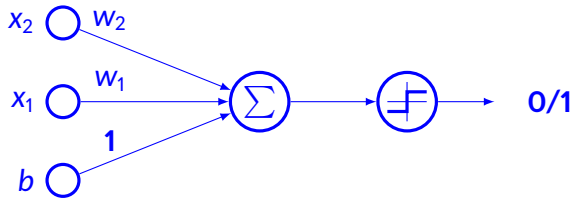
# Faculty Development Program
# on
# Machine Learning and Image Processing

# Neural Network

# History of basic model

- The first learning machine: the Perceptron
  - Built at Cornell, 1960
- Perceptron was linear classifier on top of simple feature extractor
- Most of the practical applications of ML today use glorified linear classifiers or glorified template matching.
- Significant effort is required for identifying relevant features
- Typically it will solve $y = sign\left(\sum_{i=1}^{N}(w_i \times f_i(X) + b)\right)$



# Biological Neuron

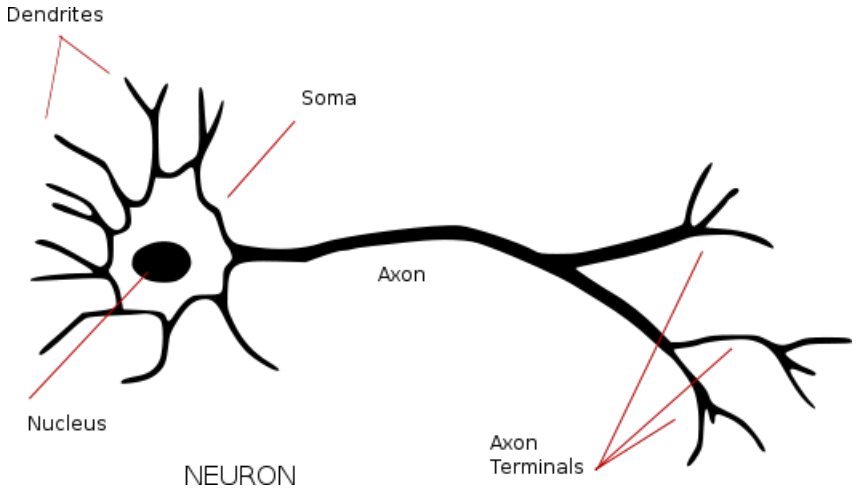



Image source: Internet

# Cerebral cortex

- It is a flat sheet of neurons about 2-3 millimeter thick with surface area is 2200 $cm^2$
  - Twice the area of computer keyboard
- It contains around $10^{11}$ neurons
  - Number of stars in the Milky-way
- Each neuron is connected to $10^3$-$10^4$ other neurons
- Total connections is around $10^{14}$-$10^{15}$
- Connectionist model

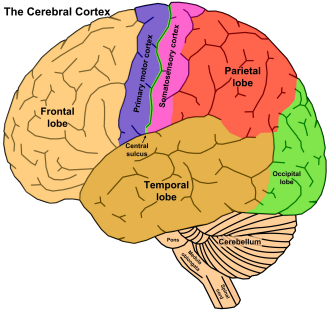
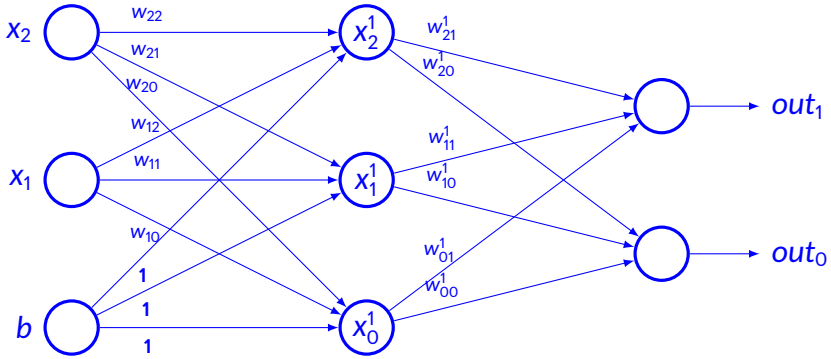


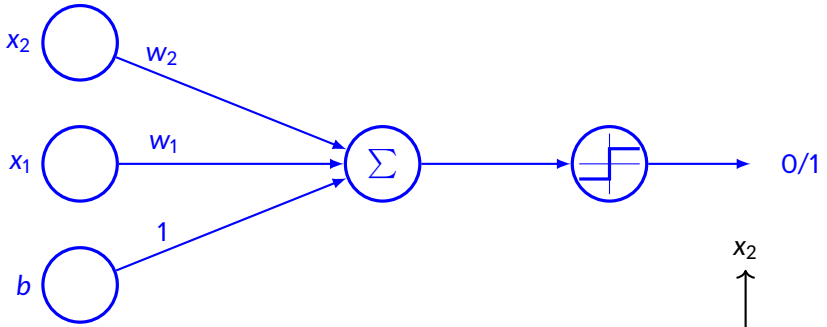

Image source: Internet

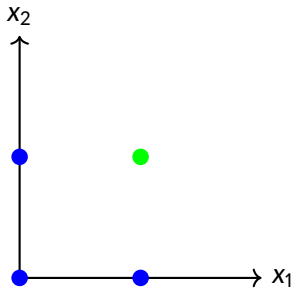
# Artificial Neural Network

- **A simple model**

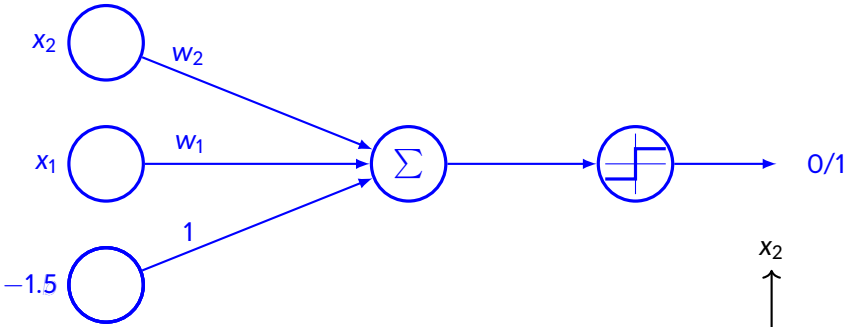


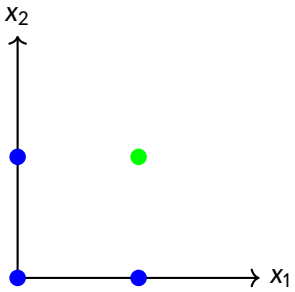
# Example NN: AND gate



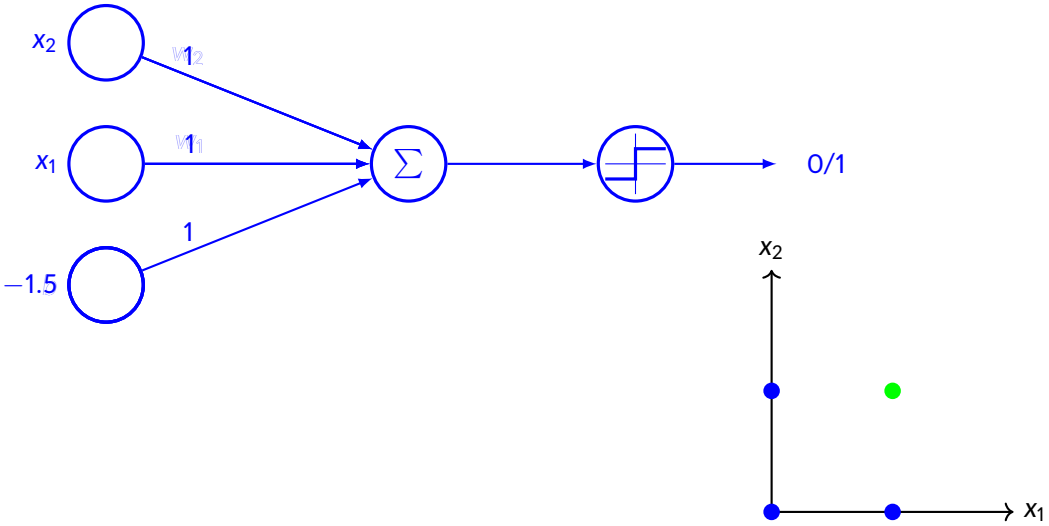



# Example NN: AND gate

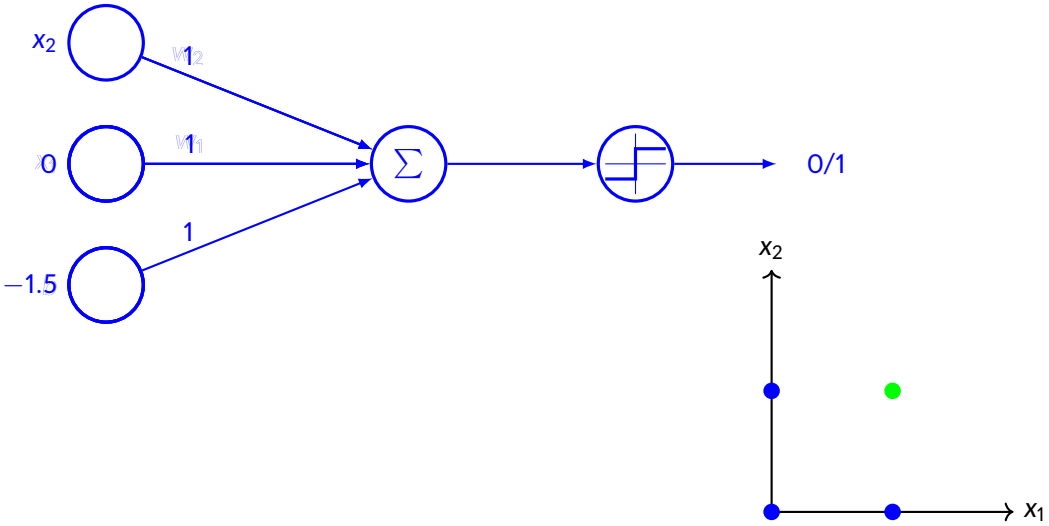


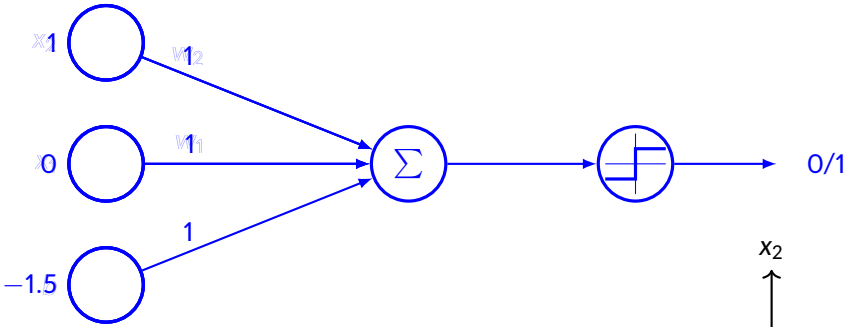
# Example NN: AND gate

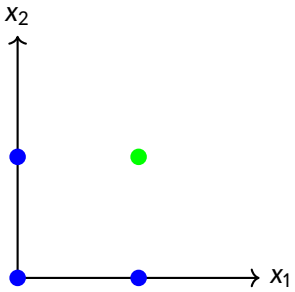


# Example NN: AND gate

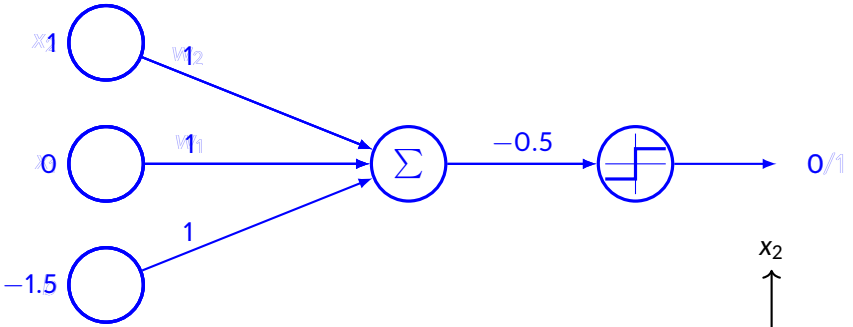


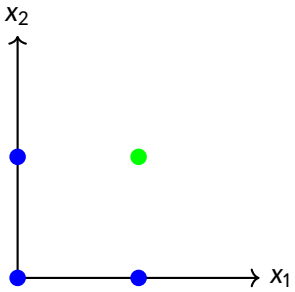
# Example NN: AND gate



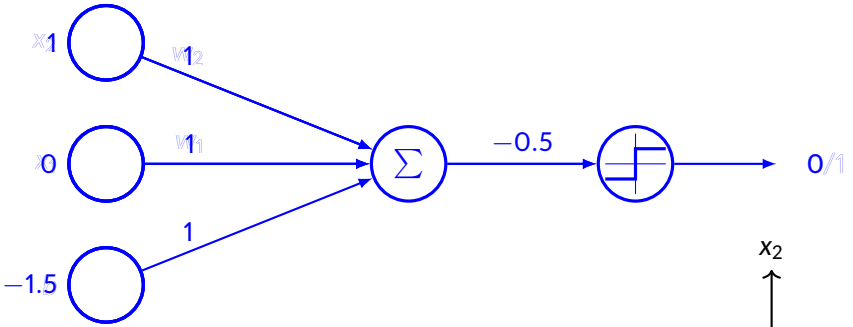



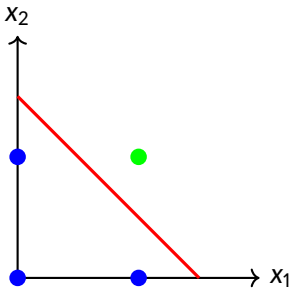
# Example NN: AND gate



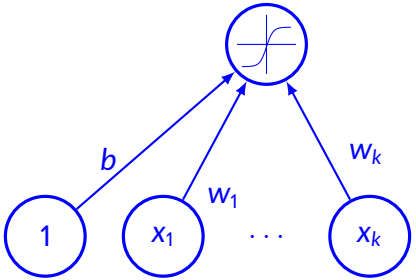
# Example NN: AND gate



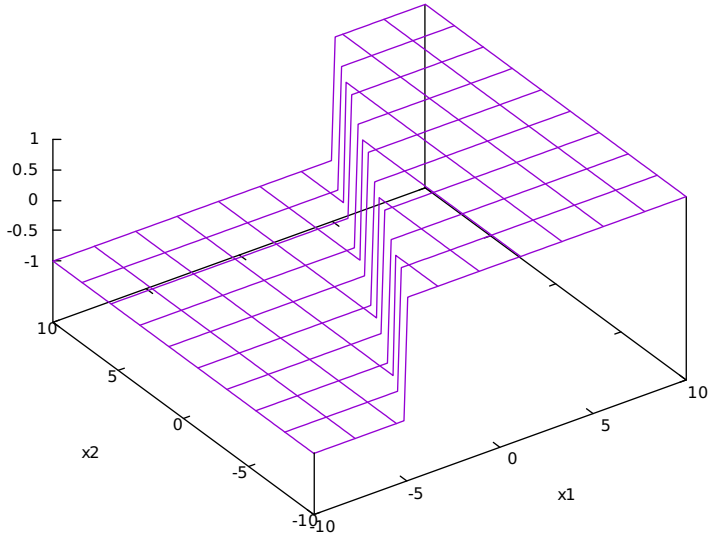



# Artificial Neuron

- **Neuron pre-activation function**
  - $a(\boldsymbol{x}) = \sum_i w_i x_i + b = b + \boldsymbol{w}^T \boldsymbol{x}$
- **Neuron output activation function**
  - $h(\boldsymbol{x}) = g(a(\boldsymbol{x})) = g\left(\sum_i w_i x_i + b\right)$
- **Notations**
  - $\boldsymbol{w}$ — Weight vector
  - $b$ — Neuron bias
  - $g(.)$ — Activation function

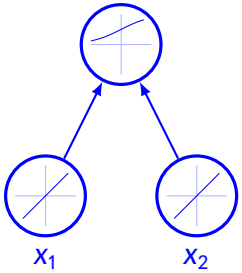


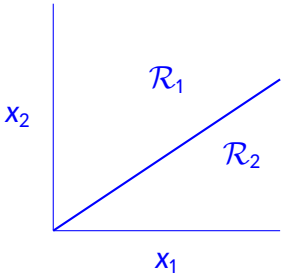
# Physical interpretation



# Classification using single neuron

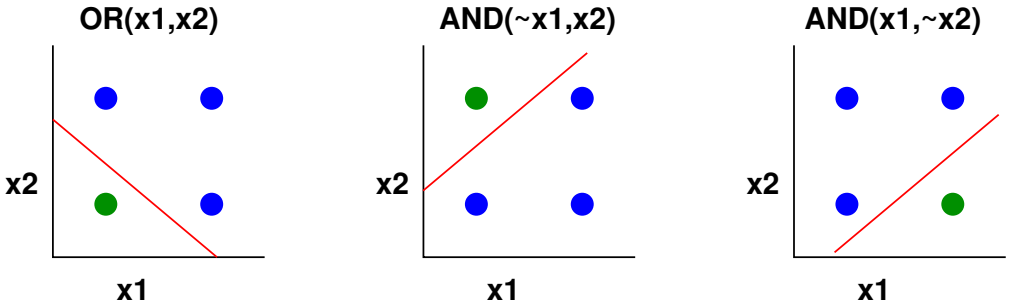
- **Single neuron can do binary classification**
  - **Also known as logistic regression classifier**



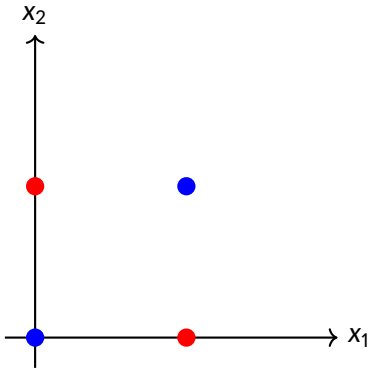



# Artificial neuron

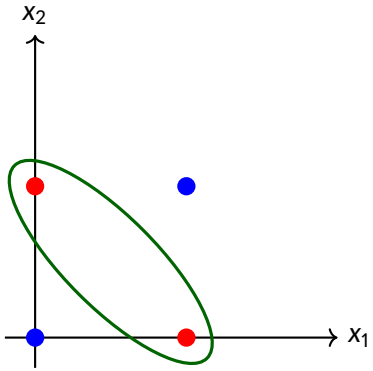
- Can solve linearly separable problems



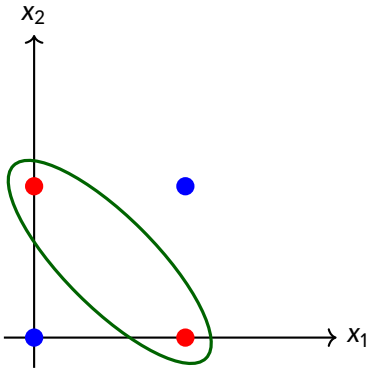
# Example NN: XOR gate

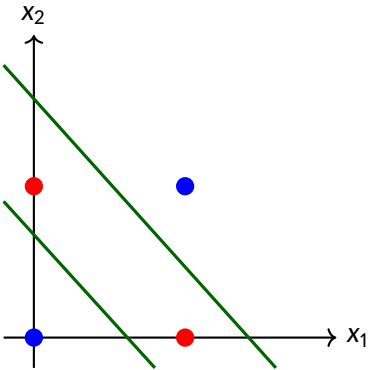


# Example NN: XOR gate



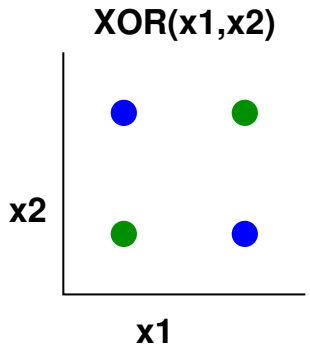
# Example NN: XOR gate

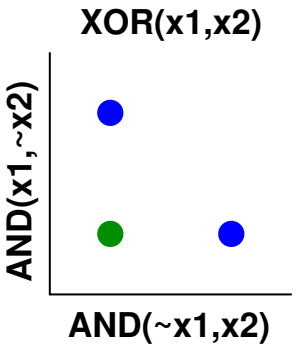


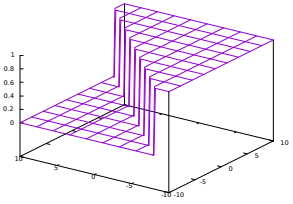
# Artificial neuron: XOR problem

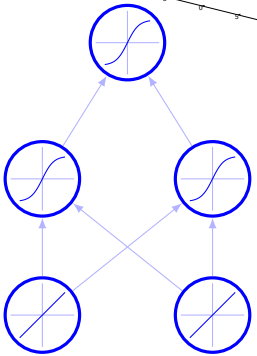
- There are issues for linear separation
- Transformation of representation can help in obtaining linearly separable data

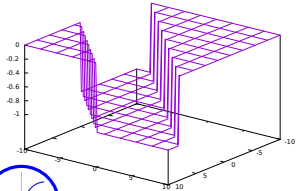


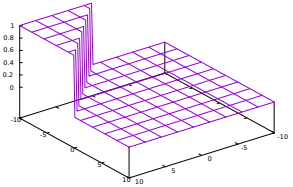



# Geometrical view of NN





# Capacity of neural network

- **Universal approximation theorem (Hornik,1991)**
  - **A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units.**
- **The result is applicable for other hidden layer activation functions such as sigmoid, tanh, etc.**
- **This is a promising result, but it does not say that there is a learning algorithm to find the necessary parameter values!**

# Deep Neural Network

# Deep feedforward networks

- **Also known as feedforward neural network or multilayer perceptron**

# Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function $f^*$
  - For a classifier, $\boldsymbol{x}$ is mapped to category $y$ ie. $y = f^*(\boldsymbol{x})$
  - A feedforward network maps $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation

# Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function $f^*$
  - For a classifier, $\boldsymbol{x}$ is mapped to category $y$ ie. $y = f^*(\boldsymbol{x})$
  - A feedforward network maps $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation
- Information flows from input to intermediate to output
  - No feedback, directed acyclic graph
  - For general model, it can have feedback and known as *recurrent neural network*

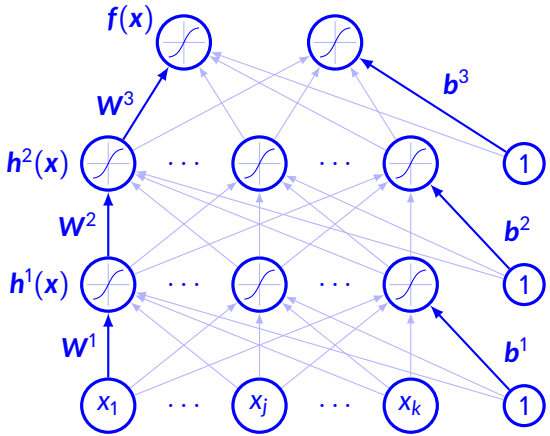
# Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function $f^*$
  - For a classifier, $\boldsymbol{x}$ is mapped to category $y$ ie. $y = f^*(\boldsymbol{x})$
  - A feedforward network maps $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation
- Information flows from input to intermediate to output
  - No feedback, directed acyclic graph
  - For general model, it can have feedback and known as *recurrent neural network*
- Typically it represents composition of functions
  - Three functions $f^{(1)}, f^{(2)}, f^{(3)}$ are connected in chain
  - Overall function realized is $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
  - The number of layers provides the depth of the model

# Deep feedforward networks

- Also known as feedforward neural network or multilayer perceptron
- Goal of such network is to approximate some function $f^*$
  - For a classifier, $\boldsymbol{x}$ is mapped to category $y$ ie. $y = f^*(\boldsymbol{x})$
  - A feedforward network maps $y = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns $\boldsymbol{\theta}$ for which the result is the best function approximation
- Information flows from input to intermediate to output
  - No feedback, directed acyclic graph
  - For general model, it can have feedback and known as *recurrent neural network*
- Typically it represents composition of functions
  - Three functions $f^{(1)}, f^{(2)}, f^{(3)}$ are connected in chain
  - Overall function realized is $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$
  - The number of layers provides the depth of the model
- Goal of NN is not to model brain accurately!

# Multilayer neural network



# Single hidden layer neural network

- **Hidden layer pre-activation**

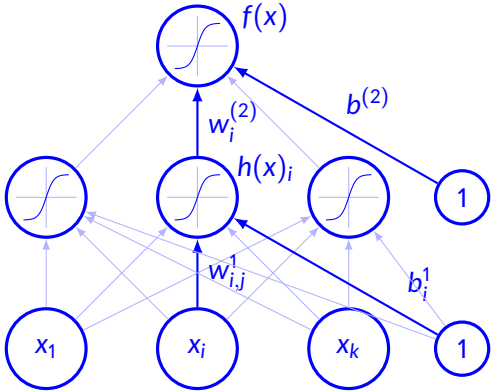
$$a(x) = b^1 + w^1 x$$

- **Hidden layer activation**

$$h(x) = g(a(x))$$

- **Output layer activation**

$$f(x) = o(b^{(2)} + w^{(2)} h^1(x))$$



# Multi layer neural network

- **Pre-activation in layer** $k > 0$ $(\boldsymbol{h}^{(0)}(\boldsymbol{x}) = x)$

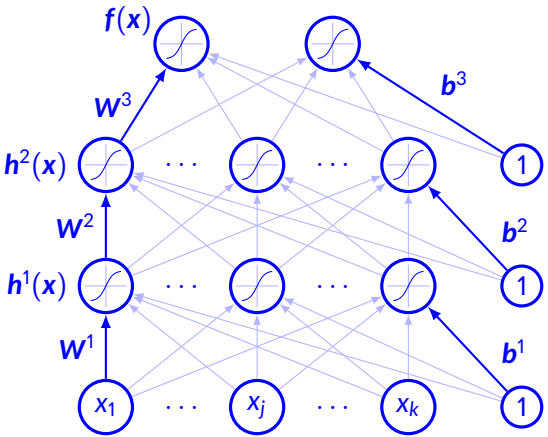
$$\boldsymbol{a}^{(k)}(\boldsymbol{x}) = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}\boldsymbol{x}$$

- **Hidden layer activation**

$$\boldsymbol{h}^{(k)}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{a}^{(k)}(\boldsymbol{x}))$$

- **Output layer activation**

$$\boldsymbol{h}^{(L+1)}(\boldsymbol{x}) = o(\boldsymbol{a}^{(L+1)}(\boldsymbol{x})) = \boldsymbol{f}(\boldsymbol{x})$$



# Issues with linear FFN

- Fits well for linear and logistic regression
- Convex optimization technique may be used
- Capacity of such function is limited
- Model cannot understand interaction between any two variables

# Overcome issues of linear FFN

- **Transform *x* (input) into $\phi(x)$ where $\phi$ is nonlinear transformation**

# Overcome issues of linear FFN

- Transform $\boldsymbol{x}$ (input) into $\phi(\boldsymbol{x})$ where $\phi$ is nonlinear transformation
- How to choose $\phi$?

# Overcome issues of linear FFN

- Transform $\boldsymbol{x}$ (input) into $\phi(\boldsymbol{x})$ where $\phi$ is nonlinear transformation
- How to choose $\phi$?
  - Use a very generic $\phi$ of high dimension
    - Enough capacity but may result in poor generalization
    - Very generic feature mapping usually based on principle of local smoothness
    - Do not encode enough prior information

# Overcome issues of linear FFN

- Transform $\boldsymbol{x}$ (input) into $\phi(\boldsymbol{x})$ where $\phi$ is nonlinear transformation
- How to choose $\phi$?
  - Use a very generic $\phi$ of high dimension
    - Enough capacity but may result in poor generalization
    - Very generic feature mapping usually based on principle of local smoothness
    - Do not encode enough prior information
  - Manually design $\phi$
    - Require domain knowledge

# Overcome issues of linear FFN

- Transform $\boldsymbol{x}$ (input) into $\phi(\boldsymbol{x})$ where $\phi$ is nonlinear transformation
- How to choose $\phi$?
  - Use a very generic $\phi$ of high dimension
    - Enough capacity but may result in poor generalization
    - Very generic feature mapping usually based on principle of local smoothness
    - Do not encode enough prior information
  - Manually design $\phi$
    - Require domain knowledge
  - Strategy of deep learning is to learn $\phi$

# Goal of deep learning

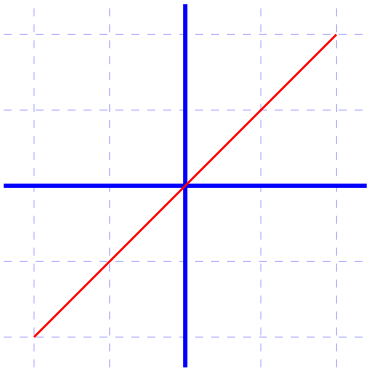
- We have a model $y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^T \boldsymbol{w}$
- We use $\boldsymbol{\theta}$ to learn $\phi$
- $\boldsymbol{w}$ and $\phi$ determines the output. $\phi$ defines the hidden layer
- It looses the convexity of the training problem but benefits a lot
- Representation is parameterized as $\phi(\boldsymbol{x}, \boldsymbol{\theta})$
  - $\boldsymbol{\theta}$ can be determined by solving optimization problem
- Advantages
  - $\phi$ can be very generic
  - Human practitioner can encode their knowledge to designing $\phi(\boldsymbol{x}; \boldsymbol{\theta})$

# Design issues of feedforward network

- Design of architecture - number of layers, number of units in each layer
- Choice of activation function
- The form of output unit
- Cost function
- Choice of optimizer
- Computation of gradients

# Activation function

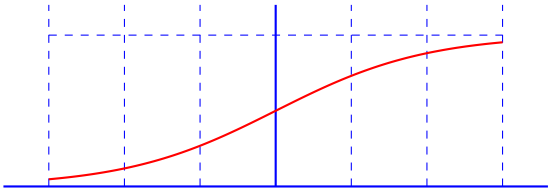
- **Linear activation function**
  - Not very interesting
  - No change in values
  - Huge range
- **Uniform gradient**



$g(x) = x$

# Activation function

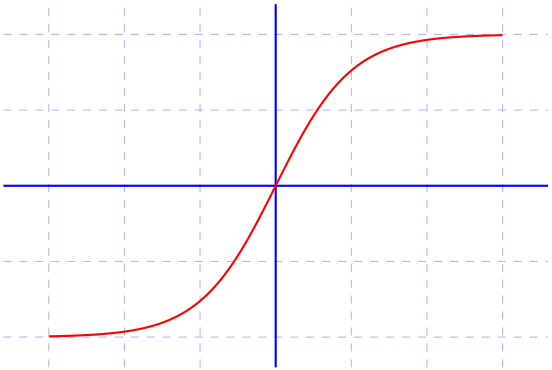
- **Sigmoid function**
  - Values lie between 0 and 1
  - Strictly increasing function
  - Bounded
- **Gradient saturates at the extreme end**



$$g(x) = \mathbf{sigm}(x) = \frac{1}{1 + \exp(-x)}$$

# Activation function

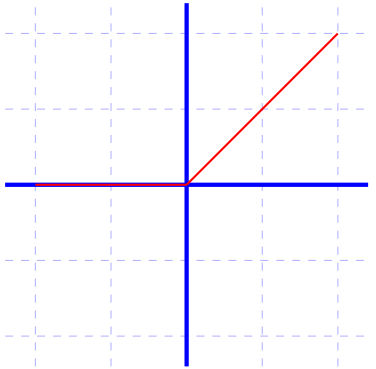
- **Hyperbolic Tangent (Tanh) function**
  - Can be positive or negative
  - Values lie between -1 and 1
  - Strictly increasing function
  - Bounded
- **Gradient saturates at the extreme end**



$$g(x) = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

# Activation function

- **Rectified linear activation function**
  - **Bounded below by 0**
  - **Strictly increasing function**
  - **Not upper bounded**
- **Gradient is 0 in left half**
- **Gradient is constant in right half**



$g(x) = \textbf{reclin}(x) = \max(0, x)$

# Generalization of ReLU

- **ReLU is defined as** $g(z) = \max\{0, z\}$
- **Using non-zero slope,** $h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$
  - **Absolute value rectification fixes** $\alpha_i = -1$ **to obtain** $g(z) = |z|$
- **Leaky ReLU assumes very small values for** $\alpha_i$
- **Parametric ReLU tries to learn** $\alpha_i$ **parameters**
- **Maxout unit** $g(\boldsymbol{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j$
  - **Suitable for learning piecewise linear function**

# Logistic sigmoid & hyperbolic tangent

- **Logistic sigmoid** $g(z) = \sigma(z)$
- **Hyperbolic tangent** $g(z) = tanh(z)$
  - $tanh(z) = 2\sigma(2z) - 1$
- **Widespread saturation of sigmoidal unit is an issue for gradient based learning**
  - **Usually discouraged to use as hidden units**
- **Usually, hyperbolic tangent function performs better where sigmoidal function must be used**
  - **Behaves linearly at 0**
  - **Sigmoidal activation function are more common in settings other than feedforward network**

# Other hidden units

- **Differentiable functions are usually preferred**
- **Activation function $h = \cos(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ performs well for MNIST data set**
- **Sometimes no activation function helps in reducing the number of parameters**
- **Radial Basis Function -** $\phi(\boldsymbol{x}, \boldsymbol{c}) = \phi(\|\boldsymbol{x} - \boldsymbol{c}\|)$
  - **Gaussian -** $\exp(-(\varepsilon r)^2)$
- **Softplus -** $g(x) = \zeta(x) = \log(1 + exp(x))$
- **Hard tanh -** $g(x) = \max(-1, \min(1, x))$

# Hidden units

- Active area of research and does not have good guiding theoretical principle
- Usually rectified linear unit (ReLU) is chosen in most of the cases
- Design process consists of trial and error, then the suitable one is chosen
- Some of the activation functions are not differentiable (eg. ReLU)
  - Still gradient descent performs well
  - Neural network does not converge to local minima but reduces the value of cost function to a very small value

# Output units

- Choice of cost function is directly related with the choice of output function
- In most cases cost function is determined by cross entropy between data and model distribution
- Any kind of output unit can be used as hidden unit

# Linear units

- Suited for Gaussian output distribution
- Given features $\boldsymbol{h}$, linear output unit produces $\hat{\boldsymbol{y}} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$
- This can be treated as conditional probability $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$
- Maximizing log-likelihood is equivalent to minimizing mean square error

# Sigmoid unit

- Mostly suited for binary classification problem that is Bernoulli output distribution
  - Discrete probability distribution of a random variable which takes the value either 0 or 1
- The neural networks need to predict $p(y = 1|\boldsymbol{x})$
  - If linear unit has been chosen, $p(y = 1|\boldsymbol{x}) = \max\left\{0, \min\{1, \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}\}\right\}$
  - Gradient?
- Model should have strong gradient whenever the answer is wrong
- Let us assume unnormalized log probability is linear with $z = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$
- Therefore, $\log \tilde{P}(y) = yz \Rightarrow \tilde{P}(y) = \exp(yz) \Rightarrow P(y) = \frac{\exp(yz)}{\sum_{y' \in \{0,1\}} \exp(y'z)}$
  - It can be written as $P(y) = \sigma((2y - 1)z)$
- The loss function for maximum likelihood is $J(\boldsymbol{\theta}) = -\log P(y|\boldsymbol{x}) = -\log \sigma((2y - 1)z) = \zeta((1 - 2y)z)$

# Softmax unit

- Similar to sigmoid. Mostly suited for multinoulli distribution
  - Discrete probability distribution that describes the possible results of a random variable that can take on one of K possible categories
- We need to predict a vector $\hat{\boldsymbol{y}}$ such that $\hat{y}_i = P(Y = i|\boldsymbol{x})$
- A linear layer predicts unnormalized probabilities $\boldsymbol{z} = \boldsymbol{W}^T\boldsymbol{h} + \boldsymbol{b}$ that is $z_i = \log \tilde{P}(y = i|\boldsymbol{x})$
- Formally, $\textbf{softmax}(\boldsymbol{z})_i = \dfrac{\exp z_i}{\sum_j \exp(z_j)}$
- Log in log-likelihood can undo exp $\log \ \textbf{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j \exp(z_j)$
  - Does it saturate?
  - What about incorrect prediction?
- Invariant to addition of some scalar to all input variables ie. $\textbf{softmax}(\boldsymbol{z}) = \textbf{softmax}(\boldsymbol{z} + \boldsymbol{c})$

# Multiclass classification

- Need multiple outputs that is one neuron for each class
- Need to determine probability of $p(y = c|\boldsymbol{x})$
- Softmax activation function is used at the output

$$\boldsymbol{o}(\boldsymbol{a}) = \textbf{softmax}(a) = \left[ \frac{\exp(a_1)}{\sum_c \exp(a_c)} \quad \frac{\exp(a_2)}{\sum_c \exp(a_c)} \quad \cdots \quad \frac{\exp(a_c)}{\sum_c \exp(a_c)} \right]^T$$

- Strictly positive
- Sum to 1
- Class having the highest probability will be the predicted output

# Example

- Let us choose XOR function
- Target function is $y = f^*(\boldsymbol{x})$ and our model provides $y = f(\boldsymbol{x}; \boldsymbol{\theta})$
- Learning algorithm will choose the parameters $\boldsymbol{\theta}$ to make $f$ close to $f^*$

# Example

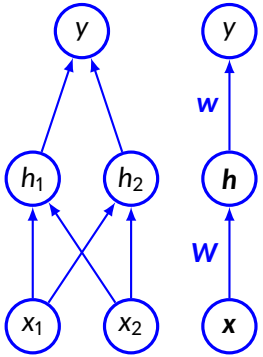
- Let us choose XOR function
- Target function is $y = f^*(\boldsymbol{x})$ and our model provides $y = f(\boldsymbol{x}; \boldsymbol{\theta})$
- Learning algorithm will choose the parameters $\boldsymbol{\theta}$ to make $f$ close to $f^*$
- Target is to fit output for $\boldsymbol{X} = \{[0, 0]^T, [0, 1]^T, [1, 0]^T, [1, 1]^T\}$
- This can be treated as regression problem and MSE error can be chosen as loss function ($J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \boldsymbol{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}))^2$)
- We need to choose $f(\boldsymbol{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ depends on $\boldsymbol{w}$ and $b$
- Let us consider a linear model $f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^T \boldsymbol{w} + b$

# Example

- Let us choose XOR function
- Target function is $y = f^*(\boldsymbol{x})$ and our model provides $y = f(\boldsymbol{x}; \boldsymbol{\theta})$
- Learning algorithm will choose the parameters $\boldsymbol{\theta}$ to make $f$ close to $f^*$
- Target is to fit output for $\boldsymbol{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$
- This can be treated as regression problem and MSE error can be chosen as loss function ($J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\boldsymbol{x}\in\boldsymbol{X}}(f^*(\boldsymbol{x}) - f(\boldsymbol{x};\boldsymbol{\theta}))^2$)
- We need to choose $f(\boldsymbol{x}; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ depends on $\boldsymbol{w}$ and $b$
- Let us consider a linear model $f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^T\boldsymbol{w} + b$
- Solving these, we get $\boldsymbol{w} = 0$ and $b = \frac{1}{2}$

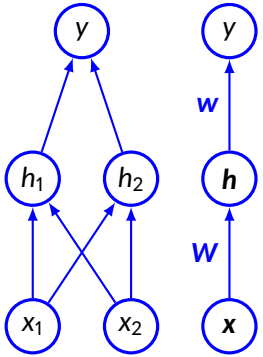
# Simple FFN with hidden layer

- Let us assume that the hidden unit ***h*** computes $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$



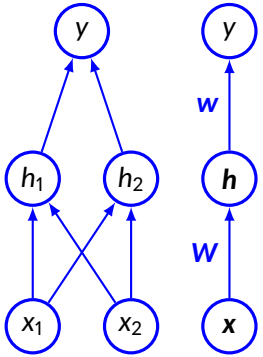
# Simple FFN with hidden layer

- Let us assume that the hidden unit $\boldsymbol{h}$ computes $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$
- In the next layer $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$ is computed



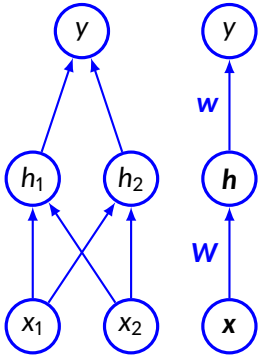
# Simple FFN with hidden layer

- Let us assume that the hidden unit $\boldsymbol{h}$ computes $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$
- In the next layer $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$ is computed
- Complete model $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$



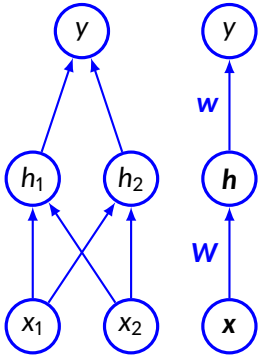
# Simple FFN with hidden layer

- Let us assume that the hidden unit $\boldsymbol{h}$ computes $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$
- In the next layer $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$ is computed
- Complete model $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$
- Suppose $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^T\boldsymbol{x}$ and $f^2(\boldsymbol{h}) = \boldsymbol{h}^T\boldsymbol{w}$



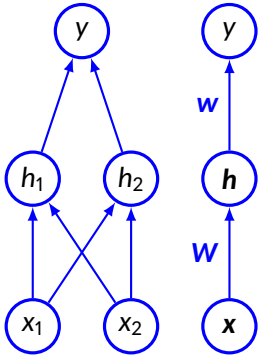
# Simple FFN with hidden layer

- Let us assume that the hidden unit $\boldsymbol{h}$ computes $f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$
- In the next layer $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$ is computed
- Complete model $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$
- Suppose $f^{(1)}(\boldsymbol{x}) = \boldsymbol{W}^T\boldsymbol{x}$ and $f^2(\boldsymbol{h}) = \boldsymbol{h}^T\boldsymbol{w}$ then $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{W}^T\boldsymbol{x}$



# Simple FFN with hidden layer (contd.)

- We need to have nonlinear function to describe the features
- Usually NN have affine transformation of learned parameters followed by nonlinear activation function
- Let us use $\boldsymbol{h} = g(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c})$
- Let us use ReLU as activation function $g(z) = \max\{0, z\}$
- $g$ is chosen element wise $h_i = g(\boldsymbol{x}^T\boldsymbol{W}_{:,i} + c_i)$



# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{c}\} + b$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X}$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$,

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW}$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix},$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, **add bias** $\boldsymbol{c}$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, **add bias** $\boldsymbol{c}$ $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$,

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, **add bias** $\boldsymbol{c}$ $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, **apply** $\boldsymbol{h}$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, **add bias** $\boldsymbol{c}$ $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, **apply** $\boldsymbol{h}$ $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$,

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, **add bias** $\boldsymbol{c}$ $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, **apply** $\boldsymbol{h}$ $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, **multiply with** $\boldsymbol{w}$

# Simple FFN with hidden layer (contd.)

- **Complete network is** $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^T \max\{0, \boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{c}\} + b$
- **A solution for XOR problem can be as follows**
  - $\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$
- **Now we have**
  - $\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$, **add bias** $\boldsymbol{c}$ $\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, **apply** $\boldsymbol{h}$ $\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$, **multiply with** $\boldsymbol{w}$ $\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

# Performance measure

- Accuracy is one of the key measures
  - The proportion of examples for which the model produces correct outputs
  - Similar to error rate
    - Error rate often referred as expected 0-1 loss
- Mostly interested how ML/DL algorithm performs on unseen data
- Choice of performance measure may not be straight forward
  - Transcription
    - Accuracy of the system at transcribing entire sequence
    - Any partial credit for some elements of the sequence are correct
- Typically MSE is used as performance measure for regression problem and cross-entropy for classification

# Cost function

- Similar to other parametric model like linear models
- Parametric model defines distribution $p(\boldsymbol{y}|\boldsymbol{x};\theta)$
- Principle of maximum likelihood is used (cross entropy between training data and model prediction)
- Instead of predicting the whole distribution of *y*, some statistic of *y* conditioned on *x* is predicted
- It can also contain regularization term

# Maximum likelihood estimation

- Consider a set of $m$ examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\boldsymbol{x})$
- Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution

# Maximum likelihood estimation

- Consider a set of $m$ examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\boldsymbol{x})$
- Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{model}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{model}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

# Maximum likelihood estimation

- Consider a set of $m$ examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\boldsymbol{x})$
- Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{model}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{model}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

- It can be written as $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{model}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$

# Maximum likelihood estimation

- Consider a set of $m$ examples $\mathbb{X} = \{x^{(1)}, \ldots, x^{(m)}\}$ drawn independently from the true but unknown data generating distribution $p_{data}(\boldsymbol{x})$
- Let $p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$ be a parametric family of probability distribution
- Maximum likelihood estimator for $\boldsymbol{\theta}$ is defined as

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} p_{model}(\mathbb{X}; \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{model}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

- It can be written as $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{model}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$
- By dividing $m$ we get $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim p_{data}} \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$

# Maximum likelihood estimation (cont.)

- **Minimizing dissimilarity between the empirical $\hat{p}_{data}$ and model distribution $p_{model}$ and it is measured by KL divergence**

$$D_{KL}(\hat{p}_{data} || p_{model}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{data}} [\log \hat{p}_{data}(\boldsymbol{x}) - \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta})]$$

# Maximum likelihood estimation (cont.)

- **Minimizing dissimilarity between the empirical $\hat{p}_{data}$ and model distribution $p_{model}$ and it is measured by KL divergence**

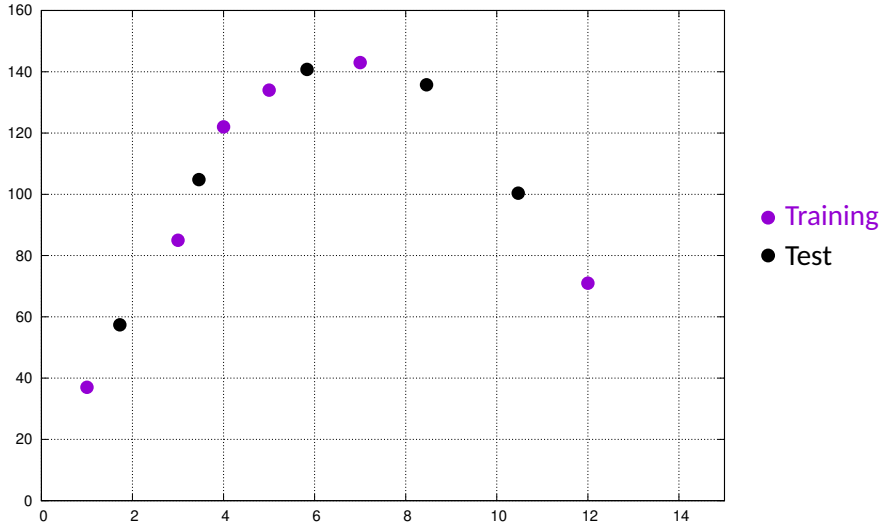
  $$D_{KL}(\hat{p}_{data} || p_{model}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{data}} [\log \hat{p}_{data}(\boldsymbol{x}) - \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta})]$$

- **We need to minimize** $-\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x} \sim \hat{p}_{data}} \log p_{model}(\boldsymbol{x}; \boldsymbol{\theta})$

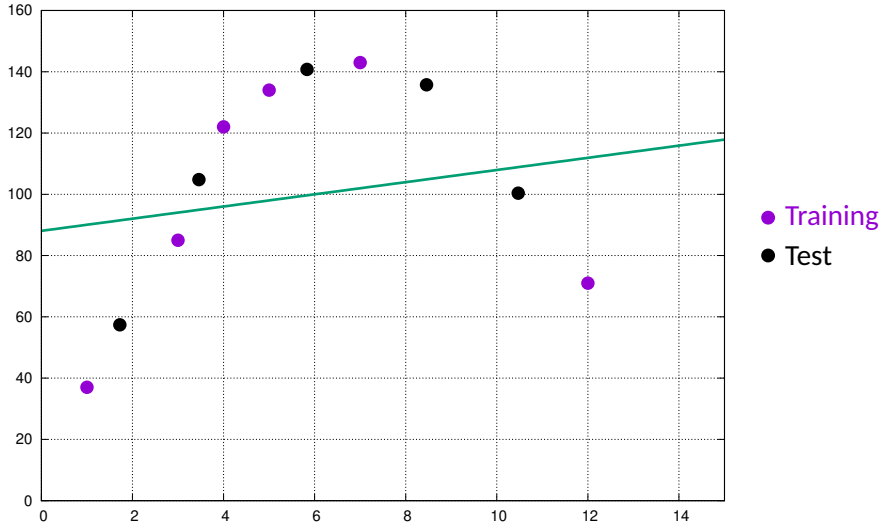
# Error

- Training error - Error obtained on a training set
- Generalization error - Error on unseen data
- Data assumed to be independent and identically distributed (iid)
  - Each data set are independent of each other
  - Train and test data are identically distributed
- Expected training and test error will be the same
- It is more likely that the test error is greater than or equal to the expected value of training error
- Target is to make the training error is small. Also, to make the gap between training and test error smaller
  - Overfitting vs Underfitting - described next

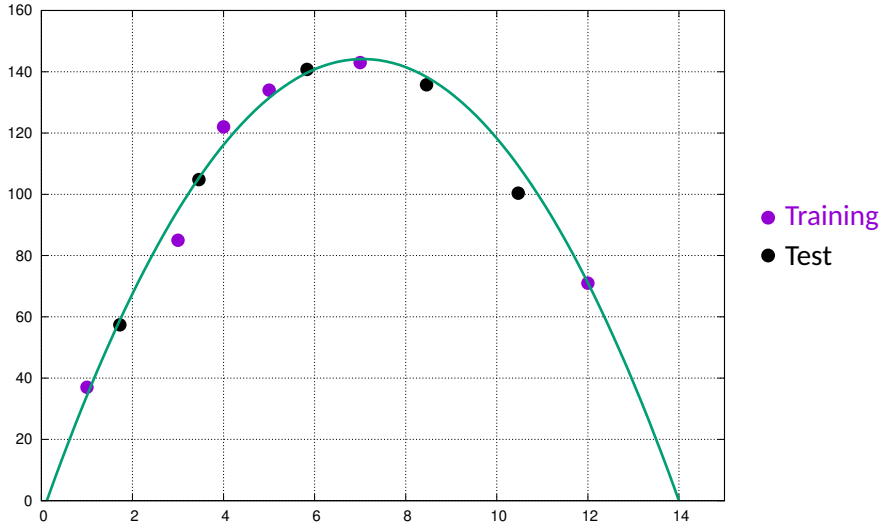
# Regression example



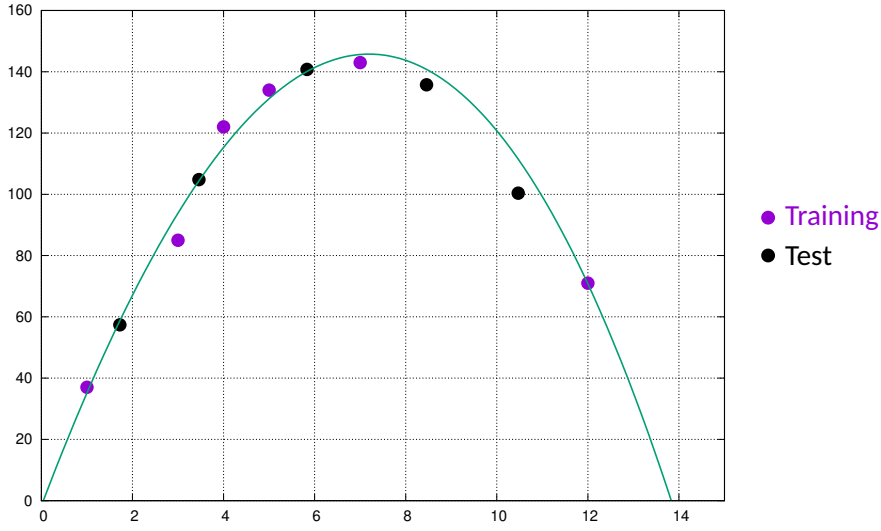
# Regression example: degree 1



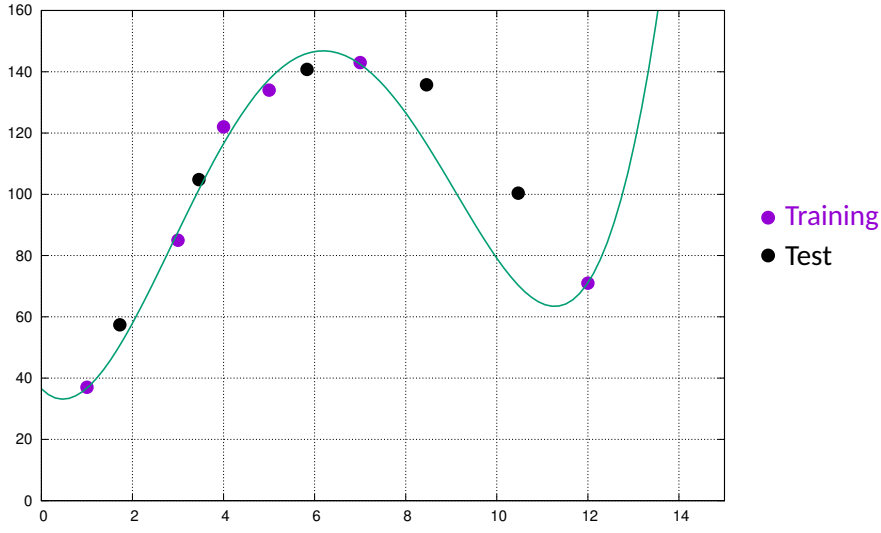
# Regression example: degree 2



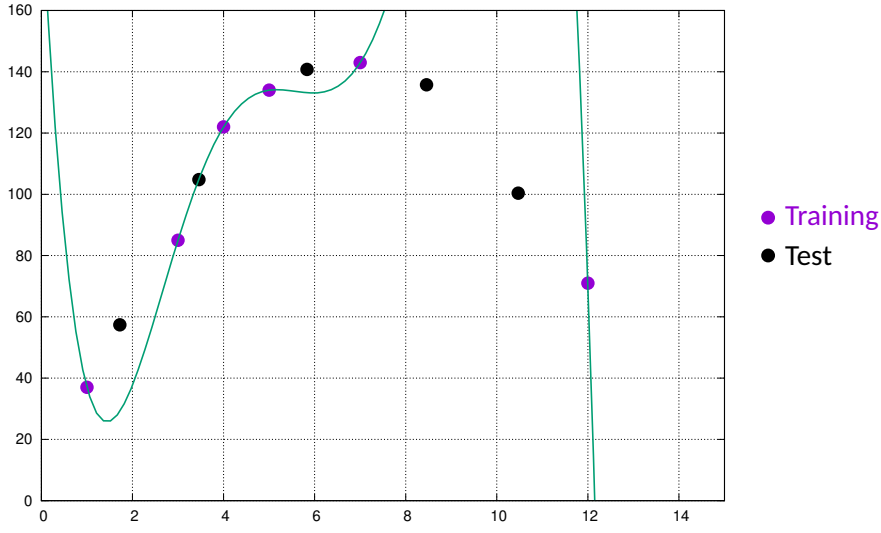
# Regression example: degree 3



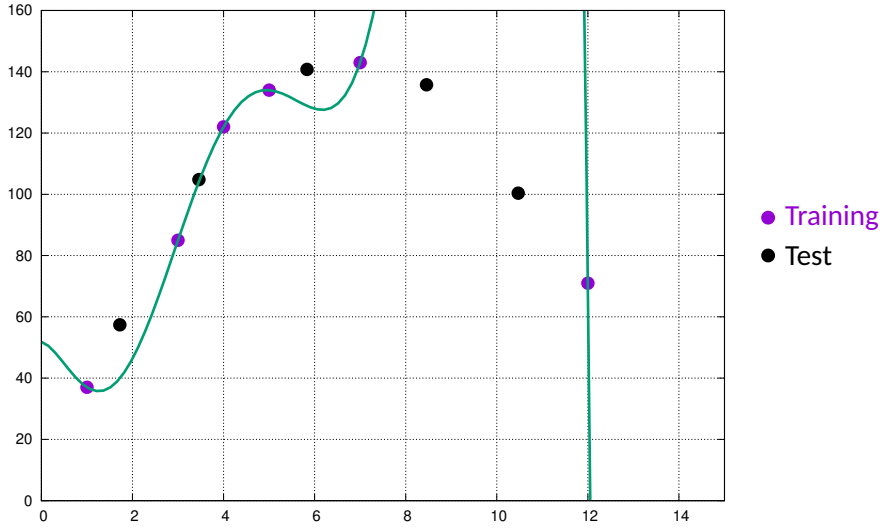
# Regression example: degree 4



# Regression example: degree 5



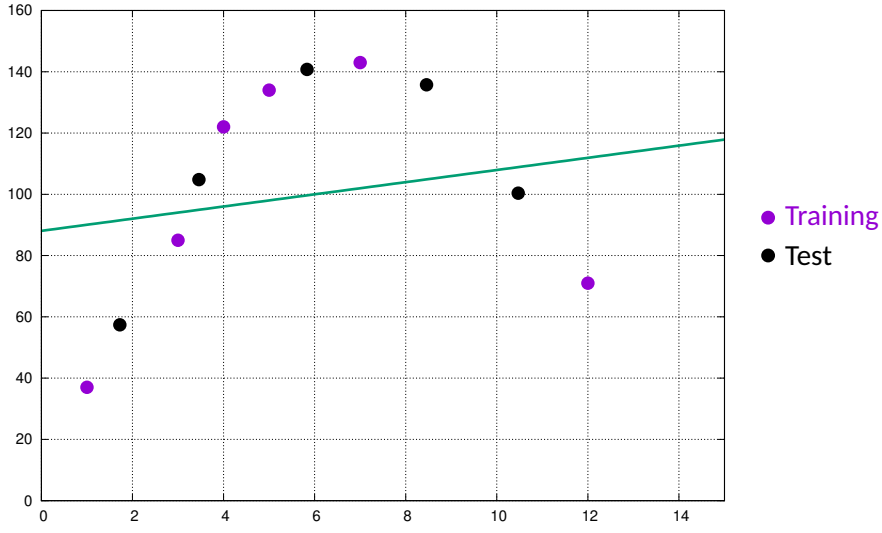
# Regression example: degree 6



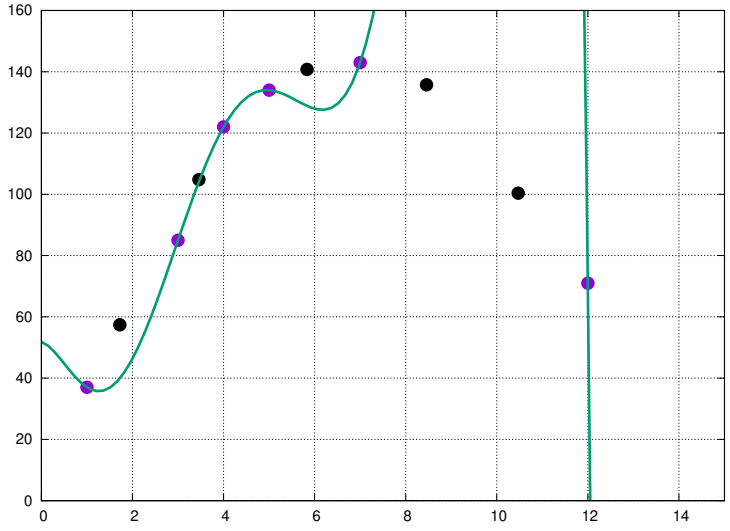
# Underfitting & Overfitting

- **Underfitting**
  - When the model is not able to obtain sufficiently low error value on the training set
- **Overfitting**
  - When the gap between training set and test set error is too large

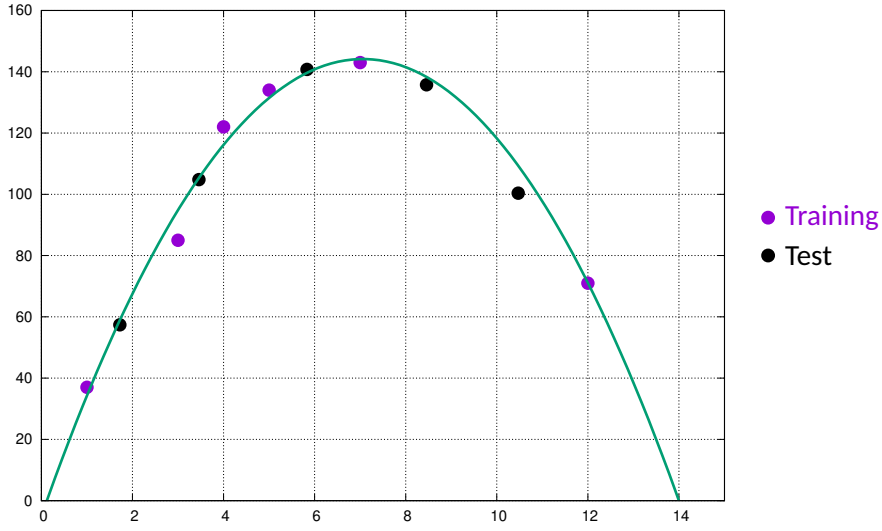
# Underfitting example



# Overfitting example



# Better fit



# Capacity

- **Ability to fit wide variety of functions**
  - **Low capacity will struggle to fit the training set**
  - **High capacity will can overfit by memorizing the training set**
- **Capacity can be controlled by choosing hypothesis space**
  - **A polynomial of degree 1 gives linear regression** $\hat{y} = b + wx$
  - **By adding** $x^2$ **term, it can learn quadratic curve** $\hat{y} = b + w_1 x + w_2 x^2$
    - **Output is still a linear function of parameters**
- **Capacity of is determined by the choice of model (Representational capacity)**
- **Finding best function is a very difficult optimization problem**
  - **Learning algorithm does not find the best function but reduces the training error**
  - **Imperfection in optimization algorithm can further reduce the capacity of model (effective capacity)**

# Error vs Capacity

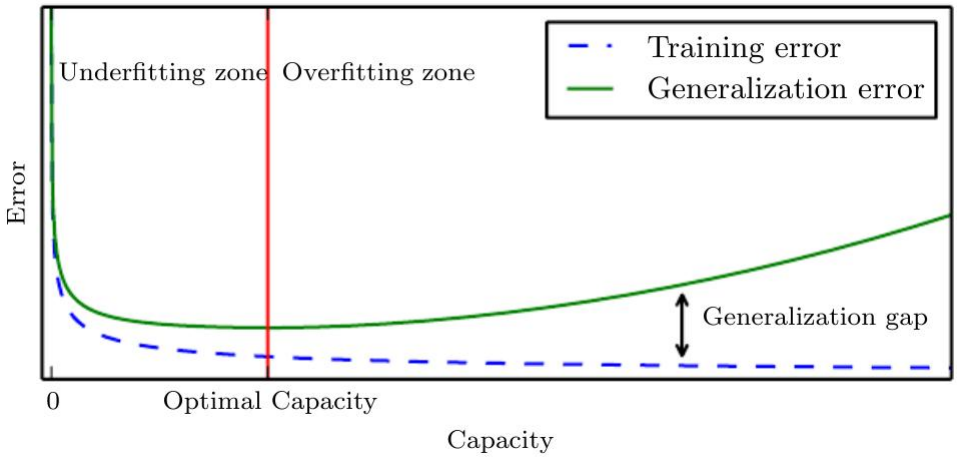



Image source: Deep Learning Book

# Note

- Training and generalization error varies as the size of training set varies
- Expected generalization error can never increase as the number of training example increases
- Any fixed parametric model with less than the optimal capacity will asymptote to an error value that exceeds the Bayes error
- It is possible to have optimal capacity but have large gap between training and generalization error
  - Need more training examples

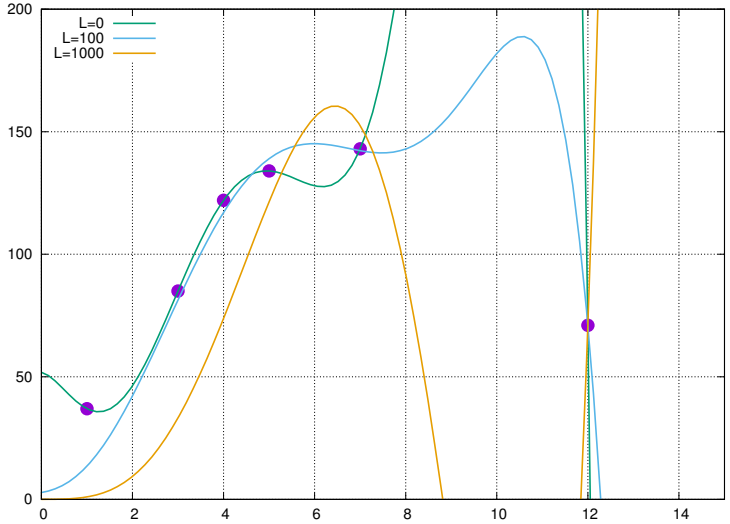
# Regularization

- A set of preferences is applied to learning algorithm so that it performs well on a specific task
- Weight decay - In linear regression, preference on the weights is introduced
  - Sum of MSE and squared $L^2$ norms of the weight is minimized ie.

$$J(\boldsymbol{w}) = \text{MSE}_{train} + \lambda \boldsymbol{w}^T \boldsymbol{w}$$

  - $\lambda = 0$ - No preference
  - $\lambda$ becomes large - weight becomes smaller
- Regularization is intended to reduce test error not training error

# Example: Weight decay



# Hyperparameters

- Settings that are used to control the behavior of learning algorithm
  - Degree of polynomial
  - $\lambda$ for decay weight
- Hyperparameters are usually not adapted or learned on the training set

# Validation set

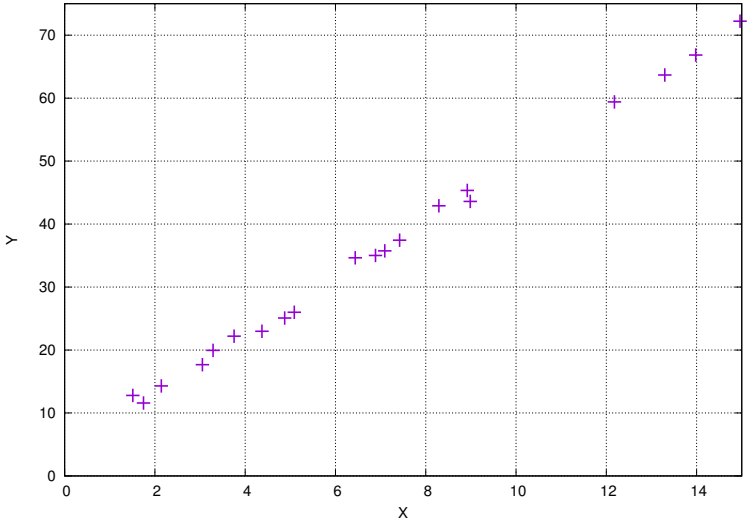
- Test data should not be used to choose the model as well as hyperparameters
- Validation set is constructed from training set
  - Typically 80% will be used for training and rest for validation
- Validation set may be used to train hyperparameters

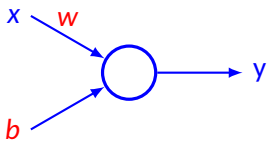
# Gradient descent, SGD

# Gradient based learning

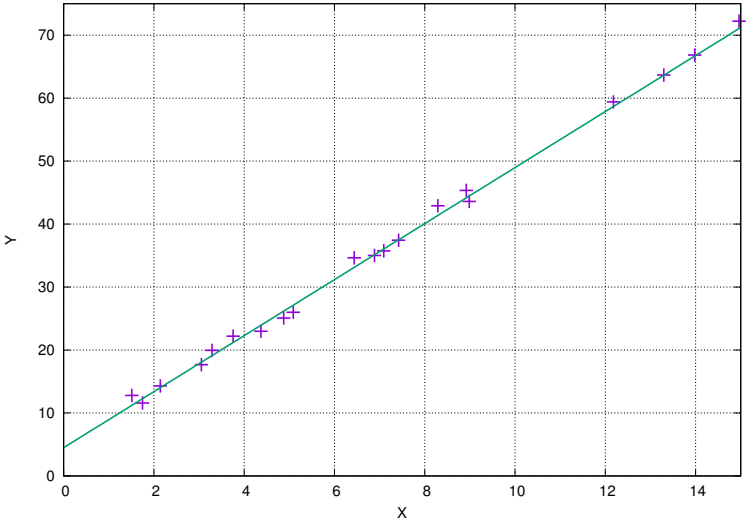
- Similar to machine learning tasks, gradient descent based learning is used
  - Need to specify optimization procedure, cost function and model family
- For NN, model is nonlinear and function becomes nonconvex
  - Usually trained by iterative, gradient based optimizer
- Solved by using gradient descent or stochastic gradient descent (SGD)

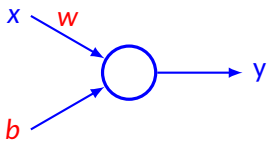
# Regression example



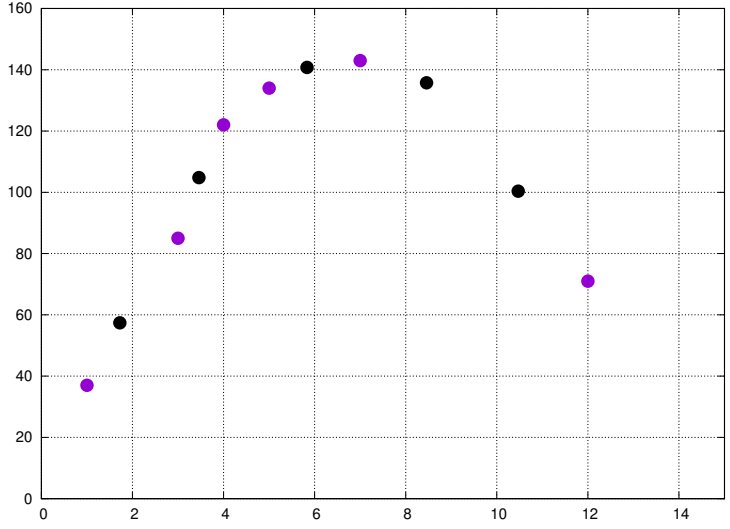



# Regression example



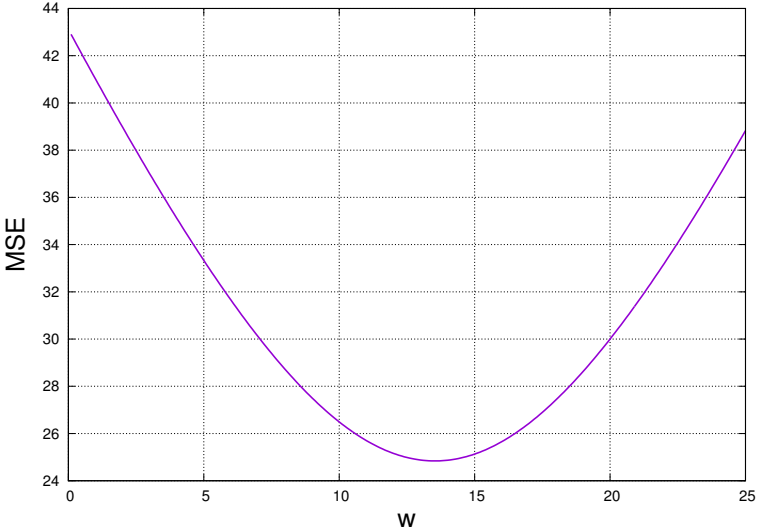
# Example



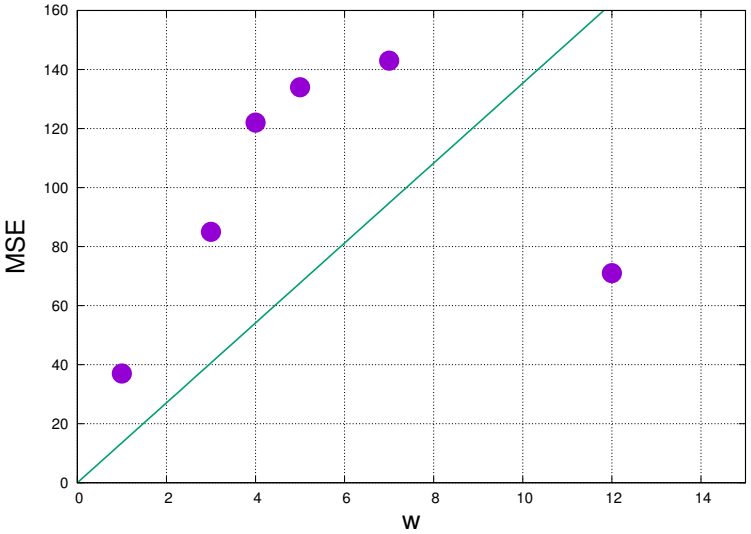
NOTE:

- The points are taken from a square curve. We would like to fit a straight line for these set of points passing through the origin. It means we are trying to fit a line of the form $y = w \times x$. Let us plot MSE value by varying $w$.

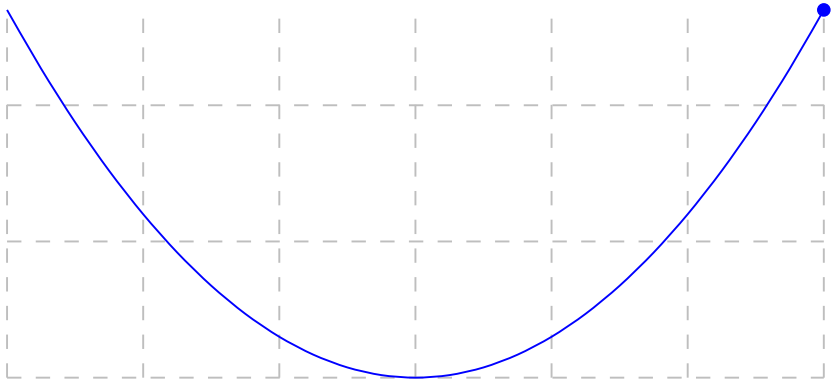
# Example: Variation of MSE wrt *w*



# Example: Best fit



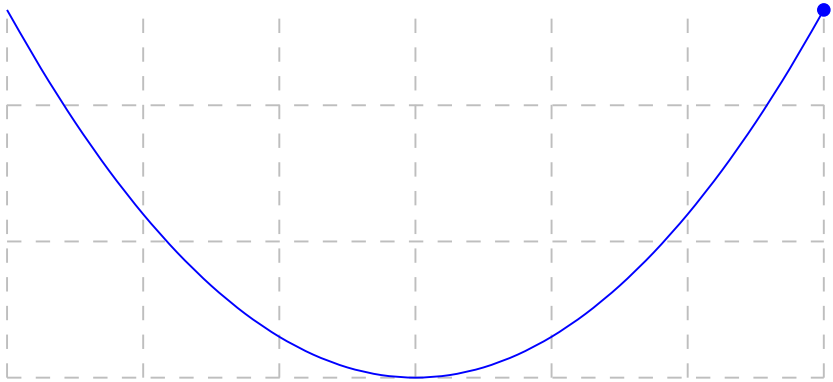
# Gradient descent



$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=1.80002

# Gradient descent

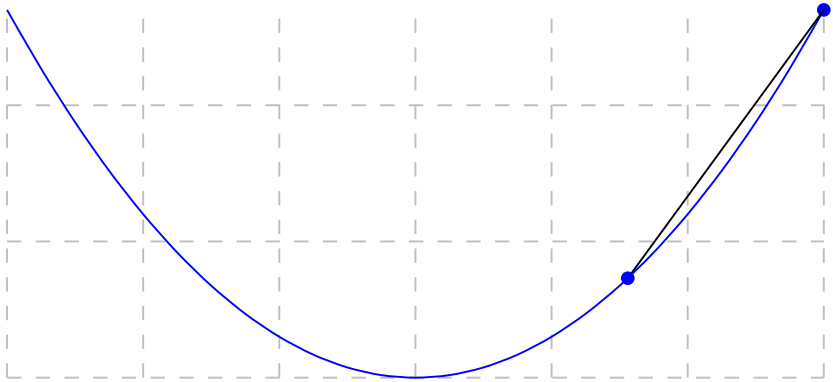


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=1.80002

$x_{new}$=1.56001

# Gradient descent

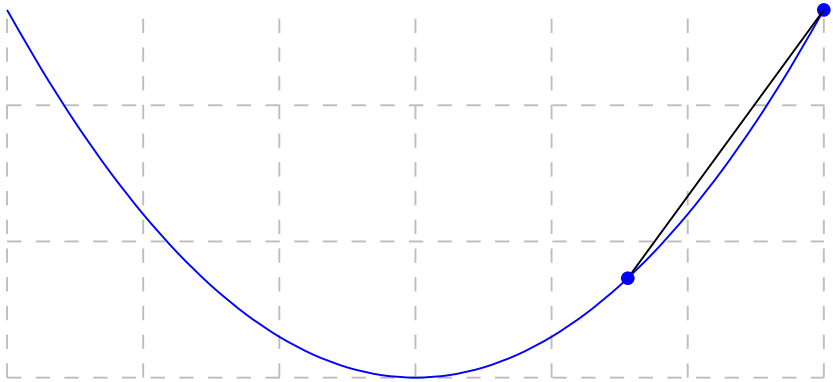


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=1.80002

$x_{new}$=1.56001

# Gradient descent

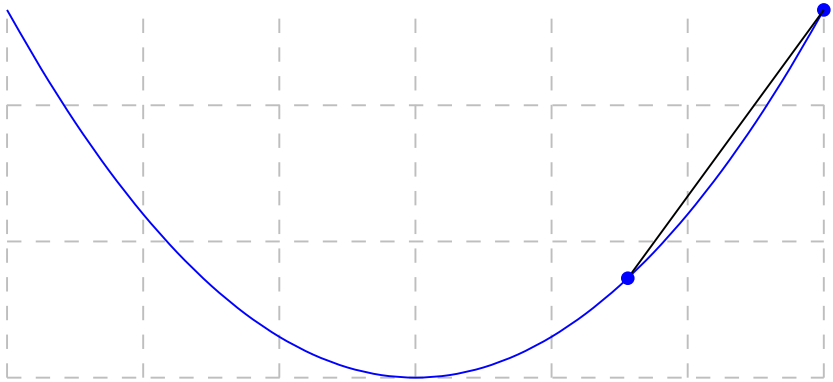


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.936

$x_{new}$=1.56001

# Gradient descent

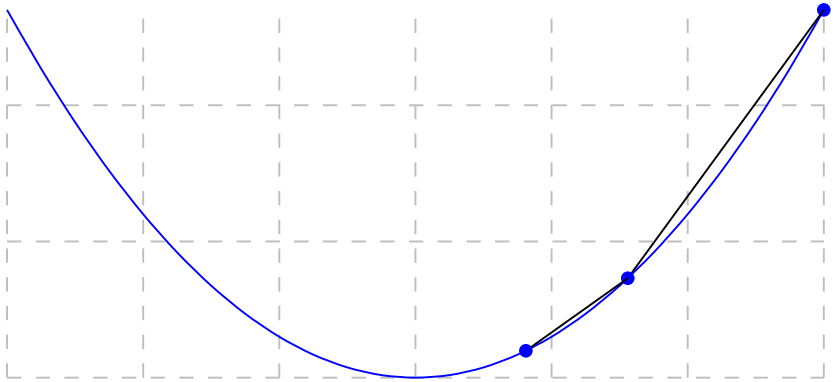


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.936

$x_{new}$=0.81122

# Gradient descent

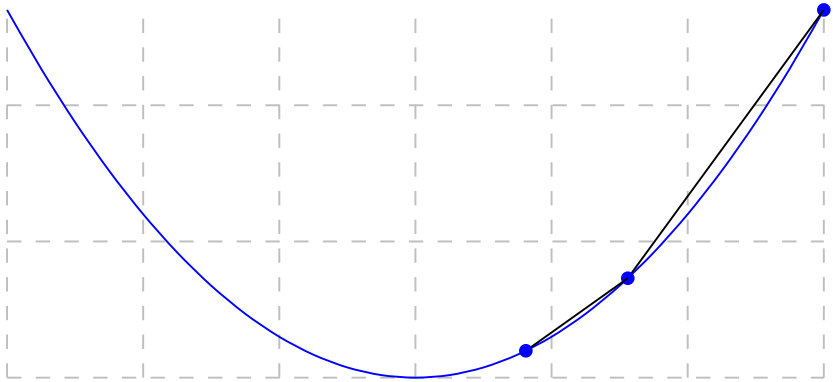


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.936

$x_{new}$=0.81122

# Gradient descent

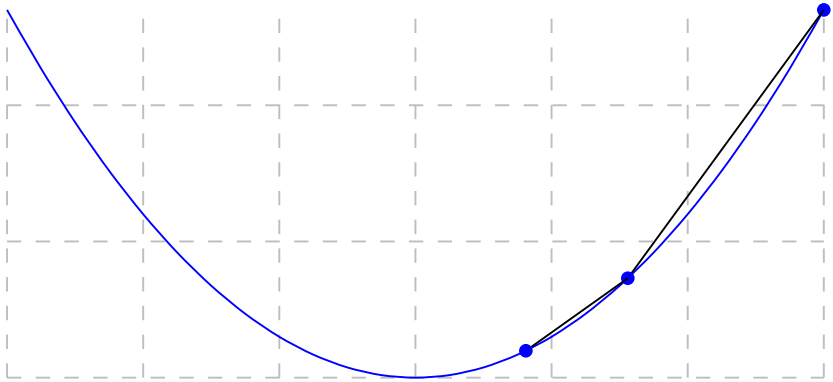


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.48672

$x_{new}$=0.81122

# Gradient descent

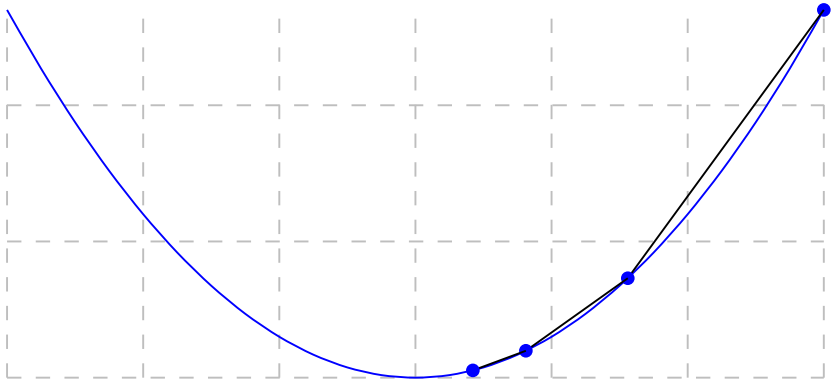


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.48672

$x_{new}$=0.42184

# Gradient descent

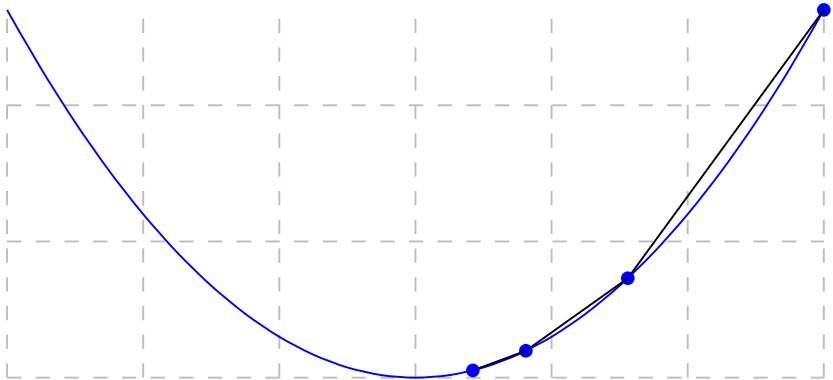


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.48672

$x_{new}$=0.42184

# Gradient descent

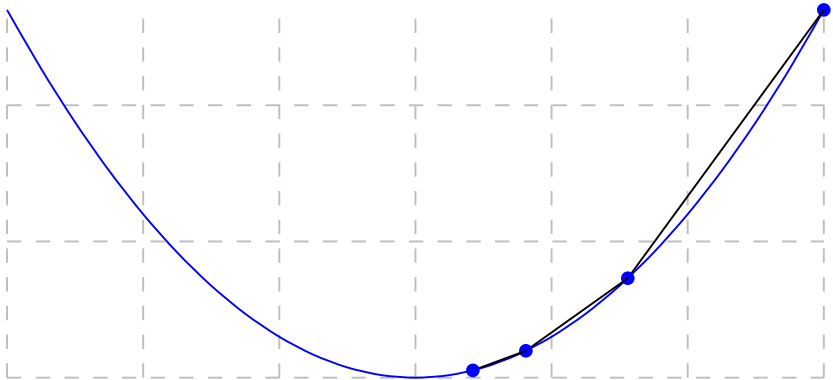


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.2531

$x_{new}$=0.42184

# Gradient descent

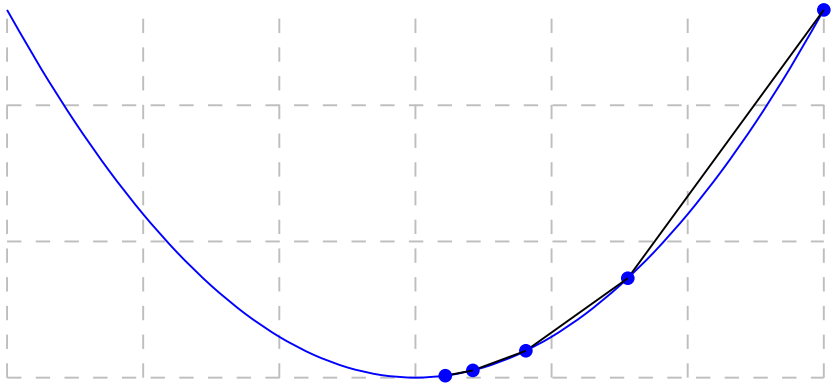


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.2531

$x_{new}$=0.21938

# Gradient descent

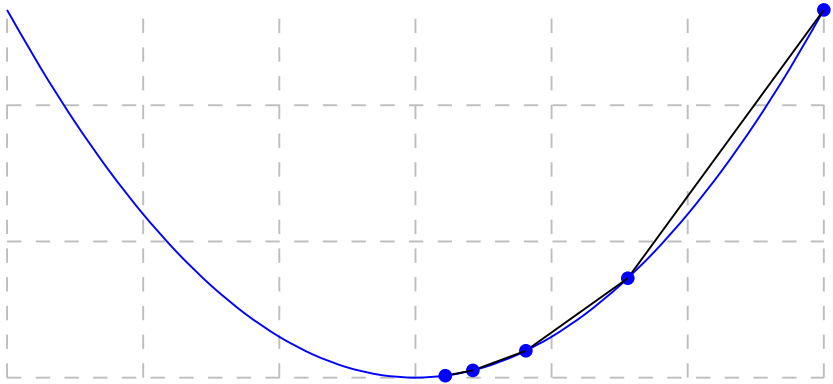


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.2531

$x_{new}$=0.21938

# Gradient descent

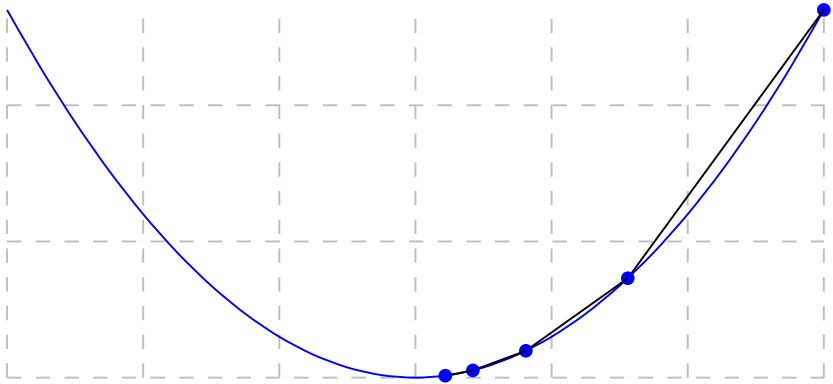


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.13162

$x_{new}$=0.21938

# Gradient descent

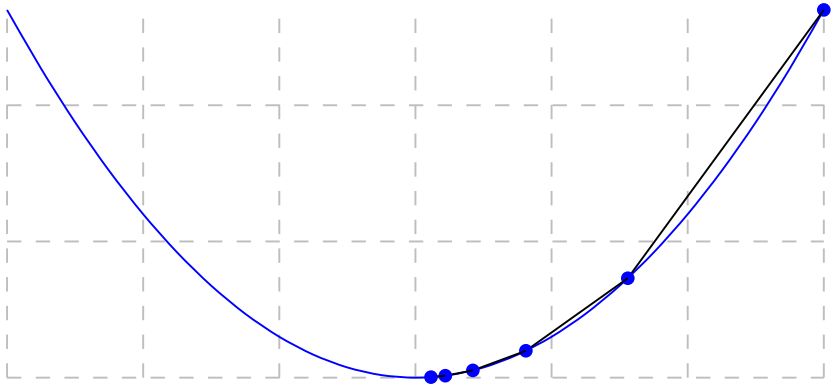


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.13162

$x_{new}$=0.11409

# Gradient descent

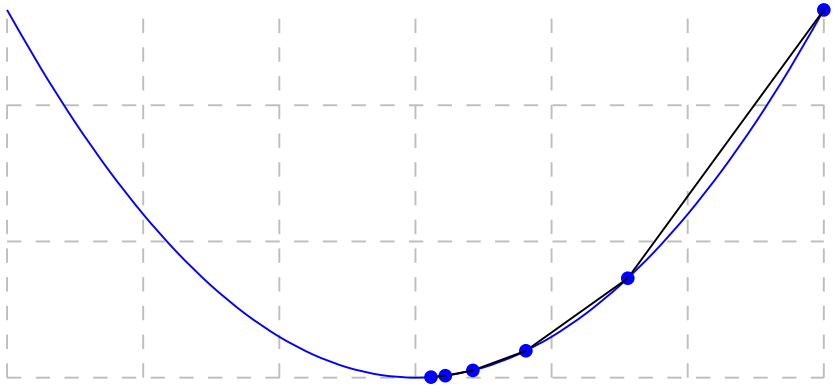


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.13162

$x_{new}$=0.11409

# Gradient descent

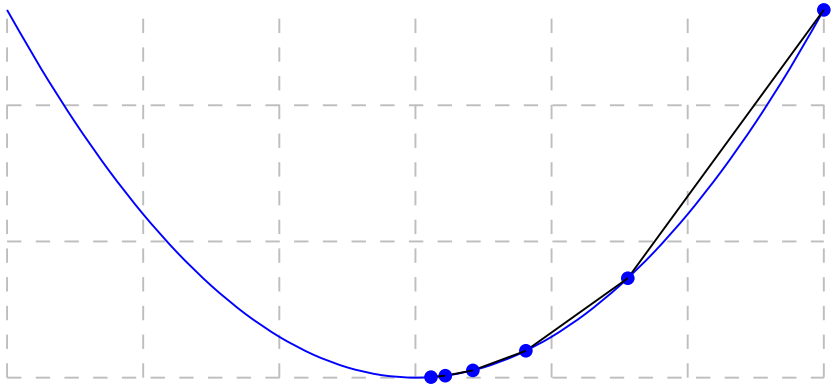


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.06845

$x_{new}$=0.11409

# Gradient descent

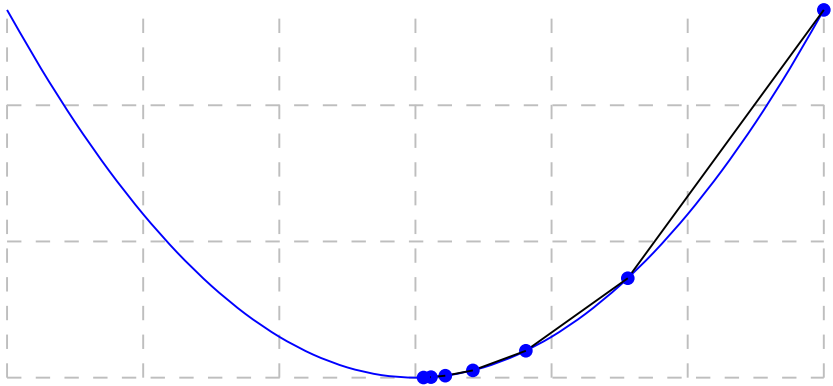


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.06845

$x_{new}$=0.05934

# Gradient descent

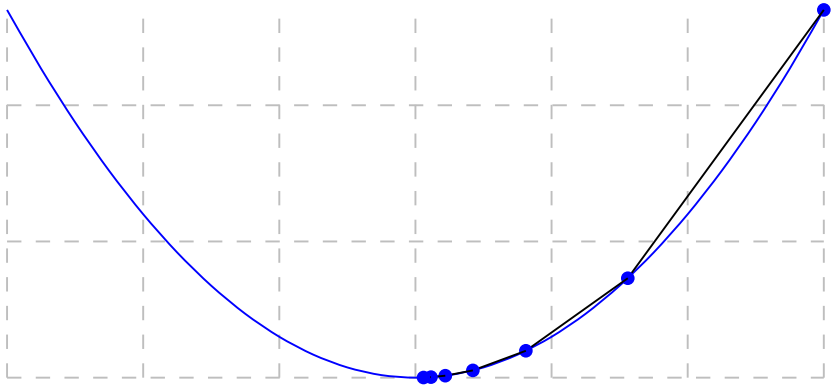


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.06845

$x_{new}$=0.05934

# Gradient descent

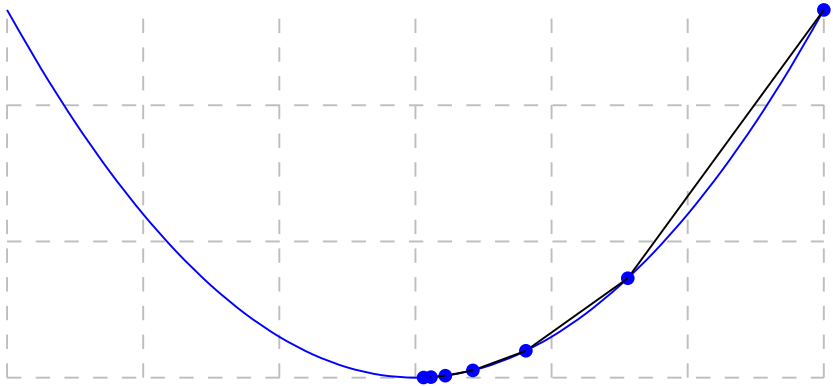


$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.0356

$x_{new}$=0.05934

# Gradient descent



$y = 0.3x^2, x_0 = 3, \alpha = 0.8$

gradient=0.0356

$x_{new}$=0.03087

# Minimization of MSE: Gradient descent

- **Assuming $\text{MSE}_{\text{(train)}} = J(w_1, w_2)$**
- **Target is to $\min_{w_1, w_2} J(w_1, w_2)$**
- **Approach**
  - **Start with some $w_1, w_2$**
  - **Keep modifying $w_1, w_2$ so that $J(w_1, w_2)$ reduces till the desired accuracy is achieved**

# Minimization of MSE: Gradient descent

- Assuming $\text{MSE}_{\text{(train)}} = J(w_1, w_2)$
- Target is to $\min_{w_1, w_2} J(w_1, w_2)$
- Approach
  - Start with some $w_1, w_2$
  - Keep modifying $w_1, w_2$ so that $J(w_1, w_2)$ reduces till the desired accuracy is achieved
- Algorithm
  - Repeat the following until convergence

$$w_j = w_j - \frac{\partial}{\partial w_j} J(w_1, w_2)$$

# Gradient descent

- For a function $y = f(x)$, derivative (slope at point $x$) of it is $f'(x) = \frac{dy}{dx}$
- A small change in the input can cause output to move to a value given by $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$
- We need to take a jump so that $y$ reduces (assuming minimization problem)
- We can say that $f(x - \epsilon \textbf{sign}(f'(x)))$ is less than $f(x)$
- For multiple inputs partial derivatives are used ie. $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$
- Gradient vector is represented as $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
- Gradient descent proposes a new point as $\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(x)$ where $\epsilon$ is the learning rate

# Stochastic gradient descent

- Large training set are necessary for good generalization
- Cost function used for optimization is $J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Gradient descent requires $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$

# Stochastic gradient descent

- Large training set are necessary for good generalization
- Cost function used for optimization is $J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Gradient descent requires $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
  - Computation cost is $O(m)$

# Stochastic gradient descent

- Large training set are necessary for good generalization
- Cost function used for optimization is $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Gradient descent requires $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
  - Computation cost is $O(m)$
- For SGD, gradient is an expectation estimated from a small sample known as mini-batch ($\mathbb{B} = \{x^{(1)}, \ldots, x^{(m')}\}$)
- Estimated gradient is $\boldsymbol{g} = \frac{1}{m'}\sum_{i=1}^{m'} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- New point will be $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon \boldsymbol{g}$

# SGD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with 4, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $x(w \times x - y)$**

| Step | Point | Derivative | New w |
|---|---|---|---|

# SGD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with 4, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $x(w \times x - y)$**

| Step | Point | Derivative | New w |
|---|---|---|---|
| 1 | (1,2) | 1*(4.0*1-2)=2.0 | 3.80 |

# SGD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with 4, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $x(w \times x - y)$**

| Step | Point | Derivative | New w |
|---|---|---|---|
| 1 | (1,2) | 1*(4.0*1-2)=2.0 | 3.80 |
| 2 | (2,4) | 2*(3.8*2-4)=7.2 | 3.08 |

# SGD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with 4, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $x(w \times x - y)$**

| Step | Point | Derivative | New w |
|---|---|---|---|
| 1 | (1,2) | 1*(4.0*1-2)=2.0 | 3.80 |
| 2 | (2,4) | 2*(3.8*2-4)=7.2 | 3.08 |
| 3 | (3,6) | 3*(3.1*3-6)=9.7 | 2.11 |
| 4 | (4,8) | 4*(2.1*4-8)=1.7 | 1.94 |
| 5 | (1,2) | 1*(1.9*1-2)=-0.1 | 1.94 |
| 6 | (2,4) | 2*(1.9*2-4)=-0.2 | 1.97 |
| 7 | (3,6) | 3*(2.0*3-6)=-0.3 | 1.99 |
| 8 | (4,8) | 4*(2.0*4-8)=-0.1 | 2.00 |
| 9 | (4,8) | 1*(2.0*1-2)=0.0 | 2.00 |

# GD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with $4$, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $\frac{1}{4} \sum_i x_i(w \times x_i - y_i)$**

```
Step     Derivative   New w
```

# GD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with $4$, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $\frac{1}{4} \sum_i x_i(w \times x_i - y_i)$**

| Step | Derivative | New w |
|---|---|---|
| 1 | 15 | 2.5 |

# GD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with $4$, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $\frac{1}{4} \sum_i x_i(w \times x_i - y_i)$**

| Step | Derivative | New w |
|---|---|---|
| 1 | 15 | 2.5 |
| 2 | 3.75 | 2.13 |

# GD example

- **Consider the following pair $(x, y)$ of points - $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$**
- **Let us try to fit a curve as follows $y = w \times x$ where $w$ is initialized with $4$, learning rate as $0.1$**
- **MSE as cost function. Derivative will be $\frac{1}{4} \sum_i x_i(w \times x_i - y_i)$**

| Step | Derivative | New w |
|---|---|---|
| 1 | 15 | 2.5 |
| 2 | 3.75 | 2.13 |
| 3 | 0.94 | 2.03 |
| 4 | 0.23 | 2.01 |
| 5 | 0.06 | 2.00 |

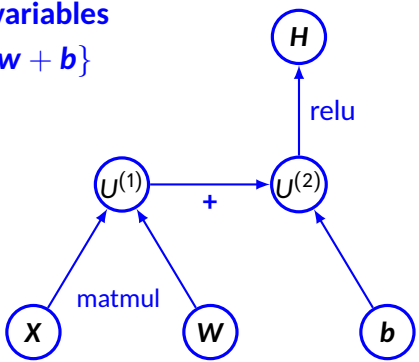
# Back propagation

# Back propagation

- In a feedforward network, an input ***x*** is read and produces an output $\hat{y}$
  - This is forward propagation
- During training forward propagation continues until it produces cost $J(\theta)$
- Back-propagation algorithm allows the information to flow backward in the network to compute the gradient
- Computation of analytical expression for gradient is easy
- We need to find out gradient of the cost function with respect to the parameters ie. $\nabla_{\theta} J(\theta)$

# Computational graph

- **Computational graph is used to represent basic operations and to be used for back propagation**
- **Each node represents a variable (scalar, vector, etc.)**
- **Operation decribes relation between variables**
- **Computational graph of $\boldsymbol{H} = \max\{0, \boldsymbol{xw} + \boldsymbol{b}\}$**



# Chain rule of calculus

- Back-propagation algorithm heavily depends on it
- Let $x$ be a real number and $y = g(x)$ and $z = f(g(x)) = f(y)$
- Chain rule says $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$
- This can be generalized: Let $\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ and $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(\boldsymbol{y})$ then $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$
- In vector notation it will be where $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the $n \times m$ Jacobian matrix of $g$

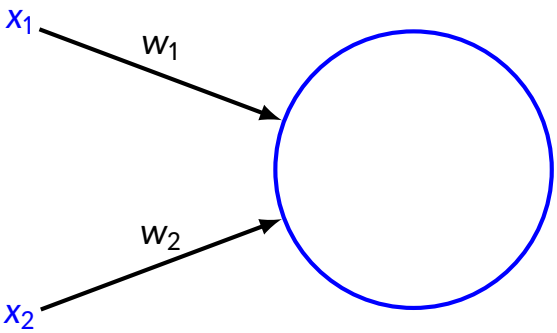
$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^T \nabla_{\boldsymbol{y}} z$$

# Backpropagation

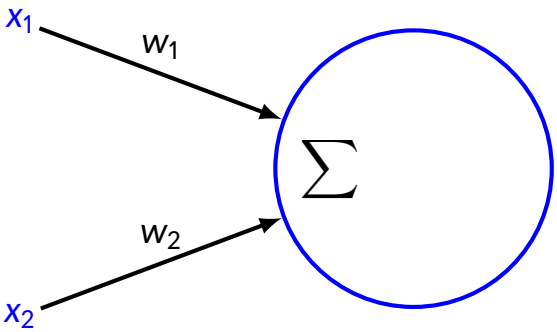
$x_1$

$x_2$

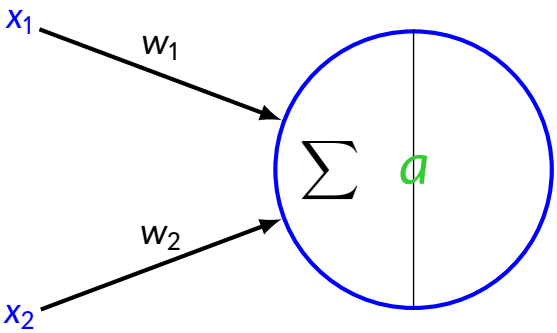
# Backpropagation



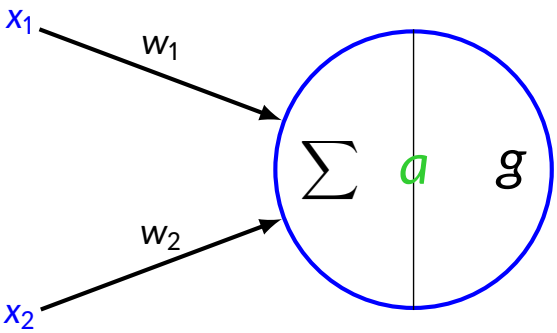
# Backpropagation



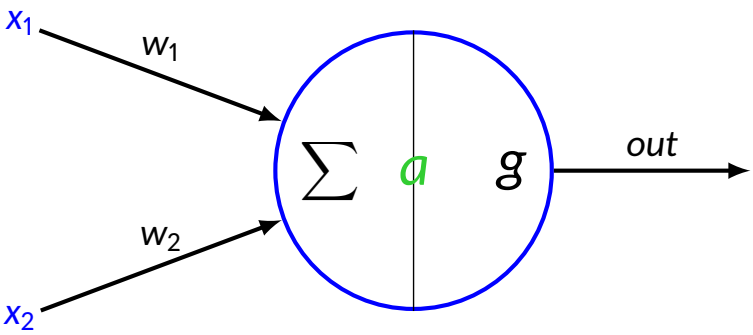
# Backpropagation



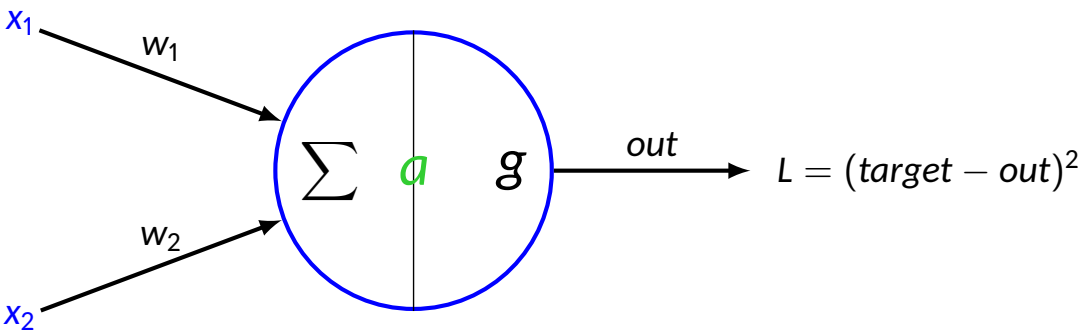
# Backpropagation



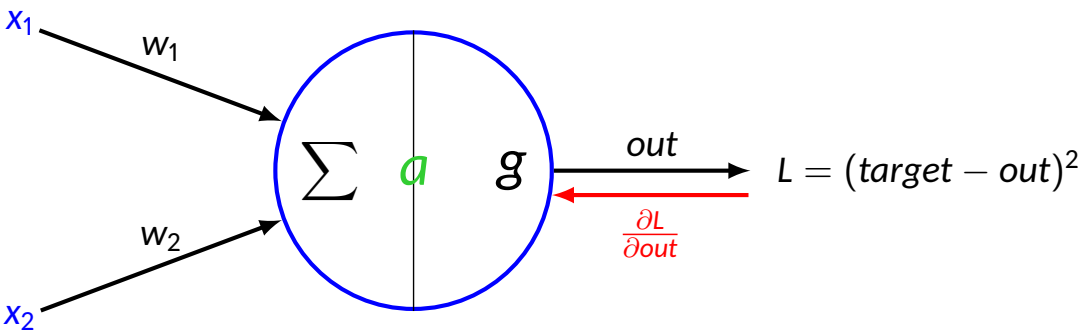
# Backpropagation



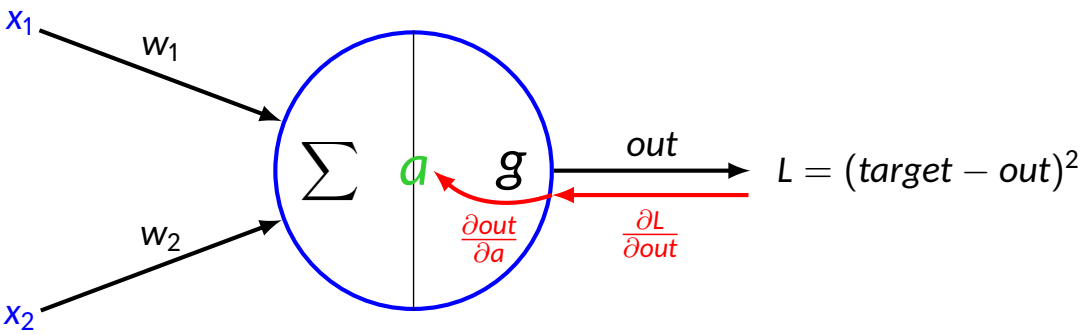
# Backpropagation



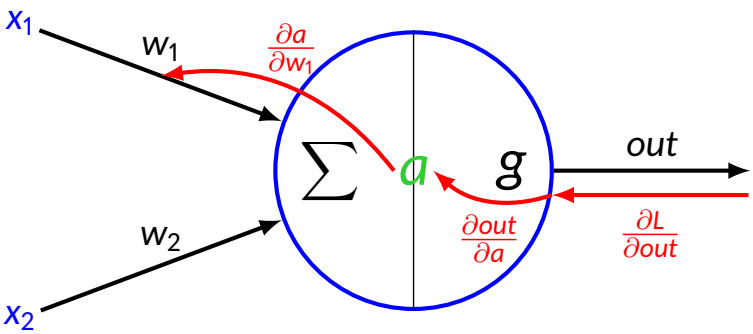
# Backpropagation



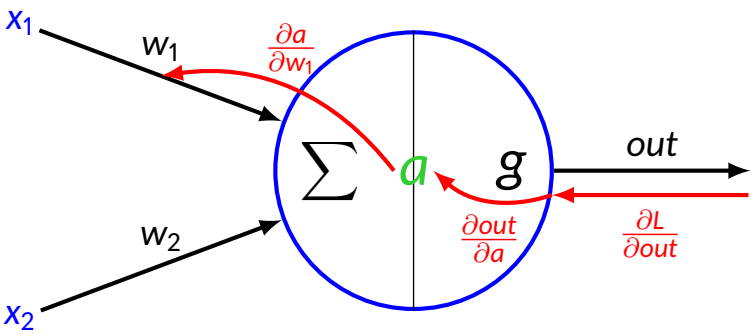
# Backpropagation



# Backpropagation



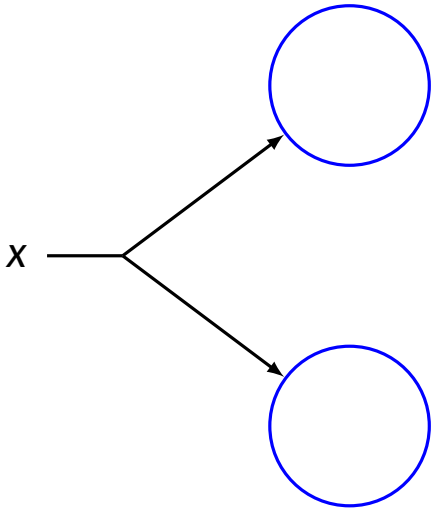
# Backpropagation



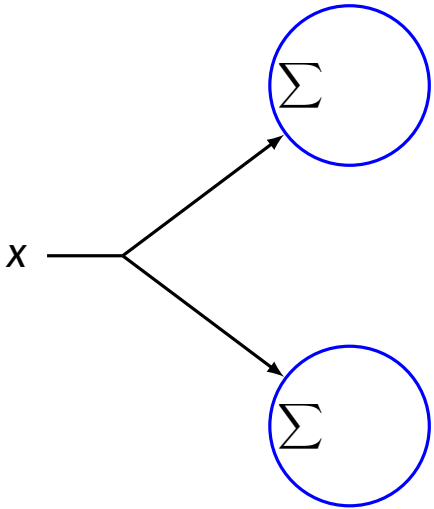

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial out}\frac{\partial out}{\partial a}\frac{\partial a}{\partial w_1}$$

# Backpropagation

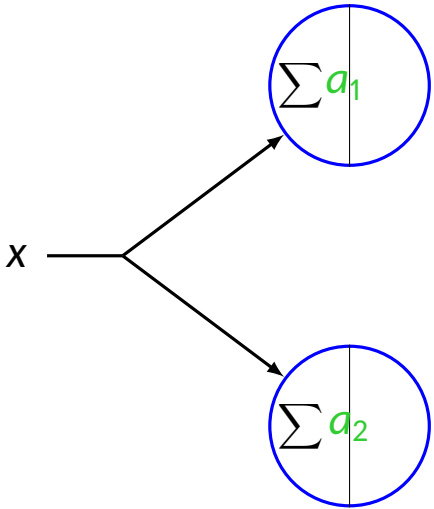
# Backpropagation



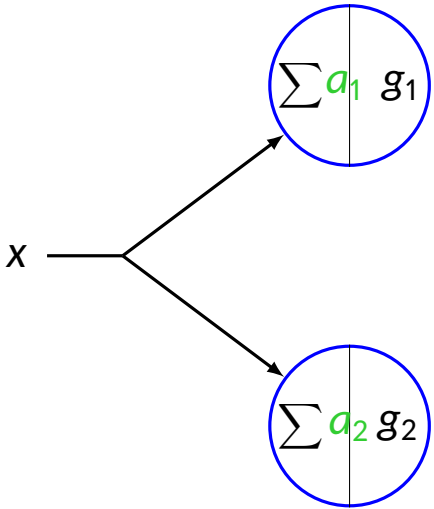
# Backpropagation



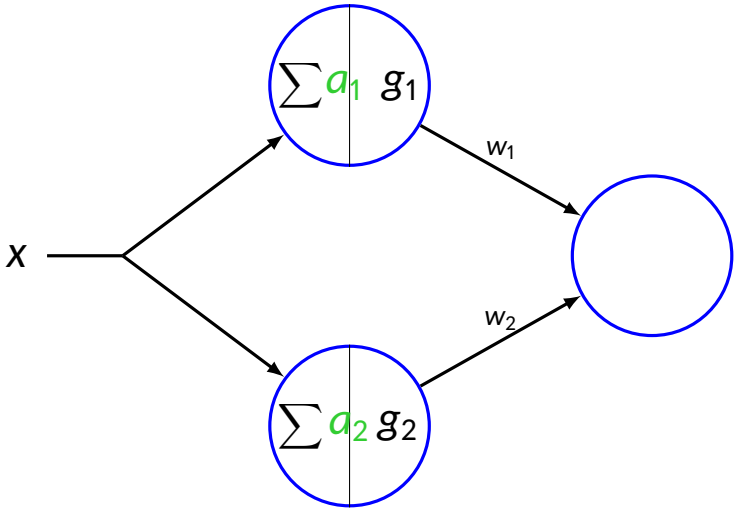
# Backpropagation



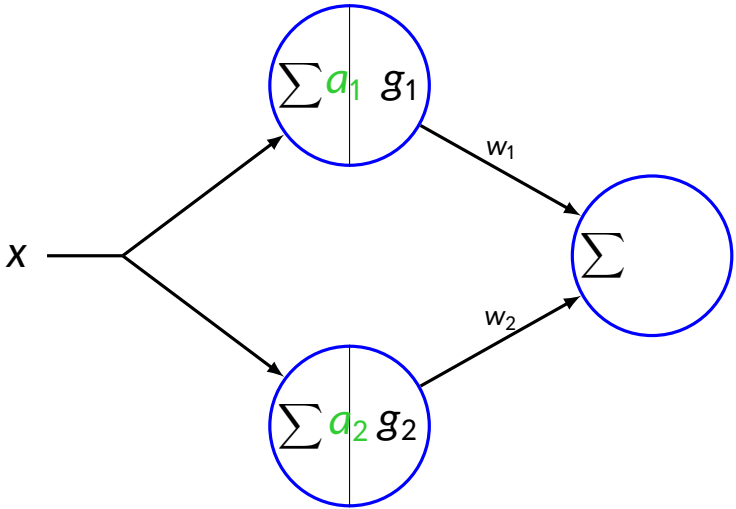
# Backpropagation



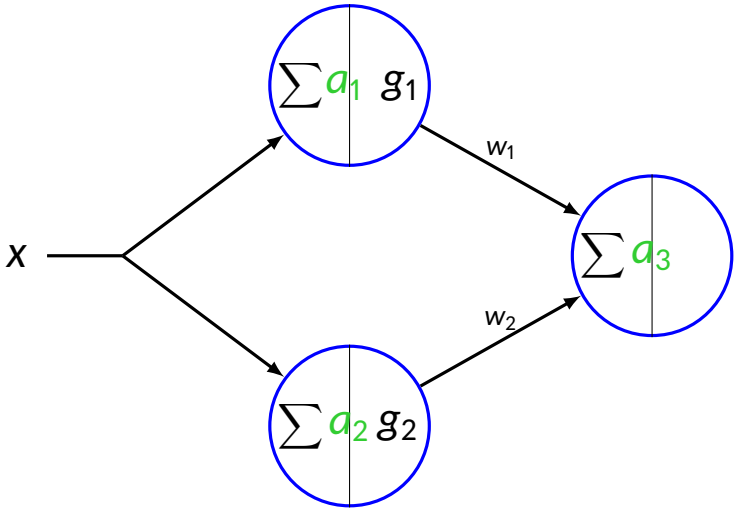
# Backpropagation



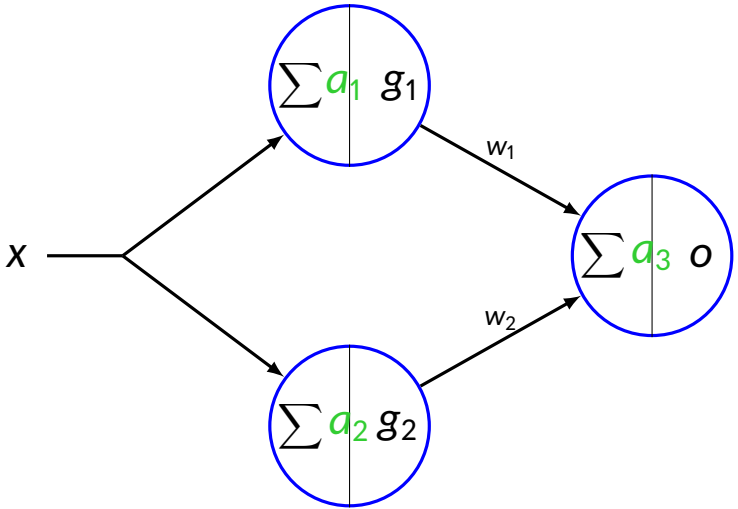
# Backpropagation



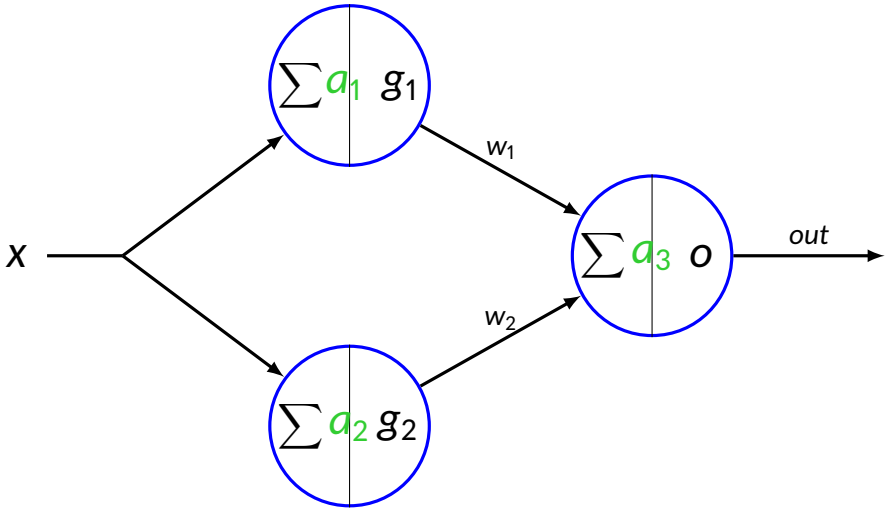
# Backpropagation



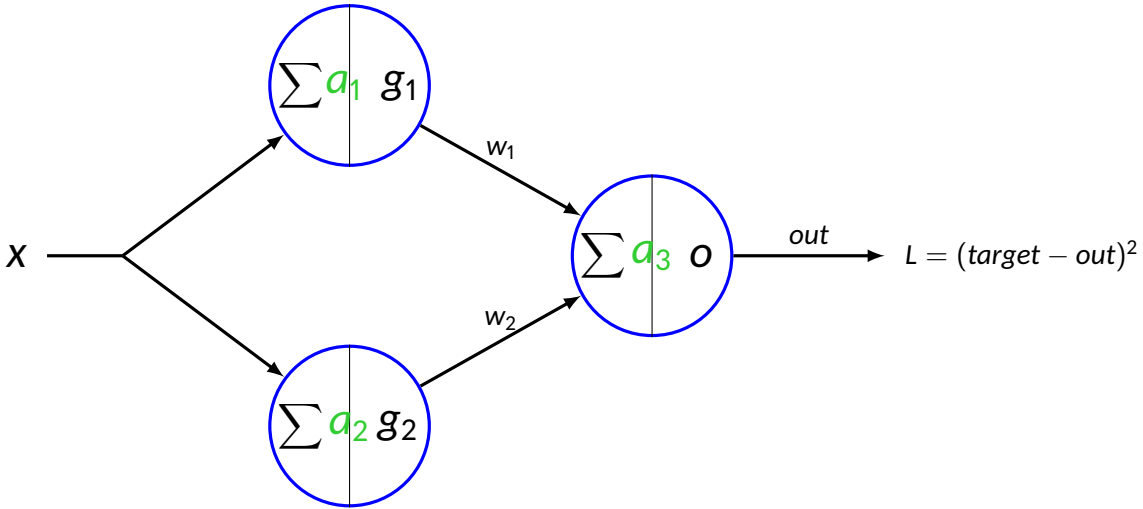
# Backpropagation



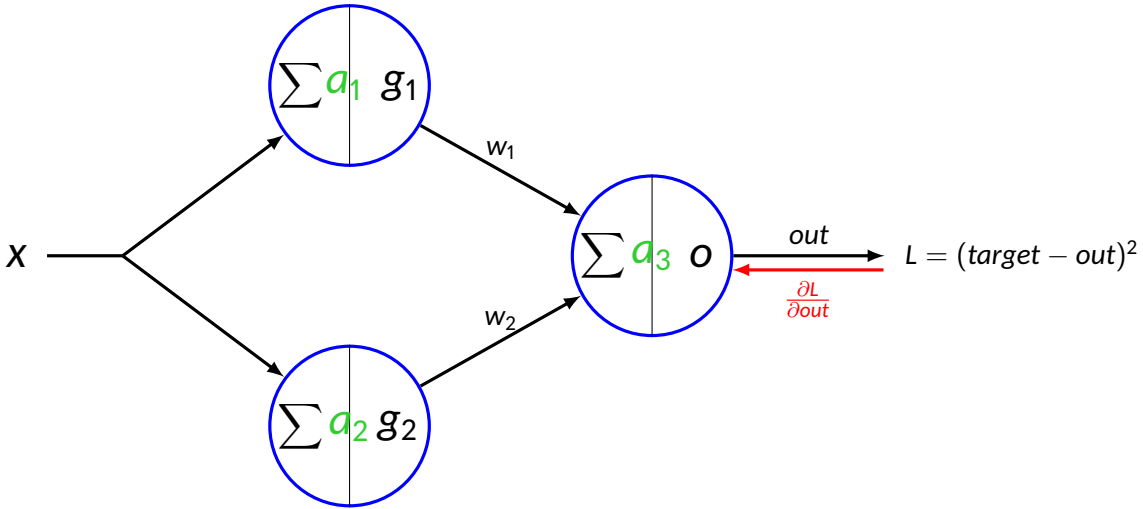
# Backpropagation



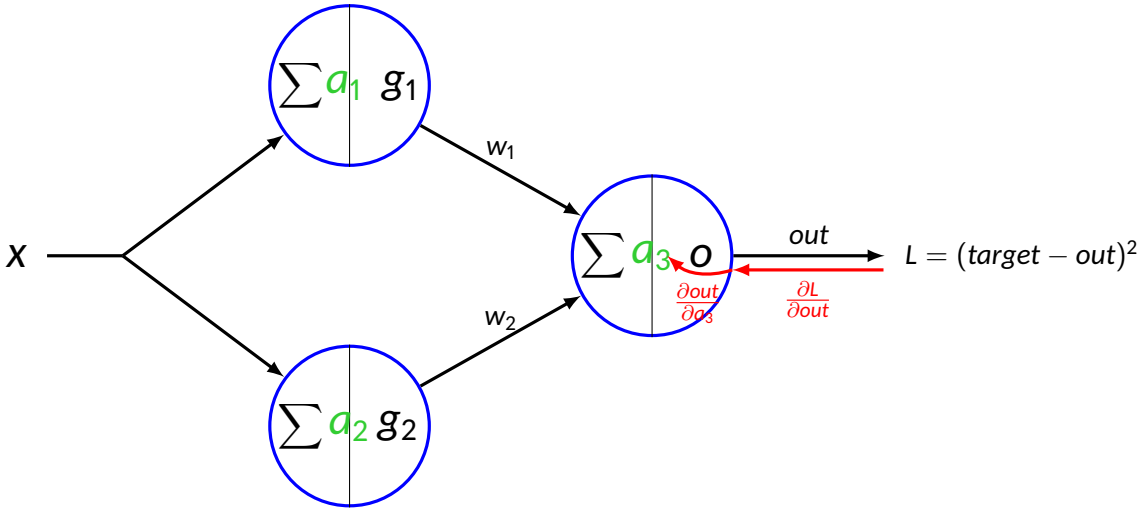
# Backpropagation



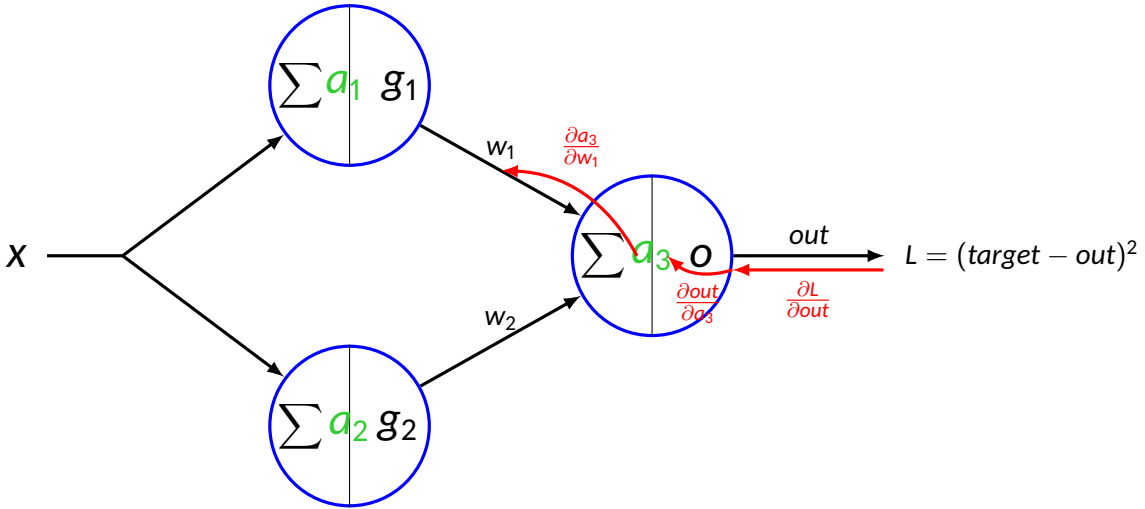
# Backpropagation



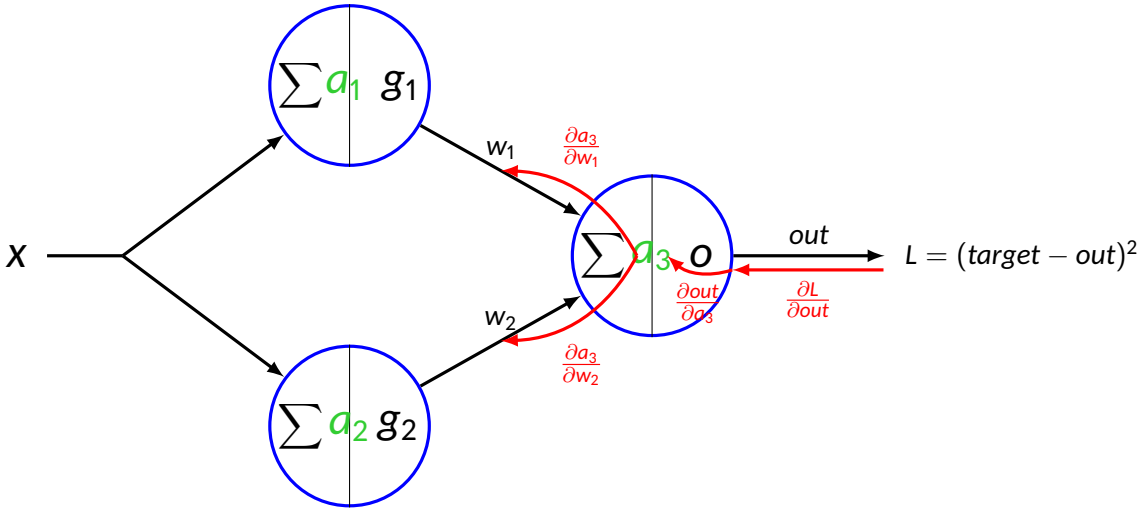
# Backpropagation



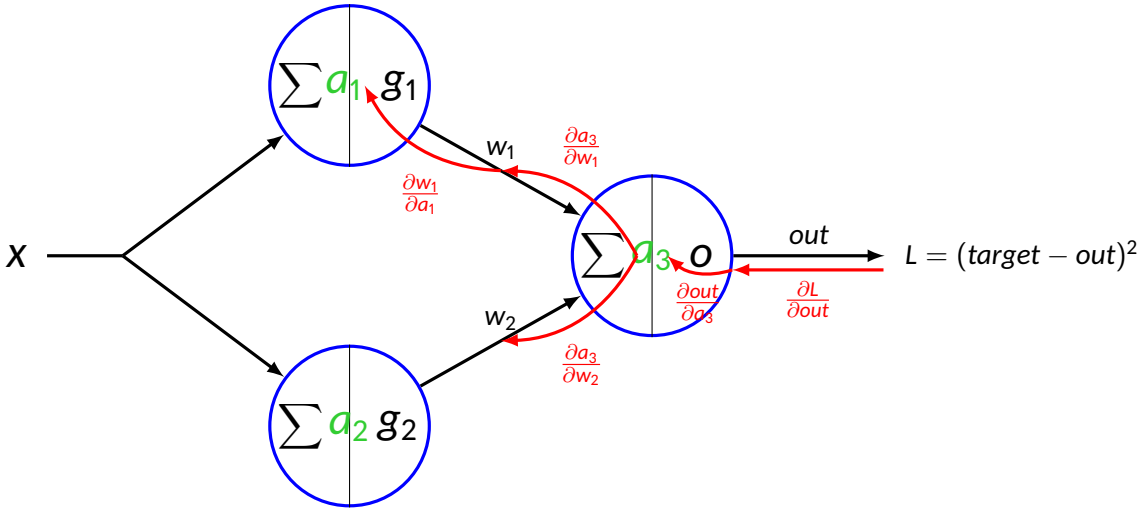
# Backpropagation



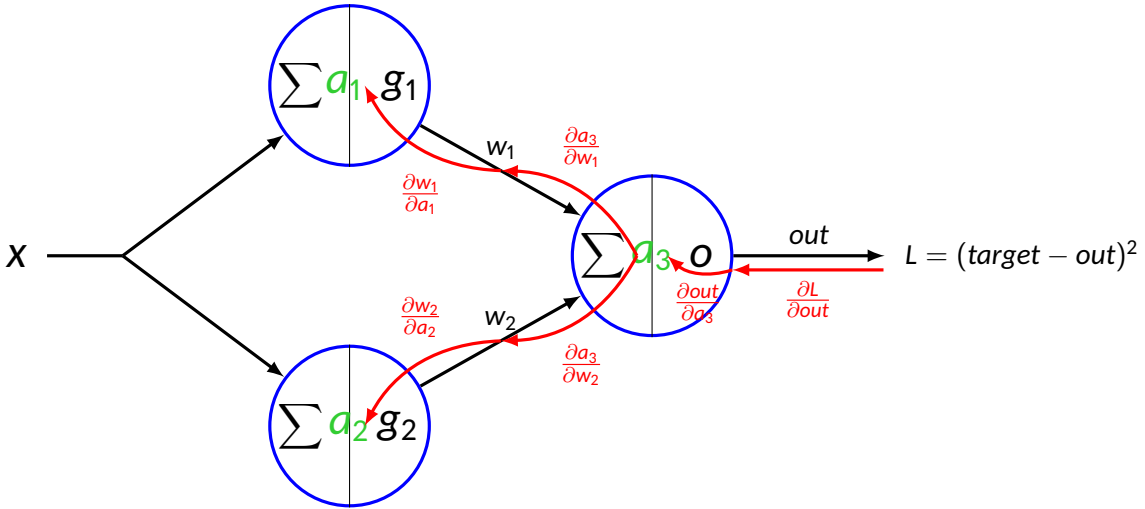
# Backpropagation



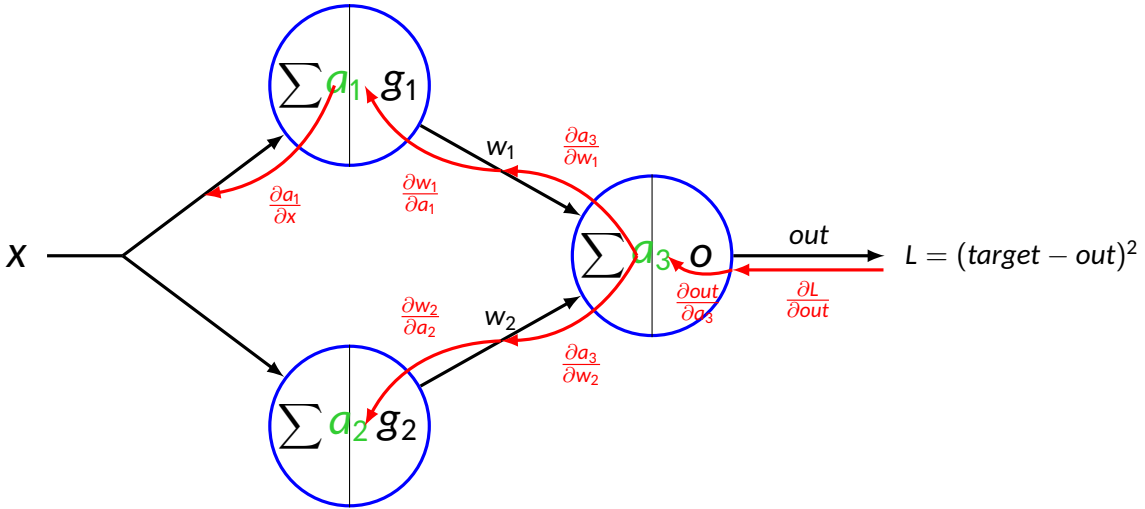
# Backpropagation



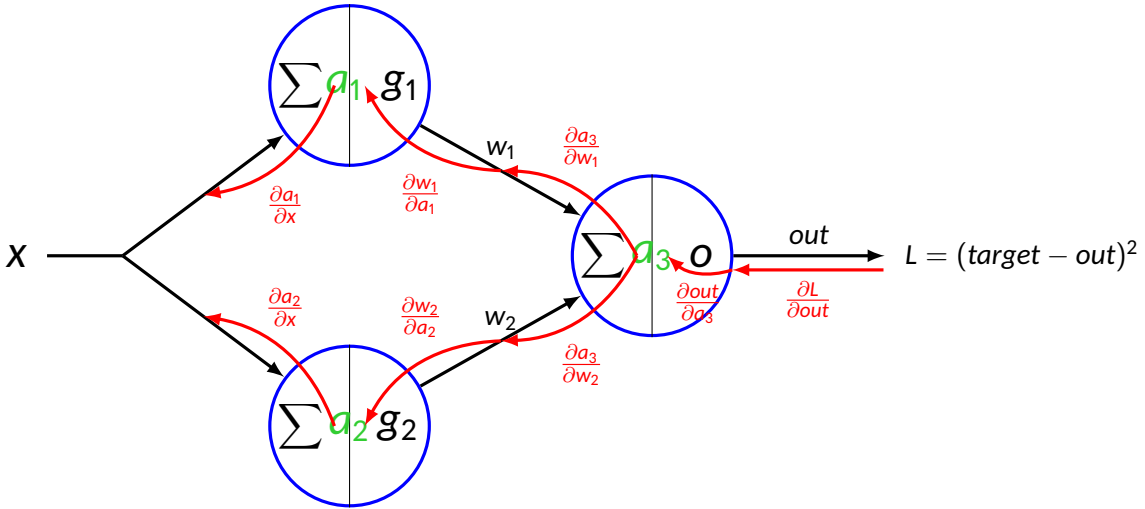
# Backpropagation



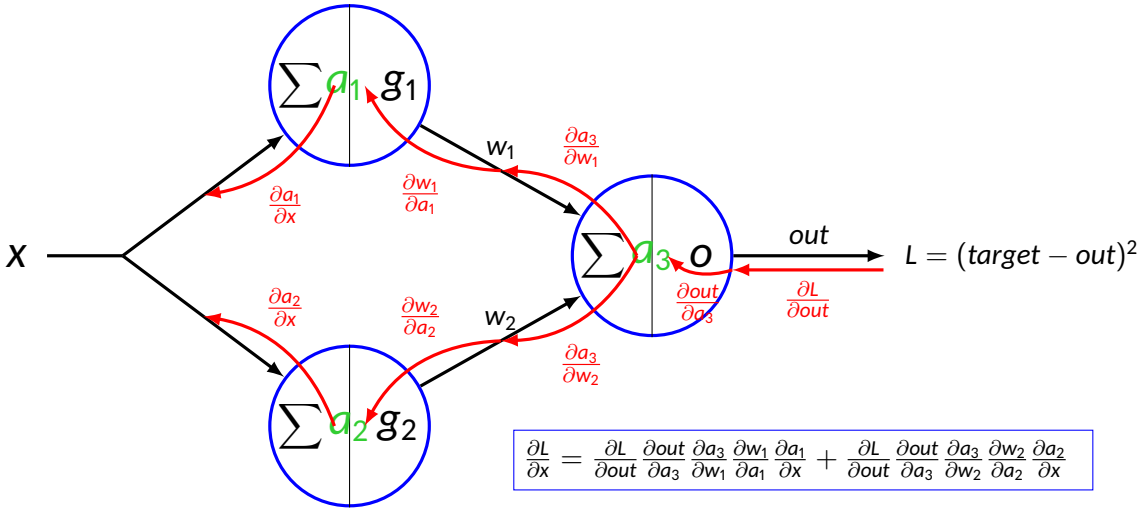
# Backpropagation



# Backpropagation

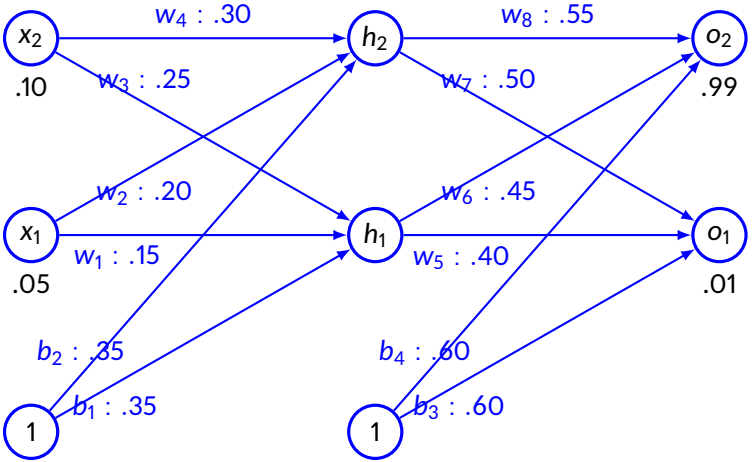


# Backpropagation



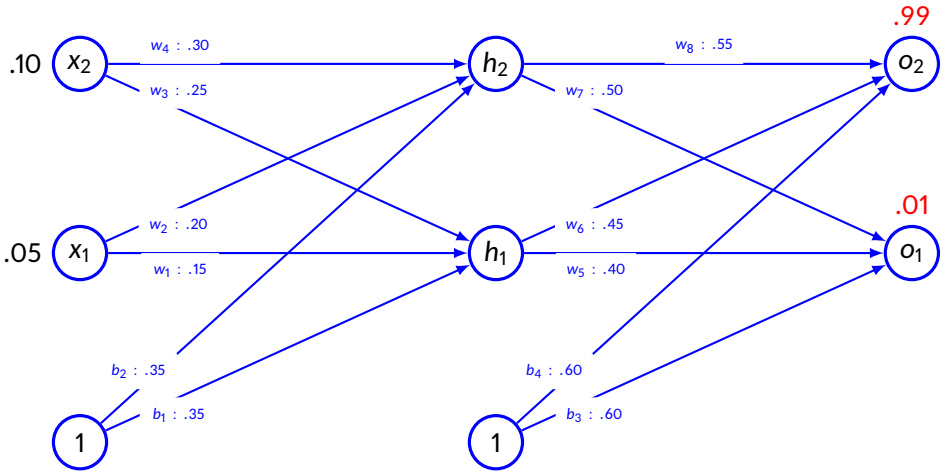

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial out}\frac{\partial out}{\partial a_3}\frac{\partial a_3}{\partial w_1}\frac{\partial w_1}{\partial a_1}\frac{\partial a_1}{\partial x} + \frac{\partial L}{\partial out}\frac{\partial out}{\partial a_3}\frac{\partial a_3}{\partial w_2}\frac{\partial w_2}{\partial a_2}\frac{\partial a_2}{\partial x}$$

# Example

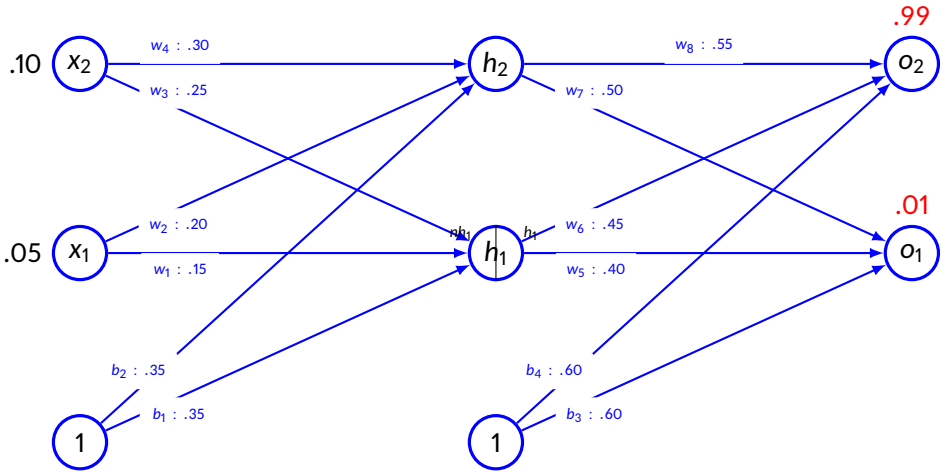



Hidden and output layer have sigmoid activation function. Loss function - MSE.

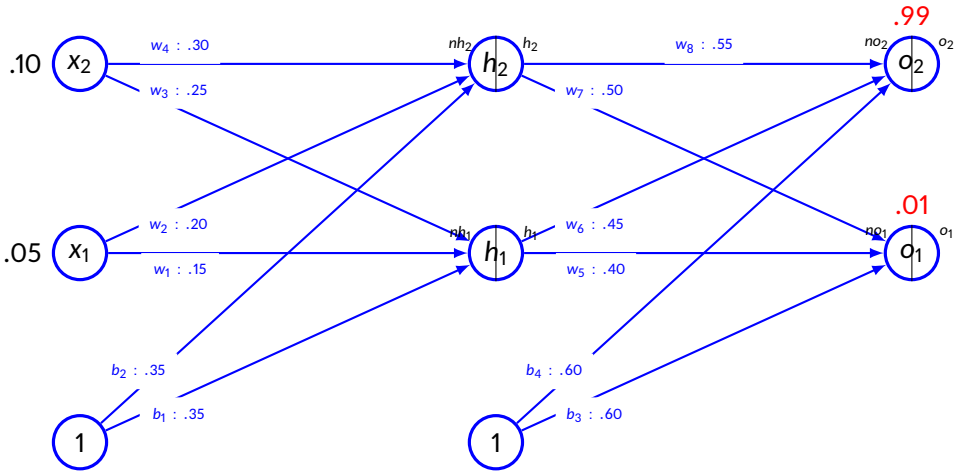
# Example


.10
$x_2$
.05
$x_1$
1
$h_2$
$h_1$
1
$o_2$
$o_1$
.99
.01
$w_4$ : .30
$w_3$ : .25
$w_2$ : .20
$w_1$ : .15
$b_2$ : .35
$b_1$ : .35
$w_8$ : .55
$w_7$ : .50
$w_6$ : .45
$w_5$ : .40
$b_4$ : .60
$b_3$ : .60

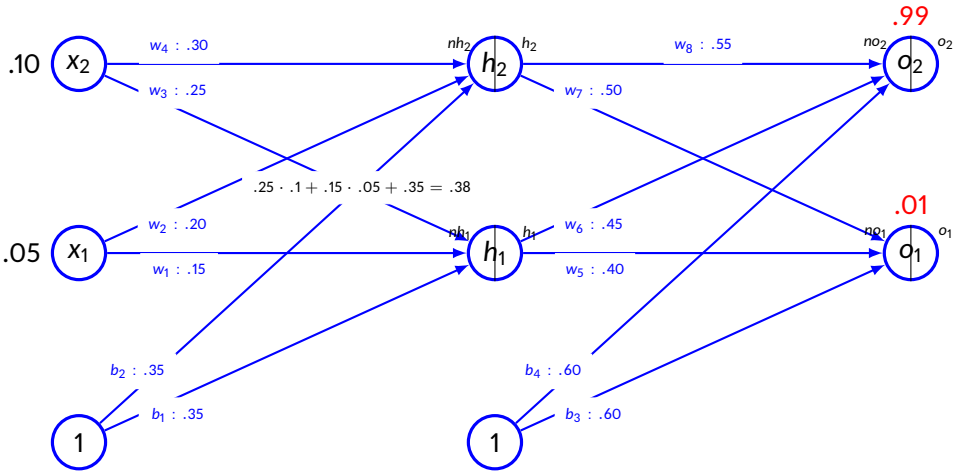
# Example


.10
$x_2$
.05
$x_1$
1
$w_4$ : .30
$w_3$ : .25
$w_2$ : .20
$w_1$ : .15
$b_2$ : .35
$b_1$ : .35
$h_2$
$h_1$
$h_1$
$w_8$ : .55
$w_7$ : .50
$w_6$ : .45
$w_5$ : .40
$b_4$ : .60
$b_3$ : .60
1
.99
$o_2$
.01
$o_1$

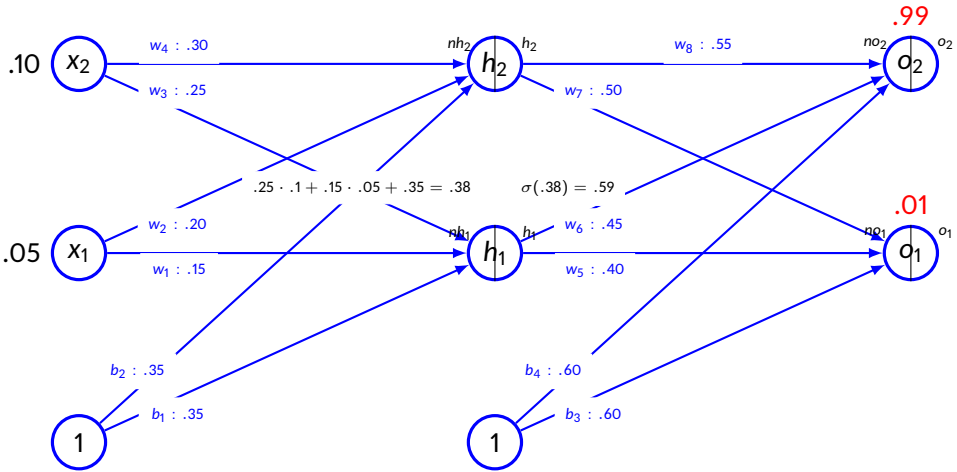
# Example


.10
$x_2$
.05
$x_1$
1
$w_4$ : .30
$w_3$ : .25
$w_2$ : .20
$w_1$ : .15
$b_2$ : .35
$b_1$ : .35
$nh_2$
$h_2$
$nh_1$
$h_1$
1
$w_8$ : .55
$w_7$ : .50
$w_6$ : .45
$w_5$ : .40
$b_4$ : .60
$b_3$ : .60
.99
$no_2$
$o_2$
.01
$no_1$
$o_1$

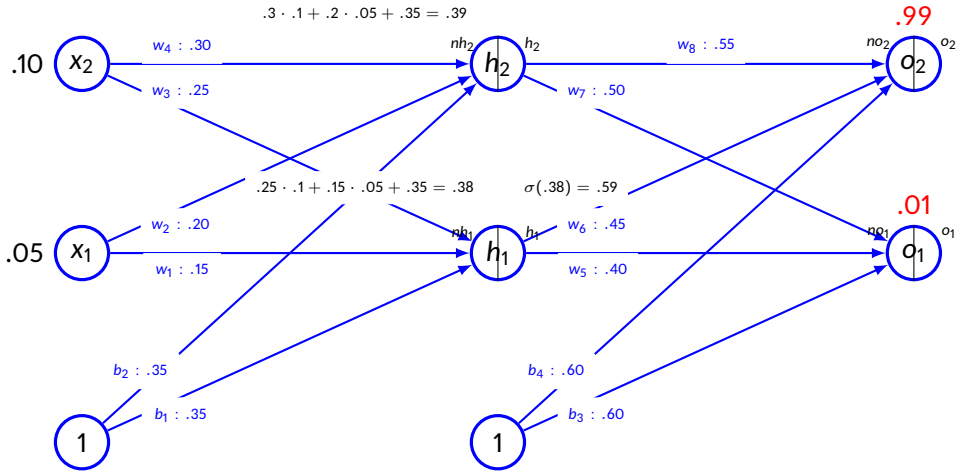
# Example


.10
$x_2$
.05
$x_1$
1
1
$w_4$ : .30
$w_3$ : .25
$w_2$ : .20
$w_1$ : .15
$b_2$ : .35
$b_1$ : .35
$nh_2$
$h_2$
$nh_1$
$h_1$
.25 · .1 + .15 · .05 + .35 = .38
$w_8$ : .55
$w_7$ : .50
$w_6$ : .45
$w_5$ : .40
$b_4$ : .60
$b_3$ : .60
.99
.01
$no_2$
$o_2$
$no_1$
$o_1$

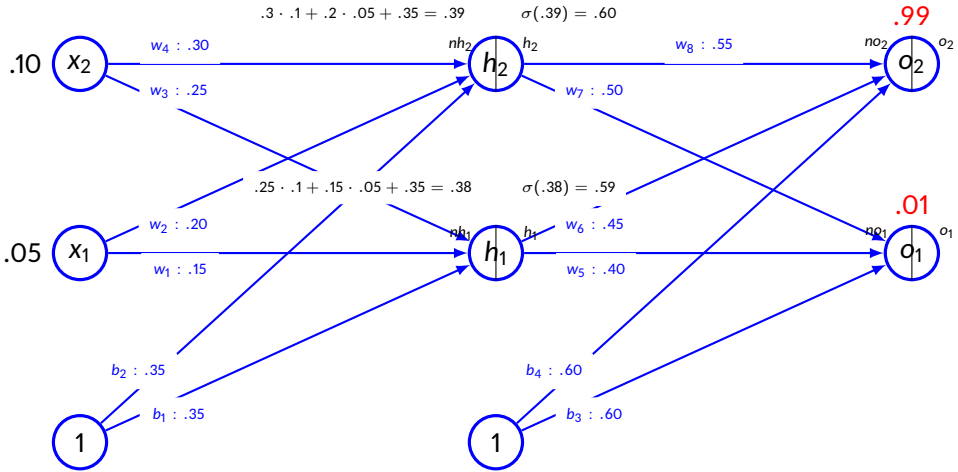
# Example


.10
$x_2$
.05
$x_1$
1
1
$w_4$ : .30
$w_3$ : .25
$w_2$ : .20
$w_1$ : .15
$b_2$ : .35
$b_1$ : .35
$nh_2$
$h_2$
$nh_1$
$h_1$
$.25 \cdot .1 + .15 \cdot .05 + .35 = .38$
$\sigma(.38) = .59$
$w_8$ : .55
$w_7$ : .50
$w_6$ : .45
$w_5$ : .40
$b_4$ : .60
$b_3$ : .60
.99
.01
$no_2$
$o_2$
$no_1$
$o_1$

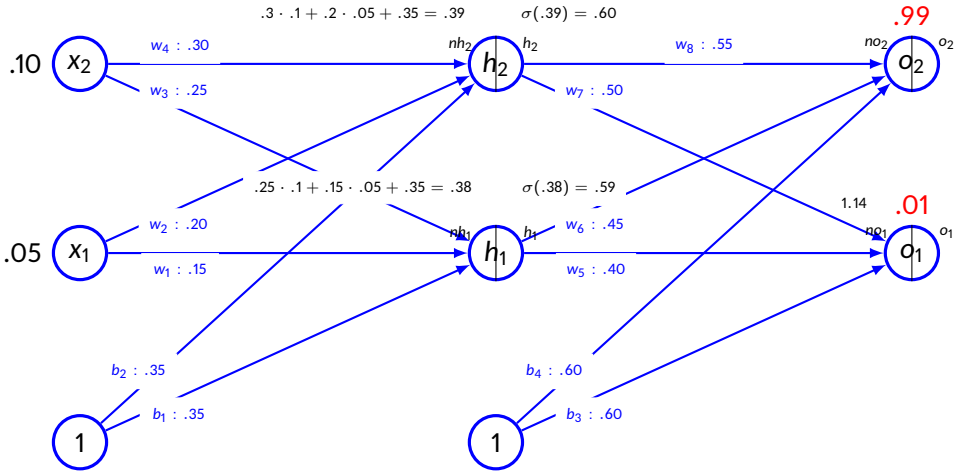
# Example


.3 · .1 + .2 · .05 + .35 = .39
.99
.10
$x_2$
$w_4$ : .30
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_3$ : .25
$w_7$ : .50
.25 · .1 + .15 · .05 + .35 = .38
$\sigma(.38) = .59$
.01
$w_2$ : .20
$nh_1$
$h_1$
$w_6$ : .45
$no_1$
$o_1$
.05
$x_1$
$w_1$ : .15
$w_5$ : .40
$b_2$ : .35
$b_4$ : .60
$b_1$ : .35
$b_3$ : .60
1
1

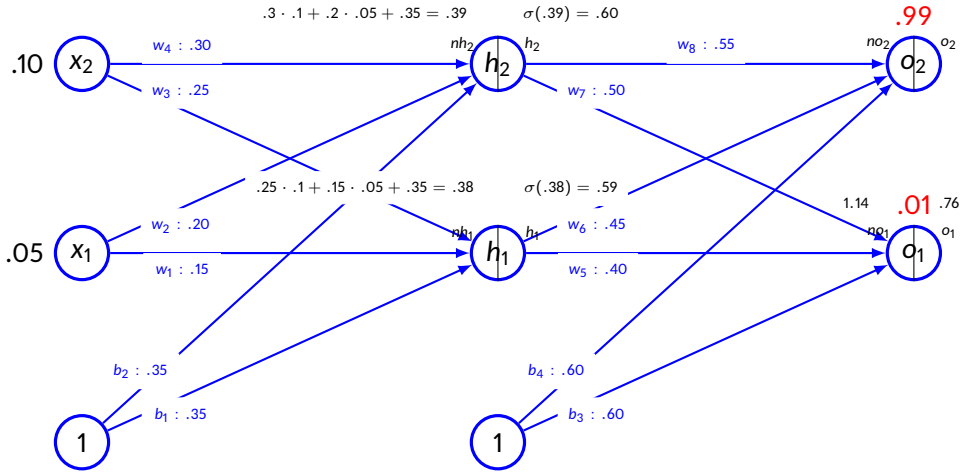
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
.99
.10
x2
w4 : .30
w3 : .25
nh2
h2
h2
w8 : .55
no2
o2
o2
w7 : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
.01
w2 : .20
.05
x1
w1 : .15
nh1
h1
h1
w6 : .45
no1
o1
o1
w5 : .40
b2 : .35
b1 : .35
1
b4 : .60
b3 : .60
1

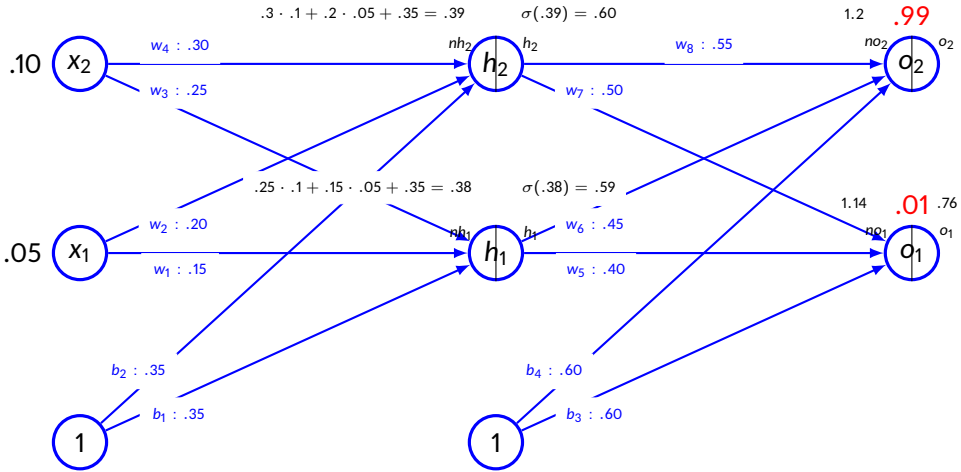
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
.99
.10
x2
w4 : .30
w3 : .25
nh2
h2
h2
w8 : .55
no2
o2
o2
w7 : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
w2 : .20
.05
x1
w1 : .15
nh1
h1
h1
w6 : .45
no1
o1
o1
w5 : .40
b2 : .35
b1 : .35
b4 : .60
b3 : .60
1
1

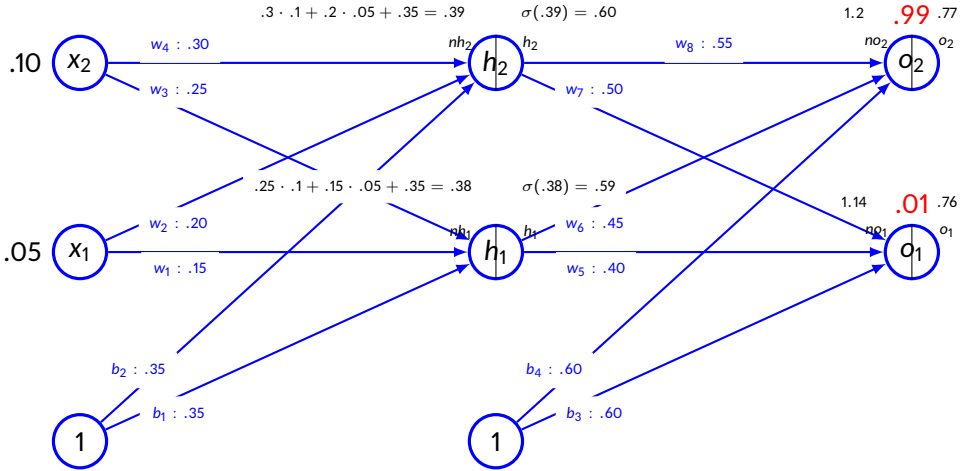
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
.99
.10
x2
w4 : .30
w3 : .25
nh2
h2
h2
w8 : .55
no2
o2
o2
w7 : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
x1
w2 : .20
w1 : .15
nh1
h1
h1
w6 : .45
w5 : .40
no1
o1
o1
b2 : .35
b1 : .35
b4 : .60
b3 : .60
1
1

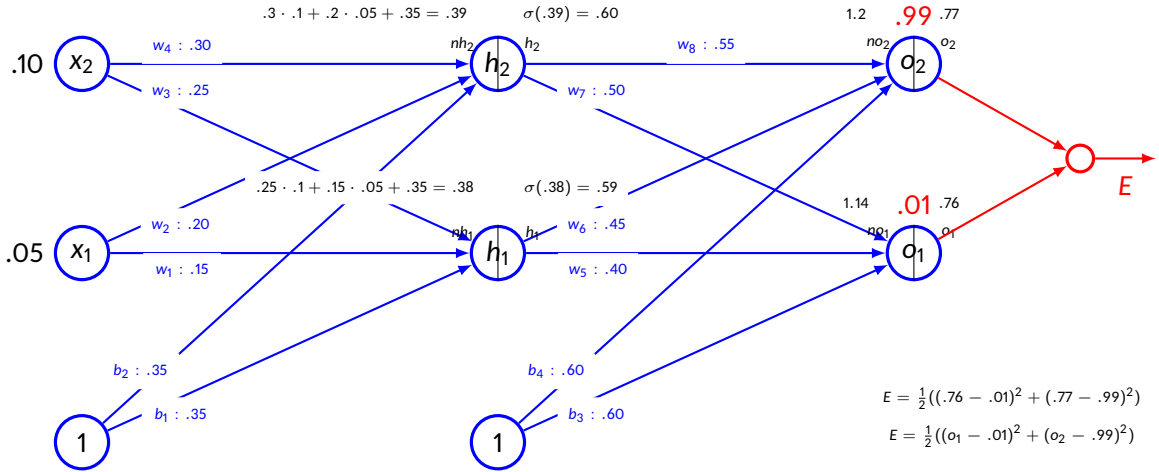
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.10
x2
w4 : .30
w3 : .25
nh2
h2
h2
w8 : .55
no2
o2
o2
w7 : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
x1
w2 : .20
w1 : .15
nh1
h1
h1
w6 : .45
w5 : .40
no1
o1
o1
b2 : .35
b1 : .35
1
b4 : .60
b3 : .60
1

# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
$x_2$
$w_4$ : .30
$w_3$ : .25
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_7$ : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
$x_1$
$w_2$ : .20
$w_1$ : .15
$nh_1$
$h_1$
$w_6$ : .45
$w_5$ : .40
$no_1$
$o_1$
$b_2$ : .35
$b_1$ : .35
$b_4$ : .60
$b_3$ : .60
1
1

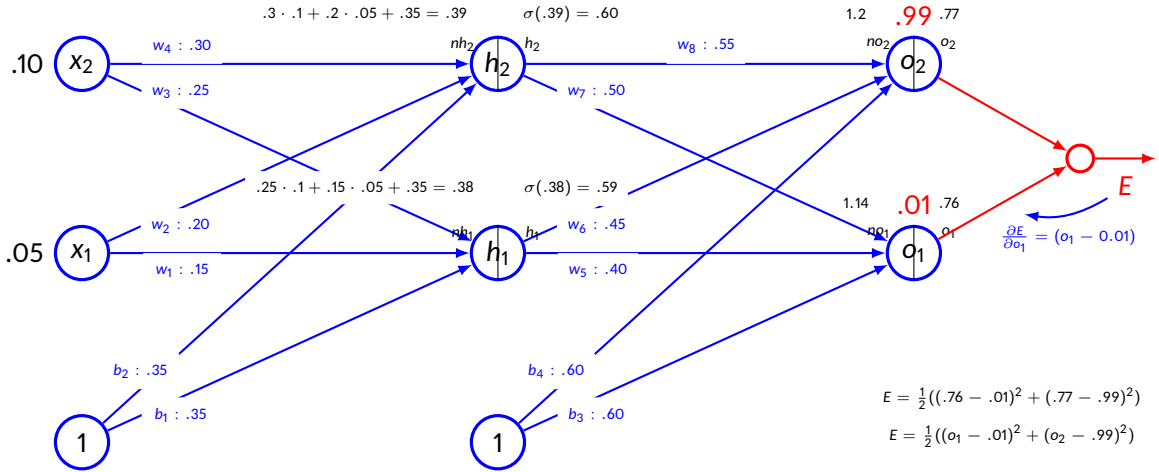
# Example



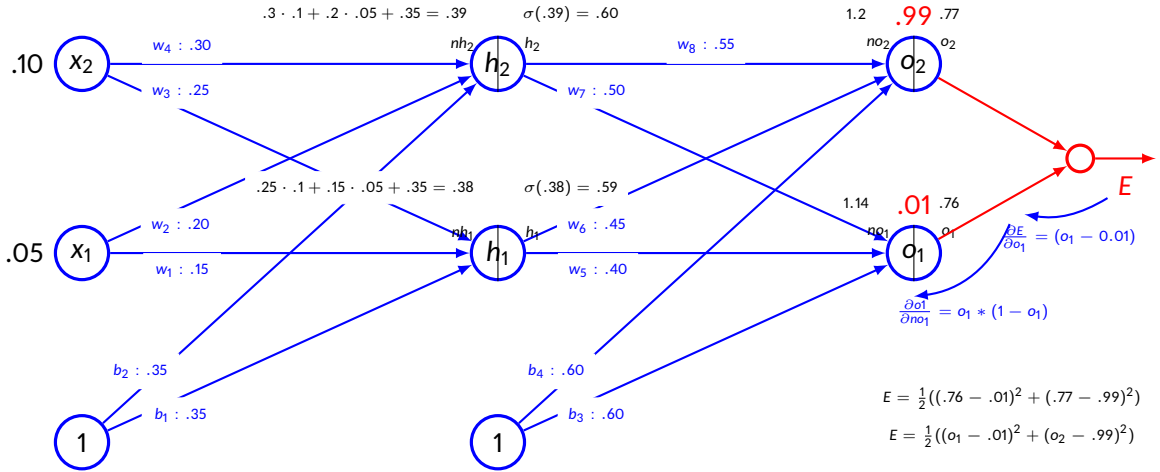

$$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$$

$$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$$

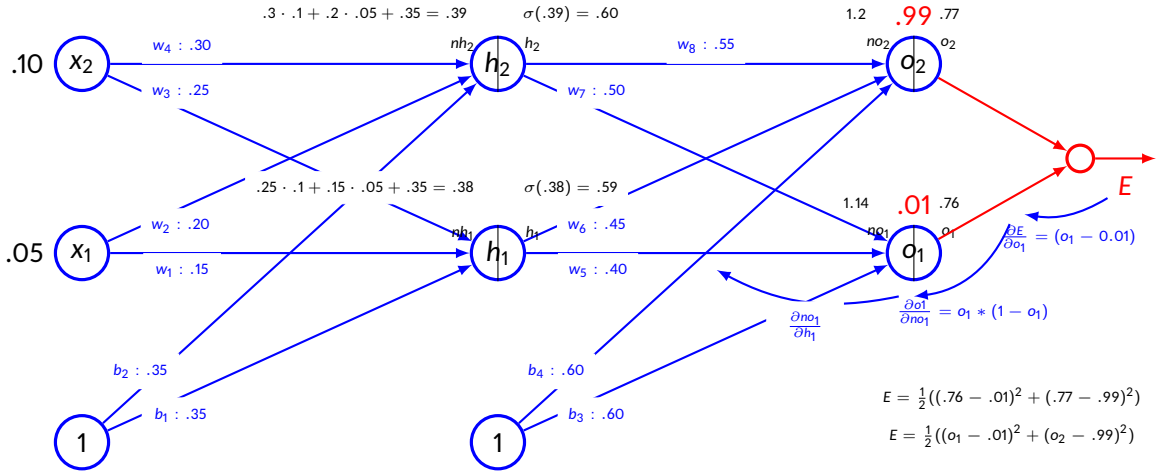
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
x2
w4 : .30
w3 : .25
nh2
h2
w8 : .55
no2
o2
w7 : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
E
.05
x1
w2 : .20
w1 : .15
nh1
h1
w6 : .45
w5 : .40
no1
o1
∂E/∂o1 = (o1 − 0.01)
b2 : .35
b1 : .35
b4 : .60
b3 : .60
1
1
E = 1/2((.76 − .01)^2 + (.77 − .99)^2)
E = 1/2((o1 − .01)^2 + (o2 − .99)^2)

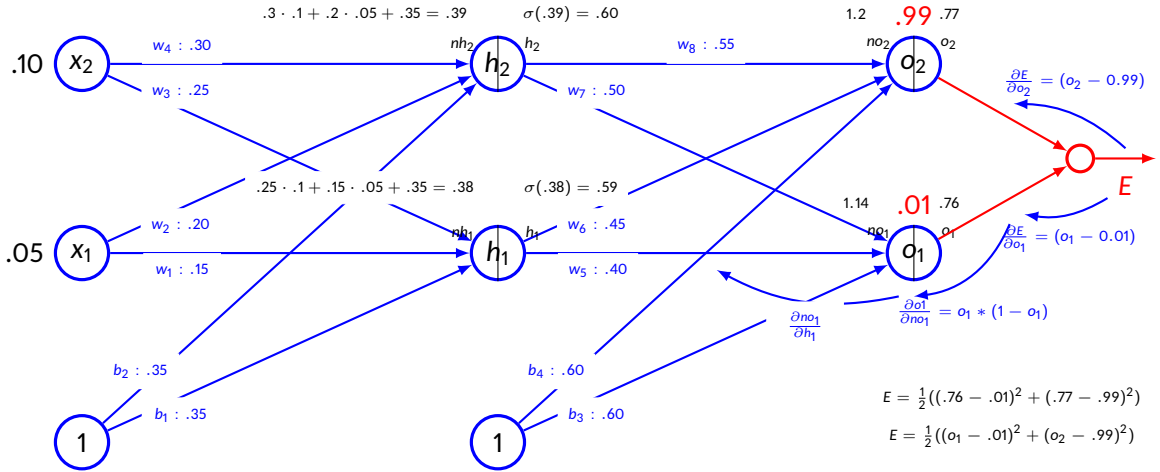
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
$x_2$
$w_4$ : .30
$w_3$ : .25
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_7$ : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
E
1.14
.01
.76
.05
$x_1$
$w_2$ : .20
$w_1$ : .15
$nh_1$
$h_1$
$w_6$ : .45
$w_5$ : .40
$no_1$
$o_1$
$\frac{\partial E}{\partial o_1} = (o_1 - 0.01)$
$\frac{\partial o1}{\partial no_1} = o_1 * (1 - o_1)$
$b_2$ : .35
$b_1$ : .35
$b_4$ : .60
$b_3$ : .60
1
1
$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$
$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$

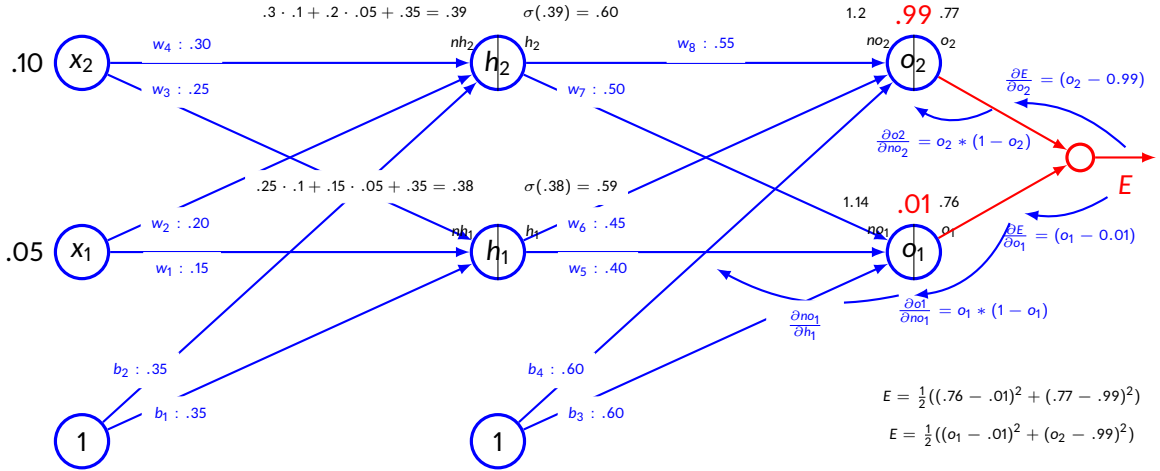
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
$x_2$
$w_4$ : .30
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_3$ : .25
$w_7$ : .50
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
E
1.14
.01
.76
.05
$x_1$
$w_2$ : .20
$nh_1$
$h_1$
$w_6$ : .45
$no_1$
$o_1$
$w_1$ : .15
$w_5$ : .40
$\frac{\partial E}{\partial o_1} = (o_1 - 0.01)$
$\frac{\partial o_1}{\partial no_1} = o_1 * (1 - o_1)$
$\frac{\partial no_1}{\partial h_1}$
$b_2$ : .35
$b_4$ : .60
$b_1$ : .35
$b_3$ : .60
1
1
$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$
$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$

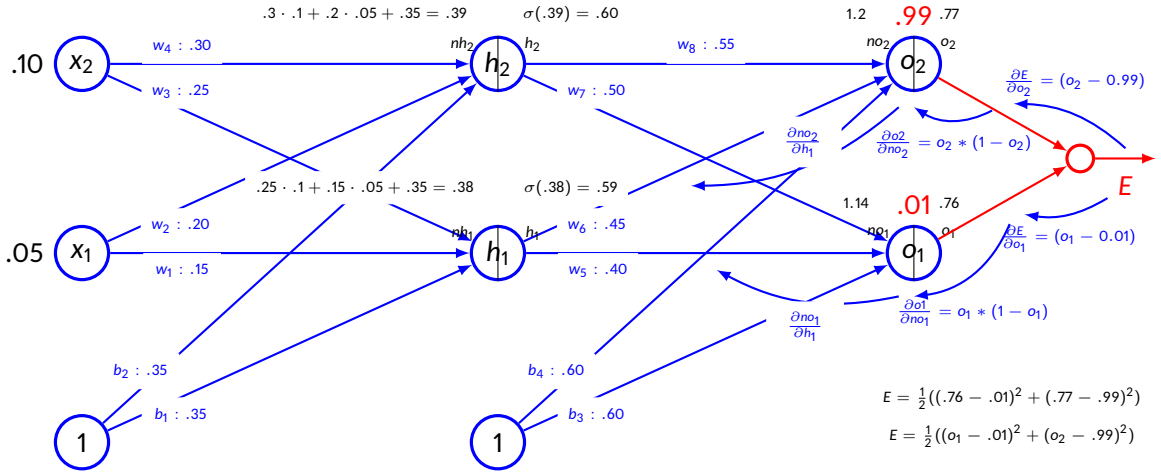
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
x2
w4 : .30
w3 : .25
nh2
h2
w8 : .55
no2
o2
w7 : .50
∂E/∂o2 = (o2 − 0.99)
E
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
x1
w2 : .20
w1 : .15
nh1
h1
w6 : .45
w5 : .40
no1
o1
∂E/∂o1 = (o1 − 0.01)
∂no1/∂h1
∂o1/∂no1 = o1 ∗ (1 − o1)
b2 : .35
b1 : .35
b4 : .60
b3 : .60
1
1
E = 1/2((.76 − .01)^2 + (.77 − .99)^2)
E = 1/2((o1 − .01)^2 + (o2 − .99)^2)

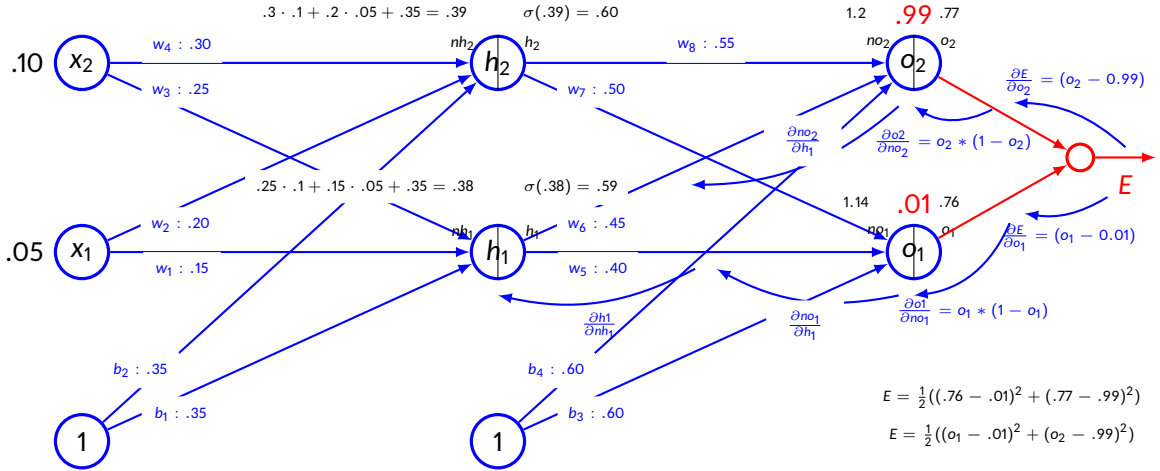
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
$x_2$
$w_4$ : .30
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_3$ : .25
$w_7$ : .50
$\frac{\partial E}{\partial o_2} = (o_2 - 0.99)$
$\frac{\partial o2}{\partial no_2} = o_2 * (1 - o_2)$
E
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
$x_1$
$w_2$ : .20
$nh_1$
$h_1$
$w_6$ : .45
$no_1$
$o_1$
$w_1$ : .15
$w_5$ : .40
$\frac{\partial E}{\partial o_1} = (o_1 - 0.01)$
$\frac{\partial no_1}{\partial h_1}$
$\frac{\partial o1}{\partial no_1} = o_1 * (1 - o_1)$
$b_2$ : .35
$b_4$ : .60
$b_1$ : .35
$b_3$ : .60
1
1
$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$
$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$

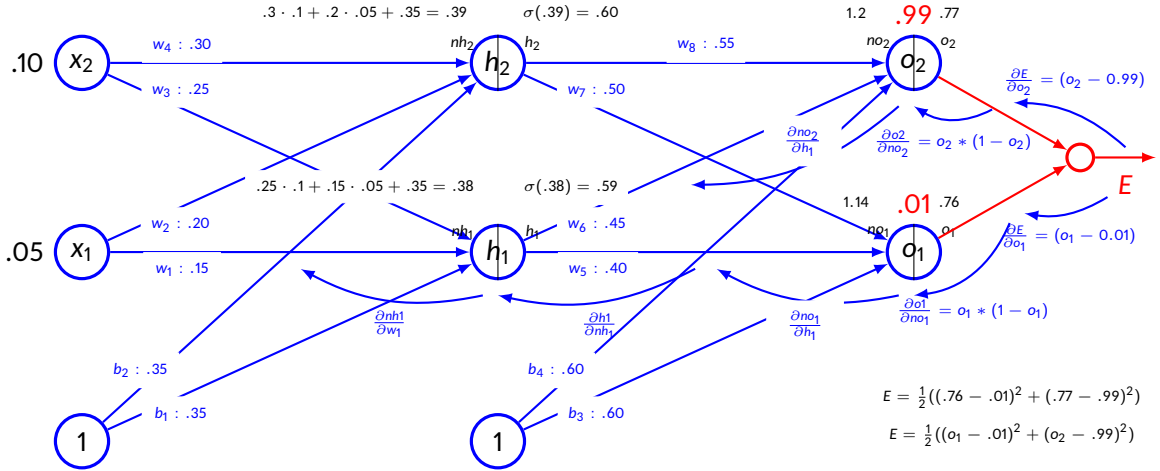
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
$x_2$
$w_4$ : .30
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_3$ : .25
$w_7$ : .50
$\frac{\partial E}{\partial o_2} = (o_2 - 0.99)$
$\frac{\partial no_2}{\partial h_1}$
$\frac{\partial o2}{\partial no_2} = o_2 * (1 - o_2)$
E
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
$x_1$
$w_2$ : .20
$nh_1$
$h_1$
$w_6$ : .45
$no_1$
$o_1$
$\frac{\partial E}{\partial o_1} = (o_1 - 0.01)$
$w_1$ : .15
$w_5$ : .40
$\frac{\partial no_1}{\partial h_1}$
$\frac{\partial o1}{\partial no_1} = o_1 * (1 - o_1)$
$b_2$ : .35
$b_4$ : .60
$b_1$ : .35
$b_3$ : .60
1
1
$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$
$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$

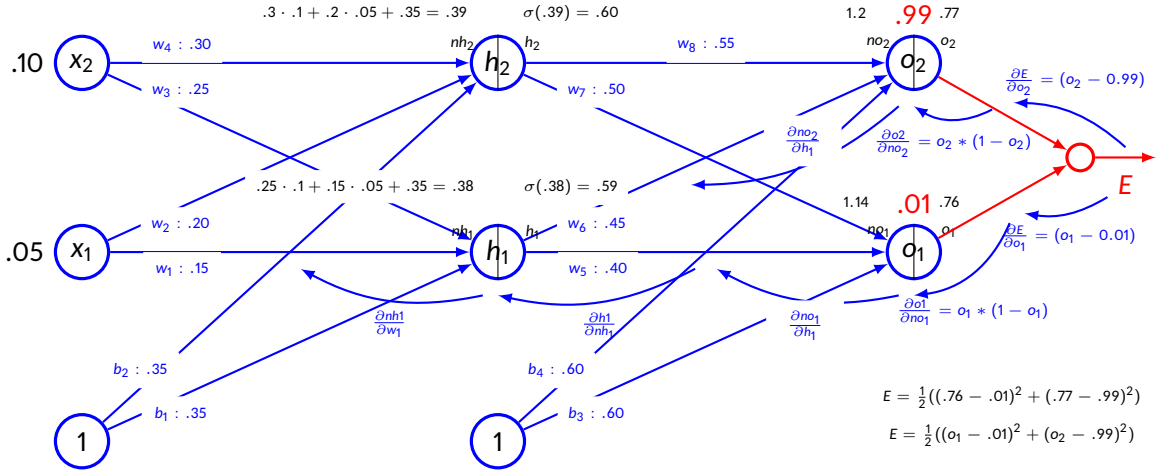
# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
$x_2$
$w_4$ : .30
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_3$ : .25
$w_7$ : .50
$\frac{\partial E}{\partial o_2} = (o_2 - 0.99)$
$\frac{\partial no_2}{\partial h_1}$
$\frac{\partial o2}{\partial no_2} = o_2 * (1 - o_2)$
E
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
$x_1$
$w_2$ : .20
$nh_1$
$h_1$
$w_6$ : .45
$no_1$
$o_1$
$\frac{\partial E}{\partial o_1} = (o_1 - 0.01)$
$w_1$ : .15
$w_5$ : .40
$\frac{\partial h1}{\partial nh_1}$
$\frac{\partial no_1}{\partial h_1}$
$\frac{\partial o1}{\partial no_1} = o_1 * (1 - o_1)$
$b_2$ : .35
$b_4$ : .60
$b_1$ : .35
$b_3$ : .60
1
1
$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$
$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$

# Example


.3 · .1 + .2 · .05 + .35 = .39
σ(.39) = .60
1.2
.99
.77
.10
$x_2$
$w_4$ : .30
$nh_2$
$h_2$
$w_8$ : .55
$no_2$
$o_2$
$w_3$ : .25
$w_7$ : .50
$\frac{\partial E}{\partial o_2} = (o_2 - 0.99)$
$\frac{\partial no_2}{\partial h_1}$
$\frac{\partial o2}{\partial no_2} = o_2 * (1 - o_2)$
E
.25 · .1 + .15 · .05 + .35 = .38
σ(.38) = .59
1.14
.01
.76
.05
$x_1$
$w_2$ : .20
$nh_1$
$h_1$
$w_6$ : .45
$no_1$
$o_1$
$\frac{\partial E}{\partial o_1} = (o_1 - 0.01)$
$w_1$ : .15
$w_5$ : .40
$\frac{\partial nh1}{\partial w_1}$
$\frac{\partial h1}{\partial nh_1}$
$\frac{\partial no_1}{\partial h_1}$
$\frac{\partial o1}{\partial no_1} = o_1 * (1 - o_1)$
$b_2$ : .35
$b_4$ : .60
$b_1$ : .35
$b_3$ : .60
1
1
$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$
$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$

# Example




$$E = \frac{1}{2}((.76 - .01)^2 + (.77 - .99)^2)$$

$$E = \frac{1}{2}((o_1 - .01)^2 + (o_2 - .99)^2)$$

$$\frac{\partial E}{\partial w_1} = \left(\frac{\partial E}{\partial o_1} \times \frac{\partial o_1}{\partial no_1} \times \frac{\partial no_1}{\partial h1} + \frac{\partial E}{\partial o_2} \times \frac{\partial o_2}{\partial no_2} \times \frac{\partial no_2}{\partial h1}\right) \times \frac{\partial h1}{\partial nh1} \times \frac{\partial nh1}{\partial w_1}$$

# Optimization

# Minimization of cost function

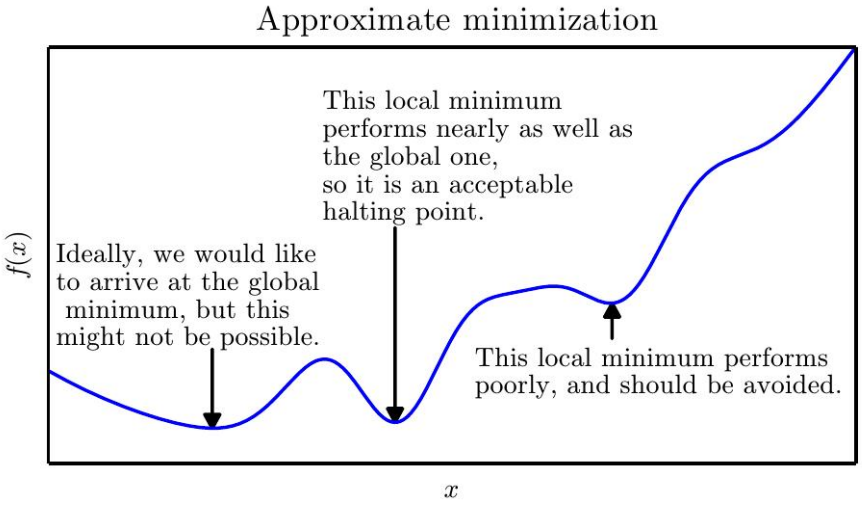



Image source: Deep Learning Book

# Problem of optimization

- Differs from traditional pure optimization problem
- Performance of a task is optimized indirectly
- We optimize $J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x}, \boldsymbol{\theta}), y)$ where $\hat{p}$ is the empirical distribution
- We would like to optimize $J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim p_{\text{data}}} L(f(\boldsymbol{x}, \boldsymbol{\theta}), y)$ where $p$ is the data generating distribution
  - Also known as risk
- We hope minimizing $J$ will minimize $J^*$

# Surrogate loss function

- Loss function may not be optimized efficiently
  - Exact minimization of 0-1 loss is typically intractable
- Surrogate loss function is used
  - Proxy function for the actual loss function
  - Negative log likelihood of correct class used as surrogate function
- There are cases when surrogate loss function results in better learning
  - 0-1 loss of test set often continues to decrease for a long time after training set 0-1 loss has reached to 0
- A training algorithm does not halt at local minima usually
  - Tries to minimize surrogate loss function but halts when validation loss starts to increase
- Training function can halt when surrogate function has huge derivative

# Batch

- Objective function usually decomposes as a sum over training example
- Typically in machine learning update of parameters is done based on an expected value of the cost function estimated using only a subset of the terms of full cost function
- Maximum likelihood problem $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Maximizing this sum is equivalent to maximizing the expectation over empirical distribution $J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}, y, \boldsymbol{\theta})$

# Batch (contd.)

- Common gradient is given by $\nabla_\theta = \mathbb{E}_{(x,y)\sim\hat{p}_{\text{data}}} \nabla_\theta \log p_{\text{model}}(x, y, \theta)$
  - It becomes expensive as we need to compute for all examples
  - Random sample is chosen, then average of the same is taken
  - Standard error in mean is $\frac{\sigma}{\sqrt{n}}$ where $\sigma$ is the true standard deviation
  - Redundancy in training examples is an issue
- Optimization algorithm that uses entire training set is called batch of deterministic gradient descent
- Optimization algorithm that uses single example at a time is known as stochastic gradient descent or online method

# Minibatch

- Larger batch provides more accurate estimate of the gradient but with lesser than linear returns
- Multicore architecture are usually underutilized by small batches
- If all examples are to be processed parallelly then the amount of memory scales with batch size
- Sometime, better run time is observed with specific size of the array
- Small batch can add regularization effect due to noise they add in learning process
- Methods that update the parameters based on $g$ only are usually robust and can handle small batch size $\sim 100$

# Issues in optimization

- Ill conditioning
- Local minima
- Plateaus
- Saddle points
- Flat region
- Cliffs
- Exploding gradients
- Vanishing gradients
- Long term dependencies
- Inexact gradients

# Stochastic gradient descent

- **Inputs — Learning rate ($\epsilon_k$), weight parameters ($\boldsymbol{\theta}$)**
- **Algorithm for SGD:**

  **while stopping criteria not met**

  **Sample a minibatch $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$**

  **Estimate of gradient** $\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta} L\left(f\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right), y^{(i)}\right)$

  **Update parameters** $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon_k \hat{g}$

  **end while**

# Stochastic gradient descent

- Learning rate is a crucial parameter
- Learning rate $\epsilon_k$ is used in the $k$th iteration
- Gradient does not vanishes even when we reach minima as minibatch can introduce noise
- True gradient becomes small and then 0 when batch gradient descent is used
- Sufficient condition on learning rate for convergence of SGD
  - $\sum_{k=1}^{\infty} \epsilon_k = \infty, \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$
- Common way is to decay the learning rate $\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$ with $\alpha = \frac{k}{\tau}$

# Stochastic gradient descent

- Choosing learning rate is an art than science!
  - Typically $\epsilon_\tau$ is 1% of $\epsilon_0$
- SGD usually performs well for most of the cases
- For large task set SGD may converge within the fixed tolerance of final error before it has processed all training examples

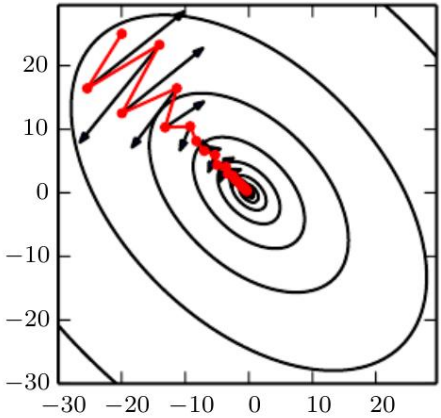
# Momentum

- SGD is the most popular. However, learning may be slow sometime
- Idea is to accelerate learning especially in high curvature, small but consistent gradients
- Accumulates an exponential decaying moving average of past gradients and continue to move in that direction
- Introduces a parameter $v$ that play the role of velocity
  - The velocity is set to an exponentially decaying average of negative gradients
- Update is given by

$$\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left( \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

  - $\alpha$ — hyperparameter, denotes the decay rate

# Momentum



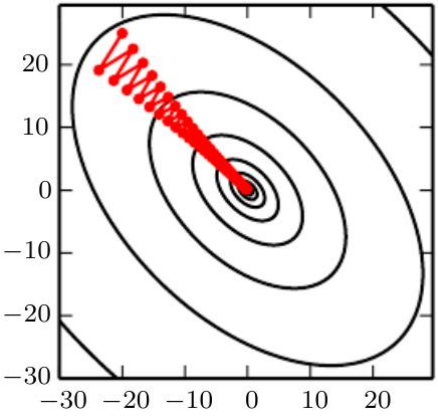




Image source: Deep Learning Book

# SGD with momentum

- **Inputs — Learning rate ($\epsilon$), weight parameters ($\theta$), momentum parameter ($\alpha$), initial velocity ($v$)**
- **Algorithm:**

  **while stopping criteria not met**

  **Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$**

  **Estimate of gradient: $g = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(x^{(i)}, \theta), y^{(i)})$**

  **Update of velocity: $v = \alpha v - \epsilon g$**

  **Update parameters: $\theta = \theta + v$**

  **end while**

# Momentum

- The step size depends on how large and how aligned a sequence gradients are
- Largest when many successive gradients are in same direction
- If it observes $\boldsymbol{g}$ always, then it will accelerate in $-\boldsymbol{g}$ with terminal velocity $\frac{\epsilon|\boldsymbol{g}|}{1-\alpha}$
- Typical values for $\alpha$ is 0.5, 0.9, 0.99. However this parameter can be adapted.

# Parameter initialization

- Training algorithms are iterative in nature
- Require to specify initial point
- Training deep model is difficult task and affected by initial choice
  - Convergence
  - Computation time
  - Numerical instability
- Need to break symmetry while initializing the parameters

# Adaptive learning rate

- Learning rate can affect the performance of the model
- Cost may be sensitive in one direction and insensitive in the other directions
- If partial derivative of loss with respect to model remains the same sign then the learning rate should increase
  - Applicable for full batch optimization

# AdaGrad

- Adapts the learning rate of all parameters by scaling them inversely proportional to the square root of the sum of all historical squared values of the gradient
  - Parameters with largest partial derivative of the loss will have rapid decrease in learning rate and vice-versa
  - Net effect is greater progress
- It performs well on some models

# Steps for AdaGrad

- **Inputs — Global learning rate ($\epsilon$), weight parameters ($\theta$), small constant ($\delta$), gradient accumulation ($r$)**
- **Algorithm:**

  **while stopping criteria not met**

  **Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$**

  **Gradient: $g = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(x^{(i)}, \theta), y^{(i)})$**

  **Accumulated squared gradient: $r = r + g \odot g$**

  **Update: $\Delta\theta = -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$**

  **Apply update: $\theta = \theta + \Delta\theta$**

  **end while**