Multilayer Perceptron

MGTF 495

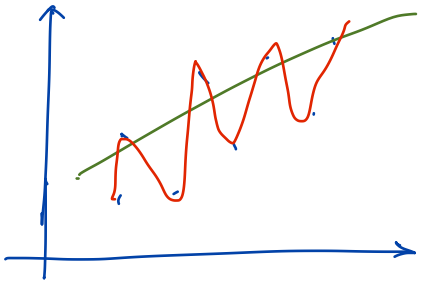
Class Outline

- Multi-layer Perceptron
  - ***Hands-On / Self – practice***
- Convolutional Neural Networks
  - ***Hands-On / Self – practice***
- Recurrent Neural Networks
  - ***Hands-On / Self – practice***

Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

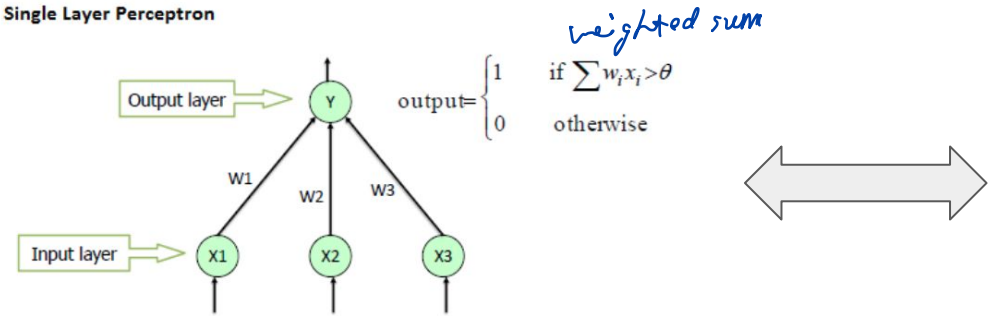
regression

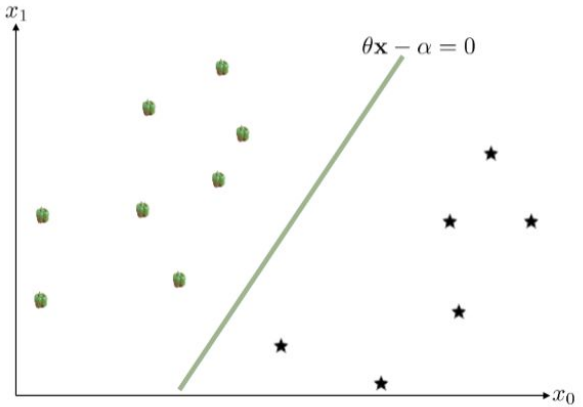


red line is overfitting
not capture the feature

Perceptron

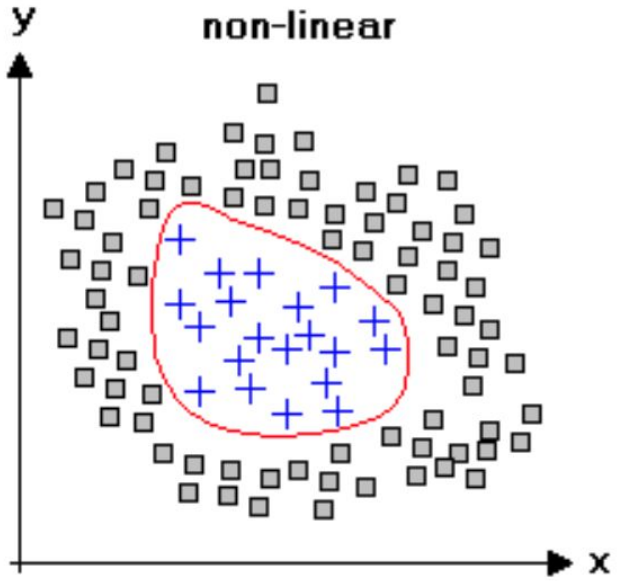
- Given input data with n features $X = (x_1, x_2)^T$
- Compute weighted sum of input.
- The sum is compared against a threshold.
- It **fires**(classifies as **1**), if the sum is greater than threshold





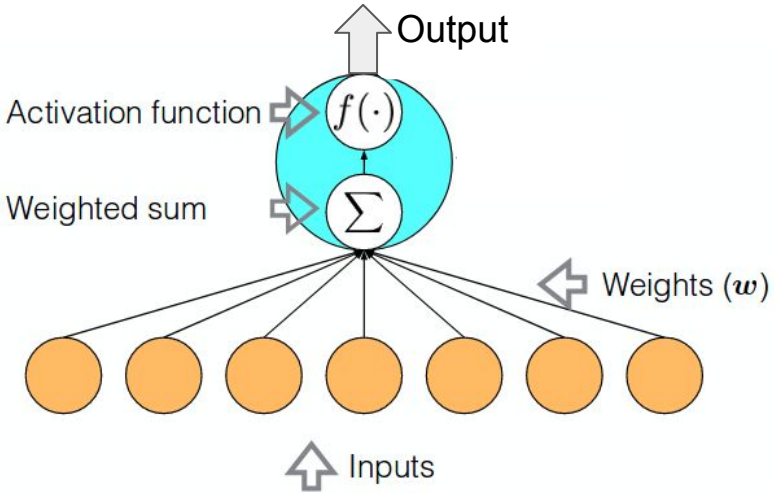
Perceptron: Limitation

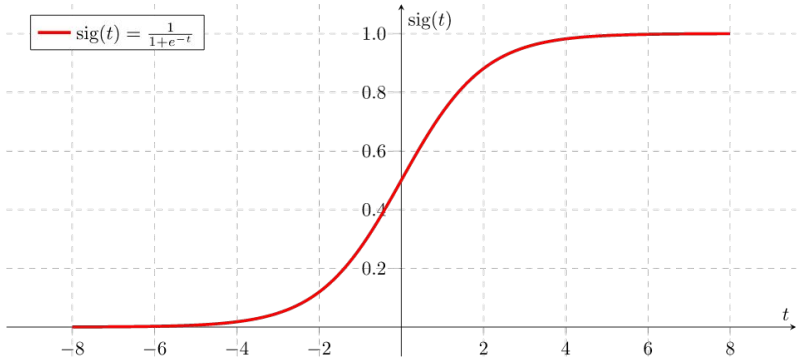
- Cannot classify **non-Linearly separable** data.
- Solution - Introduce **nonlinearity** in perceptron



Perceptron & Activation function

- We need to incorporate **non-Linearity**.
- **Activation function** introduces nonlinearity which increases the representation capability of network.





Activation Functions

- Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$y' = y(1 - y)$$

Squeezes input in range (0,1)

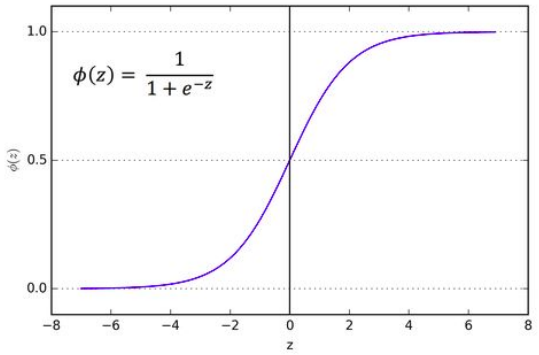


Fig: Sigmoid Function

- Tanh

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$f(x) = tanh(x) = 2\ sigmoid(2x) - 1$$

Squeezes input in range (-1,1)

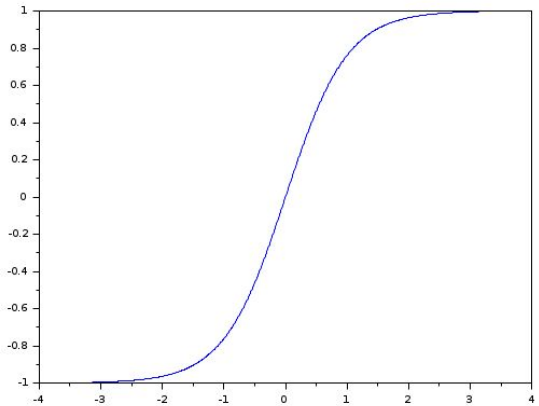


Fig: Tanh function

Activation Functions

- ReLU (Rectified Linear Unit)

$$f(x) = max(0, x)$$

  - Computationally efficient
  - Most widely used as of today

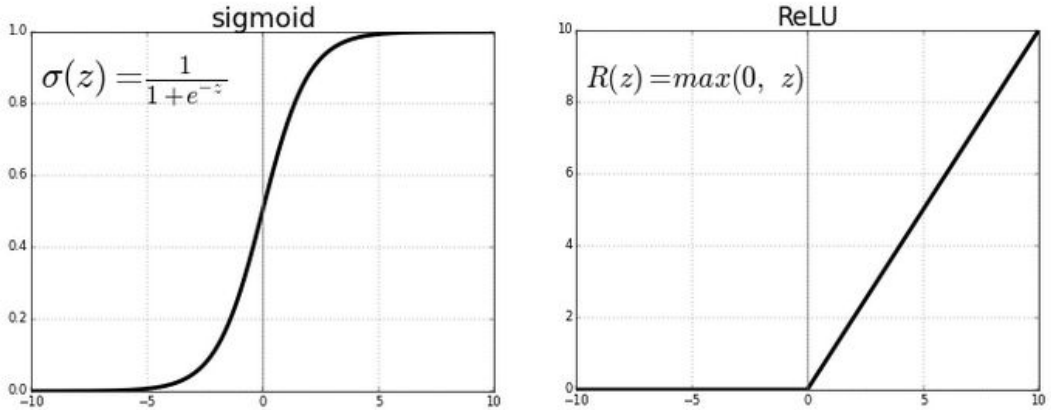


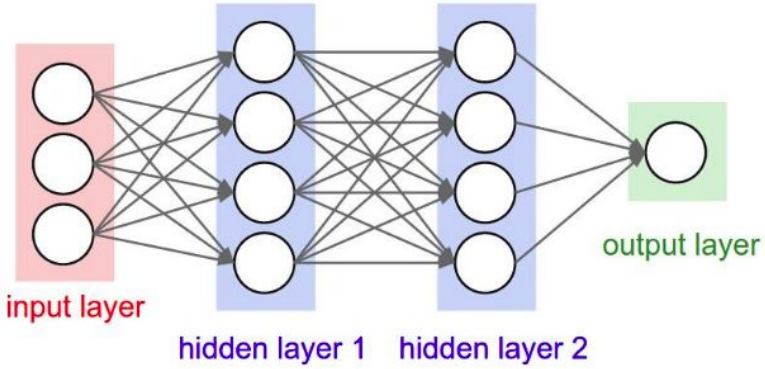
Fig: ReLU v/s Logistic Sigmoid

Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Multilayer Perceptron(Neural Network)

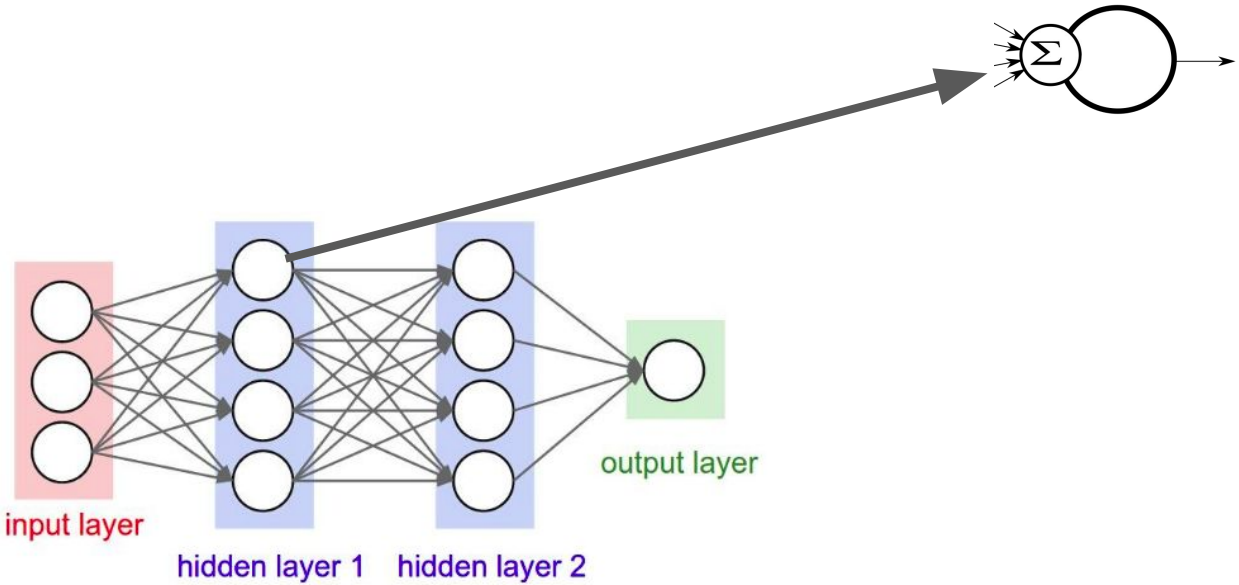
- The network is organized in terms of layers.



Multilayer Perceptron(Neural Network)

- The network is organized in terms of layers.
- Each layer has:
  - **Neurons**.

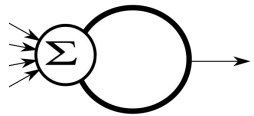
Artificial Neuron



Multilayer Perceptron(Neural Network)

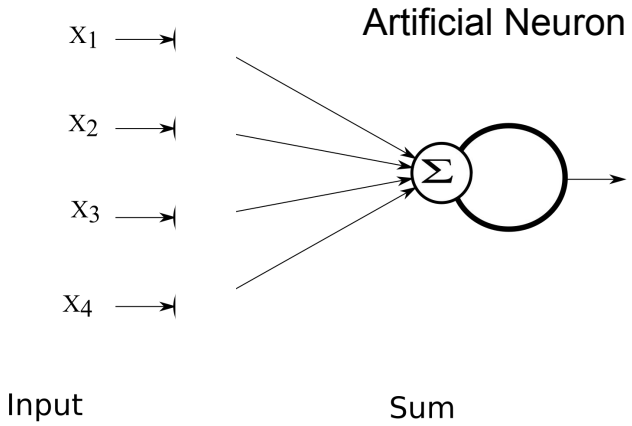
- The network is organized in terms of layers.
- Each layer has:
  - **Neurons**.

Artificial Neuron



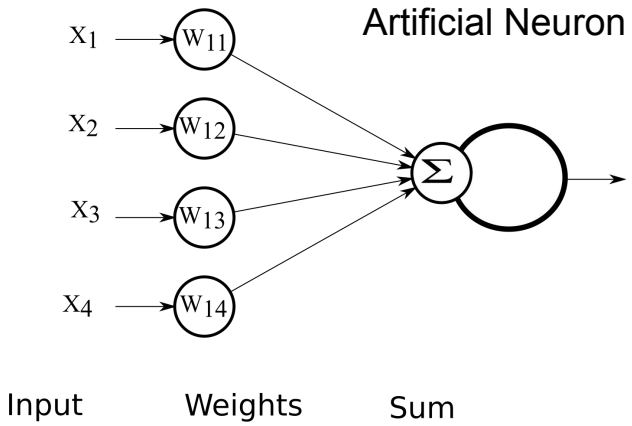
Multilayer Perceptron(Neural Network)

- The network is organized in terms of layers.
- Each layer has:
  - **Neurons**.
  - Neurons have **inputs**



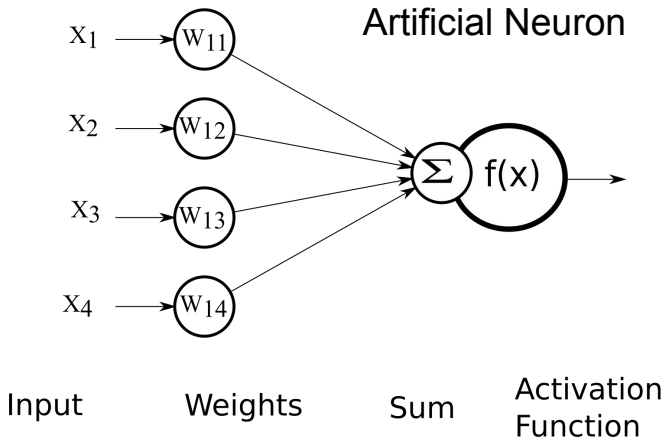
Multilayer Perceptron(Neural Network)

- The network is organized in terms of layers.
- Each layer has:
  - **Neurons**.
  - Neurons have **inputs**
  - Each input is **weighted.**



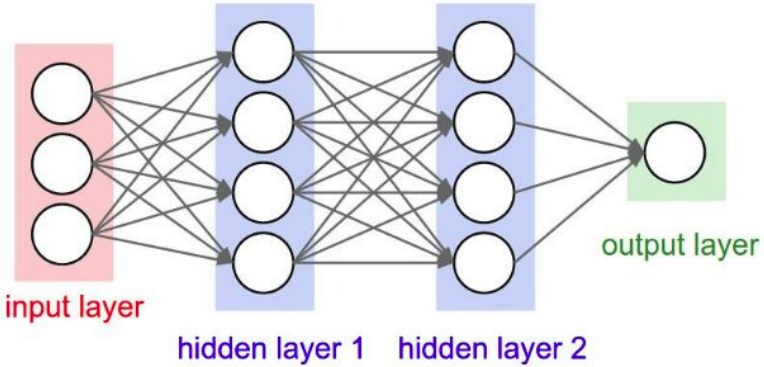
Multilayer Perceptron(Neural Network)

- The network is organized in terms of layers.
- Each layer has:
  - **Neurons**.
  - Neurons have **inputs**
  - Each input is **weighted.**
  - The weighted input is transformed by an **activation function.**



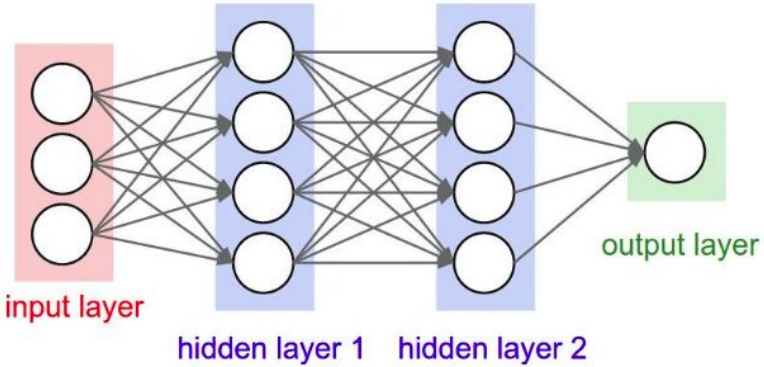
Multilayer Perceptron(Neural Network)

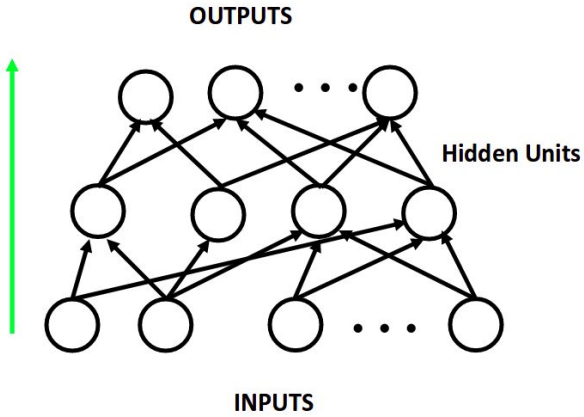
- The network is organized in terms of layers.
  - Input Layer(s)
  - Hidden Layer(s)
  - Output Layer(s)
- Hidden layers learn representation of input data by introducing non-linearity.
  - Transforms input features into higher level of abstraction.
  - Eg. Pixels' RGB values are transformed into more abstract features.



Multilayer Perceptron(Neural Network)

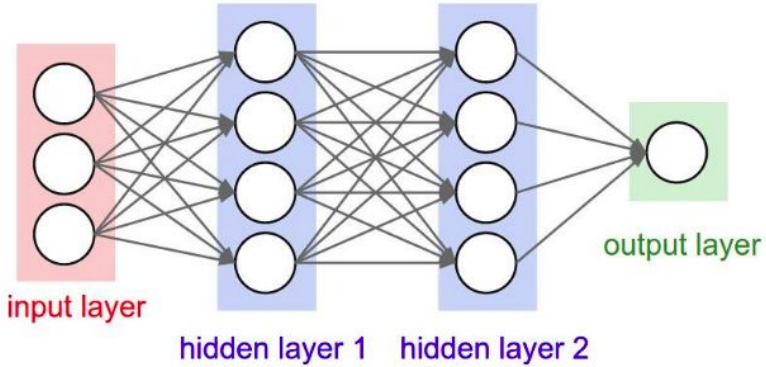
- The network is organized in terms of layers.
  - Input Layer(s)
  - Hidden Layer(s)
  - Output Layer(s)
- Hidden layers learn representation of input data by introducing non-linearity.
  - Transforms input features into higher level of abstraction.
  - Eg. Pixels RGB values are transformed into more abstract features.
- Eg: Input -> Hidden Layer -> Hidden Layer -> Output layer





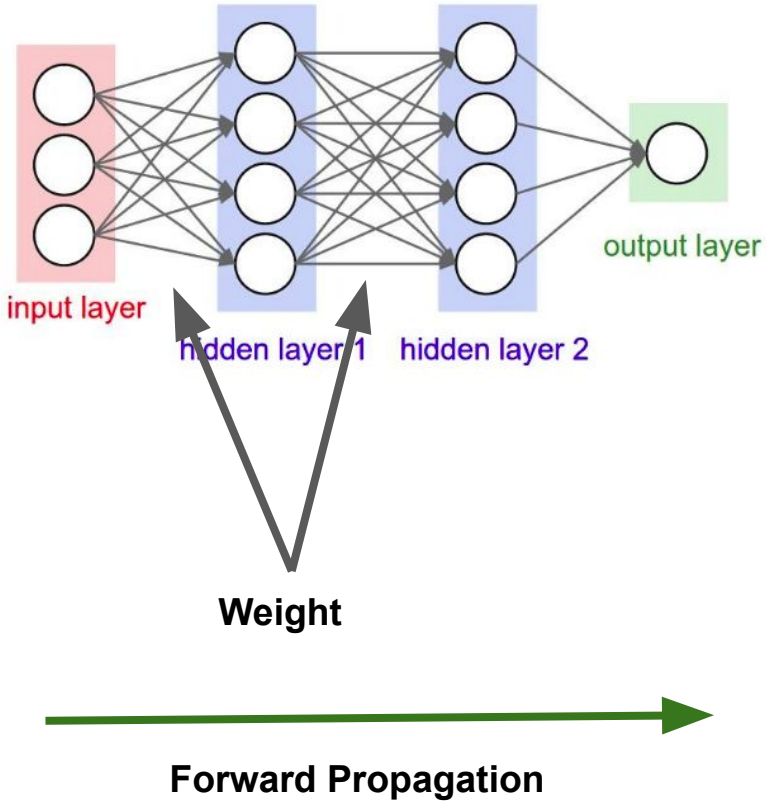
Multilayer Perceptron - Forward Propagation

- Information from from one layer to another.
  - **Forward Propagation.**



Multilayer Perceptron - Forward Propagation

- Information from from one layer to another.
  - **Forward Propagation.**



Multilayer Perceptron(Neural Network)

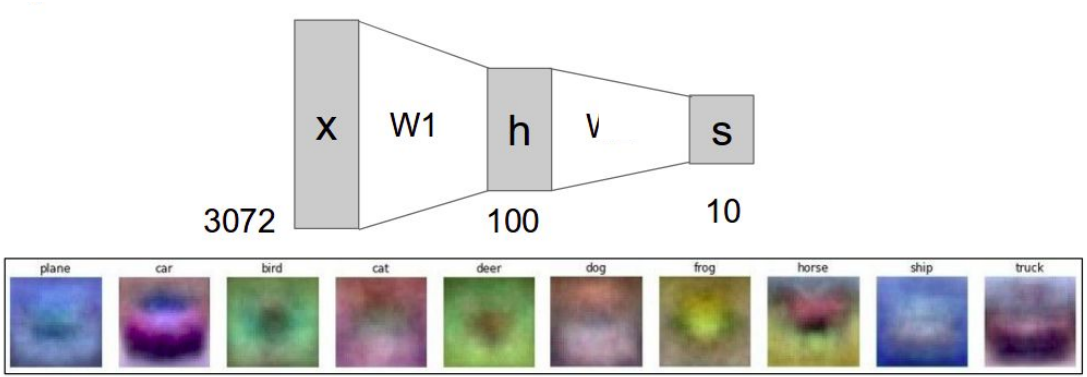
- Fully connected layer/ Multilayer Perceptron
  - Image size: 32 x 32 x 3

Multilayer Perceptron(Neural Network)

- Fully connected layer/ Multilayer Perceptron
  - Image size: 32 x 32 x 3
  - Flatten: 32 x 32 x 3 → 3072 x 1

Multilayer Perceptron(Neural Network)

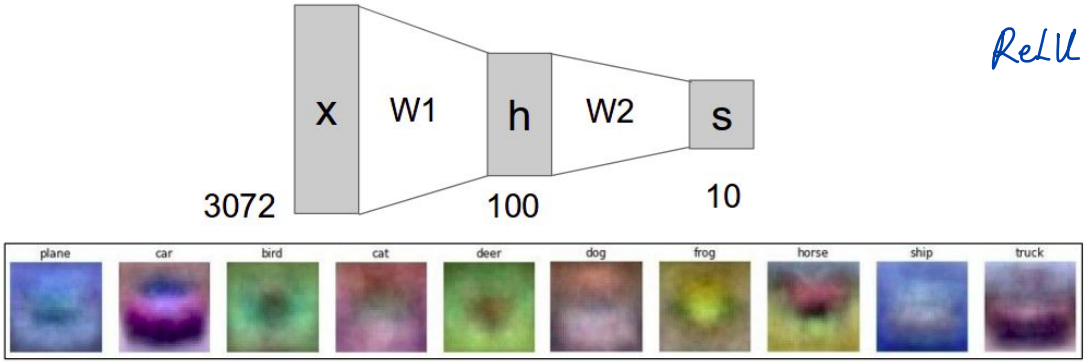
- Fully connected layer/ Multilayer Perceptron
  - Image size: 32 x 32 x 3
  - Flatten: 32 x 32 x 3 → 3072 x 1



Multilayer Perceptron(Neural Network)

- Fully connected layer/ Multilayer Perceptron
  - Image size: 32 x 32 x 3
  - Flatten: 32 x 32 x 3 → 3072 x 1

数据整理

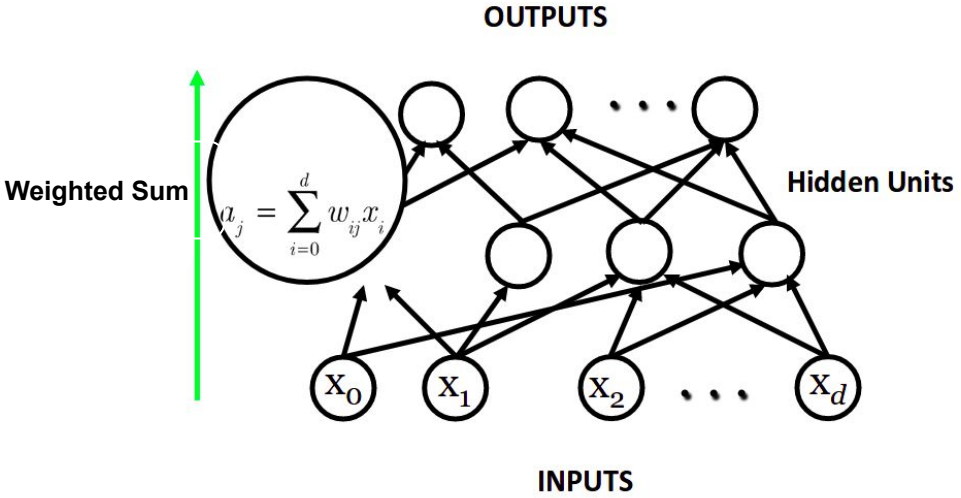


ReLU

$$f_x = W_1 X$$
$$f_h = max(0, f_x)$$
$$f_s = W_2 f_h$$
$$f_o = Softmax(f_s)$$

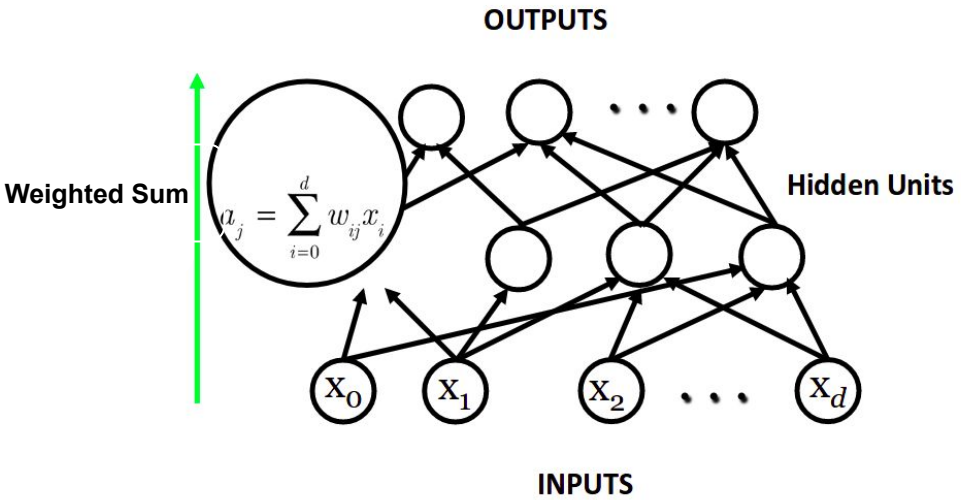
Hidden layer

- Input to hidden layer(a_j): weighted sum of all inputs (x_i , i=0..**d**)

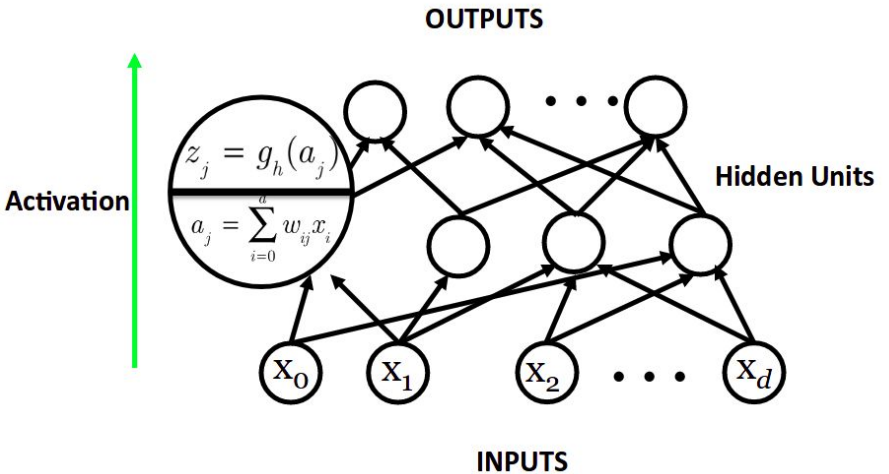


Hidden layer

- Input to hidden layer(a_j): weighted sum of all inputs (x_i , i=0..**d**)

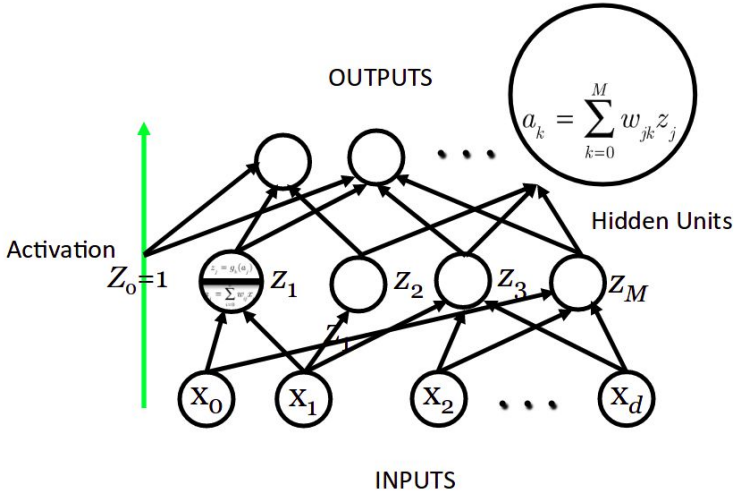


- Output of hidden layer: $z_j = g_h(a_j)$
  - g_h introduces nonlinearity
  - g_h : reLU/sigmoid/tanh



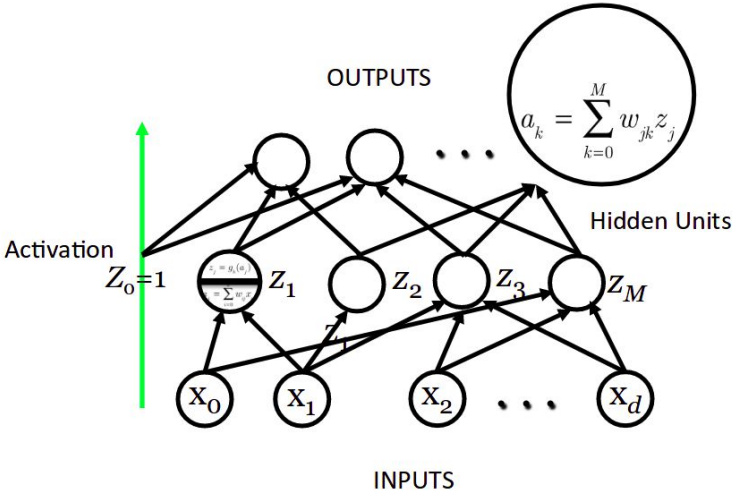
Output layer

- Input to Output layer(a_k): weighted sum of all inputs (z_i , i=0..**M**)

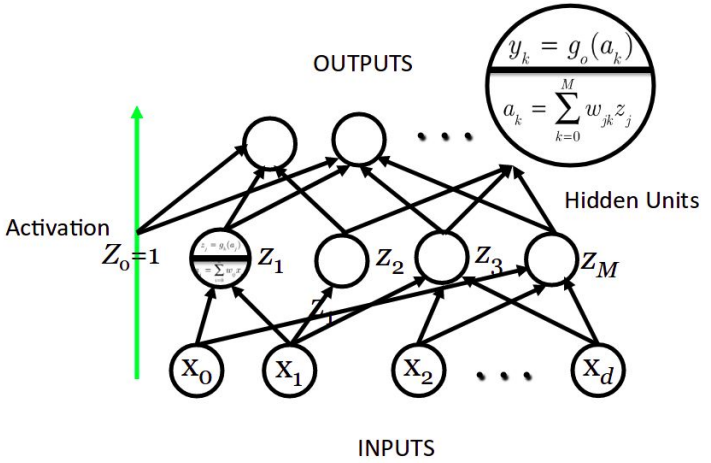


Output layer

- Input to Output layer(a_k): weighted sum of all inputs (z_i , i=0..**M**)

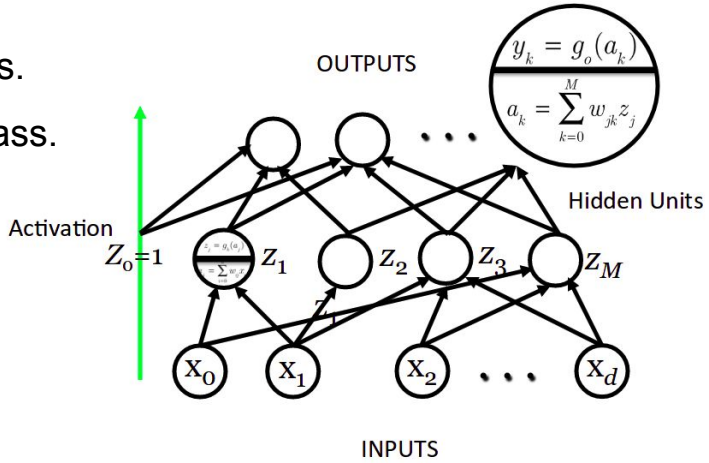


- Output of Output layer: $y_k = g_o(a_k)$
  - Prediction y_k is compared against target value.
  - G_o is activation function of outputs ~ **softmax**, linear, sigmoid, etc ...



Multi-Class Classification

- Input feature belongs to one of the many possible classes.
- Output is probability of input belonging to any of these class.



Multi-Class Classification

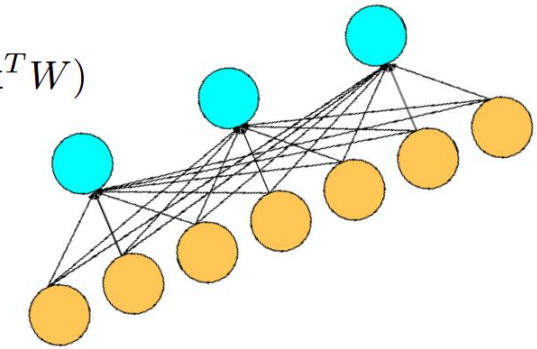
- Input feature belongs to one of the many possible classes.
- Output is probability of input belonging to any of these classes.

Softmax

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- Represent labels as one hot vector
  - [0 0 1 0 0]
- Prediction
  - [0.02, 0.05, 0.53, 0.32, 0.08]

- Label: $y \in 1, ..., k$
- Output: $\hat{y} = \text{softmax}(\mathbf{x}^T W)$
- Weights: $W \in \mathbb{R}^{d \times k}$
- Input: $\boldsymbol{x} \in \mathbb{R}^d$



Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Loss Functions

different between dog and cats
need to find a way updating weights

- Squared loss
  - $\hat{y}$: prediction
  - y: true value

we want zero loss

$$l(y, \hat{y}) = \sum (y - \hat{y})^2$$

minimize errors
least square

$$Loss = \sum_x \sum_k (y_k - \hat{y}_k)^2$$

Loss Functions

- Squared loss
  - $\hat{y}$: prediction
  - y: true value

$$l(y, \hat{y}) = \sum (y - \hat{y})^2$$

$$Loss = \sum_x \sum_k (y_k - \hat{y}_k)^2$$

- Cross Entropy
  - $\hat{y}$: prediction
  - y : true value

$$l(y_k, \hat{y}_k) = -\sum_k y_k \log \hat{y}_k$$

you can come up with other loss function

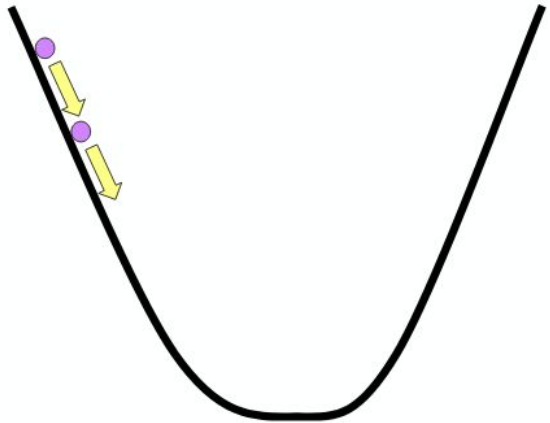
$$Loss = -\sum_x \sum_k y_k \log \hat{y}_k$$

Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Gradient Descent

- Prediction is a function of weights.
- **Aim:** Find optimal weights, so that the predictions are closer to true values.(Minimize loss)



Calculate prediction $\hat{y}_i = \boldsymbol{w}^T \boldsymbol{x}_i$

Calculate loss $\ell(y_i, \hat{y}_i) = ||y_i - \hat{y}_i||_2^2$

Apply update $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \ell(y_i, \hat{y}_i)$

step size

direction

find the weights minimize the loss function.

$\nabla_{\boldsymbol{w}} \ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \dfrac{d\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})}{d\boldsymbol{w}}$

can be difficult

Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

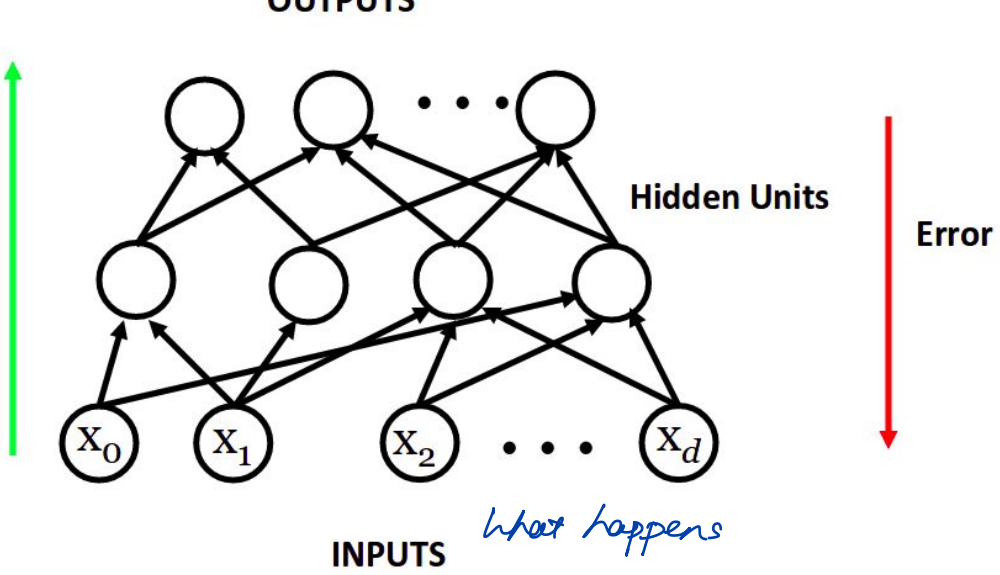
Backpropagation

$$w_{i,j} = w_{i,j} - \eta \frac{\partial L}{\partial w_{i,j}}$$

use the loss function to update w in order to find minimum w.

$w_{i,j}$ is any weight in the network

for each layer, we'll use chain rule to add information



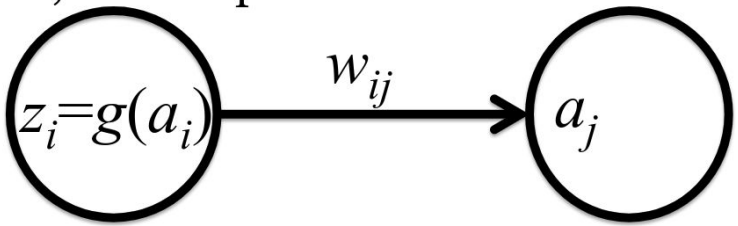
Backpropagation

- J is the objective function.
- We want: $\frac{\partial J}{\partial w_{ij}}$ where w_{ij} is any weight in the network
- As usual, we start with:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}}$$

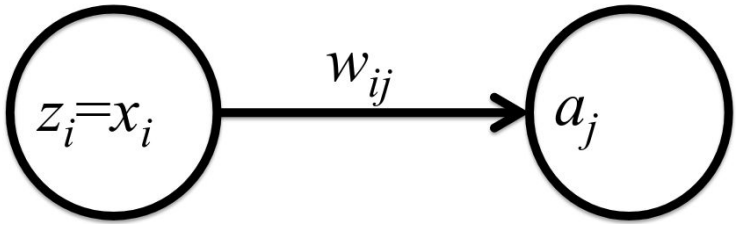
- **Notation**: z_i will mean *either* the output of a hidden unit, or an input.

i is a hidden unit



OR:

i is an input



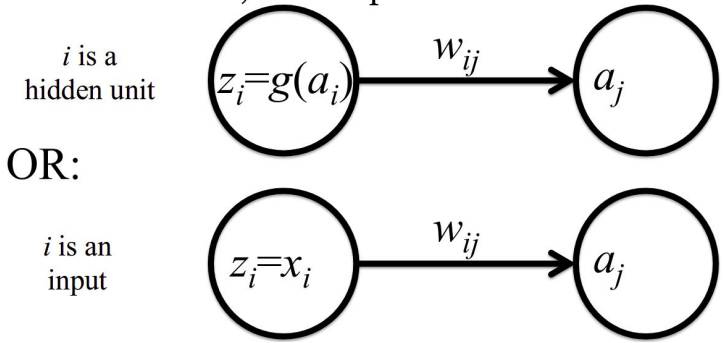
Backpropagation

- So we have:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial J}{\partial a_j} z_i$$

- As before, we have a term that is the input on the line from i to j.

- **Notation**: z_i will mean *either* the output of a hidden unit, or an input.



Backpropagation

- So we have:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial J}{\partial a_j} z_i$$

$$\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial \sum_{k=0}^{H} w_{kj} z_k}{\partial w_{ij}} = \sum_{k=0}^{H} \frac{\partial w_{kj} z_k}{\partial w_{ij}} = z_i$$

Derivative of the sum is the sum of the derivatives

In partial derivatives, all other variables (w_{kj} when $k \neq i$) are considered constants

- As before, we have a term that is the input on the line from i to j.

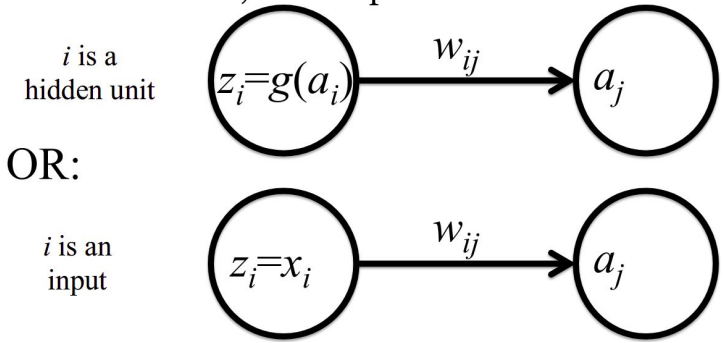
Backpropagation

- So we have:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial J}{\partial a_j} z_i$$

- As before, we have a term that is the input on the line from i to j.

- **Notation**: z_i will mean *either* the output of a hidden unit, or an input.

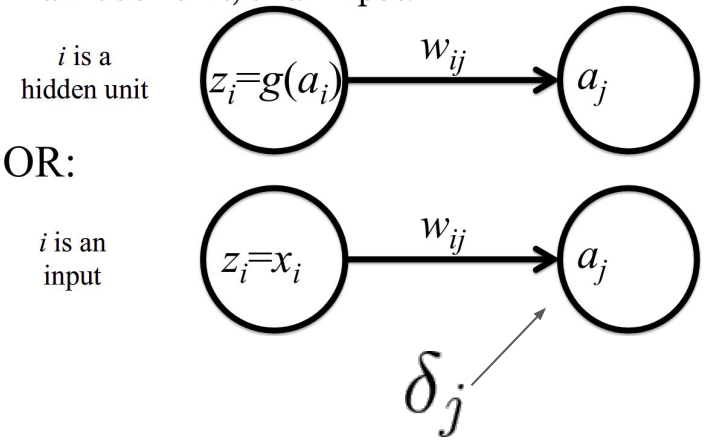


Backpropagation

- Now, we *define* $\delta_j \triangleq -\frac{\partial J}{\partial a_j}$
- So we have: $\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = -\delta_j z_i$

$$w_{ij} = w_{ij} - \frac{\partial J}{\partial w_{ij}}$$
$$= w_{ij} + \delta_j z_i$$

- **Notation**: z_i will mean *either* the output of a hidden unit, or an input.



Backpropagation

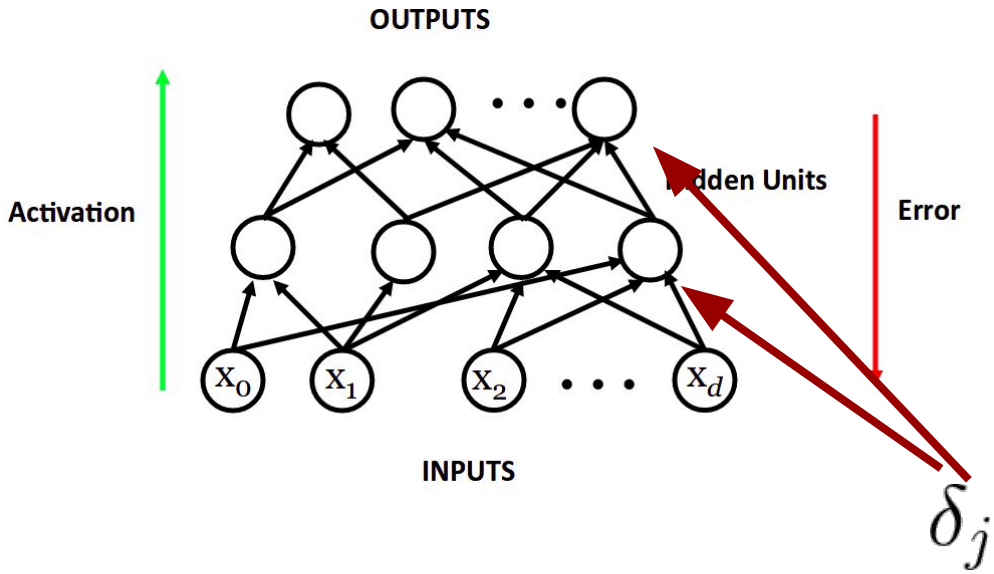
δ_j can be for the output layers or hidden layer

- Output layer: Softmax
- Hidden layer: Relu, Sigmoid, tanh

- Output units - Cross entropy loss

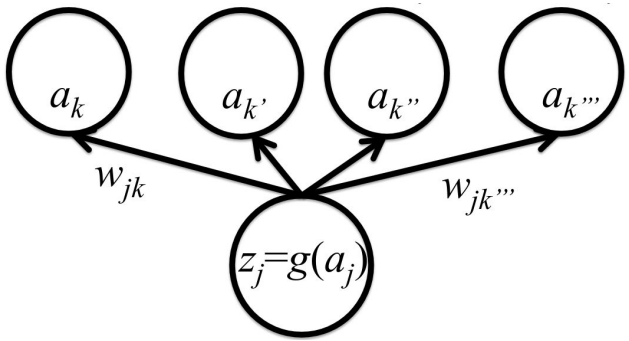
$$-\delta_j = (t_j - y_j)$$

$$w_{i,j} = w_{i,j} + \eta(t_j - y_j)z_i$$



Backpropagation

- Hidden units



$$\frac{\partial J}{\partial a_j} = \sum_k \frac{\partial J}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

$$= -\sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$ Definition of Delta

$$= -\sum_k \delta_k \frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j}$$ Chain Rule

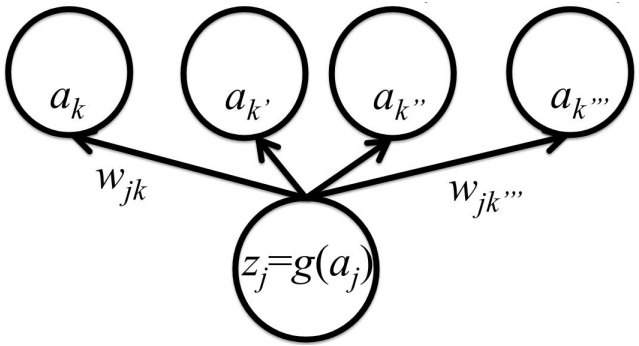
$$= -\frac{\partial z_j}{\partial a_j}\sum_k \delta_k \frac{\partial a_k}{\partial z_j} = -g'(a_j)\sum_k \delta_k \frac{\partial a_k}{\partial z_j}$$ j is independent of k and also $z_j = g(a_j)$

$$= -g'(a_j)\sum_k \delta_k \sum_i \frac{\partial w_{ik} z_i}{\partial z_j} = -g'(a_j)\sum_k \delta_k w_{jk}$$ Definition of a_k

Every term in the sum is 0 except when i = j

Backpropagation

- Hidden units



- SO: $\frac{\partial J}{\partial a_j} = -g'(a_j)\sum_k \delta_k w_{jk}$ if j is a hidden unit
- But since delta is $-\frac{\partial J}{\partial a_j}$, $\delta_j = g'(a_j)\sum_k \delta_k w_{jk}$
- Here: $\delta_j = (t_j - y_j)$ if j is an output unit*

  $\delta_j = g'(a_j)\sum_k \delta_k w_{jk}$ if j is a hidden unit
- So, we have a recursive definition of delta

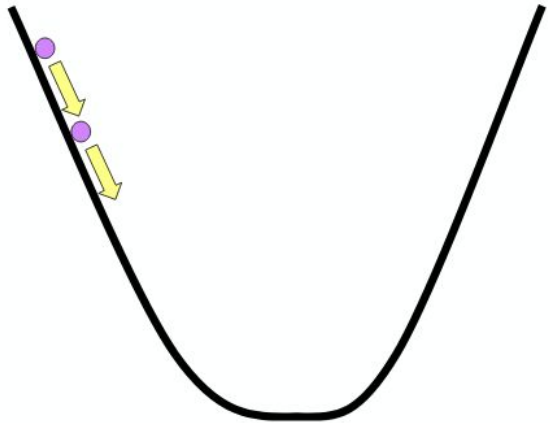
*assuming the right combination of objective function and output activation function

Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Optimizers

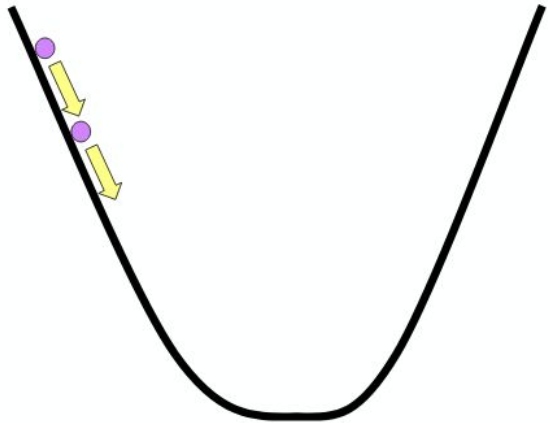
- Gradient Descent: most fundamental technique to train Neural Networks
    - Slow.



$$\nabla_{\boldsymbol{w}} \ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{d\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})}{d\boldsymbol{w}}$$

Optimizers

- Gradient Descent: most fundamental technique to train Neural Networks
    - Slow.
- Variants:
    - Momentum
    - Nesterov Momentum
    - AdaGrad
    - AdaDelta
    - RMSprop
    - Adam
    - Nadam



$$\nabla_{\boldsymbol{w}} \ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{d\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})}{d\boldsymbol{w}}$$

Momentum

SGD without momentum

$$v_t = \eta \nabla_\theta J(\theta)$$

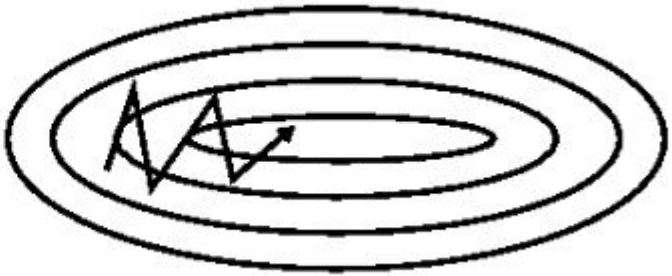
$$\theta = \theta - v_t$$

Momentum

SGD without momentum

$$v_t = \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$



SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)

SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)

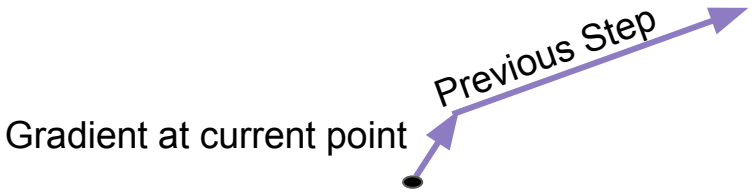
Gradient at current point 

SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)

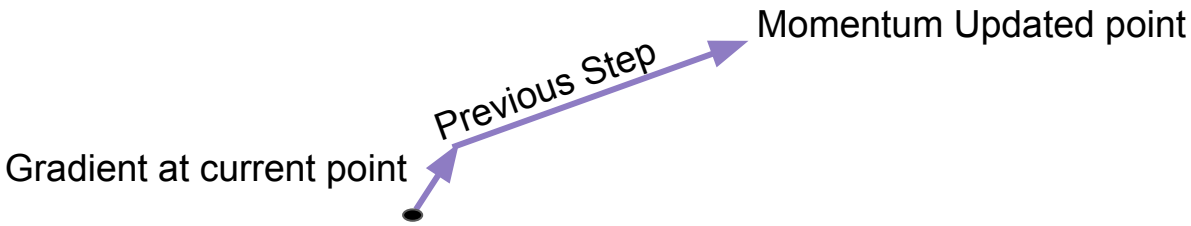


SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)

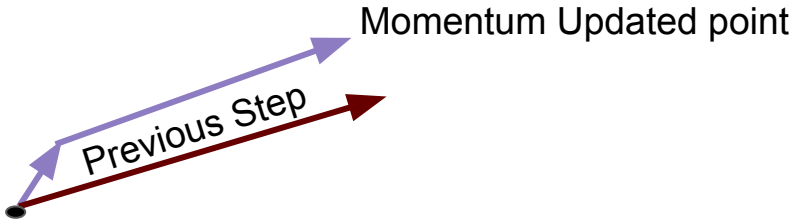


SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)



SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

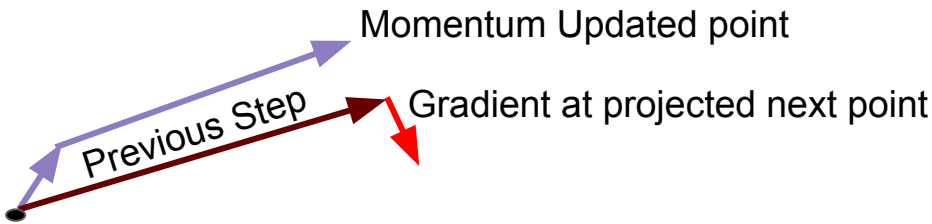
$$\theta = \theta - v_t$$

SGD with Nesterov momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)



SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

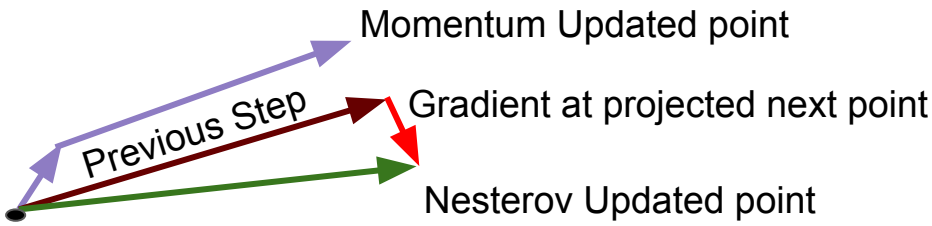
$$\theta = \theta - v_t$$

SGD with Nesterov momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)



SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

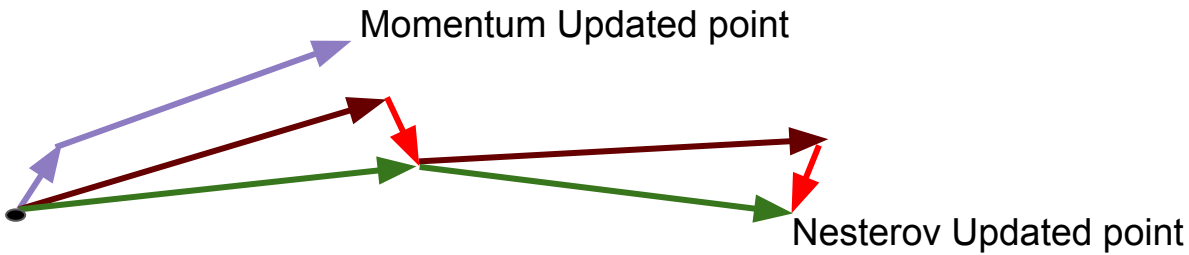
$$\theta = \theta - v_t$$

SGD with Nesterov momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

Nesterov accelerated Gradient (Momentum)



SGD with momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta)$$

$$\theta = \theta - v_t$$

SGD with Nesterov momentum

$$v_t = \gamma v_{t-1} + \eta \Delta_\theta J(\theta - \gamma v_{t-1})$$

$$\theta = \theta - v_t$$

Optimizers

- Now we would like to adapt our updates to individual parameters.
- Some parameters need larger updates, while some need smaller updates (step size).

Optimizers

- Now we would like to adapt our updates to individual parameters.
- Some parameters need larger updates, while some need smaller updates (step size).

Adagrad

- Adapts learning rate.
- Larger updates for infrequent parameters
- Smaller updates for frequent parameters.
- Earlier all parameters had same learning rate: η
- Now different learning rate for:
    - Different parameters
    - At different timesteps.

Adagrad

$$g_{t,i} = \Delta_\theta J(\theta_{t,i})$$

SGD Update

$$\theta_{t+1,i} = \theta_{t,i} - \eta g_{t,i}$$

Adagrad Update

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

G_t is sum of gradients wrt parameter till time step t.

Cons of Adagrad

- Accumulating gradient in denominator.
- Causes learning rate to shrink

AdaDelta

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

AdaGrad Update

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

AdaDelta Update

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Root mean square $RMS[g]_t$

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$

η

RMSprop

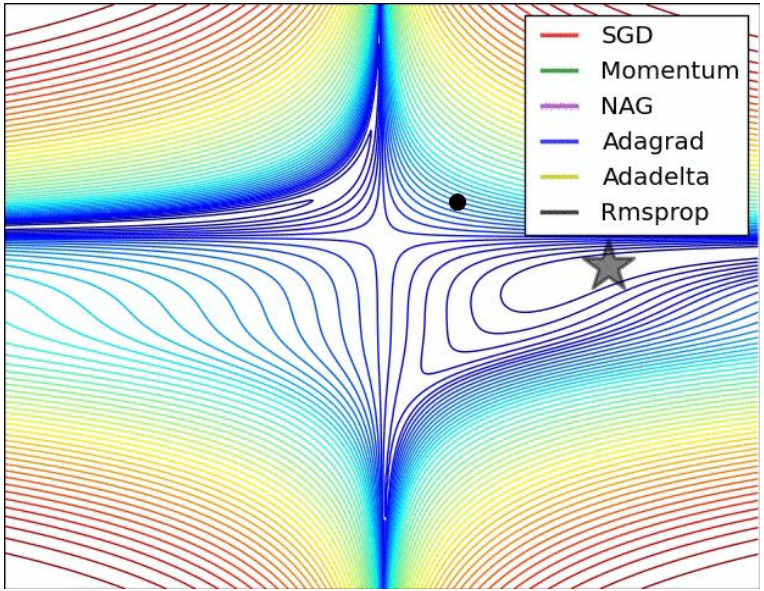
- Unpublished, adaptive learning rate.
- Proposed by Geoff Hinton in **Lecture 6e** of his **Coursera** course.
- Similar to **AdaDelta**, developed independently.

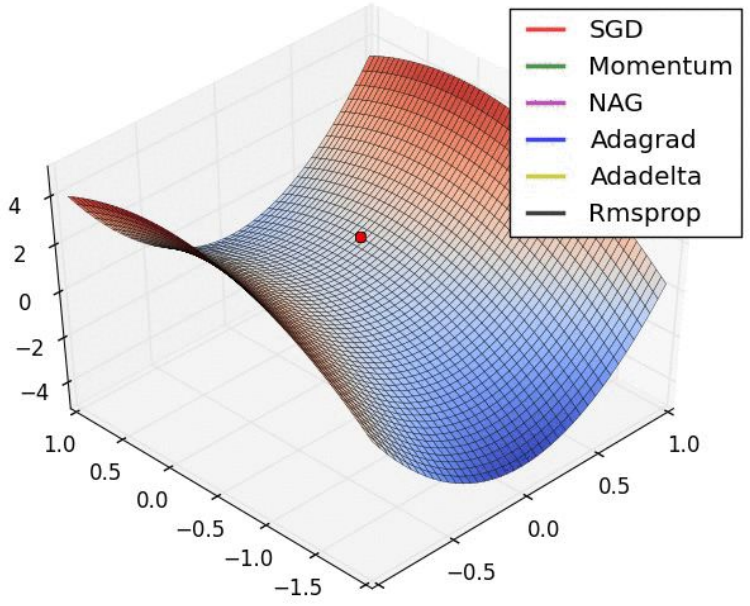
$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

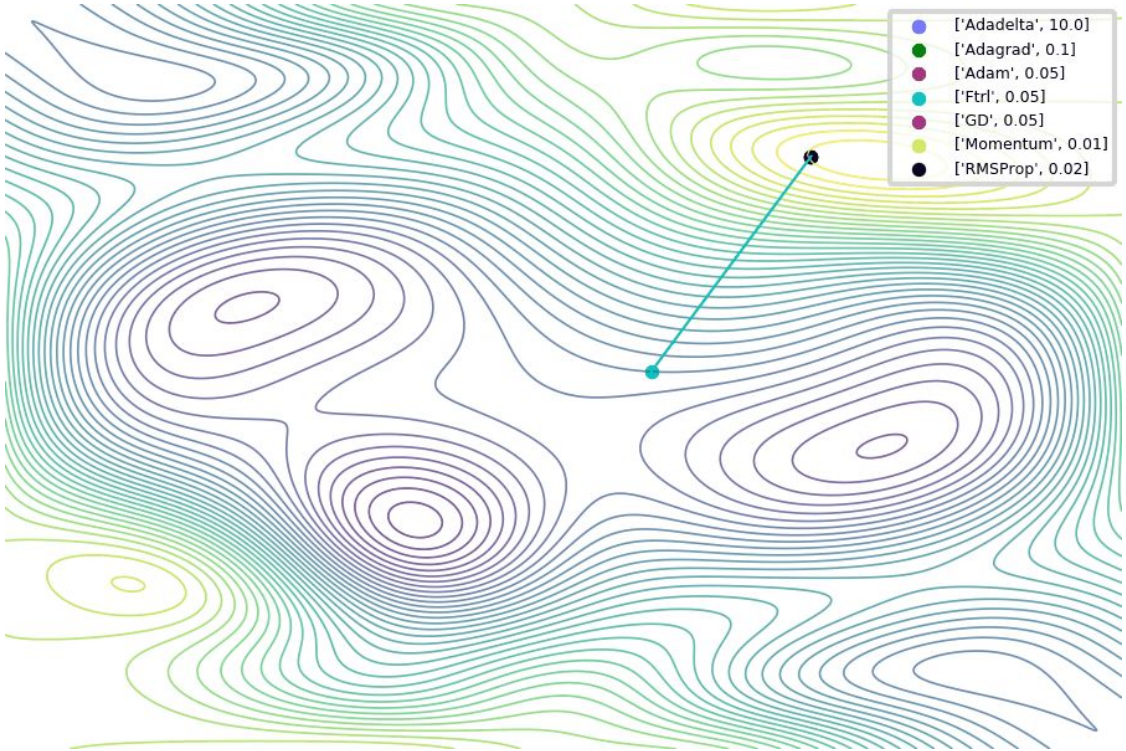
- Good η value: = 0.001

Optimizers


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Optimizers



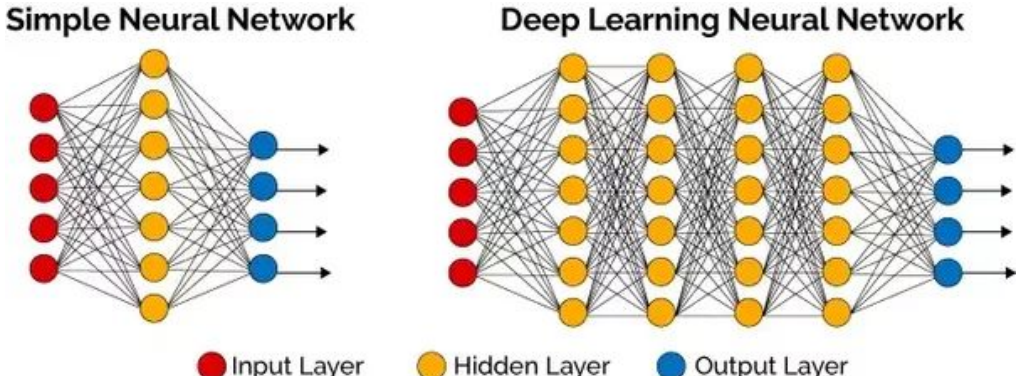
Credit: https://github.com/Jaewan-Yun/optimizer-visualization

Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

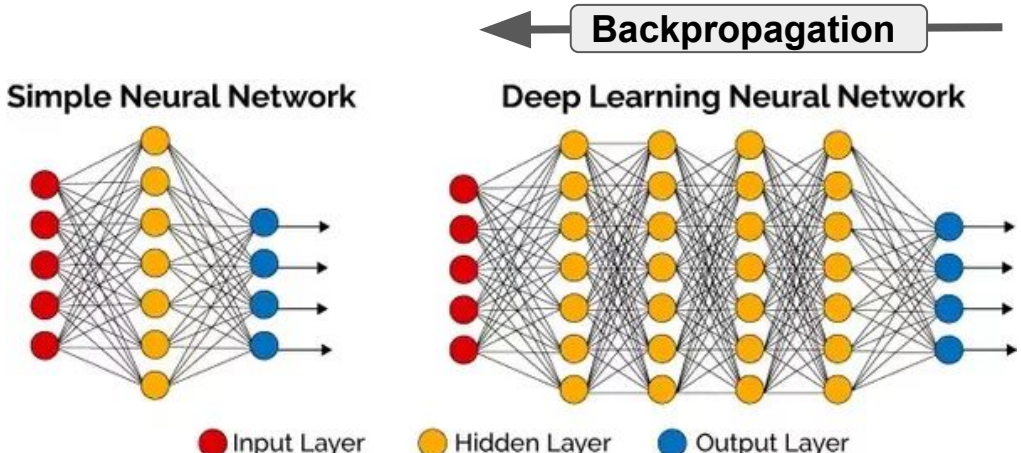
Vanishing Gradient

- Deep neural network have more representational capability.

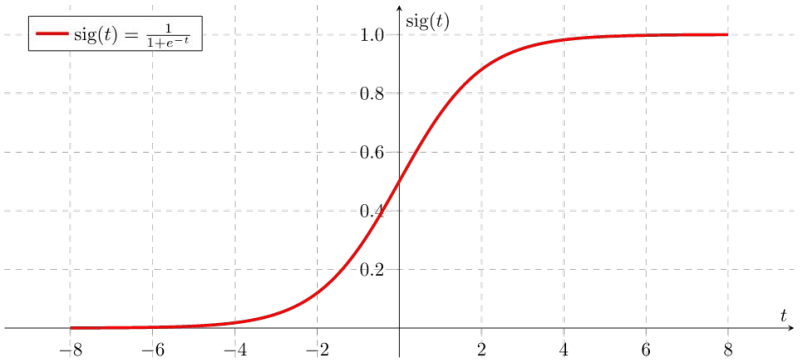


Vanishing Gradient

- Deep neural network have more representational capability.
- Practical problems with deeper networks:
  - Vanishing gradient:
    - Gradient flow from output towards input(in left direction) may vanish.



Vanishing Gradient

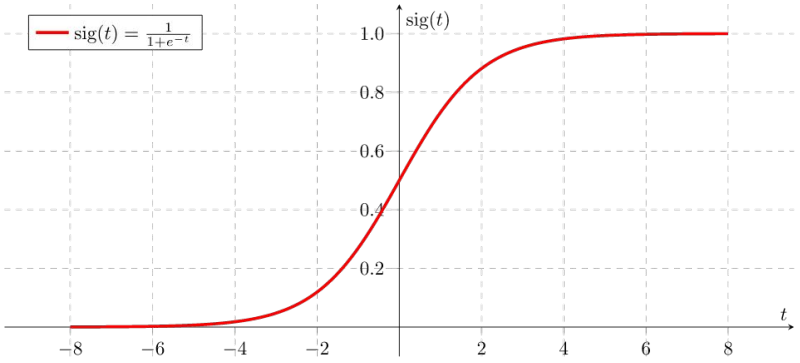


$$y = \frac{1}{1 + e^{-x}}$$

$$\nabla = \frac{dy}{dx} = y(1 - y)$$

$$\nabla_{max} = \frac{1}{4}$$

Vanishing Gradient

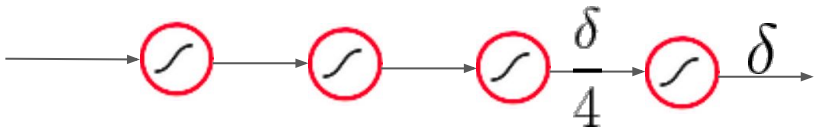


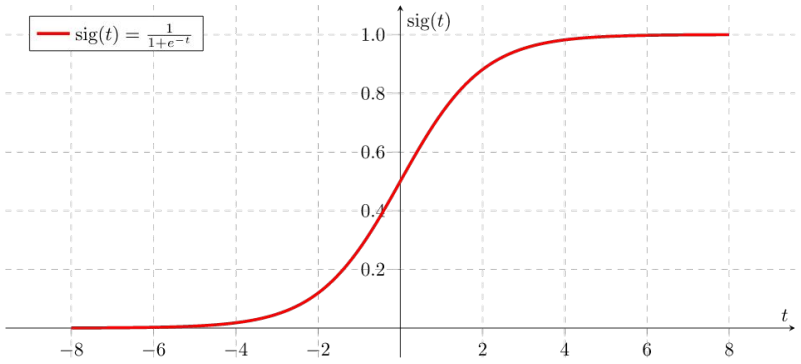
$$y = \frac{1}{1 + e^{-x}}$$

$$\nabla = \frac{dy}{dx} = y(1 - y)$$

$$\nabla_{max} = \frac{1}{4}$$

Vanishing Gradient



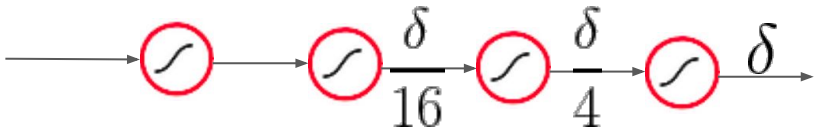


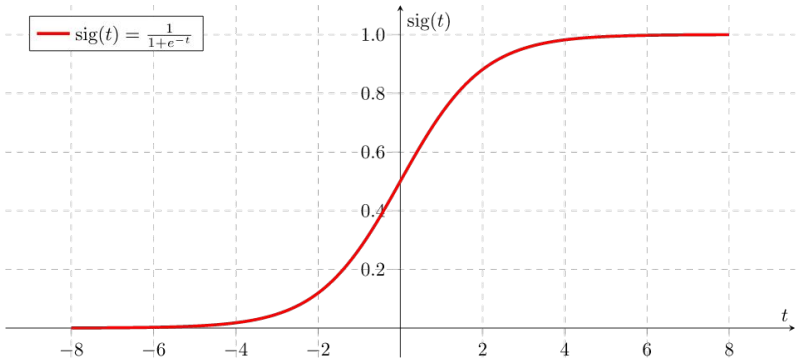
$$y = \frac{1}{1 + e^{-x}}$$

$$\nabla = \frac{dy}{dx} = y(1 - y)$$

$$\nabla_{max} = \frac{1}{4}$$

Vanishing Gradient



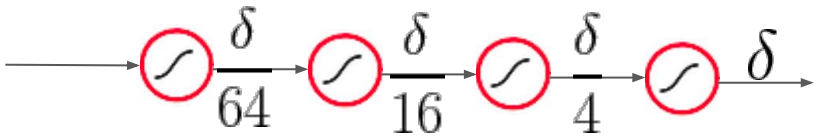


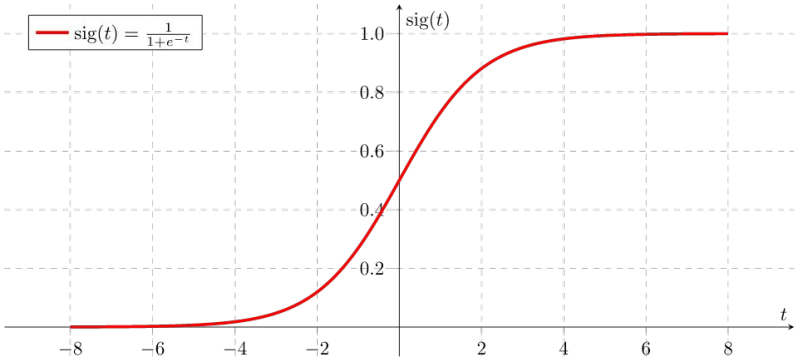
$$y = \frac{1}{1 + e^{-x}}$$

$$\nabla = \frac{dy}{dx} = y(1 - y)$$

$$\nabla_{max} = \frac{1}{4}$$

Vanishing Gradient





$$w = w - \alpha \frac{dJ}{dw}$$

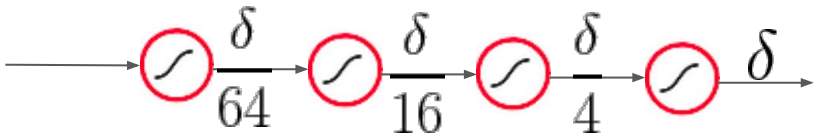
Gradient decreases along shallow/earlier layers.

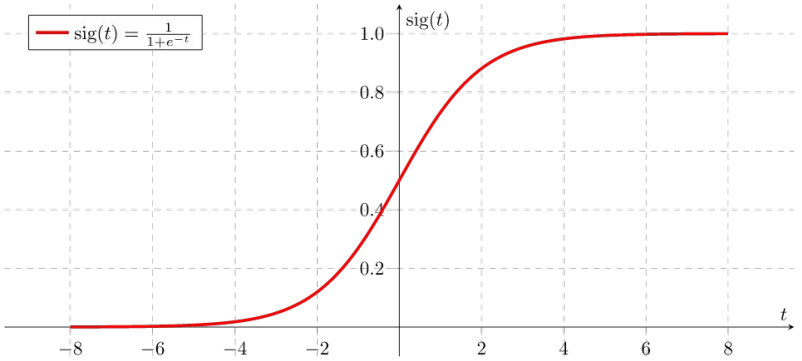
$$y = \frac{1}{1 + e^{-x}}$$

$$\nabla = \frac{dy}{dx} = y(1 - y)$$

$$\nabla_{max} = \frac{1}{4}$$

Vanishing Gradient





$$w = w - \alpha \frac{dJ}{dw}$$

Gradient decreases along shallow/earlier layers.

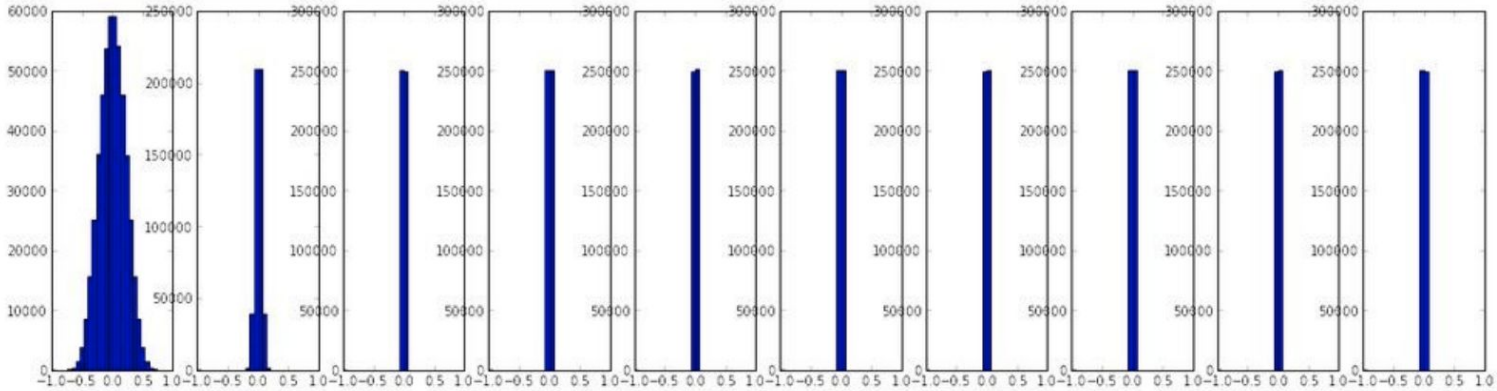
$$y = \frac{1}{1 + e^{-x}}$$

$$\nabla = \frac{dy}{dx} = y(1 - y)$$

$$\nabla_{max} = \frac{1}{4}$$

- Very small updates of weights in shallow layers.
  - It is difficult to train deeper networks.

Vanishing Gradient



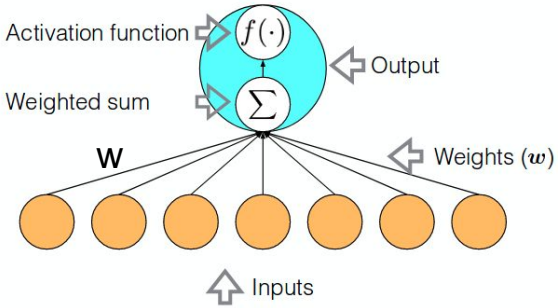
Shallow/Earlier Layers. Gradient Decreases

Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Vanishing Gradient: Xavier Initialization

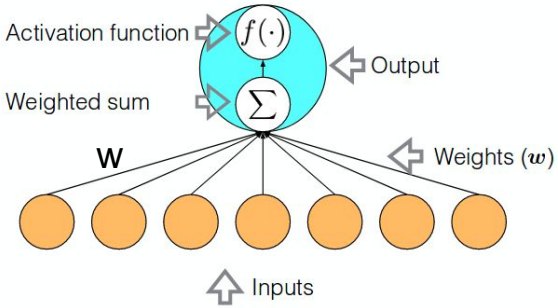
- Initialize weights with a particular initialization strategy.
  - Mean of input(weighted sum of inputs) = 0
  - Variance of input = 1



Vanishing Gradient: Xavier Initialization

- Initialize weights with a particular initialization strategy.
  - Mean of input(weighted sum of inputs) = 0
  - Variance of input = 1

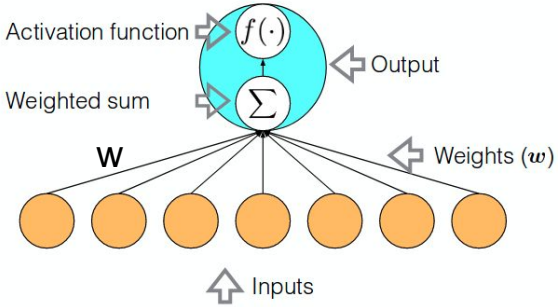
$$w \approx N(0, \frac{1}{\sqrt{fan_{in}}})$$

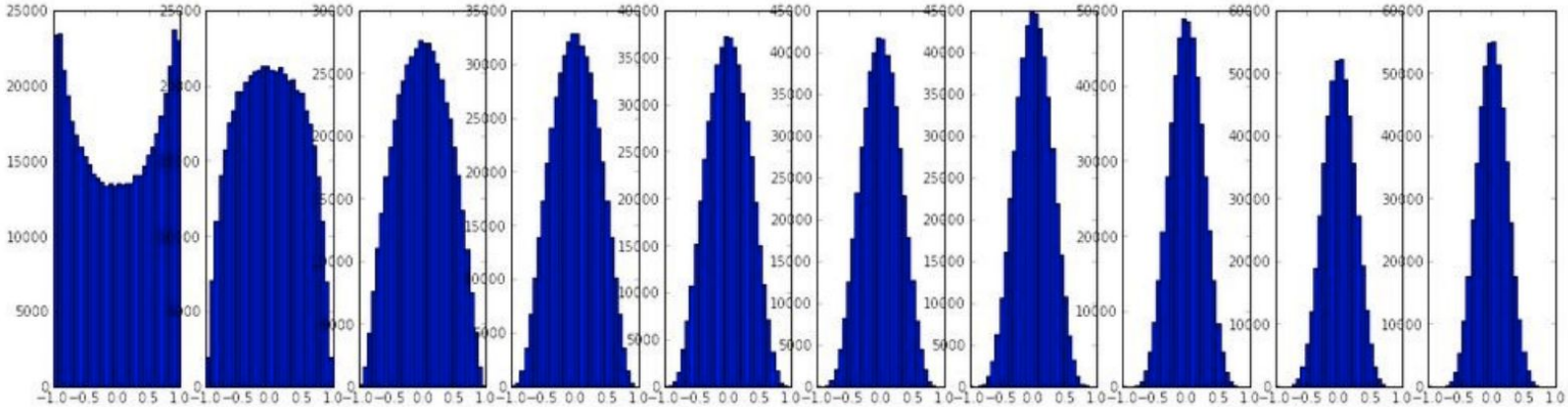


Vanishing Gradient: Xavier Initialization

- Initialize weights with a particular initialization strategy.
  - Mean of input(weighted sum of inputs) = 0
  - Variance of input = 1

$$w \approx N(0, \frac{1}{\sqrt{fan_{in}}})$$



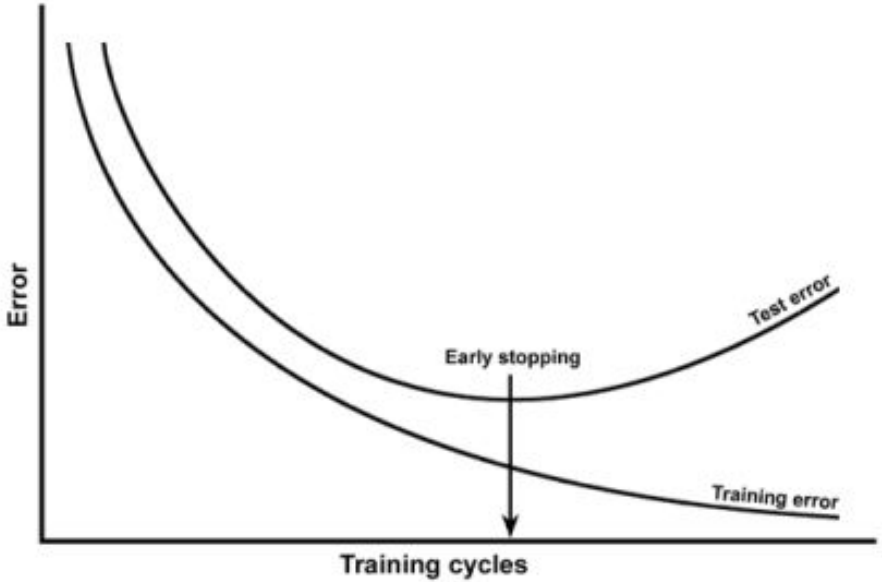


Agenda

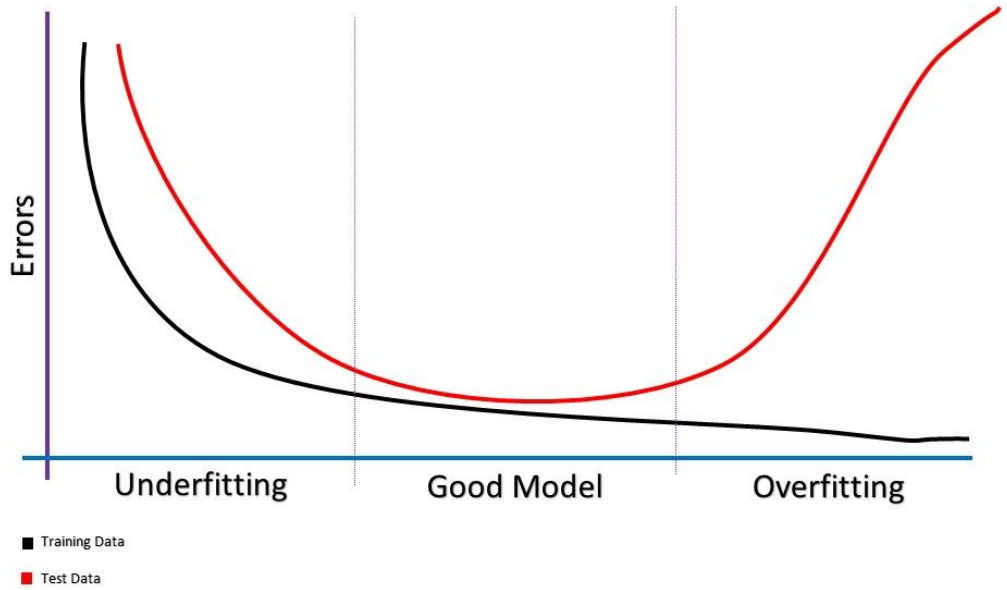
- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Network Performance Analysis

- Why is test error increasing ?

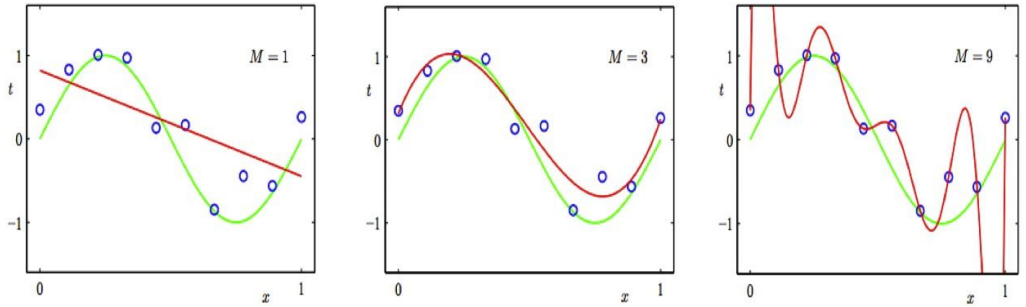


Overfitting vs Underfitting



Overfitting vs Underfitting

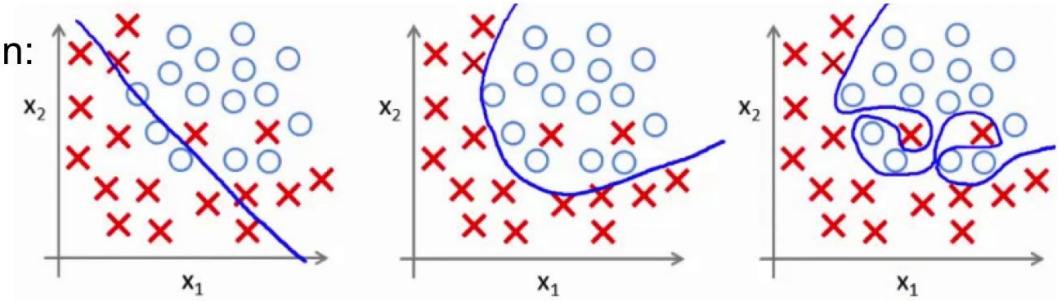
Regression:



predictor too inflexible:
cannot capture pattern

predictor too flexible:
fits noise in the data

Classification:



Agenda

- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Regularization: Dropout

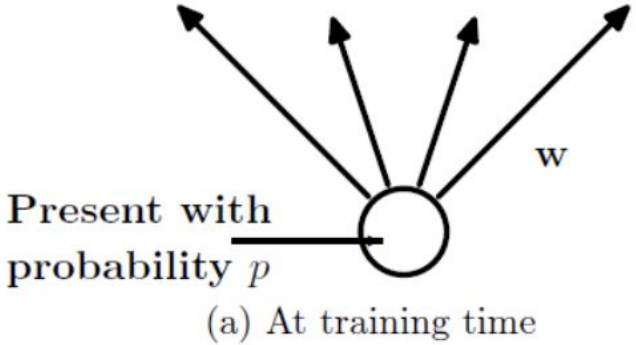
- Original/Standard network: All features are fed to classifier.

Regularization: Dropout

- Original/Standard network: All features are fed to classifier.
- **Dropout**: Randomly drop some of the inputs.

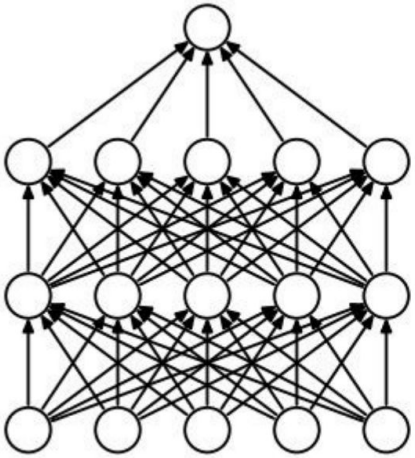
Regularization: Dropout

- Original/Standard network: All features are fed to classifier.
- **Dropout**: Randomly drop some of the inputs.
  - Neurons are kept/dropped with a probability **p**

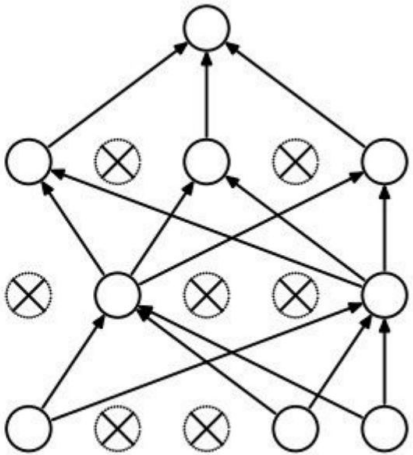


Regularization: Dropout

- Original/Standard network: All features are fed to classifier.
- **Dropout**: Randomly drop some of the inputs.
  - Neurons are kept/dropped with a probability **p**



(a) Standard Neural Net



(b) After applying dropout.

Regularization: Dropout

- Benefits:
  - Some neurons/inputs are dropped.
  - Other neurons are forced to compensate, learn better/generic representation.
  - Better/generic representation: Prevents **Overfitting**.

Regularization: Dropout

- Benefits:
  - Some neurons/inputs are dropped.
  - Other neurons are forced to compensate, learn better/generic representation.
  - Better/generic representation: Prevents **Overfitting**.

- Adds noise to Gradients:
  - Prevent Overfitting
  - Helps generalize with noise(to represent unseen data from similar distribution)

Regularization: Dropout

- Benefits:
  - Some neurons/inputs are dropped.
  - Other neurons are forced to compensate, learn better/generic representation.
  - Better/generic representation: Prevents **Overfitting**.

- Adds noise to Gradients:
  - Prevent Overfitting
  - Helps generalize with noise(to represent unseen data from similar distribution)

- Lower model complexity:
  - Less number of neurons
  - Prevents overfitting.

Regularization: Dropout

- Benefits:
  - Some neurons/inputs are dropped.
  - Other neurons are forced to compensate, learn better/generic representation.
  - Better/generic representation: Prevents **Overfitting**.

- Adds noise to Gradients:
  - Prevent Overfitting
  - Helps generalize with noise(to represent unseen data from similar distribution)

- Lower model complexity:
  - Less number of neurons
  - Prevents overfitting.

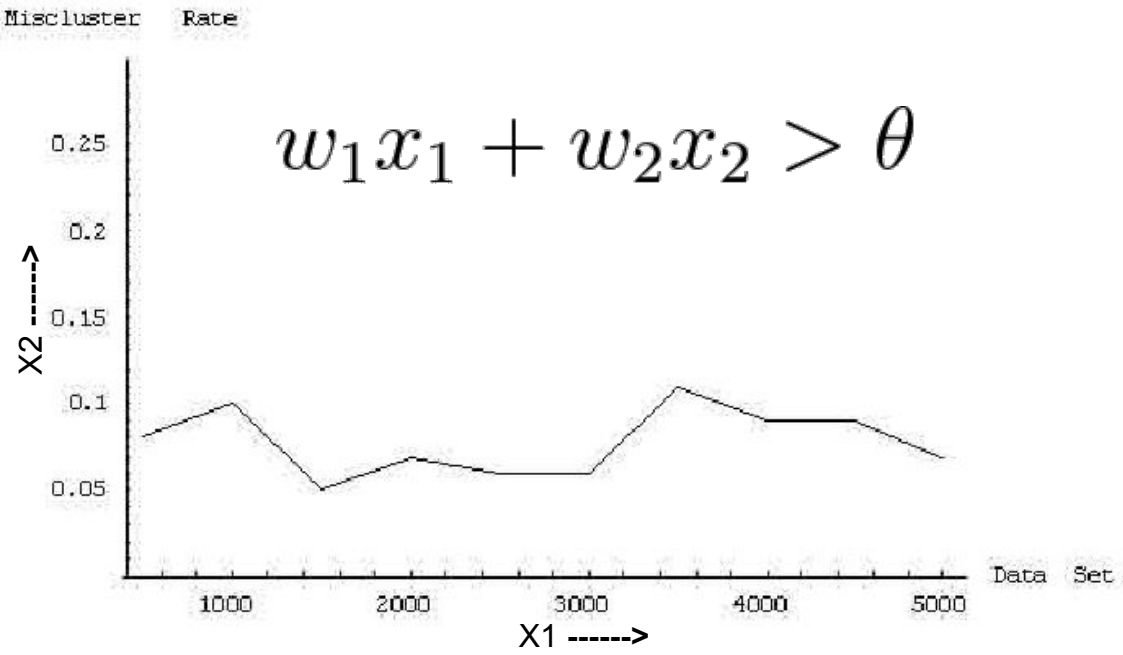
- Test time mode:
  - All the neurons are retained(not dropped).

Agenda

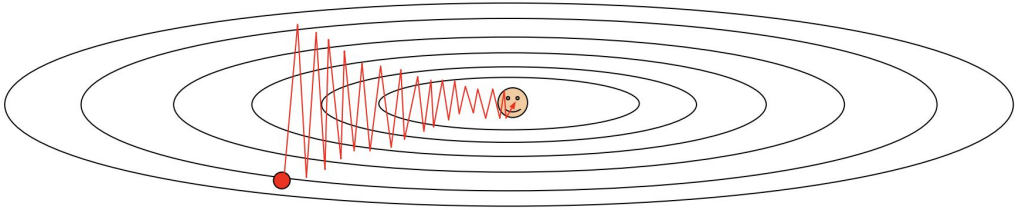
- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Feature Scaling

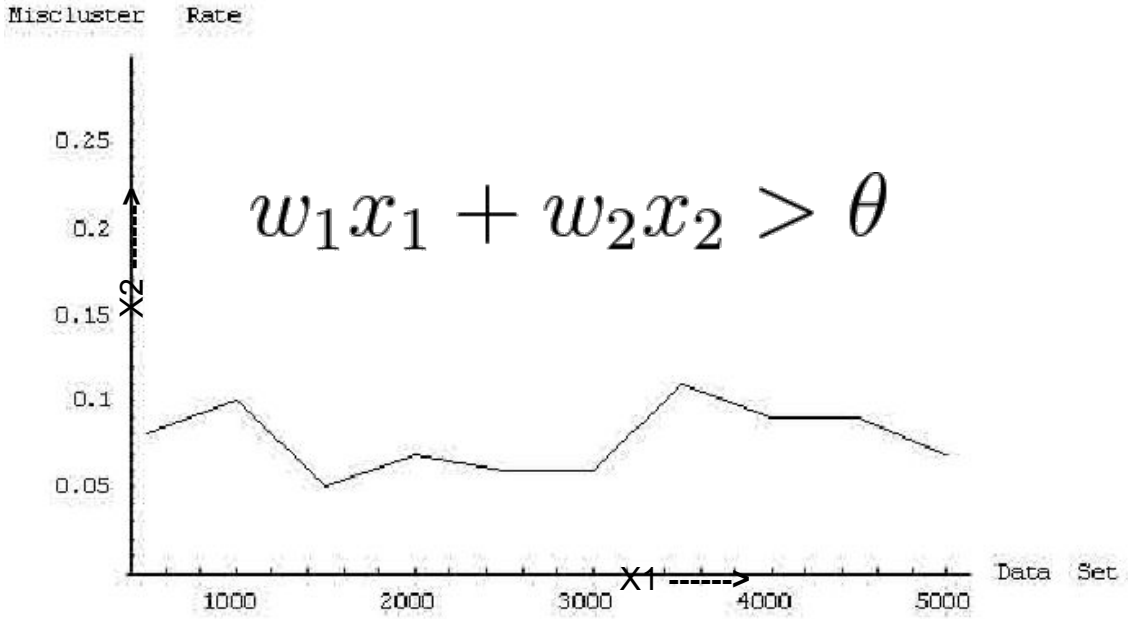
- x1: Large scale
  - w1 : small values
- x2: Small scale:
  - w2: Large values



Feature Scaling



- x1: Large scale
  - w1 : small values
- x2: Small scale:
  - w2: Large values

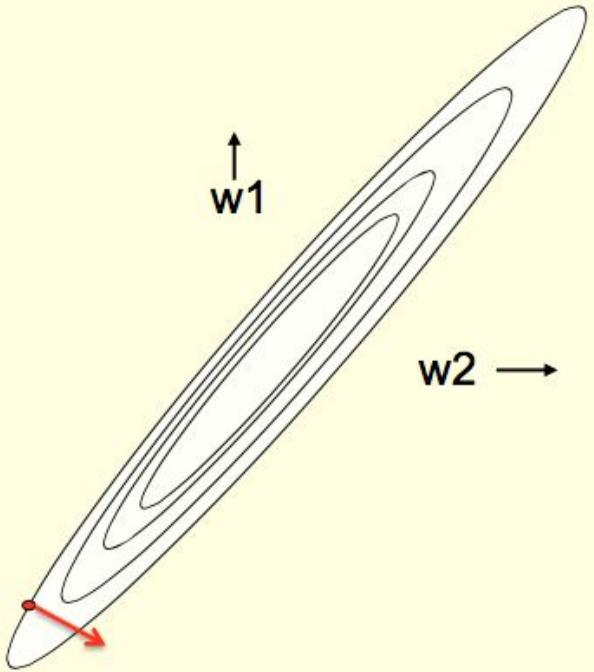


$$w_1 x_1 + w_2 x_2 > \theta$$

Feature Scaling

Why learning can be slow

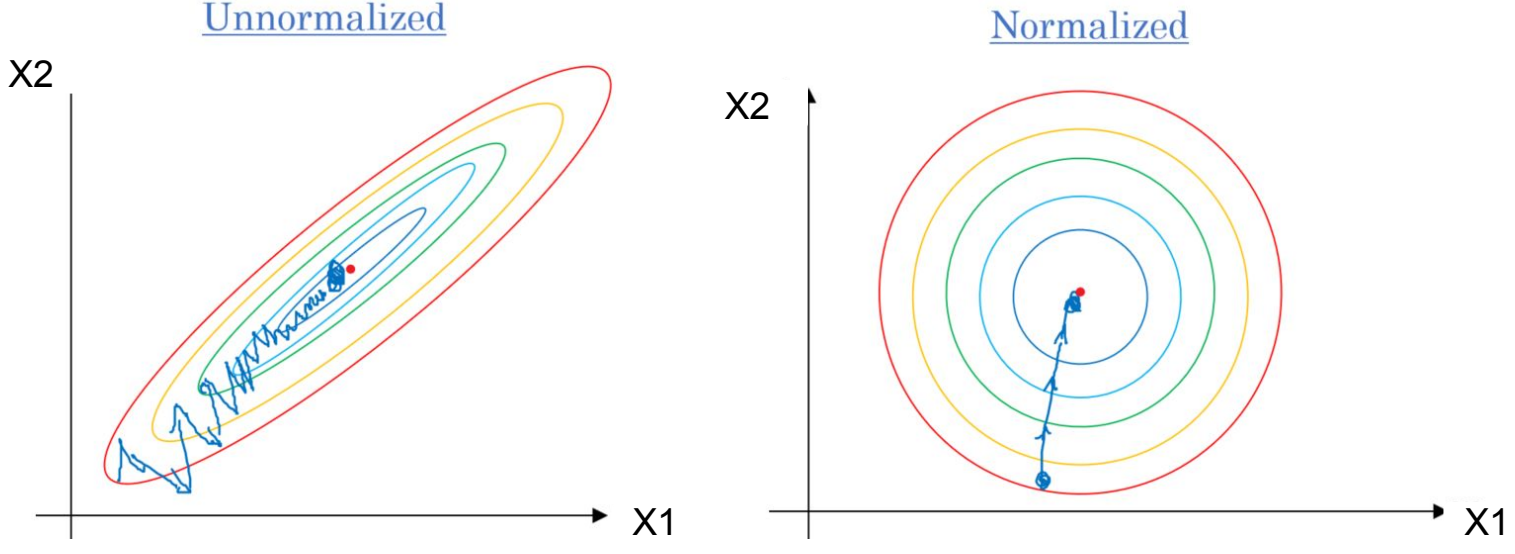
- If the ellipse is very elongated, the direction of steepest descent is almost perpendicular to the direction towards the minimum!
  - The red gradient vector has a large component along the short axis of the ellipse and a small component along the long axis of the ellipse.
  - This is just the opposite of what we want.



Geoff Hinton - Coursera slide

Feature Scaling

- Bring all data in similar Scale:
  - Normalization : Mean 0, Variance 1
  - The error surface changes from an ellipse to a circle
- Faster learning with feature scaling.



Common Normalizations:

- Two methods are usually used for re-scaling or normalizing data:
  - Scaling data all numeric variables to the range [0,1]. One possible formula is given below:

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  - To make data zero mean and unit variance:

$$x_{new} = \frac{x - \mu}{\sigma}$$

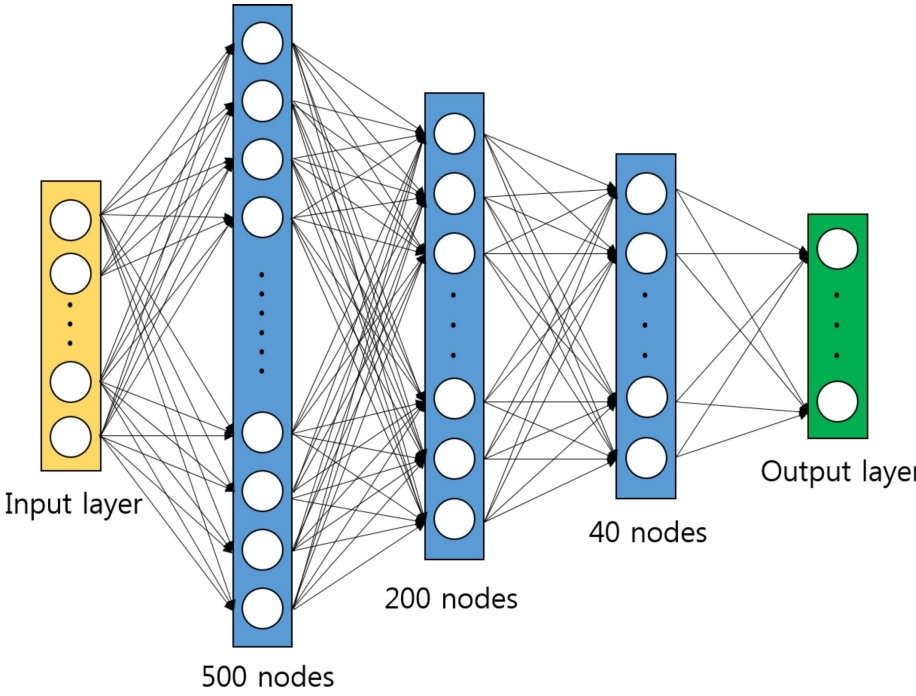
    - ❏ Most commonly used technique.

- This is also called *Whitening.*

Agenda

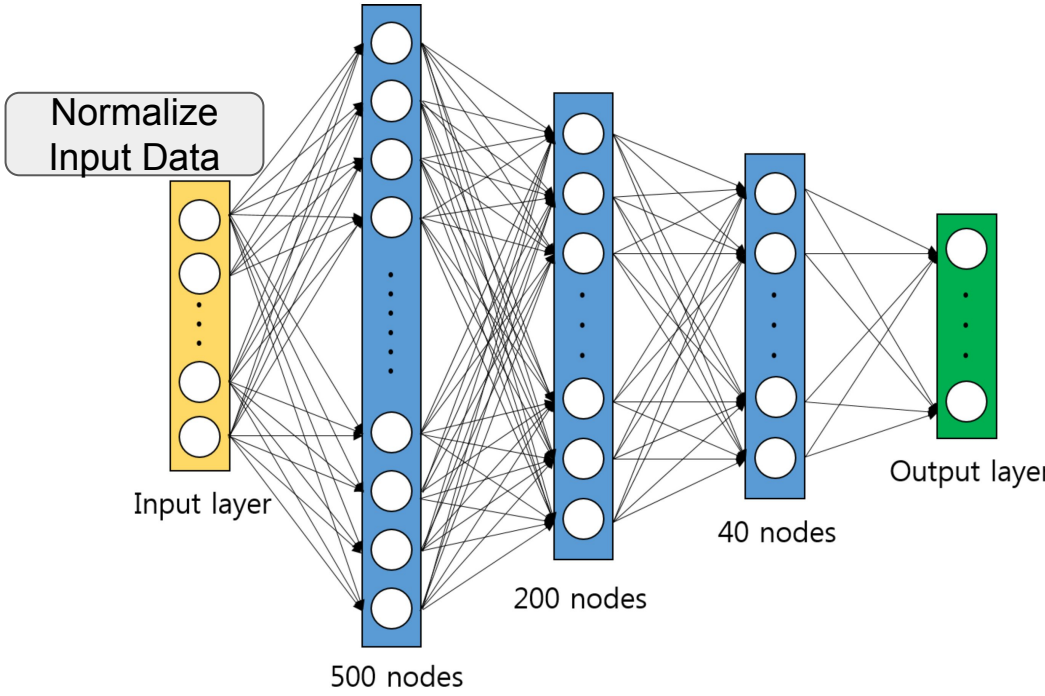
- Perceptron
  - Limitations
  - Activation Functions
- Multilayer Perceptron
  - Neurons/Weights/Hidden Layers
  - Forward Propagation
  - Loss Functions
  - Gradient Descent
  - Backward Propagation
  - Optimizers
  - Vanishing Gradient Problem
    - Xavier Initialization
  - Network performance analysis
  - Regularization - Dropout
  - Feature Scaling
  - Batch Normalization

Batch Normalization



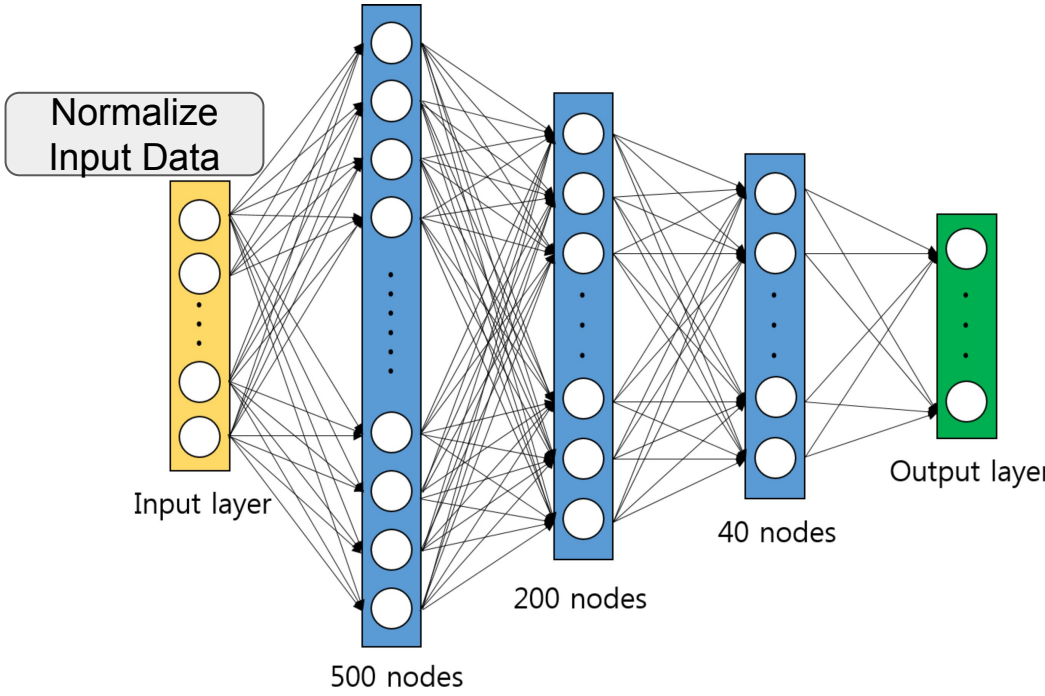
Batch Normalization

- Normalize input data



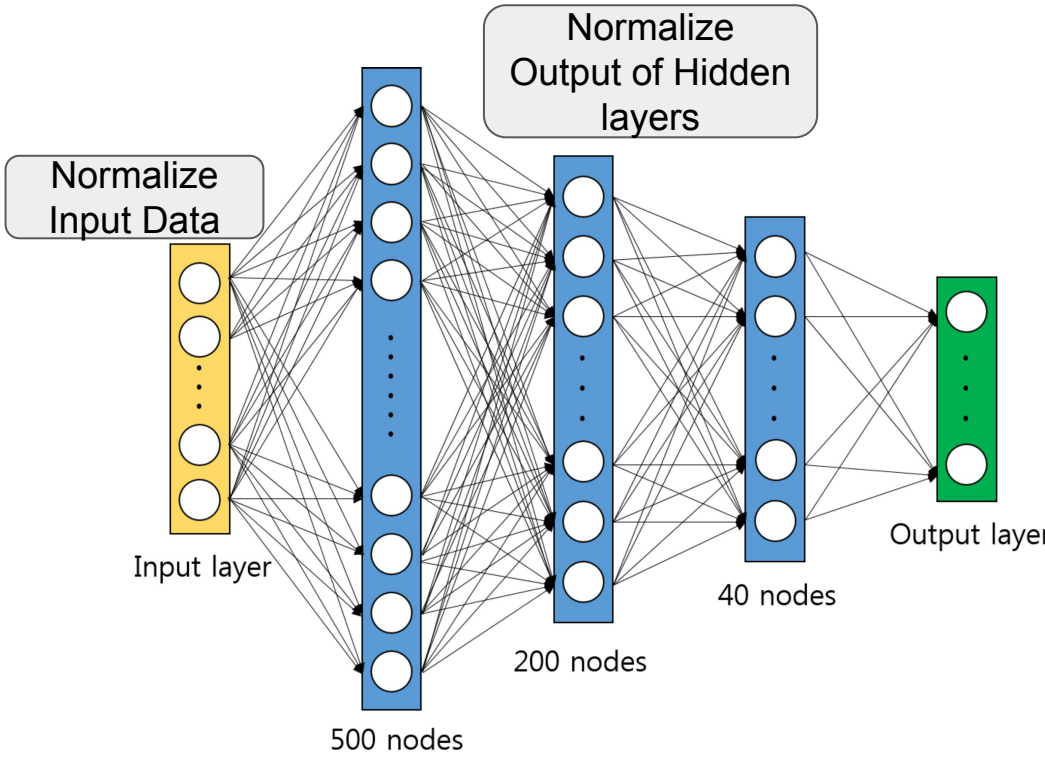
Batch Normalization

- Normalize input data
- Why not normalize output of all layers.



Batch Normalization

- Normalize input data
- Why not normalize output of all layers.
- Batch Normalization.
  - Normalize hidden layer outputs.



Batch Normalization

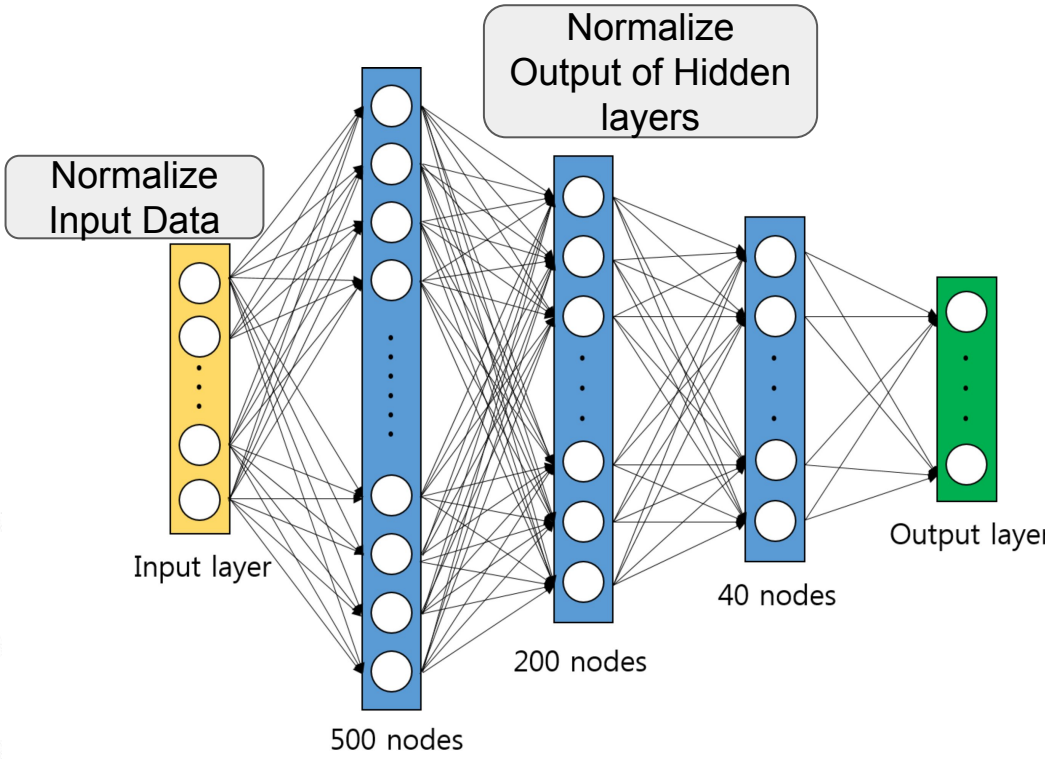
- Normalize input data
- Why not normalize output of all layers.
- Batch Normalization.
  - Normalize hidden layer outputs.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



Batch Normalization

- γ and β are learnt parameters.
- If required, Backpropagation can undo the effect of Batch normalization.

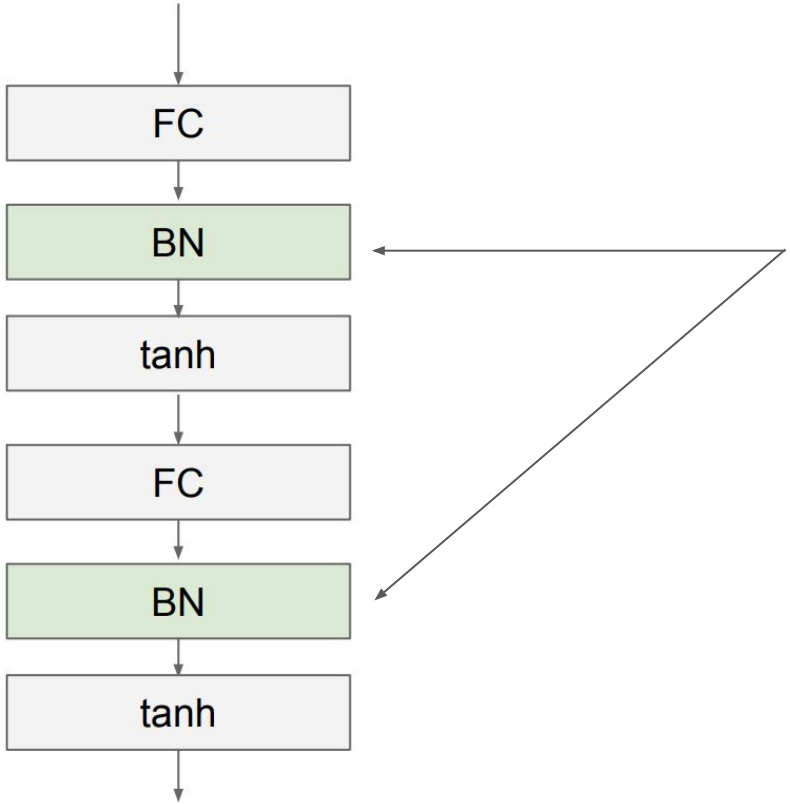
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization



Usually inserted after Fully Connected / (or Convolutional, as we'll see soon) layers, and before nonlinearity.