Google Classroom Code: mhxgl24

Basics of Neural Networks

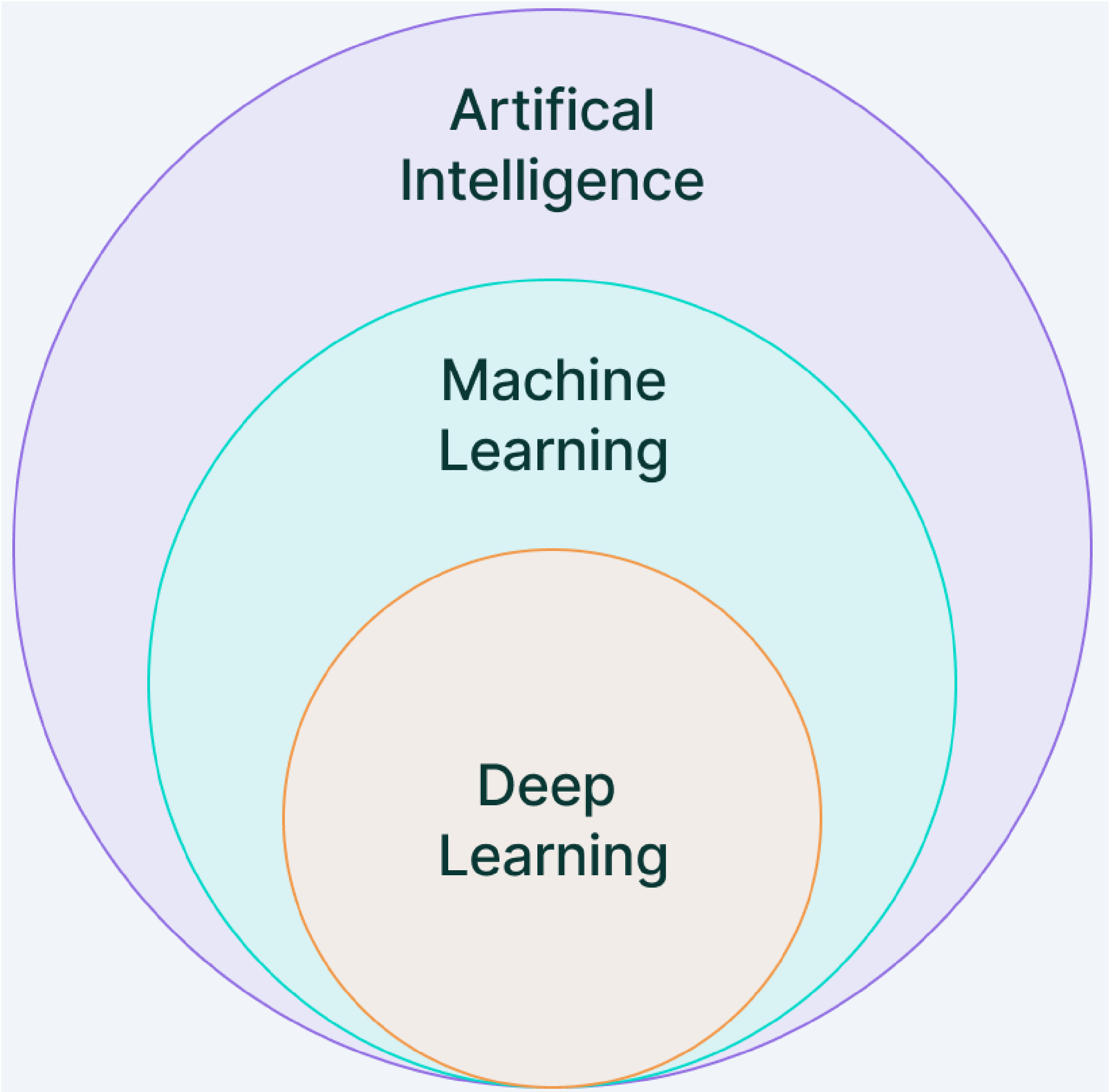
Deep Learning (DS-5006)
Dr. Adeel Mumtaz
Lecture 3
Fall, 2022

National University
Of Computer and Emerging Sciences

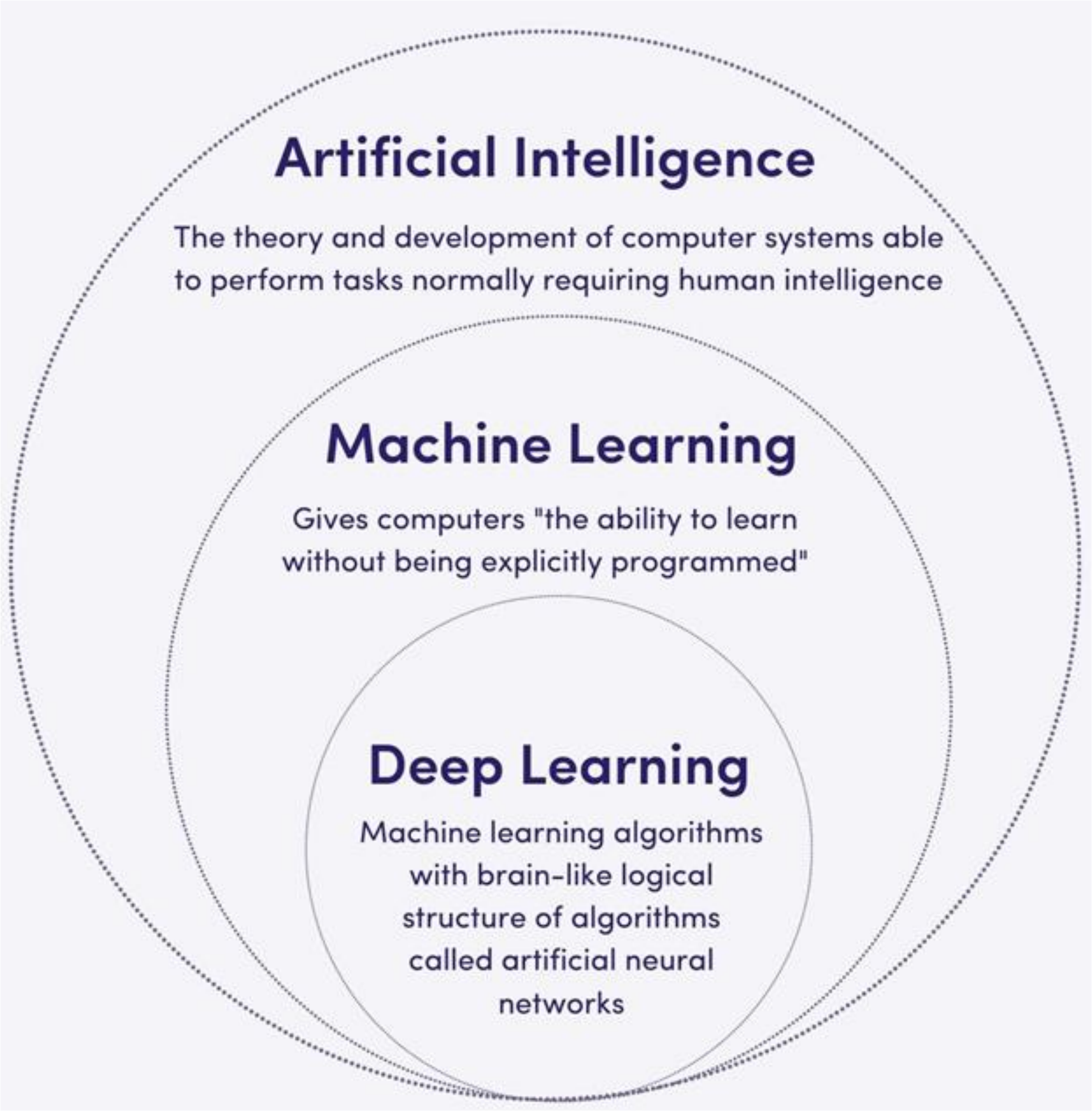
Contents

BASICS OF NEURAL NETWORKS

Deep Learning



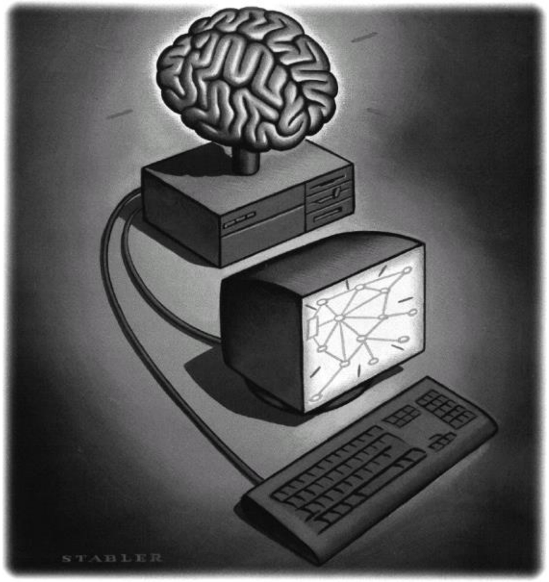
Deep Learning



- **The term Deep Learning refers to training very large Neural Network**

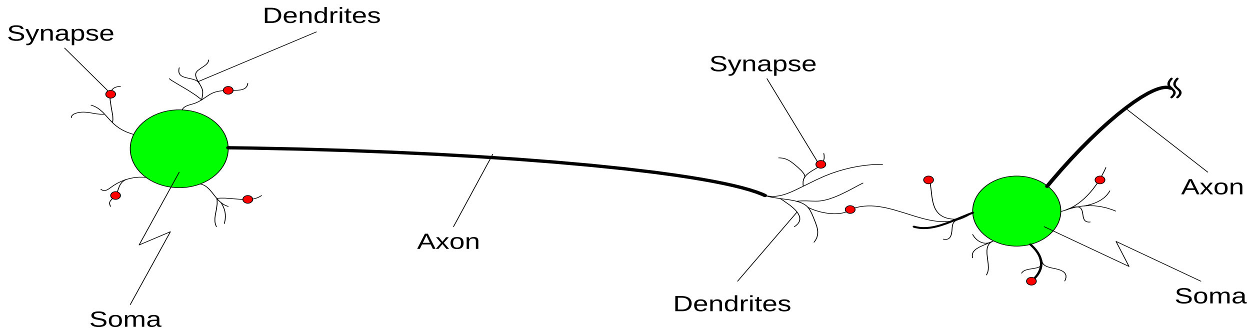
What is a Neural Network

- Biologically motivated approach to machine learning
- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial **neurons**
- Can be used for
  - Classification
  - Regression
  - Clustering
  - Association
  - Optimization



Biological Neural Networks

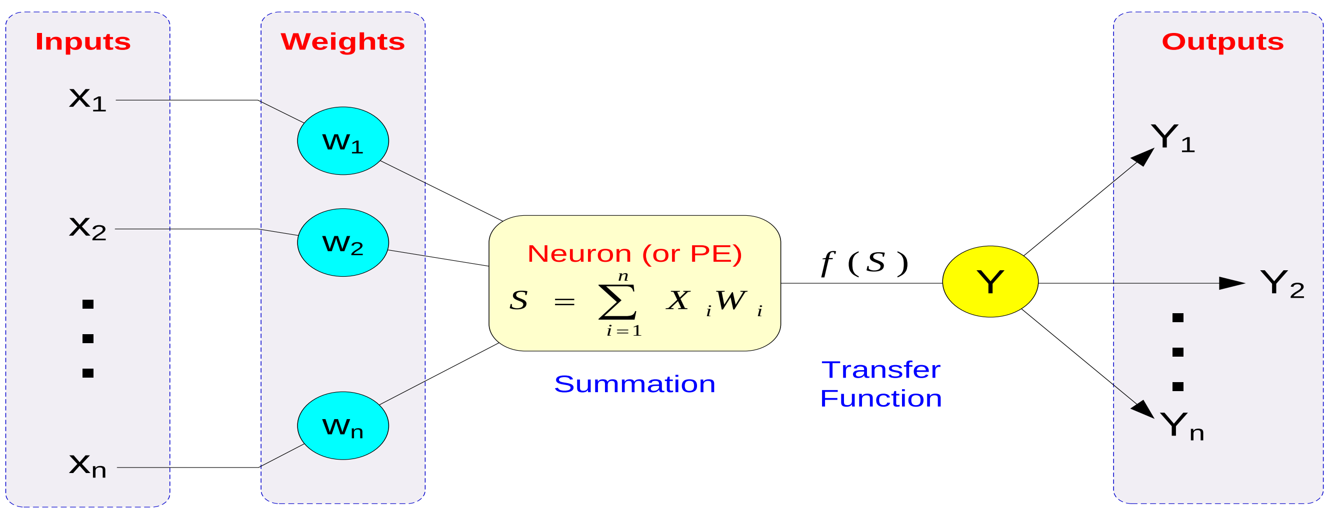
- Fundamental processing element of a neural network is a ***neuron***
  - Dendrites, accept inputs
  - Soma, process the inputs
  - Axon, turns the processed inputs into outputs
  - Synapses, the electrochemical contacts between neurons
- A human brain has 100 billion neurons
- An ant brain has 250,000 neurons



Two interconnected brain cells (neurons)

Artificial Neural Networks

- A single neuron (processing element – PE) with inputs and outputs
  - – Inputs
  - – Weights
  - – Summation S or V
  - – Transfer function or Activation function, φ *or* f



Comparison between ANN & BNN

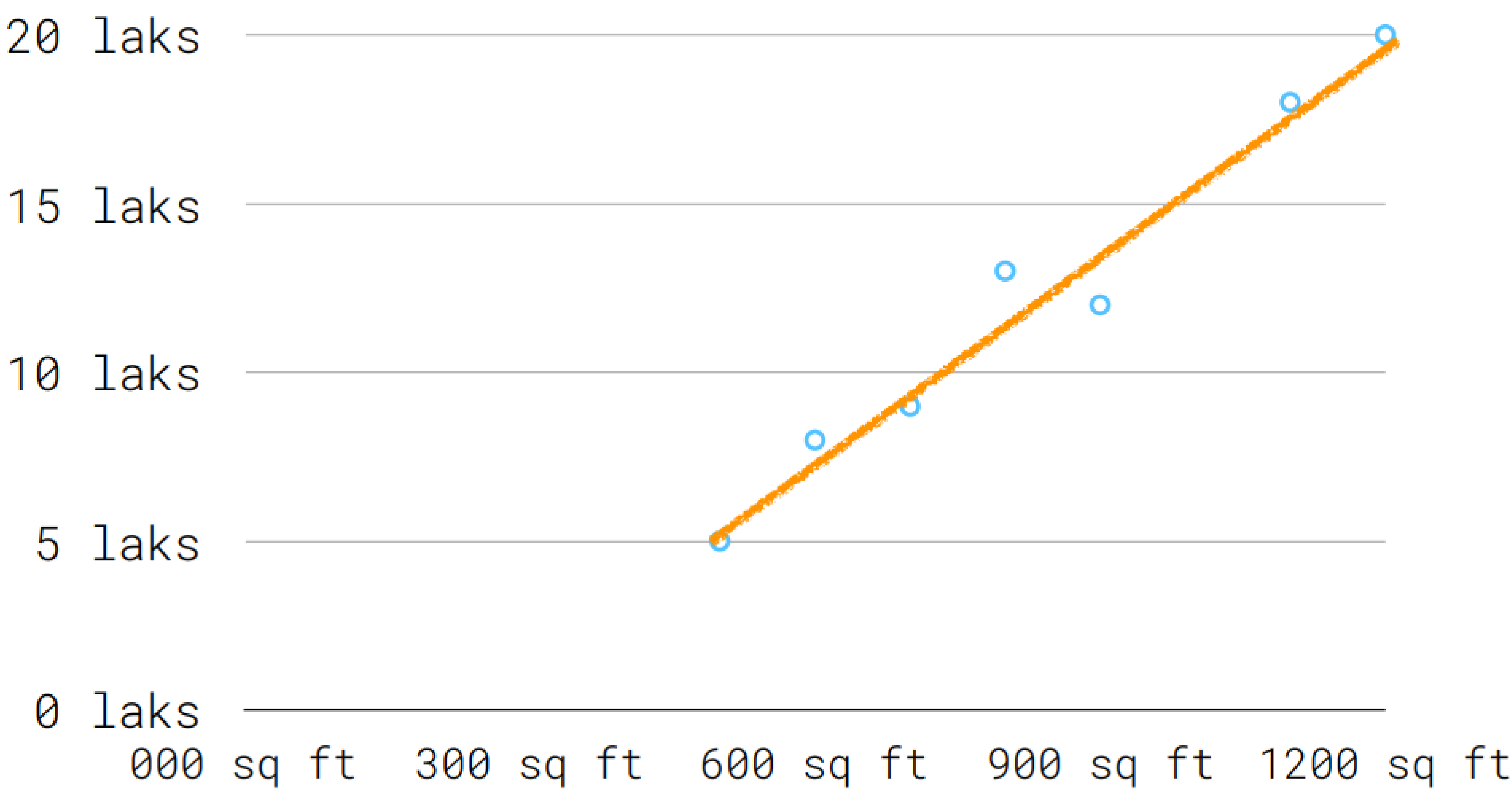
Biological versus	Artificial NNs
Soma	Node
Dendrites	Input
Axon	Output
Synapse	Weight
Slow	Fast
Many neurons (10^9)	Few neurons (~100s)

An Intuitive example (housing price prediction)

- Fit a function to predict the price of the house as a function of the size
  - linear regression?

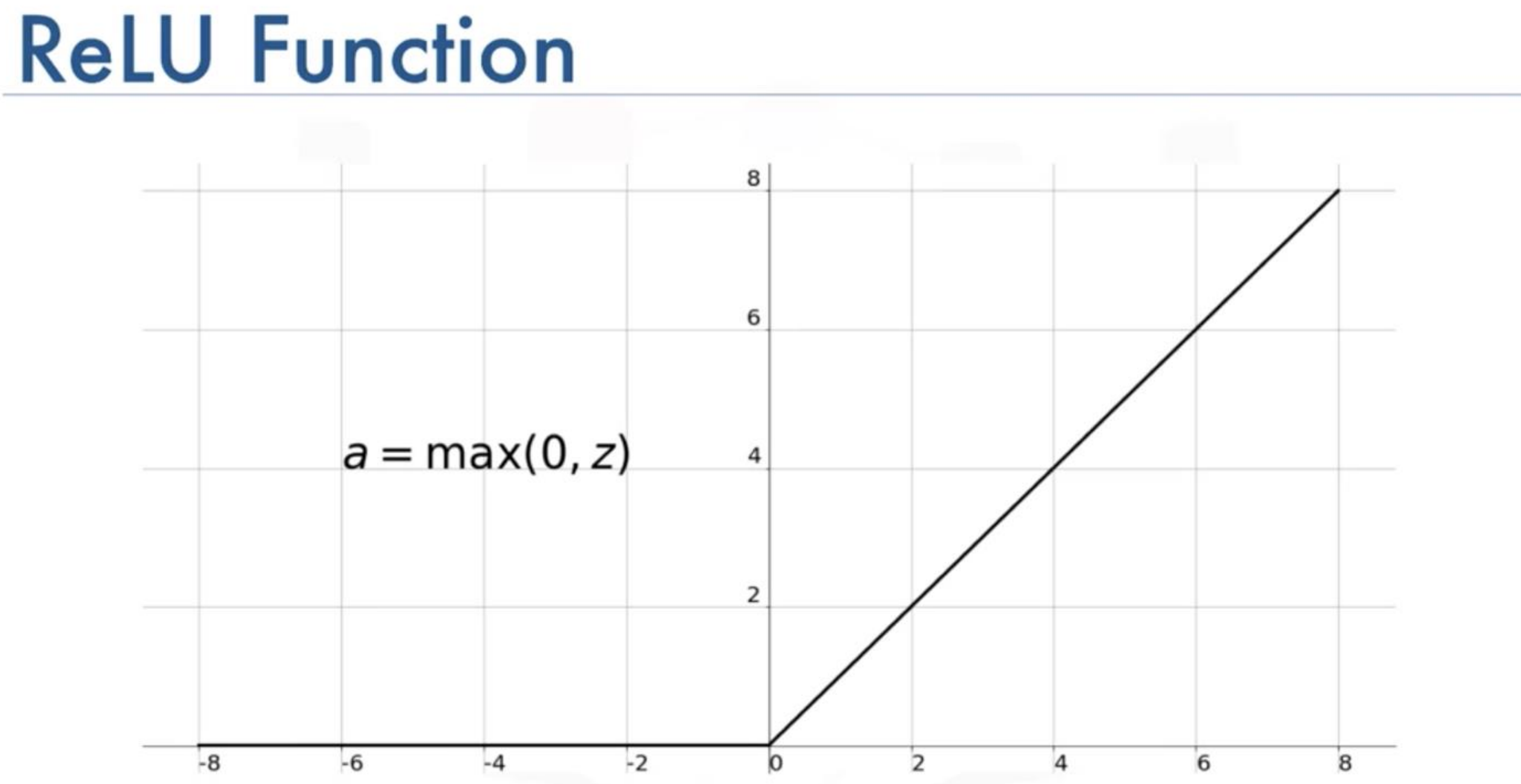
House Size (X)	Price (Y)
500	5 Laks
600	8 Laks
700	9 Laks
800	13 Laks
900	12 Laks
1100	18 Laks
1200	20 Laks

Housing Price Prediction



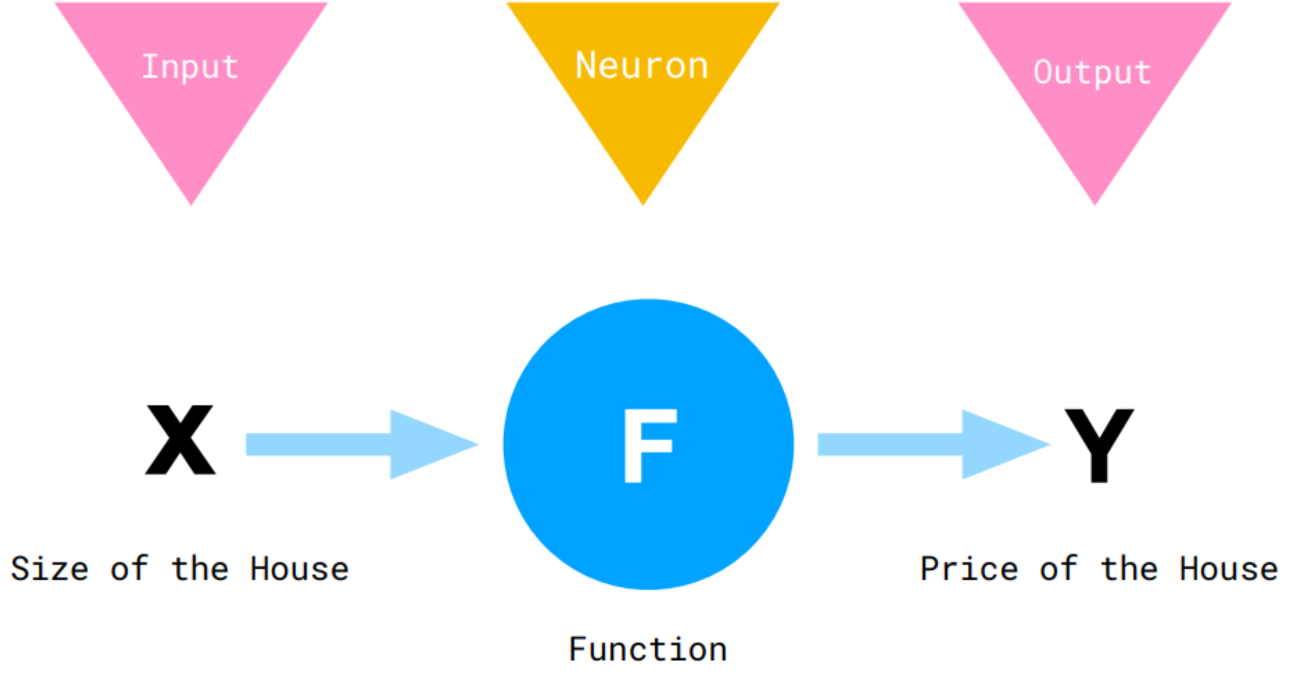
An Intuitive example (housing price prediction)

- Price can’t be negative



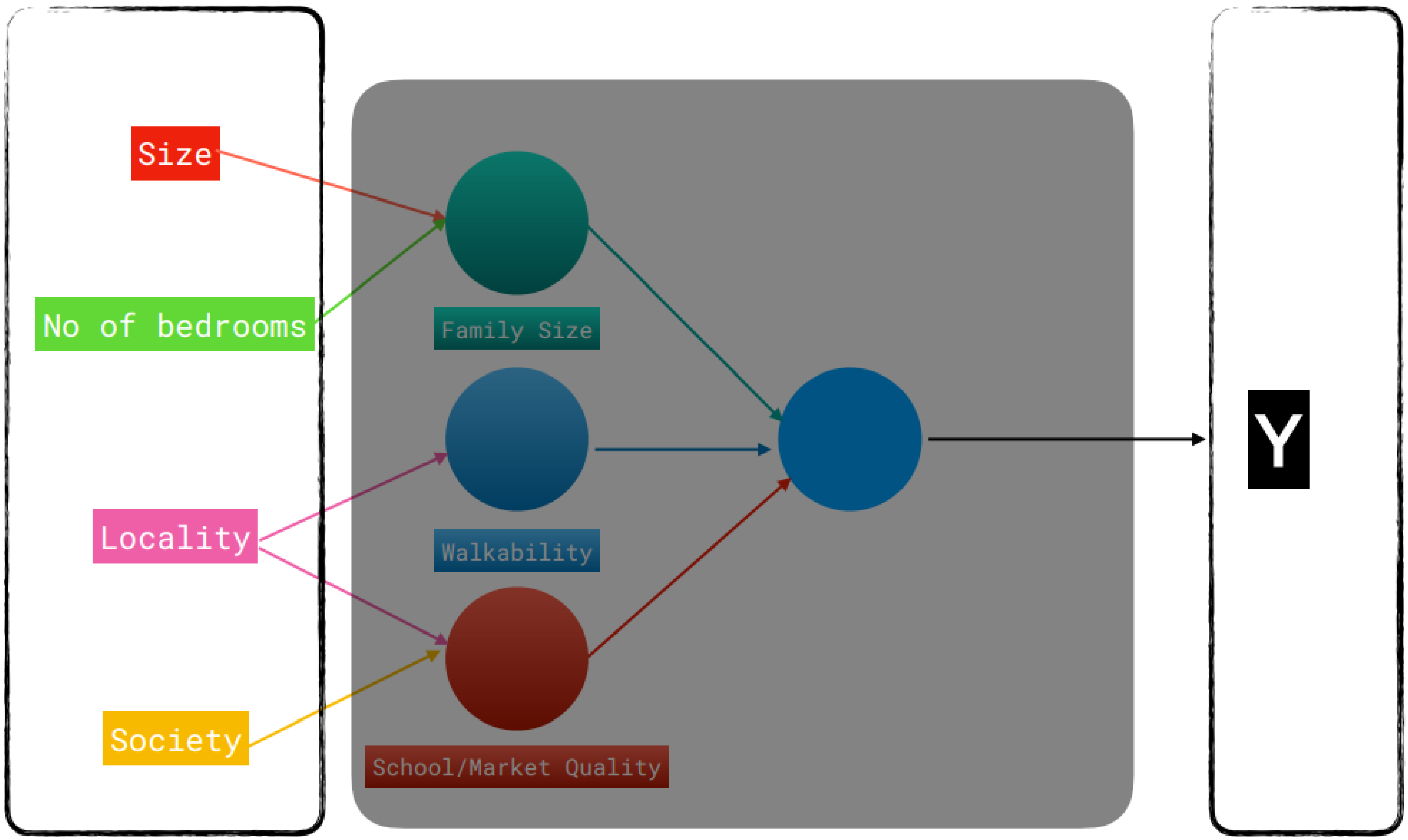
An Intuitive example (housing price prediction)

- a very simple neural network
- Neuron takes inputs the size, computes the linear function, and then outputs the estimated price.



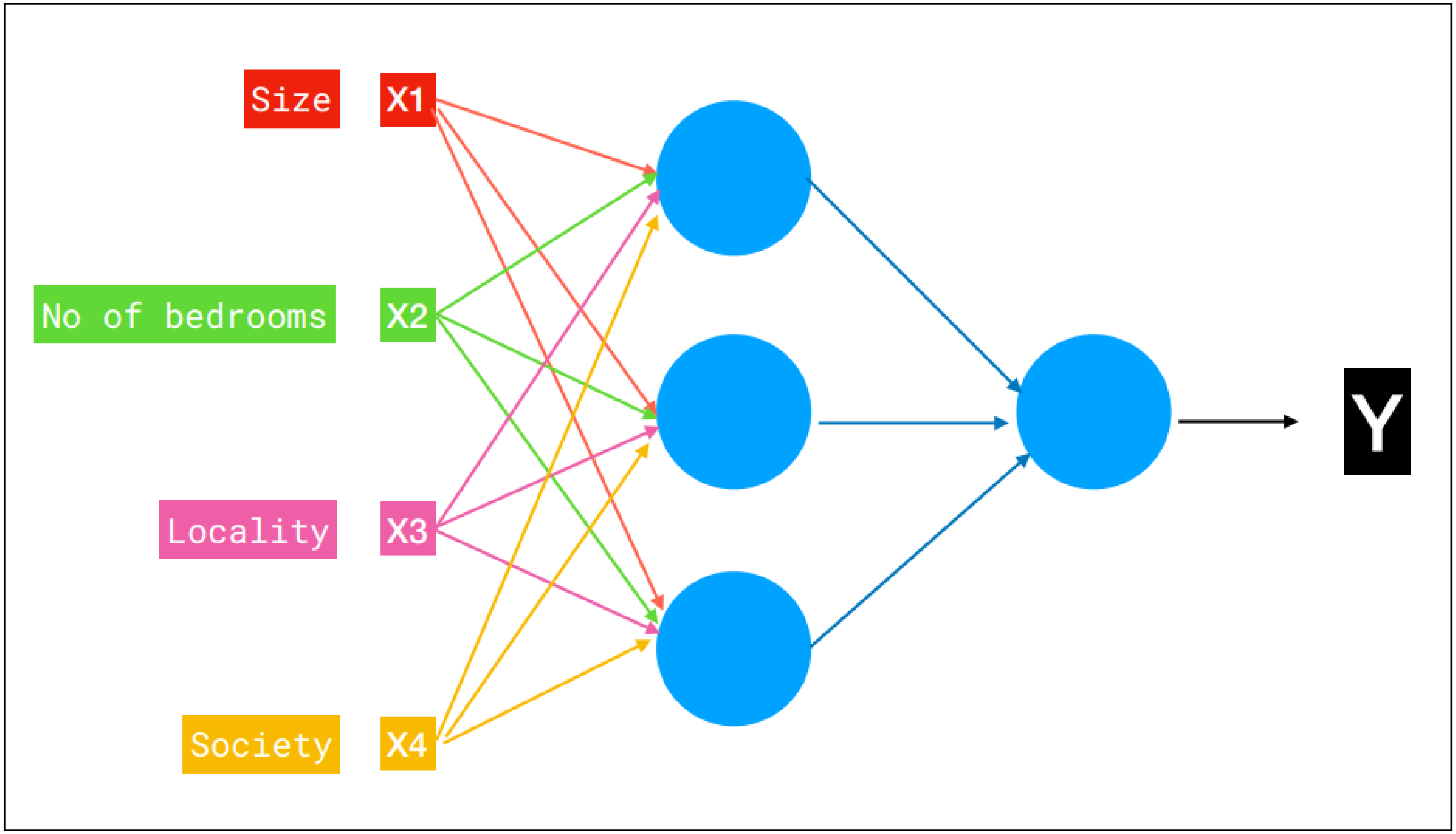
An Intuitive example (housing price prediction)

- Other features that can affect the Price



An Intuitive example (housing price prediction)

- A neural network with four inputs and one output
- One hidden layer with three hidden units
- Note that each of the hidden unit takes its inputs of all four features (Dense, Linear, Fully connected)

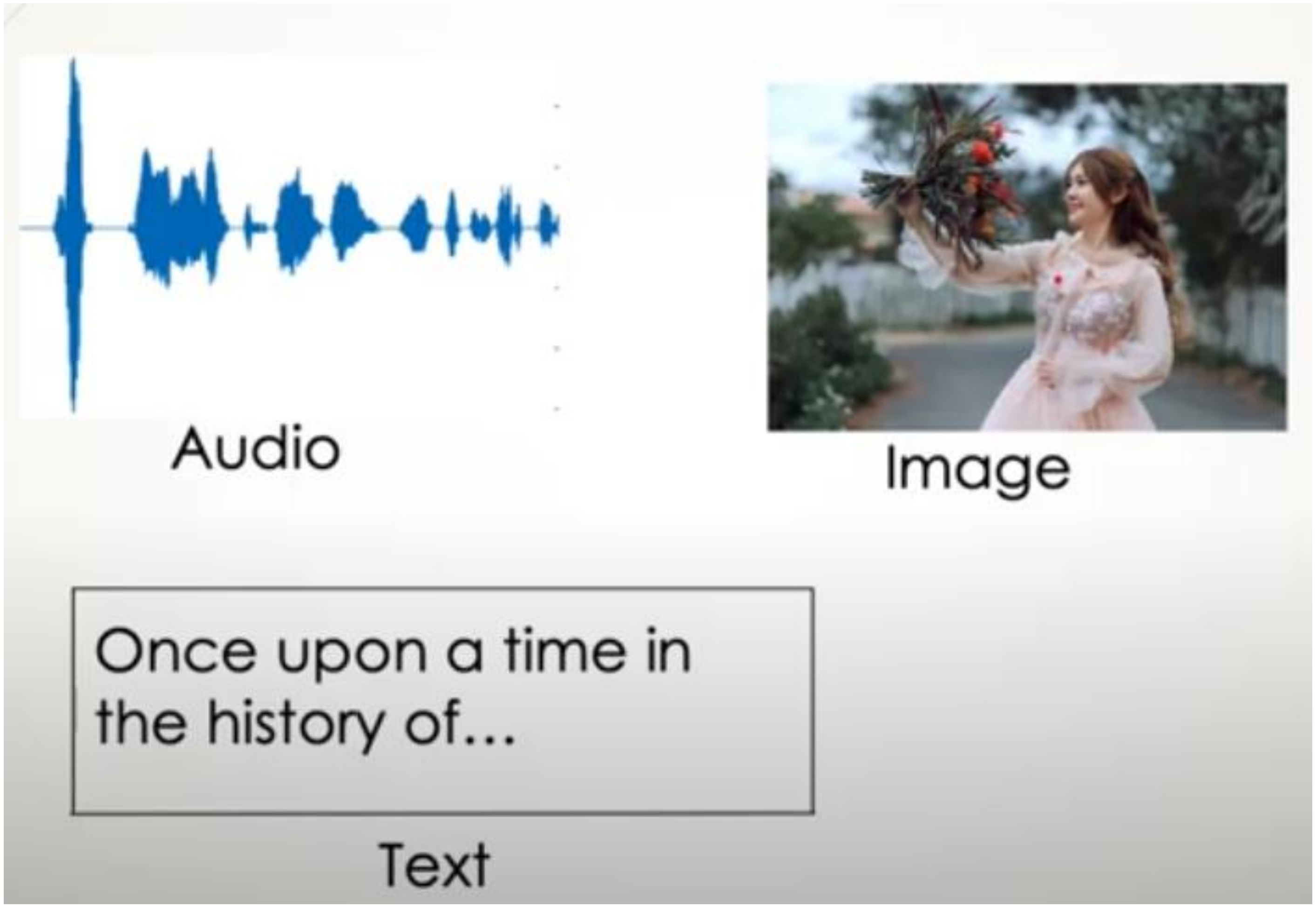


Structured Data

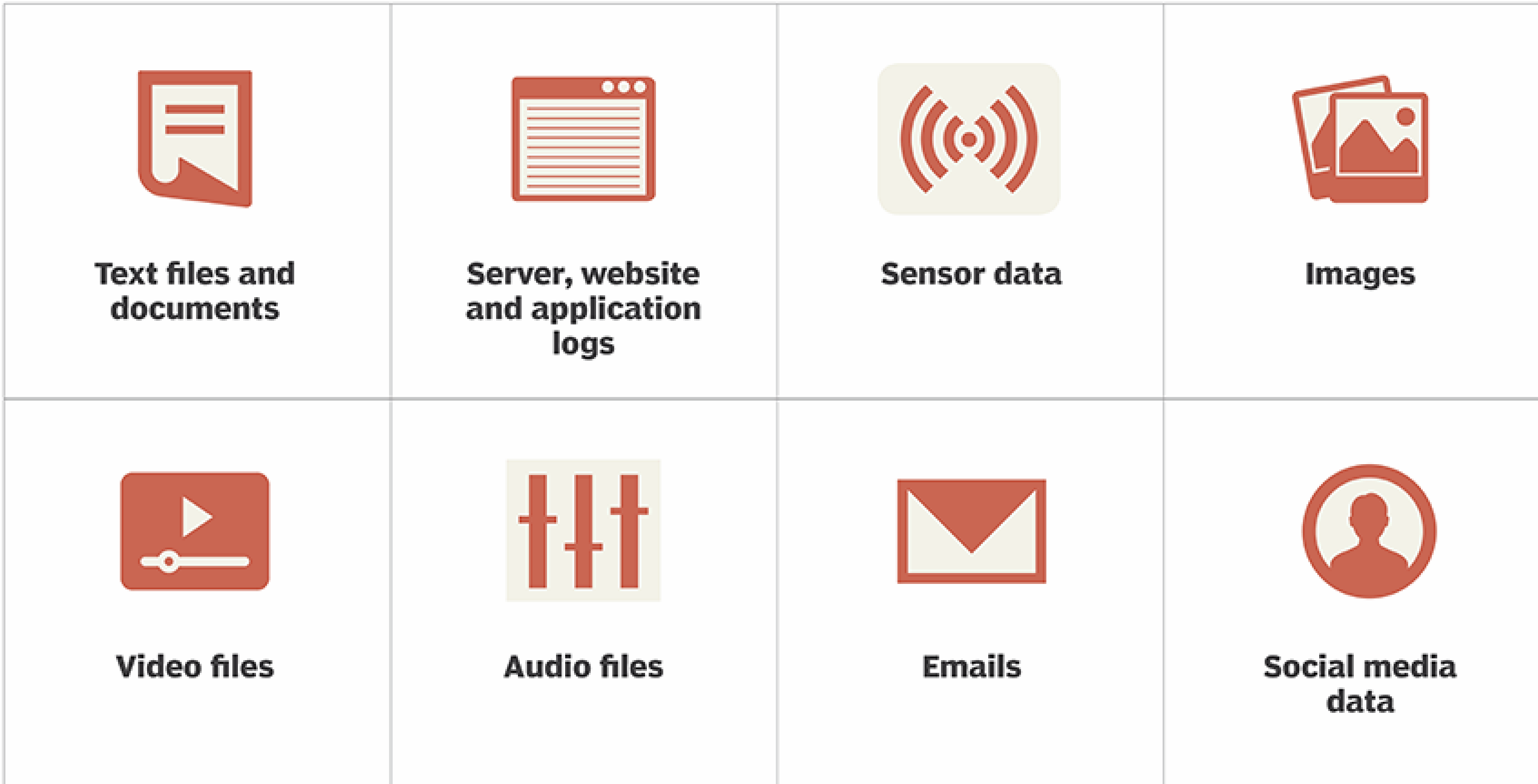
Size	#bedrooms	...	Price (1000$s)
1200	2		300
1500	3		400
2000	3		480
⋮	⋮		⋮
3000	4		520

User Region	Ad Id	...	Ad revenue($)
USA	1005		0.5
UK	1009		0.3
USA	998		0.5
⋮	⋮		⋮
CAN	2104		0.45

Unstructured Data

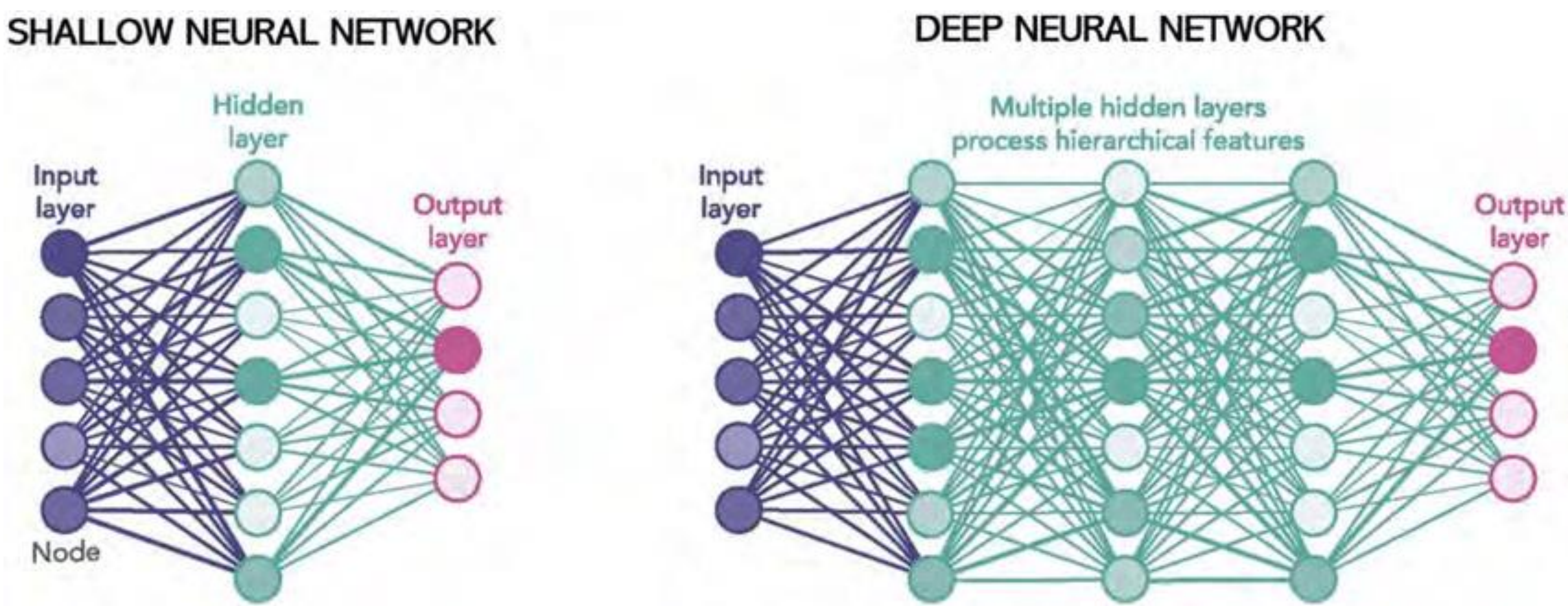


Unstructured Data



Deep Neural Networks are very good at processing these Unstructured data

Shallow Vs Deep Neural Networks

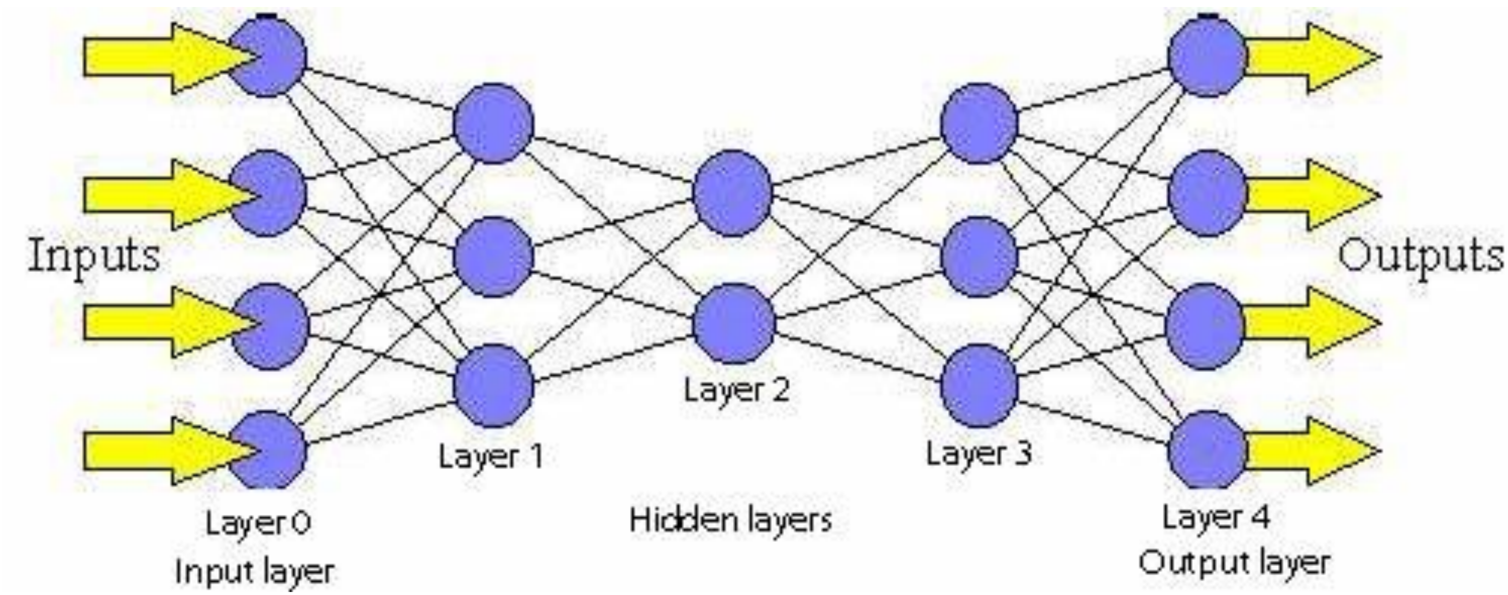


Shallow Vs Deep Neural Networks

Factors	Shallow Neural Network (SNN)	Deep Neural Network (DNN)
Feature Engineering	1. Individual feature extraction process is required. Various features cited in the literature are histogram oriented gradients, speeded up robust features, and local binary patterns.	1. Replace the hand-crafted features and directly work on the entire input. Thus, more practical for complex datasets.
Data Size Dependency	2. Needs a lesser quantity of data.	2. Needs vast volumes of data.

Deep Learning Architectures

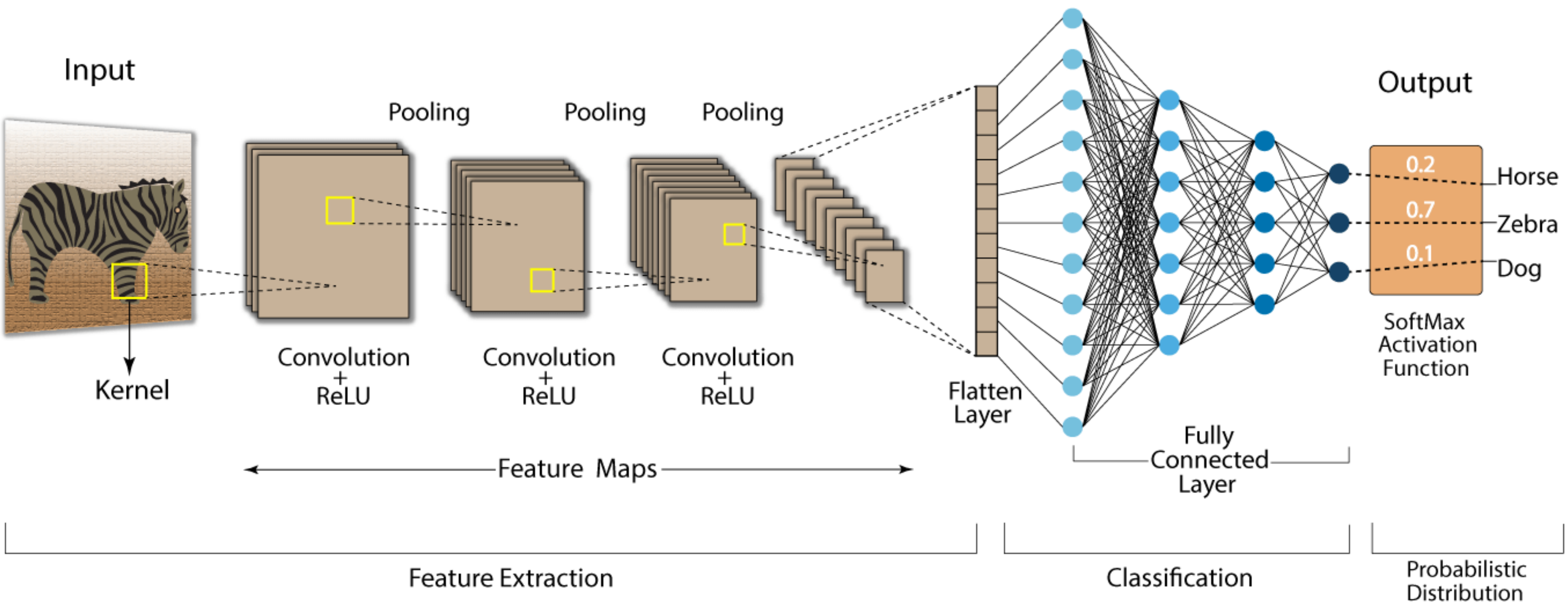
- Universal Function Approximators
- The number of trainable parameters increases drastically with an increase in the size of the image



Artificial Neural Network (ANN or MLP)

Deep Learning Architectures

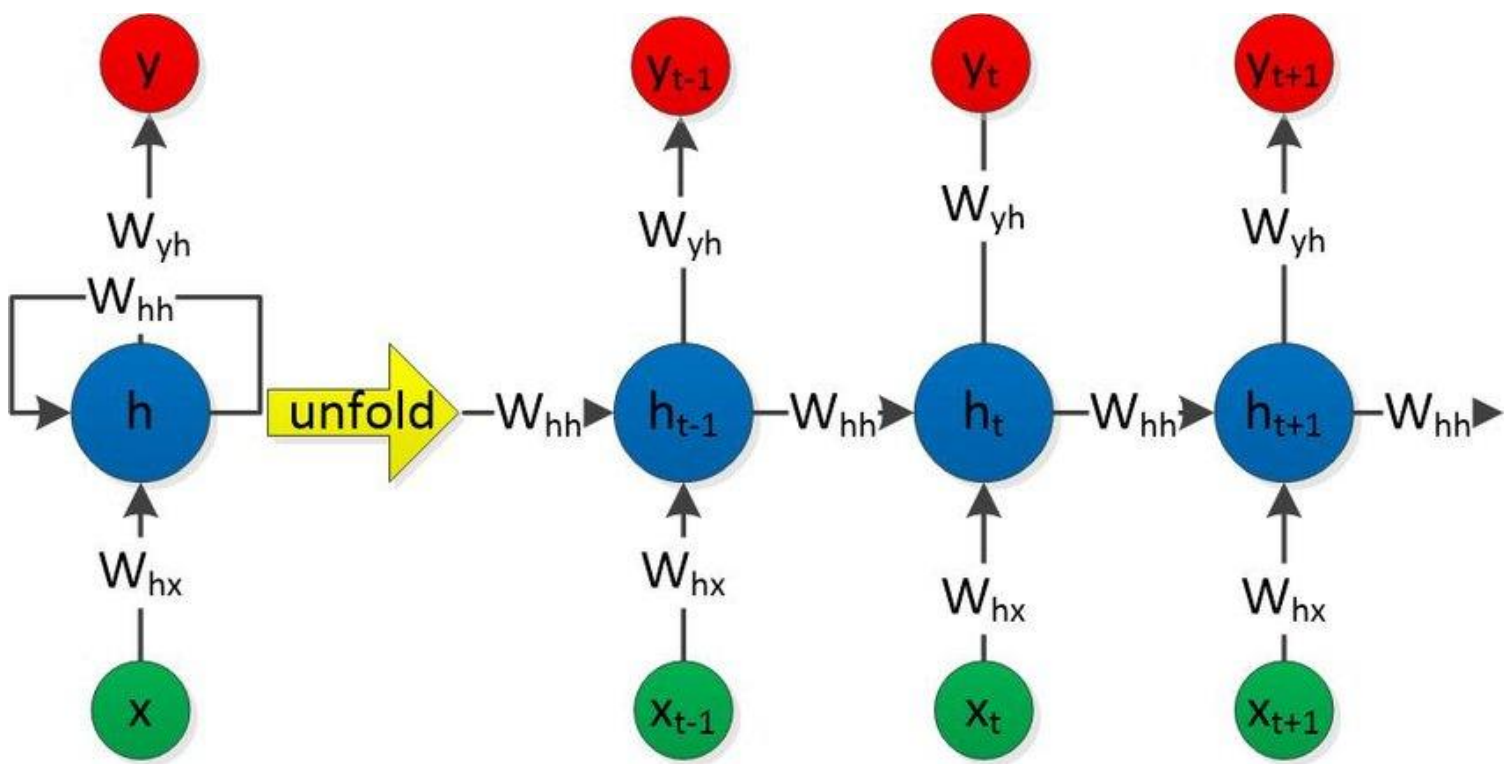
- CNN learns the filters automatically
- CNN captures the spatial features from an image, identify object location accurately
- CNN follows the concept of parameter sharing



Convolutional Neural Network (CNN)

Deep Learning Architectures

- RNN captures the sequential information present in the input data
- RNNs share the parameters across different time steps

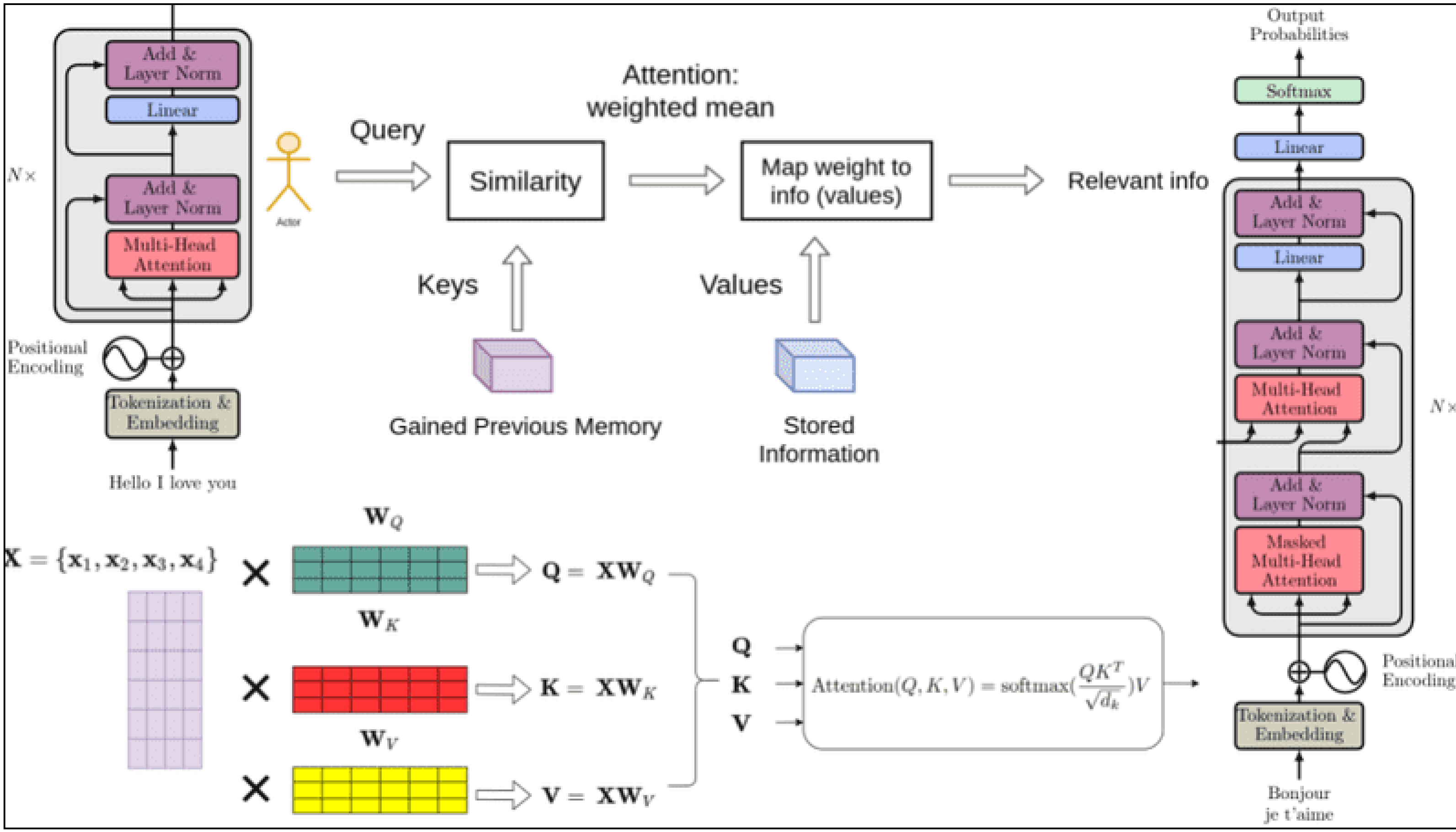


Recurrent Neural Network (RNN)

Deep Learning Architectures

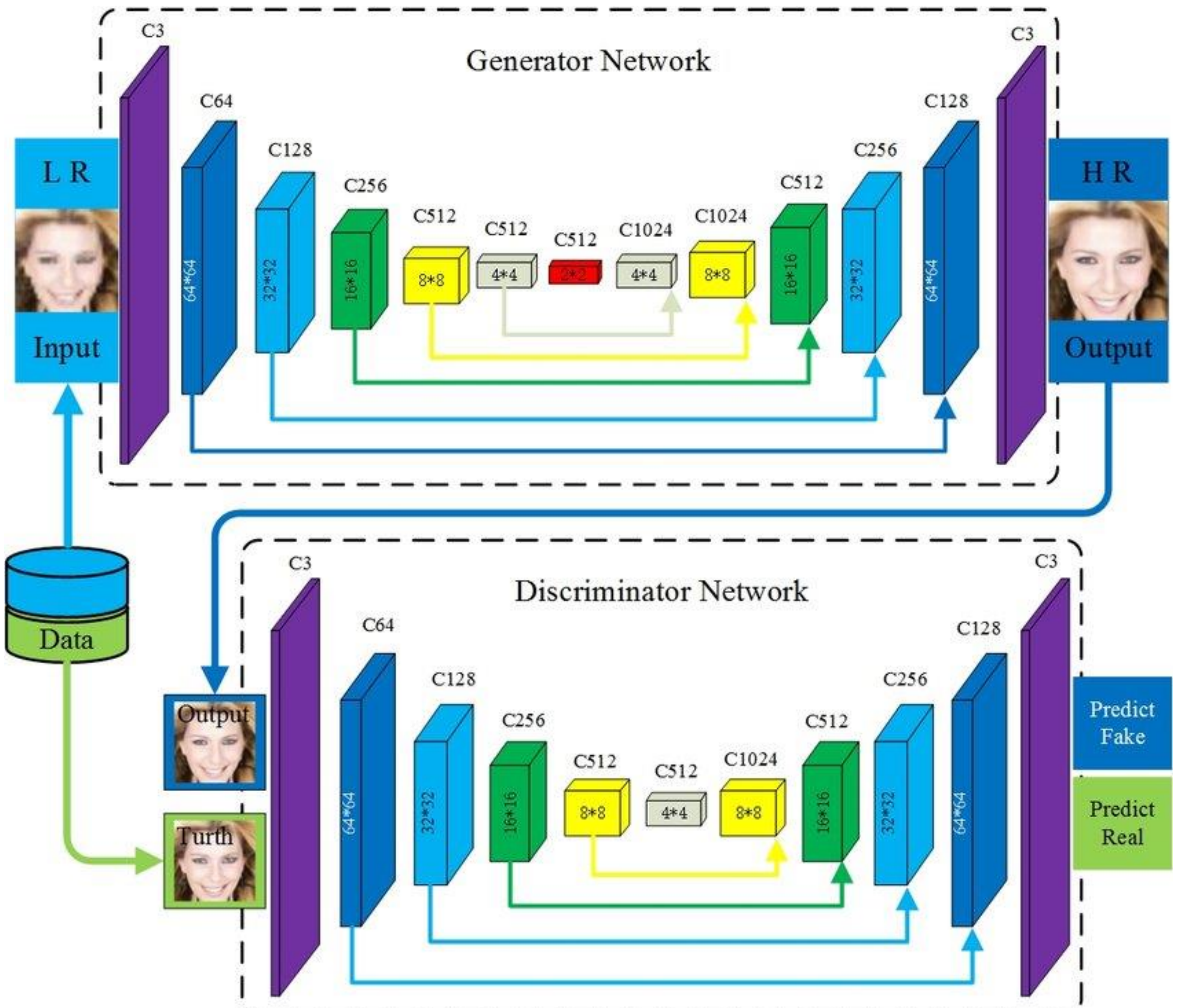
	MLP	RNN	CNN
Data	Tabular data	Sequence data (Time Series,Text, Audio)	Image data
Recurrent connections	No	Yes	No
Parameter sharing	No	Yes	Yes
Spatial relationship	No	No	Yes
Vanishing & Exploding Gradient	Yes	Yes	Yes

Deep Learning Architectures



Attention Based Neural Network (Transformer)

Deep Learning Architectures



Generative Adversarial Network (GAN)

A SIMPLE REGRESSION PROBLEM (THEORY)

Simple Linear Regression

$$\hat{y} = b + wX$$

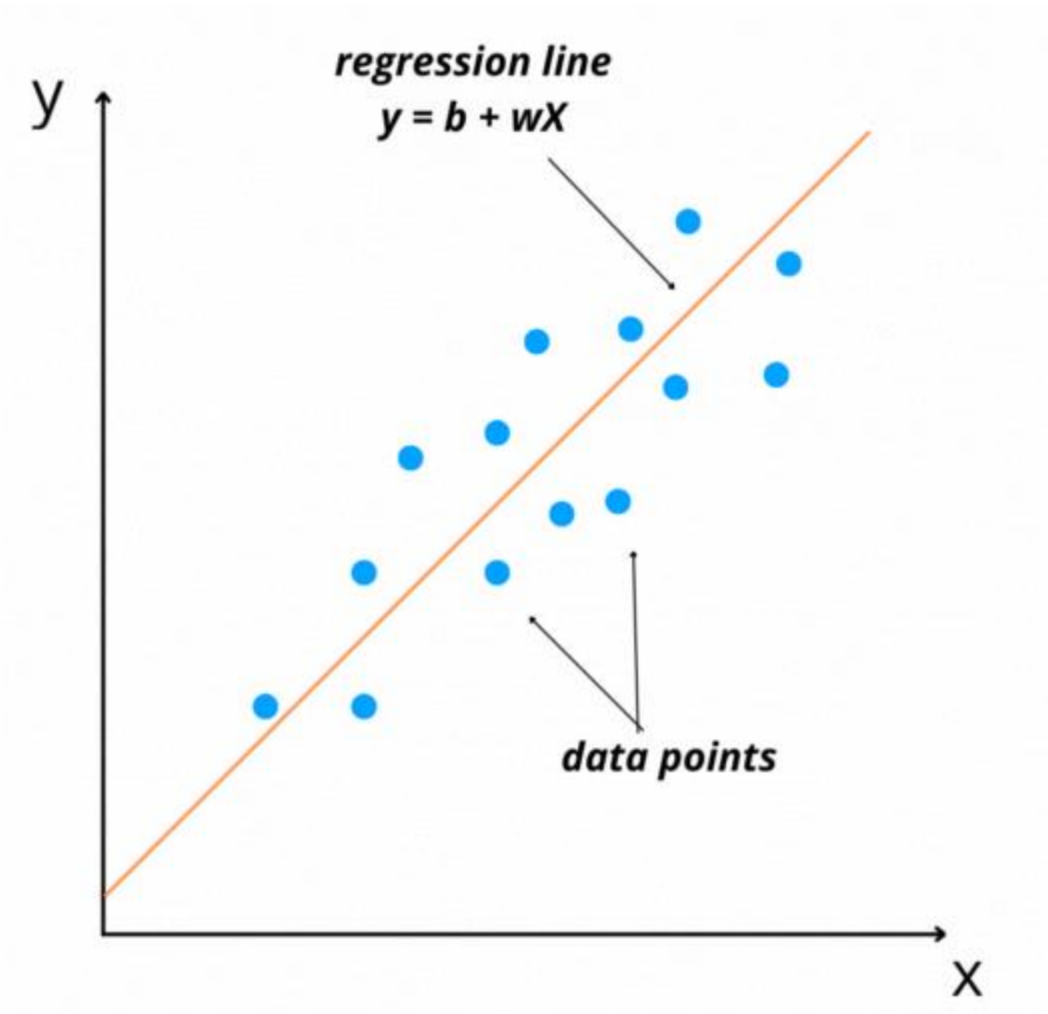
Dependent Variable/ output

y-intercept/ bias constant

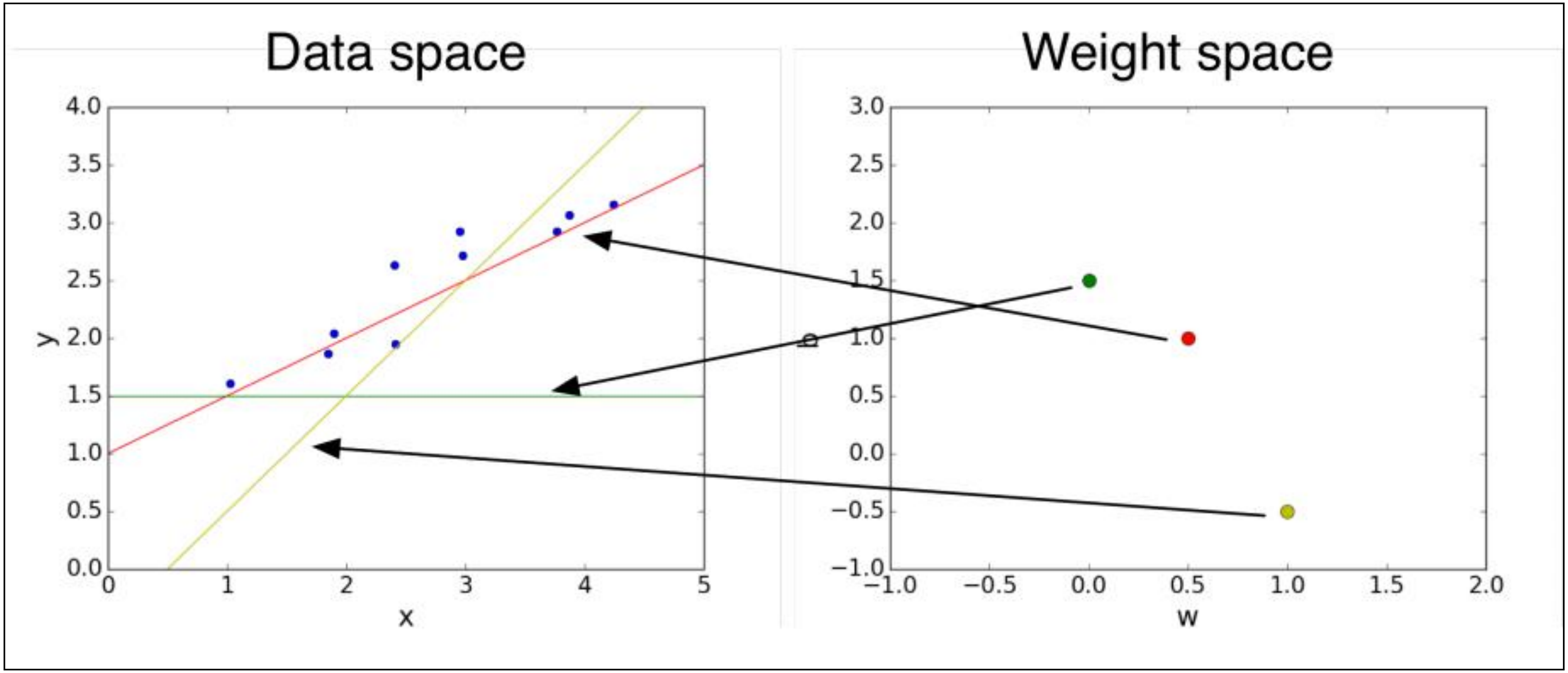
Coefficient of regression

Features or Independent variable

Simple Linear Regression



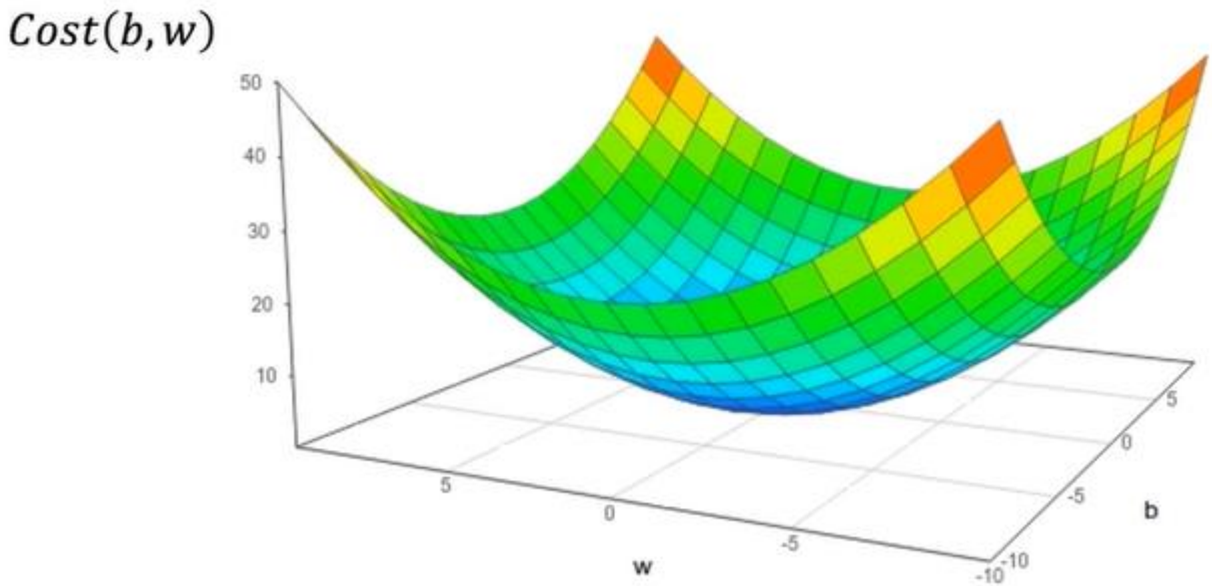
Simple Linear Regression



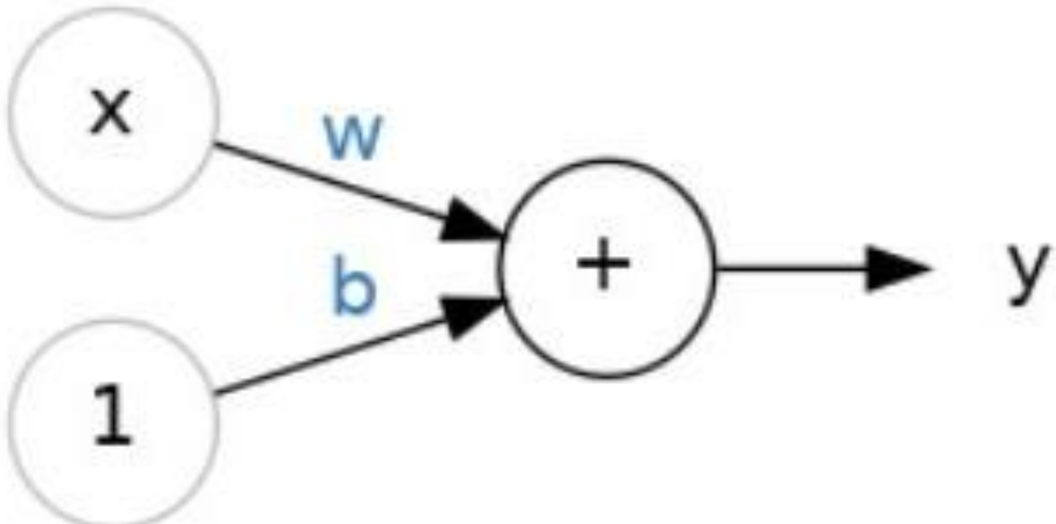
Simple Linear Regression

$$Cost(b,w) = \mathcal{L}(b,w) = \frac{1}{N}\sum_{i=1}^{N}(\widehat{y_i} - y_i)^2$$

$$= \mathcal{L}(x,b,w) = \frac{1}{N}\sum_{i=1}^{N}(wx_i + b - y_i)^2$$



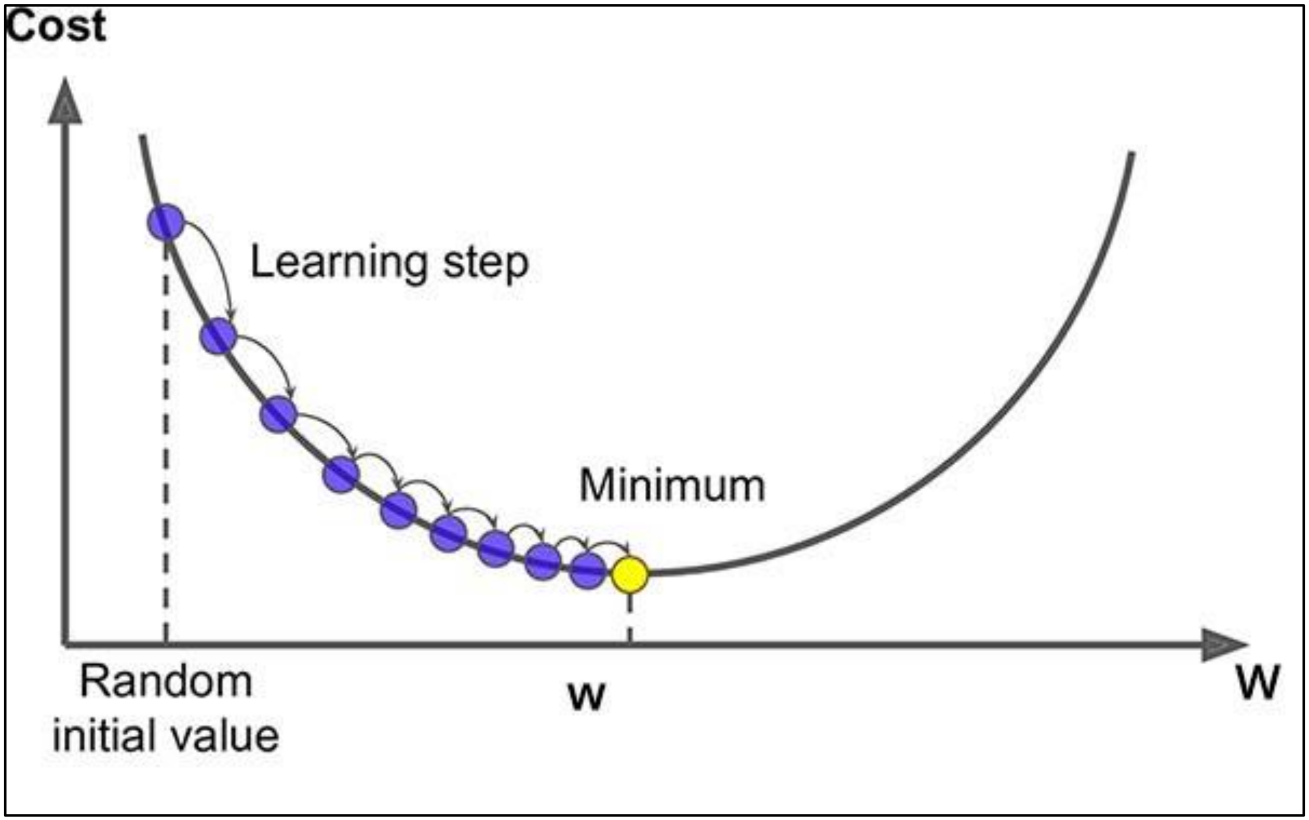
Simple Linear Regression



The Linear Unit: $y = wx + b$

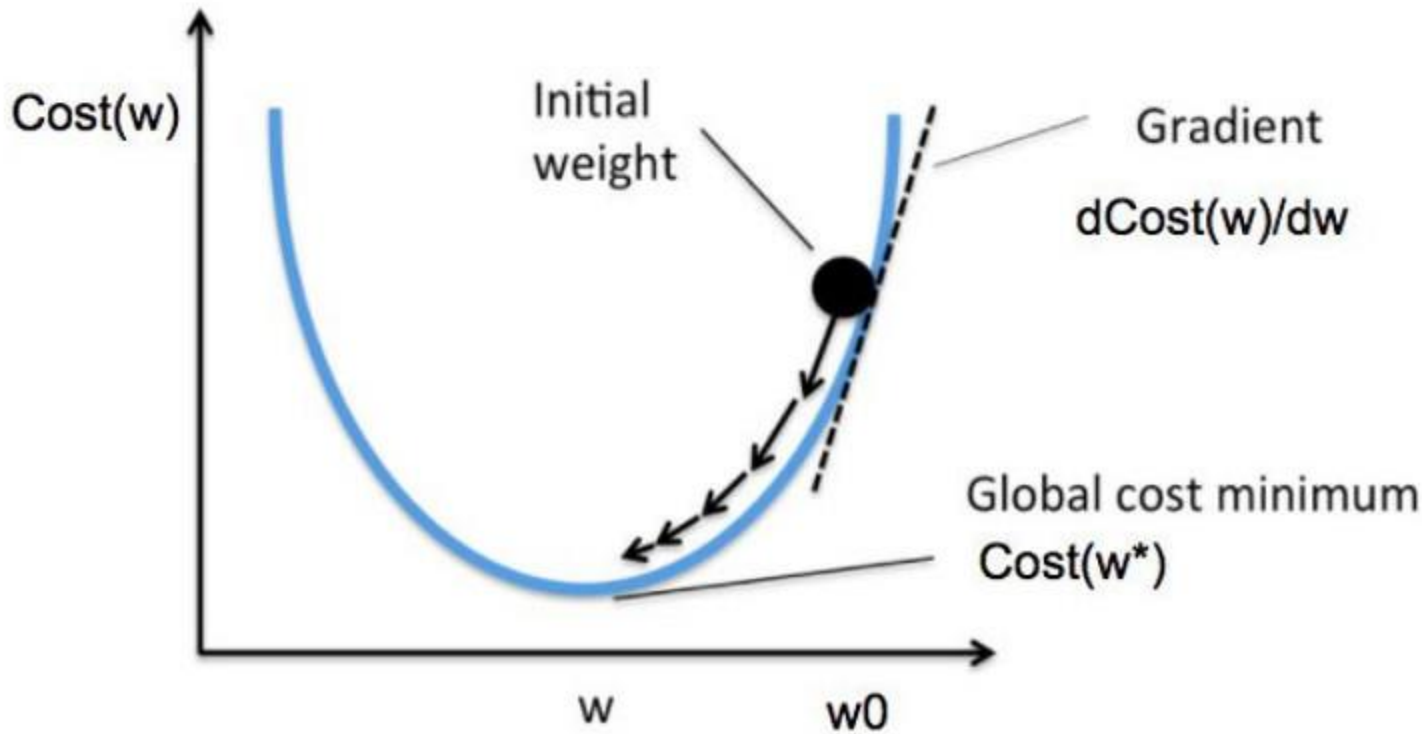
- **NN=Architecture + Parameters**
- **Training NN = Given data learn best Parameters which gives minimum loss (usually an iterative process/algorithm)**

Simple Linear Regression



Gradient Descent Algorithm

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$



Simple Linear Regression

$$\mathcal{L}(x,b,w) = \frac{1}{N}\sum_{i=1}^{N}(wx_i + b - y_i)^2$$

$$dw = \frac{\partial \mathcal{L}(x,b,w)}{\partial w} = 2 * \frac{1}{N}\sum_{i=1}^{N}(wx_i + b - y_i)(x_i)$$

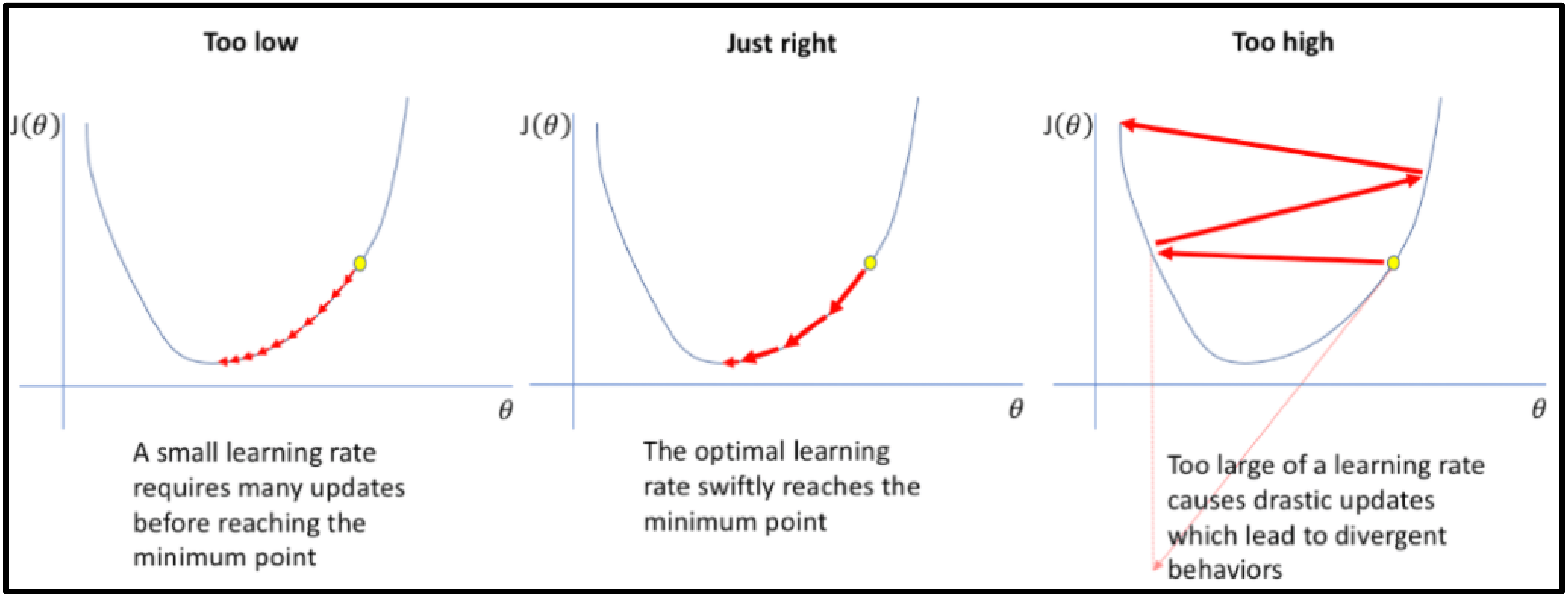
$$= 2 * \frac{1}{N}\sum_{i=1}^{N}(error_i)(x_i) = 2 * mean(error * x)$$

$$db = \frac{\partial \mathcal{L}(x,b,w)}{\partial b} = 2 * \frac{1}{N}\sum_{i=1}^{N}(error_i) = 2 * mean(error)$$

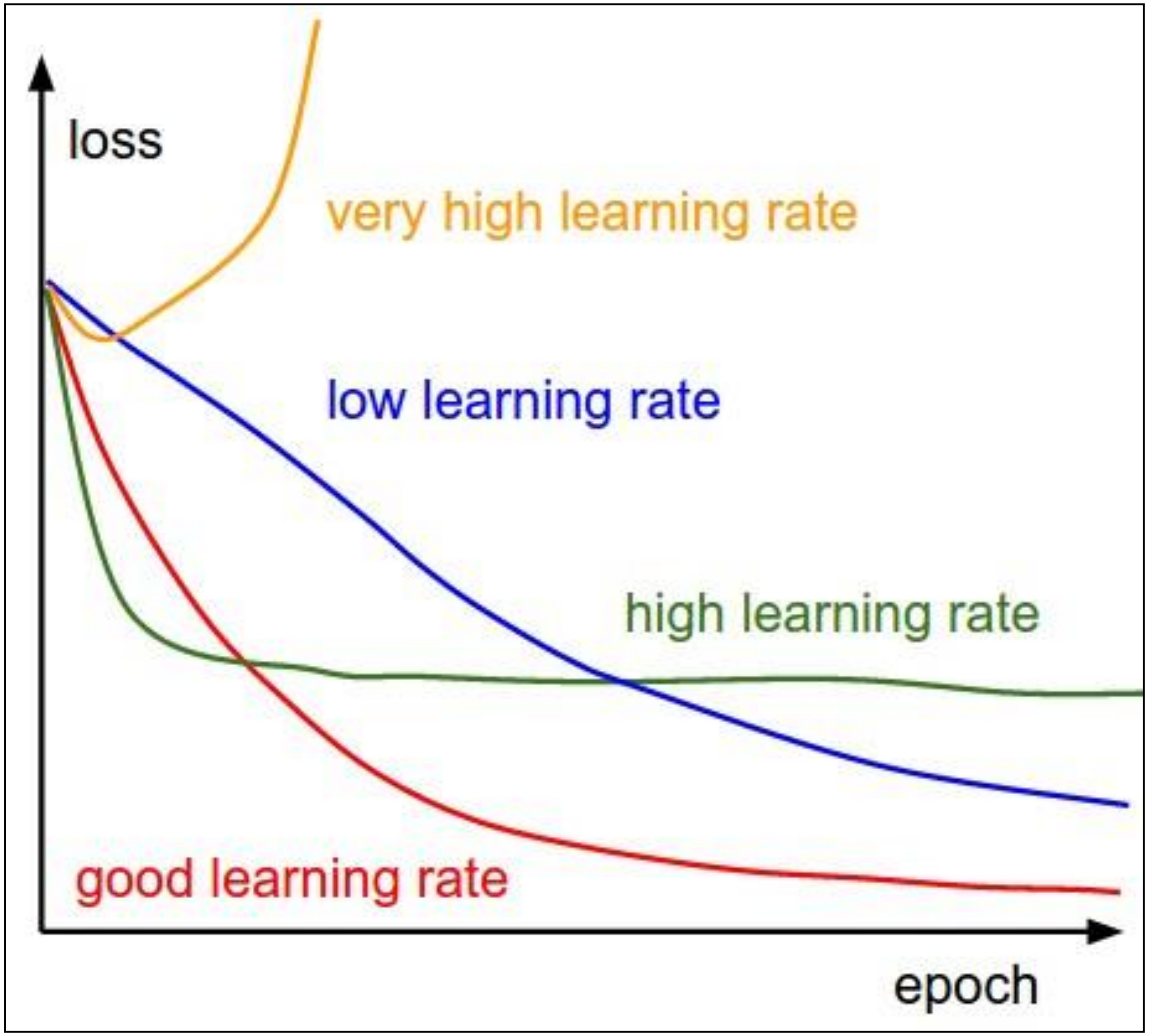
$$w_{new} = w_{old} - \alpha * dw$$

$$b_{new} = b_{old} - \alpha * db$$

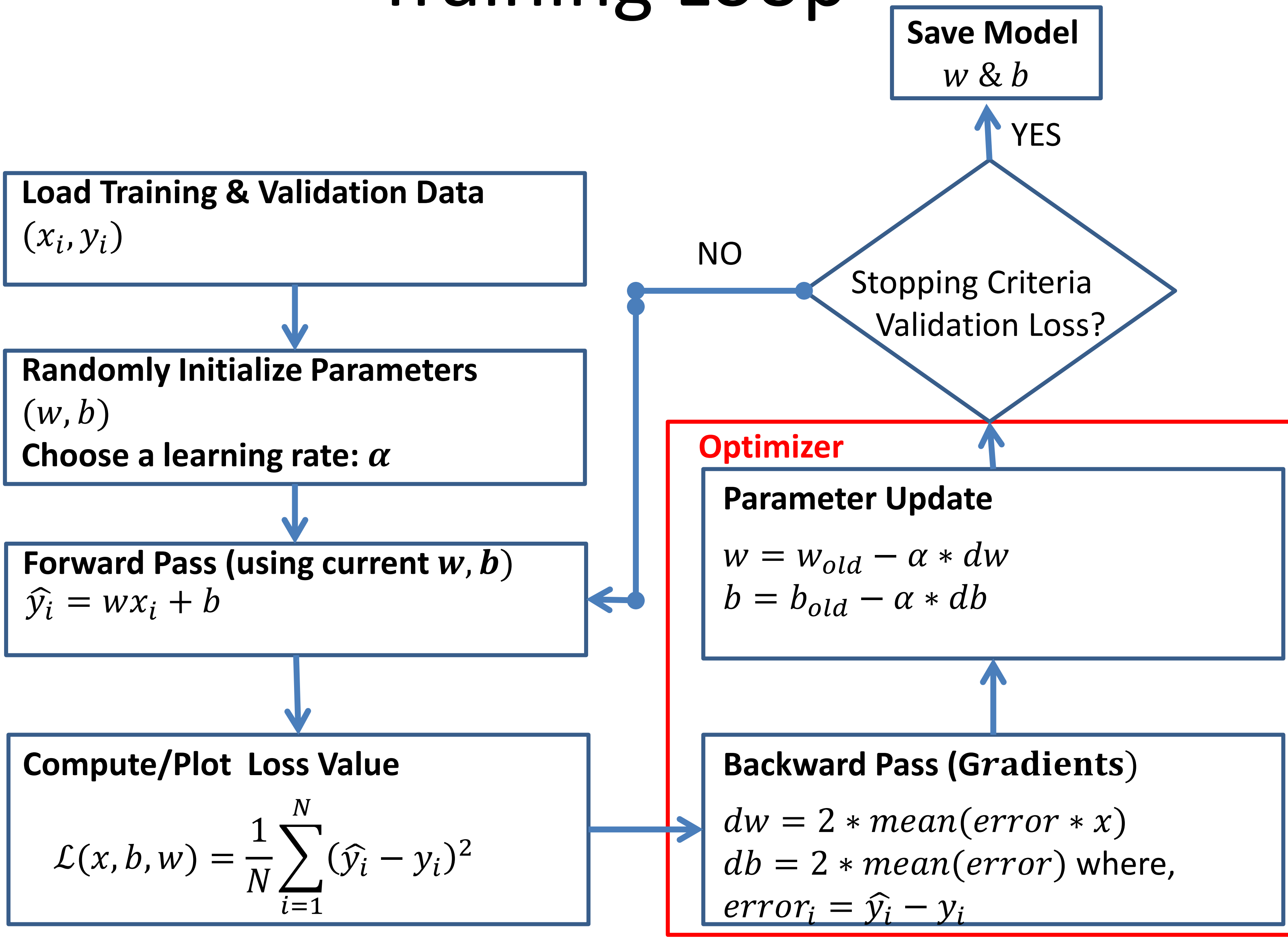
Effect of Learning Rate



Effect of Learning Rate



Training Loop



Inference/Testing

Load Test Data
(x_i, y_i)

↓

Load Saved Model
(w, b)

↓

Forward Pass (using saved $\boldsymbol{w}, \boldsymbol{b}$)
$\hat{y}_i = wx_i + b$

↓

Compute/Plot Loss Value

$$\mathcal{L}(x, b, w) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Batch Gradient Descent

- An epoch refers to one cycle through the full training dataset
- All the training data is taken into consideration to update parameters
- Uses mean of gradients to update parameters
- One step/iteration of gradient descent in one epoch
- Great for convex or relatively smooth error surfaces
- The graph of cost vs epochs is also quite smooth

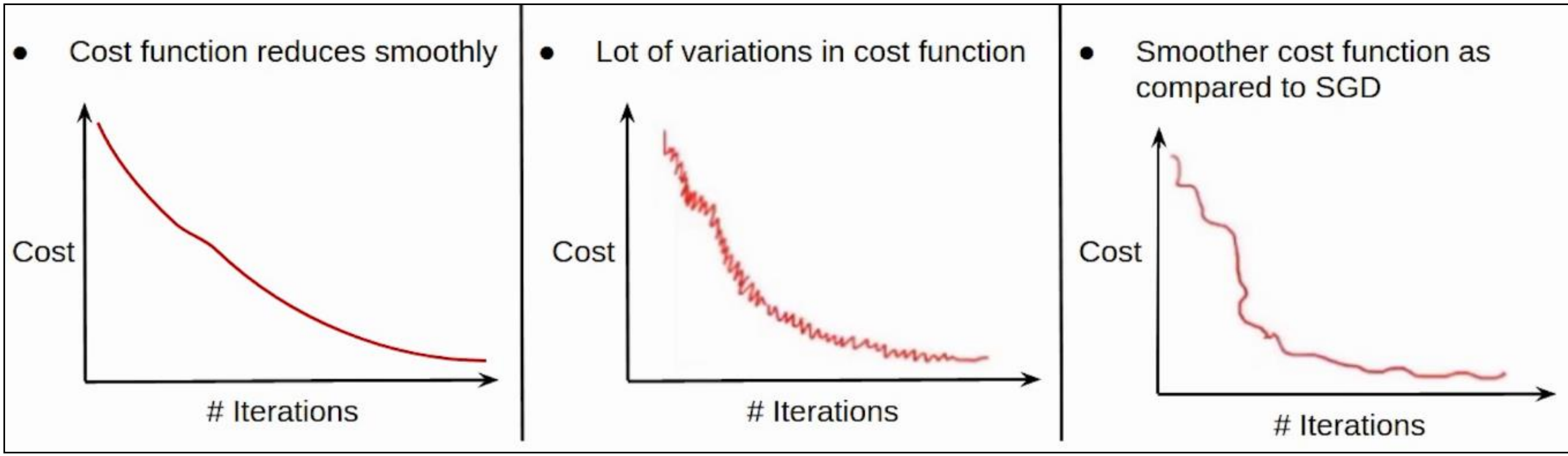
Stochastic Gradient Descent

- If our dataset has 5 million examples, then just to take one step the model will calculate the gradients 5 million examples (inefficient)
- In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step.
- Cost will fluctuate over the training examples
- N steps/iterations of gradient descent in one epoch

Mini Batch Gradient Descent

- We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch
- Average cost over the epochs in mini-batch gradient descent fluctuates because we are averaging a small number of examples at a time.
- N/batch_size steps/iterations of gradient descent in one epoch

Batch vs Stochastic vs Mini-Batch



Batch GD **Stochastic GD** **Mini-Batch GD**

We can divide the dataset of 2000 examples into batches of 500 then it will take ? iterations to complete 1 epoch.

A SIMPLE REGRESSION PROBLEM (NUMPY IMPLEMENTATION)

Dataset Generation

```
import numpy as np
import matplotlib.pyplot as plt

#data generation
true_w=2
true_b=1
N=100

np.random.seed(100)
#get N uniformly distributed values
x=np.random.rand(N,1)
#get N noise values from standard normal distribution
epsilon=0.1*np.random.randn(N,1)
y=true_w*x+true_b+epsilon
```

Splitting Dataset into train/validation sets

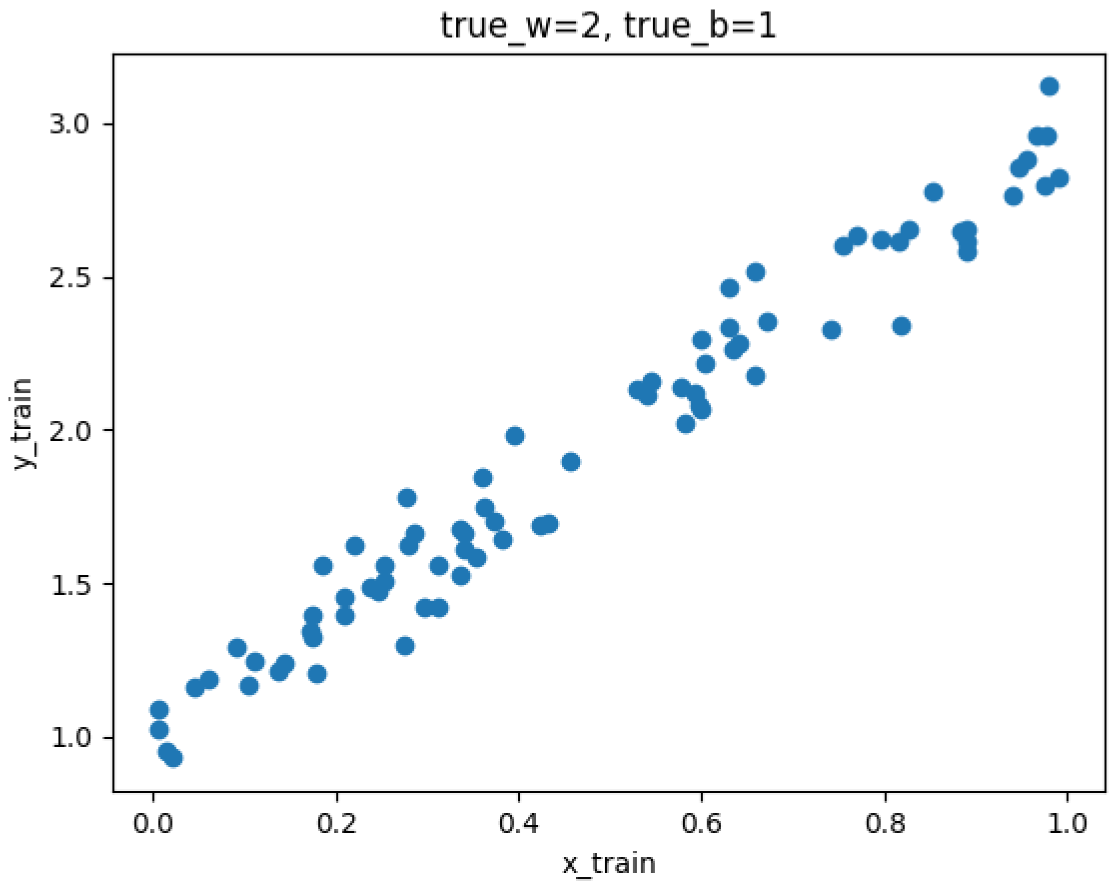
```
#Splitting Data into train and validation
idx=np.arange(N)
np.random.shuffle(idx)
idx_train=idx[:int(0.8*N)]
idx_test=idx[int(0.8*N):]
x_train, y_train = x[idx_train],y[idx_train]
x_val, y_val = x[idx_test],y[idx_test]
```

```
WATCH
> x_val.shape: (20, 1)
> y_val.shape: (20, 1)
> x_train.shape: (80, 1)
> x_train.shape: (80, 1)
```

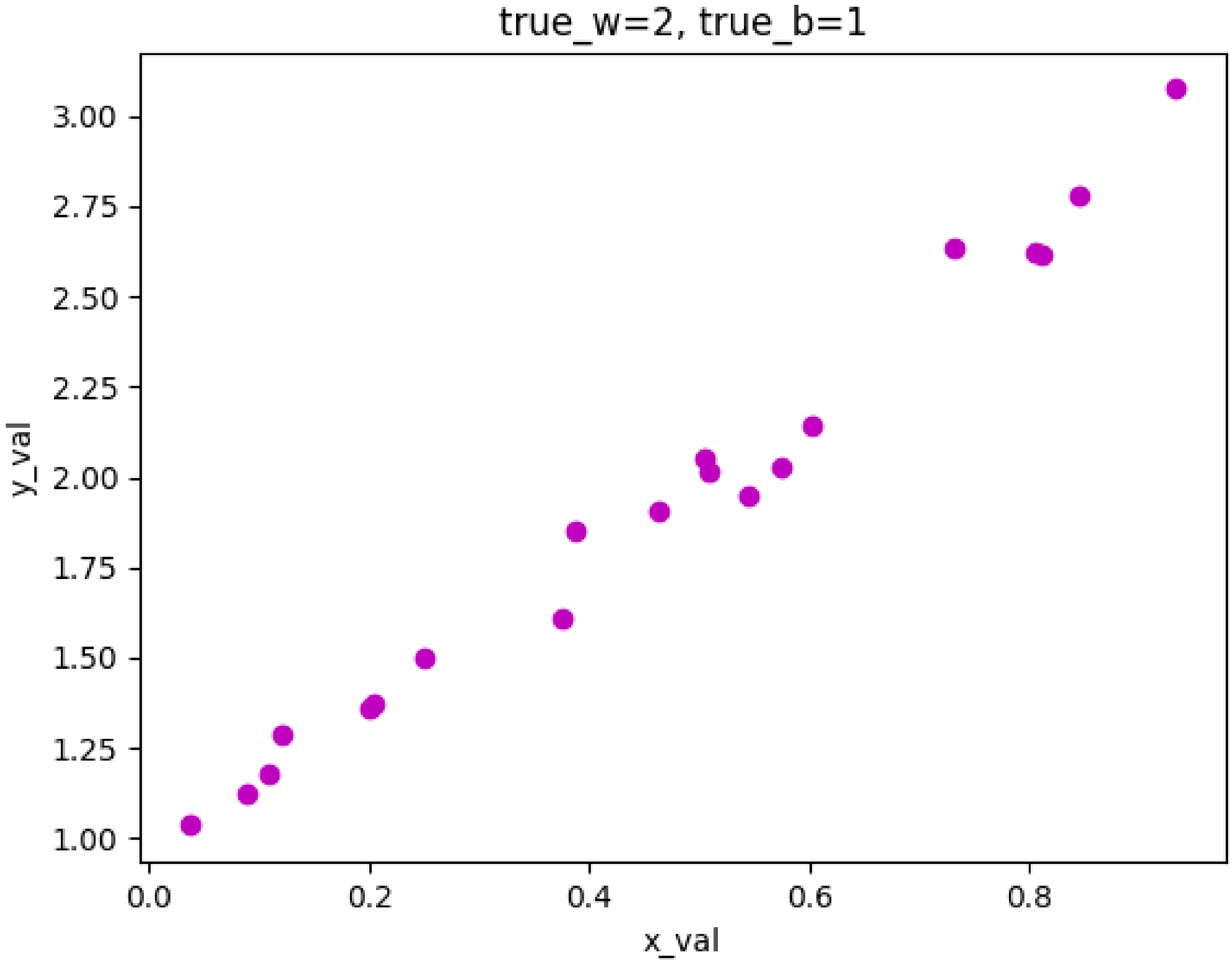
Splitting Dataset into train/validation sets

```
#plotting tain and val data
plt.figure('1')
plt.scatter(x_train,y_train)
plt.xlabel('x_train')
plt.ylabel('y_train')
plt.title(f'true_w={true_w}, true_b={true_b}')
plt.figure('2')
plt.scatter(x_val,y_val,color = 'm')
plt.xlabel('x_val')
plt.ylabel('y_val')
plt.title(f'true_w={true_w}, true_b={true_b}')
plt.show(block=True)
```

Visualizing Datasets



Visualizing Datasets



Training Loop

```
#training loop
#initializing parameters
trainLosses=[]
valLosses=[]
lr=0.1
w=np.random.randn(1)
b=np.random.randn(1)
for i in range(100):
    #forward pass
    yhat=w*x_train+b #note vectorized operation
    #MSE loss
    error=yhat-y_train
    loss= (error**2).mean()
    trainLosses.append(loss)
    #computing gradients
    db=2*error.mean()
    dw=2*(x_train*error).mean()
    #weight update
    b=b-lr*db
    w=w-lr*dw
```

Validation Loss

```
    #val MSE loss
    yhatVal=w*x_val+b
    errorVal=yhatVal-y_val
    valLoss= (errorVal**2).mean()
    valLosses.append(valLoss)

    #stopping condition
    if(valLoss<0.0001):
        break

print(f'train loss={loss}, val loss={valLoss}, w={w}, b={b}')
```

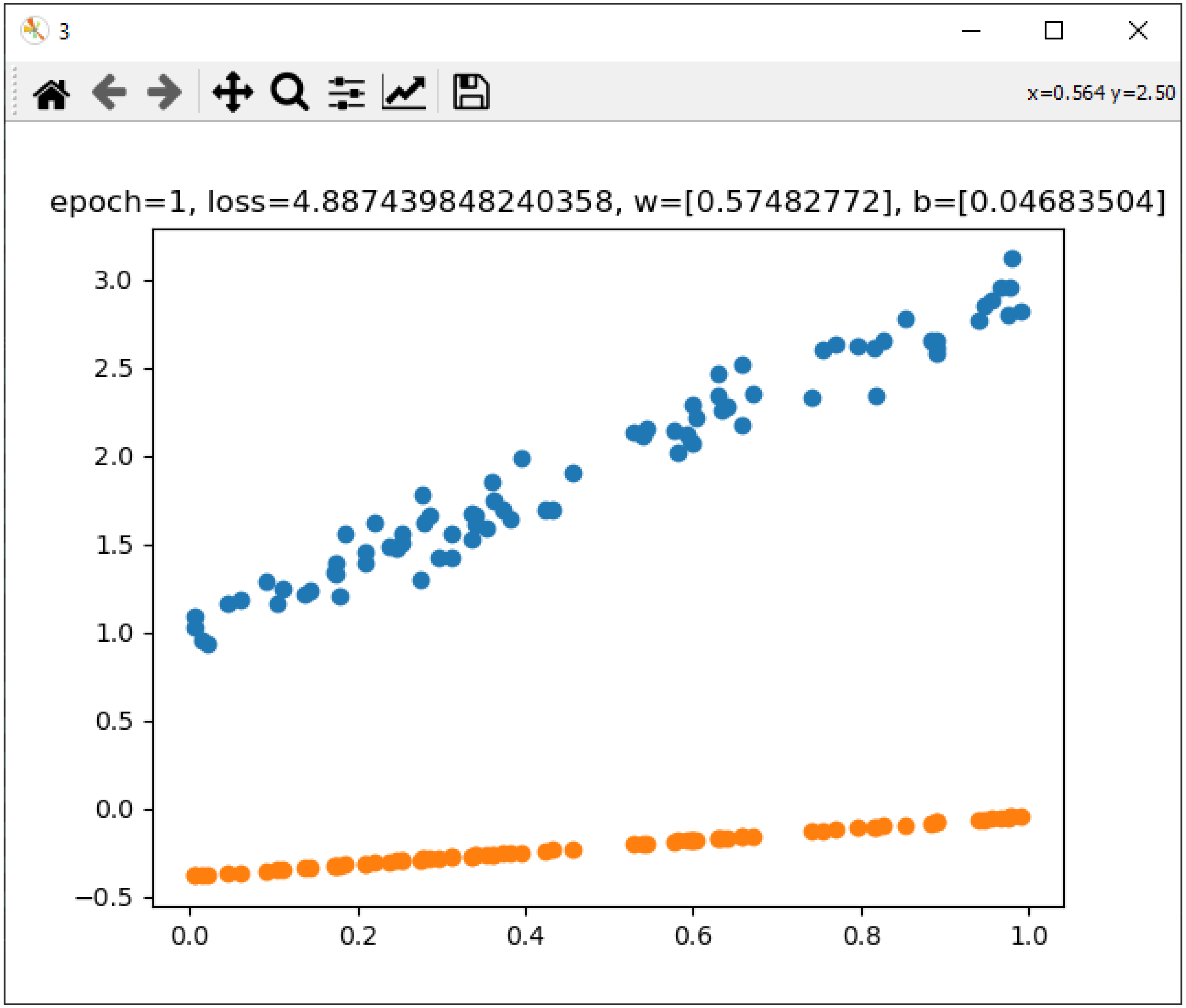
Plots (Regression Fitting)

```
#training data plot
plt.figure('3')
plt.cla()
plt.scatter(x_train,y_train)
plt.scatter(x_train,yhat)
plt.title(f'epoch={i}, loss={loss}, w={w}, b={b}')
plt.show(block=False)
plt.pause(1)

#validation data plot
plt.figure('4')
plt.cla()
plt.scatter(x_val,y_val)
plt.scatter(x_val,yhatVal)
plt.title(f'epoch={i}, ValLoss={valLoss}, w={w}, b={b}')
plt.show(block=False)
plt.pause(1)
```

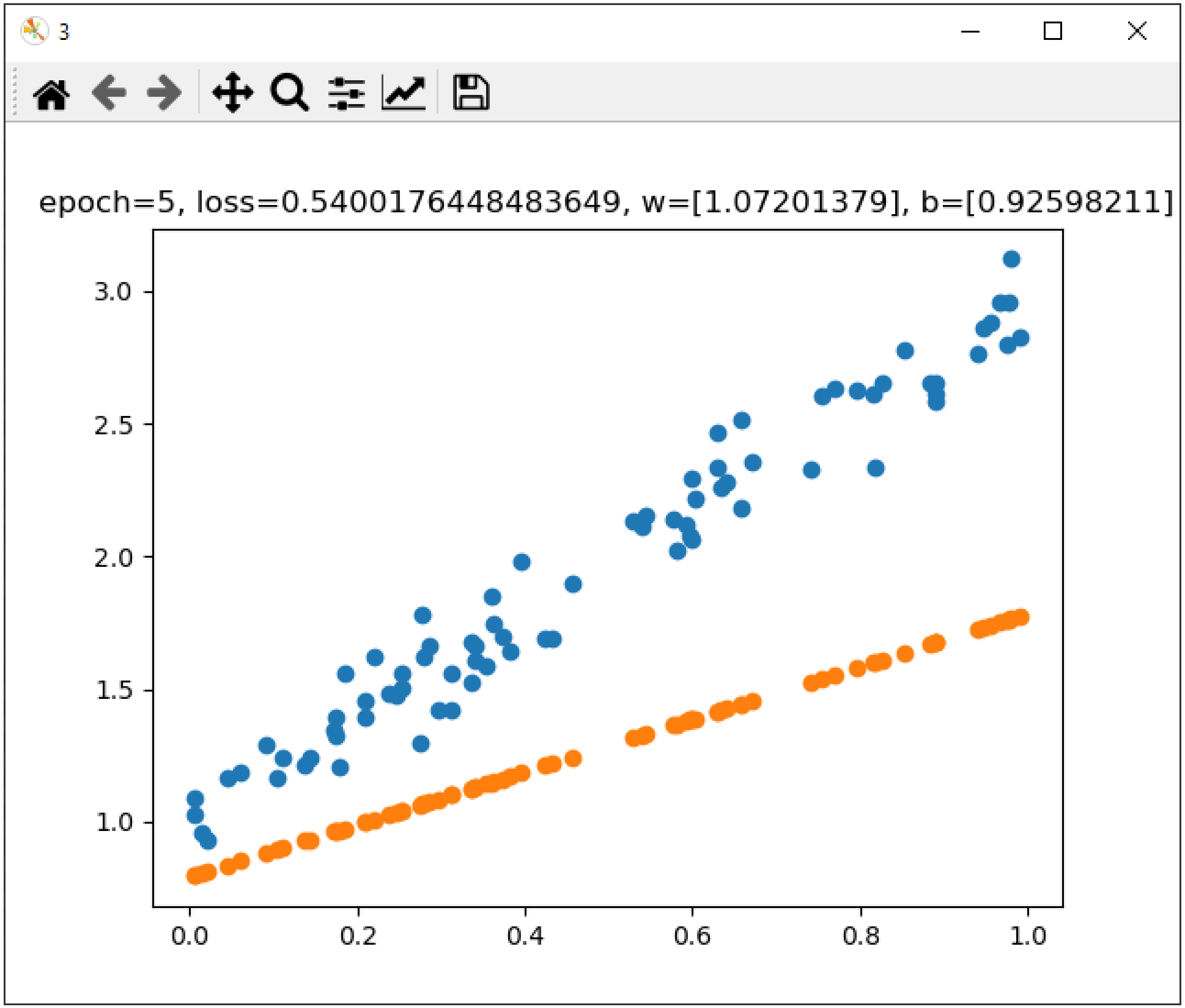
Plots (Loss vs Epoch)

```
    #trainLoss vs Epoch
    plt.figure('5')
    plt.cla()
    plt.plot(trainLosses)
    plt.xlabel('Epoch')
    plt.ylabel('trainLoss')
    plt.title(f'Training Loss Vs Epoch')
    plt.show(block=False)
    plt.pause(1)

    #validationLoss vs Epoch
    plt.figure('6')
    plt.cla()
    plt.plot(valLosses,color='m')
    plt.xlabel('Epoch')
    plt.ylabel('valLoss')
    plt.title(f'Validation Loss Vs Epoch')
    plt.show(block=False)
    plt.pause(1)
```

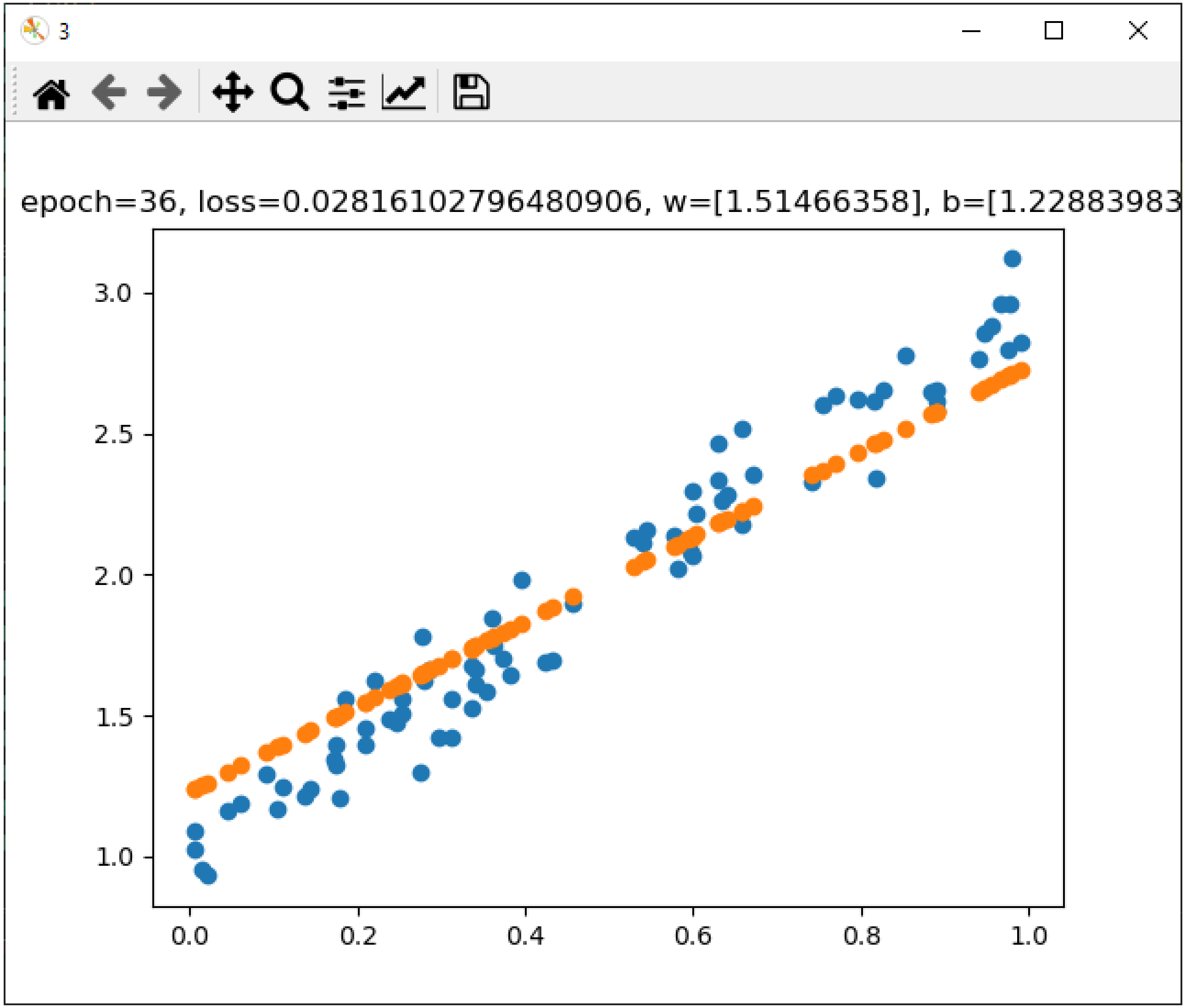
Plots (Convergence of regression)



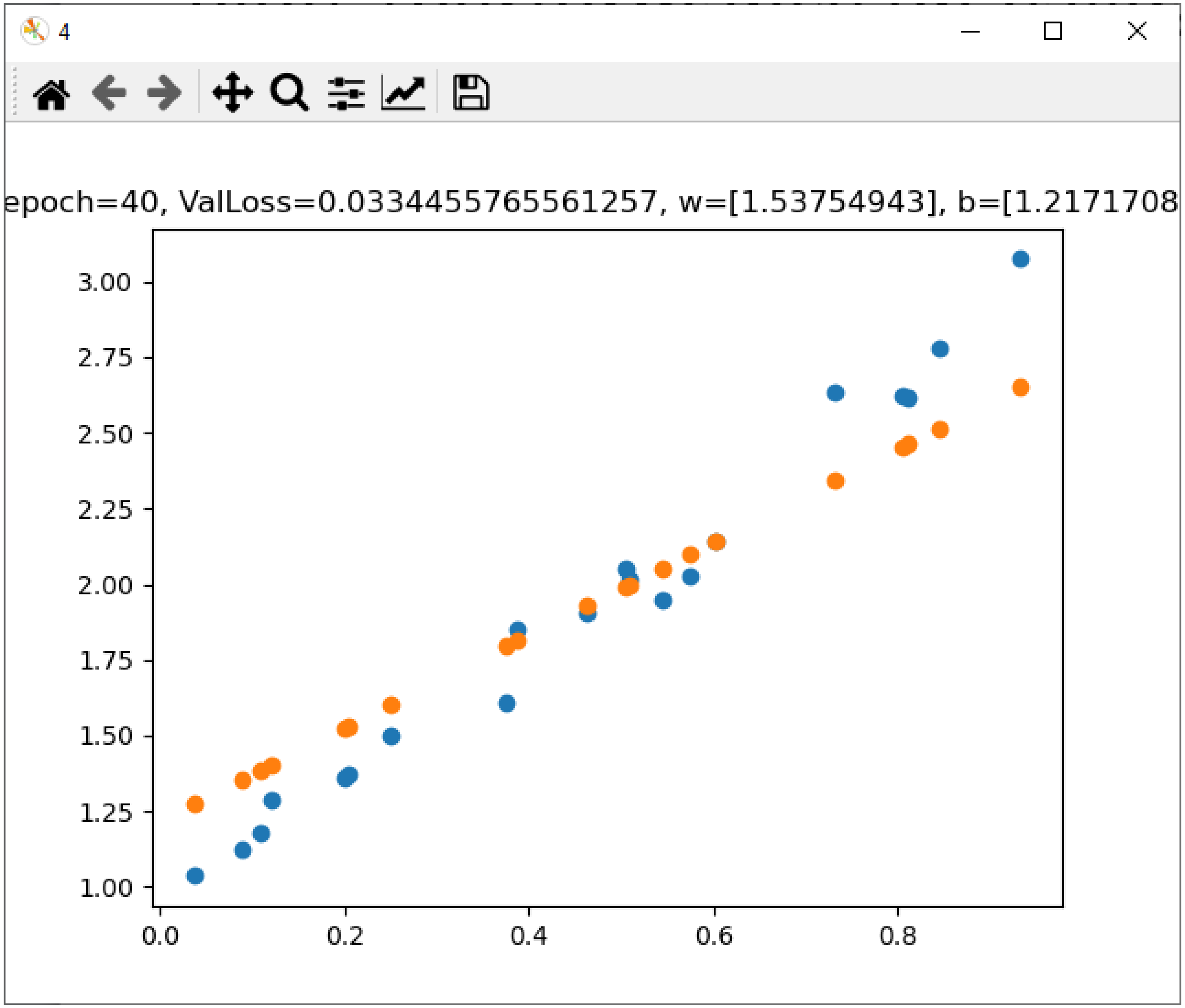
Plots (Convergence of regression)



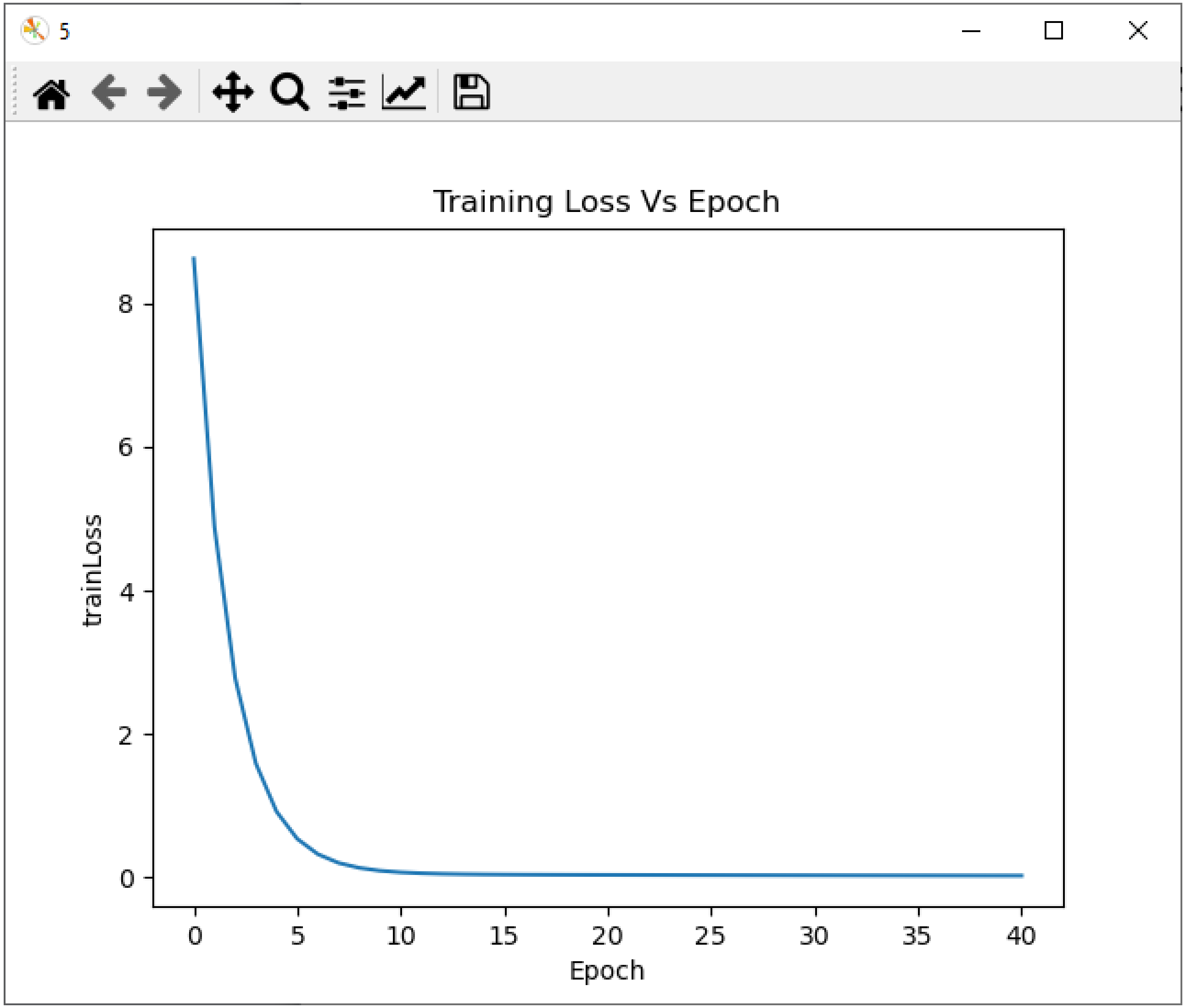
Plots (Convergence of regression)



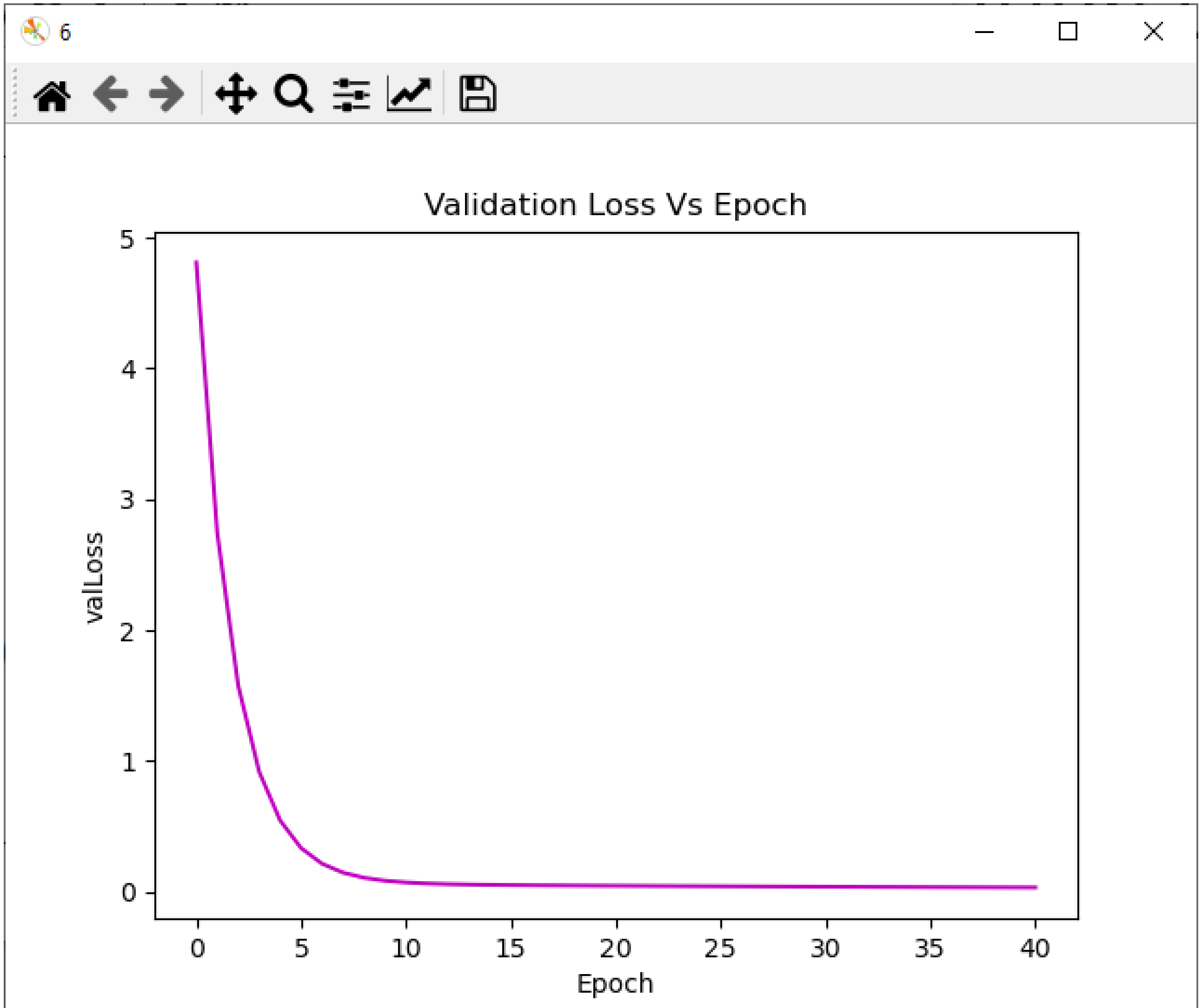
Plots (Performance on Validation Set)



Epoch vs Train Loss



Epoch vs Validation Loss



NN Summary

- Data Set, Training, Validation, Test
- Cost/Loss Function
  - MSE Loss for regression
- Training Loop
- Optimizer
- Parameters
- Learning Rate
- Epoch
- Batch
- Loading/Saving Model

Home Task

- Compare learning curves for different values of learning rate
- Convert code from Batch Gradient Descent to Stochastic Gradient Descent and compare learning curves