LECTURE 20

Neural Networks

Building neural networks in Using Keras

Data Science, Spring 2024 @ Knowledge Stream

Sana Jabbar

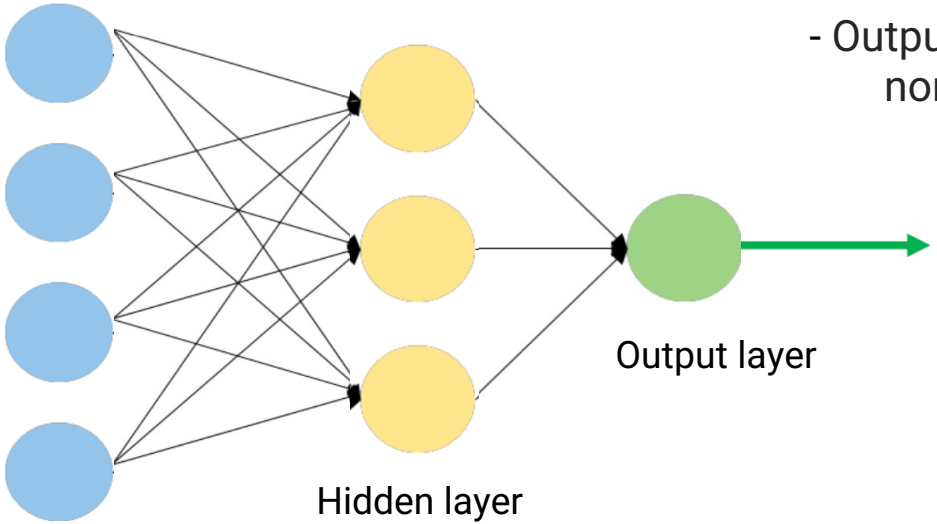
Neural Networks

Lecture 20

- Introduction to Neural Networks
- Neural network forward pass
- Activation functions
- Back Propagation

Neural Networks

A neural network is a set of neurons organized in layers



- Output is a
  non-linear function
  of a linear combination
  of non-linear functions
  of a linear combination of inputs

Each unit in the network is called a neuron and the network of neurons is called **Neural Network.**

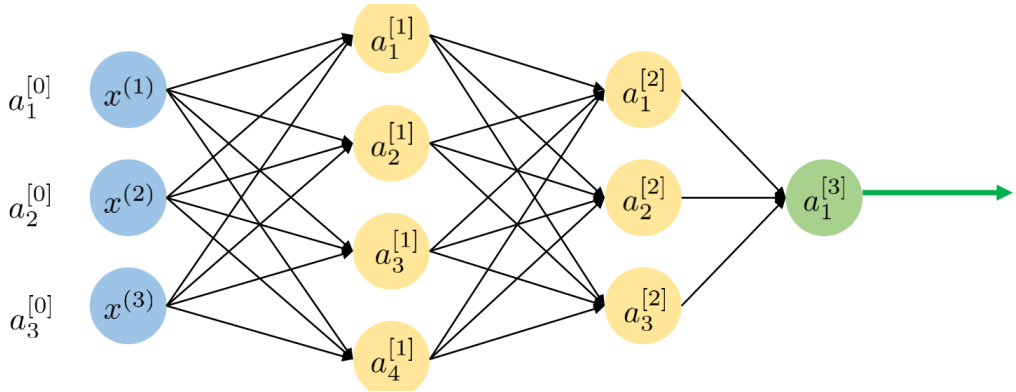
- Given the input and parameters of the neurons, we can determine the output by traversing layers from input to output. This is referred to as **Forward Pass**.

Neural Networks

- L - number of layers.
- $\mathbf{a}^{[\ell]} = \mathbf{x}$ input layer.
- $\mathbf{a}^{[L]} = y$ output layer.
- Number of nodes in the ℓ-th layer, $m^{[\ell]}$
- $\mathbf{a}^{[\ell]}$ - vector of outputs of ℓ-th node.
- $a_i^{[\ell]}$ denotes the output of i-th node in the ℓ-th layer.

Example: 3-layer network, 2 hidden layers

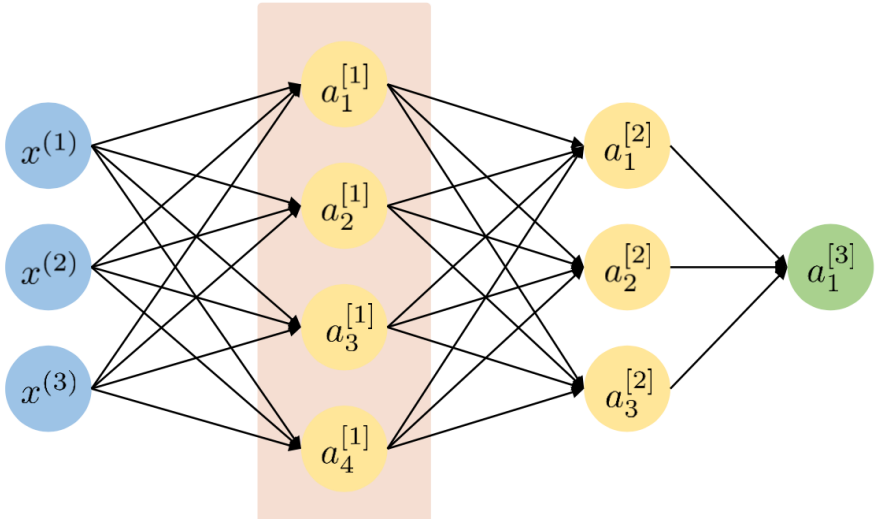
- $L = 3$
- $m^{[1]} = 4,\ m^{[2]} = 3,\ m^{[3]} = 1$



Forward Pass

- $\mathbf{w}_i^{[\ell]}$ and $b_i^{[\ell]}$ denote the weight and bias associated with the i-th node in the ℓ-th layer respectively.
- $w_{i,j}^{[\ell]}$ denote the weight and bias associated with the $j-th$ input of the i-th node in the ℓ-th layer respectively.

Example: 3-layer network, 2 hidden layers



Layer 1 output

$$a_1^{[1]} = g(z_1^{[1]}), \quad z_1^{[1]} = \mathbf{w}_1^{[1]T}\mathbf{x} + b_1^{[1]}$$

$$a_2^{[1]} = g(z_2^{[1]}), \quad z_2^{[1]} = \mathbf{w}_2^{[1]T}\mathbf{x} + b_2^{[1]}$$

$$a_3^{[1]} = g(z_3^{[1]}), \quad z_3^{[1]} = \mathbf{w}_3^{[1]T}\mathbf{x} + b_3^{[1]}$$

$$a_4^{[1]} = g(z_4^{[1]}), \quad z_4^{[1]} = \mathbf{w}_4^{[1]T}\mathbf{x} + b_4^{[1]}$$

Forward Pass

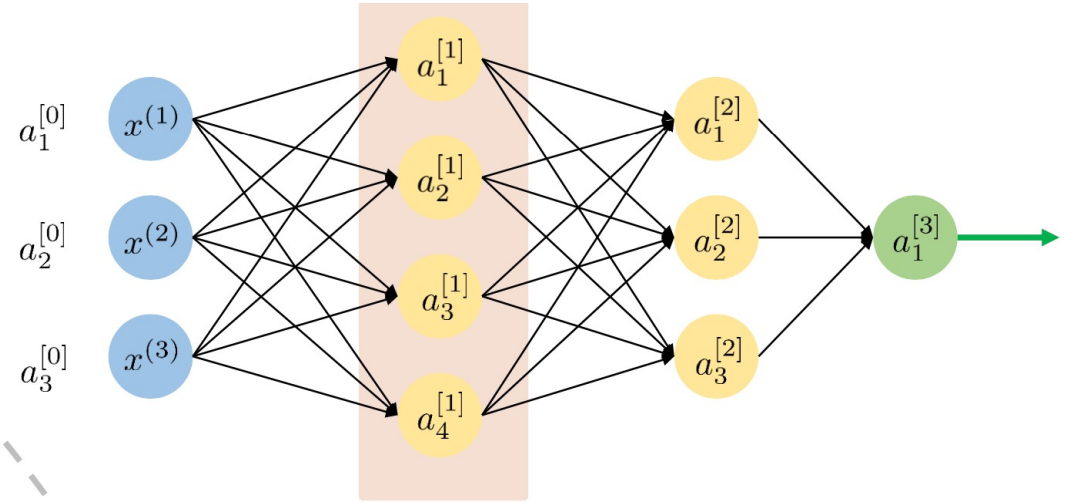
Layer 1 output

$$a_1^{[1]} = g(z_1^{[1]}), \quad z_1^{[1]} = \mathbf{w}_1^{[1]T}\mathbf{x} + b_1^{[1]}$$

$$a_2^{[1]} = g(z_2^{[1]}), \quad z_2^{[1]} = \mathbf{w}_2^{[1]T}\mathbf{x} + b_2^{[1]}$$

$$a_3^{[1]} = g(z_3^{[1]}), \quad z_3^{[1]} = \mathbf{w}_3^{[1]T}\mathbf{x} + b_3^{[1]}$$

$$a_4^{[1]} = g(z_4^{[1]}), \quad z_4^{[1]} = \mathbf{w}_4^{[1]T}\mathbf{x} + b_4^{[1]}$$



$$\mathbf{W}^{[1]} = \begin{bmatrix} \mathbf{w}_1^{[1]T} \\ \mathbf{w}_2^{[1]T} \\ \mathbf{w}_3^{[1]T} \\ \mathbf{w}_4^{[1]T} \end{bmatrix} \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]T} \\ b_2^{[1]T} \\ b_3^{[1]T} \\ b_4^{[1]T} \end{bmatrix} \quad \mathbf{z}^{[1]} = \begin{bmatrix} z_1^{[1]T} \\ z_2^{[1]T} \\ z_3^{[1]T} \\ z_4^{[1]T} \end{bmatrix}$$

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

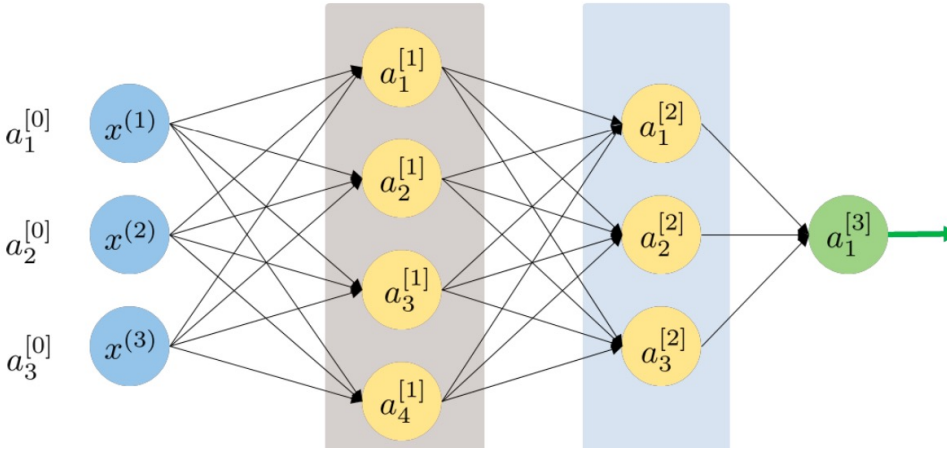
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

- $\mathbf{W}^{[1]}$ and $\mathbf{b}^{[1]}$ are the parameters of the first layer.

Forward Pass

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$



- Q. What is the size of $\mathbf{W}^{[1]}$?

  A. No. of nodes $\times$ No. of inputs. 4×3

  No. of nodes $\times$ No. nodes in the previous layer.

- Q. Can we write output of second layer using the notation we have defined?

$$\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]}), \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

- Q. What is the size of $\mathbf{W}^{[2]}$? 3×4
- Q. What is the size of $\mathbf{W}^{[3]}$? 1×3

$$\mathbf{a}^{[3]} = g(\mathbf{z}^{[3]}), \quad \mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{a}^{[2]} + \mathbf{b}^{[3]}$$

Forward Pass

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

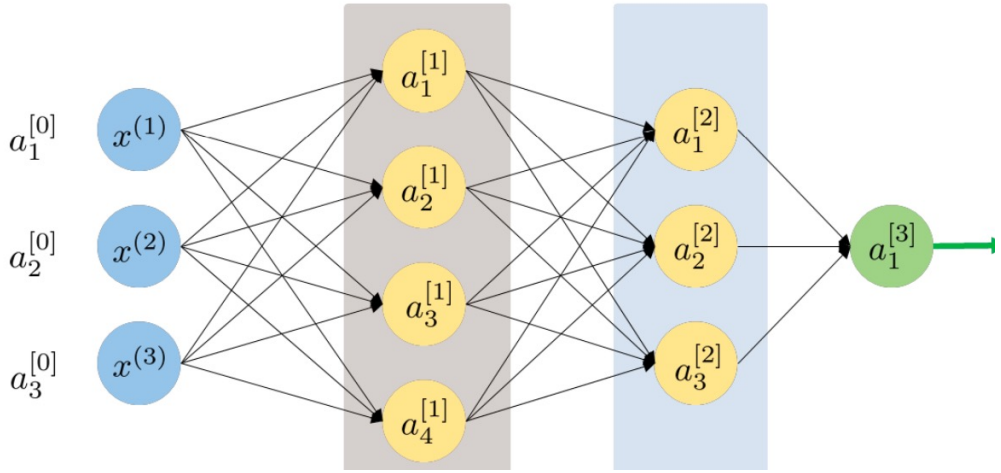
$$(4 \times 1) = (4 \times 3)(3 \times 1) + (4 \times 1)$$

$$\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]}), \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$(3 \times 1) = (3 \times 4)(4 \times 1) + (3 \times 1)$$

$$\mathbf{a}^{[3]} = g(\mathbf{z}^{[3]}), \quad \mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{a}^{[2]} + \mathbf{b}^{[3]}$$

$$(1 \times 1) = (1 \times 3)(3 \times 1) + (1 \times 1)$$



- How many parameters do we have by the way?

Activation Functions

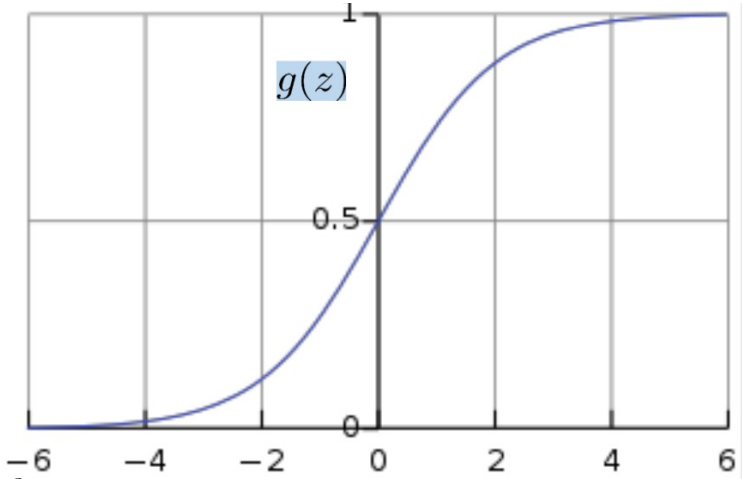
Lecture 20

- Introduction to Neural Networks
- Neural network forward pass
- **Activation functions**
- Back Propagation

Sigmoid

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$ (Sigmoid: because of S shaped curve)

- Squishes the input between (0,1)
- Suitable for output neuron when classification is applied
- We have already used this function: Logistic regression



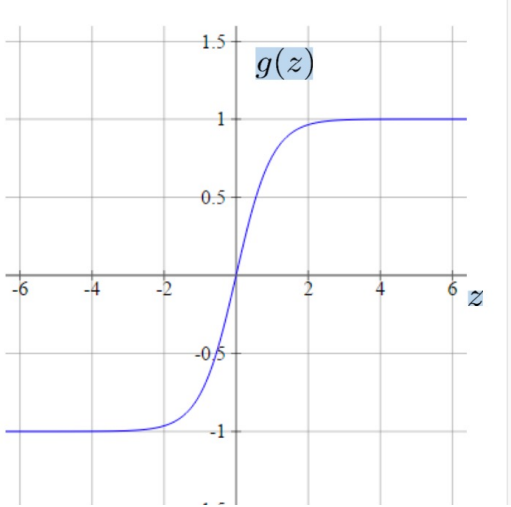
Issues

- **Vanishing Gradient:** As z becomes very large or very small, the sigmoid function approaches 1 or 0
- **Output Not Zero-Centric**: it can slow down training. Neural networks typically learn better when the outputs are centered around zero because this facilitates faster convergence during training.

tanh

$$g(z) = \tanh(z) = 2\sigma(2z) - 1 = \frac{2}{1 + e^{-2z}} - 1 = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

- Squishes the input between (-1,1)
- Suitable for output neurons when classification is applied

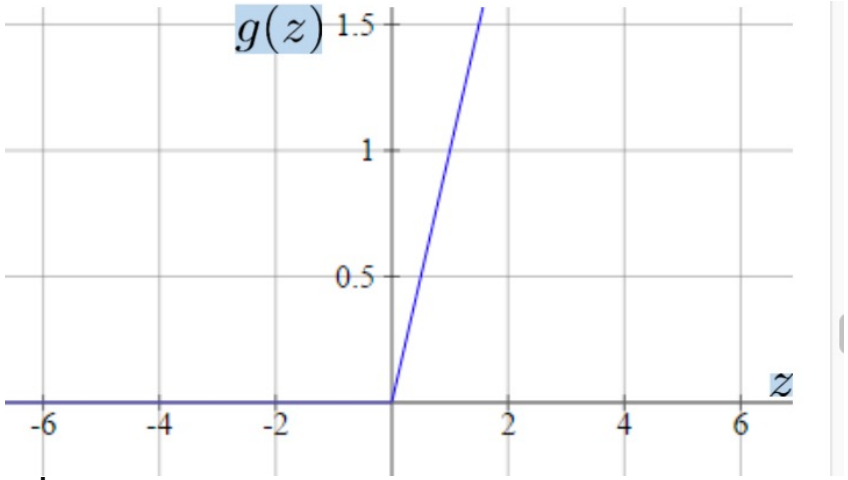


Issues

- **Vanishing Gradient:** As z becomes very large or very small, the sigmoid function approaches 1 or 0
- **Output Zero-Centric**: The Network will learn better because the output is zero-centric, it can solve the problem of sigmoid.

Rectified Linear Unit (ReLu)

$$g(z) = \max(0, z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases}$$



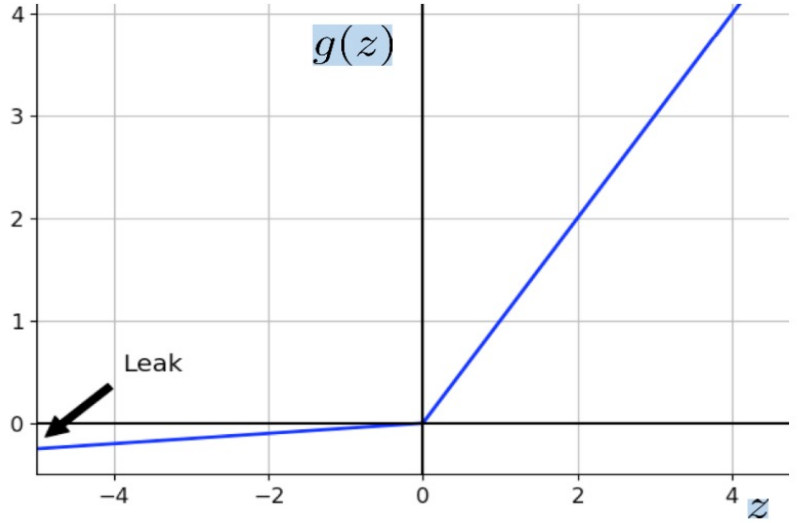
The most used activation function

Why?

- Unlike Sigmoid and tanh that required the computation of exponents, easier to compute
- It only activates the output is non-negative.
- It deactivates the neurons with values lass than 0.

Leaky Rectified Linear Unit (Leaky ReLu)

$$g(z) = \begin{cases} z & z \geq 0 \\ \alpha z & z < 0 \end{cases}$$



- It resolves the problem of ReLU
- Due to non-zero output for negative values, It keep neurons alive
- It is differentiable

$$g'(z) = \begin{cases} 1 & z \geq 0 \\ \alpha & z < 0 \end{cases}$$

We use rectified linear (ReLU), leaky rectified (Leaky ReLU), linear sigmoid, and tanh for hidden layer.

For Regression tasks

- Identify the function if your problem is regression.
- **Linear Activation**: This function simply outputs the weighted sum of the inputs without applying any transformation. It is often used when the output value can be any real number.
- Linear functions are commonly used for regression tasks because they allow the model to output continuous values without any specific range or transformation.

For Classification tasks

- We prefer to use sigmoid, tanh functions and their combinations.
- Due to the saturation problem, sigmoids and tanh functions are sometimes avoided.
- As indicated earlier, ReLU function is mostly used (computationally fast).
- ReLU variants are used to resolve a dead neuron issue (e.g., Leaky ReLU).
- It must be noted that ReLU function is only used in the hidden layers.
- Start with ReLU or leaky/randomized ReLU; if the results are unsatisfactory, you may try other activation functions.

Back Propagation

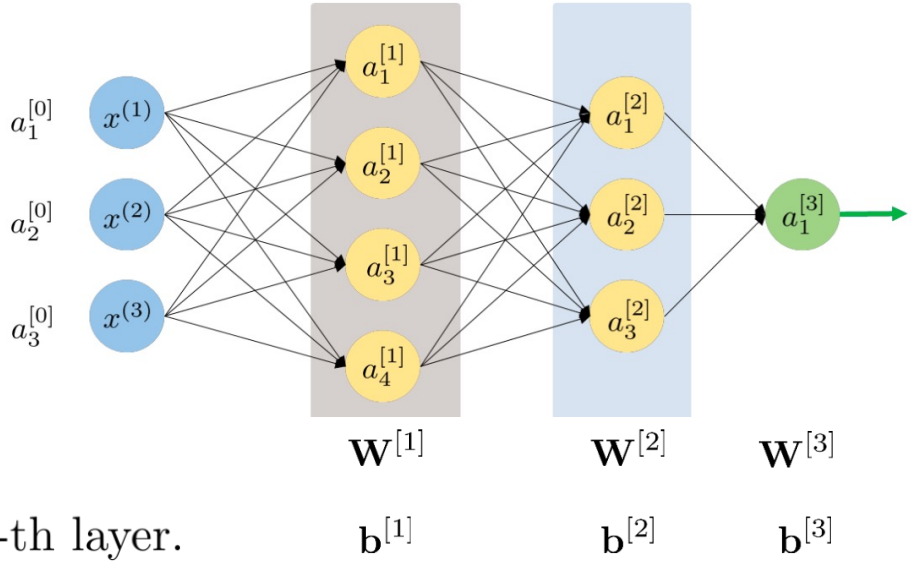
Lecture 20

- Introduction to Neural Networks
- Neural network forward pass
- Activation functions
- **Back Propagation**

Learning Weights

Given the training data, we want to learn the weights (weight matrices+bias vectors) for hidden layers and output layer.

Example:



Notation Revisit

- L - number of layers.
- $\mathbf{a}^{[\ell]} = \mathbf{x}$ input layer.
- $\mathbf{a}^{[L]} = y$ output layer.
- Number of nodes in the ℓ-th layer, $m^{[\ell]}$
- $\mathbf{a}^{[\ell]}$ - vector of outputs of ℓ-th node.
- $a_i^{[\ell]}$ denotes the output of i-th node in the ℓ-th layer.

Parameters need to be learned!

Learning Weights

- We assume we have training data D given by

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

- Given our prior knowledge, output y is a composite function of input $\mathbf{x}$. Therefore, it is continuous and differentiable and we can use chain rule to compute the gradient.

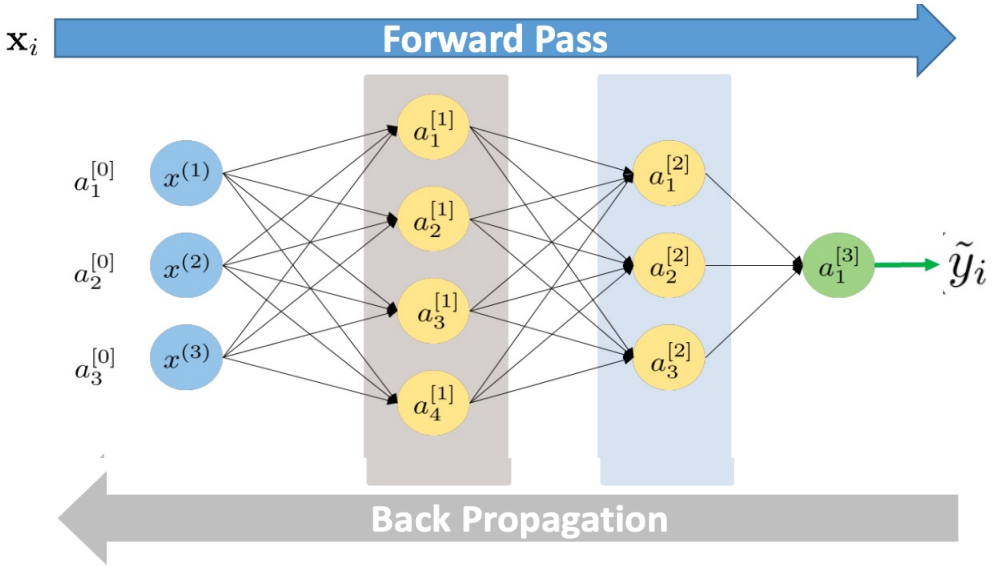
  where $\tilde{y}_i$ denotes the output of the neural network for i-th input.

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{n} \left(\tilde{y}_i - y_i\right)^2$$

- We can use gradient descent to learn the weight matrices and bias vectors.

We use a method called 'Back Propagation' to implement the chain rule for the computation the gradient.

Learning Weights



$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n}\left(\tilde{y}_i - y_i\right)^2$$

$$w_{i,j}^{[\ell]} = w_{i,j}^{[\ell]} - \alpha\frac{\partial\mathcal{L}}{\partial w_{i,j}^{[\ell]}}$$

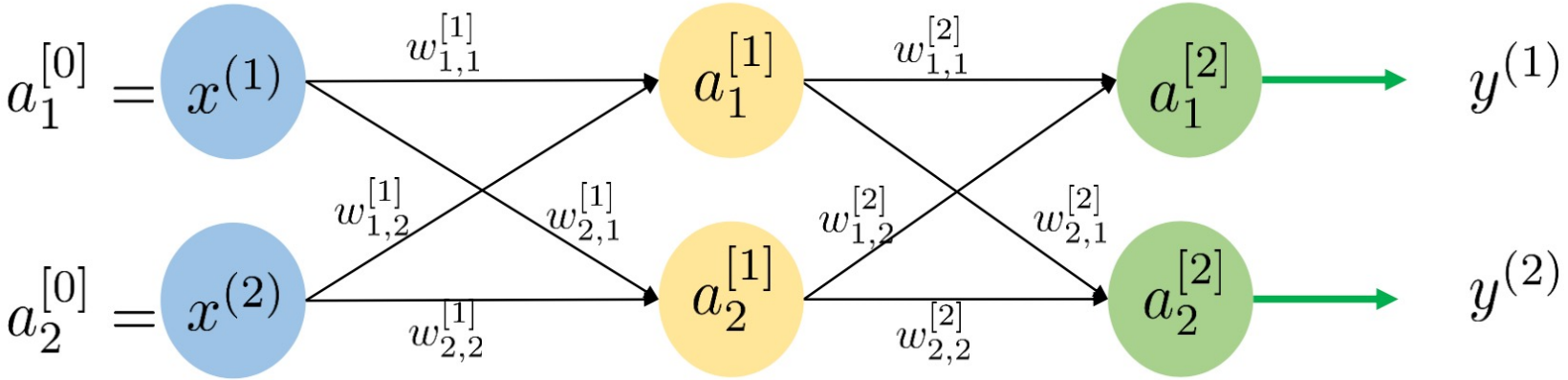
The weights are the only parameters that can be modified to make the loss function as low as possible.

Learning problem reduces to the question of calculating gradient (partial derivatives) of loss function.

- We compute the derivate by propagating the total loss at the output node back into the neural network to determine the contribution of every node in the loss.

Learning Weights

- 2 layer with 2 neurons in the hidden layer , 2 inputs, 2 outputs network.
- Assuming sigmoid as activation function, that is, $g(z) = \sigma(z)$.



- Given training data

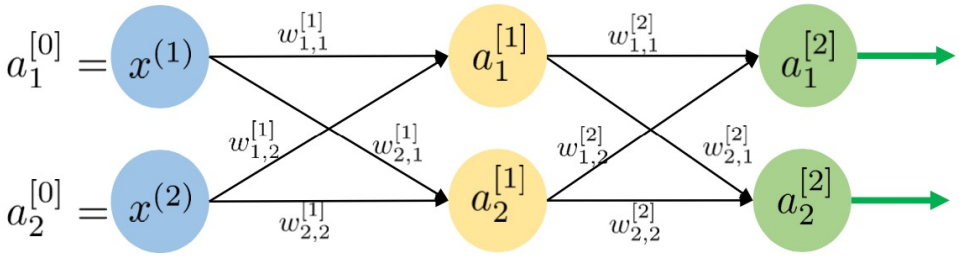
  $x^{(1)} = 0.05, \quad x^{(2)} = 0.1, \quad y^{(1)} = 0.01, \quad y^{(2)} = 0.99$

- Initial values of weights and biases:

  $w_{1,1}^{[1]} = 0.15,\ w_{1,2}^{[1]} = 0.2,\ w_{2,1}^{[1]} = 0.25,\ w_{2,2}^{[1]} = 0.3,\ b_1^{[1]} = 0.35,\ b_2^{[1]} = 0.35.$

  $w_{1,1}^{[2]} = 0.4,\ w_{1,2}^{[2]} = 0.45,\ w_{2,1}^{[2]} = 0.5,\ w_{2,2}^{[2]} = 0.55,\ b_1^{[1]} = 0.6,\ b_2^{[1]} = 0.6.$

Learning Weights



- Loss function (noting output is a vector):

$$\mathcal{L} = \frac{1}{2}\|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2$$

$$\mathcal{L} = \frac{1}{2}\|(0.01, 0.99) - (0.7514, 0.7729)\|^2 = 0.2984$$

Forward Pass

$$a_1^{[1]} = g(z_1^{[1]}), \quad z_1^{[1]} = \mathbf{w}_1^{[1]T}\mathbf{x} + b_1^{[1]}$$

$$a_2^{[1]} = g(z_2^{[1]}), \quad z_2^{[1]} = \mathbf{w}_2^{[1]T}\mathbf{x} + b_2^{[1]}$$

$$a_1^{[2]} = g(z_1^{[2]}), \quad z_1^{[2]} = \mathbf{w}_1^{[2]T}\mathbf{x} + b_1^{[2]}$$

$$a_2^{[2]} = g(z_2^{[2]}), \quad z_2^{[2]} = \mathbf{w}_2^{[2]T}\mathbf{x} + b_2^{[2]}$$

$$z_1^{[1]} = w_{1,1}^{[1]}x^{(1)} + w_{1,2}^{[1]}x^{(2)} + b_1^{[1]} = 0.3775, \quad a_1^{[1]} = g(0.3775) = 0.5933$$

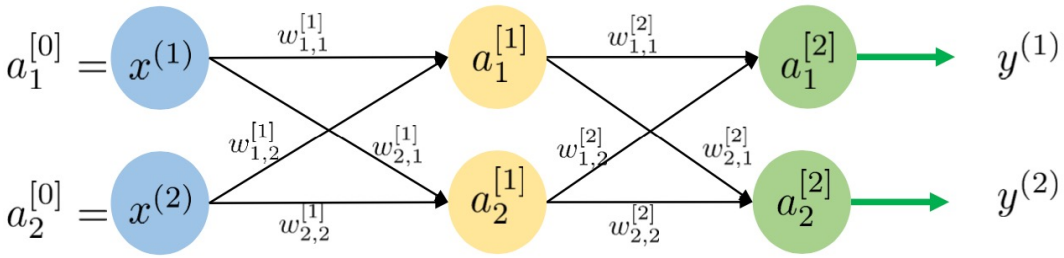
$$z_2^{[1]} = \mathbf{w}_2^{[1]T}\mathbf{x} + b_2^{[1]} = 0.3925, \quad a_2^{[1]} = g(0.3925) = 0.5969$$

$$z_1^{[2]} = \mathbf{w}_1^{[2]T}\mathbf{x} + b_1^{[2]} = 1.106, \quad a_1^{[2]} = g(1.106) = 0.7514 = \tilde{y}^{(1)}$$

$$z_2^{[2]} = \mathbf{w}_2^{[2]T}\mathbf{x} + b_2^{[2]} = 1.225, \quad a_2^{[2]} = g(1.225) = 0.7729 = \tilde{y}^{(2)}$$

Nothing fancy so far, we have computed the output and loss by traversing neural network. Let's compute the contribution of loss by each node; back propagate the loss.

Learning Weights



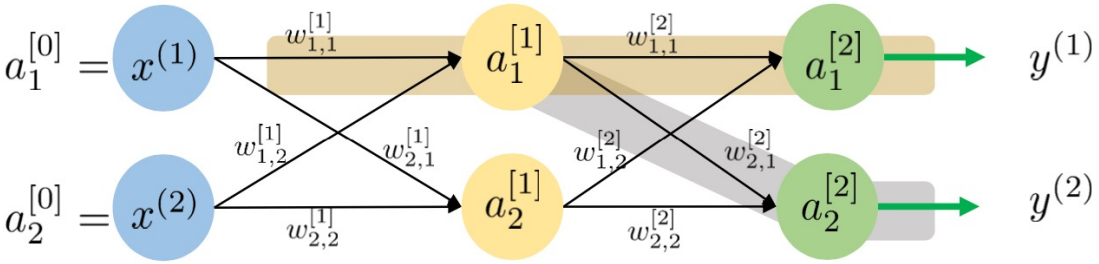
- Consider a case when we want to compute $\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}}$
- Traverse the path from the loss function back to the weight $w_{1,1}^{[2]}$:

$$\left.\begin{aligned} \mathcal{L} &= \frac{1}{2}\|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2 \\ \tilde{y}^{(2)} &= \sigma(z_1^{[2]}) \\ z_1^{[2]} &= w_{1,1}^{[2]} a_1^{[1]} + w_{1,2}^{[2]} a_2^{[1]} + b_1^{[2]} \end{aligned}\right\}$$

$$\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} = 0.0821$$

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} = \tilde{y}^{(1)} - y^{(1)} = 0.7414 \\ \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} = \sigma(\partial z_1^{[2]})\left(1 - \sigma(\partial z_1^{[2]})\right) = 0.1868 \\ \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} = a_1^{[a]} = 0.5933 \end{cases}$$

Learning Weights



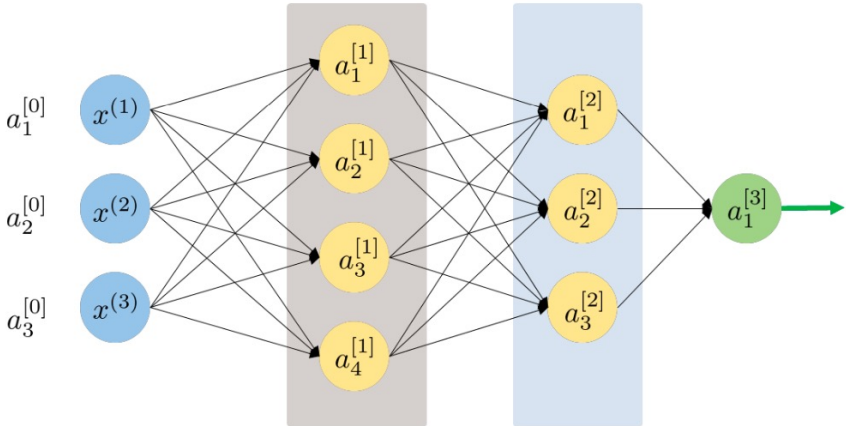
$$\mathcal{L} = \frac{1}{2}\|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2$$

- Consider a case when we want to compute $\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[1]}}$
- Traverse the path from the loss function back to the weight $w_{1,1}^{[1]}$. There are two paths from the output to the weight $w_{1,1}^{[1]}$. In other words, $w_{1,1}^{[1]}$ is contributing to both the outputs.

$$\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}} + \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(2)}} \frac{\partial \tilde{y}^{(2)}}{\partial z_2^{[2]}} \frac{\partial z_2^{[2]}}{\partial a_1^{[1]}} \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}}$$

- Looking tedious but the concept is very straightforward. I encourage you to write one partial derivative using the same approach to strengthen the concept.

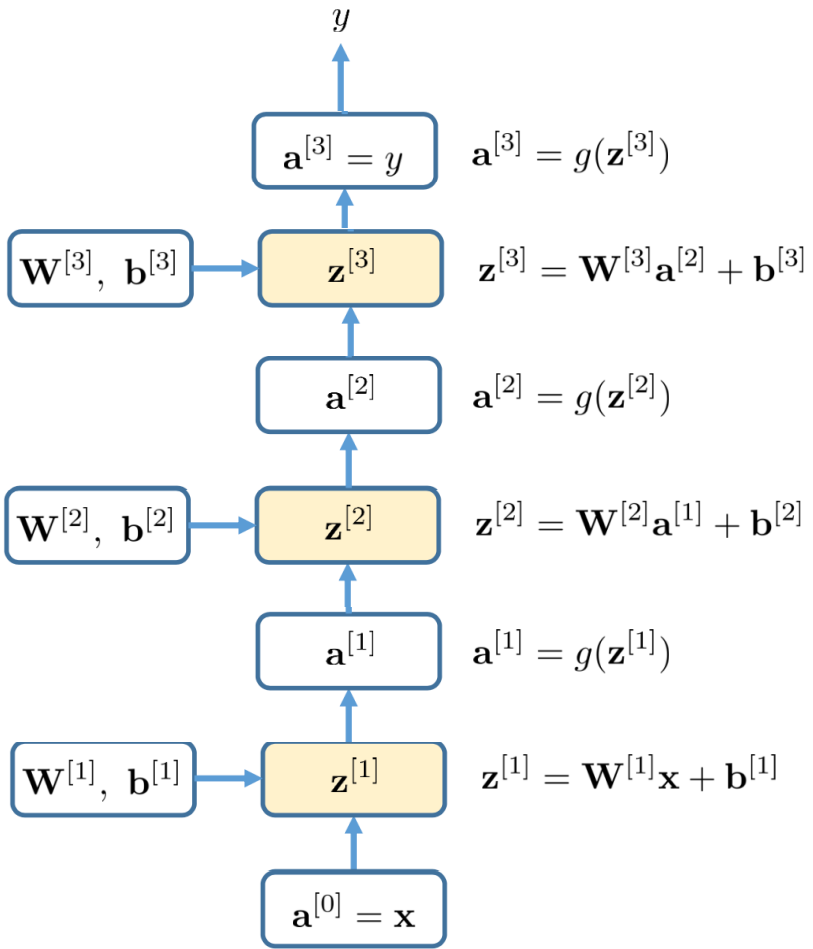
Back Propagation: Vectorization



$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]}), \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$\mathbf{a}^{[3]} = g(\mathbf{z}^{[3]}), \quad \mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{a}^{[2]} + \mathbf{b}^{[3]}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$

Partial Derivatives:

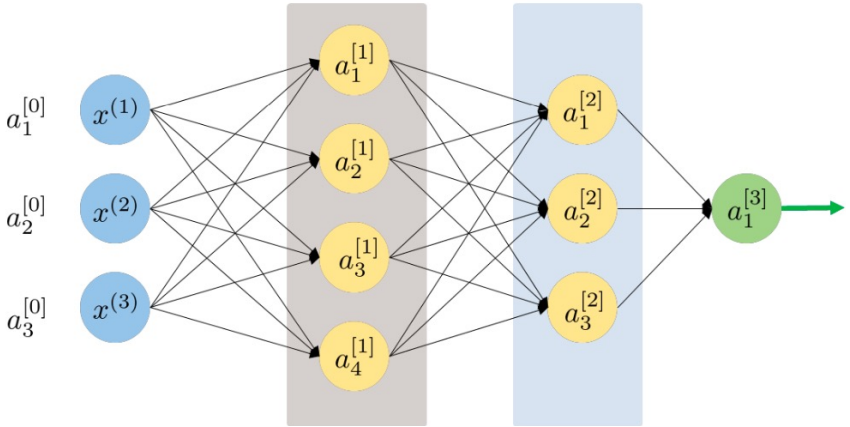
$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$



Back Propagation: Vectorization

- We compute loss function $\mathcal{L}$ using forward pass.
- We update $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using gradient descent as:

$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \qquad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$

Partial Derivatives:

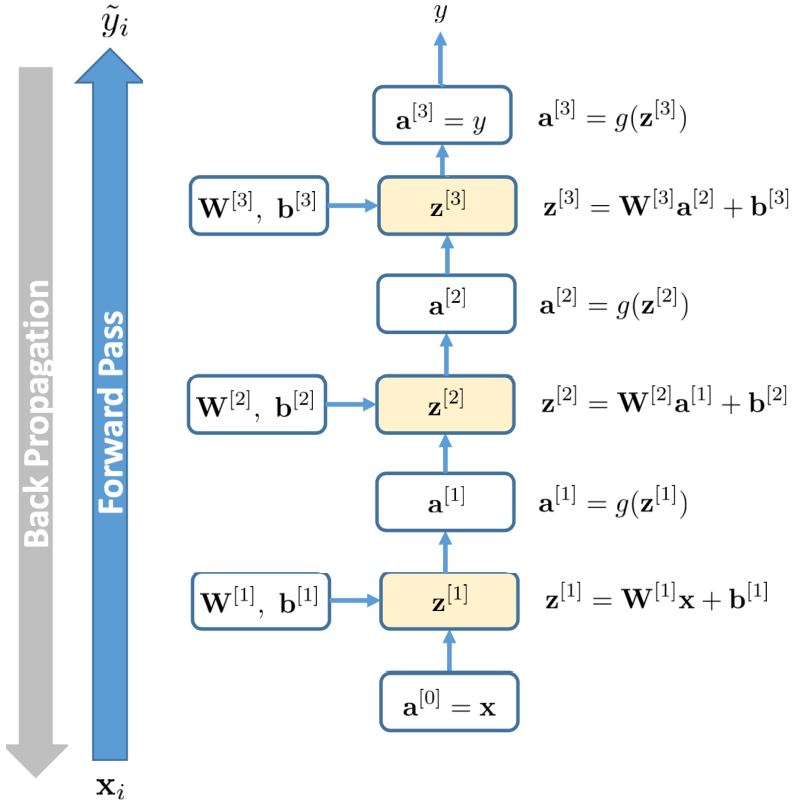
$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \qquad \frac{\partial \mathcal{L}}{\mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\mathbf{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$



Neural Networks In Keras

```
from keras.models import Sequential
from keras.layers import Dense

# built model
model = Sequential([
Dense(32, activation='relu', input_shape=(10,)),
Dense(32, activation='relu'),
Dense(1, activation='sigmoid'),
])

# compile your model
model.compile(optimizer='sgd',
loss='binary_crossentropy',
metrics=['accuracy'])
```

Neural Networks In Keras

```
# now train your model using fit
hist = model.fit(X_train, Y_train,
batch_size=32, epochs=100,
validation_data=(X_val, Y_val))
```

```
# evaluate your model using model.evaluate
model.evaluate(X_test, Y_test)[1]
```