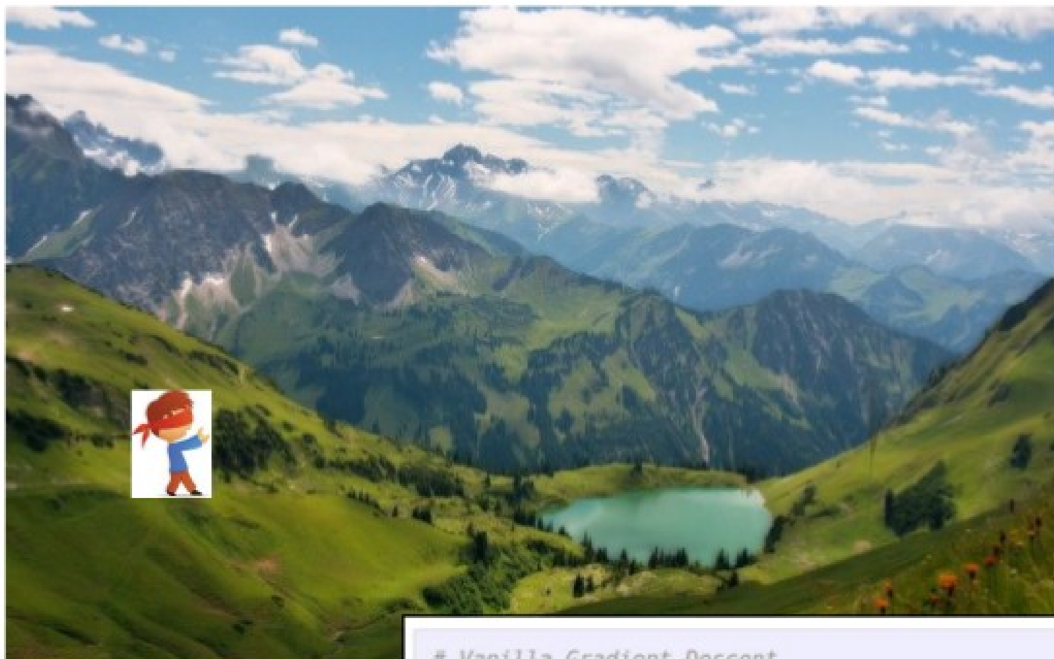
Multilayer Neural Network

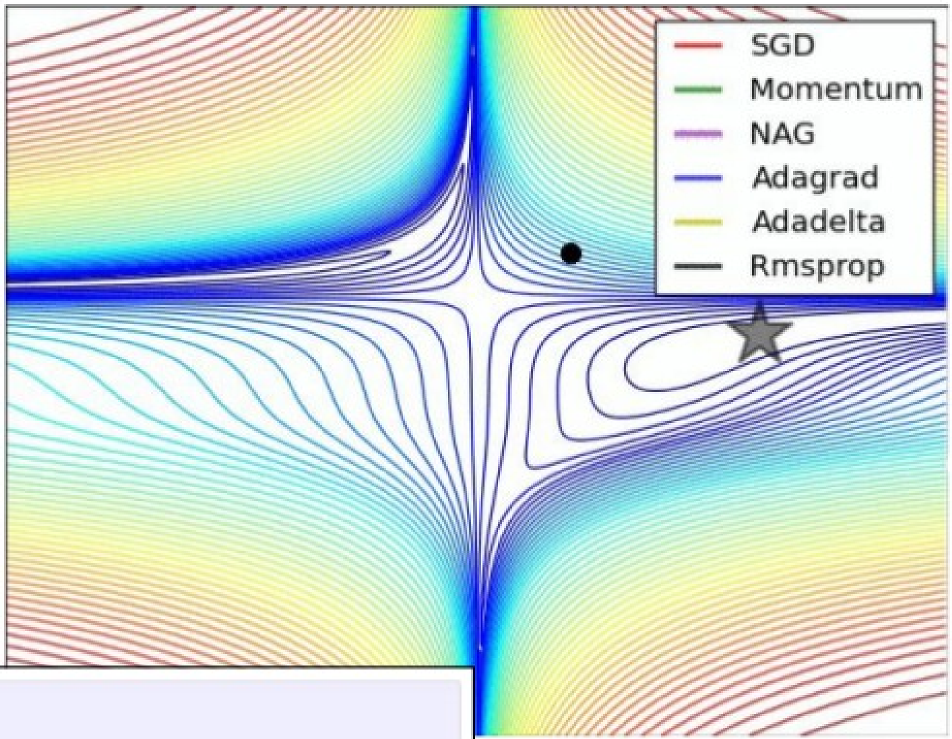
Jony Sugianto
jony@evolvemachinelearners.com
0812-13086659
github.com/jonysugianto

Neural Network Application

https://www.youtube.com/watch?v=hPKJBXkyTKM

Visualizing Error Function






```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

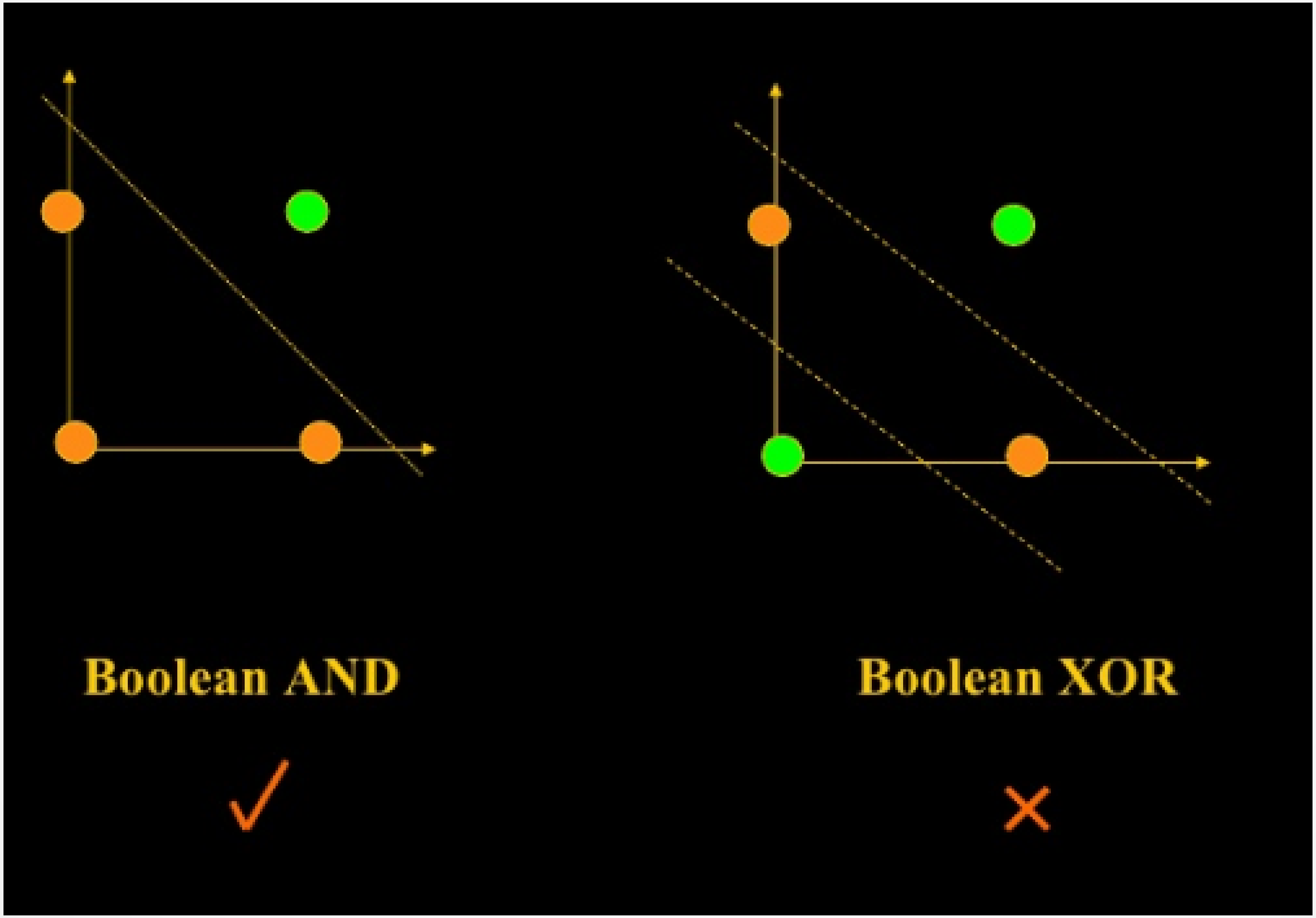
Limitation of Perceptron

- The perceptron can only model linearly separable functions,
  - those functions which can be drawn in 2-dim graph and single straight line separates values in two part.

  Boolean functions given below are linearly separable:
  - AND
  - OR
  - COMPLEMENT

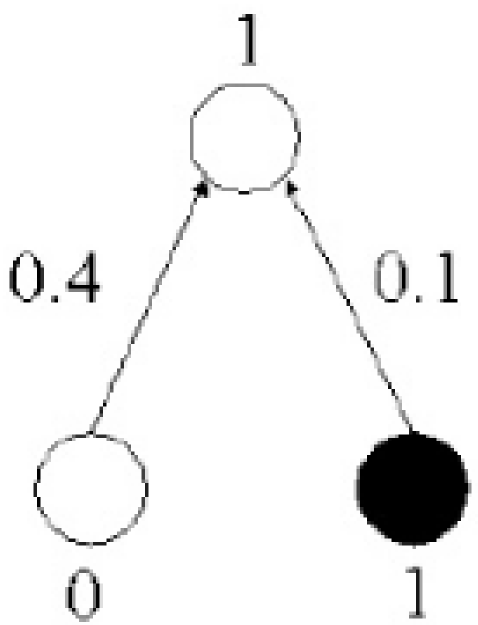
  It cannot model XOR function as it is non linearly separable.
  - When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

Limitation of Perceptron

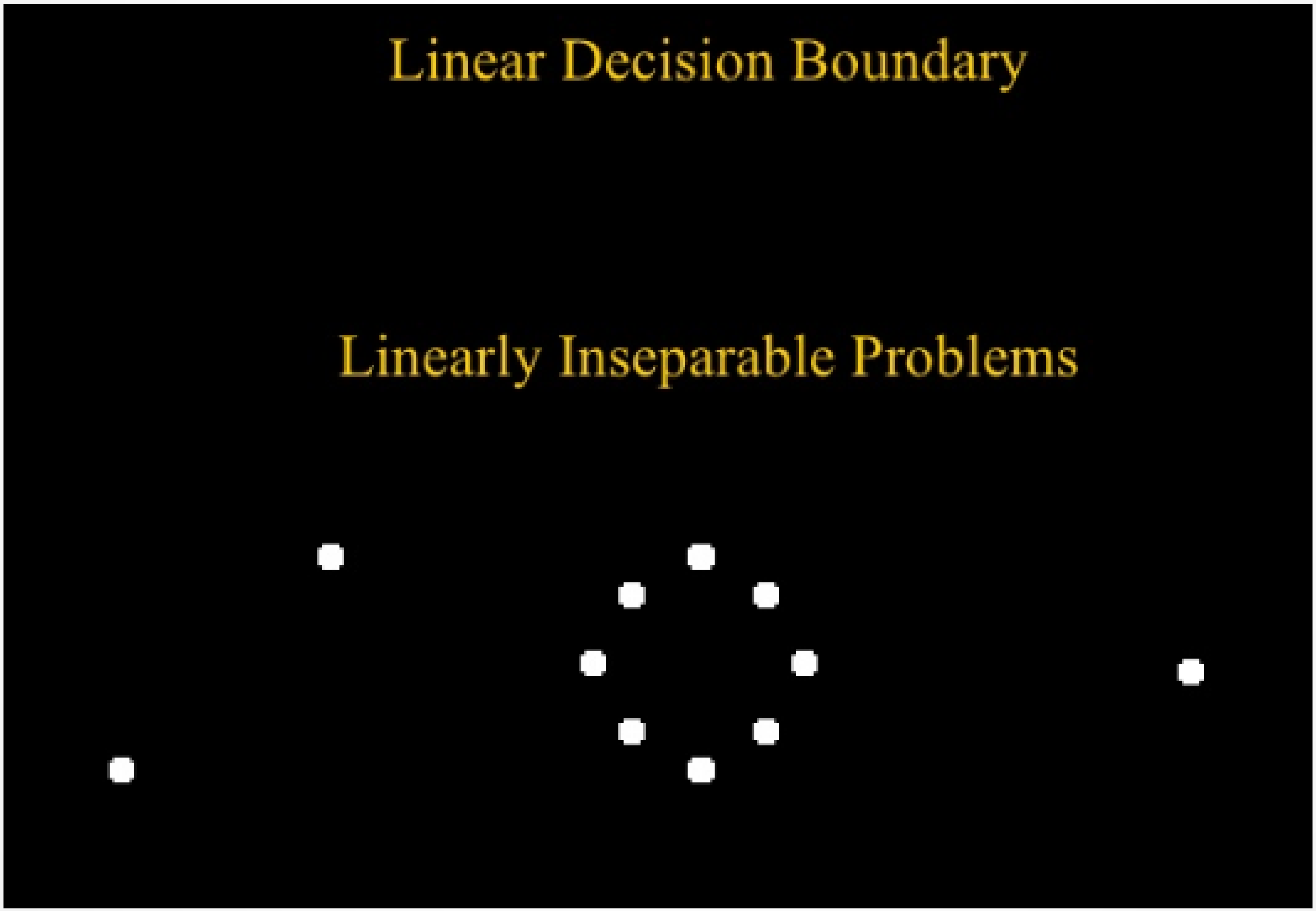


Limitation of Perceptron

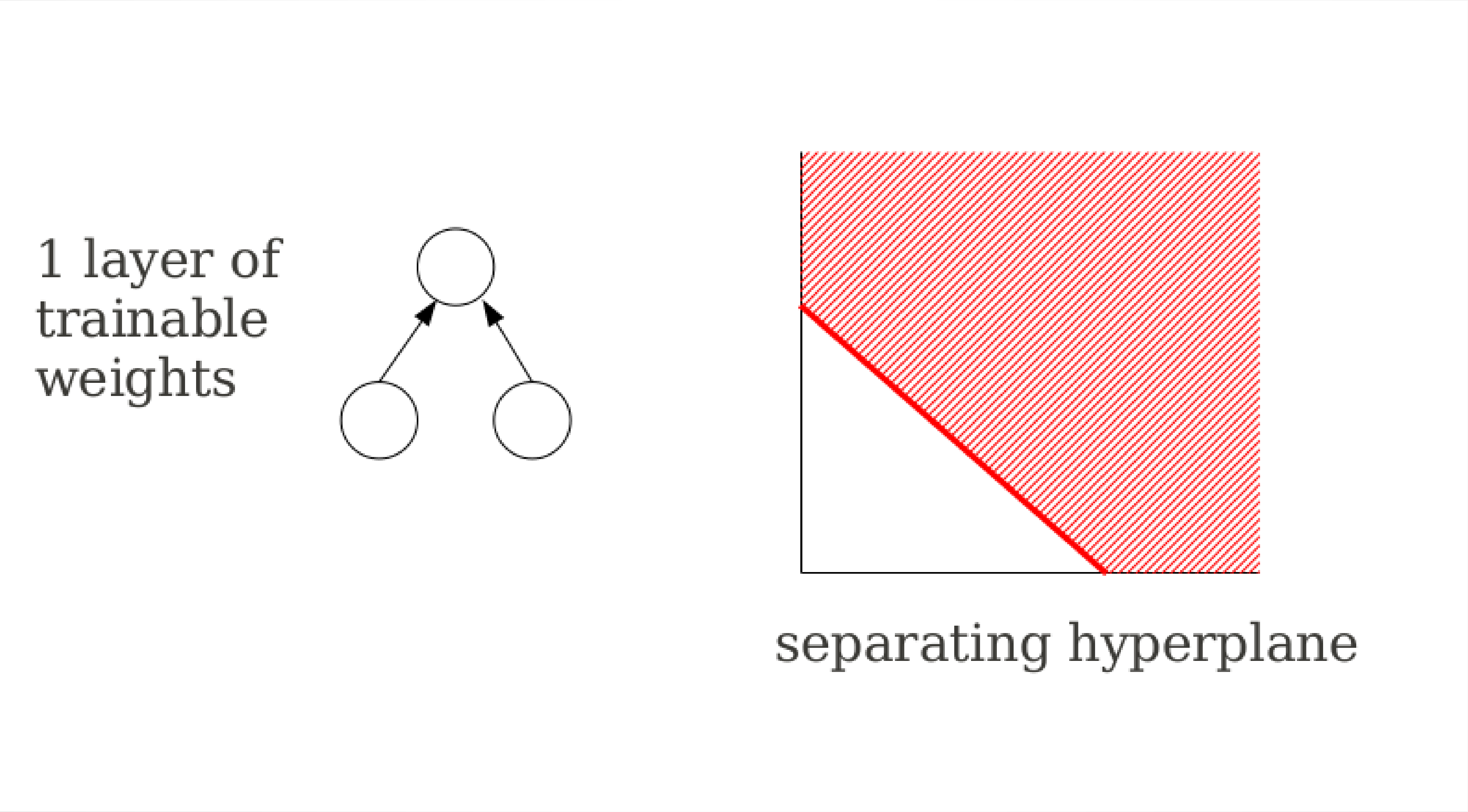
In	Out
0 1	1
1 0	1
1 1	0
0 0	0



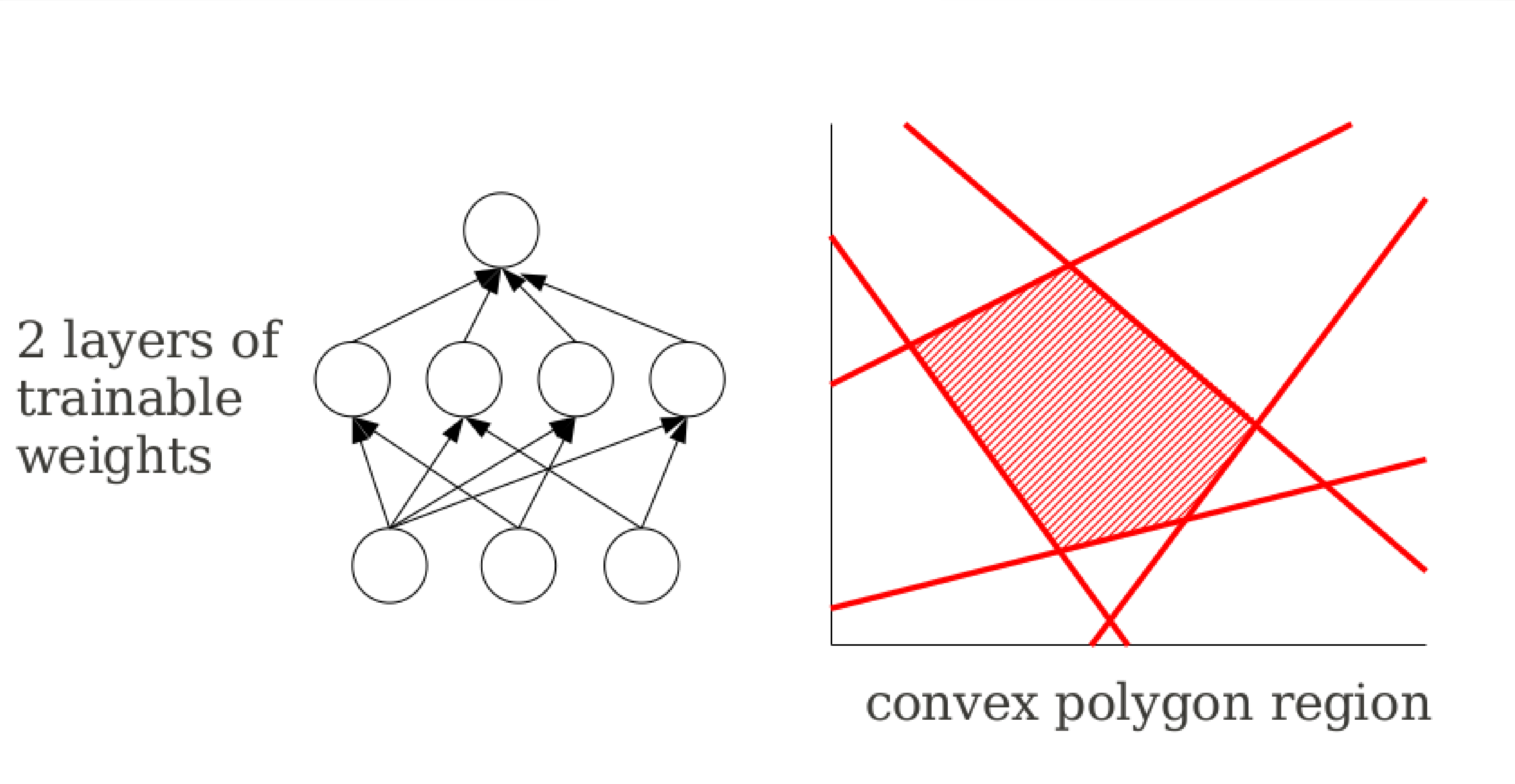
Limitation of Perceptron



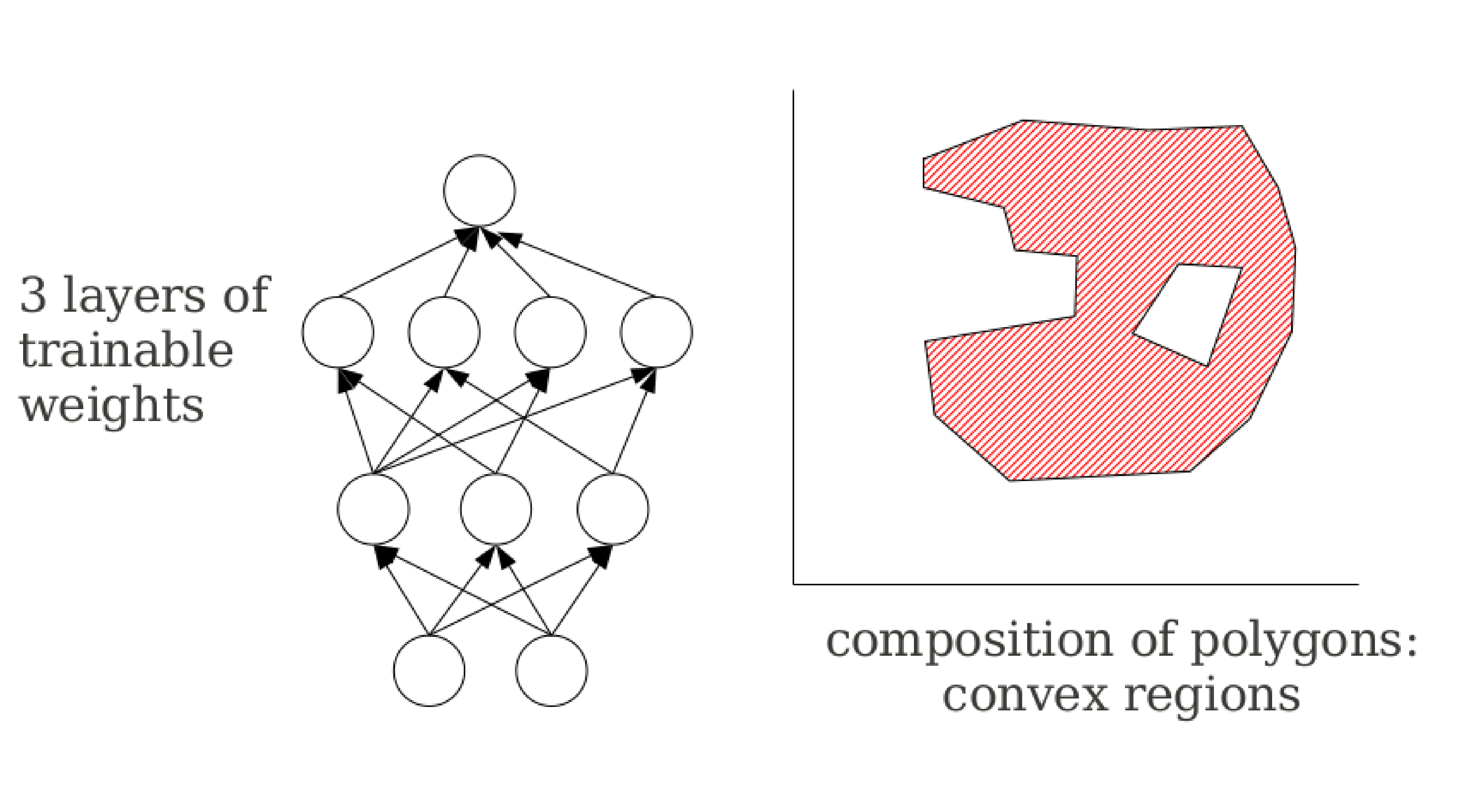
From Single Layer to Multilayer



From Single Layer to Multilayer

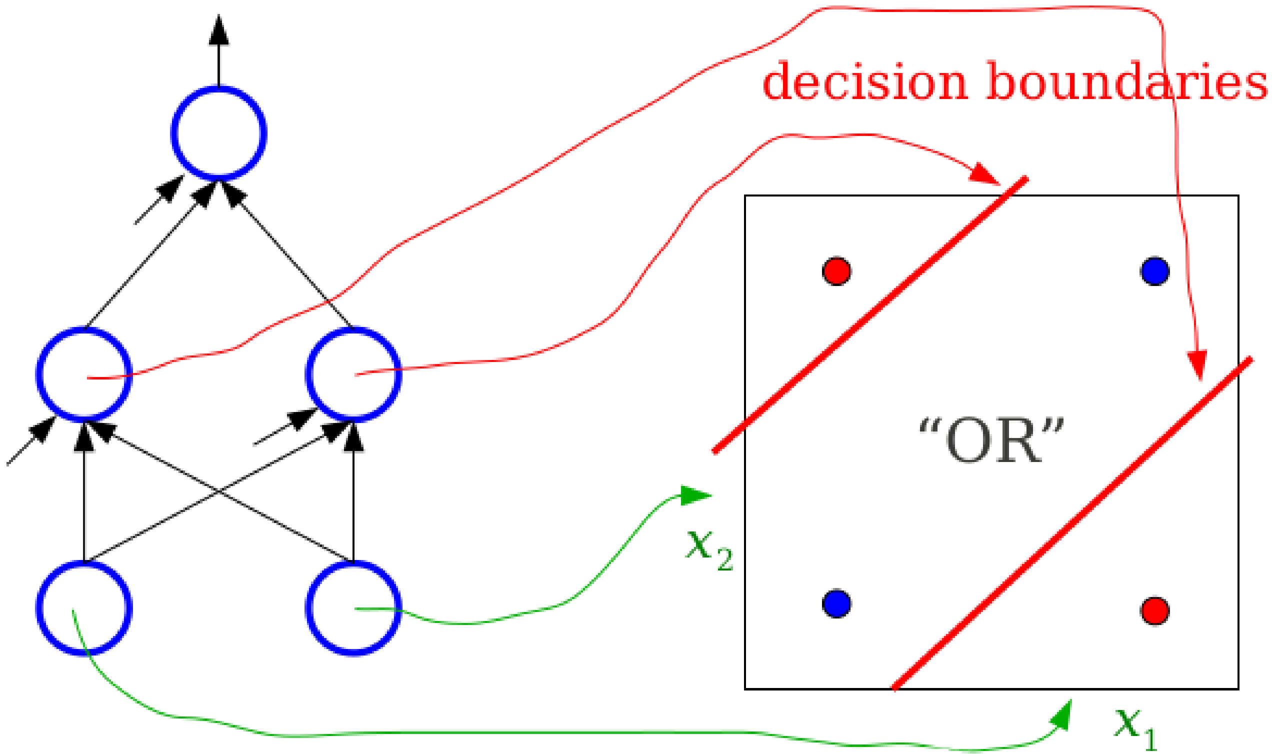


From Single Layer to Multilayer

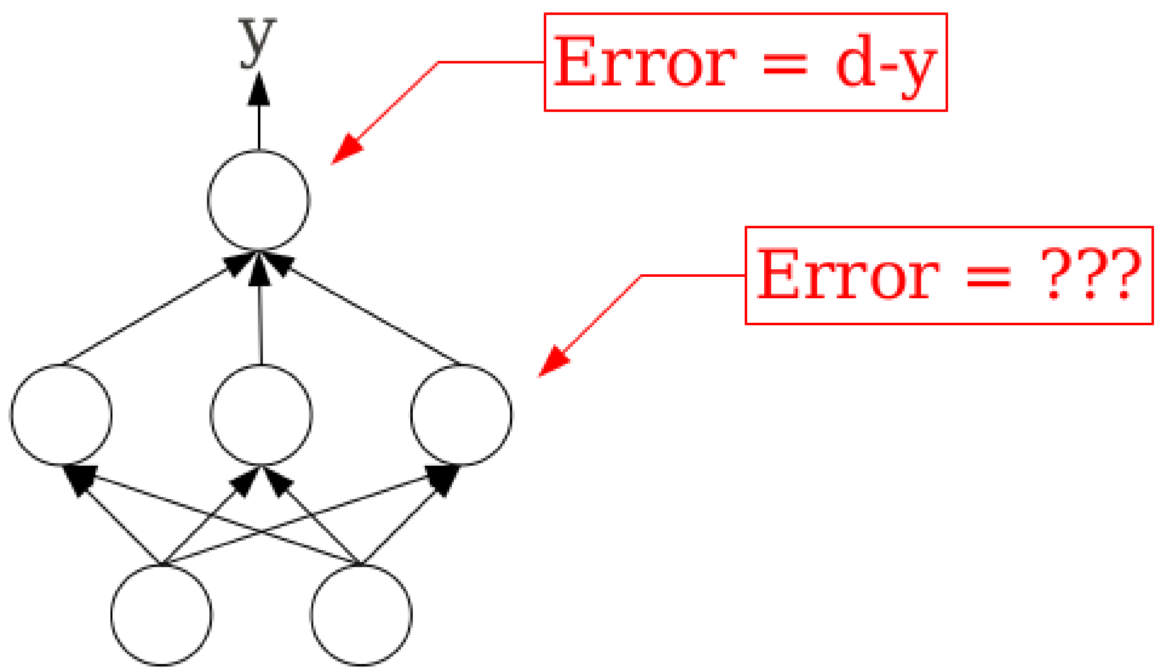


XOR Solution

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



How Do We Train A Multi-Layer Network?



Can't use perceptron training algorithm because we don't know the 'correct' outputs for hidden units.

How Do We Train A Multi-Layer Network?

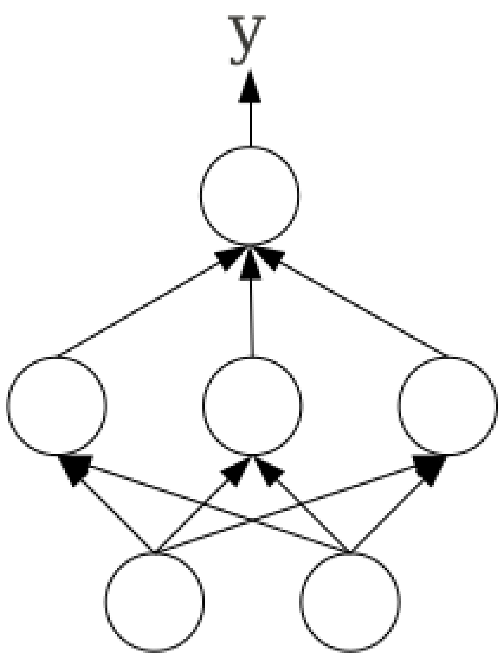
Define sum-squared error:

$$E = \frac{1}{2}\sum_{p}(d^p - y^p)^2$$

Use gradient descent error minimization:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Works if the nonlinear transfer function is differentiable.



Switch to Smooth Nonlinear Units

$$\text{net}_j = \sum_i w_{ij} y_i$$

$$y_j = g(\text{net}_j)$$

g must be differentiable

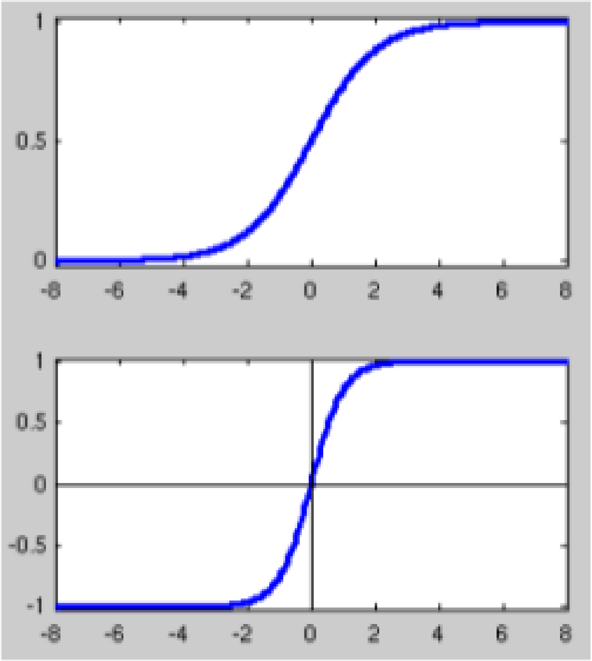
Common choices for g:

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

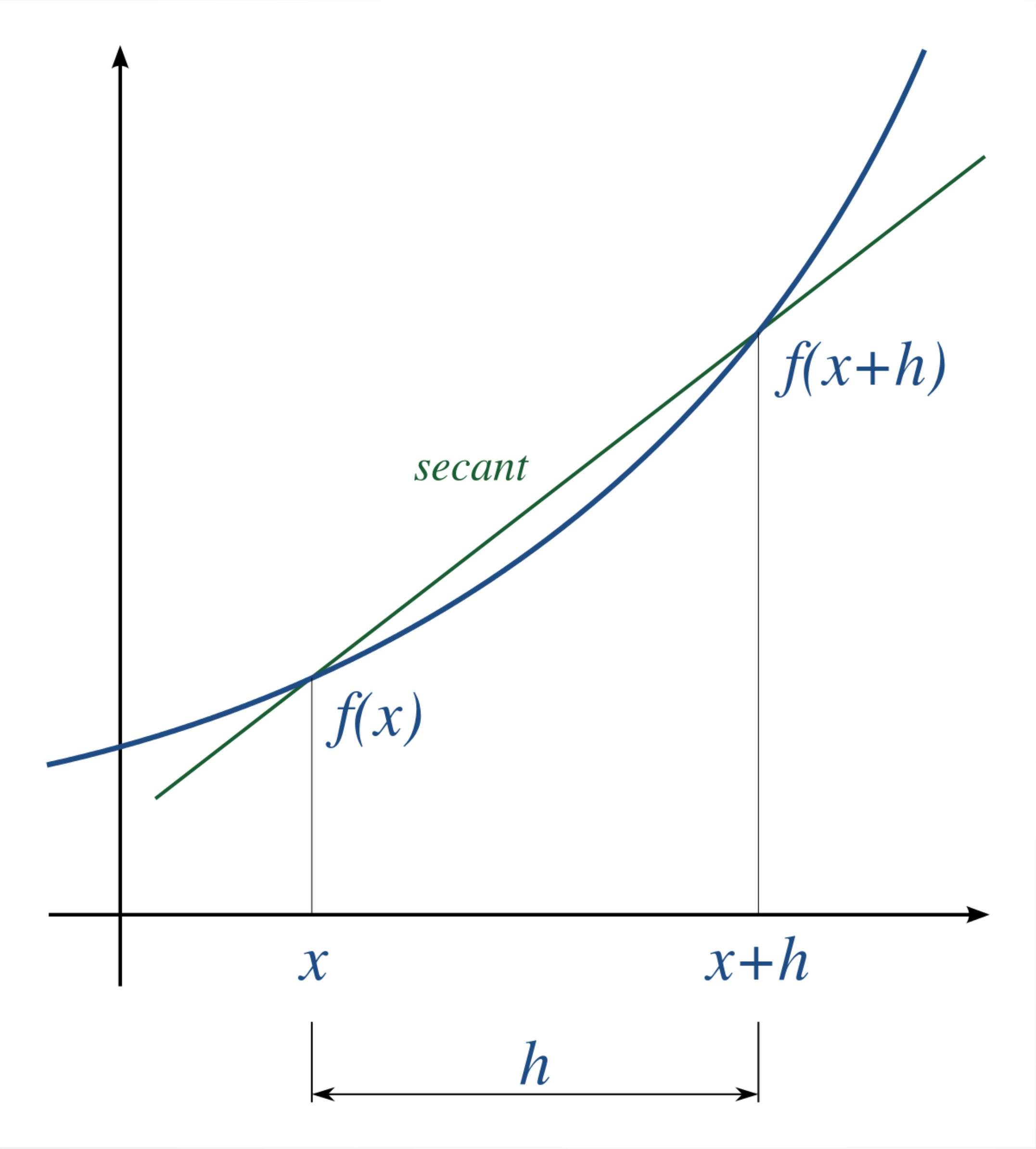
$$g'(x) = g(x) \cdot (1 - g(x))$$

$$g(x) = \tanh(x)$$

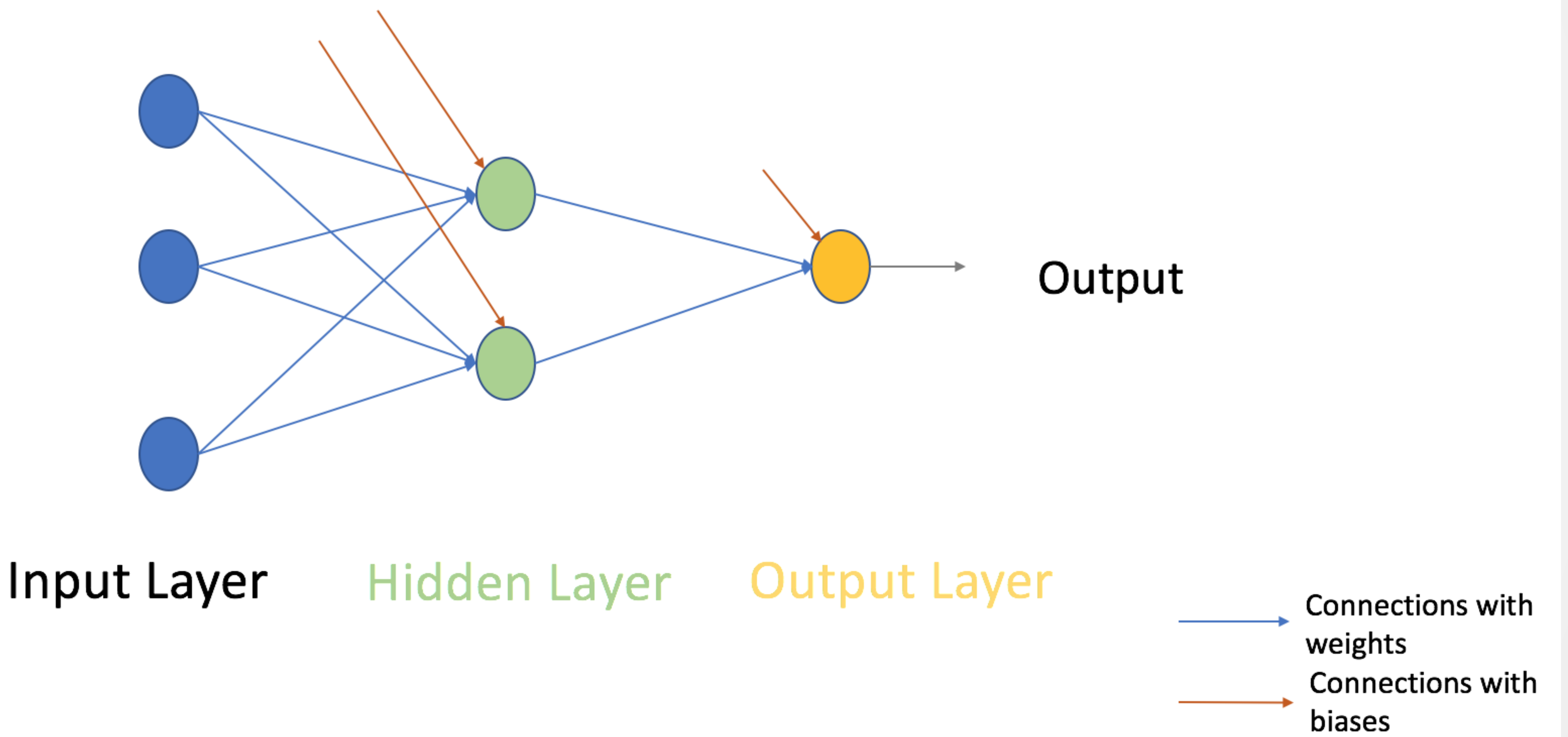
$$g'(x) = 1/\cosh^2(x)$$



Intermezzo Computing Derivative using numerical Differentiation



Multi Layer Neural Network



MLP Definitions

- X as an input matrix
- y as an output matrix
- wh as weight matrix to the hidden layer
- bh as bias matrix to the hidden layer
- wout as weight matrix to the output layer
- bout as bias matrix to the output layer

Forward Propagation

- sigmoid activation function:

  1/(1+exp(-x))

- hiddenlayer_input=matrix_dot_product(X, wh)+bh
- hiddenlayer_activations=sigmoid(hiddenlayer_input)

- outputlayer_input=matrix_dot_product(hiddenlayer_activations,wout)+bout
- output=sigmoid(outputlayer_input)

Backward Propagation

- Mean Square loss=((y-output)^2)/2
- Derivative of Mean Square Error:

  y-output
- sigmoid activation function:

  sigmoig(x)=1/(1+exp(-x))
- Derivatives of sigmoid function:

  sigmoid(x)*(1-sigmoid(x))

Backward Propagation

- Compute the gradient of error:

  E=y-output

- Compute the slope/gradient of outputlayer:

  slope_outputlayer=derivatives_sigmoid(output)

- Compute the slope/gradient of hiddenlayer:

  slope_hiddenlayer=derivatives_sigmoid(hiddenlayer_activations)

- Compute the change factor(delta) at output layer:

  d_output=E*slope_output_layer

- Compute the error at hidden layer:

  error_at_hiddenlayer=matrix_dot_product(d_output, wout.Transpose)

- Compute change factor(delta) at hiddenlayer:

  d_hiddenlayer=error_at_hiddenlayer*slope_hiddenlayer

Backward Propagation

- Update weights at outputlayer:

  wout=wout+matrix_dot_product(hiddenlayer_activations.Transpose, d_output)*learningrate

- Update biases at outputlayer:

  bout=bout+sum(d_output)*learningrate

- Update weights at hiddenlayer:

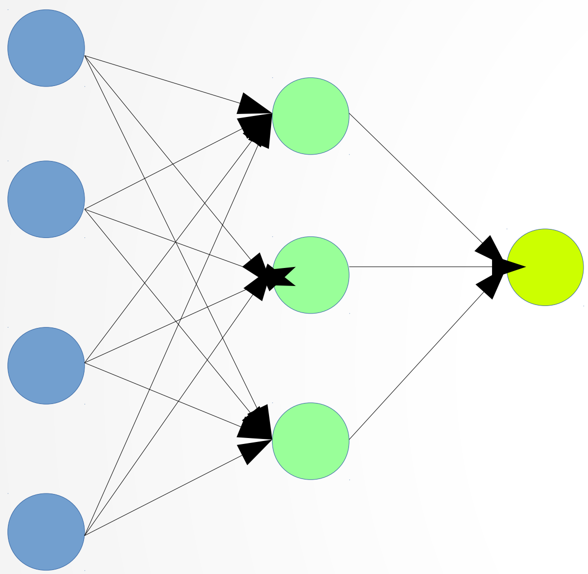
  wh=wh+matrix_dot_product(X.Transpose, d_hiddenlayer)*learningrate

- Update biases at hiddenlayer:

  bh=bh+sum(d_hiddenlayer)*learningrate

Visualization steps for neural network

MLP with 2 Layers



Dataset

```
#Input array
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])

#Output
y=np.array([[1],[1],[0]])
```

Visualization steps for neural network

Step-0: Read input and output

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0																1	
1	0	1	1																1	
0	1	0	1																0	

Visualization steps for neural network

Step-1: Initialize weights and biases

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08							0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68										0.25			1	
0	1	0	1	0.60	0.18	0.47										0.23			0	
				0.92	0.11	0.52														

Visualization steps for neural network

Step-2: Calculate hidden layer input
hiddenlayer_input=matrix_dot_product(X,wh)+bh

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10				0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61				0.25			1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27				0.23			0	
				0.92	0.11	0.52														

Visualization steps for neural network

Step-3: perform non-linear transformation on hiddenlayer_input
hiddenlayer_activations=sigmoid(hiddenlayer_input)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69		1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25			1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23			0	
				0.92	0.11	0.52														

Visualization steps for neural network

Step-4: perform linear and non-linear transformation of hiddenlayer_activation at outputlayer
outputlayer_input=matrix_dot_product(hiddenlayer_activations, wout)+bout
output=sigmoid(outpulayer_input)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	
				0.92	0.11	0.52														

Visualization steps for neural network

Step-5: Calculate gradient of Error(E) at outputlayer
E=y-output

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Visualization steps for neural network

Step-6: Compute slope at output and hiddenlayer
slope_outputlayer=derivatives_sigmoid(output)
slope_hiddenlayer=derivatives_sigmoid(hiddenlayer_activations)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer		
0.15	0.12	0.19
0.08	0.11	0.14
0.15	0.14	0.17

Slope Output
0.17
0.16
0.17

Visualization steps for neural network

Step-7: Compute delta at outputlayer
d_output=E * slope_outputlayer*learningrate

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer		
0.15	0.12	0.19
0.08	0.11	0.14
0.15	0.14	0.17

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

delta output
0.04
0.03
-0.13

Visualization steps for neural network

Step-8: Calculate error at hiddenlayer
error_at_hiddenlayer=matrix_dot_product(d_output, wout.Transpose)

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

delta output
0.04
0.03
-0.13

Visualization steps for neural network

Step-9: Compute delta at at hiddenlayer
d_hiddenlayer=error_at_hiddenlayer*slope_hiddenlayer

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.30	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.52														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

delta output
0.04
0.03
-0.13

Visualization steps for neural network

Step-10: Update weights at output and hiddenlayer
wout = wout + matrix_dot_product(hiddenlayer_activations.Transpose, d_output)*learning_rate
wh = wh+ matrix_dot_product(X.Transpose,d_hiddenlayer)*learning_rate

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.29	0.69	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.51														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

Learning Rate	0.1

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

delta output
0.035
0.033
-0.131

Visualization steps for neural network

Step-11: Update biases at output and hiddenlayer
bh = bh + sum(d_hiddenlayer, axis=0) * learning_rate
bout = bout + sum(d_output, axis=0)*learning_rate

X				wh			bh			hidden_layer_input			hidden_layer_activations			wout	bout	output	y	E
1	0	1	0	0.42	0.88	0.55	0.46	0.72	0.08	1.48	1.78	1.10	0.81	0.86	0.75	0.29	0.68	0.79	1	0.21
1	0	1	1	0.10	0.73	0.68				2.40	1.89	1.61	0.92	0.87	0.83	0.25		0.80	1	0.20
0	1	0	1	0.60	0.18	0.47				1.48	1.56	1.27	0.81	0.83	0.78	0.23		0.79	0	-0.79
				0.92	0.11	0.51														

Slope hidden layer			error at hidden layer		
0.15	0.12	0.19	0.010	0.009	0.008
0.08	0.11	0.14	0.010	0.008	0.008
0.15	0.14	0.17	-0.039	-0.033	-0.031

Slope Output
0.17
0.16
0.17

E
0.21
0.20
-0.79

Learning Rate	0.1

delta hidden layer		
0.002	0.001	0.002
0.001	0.001	0.001
-0.006	-0.005	-0.005

delta output
0.035
0.033
-0.131

Visualization steps for neural network

Dataset

```
import numpy as np
#Input array
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])
#Output
y=np.array([[1],[1],[0]])
```

Visualization steps for neural network

Sigmoid and derivative of sigmoid

```
#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))
#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)
```

Visualization steps for neural network

Variable Initialization

```
#Variable initialization
epoch=5000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = X.shape[1] #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
```

Visualization steps for neural network

Weights and bias Initialization

```
#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
```

Visualization steps for neural network

Forward Propagation

```
#Forward Propogation
hidden_layer_input1=np.dot(X,wh)
hidden_layer_input=hidden_layer_input1 + bh
hiddenlayer_activations = sigmoid(hidden_layer_input)
output_layer_input1=np.dot(hiddenlayer_activations,wout)
output_layer_input= output_layer_input1+ bout
output = sigmoid(output_layer_input)
```

Visualization steps for neural network
Forward Propagation

```
#Backward Propagation
E = y-output
slope_output_layer = derivatives_sigmoid(output)
slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)
d_output = E * slope_output_layer
Error_at_hidden_layer = d_output.dot(wout.T)
d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer
wout += hiddenlayer_activations.T.dot(d_output) *lr
bout += np.sum(d_output, axis=0,keepdims=True) *lr
wh += X.T.dot(d_hiddenlayer) *lr
bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr
```

Visualization steps for neural network

Training

```
#Training
for i in range(epoch):
    #forward_propagation
    ...
    #backward_propagation
    ...

print(output)
```

Initialize MLP

EVOLVE MACHINE LEARNERS

```
from random import seed
from random import random

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
	network = list()
	hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)]} for i in range(n_hidden)]
	network.append(hidden_layer)
	output_layer = [{'weights':[random() for i in range(n_hidden + 1)]} for i in range(n_outputs)]
	network.append(output_layer)
	return network
```

Test initialize MLP

```
seed(1)
network = initialize_network(2, 1, 2)
for layer in network:
	print(layer)
```

Layer Activation

```
# Calculate neuron activation for an input
def activate(weights, inputs):
	activation = weights[-1]
	for i in range(len(weights)-1):
		activation += weights[i] * inputs[i]
	return activation
```

Activation Function

```
# activation function
def sigmoid(activation):
	return 1.0 / (1.0 + exp(-activation))

def tanh(activation):
	return (2*sigmoid(activation)) – 1.0
```

Forward Propagation

```
# Forward propagate input to a network output
def forward_propagate(network, row):
	inputs = row
	for layer in network:
		new_inputs = []
		for neuron in layer:
			activation = activate(neuron['weights'], inputs)
			neuron['output'] = transfer(activation)
			new_inputs.append(neuron['output'])
		inputs = new_inputs
	return inputs
```

Test Forward Propagation

```
# test forward propagation
network = [[{'weights': [0.13436424411240122, 0.8474337369372327, 0.763774618976614]}],
		[{'weights': [0.2550690257394217, 0.49543508709194095]}, {'weights':
[0.4494910647887381, 0.651592972722763]}]]
row = [1, 0, None]
output = forward_propagate(network, row)
print(output)
```

Activation Function derivative

```
# Calculate the derivative of sigmoid
def sigmoid_derivative(output):
	return output * (1.0 - output)
```

Backward Propagation

```
# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
	for i in reversed(range(len(network))):
		layer = network[i]
		errors = list()
		if i != len(network)-1:
			for j in range(len(layer)):
				error = 0.0
				for neuron in network[i + 1]:
					error += (neuron['weights'][j] * neuron['delta'])
				errors.append(error)
		else:
			for j in range(len(layer)):
				neuron = layer[j]
				errors.append(expected[j] - neuron['output'])
		for j in range(len(layer)):
			neuron = layer[j]
			neuron['delta'] = errors[j] * sigmoid_derivative(neuron['output'])
```

Test Backward propagation

```
# test backpropagation of error
network = [[{'output': 0.7105668883115941, 'weights': [0.13436424411240122,
0.8474337369372327, 0.763774618976614]}],
		[{'output': 0.6213859615555266, 'weights': [0.2550690257394217,
0.49543508709194095]}, {'output': 0.6573693455986976, 'weights': [0.4494910647887381,
0.651592972722763]}]]
expected = [0, 1]
backward_propagate_error(network, expected)
for layer in network:
	print(layer)
```

Update Weights

```
# Update network weights with error
def update_weights(network, row, l_rate):
	for i in range(len(network)):
		inputs = row[:-1]
		if i != 0:
			inputs = [neuron['output'] for neuron in network[i - 1]]
		for neuron in network[i]:
			for j in range(len(inputs)):
				neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
			neuron['weights'][-1] += l_rate * neuron['delta']
```

Train Network

```
# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):
	for epoch in range(n_epoch):
		sum_error = 0
		for row in train:
			outputs = forward_propagate(network, row)
			expected = [0 for i in range(n_outputs)]
			expected[row[-1]] = 1
			sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
			backward_propagate_error(network, expected)
			update_weights(network, row, l_rate)
		print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
```

Test Train Network

```
# Initialize a network
# Test training backprop algorithm
seed(1)
dataset = [[2.7810836,2.550537003,0],
	[1.465489372,2.362125076,0],
	[3.396561688,4.400293529,0],
	[1.38807019,1.850220317,0],
	[3.06407232,3.005305973,0],
	[7.627531214,2.759262235,1],
	[5.332441248,2.088626775,1],
	[6.922596716,1.77106367,1],
	[8.675418651,-0.242068655,1],
	[7.673756466,3.508563011,1]]
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
for layer in network:
	print(layer)
```

Prediction

```
# Make a prediction with a network
def predict(network, row):
	outputs = forward_propagate(network, row)
	return outputs.index(max(outputs))
```

Test Prediction

```
# Test making predictions with the network
ataset = [[2.7810836,2.550537003,0],
	[1.465489372,2.362125076,0],
	[3.396561688,4.400293529,0],
	[1.38807019,1.850220317,0],
	[3.06407232,3.005305973,0],
	[7.627531214,2.759262235,1],
	[5.332441248,2.088626775,1],
	[6.922596716,1.77106367,1],
	[8.675418651,-0.242068655,1],
	[7.673756466,3.508563011,1]]
etwork = [[{'weights': [-1.482313569067226, 1.8308790073202204, 1.078381922048799]},
'weights': [0.23244990332399884, 0.3621998343835864, 0.40289821191094327]}],
	[{'weights': [2.5001872433501404, 0.7887233511355132, -1.1026649757805829]}, {'weights':
-2.429350576245497, 0.8357651039198697, 1.0699217181280656]}]]
or row in dataset:
	prediction = predict(network, row)
	print('Expected=%d, Got=%d' % (row[-1], prediction))
```