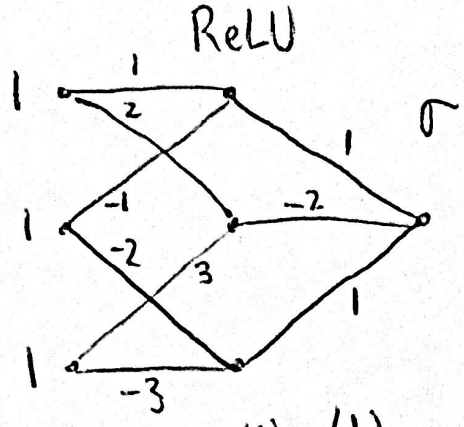
E.g.

$\mathcal{N} =$



$$b^{(0)} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad b^{(1)} = 0$$

$$\mathcal{N}\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \sigma F^{(1)} \text{ReLU } F^{(0)} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \qquad F^{(0)}\vec{x} = \begin{pmatrix} 1 & -1 & 0 \\ 2 & 0 & 3 \\ 0 & -2 & -3 \end{pmatrix}\vec{x} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$= \sigma F^{(1)} \text{ReLU} \begin{pmatrix} 1 \\ 6 \\ -4 \end{pmatrix} \qquad F^{(1)}\vec{x} = (1 \ -2 \ 1)\,\vec{x}$$

$$= \sigma F^{(1)} \begin{pmatrix} 1 \\ 6 \\ 0 \end{pmatrix}$$

$$= \sigma(-10)$$

$$= \frac{1}{1+e^{10}}$$

$$\cong .0000\vec{0}1 \quad \text{(small)}$$

E.g. Suppose $\mathcal{N}$ has depth D and $G^{(1)}_{\mathbb{R}^1 \to \mathbb{R}^1} \cdots G^{(D)}_{\mathbb{R}^1 \to \mathbb{R}^1}$ are all linear activations ($G^{(i)}_{\mathbb{R}^1}(\vec{x}) = \vec{x} \ \forall i$)

Then $\mathcal{N}\vec{x} = F^{(D-1)} F^{(D-2)} \cdots F^{(1)} F^{(0)} \vec{x}$

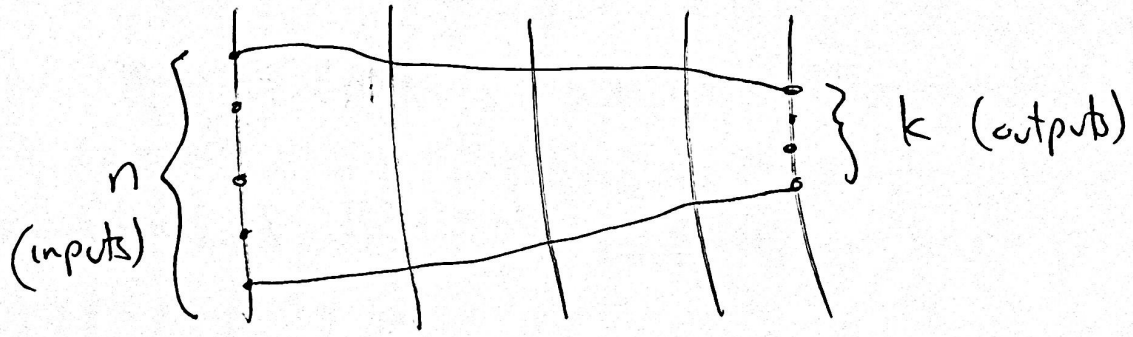
Remark The composition of affine maps is an affine map, so $\exists\ W, \vec{b}$ s.t.

$$\mathcal{N}\vec{x} = W\vec{x} + \vec{b}$$

(so this NN can only fit affine functions
i.e. this is essentially linear regression)

Neural Nets as Learning Algorithms

Suppose there is a function $f^*: \mathbb{R}^n \to \mathbb{R}^k$ (often $k=1$) and we wish to "approximate" or "model" f^*. Let $\mathcal{N}$ be a neural network $\mathcal{N} = (\mathcal{H}, F, G)$ with depth D & $\mathcal{H} = (P, \leq, \varphi, \leq')$ s.t. $\#P^{(0)} = n$ and $\#P^{(D)} = k$



$F^{(i)}(\vec{x}) = W^{(i)}\vec{x} + \vec{b}^{(i)}$ where nonzero entries of $W^{(i)}, \vec{b}^{(i)}$ are treated as parameters

Let $W = (W^{(0)}, \ldots, W^{(D-1)})$

$B = (\vec{b}^{(0)}, \ldots, \vec{b}^{(D-1)})$

& define $f_{model}(\vec{x}; W, B) = N\vec{x}$

(feedforward of $\vec{x}$)

How to find appropriate values of W, B?

Given a dataset $X \in \mathbb{R}^{m \times n}$, $Y \in \mathbb{R}^{m \times k}$ we define a cost function

$$J(W,B) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{L\left(f_{model}(\vec{x}^{(i)}; W, B), \vec{y}^{(i)}\right)}_{\text{per-example error function or "loss function"}}$$

& try to find values for W, B which minimize this using some optimization algorithm

Note: We will usually need to add a regularization term (soon)

Note: A common choice for L is

$$L(f_{model}(\vec{x}; W, B), \vec{y}) = \| f_{model}(\vec{x}; W, B) - \vec{y} \|_2^2$$

As a maximum likelihood estimator:

Lets assume X, Y are sampled from a distribution $p_{data}(\vec{x}, \vec{y})$ under I.I.D. assumptions.

Let $p_{model}(\vec{y} \mid \vec{x}; \theta)$ be a model for $p_{data}(\vec{y} \mid \vec{x})$, parameterized by $\theta \in \mathbb{R}^N$ some N.

Then $$\theta_{ML} = \operatorname{argmin}_\theta \left(-\sum_{i=1}^{m} \log\left(p_{model}(\vec{y}^{(i)} \mid \vec{x}^{(i)}; \theta) \right) \right)$$

E.g. Suppose we choose p_{model} to be Gaussian:

$$p_{model}(\vec{y} \mid \vec{x}; W, B) = N(\vec{y}; N\vec{x}, \Sigma)$$

(N: Normal distribution; $N\vec{x}$: Neural network)

$$= \frac{1}{\sqrt{2\pi^n \det \Sigma}} \exp\left(-\frac{1}{2} (\vec{y} - N\vec{x}) \Sigma^{-1} (\vec{y} - N\vec{x}) \right)$$

where $\mathcal{N} = (H, F, G)$ $\quad F^{(i)}_{\vec{x}} = W^{(i)}\vec{x} + \vec{b}^{(i)}$

$$W = \left(W^{(0)}, \ldots, W^{(D-1)}\right)$$

$$B = \left(\vec{b}^{(0)}, \ldots, \vec{b}^{(D-1)}\right)$$

Then finding $\theta_{ML} = (W, B)_{ML}$ is equivalent to minimizing

$$J(W,B) = -\sum_{i=1}^{m} \log\left(\frac{1}{\sqrt{2\pi^n \det\Sigma}} \exp\left(-\frac{1}{2}\left(\vec{y}^{(i)} - \mathcal{N}\vec{x}^{(i)}\right)\Sigma^{-1}\left(\vec{y}^{(i)} - \mathcal{N}\vec{x}^{(i)}\right)\right)\right)$$

$$= \frac{1}{m}\sum - \log\left(\frac{1}{\text{const}}\right) + \frac{1}{2}\left(\vec{y}^{(i)} - \mathcal{N}\vec{x}^{(i)}\right)\Sigma^{-1}\left(\vec{y}^{(i)} - \mathcal{N}\vec{x}^{(i)}\right)$$

↖ Does not affect argmin (throw it away)

$$J(W,B) = \frac{1}{m}\sum_{i=1}^{m} \frac{1}{2}\left(\vec{y}^{(i)} - \mathcal{N}\vec{x}^{(i)}\right)\Sigma^{-1}\left(\vec{y}^{(i)} - \mathcal{N}\vec{x}^{(i)}\right)$$

(Note: Can usually take $\Sigma = id$ by first normalizing the dataset if needed)

in which case

$$J(W,B) = \frac{1}{m}\sum_{i=1}^{m} \| \vec{y}^{(i)} - \mathcal{N}\vec{x}^{(i)} \|_2^2$$

(Naive)

so we recover our cost function from before.

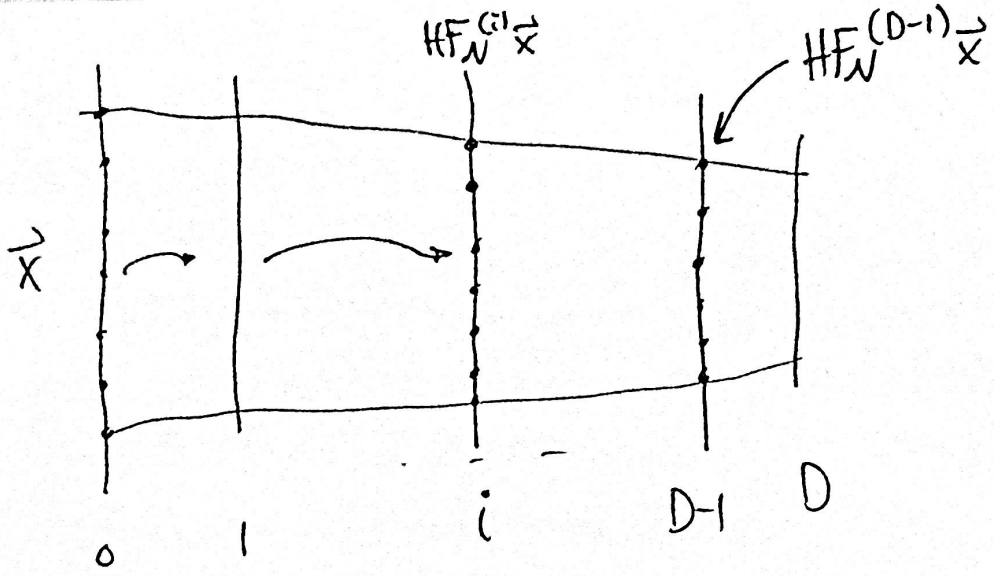
(Note: We see we can either specify a cost or specify a model for p_{data} and can usually go back & forth between these)

Output Neurons

Def: Given a neural network $\mathcal{N} = (\mathcal{H}, F, G)$ with depth D, for $\vec{x} \in \mathbb{R}^{\#P_0} \cong V_P^{(0)}$ (an input vector), the hidden features associated to $\vec{x}$ at layer i are

$$HF_{\mathcal{N}}^{(i)}\vec{x} = G_i F_{i-1} \cdots F_1 G_1 F_0 \vec{x}$$

(also set $HF_{\mathcal{N}}^{(0)}\vec{x} = \vec{x}$)



So the output neurons are responsible for predicting the target from $HF_N^{(D-1)}\vec{x}$. The choice of activation at the output layer depends on the type of targets possible

(i.e. on $p_{data}(\vec{y})$)

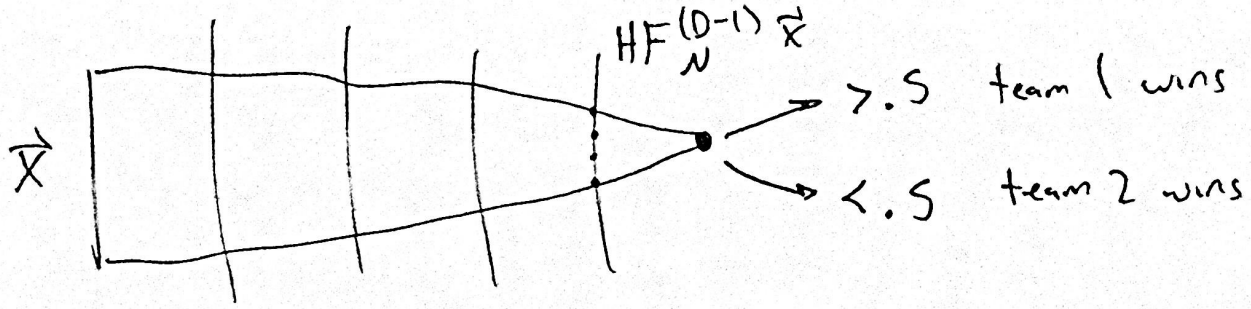
E.g. Input $\vec{x}$ = (some statistics on two sports teams)

$$y = \begin{cases} 0 & \text{if team 1 wins} \\ 1 & \text{if team 2 wins} \end{cases}$$

Dataset $X, \vec{y}$ of previous results. Goal: predict winners of upcoming games.

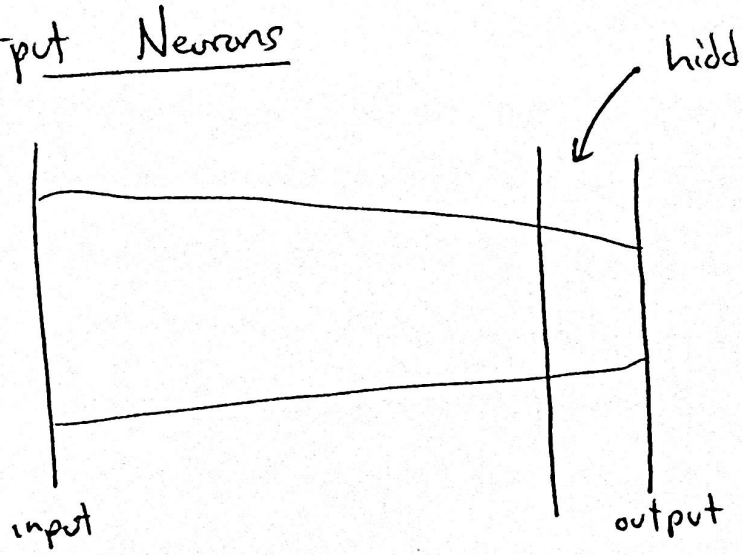
Idea: Apply a neural network $\mathcal{N}$ so that

$$\mathcal{N}\vec{x} = p(y=1 \mid \vec{x})$$



Sigmoid activation makes sense here.

Output Neurons

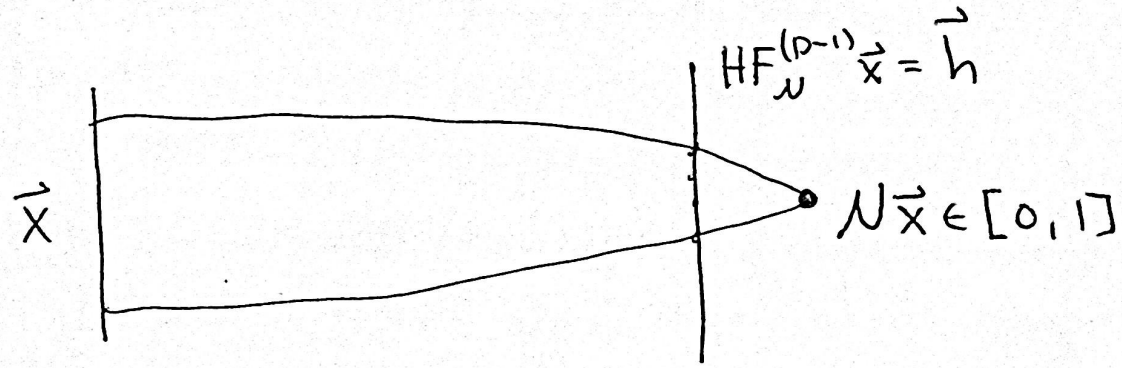


$$\vec{x}^{(i)} = HF_{\mathcal{N}}^{i}(\vec{x}) = G^{(i)}F^{(i-1)} \cdots G^{(1)}F^{(0)}\vec{x}$$

Eg. Input $\vec{x}$ statistics on sports teams 1 & 2

$$y = \begin{cases} 1 & \text{team 1 wins} \\ 0 & \text{teams 2 wins} \end{cases}$$

Idea: want $\mathcal{N}\vec{x} = p(y=1 \mid \vec{x})$

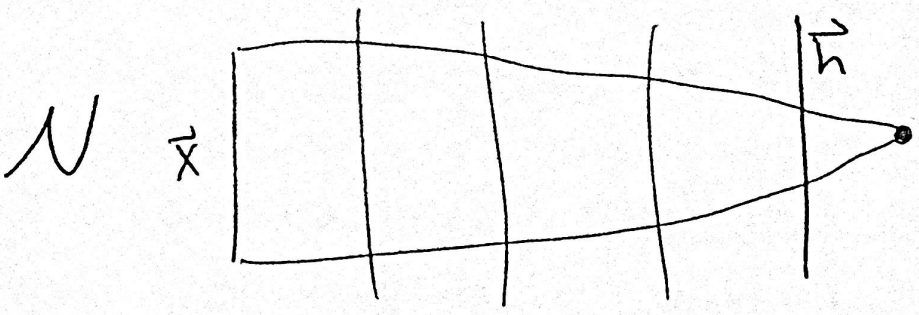


Makes sense to use sigmoid activation for $G^{(D)}$

Note: This is equivalent to transforming $\vec{x}$ to new features $\vec{h}$ and performing logistic regression on $\vec{h}$.

E.g. $\vec{x}$ statistics about a company
y stock price in 1 week

(y has continuous distribution e.g. Gaussian)



linear activation $G^{(D)}(y) = y$ makes sense here.

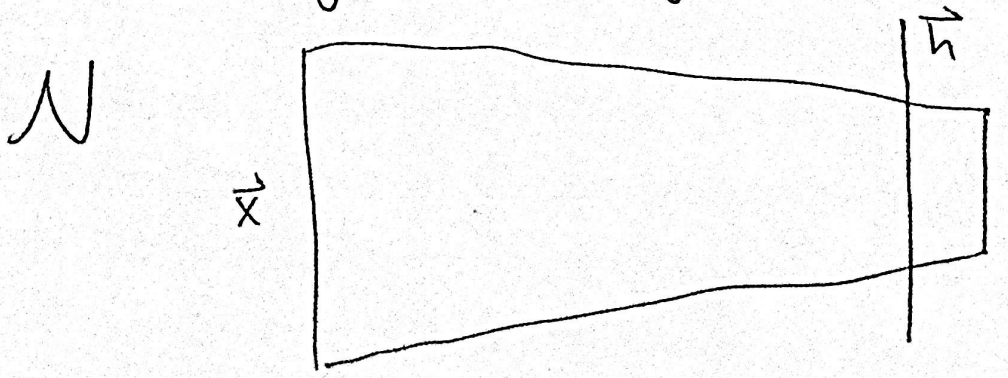
Note: this is equivalent to performing linear regression on $HF_{\mathcal{N}}^{(D-1)}(\vec{x})$

E.g. $\vec{x}$ = vectorized image of an animals (k types)

$$y = \begin{cases} e_1 & \text{if cat} \\ e_2 & \text{if dog} \\ \vdots & \\ e_k & \text{if llama} \end{cases} \in \mathbb{R}^k = \langle e_1, \dots, e_k \rangle$$

(called a one-hot encoding)

want $\mathcal{N}\vec{x} = \hat{y}$ where $\hat{y}_i = p(y = e_i \mid \vec{x})$ ($\therefore \sum_{i=1}^{k} \hat{y}_i = 1$)



Define $G^{(l)} = \text{softmax}$ where $\text{softmax}(\vec{h}) = \dfrac{\exp(h_i)}{\sum_{j=1}^{k} \exp(h_j)}$

Softmax makes sense here.

Remarks In each case we are choosing an activation function which will give the gradient nice properties. We want the gradient to have the property that when W, B are far from a local min, the gradient stays sufficiently large, (so GD does not "get stuck").

Optimization for Neural Networks

Let $\mathcal{N}$ be a neural network and $J(W,B)$ a cost function to measure "proximity" between $\mathcal{N}\vec{x}$ and target $\vec{y}$ on a dataset X, Y.
To run gradient descent to find good values for W, B we need to compute

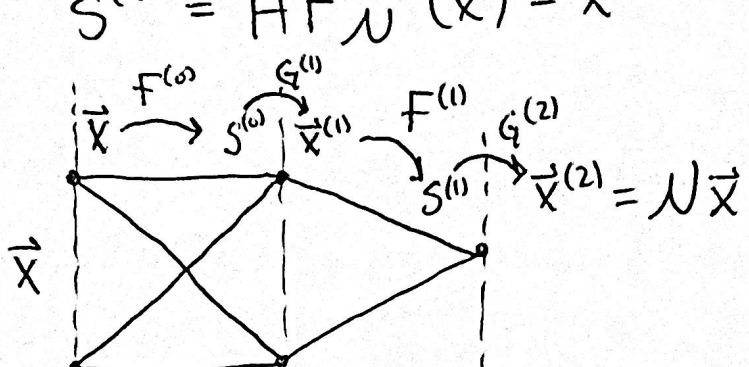
$$\frac{\partial J}{\partial W^{(i)}}, \frac{\partial J}{\partial \vec{b}^{(i)}} \qquad 0 \leq i \leq D-1$$

Definition The signal out of layer i for input $\vec{x}$ is

$$S^{(i)} = F^{(i)} G^{(i)} \cdots G^{(1)} F^{(0)} \vec{x} \qquad S^{(0)} = F^{(0)} \vec{x}$$

Note: $G^{(i+1)} S^{(i)} = H F_{\mathcal{N}}^{(i+1)}(\vec{x}) = \vec{x}^{(i+1)}$

E.g.



$$W^{(0)} = \begin{pmatrix} w_{11}^0 & w_{12}^0 \\ w_{21}^0 & w_{22}^0 \end{pmatrix} \qquad \vec{b}^{(0)} = \begin{pmatrix} b_1^0 \\ b_2^0 \end{pmatrix}$$

$$W^{(1)} = \begin{pmatrix} w_{11}^1 & w_{12}^1 \end{pmatrix} \qquad \vec{b}^{(1)} = \begin{pmatrix} b_1^1 \end{pmatrix}$$

$$S^{(0)} = W^{(0)} \vec{x} + \vec{b}^{(0)}$$

$$\vec{x}^{(1)} = G^{(1)}(S^{(0)}) = G^{(1)}\left(W^{(0)} \vec{x} + \vec{b}^{(0)}\right)$$

$$S^{(1)} = W^{(1)} G^{(1)}(S^{(0)}) + b^{(1)} = W^{(1)} \vec{x}^{(1)} + b^{(1)}$$

$$\vec{x}^{(2)} = G^{(2)}(S^{(1)}) = \mathcal{N}\vec{x}$$

Cost function $J(W,B) = \sum_{i=1}^{m} L(N\vec{x}_i, \vec{y}_i)$

Need to find derivatives of $L(N\vec{x}, \vec{y})$

Note: $\frac{\partial}{\partial v}(f \circ g)(v) = \left.\frac{\partial f(w)}{\partial w}\right|_{w=g(v)} \frac{\partial}{\partial v} g(v)$

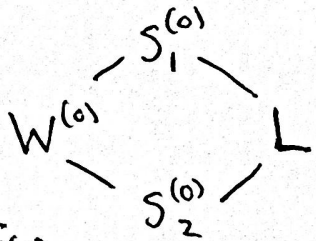
write $\left.\frac{\partial f(w)}{\partial w}\right|_{w=g(v)} = \frac{\partial f}{\partial g(v)}$

$$\frac{\partial L}{\partial b^{(1)}} = \frac{\partial L}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial b^{(1)}} = \frac{\partial L}{\partial s^{(1)}} := \delta^{(1)}$$

$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial W^{(1)}} = \frac{\partial L}{\partial s^{(1)}} (\vec{x}^{(1)})^T = \delta^{(1)} (\vec{x}^{(1)})^T$$

$$\frac{\partial L}{\partial b^{(0)}} = \frac{\partial L}{\partial s^{(0)}} \frac{\partial s^{(0)}}{\partial b^{(0)}} = \frac{\partial L}{\partial s^{(0)}} := \delta^{(0)}$$

$$\frac{\partial L}{\partial W^{(0)}} = \frac{\partial L}{\partial s^{(0)}} \frac{\partial s^{(0)}}{\partial W^{(0)}} = \frac{\partial L}{\partial s_1^{(0)}} \frac{\partial s_1^{(0)}}{\partial W^{(0)}} + \frac{\partial L}{\partial s_2^{(0)}} \frac{\partial s_2^{(0)}}{\partial W^{(0)}}$$



problem: this is a 3-tensor

$$= \frac{\partial L}{\partial s_1^{(0)}} \left[\frac{\partial}{\partial W^{(1)}} \left(W_{11}^{(0)} x_1 + W_{12}^{(0)} x_2 \right) \right] + \frac{\partial L}{\partial s_2^{(0)}} \left[\frac{\partial}{\partial W^{(0)}} \left(W_{21}^{(0)} x_1 + W_{22}^{(0)} x_2 \right) \right]$$

$$= \frac{\partial L}{\partial s_1^{(0)}} \begin{bmatrix} x_1 & x_2 \\ 0 & 0 \end{bmatrix} + \frac{\partial L}{\partial s_2^{(0)}} \begin{bmatrix} 0 & 0 \\ x_1 & x_2 \end{bmatrix}$$

$$= \begin{pmatrix} \frac{\partial L}{\partial s_1^{(0)}} \\ \frac{\partial L}{\partial s_2^{(0)}} \end{pmatrix} (x_1 \;\; x_2) = [redacted]$$

$$= (\delta^{(0)})^T \vec{x}^T = \text{outer}(\delta^{(0)}, \vec{x})$$

Still need $\delta^{(0)}$ and $\delta^{(1)}$

$$\delta^{(0)} = \frac{\partial L}{\partial s^{(0)}} = \frac{\partial L}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial s^{(0)}} \qquad s^{(1)} = W^{(1)} G^{(1)} s^{(0)} + b^{(1)}$$

$$= \delta^{(1)} W^{(1)} \frac{\partial G^{(1)}}{\partial s^{(0)}}$$

$$\delta^{(1)} = \frac{\partial L}{\partial s^{(1)}} = \frac{\partial L}{\partial N\vec{x}} \frac{\partial N\vec{x}}{\partial s^{(1)}} \qquad N\vec{x} = G^{(2)}(s^{(1)})$$

$$= \frac{\partial L}{\partial N\vec{x}} \frac{\partial G^{(2)}}{\partial s^{(1)}}$$

Backpropagation for $P(n_0, \ldots, n_D)$

(each layer fully connected to adjacent layers)

Cost function $J(W,B) = \frac{1}{m}\sum_{i=1}^{m} L(\mathcal{N}\vec{x}_i, \vec{y}_i)$

$$\delta^{(D-1)} = \frac{\partial L}{\partial s^{(D-1)}} = \frac{\partial L}{\partial \mathcal{N}\vec{x}} \frac{\partial \mathcal{N}\vec{x}}{\partial s^{(D-1)}}$$

$$= \frac{\partial L}{\partial \mathcal{N}\vec{x}} \frac{\partial G^{(D)}}{\partial s^{(D-1)}}$$

$$\mathcal{N}\vec{x} = G^{(D)}(s^{(D-1)})$$

and for $0 \le l < D-1$

$$\delta^{(l)} = \frac{\partial L}{\partial s^{(l)}} = \frac{\partial L}{\partial s^{(l+1)}} \frac{\partial s^{(l+1)}}{\partial s^{(l)}}$$

$$= \delta^{(l+1)} W^{(l+1)} \frac{\partial G^{(l+1)}}{\partial s^{(l)}}$$

$$s^{(l+1)} = W^{(l+1)} G^{(l+1)}(s^{(l)}) + b^{(l+1)}$$

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial s^{(l)}} \frac{\partial s^{(l)}}{\partial W^{(l)}}$$

$$= \sum_{k=1}^{n_{l+1}} \frac{\partial L}{\partial s_k^{(l)}} \frac{\partial s_k^{(l)}}{\partial W^{(l)}}$$

$$= \sum_{k=1}^{n_{l+1}} \delta_k^{(l)} \sum_{j=1}^{n_l} E_{kj} x_j^{(l)}$$

$$= \sum_{k,j} \delta_k^{(l)} x_j^{(l)} E_{kj}$$

$$= \text{outer}(\delta^{(l)}, x^{(l)})$$

$$s^{(l)} = W^{(l)} x^{(l)} + b^{(l)}$$

$$s_k^{(l)} = W_{k*}^{(l)} x^{(l)} + b_k^{(l)}$$

$$= \sum_{j=1}^{n_l} W_{kj}^{(l)} x_j^{(l)} + b_k^{(l)}$$

Note $\frac{\partial}{\partial W^{(l)}} W_{kj}^{(l)} = E_{kj}$

(E_{kj}: matrix with 1 in (k,j), 0 elsewhere)

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial s^{(l)}} \frac{\partial s^{(l)}}{\partial b^{(l)}} = \frac{\partial L}{\partial s^{(l)}} = \delta^{(l)}$$

Gradient for $\mathcal{N}$ with architecture $P(n_0, \ldots, n_D)$

Dataset X, Y with $X \in \mathbb{R}^{m \times n}$ $Y \in \mathbb{R}^{m \times k}$

$$J(W,B) = \frac{1}{m} \sum_{i=1}^{m} L(\mathcal{N}\vec{x}_i, \vec{y}_i)$$

$$\frac{\partial J}{\partial W^{(l)}} = \frac{1}{m} \sum_{i=1}^{m} \text{outer}(\delta_i^{(l)}, \vec{x}_i^{(l)})$$

$$\frac{\partial J}{\partial b^{(l)}} = \frac{1}{m} \sum_{i=1}^{m} \delta_i^{(l)}$$

where $\delta_i^{(D-1)} = \frac{\partial L}{\partial \mathcal{N}\vec{x}_i} \frac{\partial G^{(D)}}{\partial s_i^{(D-1)}}$

$$\delta_i^{(l)} = \delta_i^{(l+1)} W^{(l+1)} \frac{\partial G^{(l+1)}}{\partial s_i^{(l)}} \qquad 0 \le l < D-1$$

Regularization (See HW 4)

A Stochastic Gradient Descent Step

Pick $\alpha > 0$

Pick a batch $Ba \subseteq \{1, \ldots, m\}$ and approximate

$$J(W,B) \approx \frac{1}{|Ba|} \sum_{i \in Ba} L(\mathcal{N}\vec{x}_i, \vec{y}_i)$$ (add to list of costs)

for $i \in Ba$: x_i $\qquad\qquad$ $\mathcal{N}\vec{x}_i$

compute $[(x_i^{(0)}, s_i^{(0)}), (x_i^{(1)}, s_i^{(1)}), \ldots, (x_i^{(D-1)}, s_i^{(D-1)}), [x_i^{(D)}]]$

compute $\delta_i^{(D-1)}, \delta_i^{(D-2)}, \ldots, \delta_i^{(0)}$

compute $\frac{\partial J}{\partial W^{(l)}} \approx \frac{1}{|Ba|} \sum_{i \in Ba} \text{outer}(\delta_i^{(l)}, x_i^{(l)})$

$\frac{\partial J}{\partial b^{(l)}} \approx \frac{1}{|Ba|} \sum_{i \in Ba} \delta_i^{(l)}$

for l in range(D):

$W^{(l)} = W^{(l)} - \alpha \frac{\partial J}{\partial W^{(l)}}$

$b^{(l)} = b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}$

Common Cost (loss) functions : $J(W,B) = \frac{1}{m} \sum_{i=1}^{m} L(N\vec{x}_i, \vec{y}_i)$

Square Error Cost $L_{SE}(N\vec{x}, \vec{y}) = \|N\vec{x} - \vec{y}\|_2^2$ (MLE for gaussian p_{model})

Used for regression (mostly)

Cross Entropy Cost $L_{CE}(N\vec{x}, \vec{y}) = -\vec{y}^T \text{Log}(N\vec{x})$

$= -\sum_{i=1}^{n_D} y_i \log((N\vec{x})_i)$

Used for multiclass classification with softmax in last layer

Binary Cross Entropy Cost $L_{BCE}(N\vec{x}, \vec{y}) = -y \log(N\vec{x}) - (1-y) \log(1-N\vec{x})$

Used for binary classification with sigmoid output neurons (one output neuron $n_D = 1$)

(MLE ~~for~~ for ? p_{model} Bernoulli)

How do these affect backprop?

They only affect $\delta^{(D-1)}$

$$\delta^{(D-1)} = \frac{\partial L}{\partial \mathcal{N}\vec{x}} \frac{\partial \mathcal{N}\vec{x}}{\partial s^{(D-1)}}$$

for SE: $\frac{\partial L_{SE}}{\partial \mathcal{N}\vec{x}} = 2(\mathcal{N}\vec{x} - \vec{y})^T$

for CE: $\frac{\partial L_{CE}}{\partial \mathcal{N}\vec{x}} = \frac{\partial}{\partial \mathcal{N}\vec{x}}\left(-\vec{y}^T \text{Log}(\mathcal{N}\vec{x})\right)$

$$= -\vec{y}^T \begin{pmatrix} \frac{1}{(\mathcal{N}\vec{x})_1} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{(\mathcal{N}\vec{x})_{n_D}} \end{pmatrix}$$

for BCE: $\frac{\partial L_{BCE}}{\partial \mathcal{N}\vec{x}} = \frac{-y}{\mathcal{N}\vec{x}} + \frac{1-y}{1-\mathcal{N}\vec{x}}$

These lead to nice expressions for $\delta^{(D-1)}$ when you add specific output layer neurons (activations)

(see HW4)