Master Lecture 3

MLP, BP and Kernels

Pipeline

Raw Data → **Features Representation Embedding** → **ML Algorithm** →

Text

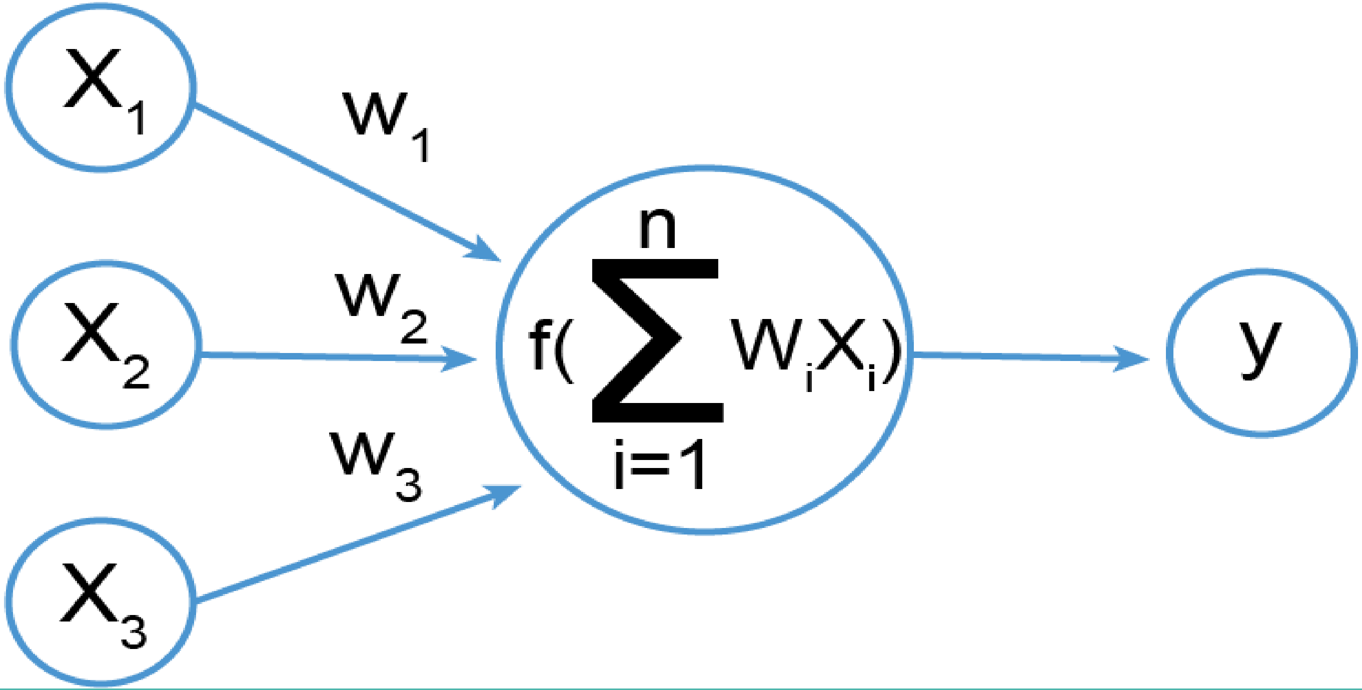
Speech

Image

[.. .. ..]

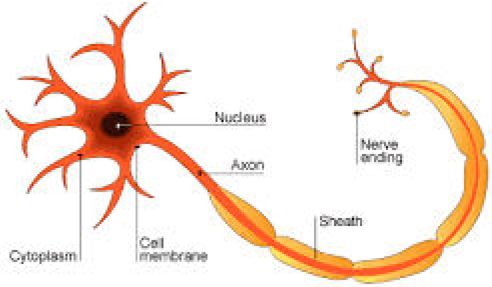
Focus:
Simple Methods
Linear Methods

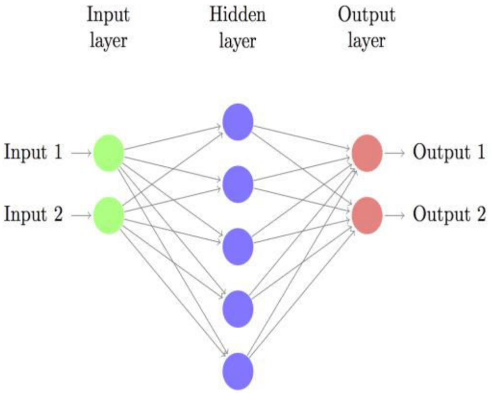
Understanding Linear Classifier as "Neuron"



Neural Networks

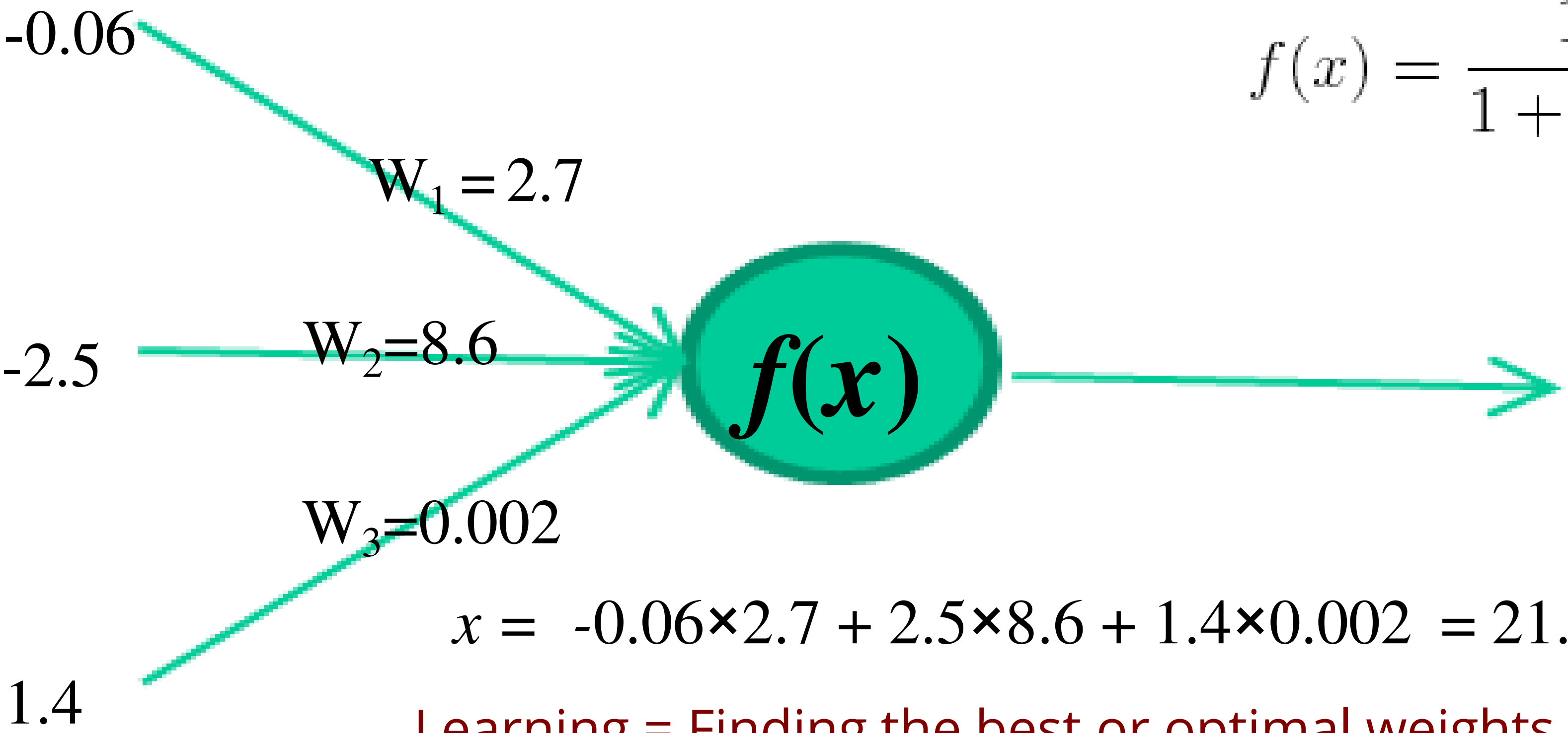
- Biologically inspired networks.
- Complex function approximation through composition of functions.
- Can learn arbitrary Nonlinear decision boundary





Learning in Neural Networks

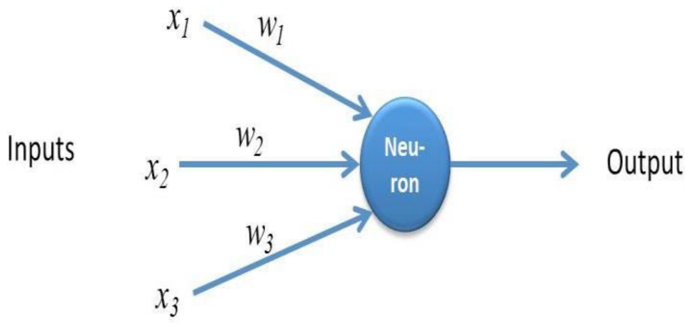
$$f(x) = \frac{1}{1+e^{-x}}$$

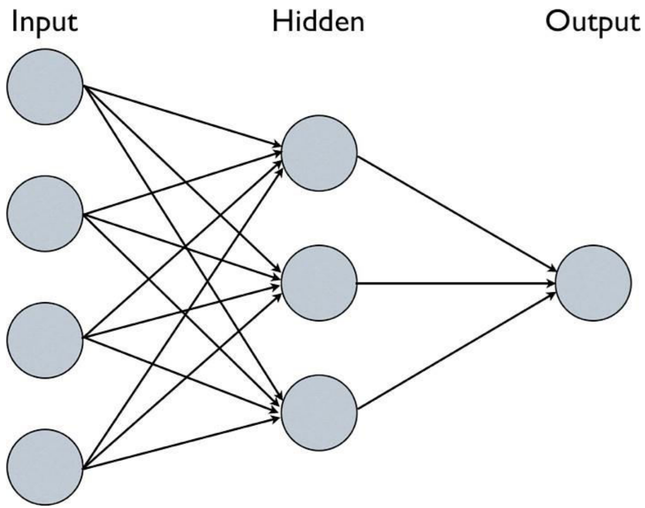


$$x = -0.06\times2.7 + 2.5\times8.6 + 1.4\times0.002 = 21.34$$

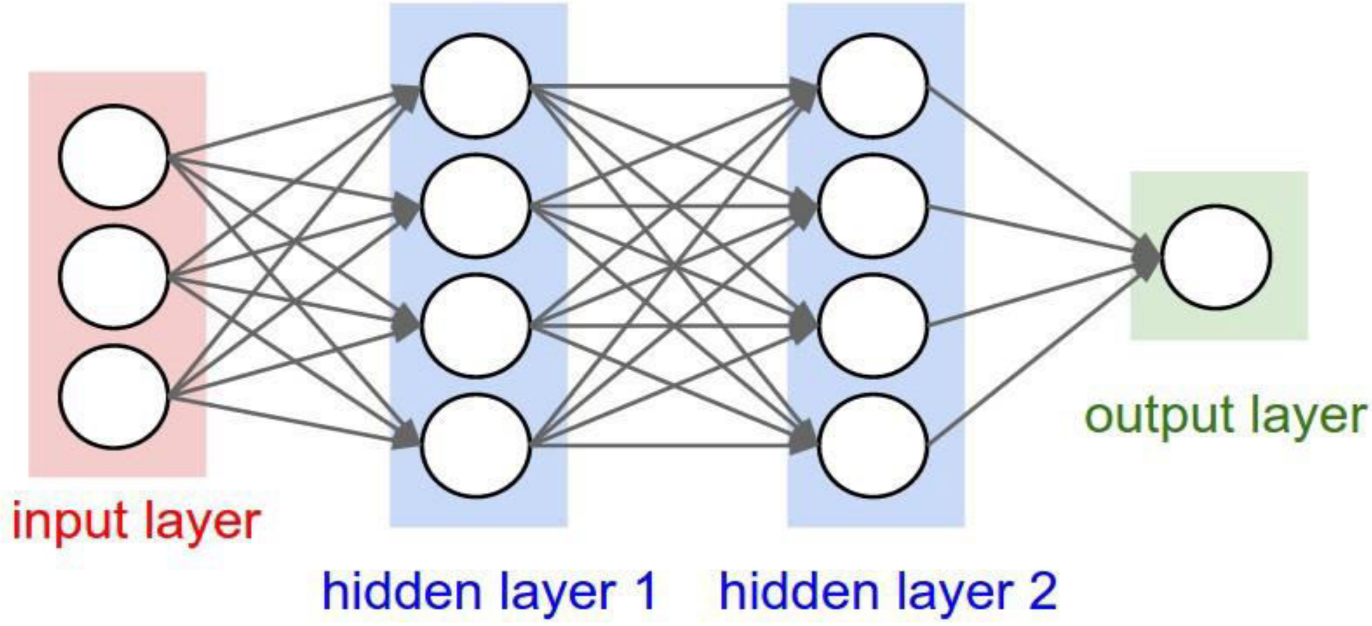
Learning = Finding the best or optimal weights

Single Layer Perceptron and Multi Layer Perceptron





Deep Neural Networks(Multi Layer Perceptron)

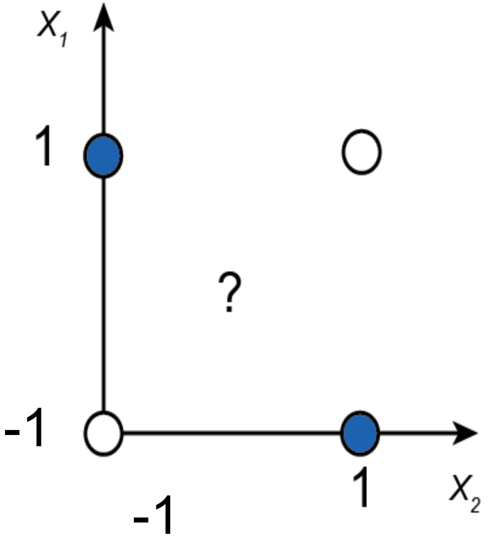


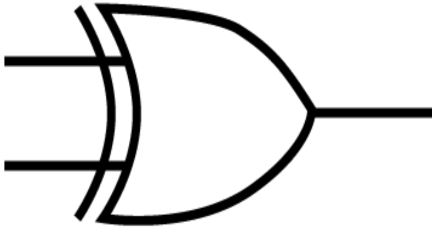
Multi layer Perceptron

Popular Artificial Neural Network

XOR: Limitation of Linear Methods

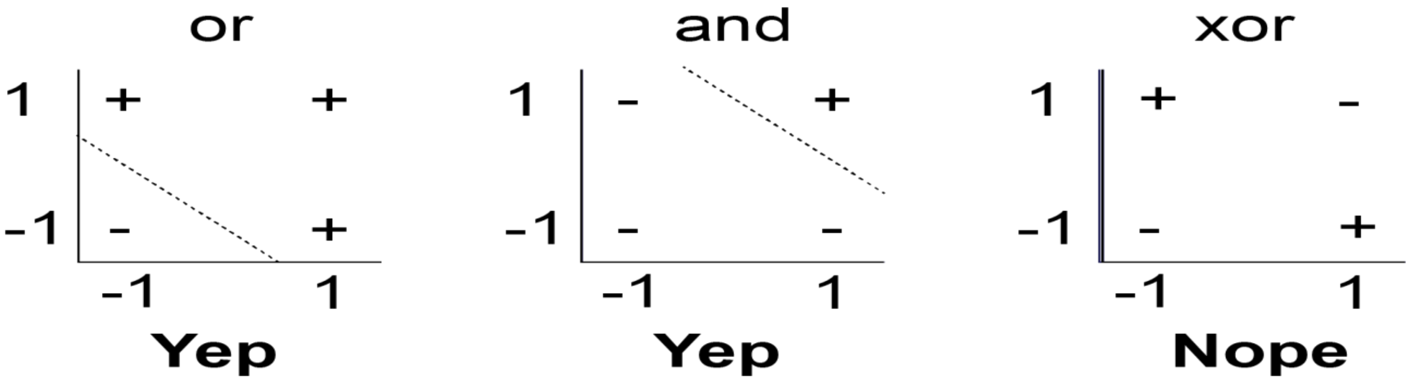
x_1	x_2	$x_1\, XOR\, x_2$
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1





XOR: Linear "Non Separability"

Not all concepts can be represented with linear functions.



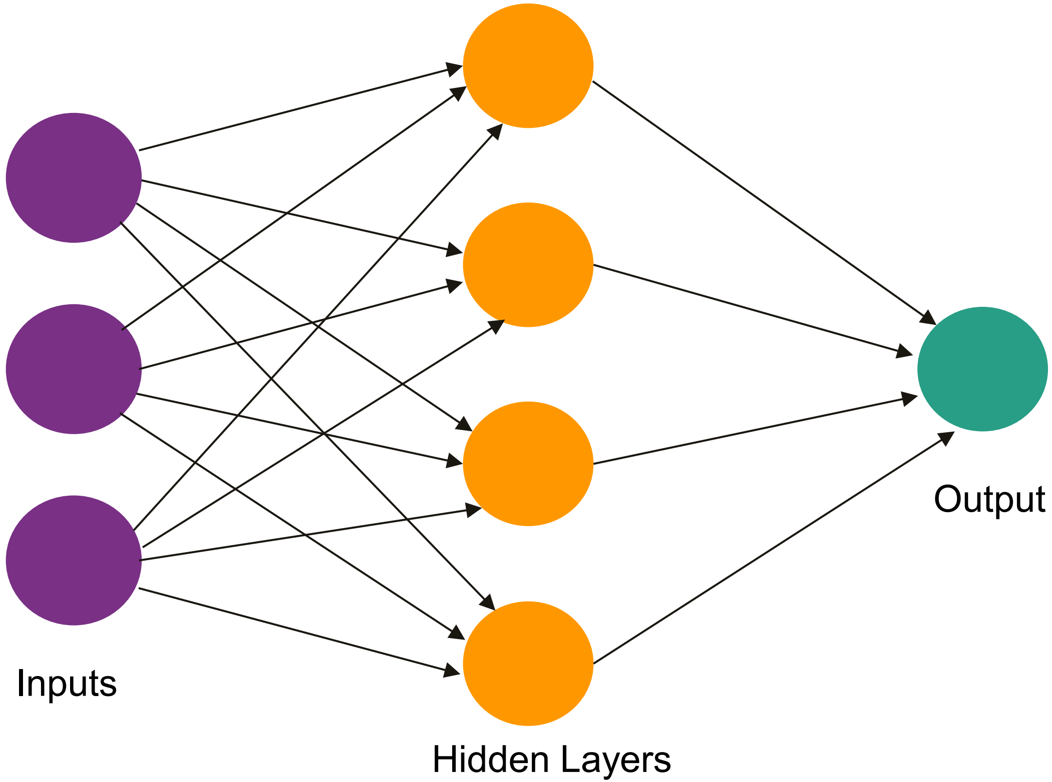
Example:
Estimate the price range of a house based on it’s attributes.

Bedrooms	Sq. Feet	Neighborhood (no. of houses in the locality)	Price high or low? High (1) , Low (0)
3	2000	90	1
2	800	143	0
2	850	167	0
1	550	267	0
4	2000	396	1

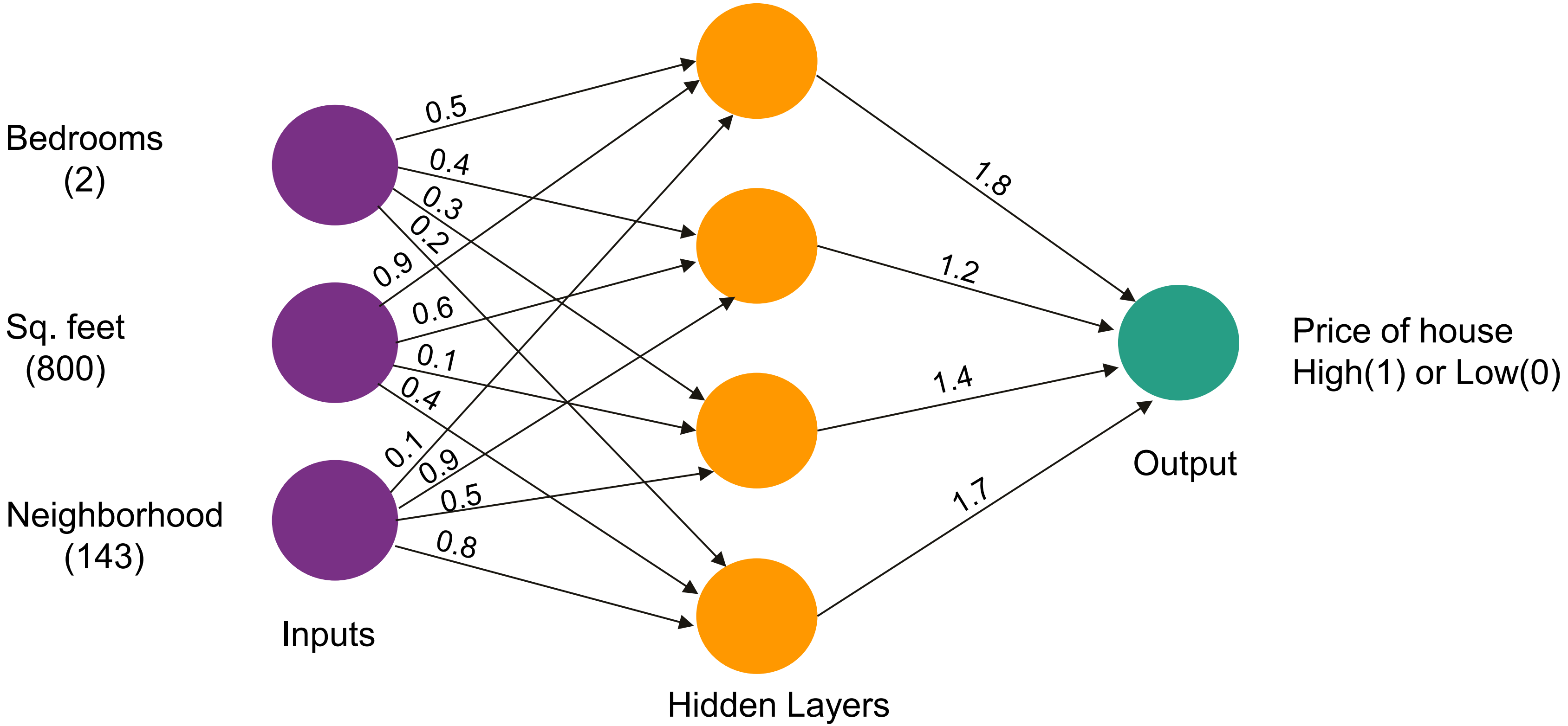
Use MLP to classify if price is high or low

- MLP consists of at least three layers of nodes. (input layer + hidden layer + output node)
- Each node (except input nodes) is a neuron that uses a nonlinear **activation function.**
- **Backpropagation** is used for training;

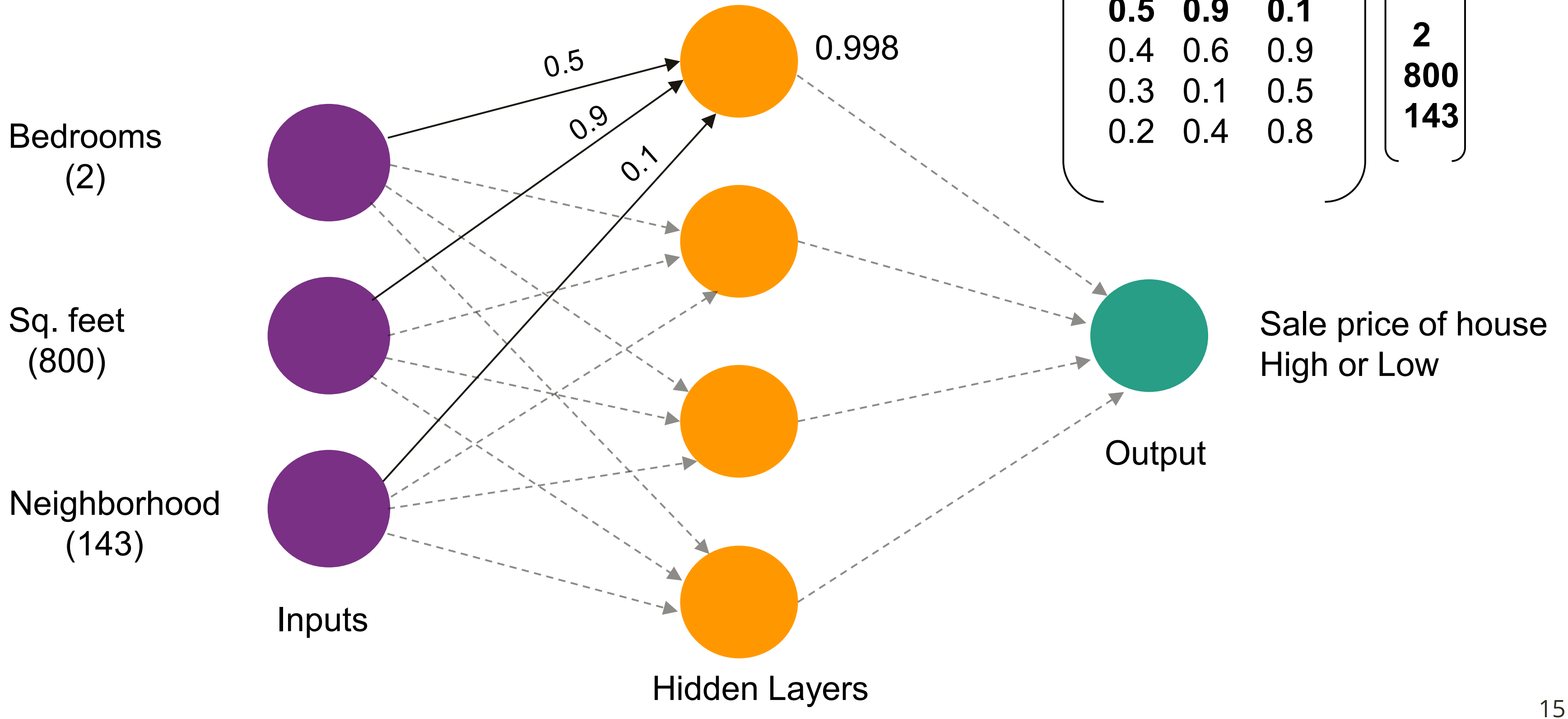
MLP



Initialize weights



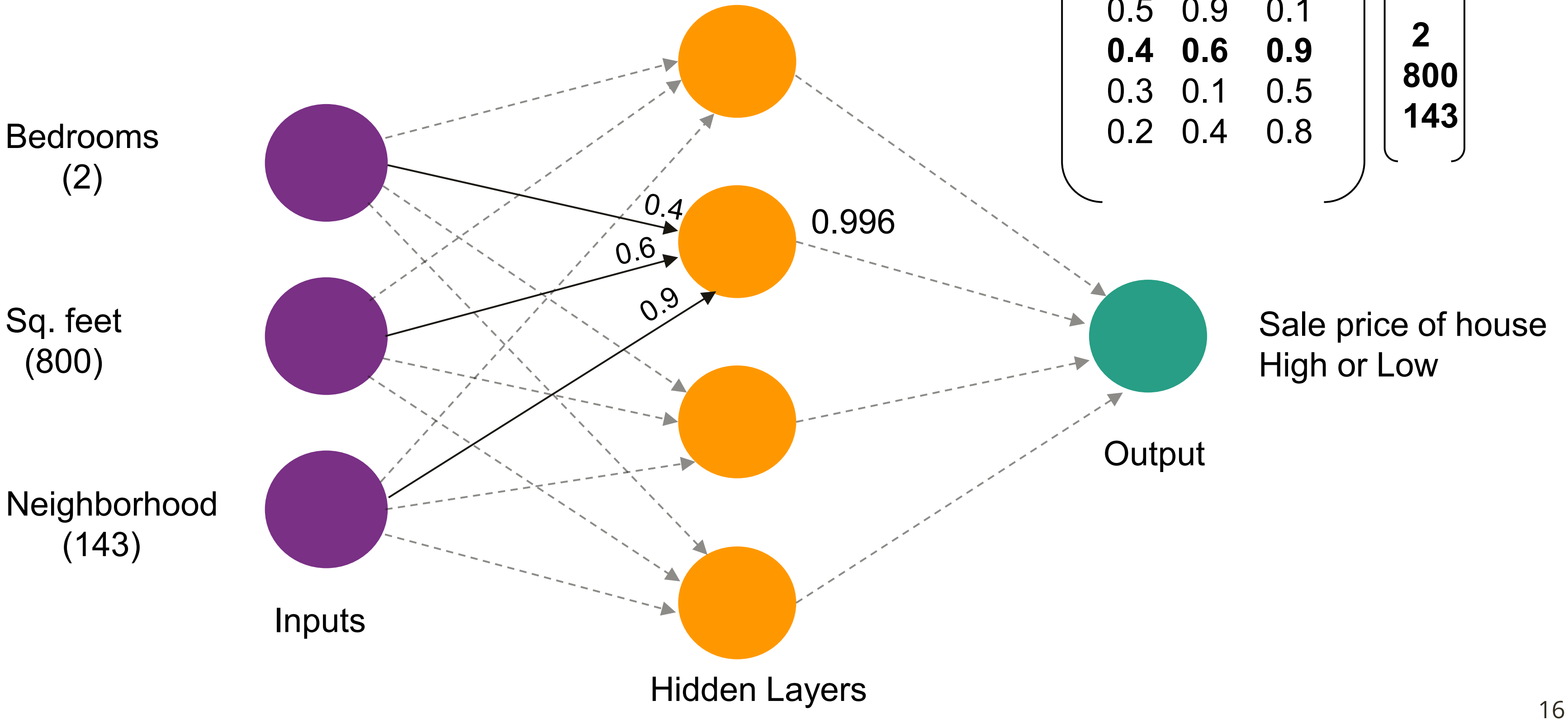
Weights at the first neuron



$$\begin{pmatrix} \mathbf{0.5} & \mathbf{0.9} & \mathbf{0.1} \\ 0.4 & 0.6 & 0.9 \\ 0.3 & 0.1 & 0.5 \\ 0.2 & 0.4 & 0.8 \end{pmatrix} \begin{pmatrix} \mathbf{2} \\ \mathbf{800} \\ \mathbf{143} \end{pmatrix}$$

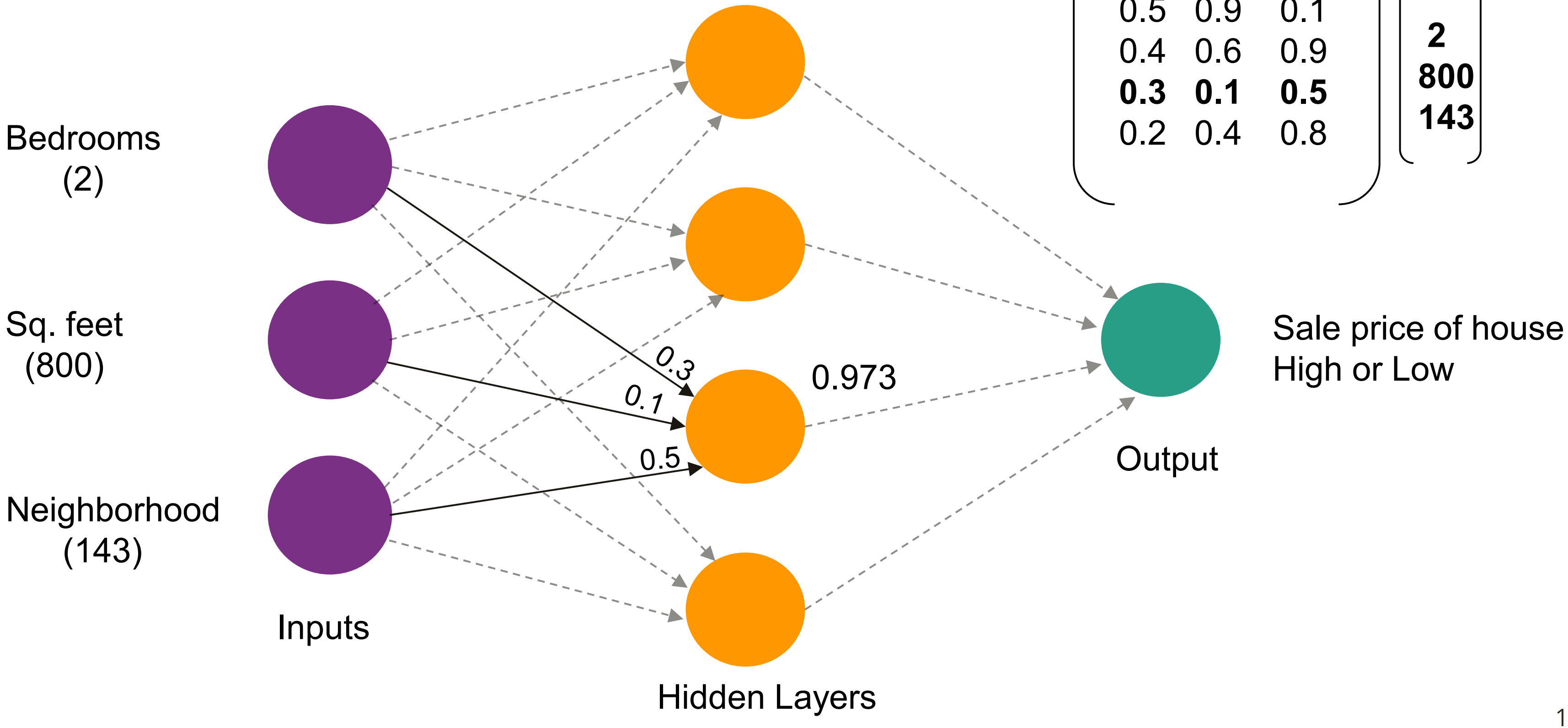
Weights at the second neuron

$$\begin{pmatrix} 0.5 & 0.9 & 0.1 \\ \mathbf{0.4} & \mathbf{0.6} & \mathbf{0.9} \\ 0.3 & 0.1 & 0.5 \\ 0.2 & 0.4 & 0.8 \end{pmatrix} \begin{pmatrix} \mathbf{2} \\ \mathbf{800} \\ \mathbf{143} \end{pmatrix}$$



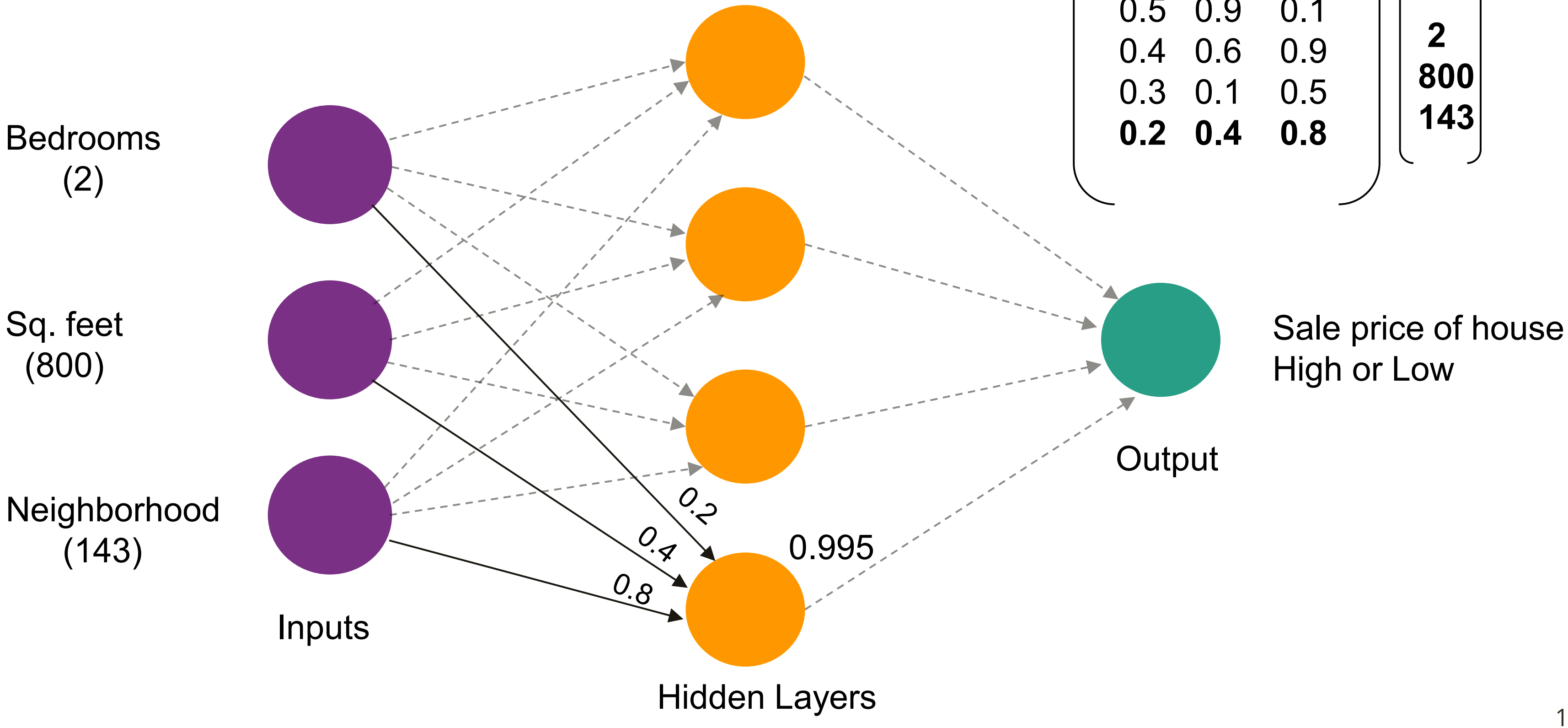
Weights at the third neuron

$$\begin{pmatrix} 0.5 & 0.9 & 0.1 \\ 0.4 & 0.6 & 0.9 \\ \mathbf{0.3} & \mathbf{0.1} & \mathbf{0.5} \\ 0.2 & 0.4 & 0.8 \end{pmatrix} \begin{pmatrix} \mathbf{2} \\ \mathbf{800} \\ \mathbf{143} \end{pmatrix}$$

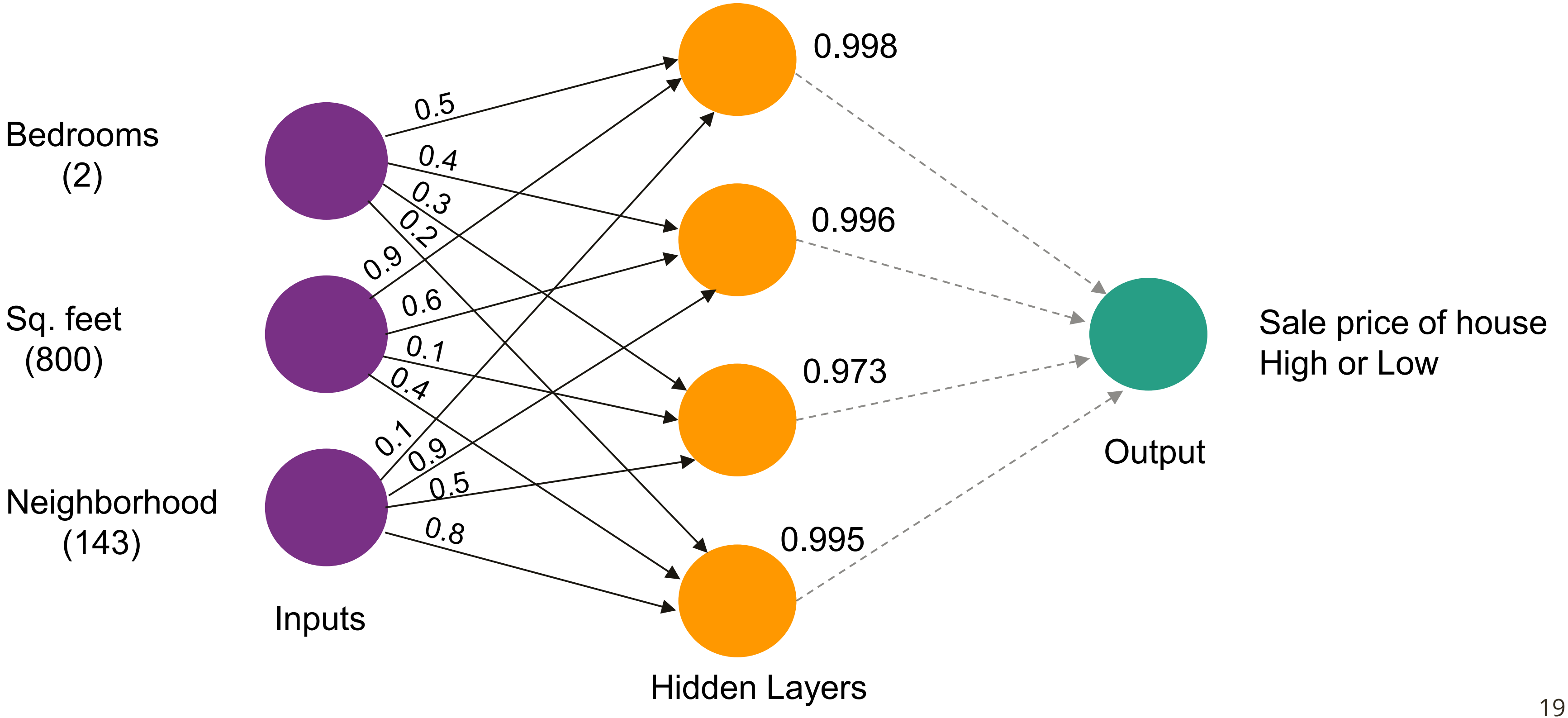


Weights at the fourth neuron

$$\begin{pmatrix} 0.5 & 0.9 & 0.1 \\ 0.4 & 0.6 & 0.9 \\ 0.3 & 0.1 & 0.5 \\ \mathbf{0.2} & \mathbf{0.4} & \mathbf{0.8} \end{pmatrix} \begin{pmatrix} \mathbf{2} \\ \mathbf{800} \\ \mathbf{143} \end{pmatrix}$$

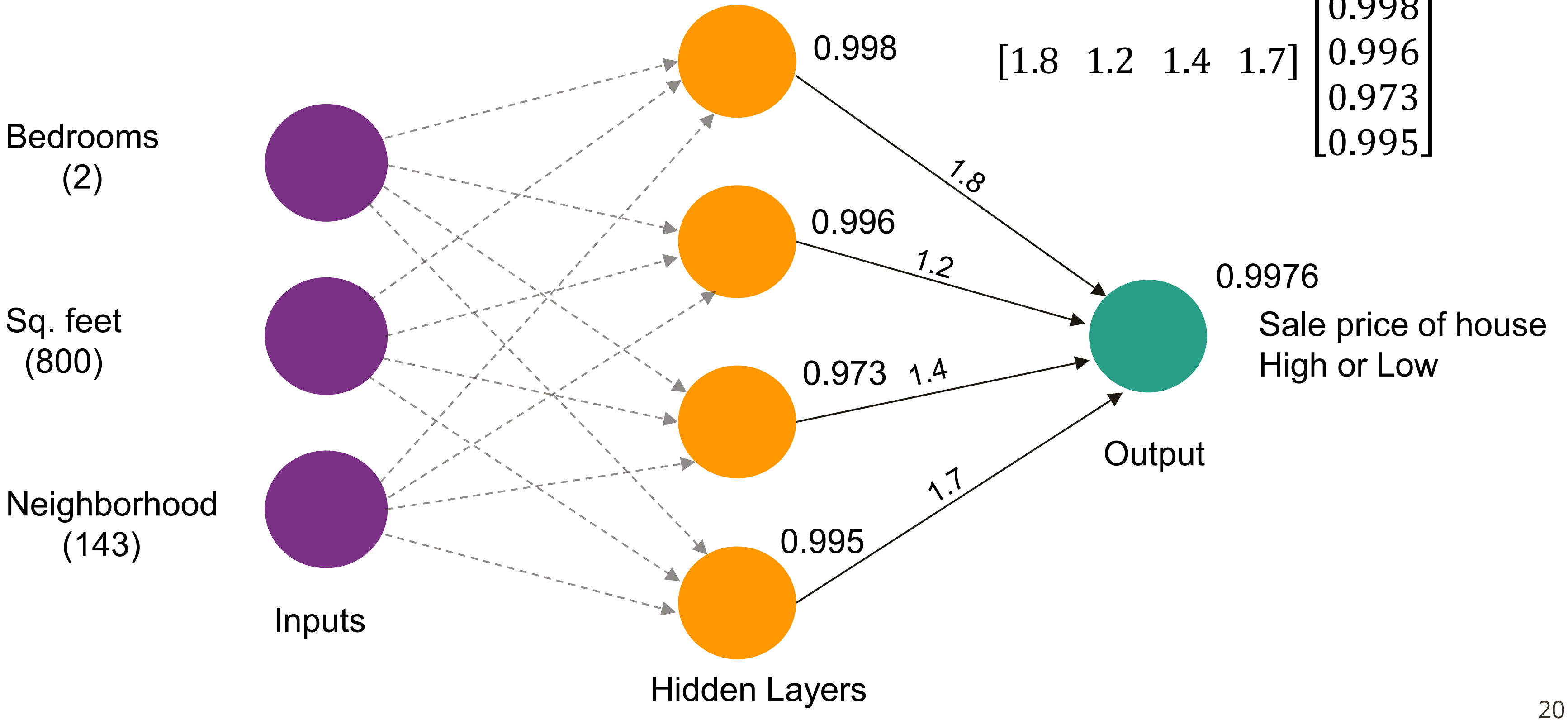


Inputs to the next layer

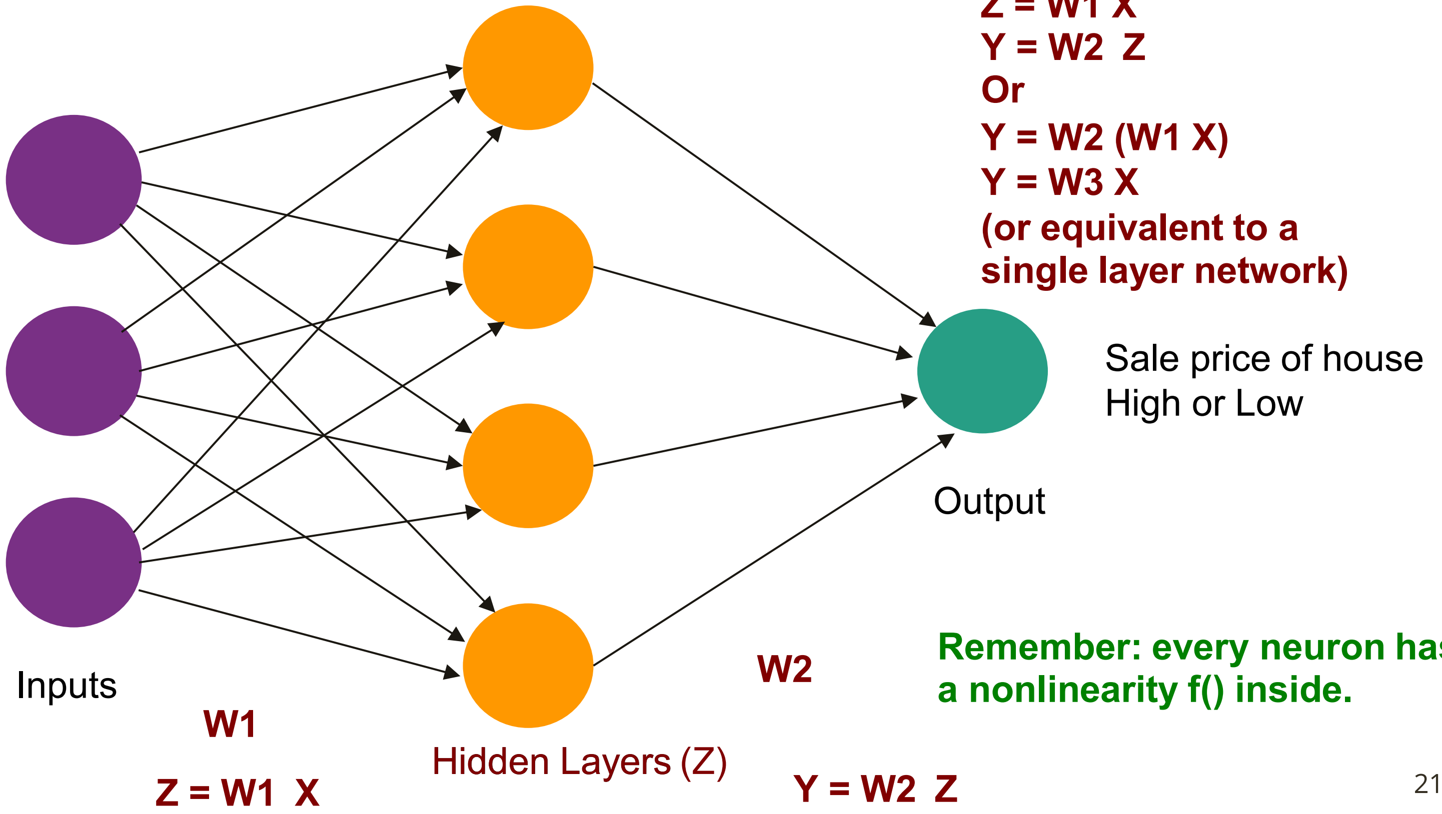


Computations in next layer

$$\begin{bmatrix} 1.8 & 1.2 & 1.4 & 1.7 \end{bmatrix} \begin{bmatrix} 0.998 \\ 0.996 \\ 0.973 \\ 0.995 \end{bmatrix}$$



Comment: Limitation of “Linear MLP”

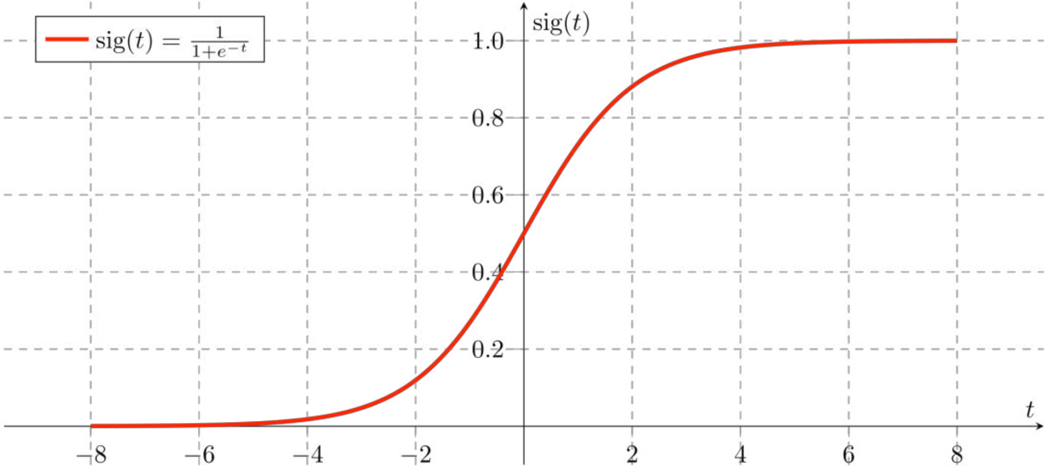


Z = W1 X
Y = W2 Z
Or
Y = W2 (W1 X)
Y = W3 X
(or equivalent to a single layer network)

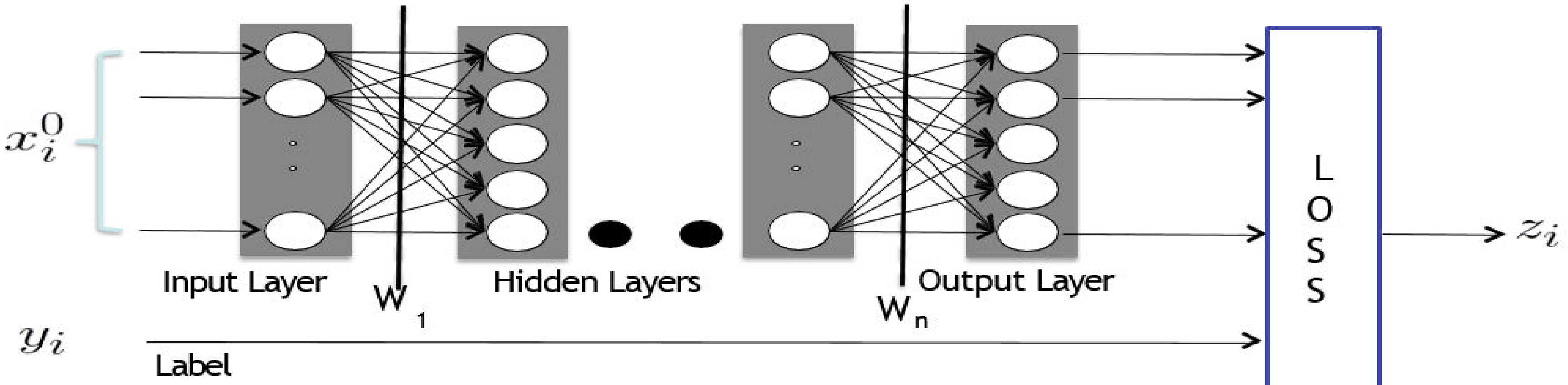
Remember: every neuron has a nonlinearity f() inside.

The nonlinear f(): Sigmoid

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



Loss or Objective

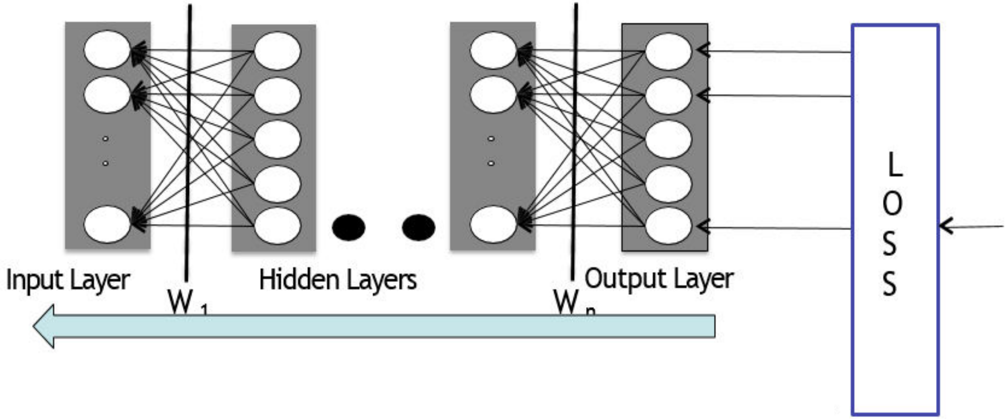


Objective: Find out the best parameters which will minimize the loss.

$$W^* = arg\min_W \sum_{i=1}^{N} L(x_i^n, y_i; W)$$ Weight Vector

$$z_i = \frac{1}{2} \| x_i^n - y_i \|_2^2$$ E.g. Squared Loss

Back propagation

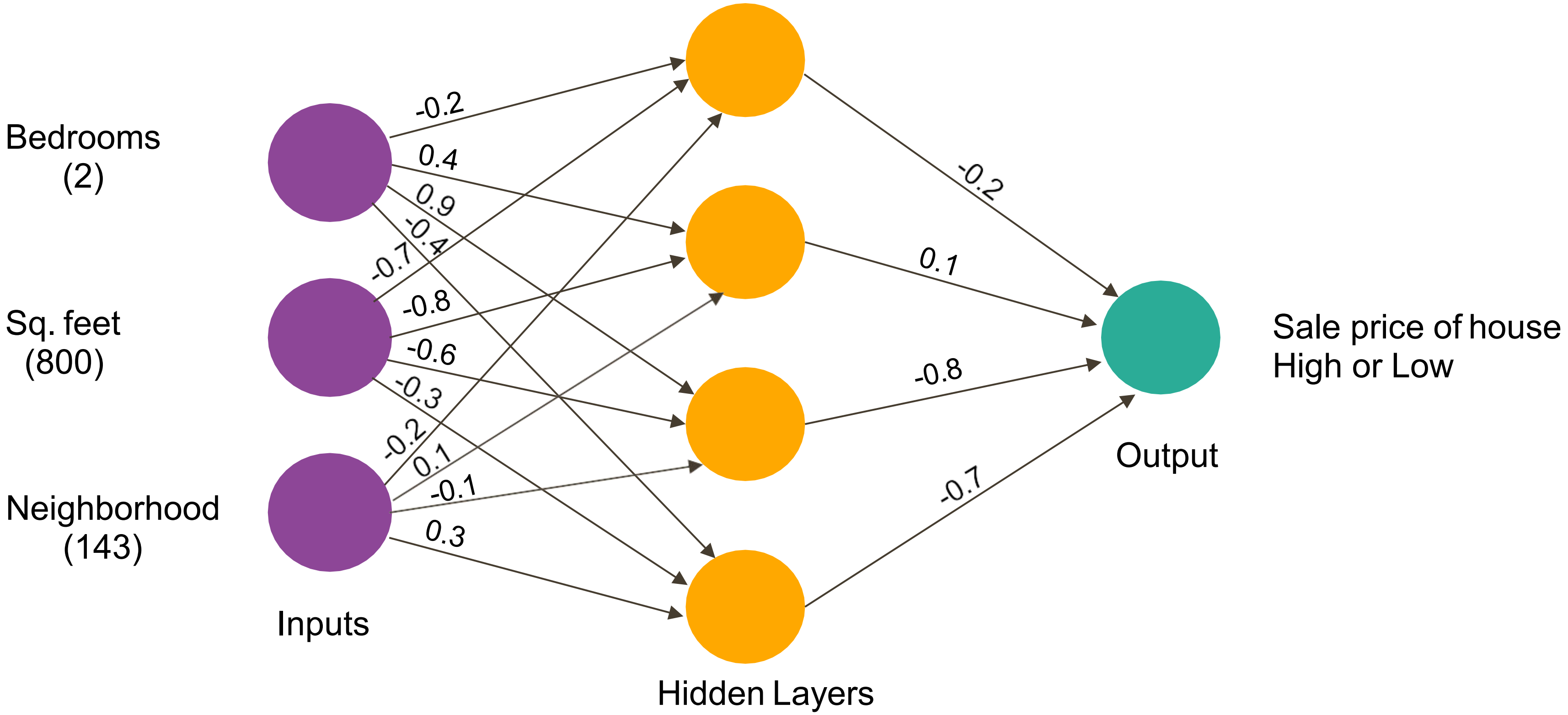


Solution: Iteratively update W along the direction where loss decreases.
Each layer weights are updated based on the derivative of its output w.r.t input and weights

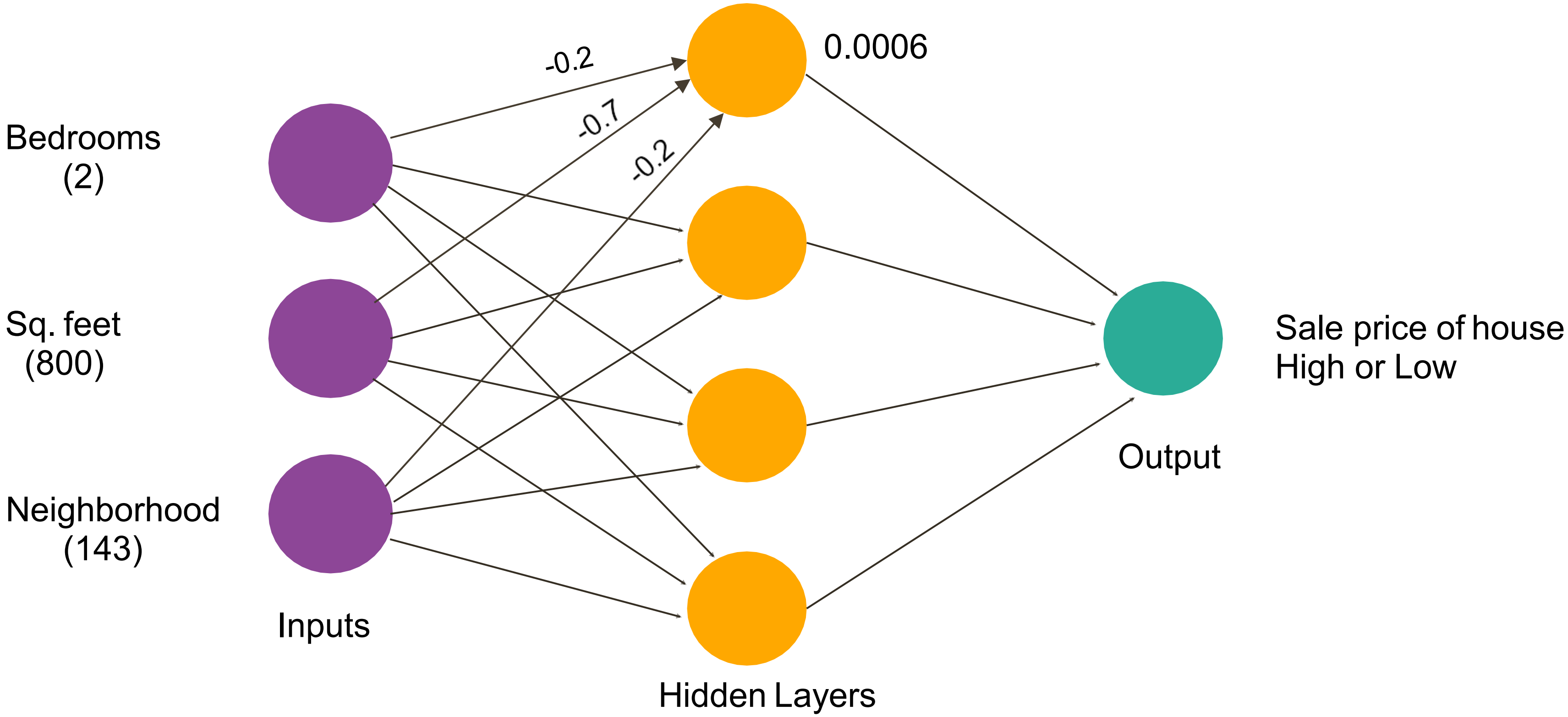
Loss or Error

- 0.998 * 1.8 + 0.996 * 1.2 + 0.973 * 1.4 + 0.995 * 1.7
  = f(6.04) = 0.9976
- The actual class (0) deviates from the predicted class (1)
- The squared error or loss is 0.9976* 0.9976 = 0.9952
- The weights need to be updated by **backpropagation to reduce the error**

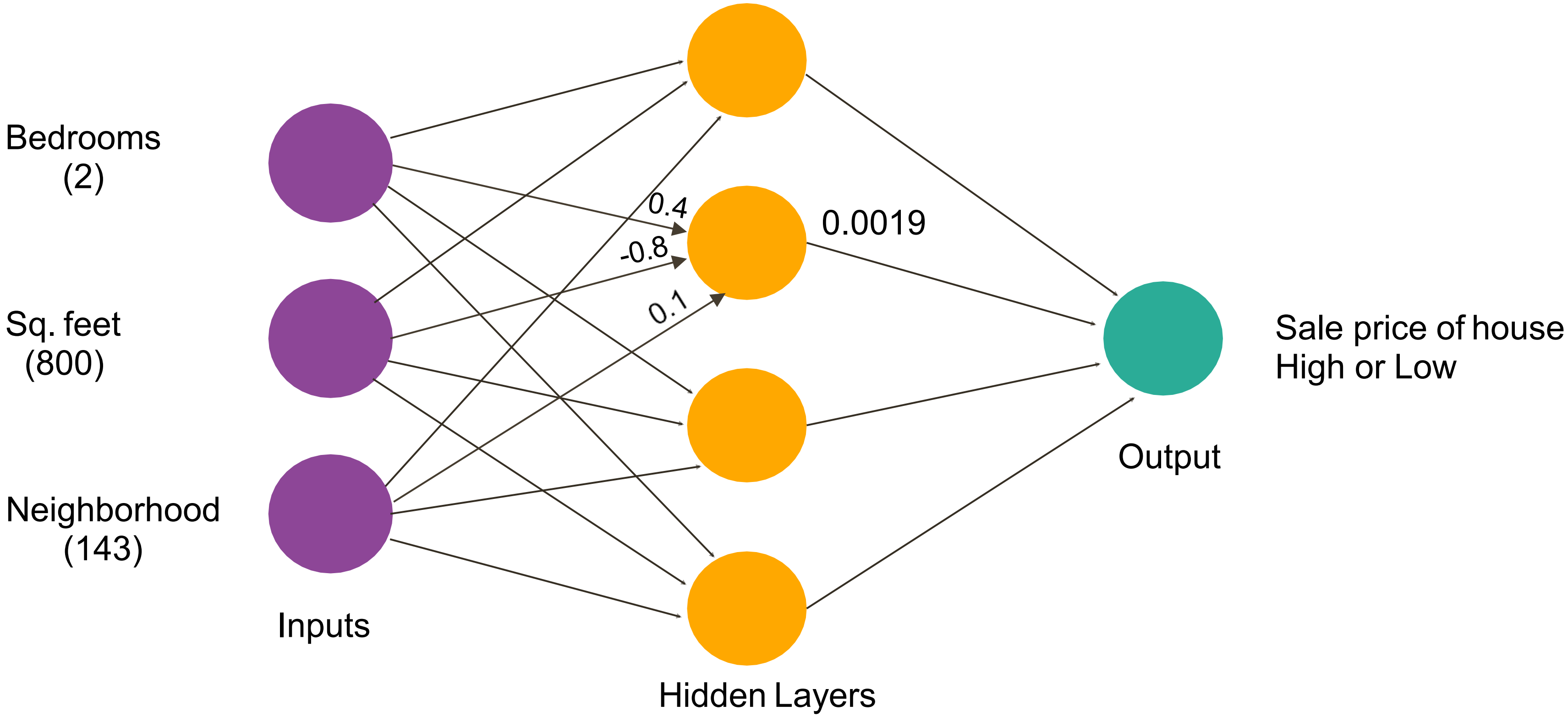
Weights obtained by backpropagation



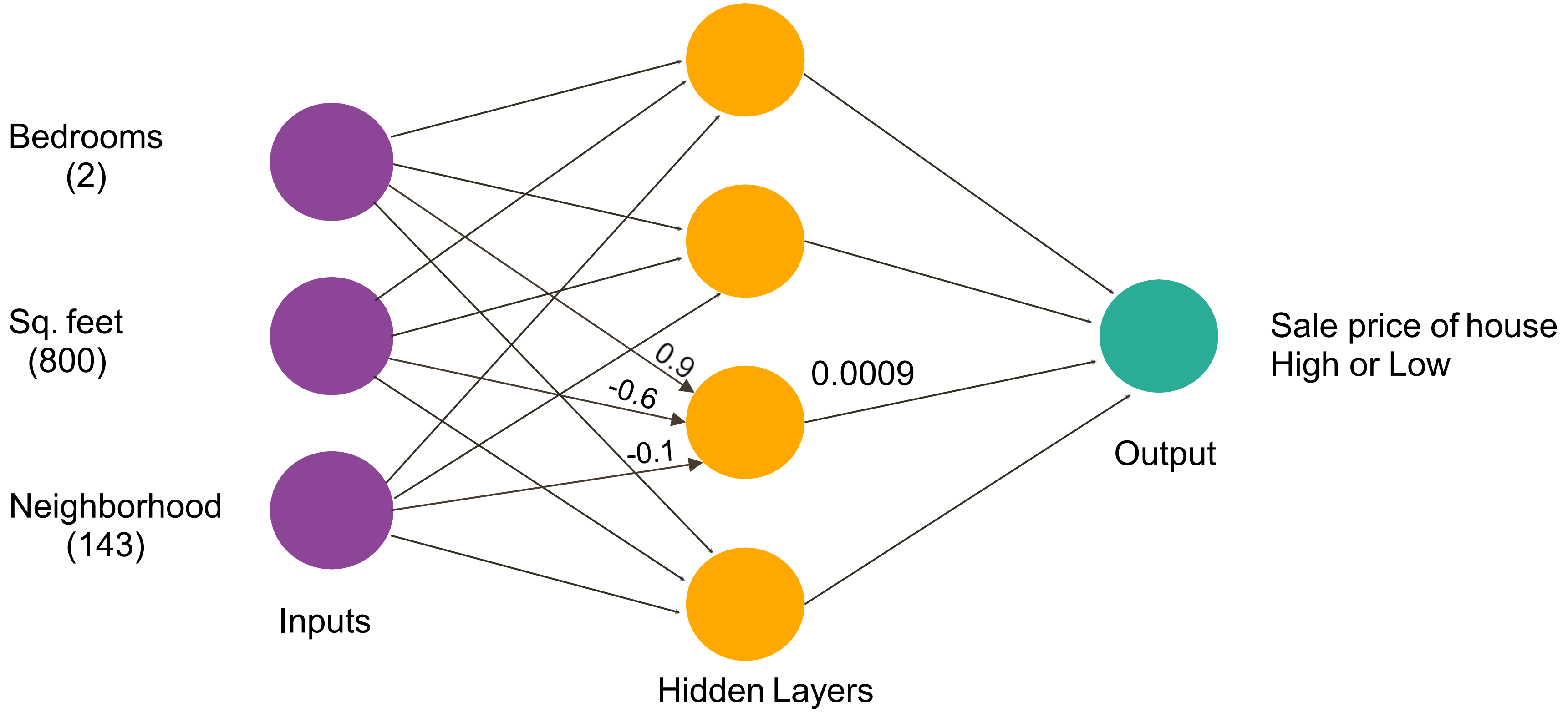
Weights at first neuron



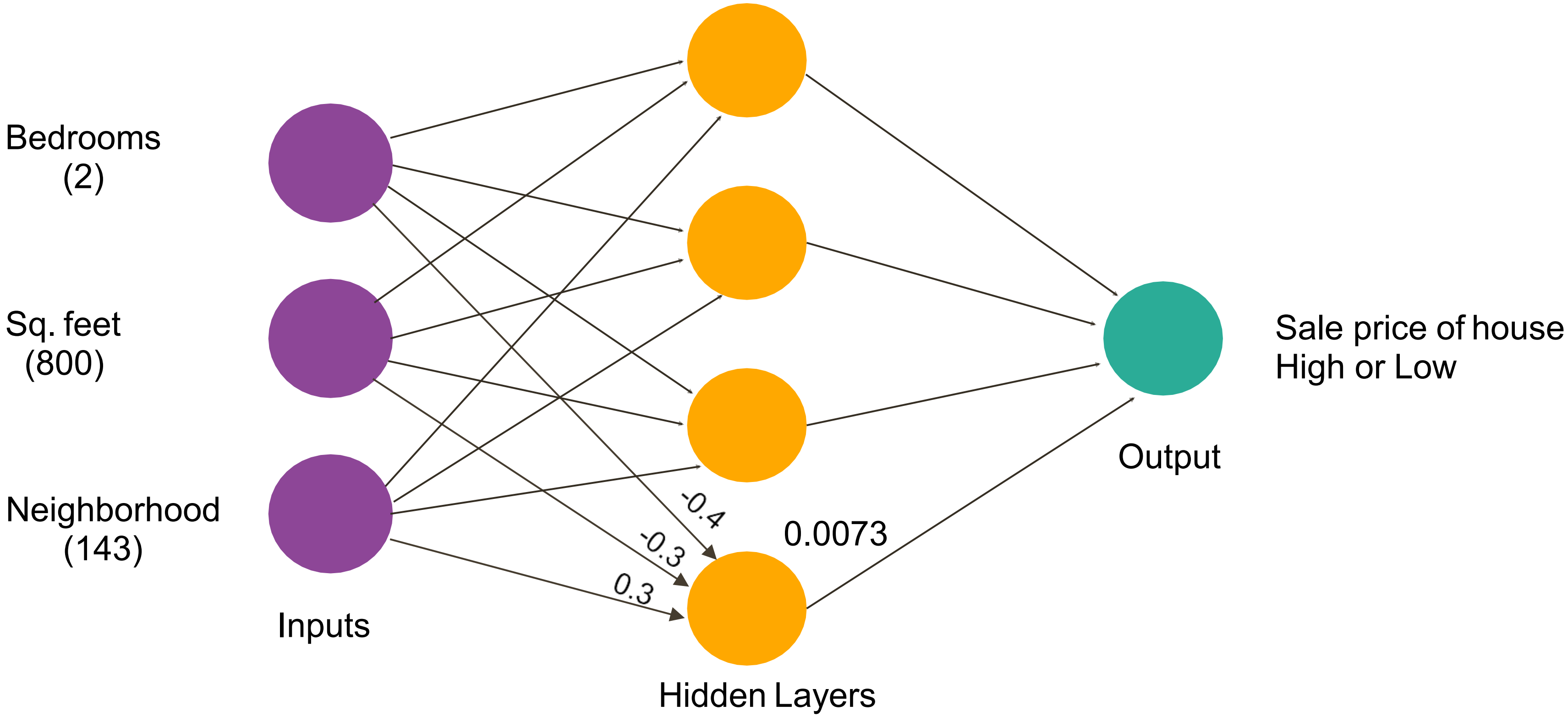
Weights at second neuron



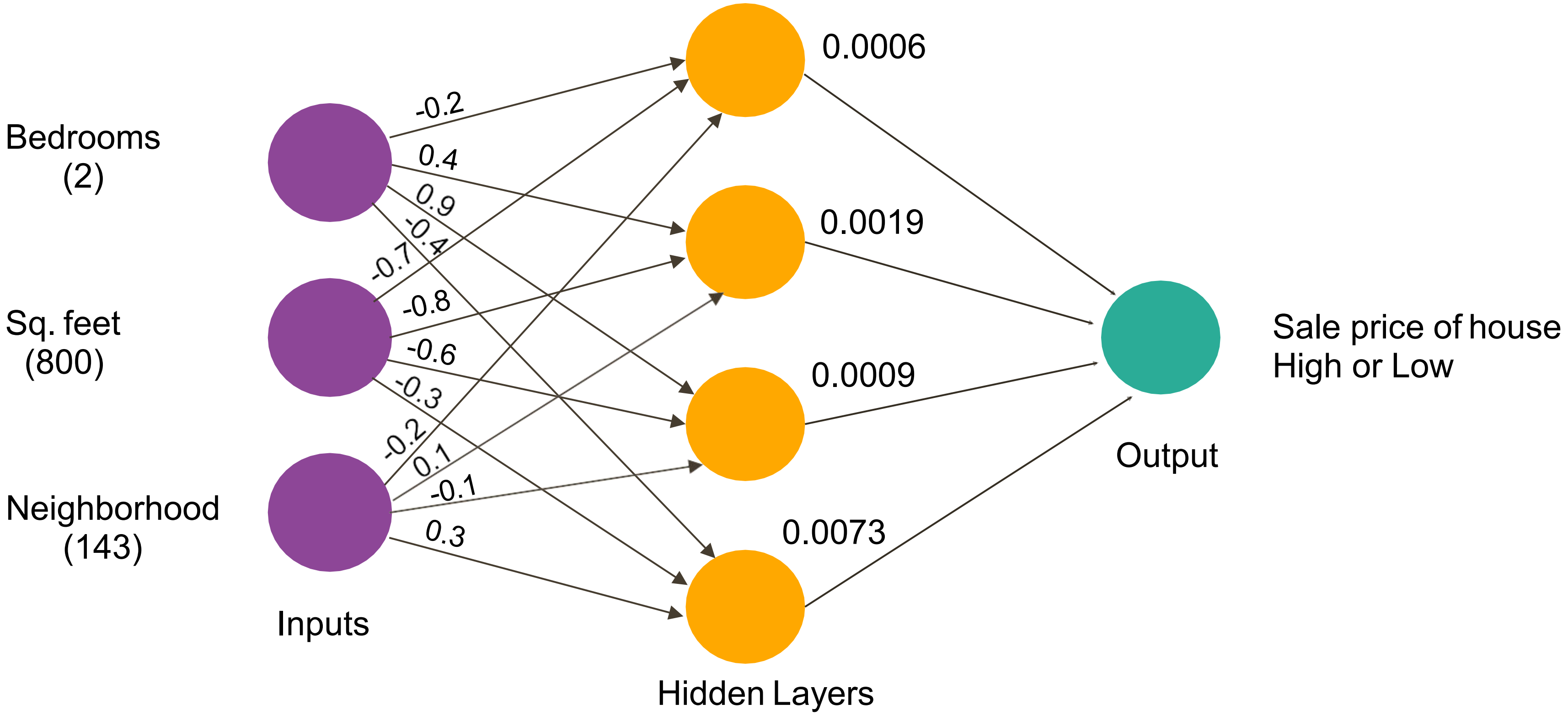
Weights at third neuron



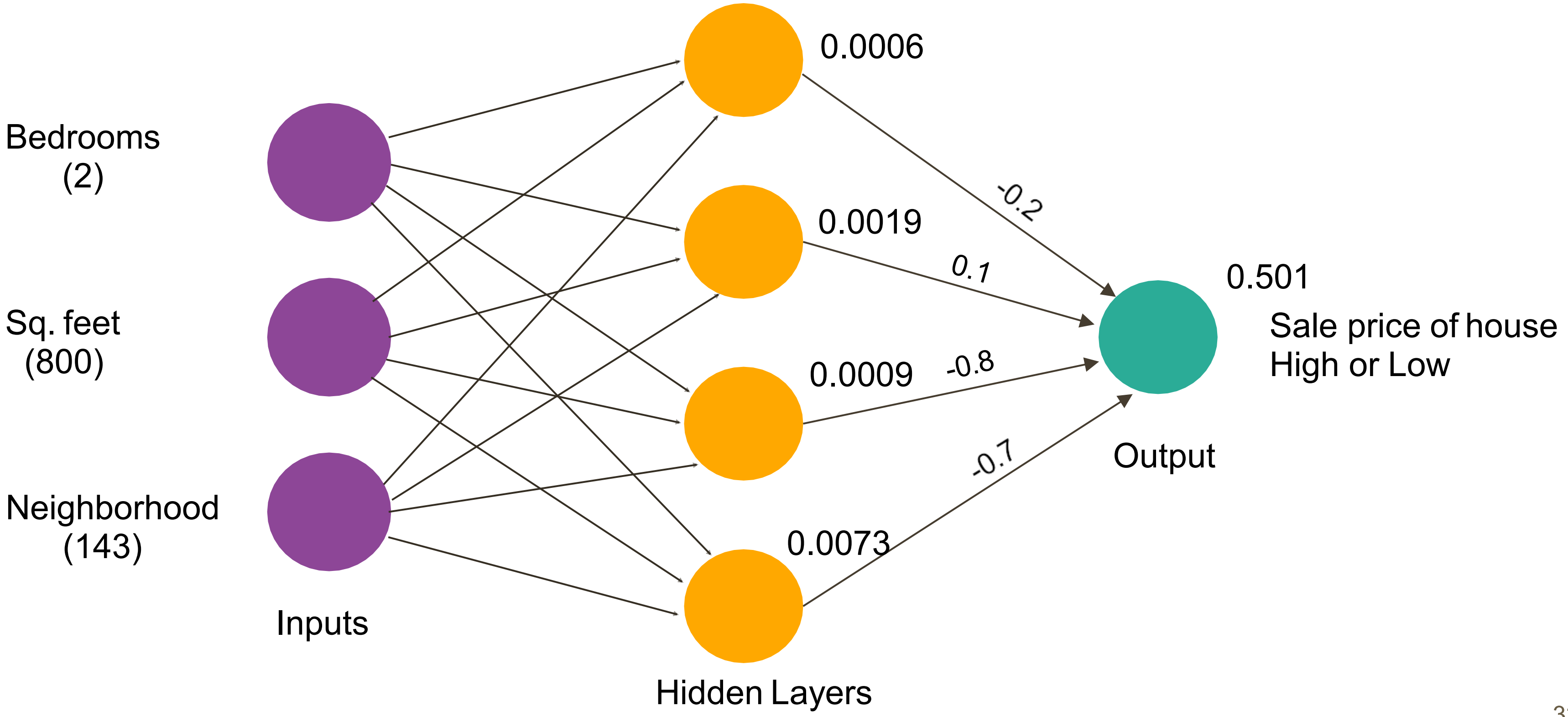
Weights at fourth neuron



Activation function applied at first layer



Activation function applied at second layer



Loss/ Error at this stage

- 0.0006 * -0.2 + 0.0019 * 0.1 + 0.0009 * -0.8 + 0.0073 * -0.7

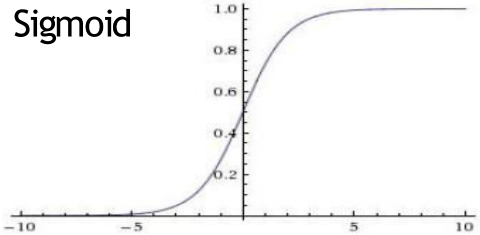
  = f(0.0057) = 0.501

- The square error or Loss is 0.501* 0.501 = 0.251
- After the weights have been updated by backpropagation the **error has reduced from 0.99 to 0.25**

Questions?

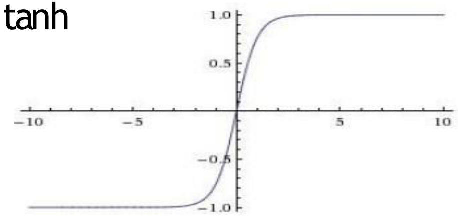
Activation Functions/ Nonlinearities

Sigmoid



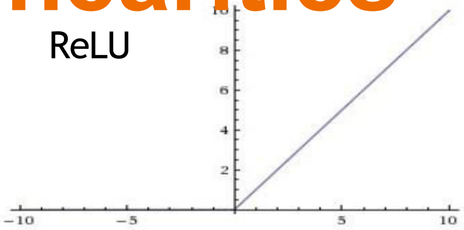
$$y = \frac{1}{1 + e^{-x}}$$

tanh



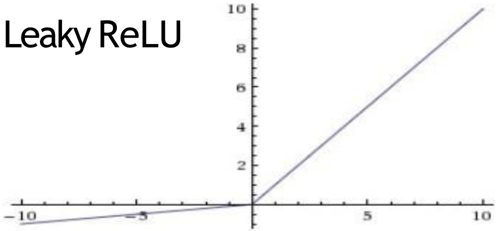
$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU



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

Leaky ReLU



$$y = \begin{cases} x & \text{if } x < 0 \\ 0.01x & \text{if } otherwise \end{cases}$$

maxout

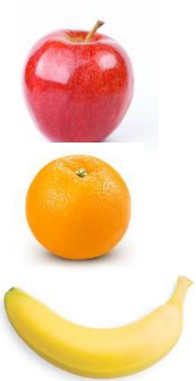
$$max(w_1^T x + b_1, w_2^T x + b_2)$$

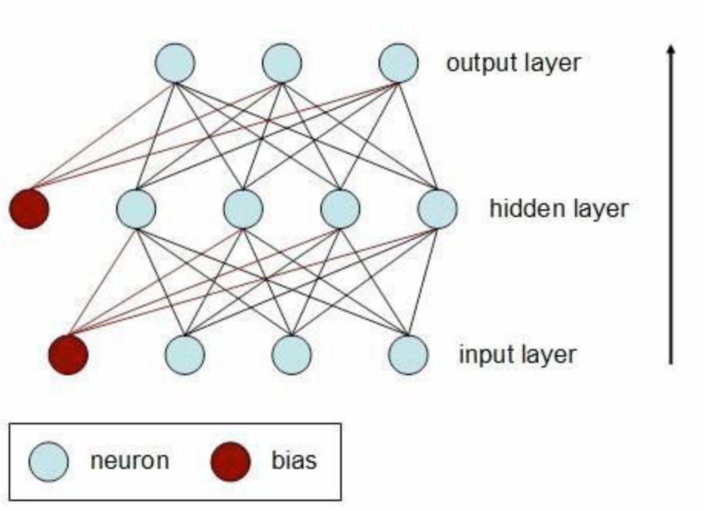
Sigmoid

Multi Class Classification using MLP

- Input: (x_i, y_i)

  $x = [x_1, x_2, x_3, ..., x_d]$
- Encode label y as
  - [1,0,0] for class 1
  - [0,1,0] for class 2
  - [0,0,1] for class 3





Multi Class Classification using MLP

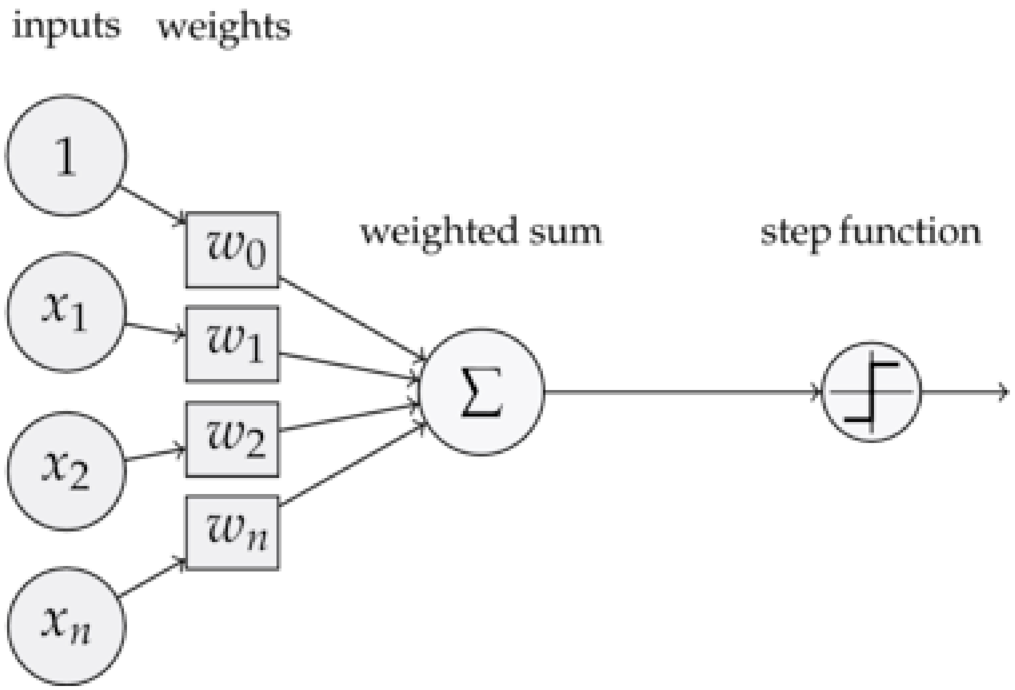
- Loss
  - MSE (Mean square error)
  - Let predicted label be z .
  - Remains the same even for regression.
- Our objective:
  - Minimize the difference between z_i and y_i for all i

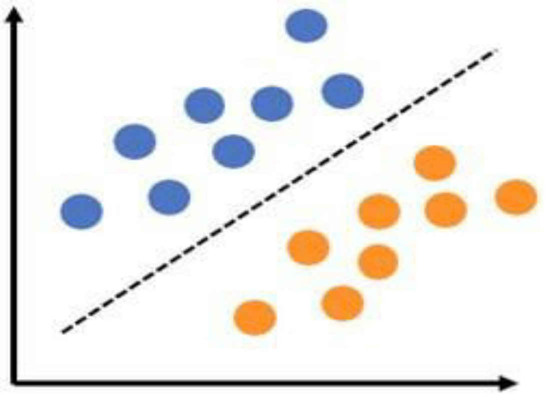
$$L(W) = \sum_i \|z_i - y_i\| = \sum_i \sum_j (z_{ij} - y_{ij})^2$$

Questions?

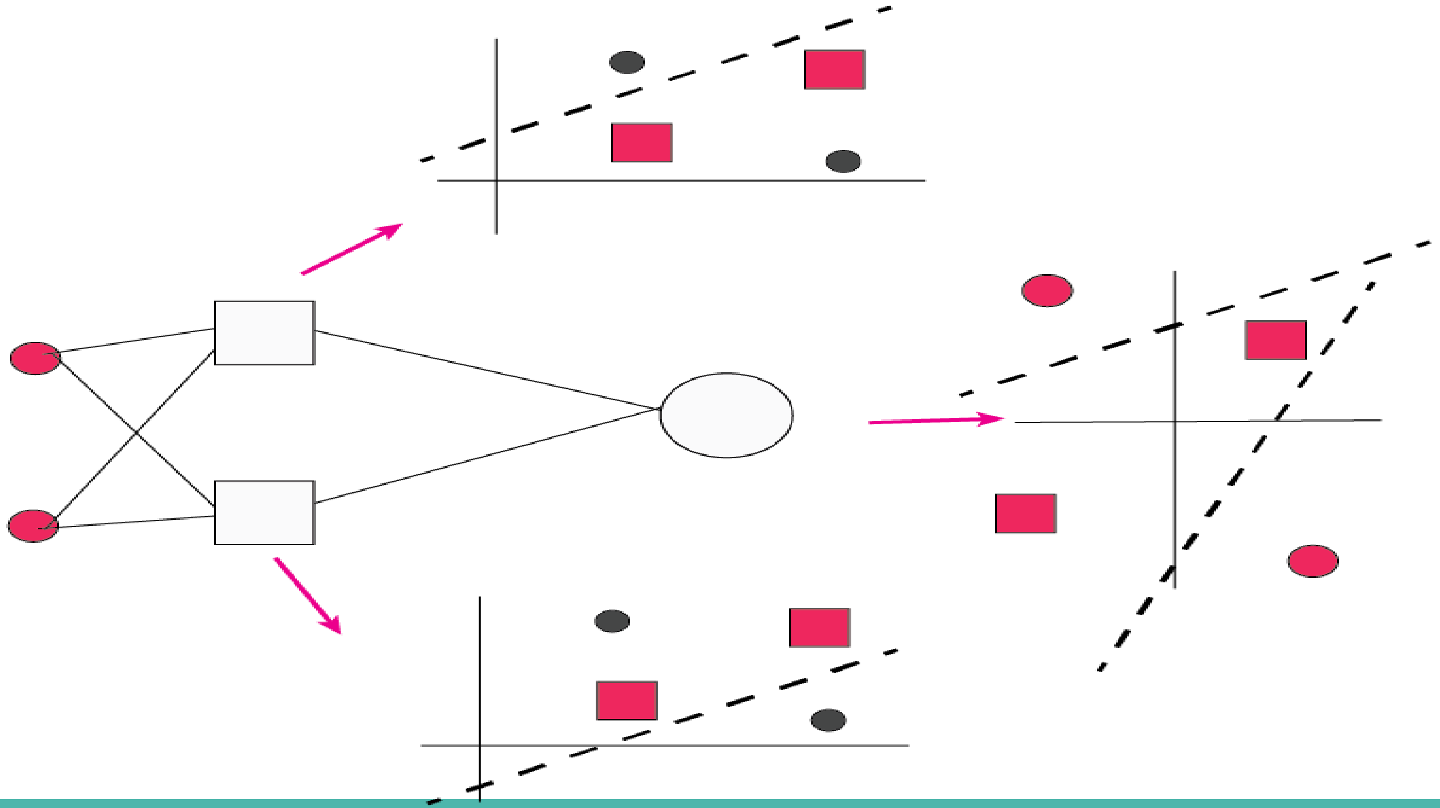
Intuitive Explanation

Decision Boundaries and Perceptrons

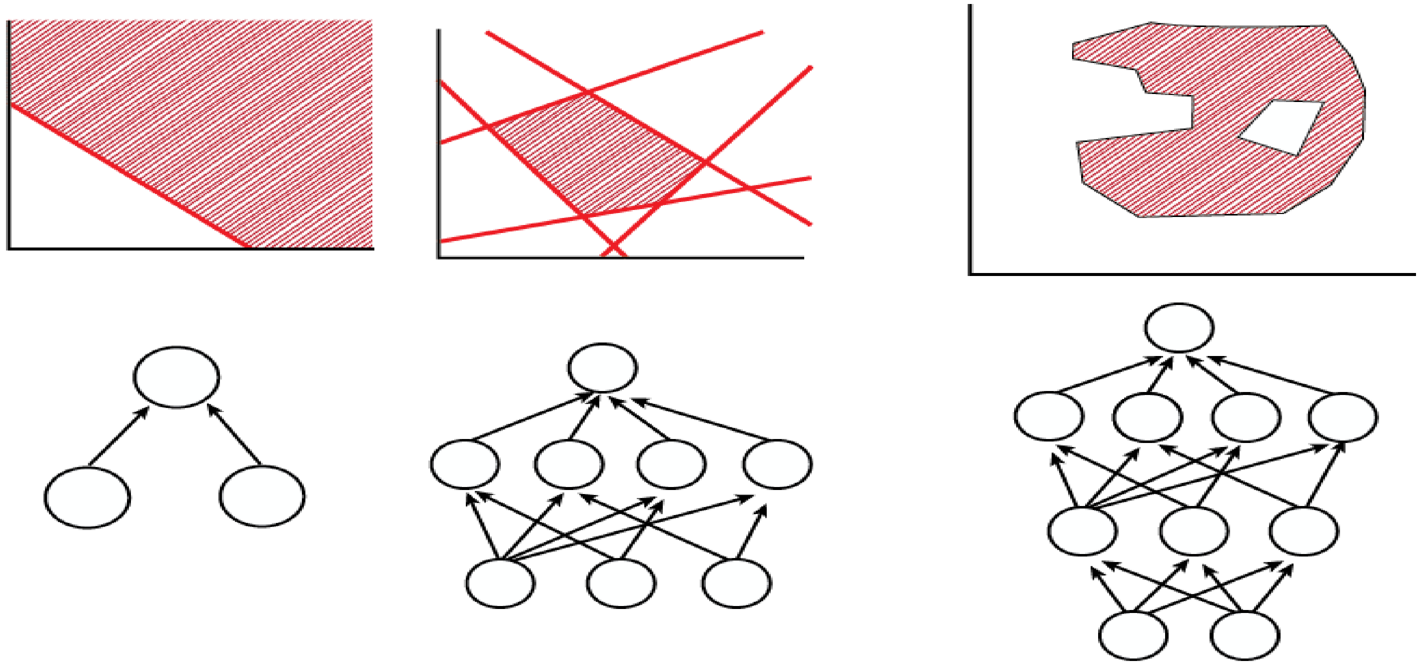




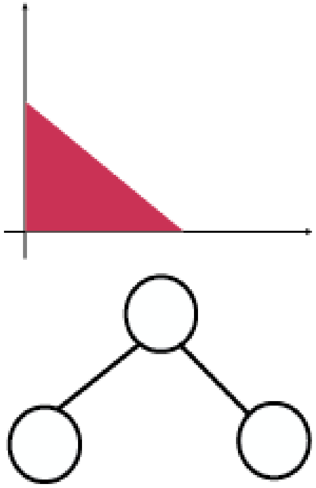
How MLP Works? (A naïve view)



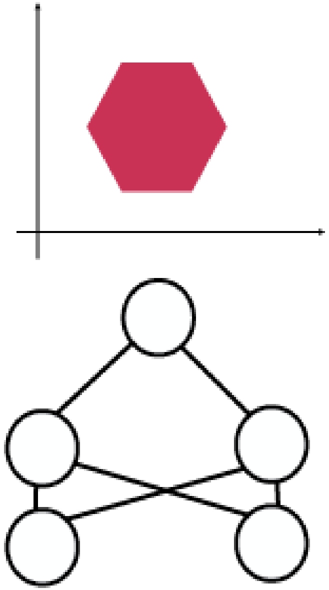
Deeper Networks



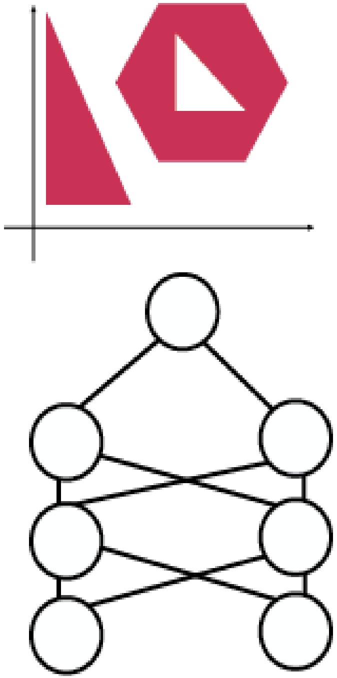
What do layers do?



1st layer draws linear boundaries



2nd layer combines the boundaries



3rd layer can generate arbitrarily complex boundaries

Summary

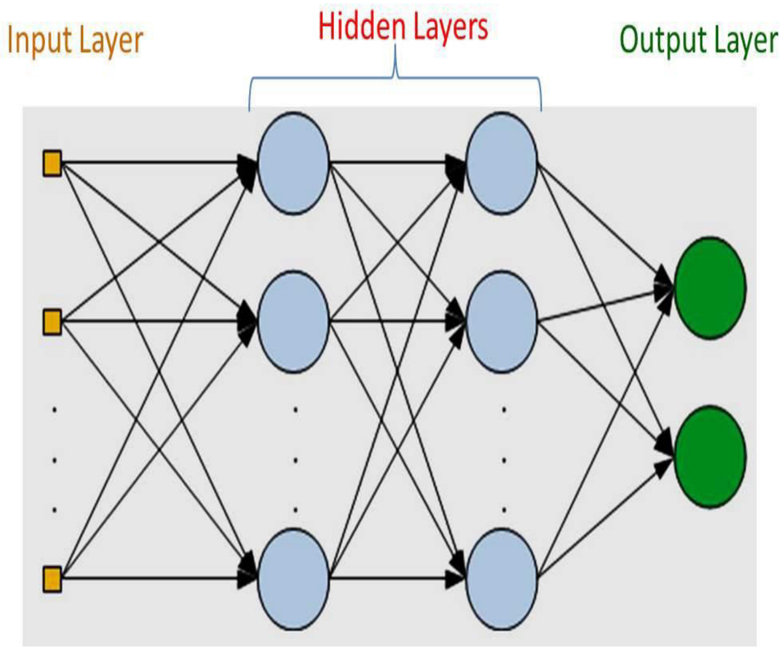
- Many “perceptron” networks can be stacked to generate Multi Layer Perceptron (MLP).
- Any arbitrary function can be approximated.
  - Given that we can train!! (this could be tricky)
- Classically the nonlinearity is a simple sigmoid or similar fns.
- Often people use MLPs’ with one or two hidden layers.
  - Not very deep.

Multi Layer Perceptron

- Two computational blocks/steps

$$y = W^T x$$

$$z = \phi(\alpha) = \frac{1}{1 + e^{-x}}$$

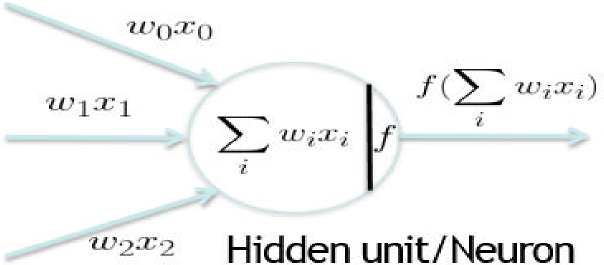


Questions?

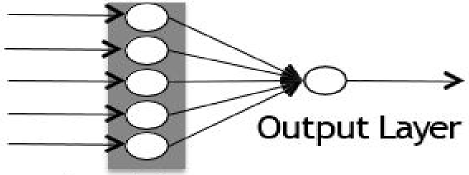
Back Propagation

Training MLP

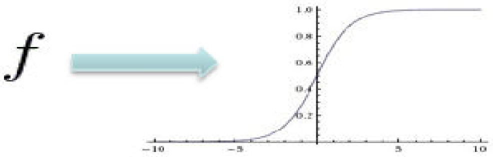
Neuron, Perceptron and MLP



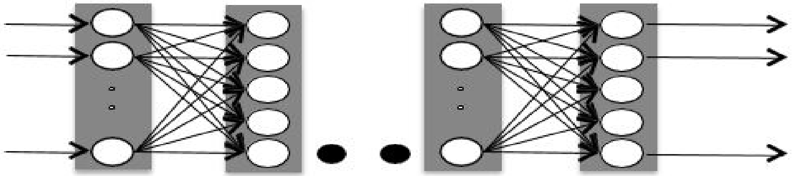
Hidden unit/Neuron



Perceptron

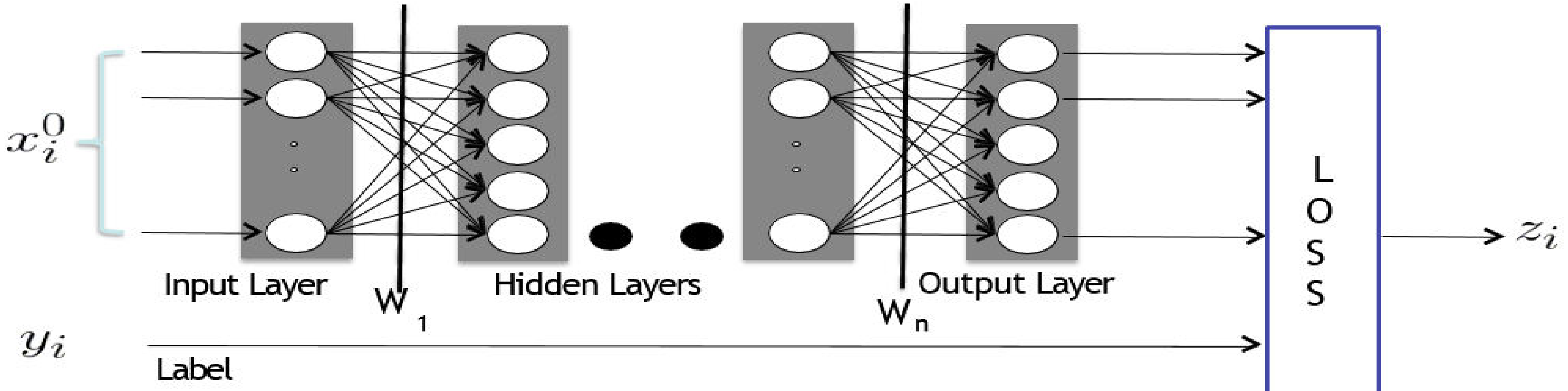


E.g. Sigmoid Activation Function



Multi Layer Perceptron

Loss or Objective

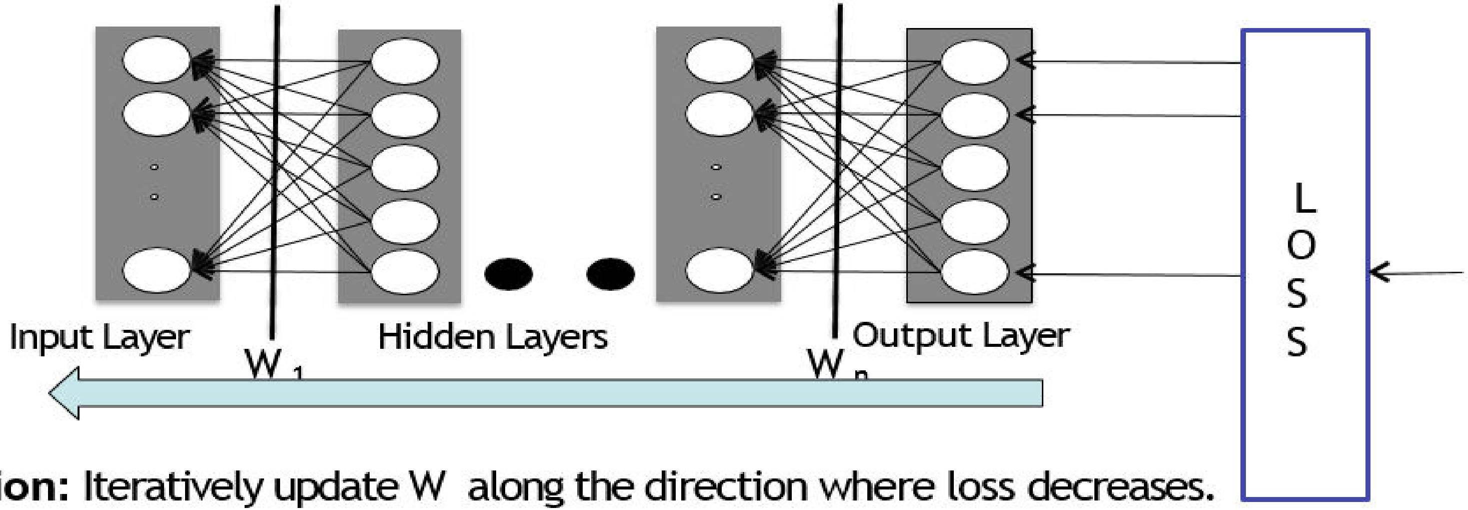


Objective: Find out the best parameters which will minimizes the loss.

$$W^* = \arg\min_{W} \sum_{i=1}^{N} L(x_i^n, y_i; W)$$ Weight Vector

$$z_i = \frac{1}{2} \| x_i^n - y_i \|_2^2$$ E.g. Squared Loss

Backpropagation

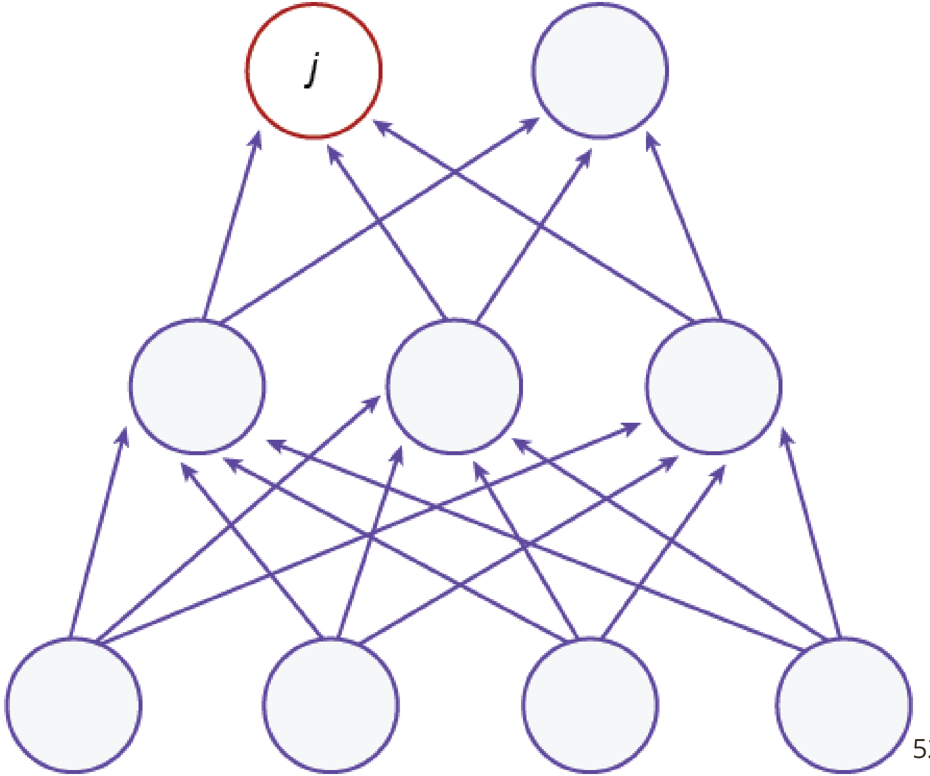


Solution: Iteratively update W along the direction where loss decreases.

Each layer weights are updated based on the derivative of its output w.r.t. input and weights

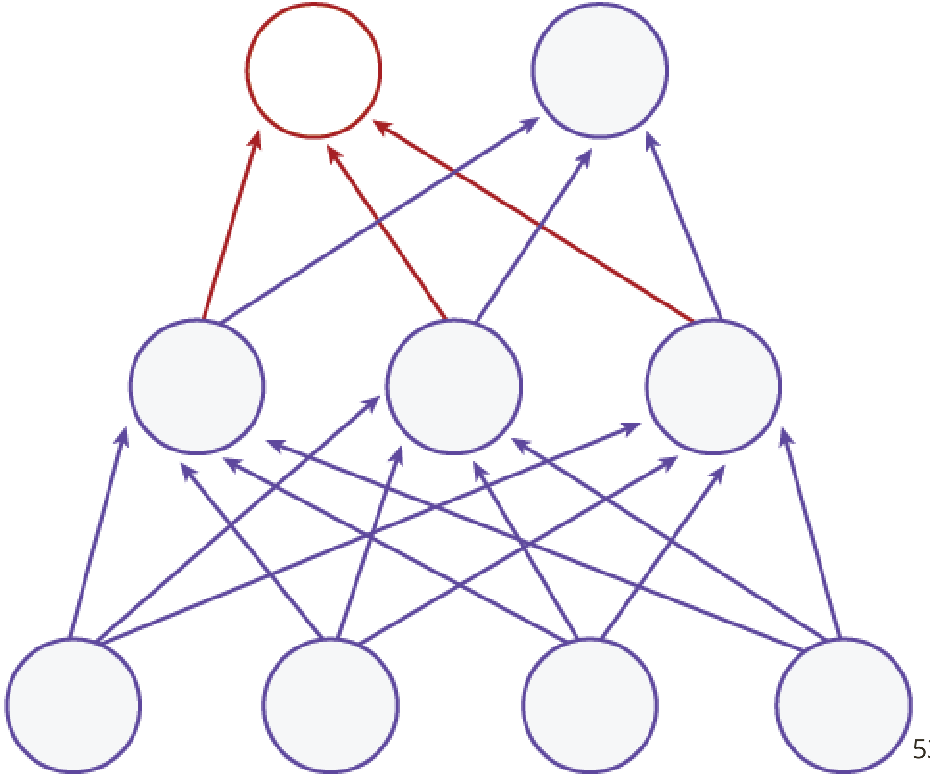
Error Backpropagation

- Calculate error



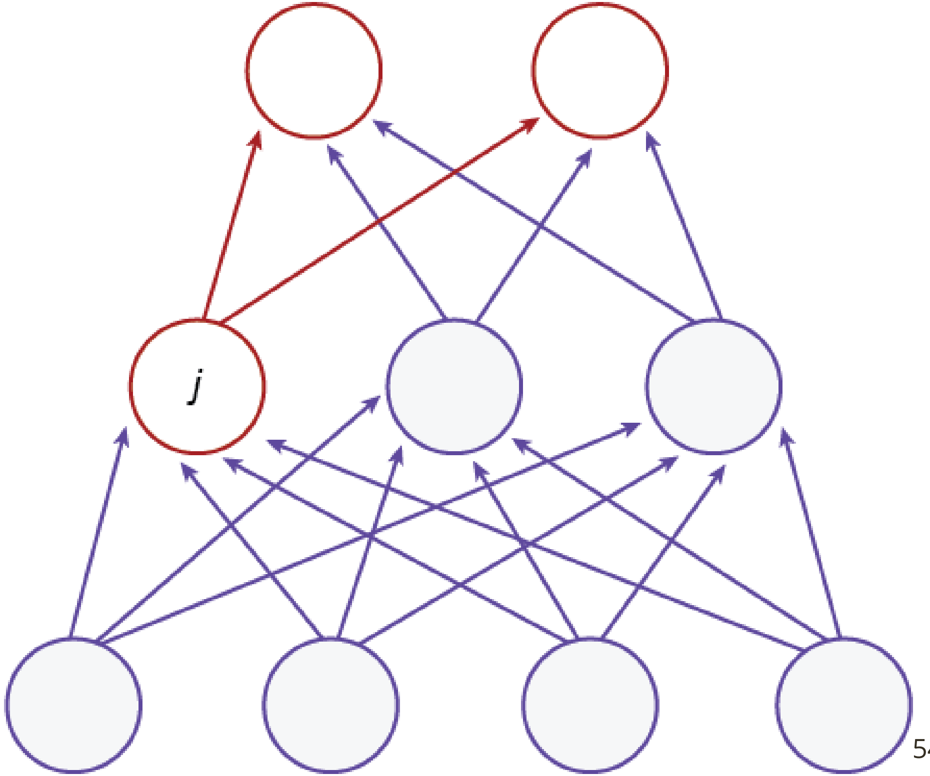
Error Backpropagation

- Determine updates for weights going to outputs.



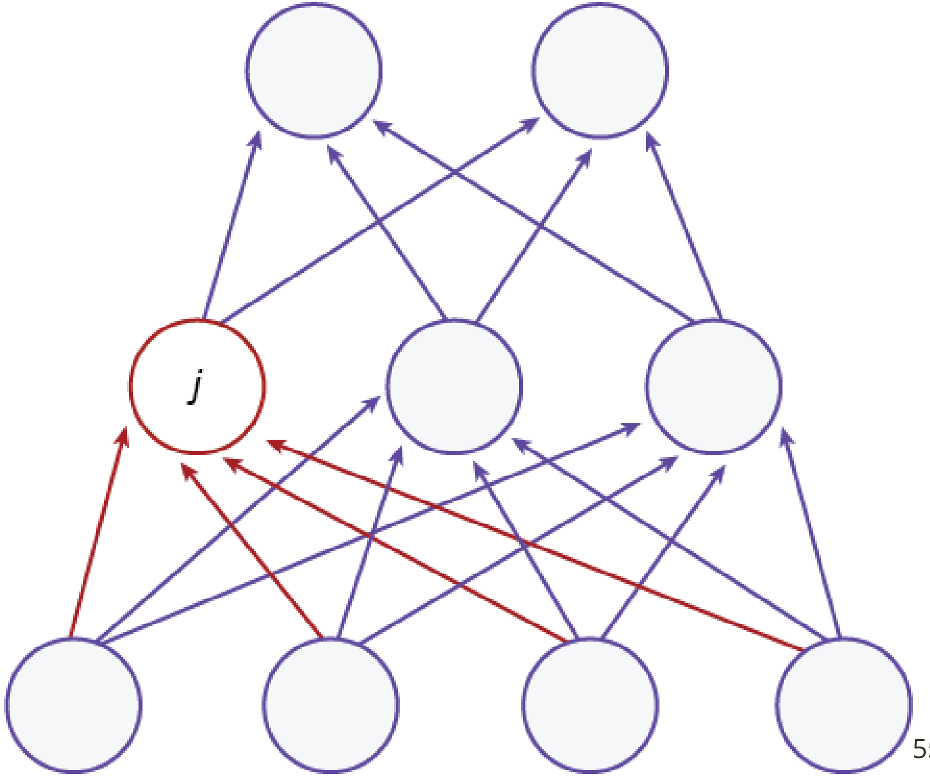
Error Backpropagation

- Calculate error for hidden units



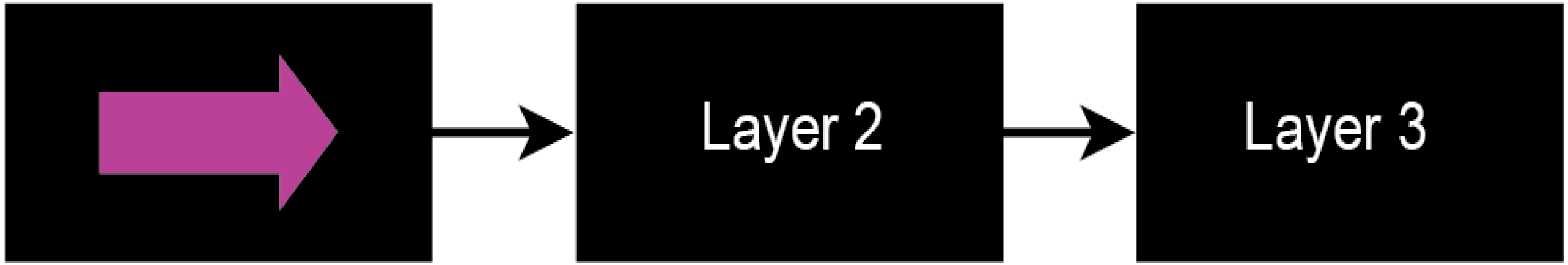
Error Backpropagation

- Determine updates for weights to hidden units using hidden-unit errors.



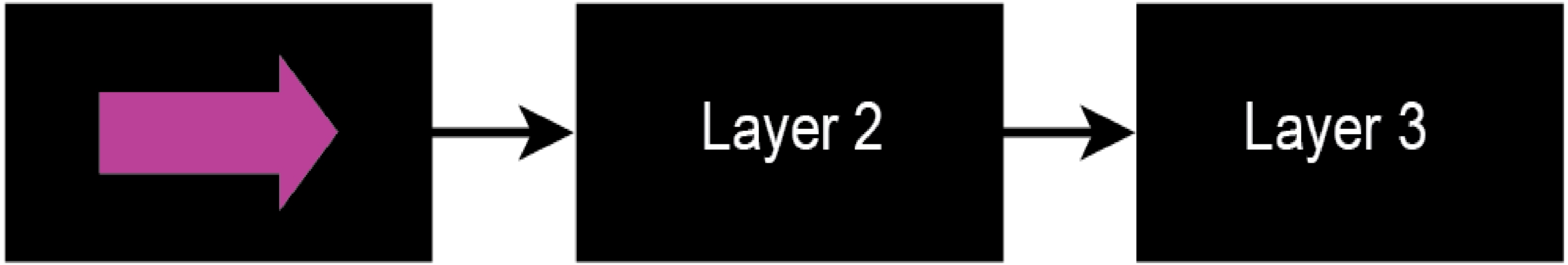
Neural Network Training

- Step 1: Compute loss on mini-batch [F-Pass]



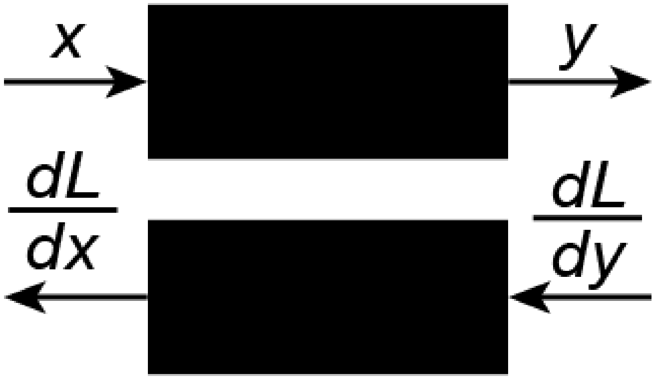
Neural Network Training

- Step 1: Compute loss on mini-batch [F-Pass]
- Step 2: Compute gradients w.r.t parameters[B-Pass]



$$W \leftarrow W - \eta \frac{dL}{dW}$$

Chain Rule



Given y(x) and dLIdy,

What is dLIdx ?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

Chain Rule

Given y(x) and dL/dy,

What is dL/dx ?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

For each block/parameters, we only need to find dy/dx

Summary

Step 0:

Initialize the Network (MLP), weights

Step 1:

- Do forward pass for a batch of randomly selected samples.
- Predict outputs with the existing weights.

Summary

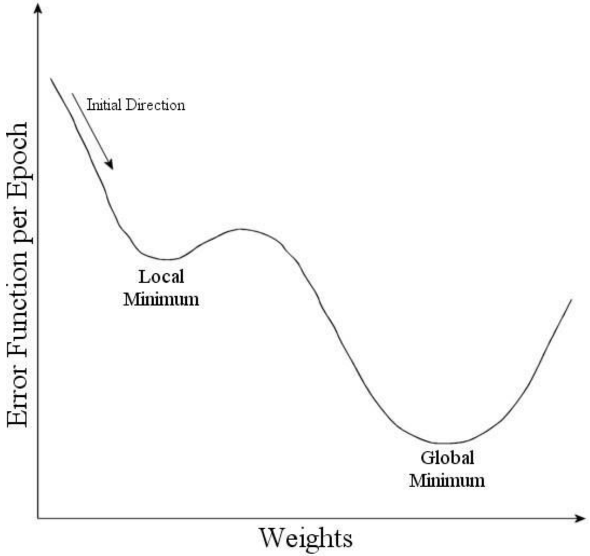
- Step 2:
  - Compute Loss for the set of samples.
- Step 3:
  - Update all the weights using gradient descent.

$$W \leftarrow W - \eta \frac{dL}{dW}$$

Practical Tricks in BP

- Control learning rate
  - Usually small learning rate.
  - Vary learning rate over time.
- Remember the past and adjust the speed
  - Eg. Momentum (borrow ideas from physics!!)
- Good initialization
  - Not random. Not zero. Not equal.

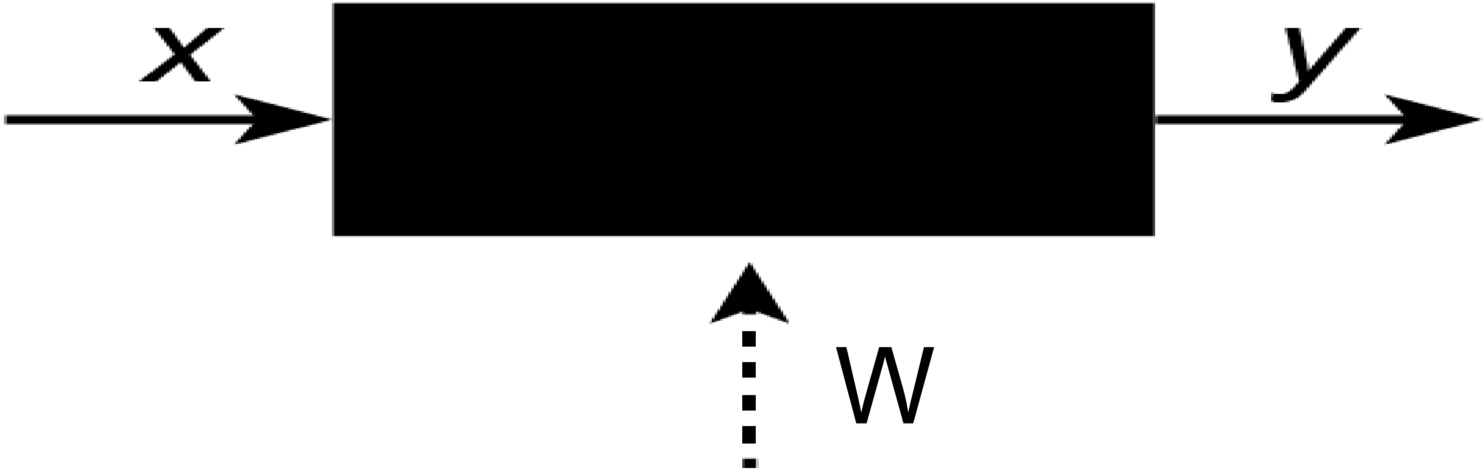
Momentum



- Have two terms.
- One from gradients
- One from the previous

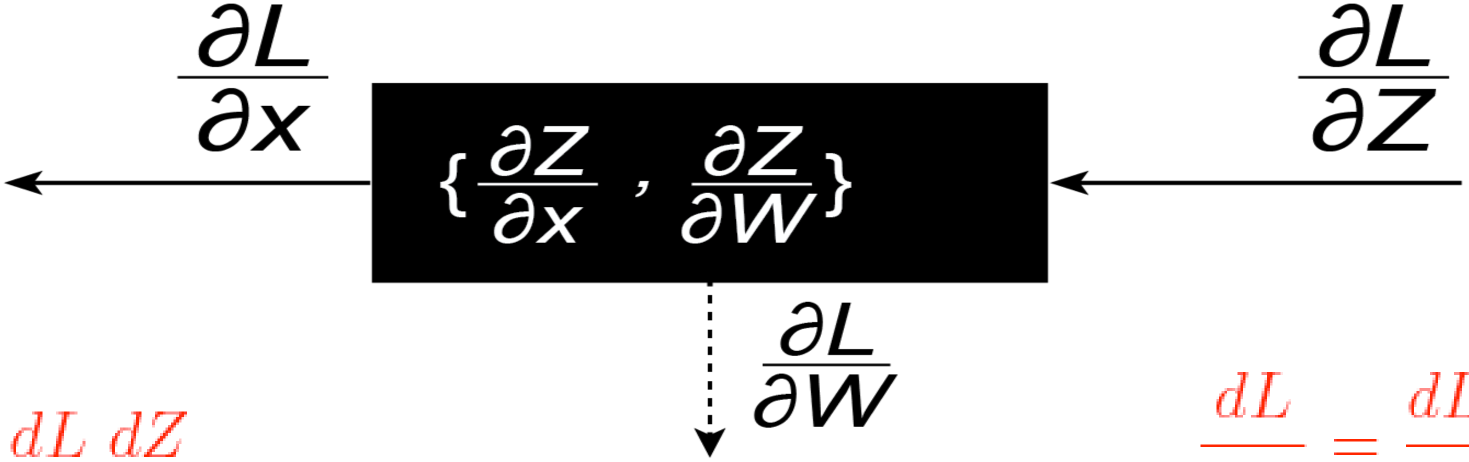
Questions?

Key Computation:Forward-Propagation



W is the parameters, say the weights within the box.

Key Computation:Backward-Propagation



$$\frac{dL}{dX} = \frac{dL}{dZ}\frac{dZ}{dX}$$

$$\frac{dL}{dW} = \frac{dL}{dZ}\frac{dZ}{dW}$$

$$W \leftarrow W - \eta\frac{dL}{dW}$$

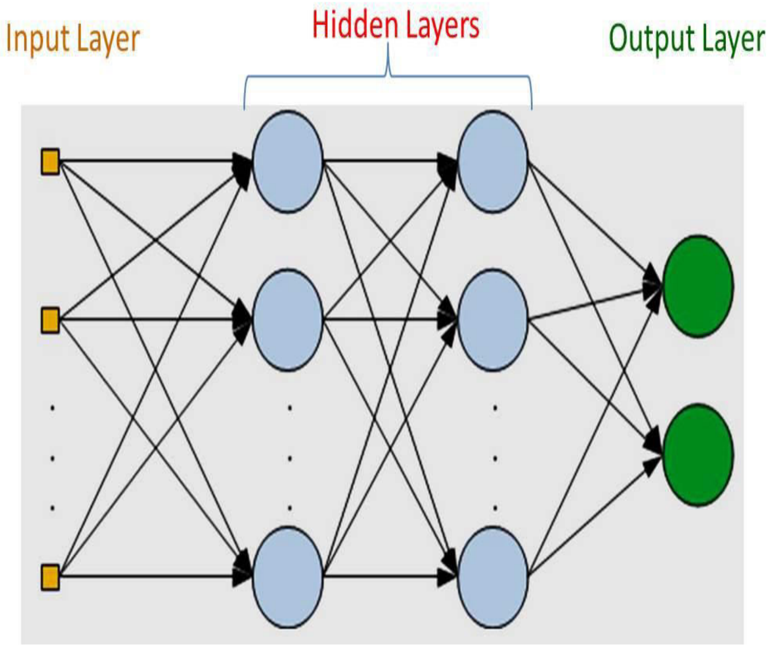
Back Propagation for MLP

Two Computational Blocks/Steps

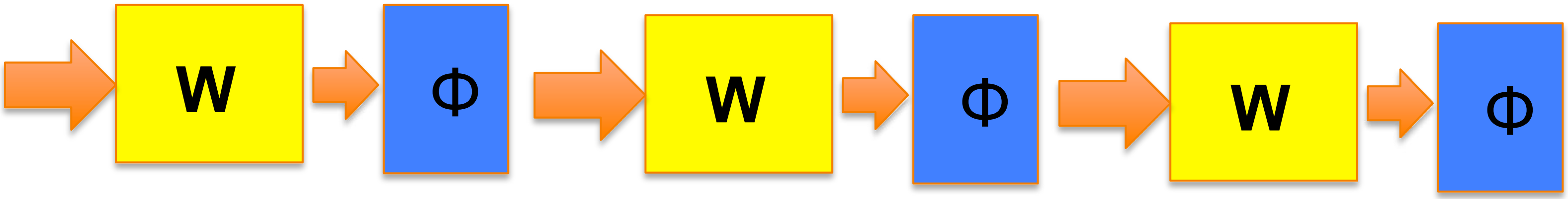
$$y = W^T x$$

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

In either case we can compute $\frac{dy}{dx}$ easily.



A simpler view point



L
o
s
s

Blocks with Learnable parameters Matrix Multiplication

Nonlinear functions (often non learnable)

Backpropagation

$$\frac{dL}{dx} = \frac{dL}{dy} . \frac{dy}{dx} (1)$$

$$\frac{dL}{dW} = \frac{dL}{dy} . \frac{dy}{dW} (2)$$

$$W^{n+1} = W^{n} - \eta \frac{dL}{dW} (3)$$

Backpropagation

Let there be N stages. For a computational block *l*,

1. Compute dL/dx using equation 1.
2. If the block as a learnable parameter W, then
   1. Compute dl/dW using equation 2.
   2. Update the parameters using equation 3.
3. Set the dL/ dx of stage l as dL/dy of stage l – 1, and repeat the steps 1-3, until we reach first block

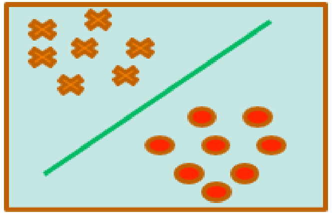
Questions?

Kernels

Nonlinearity in Linear Methods

“Linear” Learning techniques

- Linear classifier



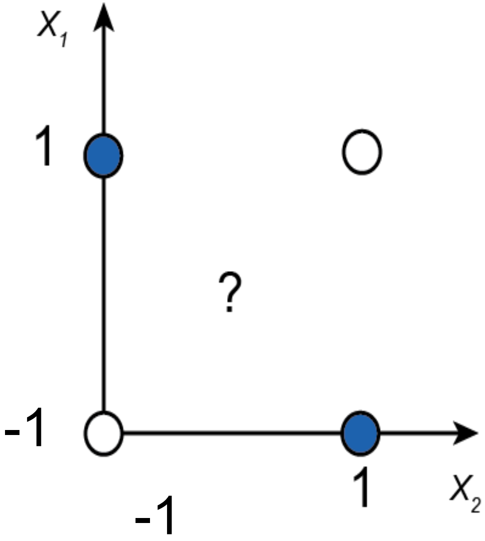
$$g(x_n) = sign(w^T x_n)$$

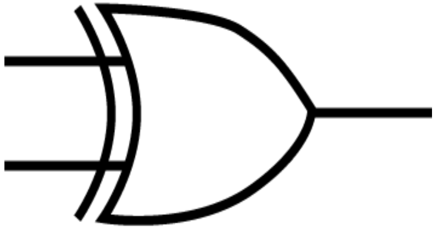
where *w* is an *d*-dim vector (learned)

- Techniques:
    - Perceptron
    - Logistic regression
    - Support vector machine (SVM)
    - Etc.

XOR: Limitation of Linear Methods

x_1	x_2	$x_1 \, XOR \, x_2$
-1	-1	-1 (-)
-1	1	1 (+)
1	-1	1(+)
1	1	-1(-)



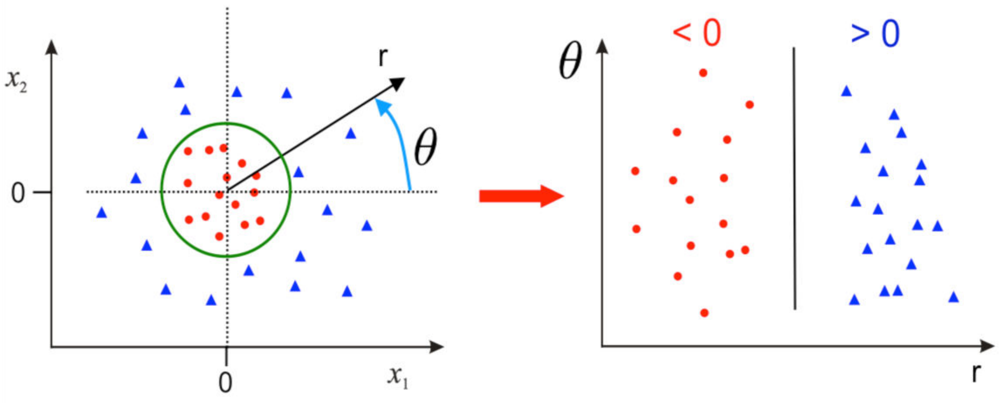


Consider a new feature $x_3 = x_1x_2$

X3	XOR
1	-
-1	+
-1	+
1	-

- With a “new feature”, a problem that is linearly non separable has become separable!!
- A difficult problem became easy!!

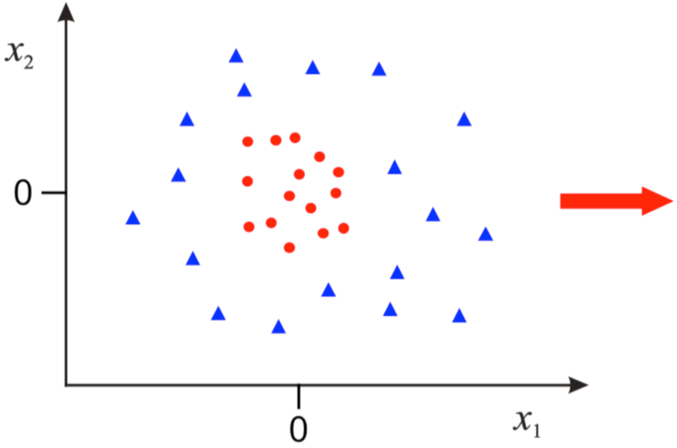
Nonlinearity with Feature Maps



$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} r \\ \theta \end{pmatrix}$$

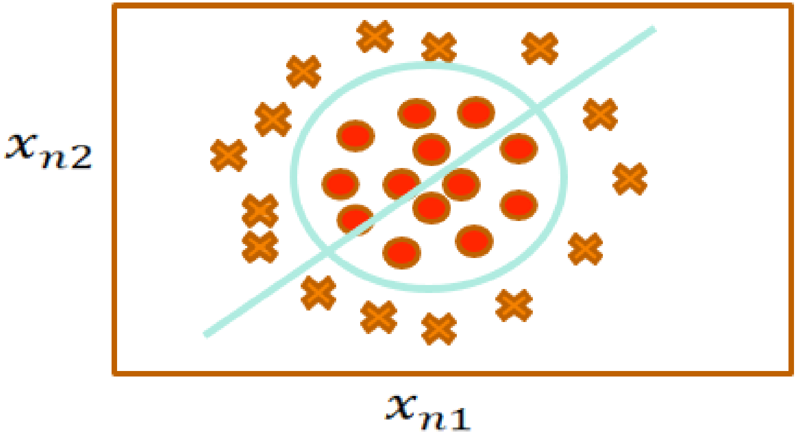
With a "smart" feature map, a linearly non-separable problem can be converted to a separable problem.!!

Q: Transform to (x_1^2, x_2^2)



More General

- Non- linear case



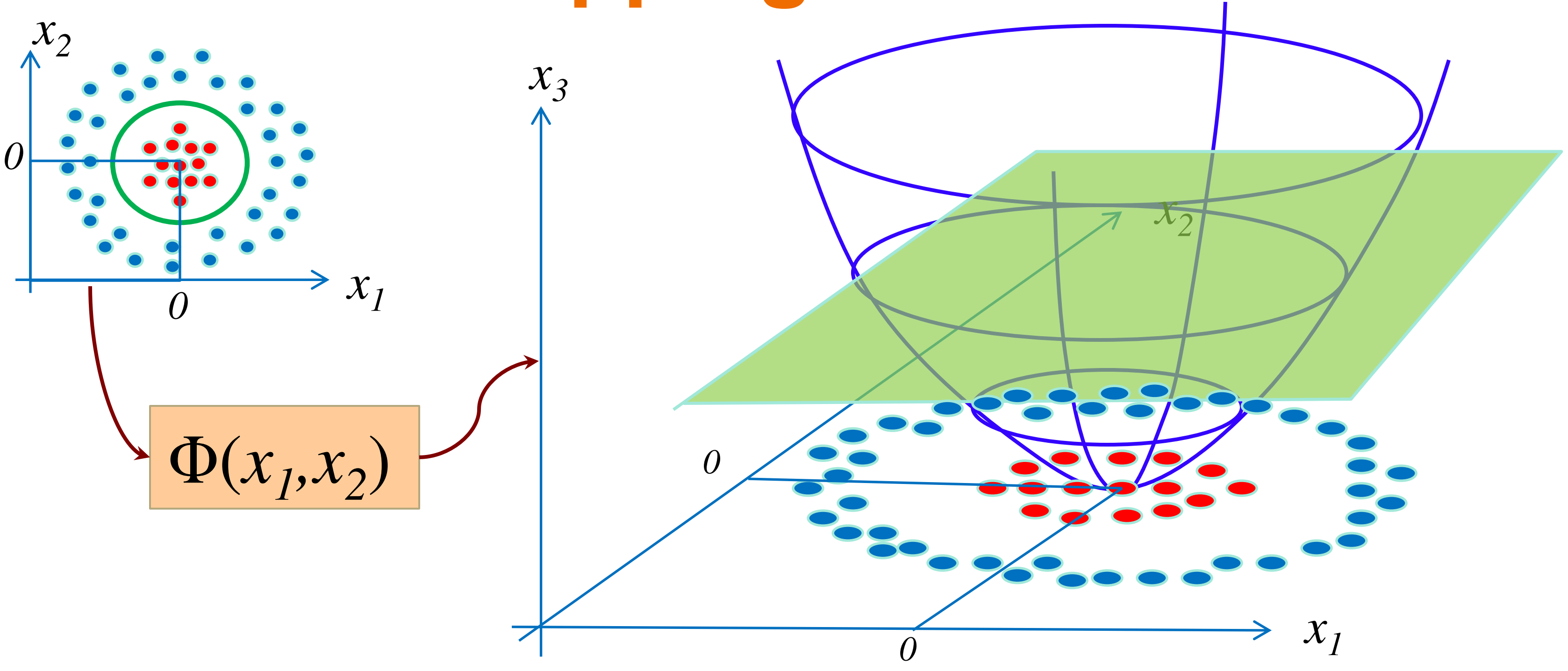
$$x_n = [x_{n1}, x_{n2}]$$

$$x_n = [x_{n1}, x_{n2}, x_{n1} * x_{n2}, {x_{n1}}^2, {x_{n2}}^2]$$

$$g(x_n) = sign(w^T x_n)$$

Non-linear Mapping

$$x_3 = x_1^2 + x_2^2$$



Φ is a non-linear mapping into a possibly high-dimensional space

Summary

- If features can be transformed appropriately, simple linear algorithms (classification, regression) is enough.
- How do we find the feature transformation?
  - Make a reasonable guess?
  - Ans: Use some popular complex function.
- Why linear?
  - Nice algorithms.
  - Speed of evaluation!!

SVMs and Kernels

SVM: Formulation

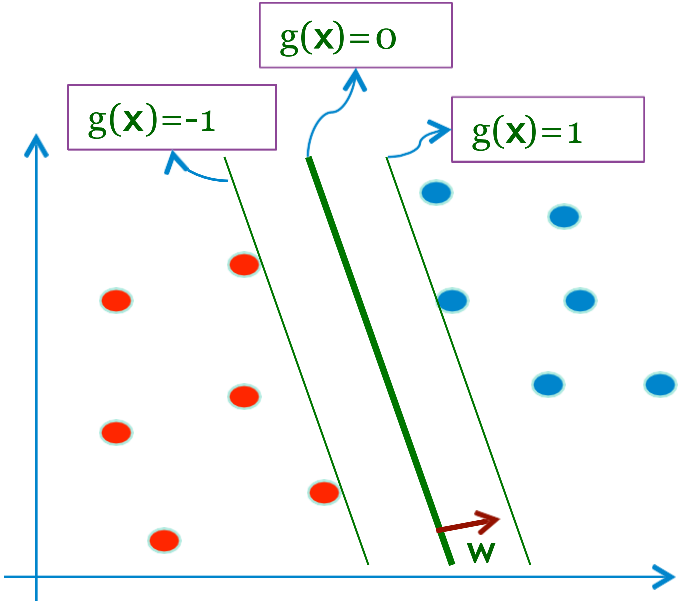
- Let $g(x)=\mathbf{w}^T\mathbf{x}+b$.
- We want to maximize margin such that:
  - $\mathbf{w}^T\mathbf{x}_i+b \geq 1$ for $y_i=1$
  - $\mathbf{w}^T\mathbf{x}_i+b \leq -1$ for $y_i=-1$
  - $y_i(\mathbf{w}^T\mathbf{x}_i+b) \geq 1$ for all i

Eg. Distance from $(0,0)$ to $a \cdot x + b \cdot y + c = 0$ is $\frac{c}{\sqrt{a^2+b^2}}$.

Similarly distance from origin to $w^Tx + b = 1$ is $\frac{b-1}{||w||}$. And to $w^Tx + b = -1$ is $\frac{b+1}{||w||}$. The distance between the the two lines/planes (margin) is

$$\frac{b+1}{||w||} - \frac{b-1}{||w||} = \frac{2}{||w||}$$

The objective of maximize the margin is same as minimize $\frac{1}{2}||w||$, such that all positive samples and negative samples are beyond the respective half planes.



SVM: Primal and Dual

$$min\ \frac{1}{2}W^TW$$

subject to $y_i(W^Tx_i + b) - 1 \geq 0 \forall i$

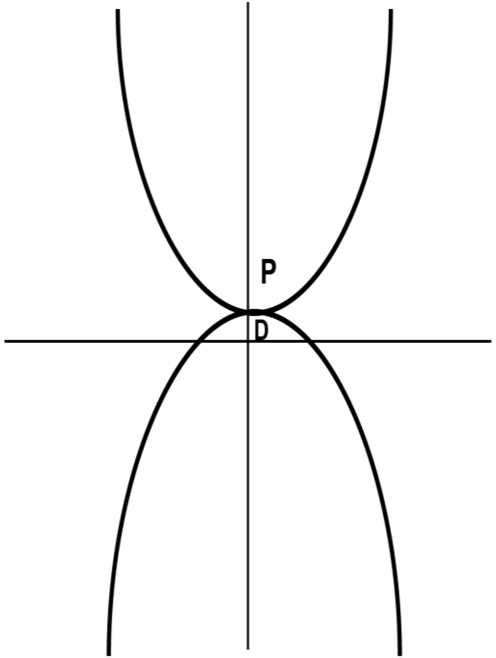
This results in $y_i \epsilon \{1, -1\}$

maximization of

$$J_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\alpha_i \alpha j y_i y_j x_i^T x_j$$

$$W = \sum_{i=1}^{N} \alpha_i y_i x_i$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$



Kernel Strategy

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

What we need is only $w^T x = \sum_{i=1}^{N} \alpha_i y_i x_i^T x$

We can do the same in a new feature space:

$$w^T x = \sum_{i=1}^{N} \alpha_i y_i \phi(x_i)^T \phi(x) \qquad w^T x = \sum_{i=1}^{N} \alpha_i y_i K(x_i, x)$$

Kernels

- Interestingly, it is possible to train and test SVMs this without explicitly doing the non-linear mapping to high dimensions $K(x_i, x_j)$
- We need only a kernel function

$$K(s_i, x_i) = \phi(s_i).\phi(x_i)$$

Popular Kernels

- Polynomial:

$$\mathrm{K}_p(\mathbf{X},\mathbf{Y}) = (1 + \mathbf{X} \bullet \mathbf{Y})^p$$

- Radial Basis Function (RBF) or Gaussian:

$$\mathrm{K}_r(\mathbf{X},\mathbf{Y}) = e^{-\frac{1}{2\sigma^2}\|\mathbf{X}-\mathbf{Y}\|_2^2}$$

- Hyperbolic Tangent:

$$\mathrm{K}_s(\mathbf{X},\mathbf{Y}) = \tanh(\beta_0 \mathbf{X} \bullet \mathbf{Y} + \beta_1)$$

Summary

- Linear SVMs generalize well, but cannot separate non- linear data
- Kernels (nonlinear) SVMs are also good at generalization, and can deal with non-linear data.
- Need not be as efficient/compact.

Questions?

Predicting in Time

Time Series

- Time series is a sequence of observations often ordered in time.
- Problem: Given a sequence, predict future samples.
- Applications:
    - Meteorology, Finance, Marketing etc.
- Notation: x[0], x[1], x[2], ..., x[N].
- X[t]. Where t is the time or index in the sequence

A simple Model

- X[t] = a1 X[t-1] + a2 X[t-2] + a3 X[t-3] + n
  - Where n is noise.
- Problem:
  - Given the sequence X[0], X[1], …… X[N]
  - Find coefficients a1, a2, a3
- Find the coefficients a1,a2,a3 such that prediction error is minimal

Model the Problem

- Our familiar (x,y) may be seen as:
  - (<x[0],x[1],x[2]>, x[3])
  - (<x[1],x[2],x[3]>, x[4])
- Problem is now cast in a 3D space
  - Regression.

$$\underset{a1,a2,a3}{\text{Minimize}} \sum_{t}^{N} ((x[t]-a1)(x[t-1]-a2)(x[t-2]-a3)(x[t-3]))^2$$

More Powerful Model

- X[t] = f(W, X[t-1], X[t-2], X[t-3]) + n
- Problem:
  - Given the sequence X[0], X[1], ... X[N]
  - Find coefficients W
- Data may be modeled as in the above linear case.
- f() may be seen as a MLP

$$\text{Minimize}_W \sum_{t=3}^{N} (x[t] - f(W, x[t-1], x[t-2], x[t-3]))^2$$

Summary

- Predicting future samples is a new problem
- However, solution is similar to what we know.
  - Cast as regression.
- Model can be linear
  - Linear regression
- Or nonlinear
  - MLP
- On how many past samples, the future sample will depend?
  - Order/model to be guessed?

Review: Pipeline/ Concept Map

DATA

Structured (numerical, categorical attributes)
Digital Logs (Tweets, SMS)
RawData/ Sensors (Image/Speech)
User behaviors
Etc.

FEATURE

Intuitive User defined
Raw data itself

Statistics (Histograms, PCA)

Signal Process (Fourier Xform)

FEATURE XFORMATIONS

Feature Selection

Feature Extraction

Dimensionality Reduction

Data Visualization
Eg. PCA, tSNE

ML PROBLEM

1. Classification
   a. Binary
   b. Multiclass
2. Regression
3. Clustering
4. Prediction (time series)

Thanks!

Questions?

Multi Class Classification using MLP

- Input: (x_i, y_i)

  $x = [x_1, x_2, x_3, ..., x_d]$
- Encode label y as
  - [1,0,0] for class 1
  - [0,1,0] for class 2
  - [0,0,1] for class 3

Multi Class Classification using MLP

- Loss
  - MSE (Mean square error)
  - Let predicted label be .
  - Remains the same even for regression.
- Our objective:
  - Minimize the difference between z_i and y_i for i each

$$L(W) = \sum_i \|z_i - y_i\| = \sum_i \sum_j (z_{ij} - y_{ij})^2$$

Kernels

Similarity Function

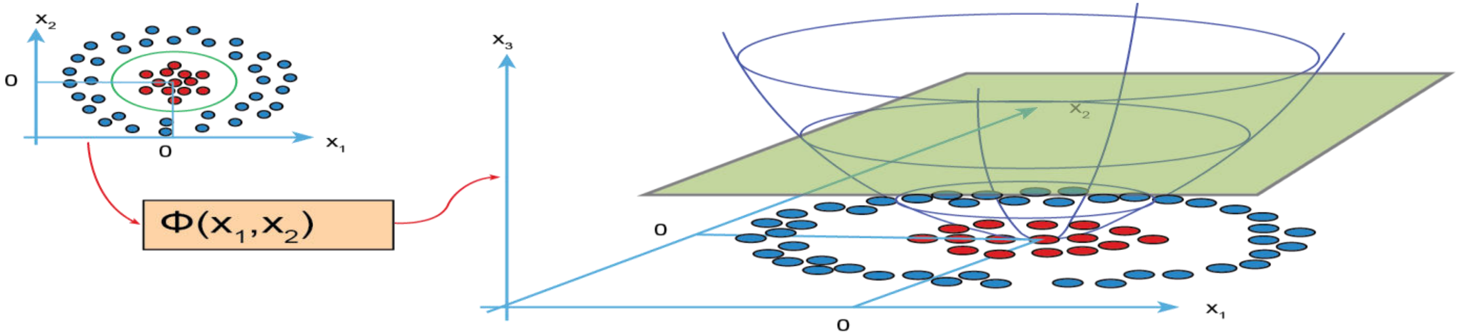
Kernels

Interestingly, the vectors appear as only dot product in the formulation. This allows us to solve the problem in a very high dimension (where the data set will well behave) without explicitly bothering about the mapping which converts into higher dimension.

We need only a kernel function $K(x_i, x_j)$

$$K(s_i, x_i) = \phi(s_i).\phi(x_i)$$

Kernels



Φ is a non-linear mapping into a possibly high-dimensional space

A Simple Quadratic Kernel

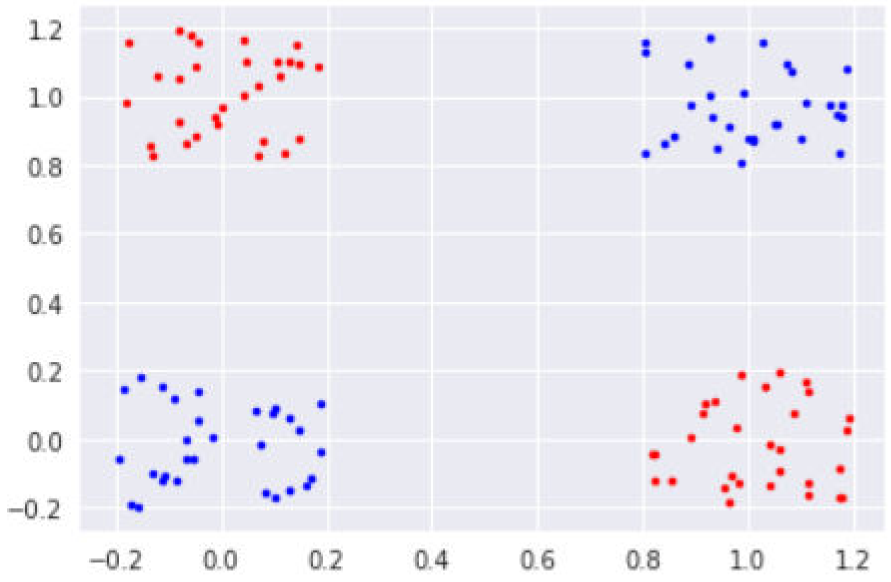
$$Let\ \Phi(X) = \left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_2x_1 \\ x_2x_2 \end{bmatrix}$$

We can compute K(X, Y)=(X.Y) instead of mapping with ø explicitly and then computing dot product.

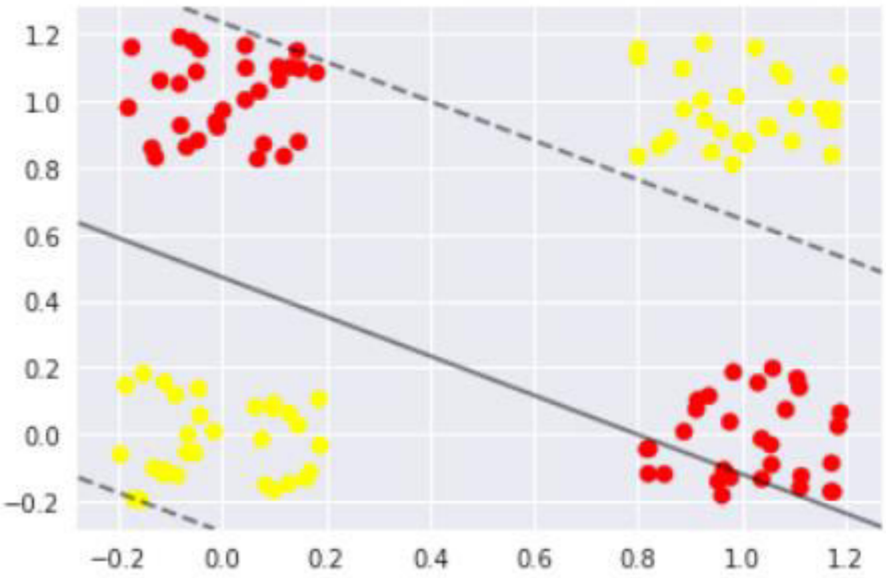
$$Let\ K(X,Y) = \Phi(X).\Phi(Y) = \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2x_1 \\ x_2^2 \end{bmatrix} . \begin{bmatrix} y_1^2 \\ y_1y_2 \\ y_2y_1 \\ y_2^2 \end{bmatrix}$$

$$= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 = (x_1y_1 + x_2y_2)^2 = (X.Y)^2$$

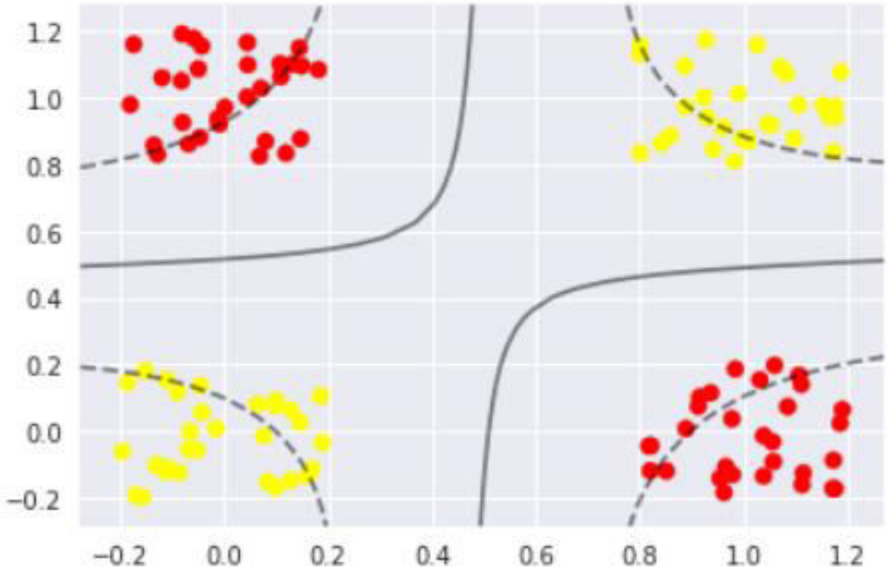
XOR Data



Classification with linear kernel



Classification with non-linear kernel



Notes

- Linear SVMs are very good. Fast.
- Kernel SVMs (nonlinear SVMs) were the best until the popularity of Deep Learning.
- Popular implementations use Quadratic Programming for training. Gradient descent based solvers (pegasos) also exists.
- Limitation:
    - Primarily binary (two class). Multi class extensions exists.

Summary

- SVMs are very good.
- Convex optimization. No worries about local minima.
- Many excellent solvers. (Often we never code ourselves.)
- Linear SVMs are efficient for training and testing (both memory and flops).
- Kernels (nonlinear) SVMs are accurate. Need not be efficient/compact.

Formulation

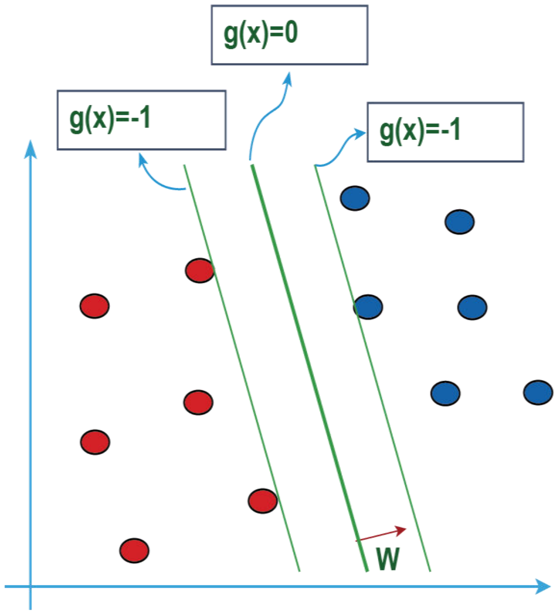
Let $g(X) = W^T X + b$

We want to maximize margin such that:

$W^T X_i + b \geq 1$ For $y_i = 1$

$W^T X_i + b \leq -1$ For $y_i = -1$

$y_i(W^T X_i + b \geq 1)$ For i all



Formulation

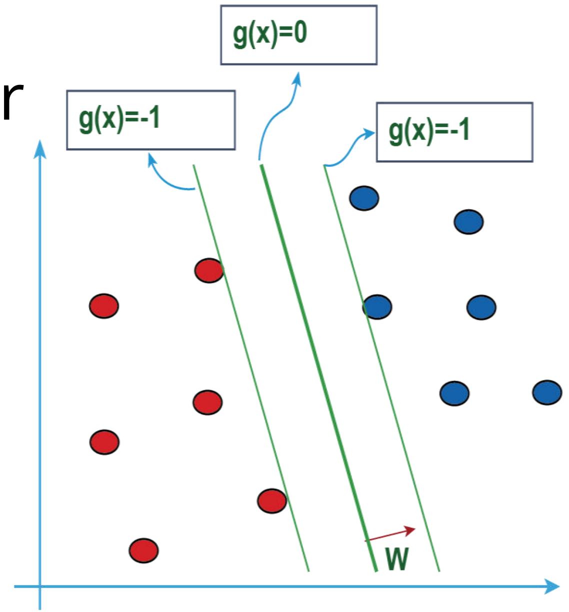
Value of $g(x)$ depends on $||w||$

- Keep $||w||$ = 1 and maximize $g(x)$ or
- Let $g(x) \geq 1$ and minimize $||w||$

We use approach(2) and formulate the problem as

- Minimize $\frac{1}{2} W^T W$
- Subject to $d_i(w^T x_i + b) \geq 1$

  for i = 1...N



Solving the Dual form

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j x_i^T x_j$$

$$Subject\ to\ \alpha_i \geq 0\ \forall_i\ and\ \sum_{i=1}^{N} \alpha_i d_i = 0$$

$\rightarrow$ QP Server $\rightarrow$ α_i $\rightarrow$

$$W_o = \sum_{i=1}^{N} \alpha_i d_i x_i$$

$$\alpha_i \left[d_i (W_o x_i \ + \ b) - 1\right] = 0$$

$$b_o = 1 - W_o^T X_{s+}$$

- The only unknowns(variables) are α_is
- The constraints are also on α_is only
- Data vectors appear only as dot products
- Objective is convex, subject to linear constraints
- Can be solved using standard convex quadratic program solvers

Solving the SVM with a mapping

- Data vectors occur only as dot products in SVM-learning and testing

$$Label = sign(W_o \,.\, \Phi(x_test) + b_o)$$

$$W_o = \sum_{i=1}^{N} \alpha_i d_i \Phi(x_i)$$

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i \; - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j x_i^T x_j K(x_i, x_j)$$

$$Label = sign(\sum_{i=1}^{N} \alpha_i d_i K(x_i, x_{test}) + b_o)$$

Thanks!

Questions?