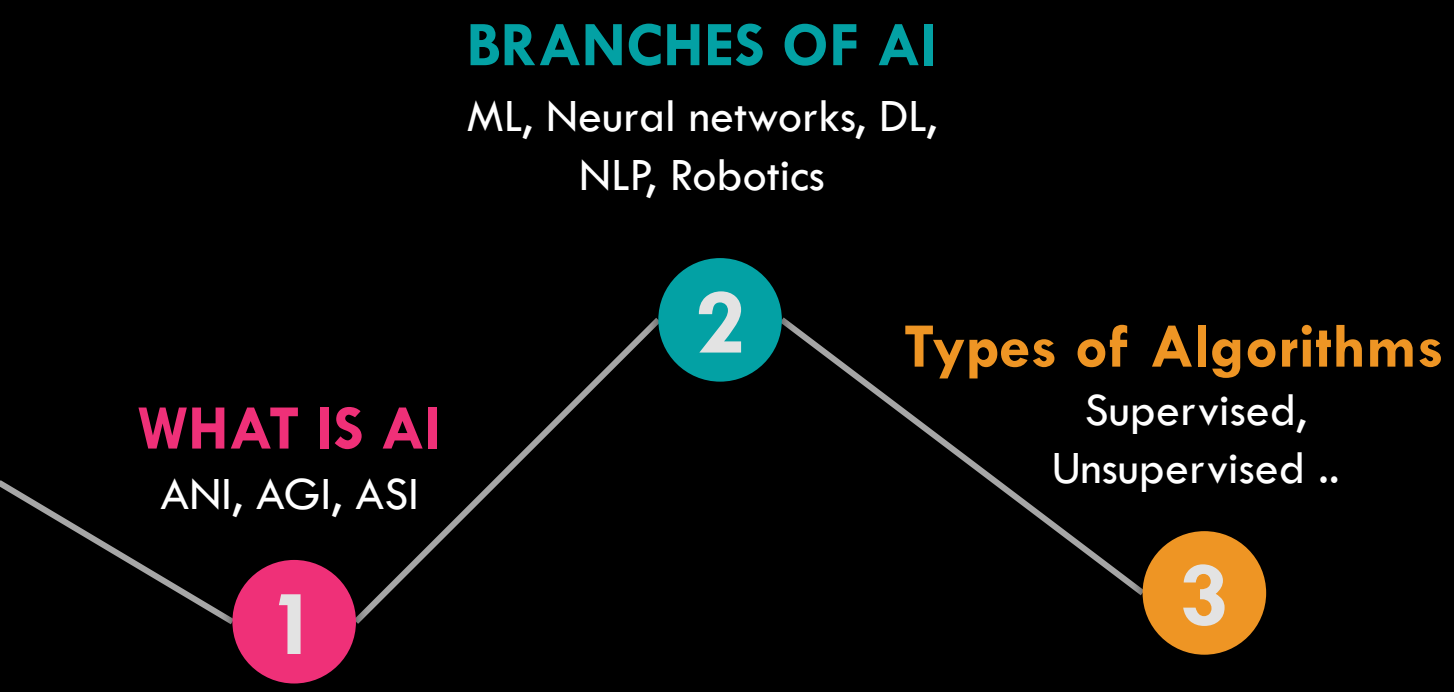
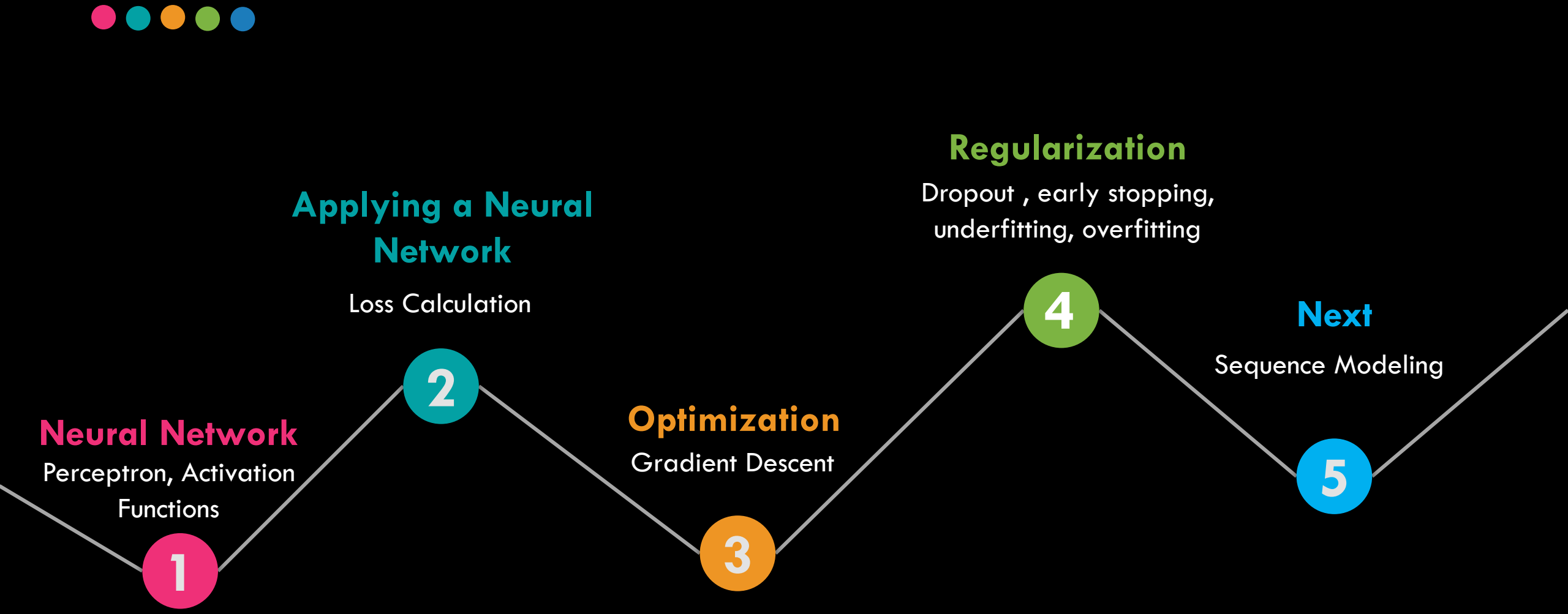
Fundamentals of AI/ML

Session-2

Recap


BRANCHES OF AI
ML, Neural networks, DL,
NLP, Robotics
2
Types of Algorithms
Supervised,
Unsupervised ..
3
WHAT IS AI
ANI, AGI, ASI
1

TODAY – Introduction to Deep Learning

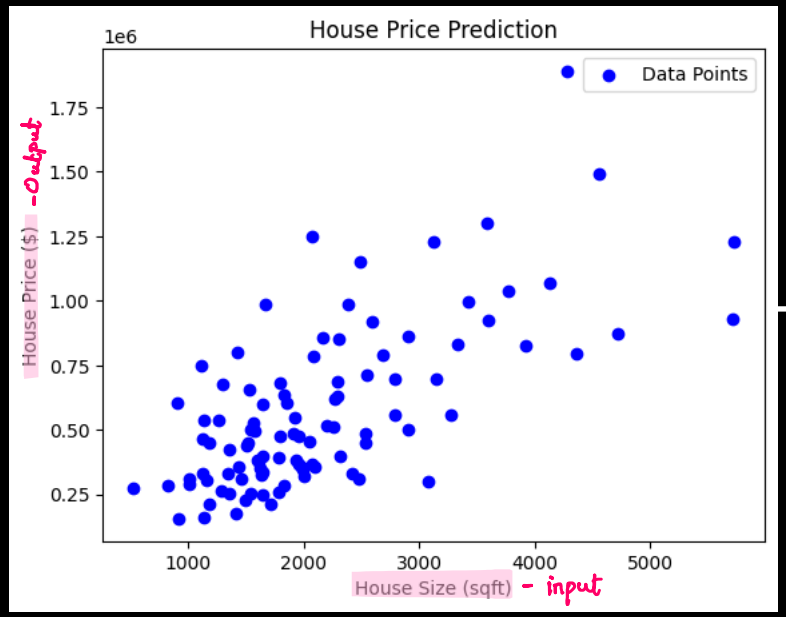


Neural Network

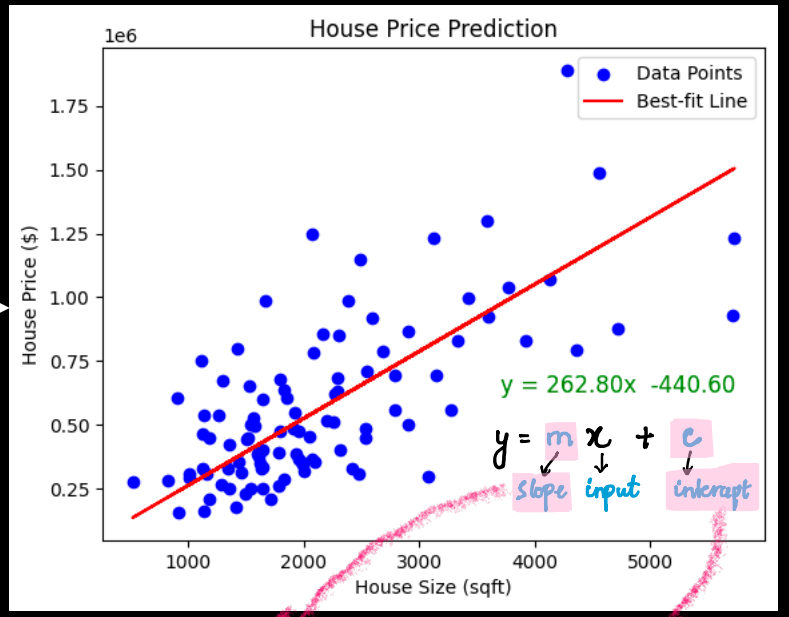
We can think of Neural Network as,
A line fitting model at its simplest.

Dataset

sqft_living	price
2250.00	292500.00
2420.00	301000.00
3250.00	951000.00
1850.00	430000.00
2150.00	650000.00
1260.00	289000.00
2519.00	505000.00
1540.00	549000.00
1660.00	425000.00
2770.00	317625.00
2720.00	975000.00
2240.00	287000.00
1000.00	204000.00
3200.00	1330000.00
4770.00	1040000.00
1260.00	325000.00
2750.00	571000.00
2380.00	360000.00
1790.00	349000.00
3430.00	832500.00



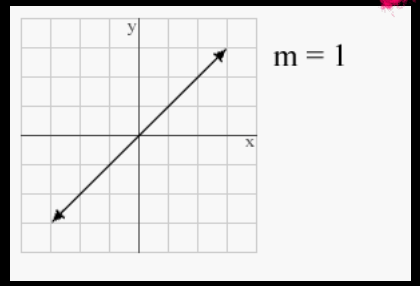
Train a **neural network** to ***predict*** House Price given House size as ***Input***

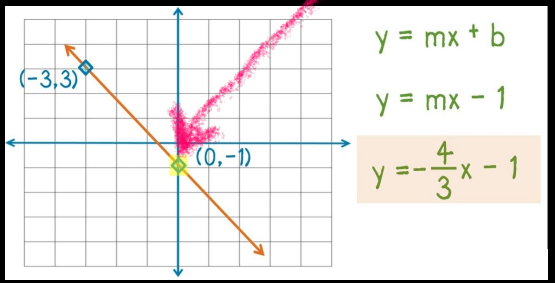


Terminology Alert!

Slope in neural networks is weight

intercept in neural network is Bias



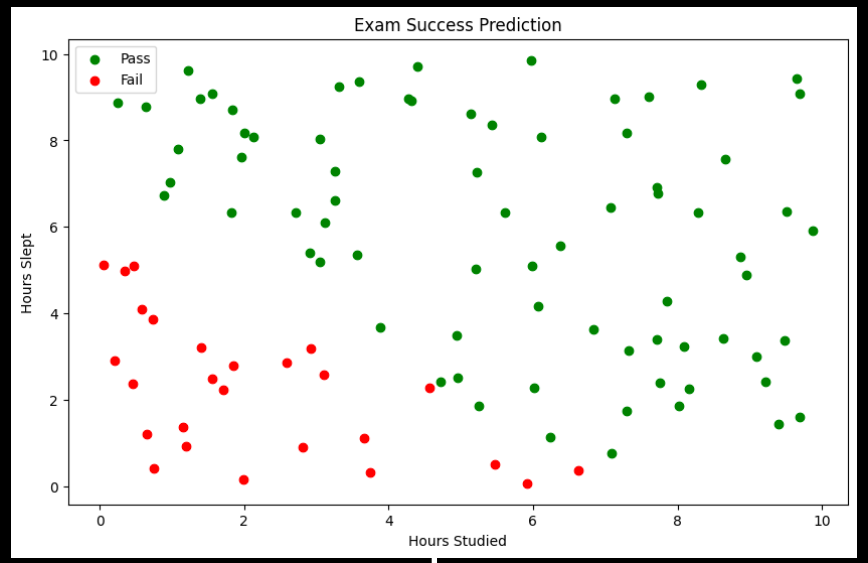


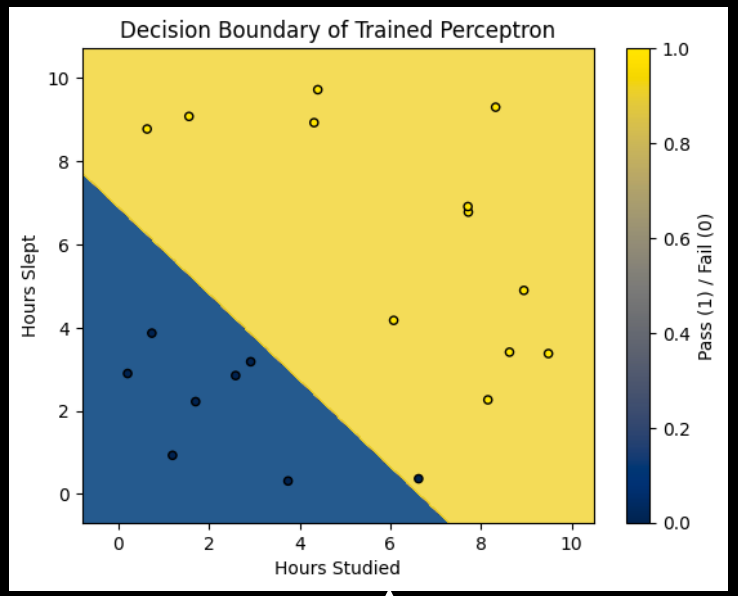
Dataset: https://github.com/RohanMudgal/House-Price-Prediction/blob/main/Kc_Price.csv

Neural Network

We can think of Neural Network as,
A line fitting model at its simplest.

HoursStudied	HoursSlept	Pass
3.75	0.31	0.00
9.51	6.36	1.00
7.32	3.14	1.00
5.99	5.09	1.00
1.56	9.08	1.00
1.56	2.49	0.00
0.58	4.10	0.00
8.66	7.56	1.00
6.01	2.29	1.00
7.08	0.77	1.00
0.21	2.90	0.00
9.70	1.61	1.00
8.32	9.30	1.00
2.12	8.08	1.00
1.82	6.33	1.00
1.83	8.71	1.00
3.04	8.04	1.00
5.25	1.87	1.00
4.32	8.93	1.00
2.91	5.39	1.00





Train a **neural network** to ***predict*** if a student will Pass given time slept and studied as ***Input***

Terminology Alert!

This simplest model is referred as

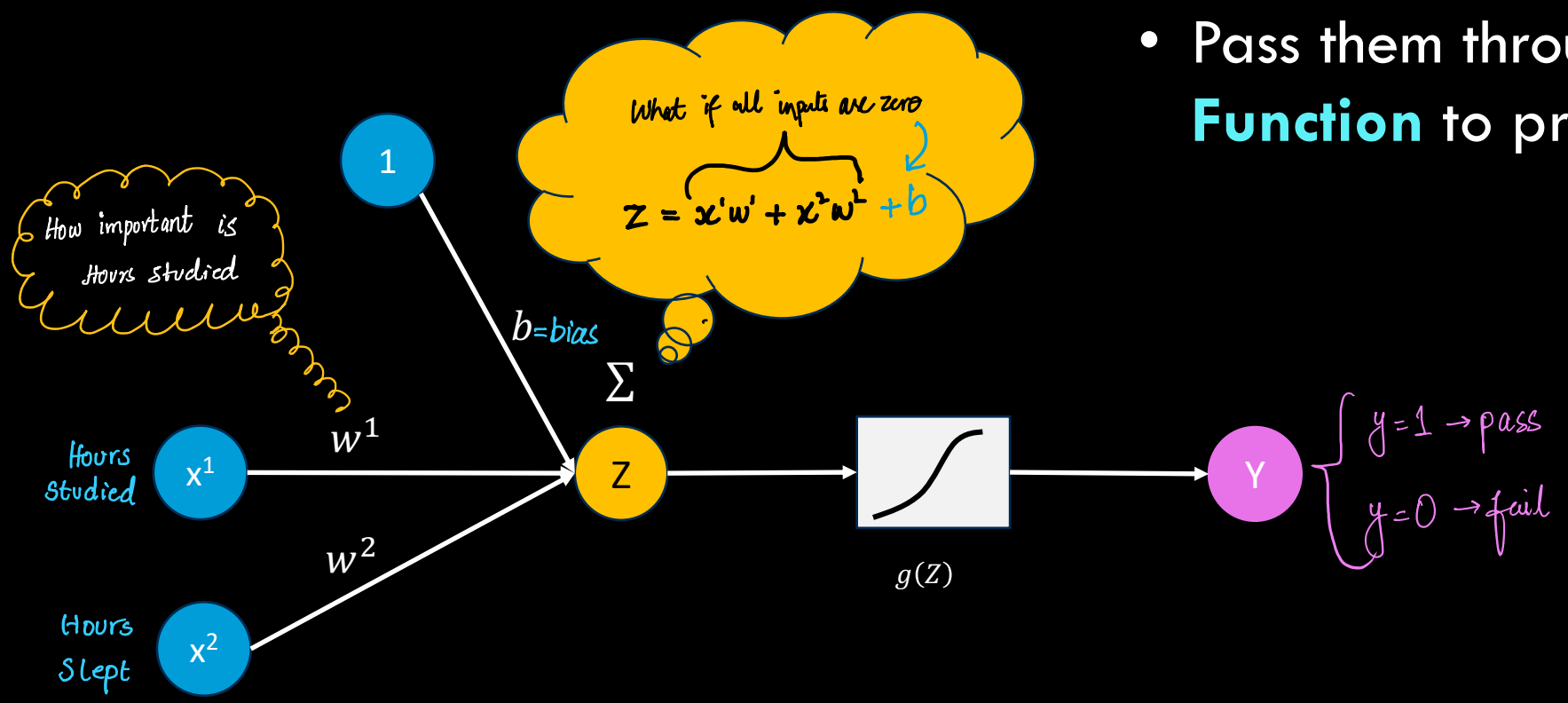
Perceptron

$$Pass = g(w^1 \times hours\_studied + w^2 \times hours\_slept + b)$$

Perceptron

A **single computational Unit** of Neural Networks;

- Takes **Inputs**
- Applies **Weights and bias** to them
- Pass them through **Activation Function** to produce an **Output**

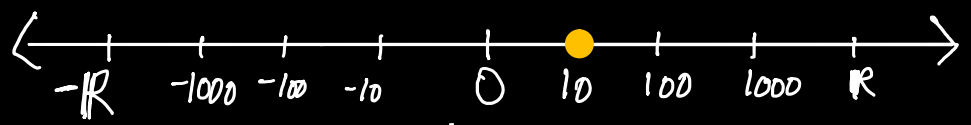


INPUTS **WEIGHTS And bias** **SUMMATION** **Activation Function** **OUTPUT**

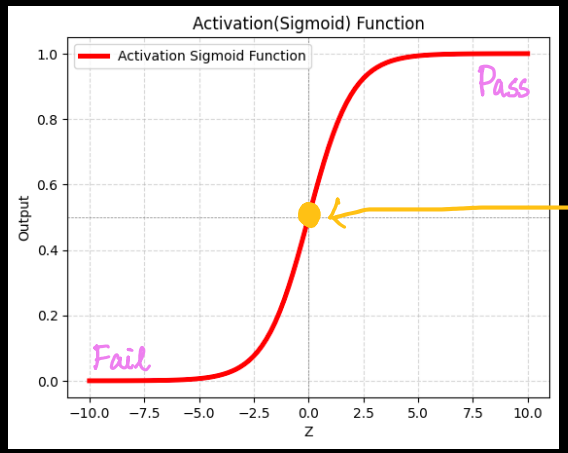
Activation Function

$$Z = x^1w^1 + x^2w^2 + x^0w^0$$

Bias



$$g(Z) = \frac{1}{1 + e^{Z}}$$




```
import torch.nn as nn
input_size = 2
fc = nn.Linear(input_size, 1)  # Single output neuron
```

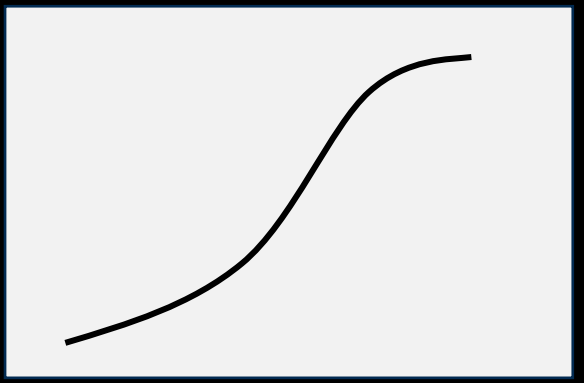
```
x = torch.Tensor([0.6356, 8.7734]) # Hours Studied, Hours Slept

Z = fc(x)
```

```
Y = torch.sigmoid(Z)  # g(z)
Y
✓ 0.0s
tensor([0.5160], grad_fn=<SigmoidBackward0>)
```

0.5 is neutral.

50% confident that student **failed**

50% confident that student **Passed**

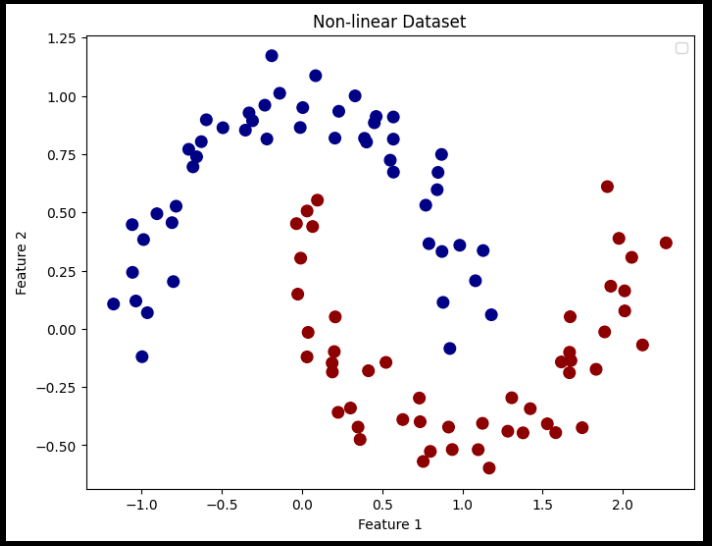


$g(Z)$

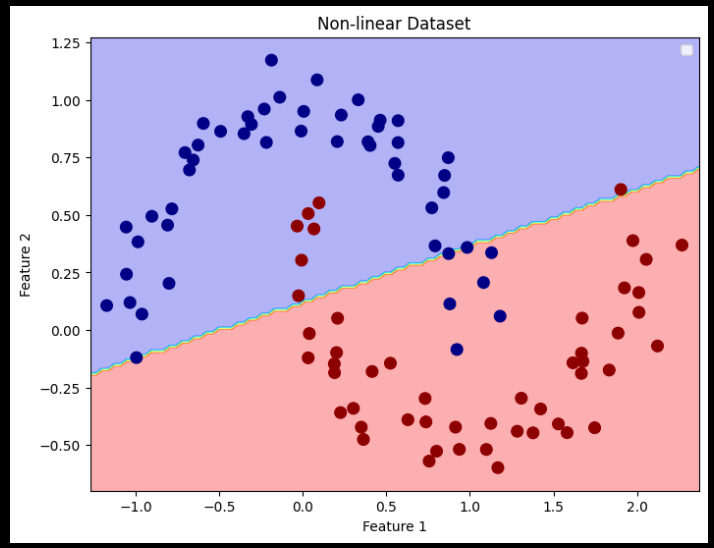
Importance of Activation Function

$g(Z)$

Activation Function introduces non-linearity to the network

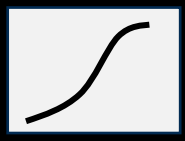



```python
# Define a multi-layer perceptron (MLP) model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(2, 10)  # Input
        self.fc2 = nn.Linear(10, 2)
        self.fc3 = nn.Linear(2, 1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return x
```



Without activation function, model will ***learn linear decision*** boundary , no matter the ***network size***

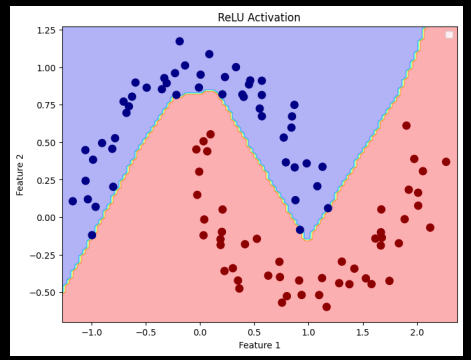
Importance of Activation Function

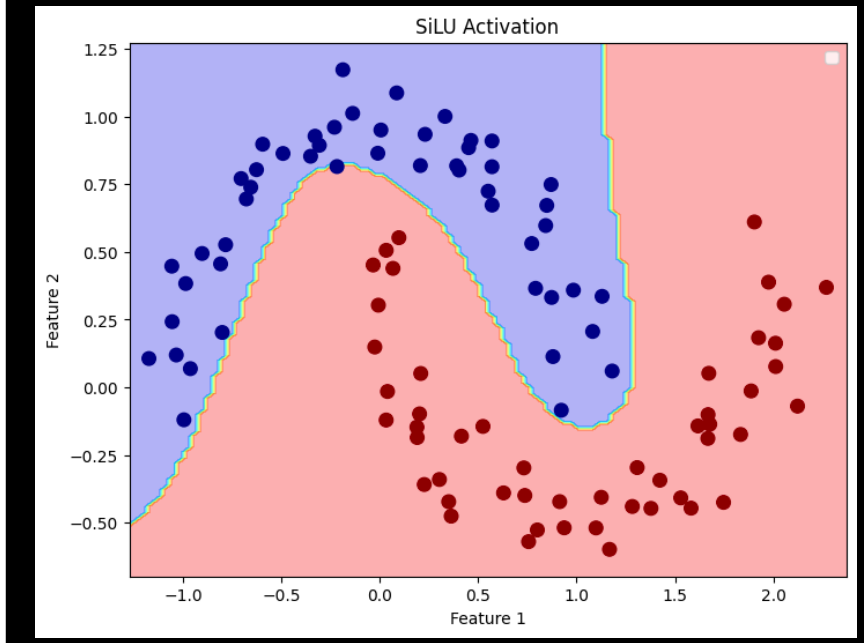


$g(Z)$

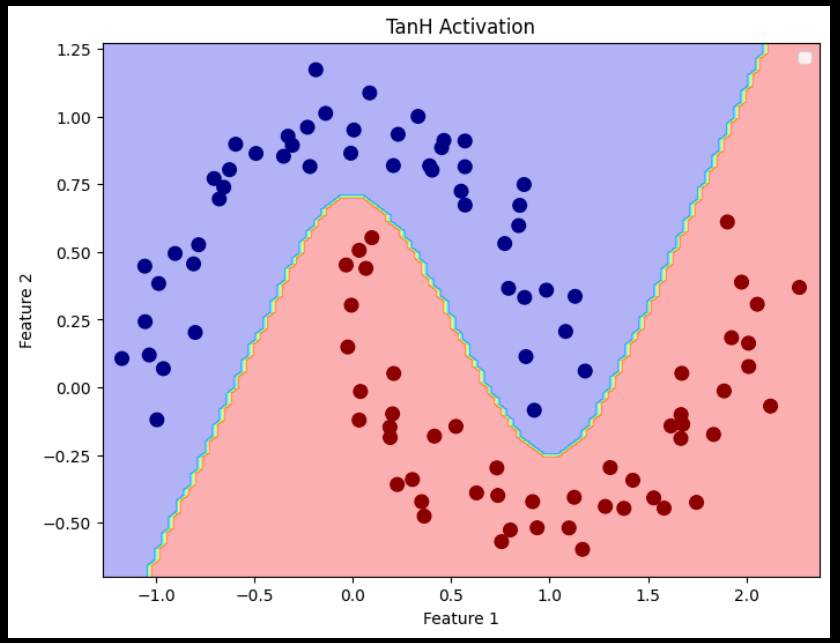
Activation Function introduces non-linearity to the network

```python
# # Define a multi-layer perceptron (MLP)
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(2, 10)  # In
        self.fc2 = nn.Linear(10, 1)   # H
        self.activation = nn.ReLU()   # S

    def forward(self, x):
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        return x
```



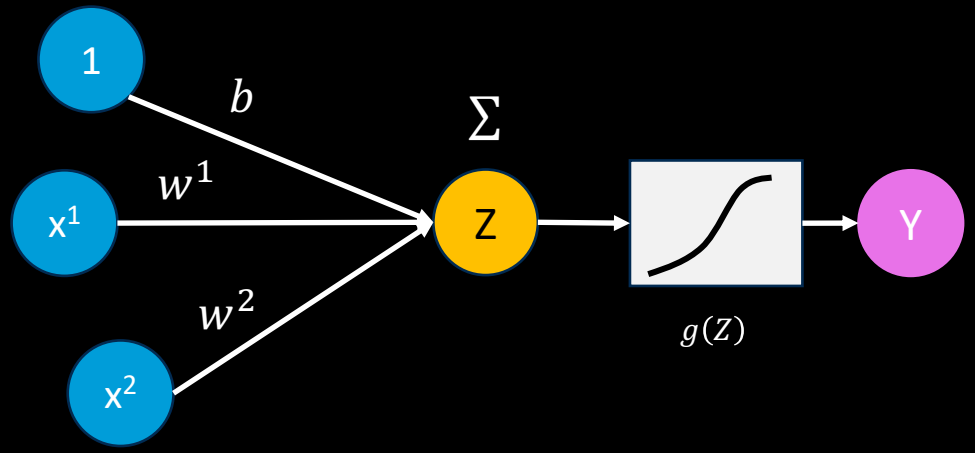



```python
self.activation = torch.nn.SiLU()
```

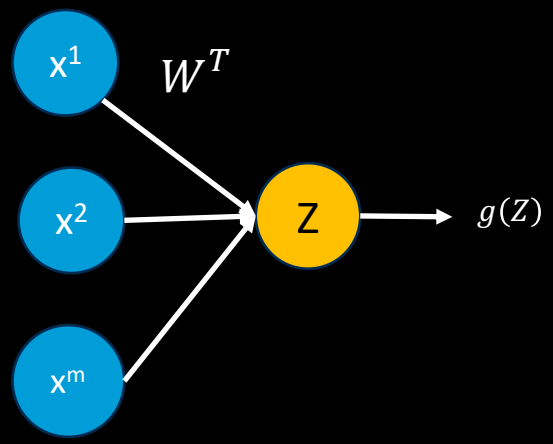



```python
self.activation = torch.nn.Tanh()
```

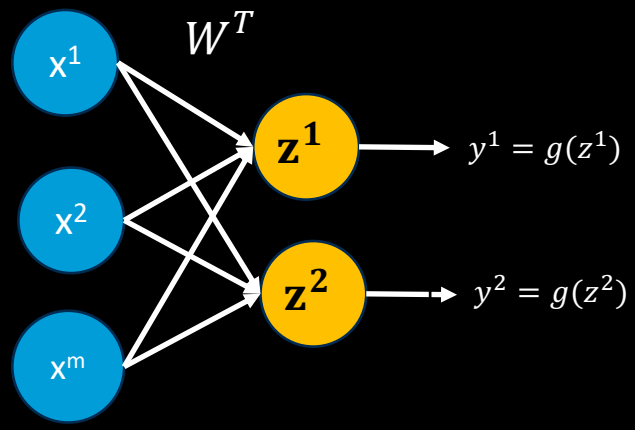
Building Neural Network using Perceptron



Building Neural Network using Perceptron

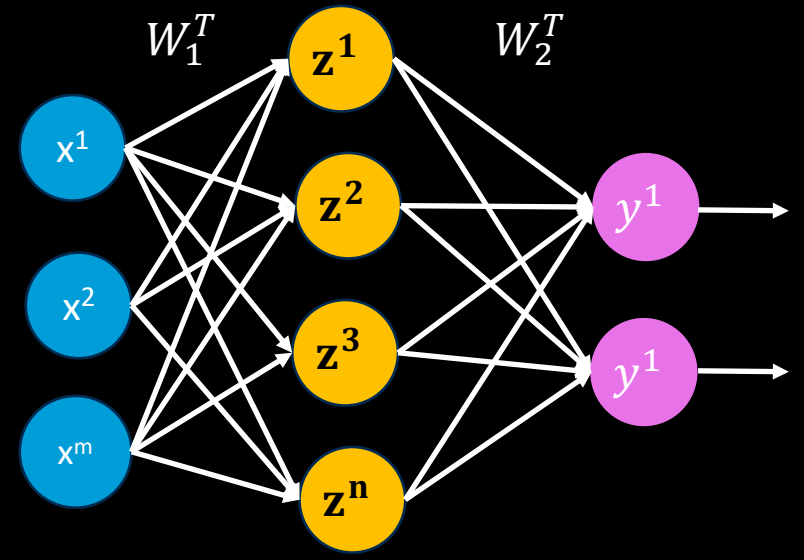


The Perceptron: Simplified



Multi Output Perceptron

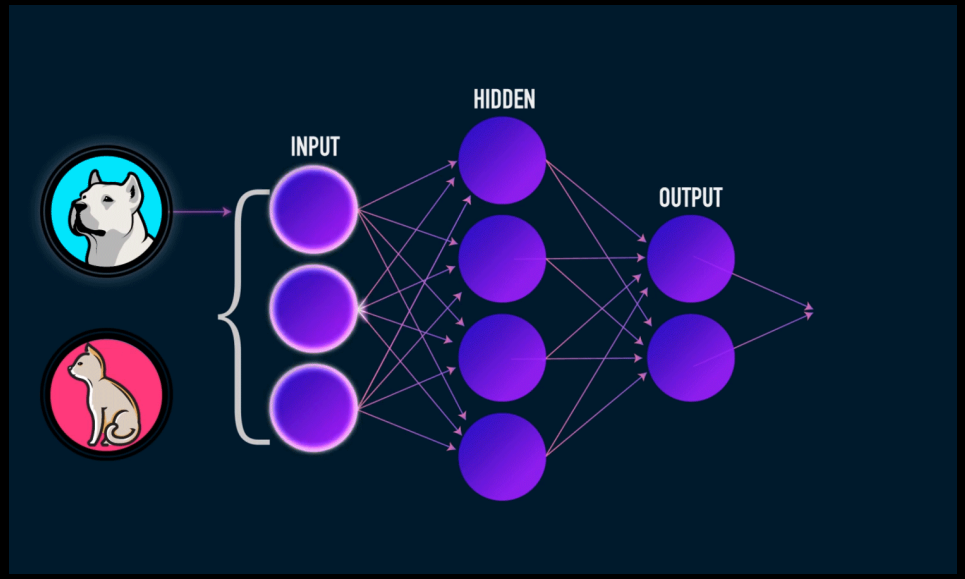
Dense Layers : All inputs are densely connected to all outputs

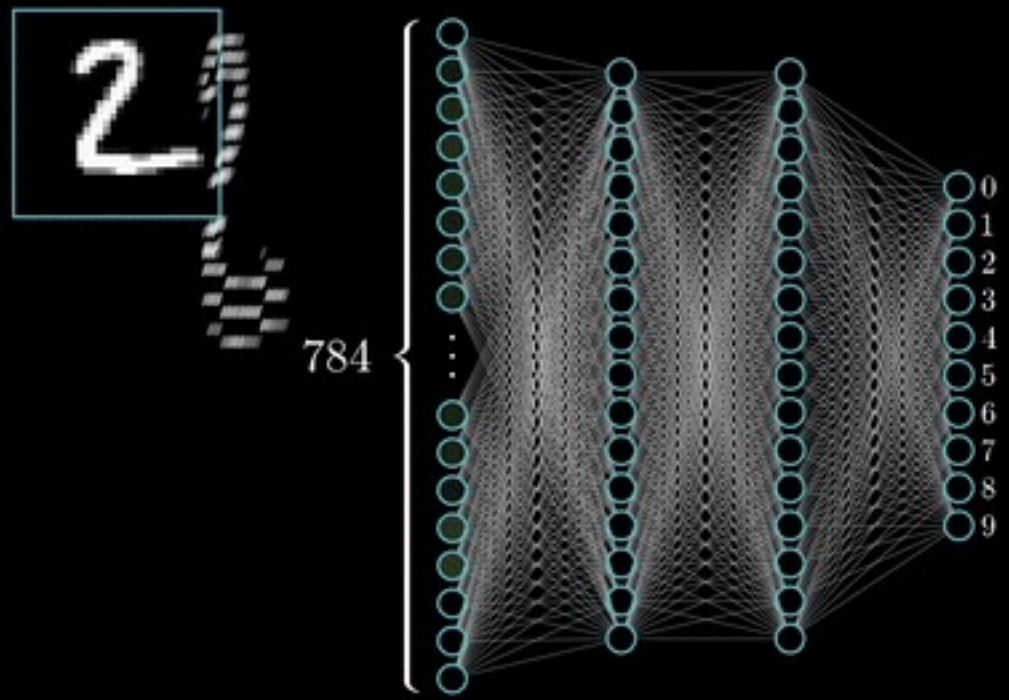


Single Layer Neural Network

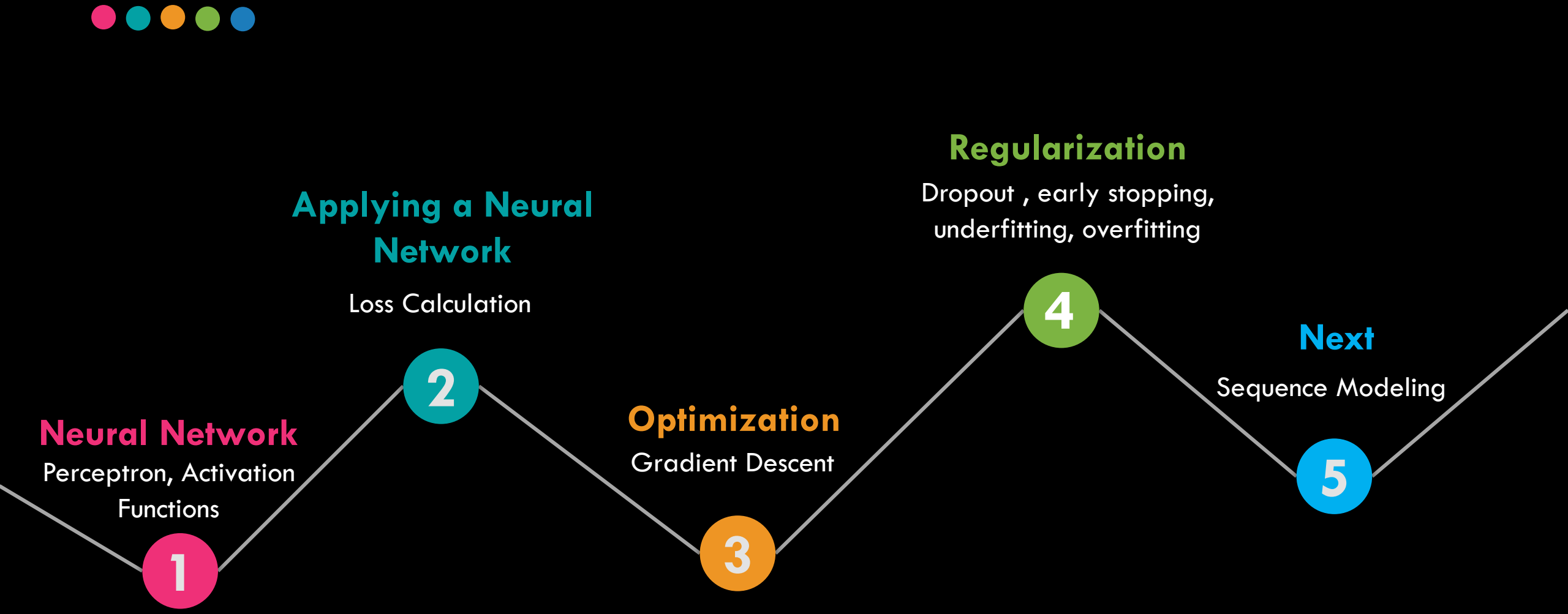
Multi-Layer Perceptron

Deep Neural Network

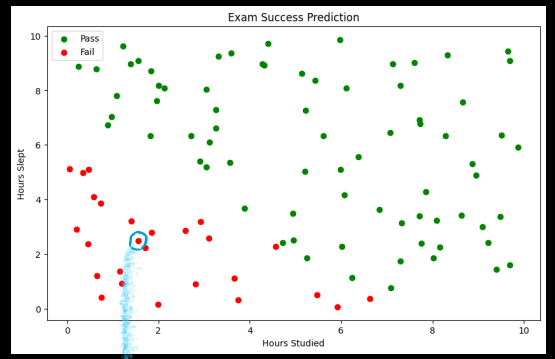




TODAY – Introduction to Deep Learning



Applying a Neural Network

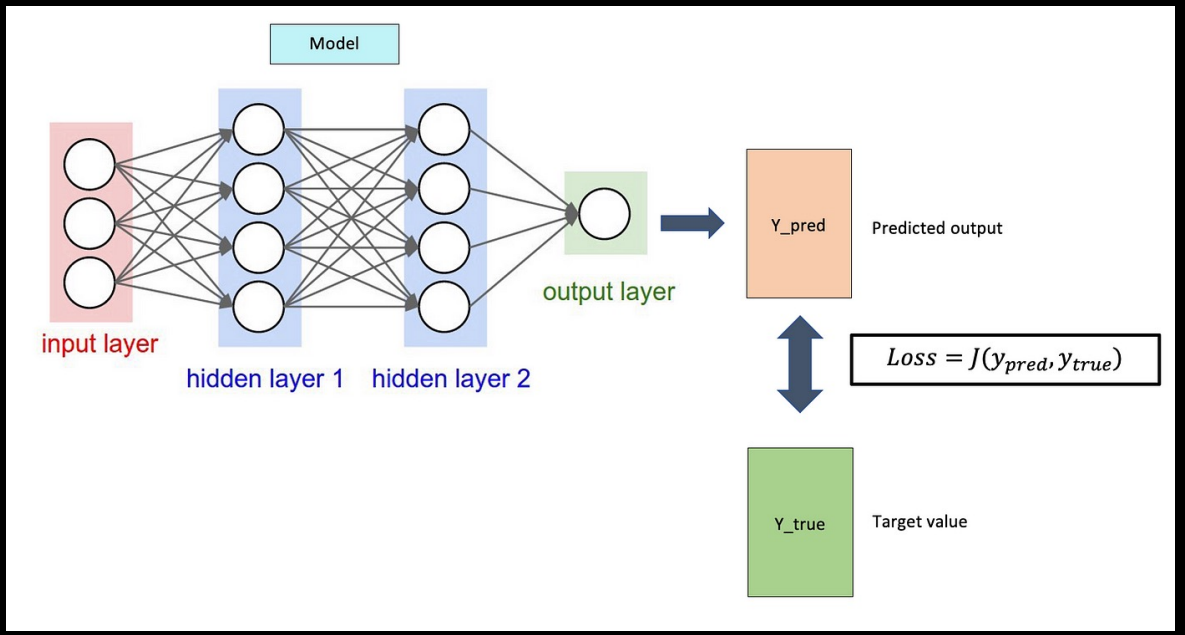


input $\begin{bmatrix} \text{studied} \\ \text{slept} \end{bmatrix} = \begin{bmatrix} 1.56 \\ 2.49 \end{bmatrix}$

Predicted $y = 1.0$

Actual $y = 0.0$

Calculating Loss



$$Loss = J(y_{pred}, y_{true})$$

Binary Cross Entropy Loss

```
criterion = torch.nn.BCEWithLogitsLoss()  # Bi
outputs = model(X_tensor)  # Forward pass
loss = criterion(outputs.squeeze(), Y_tensor)
```

- Binary Classification
- Output is probability between 0 and 1

Mean Squared Error

```
# Define loss function
criterion = torch.nn.MSELoss()
# Forward pass
outputs = model(X_train)
loss = criterion(outputs, y_train)
```

- Regression
- Penalizes larger errors more heavily than small errors

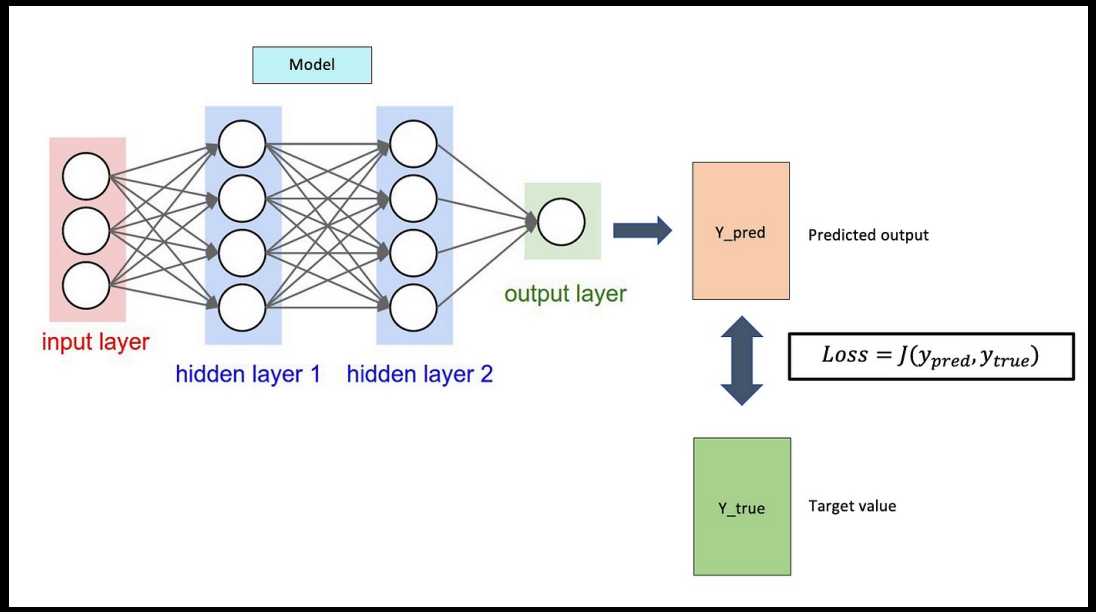
Mean Absolute Error

```
import torch.nn as nn

mae_loss = nn.L1Loss()
```

- Regression
- Sensitive to outlier data

Calculating Loss

$$Loss = J(y_{pred}, y_{true})$$



Huber Loss

```
import torch.nn as nn

huber_loss = nn.SmoothL1Loss()
```

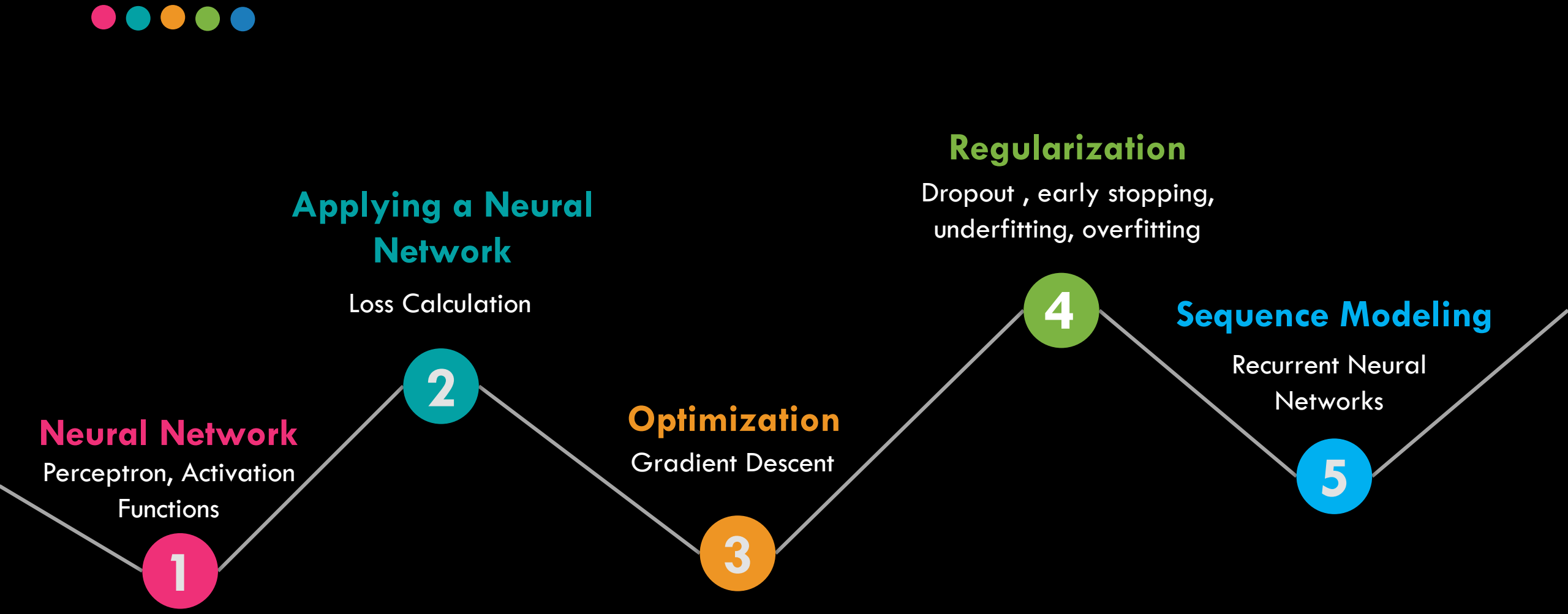
- Regression
- Combination of L1 and L2 loss

KL Divergence

```
import torch.nn as nn

kl_div_loss = nn.KLDivLoss(reduction='batchmean')
```

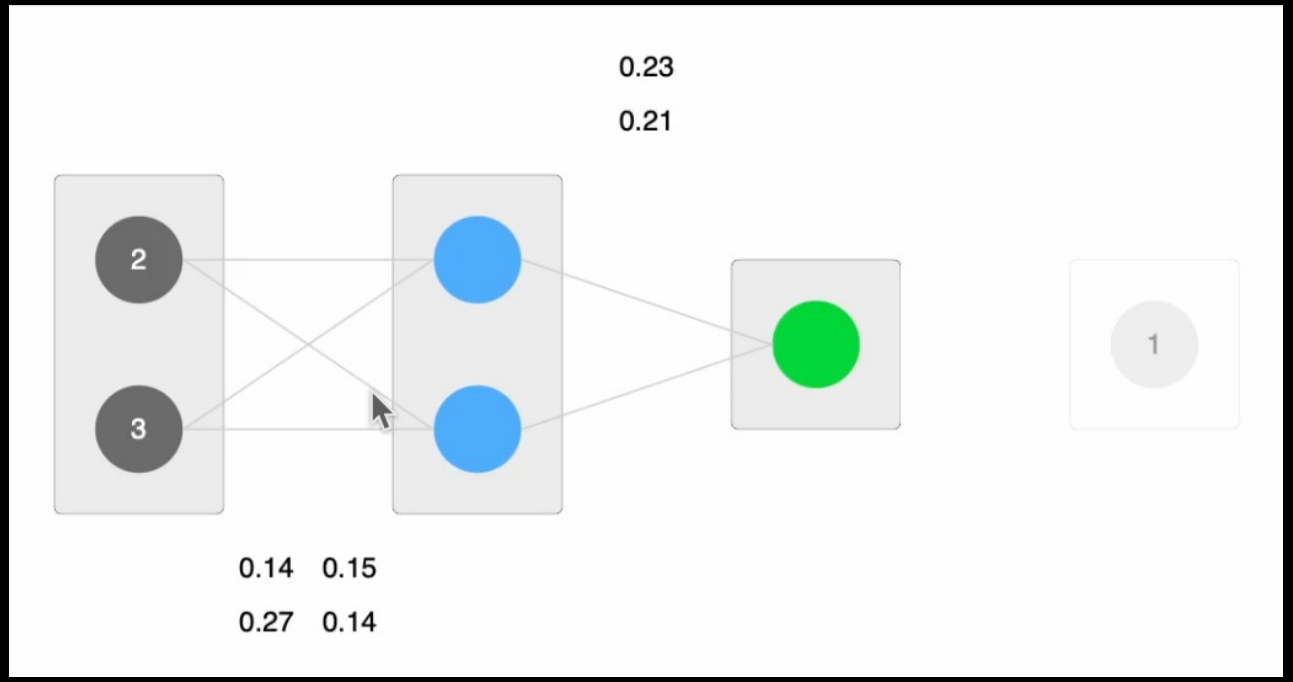
- Generative Modeling
- Measures difference between two probability distributions

TODAY – Introduction to Deep Learning



Optimization

Forward Pass



Our **OBJECTIVE** is to find new set of weights that decrease the loss.

Loss Optimization

Optimization

HoursStudied	HoursSlept	Pass
3.75	0.31	0.00
9.51	6.36	1.00
7.32	3.14	1.00
5.99	5.09	1.00
1.56	9.08	1.00
1.56	2.49	0.00
0.58	4.10	0.00
8.66	7.56	1.00
6.01	2.29	1.00
7.08	0.77	1.00
0.21	2.90	0.00
9.70	1.61	1.00
8.32	9.30	1.00
2.12	8.08	1.00
1.82	6.33	1.00
1.83	8.71	1.00
3.04	8.04	1.00
5.25	1.87	1.00
4.32	8.93	1.00
2.91	5.39	1.00

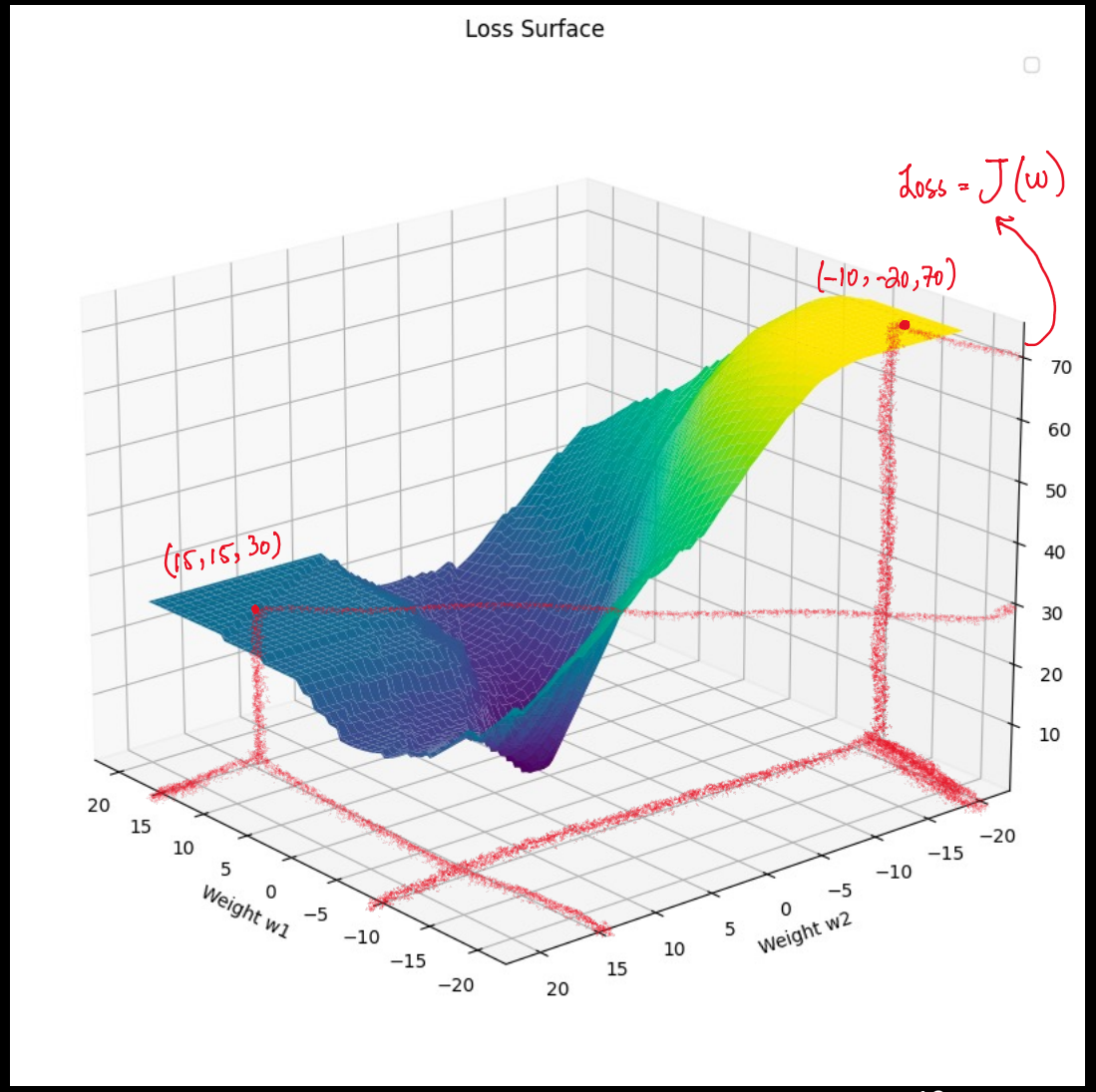
$$Pass = g(w^1 \times hours\_studied + w^2 \times hours\_slept + b)$$

```
# Define a mesh grid for w1 and w2
w1_values = np.linspace(-20, 20, 100)
w2_values = np.linspace(-20, 20, 100)
```

Loop → select w_1, w_2 from above list.

- Calculate Loss $J(w)$ for all input data.

Looping over the all w_1, w_2 values we get this 3D plot



Optimization

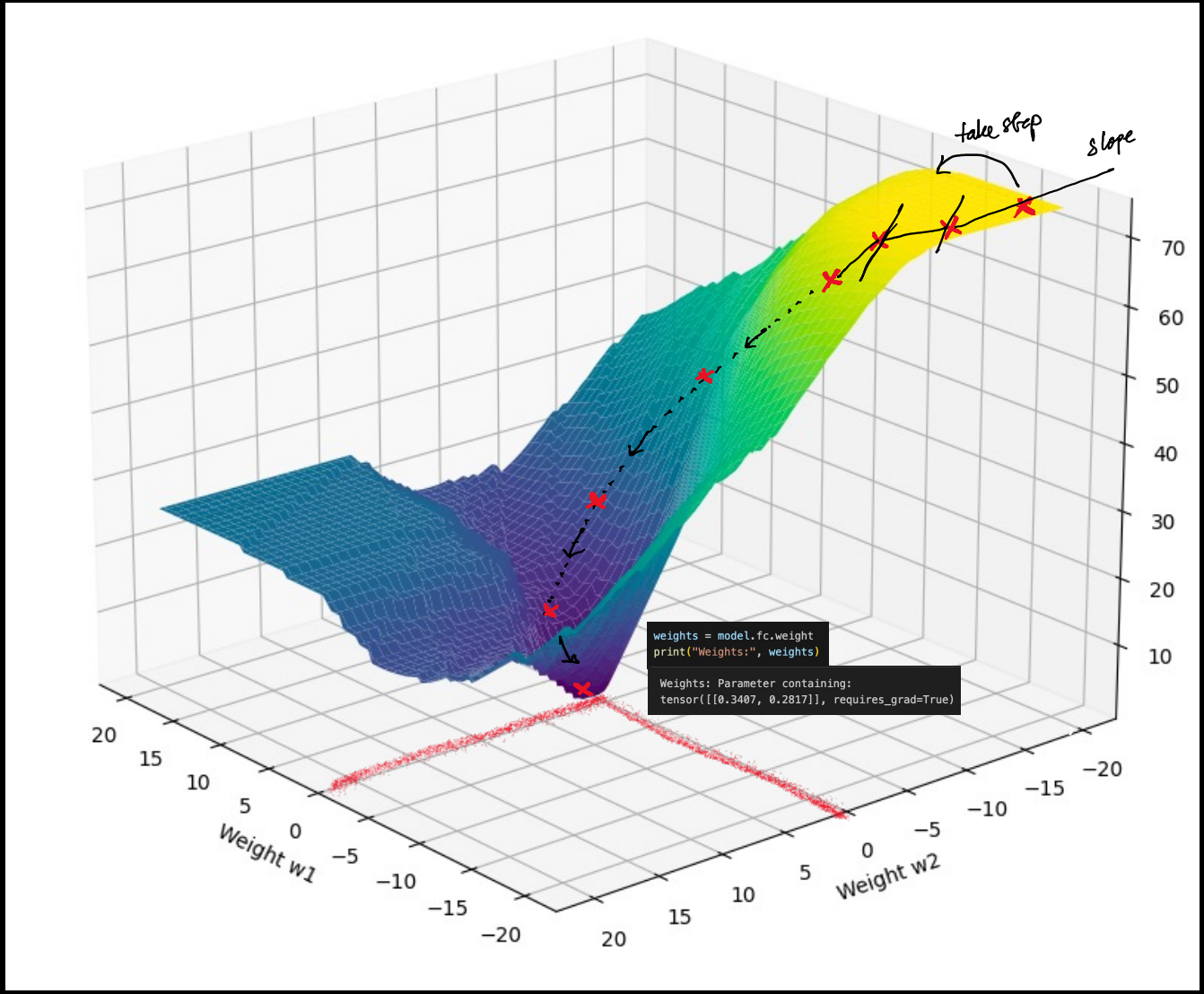
Loss Optimization

- Randomly select value for w_1, w_2

  $w_1 = -20$, $w_2 = 20$

- Calculate direction of change at this point - gradient

- Take next step opposite direction of slope ↳ Repeat above 3 steps



Terminology Alert!

These repeated set of actions are Gradient Descent

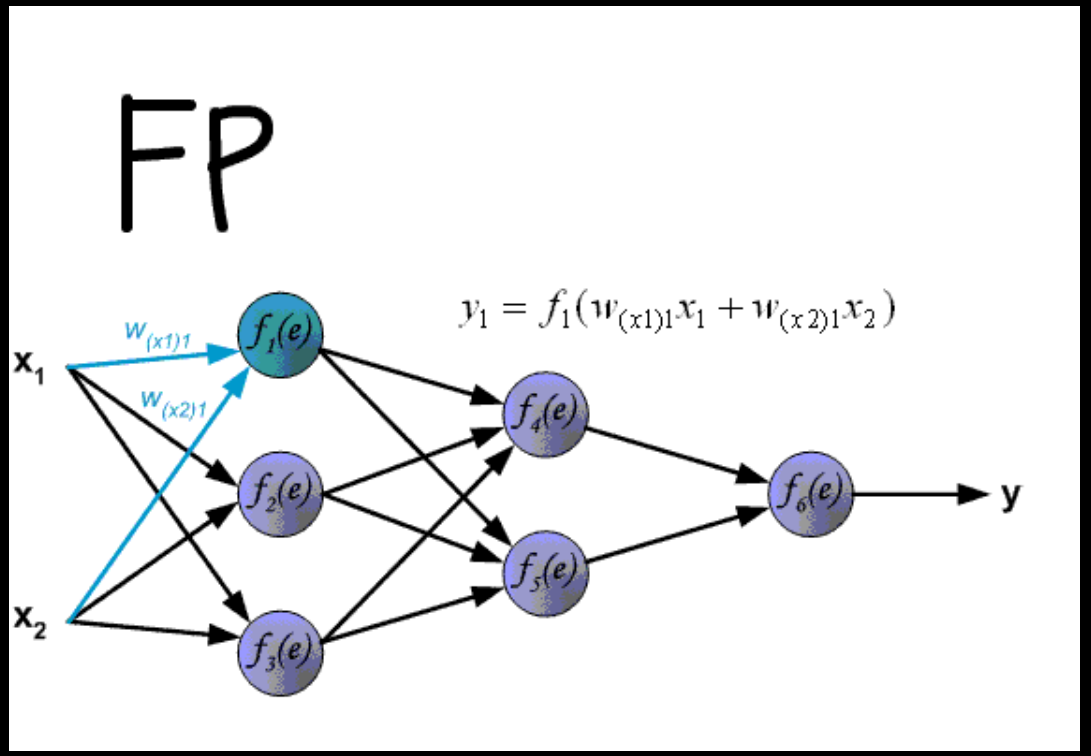
Step Size is the Learning Rate

Optimization

Gradient Descent

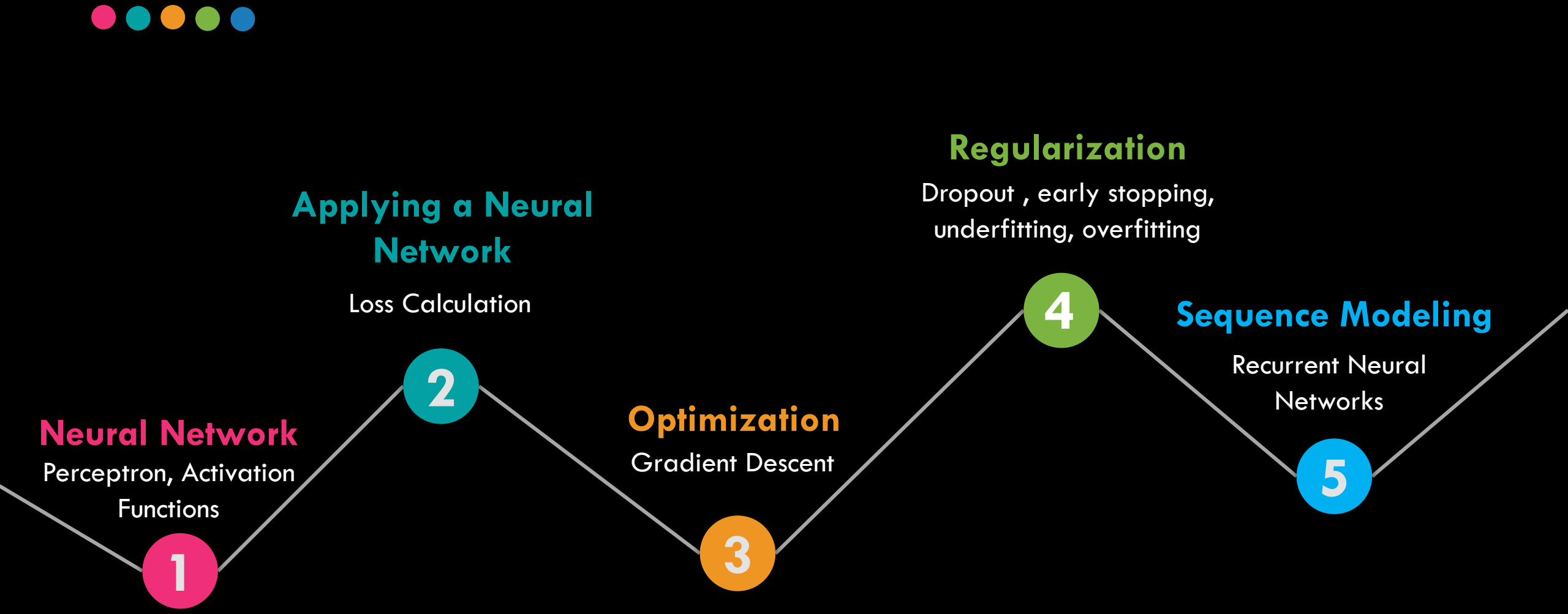
Algorithm

1. Initialize weights randomly
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ → slope
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights



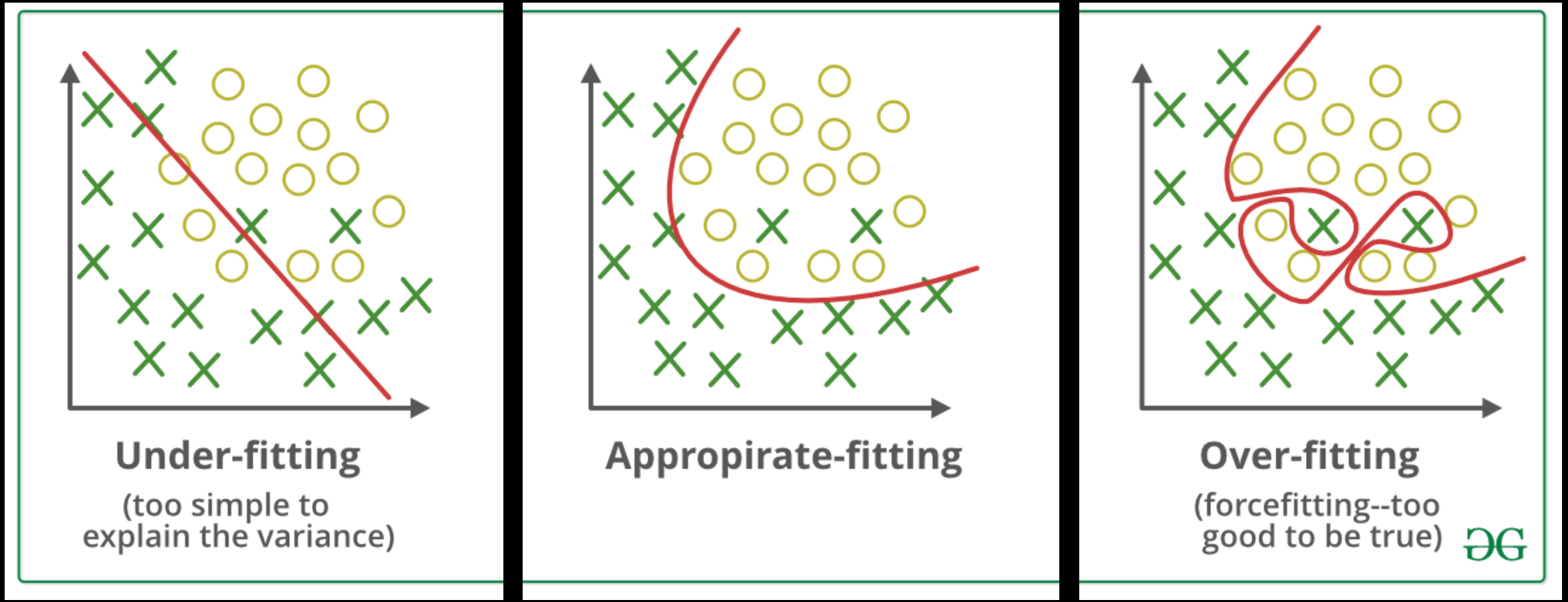
https://miro.medium.com/v2/resize:fit:1144/1*x73aS7hzk6tjv_fhh2IK3g.gif

TODAY – Introduction to Deep Learning



Regularization

Dropout , early stopping, underfitting, overfitting

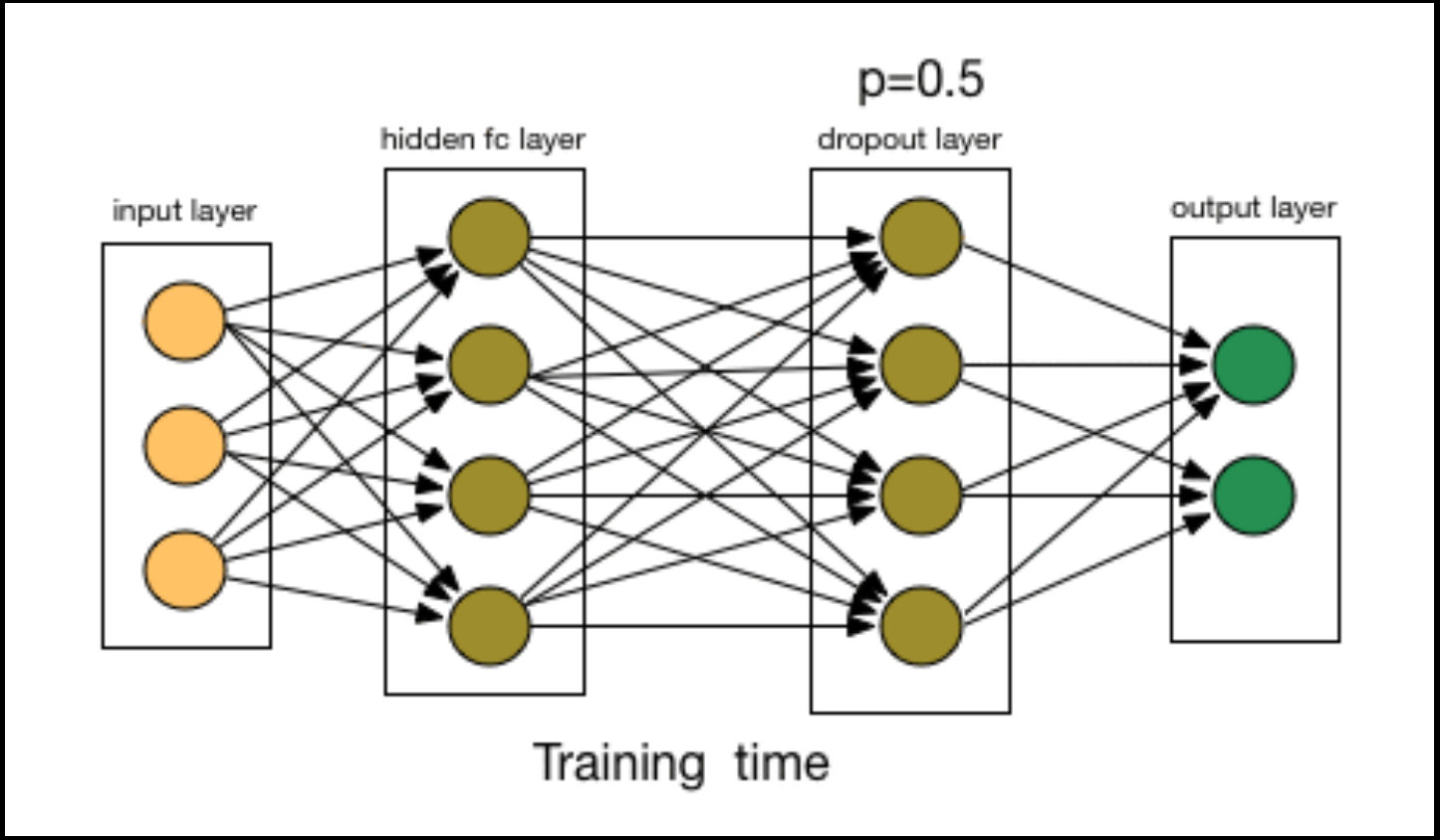


Regularization

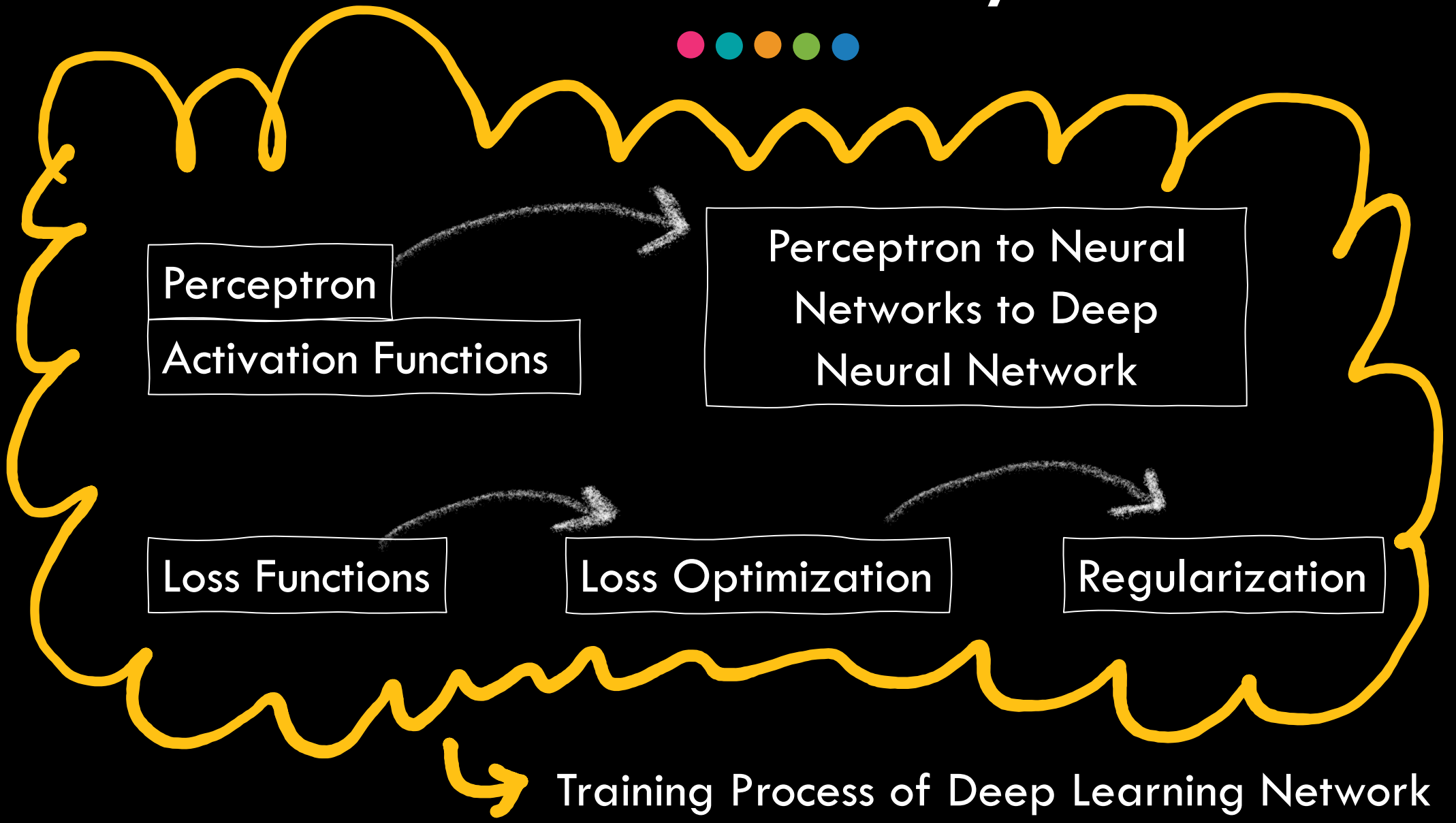
Dropout

During Training randomly set some activations to 0

- Typically drops 50% of activations in a layer
- Forces network to not rely on any one node



Reviewed Today



Putting it all together in Pytorch

```python
import torch
import torch.nn as nn
import torch.
from torch.ut
from sklearn.model_selection import train_test_split
import numpy as np

# Convert DataFrame to PyTorch tensors
X = torch.tensor(exam_data[['HoursStudied', 'HoursSlept']].values, dtype=torch.float32)
y = torch.tensor(exam_data['Pass'].values, dtype=torch.float32)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create DataLoader for training and testing sets
train_dataset = TensorDataset(X_train, y_train)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_dataset = TensorDataset(X_test, y_test)
test_loader = DataLoader(test_dataset, batch_size=32)

# Define the perceptron model
class Perceptron(nn.Module):
    def __init__(self, input_size):
        super(Perceptron, self).__init__()
        self.fc = nn.Linear(input_size, 1)  # Single output neuron

    def forward(self, x):
        x = torch.sigmoid(self.fc(x))  # Apply sigmoid activation function
        return x

# Initialize the model, loss function, and optimizer
model = Perceptron(input_size=2)
criterion = nn.BCELoss()  # Binary cross-entropy loss
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

(module) model_selection

Prepare Dataset

input vector

output vector

Break the data into train, test

pass only 32 input points in one iteration

Define Model

Initialize Model

```python
# Training loop
num_epochs = 100
for epoch in range(num_epochs):
    model.train()  # Set model to training mode
    for inputs, targets in train_loader:
        optimizer.zero_grad()  # Zero the gradients
        outputs = model(inputs)  # Forward pass
        loss = criterion(outputs.squeeze(), targets)  # Calculate the loss
        loss.backward()  # Backward pass
        optimizer.step()  # Update weights

    # Evaluate the model on the test set
    model.eval()  # Set model to evaluation mode
    with torch.no_grad():
        test_loss = 0
        total_correct = 0
        total_samples = 0
        for inputs, targets in test_loader:
            outputs = model(inputs)
            test_loss += criterion(outputs.squeeze(), targets).item()
            predicted = (outputs >= 0.5).float()  # Convert to binary predictions
            total_correct += (predicted == targets).sum().item()
            total_samples += targets.size(0)

        # Print metrics
        test_loss /= len(test_loader)
        accuracy = total_correct / total_samples
        print(f'Epoch [{epoch+1}/{num_epochs}], Test Loss: {test_loss:.4f}, Test Ac
```

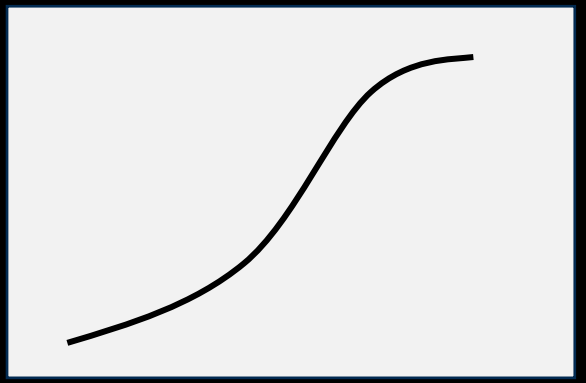
Training Loop

Model Evaluation

References

- http://introtodeeplearning.com/
- lucid by openai
- microscope by openai
- Chris Olah's blog

Activation Function



$g(Z)$