Your First MLP

Recitation 1, part 1
Fall 2022

Overview

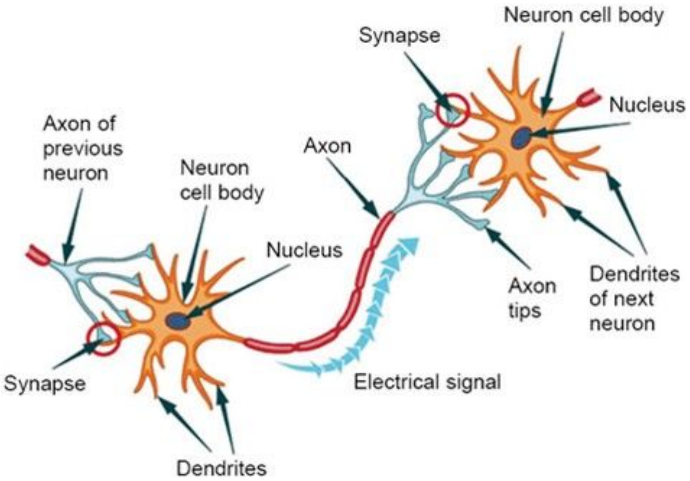
- Neural Networks
- Perceptrons
- Multilayer perceptrons
  - Forward Pass
  - Backpropagation
  - Update Weights

Neural Networks

- The brain, made up of connected neurons, are the inspirations for artificial neural networks.

Neural Networks

- A neuron is a node with many inputs and one output.
- A neural network consists of many interconnected neurons -- a “simple” device that receives data at the input and provides a response.
- Information are transmitted from one neuron to another by electrical impulses and chemical signals.



Perceptrons

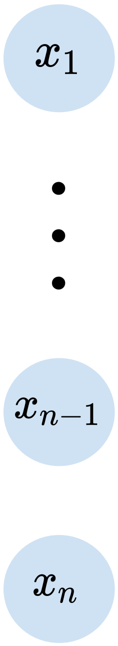
- Perceptron is a single layer neural network.

Perceptrons

- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.

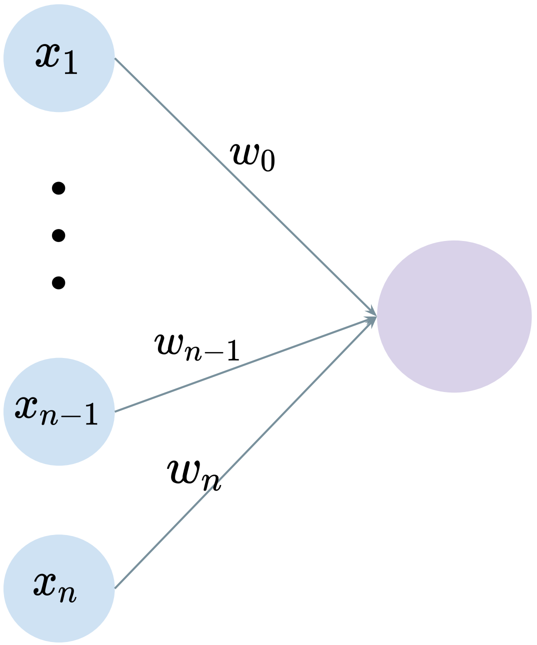
Perceptrons

- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.
    - Input values



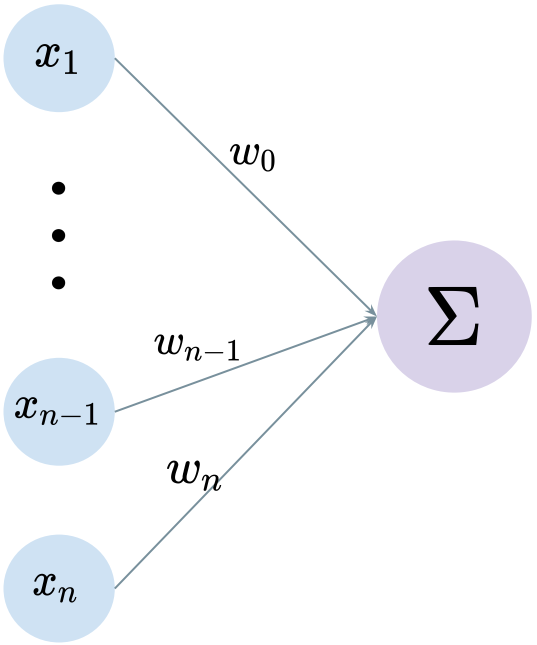
Perceptrons

- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.
    - Input values
    - Weights



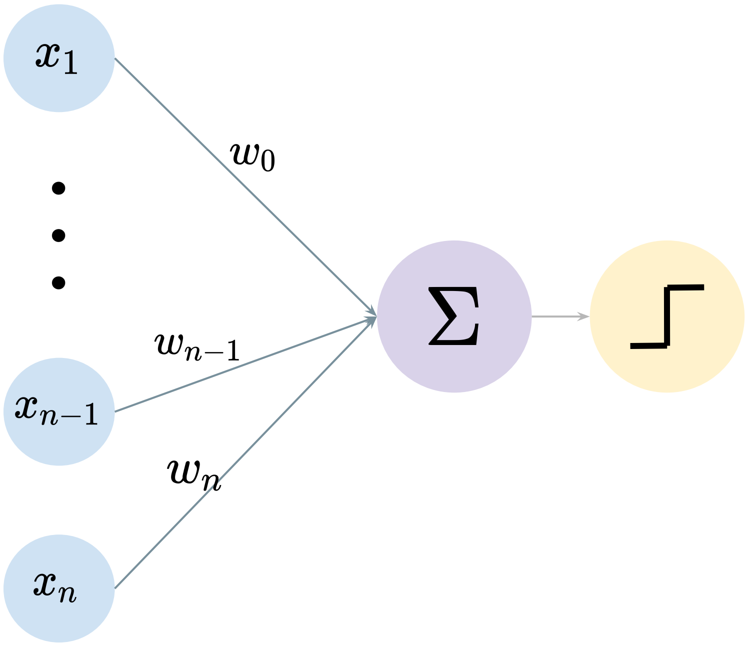
Perceptrons

- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.
  - Input values
  - Weights
  - Weighted sums



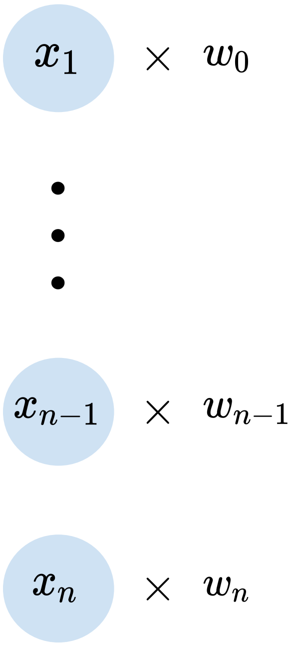
Perceptrons

- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.
  - Input values
  - Weights
  - Weighted sums
  - Threshold / Activation functions



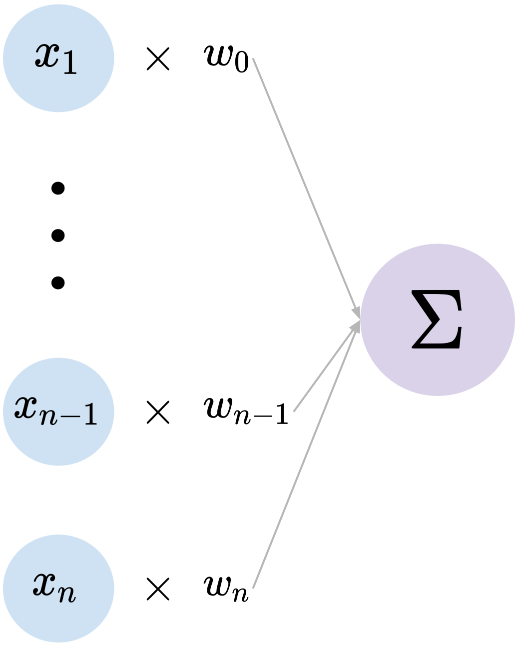
Perceptrons

- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.
- The perceptron works on the following steps:
    - Multiply all inputs with their weights



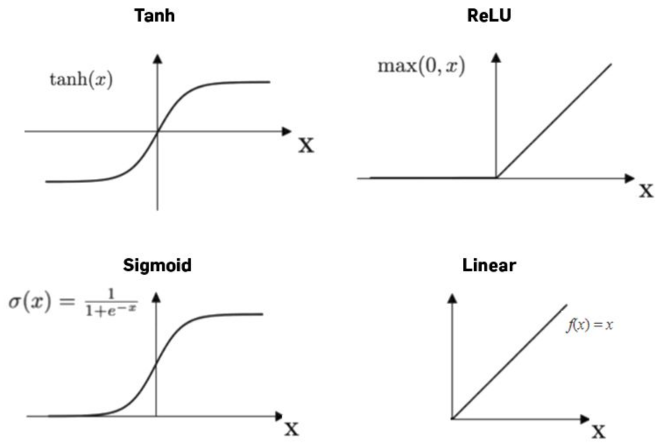
Perceptrons

- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.
- The perceptron works on the following steps:
  - Multiply all inputs with their weights
  - Add all multiplies values → weighted sum



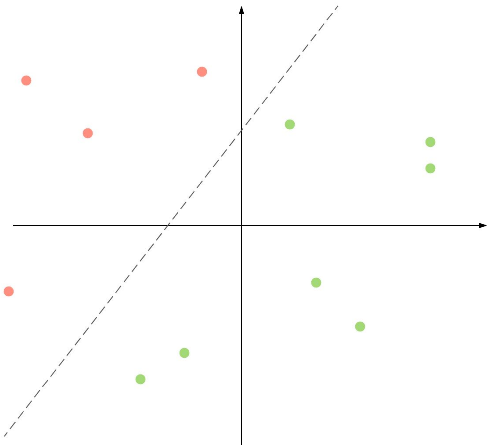
Perceptrons

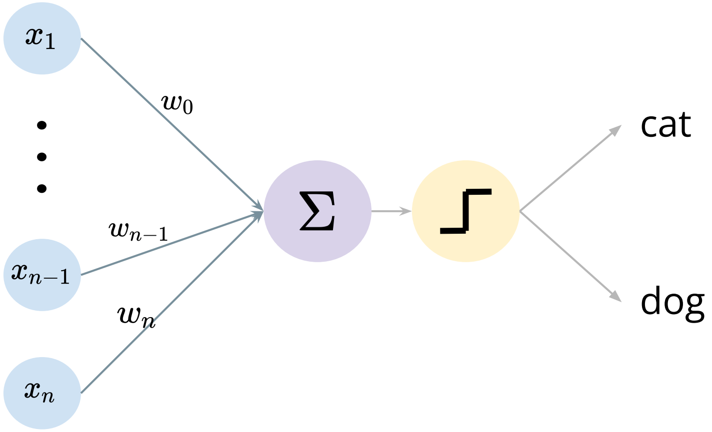
- Perceptron is a single layer neural network.
- The perceptron consists of 4 parts.
- The perceptron works on the following steps:
    - Multiply all inputs with their weights
    - Add all multiplies values → weighted sum
    - Apply the weighted sum to activation function



Perceptrons

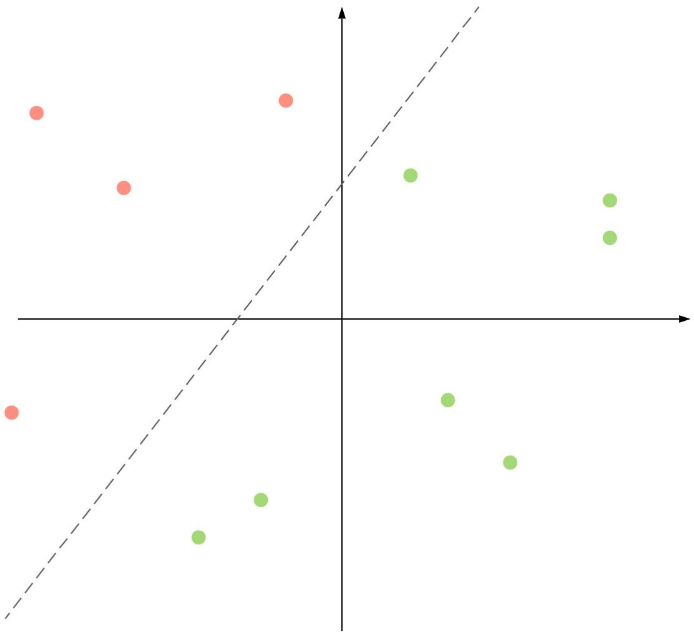
- Perceptron is usually used to classify the data into two parts -- **Linear Binary Classifier**.





Perceptrons

- Perceptron is usually used to classify the data into two parts -- **Linear Binary Classifier**.
    - **Weights** shows the **strength** of the particular node.
    - **Activation functions** are used to map the input between the required values



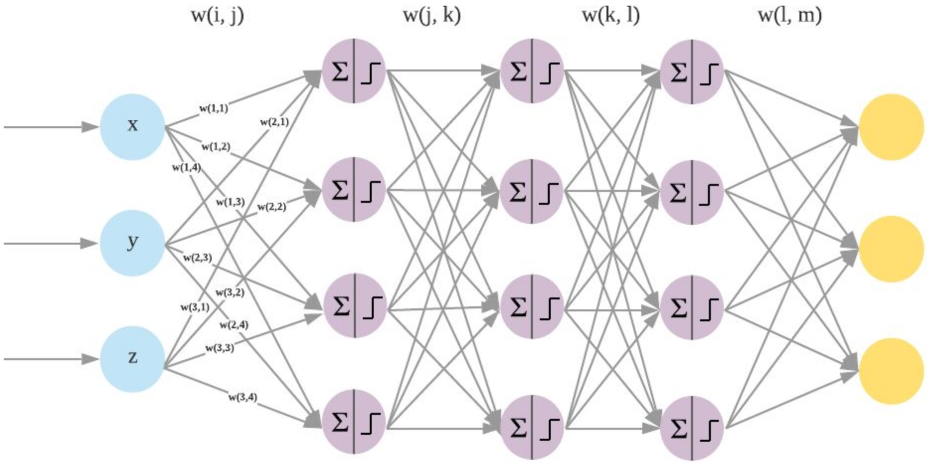
Multilayer Perceptrons

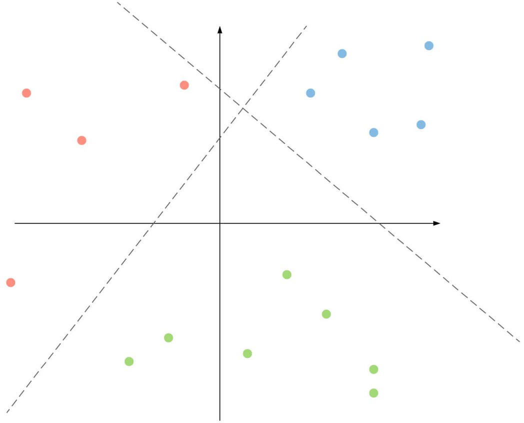
What if we want to be able to distinguish between more classes?

Multilayer Perceptrons

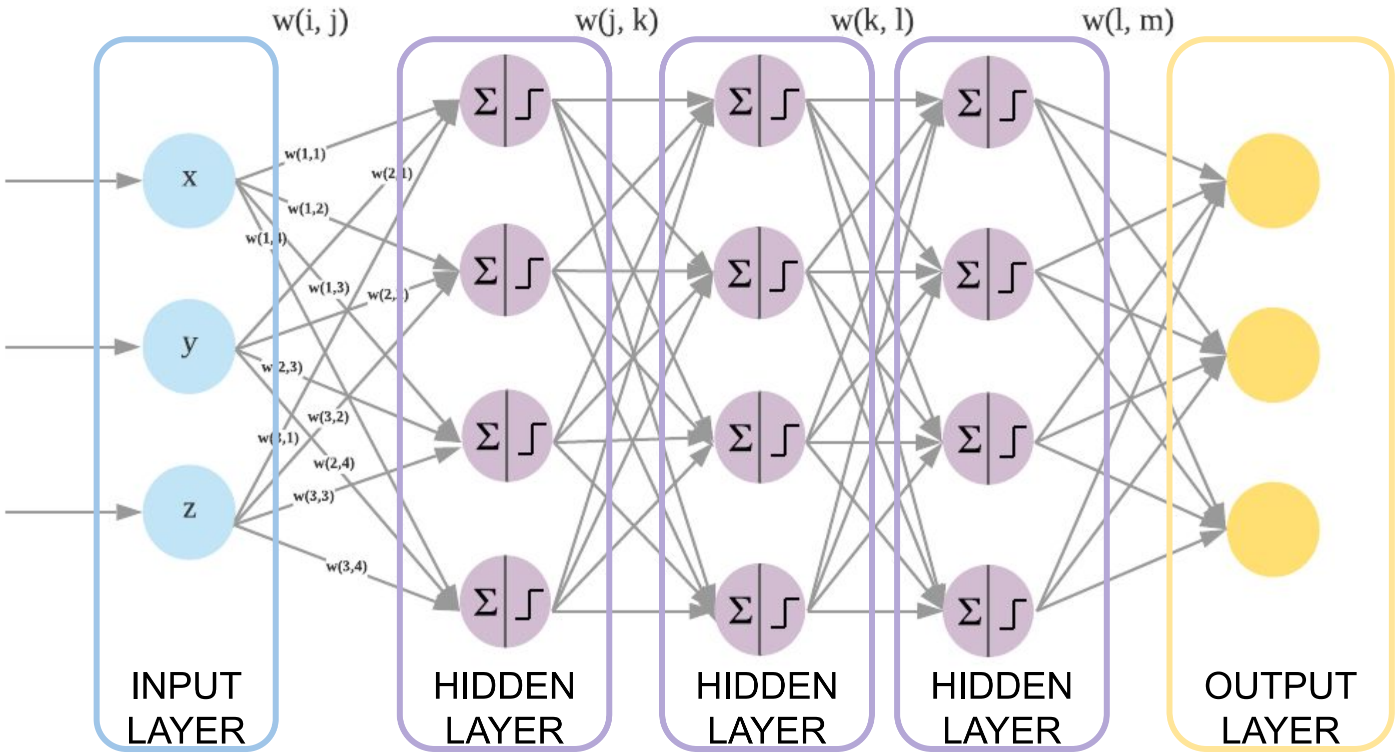
What if we want to be able to distinguish between more classes?

- Introduce more perceptrons !





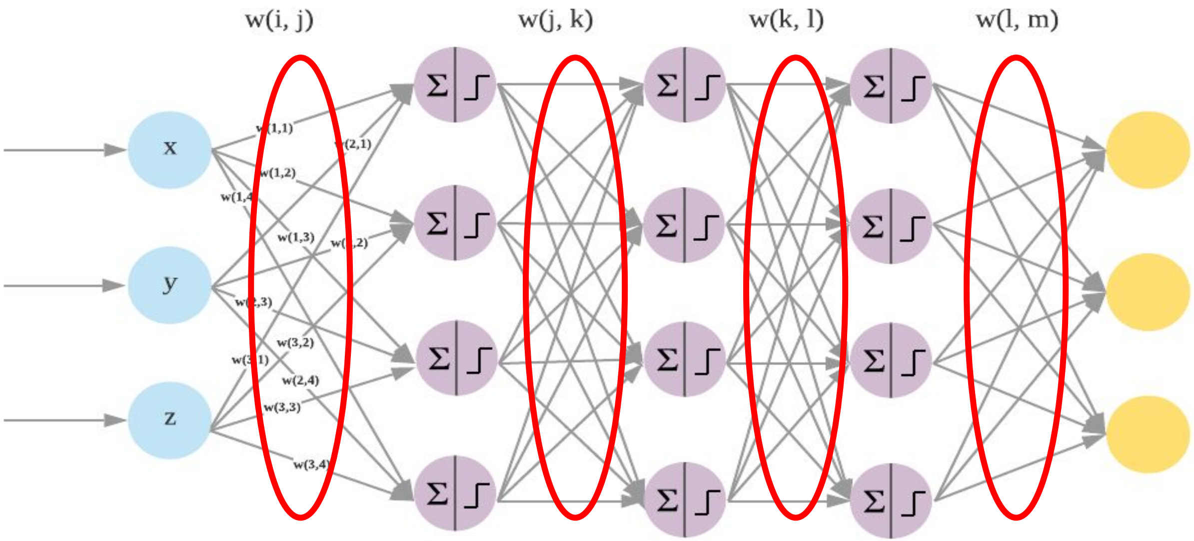
Multilayer Perceptrons



In order to correctly classify things, the network must be **learned**.

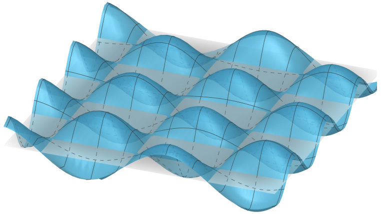
But first, **what** do we need to learn?

The parameters (or the weights)



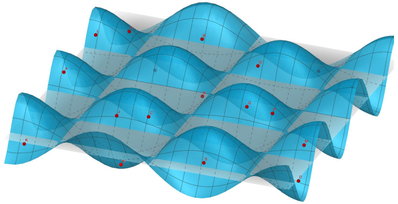
How do we learn?

➔ Actual Function that we are trying to model:



- Note: We don’t know the actual function.

➔ We only have several sample data points on this function.



➔ Our goal:

- **Estimate the function with the given samples.**

How do we learn?

- A measurement of **error**
  - How much off is the **network output** with respect to the **desired output**

For each sample

MLP

Network Output

$$Loss(W) = \frac{1}{N}\sum_i div(\boxed{f(X_i, W)}, d_i)$$

Desired Output

Number of samples

Divergence function

Sample value

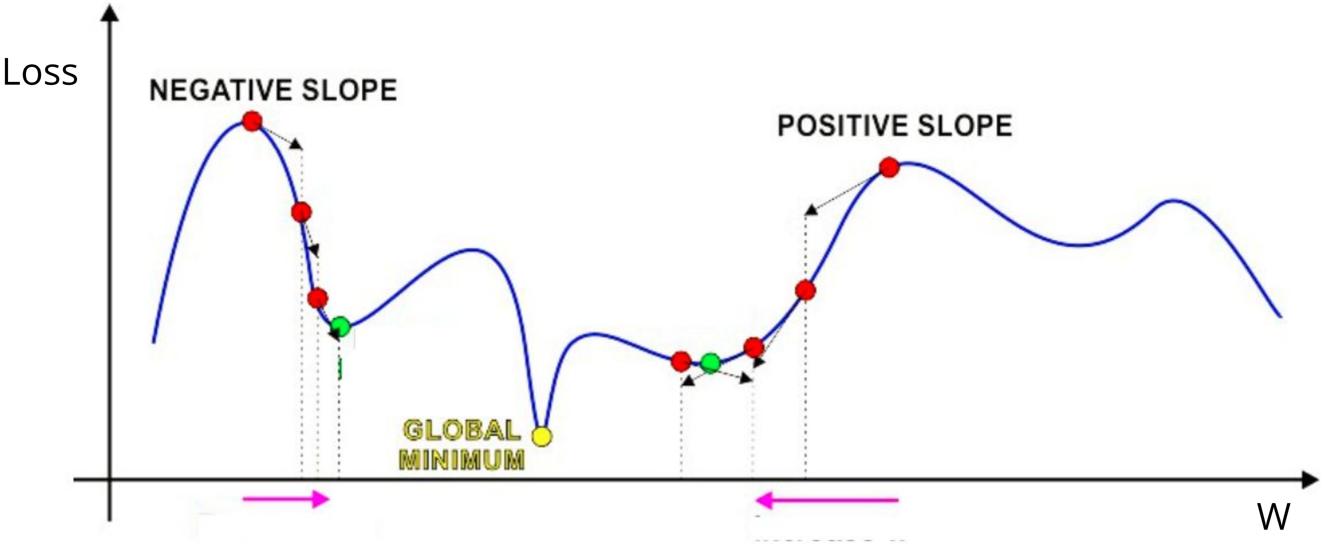
Current weights of estimated function

- Our goal (more specifically):
  - Minimize the loss

$$\hat{W} = \arg\min_{W} \; Loss(W)$$

How do we learn?

➔ Gradient Descent

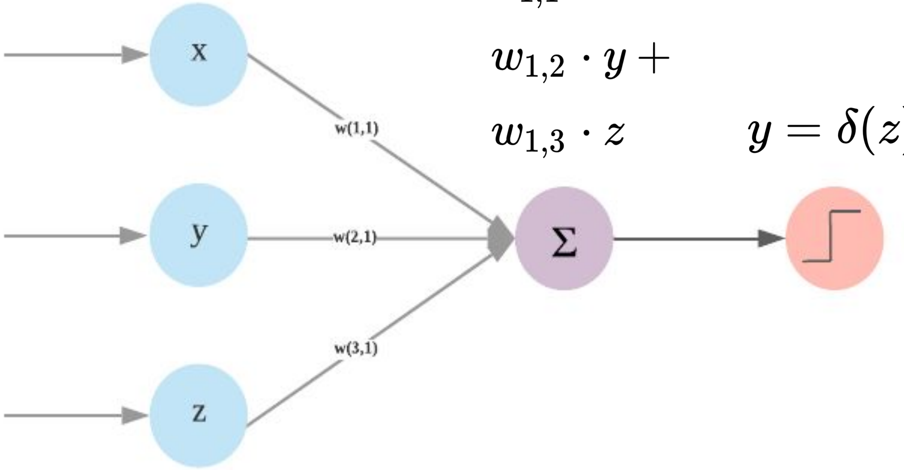


Forward Pass

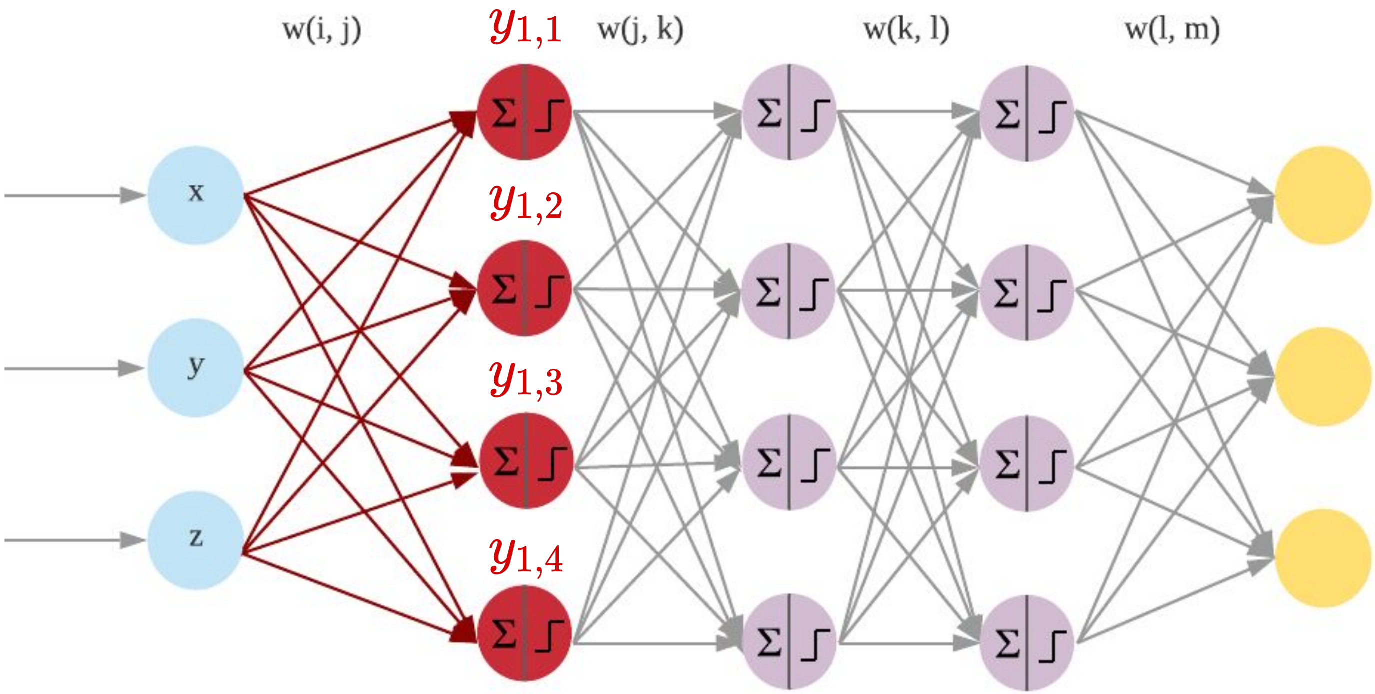
- For each single perceptron

$$z = w_{1,1} \cdot x + w_{1,2} \cdot y + w_{1,3} \cdot z$$

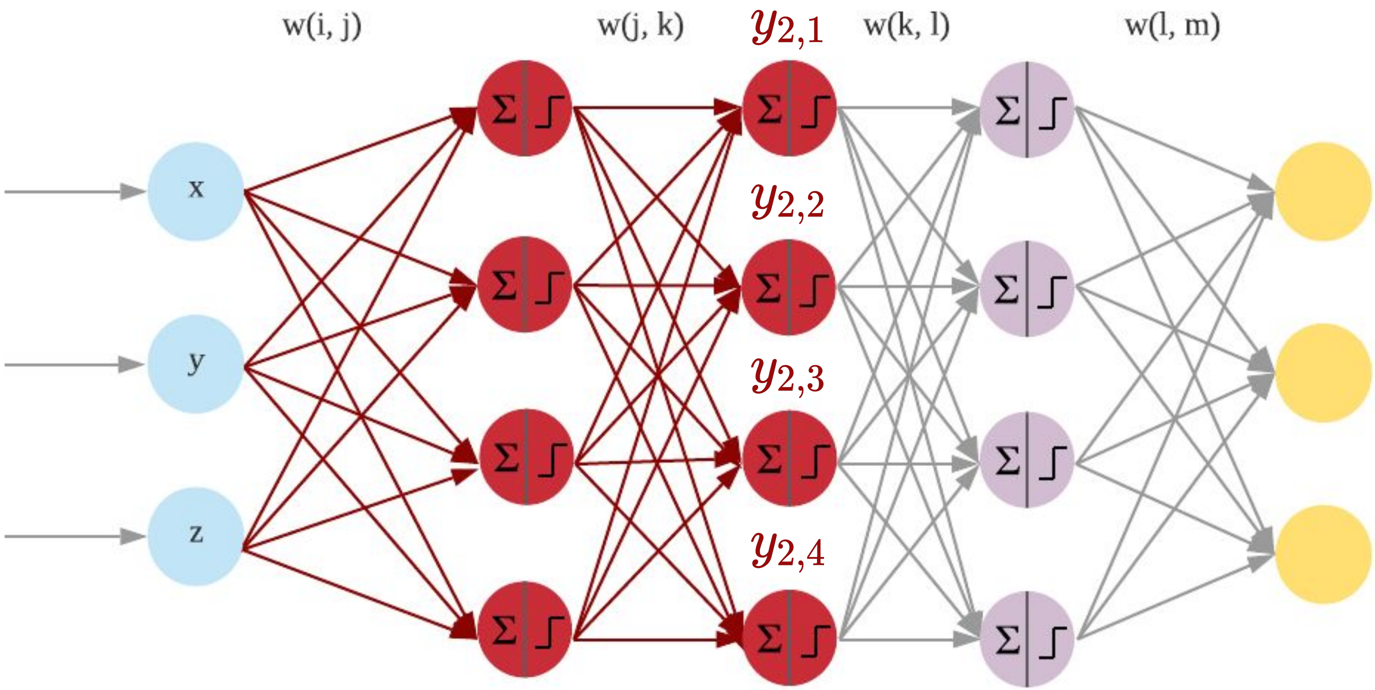
$$y = \delta(z)$$



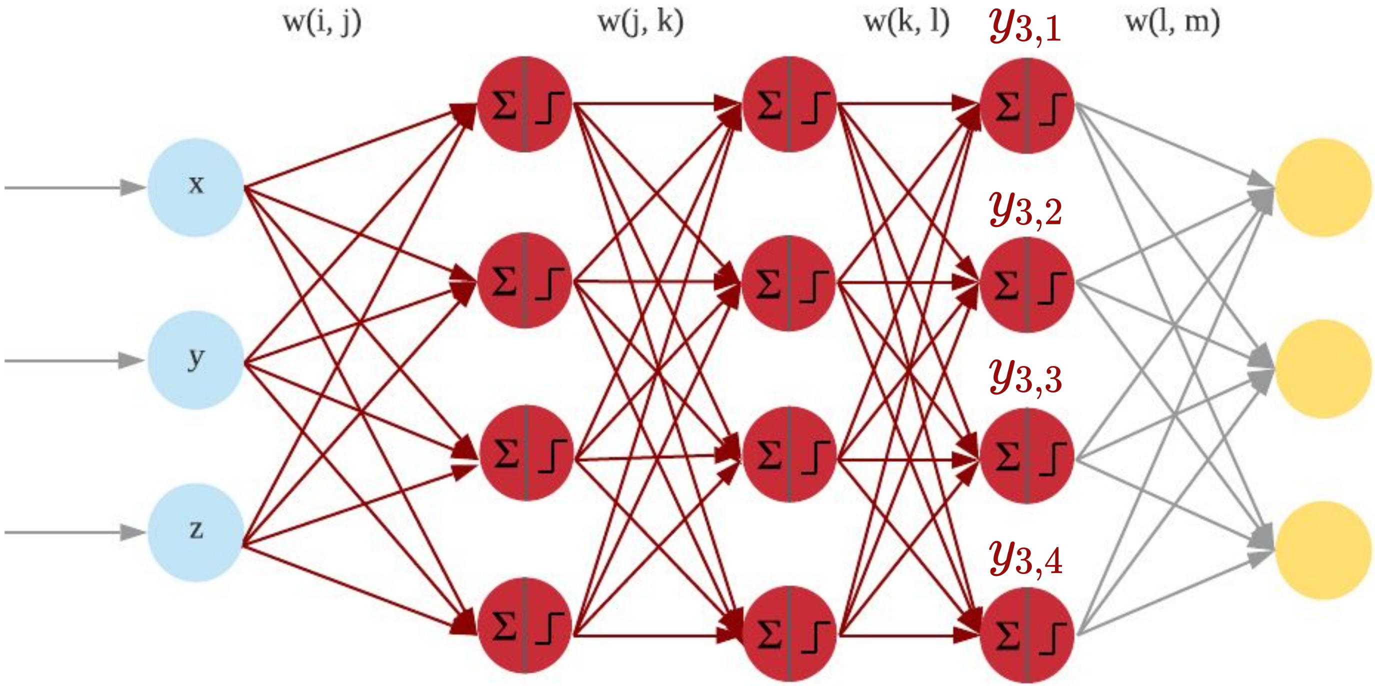
Forward Pass



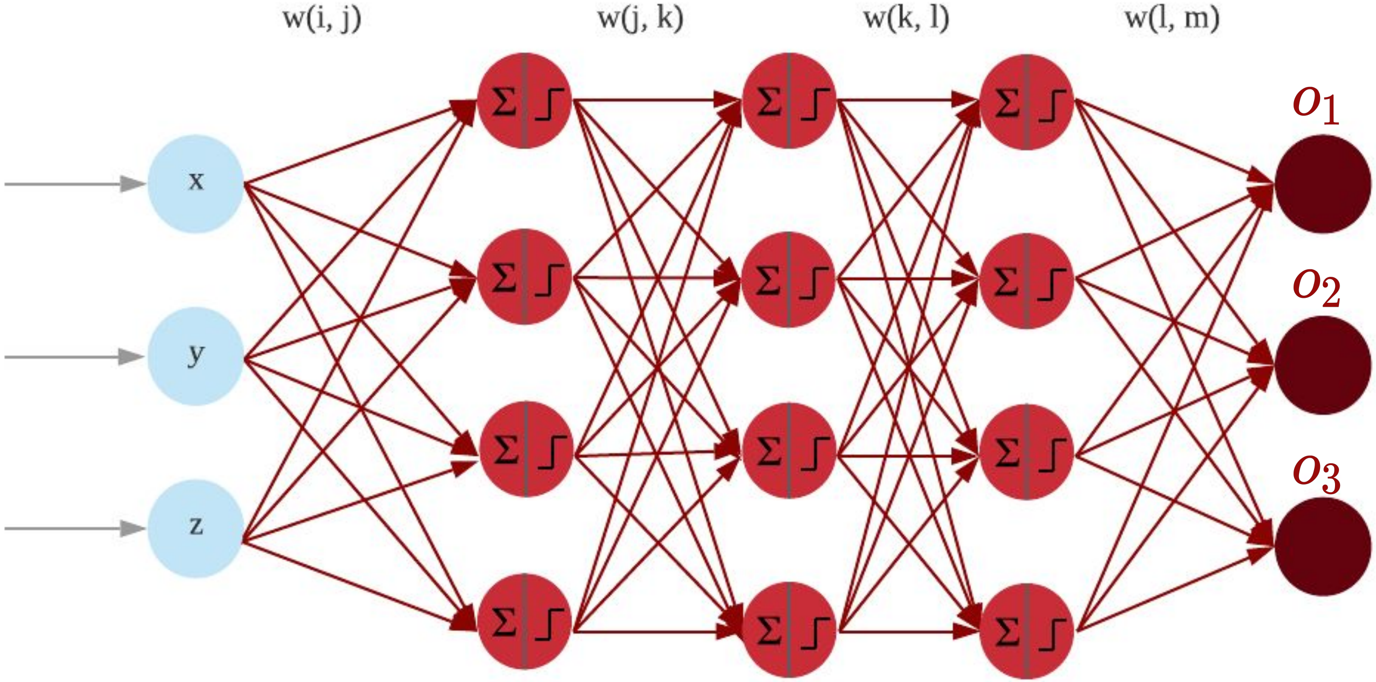
Forward Pass



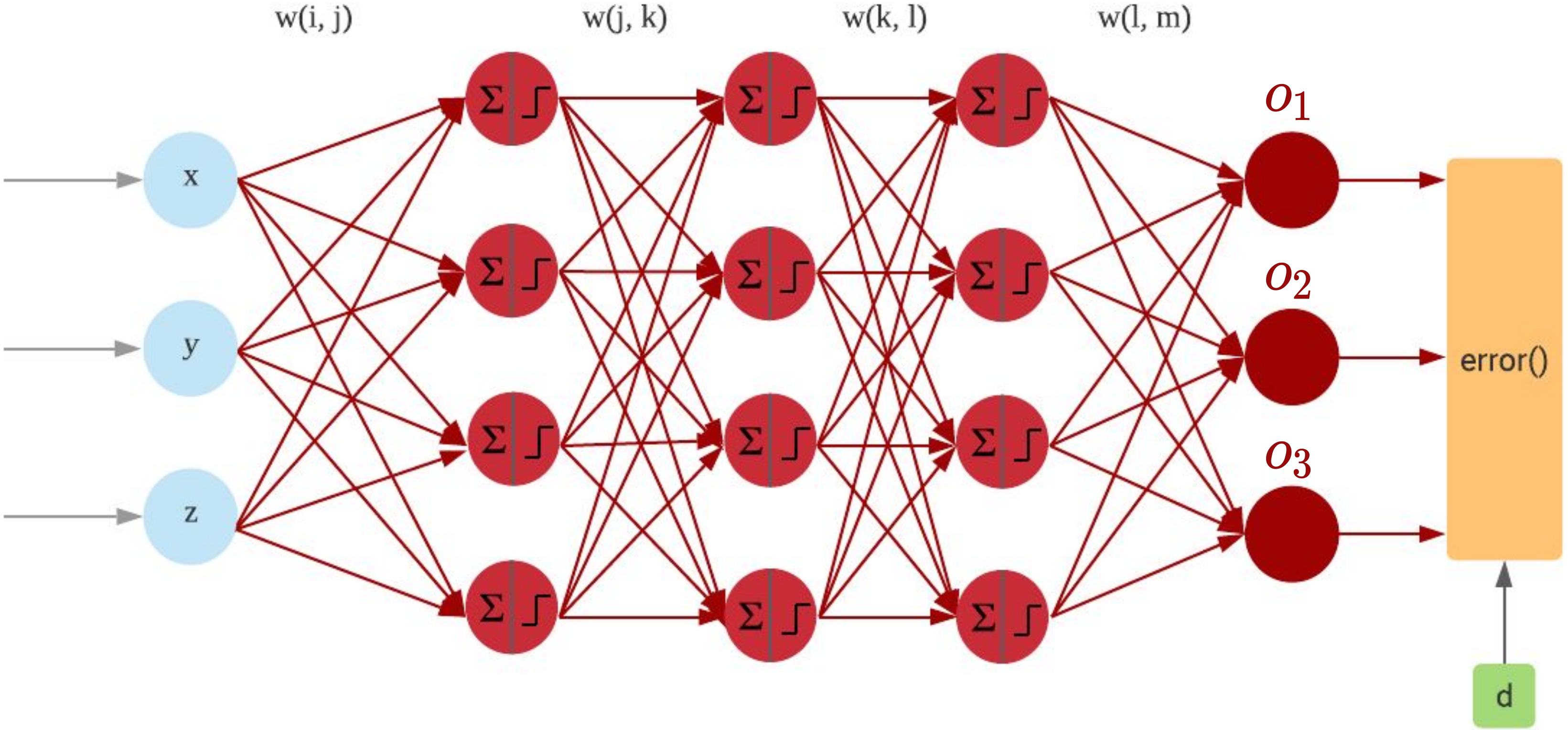
Forward Pass



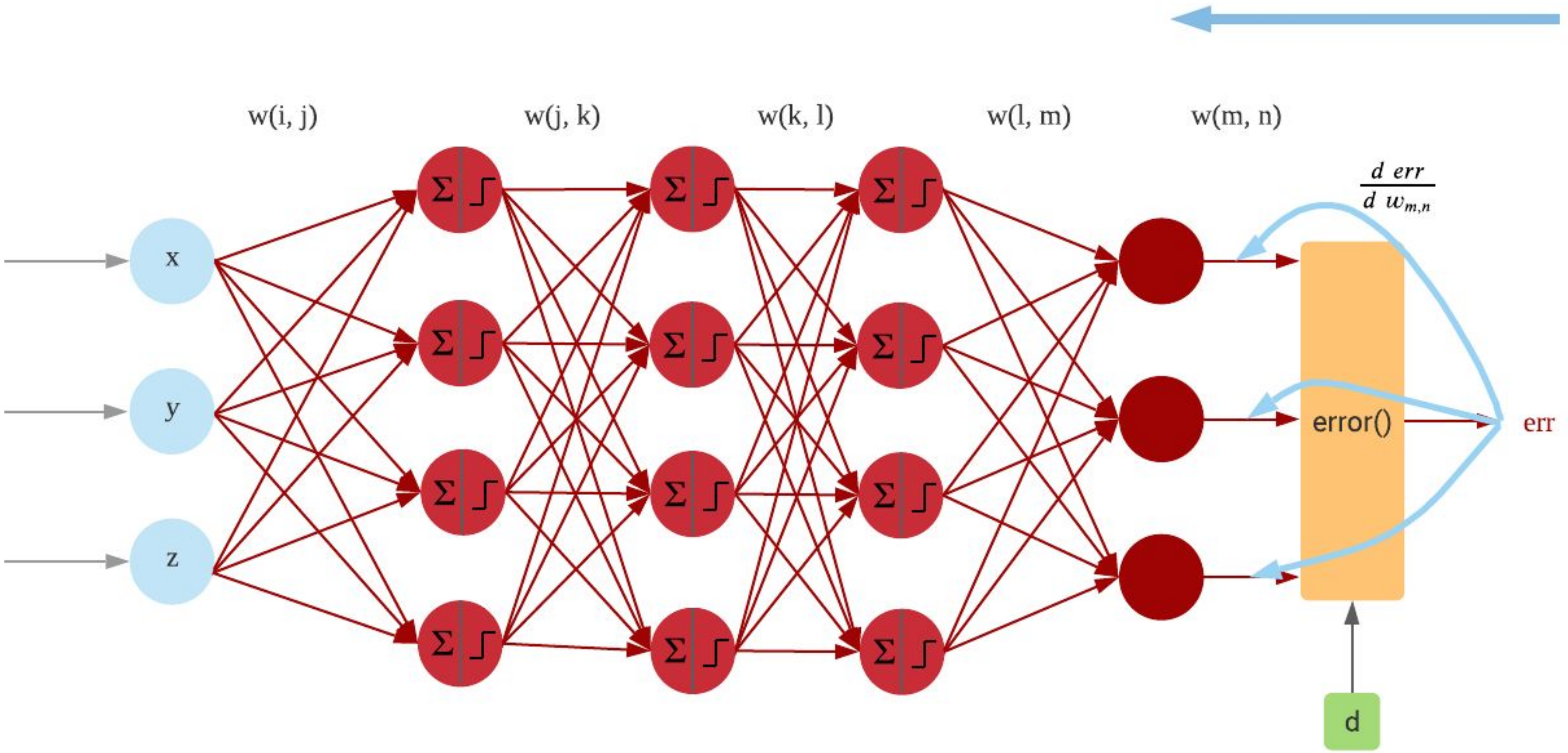
Forward Pass



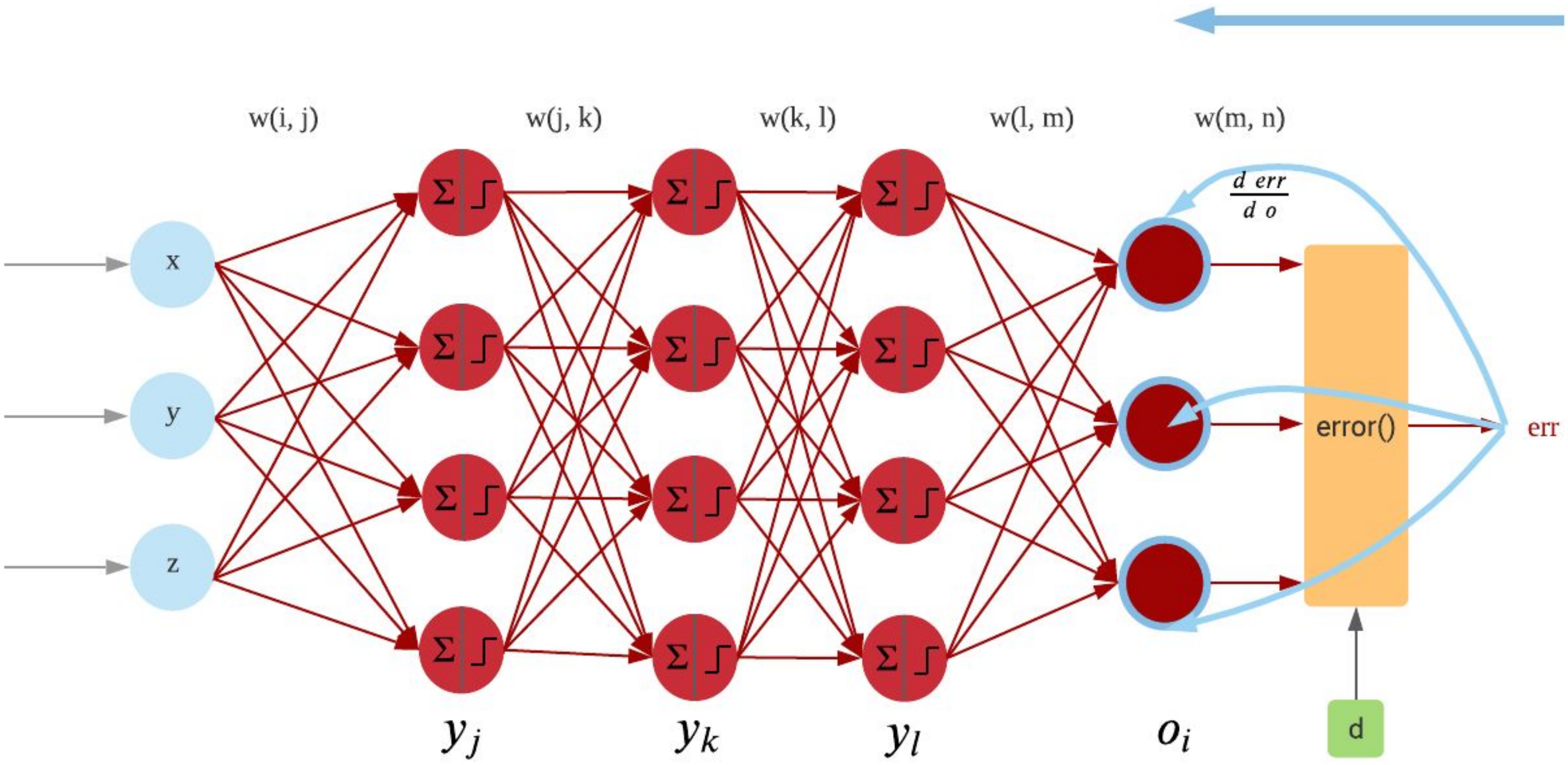
Error



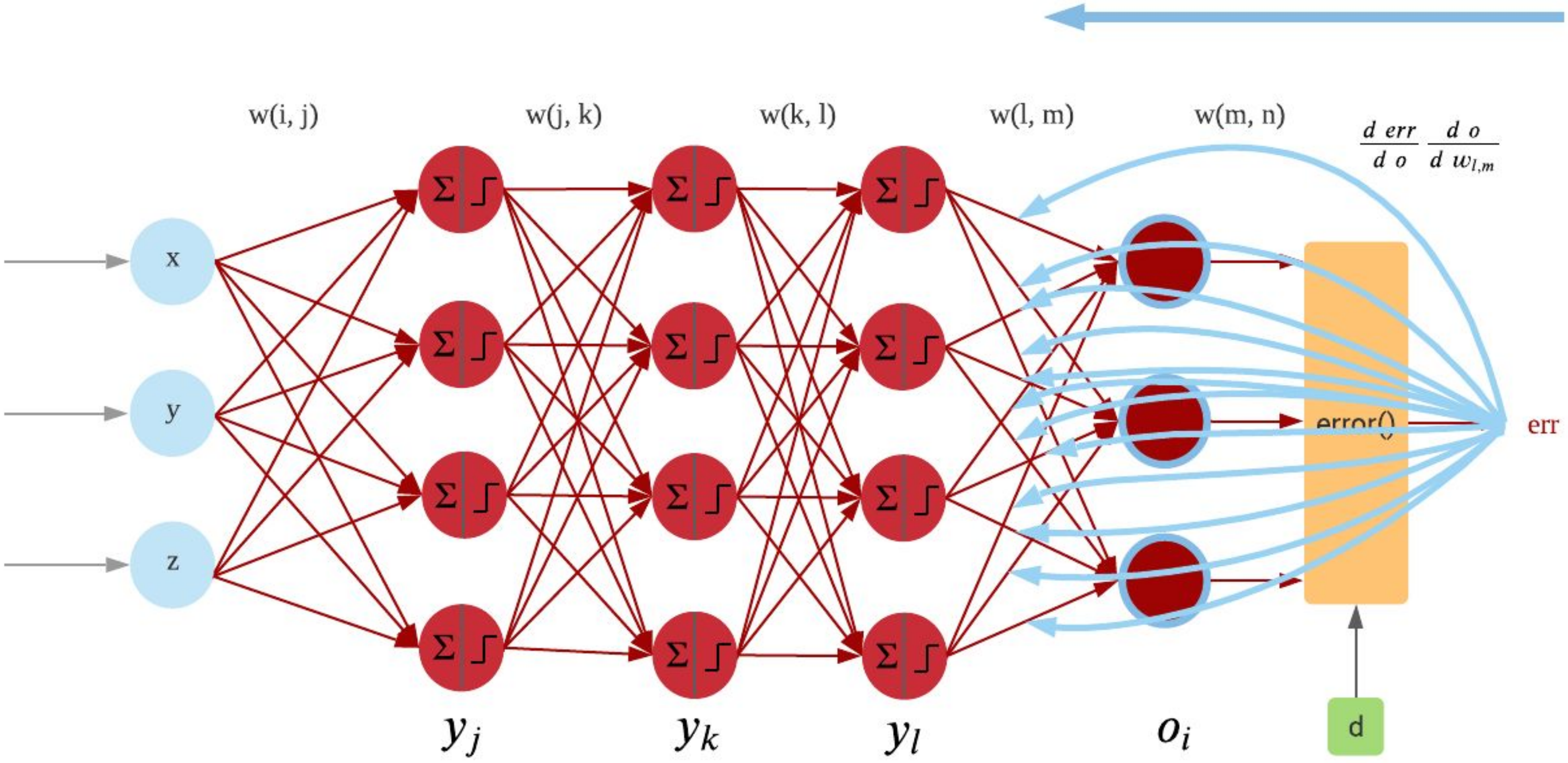
Backpropagation



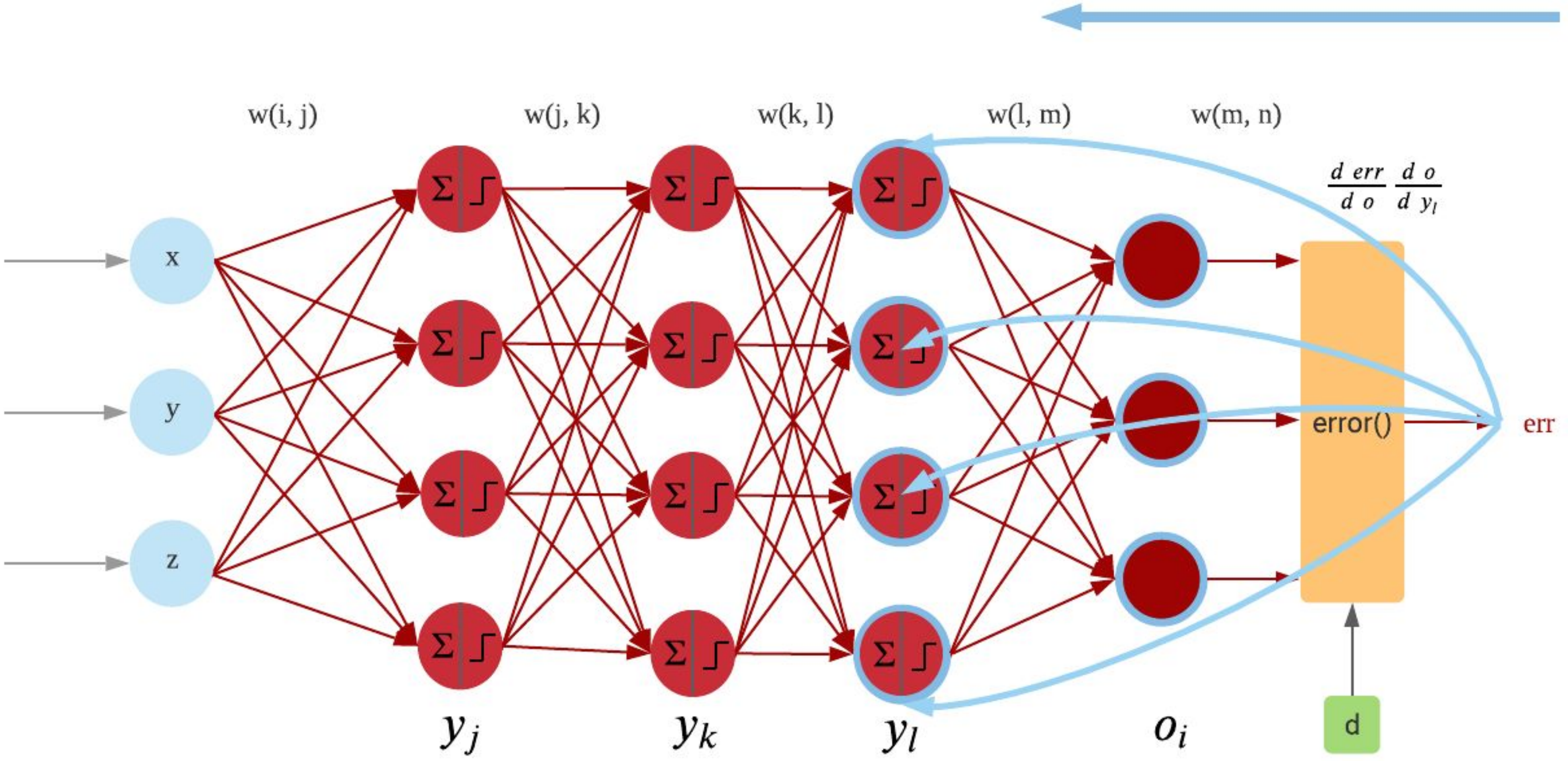
Backpropagation



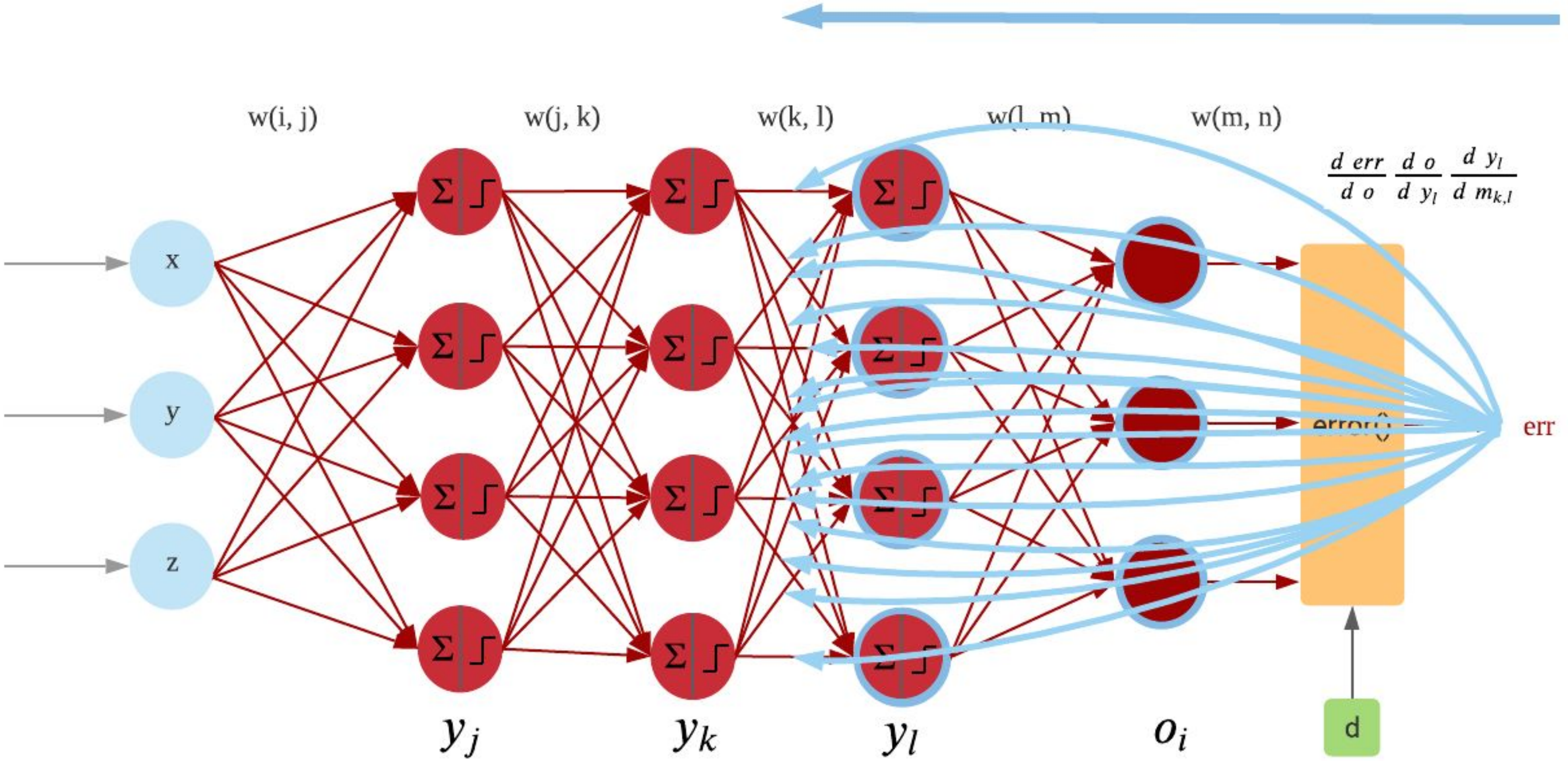
Backpropagation



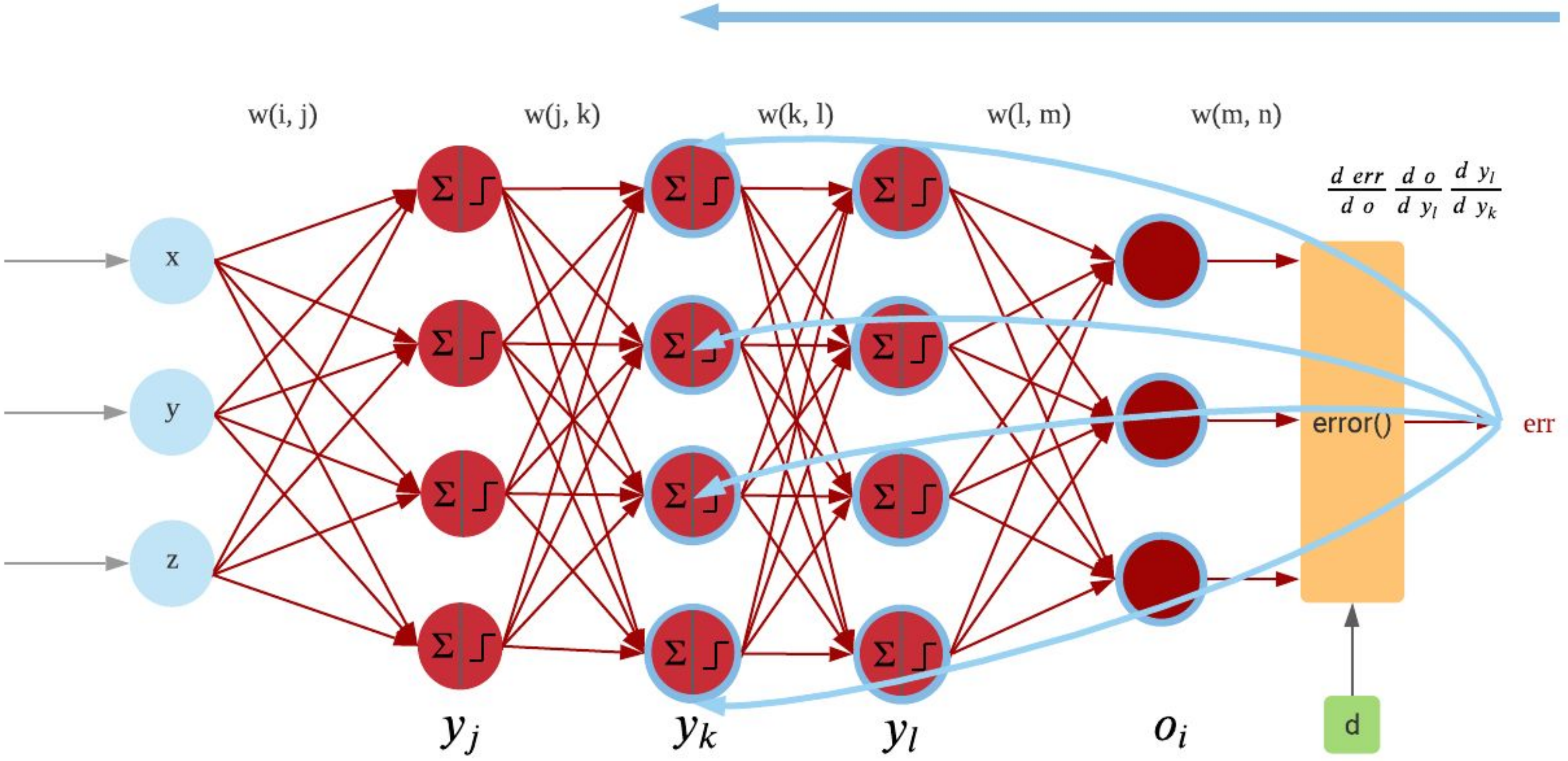
Backpropagation



Backpropagation



Backpropagation



Backpropagation

……

Backpropagation

All gradients of weights w.r.t error are calculated!

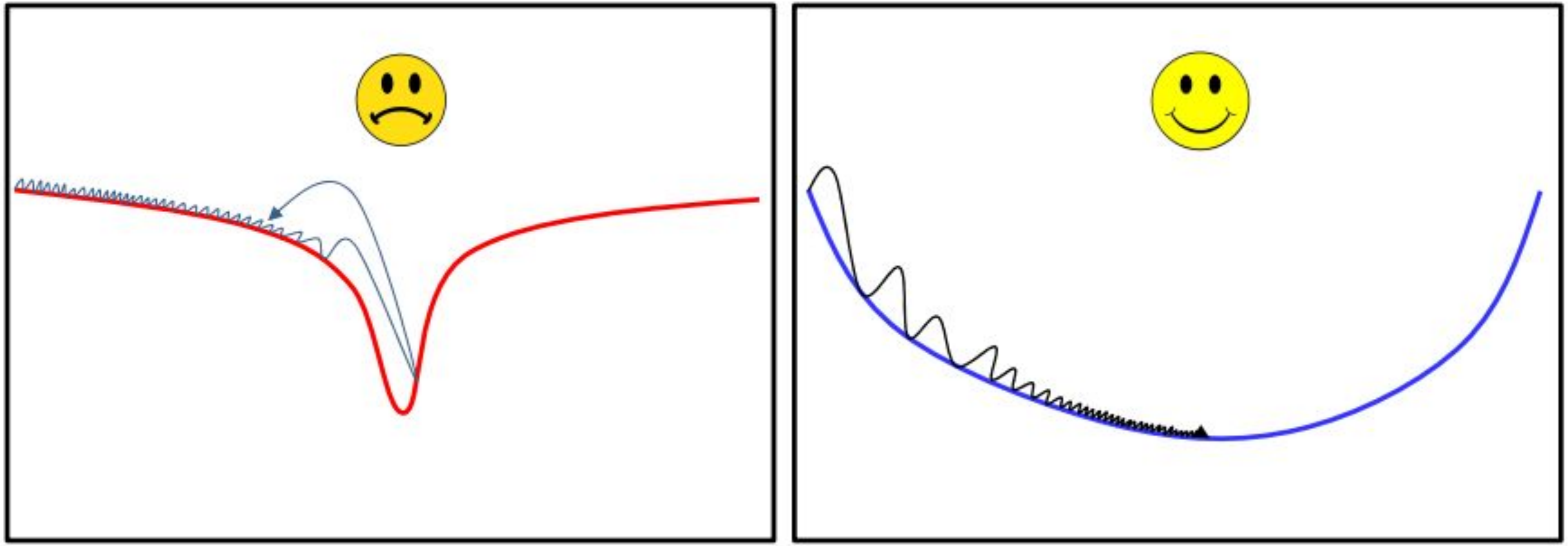
Update Weights

$$W \leftarrow W - \eta \cdot \nabla_W Loss(W)$$

learning rate

gradient

What should be the learning rate?



https://deeplearning.cs.cmu.edu/F21/document/slides/lec8.optimizersandregularizers.pdf

Optimizers

Gradient Descent:

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta J(\theta)$$

Momentum (http://proceedings.mlr.press/v28/sutskever13.pdf):

$$m_{t+1} = \mu \cdot m_t + \alpha \cdot \nabla_\theta J(\theta)$$
$$\theta_{t+1} = \theta_t - m_{t+1}$$

Adagrad (https://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf):

$$g \leftarrow \nabla_\theta J(\theta)$$
$$r \leftarrow r + g^2$$
$$\triangle\theta \leftarrow \frac{\delta}{\sqrt{r+\epsilon}} \cdot g$$
$$\theta \leftarrow \theta - \triangle\theta$$

Optimizers (Cont')

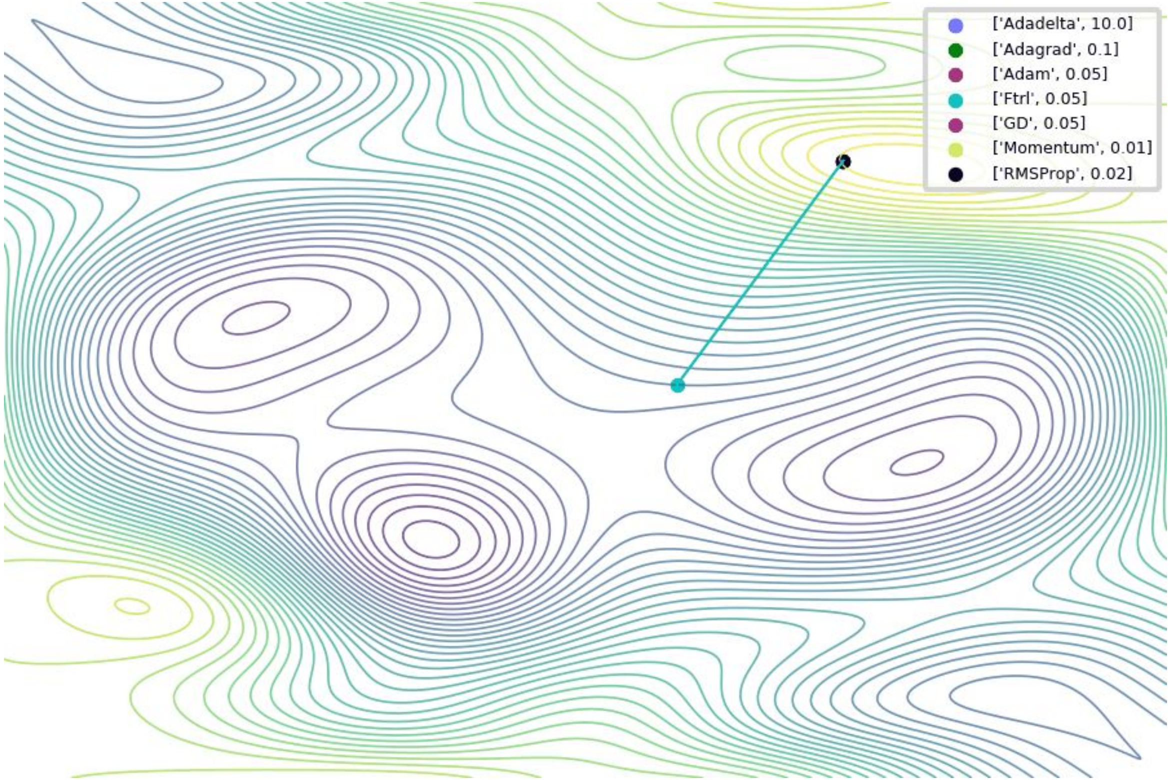
Adam (https://arxiv.org/pdf/1412.6980.pdf):

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$
$$\hat{v}_t = v_t / (1 - \beta_2^t)$$

$$\theta_t = \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t + \epsilon})$$

Visualization



https://github.com/Jaewan-Yun
/optimizer-visualization

Some fun with TF Playground