دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر و فنآوری اطلاعات

گزارش فنی

بررسی شبکه باور عمیق و ارائه یک جعبه ابزار شئ گرا در MATLAB (DeeBNet Ver2.1)

نگارش:

محمدعلی کیوان‌راد

استاد راهنما:

دکتر محمدمهدی همایون‌پور

مرداد سال ۱۳۹٤


بسم الله الرحمن الرحيم

چکیده

همواره دسته‌بندی و استخراج ویژگی از مهمترین مسائل مطرح در هوش مصنوعی بوده‌اند. استخراج ویژگی مرحله‌ای است که در آن از داده خام اولیه مانند صوت و تصویر، بردارهای ویژگی برای مقاصدی چون دسته‌بندی استخراج می‌شود. امروزه شبکه‌های باور عمیق ابزاری هستند که هم در یادگیری ویژگی و هم در طبقه‌بندی داده‌ها بکار گرفته می‌شوند.

مطالعات انجام شده نشان داده‌اند که شبکه‌های باور عمیق در لایه‌های خود می‌توانند به ویژگی‌هایی مشابه آنچه در لایه‌های مختلف مغز استخراج می‌شود، دست یابند. نتایج نشان داده شده در مقالات حاکی از آن است که این شبکه‌ها در حوزه صوت و گفتار دارای نتایج خوبی در یادگیری ویژگی و طبقه‌بندی هستند به طوری که قادرند با بهترین ویژگی‌های موجود در این حوزه و الگوریتم‌های یادگیری رقابت کنند. در مواردی که داده برچسب خورده کمی در اختیار باشد و یا در کاربردهای جدید صوت و گفتار مثل تعیین نوع موسیقی و رخدادهای صوتی وارد شویم، شبکه‌های باور عمیق برتری قابل توجهی نسبت به بقیه روش‌ها از خود نشان می دهند.

شبکه‌های باور عمیق از لایه‌هایی به نام ماشین بولتزمن محدود ساخته شده‌اند. در کنار کارهای زیادی که در راستای بهبود شبکه‌های باور عمیق صورت گرفته است، دستیابی به یک روش آموزش مناسب برای این نوع شبکه‌ها یک چالش مهم محسوب شده و بسیاری از مقالات به این موضوع مهم پرداخته‌اند.

از جمله این مشکلات در آموزش شبکه‌های باور عمیق، محاسبه گرادیان لگاریتم احتمال داده آموزشی است. با توجه به تعریف لگاریتم احتمال داده آموزشی، استفاده از روش‌های معمول در محاسبه گرادیان محاسبه پذیر نبوده و لذا روش‌های تخمین با کمک نمونه‌برداری از توزیع مدل مورد توجه قرار گرفته‌اند. در ادامه این گزارش به بررسی چندین روش نمونه‌برداری برای محاسبه تخمینی گرادیان نیز پرداخته شده است.

بعد از این توضیح شبکه‌های باور عمیق، جعبه ابزار شئ گرای نوشته شده به نام DeeBNet (Deep Belief Network) در نرم‌افزار MATLAB توضیح داده شده و نمونه‌هایی از خروجی توابع و کلاس‌های ساخته شده برای آن، نمایش داده شده است. با کمک این جعبه ابزار علاوه بر امکان ساخت یک مدل به صورت مولد که امکان تولید داده از روی مدل را فراهم می‌کند، می‌توان از آن به صورت تمایزی نیز استفاده نمود که دقت بسیار بالایی را در دسته-بندی دادگان معروف MNIST نشان می‌دهد که تمامی کدهای نمونه برای رسیدن به این دقت و همچنین تولید مجدد داده‌ها از روی مدل و چندین کاربرد دیگر، به همراه جعبه ابزار ارائه شده است.

دراین جعبه ابزار سعی شده است تا بسیاری از انواع RBM که در مقالات مختلف به آن پرداخته شده است، پیاده‌سازی و تست گردد. همچنین استفاده از شئ‌گرایی در این جعبه ابزار باعث شده تا امکان استفاده مجدد از کدها بسیار بالا رفته و همچنین اصلاح و یا توسعه جعبه‌ابزار بسیار راحت و سریع باشد.

کلمات کلیدی:شبکه‌های باور عمیق، ماشین بولتزمن محدود، یادگیری ویژگی، تنک‌سازی، پردازش صوت و گفتار

فهرست علائم اختصاری

علائم اختصاری مورد استفاده در این گزارش در جدول ذیل لیست شده‌اند. واژه‌نامه‌ای نیز در انتهای گزارش ارائه شده است که در آن معادل انگلیسی اصطلاحات و کلمات مهم این پیشنهاد رساله قید گردیده است.

Boltzmann Machine	BM
Contrastive Divergence	CD
Convolutional Deep Belief Network	CDBN
Convolutional RBM	CRBM
Deep Belief Network	DBN
Discriminative RBM	DRBM
Fast Persistent Contrastive Divergence	FPCD
Markov Random Field	MRF
Maximum A Posteriori probability estimate	MAP
Neural Network	NN
Persistent Contrastive Divergence	PCD
Persistent Contrastive Divergence with Partial Smoothing	PCD PS
Principal Component Analysis	PCA
Restricted Boltzmann Machine	RBM
Stochastic Maximum Likelihood	SML

فهرست نمادها

نمادهای مورد استفاده در این گزارش در جدول ذیل لیست شده‌اند.

X	ماتریس بردارهای آموزشی
$\boldsymbol{x}$	بردار داده آموزشی
y	برچسب (کلاس) داده آموزشی
$\boldsymbol{v}$	بردار واحدهای مشاهده‌پذیر
$\boldsymbol{h}$	بردار واحدهای مخفی
$\boldsymbol{L}$	ماتریس وزن اتصالات بین واحدهای مشاهده‌پذیر با مشاهده‌پذیر
$\boldsymbol{J}$	ماتریس وزن اتصالات بین واحدهای مخفی با مخفی
$\boldsymbol{W}$	ماتریس وزن اتصالات بین واحدهای مشاهده‌پذیر با مخفی
$\boldsymbol{a}$	بردار بایاس واحدهای مشاهده‌پذیر
$\boldsymbol{b}$	بردار بایاس واحدهای مخفی
E	مقدار انرژی
Z	مقدار ثابت نرمالسازی
$< >$	امید ریاضی
ϵ	نرخ یادگیری
g	تابع سیگموئید لاجستیک
$P(v)$	احتمال بردار v
ϕ	لگاریتم احتمال داده آموزشی
θ	پارامترهای مدل
m	ضریب ممنتم
$\boldsymbol{D}$	ماتریس عناصر دیکشنری در کدینگ تنک
α	بردار ضرایب کدینگ تنک (آلفا)
g_x	ابعاد بردار x
σ	انحراف معیار در توزیع گوسی
p	ثابت کنترل‌کننده میزان فعالیت واحدهای مخفی ($p \ll 1$)
q	میزان واقعی فعالیت واحدهای مخفی
λ	ثابت کنترل‌کننده میزان تاثیر عبارت تنک‌کننده
l_j	ورودی واحد j ام

فهرست مطالب

۱ – مقدمه

از سال‌ها پیش استفاده از شبکه‌های عصبی به عنوان یکی از ابزارهای مهم در کاربردهای هوش مصنوعی مطرح بوده است. کاربردهایی همچون تشخیص اشیاء، آنالیز صوت و گفتار و بررسی متون از جمله کاربردهایی است که در آن‌ها از شبکه‌های عصبی استفاده می‌شود. بر اساس دلائل تئوری و بیولوژیکی پیشنهاد می‌شود تا برای چنین سیستم‌هایی از معماری عمیق شامل تعداد زیادی لایه پردازشی غیر خطی استفاده شود.

اما این مدل‌های عمیق دارای لایه‌های زیاد مخفی و تعداد زیادی پارامتر هستند که باید آموزش داده شوند. این پیچیدگی محاسباتی و فضای بزرگ پارامترها موجب شده تا در روش‌های رایج در شبکه‌های عصبی کمتر از لایه‌های با تعداد زیاد استفاده گردد. مشکل تعداد لایه‌های زیاد در این نوع شبکه‌ها، علاوه بر سرعت پایین آموزش، قرار گرفتن در کمینه‌های محلی را به دنبال دارد که در اکثر مواقع ما را به نتیجه مطلوب نمی‌رساند. یکی از ابزارهای موجود برای حل این مشکل استفاده از شبکه‌های باور عمیق یا DBN[۱] است که امکان ساخت شبکه‌هایی با تعداد لایه زیاد را فراهم می‌کند [۱].

استفاده از شبکه‌های باور عمیق نه تنها در کارهای دسته‌بندی قابل استفاده است بلکه بیشتر، از آن به عنوان یک روش استخراج ویژگی استفاده می‌شود. بازنمایی داده یک نقش تعیین‌کننده در اکثر روش‌های یادگیری ماشین دارد. به همین علت بسیاری از کارهای انجام شده در گسترش الگوریتم‌های یادگیری ماشین به سمت پیش پردازش، استخراج ویژگی و یادگیری ویژگی تمایل داشته‌اند. در یادگیری ویژگی سعی می‌شود تا با کمک داده ورودی، سیستمی برای استخراج ویژگی ساخته شده تا از خروجی آن برای دسته‌بندی و کاربردهای دیگر استفاده شود. از مزایای شبکه‌های باور عمیق در یادگیری ویژگی آن است که با کمک داده‌های برچسب نخورده می‌توانند ویژگی‌های سطح بالایی از داده‌های آموزشی را استخراج [۲] و قدرت تمایز بین دسته‌های مختلف در داده‌ها را افزایش دهند [۳].

لایه‌های شبکه‌های باور عمیق از ماشین بولتزمن محدود یا RBM[۲] ساخته شده است که هر ماشین بولتزمن محدود یک مدل احتمالاتی مولد و بدون جهت است که از یک لایه مخفی برای مدل کردن یک توزیع بر روی متغیرهای مشاهده‌پذیر خود استفاده می‌کند. در واقع با روی هم قرار دادن ماشین‌های بولتزمن محدود، می‌توان شبکه‌های باور عمیق را برای پردازش‌های سلسله مراتبی بوجود آورد. لذا بیشتر تغییرات و اصلاحات انجام شده در جهت بهبود این نوع شبکه‌ها در واقع به اصلاح ماشین‌های بولتزمن محدود منتهی می‌گردد. در این گزارش با بررسی شبکه باور عمیق سعی شده است تا تعدادی از اینگونه اصلاحات که به تولید انواع RBM منتهی گردیده

[۱] Deep Belief Network (DBN)

[۲] Restricted Boltzmann Machine (RBM)

است مورد بررسی قرار گیرند. از جمله این اصلاحات می‌توان به روش‌های محاسبه گرادیان لگاریتم احتمال داده آموزشی جهت آموزش مدل، تضعیف وزن، استفاده از واحدهای گوسی، تنک‌سازی و دسته‌بندی به کمک این نوع ماشین‌های بولتزمن محدود اشاره نمود.

بیشترین کارهای انجام شده در حوزه بهبود ماشین‌های بولتزمن محدود مربوط به نحوه یادگیری و ساخت مدل است. یکی از این اصلاحات مربوط به تنک‌سازی ماشین بولتزمن محدود است که در آن سعی می‌شود تا میزان فعالیت واحدهای مخفی تا حد امکان کاهش یابد. مزایای موجود در بازنمایی تنک موجب می‌شود تا این رویکرد در بسیاری از حوزه‌های هوش مصنوعی وارد شود. از مزایای این رویکرد در شبکه‌های باور عمیق می‌توان تعمیم بهتر داده‌های آموزشی، تفسیرپذیری بهتر خروجی‌های آن و بدست آوردن خروجی‌هایی منطبق با خروجی‌های لایه‌های مختلف مغز اشاره نمود.

روش‌های ارائه شده برای بازنمایی تنک را می‌توان در دو روش ارائه شده در [٤] ,[٢] خلاصه نمود که در واقع هر دو از فاصله بین میانگین فعالیت واحدها از یک مقدار مطلوب، به عنوان معیار فاصله از میزان تنک بودن استفاده می‌کنند (توضیحات بیشتر در بخش ٢-١-٤).

دومین اصلاح اساسی صورت گرفته در شبکه‌های باور عمیق به نحوه آموزش این شبکه‌ها و محاسبه گرادیان لگاریتم احتمال داده آموزشی مربوط می‌شود. با توجه به تعریف لگاریتم احتمال داده آموزشی، استفاده از روش‌های معمول در محاسبه گرادیان محاسبه‌ناپذیر بوده و لذا به سراغ روش‌های تخمین با کمک نمونه‌برداری از توزیع مدل رفته‌اند. در یکی از این روش‌ها از چندین زنجیره نمونه‌برداری گیبز برای یافتن یک نمونه مناسب از مدل استفاده می‌شود (که در بخش ٢-١-١ توضیح داده شده است). در این روش در هنگام اصلاح پارامترها، از نمونه‌های تولید شده توسط این زنجیره‌ها استفاده می‌گردد.

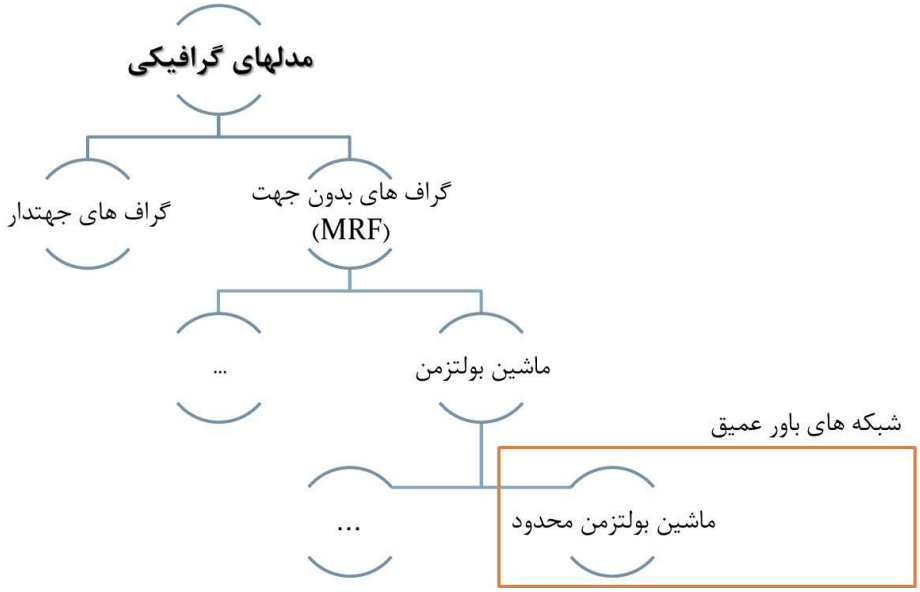
در ادامه این بررسی شبکه‌های باور عمیق، در فصل ٢ به بررسی بیشتر شبکه‌های باور عمیق و انواع اصلاحات انجام شده در آن می‌پردازیم. از آن‌جا که اصلاحات انجام شده بیشتر مربوط به ماشین بولتزمن محدود است لذا در ابتدا این اصلاحات بیان شده و سپس نحوه ترکیب این مدل‌ها برای ساخت شبکه‌های باور عمیق مطرح گردیده است.

فصل ٣ به نحوه استفاده از جعبه ابزار شبکه باور عمیق طراحی شده می‌پردازد. در این قسمت کلاس‌ها و توابع طراحی شده به طور کامل توضیح داده شده است و نمونه‌هایی از نحوه استفاده و همچنین بعضی از خروجی‌ها نیز نمایش داده شده است.

در پایان نیز بعد از ارائه مراجع، در بخش ضمائم روش‌های دیگری در اصلاح ماشین‌های بولتزمن محدود، توضیح مختصری از کدینگ تنک، دادگان ارقام دستنویس (MNIST) و چند تعریف که در این گزارش مورد استفاده قرار گرفته است و بدانها اشاره شده، آورده شده است.

۲ – ماشین بولتزمن محدود و شبکه‌های باور عمیق

شبکه‌های باور عمیق از چند لایه ماشین بولتزمن محدود ساخته شده‌اند. ماشین بولتزمن محدود نیز نوعی ماشین بولتزمن است که اتصالات بین واحدهای مخفی و بین واحدهای مشاهده‌پذیر قطع می‌باشد. ماشین بولتزمن نوعی مدل گرافیکی بدون جهت می‌باشد که به آن میدان تصادفی مارکوف یا MRF[۳] می‌گویند. در ادامه ابتدا ماشین بولتزمن محدود و انواع اصلاح شده آن را بیان نموده و سپس نحوه ترکیب آن‌ها برای ساخت شبکه‌های باور عمیق را بیان خواهیم نمود. البته چند اصلاح دیگر موجود در شبکه‌های باور عمیق نیز در که در پیشنهادات به آن اشاره نشده است، در قسمت ضمائم قرار گرفته‌اند.



شکل ۲-۱ : شبکه‌های باور عمیق از چند لایه ماشین بولتزمن محدود ساخته شده‌اند. ماشین بولتزمن محدود نیز نوعی ماشین بولتزمن است که اتصالات بین واحدهای مخفی و بین واحدهای مشاهده‌پذیر قطع می‌باشد. ماشین بولتزمن نیز نوعی مدل گرافیکی بدون جهت می‌باشد.

۲-۱ ماشین بولتزمن محدود

ماشین‌های بولتزمن[٤] نوع خاصی از میدان‌های تصادفی مارکوف هستند. یک ماشین بولتزمن، شبکه‌ای متقارن با واحدهای تصادفی باینری است. این شبکه دارای مجموعه‌ای از واحدهای قابل مشاهده $v \in \{0,1\}^{g_v}$ و مجموعه‌ای از واحدهای مخفی $h \in \{0,1\}^{g_h}$ می‌باشد که g_v و g_h به ترتیب تعداد واحدهای مخفی و تعداد واحدهای مشاهده‌پذیر است (شکل ۲-۲). انرژی حالت $\{v, h\}$ در ماشین بولتزمن به صورت زیر تعریف می‌شود:

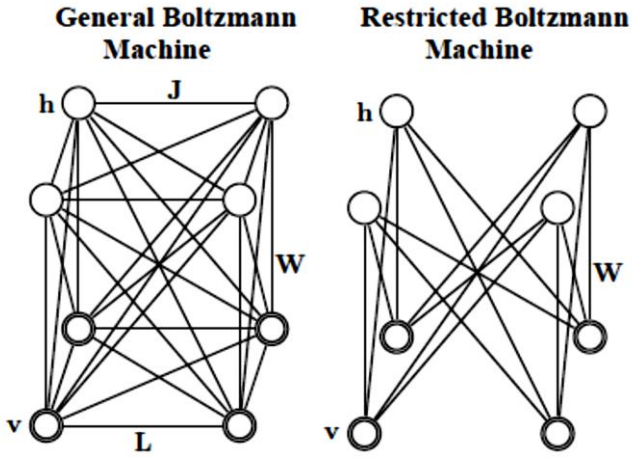
$$E(v,h) = -\frac{1}{2}v^T L v - \frac{1}{2}h^T J h - v^T W h \qquad (۱-۲)$$

[۳] Markov Random Field (MRF)

[٤] Boltzmann Machine (BM)

دراینجا برای نمایش ساده‌تر بایاس حذف شده است. پارامترهای W، L و J به ترتیب نشاندهنده وزن‌های متقارن بین واحدهای مشاهده‌پذیر به مخفی، مشاهده‌پذیر به مشاهده‌پذیر و مخفی به مخفی است. عناصر قطری L و J برابر صفرند.

به دلیل پیچیدگی موجود در محاسبات ماشین بولتزمن، از نوع ساده‌تری از آن به نام ماشین بولتزمن محدود[۵] استفاده می‌شود. تنظیم $J = 0$ و $L = 0$ مدل مشهور ماشین بولتزمن محدود را می‌دهد (تصویر سمت راست شکل ۲-۲).



شکل ۲-۲: سمت چپ: یک ماشین بولتزمن کلی. لایه بالا نشاندهنده ویژگی‌های مخفی باینری تصادفی است و لایه پایین نشاندهنده بردارهای متغیرهای قابل مشاهده باینری تصادفی است. سمت راست: یک ماشین بولتزمن محدود، بدون اتصالات بین واحدهای مخفی به مخفی و مشاهده‌پذیر به مشاهده‌پذیر. [5]

در ماشین بولتزمن محدود انرژی حالت $\{v, h\}$ با در نظر گرفتن بایاس برابر است با :

$$E(v,h) = -v^T W h - a^T v - b^T h = -\sum_{i=1}^{g_v}\sum_{j=1}^{g_h} W_{ij} v_i h_j - \sum_{i=1}^{g_v} a_i v_i - \sum_{j=1}^{g_h} b_j h_j \qquad (۲-۲)$$

که در آن W_{ij} نشاندهنده ترم تراکنشی متقارن بین واحد مشاهده‌پذیر i و واحد مخفی j است، b_i و a_j به ترتیب ترم بایاس برای واحد مخفی و مشاهده‌پذیر هستند. این شبکه به هر حالتِ ممکنِ مقادیرِ بردارهای مشاهده‌پذیر و مخفی با تابع انرژی، یک مقدار احتمال نسبت می‌دهد. تعریف توابع پتانسیل در MRF ها و ویژگی مثبت بودن آن، ما را به سمت استفاده از تابع پتانسیل به صورت نمایی و به شکل توزیع بولتزمن هدایت می‌کند. توزیع توام به صورت ضرب توابع پتانسی تعریف می‌گردد که بدین ترتیب انرژی کل با جمع انرژی توابع پتانسیل بدست می‌آید [٦]. در نتیجه، توزیع توام بر روی واحدهای قابل مشاهده و مخفی به صورت زیر تعریف می‌شود:

$$P(v,h) = \frac{1}{Z}\exp\big(-E(v,h)\big) \qquad (۳-۲)$$

مقدار Z به عنوان تابع تقسیم یا ثابت نرمال‌سازی شناخته می‌شود که از جمع تمام حالات بردارهای مشاهده‌پذیر و مخفی بدست می‌آید.

[۵] Restricted Boltzmann Machine (RBM)

$$Z = \sum_{v}\sum_{h} \exp\big(-E(v,h)\big) \qquad (۴-۲)$$

مقدار Z به عنوان تابع تقسیم یا ثابت نرمال‌سازی شناخته می‌شود. احتمالی که مدل به بردار قابل مشاهده v تخصیص می‌دهد برابر با جمع بر روی تمام حالات ممکن بردارهای مخفی h می‌باشد.

$$P(v) = \sum_{h} P(v,h) = \frac{1}{Z}\sum_{h} \exp\big(-E(v,h)\big) \qquad (۵-۲)$$

احتمالی را که شبکه به داده آموزشی نسبت می‌دهد را می‌توان با تنظیم وزن‌ها و بایاس برای رسیدن به انرژی کمتر برای این داده و انرژی بیشتر برای داده‌های دیگر که باعث تغییر در Z می‌شوند، بالا برد. بر این اساس و برای یافتن پارامترهای مناسب، تابع هدف به صورت زیر تعریف می‌شود:

$$maximize_{\{w_{ij},a_i,b_j\}} \frac{1}{m}\sum_{l=1}^{m} \log\left(\sum_{h} P\big(\boldsymbol{v}^{(l)}, \boldsymbol{h}^{(l)}\big)\right) \qquad (۶-۲)$$

که در آن m تعداد نمونه‌های آموزشی است و هدف بالا بردن احتمال این مدل برای داده‌های آموزشی است. بدین منظور با گرفتن مشتق جزئی این عبارت نسبت به w_{ij} خواهیم داشت [۷]:

$$\frac{\partial}{\partial w_{ij}}\left(\frac{1}{m}\sum_{l=1}^{m} \log\left(\sum_{h} P\big(\boldsymbol{v}^{(l)}, \boldsymbol{h}^{(l)}\big)\right)\right) \qquad (۷-۲)$$
$$= \frac{1}{m}\sum_{l=1}^{m}\sum_{h} X_{il} h_j P(h|v=x) - \sum_{v'}\sum_{h'} v'_i h'_j P(v',h')$$

که در آن X_{il} نشاندهنده i امین واحد (یا بعد) از l امین داده آموزشی است. عبارت اول در مشتق بالا را می‌توان به طور دقیق محاسبه نمود ولی محاسبه عبارت دوم، امکان‌پذیر نیست. لذا به سراغ روش‌های دیگری برای تخمین این مقدار می‌روند. مشتق لگاریتم احتمال برای داده آموزشی نسبت به وزن‌های شبکه را می‌توان به صورت زیر با تخمین محاسبه نمود.

$$\frac{\partial \log P(v)}{\partial w_{ij}} = < v_i h_j >_{data} - < v_i h_j >_{model} \qquad (۸-۲)$$

که در اینجا براکت نشاندهنده محاسبه امید ریاضی بر روی ضرب مقادیر واحد مخفی و مشاهده‌پذیر مشخص شده است. بدین ترتیب و با داشتن این مقدار مشتق، می‌توان به سادگی قانون اصلاح وزن‌ها در لگاریتم احتمال برای داده آموزشی را به صورت زیر بدست آورد.

$$\Delta w_{ij} = \epsilon\,(< v_i h_j >_{data} - < v_i h_j >_{model}) \qquad (۹-۲)$$

که در آن ϵ نرخ یادگیری است. به طور مشابه می‌توان قانون اصلاح وزن‌ها در پارامترهای بایاس را نیز نوشت.

$$\Delta a_i = \epsilon\,(< v_i >_{data} - < v_i >_{model}) \qquad (۱۰-۲)$$

$$\Delta b_j = \epsilon\,(< h_j >_{data} - < h_j >_{model}) \qquad (۱۱-۲)$$

از آنجا که اتصالات مستقیم بین واحدهای مخفی وجود ندارد، لذا این واحدها به شرط واحدهای مشاهده‌پذیر از هم مستقل می‌شوند. این نکته در واقع از مسائل مربوط به میدان‌های تصادفی مارکوف[٦] گرفته شده است [٦]. بدین ترتیب با داشتن مقدار واحدهای مشاهده‌پذیر v ، حالت باینری h_j برای هر واحد مخفی با احتمال زیر مقدار 1 را می‌گیرد.

$$P(h_j = 1|\boldsymbol{v}) = \mathcal{g}\left(b_j + \sum_i v_i w_{ij}\right) \qquad (۱۲-۲)$$

که در آن $\mathcal{g}(x)$ تابع سیگموئید لاجستیک با تعریف $\mathcal{g}(x) = 1/(1+\exp(-x))$ است. بدین ترتیب محاسبه $< v_i h_j >_{data}$ به راحتی و با بدست آوردن مقادیر h_j محاسبه می‌شود.

همچنین از آنجا که اتصالات مستقیم بین واحدهای مشاهده‌پذیر وجود ندارد، می‌توان حالت واحدهای مشاهده‌پذیر را به شرط داشتن حالت واحدهای مخفی به صورت زیر بدست آورد.

$$P(v_i = 1|\boldsymbol{h}) = \mathcal{g}\left(a_i + \sum_j h_j w_{ij}\right) \qquad (۱۳-۲)$$

اما بدست آوردن مقدار $< v_i h_j >_{model}$ کمی پیچیده‌تر می‌باشد. برای محاسبه این مقدار باید از یک مقدار تصادفی در واحدهای مشاهده‌پذیر شروع کرده و نمونه‌برداری گیبز را برای مدت طولانی اجرا نماییم. اما به سبب امکانپذیر نبودن این روش و زمان اجرای بالای آن، از روش دیگری به نام واگرایی متقابل[٧] استفاده می‌شود [٨]. در بخش‌های بعدی به نحوه محاسبه این مقدار پرداخته خواهد شد.

تعریف دیگری از تابع انرژی

در بعضی دیگر از پژوهش‌های مربوط به ماشین بولتزمن محدود از نوع دیگری از تابع انرژی نیز استفاده شده است [٩], [٢]. در کل تعریف تابع انرژی به شکلی است که به کمک آن مدل به نحوی ساخته شود تا مقدار انرژی برای حالات بدست آمده از توزیع داده‌های آموزشی کم بوده و برای داده‌های غیر محتمل در این مدل، مقدار زیادی داشته باشد. بدین ترتیب تعریف دوم از تابع انرژی به صورت زیر خواهد بود.

$$\begin{aligned} E'^{(v,h)} &= -\frac{1}{\sigma^2}(v^T W h + a^T v + b^T h) \\ &= -\frac{1}{\sigma^2}\left(\sum_{i=1}^{g_v}\sum_{j=1}^{g_h} W_{ij} v_i h_j + \sum_{i=1}^{g_v} a_i v_i + \sum_{j=1}^{g_h} b_j h_j\right) \end{aligned} \qquad (۱٤-۲)$$

[٦] Markov Random Field (MRF)

[٧] Contrastive Divergence (CD)

که σ یک مقدار ثابت است و بقیه پارامترها مشابه تعریف قبلی از تابع انرژی است. بر اساس این تعریف احتمالات شرطی واحدهای مشاهده‌پذیر و مخفی نیز به صورت زیر تغییر می‌کنند.

$$P(h_j = 1|\boldsymbol{v}) = \mathcal{g}\left(\frac{1}{\sigma^2}\left(b_j + \sum_i v_i w_{ij}\right)\right) \quad (۱۵-۲)$$

$$P(v_i = 1|\boldsymbol{h}) = \mathcal{g}\left(\frac{1}{\sigma^2}\left(a_i + \sum_j h_j w_{ij}\right)\right) \quad (۱۶-۲)$$

که مشابه قبل، $\mathcal{g}(x)$ تابع سیگموئید لاجستیک با تعریف $\mathcal{g}(x) = 1/(1+\exp(-x))$ است و در بقیه موارد همانند تابع انرژی قبلی عمل می‌کند.

به سبب مزایای ماشین‌های بولتزمن محدود، در سال‌های اخیر استفاده زیادی از این مدل به خصوص در ساخت شبکه‌های باور عمیق شده است. به همین دلیل، در مقالات زیادی نیز سعی در بهبود این مدل و افزایش کارآیی آن داشته‌اند. در ادامه قصد دایم تا این بهبودها را در بخش‌های مختلف ساخت ماشین بولتزمن محدود مشاهده نماییم.

۲-۱-۱ نحوه محاسبه گرادیان لگاریتم احتمال داده آموزشی

در واقع بر اساس رابطه (۲-۵) می‌توان $\log P(v)$ را به صورت زیر نوشت [۱۰].

$$\begin{aligned}\phi &= \log P(v) = \phi^+ - \phi^- \\ \phi^+ &= \log \sum_h \exp(-E(v,h)) \\ \phi^- &= \log Z = \log \sum_v \sum_h \exp(-E(v,h))\end{aligned} \quad (۱۷-۲)$$

محاسبه گرادیان ϕ^+ نسبت به پارامترهای مدل را گرادیان مثبت و محاسبه گرادیان ϕ^- نسبت به پارامترهای مدل را گرادیان منفی می‌گویند.

$$\begin{aligned}\frac{\partial \phi^+}{\partial w_{ij}} &= v_i . P(h_j = 1|v) \\ \frac{\partial \phi^-}{\partial w_{ij}} &= P(v_i = 1, h_j = 1)\end{aligned} \quad (۱۸-۲)$$

نحوه محاسبه گرادیان مثبت به راحتی با مقادیر داده آموزشی قابل محاسبه است اما محاسبه گرادیان منفی محاسبه ناپذیر بوده و لذا در محاسبه گرادیان از روش‌های استنتاج با نمونه‌برداری استفاده می‌کنند.

بر این اساس و با توجه به مطالب گفته شده در بخش‌های قبل مشاهده نمودیم که گرادیان لگاریتم احتمال داده آموزشی از رابطه (۲-۸) بدست می‌آید. برای استفاده از این مقدار گرادیان در اصلاح پارامترهای w_{ij} بر اساس رابطه (۲-۹) نیاز به محاسبه دو مقدار $< v_i h_j >_{data}$ و $< v_i h_j >_{model}$ می‌باشد. در بسیاری از نوشته‌های مربوط

به ماشین بولتزمن محدود، به محاسبه $< v_i h_j >_{data}$ فاز مثبت[۸] و به محاسبه $< v_i h_j >_{model}$ فاز منفی[۹] می‌گویند که متناظر با گرادیان مثبت و گرادیان منفی در رابطه (۲-۱۸) می‌باشد.

نحوه صحیح محاسبه $< v_i h_j >_{data}$ به سبب نبودن اتصالات بین واحدهای مخفی و مستقل شدن این واحدها از یکدیگر، بسیار ساده بوده و با در نظر گرفتن واحدهای مشاهده‌پذیر v (که با داده آموزشی مقداردهی شده‌اند) و ۱ در نظر گرفتن واحدهای مخفی با احتمال $P(h_j = 1|v)$ که در رابطه (۲-۱۲) آمده، بدست می‌آید. بدین ترتیب با محاسبه این مقدار احتمال، در صورتی که مقدار یک عدد تصادفی با توزیع یکنواخت در بازه $[0,1]$ کمتر از اندازه احتمال محاسبه شده باشد، می‌توان این واحد مخفی را روشن (مقدار یک) در نظر گرفت.

اما مشکل اصلی در محاسبه فاز منفی است. در عمل هم تفاوت انواع روش‌های مربوط به محاسبه گرادیان لگاریتم احتمال داده آموزشی (مانند CD[۱۰]، PCD[۱۱] و FPCD[۱۲]) از تفاوت در بدست آوردن نمونه‌های مربوط به فاز منفی، ناشی می‌شوند [۱۱].

برای محاسبه $< v_i h_j >_{model}$ می‌توان از روش نمونه‌برداری گیبز[۱۳] استفاده نمود [6]. برای این منظور از یک مقدار تصادفی در واحدهای مشاهده‌پذیر شروع کرده و مراحل نمونه‌برداری گیبز را برای یک زمان طولانی اجرا می‌کنیم. یک مرحله از نمونه‌برداری گیبز معادل بهنگام کردن همه واحدهای مخفی مطابق رابطه (۲-۱۲) و سپس بهنگام کردن تمام واحدهای مشاهده‌پذیر مطابق رابطه (۲-۱۳) می‌باشد. در واقع روش نمونه‌برداری گیبز، روشی برای بدست آوردن یک نمونه مناسب از توزیع توام مربوط به v و h مربوط به این مدل می‌باشد.

---

[۸] Positive phase

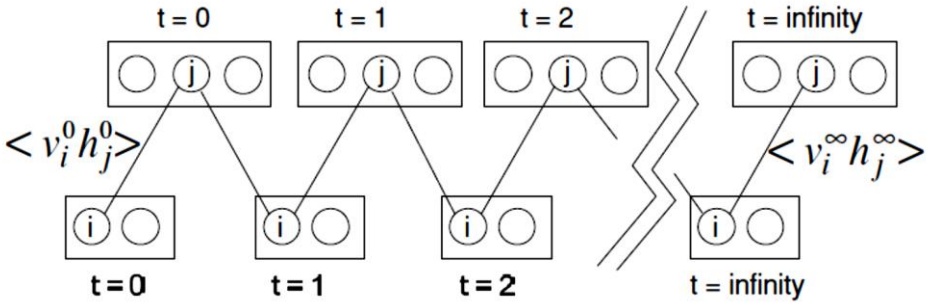
[۹] Negative phase

[۱۰] Contrastive Divergence (CD)

[۱۱] Persistent Contrastive Divergence (PCD)

[۱۲] Fast Persistent Contrastive Divergence (FPCD)

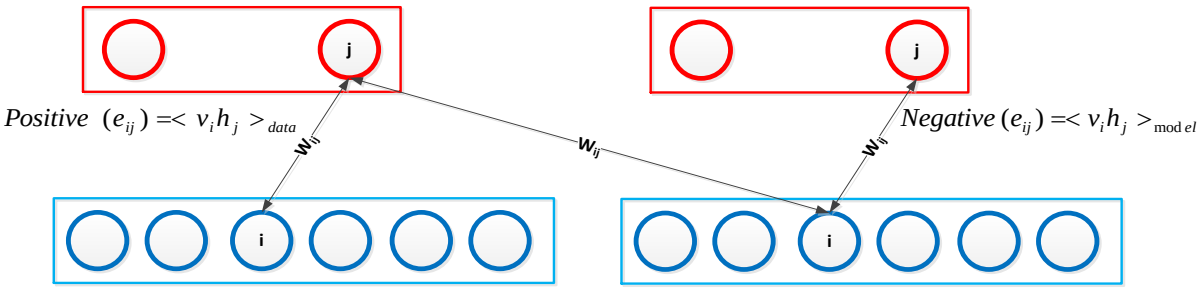
[۱۳] Gibbs sampling



شکل ۲-۳: نمونه‌برداری گیبز. یک مرحله از نمونه‌برداری گیبز معادل بهنگام کردن همه واحدهای مخفی مطابق رابطه (۲-۱۲) و سپس بهنگام کردن تمام واحدهای مشاهده‌پذیر مطابق رابطه (۲-۱۳) می‌باشد. مقدار $< v_i^0 h_j^0 >$ نشاندهنده مرحله صفر از اجرای نمونه‌برداری گیبز و محاسبه امید ریاضی ضرب $v_i h_j$ می‌باشد [۱۲].

۲-۱-۱-۱ *روش واگرایی متقابل*[۱٤]

به سبب سرعت پایین روش نمونه‌برداری گیبز و غیر عملی بودن استفاده از آن، روش سریع دیگری به نام واگرایی متقابل یا CD1 برای این منظور ارائه گردیده است [۸]. در این روش حالت واحدهای مشاهده‌پذیر به یک داده آموزشی مقداردهی اولیه می‌شوند. سپس حالت باینری واحدهای مخفی مطابق رابطه (۲-۱۲) محاسبه می‌گردد. بعد از آنکه حالت باینری واحدهای مخفی مشخص گردید، مقادیر v_i بر اساس رابطه (۲-۱۳) بازسازی می‌شوند. در انتها نیز مجددا مقادیر احتمال فعال شدن واحدهای مخفی محاسبه شده و از روی مقادیر نهایی بدست آمده از واحدهای مخفی و مشاهده‌پذیر، مقدار $< v_i h_j >_{model}$ محاسبه می‌شود. در شکل ۲-٤ نحوه عملکرد این روش به صورت گرافیکی نمایش داده شده است.



شکل ۲-٤: نحوه عملکرد روش CD1 به صورت گرافیکی. در اینجا $Positive\ (e_{ij})$ مربوط به محاسبه $< v_i h_j >_{data}$ برای یال e_{ij} است.

هر چند روش CD_1 دقیقا معادل گرادیان مورد نظر نیست اما نتایج خوبی توسط آن بدست آمده است [۸]. البته می‌توان با افزایش تعداد مراحل اجرای نمونه‌برداری گیبز به روش CD_k دست یافت که در آن k مرحله نمونه‌برداری گیبز اجرا شده است که دقت بالاتری داشته و به گرادیان مورد نظر نیز نزدیکتر است [۱۳].

[۱٤] Contrastive Divergence (CD)

۲-۱-۱-۲ روش واگرایی متقابل پایدار[۱۵]

با این همه به دلیل کمبودها و دقیق نبودن CD_k روش‌های دیگری برای استنتاج در ماشین بولتزمن محدود قابل استفاده است. یکی از این روش‌ها درستنمایی بیشینه احتمالاتی[۱۶] (یا SML) است که به آن واگرایی متقابل پایدار (یا PCD) نیز می‌گویند [۱٤]. در این روش بر خلاف روش CD به جای مقداردهی اولیه با داده آموزشی، از حالت قبلی زنجیره در بهنگام‌سازی مرحله قبل استفاده می‌شود. به عبارت دیگر در PCD برای تخمین $< v_i h_j >_{model}$ از اجرای پیوسته زنجیره گیبز استفاده می‌شود. با وجود اینکه در هر مرحله تمام پارامترهای مدل تغییر می‌کند اما به سبب تغییر کم این پارامترها، می‌توان با مراحل کمی از نمونه‌برداری گیبز، به نمونه‌هایی از توزیع مدل رسید [۱۵]. یکی از راهبردها برای داشتن نمونه در این روش آن است که چندین زنجیره موازی را با هم اجرا کرده و در زمان لازم از هر یک از آن‌ها نمونه یا اصطلاحا ذره خیالی[۱۷] را برداشت نماییم.

۲-۱-۱-۳ روش واگرایی متقابل پایدار با هموارسازی جزئی[۱۸]

یکی از روش‌ها برای کم کردن نویز حاصل از تخمین گرادیان، استفاده از میانگین‌گیری چندین تخمین گرادیان می‌باشد. البته هر چند این میانگین‌گیری نویز تخمین گرادیان را کاهش می‌دهد اما از طرفی امکان جستجوی کامل‌تر در فضای حالات مورد نیاز برای یافتن پارامتر مناسب را نیز از دست داده و کارآیی آن پایین می‌آید. یکی از بهبودهای صورت گرفته برای PCD که به این مساله توجه نموده است، روش واگرایی متقابل پایدار با هموارسازی جزئی یا PCD PS است. در این روش، هموارسازی جهتِ کاهش خطای تخمین، تنها در تخمین گرادیان مثبت استفاده شده ولی در تخمین گرادیان منفی از آن استفاده نمی‌شود تا همچنان بتوان فضای حالات را با سرعت بیشتری جستجو نمود. این روش موجب بهبود در کارآیی روش PCD می‌شود [۱۰].

۲-۱-۱-٤ روش واگرایی متقابل پایدار سریع[۱۹]

در روش واگرایی متقابل پایدار سریع یا FPCD بهبودی در PCD صورت گرفته است [۱۰]. در این روش سعی شده تا سرعت رسیدن نمونه‌های فاز منفی به نمونه‌هایی درست از توزیع مدل، افزایش یابد. برای این منظور پارامترهای استفاده شده در فاز منفی و فاز مثبت از هم متفاوت درنظر گرفته می‌شوند. در این روش از مجموعه دیگری به نام "وزن‌های سریع[۲۰]" استفاده می‌شود که تنها برای بهنگام‌سازی نمونه‌ها (یا در اصطلاح ذره‌های خیالی) در فاز منفی به کار گرفته می‌شوند. بدین ترتیب برای بهنگام‌سازی وزن‌ها در فاز منفی از جمع وزن‌های

---

[۱۵] Persistent Contrastive Divergence (PCD)

[۱۶] Stochastic Maximum Likelihood (SML)

[۱۷] Fantasy Particle

[۱۸] Persistent Contrastive Divergence with Partial Smoothing (PCD PS)

[۱۹] Fast Persistent Contrastive Divergence (FPCD)

[۲۰] Fast weight

سریع (θ_{fast}) و وزن‌های عادی ($\theta_{regular}$) یا به عبارتی وزن‌هایی که به روش PCD بدست می‌آید، استفاده می‌گردد.

تاثیر حضور وزن‌های سریع در ابتدا زیاد بوده و با گذشت زمان یک تضعیف وزن[۲۱] قوی روی آن به سمت صفر اعمال می‌شود تا اثر آن تنها به صورت موقت مشاهده شود [۱۱]. در ادامه جهت وضوح بهتر این روش، الگوریتم آن مشاهده می‌شود. البته در اینجا جهت سادگی نمایش از ممنتم و تضعیف وزن قوی استفاده نشده است.

الگوریتم ۲-۱ : نحوه اصلاح پارامترها در RBM به روش FPCD

مقداردهی اولیه

- مقداردهی $\boldsymbol{\theta_{regular}}$ به یک مقدار تصادفی کوچک
- مقداردهی $\boldsymbol{\theta_{fast}}$ به صفر
- ساخت ۱۰۰ زنجیره گیبز $\boldsymbol{v^-}$ در حالت صفر

تکرار مراحل زیر

۱. گرفتن یک گروه کوچک از داده‌های آموزشی به نام $\boldsymbol{v^+}$

۲. محاسبه $\boldsymbol{h^+ = P(h|v^+, \theta_{regular})}$ ؛ یعنی استنتاج از پارامترهای عادی استفاده می‌کند. محاسبه گرادیان مثبت $\boldsymbol{g^+ = v^{+T} h^+}$.

۳. محاسبه $\boldsymbol{h^- = P(h|v^-, \theta_{regular} + \theta_{fast})}$ ؛ یعنی استنتاج از جمع پارامترهای عادی و وزن‌های سریع جهت سرعت بیشتر، استفاده می‌کند. محاسبه گرادیان منفی $\boldsymbol{g^- = v^{-T} h^-}$.

٤. **بهنگام‌سازی $\boldsymbol{v^-}$ با نمونه‌برداری از $\boldsymbol{P(v|h^-, \theta_{regular} + \theta_{fast})}$ ؛ یعنی یک مرحله کامل از نمونه‌برداری گیبز با استفاده از وزن‌های سریع.**

۵. محاسبه گرادیان کامل $\boldsymbol{g = g^+ - g^-}$.

٦. بهنگام‌سازی $\boldsymbol{\theta_{regular} = \theta_{regular} + g}$. وجود نرخ یادگیری عادی در این تکرار.

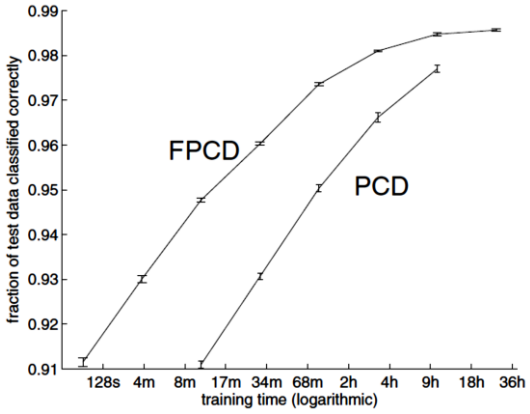
۷. بهنگام‌سازی $\boldsymbol{\theta_{fast} = \theta_{fast} . \frac{19}{20} + g}$. وجود نرخ یادگیری سریع در این تکرار.

در این الگوریتم ابتدا مقادیر اولیه برای $\theta_{regular}$ و θ_{fast} تعیین می‌شود. سپس ۱۰۰ زنجیره گیبز به طور موازی مشابه آنچه در PCD گفته شد، اجرا می‌شوند تا بتوان از هر یک از آن‌ها در اصلاح وزن‌ها استفاده نمود. سپس در یک حلقه ابتدا یک مجموعه کوچک از داده آموزشی انتخاب می‌گردد. سپس از روی پارامترهای مدل و مشابه روش‌های قبل مثل CD، مقدار h^+ از روی داده آموزشی بدست می‌آید. با این اطلاعات مقدار گرادیان مثبت محاسبه می‌شود. سپس مقدار h^- بدست می‌آید که در آن از وزن‌های سریع نیز در ساخت مدل استفاده شده است. و مجددا به کمک این اطلاعات مقدار گرادیان منفی محاسبه می‌شود. مقدار v^- نیز برای استفاده در مراحل بعد با

[۲۱] Weight decay

نمونه‌برداری بدست می‌آید. سپس مقدار $g = g^+ - g^-$ با استفاده از g^+ و g^- بدست آمده تا بتوانیم با کمک آن پارامترهای عادی مدل و پارامتر مربوط به وزن‌های سریع را بهنگام نماییم. البته همانطور که گفته شد، الگوریتم کامل این روش باید دارای تضعیف وزن قوی و ممنتوم باشد که در اینجا برای سادگی نمایش داده نشده است.

در شکل ۲-۵ دو روش PCD و FPCD مقایسه شده است. دراین مقایسه که از دادگان MNIST برای بازشناسی ارقام دستنویس استفاده شده است، در همه حالات، در زمان اجرای مساوی، روش FPCD بهتر از PCD عمل کرده است [۱۰].



شکل ۲-۵ : دقت دسته‌بندی با RMB دسته‌بندی کننده و روش‌های اصلاح پارامتر مدل PCD و FPCD در دادگان MNIST. مشاهده می‌گردد که در همه حالات، در زمان اجرای مساوی، روش FPCD بهتر عمل کرده است [۱۰].

۲-۱-۱-۵ *روش استفاده از انرژی آزاد در واگرایی متقابل پایدار (FEPCD)*

یکی از چالش‌های مطرح در ماشین‌های بولتزمن محدود، نحوه آموزش پارامترهای آن است. همانطور که قبلا توضیح داده شد، محاسبه گرادیان مدل محاسبه ناپذیر بوده و به همین دلیل از روش‌های نمونه‌برداری برای تخمین گرادیان استفاده می‌شود. علت اصلی در استفاده از روش‌های نمونه‌برداری آن است که در تخمین گرادیان نیاز به نمونه‌هایی از توزیع تعریف شده توسط مدل داریم. از آنجا که هر یک از واحدهای یک لایه در ماشین بولتزمن محدود به شرط بقیه واحدها در لایه دیگر، مستقل از یکدیگر می‌باشند، لذا بهترین روش استفاده از نمونه‌برداری گیبز است. لیکن برای رسیدن به نمونه‌های مناسب از توزیع مدل، نمونه‌برداری گیبز نیاز به اجرای تعداد زیادی نمونه‌برداری دارد که در عمل غیر ممکن است. به همین علت روش‌های مختلفی همچون CD و PCD برای حل این مشکل مطرح شده‌اند. در ادامه قصد داریم روشی را ارائه نماییم که به نمونه‌های بهتر از مدل دست یابد.

همانطور که در بخش قبل توضیح داده شد، در روش واگرایی متقابل پایدار یا PCD، چندین زنجیره نمونه‌برداری گیبز به طور موازی اجرا شده و در هر مرحله از یکی از آن‌ها به عنوان نمونه مناسب از مدل استفاده می‌شود. اما این کار به صورت کورکورانه صورت گرفته و لزوما نمونه‌ای از بهترین زنجیره انتخاب نمی‌شود. اکنون در صورتی که بتوان از یک معیار برای انتخاب زنجیره مناسب استفاده نمود، می‌توان به شکل بهتری گرادیان را محاسبه کرد.

معیار پیشنهادی برای انتخاب زنجیره مناسب‌تر استفاده از مقدار انرژی آزاد می‌باشد. انرژی آزاد بردار مشاهده‌پذیر v به صورت زیر تعریف می‌شود [16].

$$e^{-F(v)} = \sum_{h} e^{-E(v,h)} \qquad (۱۹-۲)$$

که $F(v)$ مقدار انرژی آزاد است. بدین ترتیب مقدار $F(v)$ را می‌توان به صورت زیر محاسبه نمود [16].

$$F(v) = -\sum_{i} v_i a_i - \sum_{j} q_j I_j + \sum_{j} \left(q_j \log q_j + (1-q_j)\log(1-q_j)\right) \qquad (۲۰-۲)$$

که در آن $I_j = b_j + \sum_i v_i w_{ij}$ برابر کل ورودی واحد مخفی j است و $q_j = g(I_j)$ نیز برابر احتمال فعال شدن واحد h_j به شرط v می‌باشد. یک راه مناسب برای محاسبه $F(v)$ استفاده از عبارت زیر برای محاسبه انرژی آزاد است [16].

$$F(v) = -\sum_{i} v_i a_i - \sum_{j} \log\left(1 + e^{I_j}\right) \qquad (۲۱-۲)$$

همانطور که در ابتدا گفته شد، هدف ما رسیدن به یک معیار برای انتخاب زنجیره مناسب‌تر برای استفاده از نمونه آن در محاسبه گرادیان است. یک راه مناسب استفاده از $P(v_{gen})$ است که در آن v_{gen} نمونه تولید شده توسط مدل بوده و نمونه‌ای که بیشترین احتمال را تولید کند انتخاب می‌گردد که بدین ترتیب هدف ما از روش نمونه‌برداری که رسیدن به نمونه‌هایی از مدل ساخته شده است، برآورده می‌گردد. همان طور که قبلا نیز گفته شد، محاسبه $P(v_{gen})$ امکان‌پذیر نیست و لذا از معیار $F(v_{gen})$ استفاده می‌کنیم. از آنجا که همه زنجیره‌ها دارای پارامترهای یکسانی هستند لذا مقدار Z در $P(v_{gen})$ برای همه آن‌ها یکسان است و در نتیجه می‌توانیم با مقایسه $F(v_{gen})$ (که هر چه کمتر باشد، بهتر خواهد بود) به نمونه بهتر برسیم. بدین ترتیب می‌توان در حین آموزش بهترین زنجیره‌ها را برای تولید نمونه از مدل، با این معیار بدست آورد.

۲-۱-۲ ماشین بولتزمن محدود گوسی

از کاربردهای رایج در استفاده ماشین بولتزمن محدود، در کاربردهای تصویر و صوت است که با بردارهای حقیقی یعنی $x \in \mathbb{R}^{g_x}$ کار می‌کنند. آسان‌ترین روش و همچنان رایج‌ترین رویکرد برای مدل کردن این نوع از مشاهدات با مقادیر حقیقی در قالب ماشین بولتزمن محدود، همان ماشین بولتزمن محدود گوسی یا GRBM[۲۲] نام دارد [۱۵]. اگر واحدهای مشاهده‌پذیر $v \in \mathbb{R}^{g_v}$ بوده و $h \in \{0,1\}^{g_h}$ واحدهای مخفی تصادفی باینری باشند، آنگاه می‌توان انرژی حالت $\{v,h\}$ از GRBM را به صورت زیر تعریف کرد [۱۷].

$$E(v,h) = -\sum_{i=1}^{g_v}\sum_{j=1}^{g_h} W_{ij} h_j \frac{v_i}{\sigma_i} - \sum_{i=1}^{g_v} \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j=1}^{g_h} b_j h_j \qquad (۲۲-۲)$$

[۲۲] Gaussian RBM

که در آن W_{ij} نشاندهنده ترم تراکنشی متقارن بین واحد مشاهده‌پذیر i و واحد مخفی j است، b_i و a_j به ترتیب ترم بایاس برای واحد مخفی و مشاهده‌پذیر بوده و σ_i انحراف معیار یک گوسی درنظر گرفته شده برای واحد مشاهده‌پذیر i است. بدین ترتیب توزیع حاشیه‌ای بر روی بردار مشاهدات همان رابطه (۲-۵) به صورت زیر خواهد بود.

$$P(v) = \sum_h P(v,h) = \frac{1}{Z}\sum_h \exp\big(-E(v,h)\big) \qquad (۲۳-۲)$$

که در آن Z تغییر کرده و به صورت زیر است.

$$Z = \int_v^{'} \sum_h \exp\big(-E(v,h)\big)\, dv' \qquad (۲۴-۲)$$

بدین ترتیب می‌توان توزیع شرطی را به صورت زیر بیان کرد.

$$P(v_i = x_i|\boldsymbol{h}) = \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{\left(x - a_i - \sigma_i \sum_j h_j W_{ij}\right)^2}{2\sigma_i^2}\right) \qquad (۲۵-۲)$$

و همچنین برای واحدهای مخفی خواهیم داشت،

$$P\big(h_j = 1|\boldsymbol{v}\big) = \mathcal{g}\left(b_j + \sum_i w_{ij}\frac{v_i}{\sigma_i}\right) \qquad (۲۶-۲)$$

که در آن $\mathcal{g}(x)$ تابع سیگموئید لاجستیک با تعریف $\mathcal{g}(x) = 1/(1+\exp(-x))$ است. مشاهده می‌شود که با شرط بر روی واحدهای مخفی در رابطه (۲-۲۵)، هر واحد مشاهده‌پذیر با یک توزیع گوسی مدل شده است که میانگین آن با ترکیب وزن‌های ورودی از سمت واحدهای مخفی جابجا می‌شود [۱۷]. مشتق لگاریتم درستنمایی نسبت به W به صورت زیر بدست می‌آید.

$$\frac{\partial \log P(v)}{\partial w_{ij}} = < \frac{1}{\sigma_i} v_i h_j >_{data} - < \frac{1}{\sigma_i} v_i h_j >_{model} \qquad (۲۷-۲)$$

که در اینجا براکت نشاندهنده محاسبه امید ریاضی بر روی ضرب مقادیر واحد مخفی و مشاهده‌پذیر مشخص شده است. معمولا پارامتر σ_i مقدار ثابتی در نظر گرفته شده و برای آن آموزش صورت نمی‌گیرد. لذا اگر σ_i را برابر یک در نظر بگیریم، روش اصلاح وزن‌ها هیچ تغییری نکرده و تنها باید برای محاسبه v_i از یک تابع گوسی با میانگین $a_i + \sum_j h_j W_{ij}$ نمونه‌برداری نماییم [۳].

می‌توان حاشیه بر روی واحدهای مشاهده‌پذیر را به صورت زیر بیان نمود.

$$P(v) = \sum_h P(v|h)P(h) \qquad (۲۸-۲)$$

بدین ترتیب در واقع یک GRBM را می‌توان به عنوان یک مدل مخلوط گوسی در نظر گرفت که هر وضعیت واحدهای مخفی، موقعیت عناصر مخلوط را مشخص می‌کند [۱۵]. این رابطه بدان دلیل درست است که در

$P(v|h)$ می‌توان $P(v_i|h)$ ها را به سبب ساختار RBM، مستقل در نظر گرفت و در نتیجه $P(v|h) = \prod_{v_i} P(v_i|h)$ و از آنجایی که ضرب چند تابع گوسی، خود یک تابع گوسی است، لذا $P(v|h)$ یک تابع گوسی خواهد بود و لذا نهایتا رابطه (۲-۲۸) بیانگر یک مخلوط گوسی می‌گردد.

از آنجا که تعداد عناصر مخلوط بالقوه بسیار زیاد است و به صورت نمایی با توجه به تعداد واحدهای مخفی رشد می‌کند، ظرفیت آن‌ها با به اشتراک‌گذاری تعداد نسبتا کمی از پارامترهای عناصر مخلوط، کنترل می‌شود [۱۵].

اثبات شده است که GRBM چندان مناسب برای مدل کردن تصاویر حقیقی نیست و ویژگی‌های یادگرفته شده معمولا بیانگر لبه‌های تیز که در محدوده اشیاء وجود دارد نیستند. لذا این روش منجر به بازنمایی مبهمی می‌شود که مناسب برای ویژگی‌های کاربرد دسته‌بندی نیست. بر اساس تحقیقات صورت گرفته، مشکل GRBM در تعیین مناسب ساختار آماری از تصاویر طبیعی به سبب آن است که تنها از ظرفیت میانگین شرطی در آنها استفاده شده و کواریانس شرطی در آن در نظر گرفته نشده است [۱۸]. بر اساس این تحقیقات تصاویر واقعی بیشتر با کواریانس مقادیر پیکسل و نه با مقادیر خالص آن‌ها توصیف می‌شوند. این دیدگاه با استفاده رایجی که در روش‌های پیش‌پردازش وجود دارد که مقادیر پیکسل‌ها را به طور عمومی مقیاس‌بندی مجدد می‌کنند، تطبیق می‌کند. بدین منظر روش‌های دیگری نیز وجود دارند که سعی می‌کنند تا بهتر بتوانند از کواریانس استفاده نمایند.

نوع دیگری از تعریف ماشین بولتزمن گوسی

اگر در تعریف تابع انرژی از تعریف دوم که در رابطه (۲-۱٤) آمده است استفاده نماییم، باید ماشین بولتزمن گوسی را با کمی تغییر بیاوریم. با توجه به این تعریف، تابع انرژی نوع گوسی آن به صورت زیر تعریف شده است [۲], [۹].

$$E'^{(v,h)} = -\frac{1}{2\sigma^2}\sum_{i=1}^{g_v} v_i^2 - \frac{1}{\sigma^2}\left(\sum_{i=1}^{g_v}\sum_{j=1}^{g_h} W_{ij} v_i h_j + \sum_{i=1}^{g_v} a_i v_i + \sum_{j=1}^{g_h} b_j h_j\right) \qquad (۲-۲۹)$$

که تعریف پارامترهای آن مشابه تابع انرژی گوسی قبلی است که تعریف گردید. بدین ترتیب احتمالات شرطی نیز به صورت زیر تغییر می‌کند.

$$P(v_i = x_i|\boldsymbol{h}) = \frac{1}{\sqrt{2\pi}\sigma_i}\exp\left(-\frac{\left(x - a_i - \sum_j h_j W_{ij}\right)^2}{2\sigma_i^2}\right) \qquad (۲-۳۰)$$

$$P(h_j = 1|\boldsymbol{v}) = g\left(\frac{1}{\sigma_i^2}\left(b_j + \sum_i w_{ij}\frac{v_i}{\sigma_i}\right)\right) \qquad (۲-۳۱)$$

که در آن مشابه قبل، $g(x)$ تابع سیگموئید لاجستیک با تعریف $g(x) = 1/(1+\exp(-x))$ می‌باشد.

۲-۱-۳ ماشین بولتزمن محدود با واحدهای نرمِ بیشینه[۲۳]

گاهی اوقات با متغیرهایی برخورد می‌کنیم که نه باینری هستند و نه مقدار حقیقی دارند بلکه می‌توانند مقادیر صحیح بین 1 تا K بگیرند. در این موارد از واحدهایی به نام واحد نرم بیشینه یا softmax استفاده می‌شود. یک واحد نرم بیشینه را می‌توان به صورت یک مجموعه از واحدهای باینری در نظر گرفت که تنها یکی از K واحد این مجموعه مقدار یک گرفته و بقیه مقدار صفر می‌گیرند [۱۶]. برای رسیدن به این شرط از تابع فعالیت نرم بیشینه با تعریف زیر استفاده می‌کنند.

$$p_j = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}} \qquad (۳۲-۲)$$

که p_j احتمال فعال شدن واحد j بوده و می‌توانیم در این‌جا x_j را مقدار وارد شده به هر واحد در نظر بگیریم (مثلا اگر این واحدها برای واحدهای مشاهده‌پذیر درنظر گرفته شود خواهیم داشت [۱۷]: $x_j = a_j + \sum_i h_i w_{ij}$). این رابطه باعث می‌شود تا برای مقادیری که از بیشینه فاصله دارند خروجی نزدیک صفر و برای مقدار بیشینه خروجی نزدیک یک داشته باشد.

با این تعریف از احتمال فعال شدن واحدها، قوانین آموزش برای واحدهای باینری نرم بیشینه، مشابه همان قوانین واحدهای باینری معمولی است، با این تفاوت که نحوه محاسبه احتمال حالات و همچنین نمونه‌های گرفته شده از آن تغییر می‌کند [۱۶].

۲-۱-٤ تنک‌سازی[۲٤]

تنک‌سازی یکی از مفاهیمی است که اخیرا توجه زیادی به آن شده است. این روش در علوم محاسبات نورولوژی تحت عنوان کدگذار تنک[۲۵] در سیتم بینایی [۱۹] معرفی گردیده است (توضیح بیشتر درباره کدگذار تنک در ضمیمه ۵-۱-۱ آمده است). همچنین بازنمایی تنک به عنوان یکی از ملزومات لازم در شبکه‌های باور عمیق نیز مطرح شده است [۲]. البته رویکرد تنک‌سازی در شبکه‌های باور عمیق کاملا متفاوت از چیزی است که در کدگذار تنک مشاهده می‌گردد که در ادامه این تفاوت‌ها را مشاهده خواهیم نمود.

اگر کسی بخواهد بازنمایی با اندازه ثابت[۲٦] داشته‌باشد، بازنمایی تنک نسبت به حالت غیر تنک با توجه به تئوری اطلاعات دارای دقت بالاتری است، چرا که اجازه می‌دهد تا تعداد بیت‌های لازم برای هر نمونه تغییر کند [۱۳]. بر اساس تئوری اطلاعات برای رسیدن به تعمیم خوب، کافی است مجموع تعداد بیت‌های لازم برای

---

[۲۳] Softmax units

[۲٤] Sparsifying

[۲۵] Sparse Coding

[۲٦] Fixed size

کدگذاری کل داده آموزشی را نسبت به اندازه داده آموزشی کم کنیم. در بسیاری از حوزه‌های مطرح نیز نمونه‌های متفاوت نیاز به تعداد بیت‌های متفاوتی برای فشرده‌سازی دارند.

به عبارت دیگر الگوریتم‌های کاهش بعد، چه خطی باشند مانند PCA[۲۷] یا ICA[۲۸] [٦] یا غیر خطی مانند LLE[۲۹] و ISOMAP، هر نمونه را به فضای با ابعاد کمترِ مشابهی نگاشت می‌دهند. برای روشن شدن بحث بالا، بهتر است تا هر نمونه به بازنمایی با طول متغیر نگاشت شود. برای ساده کردن بحث فرض کنید این بازنمایی یک بردار باینری باشد. اگر لازم دیده شد تا هر نمونه را به یک بازنمایی با طول ثابت نگاشت دهیم، یک راه حل خوب آن است که آن بازنمایی درجه آزادی کافی برای بازنمایی تمام نمونه‌ها را داشته باشد. در حالیکه با امکان فشرده‌سازی برای بیشتر نمونه‌ها می‌توان کدهای کوچکتر با اندازه متغیر برای بردار بیتی با اندازه ثابت بدست آورد.

بدین ترتیب دو بازنمایی وجود خواهد داشت: اول بازنمایی با طول ثابت که می‌تواند به عنوان ورودی برای تخمین زدن و تصمیم‌گیری استفاده شود، و دوم بازنمایی کوچکتر با طول متغیر که در اصل می‌تواند با استفاده از بازنمایی با طول ثابت در یک گام فشرده‌سازی بدست آید. برای مثال اگر بیت‌های لازم در بردار بازنمایی با طول ثابت دارای احتمال بالای صفر باشند (یعنی شرط تنک بودن)، آنگاه برای بیشتر نمونه‌ها، فشرده‌سازی بردار با طول ثابت بسیار ساده است (البته به طور میانگین و با توجه به میزان تنک بودن). برای یک سطح خاص از تنک بودن، تعداد پیکربندی‌های[۳۰] بردارهای تنک بسیار کمتر از وقتی است که تنک بودن کمتری داریم و یا اصلا این ویژگی را نداریم. لذا آنتروپی کدهای تنک کوچکتر است [۱۳].

بحث بعدی در مزایای تنک بودن آن است که بازنمایی با طول ثابت به عنوان ورودی برای پردازش‌های بیشتر استفاده می‌شود و لذا به سادگی تفسیر می‌گردد. یک کدگذاری شدیدا فشرده معمولا به شدت درهم آمیخته است، لذا هیچ زیر مجموعه‌ای از بیت‌های داخل کد را نمی‌توان به درستی تفسیر کرد مگر آنکه تمام بیت‌های دیگر نیز استفاده شوند. در عوض بازنمایی تنک با طول ثابت به سبب این ویژگی که هر بیت یا مجموعه کوچکی از این بیت‌ها می‌تواند تفسیرپذیر باشد، مورد توجه است. یعنی با جنبه‌های معنایی ورودی مربوط بوده و فاکتورهای متغیر در داده را نگه می‌دارد. برای مثال در یک ورودی از نوع سیگنال گفتار، اگر تعدادی از بیت‌ها کدکننده ویژگی‌های گوینده و بقیه بیت‌ها، کدکننده‌ی ویژگی‌های عمومی از واج گفته شده باشد، می‌توان بعضی از فاکتورهای متغیر در داده را حذف کرد و بعضی از زیرمجموعه‌های فاکتورها می‌تواند برای بعضی از کاربردهای خاص کافی باشد [۱۳]. مثلا می‌توان با حذف فاکتورهای مربوط به گوینده، از بقیه فاکتورها برای تشخیص گفتار استفاده کنیم.

---

۲۷ Principal Component Analysis (PCA)

۲۸ Independent Component Analysis (ICA)

۲۹ Locally-Linear Embedding (LLE)

۳۰ Configuration

بر اساس تحقیقات [۱۹]، کدینگ تنک می‌تواند ویژگی‌های سطح پایین مناسبی از داده برچسب نخورده را یاد بگیرد. در صورتی که بخواهیم، ویژگی‌های سطح بالاتری را بسازیم، امکان استفاده از کدینگ تنک کار ساده‌ای نیست. هر چند کارهایی در این زمینه برای ساخت کدینگ تنک سلسله مراتبی انجام شده است، اما آن‌ها نمی‌توانند به راحتی به صورت پشته درآمده و استفاده شوند و بار محاسباتی بالایی نیز دارند. در واقع ساخت بازنمایی سلسله مراتبی با پشته کردن لایه‌های بازنمایی تنک، به چند دلیل کار سخت و پیچیده‌ای است [۹]. اول آنکه کدینگ تنک فرض می‌کند که توزیع ورودی آن غیر تنک است در حالیکه خروجی یک کدینگ تنک بسیار تنک بوده و در نتیجه قرار دادن یک کدینگ تنک بر روی آن، فرض مدلسازی را رعایت نمی‌کند. دوم آنکه بهینه‌سازی یا استنتاج در کدینگ تنک نسبتا هزینه‌بر است چرا که دارای یک مساله بهینه‌سازی از نوع L_1 است.

لذا به سراغ روش جایگزینی برای ساخت بازنمایی سلسله‌مراتبی رفته‌اند. ایده اصلی استفاده از الگوریتم‌های یادگیری عمیق است که امکان ساخت بازنمایی‌های سلسله‌مراتبی را فراهم می‌سازد. یکی از این نوع روش‌های الگوریتم یادگیری عمیق، شبکه‌های باور عمیق است. بر خلاف کدینگ تنک، ماشین‌های بولتزمن محدود به راحتی به صورت پشته درآمده و بازنمایی سلسله‌مراتبی ایجاد می‌کنند. همچنین استنتاج در RBM و DBN از لحاظ محاسباتی کارآمد بوده و نیازی به حل مساله بهینه‌سازی در آن نیست.

به طور خلاصه می‌توان تنظیم‌کننده تنک‌سازی در RBM و DBN را دارای آثار مختلفی دانست که در ادامه به آن اشاره می‌کنیم [۱۳] ,[۹].

- بازنمایی تنک نسبت به حالت غیر تنک با توجه به تئوری اطلاعات دارای دقت بالاتری است چرا که تعداد بیت‌های لازم برای کدگذاری را کاهش داده و نتیجه آن تعمیم بهتر داده‌های آموزشی است.
- بازنمایی تنک نسبت به حالت غیر تنک به سبب طول ثابت بازنمایی آن، دارای تفسیرپذیری بیشتری در خروجی خود است و جنبه‌های معنایی ورودی بهتر در متغیرهای خروجی ظاهر می‌شوند.
- رفتار بازنمایی تنک مشابه عملکرد مغز انسان بوده و در حوزه پردازش تصویر فیلترهایی همچون فیلترهای گابور را یاد گرفته [p. ۲ ,۲] و حتی در استفاده چند لایه از این نوع روش‌ها، ویژگی‌های بدست آمده با لایه‌های بالایی در پردازش تصویر در مغز مطابقت دارد.
- بر اساس نتایج بدست آمده در مقالات، شبکه‌های باور عمیقی که خاصیت تنک بودن در لایه‌های آن وجود دارد، دارای دقت بیشتری هستند.
- به دلیل محدودیت‌های کدینگ تنک، امکان آموزش آن به صورت عمیق و یافتن ویژگی‌های سطح بالاتر پیچیده و مشکل است. اما RBMهای تنک امکان عمیق شدن و یافتن ویژگی‌های سطح بالاتر را دارا می‌باشند.

بر اساس موارد گفته شده هدف، رسیدن به ماشین‌های بولتزمن محدود تنک است که ترکیب آن‌ها شبکه‌های باور عمیق تنک را برای ما بسازد. لذا در ادامه روش‌های ساخت RMB های تنک معرفی خواهند شد.

۲-۱-٤-۱ *روش اول برای ساخت ماشین بولتزمن محدود تنک*

به طور کلی ماشین‌های بولتزمن محدود سعی دارند تا بازنمایی‌های غیر تنک و توزیع شده را یاد بگیرند. دراینجا سعی می‌شود تا با تغییر در الگوریتم یادگیری RBM، آن را به یادگیری بازنمایی تنک قادر سازیم. منظور از تنک شدن به معنای آن است که خروجی تعداد بیشتری از واحدهای مخفی صفر شوند، اینکار معادل آن است که احتمال فعال شدن تعداد بیشتری از واحدهای مخفی به سمت صفر میل کند. برای این منظور یک عبارت تنظیم‌کننده[۳۱] تعریف می‌شود که به طور میانگین، فعالیت متغیرهای مخفی را بر روی کل نمونه‌های آموزشی کم نماید [۹]. در این روش عبارت تنظیم‌کننده به نحوی اضافه می‌شود تا هر انحرافی از امید فعالیت واحدهای مخفی از مقدار ثابت و کم p را جریمه کند. یعنی این تنظیم‌کننده سعی می‌کند که نرخ فعالیت[۳۲] نرون‌های مدل (که متناظر با متغیرهای تصادفی مخفی h_j است) در یک سطح پایین باقی مانده و در نتیجه فعالیت نرون‌های مدل، تنک خواهد بود. لذا با نمونه‌های آموزشی $\{v^{(1)},\dots,v^{(m)}\}$ که نشاندهنده m نمونه آموزشی است (در واقع $\{x^{(1)},\dots,x^{(m)}\}$ نمونه‌های آموزشی هستند که با قرار دادن مقدار آن‌ها در مکان واحدهای مشاهده‌پذیر، می‌توان آنها را به صورت $\{v^{(1)},\dots,v^{(m)}\}$ نمایش داد)، مساله بهینه‌سازی زیر مطرح می‌شود:

$$minimize_{\{w_{ij},a_i,b_j\}} \quad -\sum_{l=1}^{m}\log\left(\sum_{h}P\left(\boldsymbol{v}^{(l)},\boldsymbol{h}^{(l)}\right)\right) + \lambda\sum_{j=1}^{n}\left|p-\frac{1}{m}\sum_{l=1}^{m}\mathbb{E}\left[h_j^{(l)}\middle|\boldsymbol{v}^{(l)}\right]\right|^2 \qquad (۳۳-۲)$$

که در آن $\mathbb{E}[.]$ امید ریاضی به شرط داده‌ها است و λ ثابت تنظیم‌کننده و p ثابت کنترل‌کننده میزان تنک بودن واحدهای h_j است. لذا هدف کمینه کردن جمع عبارت لگاریتم درستنمایی و عبارت تنظیم‌کننده است (یعنی برقراری یک تعادل بین درستنمایی و تنک بودن). اساسا می‌توان از کاهش گرادیان برای این مساله استفاده کرد اما محاسبه گرادیان لگاریتم درستنمایی هزینه‌بر است. لذا از الگوریتم‌های تخمین که در بخش ۲-۱-۱ مطرح گردید (مانند روش واگرایی متقابل یا CD) برای تخمین گرادیان لگاریتم درستنمایی استفاده می‌شود. بنابر این در هر تکرار از قانون بهنگام‌سازی واگرایی متقابل استفاده کرده و به دنبال آن در یک مرحله، کاهش گرادیان با استفاده از گرادیان عبارت تنظیم‌کننده را اعمال می‌نماییم. این مراحل در الگوریتم زیر مشاهده می‌شود. در پیاده‌سازی یک تغییر برای افزایش کارآیی اعمال شده است و آن اینکه از کاهش گرادیان آماری استفاده شده و دسته‌های کوچک از داده آموزشی برای تخمین به کار گرفته شده است. همچنین در بهنگام کردن پارامترها فقط $\boldsymbol{b}_j$ ها (که مستقیما درجه فعالیت نرون‌ها را کنترل می‌کنند) اصلاح می‌گردند و پارامترهای w_{ij} در این مرحله تغییر نمی‌کنند.

---

[۳۱] Regularization term

[۳۲] Firing rate

الگوریتم ۲-۲ : الگوریتم یادگیری RBM تنک [۹]

۱. **بهنگام کردن مقادیر پارامترها با استفاده از قانون یادگیری واگرایی متقابل**

$$\Delta w_{ij} = \epsilon\left(< v_i h_j >_{data} - < v_i h_j >_{model}\right)$$
$$\Delta a_i = \epsilon\left(< v_i >_{data} - < v_i >_{model}\right)$$
$$\Delta b_j = \epsilon\left(< h_j >_{data} - < h_j >_{model}\right)$$

که در آن ϵ نرخ یادگیری و $<.>_{model}$ امید بر روی داده‌های بازسازی شده است که با یک تکرار روش نمونه‌برداری گیبز بدست آمده است.

۲. **بهنگام کردن پارامترها با استفاده از عبارت تنظیم‌کننده**

۳. **تکرار مراحل ۲ و۳ تا زمان همگرایی**

گرادیان عبارت تنظیم‌کننده‌ی تنک بودن بر روی پارامترهای w و b به صورت زیر تعریف می‌شود.

$$-\frac{\partial}{\partial w_{ij}} L_{sparsity} \propto \left(p - \frac{1}{m}\sum_{l=1}^{m} q_j^{(l)}\right)\frac{1}{m}\sum_{l=1}^{m} q_j^{(l)}\left(1 - q_j^{(l)}\right) v_i^{(l)} \qquad (۲-۳۴)$$

$$-\frac{\partial}{\partial b_j} L_{sparsity} \propto \left(p - \frac{1}{m}\sum_{l=1}^{m} q_j^{(l)}\right)\frac{1}{m}\sum_{l=1}^{m} q_j^{(l)}\left(1 - q_j^{(l)}\right) \qquad (۲-۳۵)$$

که $q_j^{(l)} \triangleq g\left(\frac{1}{\sigma^2}\left(b_j + \sum_i v_i^{(l)} w_{ij}\right)\right)$ می‌باشد (در اینجا از تعریف دوم تابع انرژی استفاده شده است) و $L_{sparsity}$ عبارت جریمه تنظیم‌کننده در عبارت (۲-۳۳) است. اگر در یک تخمین فرض کنیم که مقدار عبارت $\sum_{l=1}^{m} q_j^{(l)}\left(1 - q_j^{(l)}\right)$ تقریبا به ازای j های مختلف ثابت است، قانون بهنگام‌سازی را به صورت ساده و شهودی می‌توان بیان نمود.

$$\Delta b_j \propto -\frac{\partial}{\partial b_j} L_{sparsity} \approx \left(p - \frac{1}{m}\sum_{l=1}^{m} q_j^{(l)}\right).constant \qquad (۲-۳۶)$$

که این متناظر است با قانون بهنگام کردنی که مقدار بایاس را در هر واحد مخفی بر اساس تفاوت بین میزان تنک بودن هدف و میانگین فعالیت هر واحد مخفی بر روی داده آموزشی، بالا و پایین می‌کند. در [۲] بر اساس رابطه (۲-۳۵) اصلاح b_j انجام شده ولی با استفاده از رابطه (۲-۳۶) نیز به نتایج مشابهی رسیده‌اند [۹] و این نشان می‌دهد که این تخمین در نظر گرفته شده، قابل قبول می‌باشد.

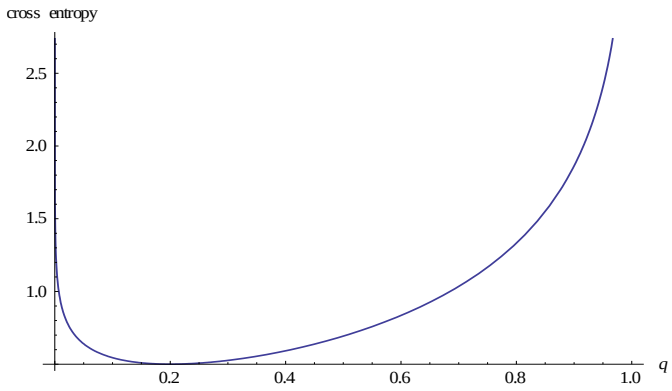
۲-۱-٤-۲ *روش دوم برای ساخت ماشین بولتزمن محدود تنک*

در این روش نیز فعالیت تنک واحدهای مخفی باینری، با یک مقدار هدف تنک بودن با $p \ll 1$ مشخص می‌گردد که این پارامتر نشاندهنده احتمال فعال بودن مورد نظر می‌باشد. بدین منظور یک عبارت جریمه اضافه

جهت ترغیب احتمال واقعی فعال شدن یعنی q به مقدار نزدیک p ، استفاده می‌شود. معیار جریمه استفاده شده برای این منظور مقدار آنتروپی‌متقابل[۳۳] بین توزیع واقعی و توزیع هدف است [٤].

$$L_{sparsity} = -p \log q - (1-p) \log(1-q) \qquad (۳۷-۲)$$

این تابع جریمه به نحوی است که در صورت فاصله داشتن مقدار q از p آن را به صورت نمایی جریمه می‌کند. به عنوان مثال در شکل ٦-۲ مشاهده می‌شود که اگر مقدار p را ۰,۲ بگیریم، هر قدر فاصله q از ۰,۲ بیشتر شود، اندازه آنتروپی متقابل یا جریمه وارد شده، بیشتر خواهد بود.



شکل ٦-۲: مقدار آنتروپی متقابل با $p = 0.2$ و مقادیر مختلف q

از آنجا که در RBM واحدهای مخفی را از نوع لاجستیک تعریف نمودیم، لذا مشتق این عبارت جریمه نسبت به کل ورودی واحد مخفی، مقدار $q - p$ خواهد بود که از آن برای تنظیم بایاس و وزن‌های ورودی به هر واحد استفاده می‌شود.

$$\begin{aligned} L_{sparsity} &= -p \log \frac{1}{1+\exp(-x)} - (1-p) \log\left(1 - \frac{1}{1+\exp(-x)}\right) \\ &= x + \log(1+\exp(-x)) - px \end{aligned} \qquad (۳۸-۲)$$

و با محاسبه مشتق نسبت به کل ورودی خواهیم داشت:

$$\frac{\partial}{\partial x} L_{sparsity} = q - p \qquad (۳۹-۲)$$

که منظور از x در اینجا، مجموع ورودی‌ها به واحد مخفی است. حتما باید مشتق را به هر دو پارامتر اعمال نمود. اگر مشتق تنها به بایاس اعمال شود، به عنوان نمونه بایاس دائما منفی‌تر می‌شود تا بتواند از فعالیت واحد مخفی جلوگیری کند اما وزن‌ها دائما مثبت‌تر می‌شوند تا بتوانند از آن واحد استفاده نمایند.

در عمل، مقدار q با استفاده از دسته‌های کوچک و به کمک رابطه زیر محاسبه می‌شود [١٦].

[۳۳] Cross entropy

$$q_{new} = \lambda q_{old} + (1-\lambda) q_{current} \quad (۴۰-۲)$$

که در آن $q_{current}$ مقدار میانگین احتمال فعالیت واحدهای مخفی بر روی دسته کوچک فعلی و λ نرخ کاهش است. مقدار مناسب برای میزان هدف تنک بودن یعنی p در حدود 0.01 تا 0.1^9 است و همچنین مقدار مناسب برای نرخ کاهش در حدود 0.9 تا 0.99 می‌باشد [۱۶]. نشان داده شده است که با این روش تنک‌سازی، می‌توان کارآیی جداسازی را با استفاده از این ویژگی‌های تنک بالا برد [٤].

در نهایت با مقایسه این دو روش تنک‌سازی برای ماشین بولتزمن محدود، مشاهده می‌کنیم که در هر دو روش تقریبا به روش اصلاح پارامترهای مشابهی رسیده‌اند. در واقع در هر دو روش (حالت ساده شده روش اول در رابطه (۲-۳۶) و روش دوم در رابطه (۲-۳۸)) از فاصله بین مقدار هدف برای تنک بودن و میانگین میزان فعالیت واحدهای مخفی در یک دسته کوچک، برای اصلاح وزن استفاده می‌شود. بدین ترتیب می‌توانیم هر دو روش را یک روش بدانیم که از دو رویکرد متفاوت به آن رسیده‌اند.

۲-۱-٤-۳ *روش سوم برای ساخت ماشین بولتزمن محدود تنک*

روش دیگر مبتنی بر تئوری نرخ اعوجاج (rate distortion theory) عمل کرده و جریمه آن مستقیما مقدار فعال شدن واحدهای مخفی است [20].

$$L_{sparsity} = -\frac{1}{m}\sum_{l=1}^{m}\left\| P\left(\boldsymbol{h}^{(l)} | \boldsymbol{v}^{(l)}\right)\right\|_1 \quad (۴۱-۲)$$

بر این اساس گرادیان این عبارت تنظیم کننده برابر رابطه زیر خواهد بود.

$$-\frac{\partial}{\partial w_{ij}} L_{sparsity} \propto -\frac{1}{m}\sum_{l=1}^{m} q_j^{(l)}\left(1-q_j^{(l)}\right) v_i^{(l)} \quad (۴۲-۲)$$

که $q_j^{(l)} \triangleq g\left(b_j + \sum_i v_i^{(l)} w_{ij}\right)$ می‌باشد.

۲-۱-٤-٤ *روش چهارم برای ساخت ماشین بولتزمن محدود تنک*

روش دیگری که توسط نویسندگان این نوشته توسعه یافته است، در آن از تابع نرمال به عنوان تابع جریمه-کننده استفاده شده است [21].

$$L_{sparsity} = \sum_{j=1}^{n} f(q_j, p, \sigma^2) \quad (۴۳-۲)$$

که در آن $q_j = \frac{1}{m}\sum_{l=1}^{m} \mathbb{E}\left[h_j^{(l)} \middle| \boldsymbol{v}^{(l)}\right]$ می‌باشد و $f(q_j, p, \sigma^2)$ نیز به صورت زیر تعریف می‌شود.

$$f(q_j, p, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{q_j - p}{\sigma}\right)^2} \quad (۴٤-۲)$$

بر این اساس گرادیان این عبارت تنظیم کننده برابر رابطه زیر خواهد بود.

$$\frac{\partial}{\partial w_{ij}} L_{sparsity} \propto \frac{1}{m}\left(p - \frac{1}{m}\sum_{l=1}^{m} q_j^{(l)}\right) f(q_j, p, \sigma^2) \sum_{l=1}^{m} q_j^{(l)}\left(1 - q_j^{(l)}\right) v_i^{(l)} \qquad (۴۵-۲)$$

۲-۱-٤-۵ *تفاوت بین ماشین بولتزمن محدود تنک و کدینگ تنک*

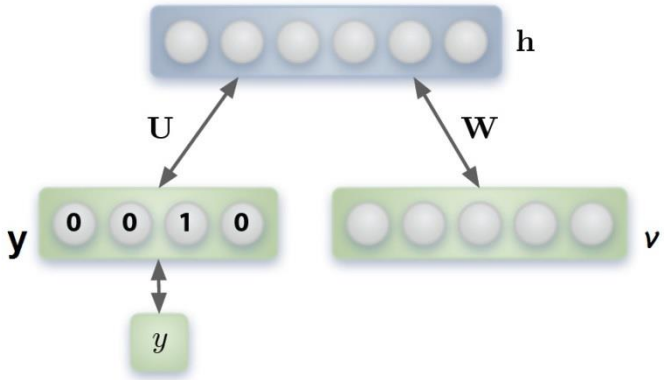
دراین بخش شباهت‌ها و تفاوت‌های بین کدینگ تنک و RBM تنک را می‌بینیم. مدل RBM تنک و کدینگ تنک از این لحاظ که هر دو داده ورودی را با استفاده از متغیرهای پنهان بازنمایی می‌کنند، مشابه هستند. بعلاوه هرچند بازنمایی RBM به طور کلی تنک نیست، اما RBM تنک می‌تواند بازنمایی‌هایی مشابه کدینگ تنک را بوجود آورد (یعنی مثلا در کاربرد پردازش تصویر، همانند کدینگ تنک، تصاویر را به صورت ترکیب لبه‌ها مدل نماید). از طرف دیگر تفاوت‌هایی نیز بین این دو وجود دارد. کدینگ تنک را می‌توان یک مدل گرافیکی جهتدار دانست که متغیرهای پنهان حقیقی دارد. در آن، تنک بودن با یک تنظیم‌کننده L_1 بر روی ضرایب کنترل می‌شود. با این حال استنتاج پسین (یعنی بدست آوردن تخمین احتمال پسین بیشینه[۳٤] یا MAP) به شرط داده‌های ورودی، محاسبات سنگینی دارد چرا که نیازمند حل یک مساله بهینه‌سازی L_1 است. در مقابل، RBM تنک یک مدل گرافیکی بدون جهت است که متغیرهای پنهان باینری دارد. تنک بودن در آن مشخصاً با یک عبارت جریمه تنظیم‌کننده بر روی میانگین فعالیت واحدهای مخفی کنترل می‌شود. توزیع پسین در آن از لحاظ محاسباتی به سادگی محاسبه می‌شود چرا که واحدهای مخفی به شرط داده‌های ورودی، مستقل شرطی هستند.

کدینگ تنک بیشتر برای کاربردهایی مناسب است که نیازمند بازسازی خوب بوده و تنها یک لایه دارند. همچنین کدینگ تنک برای حالاتی که داده‌ها نویزی هستند بسیار مناسب است. همچنین می‌تواند برای تشخیص اشیاء به خصوص وقتی تنها یک لایه بازنمایی داریم، دقت مناسبی داشته باشد. در مقابل، RBM برای زمانی مناسب‌تر است که هدف ما رسیدن به یک بازسازی دقیق نباشد چرا که RBM دارای متغیرهای تصادفی باینری مخفی است. همچنین RBM زمانی مناسب است که بخواهیم از بازنمایی‌های سلسله‌مراتبی استفاده کنیم.

۲-۱-۵ دسته‌بندی به کمک ماشین بولتزمن محدود

یک RBM یک مدل احتمالاتی مولد و بدون جهت است که از یک لایه مخفی برای مدل کردن یک توزیع بر روی متغیرهای مشاهده‌پذیر استفاده می‌کند. این مدل‌ها اغلب بدین جهت آموزش داده می‌شوند تا تنها ورودی را مدل کرده و بتوانند در یک رویکرد بدون ناظر استخراج ویژگی انجام دهند (مانند کارهای مربوط به تشخیص صوت در [۲۲]). اما این توانایی در این مدل‌ها وجود دارد تا توزیع توام ورودی و دسته متناظرشان را مدل نمایند. این توزیع توام در شکل ۲-۷ نشان داده شده است.

[۳٤] Maximum A Posteriori probability estimate (MAP)



شکل ۲-۷ : ماشین بولتزمن محدود برای مدل کردن توزیع توام ورودی $\boldsymbol{v}$ و برچسب y . حالت واحدهای مخفی با $\boldsymbol{h}$ **نشان داده شده است.**
[۲۳]

یک RBM با g_h واحد مخفی، یک مدل پارامتریک از توزیع توام بین یک لایه از متغیرهای مخفی (که به آن ویژگی می‌گویند) $\boldsymbol{h} = (h_1, \dots, h_{g_h})$ و متغیرهای مشاهده‌پذیر که ورودی را می‌سازند $\boldsymbol{v} = (v_1, \dots, v_{g_v})$ و همچنین برچسب y می‌باشد که می‌توان تابع انرژی را برای آن به صورت زیر تعریف نمود [۲۳]:

$$E(\boldsymbol{y}, \boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^T \boldsymbol{W}\boldsymbol{h} - \boldsymbol{a}^T \boldsymbol{v} - \boldsymbol{b}^T \boldsymbol{h} - \boldsymbol{d}^T \boldsymbol{y} - \boldsymbol{h}^T \boldsymbol{U}\boldsymbol{y} \qquad (۴۶-۲)$$

که در آن $\boldsymbol{W}$ نشاندهنده ماتریس وزن بین واحدهای مشاهده‌پذیر و واحدهای مخفی است، $\boldsymbol{b}$ و $\boldsymbol{a}$ به ترتیب بردار بایاس برای واحدهای مخفی و مشاهده‌پذیر هستند و $\boldsymbol{U}$ و $\boldsymbol{d}$ نیز به ترتیب بردار بایاس برای واحدهای برچسب و ماتریس وزن بین واحدهای مخفی و برچسب می‌باشند. همچنین بردار برچسب به صورت $\boldsymbol{y} = (1_{y=i})_{i=1}^{c}$ برای c دسته مختلف تعریف می‌گردد. اگر ورودی v را باینری فرض نماییم، می‌توان به راحتی همانند قبل توزیع شرطی را برای داده‌ها نوشت. توزیع $P(v_i = 1|\boldsymbol{h})$ همان رابطه (۱۳-۲) بوده و توزیع مربوط به واحدهای لایه برچسب به دلیل آنکه باید یکی از آنها یک و بقیه صفر باشد، مطابق توضیحات مربوط به واحدهای نرم بیشینه در بخش ۲-۱-۳ به صورت زیر خواهد بود:

$$P(y|\boldsymbol{h}) = \frac{e^{d_y + \sum_j U_{jy} h_j}}{\sum_{y'} e^{d_{y'} + \sum_j U_{jy'} h_j}} \qquad (۴۷-۲)$$

بدین ترتیب واحدهای مخفی اطلاعات تخمینی درباره بردار ورودی و همچنین برچسب داده‌ها را در خود نگه می‌دارند. توزیع شرطی مربوط به واحدهای مخفی به شرط ورودی و برچسب داده‌ها یعنی $P(\boldsymbol{h}|y, \boldsymbol{v})$ دارای شکل زیر خواهد بود:

$$P(\boldsymbol{h}|y, \boldsymbol{v}) = \prod_j P(h_j|y, \boldsymbol{v}) \qquad (۴۸-۲)$$

$$P(h_j = 1|y, \boldsymbol{v}) = g\left(b_j + U_{jy} + \sum_i W_{ji} v_i\right)$$

که در آن $g(.)$ تابع سیگموئید می‌باشد.

۲-۱-۵-۱ دسته‌بندی در RBM با آموزش از نوع مولد[۳۵]

از آنجا که یک RBM یک توزیع بر روی تمام متغیرها را تعریف می‌کند، لذا روش‌های مختلفی برای آموزش آن وجود دارد. رایج‌ترین روش "آموزش مولد" نام دارد. با داشتن مجموعه آموزشی $\Gamma = \{(\boldsymbol{v}^{(i)}, y^{(i)})\}$ از N_Γ زوج داده آموزشی برچسب‌دار، می‌توان این مدل را با کمینه کردن منفی لگاریتم درستنمایی داده‌ها، آموزش داد:

$$\mathcal{L}_{gen} = -\sum_{i=1}^{N_\Gamma} \log P(\boldsymbol{v}^{(i)}, y^{(i)}) \tag{۲-۴۹}$$

برای کمینه کردن منفی لگاریتم درستنمایی در رابطه (۲-۴۹) از گرادیان نسبت به مدل می‌توان استفاده کرد. گرادیان دقیق عبارت درستنمایی برای هر یک از پارامترهای $\theta \in \{\boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{d}, \boldsymbol{U}\}$ **را می‌توان به صورت زیر نوشت:**

$$\frac{\partial \log P(\boldsymbol{v}^{(i)}, y^{(i)})}{\partial \theta} \tag{۲-۵۰}$$
$$= -\mathbb{E}_{h|\boldsymbol{v}^{(i)}, y^{(i)}} \left[\frac{\partial}{\partial \theta} E(\boldsymbol{v}^{(i)}, y^{(i)}, \boldsymbol{h})\right] + \mathbb{E}_{\boldsymbol{v}, y, \boldsymbol{h}} \left[\frac{\partial}{\partial \theta} E(\boldsymbol{v}, y, \boldsymbol{h})\right]$$

در اینجا نیز همانند قبل محاسبه بخش اول نتیجه رابطه (۲-۵۰) قابل محاسبه است اما قسمت دوم محاسبه ناپذیر است. خوشبختانه یک تخمین خوب از گرادیان به نام واگرایی متقابل یا CD برای این رابطه قابل استفاده است که نحوه عملکرد این روش در بخش ۲-۱-۱ توضیح داده شده است. این تخمین قسمت دوم را با یک نمونه تولید شده بعد از اجرای تعداد محدودی از نمونه‌برداری گیبز جایگزین می‌کند که حالت اولیه آن با مقدار نمونه آموزشی ($(\boldsymbol{v}^{(i)}, y^{(i)})$) مقداردهی شده است.

محاسبه $P(y, \boldsymbol{v})$ محاسبه ناپذیر است اما می‌توان $P(y|\boldsymbol{v})$ را با گرفتن نمونه‌هایی از آن یا انتخاب دسته با بیشترین احتمال محاسبه نمود. همانطور که در [۲٤] نشان داده شده است، برای تعداد c دسته، توزیع شرطی را می‌توان به طور دقیق و کارآ به صورت زیر محاسبه نمود:

$$P(y|\boldsymbol{v}) = \frac{e^{d_y} \prod_{j=1}^{g_h} (1 + e^{b_j + U_{j,y} + \sum_i W_{j,i} v_i})}{\sum_{y'} e^{d_{y'}} \prod_{j=1}^{g_h} (1 + e^{b_j + U_{j,y'} + \sum_i W_{j,i} v_i})} \tag{۲-۵۱}$$

با محاسبه عبارت $b_j + \sum_i W_{j,i} v_i$ و استفاده مجدد آن در ضرب زیر

$$\prod_{j=1}^{g_h} (1 + e^{b_j + U_{j,y'} + \sum_i W_{j,i} v_i}) \tag{۲-۵۲}$$

برای همه برچسب‌های y' محاسبه این توزیع شرطی دارای بار محاسباتی $O(g_h d + g_h c)$ خواهد بود.

[۳۵] Generative training

۲-۱-۵-۲ دسته‌بندی در RBM با آموزش تمایزی[۳۶]

با این حال در حالت دسته‌بندی ممکن است فقط دسته‌بندی درست مد نظر باشد و نیازی به $P(\boldsymbol{v})$ مناسب نداشته باشیم. از آنجا که فرض مدلسازی $P(\boldsymbol{v})$ به طور ضمنی در RBM وجود دارد و می‌تواند نادرست باشد، لذا مفید است تا به طور مستقیم به بهینه‌سازی $P(y|\boldsymbol{v})$ به جای $P(y|\boldsymbol{v})$ بپردازیم و به سراغ کمینه کردن رابطه هزینه زیر برویم:

$$\mathcal{L}_{disc} = -\sum_{i=1}^{N_\Gamma} \log P\left(y^{(i)}|\boldsymbol{v}^{(i)}\right) \qquad (۵۳-۲)$$

این رویکرد آموزش را آموزش تمایزی می‌نامند و به RBMهایی که به این روش آموزش دیده‌اند، ماشین بولتزمن محدود تمایزی یا DRBM[۳۷] می‌گویند.

یک DRBM را می‌توان همچون گذشته و با واگرایی متقابل آموزش داد ولی از آنجا که $P(y|\boldsymbol{v})$ را می‌توان به طور دقیق محاسبه نمود، لذا گرادیان دقیق از رابطه زیر محاسبه می‌شود:

$$\frac{\partial \log P\left(y^{(i)}|\boldsymbol{v}^{(i)}\right)}{\partial \theta} \qquad (۵۴-۲)$$

$$= \sum_j g_{\cdot}\left(o_{y^{(i)},j}\left(v^{(i)}\right)\right) \frac{\partial\left(o_{y^{(i)},j}\left(v^{(i)}\right)\right)}{\partial \theta}$$

$$- \sum_{j,y'} g_{\cdot}\left(o_{y',j}\left(v^{(i)}\right)\right) p\left(y'|v^{(i)}\right) \frac{\partial\left(o_{y',j}\left(v^{(i)}\right)\right)}{\partial \theta}$$

که در آن $o_{y,j}(v) = b_j + \sum_k W_{j,k} v_k + U_{j,y}$ می‌باشد. این گرادیان را می‌توان به طور دقیق محاسبه و در بهینه‌سازی کاهش گرادیان استفاده نمود [۲۵].

۲-۱-۵-۳ دسته‌بندی در RBM با آموزش ترکیبی تمایزی/مولد[۳۸]

مزیت بدست آمده با آموزش تمایزی غالبا وابسته به میزان داده آموزشی در دسترس می‌باشد. مجموعه‌های آموزشی کوچکتر بیشتر به آموزش مولد گرایش داشته و مجموعه‌های بزرگتر به آموزش تمایزی علاقمند هستند. بدین ترتیب به جای آنکه تنها به یکی از این رویکردها اهمیت دهیم، می‌توان از روش ترکیبی تمایزی/مولدی با استفاده از معیارهای آموزش هر دو رویکرد، بهره برد. در این روش از معیار زیر استفاده می‌کنیم [۲۶]:

---

[۳۶] Discriminative training

[۳۷] Discriminative RBM (DRBM)

[۳۸] Hybrid discriminative\generative training

$$\mathcal{L}_{hybrid}(\Gamma) = \mathcal{L}_{disc}(\Gamma) + \alpha\mathcal{L}_{gen}(\Gamma) \qquad (۵۵-۲)$$

که در آن وزن معیار مولدی با پارامتر α کنترل می‌شود. در اینجا معیار مولد را می‌توان به عنوان یک تنظیم‌کننده وابسته به داده برای یک DRBM در نظر گرفت. برای آموزش یک DRBM در این رویکرد می‌توان از کاهش گرادیان آماری به همراه گرادیان محاسبه شده توسط $\mathcal{L}_{disc}$ با ضریب α استفاده نمود.

در صورتی که بخواهیم واحدهای مشاهده‌پذیر v مقادیر حقیقی بگیرند، از واحدهای گوسی مشابه آنچه در بخش ۲-۱-۱-۵ توضیح داده شد، می‌توان استفاده نمود. با این کار توزیعات شرطی روابط (۲-۴۷)، (۲-۴۸) و (۲-۵۱) بدون تغییر باقی مانده و تنها $P(v_i|\boldsymbol{h})$ همانند رابطه (۲-۲۵) یا (۲-۳۰) محاسبه می‌شود [۲۳].

بر اساس مقایسات انجام شده بهترین کارآیی مربوط به وقتی است که از هر دو رویکرد آموزشی یعنی آموزش ترکیبی تمایزی/مولد استفاده نماییم [۲۷].

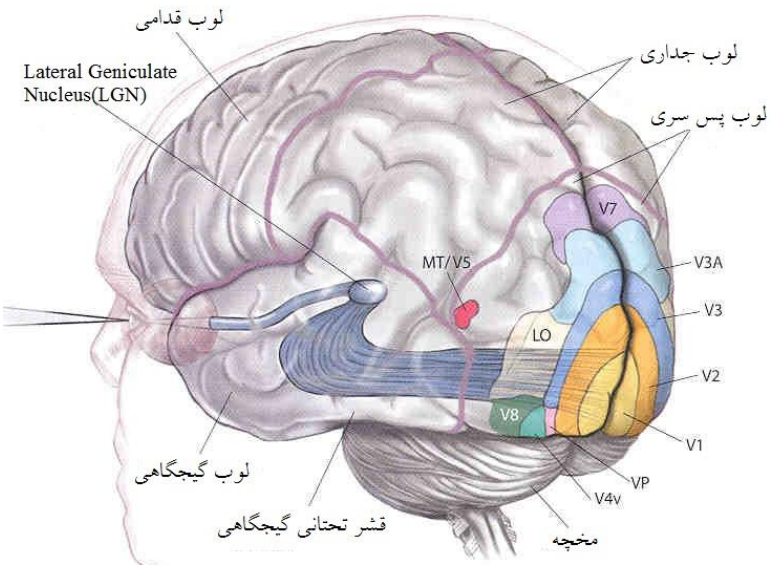
۲-۲ شبکه‌های باور عمیق

یکی از دلائل استفاده از معماری‌های عمیق برگرفته شده از علم نورولوژی است که انسان‌ها جهان را منطبق بر آن می‌بینند و درک می‌کنند. برای مثال مطابق شکل ۲-۸ اطلاعات تصویری از شبکیه به بخشی بنام "هسته خمیده کناری" یا LGN[۳۹] از تالاموس رفته و سپس به سمت قشر بینایی مخ در بخش پس‌سری[۴۰] می‌رود. قشر بینایی مخ که V1 هم نامیده می‌شود به شش لایه عملیاتی مجزا تقسیم می‌گردد که از یک تا شش نامگذاری شده است. لایه ٤ به نام لایه granular نیز خودش به چهار لایه با برچسب‌های 4A، 4B، $4C\alpha$ و $4C\beta$ تقسیم می‌شود. بخش V1 ویژگی‌های فضایی مانند جهت را حمل می‌کند و آنها را به V2 می‌فرستد. سپس V2 اتصالات قوی را به V3، V4 و V5 می‌فرستد و همچنین یک بازخوردی از آن را هم به V1 برمی‌گرداند. بنابراین حتی برای یک عمل بینایی ساده، ده‌ها لایه و حدود ۱٤۰,۰۰۰,۰۰۰ نورون درگیر هستند. در اینجا توجه و آگاهی از اشیاء که در تالاموس پردازش می‌شود، لحاظ نشده است [۱]. بررسی‌های تئوری نیز در مقایسه با معماری‌های کم عمق نشان می‌دهد که معماری‌های عمیق کارآمدتر است چون می‌تواند بیشتر توابع رایج به خصوص توابع یادگیری با تغییرات زیاد را به صورت کارآمد و کامل مدل کند.

شبکه‌های باور عمیق مدل‌های احتمالاتی مولد هستند که شامل تعداد زیادی لایه مخفی می‌باشند، که هر لایه ارتباطات مرتبه بالا بین تراکنش‌های متغیرهای مخفی در لایه زیرین را حفظ می‌کند.

[۳۹] Lateral Geniculate Nucleus

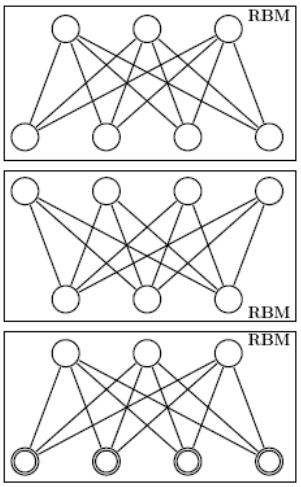
[۴۰] occipital lobe



شکل ۲-۸ : تصویری از قسمت‌های مختلف مغز مربوط به بینایی

قسمت اصلی شبکه باور عمیق یک مدل گرافیکی بدون جهت دوبخشی است که به آن ماشین بولتزمن محدود می‌گویند. در بخش‌های قبل مروری بر ماشین‌های بولتزمن محدود داشتیم و در ادامه الگوریتم آموزشی حریصانه برای شبکه‌های باور عمیق را بررسی خواهیم کرد. ویژگی کلیدی الگوریتم آموزش لایه به لایه حریصانه آن است که می‌تواند چندین بار تکرار شود تا مدل سلسله مراتب عمیقی را آموزش دهد.

ایده موجود در الگوریتم یادگیری حریصانه برای شبکه‌های باور عمیق نسبتا ساده است. الگوریتم یادگیری از پشته ماشین بولتزمن محدود مطابق شکل ۲-۹ استفاده می‌کند. ابتدا ماشین بولتزمن محدود زیرین را با پارامترهای $\mathbf{W^1}$ آموزش می‌دهد. سپس وزن‌های لایه دوم با $\boldsymbol{W^2 = W^{1^T}}$ مقداردهی اولیه می‌شوند تا مطمئن شویم که هر دو لایه مخفی شبکه باور عمیق حداقل به خوبی ماشین بولتزمن محدود اصلی هستند. بعد از استخراج مقادیر لایه h^1 می‌توان شبکه باور عمیق را با استفاده از این داده و اصلاح $\boldsymbol{W^2}$ بهتر کرد [۱۷]. البته در حالت کلی نیازی به یکسان گرفتن اندازه ماتریس وزن W^1 و W^2 نیست.



شکل ۲-۹: یادگیری حریصانه پشته‌ای از ماشین‌های بولتزمن محدود که نمونه‌های سطح پایین ماشین‌های بولتزمن محدود به عنوان داده برای آموزش ماشین بولتزمن محدود بعدی استفاده می‌شود [۱۷].

این ایده می‌تواند برای آموزش لایه سوم ماشین بولتزمن محدود روی بردار h^2 که از ماشین بولتزمن محدود دوم بدست آمده نیز استفاده شود. با مقداردهی $W^3 = W^{2^T}$ تضمین می‌کنیم که کران پایین لگاریتم درستنمایی را بهبود می‌دهیم. همچنین می‌توانیم با تغییر W^3 کران را بهتر کنیم. این روش لایه به لایه می‌تواند چندین بار تکرار شود تا یک مدل سلسله مراتبی عمیق بدست آید. این روش در الگوریتم زیر مشاهده می‌شود.

الگوریتم ۲-۳ : فرآیند یادگیری حریصانه بازگشتی در DBN [۱۷]

۱. **تعیین پارامترهای W^1 از اولین لایه *RBM* به داده**
۲. **ثابت کردن پارامتر W^1 و استفاده از نمونه‌های h^1 از $Q(h^1|v) = P(h^1|v;W^1)$ به عنوان داده برای آموزش ویژگی‌های باینری لایه بعد در یک *RBM***
۳. **ثابت کردن پارامتر W^2 که لایه دوم ویژگی‌ها را تعریف می‌کند و استفاده از نمونه‌های h^2 از $Q(h^2|h^1) = P(h^2|h^1;W^2)$ به عنوان داده برای آموزش ویژگی‌های باینری لایه سوم**
٤. **ادامه این کار به صورت بازگشتی برای لایه‌های بعد**

۲-۳ جمع‌بندی

شبکه‌های باور عمیق به عنوان یک ابزار قوی برای یادگیری ویژگی و طبقه‌بندی در حوزه‌های مختلف صوت و گفتار و همچنین پردازش تصویر مطرح می‌باشد. در این گزارش سعی شده به بررسی جنبه‌های مختلف موجود در این نوع شبکه پرداخته شود و در نهایت ابزاری برای استفاده از آن معرفی گردد.

در راستای معرفی شبکه‌هایی باور عمیق به سبب آن که این شبکه‌ها از عناصری به نام ماشین بولتزمن محدود ساخته شده‌اند، بیشتر اصلاحات انجام شده در مقالات نیز مربوط به این عنصر می‌باشد. انواع تغییرات در شبکه

بولتزمن محدود شامل استفاده از دسته‌های کوچک، نحوه مقداردهی اولیه وزن‌ها و بایاس، به کارگیری ممنتم، تضعیف وزن، ماشین بولتزمن محدود گوسی برای امکان استفاده از مقادیر حقیقی، ماشین بولتزمن محدود با واحدهای نرم بیشینه برای امکان استفاده از مقادیر صحیح، ماشین‌های بولتزمن محدود کانولوشنال برای امکان بررسی داده‌ها و تصاویر حجیم و همچنین روش‌هایی جهت امکان طبقه‌بندی با کمک ماشین بولتزمن محدود بعضی از موارد مطرح شده برای اصلاح ماشین‌های بولتزمن محدود و در نتیجه شبکه‌های باور عمیق بودند که دراین بخش و بعضی در ضمیمه ۵-۱ بررسی گردیدند.

یکی دیگر از اصلاحات انجام شده در آموزش شبکه‌های باور عمیق مربوط به نحوه محاسبه گرادیان لگاریتم احتمال داده آموزشی است. به سبب محاسبه‌ناپذیر بودن این گرادیان از روش‌های تخمین گرادیان برای این منظور استفاده می‌گردد. لذا تخمین هرچه بهتر این گرادیان از چالش‌های موجود در این نوع شبکه است و به همین علت روش‌های مختلفی برای بهبود آن ارائه شده است. روش واگرایی متقابل، روش واگرایی متقابل پایدار، روش واگرایی متقابل پایدار با هموارسازی جزئی و روش واگرایی متقابل پایدار سریع از مواردی است که در این گزارش به بررسی و توضیح آن‌ها پرداخته شد. تقریبا در همه این روش‌ها هدف یافتن یک نمونه بهتر از مدل است تا نیاز به اجرای زیاد نمونه‌برداری گیبز کاهش یابد.

اما یکی دیگر از اصلاحات انجام شده تنک‌سازی واحدهای مخفی بود. به سبب مزایایی که از تنک شدن واحدها به دست می‌آید دو روش برای آن مطرح گردید. در نهایت هر دو روش به نتیجه مشابهی برای یادگیری می‌رسیدند. در هر دو روش بعد از یک مرحله آموزش با استفاده از گرادیان تخمین زده شده از لگاریتم احتمال داده آموزشی در یک مرحله نیز به اصلاح پارامترها برای هدف تنک‌سازی می‌پردازند.

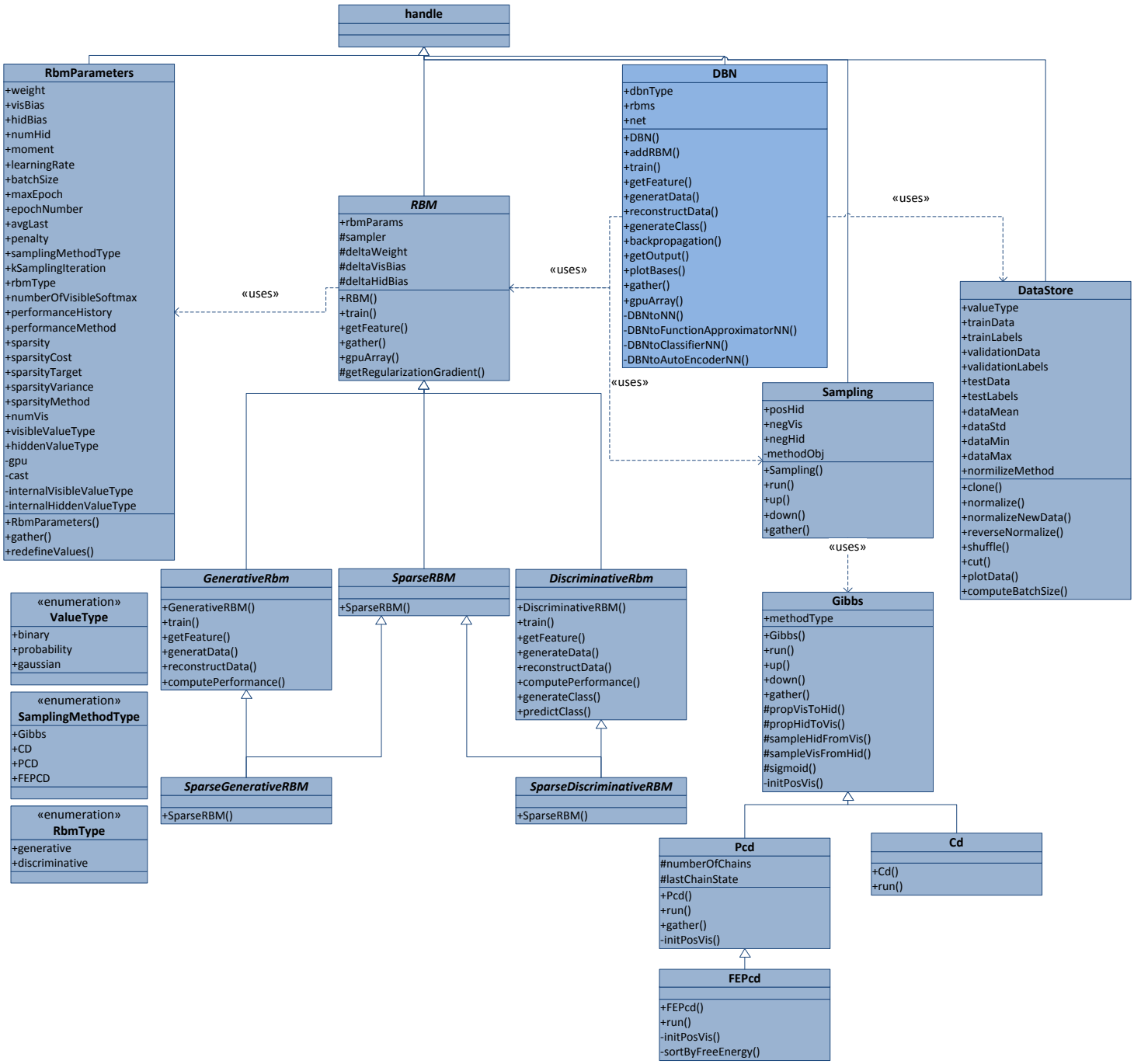
در ادامه به بررسی جعبه ابزار طراحی شده برای این مدل می‌پردازیم و توابع و کلاس‌های تعریف شده برای آن را معرفی خواهیم کرد. همچنین با آوردن مثال‌هایی از این جعبه ابزار و نمایش خروجی‌های آن، بعضی از قابلیت‌های آن به تصویر کشیده شده است.

۳ جعبه‌ابزار DeeBNet[۴۱] برای کار با شبکه‌های باور عمیق

این جعبه ابزار دارای دو پوشه به صورت بسته[۴۲] یا پکیج است که دارای تعدادی کلاس و تابع برای کار با داده‌ها و همچنین نمونه‌برداری است و در کنار آن تعدادی کلاس برای تعریف ماشین بولتزمن محدود و شبکه باور عمیق وجود دارد. در ادامه به توضیح این بسته‌ها و همچنین کلاس‌های تعریف شده در این جعبه ابزار خواهیم پرداخت. در شکل زیر نحوه ارتباط بین کلاس‌ها نشان داده شده است.

[۴۱] Deep Belief Network

[۴۲] Package



شکل ۳-۱ : نمایش نحوه ارتباط بین کلاس‌ها در جعبه ابزار DeeBNet

۳-۱ کلاس ValueType

این کلاس که در واقع یک شمارنده یا enumeration است، نوع واحدهای موجود در یک شبکه باور عمیق را تعریف می‌کند. انواع تعریف شده در این کلاس عبارتند از:

- نوع binary: واحدها تنها مقدار 0 یا 1 می گیرند.
- نوع probability : واحدها تنها مقادیر حقیقی در بازه [0,1] را می‌گیرند.

- نوع gaussian: واحدها هر مقدار حقیقی با میانگین ۰ و واریانس ۱ می‌توانند بگیرند.

مثال	**در این کد نوع مقادیر موجود در شیء data از نوع گوسی (یعنی هر مقداری با میانگین صفر و واریانس ۱) تعریف شده است.** `data.valueType=ValueType.gaussian;`

۳-۲ کلاس RbmType

این کلاس که در واقع یک شمارنده یا enumeration است، نوع RBM را تعریف می‌کند. انواع تعریف شده در این کلاس عبارتند از:

- نوع generative: نوعی از RBM که به برچسب داده‌ها کاری نداشته و در نهایت یک مدل از نوع مولد برای ما می‌سازد.
- نوع discriminative: نوعی از RBM که با برچسب داده‌ها یک مدل discriminative را ساخته و می‌توان با آن داده‌ها را دسته‌بندی نمود. در این نوع از RBM از واحدهای softmax برای افزودن امکان دسته‌بندی به RBM استفاده می‌شود. (بخش ۲-۱-۵)

مثال	**در این کد نوع RBM در پارامترهای آن از نوع discriminative تعریف شده است.** `rbmParams.rbmType=RbmType.discriminative;`

۳-۳ کلاس RbmParameters

این کلاس دارای ویژگی‌هایی است که تمامی اطلاعات مربوط به یک ماشین بولتزمن محدود را در خود دارد. در شکل زیر این ویژگی‌ها مشاهده می‌شود. بسیاری از این پارامترها و توضیح مربوط به آن‌ها در [۱۶] بیان شده است.

شکل ۳-۲: اجزای تشکیل دهنده کلاس RbmParameters

پارامتر weight یک ماتریس دو بعدی است که وزن‌های لایه مشاهده‌پذیر به لایه پنهان را نشان می‌دهد. تعداد ردیف‌های این ماتریس دو بعدی به تعداد واحدهای مشاهده‌پذیر و تعداد ستون‌های آن به تعداد واحدهای مخفی است. پارامترهای visBias و hidBias نیز بایاس‌های واحدهای مشاهده‌پذیر و مخفی را نشان می‌دهد. پارامتر numHid نیز نشاندهنده تعداد پارامترهای مخفی است.

پارامتر moment و learningRate به ترتیب مربوط به ممنتم و نرخ یادگیری در فرآیند یادگیری است. پارامتر batchSize اندازه نمونه‌های موجود در هر batch را در فرآیند محاسبه گرادیان و اصلاح وزن‌ها و بایاس نشان می‌دهد. همچنین پارامتر maxEpoch نشاندهنده ماکزیمم تعداد گام‌هایی است که در فرآیند آموزشی بر روی کل داده آموزشی اجرا می‌گردد.

پارامتر epochNumber شماره epoch فعلی را نشان می‌دهد و اگر چندبار RBM آموزش داده شود، این مقدار از حالت قبلی شمارش خود را ادامه می‌دهد. همچنین پارامتر avgLast نیز مربوط به نوعی میانگین‌گیری در پارامترهای مدل در چند Epoch آخر است که این تعداد با پارامتر avgLast مشخص می‌شود. این ایده در پایان‌نامه آقای Kevin Swersky مطرح شده است [۲۸].

پارامتر penalty مربوط به جریمه‌ای است که به بزرگ شدن اندازه وزن‌ها داده می‌شود. پارامتر samplingMethodType نشان‌دهنده روش نمونه‌برداری همچون روش گیبز و یا CD است و در نهایت پارامتر kSamplingIteration تعداد دفعاتی که نمونه‌برداری برای تولید یک نمونه اجرا می‌شود را نشان می‌دهد.

پارامتر بعدی یعنی rbmType نشاندهنده نوع RBM بوده و مقدار آن با توجه به بخش ۳-۲ تعیین می‌شود. همچنین پارامتر numberOfVisibleSoftmax نشاندهنده تعداد عناصر softmax در ساخت RBM های تمایزی است که توضیح آن در بخش ۲-۱-۵ آمده است.

پارامترهای performanceHistory و همچنین PerformanceMethod نیز به ترتیب نگهدارنده تغییر کارآیی در RBM با افزایش هر epoch و روش محاسبه کارآیی است. روش‌های مختلف محاسبه کارآیی در اینجا عبارتند از روش استفاده از اندازه انرژی آزاد، امکان بازسازی داده و همچنین دقت دسته‌بندی. البته روش آخر فقط در RBM های تمایزی قابل استفاده است.

پارامترهای sparsity، sparsityCost، sparsityTarget، sparsityVariance و sparsityMethod مربوط به تنظیم RBM برای امکان استفاده از تنک بودن در آن است که به ترتیب نشاندهنده استفاده از تنک سازی، میزان تاثیر تنک سازی، مقدار هدف در میزان فعالیت واحدهای مخفی، واریانس در روش تنک سازی نرمال و روش تنک سازی است. توضیح بیشتر این پارامترها در بخش ۲-۱-٤ آمده است.

همچنین پارامتر GPU نشاندهنده استفاده کردن و یا نکردن از GPU در محاسبات است که با تنظیم این پارامتر در پارامترهای RBM، بیشتر محاسبات در GPU با سرعتی بسیار بالاتر انجام می‌شود. پارامتر cast نیز که مقدار @double و یا @single می‌گیرد، نشاندهنده آن است که محاسبات در CPU و یا GPU با چه دقت اعشاری انجام شود.

سه پارامتر numVis، visibleValueType و hiddenValueType که به ترتیب مربوط به تعداد عناصر قابل مشاهده، نوع واحدهای قابل مشاهده و نوع واحدهای مخفی است از نوع Dependent می‌باشند. این بدان معنا است که مقادیر این پارامترها به صورت تابع مقداردهی و یا بازیابی می‌شوند. نام توابع مربوط به این عناصر در زیر آمده است. این توابع نمی‌توانند از خود پارامترهای تعریف شده به عنوان متغیر داده استفاده کنند و تنها یک نام مجازی هستند و لذا برای visibleValueType و hiddenValueType پارامترهای متناظر دیگری به صورت

خصوصی تعریف شده است (internalVisibleValueType و internalHiddenValueType) تا مقادیر این پارامترها را در خود نگه دارند.

```
function numVis=get.numVis(obj)
function visibleValueType=get.visibleValueType(obj)
function set.visibleValueType(obj,value)
function hiddenValueType=get.hiddenValueType(obj)
function set.hiddenValueType(obj,value)
```

تابع سازنده این کلاس به صورت زیر تعریف شده است.

```
function obj=RbmParameters(numHid,hiddenValueType)
```

آرگومان‌های ورودی تابع سازنده این کلاس تعداد واحدهای مخفی و نوع آن را مشخص می‌کند. در صورتی که مقداری برای این دو آرگومان مشخص نشده باشد، به صورت پیش فرض تعداد ۱۰۰ واحد مخفی از نوع باینری برای لایه مخفی در نظر گرفته می‌شود. همچنین برای پارامترهای دیگر نیز مقادیر پیش فرضی در نظر گرفته شده که از جمله آن تعیین الگوریتم CD به عنوان روش نمونه‌برداری است.

```
function [obj]=gather(obj)
```

این تابع برای استخراج مقادیر پارامترهای این شئ از حافظه GPU به حافظه محلی استفاده می‌شود. لذا در صورتی که قصد دارید DBN شاخته شده به کمک GPU را در کامپیوتر دیگری بدون GPU استفاده کنید، باید ابتدا آن را به کمک تابع gather به حالت عادی تبدیل نمایید.

```
function []=redefineValues(obj)
```

این تابع که قبل از شروع یادگیری صدا زده می‌شود، امکان استفاده از GPU و همچنین cast مورد نظر را بر روی پارامترها اعمال می‌نماید.

البته بعد از ساخت یک شئ از نوع RbmParameters، می‌توان مقادیر پارامترهای آن را همچون مثال زیر تغییر داد.

```
rbmParams=RbmParameters(250,ValueType.bionomial);
rbmParams.batchSize=150;
rbmParams.learningRate=0.01;
rbmParams.rbmType=RbmType.discriminative;
rbmParams.performanceMethod='classification';
```

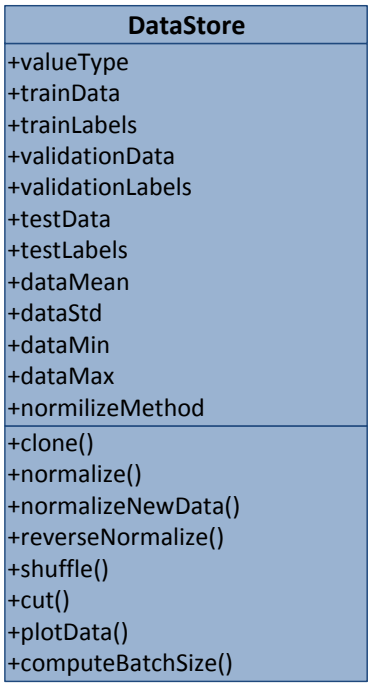
۳-٤ بسته DataClasses

در این بسته توابع و کلاس‌های مربوط به کار با داده‌ها پیاده‌سازی شده است. گذاشتن + در ابتدای نام این پوشه موجب می‌شود تا این پوشه به عنوان یک بسته یا پکیج در MATLAB شناخته شود و دیگر نیازی به اضافه

کردن مسیر این فولدر به صورت مجزا نباشد. در این بسته یک کلاس برای کار با داده‌ها قرار داده شده است که در ادامه به توضیح آن خواهیم پرداخت.

۳-٤-۱ کلاس DataStore

این کلاس جهت نگهداری مقادیر داده‌های آموزشی، تست و ارزیابی استفاده می‌شود. این کلاس از کلاس handle ارث می‌برد تا بتوانیم از یکی از ویژگی‌های مهم این کلاس بهره ببریم. با کمک این ویژگی در صورتی که یکی از آرگومان‌های یک تابع از نوع کلاس DataStore باشند، شئ جدیدی برای آن ساخته نشده و از همان شئ ساخته شده قبل استفاده می‌کند. بدین ترتیب با توجه به اینکه اشیاء مربوط به این کلاس داده‌های زیادی را در خود دارد و فضای حافظه زیادی اشغال می‌نماید، این ویژگی موجب استفاده بهینه‌تر در کار با داده‌ها را برای ما فراهم می‌کند. در شکل زیر اجزای مربوط به کلاس DataStore مشاهده می‌شود.



شکل ۳-۳ : اجزای تشکیل دهنده کلاس DataStore

در این کلاس ویژگی valueType نوع مقادیر داده‌ها در این کلاس را مشخص می‌نماید. این ویژگی از یک enumeration به نام ValueType مقدار می‌گیرد و می‌تواند مقادیری همچون باینری یا گوسی باشد. در بخش ۳-۱ کلاس ValueType معرفی شده است.

داده‌ها و همچنین برچسب‌های مربوط به آن‌ها در شش ویژگی بعدی قرار می‌گیرند. نکته قابل توجه در این ویژگی‌ها آن است که داده‌ها به صورت یک ماتریس m*n بعدی است که m تعداد نمونه‌های موجود در آن

ماتریس می‌باشد. همچنین ویژگی برچسب‌ها نیز یک بردار m بعدی است که مقادیر آن اعداد صحیح بزرگتر و مساوی صفر است که دسته داده‌های متناظر را مشخص می‌کند.

اما چهار ویژگی دیگر مربوط به زمانی است که با کمک تابع normalize داده‌ها را نرمال می‌کنیم. در واقع با نگهداری مقدار انحراف معیار و میانگین در روش meanvar (در ویژگی‌های dataMean و dataStd) و نگهداری ماکزیمم و مینیمم در روش minmax (در ویژگی‌های dataMin و dataMax) به اشیاء این کلاس اجازه می‌دهیم تا بتوانند داده‌های جدیدی را نیز نرمالیزه کرده و همچنین داده‌های نرمال شده را به حالت اول خود برگردانند.

```
function DataStoreObj=clone(obj)
```

کاربرد تابع clone در ساخت یک شئ همانند شئ موجود است. از آنجا که کلاس DataStore از نوع handle است لذا اگر تغییری در آن در هر جایی که به صورت آرگومان صدا زده شده، بوجود آید شئ اصلی تغییر می کند. لذا در جایی که بخواهیم می‌توانیم قبل از تغییر یک کپی از شئ را گرفته و در کپی بوجود آمده تغییرات لازم را بوجود آوریم.

```
function normalize(obj,method)
```

تابع normalize جهت نرمال‌سازی مقادیر داده‌های آموزشی، تست و ارزیابی استفاده می‌شود. در روش minmax مقادیر داده‌ها به بازه بین ۰ و ۱ نگاشت پیدا می‌کنند. در ضمن بعد از نرمالسازی به خاطر وجود مقادیر در بازه ۰ و ۱ مقدار valueType به نوع probability تبدیل می‌شود. همچنین در روش meanvar مقادیر واحدها دارای میانگین صفر و واریانس یک خواهد بود و مقدار valueType به نوع gaussian تبدیل خواهد شد.

```
function normalizedData=normalizeNewData(obj,data)
```

تابع normalizeNewData جهت نرمال‌سازی داده‌های جدید استفاده می‌شود. در واقع با توجه به نوع نرمال‌سازی استفاده شده در داده‌های آموزشی، داده‌های جدید نیز نرمال می‌شوند. داده‌های موجود در data نیز یک ماتریس m*n است که m تعداد نمونه‌های تست و n تعداد ابعاد ورودی است.

```
function deNormalizedData=reverseNormalize(obj,dataMatrix)
```

تابع reverseNormalize برای برگرداندن داده‌های نرمال شده به وضعیت اولیه آن است. با کمک این تابع با توجه به شیوه نرمال شدن داده‌ها، می‌توان مجددا داده‌ها را به شکل اولیه خود برگرداند. ماتریس dataMatrix نیز یک ماتریس m*n است که m تعداد نمونه‌های تست و n تعداد ابعاد ورودی است که نرمال شده است.

```
function shuffle(obj)
function cut(obj,cutRatio)
```

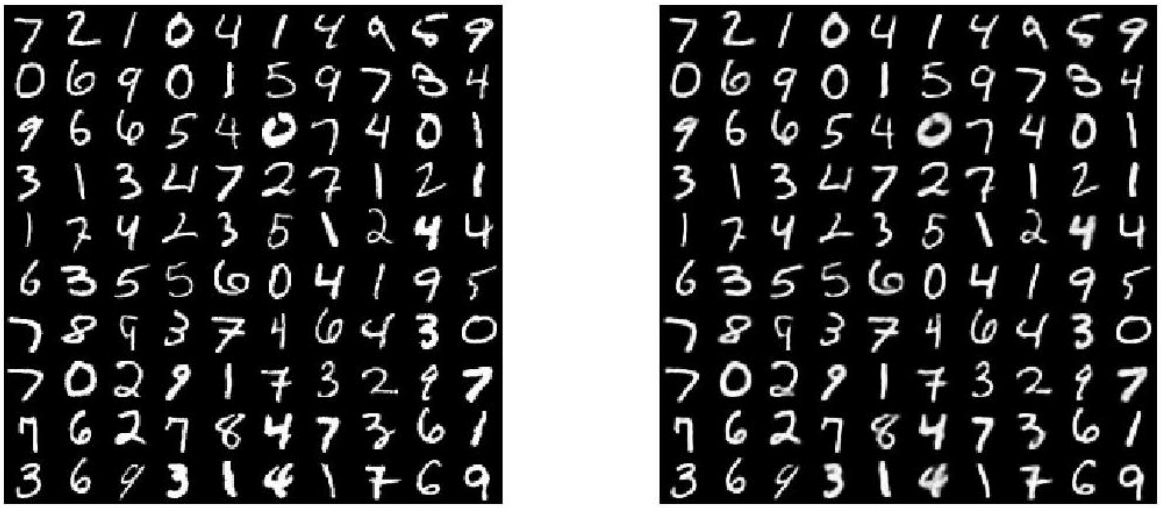
تابع shuffle تنها داده‌های آموزشی را به صورت رندوم پخش می‌کند. تابع cut نیز تنها داده‌های آموزشی را متناسب با مقدار آرگومان خود، کوچک می‌کند. به عنوان مثال اگر مقدار ورودی تابع cut برابر ٤ باشد بدین معنی است که تنها یک چهارم ابتدای داده ها آموزشی نگه داشته شده و بقیه دور ریخته می‌شود. کاربرد این تابع جهت کوچک سازی داده آموزشی برای اجرای سریعتر الگوریتم‌های یادگیری است.

```
function batchSize=computeBatchSize(obj)
```

این تابع برای محاسبه یک مقدار مناسب برای اندازه بلوک‌های مورد استفاده برای نرمال کردن و همچنین shuffle کردن داده‌ها از روی حجم دادگان و حافظه خالی سیستم می‌باشد.

```
function plotData(dataCells,isInversed,sizeX,sizeY)
```

تابع آخر نیز که از نوع Static بوده و لذا بدون ساختن شئ از کلاس مربوطه، قابل استفاده است، تابع plotData می‌باشد. این تابع آرایه‌ای سلولی از داده‌ها را گرفته و در یک تصویر در کنار هم نمایش می‌دهد. در واقع هر یک از سلول‌های آرگومان ورودی دارای یک ماتریسی از نمونه‌هایی است که برای نمایش به این تابع ارسال شده است. کاربرد این تابع بیشتر برای مواقعی است که بخواهیم نتیجه یک پردازش مشخص همچون بازسازی داده‌ها توسط شبکه باور عمیق را با داده اصلی مقایسه نماییم. نمونه‌ای از شکل رسم شده توسط این تابع را در زیر مشاهده می‌نمایید.



1 2

شکل ۳-٤: نمایش ۱۰۰ نمونه داده به کمک تابع plotData. در اینجا تصویر اول مربوط به ۱۰۰ نمونه از دادگان MNIST و تصویر دوم مربوط به تصاویر بازسازی شده متناظر آن‌ها است که هر یک از این ۱۰۰ نمونه و بازسازی شده آن در یک سلول مجزا به تابع ارسال شده است. کد مربوطه در فایل test_plotData.m قرار دارد.

گاهی اوقات داده‌های تصویری باید برای رسم درست یک چرخش داشته باشند (همچون داده‌های MNIST) که آرگومان ورودی isInversed این مشخصه را تعیین می‌کند. همچنین از آنجا که نمونه داده‌های استفاده شده در شبکه باور عمیق به صورت یک بردار ظاهر می‌شوند (مثل بردارهایی با ۷۸٤ بعد در MNIST) لذا در هنگام رسم باید اندازه‌های اصلی تصویر را نیز به تابع معرفی نمود (مثلا اندازه ۲۸*۲۸ در نمونه‌های MNIST) که آرگومان‌های sizeX و sizeY برای این منظور تعریف شده‌اند. البته در صورتی که مقداری برای این آرگومان‌ها مشخص نشده باشد، از مقدار جذر طول بردار نمونه‌ها برای این اندازه‌ها استفاده می‌شود.

۳-۵ بسته Sampling Classes

این بسته یا پکیج، شامل کلاس‌هایی از انواع روش‌های نمونه‌برداری است. در ادامه به توضیح موارد موجود در این بسته می‌پردازیم.

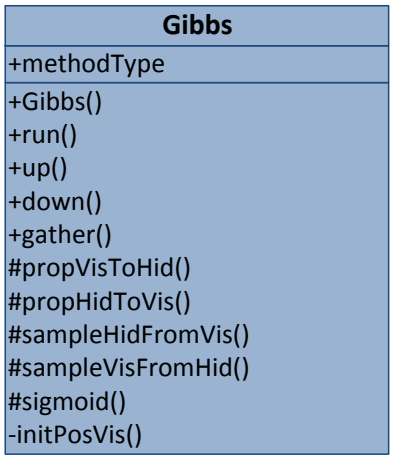
۳-۵-۱ کلاس SamplingMethodType

این کلاس یک نوع شمارنده یا enumeration است که روش نمونه‌برداری را در DBN مشخص می‌کند. انواع روش‌های نمونه‌برداری تعریف شده در این جعبه‌ابزار عبارتند از:

- روش Gibbs
- روش CD
- روش PCD
- روش FEPCD

۳-۵-۲ کلاس Gibbs

این کلاس جهت نمونه‌برداری از مدل RBM استفاده می‌شود. با کمک این کلاس می‌توان به روش نمونه‌برداری گیبز (بخش ۲-۱-۱)، از یک مدل RBM نمونه‌برداری نمود. در شکل ۳-۵ اجزای تشکیل دهنده کلاس Gibbs نمایش داده شده است.



شکل ۳-۵ : اجزای تشکیل دهنده کلاس Gibbs

در این کلاس ویژگی methodType قرار دارد که نوع روش نمونه‌برداری در آن می‌باشد.

```
function obj=Gibbs()
```

این تابع، سازنده کلاس می‌باشد. در این تابع مقدار ویژگی methodType برابر مقدار 'Gibbs' می‌شود.

```
function [obj,posHid,negVis,negHid]=run(obj,modelParams,posVis)
```
این تابع برای اجرای نمونه‌برداری به روش گیبز می‌باشد. ورودی‌های این تابع posVis و modelParams است که به ترتیب مربوط به نمونه‌های قابل مشاهده در مرحله اول اجرای الگوریتم (توضیح بیشتر در بخش ۲-۱-۱) و پارامترهای مدل می‌باشد. البته در روش گیبز نمونه‌های قابل مشاهده در مرحله اول به صورت تصادفی انتخاب می‌شود و posVis فقط برای نمونه‌های واحدهای مخفی فاز مثبت استفاده می‌گردد.

خروجی‌های مدل نیز به ترتیب عبارتند از مقادیر نمونه‌برداری شده برای واحدهای مخفی در مرحله اول و نمونه‌های بدست آمده در مرحله آخر برای واحدهای قابل مشاهده و مخفی که توضیح بیشتر آن در بخش ۲-۱-۱ آمده است.

```
function [hidSample,hidProb]=up(obj,modelParams,visSample)
```
این تابع مقادیر نمونه‌های قابل مشاهده و پارامترهای مدل را دریافت کرده و از روی آن نمونه‌های بدست آمده برای واحدهای مخفی و احتمال فعال شدن هر یک را بر می‌گرداند.

```
function [visSample,visProb]=down(obj,modelParams,hidSample)
```
این تابع نیز همچون تابع قبلی است با این تفاوت که نمونه‌های واحدهای مخفی را گرفته و نمونه‌های بدست آمده برای واحدهای قابل مشاهده و احتمال فعال شدن هر یک را بر می‌گرداند.

```
function [y]=sigmoid(~,x)
```
این تابع برای محاسبه تابع سیگموئید استفاده می‌شود. این تابع از نوع دسترسی محافظت شده (protected) می‌باشد و لذا کلاس‌هایی که از این کلاس ارث می‌برند می‌توانند از این تابع استفاده نمایند ولی در جاهای دیگر قابل دسترس نیست.

```
function [visInput]=propHidToVis(~,modelParams,hid)
```
این تابع برای محاسبه انتشار وزندار مقادیر واحدهای مخفی به واحدهای قابل مشاهده استفاده می‌شود. در واقع این تابع مقدار $a_i + \sum_j h_j w_{ij}$ را در رابطه (۲-۱۳) بدست می‌آورد.

```
function [hidInput]=propVisToHid(~,modelParams,vis)
```
این تابع برای محاسبه انتشار وزندار مقادیر واحدهای قابل مشاهده به واحدهای مخفی استفاده می‌شود. در واقع این تابع مقدار $b_j + \sum_i v_i w_{ij}$ را در رابطه (۲-۱۲) بدست می‌آورد.

```
function
[visSample,visProb]=sampleVisFromHid(obj,modelParams,visInput)
```
این تابع با توجه به مقادیر بدست آمده برای ورودی واحدهای قابل مشاهده، از روی احتمال فعال شدن هر واحد نمونه‌برداری انجام می‌دهد. بردار visInput ورودی هر واحد قابل مشاهده را طبق رابطه (۲-۱۳) نشان می‌دهد. خروجی این توابع نیز به ترتیب مربوط به نمونه بدست آمده برای واحدها و احتمال فعال شدن هر واحد می‌باشد. دراین تابع نمونه‌برداری بر اساس نوع واحدهای قابل مشاهده به صورت‌های متفاوتی انجام می‌شود.

```
function
[hidSample,hidProb]=sampleHidFromVis(obj,modelParams,hidInput)
```

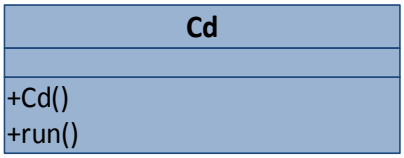
این تابع با توجه به مقادیر بدست آمده برای ورودی واحدهای مخفی، از روی احتمال فعال شدن هر واحد نمونه‌برداری انجام می‌دهد. بردار hidInput ورودی هر واحد مخفی را طبق رابطه (۲-۱۲) نشان می‌دهد. خروجی این توابع نیز به ترتیب مربوط به نمونه بدست آمده برای واحدها و احتمال فعال شدن هر واحد می‌باشد. در این تابع نمونه‌برداری بر اساس نوع واحدهای مخفی به صورت‌های متفاوتی انجام می‌شود.

```
function [posVis]=initPosVis(~,modelParams,posVis)
```

این تابع برای درست کردن مقادیر تصادفی اولیه برای نمونه‌برداری در روش گیبز استفاده می‌شود. ورودی این تابع posVis است که تنها برای بدست آوردن اندازه ماتریس مقادیر اولیه استفاده می‌شود. این تابع از نوع private تعریف شده و در نتیجه در جایی خارج از این کلاس در دسترس نمی‌باشد.

۳-۵-۳ کلاس Cd

این کلاس جهت نمونه‌برداری از مدل RBM استفاده می‌شود. با کمک این کلاس می‌توان به روش نمونه‌برداری واگرایی متقابل (Contrastive Divergence) (بخش ۲-۱-۱-۱)، از یک مدل RBM نمونه‌برداری نمود. در شکل ۳-۶ اجزای تشکیل دهنده کلاس Cd نمایش داده شده است.



شکل ۳-۶ : اجزای تشکیل دهنده کلاس Cd

این کلاس از کلاس Gibbs ارث می‌برد لذا بیشتر ویژگی‌ها و توابع آن را به ارث می‌برد و تنها توابع سازنده و run بازنویسی شده‌اند.

```
function obj=Cd()
```

این تابع، سازنده کلاس می‌باشد. در این تابع مقدار ویژگی methodType برابر مقدار 'Cd' می‌شود.

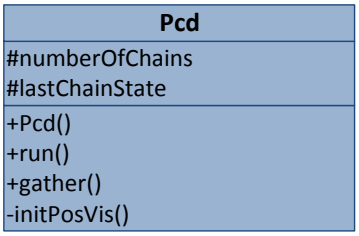
```
function [obj,posHid,negVis,negHid]=run(obj,modelParams,posVis)
```

این تابع برای اجرای نمونه‌برداری به روش واگرایی متقابل می‌باشد. ورودی‌های این تابع posVis و modelParams است که به ترتیب مربوط به نمونه‌های قابل مشاهده در مرحله اول اجرای الگوریتم (توضیح بیشتر در بخش ۲-۱-۱-۱) و پارامترهای مدل می باشد. در روش Cd نمونه‌های قابل مشاهده در مرحله اول از روی داده‌های آموزشی ارسال شده استفاده می‌کند.

خروجی‌های مدل نیز به ترتیب عبارتند از مقادیر نمونه‌برداری شده برای واحدهای مخفی در مرحله اول و نمونه‌های بدست آمده در مرحله آخر برای واحدهای قابل مشاهده و مخفی که توضیح بیشتر آن در بخش ۲-۱-۱ آمده است.

۳-۵-٤ کلاس Pcd

این کلاس جهت نمونه‌برداری از مدل RBM استفاده می‌شود. با کمک این کلاس می‌توان به روش نمونه‌برداری واگرایی متقابل پایدار (Persistent Contrastive Divergence) (بخش ۲-۱-۱-۲)، از یک مدل RBM نمونه‌برداری نمود. در شکل ۳-۷ اجزای تشکیل دهنده کلاس Pcd نمایش داده شده است.



شکل ۳-۷ : اجزای تشکیل دهنده کلاس Pcd

این کلاس از کلاس Gibbs ارث می‌برد لذا بیشتر ویژگی‌ها و توابع آن را به ارث می‌برد و برای آن توابع سازنده و run بازنویسی شده و دو ویژگی و یک تابع به آن اضافه گشته است. دو ویژگی اضافه شده در آن numberOfChains و lastChainState است. ویژگی اول نشاندهنده تعداد زنجیره‌های نمونه‌برداری PCD است و ویژگی دوم نگهدارنده زنجیره‌هایی است که در هر مرحله از نمونه‌برداری استفاده می‌شود (توضیح بیشتر در بخش ۲-۱-۱-۲).

```
function obj=Pcd()
```

این تابع، سازنده کلاس می‌باشد. در این تابع مقدار ویژگی methodType برابر مقدار 'Pcd' می‌شود.

```
function [obj,posHid,negVis,negHid]=run(obj,modelParams,posVis)
```

این تابع برای اجرای نمونه‌برداری به روش واگرایی متقابل پایدار می‌باشد. ورودی‌های این تابع posVis و modelParams است که به ترتیب مربوط به نمونه‌های قابل مشاهده در مرحله اول اجرای الگوریتم (توضیح بیشتر در بخش ۲-۱-۱-۲) و پارامترهای مدل می باشد. در روش Pcd نمونه‌های قابل مشاهده در مرحله اول برای بدست آوردن نمونه‌های واحدهای مخفی در posHid استفاده می‌شود اما برای بدست آوردن نمونه‌های مدل در ابتدا از مقادیر تصادفی شروع کرده و سپس در هر مرحله از خروجی مرحله قبل استفاده می‌کند.

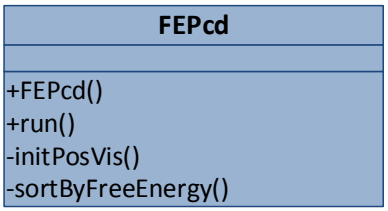
خروجی‌های مدل نیز به ترتیب عبارتند از مقادیر نمونه‌برداری شده برای واحدهای مخفی در مرحله اول و نمونه‌های بدست آمده در مرحله آخر برای واحدهای قابل مشاهده و مخفی که توضیح بیشتر آن در بخش ۲-۱-۱ آمده است.

```
function [posVis]=initPosVis(~,modelParams,numberOfChains)
```

این تابع مشابه تابعی با همین نام در کلاس Gibbs است با این تفاوت که به تعداد زنجیره‌های لازم نمونه تصادفی تولید می‌کند.

۳-۵-۵ کلاس FEPcd

این کلاس جهت نمونه‌برداری از مدل RBM استفاده می‌شود. با کمک این کلاس می‌توان به روش نمونه‌برداری انرژی آزاد در واگرایی متقابل پایدار (Free Energy in Persistent Contrastive Divergence) (بخش ۲-۱-۱-۲)، از یک مدل RBM نمونه‌برداری نمود. در شکل ۳-۸ اجزای تشکیل دهنده کلاس Pcd نمایش داده شده است.



شکل ۳-۸ : اجزای تشکیل دهنده کلاس FEPcd

این کلاس از کلاس Pcd ارث می‌برد لذا بیشتر ویژگی‌ها و توابع آن را به ارث می‌برد و برای آن توابع سازنده، run و initPosVis بازنویسی شده و یک تابع به آن اضافه گشته است. توضیح بیشتر این روش در بخش ۲-۱-۱-۲۲-۱-۱-۵ آمده است.

```
function obj=FEPcd()
```

این تابع، سازنده کلاس می‌باشد. در این تابع مقدار ویژگی methodType برابر مقدار 'FEPcd' می‌شود.

```
function [obj,posHid,negVis,negHid]=run(obj,modelParams,posVis)
```

این تابع برای اجرای نمونه‌برداری به روش انرژی آزاد در واگرایی متقابل پایدار می‌باشد. ورودی‌های این تابع posVis و modelParams است که به ترتیب مربوط به نمونه‌های قابل مشاهده در مرحله اول اجرای الگوریتم (توضیح بیشتر در بخش ۲-۱-۱-۵) و پارامترهای مدل می باشد. در روش FEPCD نمونه‌های قابل مشاهده در مرحله اول برای بدست آوردن نمونه‌های واحدهای مخفی در posHid استفاده می‌شود اما برای بدست آوردن نمونه‌های مدل در ابتدا از مقادیر تصادفی شروع کرده و سپس در هر مرحله از خروجی مرحله قبل استفاده می‌کند. تفاوت اصلی این روش با روش PCD این است که در این روش با استفاده از مقدار انرژی آزاد از زنجیره‌های مناسب‌تر برای بدست آوردن نمونه‌های مدل استفاده می‌کند.

خروجی‌های مدل نیز به ترتیب عبارتند از مقادیر نمونه‌برداری شده برای واحدهای مخفی در مرحله اول و نمونه‌های بدست آمده در مرحله آخر برای واحدهای قابل مشاهده و مخفی که توضیح بیشتر آن در بخش ۲-۱-۱ آمده است.

```
function [posVis]=initPosVis(~,modelParams,numberOfChains)
```

این تابع مشابه تابعی با همین نام در کلاس Pcd است با این تفاوت که چند برابر تعداد زنجیره‌های لازم نمونه تصادفی تولید می‌کند.

```
function [IX]=sortByFreeEnergy(~,modelParams,dataMatrix)
```

این تابع جهت محاسبه انرژی آزاد در نمونه‌های ارسال شده در ماتریس dataMatrix استفاده می‌شود. ماتریس dataMatrix شامل n داده به صورت ردیفی می‌باشد. خروجی این تابع بردار n بعدی IX است که اندیس داده‌های موجود در dataMatrix به ترتیب مقدار انرژی آزاد آن‌ها است به طوری که اندیس داده با کمترین انرژی آزاد در اولین مکان بردار IX قرار می‌گیرد.

۳-۵-۶ تابع freeEnergy

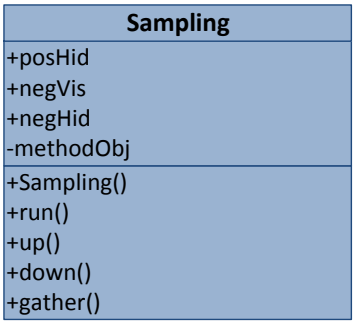
این تابع به صورت زیر در این بسته تعریف شده است.

```
function FE=freeEnergy(modelParams,dataMatrix)
```

در این تابع انرژی آزاد داده‌های موجود در dataMatrix که به صورت ردیفی در این ماتریس قرار دارد، محاسبه و در خروجی FE برگردانده می‌شود. نحوه محاسبه انرژی آزاد در بخش ۲-۱-۱-۵ توضیح داده شده است.

۳-۵-۷ کلاس Sampling

این کلاس در واقع یک کلاس واسط برای استفاده از کلاس‌های نمونه‌برداری است. اجزای استفاده شده در این کلاس در شکل ۳-۹ مشاهده می‌شود.



شکل ۳-۹ : اجزای تشکیل دهنده کلاس Sampling

در این کلاس سه ویژگی posHid، negVis و negHid به ترتیب مربوط به نمونه‌های بدست آمده برای واحدهای مخفی از روی داده‌های آموزشی، نمونه‌های بدست آمده برای واحدهای قابل مشاهده از مدل و نمونه-های بدست آمده برای واحدهای مخفی از مدل می‌باشد. ویژگی methodObj نیز دارنده شئ مربوط به نمونه‌برداری است که کلاس Sampling به کمک آن نمونه‌برداری انجام می‌دهد.

توابع موجود در این کلاس نیز مشابه توابعی است که در کلاس Gibbs تعریف شد با این تفاوت که در کلاس Sampling از توابع پیاده‌سازی شده در کلاس‌های نمونه‌برداری استفاده می‌کند.

```
function obj=Sampling(methodType)
```

این تابع، سازنده کلاس می‌باشد. در این تابع با توجه به مقدار ورودی methodType از کلاس متناظر آن یک شئ ساخته می‌شود.

```
function run(obj,modelParams,data)
```

این تابع برای اجرای نمونه‌برداری می‌باشد. ورودی‌های این تابع data و modelParams است که به ترتیب مربوط به نمونه‌های داده‌های آموزشی و پارامترهای مدل می باشد.

نمونه‌های بدست‌آمده در این تابع در ویژگی‌های این کلاس یعنی posHid، negVis و negHid قرار می‌گیرد.

```
function [hidSample,hidProb]=up(obj,modelParams,visSample)
```

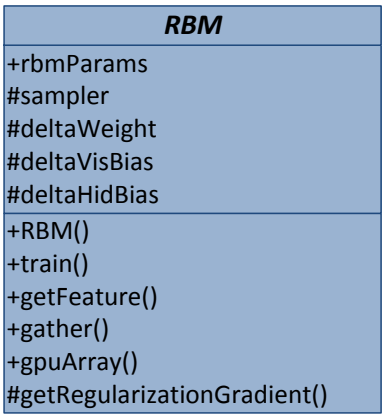
این تابع مقادیر نمونه‌های قابل مشاهده و پارامترهای مدل را دریافت کرده و از روی آن نمونه‌های بدست آمده برای واحدهای مخفی و احتمال فعال شدن هر یک را بر می‌گرداند.

```
function [visSample,visProb]=down(obj,modelParams,hidSample)
```

این تابع نیز همچون تابع قبلی است با این تفاوت که نمونه‌های واحدهای مخفی را گرفته و نمونه‌های بدست آمده برای واحدهای قابل مشاهده و احتمال فعال شدن هر یک را بر می‌گرداند.

۳-۶ کلاس RBM

این کلاس یک کلاس مجازی است و قالب تمامی انواع RBM را تعریف می‌کند. در واقع در این کلاس توابع و ویژگی‌های حداقلی که هر نوع کلاس RBM باید داشته باشد تا شبکه باور عمیق ساخته شود، تعریف شده است. در شکل ۳-۱۰ اجزای تشکیل دهنده این کلاس مشاهده می‌شود.



شکل ۳-۱۰ : اجزای تشکیل دهنده کلاس RBM

در این کلاس یک ویژگی عمومی (public) به نام rbmParams تعریف شده است. این ویژگی نگهدارنده همه پارامترهای یک RBM می‌باشد که تعریف آن در بخش ۳-۳ آمده است. چهار ویژگی بعدی که به صورت محافظت (protected) شده می‌باشند عبارتند از sampler، deltaWeigth، deltaVisBias و deltaHidBias. ویژگی

sampler دارای یک شئ از یکی از کلاس‌های نمونه‌برداری مثل Pcd است. ویژگی‌های deltaWeigth، deltaVisBias و deltaHidBias نیز برای اصلاح پارامترهای مدل RBM در زمان یادگیری مدل استفاده می‌شود.

```
function obj=RBM(rbmParams)
```

این تابع سازنده کلاس RBM است که در آن پارامترهای RBM را در ویژگی rbmParams قرار می‌دهد.

```
train(obj,trainData)
```

این تابع به صورت مجازی تعریف شده است تا همه انواع RBM تابع یادگیری را پیاده سازی نمایند.

```
[extractedFeature]=getFeature(obj,dataMatrix)
```

این تابع به صورت مجازی تعریف شده است تا همه انواع RBM تابع استخراج ویژگی را که در زمان یادگیری شبکه باور عمیق لازم می‌باشد را پیاده سازی نمایند.

```
function [obj]=gather(obj)
```

این تابع برای استخراج مقادیر پارامترهای این شئ از حافظه GPU به حافظه محلی استفاده می‌شود. لذا در صورتی که قصد دارید DBN شاخته شده به کمک GPU را در کامپیوتر دیگری بدون GPU استفاده کنید، باید ابتدا آن را به کمک تابع gather به حالت عادی تبدیل نمایید.

```
function obj=gpuArray(obj)
```

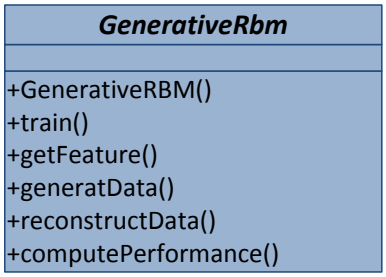
این تابع با تغییر بعضی از پارامترهای RBM امکان استفاده از GPU را در محاسبات برقرار می‌کند.

تابع مهم دیگری که در این بخش به صورت protected تعریف شده است getRegularizationGradient می‌باشد. در واقع در این تابع مقدار گرادیان برای همه ترم‌های Regularization تعریف شده در کنار تابع هدف محاسبه شده و برگردانده می‌شود. به صورت پیش‌فرض برای RBM های معمولی مقدار صفر برای این تابع برگردانده می‌شود ولی مثلا در RBM های تنک مقدار گرادیان روش تنک‌سازی برگردانده می‌شود.

```
[deltaWeightReg,deltaVisBiasReg,deltaHidBiasReg]=getRegularizationGradient(
obj,batchData,posHid)
```

۳-۷ کلاس GenerativeRBM

این کلاس یک نوع RBM است که می‌تواند یک مدل مولد (generative) از روی داده‌های آموزشی بدون برچسب بسازد. این کلاس از کلاس Rbm ارث می‌برد. اجزای این کلاس در شکل ۳-۱۱ آمده است.



شکل ۳-۱۱ : اجزای تشکیل دهنده کلاس GenerativeRBM

از آنجا که این کلاس از کلاس RBM ارث می‌برد لذا ویژگی‌های آن را دارا می‌باشد. علاوه بر این ویژگی‌ها توابعی را نیز در خود دارد که در ادامه به توضیح آن خواهیم پرداخت.

```
function obj=GenerativeRBM(rbmParams)
```

این تابع سازنده کلاس GenerativeRBM است که در آن علاوه بر صدا زدن تابع سازنده کلاس Rbm، rbmType را نیز از نوع generative مقداردهی می‌کند.

```
function train(obj,data)
```

این تابع مربوط به آموزش مدل RBM است که با کمک داده آموزشی گذاشته شده در data مدل را به صورت مولد آموزش می‌دهد. ورودی data در واقع یک شئ از نوع کلاس DataStore (بخش ۳-٤-۱) است. در این تابع بعد از مقداردهی اولیه پارامترهای مدل، با کمک داده آموزشی این پارامترها اصلاح می‌شوند. در حال حاضر شرط خاتمه در یادگیری، تعداد تکرار فرآیند آموزش (رسیدن به maxEpoch) می‌باشد. همچنین در حین آموزش اطلاعاتی درباره وضعیت فعلی یادگیری نمایش داده می‌شود. این اطلاعات شامل ساختار RBM در حال آموزش، شماره گام آموزش (epoch)، میزان کارآیی بر روی داده ارزیابی و تخمین زمان باقیمانده برای آموزش این مدل می‌باشد. در شکل ۳-۱۲ نمونه‌ای از نمایش این اطلاعات را برای یک RBM بر روی دادگان MNIST با ۷۸٤ ورودی و ۵۰۰ واحد مخفی که در۵ گام آموزش داده شده است را می‌بینید. در این RBM ، کارآیی با خطای بازسازی[٤٣] نمایش داده شده است.

```
****** RBM 1 ******* 784-500
epoch number:1   performance:0.0288815   remained RBM training time:153.802
epoch number:2   performance:0.0226055   remained RBM training time:138.783
epoch number:3   performance:0.0172301   remained RBM training time:102.35
epoch number:4   performance:0.0153325   remained RBM training time:51.7357
epoch number:5   performance:0.0132041   remained RBM training time:0
```

شکل ۳-۱۲ : خروجی آموزش یک RBM بر روی دادگان MNIST با ۷۸٤ ورودی و ۵۰۰ واحد مخفی که در ۵ گام (epoch) آموزش دیده شده است. کارآیی در اینجا به روش reconstruction می‌باشد.

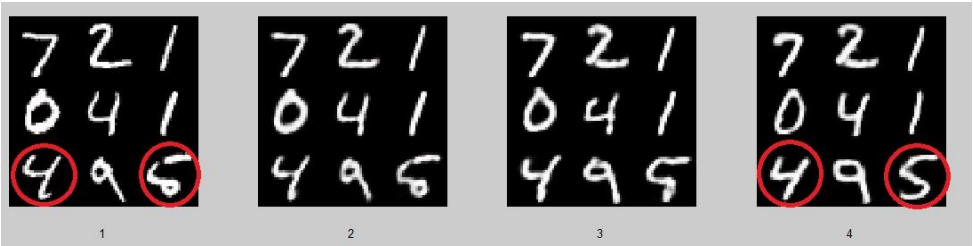
```
function [extractedFeature]=getFeature(obj,dataMatrix,k,fixedDimensions)
```

[٤٣] reconstruction

این تابع برای گرفتن ویژگی از داده ورودی dataMatrix استفاده می‌شود. ورودی dataMatrix یک ماتریس است که داده‌ها به صورت ردیفی در آن قرار دارد. این تابع نیز با روش نمونه‌برداری، نمونه‌های واحدهای مخفی که همان ویژگی‌های استخراج شده از داده ورودی است را بدست می‌آورد. پارامتر k نشاندهنده تعداد مراحل اجرای نمونه‌برداری برای رسیدن به ویژگی استخراج شده است و fixedDimensions برداری است که در آن ابعادی از داده ورودی را مشخص می‌کنیم که قصد داریم در حین نمونه‌برداری ثابت بمانند. این ورودی در مواقعی مفید خواهد بود که از صحت بعضی از ابعاد داده اطمینان داریم و قصد داریم تا در حین نمونه‌برداری ثابت بمانند تا کیفیت نهایی به سبب تغییر آنها در حین نمونه‌برداری لطمه نخورد. در نهایت نیز خروجی‌ها به صورت ردیفی در ماتریس extractedFeature برگردانده می‌شود.

```
function [generatedData]=generateData(obj,extractedFeature,k)
```

این تابع برای تولید داده از روی ویژگی‌های بدست آمده در واحدهای مخفی ساخته شده است. بدین ترتیب ویژگی‌هایی که قبلا استخراج شده است را می‌توان در ماتریس extractedFeature به تابع داده و خروجی generatedData را در آن مشاهده کرد. در اینجا پارامتر k نشاندهنده تعداد مراحل اجرای نمونه‌برداری برای رسیدن به نمونه تولید شده است. در شکل ۳-۱۳ مثال‌هایی از خروجی این تابع مشاهده می‌شود. در این تصویر یک مدل RBM با ۲۵۰ واحد مخفی برای دادگان MNIST آموزش داده شده است و سپس از روی ۲۵۰ ویژگی استخراج شده برای ۹ نمونه تست، مجددا داده‌ها رسم شده‌اند.



شکل ۳-۱۳ : مثال‌هایی از تصاویر ساخته شده از روی ویژگی‌های بدست آمده برای تعدادی از نمونه‌های تست دادگان MNIST . تصویر شماره ۱: تصویر اصلی از دادگان MNIST، تصویر شماره ۲: تصویر ساخته شده از روی ویژگی بدست آمده با کمک تابع getFeature با k=1، تصویر شماره ۳: تصویر ساخته شده شبیه تصویر شماره ۲ با k=10، تصویر شماره ٤: تصویر ساخته شده شبیه تصویر شماره ۲ با k=100. در این تصاویر مشاهده می‌شود که با افزایش k ویژگی‌هایی بدست آمده که خروجی آن به مدل عددها نزدیکتر شده و به عنوان مثال تصویر عدد ۵ یا ٤ تبدیل به تصاویری بهتر از این اعداد شده است. کد مربوط به این خروجی در فایل test_generateData.m قرار دارد.

```
function
[reconstructedData]=reconstructData(obj,dataMatrix,k,fixedDimensions)
```

این تابع برای بازسازی مجدد داده‌های dataMatrix استفاده می‌شود. در واقع عملیات استخراج ویژگی و ساخت مجدد داده در داخل این تابع صورت می‌پذیرد. ورودی dataMatrix یک ماتریس است که داده‌ها به صورت ردیفی در آن قرار دارد. این تابع نیز با روش نمونه‌برداری، نمونه‌های واحدهای مخفی که همان ویژگی-های استخراج شده از داده ورودی است را بدست آورده و سپس مجددا داده را از روی ویژگی‌ها بازسازی می‌کند. پارامتر k نشاندهنده تعداد مراحل اجرای نمونه‌برداری برای رسیدن به نمونه بازسازی شده است و fixedDimensions برداری است که در آن ابعادی از داده ورودی را مشخص می‌کنیم که قصد داریم در حین

نمونه‌برداری ثابت بمانند. این ورودی در مواقعی مفید خواهد بود که از صحت بعضی از ابعاد داده اطمینان داریم و قصد داریم تا در حین نمونه‌برداری ثابت بمانند تا کیفیت نهایی به سبب تغییر آنها در حین نمونه‌برداری لطمه نخورد. در نهایت نیز خروجی‌ها به صورت ردیفی در ماتریس reconstructedData برگردانده می‌شود. در شکل ۳-۱٤ مثالی از اجرای تابع reconstructData مشاهده می‌شود که توانسته است تاحدی نویز را در داده ورودی از بین ببرد. در مثال یک RBM با ۲۵۰ واحد مخفی بر روی دادگان تمیز MNIST آموزش داده شده و در تست، خروجی بازسازی داده نویزی بدست آمده است.

شکل ۳-۱٤: مثالی از اجرای تابع reconstructData، تصویر شماره ۱: تصویر اصلی از دادگان MNIST، تصویر شماره ۲: همان تصویر با اضافه شدن نویز گوسی با میانگین ۰ و واریانس ۰,۰۲، تصویر شماره ۳: خروجی تابع reconstructData بر روی داده نویزی. همانطور که مشاهده می‌شود، تابع reconstructData توانسته تا حدی داده نویزی را به داده تمیز تبدیل کند هر چند به دلیل وجود نویز در ورودی، خروجی دقیقا شبیه ورودی نیست. کد مربوط به این خروجی در فایل test_reconstructData.m قرار دارد.

```
function perf=computePerformance(obj,data)
```

این تابع برای محاسبه میزان کارآیی مدل از روی داده اعتبارسنجی[٤٤] استفاده می‌شود. با توجه به تنظیماتی که در پارامترهای RBM قابل انجام است، می‌توان به سه طریق مختلف کارآیی را محاسبه نمود که عبارتند از روش استفاده از اندازه انرژی آزاد، امکان بازسازی داده و همچنین دقت دسته‌بندی. البته روش آخر فقط در RBM های تمایزی قابل استفاده است و در صورت استفاده از این روش در این کلاس، یک توضیح خطا مربوط به عدم امکان محاسبه کارآیی نمایش داده می‌شود.

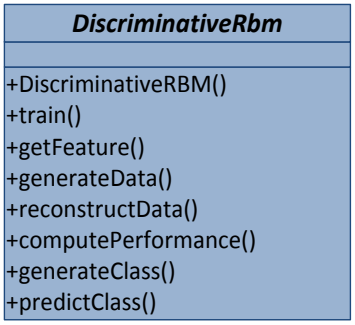
در روش استفاده از انرژی آزاد، با کمک تابع freeEnergy (بخش ۳-۵-٦) مقدار انرژی آزاد برای داده ارزیابی محاسبه شده و برگردانده می‌شود. بر اساس توضیحات ۲-۱-۱-۵ هر چه مقدار انرژی آزاد کمتر باشد، مدل برای داده‌های آموزشی بهتر ساخته شده است.

اما در روش امکان بازسازی داده برای محاسبه کارآیی، داده ارزیابی بازسازی شده و اختلاف نمونه‌های بازسازی شده از داده‌های اصلی بدست می‌آید. طبیعتا هرچه این اختلاف مقدار کمتری داشته باشد مدل بهتر آموزش یافته است.

[٤٤] validation

۳-۸ کلاس DiscriminativeRBM

این کلاس یک نوع RBM است که می‌تواند یک مدل مولد از روی داده‌های آموزشی با برچسب بسازد که امکان دسته‌بندی داده‌ها را نیز دارد. این کلاس از کلاس Rbm ارث می‌برد. اجزای این کلاس در شکل ۳-۱۵ آمده است.



شکل ۳-۱۵ : اجزای تشکیل دهنده کلاس DiscriminativeRBM

از آنجا که این کلاس از کلاس Rbm ارث می‌برد لذا ویژگی‌های آن را دارا می‌باشد. علاوه بر این ویژگی‌ها توابعی را نیز در خود دارد که در ادامه به توضیح آن خواهیم پرداخت.

```
function obj=DiscriminativeRBM(rbmParams)
```

این تابع سازنده کلاس DiscriminativeRBM است که در آن علاوه بر صدا زدن تابع سازنده کلاس Rbm، rbmType را نیز از نوع discriminative مقداردهی می‌کند.

```
function train(obj,data)
```

این تابع مربوط به آموزش مدل RBM است که با کمک داده آموزشی گذاشته شده در data مدل را به صورت تمایزی آموزش می‌دهد. ورودی data در واقع یک شئ از نوع کلاس DataStore (بخش ۳-٤-۱) است. در این تابع بعد از مقداردهی اولیه پارامترهای مدل، با کمک داده آموزشی این پارامترها اصلاح می‌شوند. از آنجا که در این کلاس مدل به صورت تمایزی آموزش می‌بیند لذا برچسب داده‌های آموزشی به صورت واحدهای softmax به مدل و داده‌های آموزشی اضافه می‌شود (توضیح بیشتر در بخش ۲-۱-۵). در حال حاضر شرط خاتمه در یادگیری، تعداد تکرار فرآیند آموزش (رسیدن به maxEpoch) می‌باشد. همچنین در حین آموزش اطلاعاتی درباره وضعیت فعلی یادگیری نمایش داده می‌شود. این اطلاعات شامل ساختار RBM در حال آموزش، شماره گام آموزش (epoch)، میزان کارآیی بر روی داده ارزیابی و تخمین زمان باقیمانده برای آموزش این مدل می‌باشد.

```
function [extractedFeature]=getFeature(obj,dataMatrix,k,fixedDimensions,
dataMatrixLabels)
```

این تابع برای گرفتن ویژگی از داده ورودی dataMatrix با برچسب dataMatrixLabels استفاده می‌شود. ورودی dataMatrix یک ماتریس است که داده‌ها به صورت ردیفی در آن قرار دارد. این تابع نیز با روش نمونه‌برداری، نمونه‌های واحدهای مخفی که همان ویژگی‌های استخراج شده از داده ورودی است را بدست می-

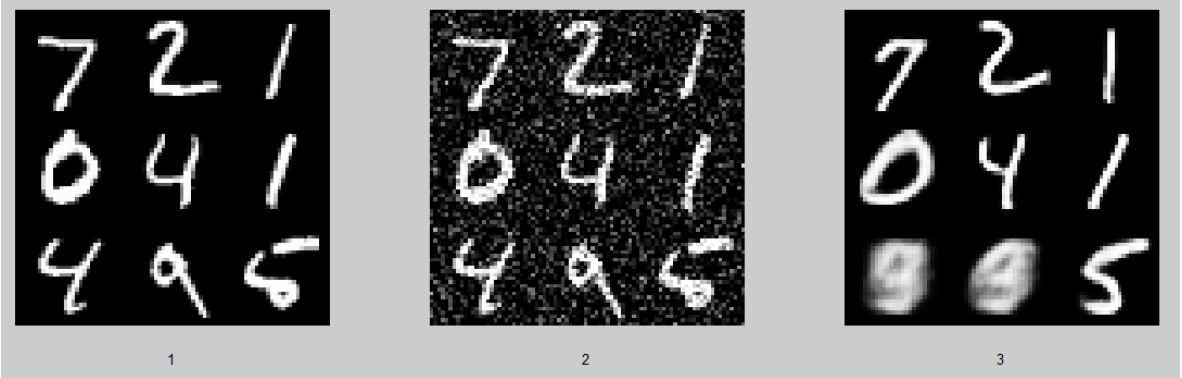
آورد. بردار dataMatrixLabels نیز برچسب هر داده موجود در dataMatrix را نشان می‌دهد تا بر اساس آن مقدار مناسب در واحدهای softmax لایه قابل مشاهده قرار گیرد. البته در صورتی که این پارامتر تهی باشد، تابع مقدار صفر را برای همه واحدهای softmax درنظر می‌گیرد. پارامتر k نشاندهنده تعداد مراحل اجرای نمونه‌برداری برای رسیدن به ویژگی استخراج شده است و fixedDimensions برداری است که در آن ابعادی از داده ورودی را مشخص می‌کنیم که قصد داریم در حین نمونه‌برداری ثابت بمانند. این ورودی در مواقعی مفید خواهد بود که از صحت بعضی از ابعاد داده اطمینان داریم و قصد داریم تا در حین نمونه‌برداری ثابت بمانند تا کیفیت نهایی به سبب تغییر آنها در حین نمونه‌برداری لطمه نخورد. در نهایت نیز خروجی‌ها به صورت ردیفی در ماتریس extractedFeature برگردانده می‌شود.

```
function [generatedData,generatedLabel]=generateData(obj,extractedFeature,
k)
```

این تابع برای تولید داده و برچسب از روی ویژگی‌های بدست آمده در واحدهای مخفی ساخته شده است. بدین ترتیب ویژگی‌هایی که قبلا استخراج شده است را می‌توان در ماتریس extractedFeature به تابع داده و خروجی داده generatedData و برچسب generatedLabel را در آن مشاهده کرد. مقدار generatedLabel در واقع مربوط به واحد softmax ای است که بیشترین احتمال فعال شدن را دارد. در اینجا پارامتر k نشاندهنده تعداد مراحل اجرای نمونه‌برداری برای رسیدن به نمونه تولید شده است.

```
function
[reconstructedData,reconstructedLabel]=reconstructData(obj,dataMatrix,k,
fixedDimensions,dataMatrixLabels)
```

این تابع برای بازسازی مجدد داده‌های dataMatrix استفاده می‌شود. در واقع عملیات استخراج ویژگی و ساخت مجدد داده در داخل این تابع صورت می‌پذیرد. ورودی dataMatrix یک ماتریس است که داده‌ها به صورت ردیفی در آن قرار دارد و بردار dataMatrixLabels نیز برچسب هر داده موجود در dataMatrix را نشان می‌دهد. این تابع نیز با روش نمونه‌برداری، نمونه‌های واحدهای مخفی که همان ویژگی‌های استخراج شده از داده ورودی است را بدست آورده و سپس مجددا داده و برچسب را از روی ویژگی‌ها بازسازی می‌کند. پارامتر k نشاندهنده تعداد مراحل اجرای نمونه‌برداری برای رسیدن به نمونه بازسازی شده است و fixedDimensions برداری است که در آن ابعادی از داده ورودی را مشخص می‌کنیم که قصد داریم در حین نمونه‌برداری ثابت بمانند. این ورودی در مواقعی مفید خواهد بود که از صحت بعضی از ابعاد داده اطمینان داریم و قصد داریم تا در حین نمونه‌برداری ثابت بمانند تا کیفیت نهایی به سبب تغییر آنها در حین نمونه‌برداری لطمه نخورد. در نهایت نیز خروجی‌ها به صورت ردیفی در ماتریس reconstructedData برگردانده می‌شود. در مثالی از اجرای تابع reconstructData مشاهده می‌شود که توانسته است تاحدی نویز را در داده ورودی از بین ببرد. در مثال یک RBM با ۲۵۰ واحد مخفی بر روی دادگان تمیز MNIST آموزش داده شده و در تست، خروجی بازسازی داده نویزی بدست آمده است.



شکل ۳-۱۶: مثالی از اجرای تابع reconstructData در کلاس RBM از نوع تمایزی، تصویر شماره ۱: تصویر اصلی از دادگان MNIST، تصویر شماره ۲: همان تصویر با اضافه شدن نویز گوسی با میانگین ۰ و واریانس ۰,۰۵، تصویر شماره ۳: خروجی تابع reconstructData بر روی داده نویزی. همانطور که مشاهده می‌شود، تابع reconstructData توانسته تا حدی داده نویزی را به داده تمیز تبدیل کند. در اینجا نیز نسبت به شکل ۳-۱۴ هرچند واریانس نویز ۲,۵ برابر شده است اما به دلیل استفاده از اطلاعات برچسب، توانسته‌ایم داده را تا حدی خالی از نویز نماییم هرچند بعضی از اعداد به تصویر درستی نرسیده‌اند. کد مربوط به این خروجی در فایل test_reconstructData2.m قرار دارد.

```
function perf=computePerformance(obj,data)
```

این تابع برای محاسبه میزان کارآیی مدل از روی داده اعتبارسنجی استفاده می‌شود. با توجه به تنظیماتی که در پارامترهای RBM قابل انجام است، می‌توان به سه طریق مختلف کارآیی را محاسبه نمود که عبارتند از روش استفاده از اندازه انرژی آزاد، امکان بازسازی داده و همچنین دقت دسته‌بندی.

در روش استفاده از انرژی آزاد، با کمک تابع freeEnergy (بخش ۳-۵-۶) مقدار انرژی آزاد برای داده ارزیابی محاسبه شده و برگردانده می‌شود. البته در اینجا مقدار صفر برای واحدهای softmax مربوط به برچسب داده‌ها در نظر گرفته می‌شود. بر اساس توضیحات ۲-۱-۱-۵ هر چه مقدار انرژی آزاد کمتر باشد، مدل برای داده‌های آموزشی بهتر ساخته شده است.

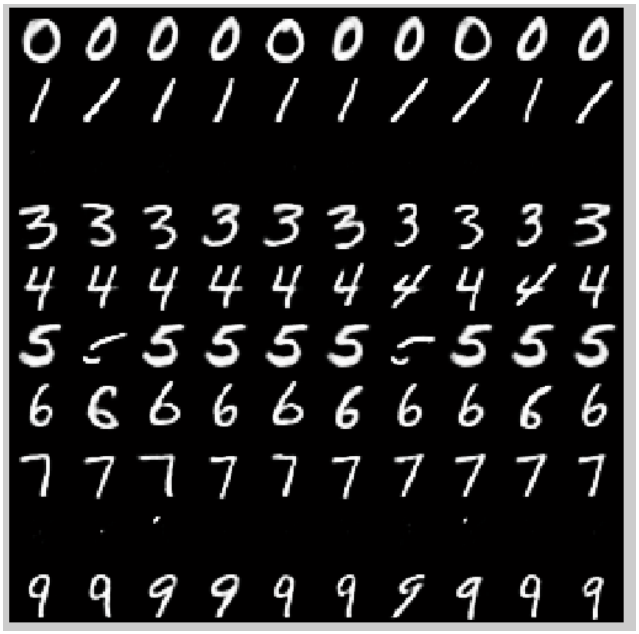
در روش امکان بازسازی داده برای محاسبه کارآیی، داده ارزیابی بازسازی شده و اختلاف نمونه‌های بازسازی شده از داده‌های اصلی بدست می‌آید. طبیعتا هرچه این اختلاف مقدار کمتری داشته باشد مدل بهتر آموزش یافته است. در اینجا مقدار برچسب داده‌ها برای بازسازی استفاده نمی‌شود.

در روش استفاده از دقت دسته‌بندی، دسته‌ی داده‌ها به کمک روش نمونه‌برداری و به صورت مولد (توضیح در تابع predictClass) مشخص شده و خطا با توجه به برچسب درست داده‌ها بدست می‌آید.

```
function [generatedData]=generateClass(obj,classNumber,k)
```

این تابع جهت تولید نمونه‌هایی از توزیع مربوط به یکی از دسته‌ها استفاده می‌شود. در واقع در این تابع از توزیع داده مربوط به کلاس‌های مشخص شده در بردار classNumber نمونه تولید می‌شود و در خروجی generatedData برگردانده می‌شود. پارامتر k نیز نشاندهنده تعداد مراحل بازسازی داده برای رسیدن به نمونه بازسازی شده است. در شکل ۳-۱۷ نمونه‌های تولید شده توسط مدل نشان داده شده است. همانطور که در شکل

مشخص است، مدل توانسته تنها با استفاده از برچسب داده، نمونه‌های مختلفی از اعداد را تولید نماید. البته در این شکل، مدل نتوانسته است اعداد ۲ و ۸ را تولید کند.



شکل ۳-۱۷ : تصاویر ساخته شده به کمک تابع generateClass. در اینجا خواسته‌ایم تا اعداد ۰ تا ۹ را توسط مدل با تنها استفاده از برچسب داده تولید نماییم. مطابق تصویر، عدد ۲ و ۸ نتوانسته‌اند تولید شوند. کد مربوط به این خروجی در فایل test_generateClass.m قرار دارد.

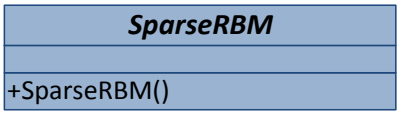
```
function classNumber=predictClass(obj,dataMatrix,method)
```

این تابع جهت بدست آوردن برچسب داده موجود در ماتریس ردیفی dataMatrix استفاده می‌شود. در واقع دسته‌بندی داده‌های بدون برچسب به کمک این تابع انجام می‌شود. ورودی method روش رسیدن به برچسب داده را مشخص می‌کند.

روش اول به نام bySampling است که با روش بازسازی داده و فعال شدن واحد softmax مربوط به کلاس داده، برچسب را تشخیص می‌دهد. روش دوم به نام byFreeEnergy است که با استفاده از قرار دادن حالات مختلف مقدار در softmax احتمال فعال شدن بیشترین حالت را با کمک انرژی آزاد بدست می‌آورد. توضیح بیشتر این دو روش در بخش ۲-۱-۵-۱ آمده است.

۳-۹ کلاس SparseRBM

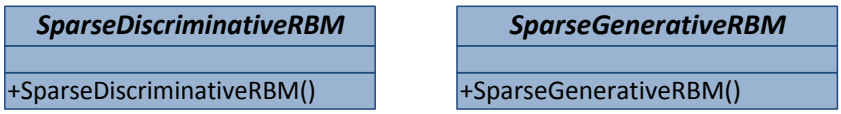
در این کلاس روابط مربوط به یک RBM تنک به سه روش مختلف نوشته شده است. روش‌های پیاده‌سازی شده در این کلاس روش‌های اول، سوم و چهارم توضیح داده شده در بخش ۲-۱-٤ می‌باشد. در واقع در این کلاس تنها تابع getRegularizationGradient به نحوی بازنویسی شده است تا مقادیر گرادیان را با توجه به ترم regularization تعریف شده در هر یک از این سه روش، به درستی محاسبه نماید.



شکل ۳-۱۸ : اجزای تشکیل دهنده کلاس SparseRBM

۳-۱۰ کلاس SparseGenerativeRBM و SparseDiscriminativeRBM

این دو کلاس پیاده سازی جدیدی نداشته و تنها از دو کلاس SparseRBM و GenerativeRBM و DiscriminativeRBM ارث گرفته اند. به این ترتیب ویژگی تنک بودن از کلاس SparseRBM به دو کلاس GenerativeRBM و DiscriminativeRBM منتقل شده و هر دو ویژگی تنک بودن را خواهند داشت.

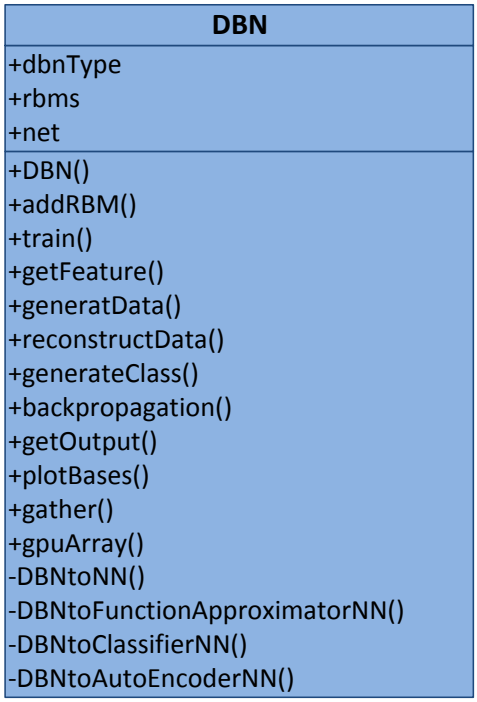


شکل ۳-۱۹ : اجزای تشکیل دهنده کلاس SparseGenerativeRBM و SparseDiscriminativeRBM

۳-۱۱ کلاس DBN

این کلاس مربوط به ساخت یک شبکه باور عمیق به کمک پشته‌ای از مدل‌های RBM است. دراین کلاس می‌توان با تعریف ساختار اولیه یک شبکه باور عمیق، RBM های تعریف شده در آن را مدل کرده و سپس در صورت نیاز با کمک الگوریتم پس انتشار خطا[۴۵] آن را اصلاح نمود. در شکل ۳-۲۰ اجزای تشکیل دهنده کلاس نمایش داده شده است.

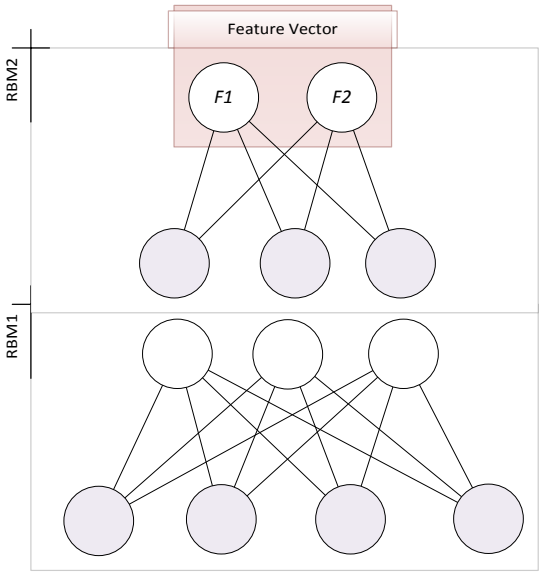
[۴۵] error backpropagation



شکل ۳-۲۰ : اجزای تشکیل دهنده کلاس DBN

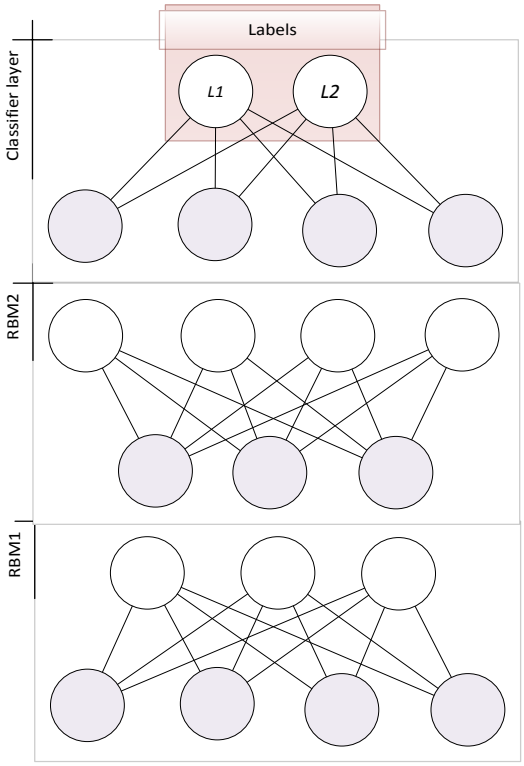
در این کلاس سه ویژگی تعریف شده است. ویژگی dbnType نشاندهنده نوع شبکه باور عمیق (مثل autoEncoder یا classifier) می‌باشد. ویژگی rbms نگهدارنده پشته‌ای از rbm ها است که می‌تواند هر یک از انواع تعریف شده rbm باشد. ویژگی بعدی net می‌باشد. این ویژگی زمان استفاده از الگوریتم پس انتشار خطا استفاده می‌شود تا شبکه عصبی مصنوعی ساخته شده در این مرحله را در خود نگهدار و در صورت نیاز در زمان گرفتن خروجی از شبکه باور عمیق مورد استفاده قرار گیرد. علاوه بر این ویژگی‌ها توابعی نیز جهت استفاده از این کلاس تعریف شده که در ادامه توضیح خواهیم داد.

اولین نوع DBN تعریف شده در این کلاس مربوط به autoEncoder می‌باشد. در این نوع از شبکه هدف رسیدن به یک مدل مولد است که بتواند ویژگی‌های جدیدی را از مدل استخراج نماید. در شکل ۳-۲۱ ساختار این مدل نمایش داده شده است که در آن با کمک دو لایه RBM ویژگی‌هایی در سطح بالاتر استخراج می‌نماییم.



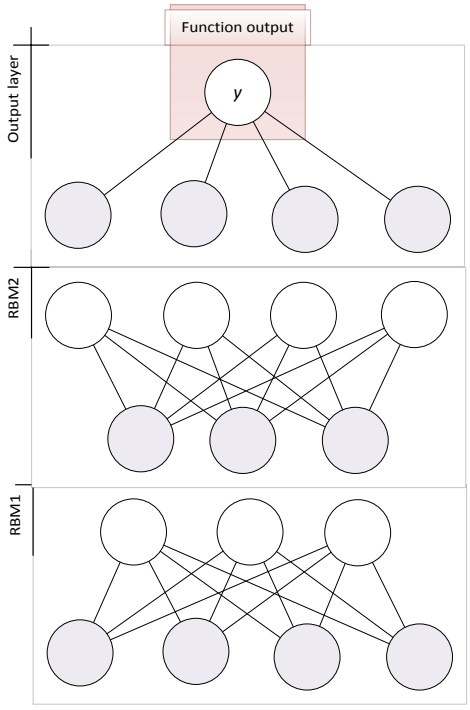
شکل ۳-۲۱ : ساختار یک شبکه باور عمیق از نوع autoEncoder

نوع دوم DBN تعریف شده در این جعبه ابزار مدل classifier می‌باشد. در این نوع هدف رسیدن به برچسب‌های داده از روی داده ورودی است. برای این منظور در عمل گویی شبکه به نحوی ساخته می‌شود که یک لایه برای عمل دسته‌بندی را در خود خواهد داشت که در توابع بعدی، نحوه اضافه شده این لایه توضیح داده خواهد شد.



شکل ۳-۲۲ : ساختار یک شبکه باور عمیق از نوع طبقه‌بندی کننده

نوع سوم شبکه‌های باور عمیق مربوط به تقریب‌زننده تابع[٤٦] است. در این نوع هدف رسیدن به یک تخمین زننده تابع است. برای این منظور در عمل گویی شبکه به نحوی ساخته می‌شود با اضافه شدن یک لایه عمل تخمین تابع را در خود داشته باشد که در توابع بعدی، نحوه اضافه شده این لایه توضیح داده خواهد شد.



شکل ۳-۲۳ : ساختار یک شبکه باور عمیق از نوع تقریب‌زننده تابع

```
function obj=DBN(dbnType,rbms)
```

این تابع سازنده کلاس DBN می‌باشد. به کمک این تابع نوع DBN تعریف شده (به کمک ورودی dbnType) و در صورتی که پشته‌ای از RBM ها قبلا ساخته شده باشد می‌توانیم با ارسال آن‌ها در این تابع (به کمک ورودی rbms)، از آنها مجددا استفاده نماییم.

```
function addRBM(obj,rbmParams)
```

این تابع جهت پشته کردن RBM ها بر روی یکدیگر استفاده می‌شود. به وسیله این تابع می‌توان یک به یک RBM ها را تعریف کرده و به DBN اضافه نمود. نمونه‌ای از استفاده این تابع در زیر آمده است.

```
dbn=DBN();
dbn.dbnType='autoEncoder';
% RBM1
rbmParams=RbmParameters(1000,ValueType.binary);
dbn.addRBM(rbmParams);
% RBM2
rbmParams=RbmParameters(500,ValueType.binary);
dbn.addRBM(rbmParams);
% RBM3
rbmParams=RbmParameters(250,ValueType.binary);
dbn.addRBM(rbmParams);
```

[٤٦] function approximator

در واقع در این تابع با توجه به نوع RBM تعریف شده در پارامتر rbmParams، شئ تعریف شده مربوط به آن RBM ساخته شده و در پشته قرار می‌گیرد.

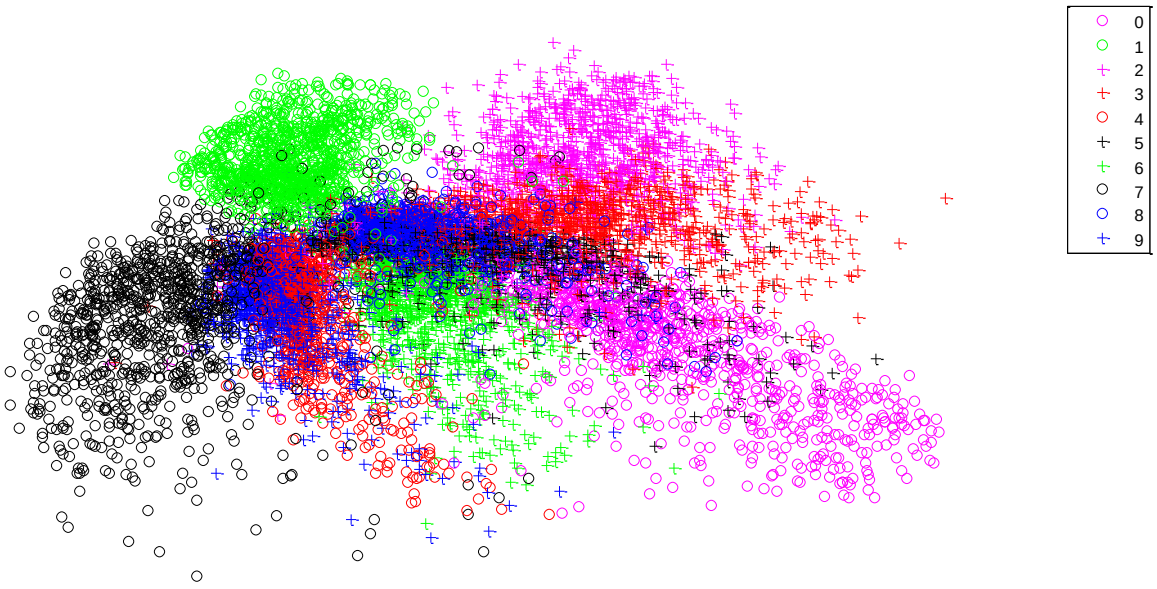
```
function train(obj,data)
```

این تابع جهت آموزش شبکه باور عمیق مورد استفاده قرار می‌گیرد. ورودی data نیز یک شئ از نوع کلاس DataStore می‌باشد که داده‌های آموزشی و ارزیابی و تست را در خود نگه داشته است. در حین اجرای این تابع، RBM ها یک به یک آموزش پیدا کرده و بر روی یکدیگر قرار می‌گیرند. در واقع این تابع بعد از آموزش هر RBM خروجی ویژگی‌های یاد گرفته شده توسط مدل را برای داده آموزشی و ارزیابی استخراج کرده و به عنوان ورودی برای RBM بعدی استفاده می‌کند.

```
function [extractedFeature]=getFeature(obj,dataMatrix,k)
```

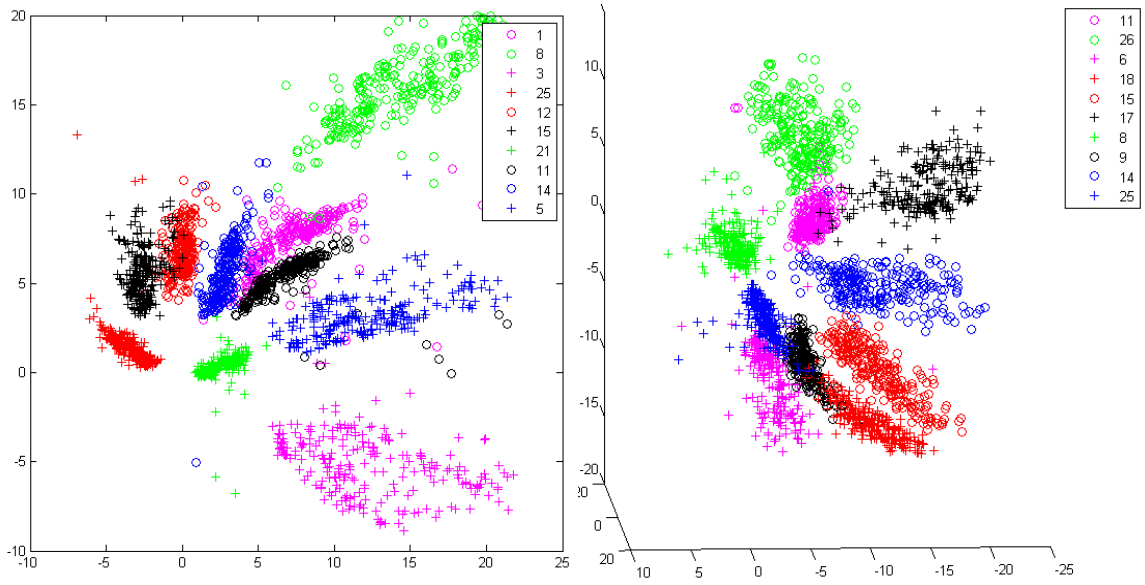
این تابع جهت استخراج ویژگی از داده ردیفی dataMatrix و از مدل DBN استفاده می‌شود. در واقع در این تابع به ترتیب ویژگی‌ها لایه به لایه از مدل‌های RBM بدست آمده و در بالاترین RBM (آخرین RBM) برای استخراج ویژگی از k مرحله نمونه‌برداری استفاده می‌شود.

در شکل ۳-۲٤ نمونه‌ای از استخراج ویژگی به کمک DBN برای دادگان MNIST مشاهده می‌شود. در اینجا یک DBN با ساختار ۷۸٤–۱۰۰۰–۵۰۰–۲۵۰–۳ ساخته شده است که فضای ورودی تصویر ۷۸٤ بعدی را به فضای ۳ بعدی نگاشت می‌دهد. در واقع در شکل ۳-۲٤ این نگاشت نمایش داده شده است. همان طور که در تصویر مشاهده می‌شود شبکه باور عمیق توانسته سه ویژگی مناسب را بدون در نظر گرفتن برچسب داده‌ها به نحوی پیدا کند که داده‌های مربوط به ارقام مختلف در این فضای جدید تا حد قابل قبولی از هم قابل تفکیک باشند. در این آزمایش هر یک از RBM ها با ۵۰ تکرار و الگوریتم پس انتشار خطا با ۲۰۰ تکرار اجرا شده است.



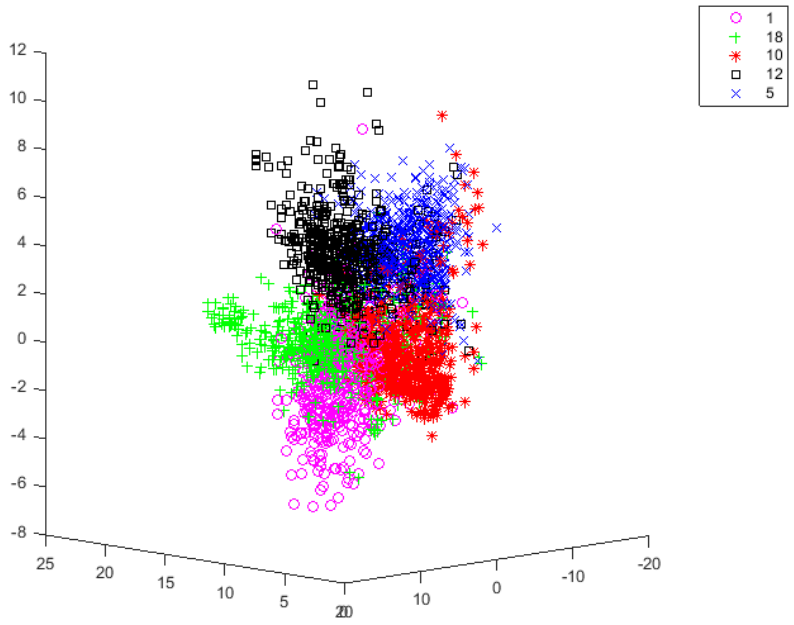
شکل ۳-۲٤ : نمایش خروجی ویژگی بدست آمده توسط یک شبکه باور عمیق با ساختار ۷۸٤–۱۰۰۰–۵۰۰–۲۵۰–۳. کد مربوط به این خروجی در فایل test_getFeature.m قرار دارد.

همچنین در تست دیگری از دادگان ISOLET برای استخراج ویژگی استفاده شد. این دادگان شامل گفتار ۱۵۰ گوینده است که هر یک از حروف زبان انگلیسی را ۲ بار تکرار کرده و ۶۱۷ ویژگی از هر نمونه استخراج شده است. در شکل ۳-۲۵ ویژگی‌های استخراج شده دو بعدی (با ساختار ۶۱۷–۲۰۰۰–۱۰۰۰–۵۰۰–۲۵۰–۲) و سه بعدی (با ساختار ۶۱۷–۲۰۰۰–۱۰۰۰–۵۰۰–۲۵۰–۳) را مشاهده می‌کنید. در اینجا نیز شبکه باور عمیق توانسته است ویژگی‌های خوبی با قابلیت تفکیک بالا برای این دادگان بوجود آورد. البته این دادگان شامل ۲۶ دسته مختلف می‌باشد که در اینجا برای نمایش بهتر، ۱۰ دسته از آن به صورت تصادفی انتخاب شده است. در این آزمایش هر یک از RBM ها با ۲۰۰ تکرار و الگوریتم پس انتشار خطا با ۱۰۰۰ تکرار اجرا شده است.



شکل ۳-۲۵ : نمایش خروجی ویژگی بدست آمده توسط یک شبکه باور عمیق دو بعدی (با ساختار ۶۱۷–۲۰۰۰–۱۰۰۰–۵۰۰–۲۵۰–۲) و سه بعدی (با ساختار ۶۱۷–۲۰۰۰–۱۰۰۰–۵۰۰–۲۵۰–۳) برای دادگان ISOLET. در اینجا ۱۰ دسته از ۲۶ دسته این دادگان به طور تصادفی انتخاب و نمایش داده شده است. کد مربوط به این خروجی در فایل test_getFeatureISOLET.m قرار دارد.

همچنین در تست دیگری از دادگان 20 Newsgroups برای استخراج ویژگی استفاده شد. این دادگان، یک دادگان متنی است که شامل ۲۰ موضوع خبری مختلف می‌باشد که هر یک از نمونه‌ها به یکی از این ۲۰ موضوع برچسب خورده‌اند. در شکل ۳-۲۶ ویژگی‌های استخراج شده سه بعدی (با ساختار ۵۰۰۰–۵۰۰–۵۰۰–۲۵۰–۳) را مشاهده می‌کنید. در اینجا نیز شبکه باور عمیق توانسته است ویژگی‌های خوبی با قابلیت تفکیک بالا برای این دادگان بوجود آورد. البته این دادگان شامل ۲۰ دسته مختلف می‌باشد که در اینجا برای نمایش بهتر، ۵ دسته از آن برای نمایش انتخاب شده است. در این آزمایش هر یک از RBM ها با ۲۰۰ تکرار و الگوریتم پس انتشار خطا با ۲۰۰ تکرار اجرا شده است.



شکل ۳-۲۶ : نمایش خروجی ویژگی بدست آمده توسط یک شبکه باور عمیق سه بعدی (با ساختار ۵۰۰۰–۵۰۰–۵۰۰–۲۵۰–۳) برای دادگان 20 Newsgroups. در اینجا ۵ دسته از ۲۰ دسته این دادگان برای نمایش انتخاب شده است. کد مربوط به این خروجی در فایل test_getFeature20Newsgroups.m قرار دارد.

```
function [generatedData]=generateData(obj,extractedFeature,k)
```

این تابع جهت تولید داده از ویژگی‌های extractedFeature و از مدل DBN می‌باشد. در این تابع در ابتدا بالاترین RBM با k مرحله نمونه‌برداری به نمونه‌ای در لایه قابل مشاهده خود می‌رسد و سپس لایه به لایه نمونه‌ها در لایه-های پایینتر تا رسیدن به داده مورد نظر، استخراج می‌شود. در پایان نیز نمونه‌های تولید شده در خروجی generatedData ارسال می‌گردند.

```
function
[reconstructedData]=reconstructData(obj,dataMatrix,k,fixedDimensions)
```

این تابع جهت بازسازی داده مورد استفاده قرار می‌گیرد. با کمک این تابع می‌توان داده ورودی dataMatrix را با مدل شبکه باور عمیق بازسازی کرده و خروجی را در reconstructedData برگرداند. نحوه عملکرد این تابع به این صورت است که در ابتدا با کمک تابع getFeature و k مرحله نمونه‌برداری، ویژگی داده‌های ورودی را استخراج کرده و سپس با کمک تابع generateData نمونه‌های مورد نظر تولید می‌شود. در این‌جا نیز تنها استفاده از ورودی fixedDimensions این خواهد بود که داده‌های خروجی reconstructedData در ابعاد مشخص شده در fixedDimensions همان مقادیر dataMatrix را خواهند داشت ولی این مساله در زمان تولید داده مورد استفاده قرار نمی‌گیرد.

```
function [generatedData]=generateClass(obj,classNumber,k)
```

این تابع جهت تولید داده با توجه به دانستن برچسب داده مورد استفاده قرار می‌گیرد. در واقع در صورتی که بالاترین RBM یک مدل DiscriminativeRBM باشد، ابتدا با کمک تابع generateClass در مدل

DiscriminativeRBM یک داده با ورودی k و calssNumber تولید کرده (بخش ۳–۸) و سپس داده تولید شده را لایه به لایه تا رسیدن به نمونه نهایی پایین می‌بریم.

```
function backpropagation(obj,data)
```

این تابع جهت اصلاح پارامترهای آموزش دیده توسط الگوریتم پس انتشار خطا مورد استفاده قرار می‌گیرد. ورودی data نیز یک شئ از نوع کلاس DataStore می‌باشد که داده‌های آموزشی و ارزیابی و تست را در خود نگه داشته است. برای انجام بهتر این الگوریتم، از ابزار پیاده‌سازی شده در خود MATLAB استفاده شده است. بدین ترتیب ابتدا شبکه باور عمیق به ساختار یک شبکه عصبی مصنوعی در MATLAB در آمده و سپس بعد از تنظیم پارامترهای یادگیری در این شبکه، با توجه به نوع شبکه باور عمیق، تابع یادگیری صدا زده می‌شود. و در نهایت در صورت امکان پارامترهای یادگرفته شده در شبکه باور عمیق قرار می‌گیرد.

```
function output=getOutput(obj,dataMatrix,method)
```

این تابع برای گرفتن خروجی از شبکه باور عمیق مورد استفاده قرار می گیرد. با کمک این تابع با توجه به نوع شبکه باور عمیق می‌توان خروجی متناظر را برای داده ورودی dataMatrix بدست آورده و در خروجی output ارسال نمود. ورودی method نیز معرف روشی خواهد بود که این تابع خروجی را بدست خواهد آورد. مثلا در شبکه باور عمیق از نوع classifier می‌توان مقدار method را روش byFreeEnergy و یا bySampling برای رسیدن به برچسب داده‌ها مقداردهی کرد.

بدین ترتیب در این تابع در صورتی‌که نوع شبکه باور عمیق autoEncoder باشد ویژگی‌های داده‌های موجود در dataMatrix را برمی‌گرداند و در صورتیکه از نوع طبقه‌بندی کننده باشد برچسب داده‌های ورودی را برمی-گرداند و در نهایت برای نوع تقریب‌زننده تابع، خروجی تابع یادگرفته شده برگردانده می‌شود.

```
function net=DBNtoNN(obj,data)
```

این تابع که به صورت خصوصی[۴۷] تعریف شده است، برای تبدیل شبکه باور عمیق به یک شئ از نوع شبکه عصبی در MATLAB مورد استفاده قرار می‌گیرد که کاربرد اصلی آن در استفاده از پس‌انتشار خطا [۴۸] می‌باشد. در این تابع با توجه به نوع شبکه باور عمیق سه تابع دیگر را برای تبدیل فراخوانی می‌نماید که در ادامه به توضیح آن-ها خواهیم پرداخت.

```
function net=DBNtoClassifierNN(obj,data)
```

این تابع برای تبدیل شبکه باور عمیق از نوع طبقه‌بندی کننده به یک شئ شبکه عصبی مصنوعی در MATLAB می‌باشد تا بتوان به کمک آن الگوریتم پس‌انتشار خطا را به راحتی اجرا نمود. در DBN های از نوع طبقه‌بندی کننده دو حالت می‌تواند وجود داشته باشد. در حالت اول اگر RBM در بالاترین لایه از نوع DiscriminativeRBM باشد، ساختار DBN بدست آمده به صورت تصویر سمت چپ از شکل ۳-۲۷ خواهد بود.

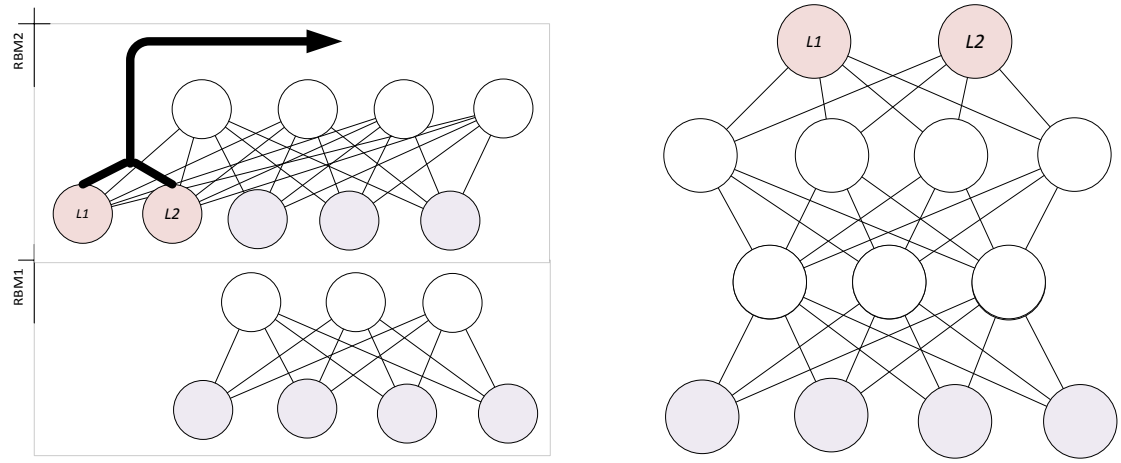
---

[۴۷] private

[۴۸] error backpropagation

در این حالت شبکه عصبی مصنوعی ساخته خواهد شد که لایه خروجی آن گرفته شده از واحدهای softmax در لایه قابل مشاهده در DiscriminativeRBM است که در نهایت تبدیل به تصویر سمت راست از شکل ۳-۲۷ تبدیل خواهد شد.

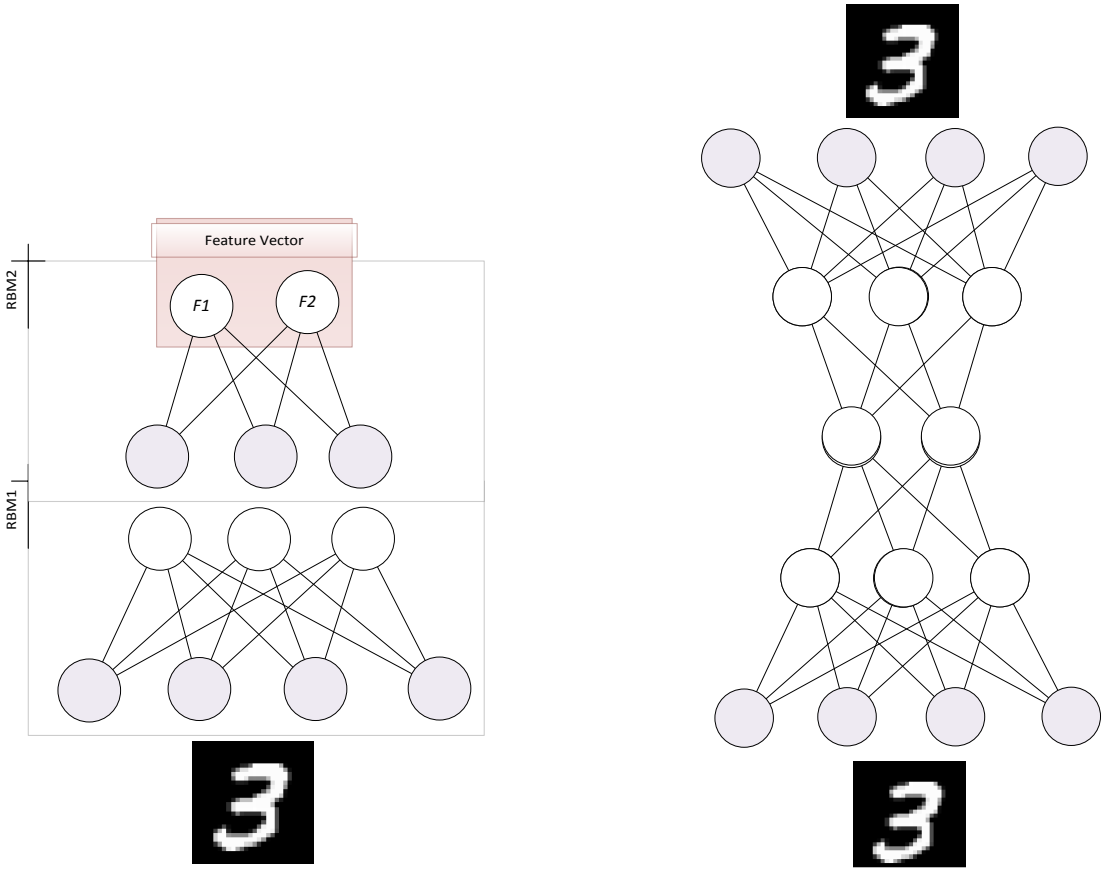
اما در حالتی که RBM در بالاترین لایه از نوع غیر تمایزی باشد، چون برچسب ها در آن لحاظ نشده است لذا برای لایه مربوط به خروجی، واحدهای جدیدی ساخته شده و وزن بین لایه خروجی و لایه آخر به صورت تصادفی مقداردهی می‌گردد. در اینجا نیز شکل خروجی به صورت تصویر سمت راست شکل ۳-۲۷ خواهد بود.



شکل ۳-۲۷ : نحوه تبدیل شبکه باور عمیق از نوع classifier به یک شبکه عصبی مصنوعی در MATLAB . تصویر سمت چپ: ساختار یک شبکه باور عمیق که لایه آخر آن از نوع DiscriminativeRBM بوده و برای تبدیل به شبکه عصبی مصنوعی، واحدهای مربوط به برچسب به همراه وزن‌های آن به عنوان لایه آخر (یا خروجی) در نظر گرفته می‌شود. تصویر سمت راست: ساختار شبکه عصبی مصنوعی که از شبکه باور عمیق ساخته شده است.

```
function net=DBNtoAutoEncoderNN(obj)
```

این تابع برای تبدیل شبکه باور عمیق از نوع autoEncoder به شبکه عصبی مصنوعی در MATLAB طراحی شده است. همانطور که گفته شد، در این نوع از شبکه باور عمیق هدف رسیدن به کدها یا ویژگی‌های جدیدی از داده ورودی است که بتوان از روی آن دوباره به داده اصلی رسید. لذا برای ساخت یک شبکه عصبی در آن و اجرای الگوریتم پس‌انتشار خطا ، مدل DBN ساخته شده را به صورت وارونه بر روی DBN قرار می‌دهیم تا ما مجددا از کد تولید شده ما را به داده اصلی برساند. در شکل ۳-۲۸نحوه این تبدیل نمایش داده شده است.

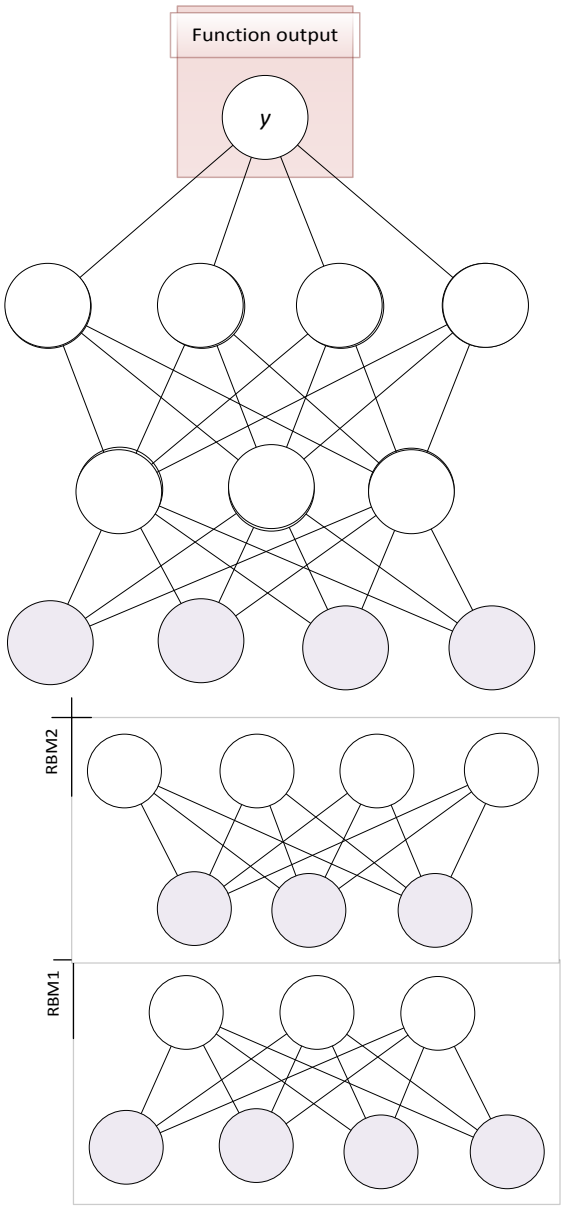


شکل ۳-۲۸ : نحوه تبدیل شبکه باور عمیق از نوع autoEncoder به یک شبکه عصبی مصنوعی در MATLAB . تصویر سمت چپ: ساختار یک شبکه باور عمیق که برای تبدیل شبکه باور عمیق ساخته شده را به صورت وارونه کپی می‌کند تا بتواند مجددا داده را بازسازی نماید. تصویر سمت راست: ساختار شبکه عصبی مصنوعی که از شبکه باور عمیق ساخته شده است.

```
function net=DBNtoFunctionApproximatorNN(obj,data)
```

در این تابع شبکه باور عمیق از نوع تقریب‌زننده تابع[۴۹] تبدیل به شبکه عصبی مصنوعی در MATLAB شده است.

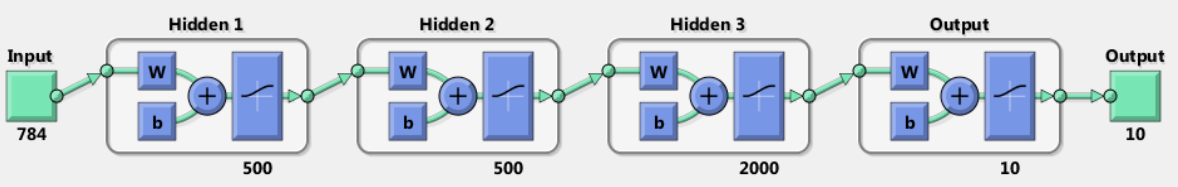
[۴۹] functionApproximator



شکل ۳-۲۹ : نحوه تبدیل شبکه باور عمیق از نوع تقریب‌زننده تابع به یک شبکه عصبی مصنوعی در MATLAB . تصویر سمت چپ: ساختار یک شبکه باور عمیق از نوع تقریب‌زننده تابع تصویر سمت راست: ساختار شبکه عصبی مصنوعی که از شبکه باور عمیق برای تخمین تابع ساخته شده است.

بعد از تبدیل شبکه باور عمیق به یک شبکه عصبی در MATLAB ، امکان استفاده از BP فراهم شده و بعد از تنظیم دقیقتر شبکه با BP، استفاده از شبکه عمیق با کمک شبکه آموزش دیده شده توسط جعبه ابزار شبکه عصبی در MATLAB قابل استفاده خواهد بود.

برای نمونه یک شبکه باور عمیق برای دسته‌بندی داده‌های MNIST به صورت ساختار شکل ۳-۳۰ طراحی شده و عمل دسته‌بندی بر روی دادگان تست انجام گشت. خطای دسته‌بندی بر روی تمام دادگان تست (۱۰۰۰۰ نمونه که در آموزش استفاده نشده است) قبل از استفاده از الگوریتم پس‌انتشار خطا برابر ۰٫۰۲۷۰ و بعد از الگوریتم پس‌انتشار خطا برابر ۰٫۰۱۲۰ بود.

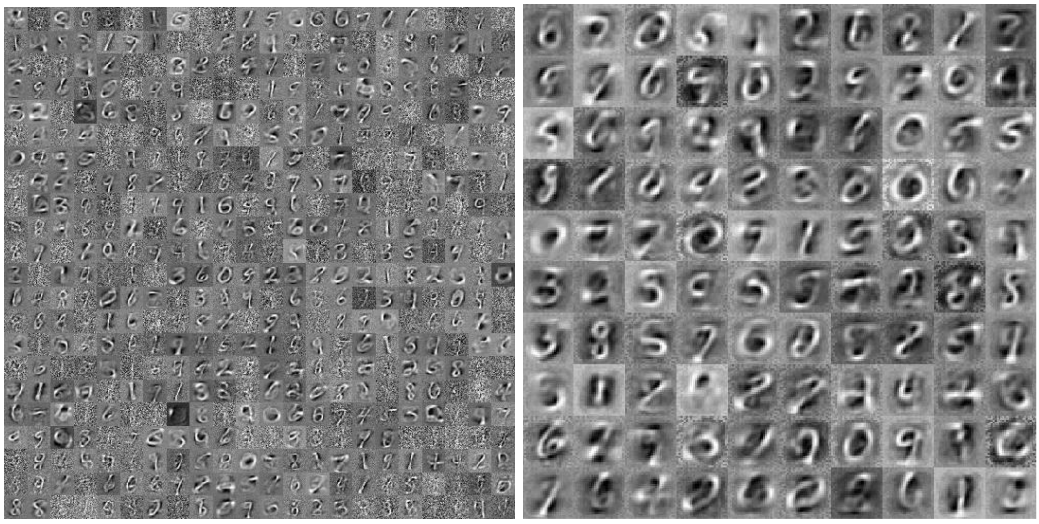


شکل ۳-۳۰ : یک ساختار نمونه برای دسته‌بندی دادگان MNIST

تابع مفید دیگری که در کلاس قابل استفاده است، تابعی برای رسم توابع پایه‌ی یادگیری شده توسط هر یک از لایه‌های DBN است.

```
function bases=plotBases(obj,rbmNumber)
```

در واقع به کمک این تابع و تعیین شماره RBM می‌توان موارد یاد گرفته شده در آن RBM را با رسم وزن-های متصل به هر واحد مخفی نشان داد. در شکل ۳-۳۱ نمونه‌ای از رسم این توابع پایه یادگیری شده را در دادگان MNIST و در یک شبکه باور عمیق می‌بینید.



شکل ۳-۳۱ : توابع پایه رسم شده توسط تابع plotBases در یک DBN دولایه تنک. شکل سمت چپ: توابع پایه یادگیری شده در لایه اول ، شکل سمت راست: توابع پایه یادگیری شده در لایه دوم

```
function [obj]=gather(obj)
```

این تابع برای استخراج مقادیر پارامترهای این شئ از حافظه GPU به حافظه محلی استفاده می‌شود. لذا در صورتی که قصد دارید DBN شاخته شده به کمک GPU را در کامپیوتر دیگری بدون GPU استفاده کنید، باید ابتدا آن را به کمک تابع gather به حالت عادی تبدیل نمایید.

```
function obj=gpuArray(obj)
```

این تابع با تغییر بعضی از پارامترهای RBM امکان استفاده از GPU را در محاسبات برقرار می‌کند.

۳-۱۲ کدهای کاربردی دیگر

در این بخش به توضیح توابع و کدهای دیگری می‌پردازیم که داخل جعبه ابزار نبوده ولی برای کار با آن طراحی شده است.

۳-۱۲-۱ تابع prepareMNIST

```
function [data] = prepareMNIST(mnistFolder)
```

این تابع به صورت مستقل در کنار جعبه ابزار قرار داده شده است. کاربرد این تابع در تبدیل فایل‌های مربوط به دادگان MNIST برای استفاده در قسمت‌های مختلف می‌باشد. در واقع این تابع داده‌ها و برچسب‌های آموزشی و تست مربوط به این دادگان را در یک شئ از نوع کلاس DataStore قرار داده و شئ ساخته شده را برمی‌گرداند. برای فایل‌های مربوط به دادگان MNIST می‌توانید آن را از آدرس http://yann.lecun.com/exdb/mnist دانلود نمایید. ورودی این تابع نیز یک رشته شامل آدرس پوشه‌ای است که فایل‌های مربوط به دادگان MNIST در آن قرار دارد. کدهای نوشته شده در این تابع الهام گرفته شده از کدهای مقاله [۳] می‌باشد. نمونه‌ای از استفاده این تابع را در خط زیر مشاهده می‌کنید.

```
data = MNIST.prepareMNIST('D:\DataSets\Image\MNIST\');
```

علت صدا زدن این تابع با MNIST. به دلیل آن است که این تابع در یک بسته یا پکیج به نام MNIST قرار داده شده است.

۳-۱۲-۲ تابع prepareISOLET

```
function [data] = prepareISOLET(isoletFolder)
```

این تابع به صورت مستقل در کنار جعبه ابزار قرار داده شده است. کاربرد این تابع در تبدیل فایل‌های مربوط به دادگان ISOLET برای استفاده در قسمت‌های مختلف می‌باشد. در واقع این تابع داده‌ها و برچسب‌های آموزشی و تست مربوط به این دادگان را در یک شئ از نوع کلاس DataStore قرار داده و شئ ساخته شده را برمی‌گرداند. برای فایل‌های مربوط به دادگان ISOLET می‌توانید آن را از آدرس https://archive.ics.uci.edu/ml/datasets/ISOLET دانلود نمایید. ورودی این تابع نیز یک رشته شامل آدرس پوشه‌ای است که فایل‌های مربوط به دادگان ISOLET در آن قرار دارد. نمونه‌ای از استفاده این تابع را در خط زیر مشاهده می‌کنید.

```
data = ISOLET.prepareISOLET('D:\DataSets\Voice\ISOLET\');
```

علت صدا زدن این تابع با ISOLET. به دلیل آن است که این تابع در یک بسته یا پکیج به نام ISOLET قرار داده شده است.

۳-۱۲-۳ تابع prepare20Newsgroups

```
function [data] = prepare20Newsgroups(newsgroupsFolder)
```

این تابع به صورت مستقل در کنار جعبه ابزار قرار داده شده است. کاربرد این تابع در تبدیل فایل‌های مربوط به دادگان 20Newsgroups برای استفاده در قسمت‌های مختلف می‌باشد. در واقع این تابع داده‌ها و برچسب‌های آموزشی و تست مربوط به این دادگان را در یک شئ از نوع کلاس DataStore قرار داده و شئ ساخته شده را برمی‌گرداند. برای فایل‌های مربوط به دادگان 20Newsgroups می‌توانید آن را از آدرس http://qwone.com/~jason/20Newsgroups/20news-bydate-matlab.tgz دانلود نمایید. ورودی این تابع نیز یک رشته شامل آدرس پوشه‌ای است که فایل‌های مربوط به دادگان 20Newsgroups در آن قرار دارد. نمونه‌ای از استفاده این تابع را در خط زیر مشاهده می‌کنید.

```
data = TwentyNewsgroups.prepare20Newsgroups('H:\DataSets\Text\The 20
Newsgroups data set\20news-bydate-matlab\');
```

علت صدا زدن این تابع با `TwentyNewsgroups.` به دلیل آن است که این تابع در یک بسته یا پکیج به نام `TwentyNewsgroups` قرار داده شده است.

٤ فهرست مراجع

[1] Y. Liu, S. Zhou, and Q. Chen, "Discriminative deep belief networks for visual data classification," *Pattern Recognition*, vol. 44, no. 10–11, pp. 2287–2296, Oct. 2011.

[2] H. Lee, C. Ekanadham, and A. Ng, "Sparse deep belief net model for visual area V2," *Advances in neural information processing systems*, vol. 20, pp. 873–880, 2008.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[4] V. Nair and G. Hinton, "3D object recognition with deep belief nets," *Advances in Neural Information Processing Systems*, vol. 22, pp. 1339–1347, 2009.

[5] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," in *Proceedings of the international conference on artificial intelligence and statistics*, 2009, vol. 5, pp. 448–455.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. 2006. Corr. 2nd printing. Springer, 2007.

[7] K. Swersky, B. Chen, B. Marlin, and N. de Freitas, "A tutorial on stochastic approximation algorithms for training Restricted Boltzmann Machines and Deep Belief Nets," in *Information Theory and Applications Workshop (ITA), 2010*, 2010, pp. 1 –10.

[8] M. A. Carreira-Perpinan and G. E. Hinton, "On contrastive divergence learning," in *Artificial Intelligence and Statistics*, 2005, vol. 2005, p. 17.

[9] H. Lee, "Unsupervised Feature Learning Via Sparse Hierarchical Representations," Stanford University, 2010.

[10] T. Tieleman and G. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Proceedings of the 26th Annual International Conference on Machine Learning*, New York, NY, USA, 2009, pp. 1033–1040.

[11] O. Breuleux, Y. Bengio, and P. Vincent, "Quickly Generating Representative Samples from an RBM-Derived Process," *Neural Computation*, vol. 23, no. 8, pp. 2058–2073, Apr. 2011.

[12] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[13] Y. Bengio, "Learning Deep Architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Jan. 2009.

[14] T. Tieleman, "Training restricted Boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the 25th international conference on Machine learning*, New York, NY, USA, 2008, pp. 1064–1071.

[15] Y. Bengio, A. Courville, and P. Vincent, "Unsupervised Feature Learning and Deep Learning: A Review and New Perspectives," *arXiv:1206.5538*, Jun. 2012.

[16] G. Hinton, "A practical guide to training restricted boltzmann machines," Machine Learning Group, University of Toronto, Technical report, 2010.

[17] R. Salakhutdinov, "Learning deep generative models," University of Toronto, 2009.

[18] M. A. Ranzato and G. E. Hinton, "Modeling pixel means and covariances using factorized third-order boltzmann machines," in *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 2551 –2558.

[19] B. A. Olshausen and others, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[20] N.-N. Ji, J.-S. Zhang, and C.-X. Zhang, "A sparse-response deep belief network based on rate distortion theory," *Pattern Recognition*, vol. 47, no. 9, pp. 3179–3191, Sep. 2014.

[21] M. A. Keyvanrad and M. M. Homayounpour, "Effective Sparsity Control in Deep Belief Networks using Normal Regularization Term," *submitted to Neural Networks*, 2015.

[22] H. Lee, Y. Largman, P. Pham, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," *Advances in neural information processing systems*, vol. 22, pp. 1096–1104, 2009.

[23] Y. Bengio, N. Chapados, O. Delalleau, H. Larochelle, X. Saint-Mleux, C. Hudon, and J. Louradour, "Detonation Classification from Acoustic Signature with the Restricted Boltzmann Machine," *Computational Intelligence*, vol. 28, no. 2, pp. 261–288, May 2012.
[24] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 791–798.
[25] G. E. Hinton, "To recognize shapes, first learn to generate images," in *Progress in Brain Research*, vol. Volume 165, T. D. and J. F. K. Paul Cisek, Ed. Elsevier, 2007, pp. 535–547.
[26] H. Larochelle and Y. Bengio, "Classification using discriminative restricted Boltzmann machines," in *Proceedings of the 25th international conference on Machine learning*, New York, NY, USA, 2008, pp. 536–543.
[27] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio, "Learning Algorithms for the Classification Restricted Boltzmann Machine," *J. Mach. Learn. Res.*, vol. 13, pp. 643–669, Mar. 2012.
[28] K. Swersky, "Inductive principles for learning Restricted Boltzmann Machines," University of Alberta, 2010.
[29] B. Mélykúti, "The mixing rate of Markov Chain Monte Carlo methods and some applications of MCMC simulation in bioinformatics," MSc thesis, 2006.
[30] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*, New York, NY, USA, 2009, pp. 609–616.
[31] R. Grosse, R. Raina, H. Kwong, and A. Y. Ng, "Shift-Invariance Sparse Coding for Audio Classification," presented at the Twenty-Third Conference on Uncertainty in Artificial Intelligence (UAI2007), Vancouver, BC, Canada, 2007.
[32] M. Elad, *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*, 1st Edition. Springer, 2010.
[33] Y. LeCun and C. Cortes, "MNIST handwritten digit database," *AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 1998.
[34] V. Nair, "Visual Object Recognition Using Generative Models of Images," University of Toronto, 2010.
[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278 –2324, Nov. 1998.
[36] M. A. Fanty and R. A. Cole, "Spoken Letter Recognition," presented at the Advances in Neural Information Processing Systems, 1991, pp. 220–226.

۵ ضمائم

۵-۱ اصلاحات دیگر انجام شده بر روی ماشین بولتزمن محدود

در این قسمت بعضی از اصلاحات دیگر مربوط به ماشین‌های بولتزمن محدود که در گزارش به آنها اشاره نشده بود، آورده شده است.

۵-۱-۱ استفاده از دسته‌های کوچک[۵۰]

بر اساس روابط لازم برای اصلاح وزن‌های ماشین بولتزمن محدود، محاسبه گرادیان می‌تواند با یک نمونه آموزشی انجام شود اما معمولا بهتر است تا داده آموزشی به دسته‌هایی با ۱۰ تا ۱۰۰ نمونه تقسیم شده و از این گروه‌های کوچک برای اصلاح وزن‌ها استفاده نماییم [۱۶]. این کار امکان استفاده از مزایای ضرب ماتریسی را فراهم می‌کند. بدین ترتیب از میانگین گرادیان مربوط به هر دسته برای اصلاح پارامترها استفاده می‌شود. نکته دیگر آن است که نباید اندازه این دسته‌ها را بزرگ در نظر گرفت چرا که هر چند تخمین پایدارتری از گرادیان را می‌دهد اما سرعت اصلاح وزنه‌ها را نیز کاهش می‌دهد. یک ایده مناسب برای ساخت دسته‌ها آن است که اگر دسته کلاس‌ها مشخص و هر کلاس دارای احتمال یکنواختی است، دسته‌ها به تعداد کلاس‌ها بوده و در هر دسته یک نمونه از هر کلاس موجود باشد.

۵-۱-۲ مقداردهی اولیه وزن‌ها و بایاس

وزن‌ها معمولا به یک مقدار کوچک تصادفی از یک گوسی با میانگین صفر و انحراف معیار ۰٫۰۱ مقداردهی اولیه می‌شوند [۱۶]. استفاده از مقادیر تصادفی بزرگتر، آموزش اولیه را سرعت می‌دهد اما ما را به مدلی نسبتاً نامناسب هدایت خواهد کرد. باید مطمئن بود که مقادیر اولیه وزن‌ها باعث نشوند تا بردارهای مشاهدات موجب شوند تا احتمال واحدهای مخفی نزدیک صفر یا یک باشند چراکه این کار سرعت یادگیری را به شدت کاهش می‌دهد.

معمولا یک مقدار مناسب برای بایاس در واحدهای مشاهده‌پذیر برای واحد i مقدار $\ln[\frac{p_i}{1-p_i}]$ است که در آن p_i نسبت فعال بودن واحد i در بردارهای مشاهدات است. با توجه به روابط زیر مشخص است که این کار موجب می‌شود تا در مراحل اولیه آموزش، واحدهای مخفی واحد i را با همان احتمال p_i فعال نمایند.

$$P(v_i = 1|h) = \sigma\left(a_i + \sum_j h_j w_{ij}\right) \xrightarrow{W\approx 0\ ,a_i=\ln[\frac{p_i}{1-p_i}]} \sigma(a_i) = p_i \qquad (۱-۵)$$

[۵۰] Mini-batch

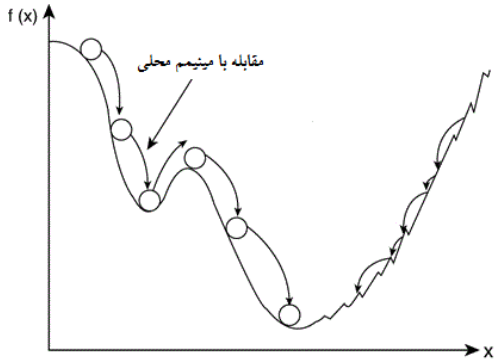
اما اگر از احتمال تنک بودن هدف t (یعنی بخواهیم واحدهای مخفی دارای احتمال فعال شدن $t \ll 1$ باشند) استفاده نماییم (که توضیح بیشتر آن در بخش ماشین بولتزمن تنک آمده است)، بهتر است مقادیر اولیه بایاس برای واحدهای مخفی را $\ln[\frac{t}{1-t}]$ در نظر بگیریم که با این مقدار، در مراحل اولیه آموزش، مشابه بایاس در نظر گرفته شده برای واحدهای مشاهده‌پذیر، احتمال فعال شدن واحدهای مخفی همان مقدار دلخواه خواهد بود. در غیر این صورت و اگر تنک بودن مد نظر ما نباشد، می‌توانیم از مقدار اولیه صفر برای این بایاس استفاده نماییم.

۵-۱-۳ به کارگیری ممنتم[۵۱]

ممنتم یک روش ساده برای افزایش سرعت یادگیری و همچنین جلوگیری از مینیمم‌های محلی است. روش ممنتم مطابق شکل ۵-۱ حرکت رو به پایین یک توپ سنگین به سمت یک دره را شبیه‌سازی می‌کند. در روش ممنتم، مشابه حرکت توپ در دره، پارامتری به نام سرعت تعریف می‌شود که در واقع برابر همان میزان تغییر در اصلاح پارامترها (یا $\Delta\theta$) است.در این روش سرعت جدید برابر جمع ضریبی از سرعت قبلی و میزان گرادیان در نقطه جدید است. بدین ترتیب می‌توان به هر یال یک جمع وزندهی شده از میزان اصلاح در این مرحله و مقدار بدست آمده در مرحله قبل، جمع کرد.

$$\Delta\theta(t) = m\,\Delta\theta(t-1) - \epsilon(< v_i h_j >_{data} - < v_i h_j >_{model}) \qquad (۵-۲)$$

که در آن θ پارامتر مدل، ϵ نرخ یادگیری و m ضریب ممنتم می‌باشد. بدین ترتیب اگر مقدار گرادیان نسبتا ثابت بماند، موجب می‌شود تا سرعت تغییر در آن جهت افزایش یافته و سرعت یادگیری بالا رود. مقدار معمول برای ضریب m برابر 0.9 است. اما از آنجا که معمولا در ابتدای یادگیری در ماشین بولتزمن محدود، به سبب مقادیر تصادفی اولیه پارامترهای مدل مقدار اندازه گرادیان بالا است، لذا بهتر است مقدار اولیه ضریب ممنتم را 0.5 درنظر گرفته و زمانی که سرعت کاهش خطای بازسازی نمونه‌ها کاهش یافت، مقدار آن را به 0.9 تنظیم نماییم [۱۶].



شکل ۵-۱ : نحوه حرکت در اصلاح وزن‌ها با کمک ممنتوم

[۵۱] Momentum

٥-١-٤ تضعیف وزن[٥٢]

تضعیف وزن با اضافه کردن یک عنصر اضافه به گرادیان معمولی صورت می‌گیرد. عنصر اضافه مشتق تابعی است که وزن‌های بزرگ را جریمه می‌کند. ساده‌ترین تابع جریمه $L2$ نام دارد که نصف جمع توان دوم مقادیر وزن است که به آن هزینه وزن[٥٣] می‌گویند.

چهار دلیل برای استفاده از تضعیف وزن در RBM وجود دارد [١٦]. اول آنکه تعمیم را برای داده جدید با کاهش بیش‌آموزش[٥٤] در داده آموزشی، افزایش می‌دهد. دوم آنکه موجب می‌شود تا میدان‌های دریافتی[٥٥] (یعنی وزن‌های متصل به یک واحد مخفی [٧]) برای واحدهای مخفی نرم‌تر بوده و با کوچک کردن وزن‌های بدون استفاده، بیشتر قابل تفسیر باشد. سوم آنکه واحدهای مخفی با وزن‌های زیاد در یادگیری که تقریبا همیشه فعال یا غیر فعال هستند را از بین می‌برد. البته یک روش مناسب‌تر برای جلوگیری از این مشکل استفاده از تنک‌سازی در RBM است.

اما دلیل چهارم، بهبود نرخ همگرایی (یعنی تعداد گام‌هایی که زنجیره مارکوف برای رسیدن به توزیع نسبتا پایدار لازم دارد [29]) در زنجیره مارکوف گیبز است. با وزن‌های کوچکتر، زنجیره مارکوف سریع‌تر همگرا می‌شود. روش یادگیری CD مبتنی بر حذف مشتقاتی است که از گام‌های آخر در زنجیره مارکوف بدست می‌آید [١٢]. لذا تضعیف وزن موجب می‌شود تا تخمین بهتری با این نرخ همگراییِ سریعتر بدست آید و مشتقات حذف شده مقدار کمی داشته باشند.

البته روش ساده‌تر دیگر آن است که یک مقدار بیشینه برای جمع قدرمطلق وزن‌های ورودی در هر واحد در نظر گرفته و در صورتی که از این مقدار تجاوز کرد، وزن‌ها را مجدداً مقیاس‌بندی نماییم.

همانطور که بیان گردید، تضعیف وزن تنها موجب جلوگیری از بیش‌آموزش نمی‌شود بلکه نرخ همگرایی در یادگیری CD را نیز بالا برده و تخمین بهتری از بیشترین درستنمایی به ما می‌دهد. لذا حتی اگر به دلیل داشتن نمونه‌های آموزشی زیاد، مشکل بیش‌آموزش وجود نداشته باشد، همچنان استفاده از تضعیف وزن می‌تواند مفید باشد.

---

[٥٢] Weight-decay

[٥٣] Weight cost

[٥٤] Overfitting

[٥٥] Receptive field

۵-۱-۵ ماشین بولتزمن محدود کانولوشنال[۵۶]

به دو دلیل استفاده از ماشین‌های بولتزمن محدود در کاربرد پردازش تصویر برای تصاویری با اندازه واقعی (مثلا ۲۰۰*۲۰۰ پیکسل) یک مساله چالش‌برانگیز است. اول آنکه در تصاویر بزرگ، شبکه باید بتواند خود را با اندازه تصویر تنظیم نماید. دوم آنکه شئ مورد نظر ممکن است در هر جایی از تصویر واقع شود و لذا ارائه یک بازنمایی از تصویر که مستقل از جابجایی‌های کم در ورودی باشد نیز از اهداف مهم است.

در روش مورد نظر در این بخش سعی می‌شود تا با به اشتراک‌گذاری تشخیص‌دهنده‌های ویژگی[۵۷] (همان واحدهای مخفی) در تمام موقعیت‌ها به هدف مورد نظر یعنی مستقل بودن از جابجایی رسید [۳۰]. این کار از آن جهت است که یک تشخیص‌دهنده ویژگی که اطلاعات مفیدی را از یک بخش تصویر استخراج می‌کند، می‌تواند اطلاعات مشابهی را از مکان‌های دیگر هم بدست بیاورد. لذا مدل می‌تواند تصاویر بزرگ را با استفاده از فقط تعداد کمی تشخیص‌دهنده ویژگی بازنمایی کند. روش ماشین بولتزمن محدود کانولوشنال نام روشی است که سعی می‌کند تا برای تصاویر با اندازه واقعی و به صورت مستقل از جابجایی، به نتایج مناسبی برسد.

در این بخش برای سادگی چند فرض در نظر گرفته شده است. اول آنکه فرض می‌شود تمام ورودی‌ها تصاویری با ابعاد $N_V \times N_V$ باشد هرچند که در عمل نیازی به وجود این شرط نیست. همچنین فرض می‌گردد که همه واحدها باینری باشند. از علامت * برای کانولوشن استفاده کرده و علامت • برای ضرب داخلی به کار گرفته می‌شود. همچنین در بالای یک آرایه مد قرار داده‌شده ($\tilde{A}$) تا نشان داده شود که آرایه به صورت افقی و عمودی معکوس شده است.

یک CRBM شبیه RBM است اما وزن‌های بین لایه مخفی و مشاهده‌پذیر آن در تمام موقعیت‌های تصویر به اشتراک گذاشته شده است. یک CRBM دارای دو لایه است: یک لایه ورودی V و یک لایه مخفی H. لایه ورودی دارای $N_V \times N_V$ واحد باینری است. لایه مخفی نیز شامل K گروه است که هر گروه دارای $N_H \times N_H$ واحد مخفی است که در کل می‌توان گفت که دارای $N_H^2 \times K$ واحد مخفی می‌باشد. هر یک از K گروه به یک فیلتر $N_W \times N_W$ ($N_W \triangleq N_V - N_H + 1$) متعلق است که وزن‌های فیلتر در بین واحدهای مخفی در گروه به اشتراک گذاشته شده است. بعلاوه هر گروه مخفی دارای یک بایاس b_k است و تمام واحدهای مشاهده‌پذیر نیز دارای یک بایاس a مشترک هستند. بدین ترتیب تابع انرژی $E(v,h)$ به صورت زیر تعریف می‌شود [۹]:

$$E(v,h) = -\sum_{k=1}^{K}\sum_{i,j=1}^{N_H}\sum_{r,s=1}^{N_W} h_{ij}^{k} W_{rs}^{k} v_{i+r-1,j+s-1} - \sum_{k=1}^{K} b_k \sum_{i,j=1}^{N_H} h_{ij}^{k} - a\sum_{i,j=1}^{N_V} v_{ij} \qquad (۳-۵)$$

[۵۶] Convolutional RBM (CRBM)

[۵۷] Feature detector

$$P(v,h)=\frac{1}{Z}\exp\big(-E(v,h)\big) \qquad (۴-۵)$$

با استفاده از عملگرهای تعریف شده می‌توان تابع انرژی را به صورت زیر بازنویسی کرد:

$$E(v,h)=-\sum_{k=1}^{K}h^k \bullet \left(\tilde{W}^k * v\right)-\sum_{k=1}^{K}b_k\sum_{i,j=1}^{N_H}h_{ij}^k-a\sum_{i,j=1}^{N_V}v_{ij} \qquad (۵-۵)$$

همانند RBM استاندارد، در اینجا نیز می‌توان نمونه‌برداری گیبز را با کمک روابط زیر داشت.

$$P\left(h_{ij}^k=1|v\right)=g\left(\left(\tilde{W}^k * v\right)_{ij}+b_k\right) \qquad (۶-۵)$$
$$P\left(v_{ij}=1|h\right)=g\left(\left(\sum_k W^k * h^k\right)_{ij}+a\right)$$

که در آن g تابع لاجستیک است.

ادغام احتمالاتی بیشینه[۵۸]

برای یادگیری بازنمایی‌های سطح بالا، CRBMها همانند شبکه باور عمیق به صورت پشته در می‌آیند. معماری استفاده شده در این نوع ترکیب، استفاده از یک عملیات به نام ادغام بیشینه احتمالاتی است [۹].

در حالت کلی تشخیص‌دهنده‌های ویژگی سطح بالاتر، نیاز به اطلاعات ناحیه‌های بزرگتری از ورودی دارند. روش‌های موجود در بازنمایی مستقل از جابجایی مانند شبکه‌های کانولوشنال، معمولا شامل دو لایه کنار هم هستند. "لایه تشخیص"[۵۹] که خروجی آن با کانوالو تشخیص‌دهنده ویژگی با لایه قبل محاسبه می‌شود و "لایه ادغام"[۶۰] که بازنمایی لایه تشخیص را با یک فاکتور ثابت فشرده می‌کند. به طور دقیقتر هر واحد در لایه ادغام، بیشینه فعالیت واحدها در یک محدوده کوچک از لایه تشخیص را محاسبه می‌کند. فشرده‌سازی بازنمایی با ادغام بیشینه اجازه می‌دهد تا بازنمایی‌های لایه بالاتر نسبت به جابجایی‌های کم در ورودی مستقل بوده و بار محاسباتی کم شود.

روش ادغام بیشینه فقط برای معماری‌های قطعی[۶۱] و روبه‌جلو[۶۲] پیشنهاد شده است و از آنجا که ادغام بیشینه یک عملگر قطعی است، اجرای استنتاج احتمالاتی (مثلا محاسبه احتمال‌های پسین) در آن سخت است. از طرفی ما

---

۵۸ Probabilistic max-pooling

۵۹ Detection layer

۶۰ Pooling layer

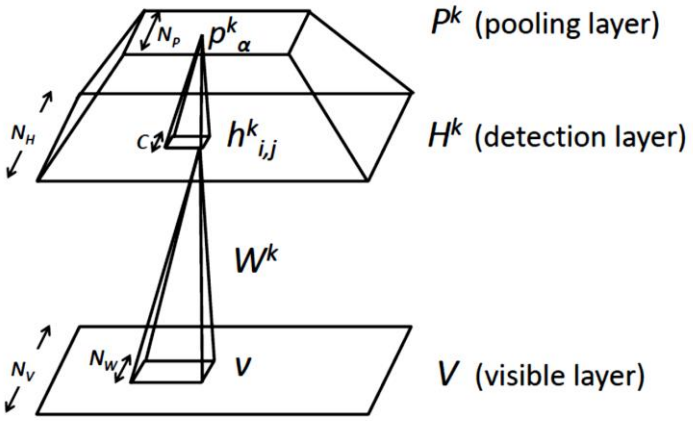
۶۱ Deterministic

با یک مدل مولد[۶۳] از داده‌ها سروکار داریم که استنتاجی کاملا احتمالاتی دارد و لذا این مدل مولد طوری طراحی خواهد شد که استنتاج در آن شامل رفتاری مشابه ادغام بیشینه باشد.

برای سادگی در علائم استفاده شده، یک مدل با لایه مشاهده‌پذیر V ، یک لایه تشخیص H و یک لایه ادغام P مطابق شکل ۲-۵ در نظر گرفته می‌شود. لایه‌های ادغام و تشخیص هردو دارای K گروه از واحدها هستند و هر گروه از لایه ادغام دارای $N_P \times N_P$ واحد باینری است. به ازای هر $k \in \{1, \dots, K\}$ ، لایه ادغام P^k بازنمایی لایه تشخیص H^k را با فاکتور c در هر بعد فشرده می‌کند، که c یک عدد صحیح کوچک مثل ۲ یا ۳ است. به عبارت دیگر لایه تشخیص H^k به بلوک‌های $c \times c$ تقسیم می‌شود و هر بلوک α به دقیقا یک واحد باینری p_α^k در لایه ادغام وصل می‌شود. بدین ترتیب می‌توان یک بلوک در واحدهای مخفی را به این صورت تعریف نمود :

$$B_\alpha \triangleq \{(i,j): h_{ij} \text{ belongs to the block } \alpha\} \qquad (۷-۵)$$

واحدهای تشخیص در بلوک B_α و واحد ادغام p_α به صورتی به هم وصل‌اند که این شرط را ایجاب می‌کند که حداکثر یکی از واحدهای تشخیص می‌تواند روشن باشد و واحد ادغام در صورتی روشن خواهد بود که یک واحد تشخیص روشن باشد. با اضافه شدن این شرط، می‌توان به صورت کارآمد از شبکه نمونه‌برداری کرد بدون اینکه نیاز باشد تا تمام 2^{c^2} حالت را بررسی نماییم که در ادامه خواهد آمد.



شکل ۲-۵ : ماشین بولتزمن محدود کانولوشنال با ادغام بیشینه احتمالاتی. برای سادگی، تنها گروه k ام از لایه تشخیص و لایه ادغام نمایش داده شده است. یک CRBM استاندارد متناظر با ساختاری است که تنها دارای لایه مشاهده‌پذیر و لایه تشخیص (یا لایه مخفی) می‌باشد [۹].

با این شرط می‌توان این $c^2 + 1$ واحد را به عنوان یک متغیر تصادفی در نظر گرفت که یکی از $c^2 + 1$ مقدار ممکن را می‌تواند بگیرد: یک مقدار به ازای روشن شدن هر یک از واحدهای تشخیص و یک مقدار نشاندهنده خاموش بودن همه واحدها.

---

[۶۲] Feed-forward

[۶۳] Generative

می‌توان به طور رسمی تابع انرژی این ادغام بیشینه احتمالاتی ساده شده را برای CRBM به صورت زیر نوشت:

$$E(v,h) = -\sum_{k=1}^{K}\sum_{i,j=1}^{N_H}\left(h_{\mathrm{ij}}^{k}\left(\widetilde{W}^{k} * v\right)_{ij} + b_k h_{\mathrm{ij}}^{k}\right) - a\sum_{i,j=1}^{N_V} v_{ij} \qquad (۸-۵)$$

$$subject\ to \sum_{(i,j)\in B_\alpha} h_{\mathrm{ij}}^{k} \le 1, \qquad \forall k,\alpha$$

اکنون به سراغ نمونه‌برداری از لایه تشخیص H و لایه ادغام P به شرط لایه مشاهده‌پذیر V می‌رویم. گروه k سیگنال زیر را از لایه V دریافت می‌کند:

$$I\left(h_{\mathrm{ij}}^{k}\right) \triangleq b_k + \left(\widetilde{W}^{k} * v\right)_{ij} \qquad (۹-۵)$$

اکنون هر بلوک به طور مستقل به صورت یک تابع چندجمله‌ای از ورودی‌ها، نمونه‌برداری می‌شود. فرض کنید h_{ij}^{k} یک واحد مخفی است که در بلوک α قرار دارد (یعنی $(i,j) \in B_\alpha$) و افزایش انرژی که موجب روشن شدن واحد h_{ij}^{k} می‌گردد برابر $-I\left(h_{\mathrm{ij}}^{k}\right)$ باشد، آنگاه می‌توان احتمال شرطی را به صورت زیر تعریف نمود:

$$P\left(h_{\mathrm{ij}}^{k} = 1|v\right) = \frac{\exp\left(I\left(h_{ij}^{k}\right)\right)}{1 + \sum_{\left(i',j'\right)\in B_\alpha}\exp\left(I\left(h_{i'j'}^{k}\right)\right)} \qquad (۱۰-۵)$$

$$P\left(\mathrm{p}_{\alpha}^{k} = 0|v\right) = \frac{1}{1 + \sum_{\left(i',j'\right)\in B_\alpha}\exp\left(I\left(h_{i'j'}^{k}\right)\right)} \qquad (۱۱-۵)$$

نمونه‌برداری از لایه مشاهده‌پذیر V به شرط لایه مخفی H مشابه آنچه قبلا گفته شد، انجام می‌شود.

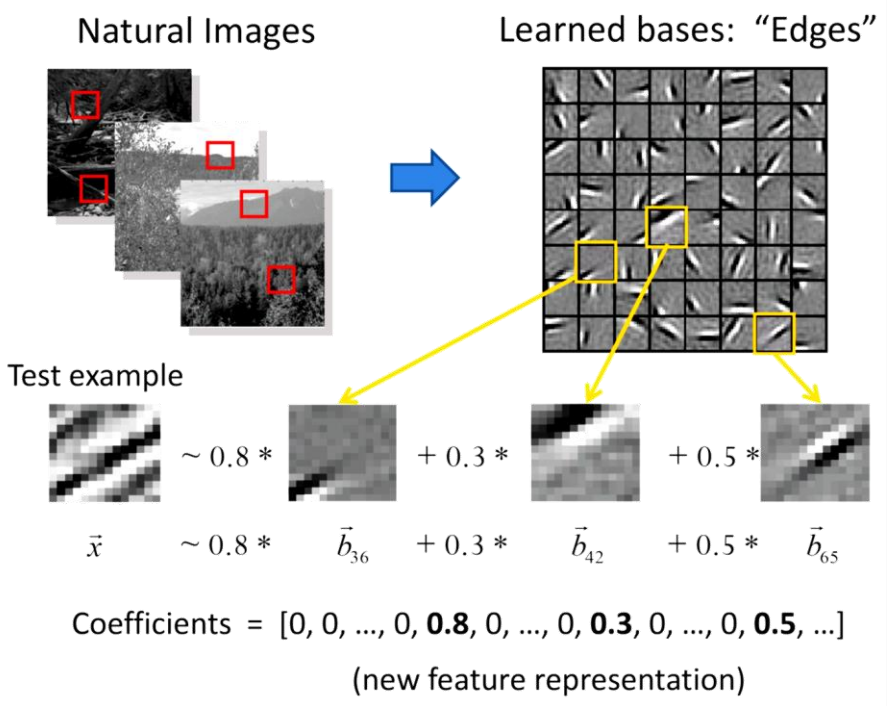
در حالت کلی مدل‌هایی مانند این مدل که بیش تکمیل[۶۴] هستند یعنی بازنمایی آن از سایز ورودی خیلی بیشتر است، خطر رسیدن به راه حل‌های بدیهی[۶۵] در آن‌ها وجود دارد. مثل اینکه تشخیص دهنده‌های ویژگی، یک پیکسل را بازنمایی نمایند. یک راه حل رایج آن است که به اجبار بازنمایی را تنک نمود تا تنها نسبت کمی از واحدها فعال باشند [۳۰]. بدین ترتیب در [۳۰] ,[۹] با کمک روش اول تنک‌سازی [۲] که در ۲-۱-٤ مطرح نمودیم، سعی شده تا از این مشکل اجتناب نماید.

[۶۴] Overcomplete

[۶۵] Trivial

۵-۲ کدینگ تنک

روش کدینگ تنک[۶۶] می‌تواند به عنوان روشی برای آموزش ویژگی به کار برده شود. دراین روش سعی می شود تا در یک فرآیند بهینه‌سازی با ساختن یک دیکشنری از روی داده‌های برچسب نخورده، داده‌های جدید بعدی را با ترکیب خطی این دیکشنری بسازد. در واقع ضرایب این عناصر دیکشنری برای هر داده می‌تواند به عنوان ویژگی استفاده شود. یک ویژگی مهم که در این روش استفاده می‌شود، تنک بودن کدینگ بدست آمده است. یعنی سعی می‌شود تا عناصر دیکشنری به نحوی انتخاب شوند که بیشتر ضرایب در حین یافتن یک ترکیب خطی برای هر داده، مقدار صفر بگیرند. در شکل ۵-۳ نمونه‌ای از استفاده از یک دیکشنری ساخته شده، نشان داده می‌شود.



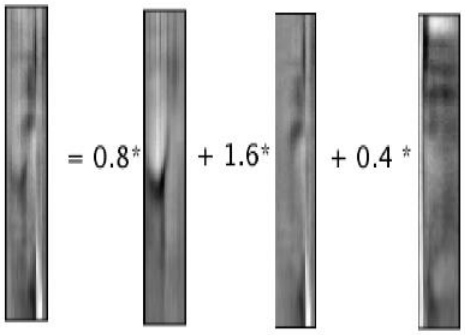
شکل ۵-۳: استفاده از یک دیکشنری ساخته شده برای ارائه تنک یک قسمت از تصویر [9]

برای فهم بهتر این روش می‌توان آن را با روش تبدیل فوریه سریع[۶۷] مقایسه نمود. در تبدیل فوریه سریع ما سعی می‌کنیم تا یک سیگنال را بر اساس ترکیب خطی از توابع سینوسی و کسینوسی بیان کنیم. اما شاید لزوما سینوس و کسینوس توابع پایه خوبی برای داده‌های ما نباشند. لذا با روش کدینگ تنک، عناصر پایه بهتری را خواهیم یافت. این روش برای کاربردهای تصویری و صوتی استفاده می‌شود. در کاربردهای تصویری، عناصر دیکشنری خود تصویر و یا قسمت‌هایی از آن هستند. در صوت و گفتار نیز ابتدا آن را مثلا به حوزه فرکانس برده و مورد استفاده قرار می‌گیرند [31]. در شکل ۵-٤ نمونه‌ای از اسپکتروگرام و عناصر دیکشنری که برای ساخته شدن آن در نظر گرفته شده مشاهده می‌شود.

---

[۶۶] Sparse Coding

[۶۷] Fast Fourier Transform (FFT)



شکل ۵-٤: نمونه‌ای از یک اسپکتروگراف که به صورت ترکیب خطی از عناصر یک دیکشنری بیان شده است [31].

در کدینگ تنک، مساله اصلی حل رابطه بهینه‌سازی زیر است:

$$\min_{\alpha}\|\alpha\|_0 \ \ subject\ to\ x = D\alpha \qquad (۱۲-۵)$$

که در آن x ماتریس مشاهدات، D دیکشنری و α ضرایب لازم برای این دیکشنری جهت رسیدن به بردار مشاهدات است. البته گاهی اوقات نیز هدف در مساله بهینه‌سازی به صورت کمینه کردن $\|x - D\alpha\|$ مطرح می‌شود.

در حالت کلی مساله یادگیری دیکشنری را هم می‌توان به این صورت بیان نمود [۳۲]. فرض کنید خطای مدل یا ϵ را می‌دانیم و هدف رسیدن به تخمین D می‌باشد. مساله بهینه‌سازی به صورت زیر تعریف می‌شود:

$$\min_{D,\{\alpha\}_{i=1}^{M}} \sum_{i=1}^{M}\|\alpha_i\|_0 \ \ subject\ to\ \|X_i - D\alpha_i\|_2 \le \epsilon\ , 1 \le i \le M \qquad (۱۳-۵)$$

این مساله بیان می‌کند که در سیگنال $\boldsymbol{X_i}$ با تنک‌ترین بازنمایی $\boldsymbol{\alpha_i}$ بر روی دیکشنری ناشناخته $\boldsymbol{D}$، می‌خواهیم بازنمایی‌ها و دیکشنری مناسب را بیابیم. اما جایگاه قیود و تابع جریمه در رابطه (۵-۱۳) می‌تواند جابجا شود و شرط را تنک بودن گرفته و بهترین تطبیق را برای آن پیدا کنیم.

$$\min_{D,\{\alpha\}_{i=1}^{M}} \sum_{i=1}^{M}\|\boldsymbol{X_i} - \boldsymbol{D\alpha_i}\|_2 \ \ subject\ to\ \|\boldsymbol{\alpha_i}\|_0 \le k_0\ , 1 \le i \le M \qquad (۱٤-۵)$$

۵-۳ دادگان

در این بخش دادگان‌های MINIST، ISOLET و 20Newsgroups معرفی شده است.

۵-۳-۱ دادگان MNIST

دادگان MNIST [33] شامل تصاویر دیجیتالی از ارقام دستنویس است که متعلق به ده دسته (۰ تا ۹) می‌باشد. اندازه ارقام، نرمال‌سازی شده و در مرکز تصاویر ۲۸*۲۸ قرار گرفته‌اند. نمونه‌هایی از آن در شکل ۵-۵ مشاهده می‌شود. نمونه‌های تست و آموزشی توسط سازنده آن از قبل تعیین و تقسیم شده است. مجموعه آموزشی در

مجموع شامل ۶۰۰۰۰ تصویر و مجموعه تست شامل ۱۰۰۰۰ تصویر می‌باشد. پیکسل‌ها دارای مقادیر حقیقی در بازه [0,1] هستند که البته بیشتر مقادیر آنها در لبه‌های بازه واقع می‌شوند [34].

شکل ۵-۵: نمونه‌های نرمال‌سازی شده در دادگان MNIST [35]

۵-۳-۲ دادگان ISOLET

دادگان ISOLET [۳۶] شامل گفتار ۱۵۰ گوینده است که هر یک از حروف زبان انگلیسی را ۲ بار تکرار کرده و ۶۱۷ ویژگی از هر نمونه استخراج شده است. در کل این دادگان شامل ۶۲۳۸ نمونه آموزشی و ۱۵۵۹ نمونه تست است. از آنجا که هدف این دادگان تشخیص صوتی حروف زبان انگلیسی است، لذا شامل ۲۶ دسته مختلف برای دسته بندی است.

۵-۳-۳ دادگان 20Newsgroups

دادگان 20Newsgroups[۶۸] یک دادگان متنی است که شامل ۲۰ موضوع خبری مختلف می‌باشد که هر یک از نمونه‌ها به یکی از این ۲۰ موضوع برچسب خورده‌اند. این دادگان یکی از رایجترین دادگان‌ها در کاربرد متنی برای تکنیک‌های یادگیری ماشین مانند دسته‌بندی و یا خوشه‌بندی متون است.

ما در آزمایشات خود ورژنی از این دادگان را با ۱۸۷۷۴ سند استفاده نمودیم[۶۹] که در آن دادگان آموزشی و تست (۶۰٪ داده آموزشی و ۴۰٪ داده تست) در زمان‌های مختلفی جمع‌آوری شده‌اند. همچنین ما در آزمایشات از ۵۰۰۰ کلمه پرکاربرد مورد استفاده در این متون، به عنوان ویژگی‌های باینری استفاده نمودیم.

[۶۸] http://qwone.com/~jason/20Newsgroups/

[۶۹] http://qwone.com/~jason/20Newsgroups/20news-bydate-matlab.tgz

واژه نامه

Auto Encoder	کدگذار خودکار
backpropagation	انتشار خطای رو به عقب
Boltzmann Machine (BM)	ماشین بولتزمن
Contrastive Divergence (CD)	واگرایی متقابل
Convolutional Deep Belief Network (CDBN)	شبکه باور عمیق کانولوشنال
Convolutional RBM (CRBM)	ماشین بولتزمن محدود کانولوشنال
Deep Belief Network (DBN)	شبکه باور عمیق
Deep Learning	یادگیری عمیق
Detection layer	لایه تشخیص
Deterministic	قطعی
Discriminative Deep Belief Network	شبکه باور عمیق تمایزی
Discriminative RBM (DRBM)	ماشین بولتزمن محدود تمایزی
Discriminative training	آموزش تمایزی
Fantasy Particle	ذره خیالی
Fast Persistent Contrastive Divergence (FPCD)	واگرایی متقابل پایدار سریع
Fast weight	وزن سریع
Feature detector	تشخیص‌دهنده ویژگی
Feed-forward	روبه‌جلو
Firing rate	نرخ فعالیت
Fitness function	تابع برازندگی
Fixed size	سایز ثابت
Free energy	انرژی آزاد
Gaussian RBM	ماشین بولتزمن محدود گوسی
Generalized RBM	RBM عمومی
Generative	مولد
Generative training	آموزش از نوع مولد
Gibbs sampling	نمونه‌برداری گیبز
Hybrid discriminative\generative training	آموزش ترکیبی تمایزی/مولد
Independent Component Analysis (ICA)	آنالیز مولفه های مستقل
Markov Random Field (MRF)	میدان تصادفی مارکوف
Maximum A Posteriori probability estimate (MAP)	تخمین احتمال پسین بیشینه

Mini-batch	دسته کوچک
Momentum	ممنتوم
Negative phase	فاز منفی
Neural Network (NN)	شبکه عصبی
Persistent Contrastive Divergence (PCD)	واگرایی متقابل پایدار
Persistent Contrastive Divergence with Partial Smoothing (PCD PS)	واگرایی متقابل پایدار با هموارسازی جزئی
Pooling layer	لایه ادغام
Positive phase	فاز مثبت
posterior distribution of underlying explanatory factors	توزیع پسین فاکتورهای توصیف‌گر اصلی
Principal Component Analysis (PCA)	آنالیز مولفه های اسصلی
Probabilistic max-pooling	ادغام احتمالاتی بیشینه
Raw data	داده خام
Receptive field	میدان دریافتی
Restricted Boltzmann Machine (RBM)	ماشین بولتزمن محدود
Softmax units	واحدهای نرم بیشینه
Sparse Coding	کدگذاری تنک
Sparsifying	تنک سازی
Stack	پشته
Stochastic Maximum Likelihood (SML)	درست‌نمایی بیشینه احتمالاتی
Support Vector Machines (SVM)	ماشین بردار پشتیبان
Trivial	بدیهی
Weight cost	هزینه وزن
Weight decay	تضعیف وزن