[SKT AI Course: Deep Learning Basics]

Gentle Introduction to Feedforward Neural Networks

Heung-Il Suk

hisuk@korea.ac.kr

http://www.ku-milab.org

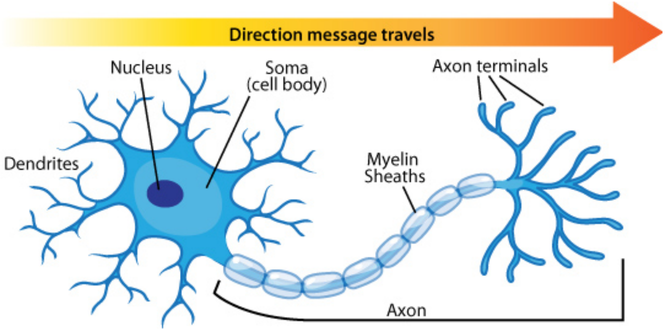
Department of Brain and Cognitive Engineering,
Korea University

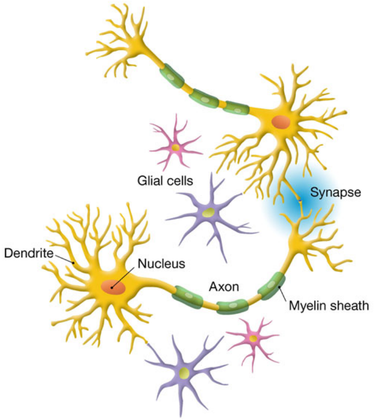
September 18-19, 2017

Contents

Towards an Artificial Neural Network

Neuron: building block of the nervous system





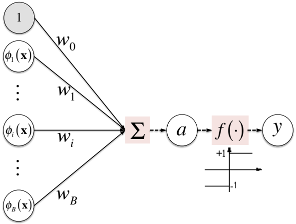
[Image source: https://askabiologist.asu.edu/neuron-anatomy]

Perceptron [Rosenblatt, 1958]

- Two-class model
  - An input vector $\mathbf{x}$ is first transformed using a fixed nonlinear transformation to give a feature vector $\phi(\mathbf{x})$
  - Then used to construct a generalized linear model

$$y(\mathbf{x}) = f\left(\mathbf{w}^\top \phi(\mathbf{x})\right)$$

$$\text{where } f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$



- Use a target coding scheme
  - $t = +1$ for class C_1, and $t = -1$ for C_2
  - Matching the choice of activation function

[Parameters Learning]

Error function minimization

- Error function: number of misclassifications
- This error function is a piecewise constant function of $\mathbf{w}$ with discontinuities (c.f., regression)
- No closed-form solution (no derivatives exist for non-smooth functions)
- Take an iterative approach

Perceptron Criterion

- Seeking $\mathbf{w}$ such that

$$\left\{ \begin{array}{l} \mathbf{x}_n \in C_1 \ (t_n = +1) \text{ will have } \mathbf{w}^\top \phi(\mathbf{x}_n) \geq 0 \\ \mathbf{x}_n \in C_2 \ (t_n = -1) \text{ will have } \mathbf{w}^\top \phi(\mathbf{x}_n) < 0 \end{array} \right\} \Rightarrow \mathbf{w}^\top \phi(\mathbf{x}_n) t_n \geq 0$$

  - Linearly bisecting the feature space

- For each misclassified sample, perceptron criterion tries to minimize

$$E_P(\mathbf{w}) = -\sum_{n \in M} \mathbf{w}^\top \phi(\mathbf{x}_n) t_n$$

M: a set of all misclassified samples

Machine Intelligence Lab

Perceptron Algorithm

- Error function: $E_P(\mathbf{w}) = -\sum_{n\in M} \mathbf{w}^\top \phi(\mathbf{x}_n) t_n$
- Batch/Stochastic Gradient descent

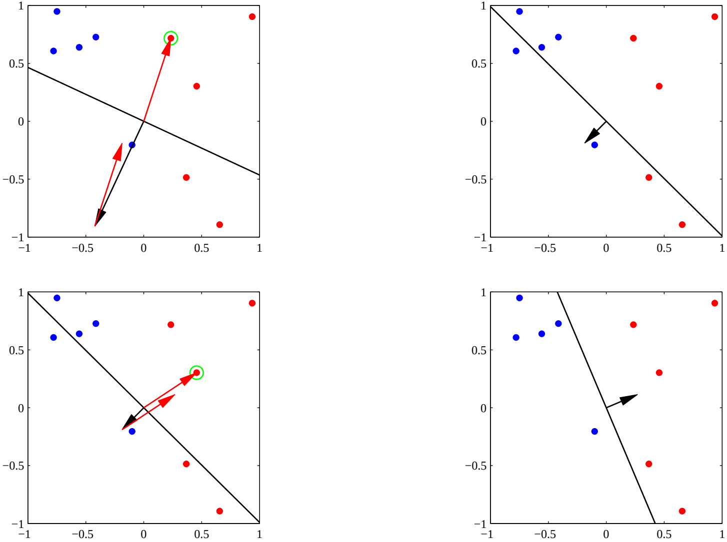
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \sum_n \phi(\mathbf{x}_n) t_n$$

η: learning rate, τ: step index

  - Since $y(\mathbf{x}, \mathbf{w})$ is unchanged if we multiply $\mathbf{w}$ by a constant, we can set η equal to 1 without loss of generality.

- Interpretation: cycle through the training samples in turn
  - If misclassified, for class C_1 add $\phi(\mathbf{x}_n)$ to $\mathbf{w}$
  - If misclassified, for class C_2 subtract $\phi(\mathbf{x}_n)$ from $\mathbf{w}$

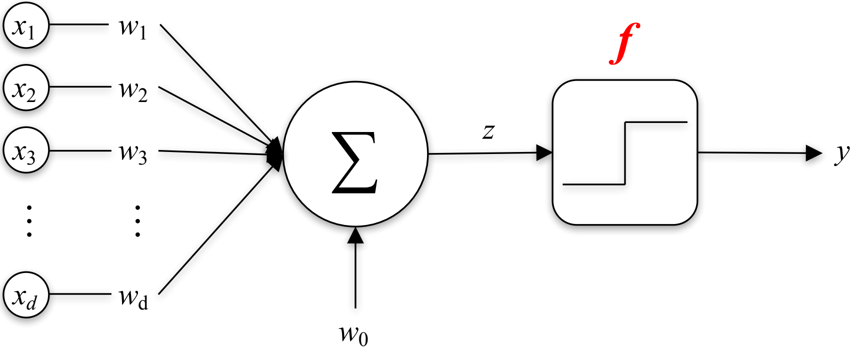
Machine Intelligence Lab



Black arrow: $\mathbf{w}$ (points towards the decision region of the red class), green point: misclassified

Changes in Non-linear Activation Function

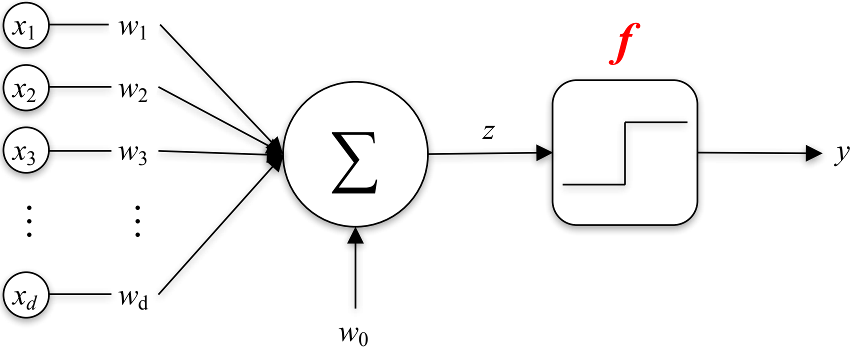
1960's



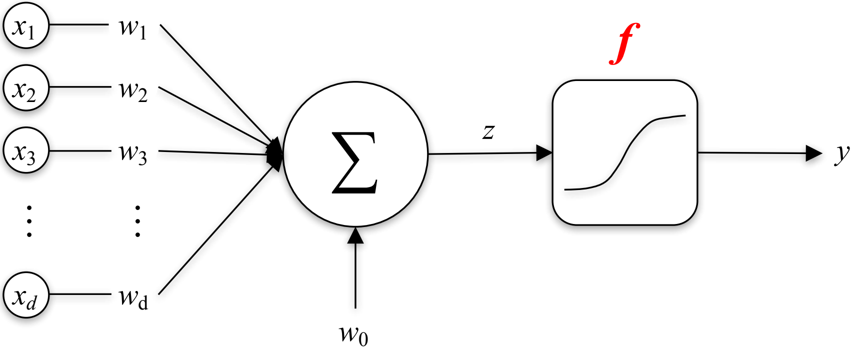
1980's

Hands on Programming: Perceptron Learning

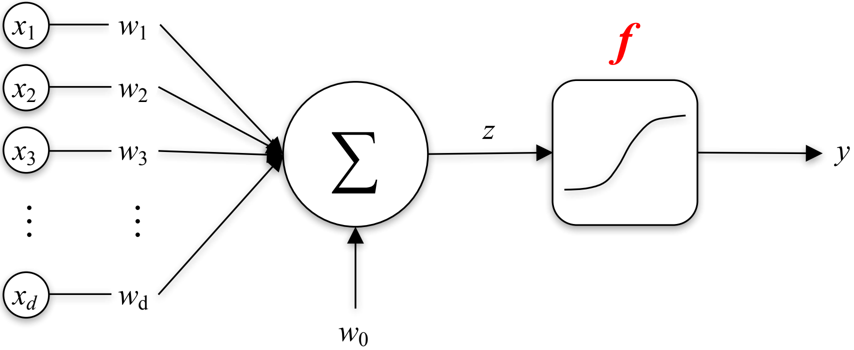
Rosenblatt's Perceptron with Vanilla Gradient



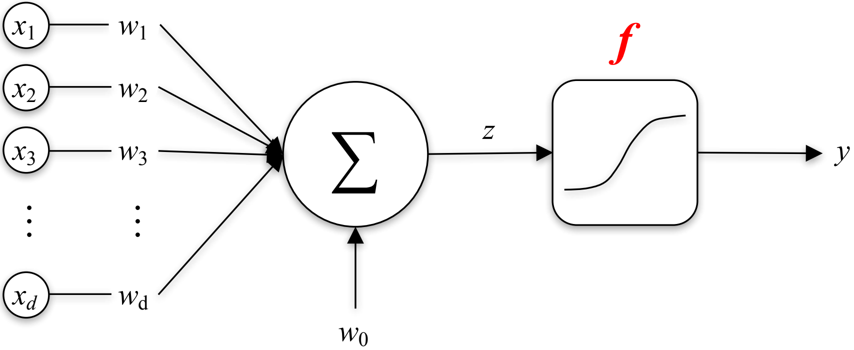
Widrow-Hoff's Algorithm with Vanilla Gradient



Widrow-Hoff's Algorithm with SDG



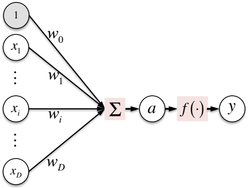
Widrow-Hoff Perceptron with Minibatch-SDG



Towards Feed-Forward Neural Networks

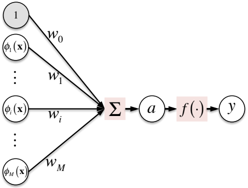
(Recap.) Linear Basis Function Models - Perceptron

$$y\left(\mathbf{x},\mathbf{w}\right) = f\left(\underbrace{\sum_{j=0}^{D} w_j x_j}_{a}\right)$$



- $f\left(\cdot\right)$: continuous output
  - (regression) identity function; (classification) sigmoidal function

$$y\left(\mathbf{x},\mathbf{w}\right) = f\left(\underbrace{\sum_{j=0}^{M} w_j \phi_j\left(\mathbf{x}\right)}_{a}\right)$$



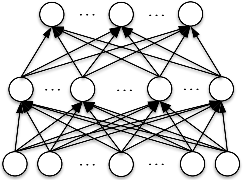
$\{\phi_j\}_{j=1}^{M}$: basis functions

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right)$$

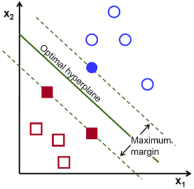
For application to large-scale problems, it is necessary *to adapt the basis functions to the data*

$$\Downarrow$$

$h(W\mathbf{x})$



Neural Networks



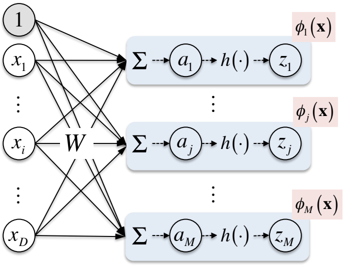
$K(\mathbf{x}, \mathbf{x}')$

Support Vector Machines

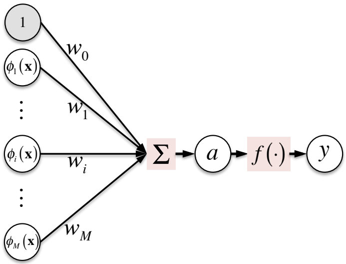
$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right)$$

- Basis functions $\phi_j(\mathbf{x})$: a parametric form
- These parameters to be adjusted along with the coefficients $\{w_j\}$

- In NN, each basis function itself is a nonlinear function of a linear combination of the inputs
  - Coefficients in the linear combination are adaptive parameters

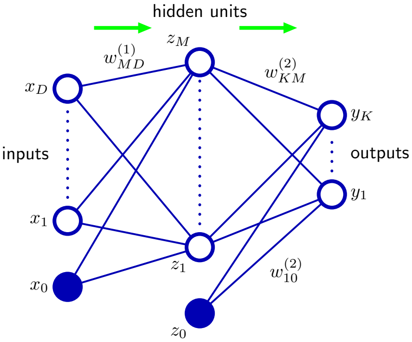


$\Rightarrow$



$h(\cdot)$: nonlinear function

Feed-Forward Network



$$\phi(\mathbf{x}) = h\left(W^{(1)}\mathbf{x}\right) = \mathbf{z}$$

$$W^{(1)} = \left[w_{ji}^{(1)}\right] \in \mathbb{R}^{M\times(D+1)}$$

$$\mathbf{y} = f\left(W^{(2)}\mathbf{z}\right)$$

$$W^{(2)} = \left[w_{kj}^{(2)}\right] \in \mathbb{R}^{K\times(M+1)}$$

Two-layer neural network

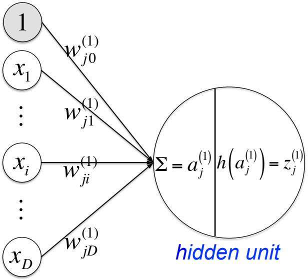
- Pre-activations: $\left\{a_j^{(1)}\right\}_{j=1}^{M}$

$$a_j^{(1)} = \sum_{i=1}^{D} \underbrace{w_{ji}^{(1)}}_{\text{weight}} x_i + \underbrace{w_{j0}^{(1)}}_{\text{bias}}$$

- Outputs of the basis functions ('*hidden units*')

$$z_j^{(1)} = h\left(a_j^{(1)}\right)$$

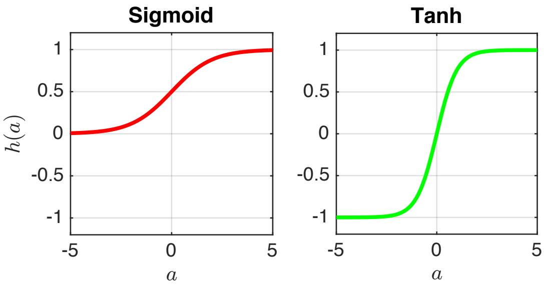
  - $h(\cdot)$: differentiable, nonlinear activation function
  - sigmoidal functions such as 'logistic sigmoid' or 'tanh'
  - or others (discussed in deep learning)



Connection between input and hidden units

$$\text{sigmoid}(a) = \frac{1}{1+\exp[-a]}$$

$$\tanh(a) = \frac{\exp[a]-\exp[-a]}{\exp[a]+\exp[-a]}$$



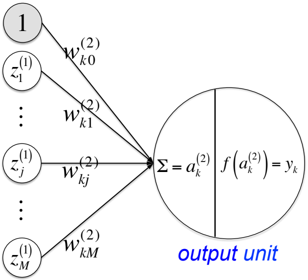
Sigmoid
Tanh
1
0.5
0
-0.5
-1
h(a)
-5
0
5
a

- Pre-activations: $\left\{a_k^{(2)}\right\}_{k=1}^{K}$

$$a_k^{(2)} = \sum_{j=1}^{M} \underbrace{w_{kj}^{(2)}}_{\text{weight}} z_j^{(1)} + \underbrace{w_{k0}^{(2)}}_{\text{bias}}$$

- Outputs of the basis functions ('*output units*')

$$y_k = f\left(a_k^{(2)}\right)$$



Connection between hidden and output units

- Choice of the output activation function $f(\cdot)$ is determined by the nature of the data and the assumed distribution of target variables
  - Regression: identity
$$y_k = a_k^{(2)}$$
  - Binary classification: logistic sigmoid
$$y_k = \mathsf{sigmoid}\left(a_k^{(2)}\right) = \frac{1}{1 + \exp\left(-a_k^{(2)}\right)}$$
  - Multiclass classification: softmax
$$y_k = \mathsf{softmax}\left(a_k^{(2)}\right) = \frac{\exp\left(a_k^{(2)}\right)}{\sum_{l=1}^{K} \exp\left(a_l^{(2)}\right)}$$

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

equivalently

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{M} w_{kj}^{(2)} \underbrace{h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)}_{\phi_j(\mathbf{x})}\right)$$

where $\mathbf{w} = \left[vec\left(W^{(1)}\right); vec\left(W^{(2)}\right)\right]$

Network Training

Overview

- Error function
- Parameter learning: (stochastic) gradient-descent method
- Gradient evaluation: backpropagation [Rumelhart *et al.*, 1986]

Error Functions

Given a set of input vectors $\{\mathbf{x}_n\}_{n=1}^N$ and target vectors $\{\mathbf{t}_n\}_{n=1}^N$

- Regression
  - Identity activation function
  - Sum-of-squares error

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\right\|^2$$

- Binary classification
  - Logistic sigmoid activation function
  - Cross-entropy error function

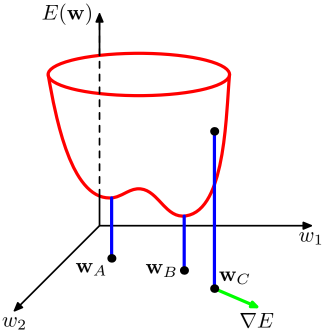
$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-t_n)\}$$

- Multiclass classification
  - Softmax function
  - Cross-entropy error function

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn}\ln y_{kn}$$

Parameter Optimization

- Finding a weight vector $\mathbf{w}$ that minimizes the error function $E(\mathbf{w})$

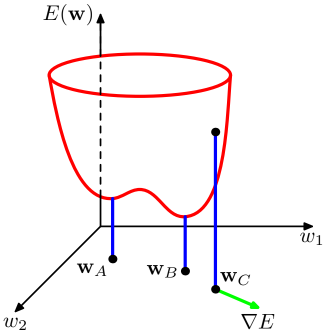


- Geometrical picture of error function
  - a surface sitting over the weight space
- Small step from $\mathbf{w}$ to $\mathbf{w} + \Delta\mathbf{w}$ leads to change in error function

$$\Delta E \approx \Delta\mathbf{w}^\top \nabla E(\mathbf{w})$$

  - At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector

$$\nabla E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \cdots, \frac{\partial E}{\partial w_d},\right]^\top$$



- ∇E: the direction of greatest rate of increase of the error function $E(\mathbf{w})$
- To reduce the error, make a small step in the direction of $-\nabla E(\mathbf{w})$
- Points at which the gradient vanishes: stationary points
  - minima, maxima, saddle points

- There is no analytical solution
- Resort to iterative numerical procedures
- Choose some initial value $\mathbf{w}^{(0)}$ and then update it

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$$

- Different algorithms involve different choices for $\Delta\mathbf{w}^{(\tau)}$.
- Weight vector update $\Delta\mathbf{w}^{(\tau)}$ is usually based on gradient $\nabla E(\mathbf{w})$ evaluated at the weight vector $\mathbf{w}^{(\tau)}$

(Recap.) Gradient Descent Optimization

- Simplest approach to using gradient information
- Take a small step in the direction of the negative gradient

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E\left(\mathbf{w}^{(\tau)}\right)$$

  - η: learning rate (*e.g.*, 0.001)

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

Different versions of gradient descent optimization

- Batch: the entire training set to be processes to evaluate ∇E (a.k.a., *gradient descent* or *steepest descent*)

- On-line: one sample at a time (a.k.a., *stochastic gradient descent* or *sequential gradient descent*)
  - Possibility of escaping from local minima
    - Stationary point w.r.t. the error function for the whole data set will generally not be a stationary point for each data point individually

- Mini-batch stochastic gradient descent: a small set of samples at a time
  - good tradeoff between batch and on-line
  - approximation of the gradient of the loss function

Gradient Evaluation

To find an efficient technique for evaluating the gradient of an error function $E(\mathbf{w})$ for a feed-forward neural network

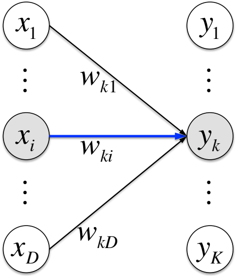
- *Local message passing* scheme
- alternative information passing: forwards and backwards through the network

'Error backpropagation' or simply 'Backprop'

Simple Linear Model

$$
\begin{aligned}
y_k &= \sum_i w_{ki} x_i \\
y_{nk} &= y_k(\mathbf{x}_n, \mathbf{w}) \\
E_n &= \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2
\end{aligned}
$$

$$
\frac{\partial E_n}{\partial w_{ki}} = \underbrace{(y_{nk} - t_{nk})}_{\text{error}} x_{ni}
$$



- 'Local' computation
  - an 'error signal' $(y_{nk} - t_{nk})$ associated with the output end of the link w_{ki}
  - the variable x_{ni} associated with the input end of the link

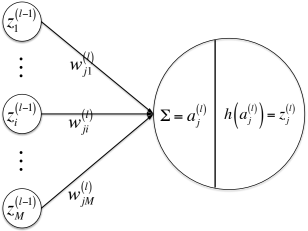
Forward Propagation in a General Feed-Forward Network

1. Each unit computes a weighted sum of its inputs

$$a_j^{(l)} = \sum_i w_{ji}^{(l)} z_i^{(l-1)}$$

2. Transformation by a nonlinear activation function $h(\cdot)$ to give the activation $z_j^{(l)}$ of unit j

$$z_j^{(l)} = h\left(a_j^{(l)}\right)$$



What is the error in this unit?

Evaluation of the derivative of E_n w.r.t. $w_{ji}^{(l)}$

$$\frac{\partial E_n}{\partial w_{ji}^{(l)}} = \underbrace{\frac{\partial E_n}{\partial a_j^{(l)}}}_{\equiv \delta_j^{(l)}} \underbrace{\frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}}}_{z_i^{(l-1)}} \qquad \text{(by chain rule)}$$

$$= \delta_j^{(l)} z_i^{(l-1)}$$

$\delta_j^{(l)}$: gradient of the error function w.r.t. the pre-activation $a_j^{(l)}$

- Note that this takes the same form as for the simple linear model
- Need only to calculate the value of $\delta_j^{(l)}$ for each hidden and output unit in the network

- For hidden unit j,

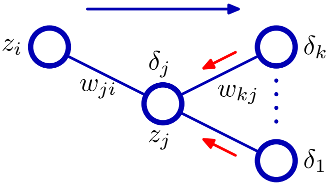
$$\delta_j^{(l)} = \frac{\partial E_n}{\partial a_j^{(l)}} = \sum_k \underbrace{\frac{\partial E_n}{\partial a_k^{(l+1)}}}_{\equiv \delta_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}}$$

  - We are making use of the fact that variations in $a_j^{(l)}$ give rise to variations in the error function only through variations in the variable $a_k^{(l+1)}$

$$\frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} = \frac{\partial \left(\sum_m w_{km}^{(l+1)} z_m^{(l)}\right)}{\partial a_j^{(l)}} = \frac{\partial \left(\sum_m w_{km}^{(l+1)} h\left(a_m^{(l)}\right)\right)}{\partial a_j^{(l)}} = h'\left(a_j^{(l)}\right) w_{kj}^{(l+1)}$$

$$\text{or } \frac{\partial a_k^{(l+1)}}{\partial a_j^{(l)}} = \frac{\partial a_k^{(l+1)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial a_j^{(l)}} = w_{kj}^{(l+1)} h'\left(a_j^{(l)}\right)$$

$$\delta_j^{(l)} = h'\left(a_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$$



Blue arrow: information flow during forward propagation
Red arrow: backward propagation of error information

The value of δ for a particular hidden unit can be obtained
by propagating the δ's backwards from units higher up in the network

- Forward propagation

$$a_j^{(l)} = \sum_i w_{ji}^{(l)} z_i^{(l-1)}$$

$$z_j^{(l)} = h\left(a_j^{(l)}\right)$$

- Backward propagation

$$\delta_j^{(l)} = h'\left(a_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

Forward propagation

$$\mathbf{x} \xrightarrow{W^{(1)}} \mathbf{a}^{(1)} \to \mathbf{z}^{(1)} \xrightarrow{W^{(2)}} \mathbf{a}^{(2)} \to \mathbf{z}^{(2)} = \mathbf{y}$$

Backward propagation

$$\begin{tikzcd}
\dfrac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{x}} & \overset{\boldsymbol{\delta}^{(1)}}{\dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} \odot \dfrac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(1)}}} \arrow[l] \arrow[d] & \overset{\boldsymbol{\delta}^{(2)}}{\dfrac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(2)}} \odot \dfrac{\partial E}{\partial \mathbf{z}^{(2)}}} \arrow[l] \arrow[d] \\
& \dfrac{\partial \mathbf{a}^{(1)}}{\partial W^{(1)}} & \dfrac{\partial \mathbf{a}^{(2)}}{\partial W^{(2)}}
\end{tikzcd}$$

$$\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} = h'\left(\mathbf{a}^{(1)}\right)$$

$$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(1)}} = W^{(1)}$$

$$\frac{\partial \mathbf{a}^{(1)}}{\partial W^{(1)}} = \mathbf{z}^{(0)} = \mathbf{x}$$

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(2)}} = h'\left(\mathbf{a}^{(2)}\right)$$

$$\frac{\partial \mathbf{a}^{(2)}}{\partial W^{(2)}} = \mathbf{z}^{(1)}$$

$$\frac{\partial E}{\partial \mathbf{z}^{(2)}} \equiv \boldsymbol{\delta}^{(2)} = f'\left(\mathbf{a}^{(2)}\right) \odot (\mathbf{y} - \mathbf{t})$$

$$f'(\mathbf{a}) = \begin{cases} 1 & \text{identity} \\ f(\mathbf{a})\,(1 - f(\mathbf{a})) & \text{logistic sigmoid} \\ 1 - f(\mathbf{a})^2 & \text{tanh} \\ \vdots & \vdots \end{cases}$$

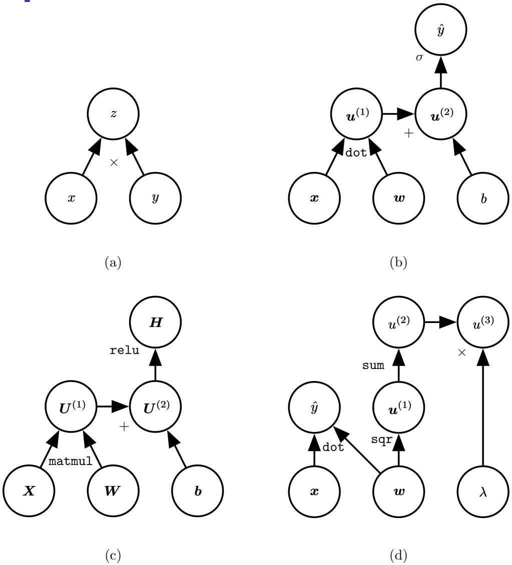
Forward propagation

$$\mathbf{x} \xrightarrow{W^{(1)}} \mathbf{a}^{(1)} \to \mathbf{z}^{(1)} \xrightarrow{W^{(2)}} \cdots \xrightarrow{W^{(L-1)}} \mathbf{a}^{(L-1)} \to \mathbf{z}^{(L-1)} \xrightarrow{W^{(L)}} \mathbf{a}^{(L)} \to \mathbf{z}^{(L)} = \mathbf{y}$$

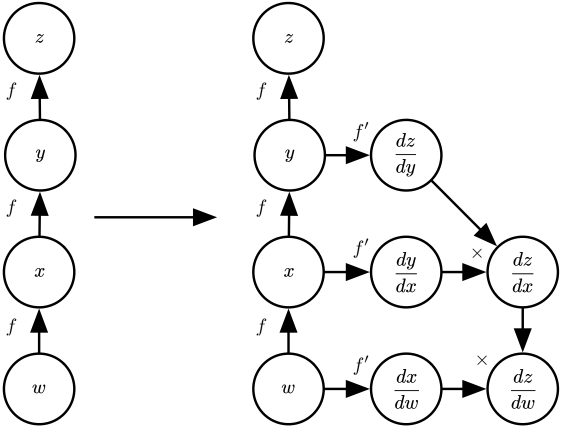
Backward propagation

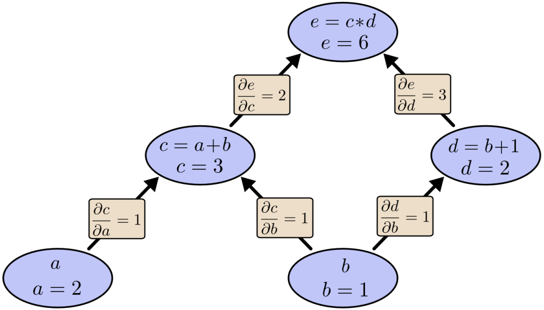
$$\begin{tikzcd}
\dfrac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{x}} & \overbrace{\dfrac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{a}^{(1)}} \odot \dfrac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(1)}}}^{\boldsymbol{\delta}^{(1)}} \arrow[l] \arrow[d] & \cdots \arrow[l] & \overbrace{\dfrac{\partial \mathbf{z}^{(L-1)}}{\partial \mathbf{a}^{(L-1)}} \odot \dfrac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L-1)}}}^{\boldsymbol{\delta}^{(L-1)}} \arrow[l] \arrow[d] & \overbrace{\dfrac{\partial \mathbf{z}^{(L)}}{\partial \mathbf{a}^{(L)}} \odot \dfrac{\partial E}{\partial \mathbf{z}^{(L)}}}^{\boldsymbol{\delta}^{(L)}} \arrow[l] \arrow[d] \\
 & \dfrac{\partial \mathbf{a}^{(1)}}{\partial W^{(1)}} & \cdots & \dfrac{\partial \mathbf{a}^{(L-1)}}{\partial W^{(L-1)}} & \dfrac{\partial \mathbf{a}^{(L)}}{\partial W^{(L)}}
\end{tikzcd}$$

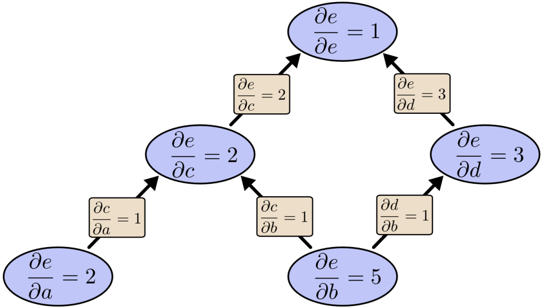
Computational Graph

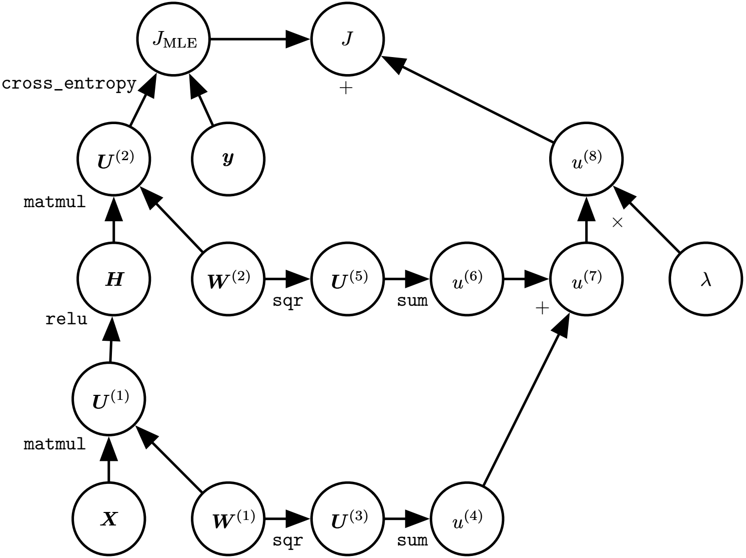


$$
\begin{aligned}
&\frac{\partial z}{\partial w} \\
=&\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \\
=&f'(y)f'(x)f'(w) \\
=&f'(f(f(w)))f'(f(w))f'(w)
\end{aligned}
$$




$e = c*d$
$e = 6$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$c = a+b$
$c = 3$
$d = b+1$
$d = 2$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
a
$a = 2$
b
$b = 1$


$\frac{\partial e}{\partial e} = 1$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
$\frac{\partial e}{\partial a} = 2$
$\frac{\partial e}{\partial b} = 5$


J_{MLE}
J
$+$
cross_entropy
$\boldsymbol{U}^{(2)}$
$\boldsymbol{y}$
$u^{(8)}$
matmul
$\times$
$\boldsymbol{H}$
$\boldsymbol{W}^{(2)}$
sqr
$\boldsymbol{U}^{(5)}$
sum
$u^{(6)}$
$u^{(7)}$
$+$
λ
relu
$\boldsymbol{U}^{(1)}$
matmul
$\boldsymbol{X}$
$\boldsymbol{W}^{(1)}$
sqr
$\boldsymbol{U}^{(3)}$
sum
$u^{(4)}$

Hands on Programming: Neural Networks

Thank you for your attention!!!

(Q & A)

hisuk (AT) korea.ac.kr

http://www.ku-milab.org

Machine
Intelligence
Lab