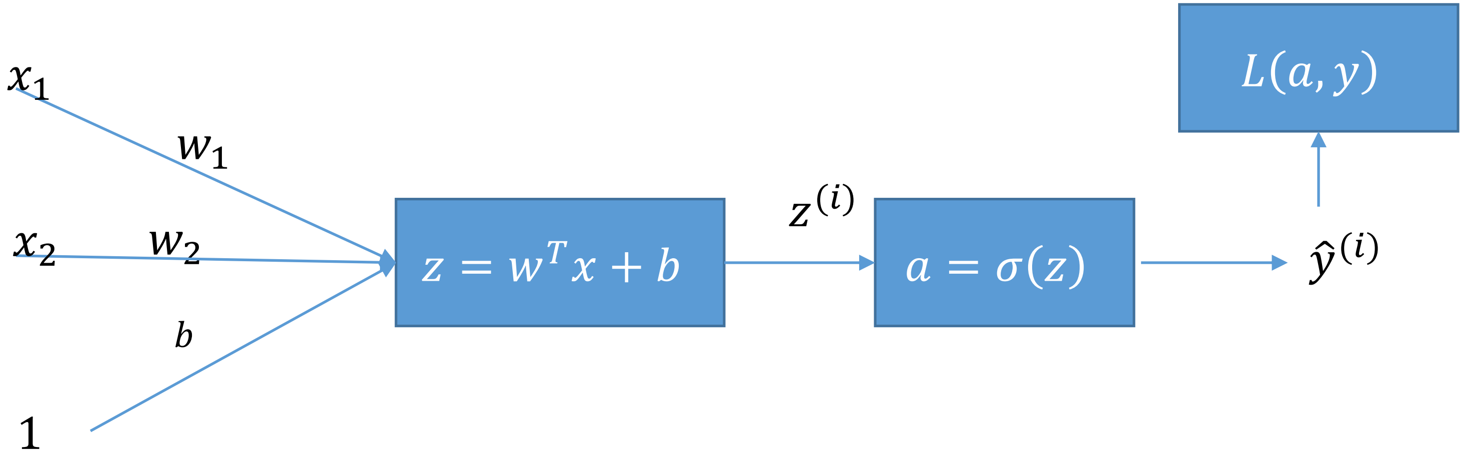
Machine Learning Practice

Shallow NNs

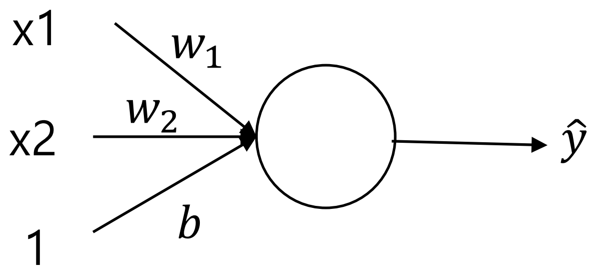
References

- Andrew Ng's ML class
  - https://class.coursera.org/ml-003/lecture
  - http://www.holehouse.org/mlclass/ (note)
- Convolutional Neural Networks for Visual Recognition.
  - http://cs231n.github.io/
- Tensorflow
  - https://www.tensorflow.org
  - https://github.com/aymericdamien/TensorFlow-Examples
- 모두의 머신러닝
- Wikipedia
- Neural Network and Deep Learning, Michael Nielsen,
  - http://neuralnetworksanddepplearning.com

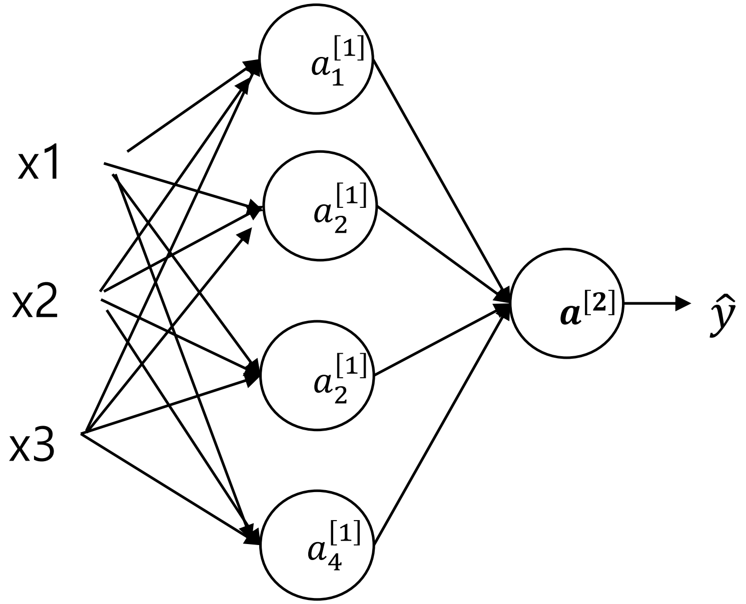
A neuron and a hyperplane



can be simplified to



Neural Networks with a hidden layer

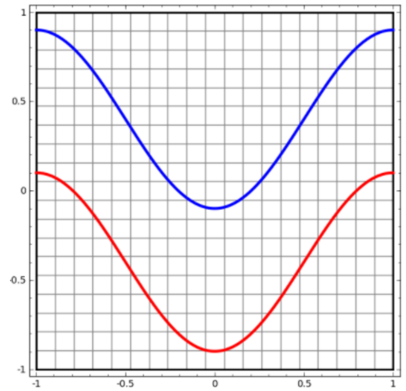


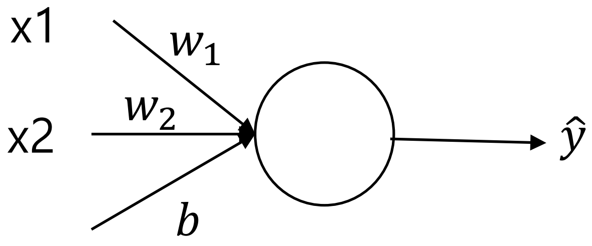
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$dW^{[1]} = \cdots$$
$$db^{[1]} = \cdots$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$
$$dW^{[2]} = \cdots$$
$$db^{[2]} = \cdots$$

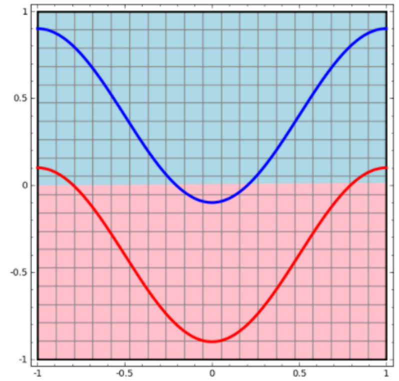
Why we need hidden layers?

- Input





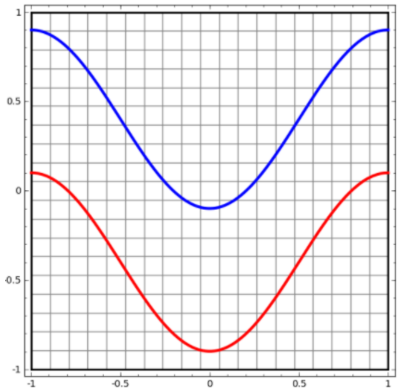
- Output

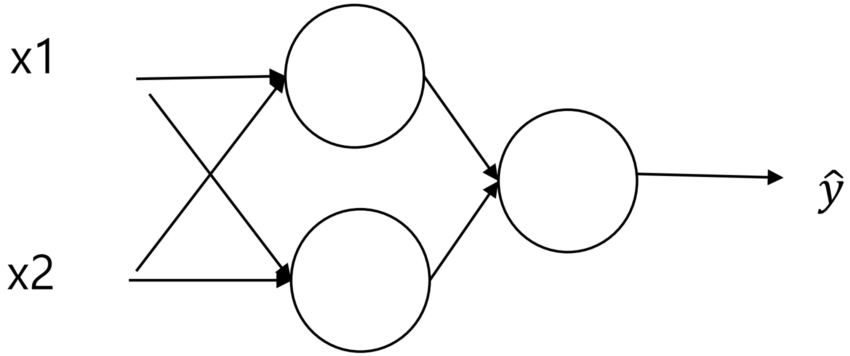


http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

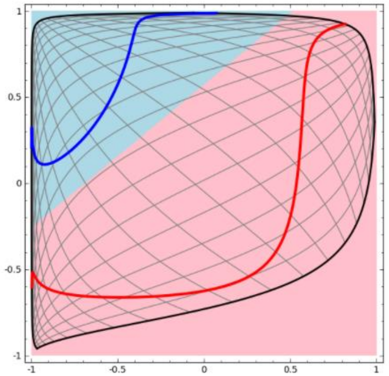
Why we need hidden layers?

- Input





- Output

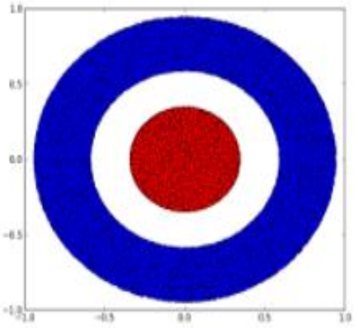


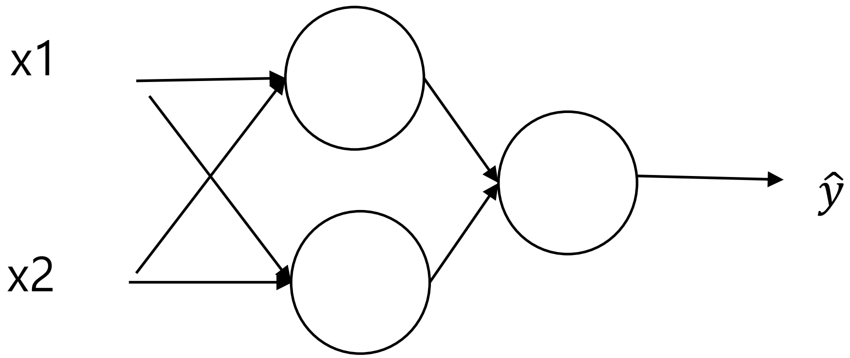
Each layer changes data representation by using
1. A linear transformation by the "weight" matrix W
2. A translation by the vector b
3. Point-wise application of activation function (nonlinear)

http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

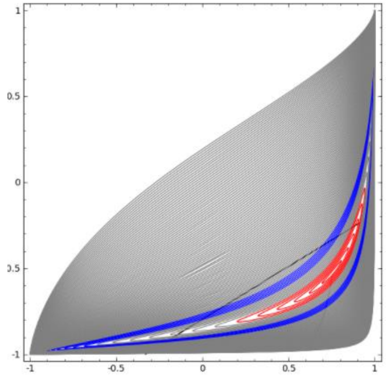
Why we need hidden layers?

- Input





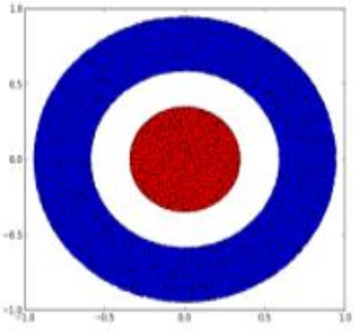
- Output



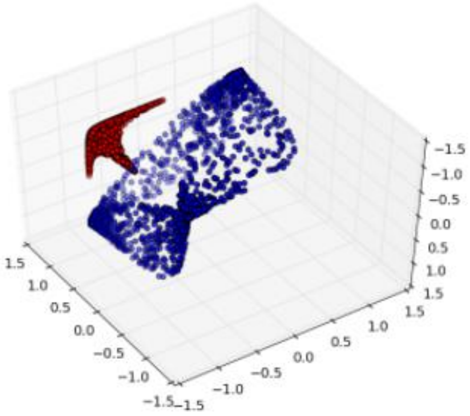
The network is topologically incapable of separating the data!

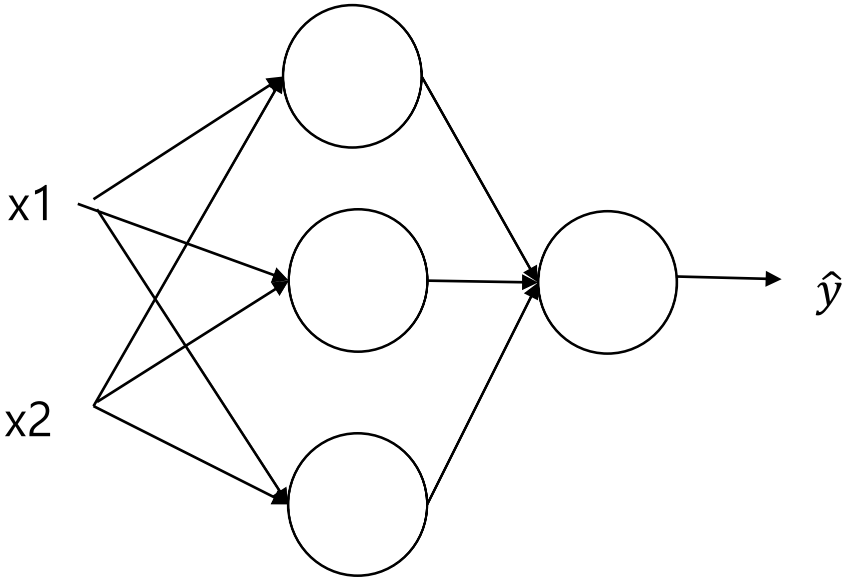
Why we need hidden layers?

- Input



- Output

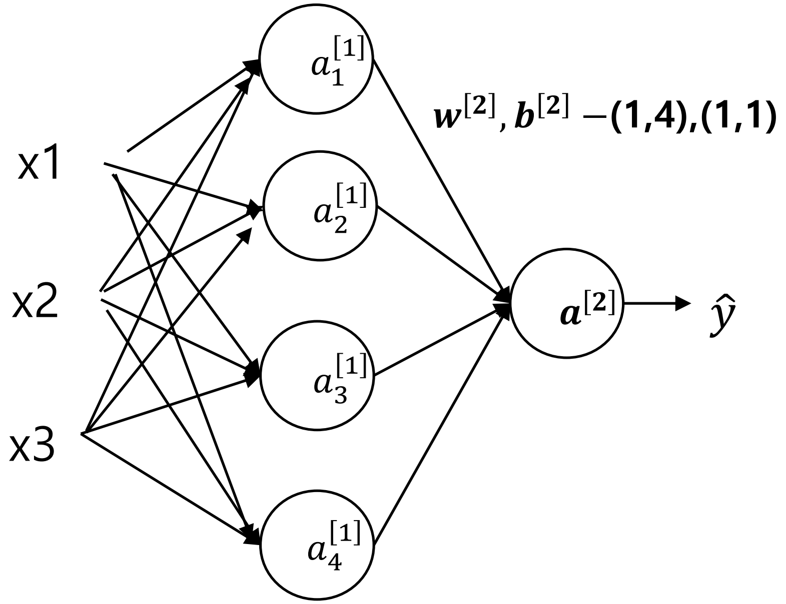




The neural network learns the new representation and it thus separate the datasets with a hyperplane.

NN Representation

$\boldsymbol{w}^{[1]}, \boldsymbol{b}^{[1]}\ - \boldsymbol{dim}: (\mathbf{4}, \mathbf{3}), (\mathbf{4}, \mathbf{1})$



input layer **hidden layer** **output layer**

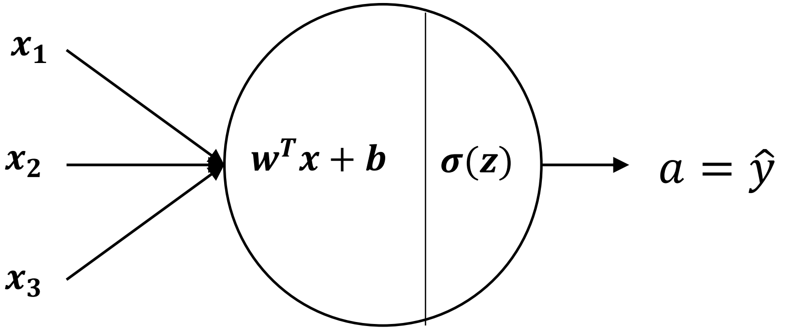
$\boldsymbol{a}^{[0]} = \boldsymbol{X}$

$$\boldsymbol{a}^{[1]} = \begin{bmatrix} \boldsymbol{a}_1^{[1]} \\ \boldsymbol{a}_2^{[1]} \\ \boldsymbol{a}_3^{[1]} \\ \boldsymbol{a}_4^{[1]} \end{bmatrix}$$

$\hat{y} = a^{[2]}$

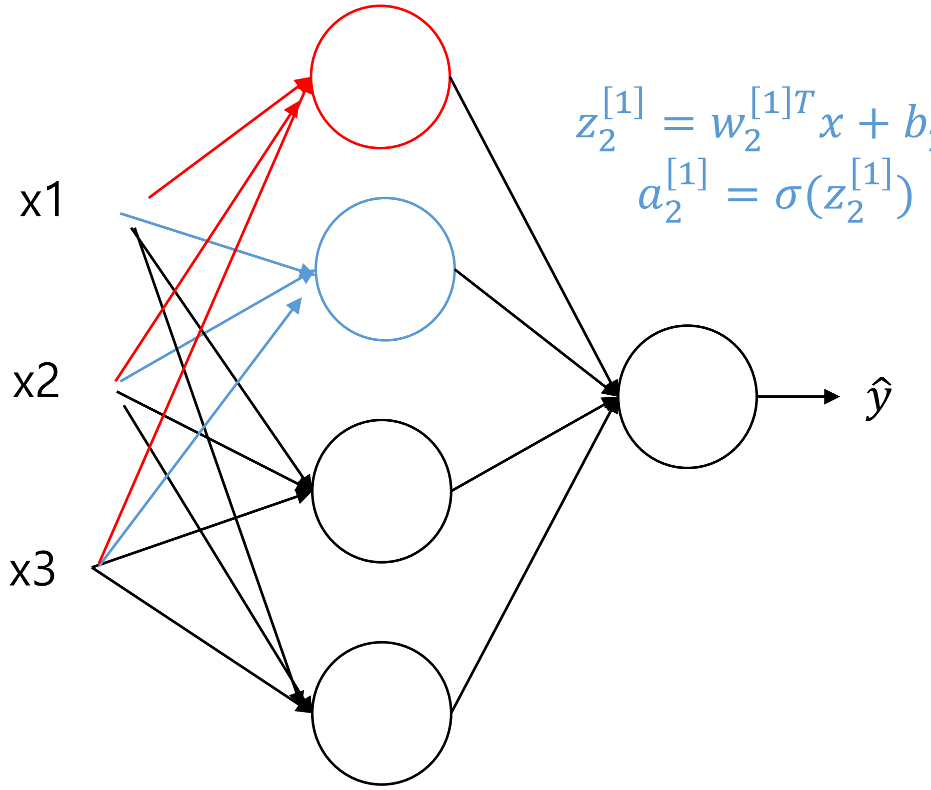
* In NN context, this is called 2-layer NN. The input layer is not counted.

NN Representation



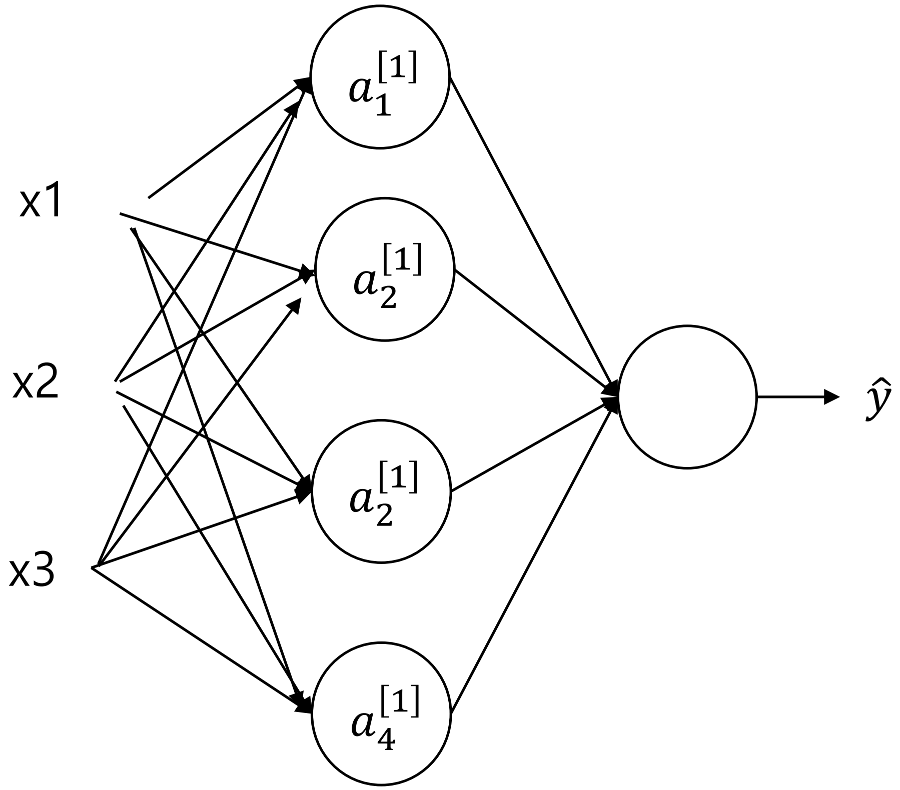
$$z = w^T x + b$$
$$a = \sigma(z)$$

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$
$$a_1^{[1]} = \sigma(z_1^{[1]})$$



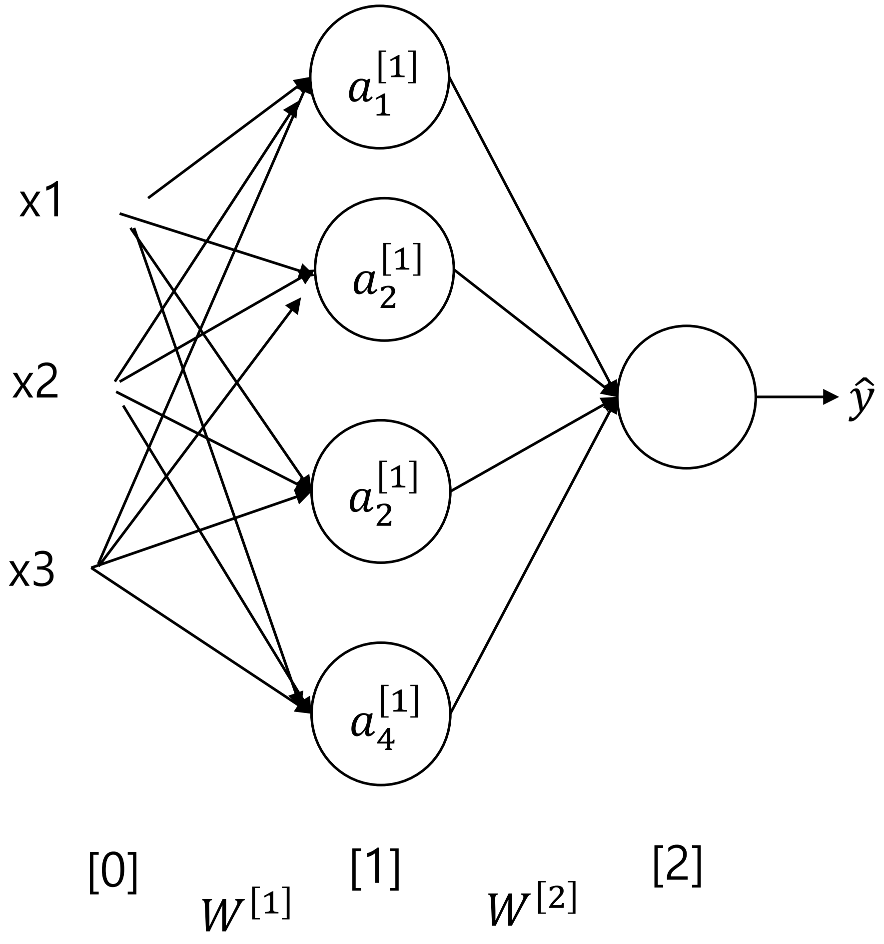
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$
$$a_2^{[1]} = \sigma(z_2^{[1]})$$

NN Representation



- $z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}$, $a_1^{[1]} = \sigma(z_1^{[1]})$
- $z_2^{[1]} = w_2^{[1]T}x + b_2^{[1]}$, $a_2^{[1]} = \sigma(z_2^{[1]})$
- $z_3^{[1]} = w_3^{[1]T}x + b_3^{[1]}$, $a_3^{[1]} = \sigma(z_3^{[1]})$
- $z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}$, $a_4^{[1]} = \sigma(z_4^{[1]})$
- $z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ \cdots & w_3^{[1]T} & \cdots \\ \cdots & w_4^{[1]T} & \cdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = W^{[1]} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \mathrm{b}^{[1]}$
- $a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$

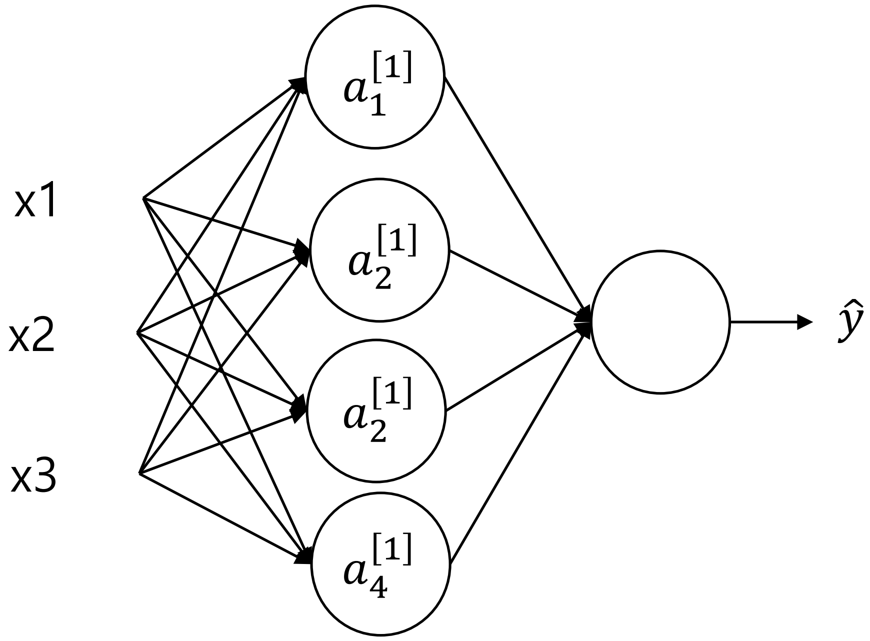
NN Representation



- Given input x
  - $z^{[1]} = W^{[1]} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \mathrm{b}^{[1]}$ Dim: (4,1)=(4,3)(3,1)+(4,1)
  - $a^{[1]} = \sigma(z^{[1]})$ Dim: (4,1)=(4,1)
  - $z^{[2]} = W^{[2]} a^{[1]} + \mathrm{b}^{[2]}$ Dim: (1,1)=(1,4)(4,1)+(1,1)
  - $a^{[2]} = \sigma(z^{[2]})$ Dim: (1,1)= (1,1)

- Expression convention
  - $z = W^T x + b$
  - $\hat{y} = a = \sigma(z)$

Vectorizing across multiple examples



- Consider
  - $z^{[1]} = W^{[1]}x + \mathrm{b}^{[1]}$
  - $a^{[1]} = \sigma(z^{[1]})$
  - $z^{[2]} = W^{[2]}a^{[1]} + \mathrm{b}^{[2]}$
  - $a^{[2]} = \sigma(z^{[2]})$
- Assume m training examples

$x \longrightarrow a^{[2]} = \hat{y}$

$x^{(1)}$ $\quad a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)}$ $\quad a^{[2](2)} = \hat{y}^{(2)}$

$x^{(m)}$ $\quad a^{[2](m)} = \hat{y}^{(m)}$

for i=1 to m:

$$\mathrm{z}^{[1](\mathrm{i})} = W^{[1]}x^{(i)} + b^{[1]}$$
$$\mathrm{a}^{[1](\mathrm{i})} = \sigma\left(z^{[1](i)}\right)$$
$$\mathrm{z}^{[2](\mathrm{i})} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$\mathrm{a}^{[2](\mathrm{i})} = \sigma(z^{[2](i)})$$

Vectorizing across multiple examples

```
for i=1 to m:
```
$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma\left(z^{[1](i)}\right)$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

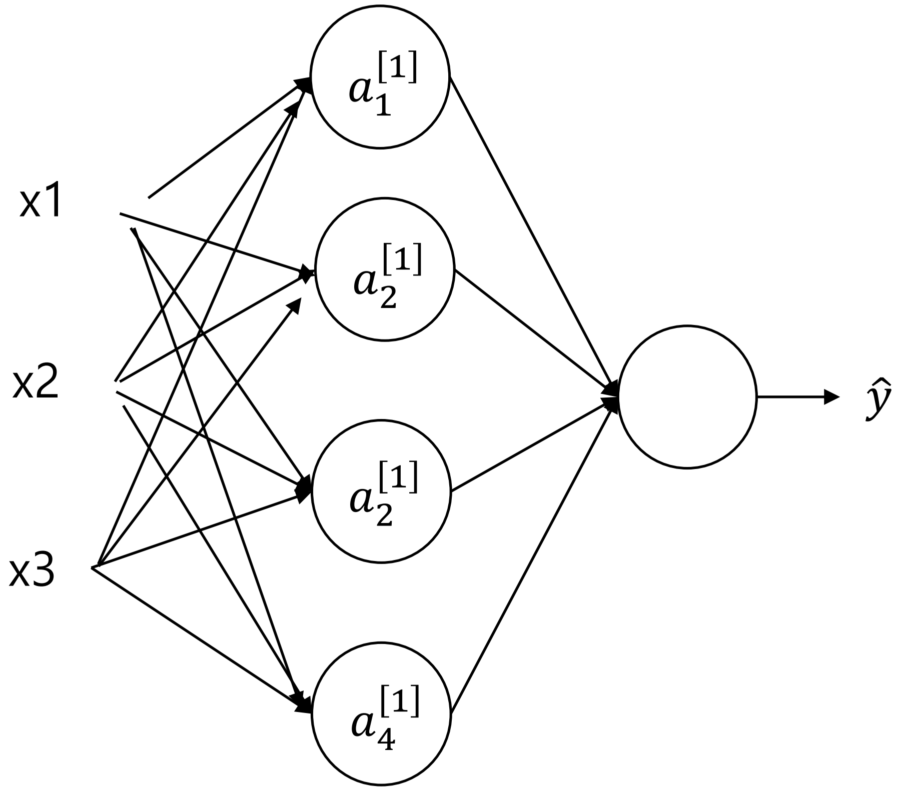
- Remember
  - $X = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & \vdots & x^{(m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$, x.shape=(n_x, m)
- Then, vetorized equations are built as:
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = \sigma(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = \sigma(Z^{[2]})$

Dim: (#node,m)=(#node,n_x)(n_x,m)+(#node,m)

Justification for vectorized implementation

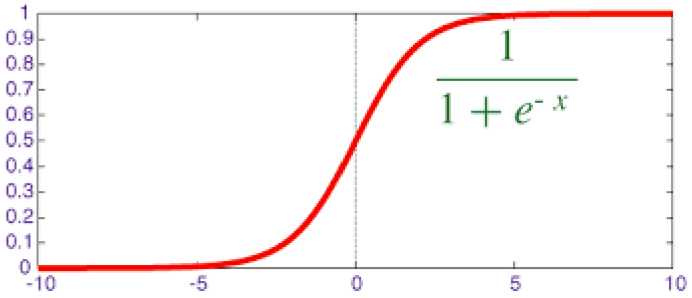
- $z^{[1](1)} = W^{[1]}x^{(1)} + b^{[1]}$, $z^{[1](2)} = W^{[1]}x^{(2)} + b^{[1]}$, $z^{[1](3)} = W^{[1]}x^{(3)} + b^{[1]}$
- To simply the justification, assume that $b^{[0]} = b^{[1]} = b^{[2]} = 0$
- $W^{[1]} = \begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$, $X = \begin{bmatrix} \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} \\ \vdots & \vdots & \vdots \end{bmatrix}$
- *Thus,*
- $W^{[1]}X = \begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ \vdots & \vdots & \vdots \end{bmatrix} = Z^{[1]}$

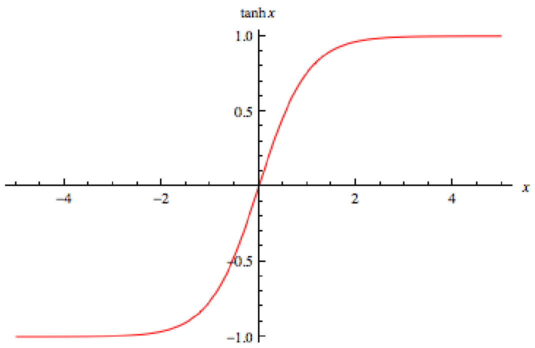
Recap



- Then, vetorized equations are built as:
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = \sigma(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = \sigma(Z^{[2]})$
  - where
    - $X = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & \vdots & x^{(m)} \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$, x.shape=(n_x, m)
    - $W^{[1]} = \begin{bmatrix} \cdots & w_1^{[1]T} & \cdots \\ \cdots & w_2^{[1]T} & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$
    - $A^{[1]} = \begin{bmatrix} \vdots & \vdots & \vdots \\ a^{[1](1)} & a^{[1](2)} & a^{[1](m)} \\ \vdots & \vdots & \vdots \end{bmatrix}$

Activation functions





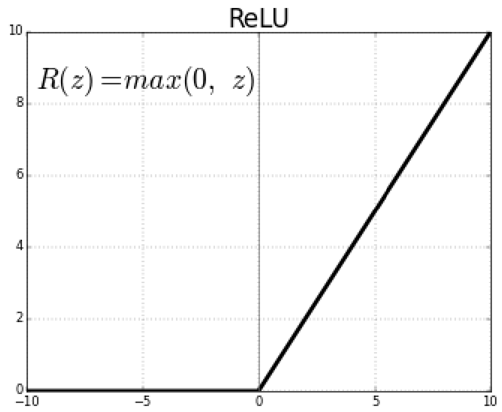
$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

- Activation function g()
    - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
    - $A^{[1]} = g(Z^{[1]})$
    - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
    - $A^{[2]} = g(Z^{[2]})$

- Another activation function : tanh(z)
    - → shifted sigmoid function
    - Generally, it works better than the sigmoid function.
    - Exception : output layer.
    - E.g., In 3 layer NN, the hidden layer uses tanh(z) and the output layer uses sigmoid functions.
    - When abs(z) is large, the derivative of tanh(z) (and sigmoid(z)) almost becomes zero, which slows down GD.

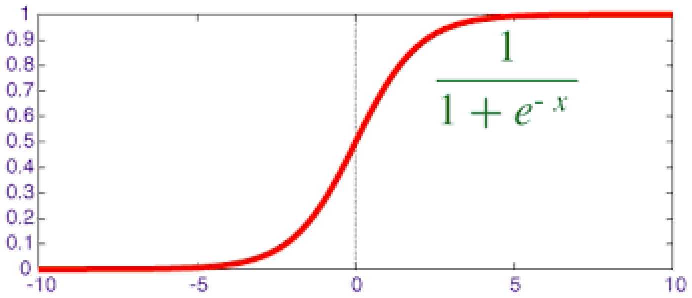
Activation functions

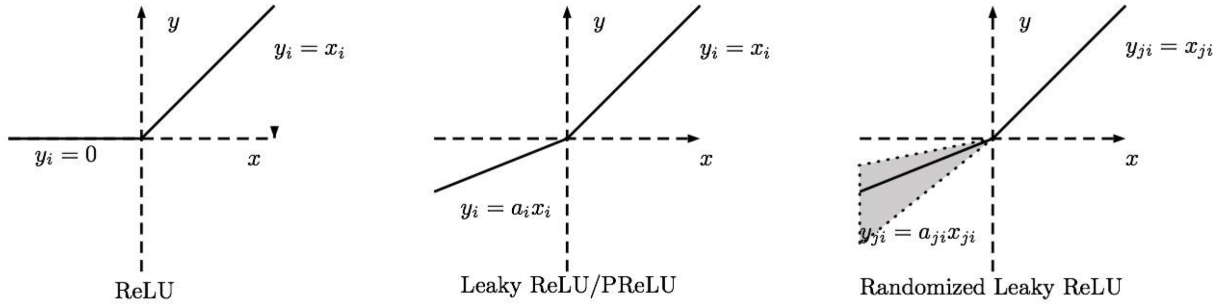
- Activation function g()
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g(Z^{[2]})$
- Another activation function : ReLU(z)
  - When z is 0, derivative ReUL(z) is not well defined. (do not worry about it in practice)
  - Strength : When z is greater than 0, derivatives becomes 1→ fast learning
  - Weakness: If z is less than 0, derivative becomes 0. → leaky ReLU
- How to choose the activation function?
  - If the output is binary, sigmoid function is a good choice.
  - Otherwise, ReLU increasingly becomes a default choice



Recap

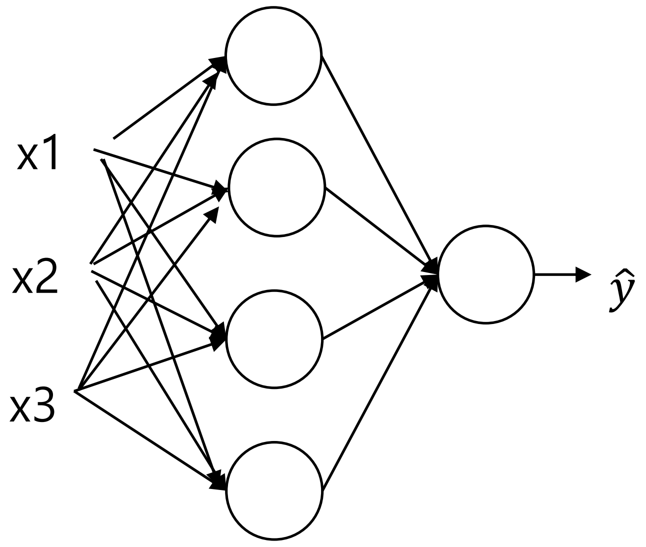
- Don't use sigmoid fn. Except the output layer that generates binary outputs.
- tanh(z) generally outperforms sigmoid()
- The most commonly used activation functions is ReLU()
- Try leakly ReLUs





https://datascience.stackexchange.com/questions/14349/difference-of-activation-functions-in-neural-networks-in-general

Why do we need non-linear activation functions?

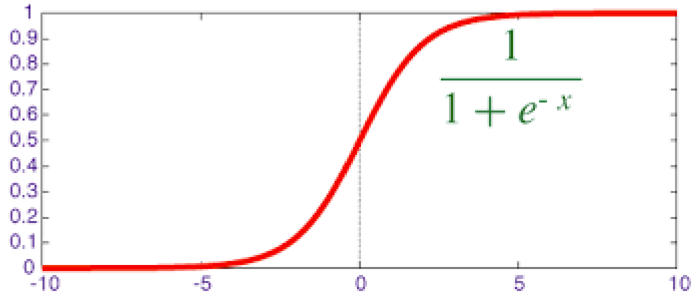


- Given x:
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$ (vs. $A^{[1]} = Z^{[1]}$)
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g(Z^{[2]})$ (vs. $A^{[2]} = Z^{[2]}$)

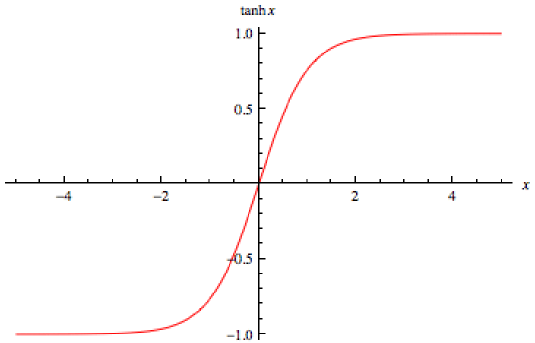
- IF we use linear activation function $g(z) = z$,
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]} = W^{[2]}\left(W^{[1]}X + \mathrm{b}^{[1]}\right) + \mathrm{b}^{[2]} = W^{[2]}W^{[1]}X + W^{[2]}\mathrm{b}^{[1]} + \mathrm{b}^{[2]}$
  - The output of NN becomes a linear function of input x.→ No matter how many layers we use, the entire NN becomes a linear function input x.

Derivatives of activation functions

- Sigmoid function
  - $\frac{d}{dz}g(z) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1 - g(z))$
  - If g(z) is large, then it becomes 0
  - If -g(z) is large, then it becomes 0.
  - If z=0, then the derivative becomes 1/4 .



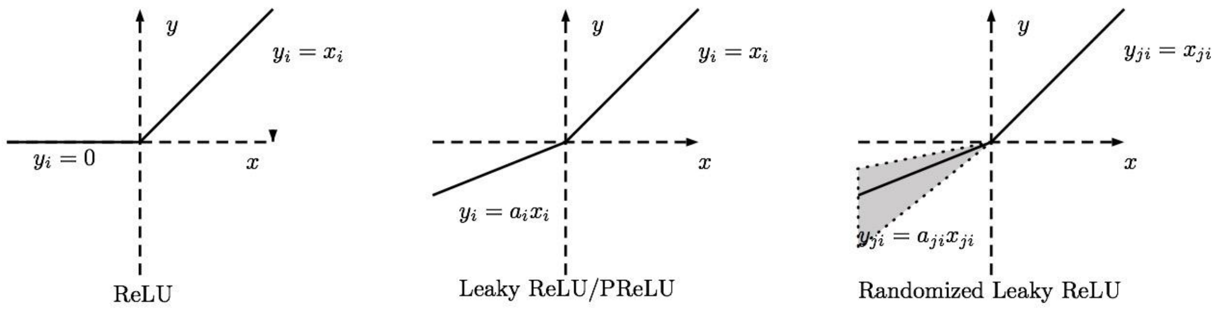
- Tanh(z)
  - $\frac{d}{dz}g(z) = 1 - (\tanh(z))^2$
  - If g(z) is large, then it becomes 0
  - If -g(z) is large, then it becomes 0.
  - If z=0, then the derivative becomes 1 .



$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Derivatives of activation functions

- ReLU function
  - $\frac{d}{dz}g(z) = \begin{cases} 0 \; if \; z < 0 \\ 1 \; if \; z \geq 0 \end{cases}$
- Leaky ReLU function
  - $\frac{d}{dz}g(z) = \begin{cases} a \; if \; z < 0 \\ 1 \; if \; z \geq 0 \end{cases}$



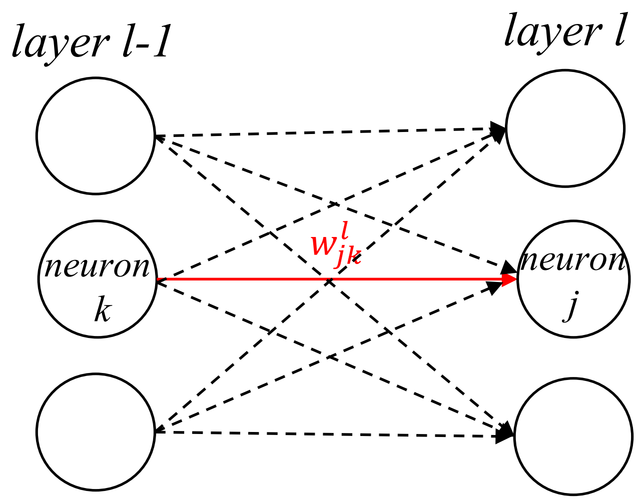
https://datascience.stackexchange.com/questions/14349/difference-of-activation-functions-in-neural-networks-in-general

Back propagation

- Objectives : We need to obtain $\frac{dJ}{dw^{[1]}}, \frac{dJ}{db^{[1]}}, \frac{dJ}{dw^{[2]}}, \frac{dJ}{db^{[2]}}$

- Strategy :
  - We compute the partial derivatives ∂L/∂w for a single training example. We then recover ∂J/∂w by averaging over training examples.
  - We first look at a generic form that considers multiple nodes in multiple layers. We then turn it into our model (3 layer model for logistic regression)

Generic form for data (i)



Element-wise

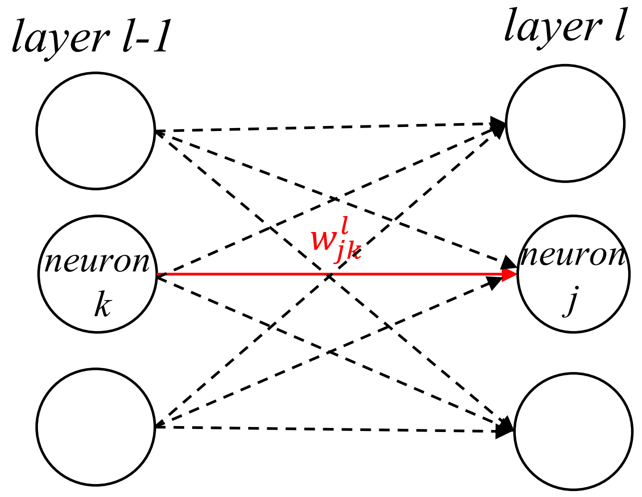
$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma\left({w_j^l}^T a^{l-1} + b_j^l\right)$$

Vectorize for a layer l

$$\begin{bmatrix} a_1^l \\ a_2^l \\ \cdots \\ a_j^l \end{bmatrix} = \sigma\left(\begin{bmatrix} {w_1^l}^T a^{l-1} + b_1^l \\ {w_2^l}^T a^{l-1} + b_2^l \\ \cdots \\ {w_j^l}^T a^{l-1} + b_j^l \end{bmatrix}\right)$$

$$a^l = \sigma(W^l a^{l-1} + b^l)$$

Generic form for data (i)



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma\left({w_j^l}^T a^{l-1} + b_j^l\right)$$

$$a^l = \sigma(W^l a^{l-1} + b^l)$$

$$\delta_j^l \overset{\text{def}}{=} \frac{\partial L}{\partial z_j^l}$$

1. Obtain δ_j^L for data (i)

Element-wise

$$\delta_j^L \overset{\text{def}}{=} \frac{\partial L}{\partial z_j^L} = \frac{\partial L}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}$$

Vectorize for layers

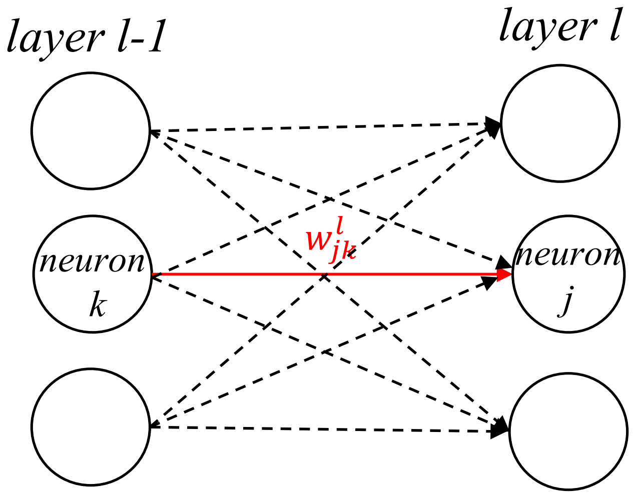
$$\delta^L = \begin{bmatrix} \delta_1^L \\ \delta_2^L \\ \cdots \\ \delta_j^L \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1^L}\frac{\partial a_1^L}{\partial z_1^L} \\ \frac{\partial L}{\partial a_2^L}\frac{\partial a_2^L}{\partial z_2^L} \\ \cdots \\ \frac{\partial L}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial a_1^L} \\ \frac{\partial L}{\partial a_2^L} \\ \cdots \\ \frac{\partial L}{\partial a_j^L} \end{bmatrix} * \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} \\ \frac{\partial a_2^L}{\partial z_2^L} \\ \cdots \\ \frac{\partial a_j^L}{\partial z_j^L} \end{bmatrix} = \nabla_{a^L} L * \sigma'(z^L)$$

Especially for cross-entropy

$$\delta_j^L = \frac{\partial L}{\partial z_j^L} = a_j^L - y_j \qquad \delta^L = a^L - y$$

* Sorry about duplicate use of L, i.e., layer L and loss L as well.

Generic form for data (i)



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma\left({w_j^l}^T a^{l-1} + b_j^l\right)$$

$$a^l = \sigma(W^l a^{l-1} + b^l)$$

$$\delta_j^l \overset{\text{def}}{=} \frac{\partial L}{\partial z_j^l}$$

2. Express δ_k^{l-1} with δ_j^l

Element-wise *Note $z_j^l = \sum_k w_{jk}^l \sigma(z_k^{l-1})$

$$\delta_k^{l-1} = \frac{\partial L}{\partial z_k^{l-1}} = \sum_j \frac{\partial L}{\partial z_j^l}\frac{\partial z_j^l}{\partial z_k^{l-1}} = \sum_j \delta_j^l \frac{\partial z_j^l}{\partial z_k^{l-1}} = \sum_j \delta_j^l w_{jk}^l \sigma'(z_k^{l-1})$$

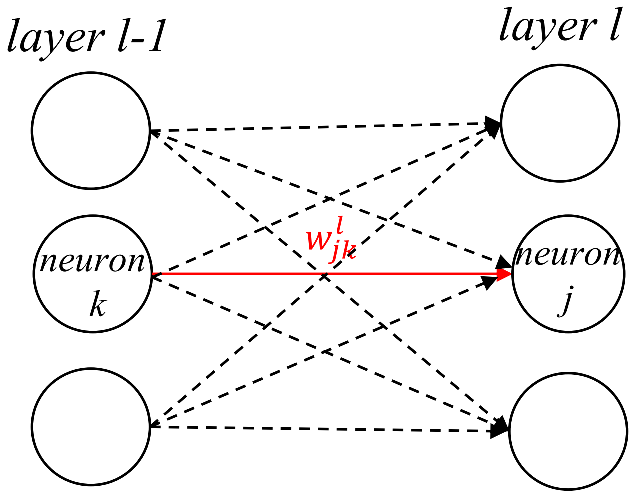
$$\delta_k^{l-1} = [w_{1k}^l w_{2k}^l \dots w_{jk}^l]\begin{bmatrix}\delta_1^l \\ \delta_2^l \\ \dots \\ \delta_j^l\end{bmatrix}\sigma'\left(z_k^{l-1}\right) = w_{\_k}^l \delta^l \sigma'\left(z_k^{l-1}\right)$$

Vectorize

$$\delta^{l-1} = \begin{bmatrix}\delta_1^{l-1} \\ \delta_2^{l-1} \\ \dots \\ \delta_k^{l-1}\end{bmatrix} = \begin{bmatrix} w_{\_1}^l \delta^l \sigma'(z_1^{l-1}) \\ w_{\_2}^l \delta^l \sigma'(z_2^{l-1}) \\ \dots \\ w_{\_k}^l \delta^l \sigma'(z_k^{l-1}) \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{21}^l \dots w_{j1}^l \\ w_{12}^l & w_{22}^l \dots w_{j2}^l \\ & \dots \\ w_{1k}^l & w_{2k}^l \dots w_{jk}^l \end{bmatrix} \delta^l * \sigma'(z^{l-1})$$

$$= {W^l}^T \delta^l * \sigma'(z^{l-1})$$

Generic form for data (i)



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma\left({w_j^l}^T a^{l-1} + b_j^l\right)$$

$$a^l = \sigma(W^l a^{l-1} + b^l)$$

$$\delta_j^l \stackrel{\text{def}}{=} \frac{\partial L}{\partial z_j^l}$$

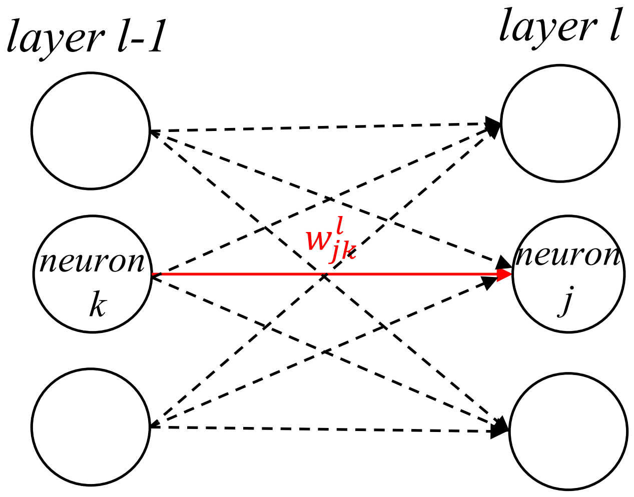
3. Compute $\frac{\partial L}{\partial w_{jk}^l}$ and $\frac{\partial L}{\partial b_j^l}$

Element-wise

$$\frac{\partial L}{\partial w_{jk}^l} = \frac{\partial L}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$$

$$\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l$$

Generic form for data (i)



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) = \sigma\left({w_j^l}^T a^{l-1} + b_j^l\right)$$

$$a^l = \sigma(W^l a^{l-1} + b^l)$$

$$\delta_j^l \stackrel{\text{def}}{=} \frac{\partial L}{\partial z_j^l}$$

$$\delta^L = \nabla_{a^L} L * \sigma'(z^L)$$

$$\delta_j^L = \frac{\partial L}{\partial z_j^L} = \frac{\partial L}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}$$

Especially for cross-entropy

$$\delta_j^L = \frac{\partial L}{\partial z_j^L} = a_j^L - y_j \qquad \delta^L = a^L - y$$

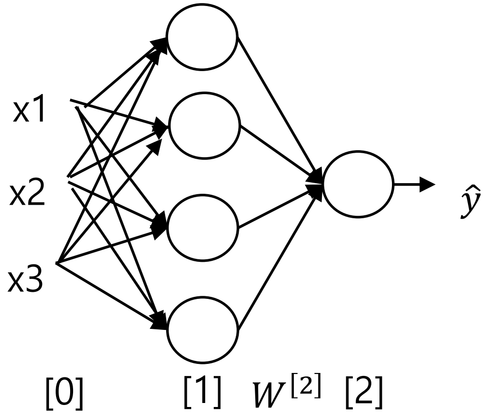
$$\delta^{l-1} = {W^l}^T \delta^l * \sigma'(z^{l-1})$$

$$\delta_k^{l-1} = \textstyle\sum_j \delta_j^l w_{jk}^l \sigma'(z_k^{l-1}) = w_{\_k}^l \delta^l * \sigma'\left(z_k^{l-1}\right)$$

$$\frac{\partial L}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$$

$$\frac{\partial L}{\partial b_j^l} = \delta_j^l$$

Gradient descent for 3 layer NNs



- Parameters : $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$
  - #input features : $\mathrm{n_x} = n^{[0]}$,
  - #hidden node : $n^{[1]}$,
  - #output node : $n^{[1]} = 1$
  - Shape of $W^{[1]}$: $(n^{[1]}, n^{[0]})$
  - Shape of $b^{[1]}$: $(n^{[1]}, 1)$
  - Shape of $W^{[2]}$: $(n^{[2]}, n^{[1]})$
  - Shape of $b^{[1]}$: $(n^{[2]}, 1)$

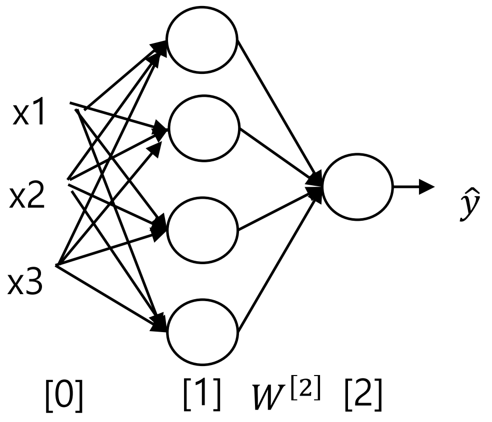
- Cost function
  - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

#Gradient Descent
Repeat:
  compute $\hat{y}^{(i)}, i = 1, \ldots, m$
  $\mathrm{dw}^{[1]} = \frac{\mathrm{dJ}}{dw^{[1]}}$, $\mathrm{db}^{[1]} = \frac{\mathrm{dJ}}{db^{[1]}}$, …
  $\mathrm{W}^{[1]} = \mathrm{W}^{[1]} - \alpha \times dw^{[1]}$
  $\mathrm{b}^{[1]} = \mathrm{b}^{[1]} - \alpha \times db^{[1]}$
  $\mathrm{W}^{[2]} = \mathrm{W}^{[2]} - \alpha \times dw^{[2]}$
  $\mathrm{b}^{[2]} = \mathrm{b}^{[2]} - \alpha \times db^{[2]}$

We need to obtain $dw^{[1]}$, $db^{[1]}$, $dw^{[2]}$, $db^{[2]}$

Gradient descent for 3 layer NNs

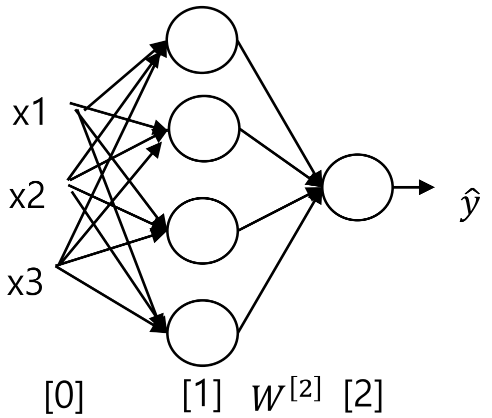
- Forward propagation
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g(Z^{[2]}) = \sigma(Z^{[2]})$
- Cost function
  - $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$



- Compute $dZ^{[2]}$
  - Note $Z^{[2]} \stackrel{\text{def}}{=} [z^{[2](1)}, z^{[2](2)}, \dots, z^{[2](m)}]$, $z^{[2](i)}: scalar,$
  - $dZ^{[2]} \stackrel{\text{def}}{=} \left[\frac{dL^{(1)}}{dz^{[2](1)}}, \frac{dL^{(2)}}{dz^{[2](2)}}, \dots, \frac{dL^{(m)}}{dz^{[2](m)}}\right]$
  - In generic form for data (i), $\delta_j^L \stackrel{\text{def}}{=} \frac{\partial J}{\partial z_j^L} = a_j^L - y_j$
  - Thus, $\frac{dL^{(i)}}{dz^{[2](i)}} = a^{[2](i)} - y^{(i)}$
  - $dZ^{[2]} = [a^{[2](1)} - y^{(1)}, \dots, a^{[2](m)} - y^{(m)}]$
    $= A^{[2]} - Y \text{ , } where\ Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$

Gradient descent for 3 layer NNs

- Forward propagation
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g\left(Z^{[2]}\right) = \sigma(Z^{[2]})$
- Cost function
  - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$



- Compute $dW^{[2]}$
  - $dW^{[2]} \overset{\text{def}}{=} \frac{dJ}{dW^{[2]}} = \left[\frac{dJ}{dw_1^{[2]}}, \frac{dJ}{dw_2^{[2]}}, \dots, \frac{dJ}{dw_{n^{[1]}}^{[2]}}\right]$
  - In generic form for data (i), $\frac{\partial L}{\partial w_{jk}^{l}} = \delta_j^l a_k^{l-1}$
  - $\frac{\partial L^{(i)}}{\partial w_k^{[2]}} = \frac{dL^{(i)}}{dz^{[2](i)}} a_k^{[1](i)}$
  - $\frac{\partial J}{\partial w_k^{[2]}} = \frac{1}{m}\left(\frac{dL^{(1)}}{dz^{[2](1)}} a_k^{[1](1)} + \dots + \frac{dL^{(m)}}{dz^{[2](m)}} a_k^{[1](m)}\right) = \frac{1}{m}\left[\frac{dL^{(1)}}{dz^{[2](1)}}, \dots, \frac{dL^{(m)}}{dz^{[2](m)}}\right]\begin{bmatrix} a_k^{[1](1)} \\ \dots \\ a_k^{[1](m)} \end{bmatrix}$

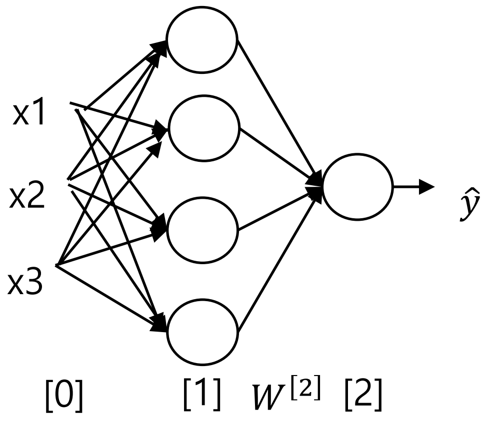
$= \frac{1}{m} dZ^{[2]} {a_k^{[1]}}^T,\ where\ a_k^{[1]} = [a_k^{[1](1)}, \dots a_k^{[1](m)}]$

- $dW^{[2]} = \left[\frac{dJ}{dw_1^{[2]}}, \frac{dJ}{dw_2^{[2]}}, \dots, \frac{dJ}{dw_{n^{[1]}}^{[2]}}\right] = \frac{1}{m}\left[dZ^{[2]} {a_1^{[1]}}^T, \dots, dZ^{[2]} {a_{n^{[1]}}^{[1]}}^T\right] =$

$$= \frac{1}{m} dZ^{[2]} \left[{a_1^{[1]}}^T, \dots, {a_{n^{[1]}}^{[1]}}^T\right] = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

Gradient descent for 3 layer NNs

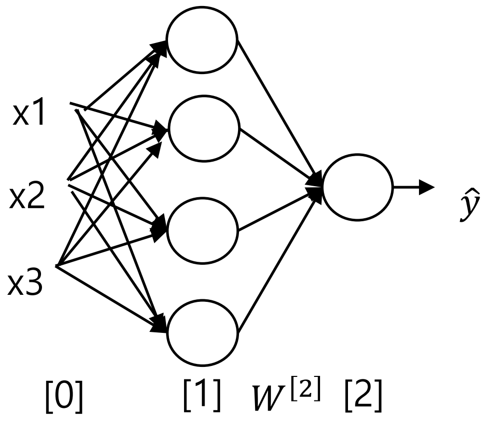
- Forward propagation
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g\left(Z^{[2]}\right) = \sigma(Z^{[2]})$
- Cost function
  - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$



- Back propagation
  - $dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$
  - $db^{[2]} = \frac{1}{m} dZ^{[2]} 1^{T} = np.sum(dZ^{[2]}, axis = 1, keepdims = True)$

Gradient descent for 3 layer NNs

- Forward propagation
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g\left(Z^{[2]}\right) = \sigma(Z^{[2]})$
- Cost function
  - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$

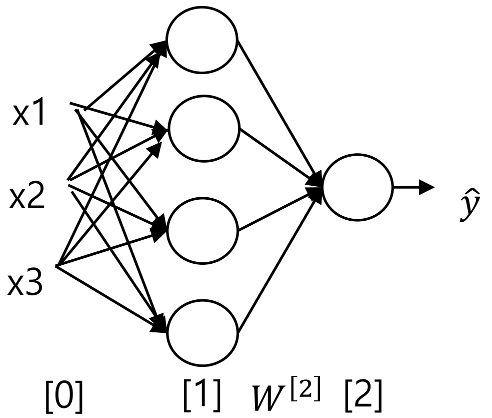


- Express $dZ^{[1]}$ with $dZ^{[2]}$
  - *In generic form*
    - $\delta^{l-1} = {W^{l}}^{T}\ \delta^{l} * \sigma'(z^{l-1})$
  - $dz^{[1](i)} = W^{[2]T}dz^{[2](i)} * \frac{dg(z^{[1](i)})}{dz^{[1](i)}}$
  - $dZ^{[1]} = [W^{[2]T}dz^{[2](1)} * \frac{dg\left(z^{[1](1)}\right)}{dz^{[1](1)}}, \dots, W^{[2]T}dz^{[2](m)} * \frac{dg(z^{[1](m)})}{dz^{[1](m)}}]$

  $= W^{[2]T}dZ^{[2]} * \frac{dg(Z^{[1]})}{dZ^{[1]}}$

Gradient descent for 3 layer NNs

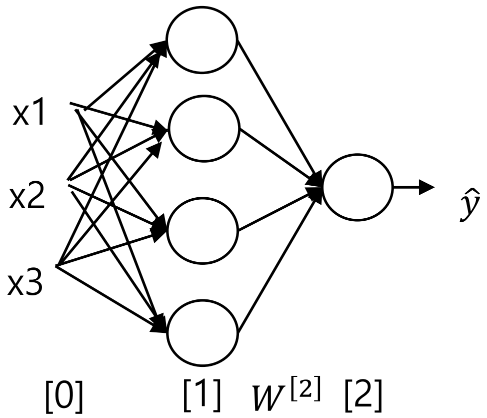
- Forward propagation
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g\left(Z^{[2]}\right) = \sigma(Z^{[2]})$
- Cost function
  - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$



- Compute $dW^{[1]}$
  - *In generic form :* $\frac{\partial L}{\partial w_{jk}^{l}} = \delta_j^l a_k^{l-1}$
  - $\frac{\partial L^{(i)}}{\partial w_{jk}^{[1]}} = \frac{dL^{(i)}}{dz_j^{[1](i)}} x_k^{(i)}$
  - $\frac{\partial J}{\partial w_{jk}^{[1]}} = \frac{1}{m}\left(\frac{dL^{(1)}}{dz_j^{[1](1)}} x_k^{(1)} + \ldots + \frac{dL^{(m)}}{dz_j^{[1](m)}} x_k^{(m)}\right) =$
    $\frac{1}{m}\begin{bmatrix}\frac{dL^{(1)}}{dz_j^{[1](1)}}, \ldots, \frac{dL^{(m)}}{dz_j^{[1](m)}}\end{bmatrix}\begin{bmatrix} x_k^{(1)} \\ \ldots \\ x_k^{(m)}\end{bmatrix} = \frac{1}{m} dz_j^{[1]} x_k^T$
  - $\frac{\partial J}{\partial W^{[1]}} = \begin{bmatrix} \frac{\partial J}{\partial w_{11}^{[1]}}, \ldots, \frac{\partial J}{\partial w_{1k}^{[1]}} \\ \ldots \\ \frac{\partial J}{\partial w_{j1}^{[1]}}, \ldots, \frac{\partial J}{\partial w_{jk}^{[1]}} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} dz_1^{[1]} x_1^T, \ldots, dz_1^{[1]} x_k^T \\ \ldots \\ dz_j^{[1]} x_1^T, \ldots, dz_j^{[1]} x_k^T \end{bmatrix} =$
    $\frac{1}{m}\begin{bmatrix} dz_1^{[1]} \\ \ldots \\ dz_j^{[1]} \end{bmatrix}[x_1^T \ldots x_k^T] = \frac{1}{m} dZ^{[1]} X^T$

Gradient descent for 3 layer NNs

- Forward propagation
  - $Z^{[1]} = W^{[1]}X + \mathrm{b}^{[1]}$
  - $A^{[1]} = g(Z^{[1]})$
  - $Z^{[2]} = W^{[2]}A^{[1]} + \mathrm{b}^{[2]}$
  - $A^{[2]} = g\left(Z^{[2]}\right) = \sigma(Z^{[2]})$
- Cost function
  - $J\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$



- Compute $db^{[1]}$
  - $In\ generic\ form: \frac{\partial L}{\partial b_j^l} = \delta_j^l$
  - $\frac{\partial L^{(i)}}{\partial b_j^{[1]}} = \frac{dL^{(i)}}{dz_j^{[1](i)}}$
  - $\frac{\partial J}{\partial b_j^{[1]}} = \frac{1}{m}\left(\frac{dL^{(1)}}{dz_j^{[1](1)}} + \ldots + \frac{dL^{(m)}}{dz_j^{[1](m)}}\right) = \frac{1}{m}\left[\frac{dL^{(1)}}{dz_j^{[1](1)}}, \ldots, \frac{dL^{(m)}}{dz_j^{[1](m)}}\right]\begin{bmatrix} 1 \\ \ldots \\ 1 \end{bmatrix} = \frac{1}{m} dz_j^{[1]} 1^T$
  - $\frac{\partial J}{\partial b^{[1]}} = \begin{bmatrix} \frac{\partial J}{\partial b_1^{[1]}} \\ \ldots \\ \frac{\partial J}{\partial w_j^{[1]}} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} dz_1^{[1]} 1^T \\ \ldots \\ dz_j^{[1]} 1^T \end{bmatrix} = \frac{1}{m}\begin{bmatrix} dz_1^{[1]} \\ \ldots \\ dz_j^{[1]} \end{bmatrix} 1^T = \frac{1}{m} dZ^{[1]} 1^T$
  - $= \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

Summary

- $dZ^{[2]} = A^{[2]} - Y \text{ , } where\ Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$
- $dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$
- $db^{[2]} = \frac{1}{m} dZ^{[2]} 1^T = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
- $dZ^{[1]} = W^{[2]T} dZ^{[2]} * \frac{dg(Z^{[1]})}{dZ^{[1]}}$
- $dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
- $db^{[1]} = \frac{1}{m} dZ^{[1]} 1^T = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

- Why backpropagation is called “fast”?
- What happen when we initialize W and b to “zero”?

What happen when we initialize W and b to “zero”?

$$dZ^{[2]} = A^{[2]} - Y \text{ , } where\ Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} dZ^{[2]} 1^T = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * \frac{dg(Z^{[1]})}{dZ^{[1]}}$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} dZ^{[1]} 1^T = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$