Neural Network Training

Given training data **D**={**x**[m], **y**[m]}, m=1,2,.., M, NN training is to learn the NN parameters Θ by minimizing a loss function L(**D**: Θ), i.e.,

$$L(\mathbf{D}:\mathbf{\Theta}) = \frac{1}{M}\sum_{m=1}^{N} l(x[m], y[m], \mathbf{\Theta}) + \lambda R(\mathbf{\Theta})$$

$$\mathbf{\Theta}^* = \arg\min_{\mathbf{\Theta}} L(\mathbf{D}:\mathbf{\Theta})$$

where

$$\mathbf{\Theta} = [\mathbf{W}^1\ \mathbf{W}_0^1\ \mathbf{W}^2\ \mathbf{W}_0^2 \ldots \mathbf{W}^{L+1}\ \mathbf{W}_0^{L+1}]$$

Loss Function for Binary Logistic Classification

For each training sample (**x**[m], **y**[m]), where y[m] is a scalar, representing the probability of y[m]=1. Let $\hat{y}$ [m] be the estimated output probability via sigmoid function through the forward propagation based on current parameters.

Commonly used loss functions:

- Squared loss

  $l(y[m], \hat{y}[m]) = \frac{1}{2}(y[m] - \hat{y}[m])^t (y[m] - \hat{y}[m]))$

Loss function for multi-class logistic classification

For each training sample (**x**[m], **y**[m]), where **y**[m] follows 1-of-K encoding and let $\hat{\mathbf{y}}$ [m] be the estimated output vector via softmax through the forward propagation based on current parameters.

Commonly used loss functions:

- Squared loss

  $l(\mathbf{y}[m], \hat{\mathbf{y}}[m]) = \frac{1}{2}(\mathbf{y}[m] - \hat{\mathbf{y}}[m])^t (\mathbf{y}[m] - \hat{\mathbf{y}}[m])$

- Cross-entropy loss: $l(\mathbf{y}[m], \hat{\mathbf{y}}[m]) = -\sum_{k=1}^{K} \mathbf{y}[m][k] * \log(\hat{\mathbf{y}}[m][k]))$

Loss Function for Regression

For each training sample (**x**[m], y[m]), where y[m] is a real number and let $\hat{y}$[m] be the estimated output value through the forward propagation based on current parameters.

Squared loss :

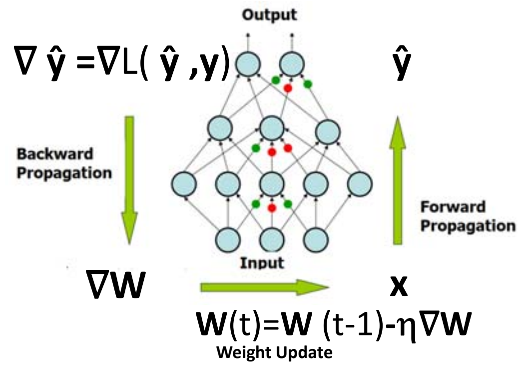
$l(y[m], \hat{y}[m]) = (y[m] - \hat{y}[m])^2$

Backpropagation algorithm

- Three Steps of computation:
  - **Forward pass**: run the NN and compute $\hat{\mathbf{y}}$
  - **Backward pass**: start at the output layer, compute the output gradient $\nabla\hat{\mathbf{y}}$, pass the output gradient backwards through the network, layer by layer, by recursively computing the weight gradients of each layer.
  - **Weight updates:** update the weights for each layer based on the estimated gradients

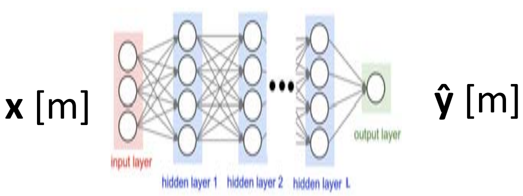
33

Neural Network Learning



Forward Propagation

1. Given current **W**s and an input x[m], compute **ŷ**[m]



$$\mathbf{H}^1[m] = \phi((\mathbf{W}^1)^t \mathbf{x}[m] + \mathbf{W}_0^1)$$
$$\mathbf{H}^2[m] = \phi((\mathbf{W}^2)^t \mathbf{H}^1[m] + \mathbf{W}_0^2)$$
....
$$\mathbf{H}^L[m] = \phi((\mathbf{W}^L)^t \mathbf{H}^{L-1}[m] + \mathbf{W}_0^L)$$
$$\hat{\mathbf{y}}[m] = g((\mathbf{W}^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1})$$

- The output is computed through a series of recursive composition from input layer through the hidden layer until the output layer.
- ϕ() is the activation function and g() is the output function. Note they apply individually to each element of the vector.

Derivatives with Matrices

Let $\mathbf{A}^{MxN}$ be a matrix and z be a scalar function of A

$$\frac{\partial z}{\partial \mathbf{A}}^{MxN} = \left[\frac{\partial z}{\partial \mathbf{A}[][1]}\ \frac{\partial z}{\partial \mathbf{A}[][2]} \cdots \frac{\partial z}{\partial \mathbf{A}[][N]}\right] = \begin{bmatrix} \frac{\partial z}{\partial \mathbf{A}[1][]} \\ \frac{\partial z}{\partial \mathbf{A}[2][]} \\ \ldots \\ \frac{\partial z}{\partial \mathbf{A}[M][]} \end{bmatrix} = \begin{pmatrix} \frac{\partial z}{\partial a_{11}} & \frac{\partial z}{\partial a_{12}} & \cdots & \frac{\partial z}{\partial a_{1N}} \\ \frac{\partial z}{\partial a_{21}} & \frac{\partial z}{\partial 2_{22}} & \cdots & \frac{\partial z}{\partial a_{2N}} \\ \ldots \\ \frac{\partial z}{\partial a_{M1}} & \frac{\partial z}{\partial a_{M2}} & \cdots & \frac{\partial z}{\partial a_{MN}} \end{pmatrix}$$

Derivatives with Matrices

- Vector by matrix . Let $\mathbf{X}^{Kx1}$ be a vector

$$\frac{\partial \mathbf{X}}{\partial \mathbf{A}} = \left[\frac{\partial \mathbf{X}_1}{\partial \mathbf{A}} \ \frac{\partial \mathbf{X}_2}{\partial \mathbf{A}} \ .... \frac{\partial \mathbf{X}_K}{\partial \mathbf{A}} \right]^{MxNxK} - \text{Tensor}$$

- Tensor vector multiplication

Let **Y** be a Kx1 vector

$$\left(\frac{\partial \mathbf{X}}{\partial \mathbf{A}}\right)^{MxNxK} \mathbf{Y}^{Kx1} = \left(\sum_{k=1}^{K} \frac{\partial \mathrm{X}_k}{\partial \mathbf{A}} \ \mathrm{Y}_k\right)^{MxN}$$

- Tensor matrix multiplication

Let **B** be a KxD matrix

$$\frac{\partial \mathbf{X}}{\partial \mathbf{A}} \mathbf{B} = \left[\frac{\partial \mathbf{X}}{\partial \mathbf{A}} \mathbf{B}_1 \ , \ \frac{\partial \mathbf{X}}{\partial \mathbf{A}} \mathbf{B}_2 ..., \frac{\partial \mathbf{X}}{\partial \mathbf{A}} \mathbf{B}_D \right]^{MxNxD}$$

where $\mathbf{B}_i$ is the ith column of **B**

Back Propagation

- The basic idea of backward propagation is to employ the gradient descent method to update the weight parameters for each layer.
- Let l(y, ŷ) be the loss function for the predicted ŷ and given y, and W^l be the weight for the lth layer.
- Using the gradient descent method, W^l at t th iteration can be updated by

$$\mathbf{W}^l[t] = \mathbf{W}^l[t-1] - \eta \nabla_{\mathbf{W}^l} l(\mathbf{y}, \hat{\mathbf{y}})$$

where

$\nabla_{\mathbf{W}^l} l(\mathbf{y}, \hat{\mathbf{y}})$ is the gradient of $l(\mathbf{y}, \hat{\mathbf{y}})$ w.r.t. W^1 and by chain rule it can be computed as follows

$$\nabla_{\mathbf{W}^l} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\mathbf{W}^l} = \frac{\partial \hat{\mathbf{y}}}{\mathbf{W}^l} \frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\hat{\mathbf{y}}} = \frac{\partial \hat{\mathbf{y}}}{\mathbf{W}^l} \nabla \hat{\mathbf{y}}$$

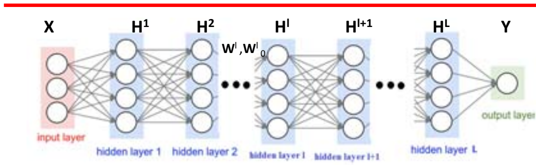
Backward Propagation

Gradient Chain rule

$$\mathbf{x} \longrightarrow \boxed{f(g(\phi(\Theta^t \mathbf{x},)))} \longrightarrow \mathbf{y}$$

$$\nabla \Theta = \frac{\partial \mathbf{y}}{\partial \Theta} = \frac{\partial f(g(\phi(\Theta^t \mathbf{x})))}{\partial \Theta} = \frac{\partial f(g(\phi(\Theta^t \mathbf{x})))}{\partial g(\phi(\mathbf{x}, \Theta))} \frac{\partial g(\phi(\Theta^t \mathbf{x}))}{\partial \phi(\mathbf{x}, \Theta)} \frac{\partial \phi(\Theta^t \mathbf{x})}{\partial (\Theta^t \mathbf{x})} \frac{\partial \Theta^t \mathbf{x}}{\partial \Theta}$$

Back Propagation-gradient chain rule



$\nabla \mathbf{W}^1$ can be recursively computed via gradient chain rule

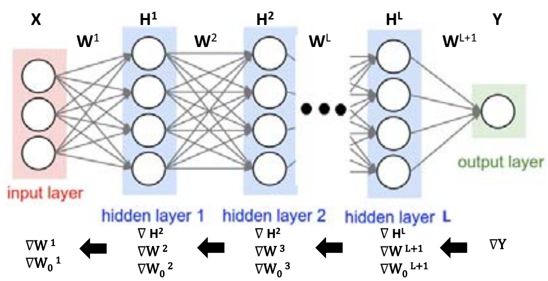
N_{l-1} x N_l $\quad$ $N_{l-1}xN_lxK$ $\quad$ K x 1 $\quad$ $N_{l-1}xN_lxN_L$ $\quad$ N_LxK $\quad$ Kx1

$$\nabla \mathbf{W}^l = \frac{\partial l(\mathbf{y}[m], \hat{\mathbf{y}}[m])}{\partial \mathbf{W}^l} = \frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{W}^l} \frac{\partial l(\mathbf{y}[m], \hat{\mathbf{y}}[m])}{\partial \hat{\mathbf{y}}} = \frac{\partial \mathbf{H}^L}{\partial \mathbf{W}^l} \frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{H}^L} \frac{\partial l(\mathbf{y}[m], \hat{\mathbf{y}}[m])}{\partial \hat{\mathbf{y}}}$$

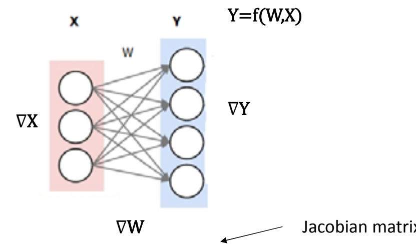
$$= \frac{\partial \mathbf{H}^{L-1}}{\partial \mathbf{W}^l} \frac{\partial \mathbf{H}^L}{\partial \mathbf{H}^{L-1}} \frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{H}^L} \frac{\partial l(\mathbf{y}[m], \hat{\mathbf{y}}[m])}{\partial \hat{\mathbf{y}}} = \frac{\partial \mathbf{H}^l}{\partial \mathbf{W}^l} \underbrace{\frac{\partial \mathbf{H}^{l+`1}}{\partial \mathbf{H}^l} \cdots \underbrace{\frac{\partial \mathbf{H}^L}{\partial \mathbf{H}^{L-1}} \underbrace{\frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{H}^L} \underbrace{\frac{\partial l(\mathbf{y}[m], \hat{\mathbf{y}}[m])}{\partial \hat{\mathbf{y}}}}_{\nabla \hat{\mathbf{y}}}}_{\nabla \mathbf{H}^L}}_{\nabla \mathbf{H}^{L-1}}}_{\nabla \mathbf{H}^{l+1}}$$

Pay attention to matrix dimensions consistency!

Back Propagation



Back Propagation for Each Layer



$$\nabla W = \frac{\partial f(\mathbf{W},\mathbf{X})}{\partial \mathbf{W}} \nabla \mathbf{Y}$$

$$\nabla \mathbf{X} = \frac{\partial f(\mathbf{W},\mathbf{X})}{\partial \mathbf{X}} \nabla \mathbf{Y}$$

Compute output gradient

For the mth training sample, given groundtruth **y**[m] and the computed **ŷ**[m] by forward propagation, and let l(**y**[m], **ŷ**[m]) be the loss function

$$\nabla \hat{\mathbf{y}}[m] = \nabla_{\hat{\mathbf{y}}} l(\mathbf{y}[m], \hat{\mathbf{y}}[m]) = \frac{\partial l(\mathbf{y}[m], \hat{\mathbf{y}}[m])}{\partial \hat{\mathbf{y}}}$$

Compute output gradient (cont'd)

1) For squared loss function

l(**y**[m], **ŷ**[m])=$\frac{1}{2}$ (**y**[m]-**ŷ**[m])^t(**y**[m]-**ŷ**[m])

$$\nabla \hat{\mathbf{y}}[m] = \frac{1}{2} \frac{\partial (\mathbf{y}[m]-\hat{\mathbf{y}}[m])^t (\mathbf{y}[m]-\hat{\mathbf{y}}[m])}{\partial \hat{\mathbf{y}}}$$
$$= -(\mathbf{y}[m]-\hat{\mathbf{y}}[m])$$

Compute output gradient (cont'd)

2) For cross-entropy loss function

l(**y**[m], **ŷ**[m])= $-\sum_{k=1}^{K} \mathbf{y}[m][k] * \log(\hat{\mathbf{y}}[m][k])$

$$\nabla\hat{\mathbf{y}}[m] = -\frac{\partial l(\mathbf{y}[m],\hat{\mathbf{y}}[m])}{\partial\hat{\mathbf{y}}}$$

$$= -\begin{pmatrix} \dfrac{\partial\sum_{k=1}^{K}\mathbf{y}[m][k]\log\hat{\mathbf{y}}[m][k]}{\hat{\mathbf{y}}[m][1]} \\ \dfrac{\partial\sum_{k=1}^{K}\mathbf{y}[m][k]\log\hat{\mathbf{y}}[m][k]}{\hat{\mathbf{y}}[m][2]} \\ \cdots\cdot \\ \dfrac{\partial\sum_{k=1}^{K}\mathbf{y}[m][k]\log\hat{\mathbf{y}}[m][K]}{\hat{\mathbf{y}}[m][K]} \end{pmatrix} = -\begin{pmatrix} \dfrac{\mathbf{y}[m][1]}{\hat{\mathbf{y}}[m][1]} \\ \dfrac{\mathbf{y}[m][2]}{\hat{\mathbf{y}}[m][2]} \\ \cdots\cdot \\ \dfrac{\mathbf{y}[m][K]}{\hat{\mathbf{y}}[m][K]} \end{pmatrix}$$

Backward Propagation-Output Layer

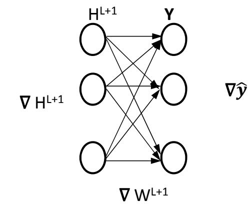
Given $\nabla\hat{\boldsymbol{y}}$[m], compute ∇H^{L} [m]and ∇W^{L+1} [m]

$$\hat{\mathbf{y}}[m] = g((\mathbf{W}^{L+1})^{t}\mathbf{H}^{L}[m] + \mathbf{W}_0^{L+1})$$

$$\nabla\mathbf{W}^{L+1}[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{W}^{L+1}}\nabla\hat{\mathbf{y}}[m]$$

$$\nabla\mathbf{W}_0^{L+1}[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{W}_0^{L+1}}\nabla\hat{\mathbf{y}}[m]$$

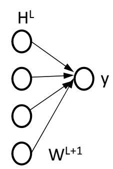
$$\nabla\mathbf{H}^{L}[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{H}^{L}[m]}\nabla\hat{\mathbf{y}}[m]$$



where g() is the output function

Backward Propagation-Output Layer

For regression, the output function is a scalar node and g() is



$$\hat{\mathbf{y}}[m] = g((\mathbf{W}^{L+1})^{t}\mathbf{H}^{L}[m]) = (\mathbf{W}^{L+1})^{t}\mathbf{H}^{L}[m] + \mathbf{W}_0^{L+1}$$

$$\nabla\mathbf{W}^{L+1}[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{W}^{L+1}}\nabla\hat{\mathbf{y}}[m] = \mathbf{H}^{L}[m]\nabla\hat{\mathbf{y}}[m]$$

$$\nabla\mathbf{W}_0^{L+1}[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{W}_0^{L+1}}\nabla\hat{\mathbf{y}}[m] = \nabla\hat{\mathbf{y}}[m]$$

$$\nabla\mathbf{H}^{L}[m] = \frac{\partial\hat{\mathbf{y}}[m]}{\partial\mathbf{H}^{L}}\nabla\hat{\mathbf{y}}[m] = \mathbf{W}^{L+1}\nabla\hat{\mathbf{y}}[m]$$

Note $\mathbf{W}^{L+1}$ is a vector and $\nabla\hat{\boldsymbol{y}}$ is a scalar.

Backward Propagation-Output Layer

For binary classification yϵ\{+1,-1\}, the output is a probability scalar node and output function is $\sigma()$

$$\hat{\mathbf{y}}[m] = \sigma(\mathbf{z}[m]), \text{ where } \mathbf{z}[m] = (\mathbf{W}^{L+1})^{t}\mathbf{H}^{L}[m] + \mathbf{W}_0^{L+1}$$

$$\nabla\mathbf{W}^{L+1}[m] = \frac{\partial\sigma(\mathbf{z}[m])}{\partial\mathbf{W}^{L+1}}\nabla\hat{\mathbf{y}}[m]$$

$$= \frac{\partial\sigma(\mathbf{z}[m])}{\partial\mathbf{z}[m]}\frac{\partial\mathbf{z}[m]}{\partial\mathbf{W}^{L+1}}\nabla\hat{\mathbf{y}}[m]$$

$$= \sigma(\mathbf{z}[m])(1-\sigma(\mathbf{z}[m]))\mathbf{H}^{L}[m]\nabla\hat{\mathbf{y}}[m]$$

$$\nabla\mathbf{W}_0^{L+1}[m] = \frac{\partial\sigma(\mathbf{z}[m])}{\partial\mathbf{W}_0^{L+1}}\nabla\hat{\mathbf{y}}[m]$$

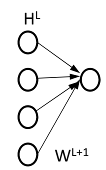
$$= \frac{\partial\sigma(\mathbf{z}[m])}{\partial\mathbf{z}[m]}\frac{\partial(\mathbf{z}[m])}{\partial\mathbf{W}_0^{L+1}}\nabla\hat{\mathbf{y}}[m]$$

$$= \sigma(\mathbf{z}[m])(1-\sigma(\mathbf{z}[m]))\nabla\hat{\mathbf{y}}[m]$$

$$\nabla\mathbf{H}^{L}[m] = \frac{\partial\sigma(\mathbf{z}[m])}{\partial\mathbf{H}^{L}}\nabla\hat{\mathbf{y}}[m]$$

$$= \frac{\partial\sigma(\mathbf{z}[m])}{\partial\mathbf{z}[m]}\frac{\partial\mathbf{z}[m]}{\partial\mathbf{H}^{L}}\mathbf{W}^{L+1}\nabla\hat{\mathbf{y}}[m]$$

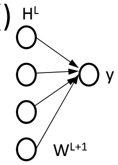
$$= \sigma(\mathbf{z}[m])(1-\sigma(\mathbf{z}[m]))\mathbf{W}^{L+1}\nabla\hat{\mathbf{y}}[m]$$



Note $\mathbf{W}^{L+1}$ is a vector and $\nabla\hat{\boldsymbol{y}}$ is a scalar.

Backward Propagation-Output Layer

For binary classification yϵ{1,0}, the output is a scalar node and output function is $\sigma()$



$$\hat{\mathbf{y}}[m] = \frac{e^{\mathbf{z}[m]}}{1+e^{\mathbf{z}[m]}}, \text{ where } \mathbf{z}[m] = (\mathbf{W}^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}$$

$$\begin{aligned}
\nabla \mathbf{W}^{L+1}[m] &= \frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{W}^{L+1}} \nabla \hat{\mathbf{y}}[m] \\
&= \frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{z}[m]} \frac{\partial \mathbf{z}[m]}{\partial \mathbf{W}^{L+1}} \nabla \hat{\mathbf{y}}[m] \\
&= (\hat{y}[m])^2 \mathbf{H}^L[m] \nabla \hat{\mathbf{y}}[m]
\end{aligned}$$

$$\text{Note } \frac{\partial \hat{y}[m]}{\partial \mathbf{z}[m]} = (\hat{y}[m])^2$$

$$\begin{aligned}
\nabla \mathbf{W}_0^{L+1}[m] &= \frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{W}_0^{L+1}} \nabla \hat{\mathbf{y}}[m] \\
&= \frac{\partial \hat{\mathbf{y}}[m]}{\partial \mathbf{z}[m]} \frac{\partial (\mathbf{z}[m])}{\partial \mathbf{W}_0^{L+1}} \nabla \hat{\mathbf{y}}[m] \\
&= (\hat{y}[m])^2 \nabla \hat{\mathbf{y}}[m]
\end{aligned}$$

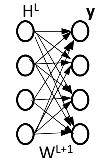
$$\begin{aligned}
\nabla \mathbf{H}^L[m] &= \frac{\partial \hat{y}[m]}{\partial \mathbf{H}^L} \nabla \hat{\mathbf{y}}[m] \\
&= \frac{\partial \hat{y}[m]}{\partial \mathbf{z}[m]} \frac{\partial \mathbf{z}[m]}{\partial \mathbf{H}^L} \mathbf{W}^{L+1} \nabla \hat{\mathbf{y}}[m] \\
&= (\hat{\mathbf{y}}[m])^2 \mathbf{W}^{L+1} \nabla \hat{\mathbf{y}}[m]
\end{aligned}$$

Note $\mathbf{W}^{L+1}$ is a vector and $\nabla\hat{y}$ is a scalar.

57

Backward Propagation-Output Layer

For multi-class classification, the output has multiple nodes and g() is multi-class Sigmoid function $\sigma_M()$



$$\text{Let } \mathbf{z}[m] = (\mathbf{W}^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1})$$

$$\hat{\mathbf{y}}[m] = \sigma_M(\mathbf{z}[m]) = \begin{pmatrix} \sigma_M(\mathbf{z}[m][1]) \\ \sigma_M(\mathbf{z}[m][2]) \\ \ldots \\ \sigma_M(\mathbf{z}[m][K]) \end{pmatrix}$$

$$\text{where } \mathbf{z}[m][k] = (\mathbf{W}_k^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[k]$$

$$\sigma_M(\mathbf{z}[m][k]) = \frac{\exp((\mathbf{W}_k^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[k])}{\sum_{j=1}^{K} \exp((\mathbf{W}_j^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[j])}$$

$$\begin{aligned}
\nabla \mathbf{W}^{L+1}[m] &= \frac{\partial \sigma_M(\mathbf{z}[m])}{\partial \mathbf{W}^{L+1}} \nabla \hat{\mathbf{y}}[m] \\
&= \sum_{i=1}^{K} \frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}^{L+1}} \nabla \hat{\mathbf{y}}[m][i] \\
&= \sum_{i=1}^{K} \left[\frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_1^{L+1}} \; \frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_2^{L+1}} \ldots \frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_K^{L+1}} \right] \nabla \hat{\mathbf{y}}[m][i]
\end{aligned}$$

where

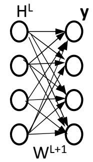
$$\frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_k^{L+1}} = \begin{cases} \sigma_M(\mathbf{z}[m][k])(1-\sigma_M(\mathbf{z}[m][k]))\mathbf{H}^L[m] & \text{if } i=k \\ -\sigma_M(\mathbf{z}[m][k])\sigma_M(\mathbf{z}[m][i])\mathbf{H}^L[m] & \text{else} \end{cases}$$

Note $\mathbf{W}^{L+1}$ is a matrix and $\nabla\hat{y}$ is a vector

59

Backward Propagation-Output Layer

For multi-class classification, the output has multiple nodes and g() is multi-class Sigmoid function $\sigma_M()$



$$\text{Let } \mathbf{z}[m] = (\mathbf{W}^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1})$$

$$\hat{\mathbf{y}}[m] = \sigma_M(\mathbf{z}[m]) = \begin{pmatrix} \sigma_M(\mathbf{z}[m][1]) \\ \sigma_M(\mathbf{z}[m][2]) \\ \ldots \\ \sigma_M(\mathbf{z}[m][K]) \end{pmatrix}$$

$$\text{where } \mathbf{z}[m][k] = (\mathbf{W}_k^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[k]$$

$$\sigma_M(\mathbf{z}[m][k]) = \frac{\exp((\mathbf{W}_k^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[k])}{\sum_{j=1}^{K} \exp((\mathbf{W}_j^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[j])}$$

$$\begin{aligned}
\nabla \mathbf{W}_0^{L+1}[m] &= \frac{\partial \sigma_M(\mathbf{z}[m])}{\partial \mathbf{W}_0^{L+1}} \nabla \hat{\mathbf{y}}[m] \\
&= \sum_{i=1}^{K} \frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^{L+1}} \nabla \hat{\mathbf{y}}[m][i] = \sum_{i=1}^{K} \left[\frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^{L+1}[1]} \; \frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^{L+1}[2]} \ldots \frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^{L+1}[K]} \right]^t \nabla \hat{\mathbf{y}}[m][i]
\end{aligned}$$

where

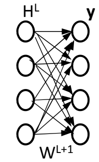
$$\frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^{L+1}[k]} = \begin{cases} \sigma_M(\mathbf{z}[m][k])(1-\sigma_M(\mathbf{z}[m][k])) & \text{if } i=k \\ -\sigma_M(\mathbf{z}[m][i])\sigma_M(\mathbf{z}[m][k]) & \text{else} \end{cases}$$

Note $\mathbf{W}^{L+1}$ is a matrix and $\nabla\hat{y}$ is a vector

61

Backward Propagation-Output Layer

For multi-class classification, the output has multiple nodes and g() is multi-class Sigmoid function $\sigma_M()$



$$\text{Let } \mathbf{z}[m] = (\mathbf{W}^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1})$$

$$\hat{\mathbf{y}}[m] = \sigma_M(\mathbf{z}[m]) = \begin{pmatrix} \sigma_M(\mathbf{z}[m][1]) \\ \sigma_M(\mathbf{z}[m][2]) \\ \ldots \\ \sigma_M(\mathbf{z}[m][K\}) \end{pmatrix}$$

$$\text{where } \mathbf{z}[m][k] = (\mathbf{W}_k^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[k]$$

$$\sigma_M(\mathbf{z}[m][k]) = \frac{\exp((\mathbf{W}_k^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[k])}{\sum_{j=1}^{K} \exp((\mathbf{W}_j^{L+1})^t \mathbf{H}^L[m] + \mathbf{W}_0^{L+1}[j])}$$

$$\begin{aligned}
\nabla \mathbf{H}^L[m] &= \frac{\partial \sigma_M(\mathbf{z}[m])}{\partial \mathbf{H}^L} \nabla \hat{\mathbf{y}}[m] \\
&= \sum_{i=1}^{K} \frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{H}^L} \nabla \hat{\mathbf{y}}[m][i]
\end{aligned}$$

where

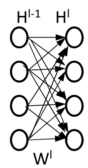
$$\frac{\partial \sigma_M(\mathbf{z}[m][i])}{\partial \mathbf{H}^L} = \sigma_M(\mathbf{z}[m][i])[\mathbf{W}_i^{L+1} - \sum_{j=1}^{K} \sigma_M(\mathbf{z}[m][j])\mathbf{W}_j^{L+1}]$$

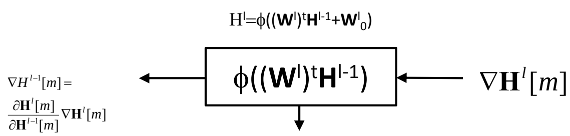
Note $\mathbf{W}^{L+1}$ is a matrix and $\nabla\hat{y}$is a vector

63

Backward Propagation-Hidden Layers

For each hidden layer l, let H^l be its output, H^{l-1} be its inputs, and W^l be its current parameters. Given $\nabla H^l[m]$, compute $\nabla H^{l-1}[m]$, $\nabla W^l[m]$, and $\nabla W_0^l[m]$



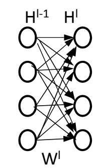


$$\nabla \mathbf{W}^l[m]=\frac{\partial \mathbf{H}^l[m]}{\partial \mathbf{W}^l}\nabla \mathbf{H}^l[m] \quad \nabla \mathbf{W}_0^l[m]=\frac{\partial \mathbf{H}^l[m]}{\partial \mathbf{W}_0^l}\nabla \mathbf{H}^l[m]$$

Backward Propagation-Hidden Layers

Let $\mathbf{z}[m]=(\mathbf{W}^l)^t\mathbf{H}^{l-1}[m]+\mathbf{W}_0^l$

$$\mathbf{H}^l[m]=\phi(\mathbf{z}[m])=\begin{bmatrix}\phi(\mathbf{z}[m][1])\\ \phi(\mathbf{z}[m][2])\\ \dots\\ \phi(\mathbf{z}[m][N_l])\end{bmatrix}, \text{ where } \mathbf{z}[m][i]=(\mathbf{W}_i^l)^t\mathbf{H}^{l-1}[m]+\mathbf{W}_0^l[i]$$



$$\nabla \mathbf{W}^l[m]=\frac{\partial \phi(\mathbf{z}[m])}{\partial \mathbf{W}^l}\nabla \mathbf{H}^l[m]$$

$$=\sum_{i=1}^{N_l}\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}^l}\nabla \mathbf{H}^l[m][i]$$

$$=\sum_{i=1}^{N_l}\left[\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_1^l}\ \frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_2^l}\dots\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_{N_l}^l}\right]\nabla \mathbf{H}_i^l[m][i], \text{ where } \mathrm{W_j} \text{ is the jth column of W}$$

For sigmoid transfer function

$$\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_k^l}=\begin{cases}\sigma(\mathbf{z}[m][k])(1-\sigma(\mathbf{z}[m][k]))\mathbf{H}^{l-1}[m] & \text{if } \mathrm{i=k}\\ 0 & \text{else}\end{cases}$$

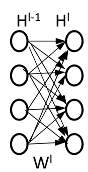
For ReLU transfer function

$$\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_k^l}=\frac{\partial \mathrm{Re}LU(\mathbf{z}[m][i])}{\partial \mathbf{W}_k^l}=\begin{cases}\mathbf{H}^{l-1}[m] & \text{if i = k and } \mathbf{z}[m][i]>0\\ 0 & \text{else}\end{cases}$$

Backward Propagation-Hidden Layers

Let $\mathbf{z}[m]=(\mathbf{W}^l)^t\mathbf{H}^{l-1}[m]+\mathbf{W}_0^l$

$$\mathbf{H}^l[m]=\phi(\mathbf{z}[m])=\begin{bmatrix}\phi(\mathbf{z}[m][1])\\ \phi(\mathbf{z}[m][2])\\ \dots\\ \phi(\mathbf{z}[m][N_l])\end{bmatrix}, \text{ where } \mathbf{z}[m][i]=(\mathbf{W}_i^l)^t\mathbf{H}^{l-1}[m]+\mathbf{W}_0^l[i]$$



$$\nabla \mathbf{W}_0^l[m]=\frac{\partial \phi(\mathbf{z}[m])}{\partial \mathbf{W}_0^l[m]}\nabla \mathbf{H}^l[m]$$

$$=\sum_{i=1}^{N_l}\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^l}\nabla \mathbf{H}^l[m][i]$$

$$=\sum_{i=1}^{N_l}\left[\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^l[1]}\ \frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^l[2]}\dots\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^l[N_l]}\dots\right]^t\nabla \mathbf{H}^l[m][i]$$

For sigmoid transfer function

$$\frac{\partial \phi(\mathbf{z}_i[m])}{\partial \mathbf{W}_0^l[k]}=\begin{cases}\sigma(\mathbf{z}_i[m])(1-\sigma(\mathbf{z}_i[m])) & \text{if } \mathrm{i=k}\\ 0 & \text{else}\end{cases}$$

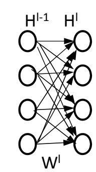
For ReLU transfer function

$$\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^l[k]}=\frac{\partial \mathrm{Re}LU(\mathbf{z}[m][i])}{\partial \mathbf{W}_0^l[k]}=\begin{cases}1 & \text{if i = k and } \mathbf{z}[m][i]>0\\ 0 & \text{else}\end{cases}$$

Backward Propagation-Hidden Layers

Let $\mathbf{z}[m]=(\mathbf{W}^l)^t\mathbf{H}^{l-1}[m]+\mathbf{W}_0^l$

$$\mathbf{H}^l[m]=\phi(\mathbf{z}[m])=\begin{bmatrix}\phi(\mathbf{z}[m][1])\\ \phi(\mathbf{z}[m][2])\\ \dots\\ \phi(\mathbf{z}[m][N_l])\end{bmatrix}, \text{ where } \mathbf{z}[m][i]=(\mathbf{W}_i^l)^t\mathbf{H}^{l-1}[m]+\mathbf{W}_0^l[i]$$



$$\nabla \mathbf{H}^{l-1}[m]=\frac{\partial \phi(\mathbf{z}[m])}{\partial \mathbf{H}^{l-1}}\nabla \mathbf{H}^l[m]$$

$$=\sum_{i=1}^{N_l}\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{H}^{l-1}}\nabla \mathbf{H}^l[m][i]$$

For sigmoid transfer function

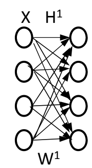
$$\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{H}^{l-1}}=\sigma(\mathbf{z}[m][i])(1-\sigma(\mathbf{z}[m][i]))\mathbf{W}_i^l$$

For ReLU transfer function

$$\frac{\partial \phi(\mathbf{z}[m][i])}{\partial \mathbf{H}^{l-1}}=\frac{\partial \mathrm{Re}LU(\mathbf{z}[m][i])}{\partial \mathbf{H}^{l-1}}=\begin{cases}\mathbf{W}_i^l & \mathbf{z}[m][i]>0\\ 0 & \text{else}\end{cases}$$

Backward Propagation-First Hidden Layer

For $l=1$, given $\nabla \mathbf{H}^1[m]$, compute $\nabla W^1[m]$



$$\mathbf{z}[m] = (\mathbf{W}^1)^t \mathbf{X}[m] + \mathbf{W}_0^1,\ \mathbf{H}^1[m] = \phi(\mathbf{z}[m])$$

$$\nabla \mathbf{W}^1[m] = \frac{\partial \mathbf{H}^1[m]}{\partial \mathbf{W}^1} \nabla \mathbf{H}^1[m] = \frac{\partial \sigma(\mathbf{z}[m])}{\partial \mathbf{W}^1} \nabla \mathbf{H}^1[m]$$

$$= \sum_{i=1}^{N_1} \frac{\partial \sigma(\mathbf{z}[m][i])}{\partial \mathbf{W}^1} \nabla \mathbf{H}^1[m][i] = \sum_{i=1}^{N_1} \left[\frac{\partial \sigma(\mathbf{z}[m][i])}{\partial \mathbf{W}_1{}^1} \frac{\partial \sigma(\mathbf{z}[m][i])}{\partial \mathbf{W}_2{}^1} \dots \frac{\partial \sigma(\mathbf{z}[m][i])}{\partial \mathbf{W}_{N_1}{}^1} \right] \nabla \mathbf{H}^1[m][i]$$

For sigmoid activation function

$$\frac{\partial \sigma(\mathbf{z}[m][i])}{\partial \mathbf{W}_k^1} = \begin{cases} \sigma(\mathbf{z}[m][k])(1-\sigma(\mathbf{z}[m][k]))\mathbf{X}[m] & \text{if } \mathrm{i}=\mathrm{k} \\ 0 & \text{else} \end{cases}$$

For ReLU activation function

$$\frac{\partial \mathrm{ReLU}(\mathbf{z}[m][i])}{\partial \mathbf{W}_k^1} = \begin{cases} \mathbf{X}[m] \text{ if } \mathrm{i}=\mathrm{k} \text{ and } \mathbf{z}[m][i] > 0 \\ 0 \text{ else} \end{cases}$$

Add Regularization

For squared of L2-norm, its gradient is

$$|\mathbf{W}^l|_2 = [\sum_{i=1}^{N_{l-1}} \sum_{j=1}^{N_l} (\mathbf{W}^l[i][j])^2]^{\frac{1}{2}}$$

$$\nabla R_{L2}(\mathbf{W}^l) = \frac{\partial(|\mathbf{W}^l|_2)^2}{\partial \mathbf{W}^l} = \frac{\partial(\sum_{i=1}^{N_{l-1}} \sum_{j=1}^{N_l} (\mathbf{W}^l[i][j])^2)}{\partial \mathbf{W}^l}$$

$$= \sum_{i=1}^{N_{l-1}} \sum_{j=1}^{N_l} \frac{\partial((\mathbf{W}^l[i][j])^2)}{\partial \mathbf{W}^l} = \sum_{i=1}^{N_{l-1}} \sum_{j=1}^{N_l} \frac{\partial((\mathbf{W}^l[i][j])^2)}{\partial \mathbf{W}^l[i][j]} \frac{\partial \mathbf{W}^l[i][j]}{\partial \mathbf{W}^l}$$

$$= \sum_{i=1}^{N_{l-1}} \sum_{j=1}^{N_l} 2\mathbf{W}^l[i][j] \frac{\partial \mathbf{W}^l[i][j]}{\partial \mathbf{W}^l}$$

$$= \{2\mathbf{W}^l[i][j]\}$$

where $\{\mathbf{W}^l[i][j]\}$ is a matrix, whose ith row and jth column entry is $2\mathbf{W}$[i][j]

Add Regularization

For L-1 norm, its gradient is

$$|\mathbf{W}^l|_1 = \max_{1 \le j \le N_l} \sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][j]| = \max(\sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][1]|, \sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][2]|, \dots, \sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][N_l]|)$$

$$\nabla R_{L1}(\mathbf{W}^l) = \frac{\partial |\mathbf{W}^l|_1}{\partial \mathbf{W}^l} = \left[\frac{\partial \max_{1<j<N_l} \sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][j]|}{\partial \mathbf{W}_1{}^l} \frac{\partial \max_{1<j<N_l} \sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][j]|}{\partial \mathbf{W}_2{}^l} \dots \frac{\partial \max_{1<j<N_l} \sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][j]|}{\partial \mathbf{W}_{N_l}{}^l} \right]$$

$$= \begin{bmatrix} 0\ 0\ sign(\mathbf{W}^l[1][j^*]) \ \dots \ 0 \\ 0\ 0\ sign(\mathbf{W}^l[2][j^*]) \ \dots. \ 0 \\ \dots \\ 0\ 0\ sign(\mathbf{W}^l[N_{l-1}][j^*]) \dots 0 \end{bmatrix} \quad j^* = \arg\max_j (\sum_{i=1}^{N_{l-1}} |\mathbf{W}^l[i][j]|)$$

Add Regularization

For regularization on $\mathbf{W}^l{}_0$, we can use the vector regularization equations to derive its L1 and L2-squared norms

Parameter Updating

- Update the parameters $\mathbf{W}^l$ for each layer
  For l=1 to L+1 do

$$\mathbf{W}^{l}[t]=\mathbf{W}^{l}[t-1]-\eta_l(\nabla\mathbf{W}^{l}[m]+\lambda\frac{\partial R(\mathbf{W}^{l})}{\partial\mathbf{W}^{l}})$$

$$\mathbf{W}_0^{l}[t]=\mathbf{W}_0^{l}[t-1]-\eta_{l0}(\nabla\mathbf{W}_0^{l}[m]+\lambda_0\frac{\partial R(\mathbf{W}_0^{l})}{\partial\mathbf{W}_0^{l}})$$

76

Back Propagation Steps by sample

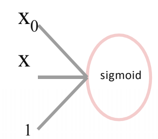
- Initialize the weights
- For each training sample $\mathbf{x}[m]$, perform forward propagation to compute $\nabla\hat{\mathbf{y}}[m]$
- For hidden layers l=L+1 to 1 do (note $\mathbf{H}^{L+1}=\mathbf{y}$, $\mathbf{H}^0=\mathbf{x}$)
  Compute $\nabla\mathbf{W}^l(m)$, $\nabla\mathbf{W}_0^l(m)$, and $\nabla H^{l-1}(m)$
  For l=1 to N_{l+1}
  Update weights for each layer
  $\mathbf{W}^l[t]=\mathbf{W}^l[t-1]-\eta_l(\nabla\mathbf{W}^l(m)+\lambda\frac{\partial R(\mathbf{W}^l)}{\partial\mathbf{W}^l})$
  Update bias weights for each layer
  $\mathbf{W}_0^l[t]=\mathbf{W}_0^l[t-1]-\eta_{l0}(\nabla\mathbf{W}_0^l(m)+\lambda\frac{\partial R(\mathbf{W}_0^l)}{\partial\mathbf{W}_0^l})$
- Repeat for all training samples until convergence

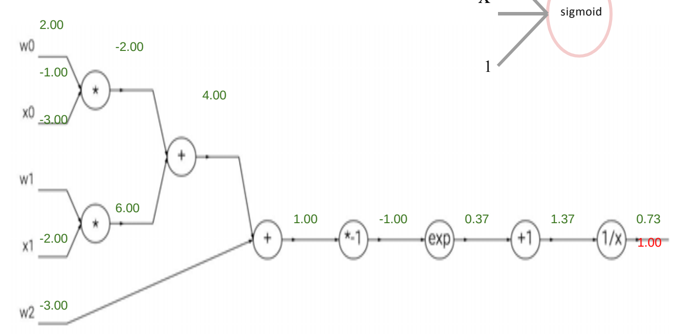
Back Propagation Steps by Batch

- Initialize the weights
- Randomly select from all training samples a mini-batch $\mathbf{D}_i$ of size M_i.
- For each training sample $\mathbf{x}[m]$ in $\mathbf{D}_i$, perform forward propagation to compute $\nabla\hat{\mathbf{y}}[m]$
- For hidden layers l=L+1 to 1 do (note $\mathbf{H}^{L+1}=\mathbf{y}$, $\mathbf{H}^0=\mathbf{x}$)
  Compute $\nabla\mathbf{W}^l(m)$, $\nabla\mathbf{W}_0^l(m)$, and $\nabla H^{l-1}(m)$
  Compute the average weight gradients for all samples in $\mathbf{D}_i$
  $\nabla\overline{\mathbf{W}}^l=\frac{1}{\mathbf{M}_i}\sum_{m=1}^{\mathbf{M}_i}\nabla\overline{\mathbf{W}}^l[m]$ $\quad\nabla\overline{\mathbf{W}}_0^l=\frac{1}{\mathbf{M}_i}\sum_{m=1}^{\mathbf{M}_i}\nabla\overline{\mathbf{W}}_0^l[m]$
  For l=1 to N_{l+1}
  Update weights for each layer $\mathbf{W}^l[t]=\mathbf{W}^l[t-1]-\eta_l(\nabla\overline{\mathbf{W}}^l+\lambda\frac{\partial R(\mathbf{W}^l)}{\partial\mathbf{W}^l})$
  Update bias weights for each layer $\mathbf{W}_0^l[t]=\mathbf{W}_0^l[t-1]-\eta_{l0}(\nabla\overline{\mathbf{W}}_0^l+\lambda\frac{\partial R(\mathbf{W}^l)}{\partial\mathbf{W}_0^l})$
- Repeat by selecting another mini-batch until convergence

Backward Computation

$f(x_0, x_1) = 1/(1+\exp(-(w_0x_0+w_1x_1+w_2)))$

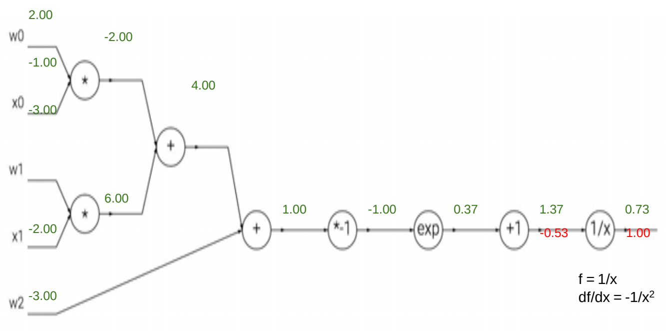




http://cs231n.github.io/optimization-2/

Backward Computation

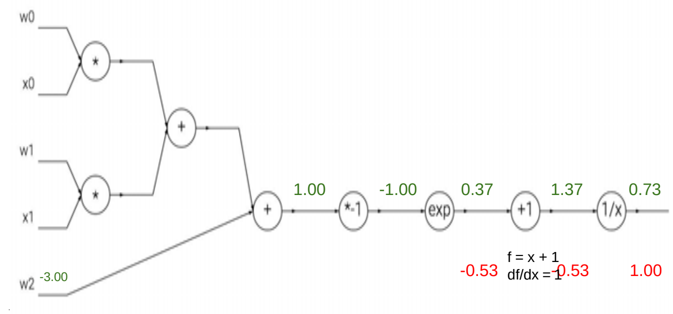
$f(x_0, x_1) = 1/\ (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$



http://cs231n.github.io/optimization-2/

Backward Computation

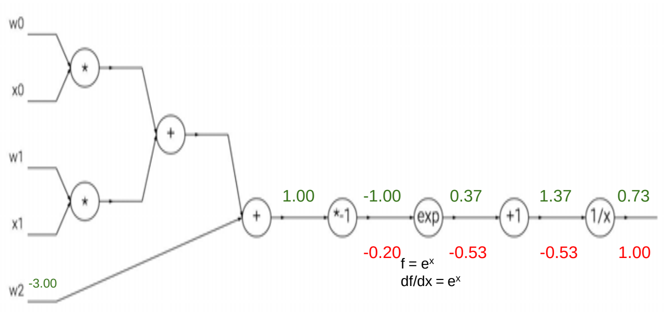
$f(x_0, x_1) = 1/\ (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$



http://cs231n.github.io/optimization-2/

Backward Computation

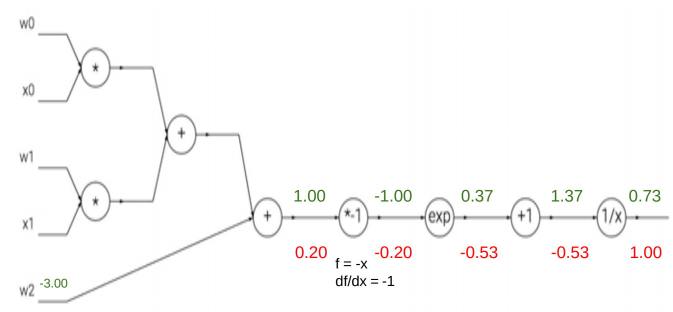
$f(x_0, x_1) = 1/\ (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$



http://cs231n.github.io/optimization-2/

Backward Computation

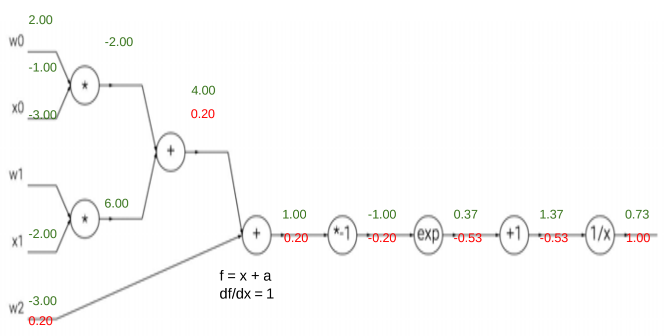
$f(x_0, x_1) = 1/\ (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$



http://cs231n.github.io/optimization-2/

Backward Computation

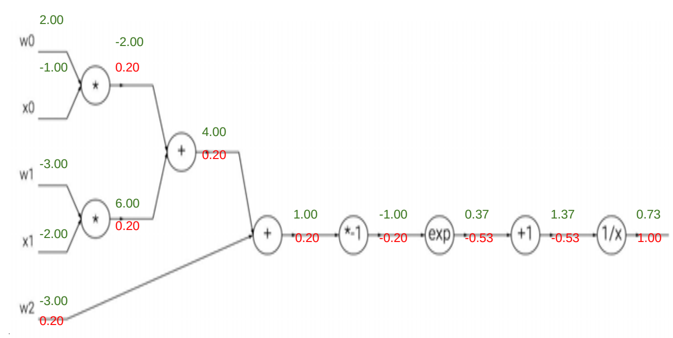
$f(x_0, x_1) = 1/\ (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$



http://cs231n.github.io/optimization-2/

Backward Computation

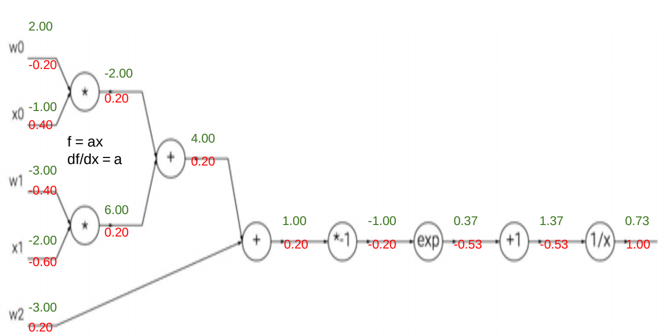
$f(x_0, x_1) = 1/\ (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$



http://cs231n.github.io/optimization-2/

Backward Computation

$f(x_0, x_1) = 1/\ (1 + \exp(-(w_0x_0 + w_1x_1 + w_2)))$



http://cs231n.github.io/optimization-2/

Training

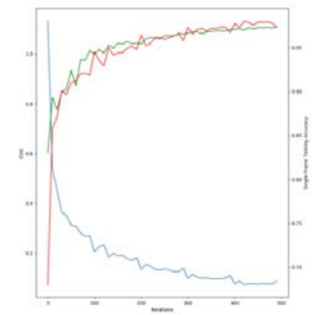
- The training continues epoch by epoch. Each epoch goes through all training samples (all mini-batches
- For each epoch, training samples order may be randomized to avoid any correlations between epochs
- The learning continues until the stopping criterion is satisfied.

98

Stopping criterions

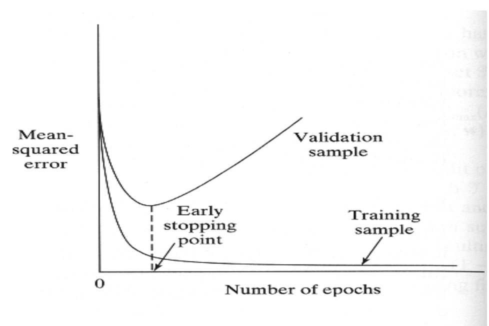
- Average loss change:
  - Back-prop is considered to have converged when the absolute rate of change in the average loss function per epoch is sufficiently small
- Classification performance
  - After each epoch, the NN is tested on
    - ➢ the training data to evaluate classification performance
    - ➢ testing data to evaluate the generalization performance
    - ➢ If the performance does not improve further, stop.

NN Training



NOTE: Red is training accuracy, green is testing accuracy, and blue is training loss. (It's not labeled on the graph.)

Early stopping



Initializations

- Initialize all biases to zero
- Initialize the weights to small non-zero random values. They should not be the same or all equal to zero. For example, w=N(0,1)

Hyper-parameter Tuning

On the validation dataset (different from testing data),

- Use the grid method to greedily tune the hyper-parameters
- The grid search method first estimates the range for each parameter and then quantizes each parameter into a finite set of intervals.
- Given initial values of hyper- parameters, it then tunes each hyper-parameter individually, while fixing other parameters.
- The process repeats until convergence.

Hyper-parameter Tuning

Adaptive learning rate

- Learning rate should vary with the iteration. It can be set as a function of the gradient magnitude. Two common choice of choosing learning rates are O(1/t) and O(1/sqrt(t)) for strong convexity. Polynomial decay or AdaGad are also widely used.
- Learning rates for different parameters should be different