Ch.2 ニューラルネットワークの基礎

§2.1 線形回帰モデル

2.1.1 最小二乗法による学習

$x \in \mathbb{R}$: 入力, $y \in \mathbb{R}$: ラベル.

$\phi(x) = (\phi_1(x), \cdots, \phi_M(x))^T$: 特徴量関数 (feature function)

$\phi_i : \mathbb{R} \to \mathbb{R}$: 基底関数 (basis function)

モデル : $y = w^T \phi(x)$ $(w \in \mathbb{R}^M$: パラメータ)

・パラメータについて線形なので線形モデル.

・N個の入力集合 $\mathcal{X} = \{x_1, \cdots, x_N\}$,

ラベル集合 $\mathcal{Y} = \{y_1, \cdots, y_N\}$ を学習用データセットとする:

$\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$.

$\mathcal{D}$ に対して最も"当てはまりが良い"線形モデルのパラメータは?

・"当てはまりが良い" = "モデルによる予測と各ラベルとの誤差2乗和が最小になる"

と考える.

誤差関数 (error function) : $E(w) := \frac{1}{2} \sum_{n=1}^{N} (y_n - w^T \phi(x_n))^2$

$\to w_{LS} := \underset{w}{\operatorname{argmin}}\, E(w)$ とすればよい. 最小2乗法.

・w_{LS} を求める. $E(w)$ は凸関数なので, $\nabla_w E(w) = 0$ となる

w が w_{LS}.

- ベクトルでの微分.

$$\nabla_{\boldsymbol{w}} f(\boldsymbol{w}) := \begin{pmatrix} \frac{\partial}{\partial w_1} f(\boldsymbol{w}) \\ \vdots \\ \frac{\partial}{\partial w_M} f(\boldsymbol{w}) \end{pmatrix}.$$

公式.

(1) $\nabla_{\boldsymbol{w}} (\boldsymbol{a}^T \boldsymbol{w}) = \nabla_{\boldsymbol{w}} (\boldsymbol{w}^T \boldsymbol{a}) = \boldsymbol{a}$.

(2) $\nabla_{\boldsymbol{w}} (\boldsymbol{w}^T A \boldsymbol{w}) = (A + A^T)\boldsymbol{w}$.

pf. (1) $\boldsymbol{a}^T \boldsymbol{w} = \sum_{i=1}^{M} a_i w_i$. $\quad \frac{\partial}{\partial w_i} \boldsymbol{a}^T \boldsymbol{w} = \sum_{j=1}^{M} a_j \frac{\partial w_j}{\partial w_i} = \sum_{j=1}^{M} a_j \delta_{ji} = a_i$.

(2) $\boldsymbol{w}^T A \boldsymbol{w} = \sum_{i=1}^{M} \sum_{j=1}^{M} a_{ij} w_i w_j$.

$$\frac{\partial}{\partial w_k} (\boldsymbol{w}^T A \boldsymbol{w}) = \sum_{i=1}^{M} \sum_{j=1}^{M} a_{ij} \left(\frac{\partial w_i}{\partial w_k} w_j + w_i \frac{\partial w_j}{\partial w_k} \right)$$

$$= \sum_{i=1}^{M} \sum_{j=1}^{M} a_{ij} w_j \delta_{ik} + \sum_{i=1}^{M} \sum_{j=1}^{M} a_{ij} w_i \delta_{jk}$$

$$= \sum_{j=1}^{M} a_{kj} w_j + \sum_{i=1}^{M} a_{ik} w_i$$

$$= (A\boldsymbol{w})_k + (A^T \boldsymbol{w})_k$$

$$= \left((A + A^T)\boldsymbol{w} \right)_k .$$

対称行列

■

- $E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(y_n^2 - 2 y_n \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) + \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{w} \right)$ より,

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \left(-2 y_n \boldsymbol{\phi}(\boldsymbol{x}_n) + 2 \boldsymbol{\phi}(\boldsymbol{x}_n) \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{w} \right)$$

$$= -\sum_{n=1}^{N} y_n \boldsymbol{\phi}(\boldsymbol{x}_n) + \sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_n) \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{w}.$$

$\nabla_w E(w_{LS}) = 0$ より

$$\sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T w_{LS} = \sum_{n=1}^{N} y_n \phi(x_n).$$

行列 $\sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T$ が正則であれば，

$$w_{LS} = \left(\sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T \right)^{-1} \sum_{n=1}^{N} y_n \phi(x_n).$$

- 学習済みモデルによる予測は，x_* に対して

$$y_* = w_{LS}^T \phi(x_*)$$

を出力する．

2.1.2 基底関数の選択

- $\phi(x) = (x^{M-1}, x^{M-2}, \cdots, x, 1)$

M を変えればいろいろな多項式でフィッティングできる．

基底関数の種類を変えることもできる．

- ラベルが D 次元 ($D>1$) でも同様にモデル化できる：

$$y = W\phi(x)$$

2.1.3 過剰適合と正則化．

- どういうモデルを使えばよいか？（モデル選択）

複雑すぎるモデルだと，データの特徴的な傾向をうまく捉えられない（過剰適合 : overfitting）

2.1.3.1 正則化項.

- L2正則化 (L2 regularization) : overfitting を防ぐ.

ペナルティ項 $\Omega_{L2}(w) = \frac{1}{2} w^T w \quad (= \frac{1}{2}\|w\|^2)$ を

誤差関数に加えたコスト関数　　ペナルティの強さを制御.

$$J(w) := E(w) + \lambda \Omega_{L2}(w) \qquad (\lambda > 0 : パラメータ)$$

を定義する.　　↑ w のとりうる値に制限をかける効果. 正則化項

→ $w_{L2} := \underset{w}{\operatorname{argmin}}\, J(w)$ で パラメータを決定.

リッジ回帰 (ridge regression)

- w_{L2} を求める. $J(w)$ も凸関数.

$$\nabla_w J(w) = \nabla_w E(w) + \lambda \nabla_w \Omega_{L2}(w).$$

$$\nabla_w \Omega_{L2}(w) = w.$$

$$\rightarrow \nabla_w J(w_{L2}) = -\sum_{n=1}^{N} y_n \phi(x_n) + \sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T w_{L2} + \lambda I_M w_{L2} = 0.$$

$$\therefore w_{L2} = \left(\sum_{n=1}^{N} \phi(x_n)\phi(x_n)^T + \lambda I_M \right)^{-1} \sum_{n=1}^{N} y_n \phi(x_n)$$

↑ これは必ず正則.

(∵ 行列 $\sum \phi(x_n)\phi(x_n)^T$ は非負定値行列.
→ 固有値は非負.
λI_M を足して 固有値が 正になるので
この行列の行列式は正.)

→ 過度な大きさの w がペナルティ項により抑制. 学習データに対して

極端に当てはまるのを防ぐ (正則化 : regularization)

・L1正則化 (L1 regularization) / LASSO

正則化項 : $\Omega_{L1}(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_{m=0}^{M-1} |w_m|$.

→ $\boldsymbol{w}$ がスパースになりやすい.

2.1.3.2 正則化による学習の問題点.

まだ問題が残っている.

1. 適切な ϕ の設定方法が不明

→ NNをつかう. Gauss過程をつかう.

2. 選んだ特徴量がデータに合っているかを判定しにくい.

→ Cross varidation

3. 正則化項 Ω の設定指針が不明瞭

4. 予測の不確実性を表現できない.

§2.2 ニューラルネットワーク

- NN：基底関数の中にパラメータをおくことで，データから基底関数自体も学習．

2.2.1 順伝播型ニューラルネットワーク．

2.2.1.1 2層の順伝播型 NN

- 順伝播型ニューラルネットワーク（feedforward NN）

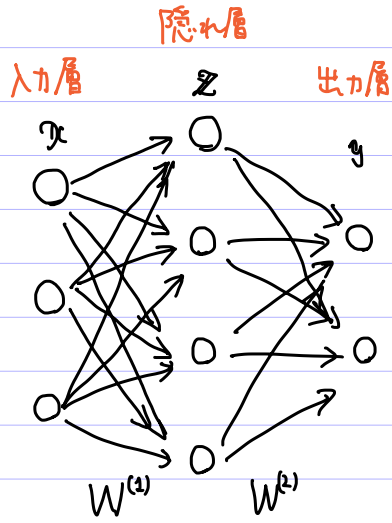
Input：$x_n \in \mathbb{R}^{H_0}$．　Output：$y_n \in \mathbb{R}^D$．

モデル：

活性 activation

$$y_{n,d} = \sum_{h_1=1}^{H_1} w^{(2)}_{d,h_1} \phi\left(\sum_{h_0=1}^{H_0} w^{(1)}_{h_1,h_0} x_{n,h_0} \right)$$

↑ 重みパラメータ



行列表記：$y_n = W^{(2)} \phi.(W^{(1)} x_n)$．

↑ 要素ごとに適用

2.2.1.2 さまざまな活性化関数．

- ϕ：基底関数のことを NN では活性化関数（activation func.）という．

- シグモイド関数：$Sig(x) = \dfrac{1}{1+e^{-x}}$．

- 双曲線正接関数：$\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$

$\tanh(x) = 2Sig(2x) - 1$

pf. $2Sig(2x) - 1 = \dfrac{2}{1+e^{-2x}} - \dfrac{1+e^{-2x}}{1+e^{-2x}} = \dfrac{1-e^{-2x}}{1+e^{-2x}}$

$= \dfrac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x)$．

$\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2}$

・標準正規分布の累積分布関数：$\Phi(x) = \int_{-\infty}^{x} \underline{\mathcal{N}(t|0,1)}\, dt$.

・Gaussの誤差関数：$Erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

$\Phi(x) = \frac{1}{2}\left(1 + Erf\left(\frac{x}{\sqrt{2}}\right)\right)$.

pf. (r.h.s.) $= \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^{\frac{x}{\sqrt{2}}} e^{-t^2} dt$　　$t = \frac{1}{\sqrt{2}}u$　　$dt = \frac{1}{\sqrt{2}} du$.　　$\begin{array}{c|cc} t & 0 & \to \frac{x}{\sqrt{2}} \\ \hline u & 0 & \to x \end{array}$

$= \frac{1}{2} + \frac{1}{\sqrt{\pi}} \int_0^x \frac{1}{\sqrt{2}} e^{-\frac{1}{2}u^2} du$.

ここで，$\int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} dt = \frac{1}{2}$ なので，

$(r.h.s.) = \int_{-\infty}^{0} \mathcal{N}(t|0,1)\, dt + \int_0^x \mathcal{N}(t|0,1)\, dt$

$= \int_{-\infty}^{x} \mathcal{N}(t|0,1)\, dt$

$= \Phi(x)$.

・正規化線形関数（ReLU）：$ReLU(x) = \max(x, 0)$.

・指数線形関数（ELU）：$ELU(x) = \begin{cases} x & (x > 0) \\ \alpha(e^x - 1) & (x \leq 0) \end{cases}$

2.2.1.3 NNで表現される関数の例.

・NNの普遍性定理（universal approximation theorem）：

層数2のfeedforward NNで H_1 の数を大きくすることで任意の連続関数を近似できる.

2.2.1.4 複数層をもつ feedforward NN.

- 多層化： $d = 1, \dots, D(=H_L)$ として.

$$y_{n,d} = \sum_{h_{L-1}=1}^{H_{L-1}} w_{d,h_{L-1}}^{(L)} \phi\left(\sum_{h_{L-2}=1}^{H_{L-2}} w_{h_{L-1},h_{L-2}}^{(L-1)} \cdots \phi\left(\sum_{h_0=1}^{H_0} w_{h_1,h_0}^{(1)} x_{n,h_0} \right) \cdots \right)$$

- 深層学習 (deep learning):

$L > 2$ となるような 深いネットワーク構造をもつモデル.

- 以下では $L = 2$ のときもその一種と考える.

- $L = 2$ のとき $w_{d,h_1}^{(2)} = 1$ としたモデル

は 一般化線形モデル (generalized linear model) と一致

- リンク関数は活性化関数の逆関数.

cf.) 一般化線形モデル (GLM)

線形モデル → $y = w^T x$ (パラメータについて線形)

y の範囲に制限がある ($y \in \{0,1\}$, $y \in \mathbb{Z}_{\geq 0}$ など) とか,

y の分散が平均に依存するような場合にはうまくいかない.

→ GLM: 線形モデル $\eta = w^T x$ と

リンク関数 g: $\mathbb{E}[y|x] = \mu = g^{-1}(\eta)$ となるもの

↑ 線形モデルによる予測値と, 分布の平均との関係を表現する.

2.2.2 勾配降下法とNewton-Raphson法.

2.2.2.1 勾配降下法

- NNでは，誤差最小となるパラメータを解析的に求められない.

→ 数値解法.

- 勾配降下法 (gradient descent method)

誤差関数 $E(w)$ $(w \in \mathbb{R}^D)$.

点 w_0 における最急増加方向： $\nabla_w E(w_0)$

pf. $d \in \mathbb{R}^D$ 方向の方向微分が最大になる d を考える. $(\|d\| = 1$ とする$)$

$$\frac{1}{\varepsilon}\left(E(w_0+\varepsilon d) - E(w_0)\right)$$

↓ Taylor展開

$$= \frac{1}{\varepsilon}\left(E(w_0) + \nabla_w E(w_0)\cdot(\varepsilon d) + O(\varepsilon^2) - E(w_0)\right)$$

$$= \nabla_w E(w_0)\cdot d + O(\varepsilon)$$

$$\xrightarrow{\varepsilon \to 0} \nabla_w E(w_0)\cdot d.$$

d が $\nabla_w E(w_0)$ と同じ方向のとき，方向微分は最大. ■

- w の適当な初期値を与えて，次式を繰り返す.

$$w \leftarrow w - \alpha \nabla_w E(w).$$

$\alpha > 0$ ： 学習率 (learning rate)

- α が大 → 学習は速いが収束が安定しない.

α が小 → 学習は遅いが収束は安定する.

2.2.2.2 Newton-Raphson法

- パラメータ数 M が多くないとき，誤差関数の2階微分を使って最適化する方法もよくある．

- **Newton-Raphson法**

最小化する誤差関数を $\bar{w}$ まわりのTaylor展開で2次近似：

$$E(w) \approx E(\bar{w}) + \nabla_w E(\bar{w})^T (w-\bar{w}) + \frac{1}{2}(w-\bar{w})^T \nabla_w^2 E(\bar{w})(w-\bar{w})$$

$$=: \tilde{E}(w).$$

E が C^2 級なら対称行列．

$$H_E(\bar{w}) := \nabla_w^2 E(\bar{w}) = \begin{pmatrix} \frac{\partial^2 E}{\partial w_1^2}(\bar{w}) & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_M}(\bar{w}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_M \partial w_1}(\bar{w}) & \cdots & \frac{\partial^2 E}{\partial w_M^2}(\bar{w}) \end{pmatrix}$$ Hesse行列

凸関数

$\tilde{E}(w)$ の最小解は解析的にとける．（停留点が最小解になる）

$$\nabla_w \tilde{E}(w) = \nabla_w E(\bar{w}) + H_E(\bar{w})(w-\bar{w}) = 0 \quad \text{とすると}$$

$$H_E(\bar{w})(w-\bar{w}) = -\nabla_w E(\bar{w}).$$

局所的最適解の十分近くなら $H_E(\bar{w})$ が正定値で $H_E(\bar{w})$ が正則になる．

$H_E(\bar{w})$ が正則なら，最小解は

$$w = \bar{w} - H_E(\bar{w})^{-1} \nabla_w E(\bar{w}).$$

Newton-Raphson法の更新式：

$$w \leftarrow w - H_E(w)^{-1} \nabla_w E(w).$$

→ 2次収束する：$\|w_{k+1} - w^*\| = O(\|w_k - w^*\|^2)$ （w^*：最適解）

- 一般には Newton-Raphson 法は大域的収束性がない.

(解の十分近くから反復をすると高速に収束する：局所的収束性)

→ 信頼領域法を利用することで大域的収束するようにする.

- $H_E(\bar{w})$ を適当な正定値行列で近似的に計算する準 Newton 法を利用することで、大域的収束したり、$H_E(\bar{w})$ の計算にかかる手間を省ける.

cf.) 非線形最適化（制約なし）とかで調べるといろいろでる.

2.2.3 誤差逆伝播法

- **誤差逆伝播法** (error back propagation method)

feedforward NN では誤差最小となるパラメータが解析的に求まらない.

→ 数値的に求める. パラメータについての勾配がほしい.

- L層のモデルを考える: $(d = 1, \cdots, D(=H_L))$

$$y_{n,d} = \sum_{h_{L-1}=1}^{H_{L-1}} w_{d,h_{L-1}}^{(L)} \phi\left(\sum_{h_{L-2}=1}^{H_{L-2}} w_{h_{L-1},h_{L-2}}^{(L-1)} \cdots \phi\left(\sum_{h_0=1}^{H_0} w_{h_1,h_0}^{(1)} x_{n,h_0} \right) \cdots \right) + \varepsilon_{n,d}$$

重みパラメータ全体 $W = (W^{(1)}, \cdots, W^{(L)})$.

$$z_{n,h_0}^{(0)} = x_{n,h_0} \quad (h_0 = 1, \cdots, H_0)$$

$$a_{n,h_l}^{(l)} = \sum_{h_{l-1}=1}^{H_{l-1}} w_{h_l,h_{l-1}}^{(l)} z_{n,h_{l-1}}^{(l-1)} \quad (l = 1, 2, \cdots, L,\ h_l = 1, \cdots, H_l)$$

$$z_{n,h_l}^{(l)} = \phi(a_{n,h_l}^{(l)}) \quad (l = 1, 2, \cdots, L-1,\ h_l = 1, \cdots, H_l)$$

学習データ数 N のときの誤差関数:

$$E(W) := \sum_{n=1}^{N} E_n(W), \quad E_n(W) := \frac{1}{2} \sum_{d=1}^{D} (y_{n,d} - a_{n,d}^{(L)})^2$$

(真の値 $y_{n,d}$, 予測値 $a_{n,d}^{(L)}$)

$\nabla_W E(W)$ を求めるには, $\nabla_W E_n(W)$ を求めればよい.

$\frac{\partial E_n}{\partial a_{n,i}^{(l)}} =: \delta_{n,i}^{(l)}$ $(l = 1, \cdots, L,\ i = 1, \cdots, H_l)$ と定める.

- 第 l 層の重み $w_{i,j}^{(l)}$ $(i = 1, \cdots, H_l,\ j = 1, \cdots, H_{l-1})$ について

偏微分を求める. $w_{i,j}^{(l)}$ が関わるのは $a_{n,i}^{(l)}$. Chain rule を使って,

$$\frac{\partial E_n}{\partial w_{i,j}^{(l)}} = \frac{\partial E_n}{\partial a_{n,i}^{(l)}} \frac{\partial a_{n,i}^{(l)}}{\partial w_{i,j}^{(l)}} = \delta_{n,i}^{(l)} z_{n,j}^{(l-1)}$$

$\delta_{n,i}^{(l)}$ が分かればよい.

$E_n(a_n^{(L)}(a_n^{(L-1)} \cdots (a_n^{(1)}) \cdots)) \Rightarrow \frac{\partial E_n}{\partial w_{i,j}^{(l)}} = \frac{\partial E_n}{\partial a_n^{(L)}} \frac{\partial a_n^{(L)}}{\partial a_n^{(L-1)}} \cdots \frac{\partial a_n^{(l+1)}}{\partial a_n^{(l)}} \frac{\partial a_n^{(l)}}{\partial w_{i,j}^{(l)}} = \frac{\partial E_n}{\partial a_n^{(l)}} \frac{\partial a_n^{(l)}}{\partial w_{i,j}^{(l)}}$.

- ところで，$\delta_{n,i}^{(L)}$ は計算できる．　$d=i$ のときだけ関係する

$$\delta_{n,i}^{(L)} = \frac{\partial E_n}{\partial a_{n,i}^{(L)}} = \frac{1}{2}\sum_{d=1}^{D} \frac{\partial}{\partial a_{n,i}^{(L)}} \left(y_{n,d} - a_{n,d}^{(L)}\right)^2$$

$$= \frac{1}{2}\cdot 2\left(y_{n,i} - a_{n,i}^{(L)}\right)(-1)$$

$$= a_{n,i}^{(L)} - y_{n,i} .$$　← 真の値と NN の出力との誤差

- $l(\geq 2)$ について $\delta_{n,i}^{(l)}$ $(i=1,\cdots,H_l)$ が求まっているとする．このとき，

$\delta_{n,i}^{(l-1)}$ $(i=1,\cdots,H_{l-1})$ を $\delta_{n,i}^{(l)}$ $(i=1,\cdots,H_l)$ から求めよう．

$a_{n,j}^{(l)} = \sum_{h=1}^{H_{l-1}} w_{j,h}^{(l)} \phi\left(a_{n,h}^{(l-1)}\right)$ なので，$a_{n,i}^{(l-1)}$ が変わると

$a_{n,j}^{(l)}$ $(j=1,\cdots,H_l)$ が全て変わる．Chain rule より

$$\delta_{n,i}^{(l-1)} = \frac{\partial E_n}{\partial a_{n,i}^{(l-1)}} = \sum_{j=1}^{H_l} \frac{\partial E_n}{\partial a_{n,j}^{(l)}} \frac{\partial a_{n,j}^{(l)}}{\partial a_{n,i}^{(l-1)}}$$

$a_{n,i}^{(l-1)}$ が関わるのは $h=i$ のところだけ

$$= \sum_{j=1}^{H_l} \delta_{n,j}^{(l)} \frac{\partial}{\partial a_{n,i}^{(l-1)}} \left(\sum_{h=1}^{H_{l-1}} w_{j,h}^{(l)} \phi\left(a_{n,h}^{(l-1)}\right)\right)$$

$$= \sum_{j=1}^{H_l} \delta_{n,j}^{(l)} w_{j,i}^{(l)} \phi'\left(a_{n,i}^{(l-1)}\right)$$

$$= \phi'\left(a_{n,i}^{(l-1)}\right) \sum_{j=1}^{H_l} \delta_{n,j}^{(l)} w_{j,i}^{(l)}$$

- $\delta_{n,i}^{(l)}$ は，全ての $a_{n,i}^{(l)}$ が求まっていれば

$$\delta_{n,i}^{(L)} \to \delta_{n,i}^{(L-1)} \to \cdots \to \delta_{n,i}^{(2)} \to \delta_{n,i}^{(1)}$$

の順に求まる！（逆伝播）

- $\frac{\partial E_n}{\partial w_{i,j}^{(l)}}$ は全ての $\delta_{n,i}^{(l)}$ と $z_{n,j}^{(l)}$ が求まっていれば計算できる！

Algo. (誤差逆伝播法)

1. 順伝播：$z_{n,i}^{(0)} \to a_{n,i}^{(1)} \to z_{n,i}^{(1)} \to \cdots \to z_{n,i}^{(L-1)} \to a_{n,i}^{(L)}$ の順に

   全ての隠れユニットの値 $\mathcal{Z} = \{z_{n,i}^{(l)}\}$ と活性 $\mathcal{A} = \{a_{n,i}^{(l)}\}$

   を計算する.

2. 逆伝播：現在のパラメータ W と活性 $\mathcal{A}$ を使って

   $$\delta_{n,i}^{(L)} \to \delta_{n,i}^{(L-1)} \to \cdots \to \delta_{n,i}^{(2)} \to \delta_{n,i}^{(1)}$$

   の順にデルタたち $\Delta = \{\delta_{n,i}^{(l)}\}$ を計算.

3. 勾配計算：デルタたち Δ と隠れユニットの値 $\mathcal{Z}$ を使って，E_n の

   パラメータ W による勾配 $\nabla_W E_n(W)$ を計算する.

4. パラメータ更新：勾配降下法によりパラメータを

   $$W \leftarrow W - \alpha \nabla_W E(W)$$

   と更新. □

- 実装上は自動微分 (automatic differentiation) を利用できる.

cf. Chain rule. (以下は雑な主張になっている. 解析学の本を読んでね)

$f(y_1, \cdots, y_n)$, $y_i = g_i(x_1, \cdots, x_m)$ のとき

$$\frac{\partial f}{\partial x_j} = \sum_{i=1}^{n} \frac{\partial f}{\partial y_i} \frac{\partial y_i}{\partial x_j} \qquad (j = 1, \cdots, m).$$

- ふつうの feedforward NN はパラメータ数が多い.

→ $E(W)$ を最小化すると過剰適合する可能性あり.

Ridge回帰と同様, $E(W)$ のかわりに

$$J(W) = E(W) + \frac{\lambda}{2}\Omega_{L2}(W)$$

を最小化することで防ぐ. ($\Omega_{L2}(W) = \sum_{w \in W} w^2$)

このときの重みの更新式は重み減衰 (weight decay) の式と一致:

$$J(W) = \nabla_W E(W) + \frac{\lambda}{2}\nabla_W \Omega_{L2}(W).$$

ここで $\nabla_W \Omega_{L2}(W) = 2W$ なので,

$$W \leftarrow W - \alpha\left(\lambda W + \nabla_W E(W)\right) = (1-\alpha\lambda)W - \alpha\nabla_W E(W)$$

と更新する.

2.2.4 Hesse行列を利用した学習

- 勾配降下法のかわりに Newton-Raphson 法も利用できる.

→ Hesse行列が要るから, パラメータ数 M に対して

$O(M^2)$ の計算時間になる.

簡略化のため, 各入力 x_n に対する出力 $a_n^{(L)}$ の勾配を使って

$$H_E(W) \simeq \sum_{n=1}^{N} (\nabla_W a_n^{(L)})(\nabla_W a_n^{(L)})^T$$

と近似することもある.

2.2.5 分類モデルの学習

- 回帰問題ではなくて，分類問題を解くには？
- 2値分類．シグモイド関数 Sig を使って，

x_n に対する出力 $a_n^{(L)} \in \mathbb{R}$ を

$$\mu_n = \mathrm{Sig}(a_n^{(L)}) \in (0,1) \leftarrow y_n = 1 \text{となる確率を表す．}$$

と変換．$y_n \in \{0,1\}$ に対する誤差を交差エントロピー誤差関数

$$E(W) = -\sum_{n=1}^{N} (y_n \log \mu_n + (1-y_n)\log(1-\mu_n))$$

で評価．

- Dクラス分類．$y_n \in \{0,1\}^D$, $\sum_{d=1}^{D} y_{n,d} = 1$ (one-hot vector)

x_n に対する出力 $a_n^{(L)} \in \mathbb{R}^D$ をソフトマックス関数

$$\pi_d(a_n^{(L)}) := \frac{\exp(a_{n,d}^{(L)})}{\sum_{d'=1}^{D} \exp(a_{n,d'}^{(L)})} \leftarrow y_{n,d} = 1 \text{となる確率を表す．}$$

で $\sum_{d=1}^{D} \pi_d(a_n^{(L)}) = 1$ となるD次元ベクトル

$$\pi(a_n^{(L)}) = \begin{pmatrix} \pi_1(a_n^{(L)}) \\ \vdots \\ \pi_D(a_n^{(L)}) \end{pmatrix}$$

と変換．誤差関数を多値版の交差エントロピー誤差

$$E(W) = -\sum_{n=1}^{N} \sum_{d=1}^{D} y_{n,d} \log \pi_d(a_n^{(L)})$$

- 上記の誤差関数を誤差逆伝播させて最小化することで学習する．

・Dクラス分類での back propagation の例.

$d=d'$ のとき

$$\frac{\partial \pi_d(a_n^{(L)})}{\partial a_{n,d'}^{(L)}} = \frac{\partial}{\partial a_{n,d}^{(L)}}\left(\frac{\exp(a_{n,d}^{(L)})}{\sum_{d'=1}^{D}\exp(a_{n,d'}^{(L)})}\right)$$

$$= \frac{1}{\left(\sum_{d'=1}^{D}\exp(a_{n,d'}^{(L)})\right)^2}\left(\exp(a_{n,d}^{(L)})\sum_{d'=1}^{D}\exp(a_{n,d'}^{(L)}) - \exp(a_{n,d}^{(L)})^2\right)$$

$$= \pi_d(a_n^{(L)}) - \pi_d(a_n^{(L)})^2$$

$$= \pi_d(a_n^{(L)})(1-\pi_d(a_n^{(L)}))$$

$d\neq d'$ のとき

$$\frac{\partial \pi_d(a_n^{(L)})}{\partial a_{n,d'}^{(L)}} = \exp(a_{n,d}^{(L)})\frac{\partial}{\partial a_{n,d'}^{(L)}}\left(\left(\sum_{d''=1}^{D}\exp(a_{n,d''}^{(L)})\right)^{-1}\right)$$

$$= \exp(a_{n,d}^{(L)})\cdot(-1)\left(\sum_{d''=1}^{D}\exp(a_{n,d''}^{(L)})\right)^{-2}\exp(a_{n,d'}^{(L)})$$

$$= -\pi_d(a_n^{(L)})\pi_{d'}(a_n^{(L)})$$

よって, ($n'=n$ のところだけ残る.)

$$\delta_{n,d}^{(L)} = \frac{\partial E(w)}{\partial a_{n,d}^{(L)}} = \frac{\partial}{\partial a_{n,d}^{(L)}}\left(-\sum_{n'=1}^{N}\sum_{d'=1}^{D} y_{n',d'}\log \pi_{d'}(a_{n'}^{(L)})\right)$$

$$= -\sum_{d'=1}^{D} y_{n,d'}\frac{\partial}{\partial a_{n,d}^{(L)}}\log \pi_{d'}(a_n^{(L)})$$

$$= -\sum_{d'\neq d} y_{n,d'}(\pi_{d'}(a_n^{(L)}))^{-1}(-\pi_{d'}(a_n^{(L)})\pi_d(a_n^{(L)}))$$

$$- y_{n,d}(\pi_d(a_n^{(L)}))^{-1}(\pi_d(a_n^{(L)})(1-\pi_d(a_n^{(L)})))$$

$$= \pi_d(a_n^{(L)})\sum_{d'\neq d} y_{n,d'} - y_{n,d}(1-\pi_d(a_n^{(L)}))$$

$$= \pi_d\left(a_n^{(L)}\right) \underbrace{\sum_{d'=1}^{D} y_{n,d'}}_{=1} - y_{n,d}$$

$$= \pi_d\left(a_n^{(L)}\right) - y_{n,d}.$$ ← 分類結果に対する真値との差分.

§2.3 効率的な学習法

- 勾配降下法での feedforward NN の学習.

→ 大規模データに対する処理速度に問題あり.

パラメータが多いとき過剰適合してしまう.

なんとかしたい...

2.3.1 確率的勾配降下法.

- データセット $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ 中の N 個のデータを全て使って勾配を計算する方法 → バッチ学習 (batch learning).

1回のパラメータ更新のたびに学習データ全てを使う

→ 大量のデータを使う際に計算効率が悪い.

- $\mathcal{D}$ を一気に処理せず, M ($< N$) 個の小規模な部分集合 $\mathcal{D}_\delta = \{(x_n, y_n)\}_{n \in \delta}$ (ミニバッチ) を取り出して使う.

・δ : ランダムに M 個選んだ添字集合.

誤差関数 $E_\delta(W) = N \cdot \frac{1}{M} \sum_{n \in \delta} E_n(W)$ を設定し,

これについて back propagation することで W を更新する.

(確率的勾配降下法 : Stochastic Gradient Descent, SGD

ミニバッチの選び方がランダム.

・ $E_S(W)$ は、S の選び方を一様にすれば ($q_D(S)$：一様分布) 期待値的に $E(W)$ と一致する：

「$E(W)$ を minimize するのと $E_S(W)$ を minimize するのは大体同じ」ということ。

$$\mathbb{E}_{q_D(S)}[E_S(W)] = E(W).$$

pf. S を選ぶ確率は、$q_D(S) = \dfrac{1}{\binom{N}{M}} = \dfrac{M!(N-M)!}{N!}$

$$\begin{aligned}\mathbb{E}_{q_D(S)}[E_S(W)] &= \sum_S E_S(W)\cdot q_D(S) \\ &= \frac{(M-1)!(N-M)!}{(N-1)!}\sum_S\sum_{n\in S} E_n(W) \\ &= \binom{N-1}{M-1}^{-1}\sum_S\sum_{n\in S} E_n(W).\end{aligned}$$

ここで、各 $E_n(W)$ 的に $\sum_S\sum_{n\in S} E_n(W)$ の中で何回加えられるかを考える。

n を含むような S の選び方は $\binom{N-1}{M-1}$ 通りあるので、

どの $E_n(W)$ も $\binom{N-1}{M-1}$ 回加えられる。

$$\therefore \mathbb{E}_{q_D(S)}[E_S(W)] = \binom{N-1}{M-1}^{-1}\binom{N-1}{M-1}\sum_{n=1}^{N} E_n(W)$$

$$= E(W). \qquad \blacksquare$$

・ 学習率を α_i とし、

$$\sum_{i=1}^{\infty}\alpha_i = \infty, \quad \sum_{i=1}^{\infty}\alpha_i^2 < \infty \qquad -(*)$$

となるように $\alpha_1, \alpha_2, \cdots$ のとる値をスケジューリングすると、

ミニバッチによる更新的に確率1で $E(W)$ の停留点に収束する。

(証明は知らない…)

(*) をみたすスケジューリングとして $\alpha_i = \frac{\alpha}{i}$ $(\alpha > 0)$ がある.

cf. $\sum_{i=1}^{\infty} \frac{1}{i}$ は発散する.

pf. $S_n := \sum_{i=1}^{n} \frac{1}{i}$ とする. $\frac{1}{i} > 0$ なので, $\sum_{i=1}^{\infty} \frac{1}{i}$ は収束するか $+\infty$ に発散する. (振動はしない)

$$S_{2n} - S_n = \sum_{i=n+1}^{2n} \frac{1}{i} \geq \sum_{i=n+1}^{2n} \frac{1}{2n} = n \cdot \frac{1}{2n} = \frac{1}{2}$$

なので, $\{S_n\}$ は Cauchy 列でなく, 収束しない. $\therefore \sum_{i=1}^{\infty} \frac{1}{i}$ は発散. ■

$\sum_{i=1}^{\infty} \frac{1}{i^2}$ は収束する.

pf. $$\sum_{i=1}^{n} \frac{1}{i^2} < 1 + \int_1^n \frac{1}{x^2} dx = 1 + \left[-\frac{1}{x} \right]_1^n = 2 - \frac{1}{n}$$

$\therefore n \to \infty$ として $\sum_{i=1}^{\infty} \frac{1}{i^2} < 2$. ■

- Robbins-Monro アルゴリズム:

学習率スケジューリング $\alpha_i = \frac{\alpha}{i}$ $(\alpha > 0)$ により確率的勾配降下法を行う.

- モメンタム法 (momentum method): SGD の最適化の効率化.

$\beta \in [0, 1)$ (過去の勾配の影響をどれだけ受けるかを指定する.) として次式で反復.

$$p \leftarrow \beta p - \alpha \nabla_w E(w);$$

$$w \leftarrow w + p$$

経験的に良くなる.

2.3.2 ドロップアウト.

- **確率的正則化** (Stochastic regularization):

モデル学習時に小量のノイズをデータや隠れユニットなどに加えて

過剰適合を抑制し，汎化性能を上げる.

- **ドロップアウト** (dropout): その一例. SGDをするときによく用いられる.

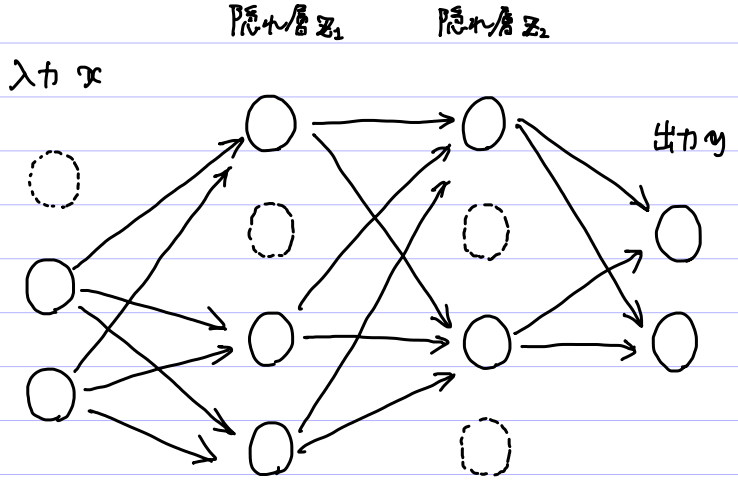
・dropoutを用いたfeedforward NNの学習.

ミニバッチ $\mathcal{D}_s = \{(x_n, y_n)\}_{n \in s}$: given.

各データ点 (x_n, y_n) に対する勾配計算時に，各ユニットを

ある独立な確率 $r \in (0,1)$ で無効にする.

→ (x_n, y_n) が与えられるごとにサブネットが構成される:



各 $(x_n, y_n) \in \mathcal{D}_s$ に対する勾配を計算した後は，それらの平均をとって

パラメータ更新のための勾配を得る.

- 予測

入力 x_* に対する予測では，ユニットを欠落させていない元々のネットワークを利用．

- 実験的には，各ユニットの出力を $1/2$ 倍にスケーリングしたものを代わりに用いると，予測性能が良くなる．

- dropout で過剰適合を防げる理由．

- サブネットを組み合わせることによる アンサンブル効果：

ユニット数を U とすると，2^U 個の異なるサブネットでアンサンブルしているようなもの．

- 遺伝的な交配のプロセスを模倣している．(?)

- その他の確率的正則化．

- ユニットを無効化するかわりに，ユニットの出力にノイズ $m \sim N(1,1)$ をかける

- 予測時のスケーリングは不要．

- DropConnect：dropout の特別な場合．

各スカラーの重み w と隠れユニット z の間の接続をランダムに欠落．

2.3.3 バッチ正規化.

確率的正則化の一つ.

- バッチ正規化 (batch normalization):

学習時に加法的なノイズと乗法的なノイズを隠れユニットに与え,

学習時の最適化の効率化とノイズの付加による正則化の効果を同時に実現.

- 多層NNの学習の難しさの要因の一つとして,

一つ前の層のパラメータの変化により,次の層への入力の分布が大幅に変化

することがある. (内部共変量シフト: internal covariate shift)

→ 学習率αを小さくすることで一度の更新量を小さくすると学習効率低下….

- batch normalization:

$\mathcal{D}_{\mathcal{S}} = \{(x_n, y_n)\}_{n\in\mathcal{S}}$: given のとき, 隠れユニットの活性$\{a_n\}_{n\in\mathcal{S}}$の各値を

M個のデータからなる

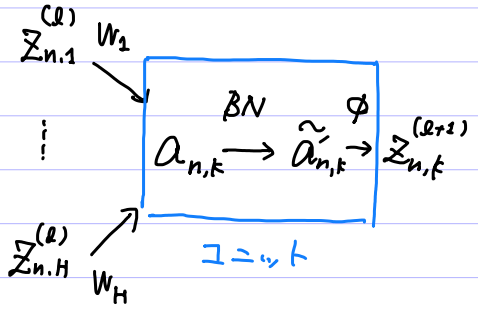
平均0, 分散1になるよう修正する:

$$\mu_{\mathcal{S}} \leftarrow \frac{1}{M}\sum_{n\in\mathcal{S}} a_n;$$

$$\sigma_{\mathcal{S}}^2 \leftarrow \frac{1}{M}\sum_{n\in\mathcal{S}} (a_n - \mu_{\mathcal{S}})^2;$$

$$\tilde{a}_n \leftarrow \frac{a_n - \mu_{\mathcal{S}}}{\sqrt{\sigma_{\mathcal{S}}^2 + c}} \quad (c>0)$$

数値的に安定にするための定数



パラメータ $\gamma, \beta \in \mathbb{R}$ によりユニットの値を

$$\tilde{a}'_n \leftarrow \gamma\tilde{a}_n + \beta$$ ← 表現力をもたせるため.

とする.

Batch normalization は非線形変換を行う前にやる: $z = \phi(BN(a))$

- Batch normalization を用いた NN で予測を行うときは，学習データ全体の
  を使って正規化する．
- Batch normalization により，各隠れユニットの入力の傾向が安定化．
  → モデルのパラメータの調整や初期化などが簡単になる
  → 学習率を上げて学習速度を上げることができる．
- 理論的な根拠はない…．

cf. Batch normalization と dropout を併用しない方が良いらしい：
[Li+2018] Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift.

§2.4 ニューラルネットワークの拡張モデル.

- feedforward NN以外にもいろいろある.

2.4.1 畳み込みNN.

- 畳み込みNN (LeCun+, 1989) : Convolutional NN

通常の重みパラメータによる行列積の代わりに畳み込み (convolution) を行う.

(ふつうの"畳み込み"とは異なる意味合いらしい.)

- 主に時系列データや画像認識のタスクで高い性能を発揮.

- 2次元データに対する畳み込み計算の例.

$X \in M_{m,n}(\mathbb{R})$: 2次元データ

$W \in M_{k,l}(\mathbb{R})$: フィルタ (filter) $(k \leq m, l \leq n)$

畳み込み後のデータ $S \in M_{m-k+1,\, n-l+1}(\mathbb{R})$ については

$$S_{ij} = (W * X)_{ij} := \sum_{a=1}^{k} \sum_{b=1}^{l} W_{a,b} X_{i+a-1,\, j+b-1}$$

と計算. S を特徴マップ (feature map) と呼ぶ.

- 疎結合 (sparse connected) なネットワークになっている.

cf.) feedforward NNは全結合 (fully connected)

- 出力 S_{ij} が入力 X の局所的な領域にのみ依存するよう制限されている.

→ パラメータ数が少ない.

- 共通のフィルタによる変換なので, X の特徴的な箇所の移動に対し, 不変な特徴抽出.

・フィルタと入力Xを適切に変換すれば，通常の行列積で表現可能．

→ backpropagationで Wを学習できる．

（計算の効率上，そうしたことはやらない）

・特徴マップは，活性化関数により変換．

・プーリング関数（pooling func.）：非線形変換．固定して使う．

ex）最大プーリング（max pooling）

・プーリングされる値は，入力のわずかな変化に対しても変化しにくい．

2.4.2 再帰型NN

・feedforward NN，CNN

・各データ点は独立という仮定をおいている．

→ 音声，動画，文字列など，各データ点が時系列的な相関をもつときは，

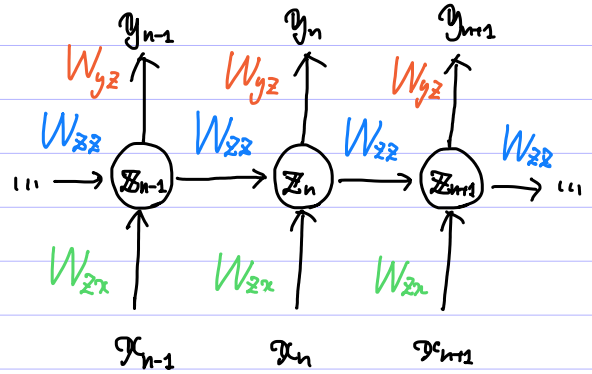
独立性を仮定できない．

・入力データやラベルの次元数は通常固定されている．

→ 長さが変わる文字列などのデータは扱いづらい．

・再帰型NN：Recurrent NN．

・系列データを扱うためのNN．



・連続値入力データ：$x_1, \cdots, x_N$

カテゴリラベルデータ：$y_1, \cdots, y_N$　（D カテゴリ）

時刻 n の隠れユニット：z_n は次のように計算される．

$$z_n \leftarrow \phi\left(W_{zx} x_n + W_{zz} z_{n-1} + b_z\right)$$

時刻 n の隠れユニットからの出力：

$$\pi_n \leftarrow \pi\left(W_{yz} z_n + b_y\right) \qquad (\pi : \text{softmax})$$

・誤差関数は各時刻のネットワークの出力とラベルの間の誤差により定義される．

パラメータ集合 $\textcircled{H} := \{W_{zx}, W_{zz}, b_z, W_{yz}, b_y\}$ とする．

時刻 n における誤差は，クロスエントロピー誤差により

$$E_n(\textcircled{H}) = -\sum_{d=1}^{D} y_{n,d} \log \pi_{n,d}.$$

時系列全体の誤差は，

$$E(\textcircled{H}) = \sum_{n=1}^{N} E_n(\textcircled{H})$$

E の $\textcircled{H}$ に関する最小化は backpropagation でできる．（*exercise*）

・いろいろな RNN の構造がある．

2.4.3 自己符号化器

- 自己符号化器（autoencoder）：

教師なし学習．入力データを低次元の空間に圧縮する次元削減に利用される．

- データ $\mathcal{X} = \{x_1, \cdots, x_N\} \subseteq \mathbb{R}^D$ を

$\mathcal{Z} = \{z_1, \cdots, z_N\} \subseteq \mathbb{R}^d$ $(d < D)$ に変換する．

各 $z_n \in \mathcal{Z}$ を 符号（code）とか 潜在変数（latent variable）という．

- 二つの NN f, g を使う．

f：符号化器（encoder）． $z_n = f(x_n)$

g：復号化器（decoder）． $\tilde{x}_n = g(z_n)$ ， $\tilde{x}_n \approx x_n$

- 損失関数：$\mathcal{L}(x_n, \tilde{x}_n) \to \min$ として学習させる．

↳ x_n がもつ本質的な情報を z_n として抽出できるようにする．

- 潜在変数 z_n は，データ圧縮や特徴量として利用される．

- f, g には通常は feedforward NN が利用される．

- f, g の表現力が高すぎたり，$d > D$ だと $g \circ f = \mathrm{id}$

となってしまう．z_n が，x_n のノイズも含めて情報をもっており，意味なし．

（過剰適合）

→ z_n が獲得する表現を抑える必要がある．

正則化：$J_n = \mathcal{L}(x_n, \tilde{x}_n) + \lambda \Omega(z_n)$

→ $g \circ f = id$ となるのを防ぐ

- $f, g, \mathcal{L}, \Omega$ の設計が明確でない．

これらの値を過剰適合しないように調整するのも難しい．

→ 変分自己符号化器（variational autoencoder, VAE）．