Focus for this lecture

Task → Classification

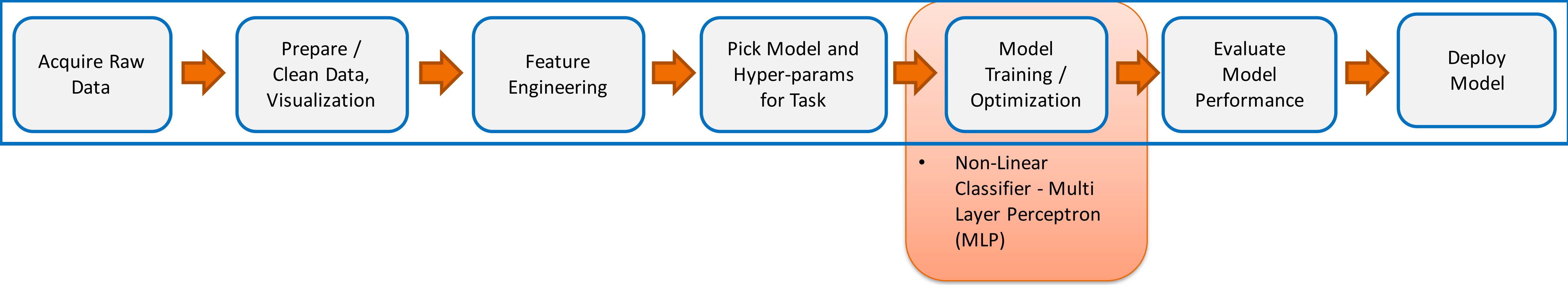
Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

Pick Model and Hyper-params for Task

- Linear and Non-linear Functions
  - Activation Functions
  - Optimization function
  - Loss Function

Model Training / Optimization

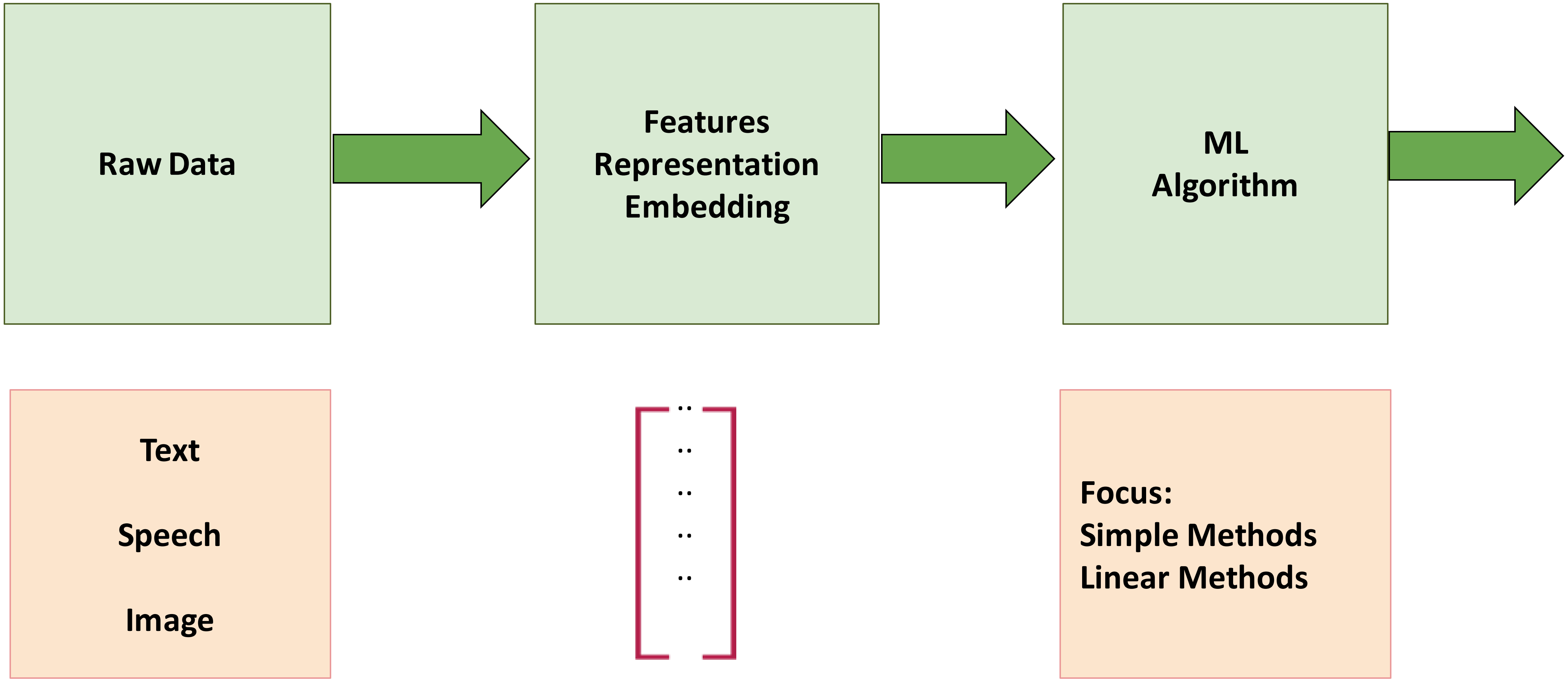
- Linear Classifier in Single Layer Perceptron
- Non-Linear Classifier - Multi Layer Perceptron(MLP)
- Forward Propagation
- Back Propagation



Non-Linear Classification

MLP

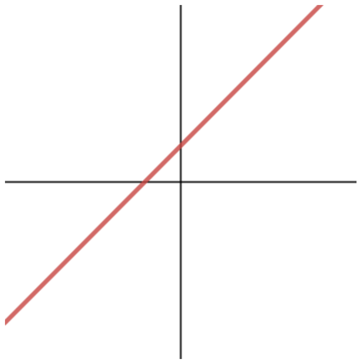
Pipeline


Raw Data
Features
Representation
Embedding
ML
Algorithm
Text
Speech
Image
Focus:
Simple Methods
Linear Methods

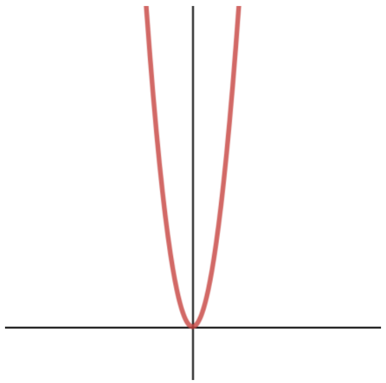
Spectrum of Classifiers

Linear and Non Linear Functions

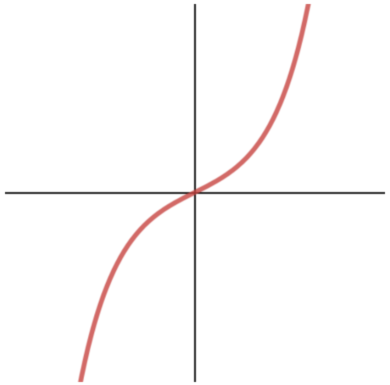
$$W^T X = w_0 + w_1 x_1 = x_1 + 2$$



$$W^T X^2 = w_1 x_1^2 = x_1^2$$

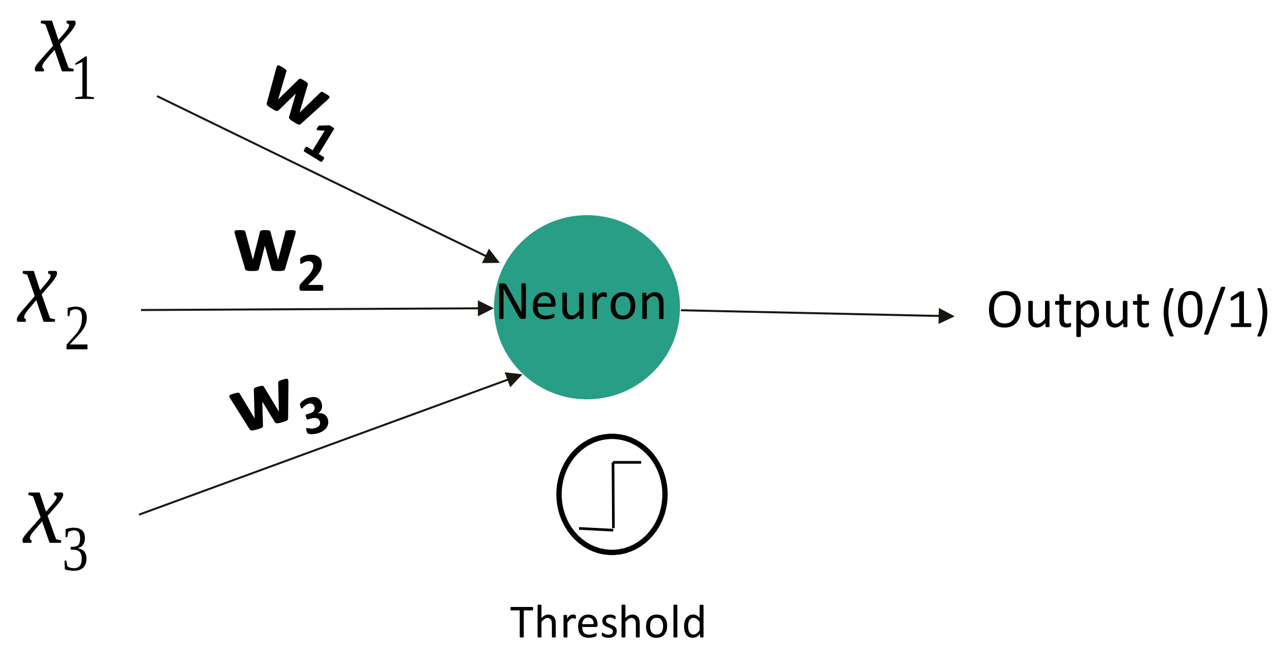


$$\sinh(W^T X) = \sinh(w_1 x_1) = \sinh(0.5 x_1)$$

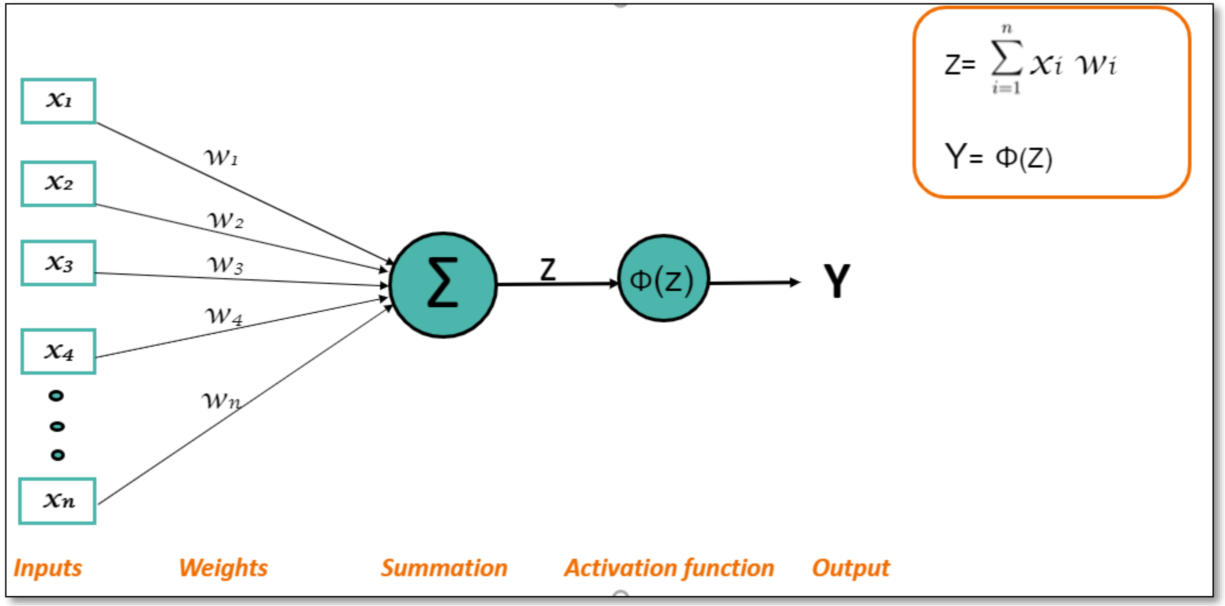


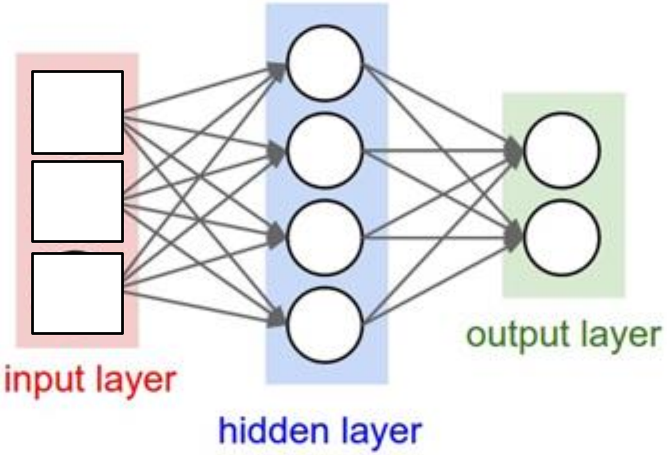
$$e^{-W^T X} = e^{(w_1 x_1)} = e^{(-2 x_1)}$$

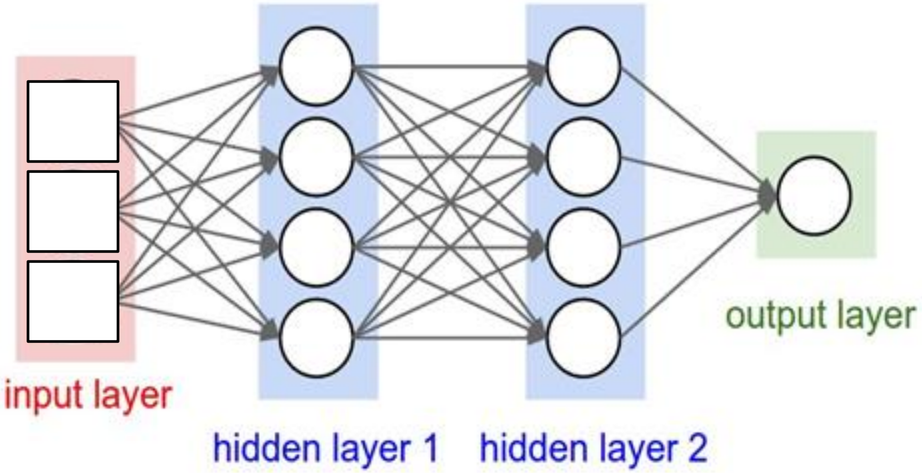
Single Layer Perceptron



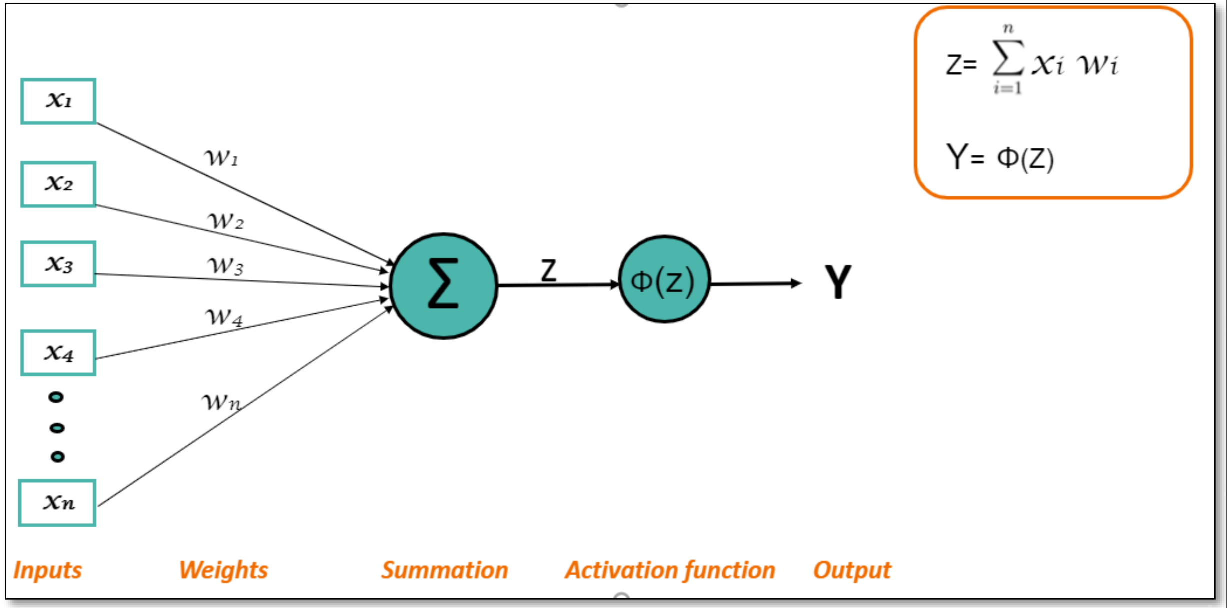
Why Use Only One Neuron ?

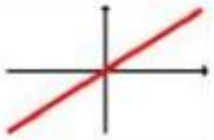
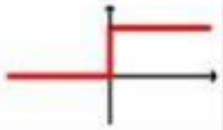
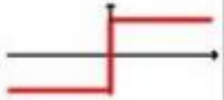




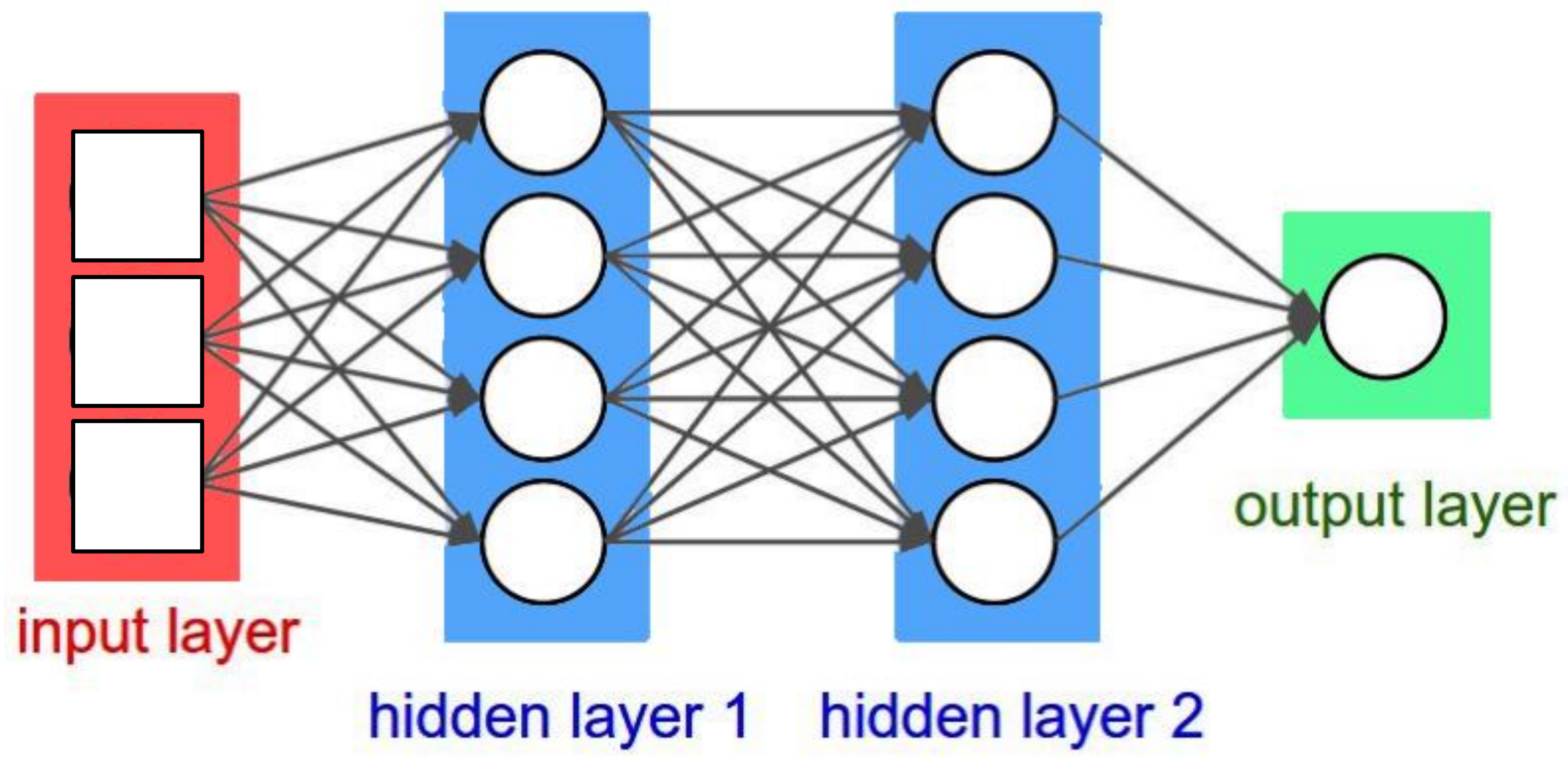


Activation Functions

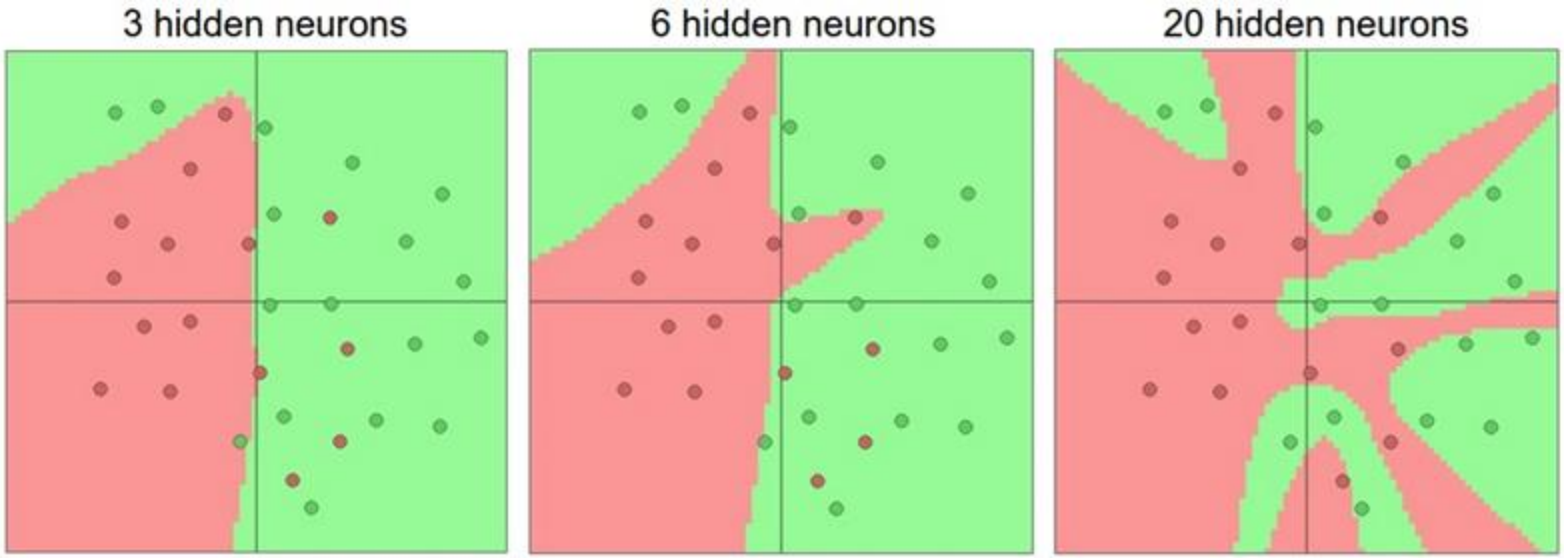


Activation Function	Equation		Example	1D Graph
Linear	$\phi(z) = z$		Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$		Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$		Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & z \geq \frac{1}{2} \end{cases}$		Support vector machine	
Logistic (sigmoid)	$\phi(z) = \dfrac{1}{1 + e^{-z}}$		Logistic regression,	
Hyperbolic Tangent (tanh)	$\phi(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$			
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$			

Deep Neural Networks (Multi Layer Perceptron (MLP))



More layers : Able to model complex decision boundaries



Note: Every neuron has a nonlinearity Φ inside. This is required to model non-linear boundaries

Multi layer Perceptron

Popular Artificial Neural Networks

Simple Problem

- Given #Bedrooms, Area (Sqft), LocalityIndex (#houses in the neighbourhood),
  - Predict whether the price of the house is “high” (1) or "low” (0).

Eg. House price from 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

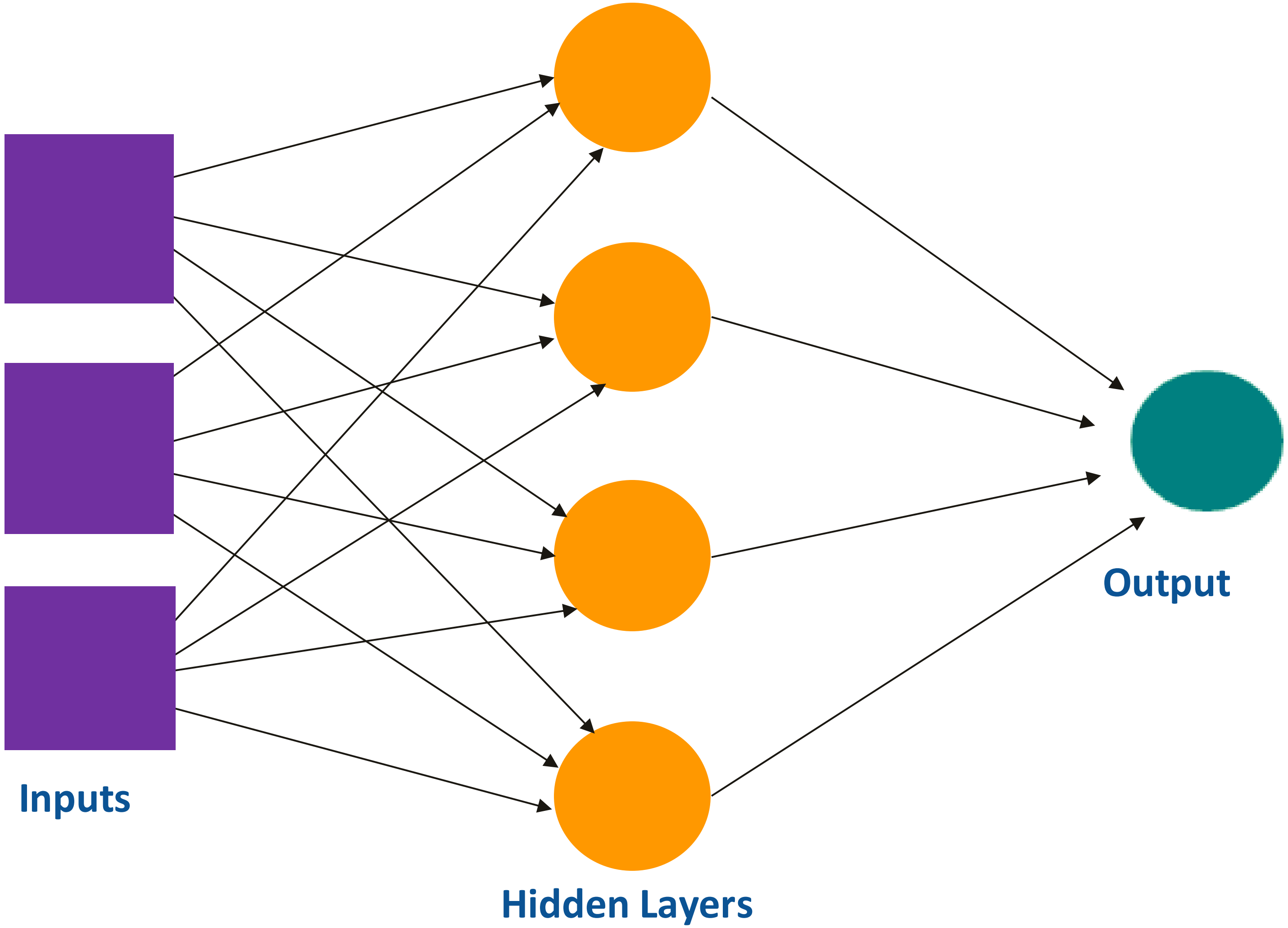
MLP Architecture

- MLP consists of at least three layers
  (input layer + hidden layer(s) + output layer)

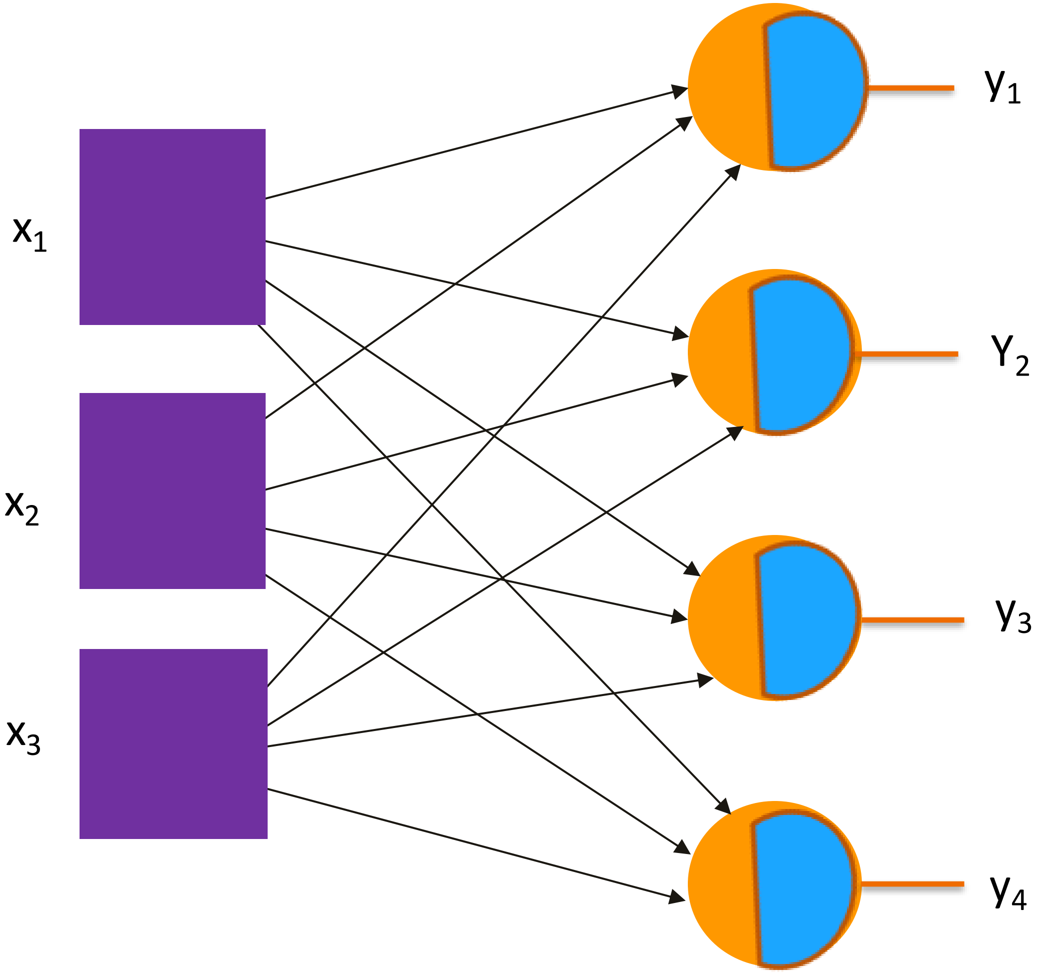
- Each layer (except input layer) consists of neurons that use an activation function (eg. step or sigmoid)

- Learning/Training: Backpropagation is used (not for today!!)

MLP

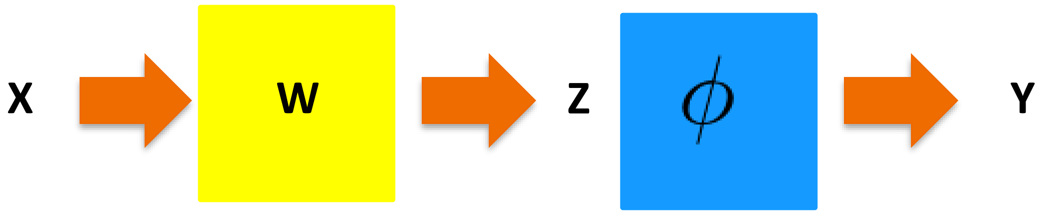


A Note on Notations

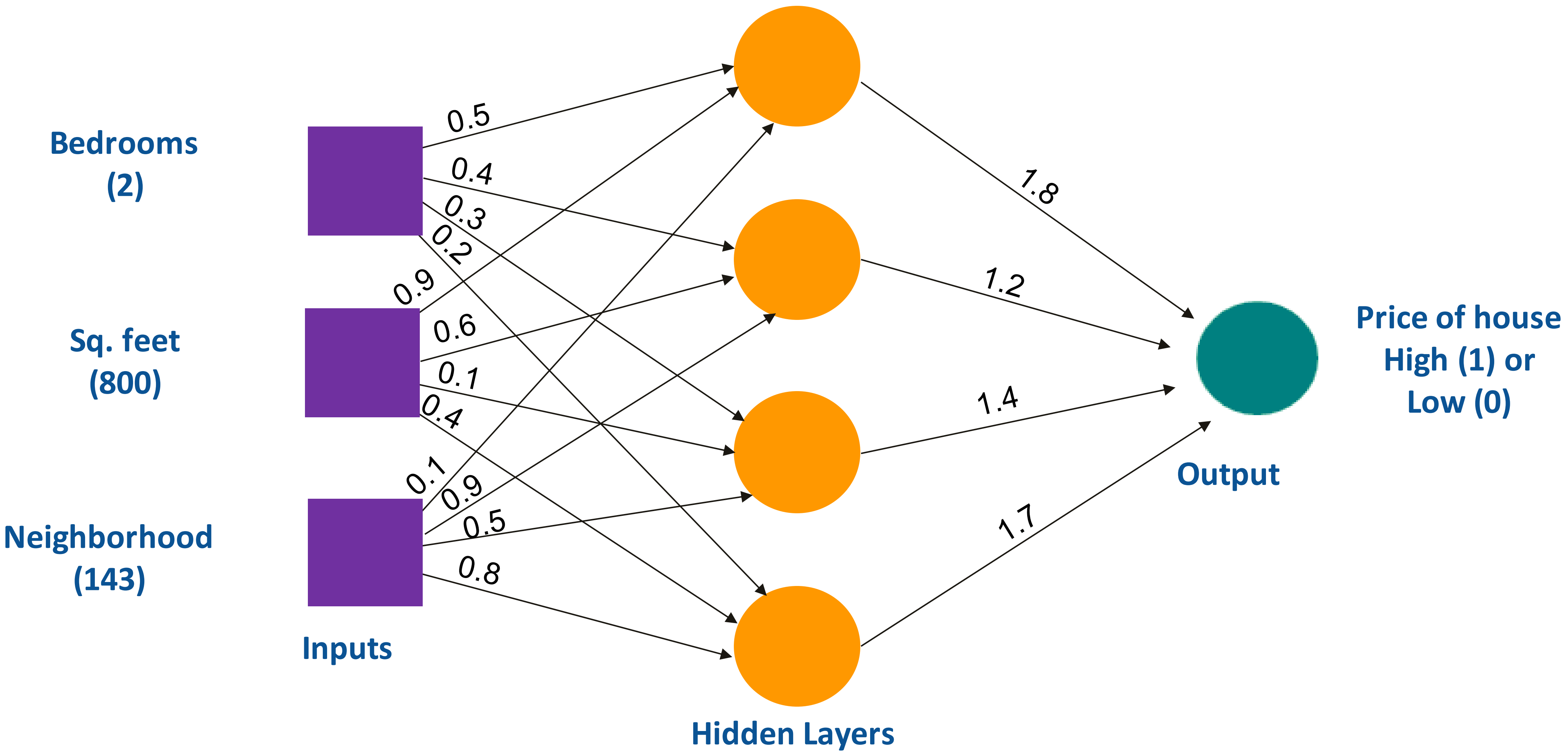


$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

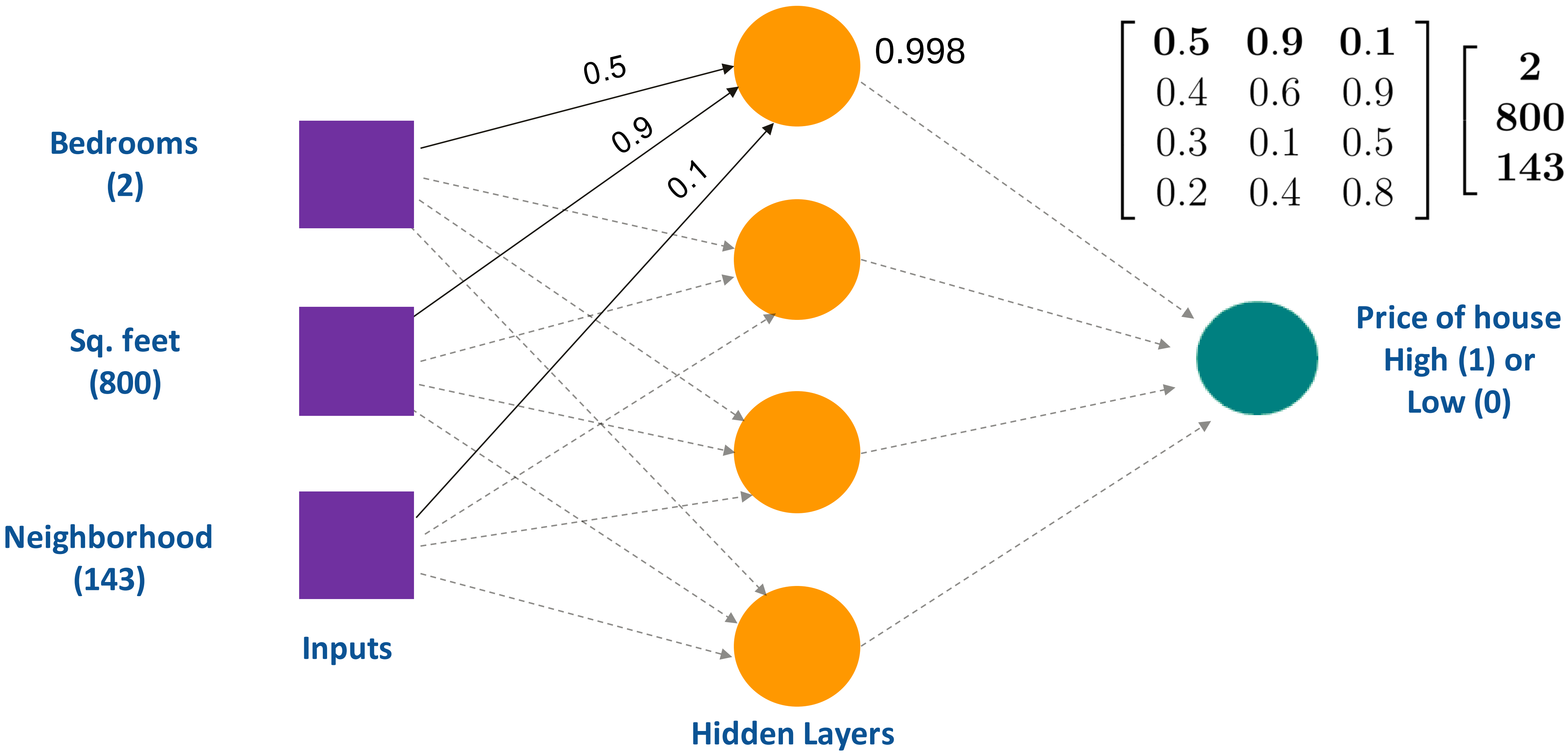
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} \phi(z_1) \\ \phi(z_2) \\ \phi(z_3) \\ \phi(z_4) \end{bmatrix}$$



Initialize weights

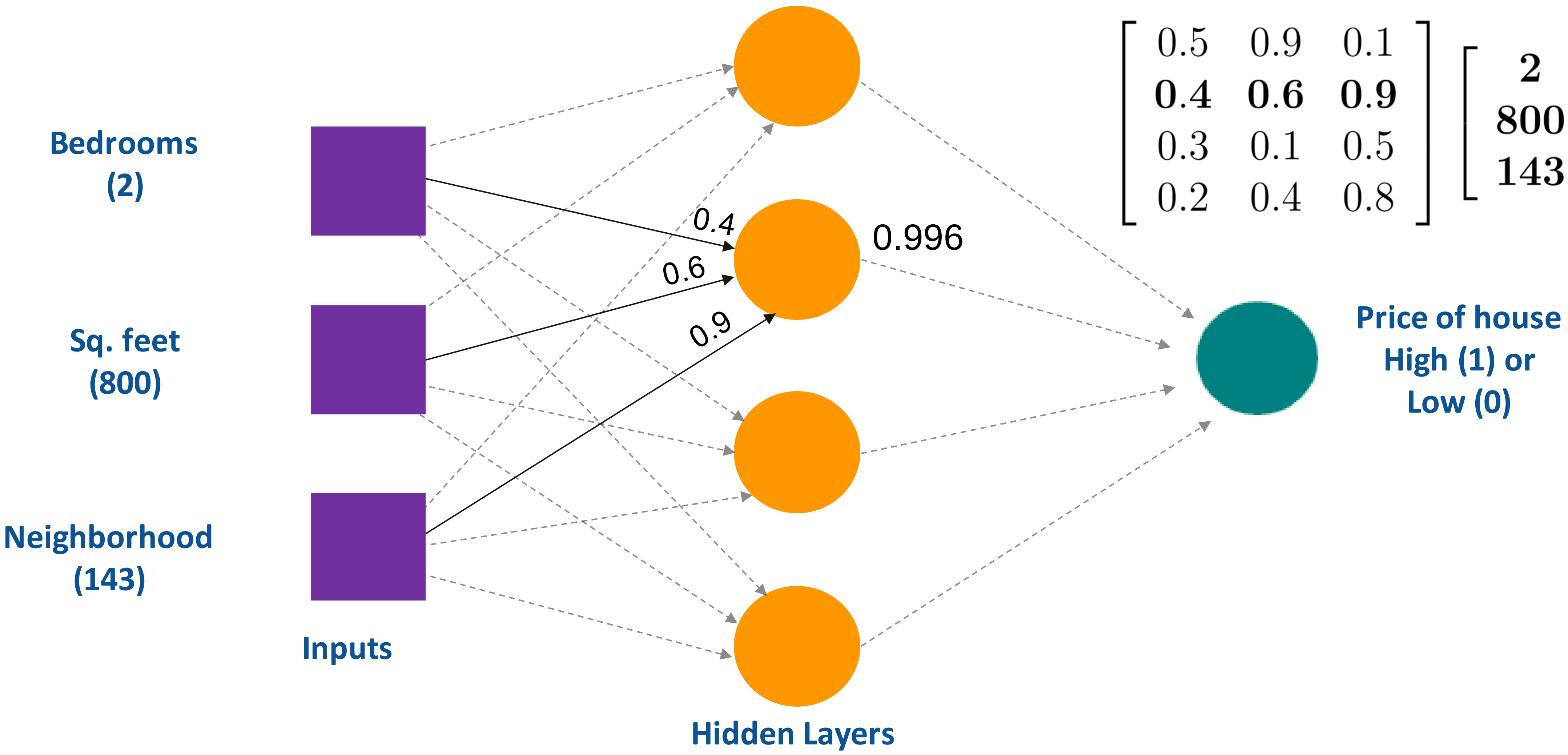


Weights at the first neuron



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

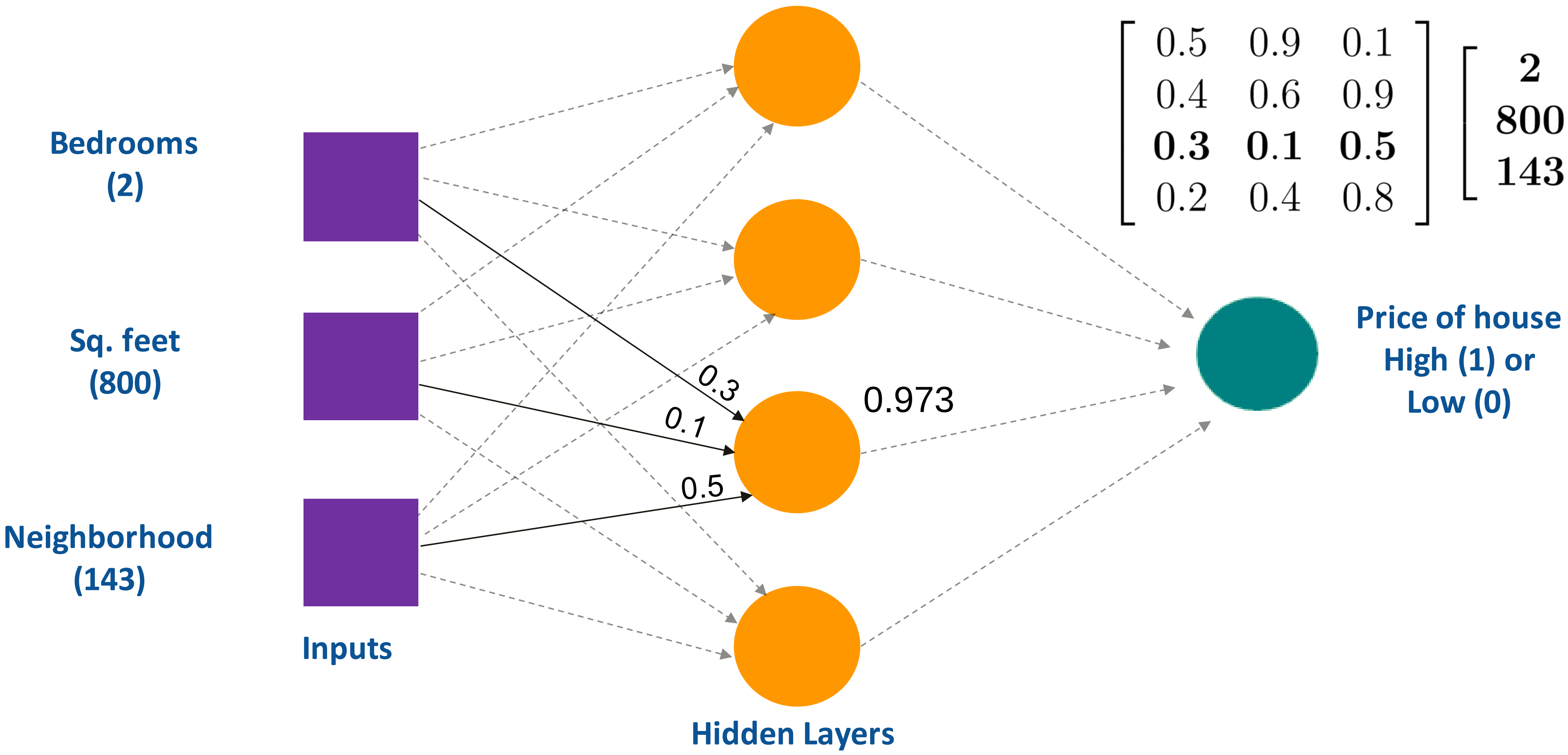
Weights at the second neuron



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

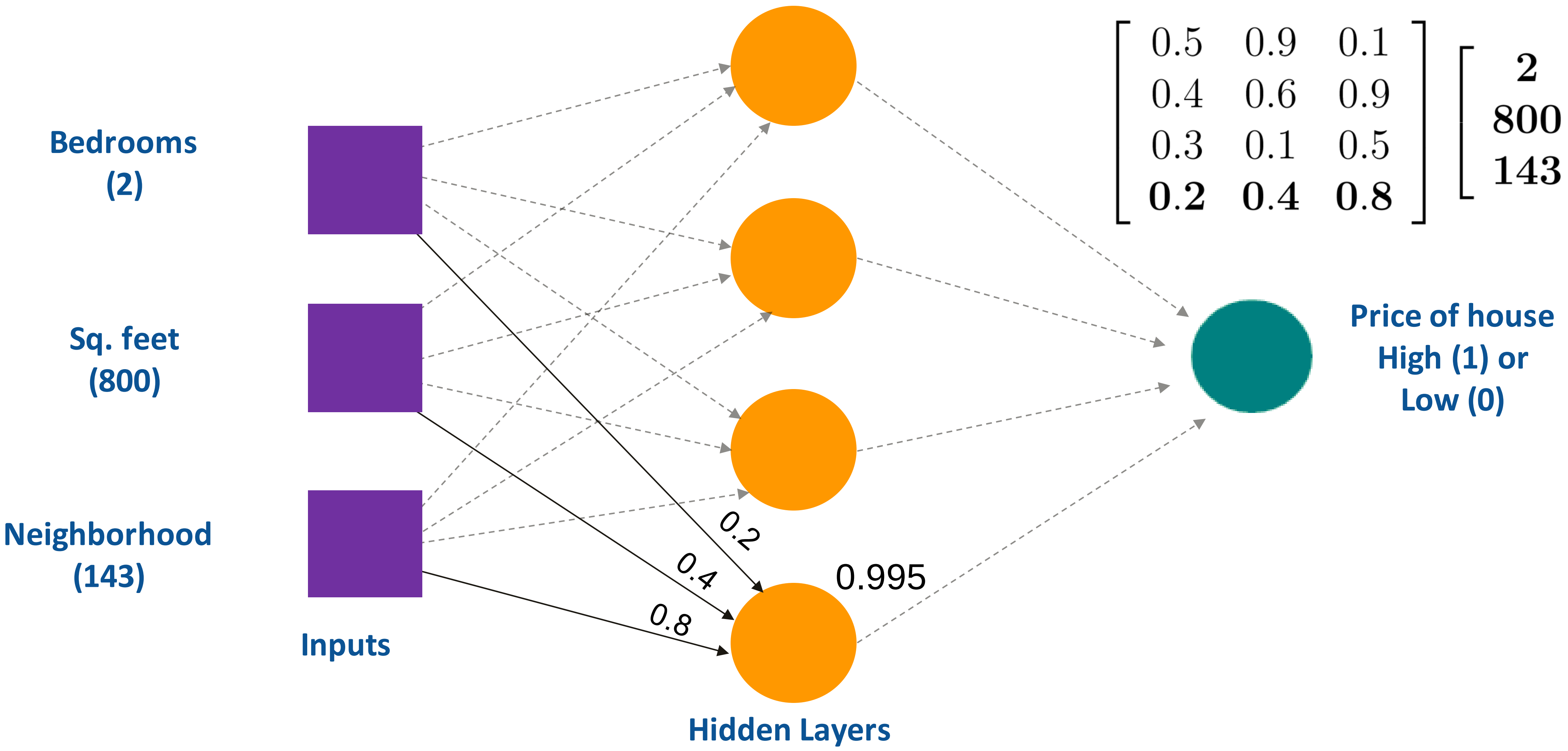
Weights at the third neuron

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

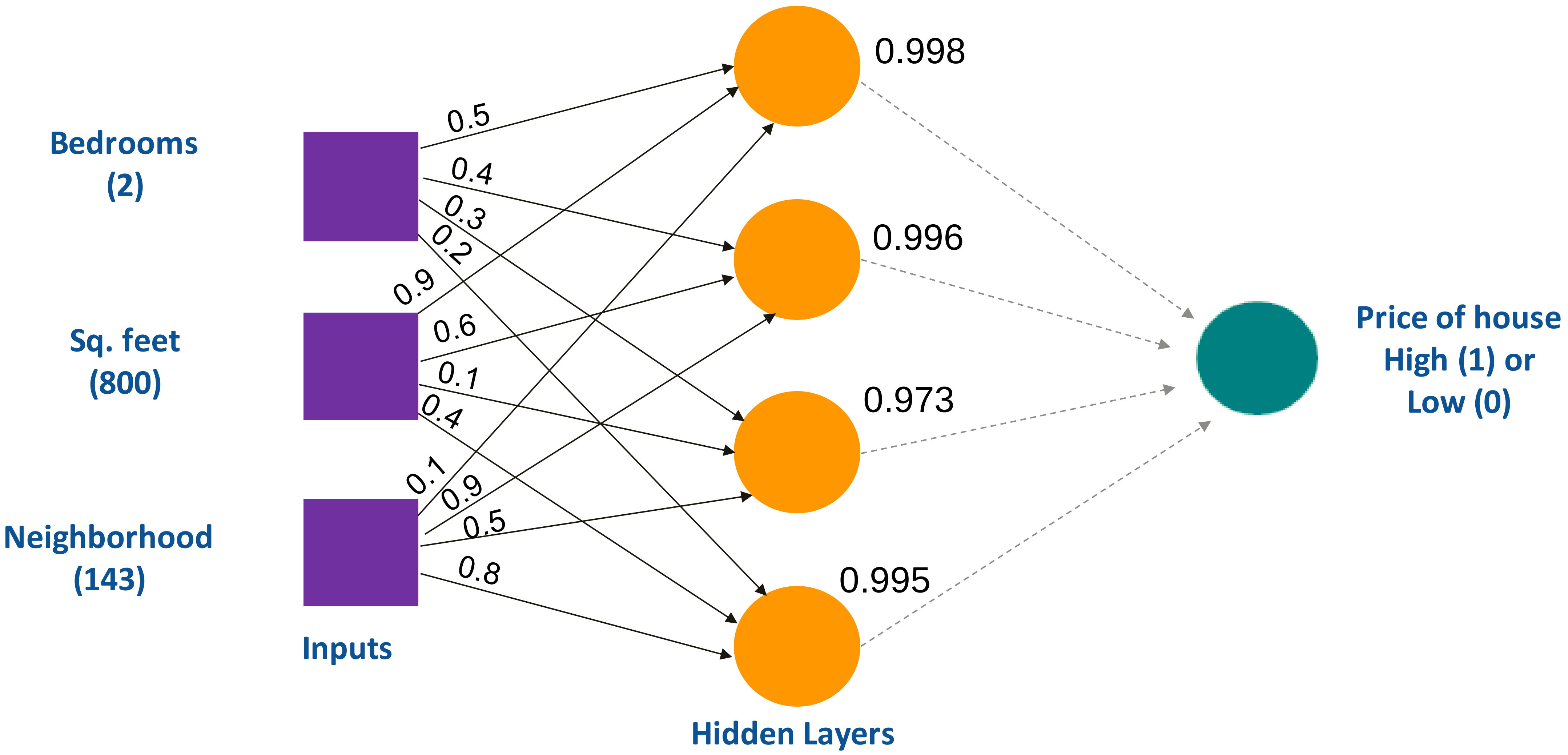


Weights at the fourth neuron

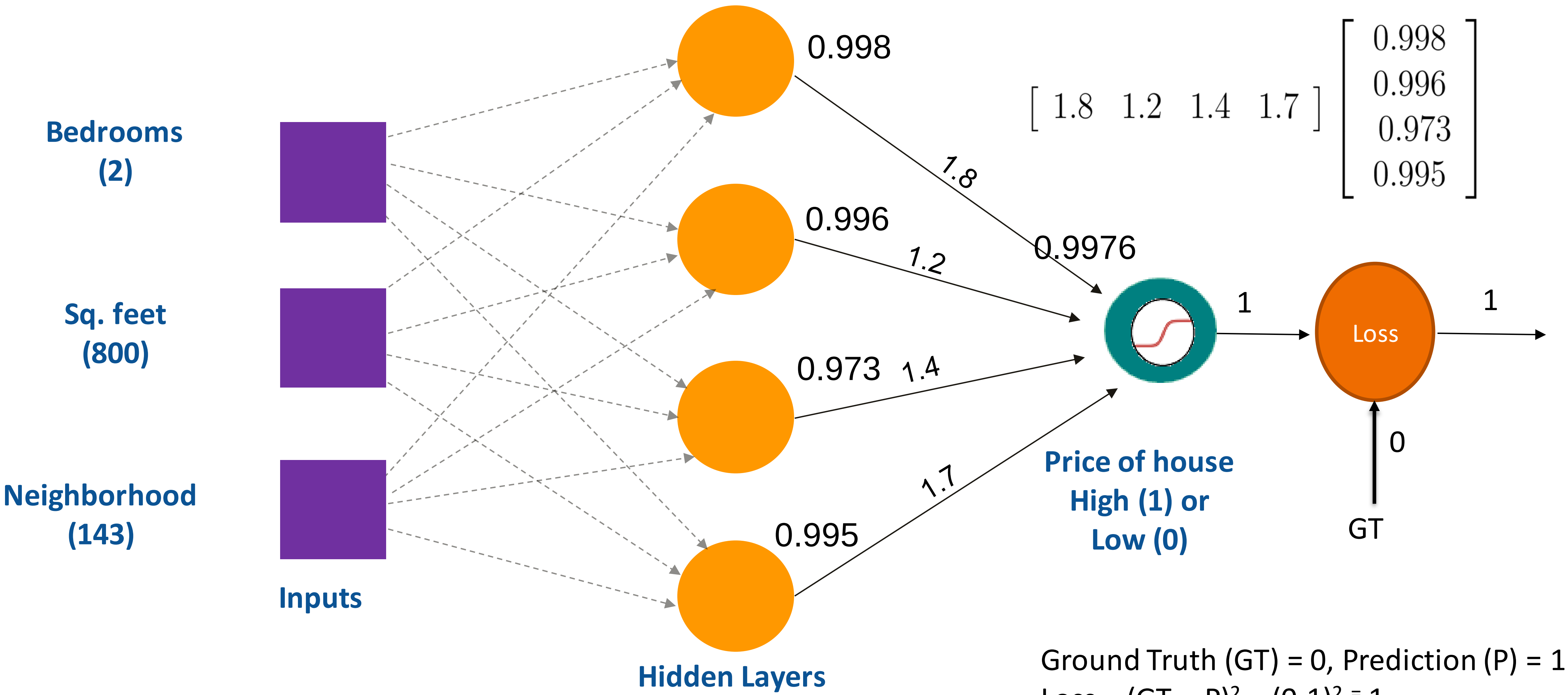
$$\begin{bmatrix} 0.5 & 0.9 & 0.1 \\ 0.4 & 0.6 & 0.9 \\ 0.3 & 0.1 & 0.5 \\ \mathbf{0.2} & \mathbf{0.4} & \mathbf{0.8} \end{bmatrix} \begin{bmatrix} \mathbf{2} \\ \mathbf{800} \\ \mathbf{143} \end{bmatrix}$$



Inputs to the next layer



Computations in next layer



Ground Truth (GT) = 0, Prediction (P) = 1

Loss = $(GT - P)^2 = (0-1)^2 = 1$

Why Nonlinearity in MLP?

Answer?

Comment: Limitation of "Linear MLP"

Z = W1 X

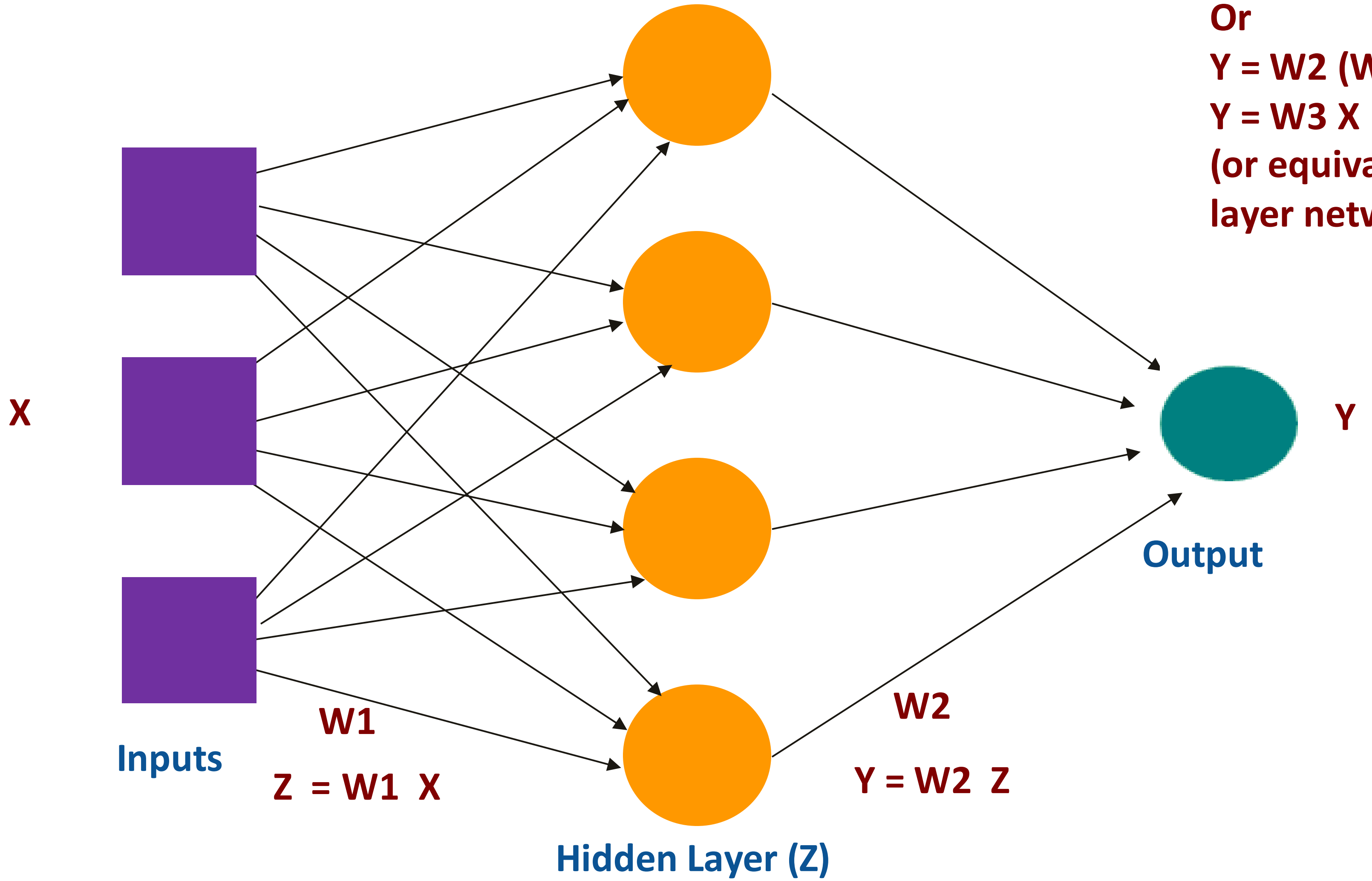
Y = W2 Z

Or

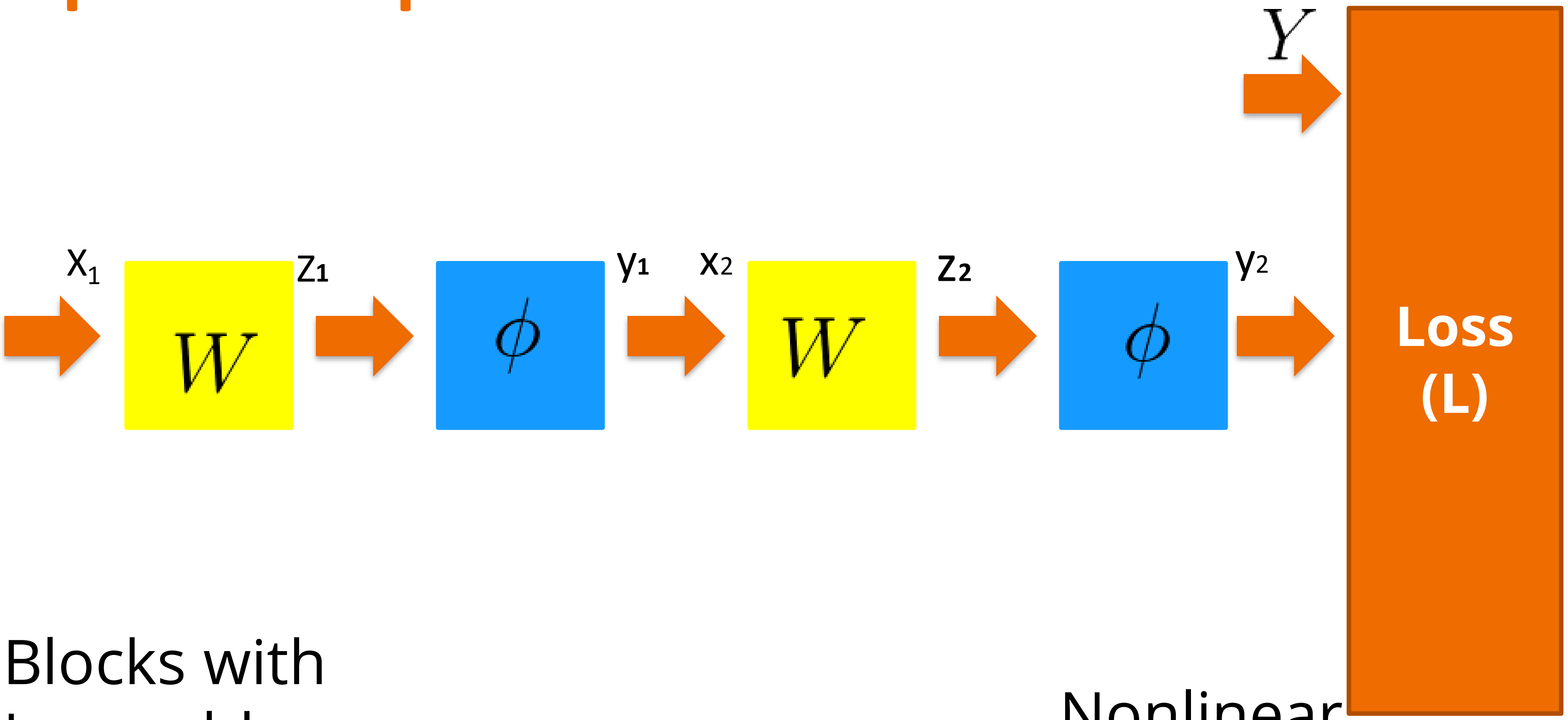
Y = W2 (W1 X)

Y = W3 X

(or equivalent to a single layer network)



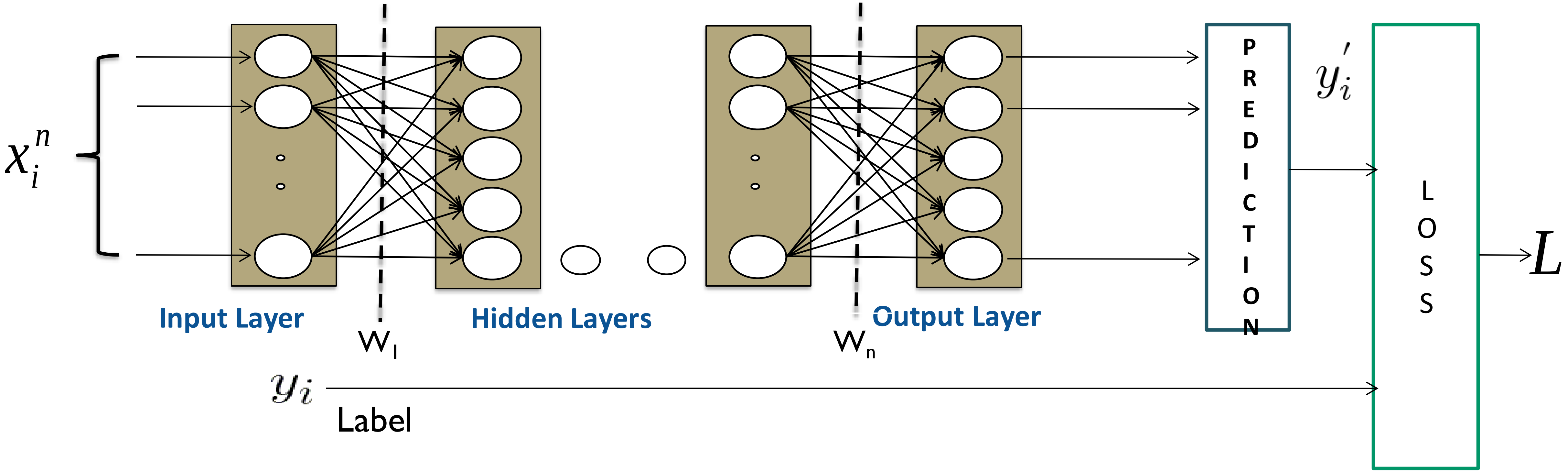
A simpler view point



Blocks with Learnable parameters Matrix Multiplication

Nonlinear functions (often non learnable)

Loss or Objective

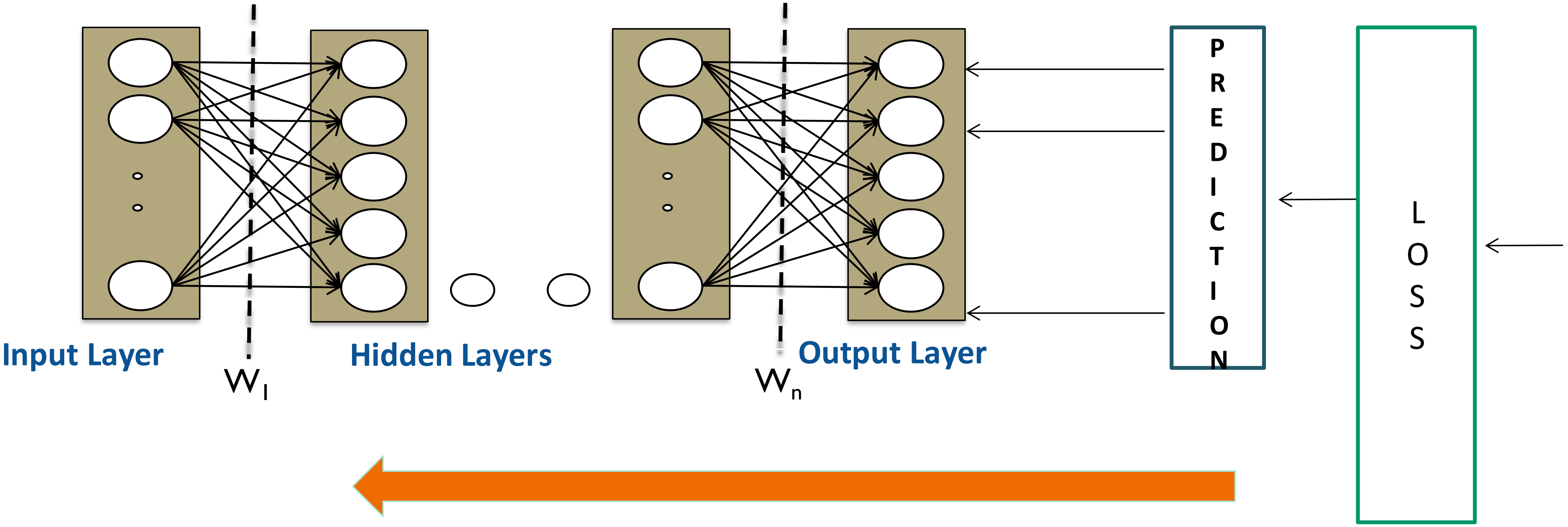


Objective: Find out the best parameters which will minimize the loss.

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

$$L = \frac{1}{2}\|y_i' - y_i\|_2^2$$ → **E.g. Squared Loss**

Back Propagation



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

Each layer's weights are updated based on the derivative of its output w.r.t. input and weights

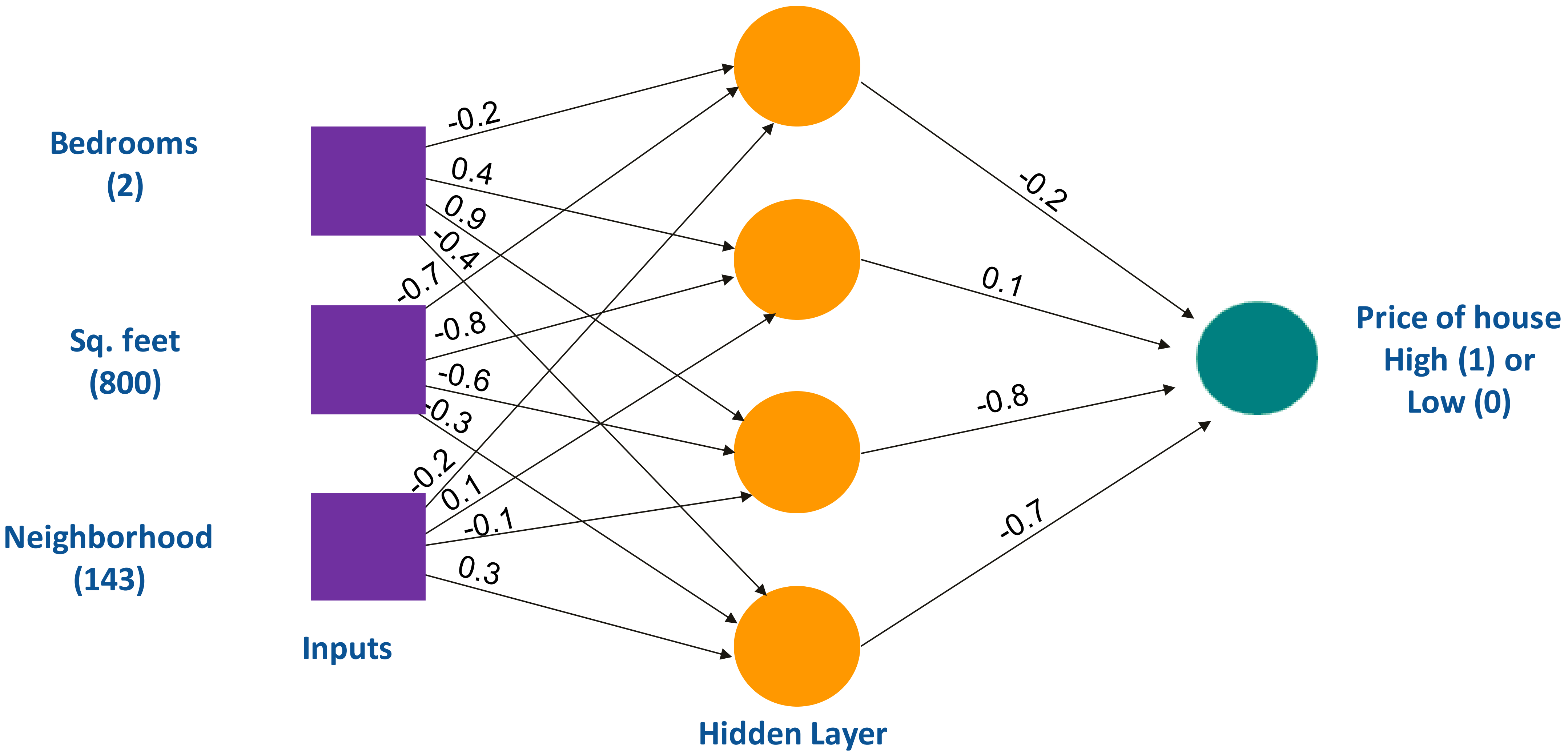
Backpropagation Algorithm

- Input: A set of examples (x_i, y_i)
- Objective: Update weights **W** so that error is small/optimal.
- How: For the set (or subset) of examples,
  1. Compute output
  2. Update weights (gradient descent)
  3. Repeat 1-3 until convergence.

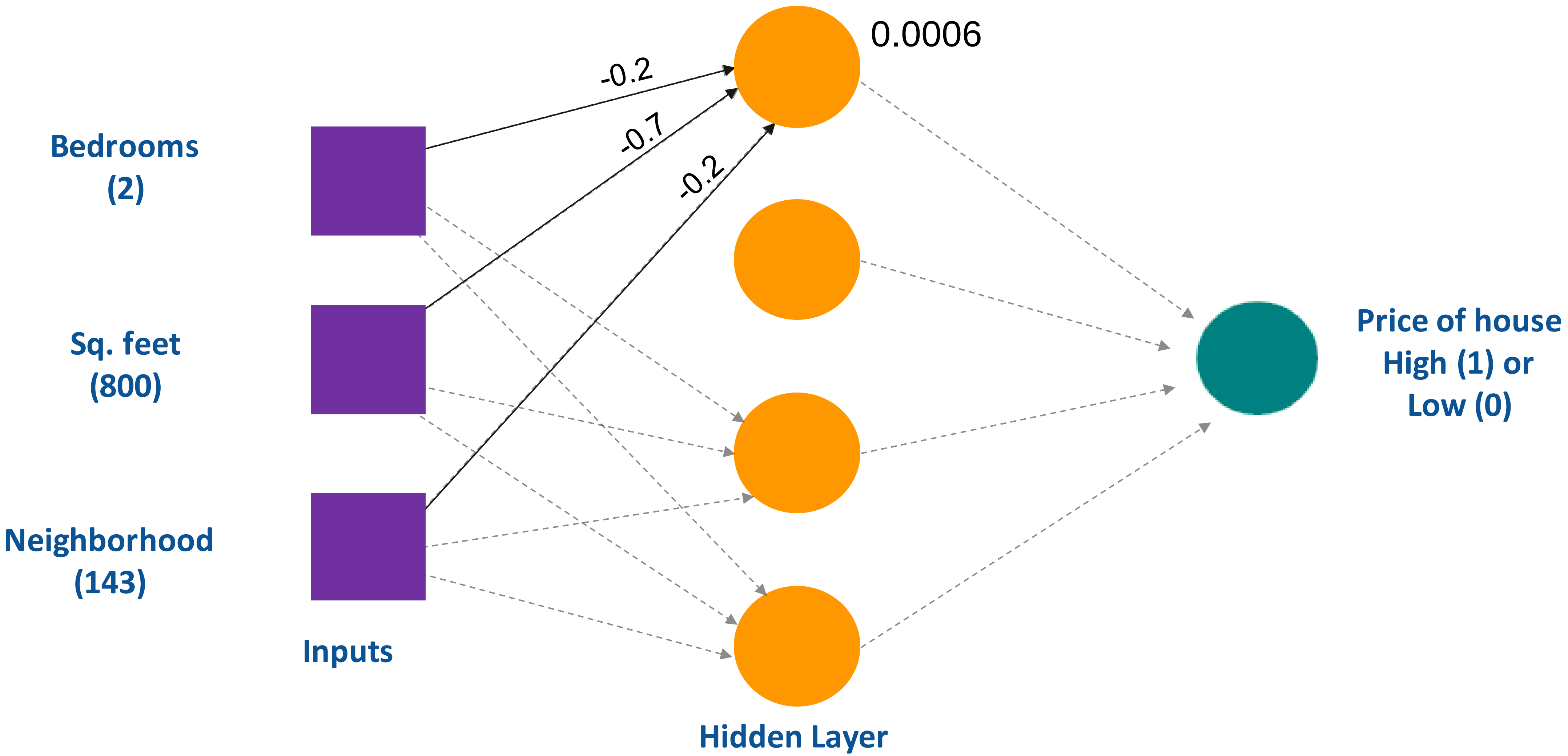
Loss or Error

- $0.998 * 1.8 + 0.996 * 1.2 + 0.973 * 1.4 + 0.995 * 1.7 = \phi(6.04) = 0.9976$
- For threshold nonlinearity Output = 1
- The actual class (0) deviates from the predicted class (1)
- The squared error or Loss is (1-0) * (1-0) = 1
- The weights to be updated by **backpropagation to reduce the error**

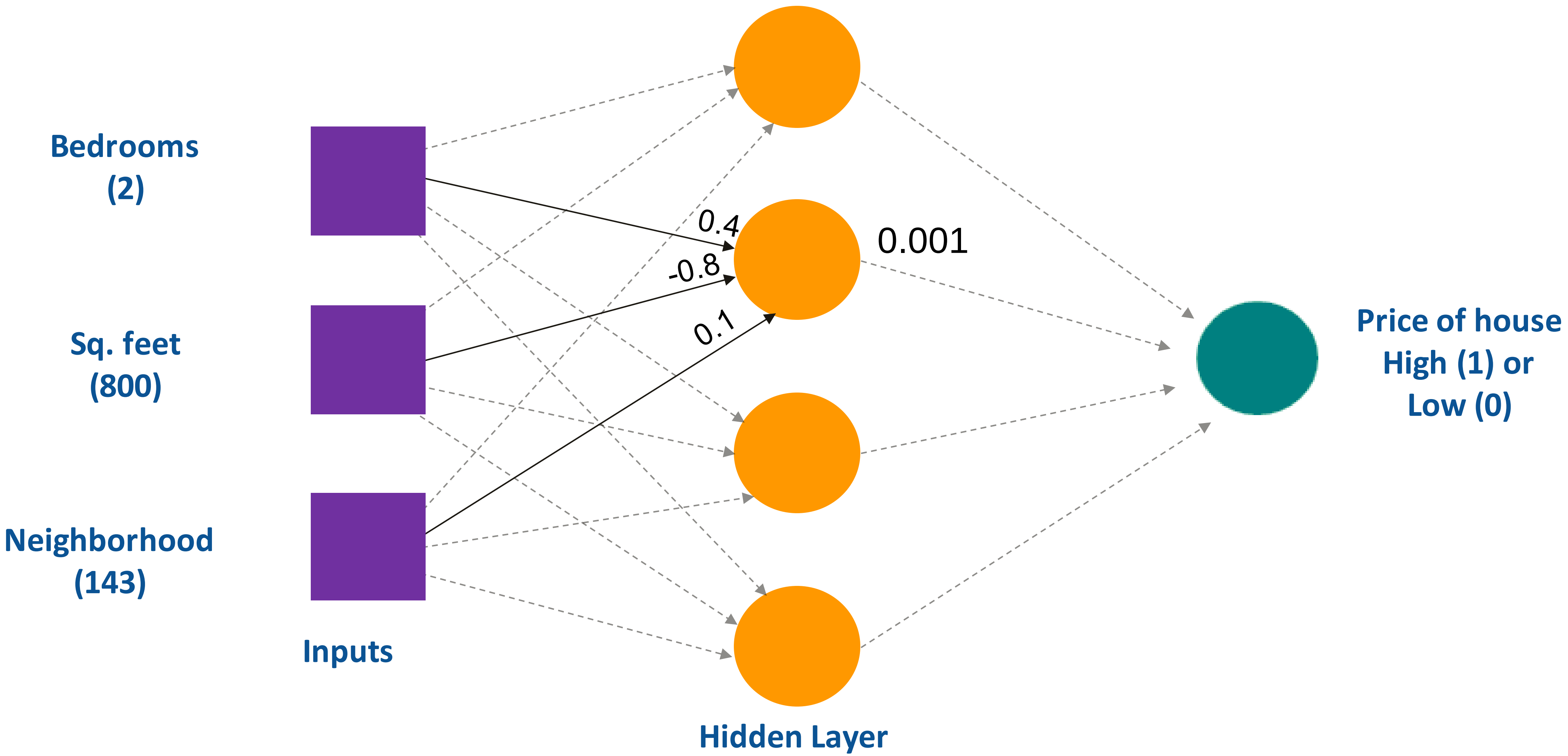
Weights obtained by backpropagation



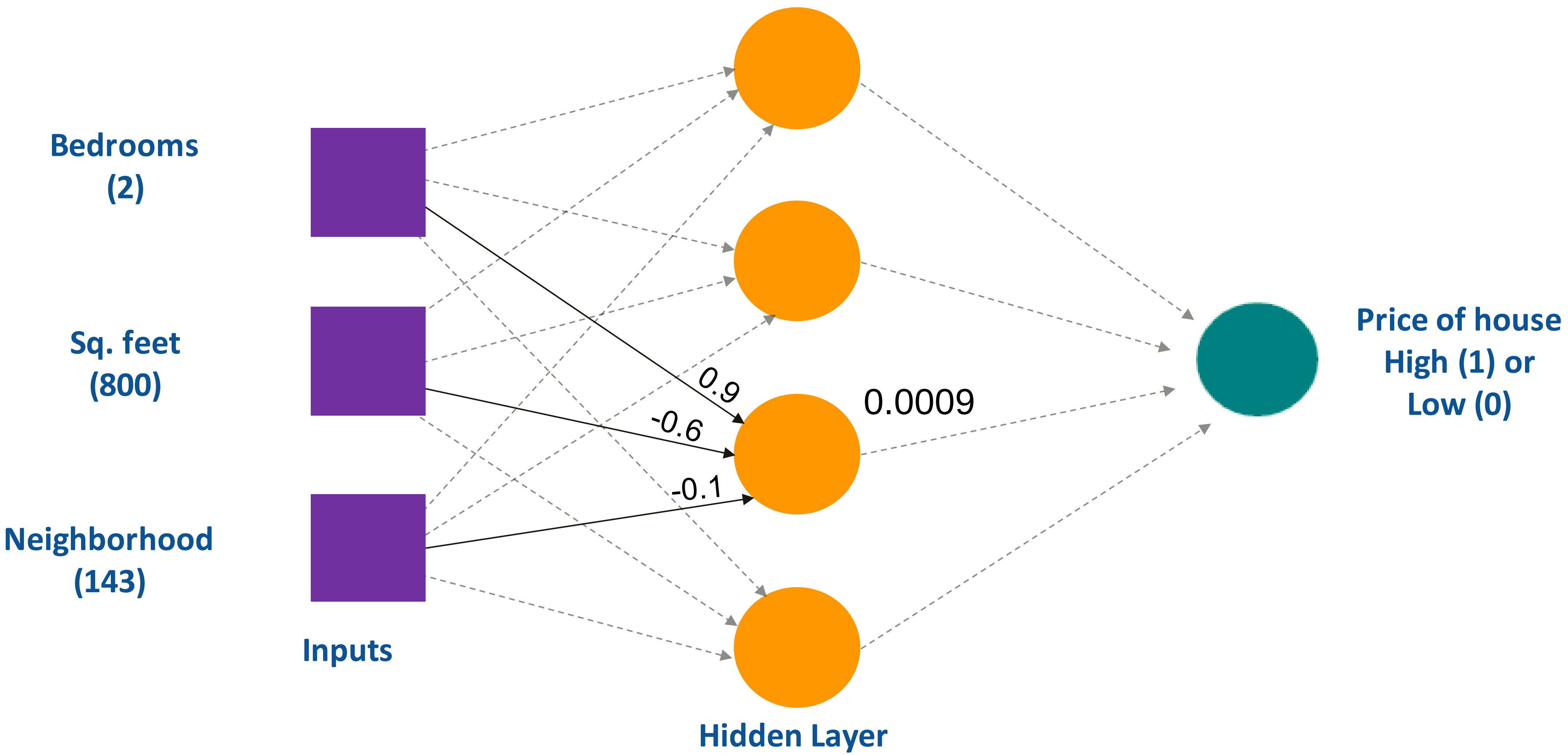
Weights at the first neuron



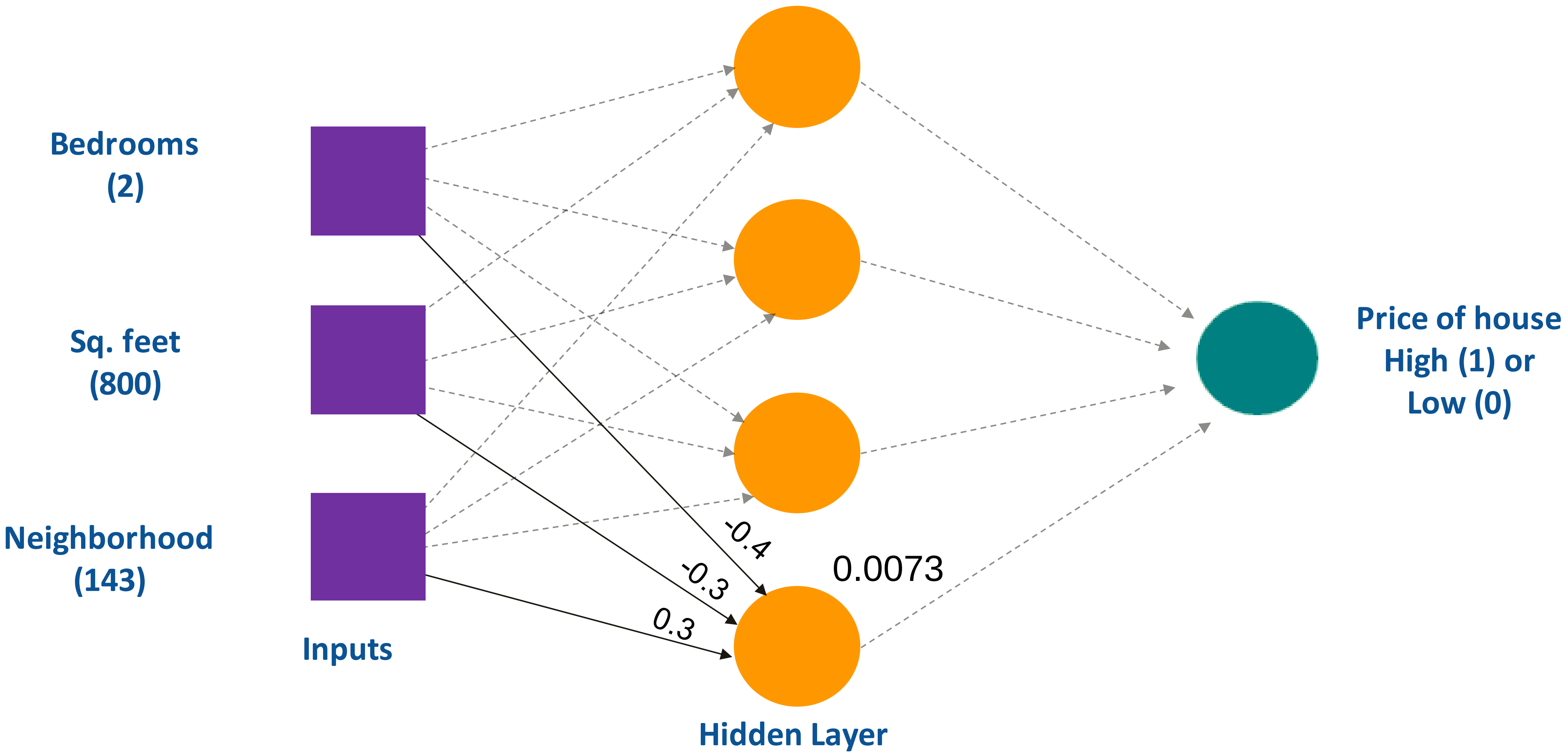
Weights at the second neuron



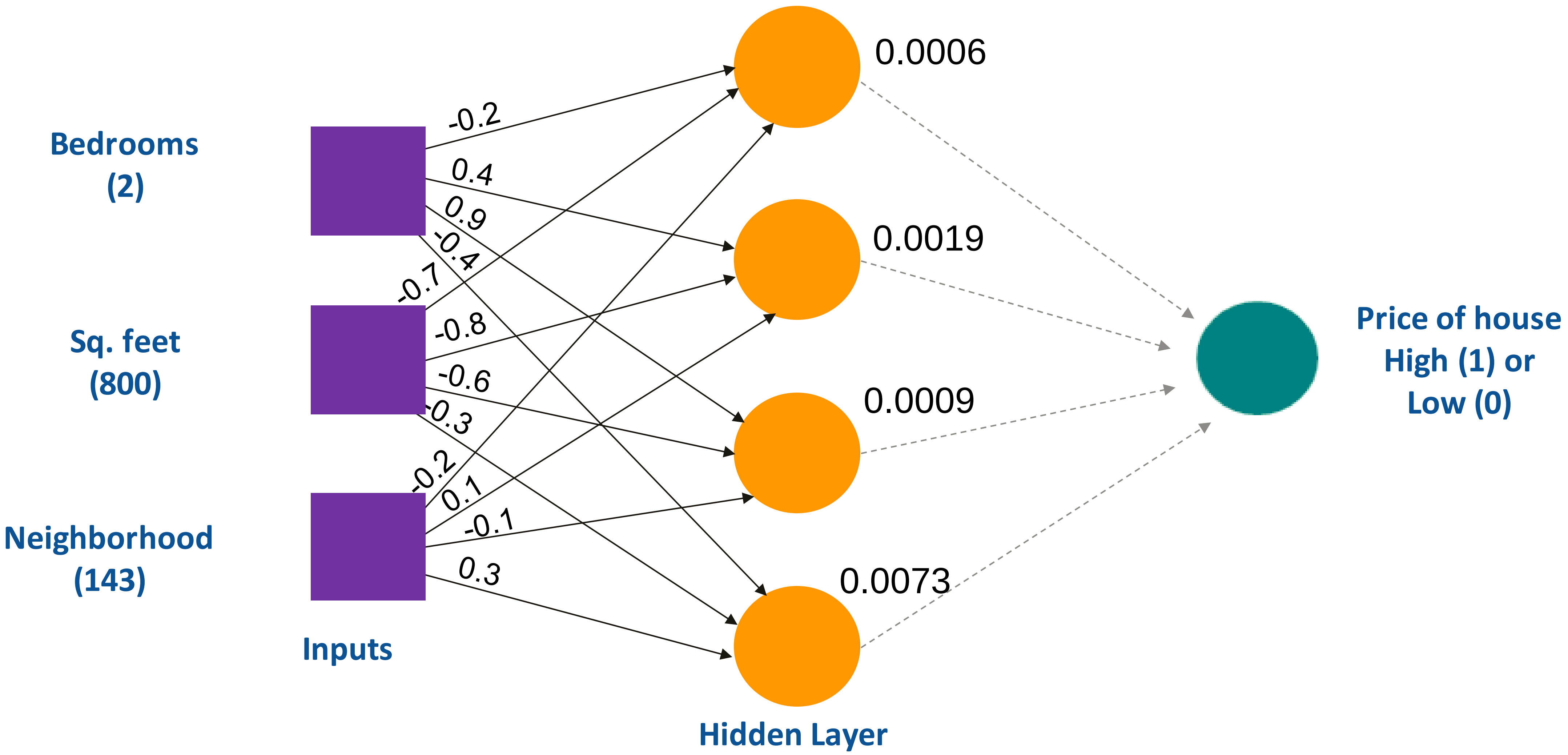
Weights at the third neuron



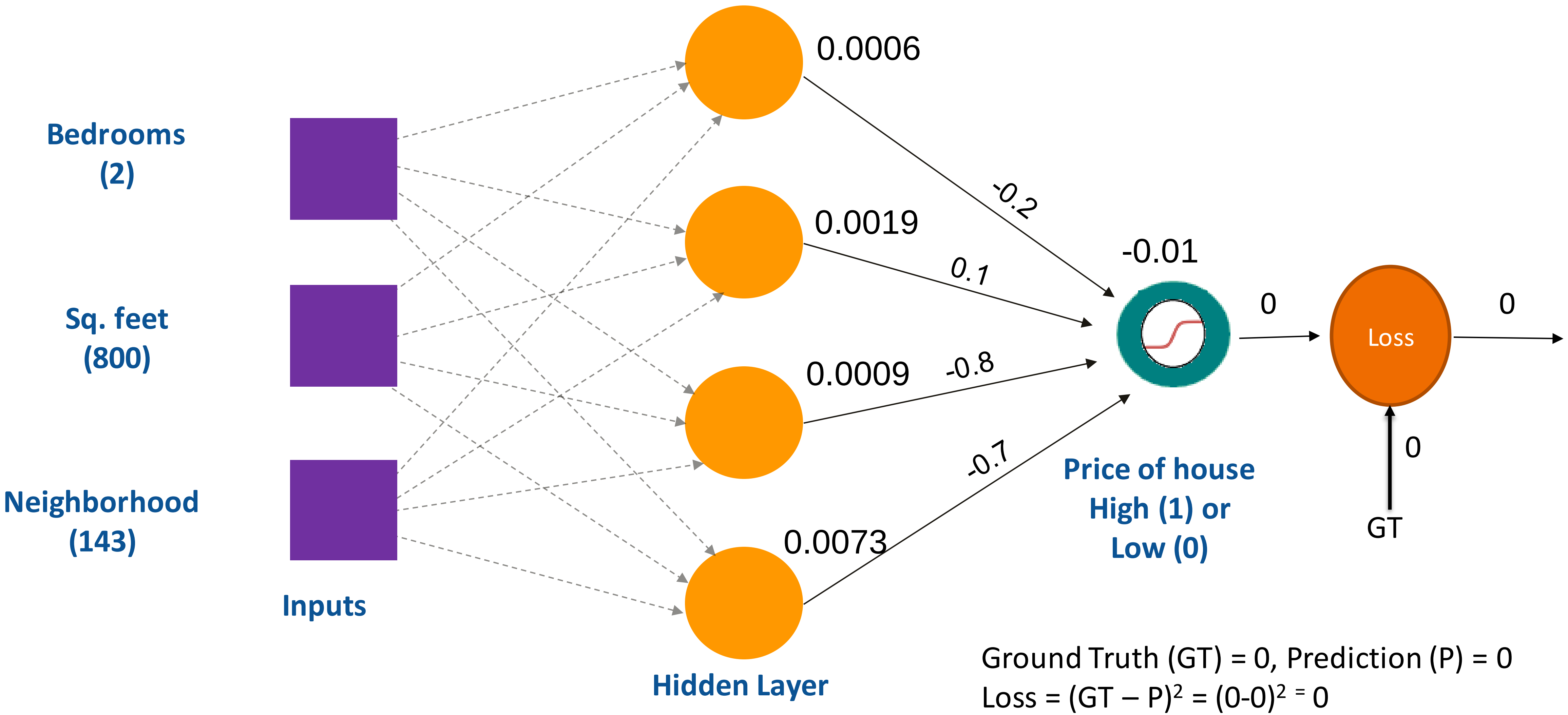
Weights at the fourth neuron



Activation function applied at first layer



Activation function applied at Second layer



Ground Truth (GT) = 0, Prediction (P) = 0

Loss = $(GT - P)^2 = (0-0)^2 = 0$

Loss/Error at this stage

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

  = -0.001
- The square error or Loss with activation is (0-0)*(0-0) = 0
- The actual class (0) and the predicted class is also (0)
- Loss is now reduced from 1 to 0; appropriate weights are also found!!

Journey so far…

Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

Pick Model and Hyper-params for Task

- ✓ Linear and Non-linear Functions
  - ✓ Activation Functions
  - ✓ Optimization function
  - ✓ Loss Function

Model Training / Optimization

- ✓ Linear and Non linear Classifiers
  - ✓ Single Layer Perceptron
  - ✓ Multi Layer Perceptron (MLP)
  - ✓ Forward propagation
  - ✓ Back Propagation

Case Study

Classification and Regression using MLP

Eg. House price from attributes (Classification)

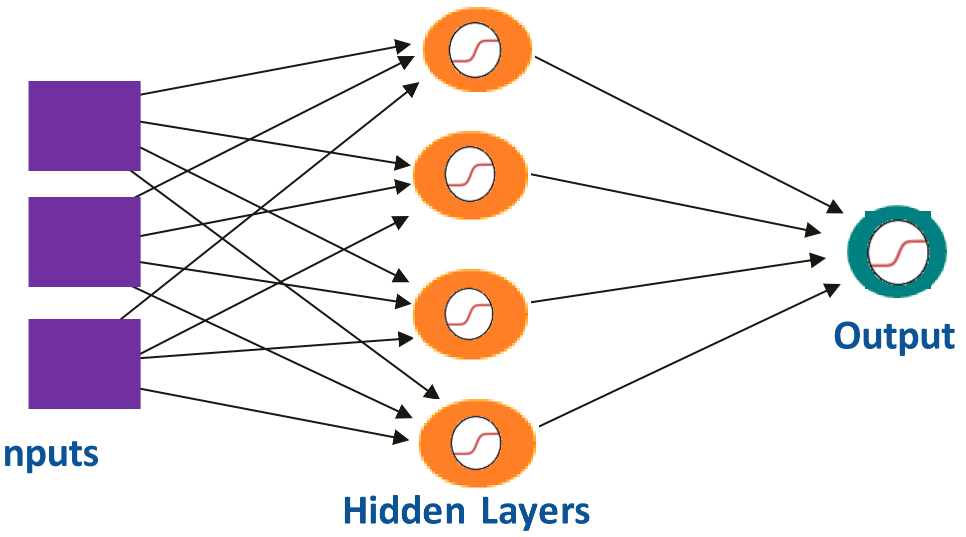
Bedrooms	Sq. Feet	Neighborhood (no. of houses in the locality)	Price high or low? High (1), Low (0)
3	2000	90	1
2	800	143	0
2	850	167	0
1	550	267	0
4	2000	396	1

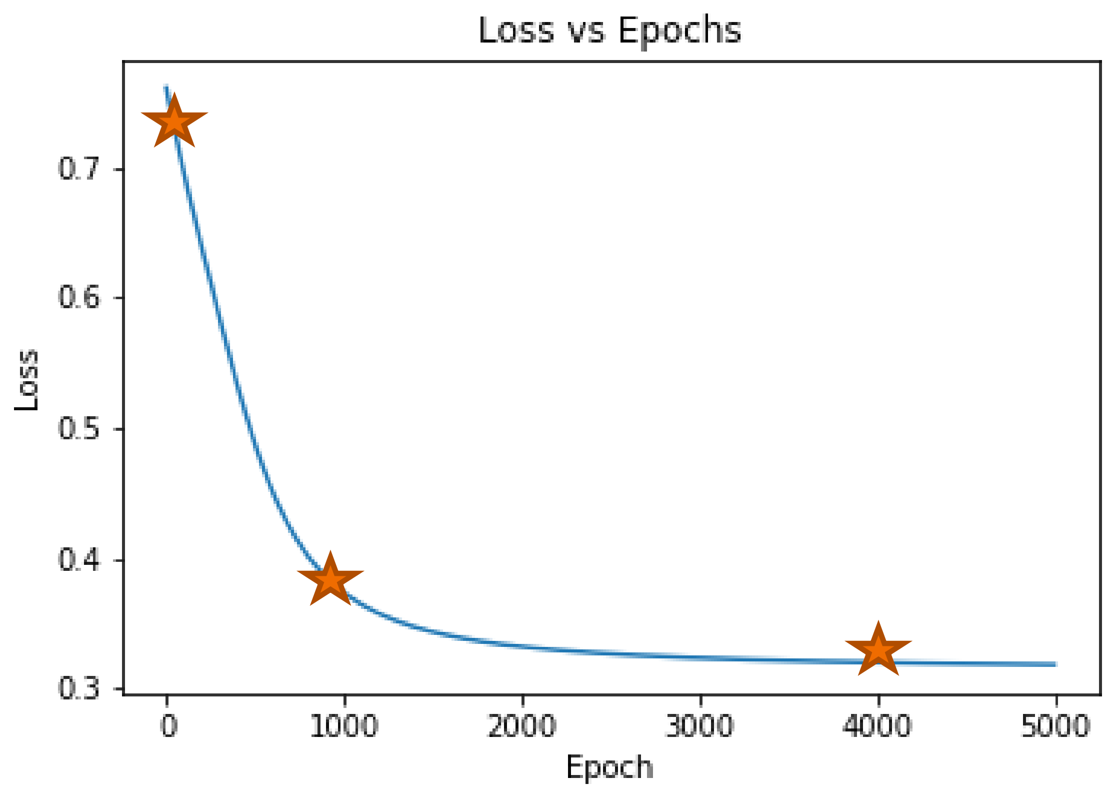
Code Available

MLP for Classification

$$W1 = \begin{bmatrix} 0.2789 & 0.3983 & 0.2913 \\ -0.4123 & 0.1979 & -0.4385 \\ 0.1928 & -0.7861 & -0.0292 \\ -0.2227 & -0.1160 & -0.2257 \end{bmatrix}$$

$$W2 = [[-0.5871, -0.0093, -0.1943, 0.1869]]$$





Code Available

Eg. House price from attributes (Regression)

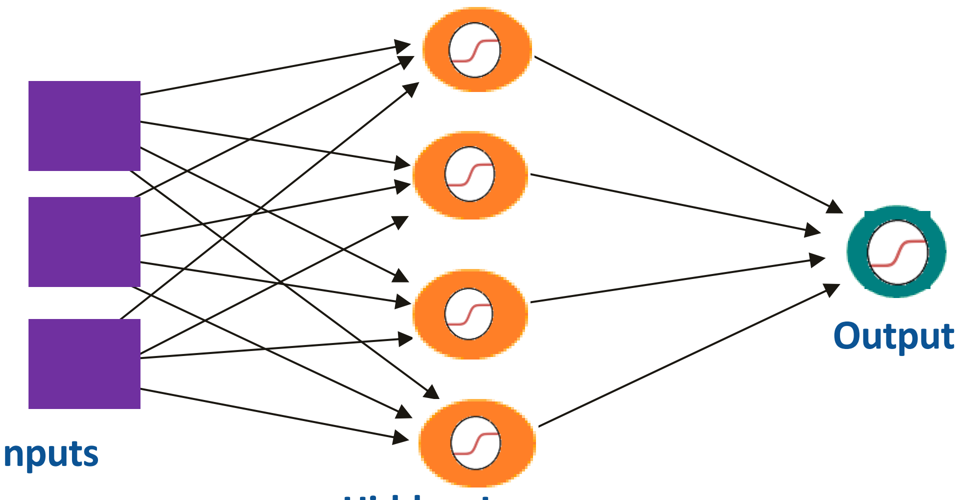
Bedrooms	Sq. Feet	Neighborhood (no. of houses in the locality)	Price (in lakhs)
3	2000	90	23.0
2	800	143	8.0
2	850	167	9.0
1	550	267	9.0
4	2000	396	25.0

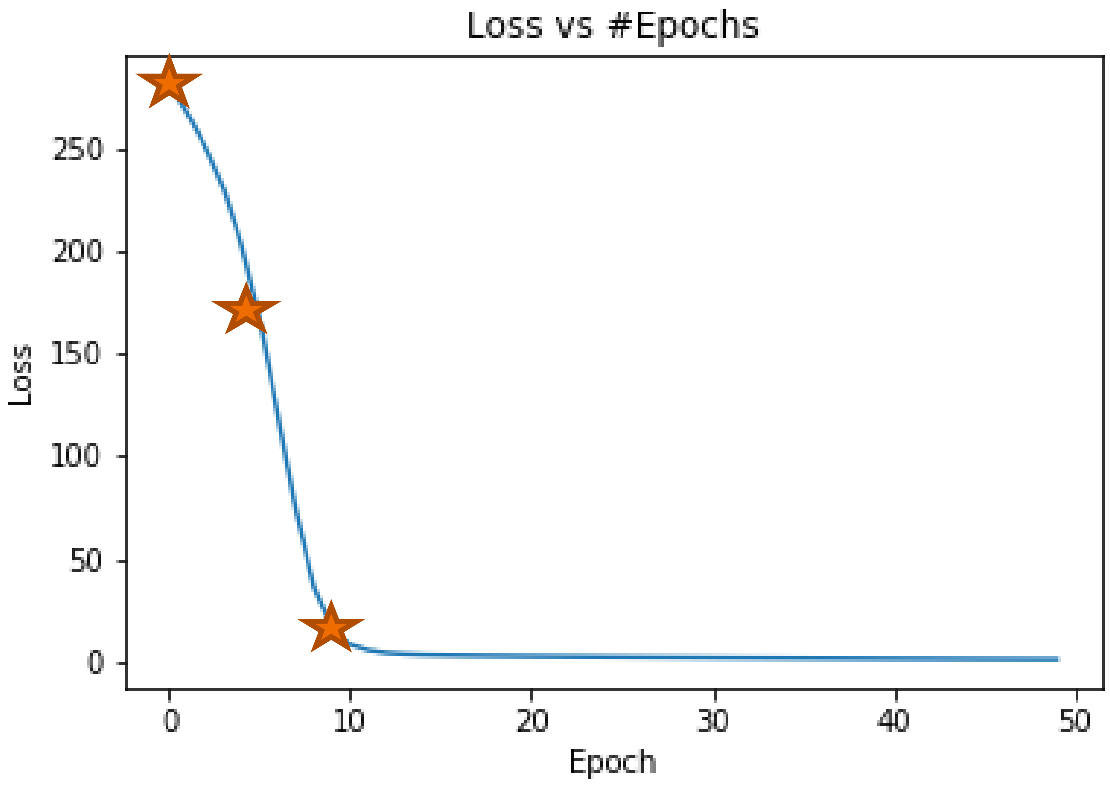
Code Available

MLP for Regression

$$W1 = \begin{bmatrix} -0.220 & -0.791 & 0.445 \\ -0.316 & -0.875 & -0.364 \\ -0.027 & 0.651 & -0.473 \\ 0.648 & 0.391 & 0.234 \end{bmatrix}$$

$$W2 = [2.4061, -1.8249, 0.4783, 2.1482]$$

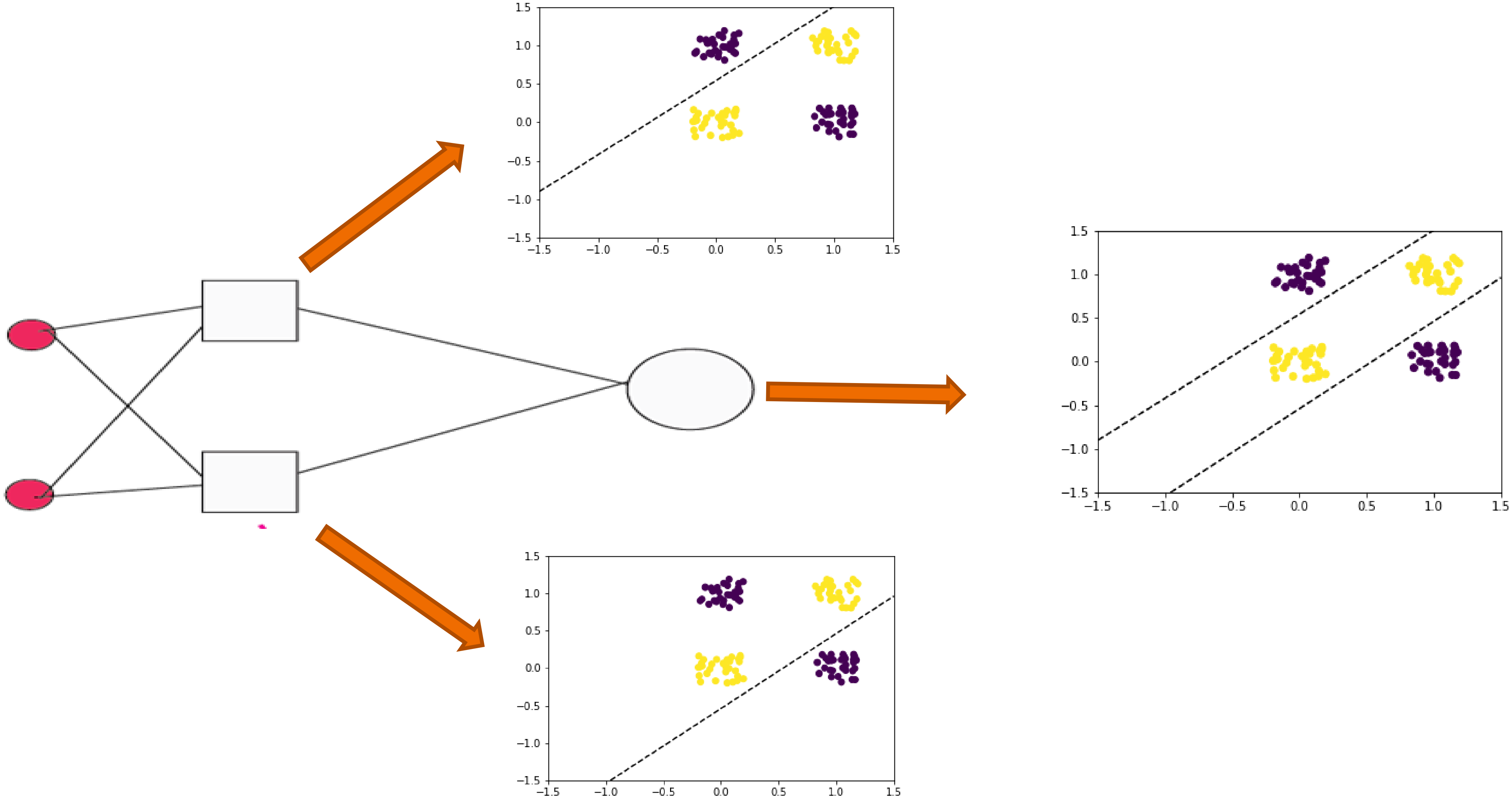




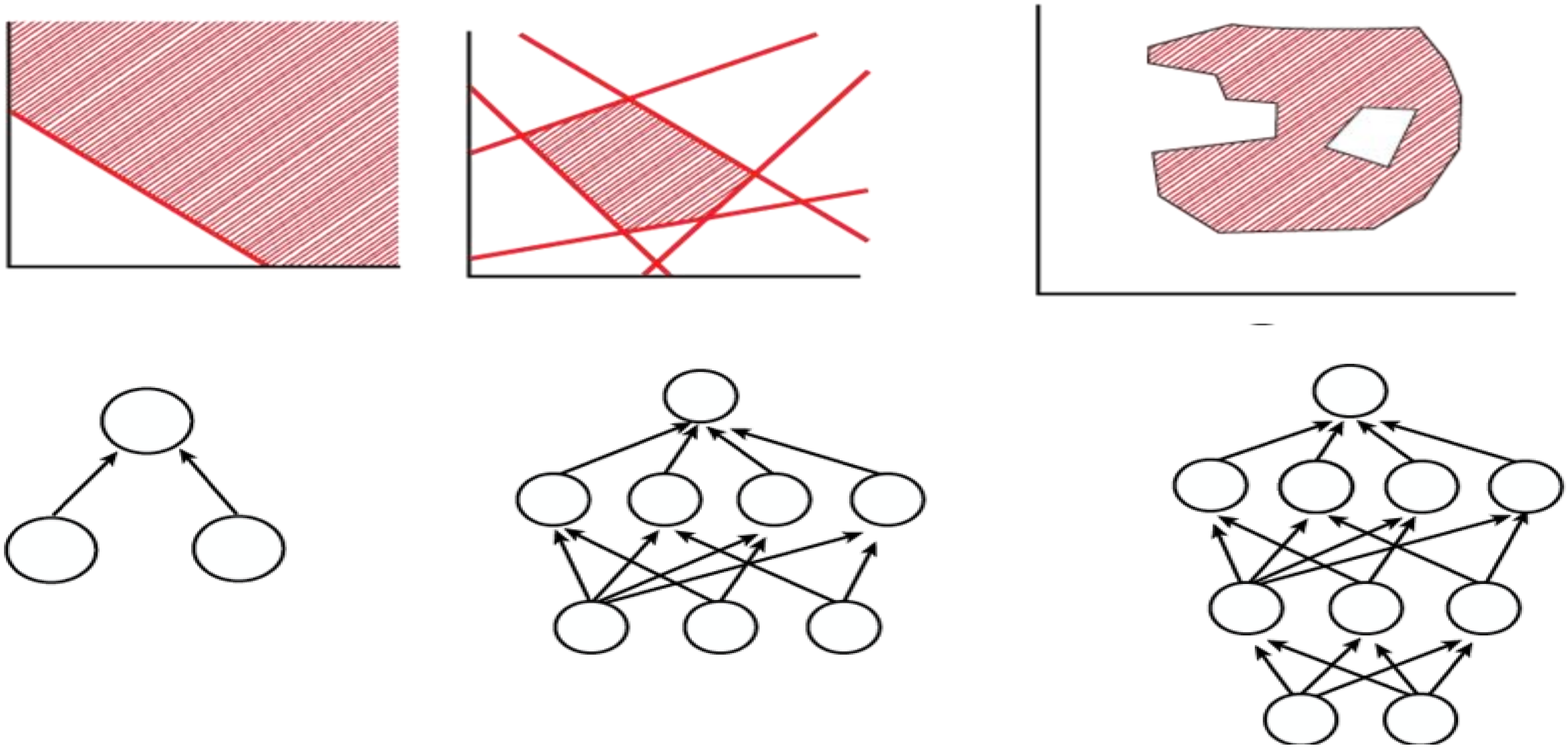
Questions?

Intuitive Explanation

How MLP Works? (A naïve view)



Deeper Networks

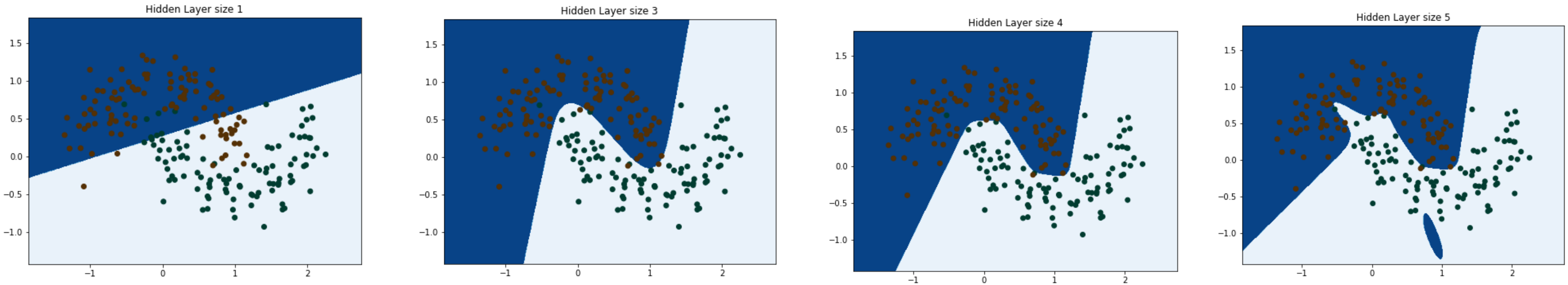


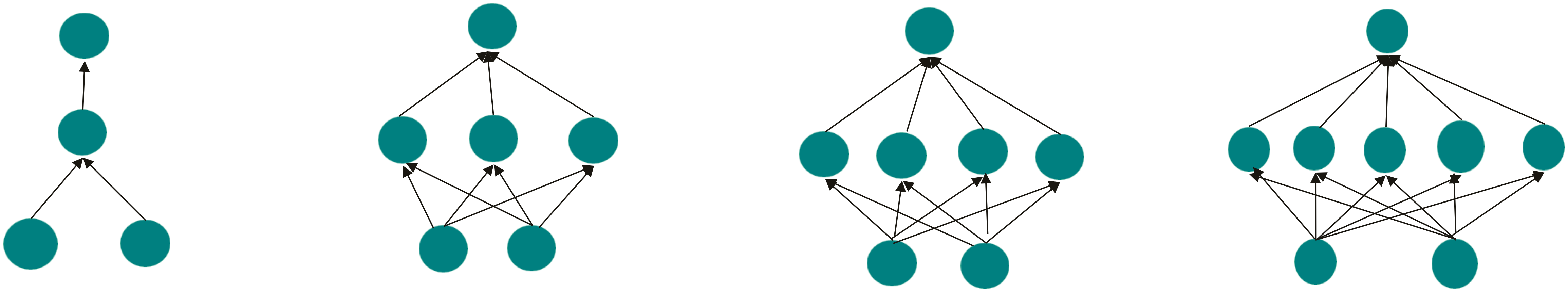
Case Study

What layers and neurons do in MLP?

Code Available

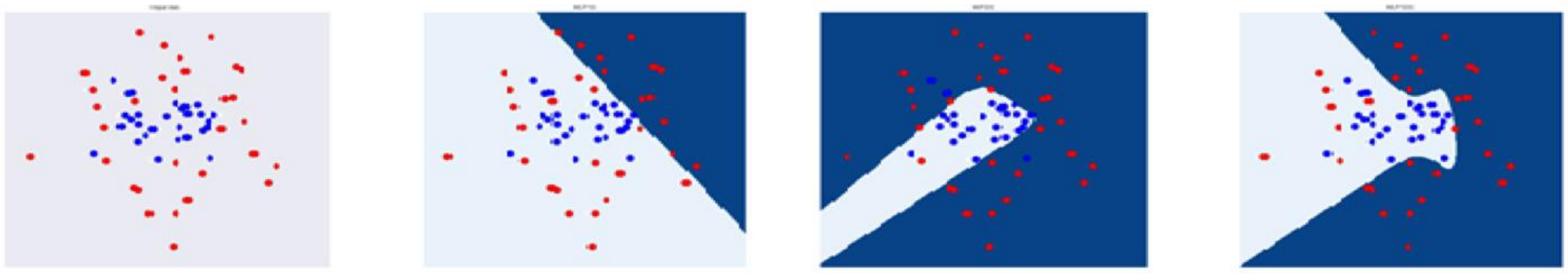
What do neurons do?



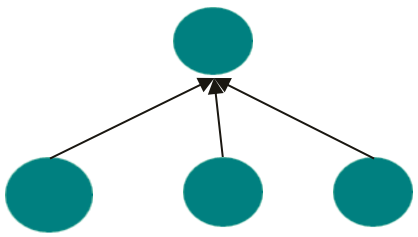


Code Available

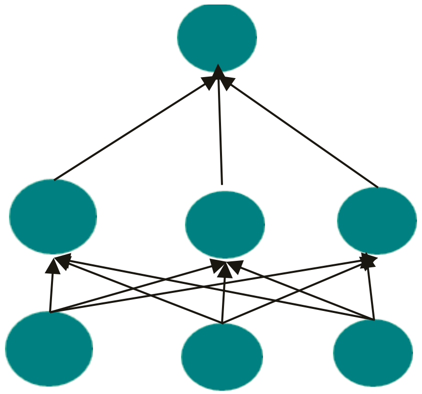
What do layers do?



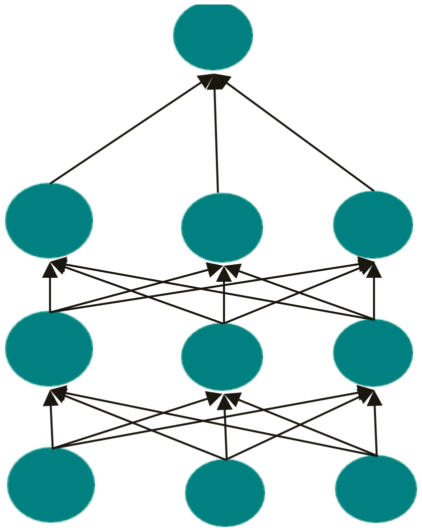
1st layer draws linear boundaries



2nd layer combines the boundaries



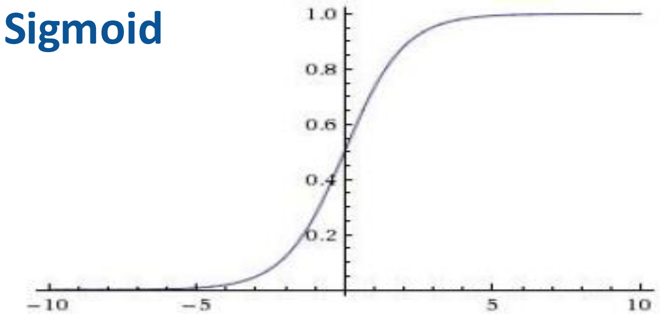
3rd layer can generate arbitrarily complex boundaries



Questions?

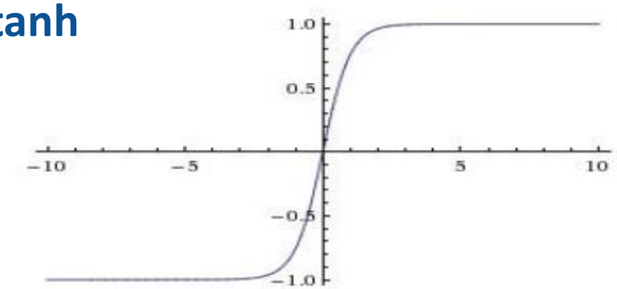
Activation Functions/Nonlinearities

Sigmoid



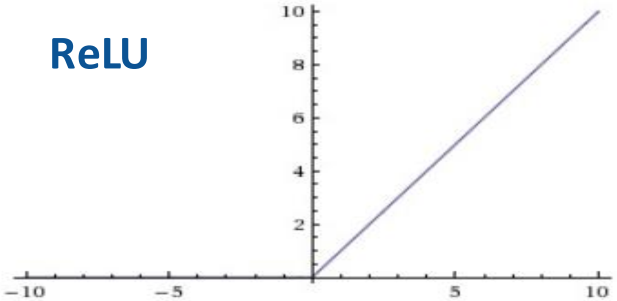
$$y = \frac{1}{1 + e^{-x}}$$

tanh



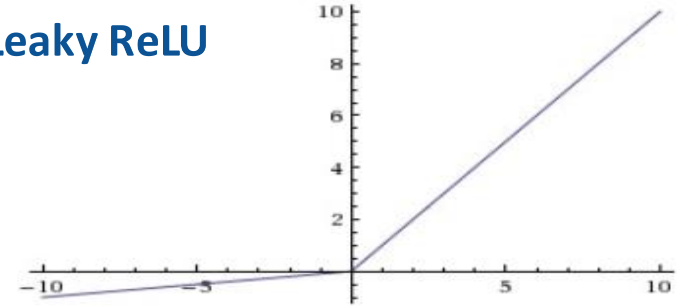
$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU



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

Leaky ReLU

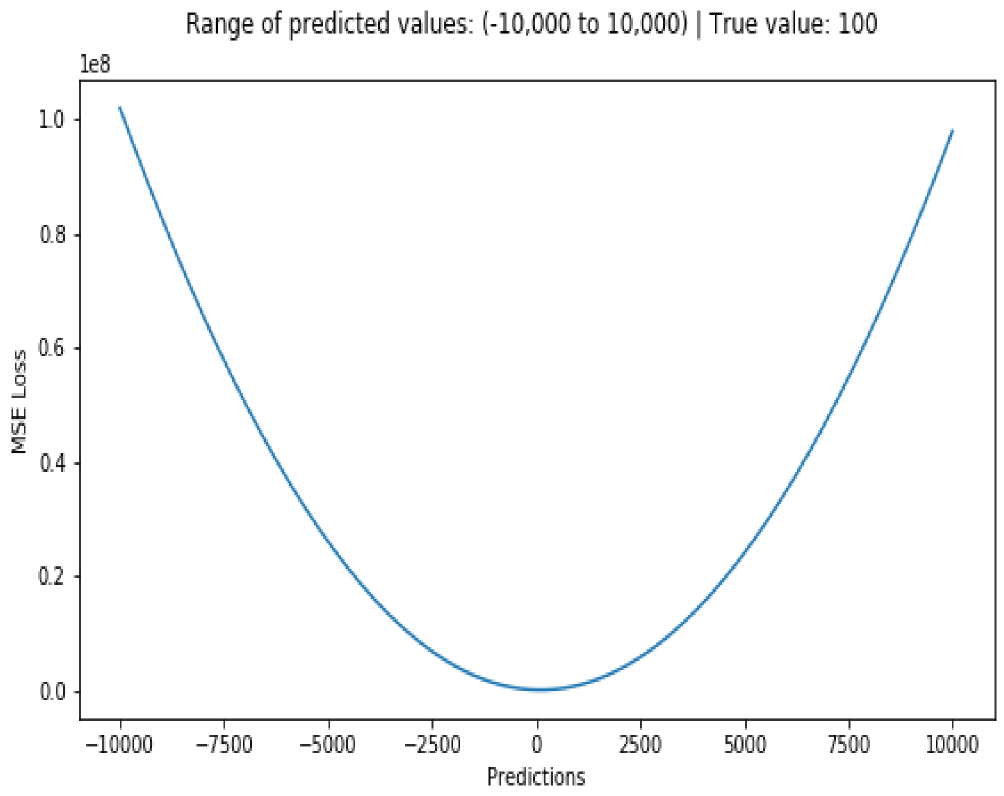


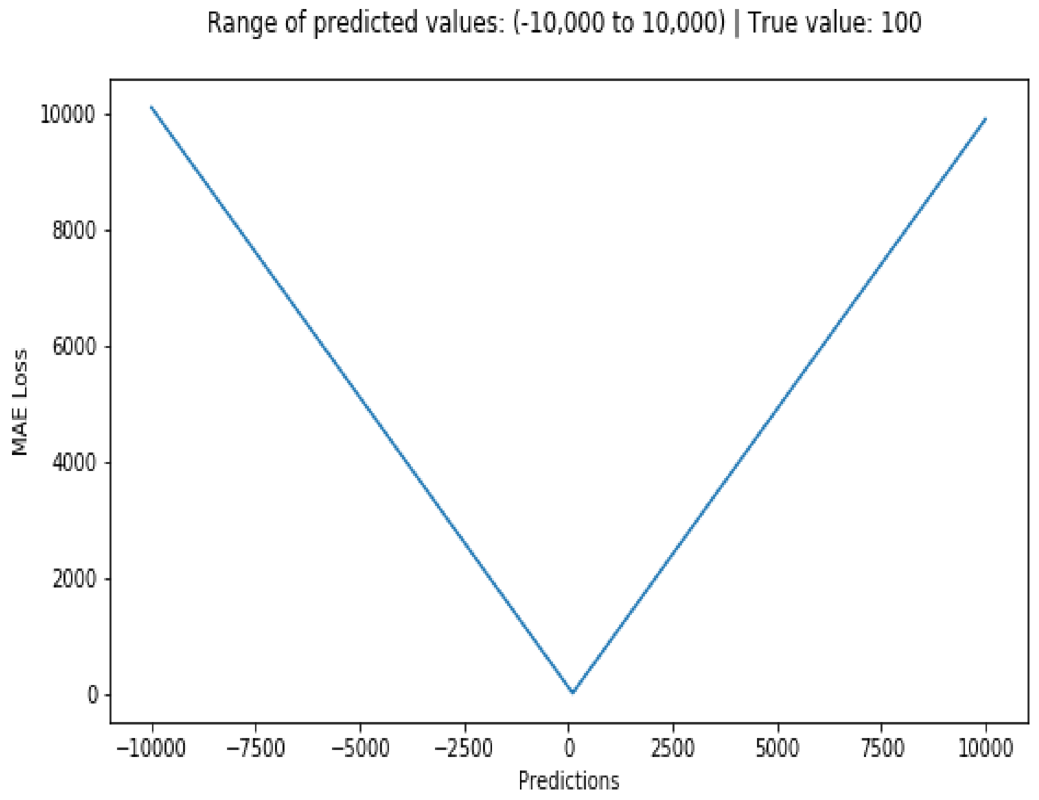

```
          x            if x > 0
y = {     0.01x  otherwise
```

maxout

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

Loss Functions





Mean Squared Loss

$$L(y, y') = (y - y')^2$$

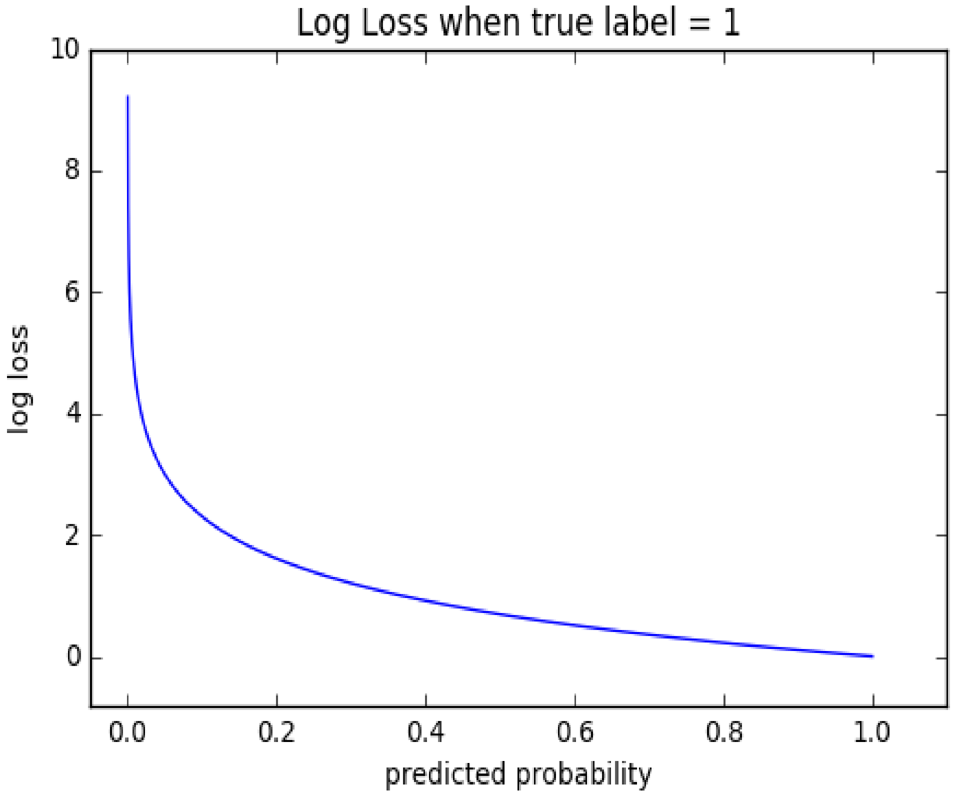
y - Actual value

y' - Predicted value

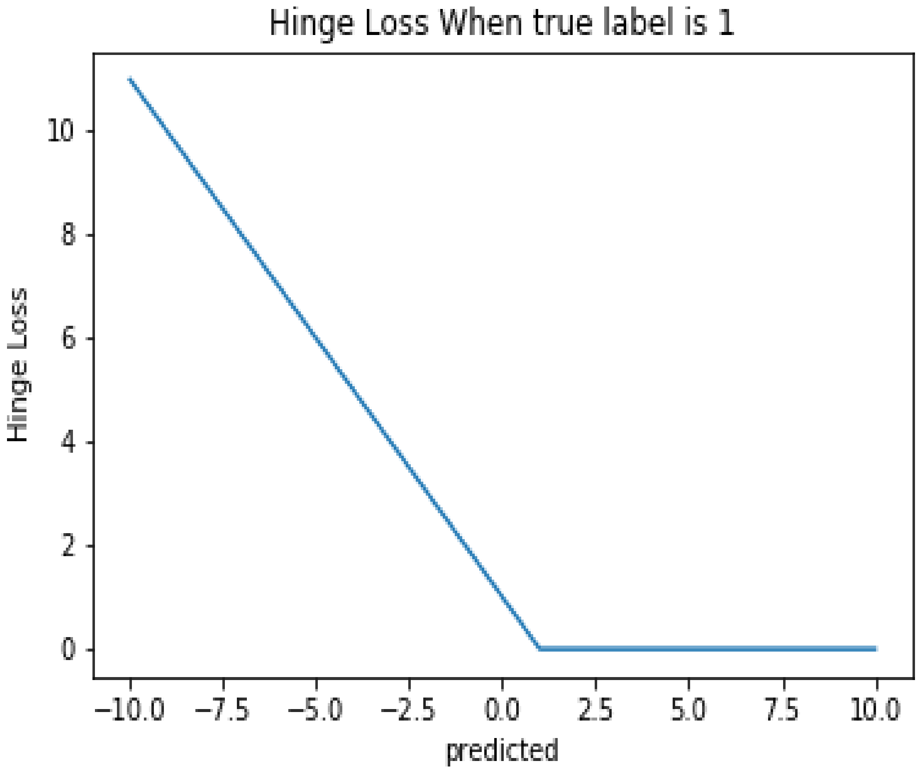
Mean Absolute Loss

$$L(y, y') = |y - y'|$$

Loss Functions



Cross Entropy Loss



Hinge Loss

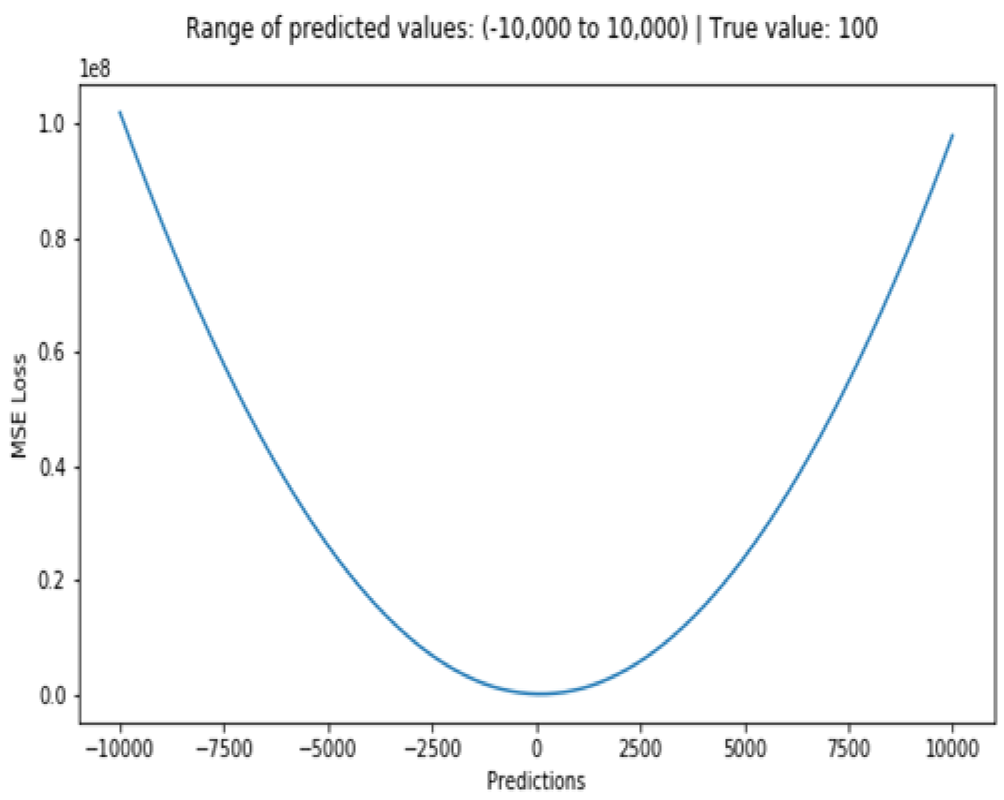
y - Actual value

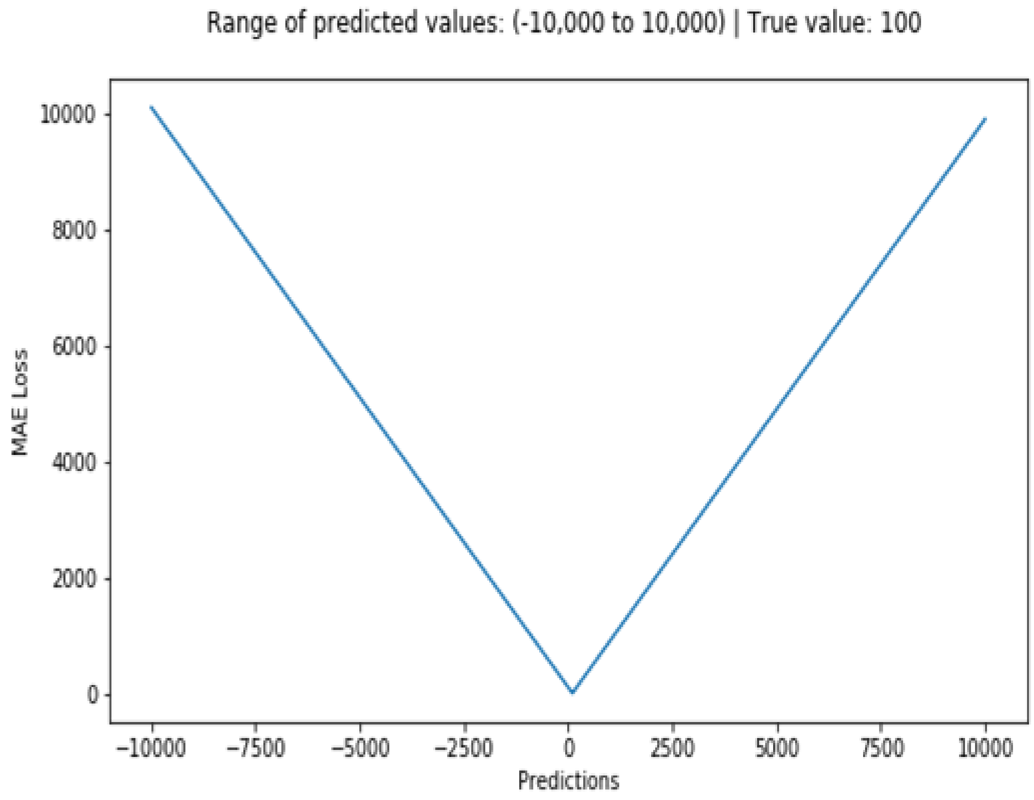
y' - Predicted value

$$L(y, y') = -(y\log(y') + (1 - y)\log(1 - y'))$$

$$L(y, y') = max(0, 1 - y * y')$$

Loss Functions (Regression)





Mean Squared Loss

$$L(y, y') = (y - y')^2$$

y - Actual value

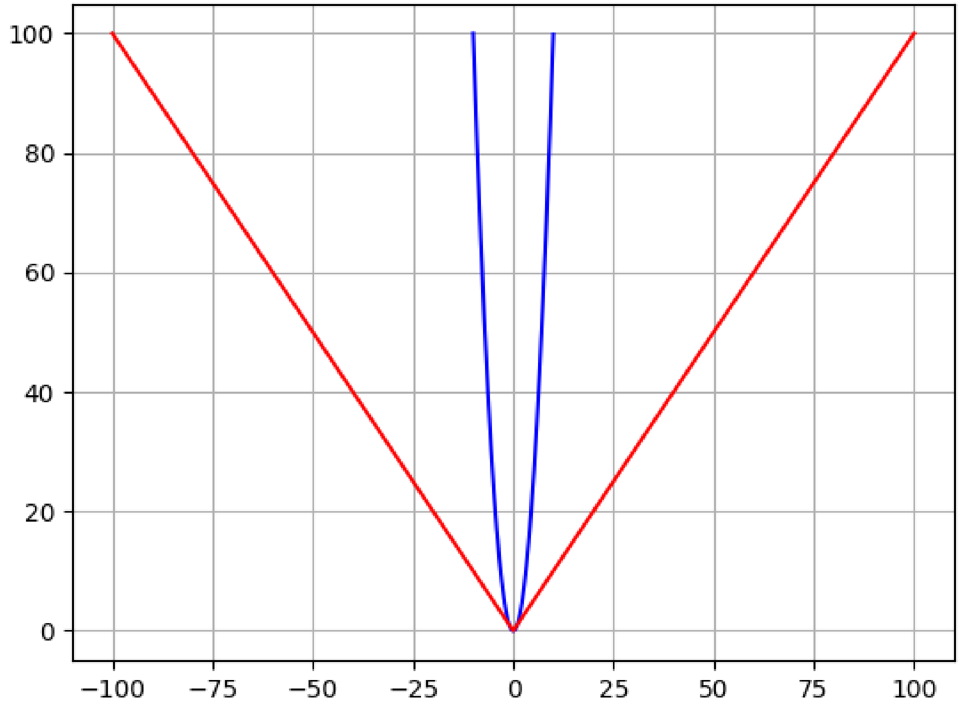
y' - Predicted value

Mean Absolute Loss

$$L(y, y') = |y - y'|$$

Loss Functions (Regression)

Mean Squared Loss (Blue)



Mean Absolute Loss (Red)

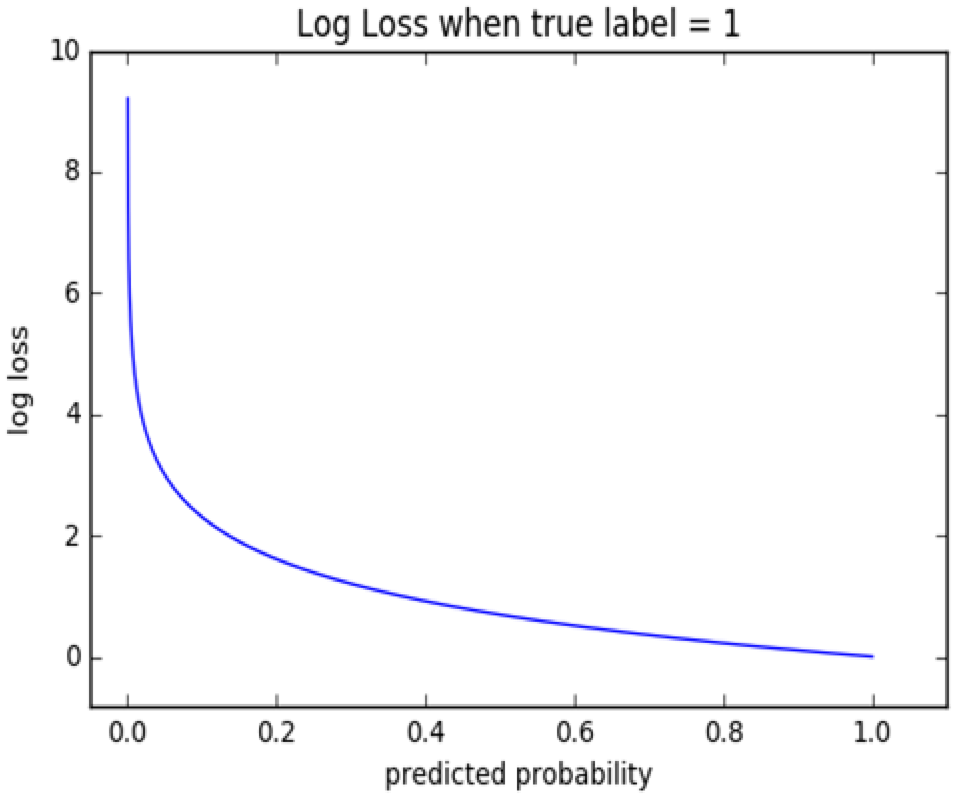
y - Actual value

y' - Predicted value

$$L(y, y') = (y - y')^2$$

$$L(y, y') = |y - y'|$$

Loss Functions (Classification)



Cross Entropy Loss

$$L(y, y') = -\left(y\,log(y') + (1-y)\,log(1-y')\right)$$

y' - Actual value

y - Predicted value

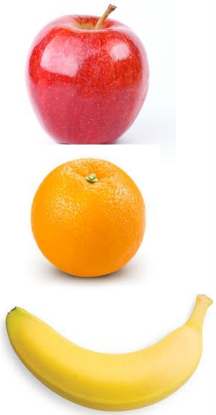
Multi Class Classification using MLP

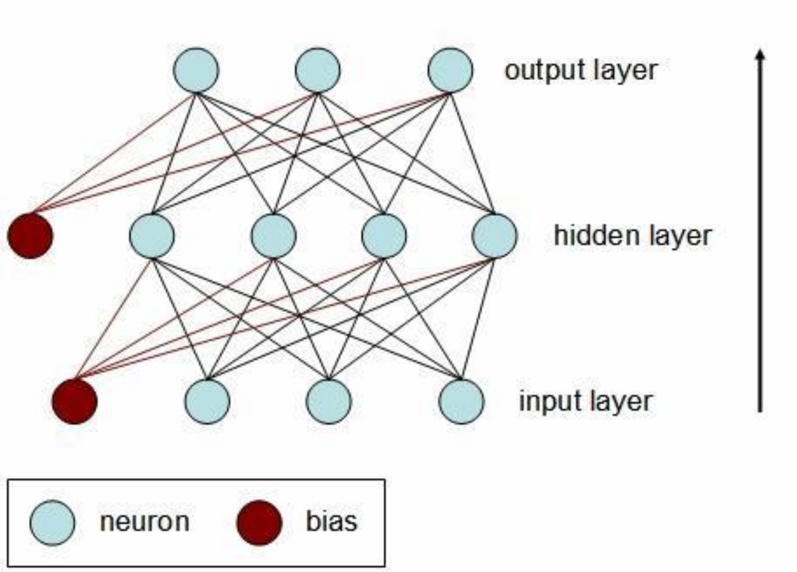
Input: (x_i, y_i)

$$x = [x_1, x_2, x_3, \ldots x_n]$$

Encode label y as

- [1,0,0] for class 1
- [0,1,0] for class 2
- [0,0,1] for class 3



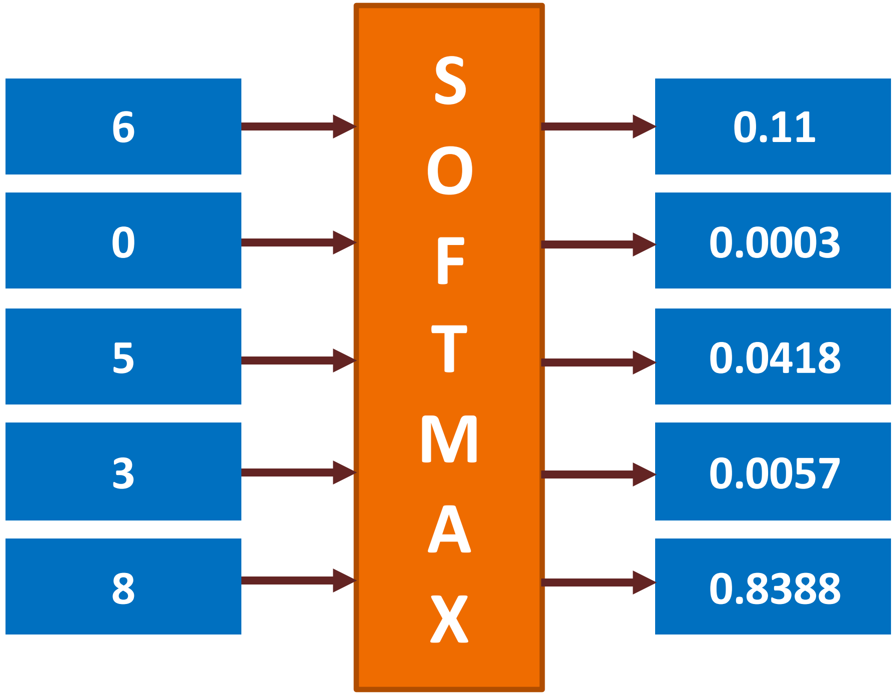


Softmax

```
Out[12]: array([ 6.,  0.,  5.,  3.,  8.])

In [8]:  exp = (np.e)**(x)
         exp

executed in 6ms, finished 01:47:23 2018-08-21

Out[8]:  array([  4.03428793e+02,   1.00000000e+00,   1.48413159e+02,
                  2.00855369e+01,   2.98095799e+03])

In [9]:  sigma_e = np.sum(exp)
         sigma_e

executed in 9ms, finished 01:47:25 2018-08-21

Out[9]:  3553.8854765602264

In [11]: z = exp/sigma_e
         z

executed in 8ms, finished 01:47:34 2018-08-21

Out[11]: array([  1.13517669e-01,   2.81382168e-04,   4.17608165e-02,
                  5.65171192e-03,   8.38788421e-01])
```

- Normalizes the output.
- K is total number of classes

$$z_n = \frac{e^{x_n}}{\sum_{i=1}^{K} e^{x_i}}$$



Multi Class Classification using MLP

Loss:

- MSE (Mean square error)
- Let predicted label be $y^{'}$.
- Remains the same even for regression.

Our objective:

- Minimize the difference between $y_i^{'}$ and y_i for all i

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

Four Cases

Perceptron

$$\mathbf{w}^T\mathbf{x}$$

Linear Features
Linear Classifiers

$$\phi(\mathbf{w})^T\phi(\mathbf{x})$$

Nonlinear Features
Linear Classifiers

Kernel-SVM

MLP

$$\psi(\mathbf{w},\mathbf{x})$$

Linear Features
Nonlinear Classifiers

$$\psi(\mathbf{w},\phi(\mathbf{x}))$$

Nonlinear Features
Nonlinear Classifiers

MLP of VGG Features

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 functions.
- Often people use MLP with one or two hidden layers
  – Not very deep.

Thanks!!

Questions?