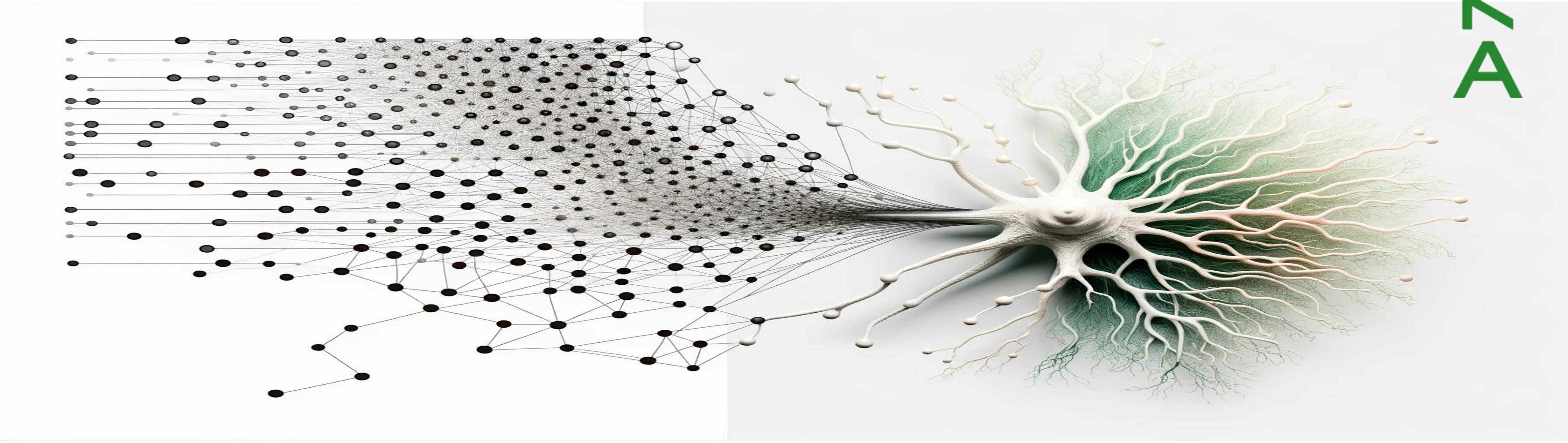
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Elektro- und
Informationstechnik

Artificial Neural Networks (ANN)



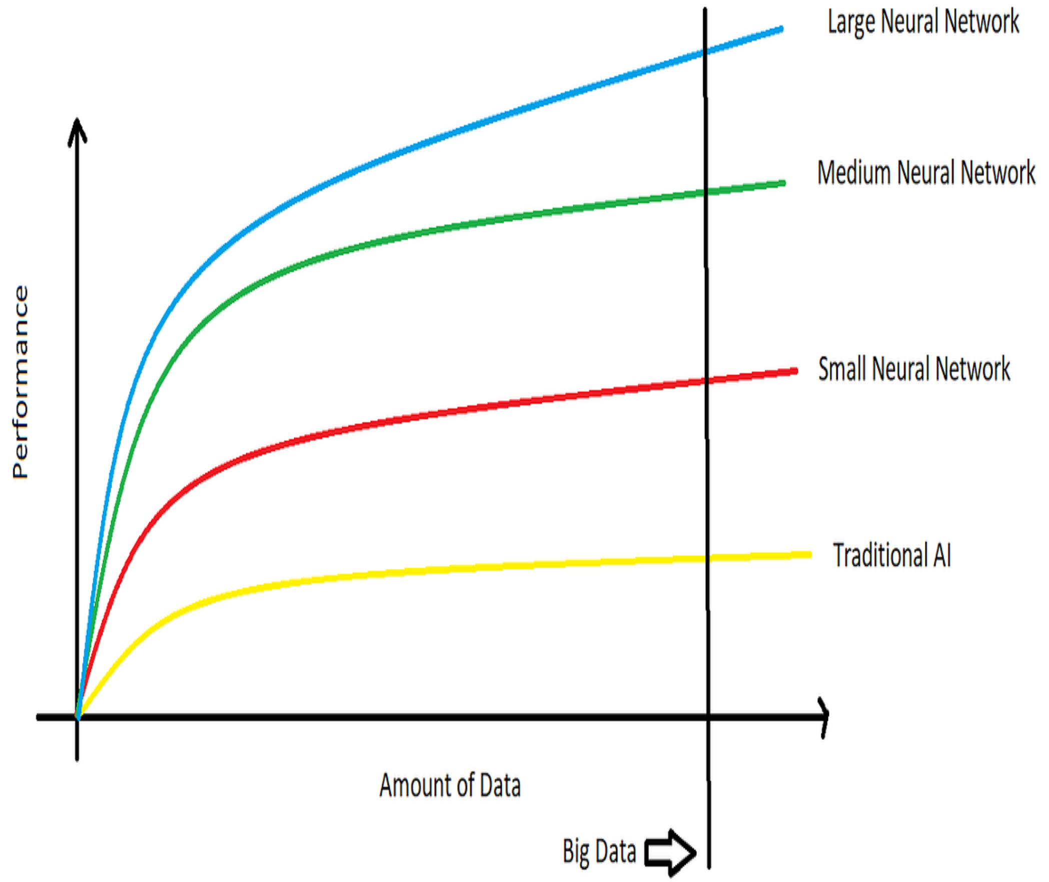
Source: DALL.E

Kawther Aboalam

Artificial Neural Networks (ANN)

Why Do We Need Artificial Neural Networks?

+ Superior Performance: Neural networks, especially larger ones, can achieve higher performance levels compared to traditional AI methods.
+ Scalability with Big Data: As the amount of data increases, the performance of neural networks continues to improve.
+ Handling Complexity: Larger neural networks can model more complex relationships within the data, which is reflected in their higher performance with increasing data amounts.
+ Neural networks are ideal for big data applications, where they can leverage large datasets to improve accuracy and efficiency.

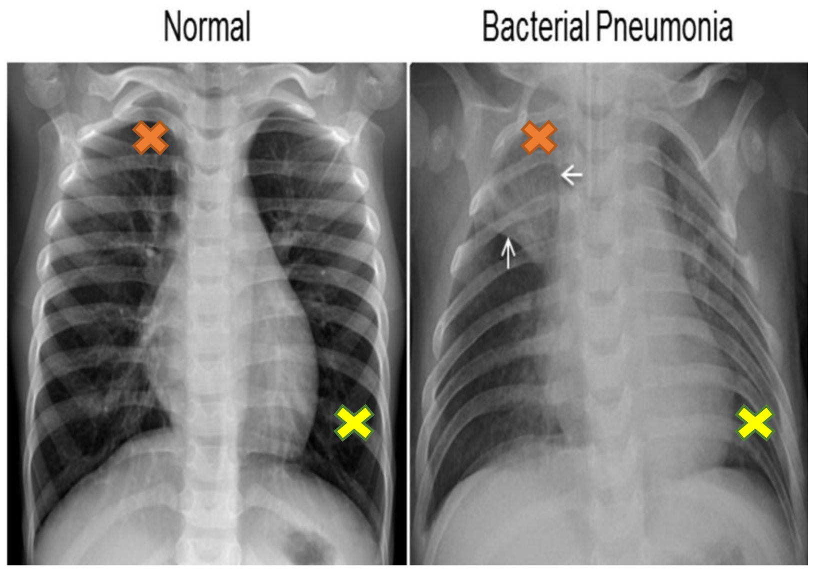


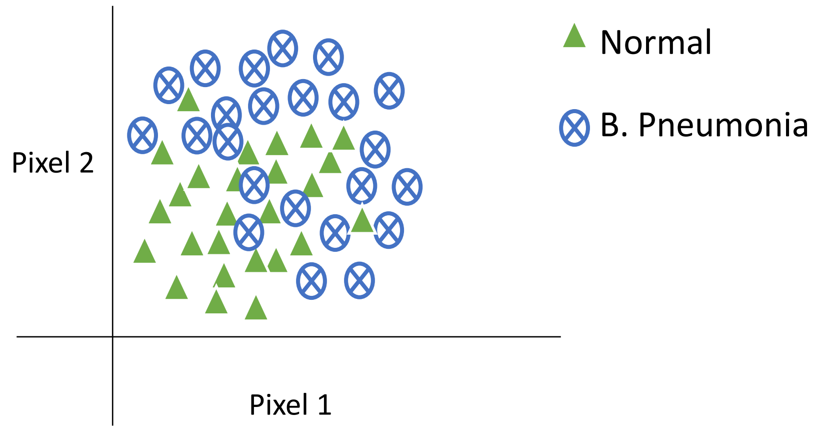
Image_source: https://medium.com/@didrichsen./demystifying-artificial-intelligence-a-basic-introduction-9f6386cb5439

Artificial Neural Networks (ANN)

Why Do We Need Artificial Neural Networks?

+ Example: Classification of Chest X-Ray images
  - Image Size is 1024x1024 pixels, which is a common resolution for medical X-ray images, providing sufficient detail for clinical analysis
  - Feature Space is 1.048.576 features
  - Simple logistic regression is not suitable for this task
+ Neural networks are essential for solving complex classification problems, especially when dealing with high-dimensional data and non-linear relationships.

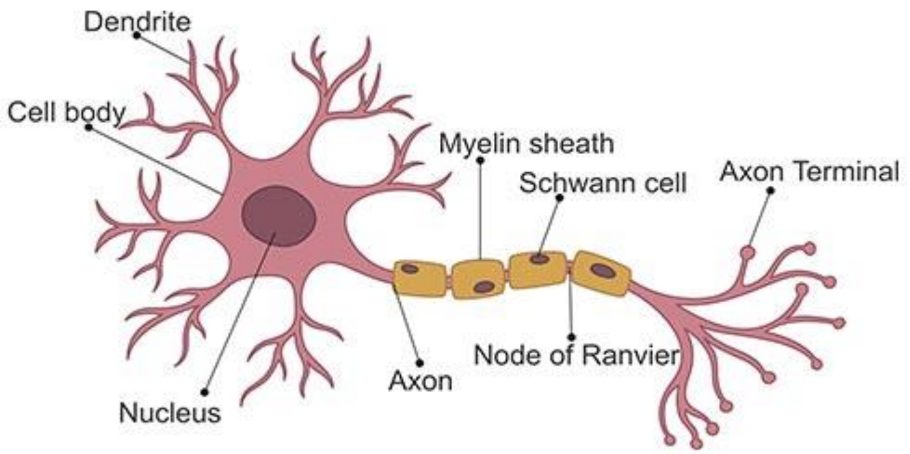




Artificial Neural Networks (ANN)

Biological Neuron And Artificial Neuron

- Biological Neuron
  - The above figure shows a biological neuron with signal flow from inputs at dendrites to outputs at axon terminals
  - The signal is a short electrical pulse called action potential or 'spike'
  - A neuron receives input signals through its dendrites, processes them, and sends the output down its axon.
  - Brain can process and learn from data from any source
- Artificial Neuron
  - It gets inputs via input wires
  - The sum of weighted inputs plus a bias is the net input
  - The net input is processed by the activation function
  - The activation function sends the output via the output wires



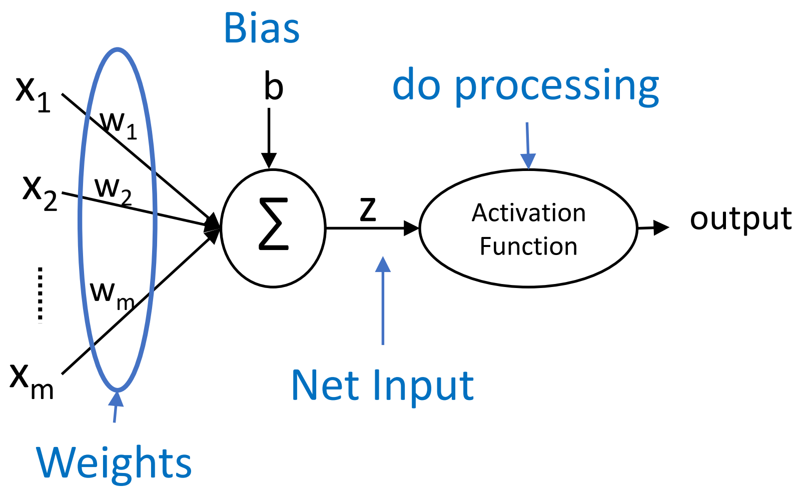


Image source: https://www.cusabio.com/Cell-Marker/Neuron-Cell.html

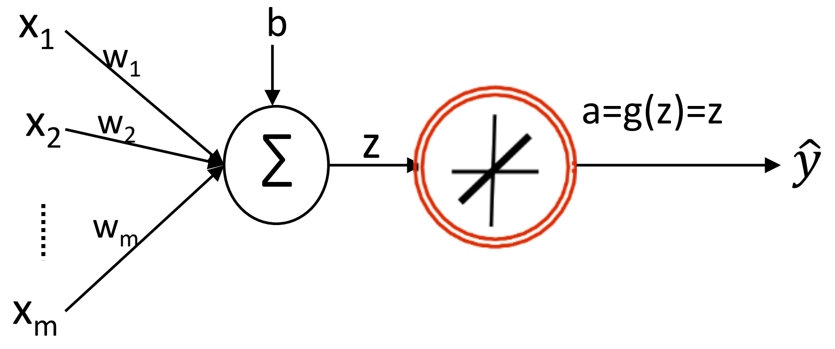
Artificial Neural Networks (ANN)

Forward Propagation For A Single Neuron

+ Forward propagation is the process of passing input data through the neural network to obtain an output (or prediction)
+ If the activation function is a linear function "pass-through", where $g(z) = z$,

  the single neuron represents a linear regression model.

  output = $\hat{y} = w_1x_1 + w_2x_2 + \cdots + w_mx_m + b = w^Tx + b$



Linear Regression Unit

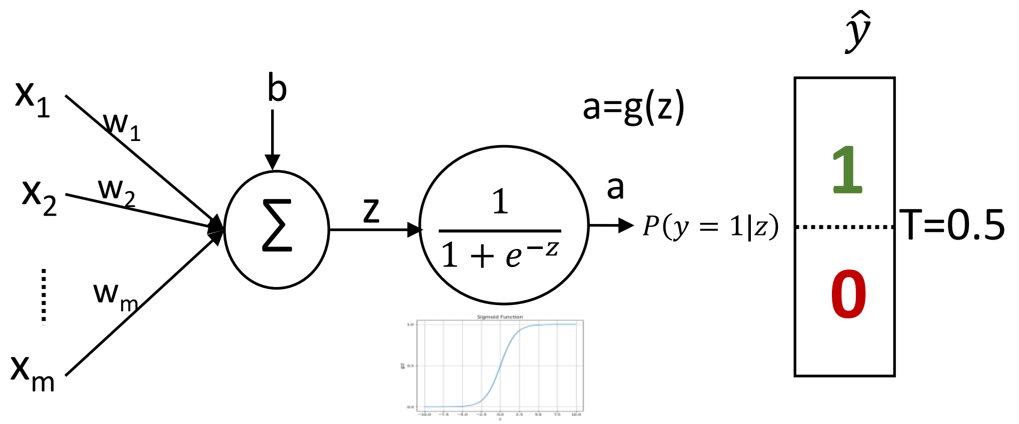
+ If the activation function is a sigmoid function , where

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

  the single neuron represents a logistic regression model.

  The output represents the probabilities and

$$\hat{y} = \begin{cases} 1 & if\ z \geq Threshold \\ 0 & if\ z < Threshold \end{cases}$$



Logistic Regression Unit

Artificial Neural Networks (ANN)

Backpropagation And Gradient Descent For A Single Neuron

- Backpropagation is the process of updating the weights and bias to minimize the error between the neuron output (a) and the actual target y. It is also called error backpropagation.
- Considering the batch gradient descent optimization algorithm:
  - For n training examples, the overall loss — also called the cost function — is the average of the individual losses over all training examples:

$$\mathrm{L}(w,b) = \frac{1}{n}\sum_{i=1}^{n} l(a^{(i)}, y^{(i)}) \longrightarrow$$

Superscript of (i) refers to the training example No. i

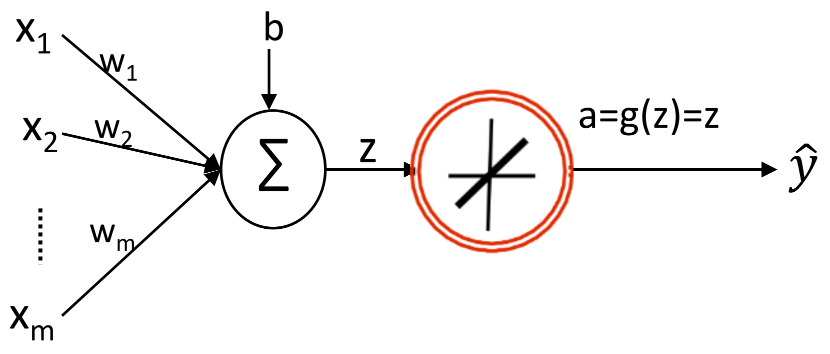
  - The gradient of the overall loss function with respect to the parameters (w , b)

$$\nabla_{w,b}\mathrm{L}(w,b) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{w,b} l(a^{(i)}, y^{(i)})$$

  - Overall loss in the case of linear regression Unit,

$$\begin{aligned}\mathrm{L}(w,b) &= \frac{1}{n}\sum_{i=1}^{n}(a^{(i)} - y^{(i)})^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{n}\sum_{i=1}^{n}(g(z^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}((w_1 {x_1}^{(i)} + \cdots + w_m {x_m}^{(i)} + b) - y^{(i)})^2\end{aligned}$$

It is the mean squared error



Linear Regression Unit

Artificial Neural Networks (ANN)

Backpropagation And Gradient Descent For A Single Neuron

+ Gradient is determined via partial derivatives of the loss function
+ The calculation of the partial derivatives is based on the chain rule

$$\frac{\partial \mathrm{L}(w,b)}{\partial w_1} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_1}$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial w_2} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_2}$$

$$\vdots \qquad \vdots \quad \vdots \qquad \vdots \qquad \vdots$$

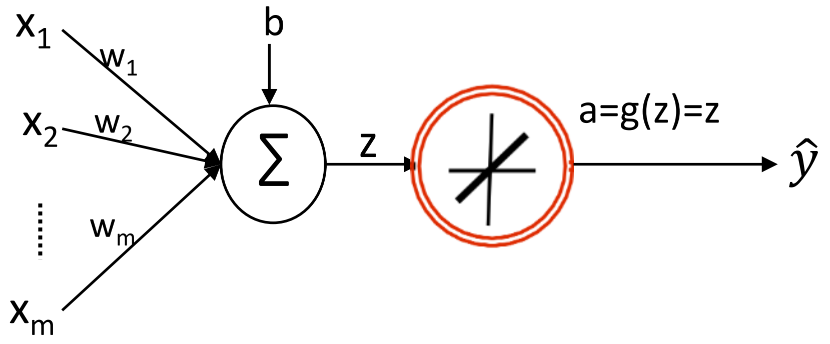
$$\frac{\partial \mathrm{L}(w,b)}{\partial w_m} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_m}$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial b} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial b}$$

$$\frac{\partial L(w,b)}{\partial g(z^{(i)})} = \frac{2}{n}\sum_{i=1}^{n}\left(g(z^{(i)}) - y^{(i)}\right) \quad , \quad \frac{\partial g(z^{(i)})}{\partial z^{(i)}} = 1, \qquad \frac{\partial z^{(i)}}{\partial w_1} = {x_1}^{(i)}, \; \frac{\partial z^{(i)}}{\partial b} = 1$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{n}\left(g(z^{(i)}) - y^{(i)}\right) {x_1}^{(i)}$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}\left(g(z^{(i)}) - y^{(i)}\right)$$



Linear Regression Unit

b_1

Artificial Neural Networks (ANN)

Backpropagation And Gradient Descent For A Single Neuron

- The steps involved in batch Gradient Descent optimization algorithms:
  - Initialize the weights and the bias with small random values to get started with the iteration process.
  - Keep on iterating for k = 0, 1, 2,……. using the update rule of the gradient descent:

$$w_1^{[k+1]} = w_1^{[k]} - \eta \cdot \frac{\partial \mathrm{L}(w,b)}{\partial w_1}$$

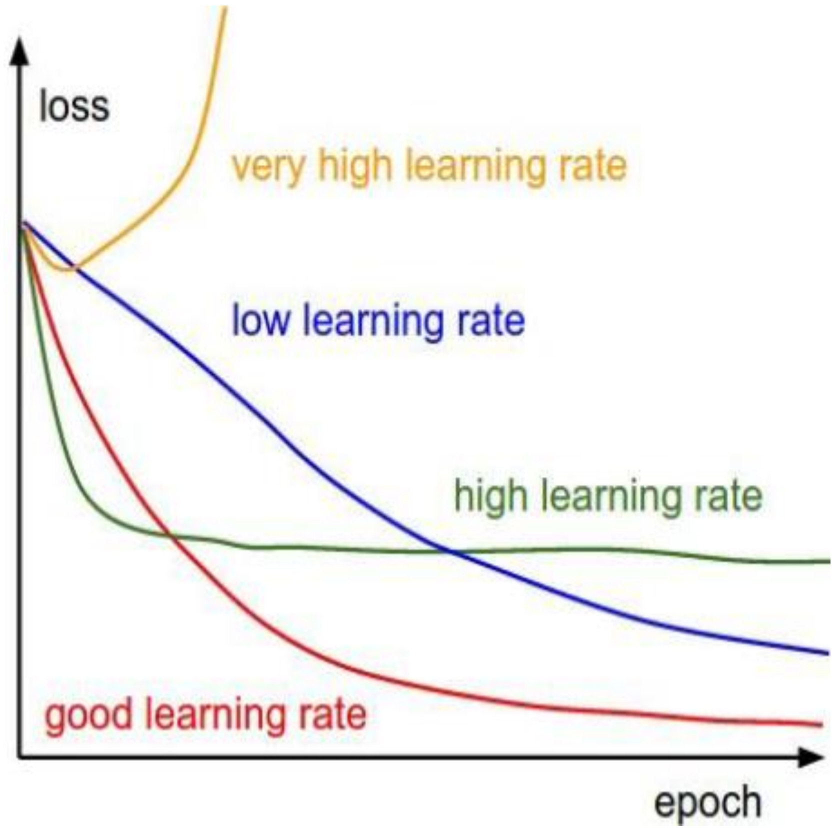
$$= w_1^{[k]} - \eta \cdot \frac{2}{n} \sum_{i=1}^{n} \left(g(z^{(i)}) - y^{(i)}\right) x_1{}^{(i)}$$

$$b^{[k+1]} = b^{[k]} - \eta \cdot \frac{\partial \mathrm{L}(w,b)}{\partial b}$$

$$= b^{[k]} - \eta \cdot \frac{2}{n} \sum_{i=1}^{n} \left(g(z^{(i)}) - y^{(i)}\right)$$

η is the learning rate or the learning step size

- The learning rate and the number of epochs can be considered as hyperparameters of this method



Effect of Learning Rate on the Loss Function

Image source: https://srdas.github.io/DLBook/GradientDescentTechniques.html

Artificial Neural Networks (ANN)

Backpropagation And Gradient Descent For A Single Neuron

- Overall loss for a logistic regression Unit,

  $$\mathrm{L}(w,b) = \frac{-1}{n}\sum_{i=1}^{n}\left(y^{(i)}\cdot\ln\left(a^{(i)}\right) + \left(1-y^{(i)}\right)\cdot\ln\left(1-a^{(i)}\right)\right)$$

  $$= \frac{-1}{n}\sum_{i=1}^{n}\left(y^{(i)}\cdot\ln\left(\mathrm{g}(z^{(i)})\right) + \left(1-y^{(i)}\right)\cdot\ln\left(1-\mathrm{g}(z^{(i)})\right)\right)$$

  It is the binary cross entropy

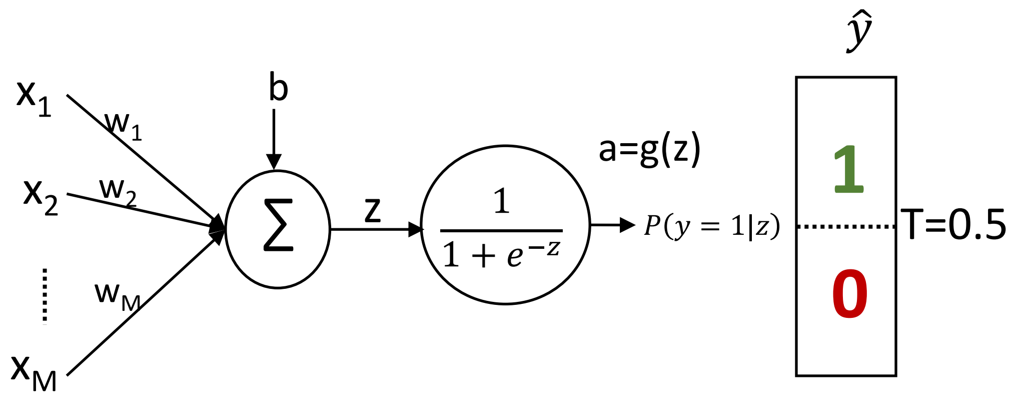
- The steps involved in batch Gradient Descent:
  - Initialize the weights and the bias with small random values to get started with the iteration process
  - Keep on iterating for k = 0, 1, 2,……. using the following update

$$w_1^{[k+1]} = w_1^{[k]} - \eta\cdot\frac{\partial\mathrm{L}(w,b)}{\partial w_1}$$

$$= w_1^{[k]} - \eta\cdot\frac{1}{n}\sum_{i=1}^{n}\left(g(z^{(i)}) - y^{(i)}\right)x_1^{(i)}$$

$$b^{[k+1]} = b^{[k]} - \eta\cdot\frac{\partial\mathrm{L}(w,b)}{\partial b}$$

$$= b^{[k]} - \eta\cdot\frac{1}{n}\sum_{i=1}^{n}\left(g(z^{(i)}) - y^{(i)}\right)$$



Logistic Regression Unit

Artificial Neural Networks (ANN)

Backpropagation And Gradient Descent For A Single Neuron

+ Gradient calculations for the logistic regression unit

$$\frac{\partial \mathrm{L}(w,b)}{\partial w_1} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_1}$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial w_2} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_2}$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

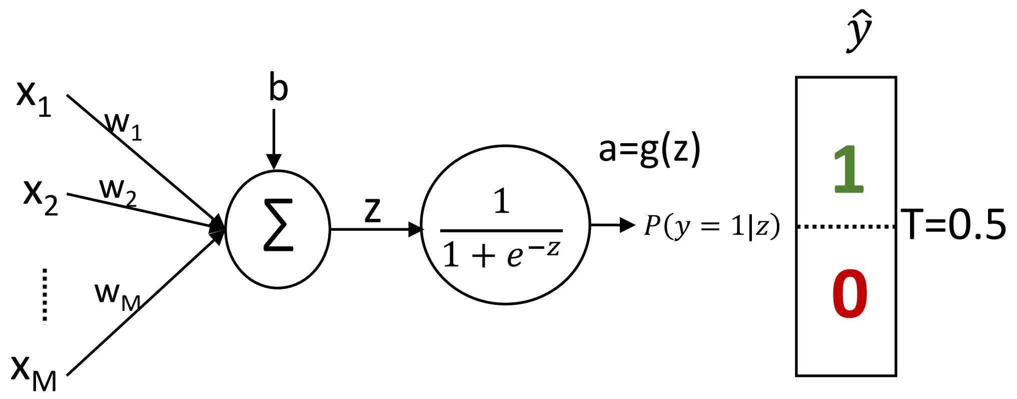
$$\frac{\partial \mathrm{L}(w,b)}{\partial w_m} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_m}$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial b} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial b}$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial g(z^{(i)})} = \frac{1}{n}\sum_{i=1}^{n} -y^{(i)} \cdot \frac{1}{g(z^{(i)})} + \left(1-y^{(i)}\right) \cdot \frac{1}{1-g(z^{(i)})}$$

$$\frac{\partial g(z^{(i)})}{\partial z^{(i)}} = \frac{1}{1+e^{-z^{(i)}}} = \frac{e^{-z^{(i)}}}{\left(1+e^{-z^{(i)}}\right)^2} = \frac{1}{1+e^{-z^{(i)}}} \cdot \left(\frac{e^{-z^{(i)}}}{1+e^{-z^{(i)}}}\right) =$$

$$\frac{1}{1+e^{-z^{(i)}}} \cdot \left(\frac{1+e^{-z^{(i)}}-1}{1+e^{-z^{(i)}}}\right) = \frac{1}{1+e^{-z^{(i)}}} \cdot \left(1 - \frac{1}{1+e^{-z^{(i)}}}\right) = g\left(z^{(i)}\right) \cdot \left(1 - g\left(z^{(i)}\right)\right)$$



Logistic Regression Unit

Artificial Neural Networks (ANN)

Backpropagation And Gradient Descent For A Single Neuron

$\frac{\partial z^{(i)}}{\partial w_1} = {x_1}^{(i)}$, and $\frac{\partial z^{(i)}}{\partial b}$ =1

+ By multiplying the partial derivatives:

$$\frac{\partial \mathrm{L}(w,b)}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n}\left(\left(-y^{(i)} \cdot \frac{1}{g(z^{(i)})} + y^{(i)} \cdot \frac{1}{1-g(z^{(i)})}\right) \cdot g\left(z^{(i)}\right) \cdot \left(1 - g\left(z^{(i)}\right)\right)\right) \cdot {x_1}^{(i)} = \frac{1}{n}\sum_{i=1}^{n}(-y^{(i)} \cdot \left(1 - g\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \cdot g\left(z^{(i)}\right)) \cdot {x_1}^{(i)}$$
$$= \frac{1}{n}\sum_{i=1}^{n}(-y^{(i)} + y^{(i)} \cdot g\left(z^{(i)}\right) + g\left(z^{(i)}\right) - y^{(i)} \cdot g\left(z^{(i)}\right)) \cdot {x_1}^{(i)} = \frac{1}{n}\sum_{i=1}^{n}(g\left(z^{(i)}\right) - y^{(i)}) \cdot {x_1}^{(i)}$$

$$\frac{\partial \mathrm{L}(w,b)}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n}(g\left(z^{(i)}\right) - y^{(i)})$$

Artificial Neural Networks (ANN)

Different Variants of Gradient Descent

+ Batch Gradient Descent (Gradient Descent)
  + It computes the gradient of the cost function with respect to the parameters (w and b) over the all training examples
  + The parameters are updated **once** for each epoch until convergence or until the maximum number of epochs is reached
  + **Algorithm**:

- Initialize parameters randomly
- Set the maximum number of epochs N or define a convergence criterion.
- For each epoch k=1 to N or until convergence:
  - Update the parameters once

$$w^{[k+1]} = w^{[k]} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_w l(a^{(i)}, y^{(i)})$$

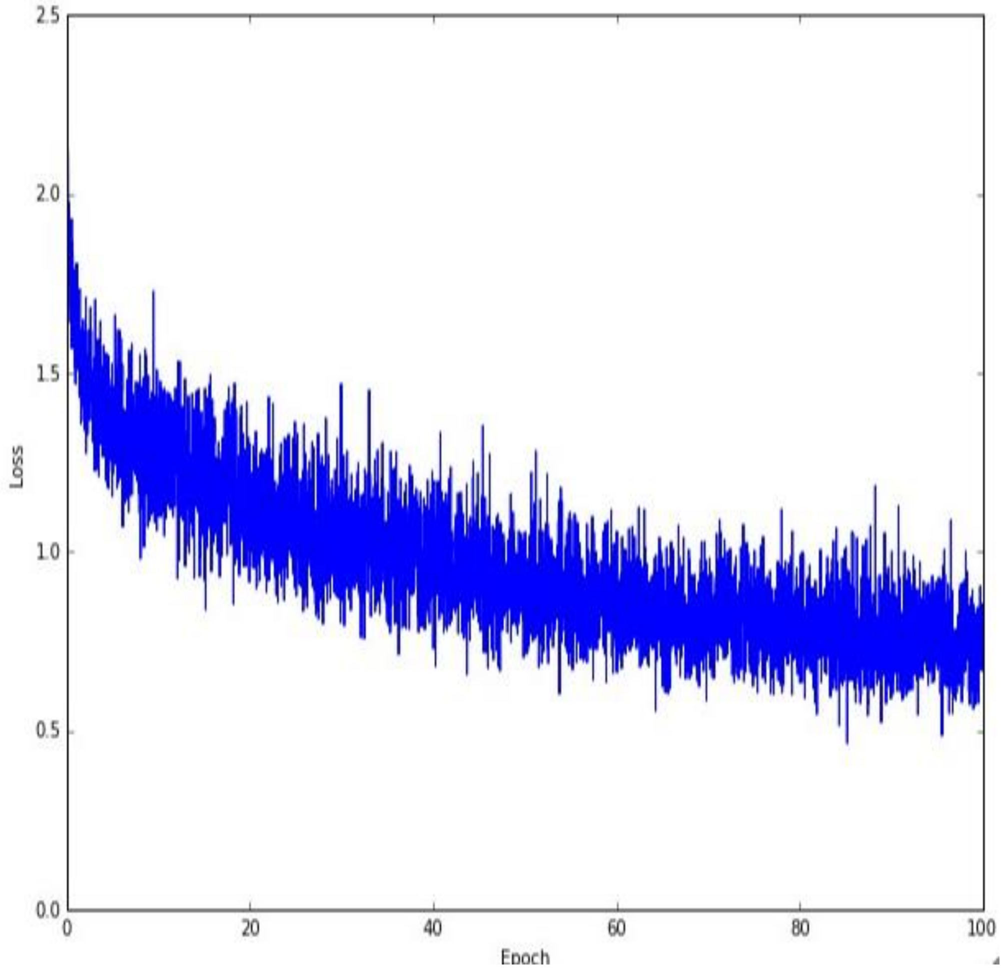
$$b^{[k+1]} = b^{[k]} - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_b l(a^{(i)}, y^{(i)})$$

Artificial Neural Networks (ANN)

Different Variants of Gradient Descent

+ Stochastic Gradient Descent (SGD)
+ It computes the gradient and updates the parameters for each training example individually
  + **Algorithm**:

```
- Initialize parameters randomly
- Set the maximum number of epochs $N$ or define a convergence criterion.
- For each epoch $k$=1 to N or until convergence:
    - Shuffle the training dataset
    - For each training example $i$:
        - Update the parameters once
          $w^{[k+1]} = w^{[k]} - \eta \cdot \nabla_w l(a^{(i)}, y^{(i)})$
          $b^{[k+1]} = b^{[k]} - \eta \cdot \nabla_b l(a^{(i)}, y^{(i)})$
```

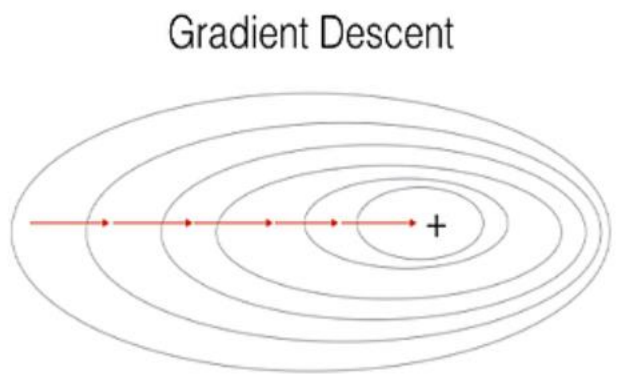


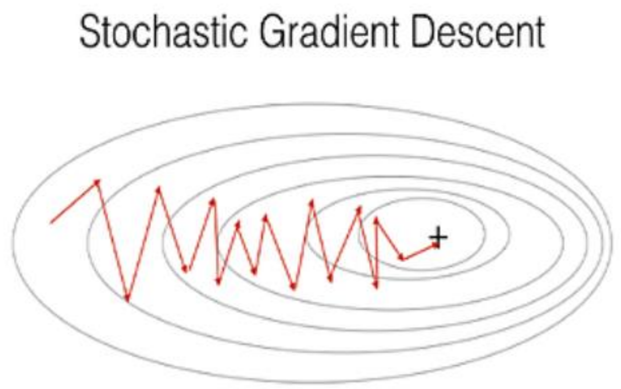
SGD fluctuation.

Artificial Neural Networks (ANN)

Different Variants of Gradient Descent

- Although both the batch gradient descent and the Stochastic Gradient Descent processed all the training examples within one epoch, stochastic gradient descent consumes more time than gradient descent with in one epoch. This is because stochastic gradient descent updated the parameters more frequently
- Stochastic gradient descent converges faster than GD in terms of number of examples processed (number of epochs), it uses more time to reach the same loss than GD because computing the gradient example by example is not as efficient
- Minibatch stochastic gradient descent is able to trade-off convergence speed and computation efficiency





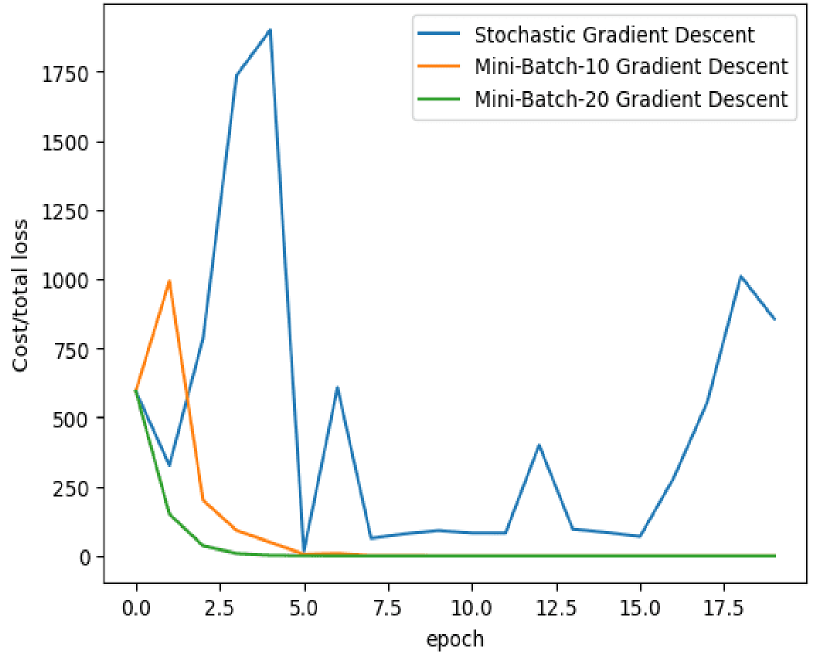
Artificial Neural Networks (ANN)

Different Variants of Gradient Descent

+ Mini-Batch Gradient Descent
+ It computes the gradient and updates the parameters for small batches of training examples
+ The gradient over a single training example is replaced by one over a small batch
  + **Algorithm**:
  - Initialize parameters randomly
  - Set the maximum number of epochs N or define a convergence criterion.
  - For each epoch k=1 to N or until convergence:
    - Shuffle the training dataset
    - Split the dataset into mini-batches
    - For each mini-batch B_t:
      - Update the parameters

$$w^{[k+1]} = w^{[k]} - \eta \cdot \frac{1}{\mathrm{n}B_\mathrm{t}} \sum_{i \in B_\mathrm{t}} \nabla_w l(a^{(i)}, y^{(i)})$$

$$b^{[k+1]} = b^{[k]} - \eta \cdot \frac{1}{\mathrm{n}B_\mathrm{t}} \sum_{i \in B_\mathrm{t}} \nabla_b l(a^{(i)}, y^{(i)})$$

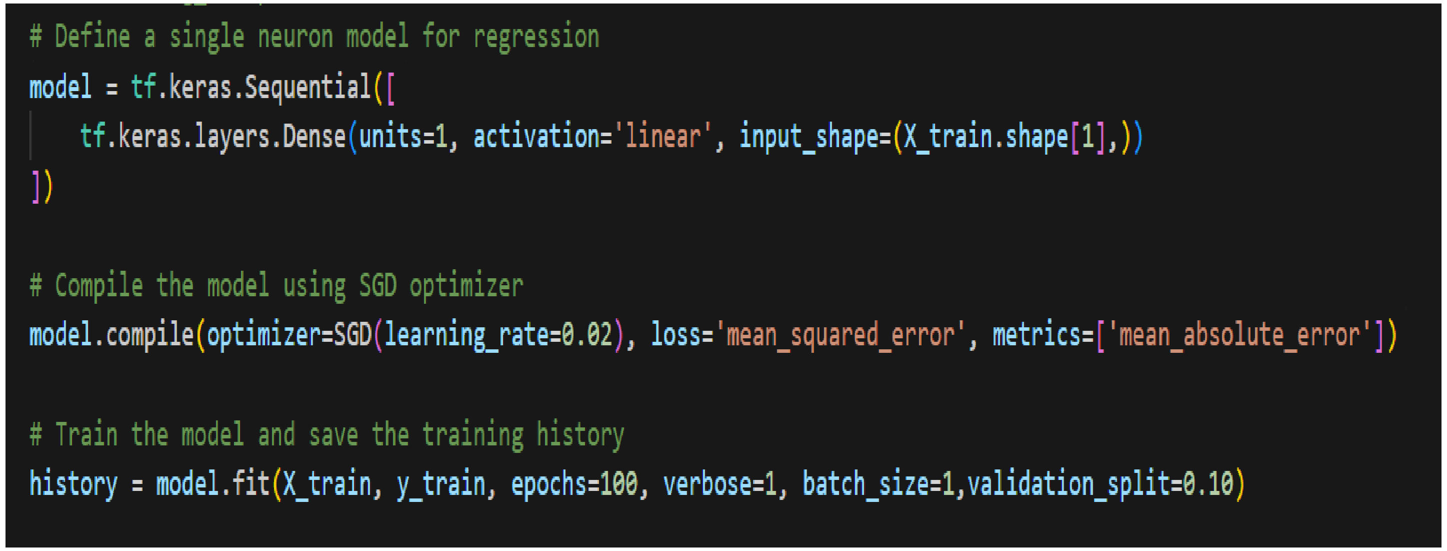


https://d2l.ai/chapter_optimization/minibatch-sgd.html
https://machinelearningmastery.com/mini-batch-gradient-descent-and-dataloader-in-pytorch/

Artificial Neural Networks (ANN)

TensorFlow Implementation For A Single Neuron

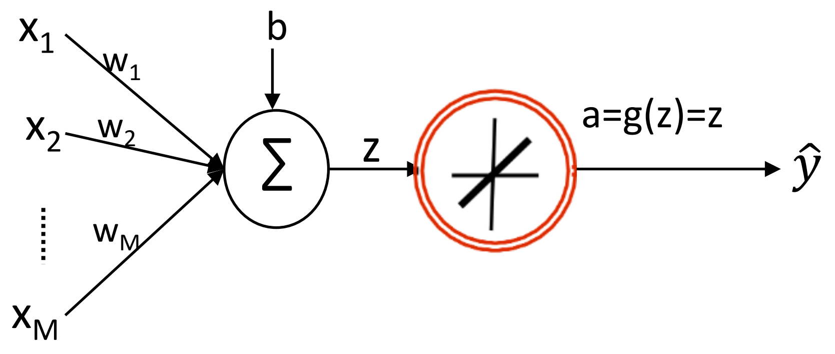
For a regression task:



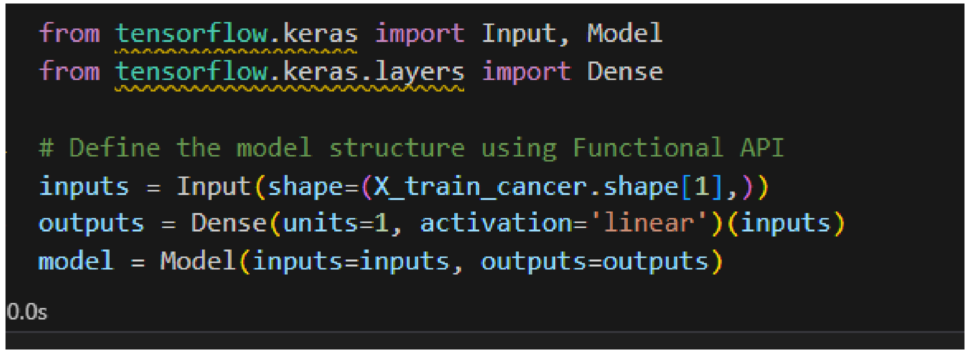
```
# Define a single neuron model for regression
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, activation='linear', input_shape=(X_train.shape[1],))
])

# Compile the model using SGD optimizer
model.compile(optimizer=SGD(learning_rate=0.02), loss='mean_squared_error', metrics=['mean_absolute_error'])

# Train the model and save the training history
history = model.fit(X_train, y_train, epochs=100, verbose=1, batch_size=1,validation_split=0.10)
```



Linear Regression Unit



```
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Dense

# Define the model structure using Functional API
inputs = Input(shape=(X_train_cancer.shape[1],))
outputs = Dense(units=1, activation='linear')(inputs)
model = Model(inputs=inputs, outputs=outputs)
0.0s
```

Artificial Neural Networks (ANN)

TensorFlow Implementation For A Single Neuron

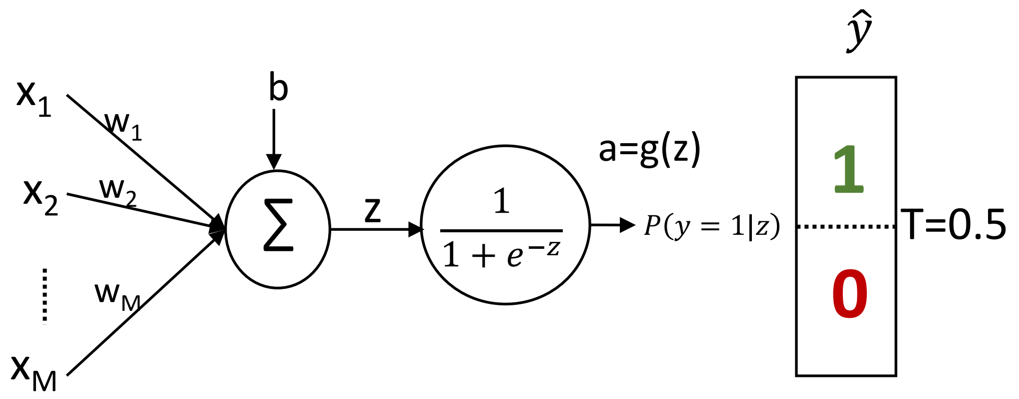
For a binary classification task:

```python
# Define a single neuron model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=(X_train.shape[1],))
])

# Compile the model
model.compile(optimizer=SGD(learning_rate=0.01), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model and save the training history
history = model.fit(X_train, y_train, epochs=100, verbose=0,batch_size=10, validation_split=0.1)
```

```python
# Define the model structure using Functional API
inputs = Input(shape=(X_train_cancer.shape[1],))
outputs = Dense(units=1, activation='sigmoid')(inputs)
model = Model(inputs=inputs, outputs=outputs)
```



Logistic Regression Unit

Artificial Neural Networks (ANN)

Multilayer Perceptron MLP

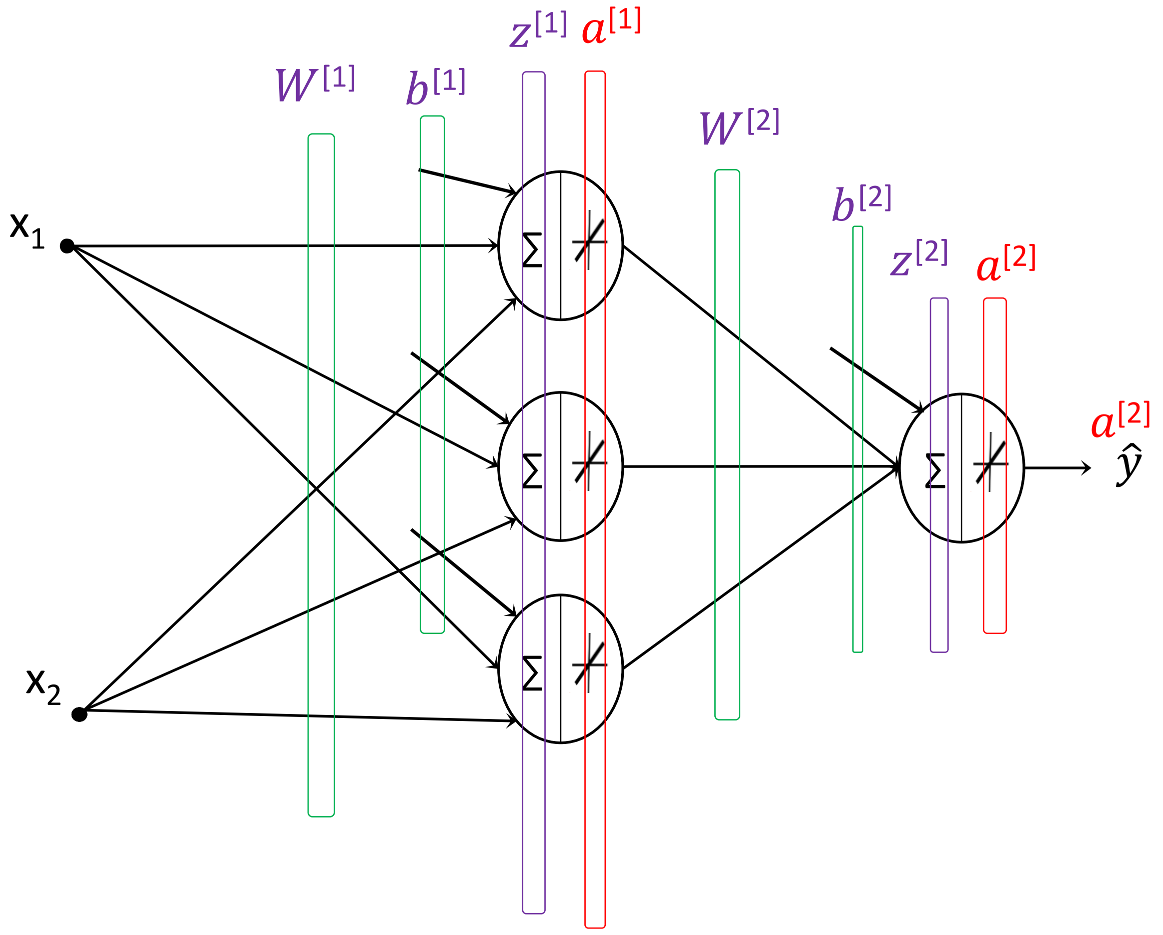
+ A multilayer perceptron (MLP) is a feedforward artificial neural network consisting of multiple layers of neurons connected in a directed graph, where each layer is fully connected to the next until the output is generated.
+ Information flows in one direction, from the input layer to the output layer, without cycles or loops.
+ Structure of an MLP:

  1. Input Layer:
     - The first layer of the MLP that receives the inputs
     - The number of neurons in this layer corresponds to the number of features in the input data
  2. Hidden Layers:
     - One or more intermediate layers between the input and output layers.
  3. Output Layer:
     - The final layer that produces the network's output.
     - The number of neurons depends on the task (one neuron for binary classification or regression tasks and multiple neurons for multiclass classification).
     - Activation functions are varied depending on the task (sigmoid for binary classification, SoftMax for multiclass classification, linear for regression).

Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

- The illustrated ANN has two inputs (x_1 and x_2), one hidden layer containing three neurons, and one output neuron.
- Each layer feeds into the layer next to it, until the output is generated. They are fully connected layers.
- For simplicity, the activation functions for all neurons in this network are linear activation functions.
- W$^{[1]}$ and b$^{[1]}$ are the weights and biases for the hidden layer.
- W$^{[2]}$ and b$^{[2]}$ are the weights and biases for the output layer.
- Z$^{[1]}$ is the sum of weighted inputs plus the biases for the hidden layer.
- Z$^{[2]}$ is the sum of weighted inputs plus the biases for the output layer.
- $a^{[1]}$ is the output of activation of the hidden layer neurons.
- $a^{[2]}$ is the output of activation of the output layer neuron and in this case it is the output the neural network $\hat{y}$.



Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

Forward Propagation Path:

+ $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ is the input vector
+ Hidden Layer

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

  Since the activation functions are linear, the activation function output $a^{[1]}$ is just $z^{[1]}$

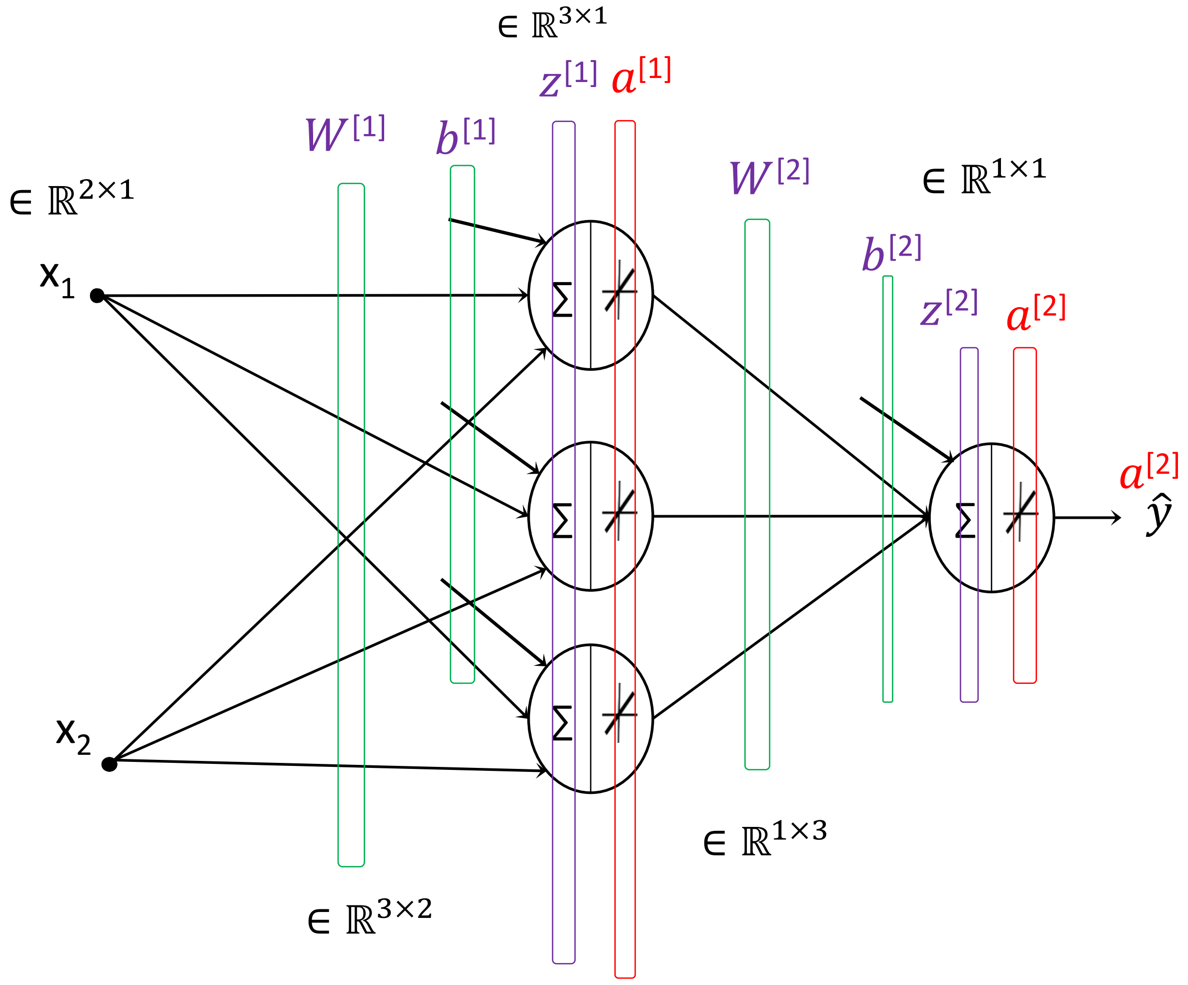
$$a^{[1]} = z^{[1]}$$

+ Output Layer

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = z^{[2]}$$

$$\hat{y} = a^{[2]}$$



$\in \mathbb{R}^{3\times1}$
$z^{[1]}$ $a^{[1]}$
$W^{[1]}$
$b^{[1]}$
$W^{[2]}$
$\in \mathbb{R}^{1\times1}$
$\in \mathbb{R}^{2\times1}$
x_1
$b^{[2]}$
$z^{[2]}$
$a^{[2]}$
Σ
$a^{[2]}$
$\hat{y}$
x_2
$\in \mathbb{R}^{1\times3}$
$\in \mathbb{R}^{3\times2}$

Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

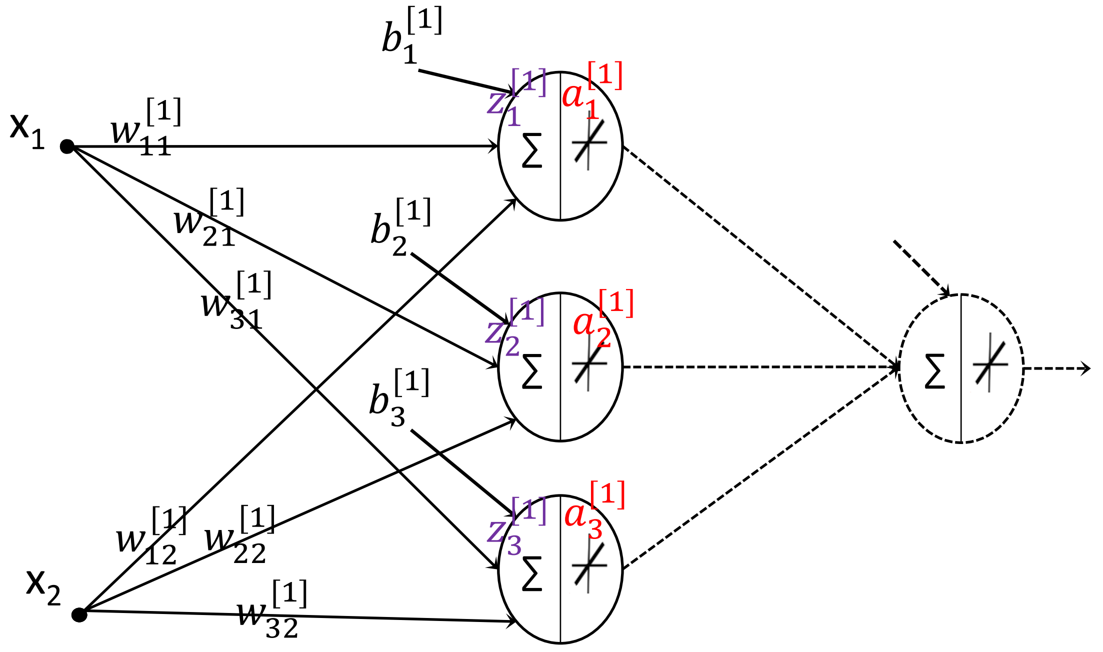
Forward Propagation Path in more details:

+ Hidden Layer

$$z^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$$

$\in \mathbb{R}^{3\times1}$ $\in \mathbb{R}^{3\times2}$ $\in \mathbb{R}^{2\times1}$ $\in \mathbb{R}^{3\times1}$

$$a^{[1]} = z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix}$$ (Linear activation)



Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

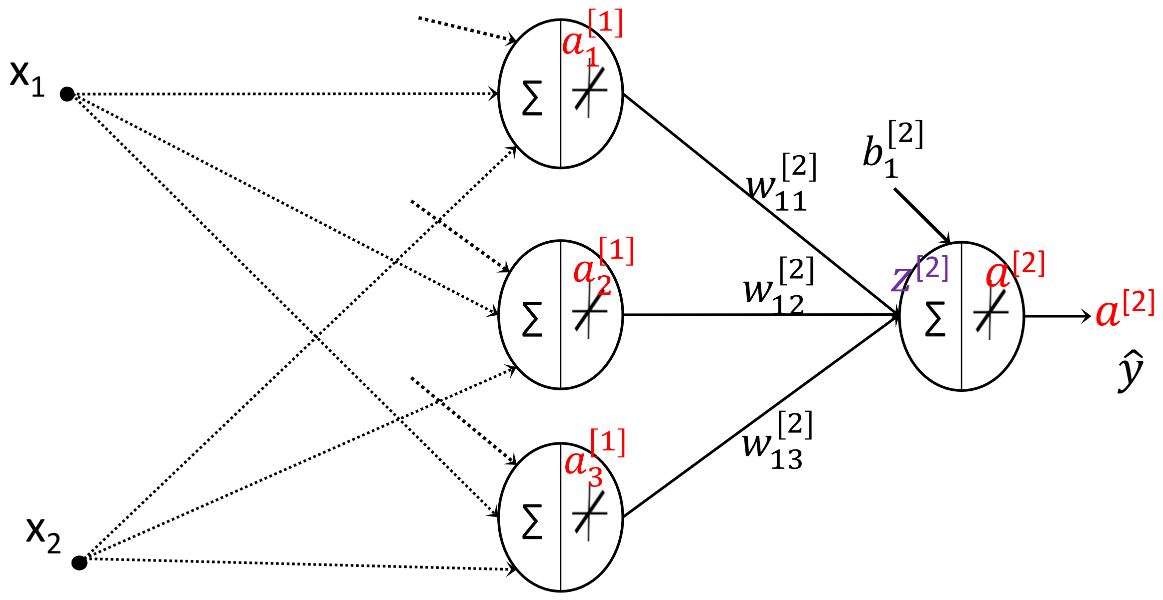
Forward Propagation Path in more details:

+ Output Layer

$$z^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} + b^{[2]}$$

$\in \mathbb{R}^{1\times1}$ $\in \mathbb{R}^{1\times3}$ $\in \mathbb{R}^{3\times1}$ $\in \mathbb{R}^{1\times1}$

$a^{[2]} = z^{[2]}$ (Linear activation)



Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

Forward Propagation Path in more details:

+ Hidden Layer

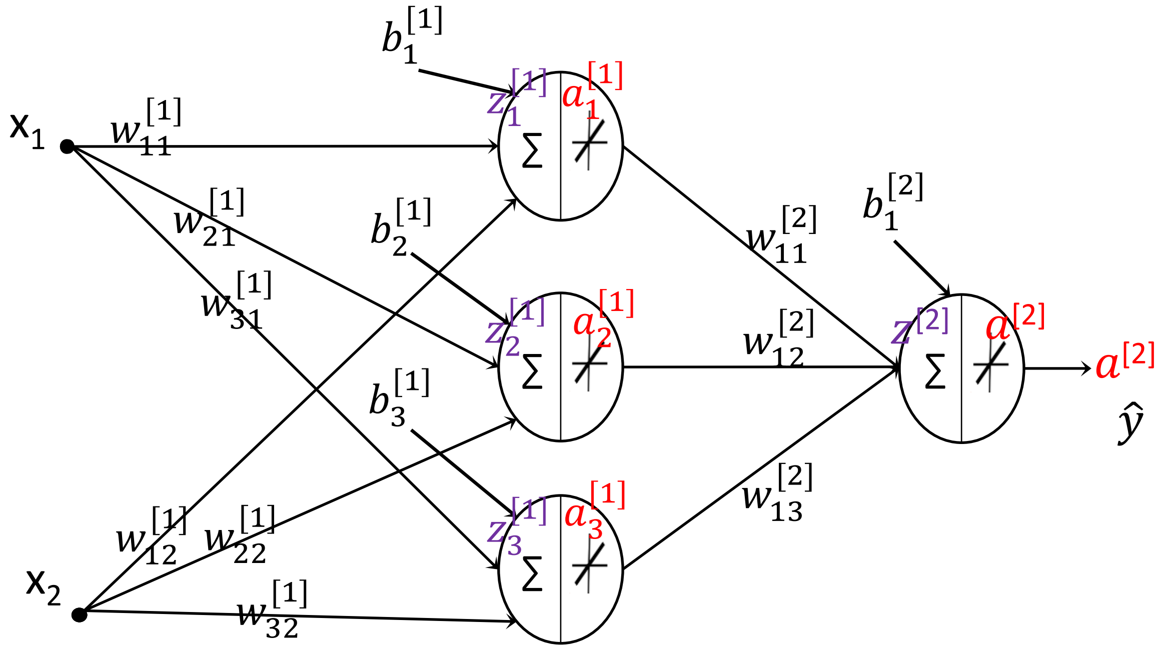
$$w_{11}^{[2]}$$

$$w_{12}^{[2]}$$ (Linear activation)

+ Output Layer

$$z^{[2]} = \begin{bmatrix} w_{11}^{[2]} & w_{12}^{[2]} & w_{13}^{[2]} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} + b^{[2]}$$

$a^{[2]} = z^{[2]}$ (Linear activation)



Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

Example: Given Input x and initial values for weights and biases, compute the output of the shown ANN.

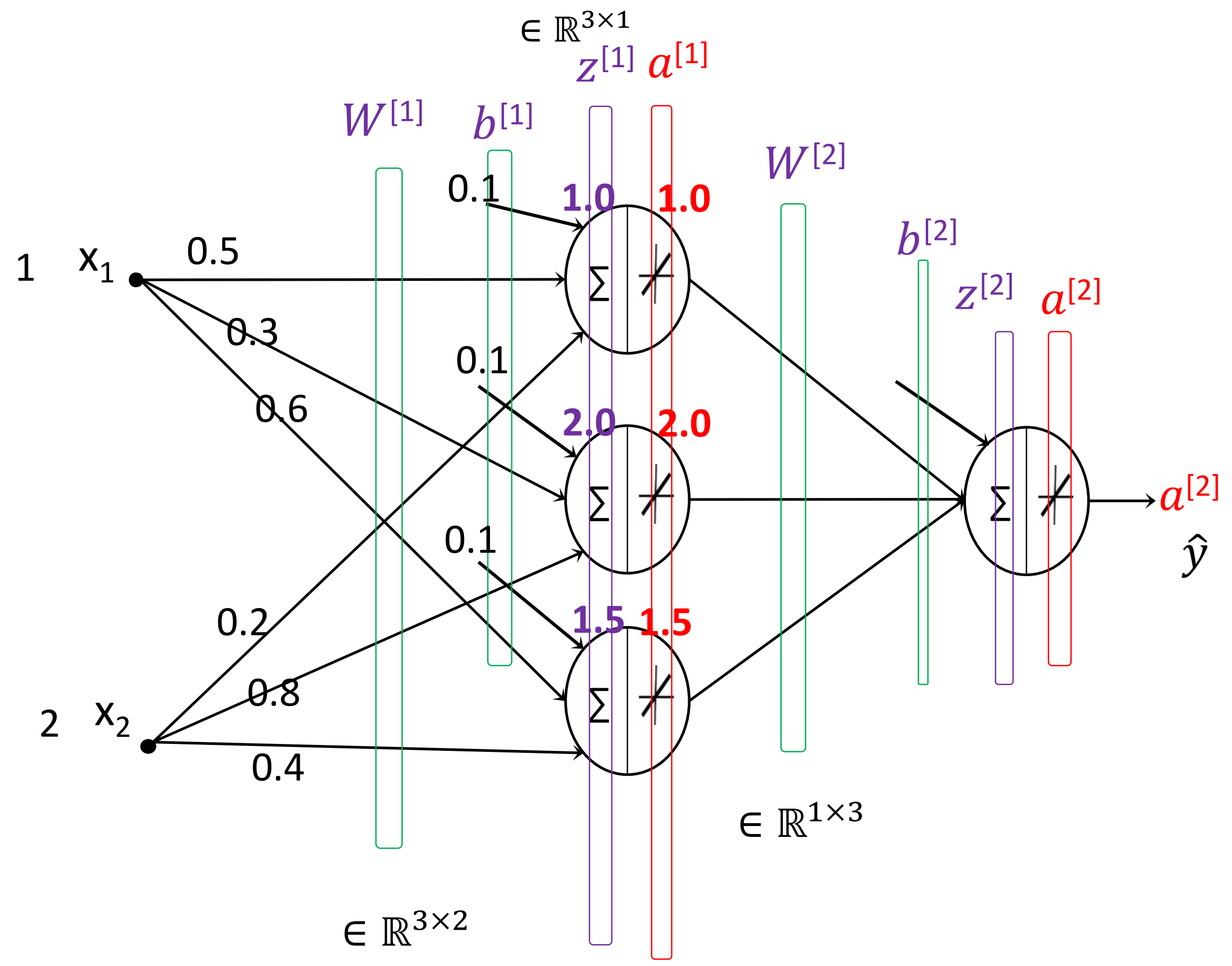
+ $x = \begin{bmatrix}1\\2\end{bmatrix}$ $\quad W^{[1]} = \begin{bmatrix}0.5 & 0.2\\0.3 & 0.8\\0.6 & 0.4\end{bmatrix}, \quad b^{[1]} = \begin{bmatrix}0.1\\0.1\\0.1\end{bmatrix}$

$$W^{[2]} = \begin{bmatrix}0.4 & 0.3 & 0.2\end{bmatrix}, \quad b^{[2]} = 0.1$$

+ The output is computed by the forward propagation:

$$z^{[1]} = W^{[1]}x + b^{[1]} = \begin{bmatrix}0.5 & 0.2\\0.3 & 0.8\\0.6 & 0.4\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} + \begin{bmatrix}0.1\\0.1\\0.1\end{bmatrix} = \begin{bmatrix}1.0\\2.0\\1.5\end{bmatrix}$$

$$a^{[1]} = z^{[1]} = \begin{bmatrix}1.0\\2.0\\1.5\end{bmatrix}$$



Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

Example: Given Input x and initial values for weights and biases, compute the output of the shown ANN.

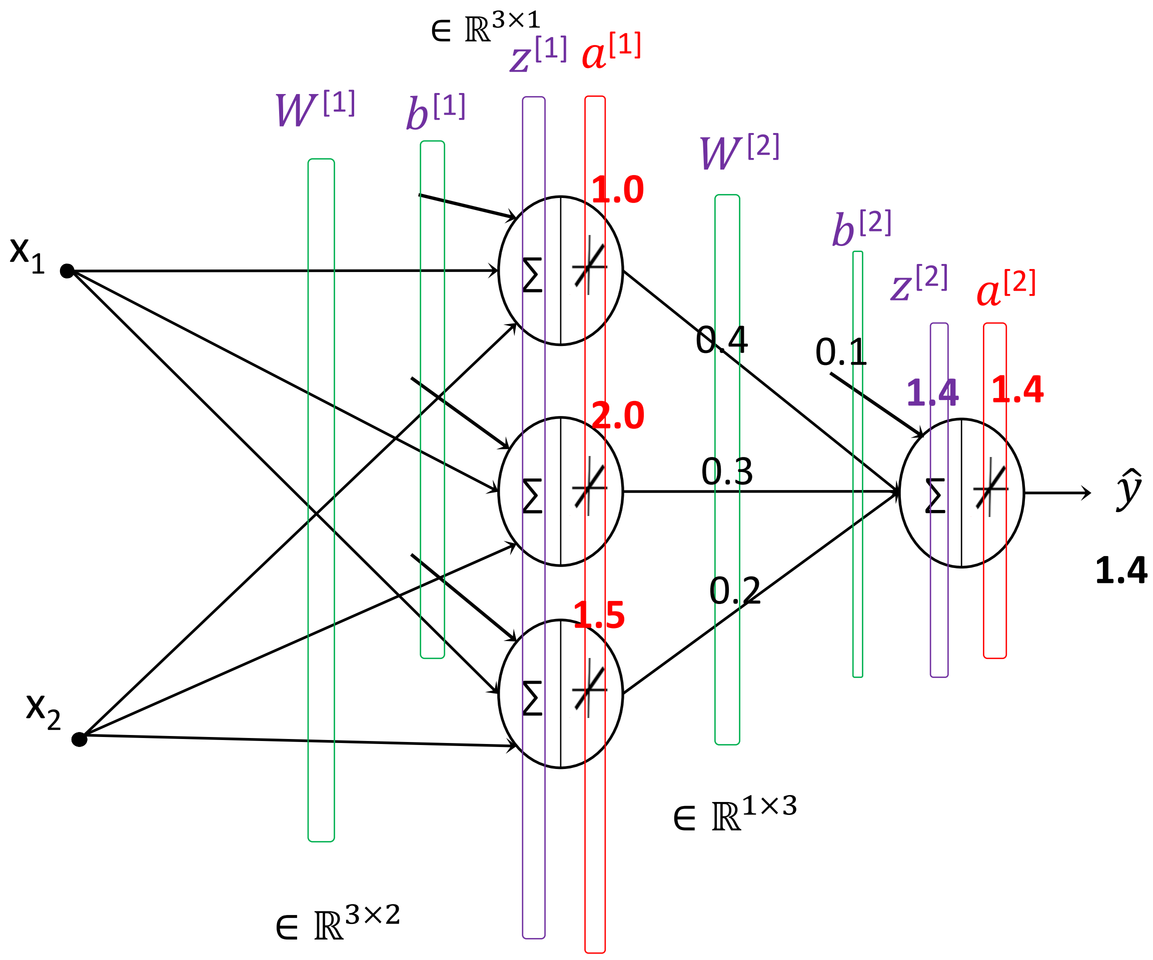
+ $x = \begin{bmatrix}1\\2\end{bmatrix}$ $\quad W^{[1]} = \begin{bmatrix}0.5 & 0.2\\0.3 & 0.8\\0.6 & 0.4\end{bmatrix}, \quad b^{[1]} = \begin{bmatrix}0.1\\0.1\\0.1\end{bmatrix}$

$$W^{[2]} = \begin{bmatrix}0.4 & 0.3 & 0.2\end{bmatrix}, \quad b^{[2]} = 0.1$$

+ The output is computed by the forward propagation:

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} = \begin{bmatrix}0.4 & 0.3 & 0.2\end{bmatrix}\begin{bmatrix}1.0\\2.0\\1.5\end{bmatrix} + 0.1 = 1.4$$

$$a^{[2]} = z^{[2]} = 1.4$$



Artificial Neural Networks (ANN)

Multilayer Perceptron – One Hidden Layer Network

Backropagation Path for parameters update:

+ To compute the gradients using the chain rule. For simplicity, the following loss function is applied:

$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$$

+ Where, $\hat{y}$ is the output of the ANN and y is the true target value

$$\mathcal{L} = \frac{1}{2}\left(a^{[2]} - y\right)^2 \qquad \text{as } \hat{y} = a^{[2]}$$

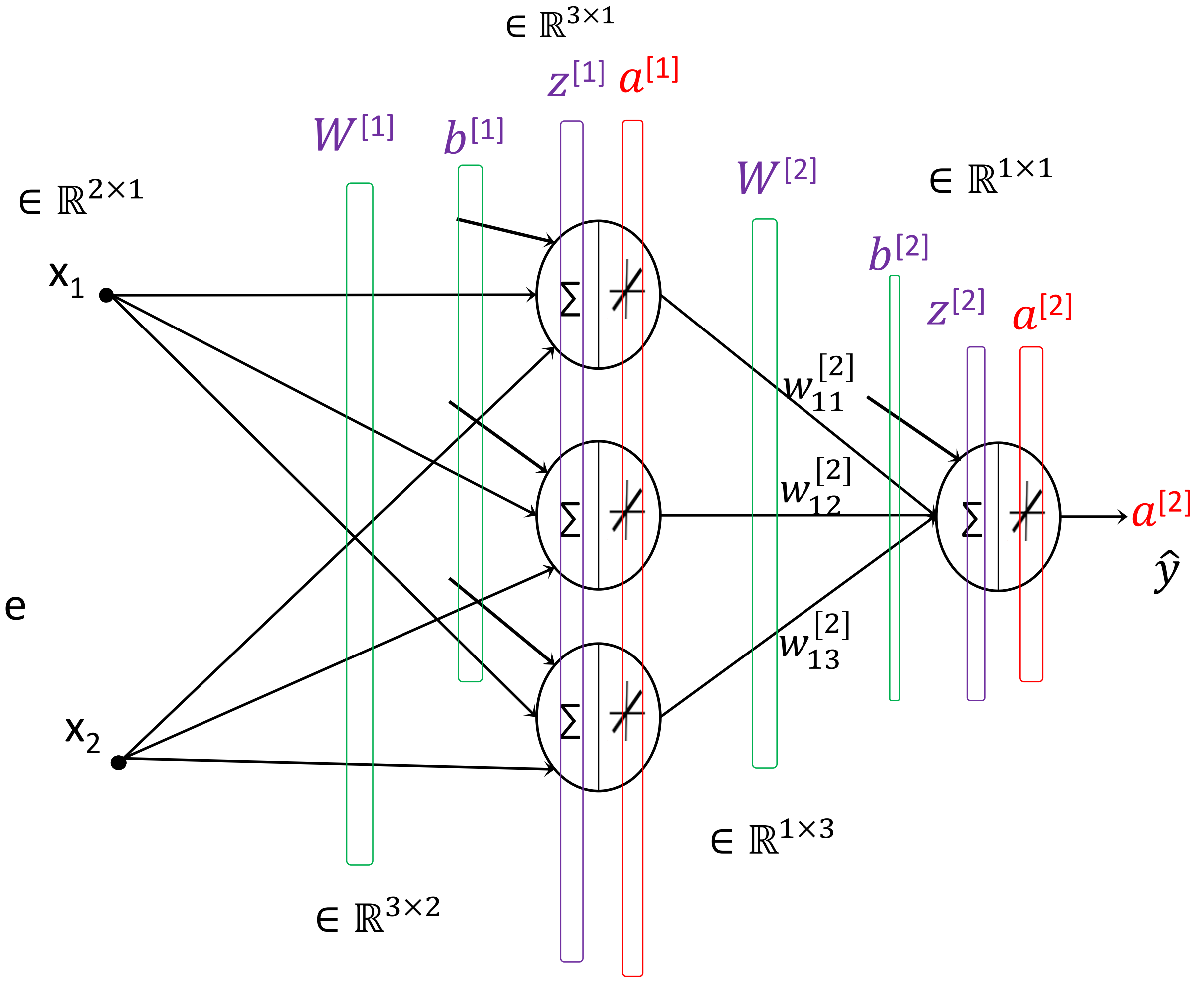
+ Gradients:

1. For output Layer parameters $W^{[2]}$ and $b^{[2]}$

$$\frac{\partial \mathcal{L}}{\partial a^{[2]}} = a^{[2]} - y$$

Since $a^{[2]} = z^{[2]}$ (Linear activation) $\frac{\partial a^{[2]}}{\partial z^{[2]}} = 1$

$\frac{\partial z^{[2]}}{\partial W^{[2]}} = a^{[1]T}$ and $\frac{\partial z^{[2]}}{\partial b^{[2]}} = 1$ where, $z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$



Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

Gradients for $W^{[2]}$:

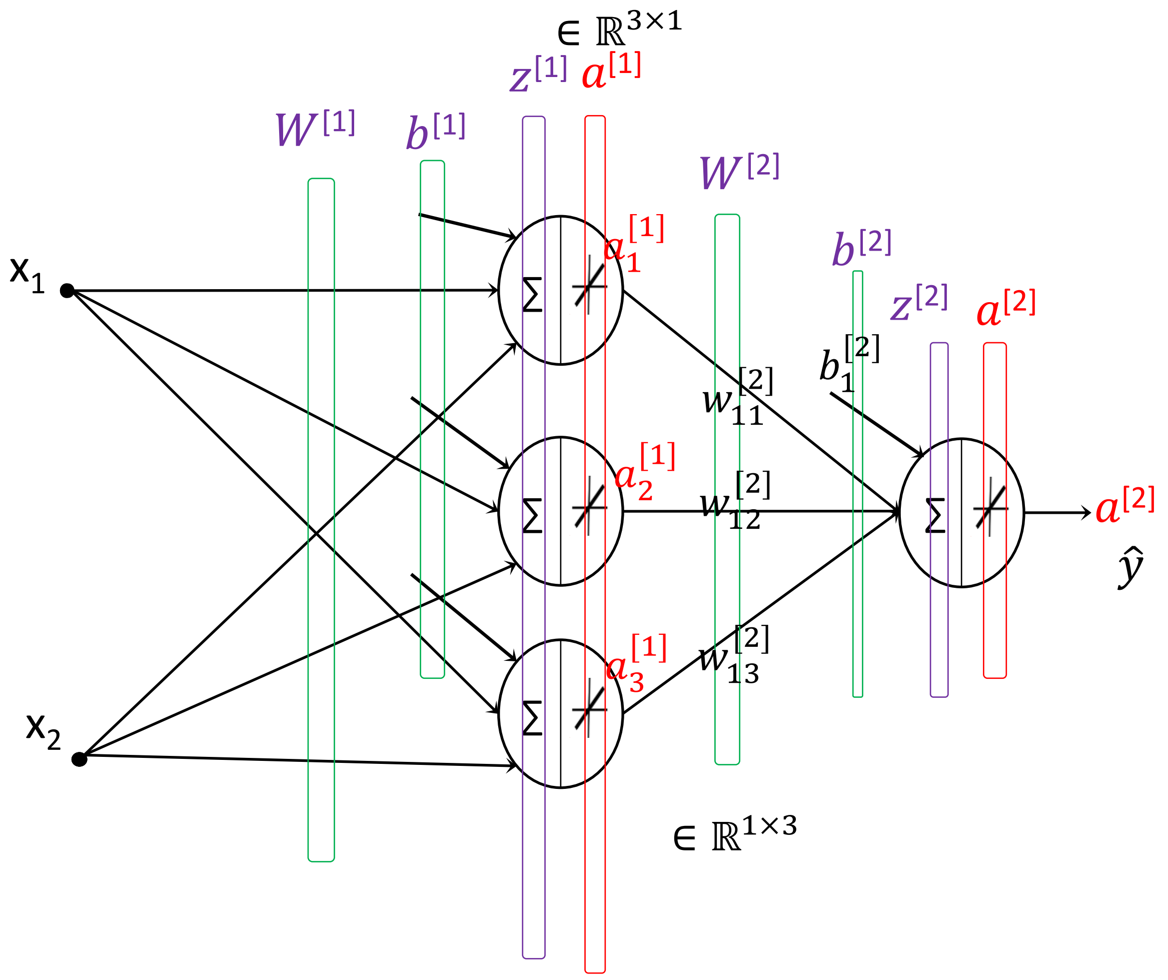
$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T}$$

$\in \mathbb{R}^{1\times 3}$ $\in \mathbb{R}^{1\times 3}$

Gradients for $b^{[2]}$:

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial b^{[2]}} = (a^{[2]} - y)$$

$\in \mathbb{R}^{1\times 1}$ $\in \mathbb{R}^{1\times 1}$



$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = z^{[2]}$$
$$\hat{y} = a^{[2]}$$

Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

2. For hidden layer parameters $W^{[1]}$ and $b^{[1]}$

Gradients for $W^{[1]}$:

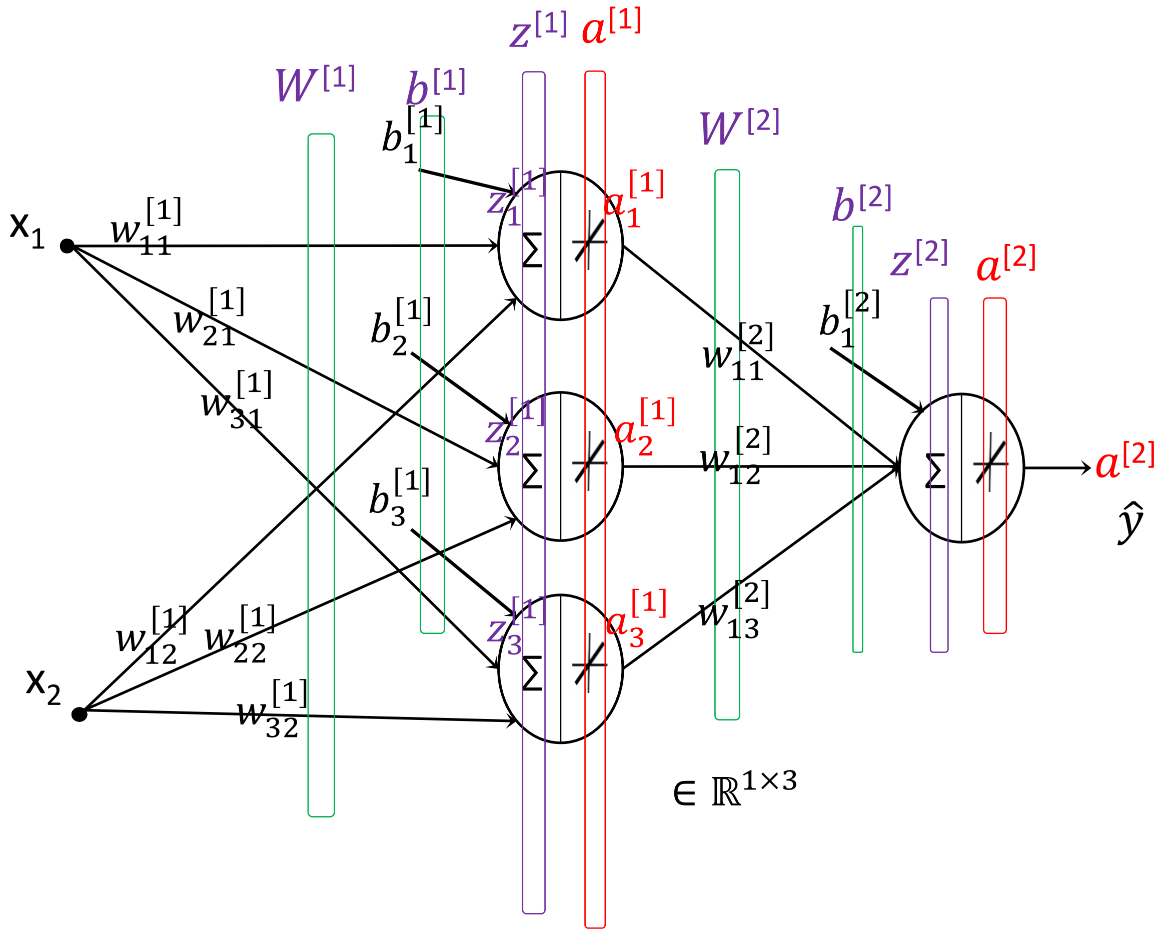
$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}} = \left(a^{[2]} - y\right) W^{[2]\mathrm{T}} x^{T}$$

$\in \mathbb{R}^{3\times 2}$ $\in \mathbb{R}^{3\times 2}$

Gradients for $b^{[1]}$:

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial b^{[1]}} = \left(a^{[2]} - y\right) W^{[2]\mathrm{T}}$$

$\in \mathbb{R}^{3\times 1}$ $\in \mathbb{R}^{3\times 1}$



$\in \mathbb{R}^{1\times 3}$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = z^{[1]}$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = z^{[2]}$$
$$\hat{y} = a^{[2]}$$

Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

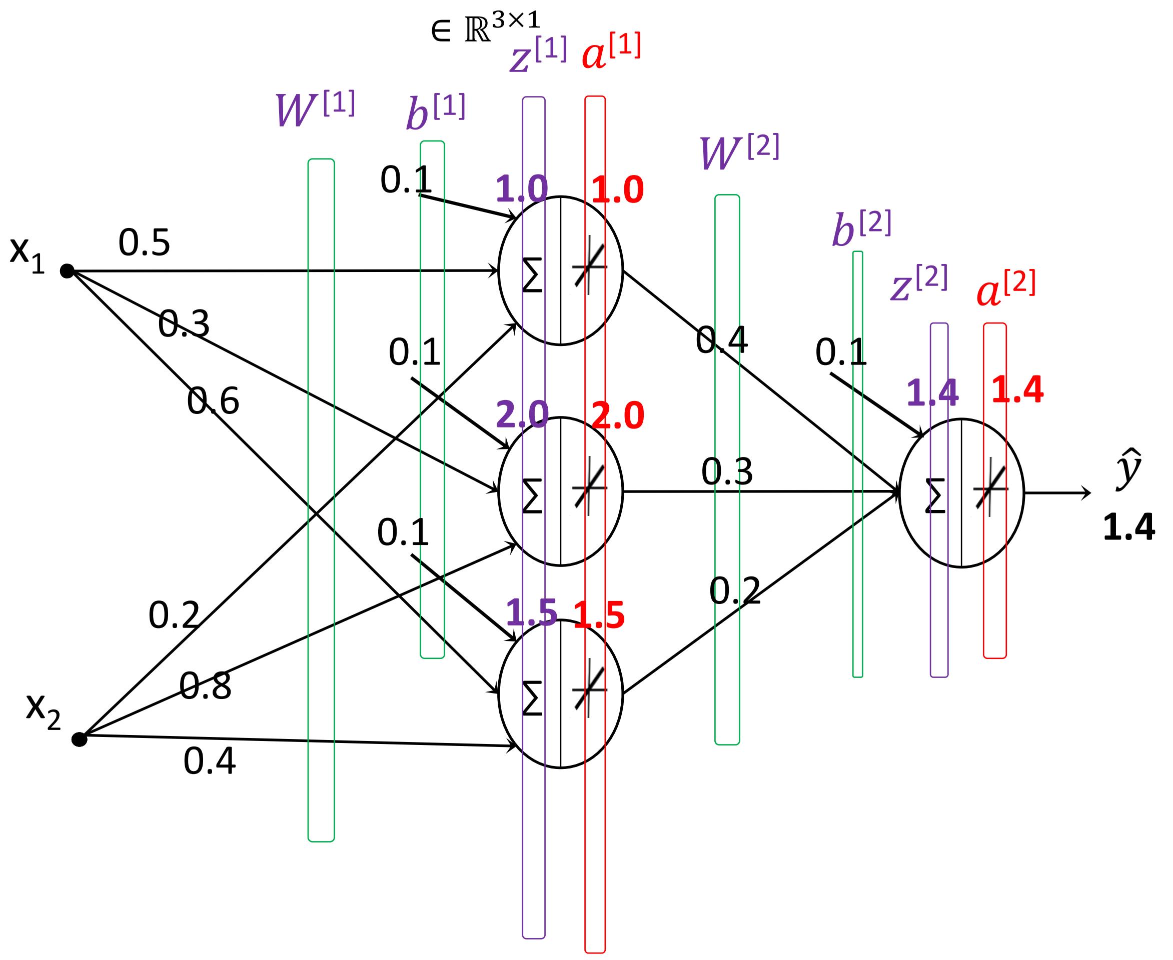
Example:

Refer to the example on slide 24. Update the parameters by one step, given that the true target value is $y = 1$ and the learning rate is $\eta = 0.1$

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad W^{[1]} = \begin{bmatrix} 0.5 & 0.2 \\ 0.3 & 0.8 \\ 0.6 & 0.4 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} 0.4 & 0.3 & 0.2 \end{bmatrix}, \quad b^{[2]} = 0.1$$

$$a^{[1]} = z^{[1]} = \begin{bmatrix} 1.0 \\ 2.0 \\ 1.5 \end{bmatrix} \quad \text{and} \quad a^{[2]} = z^{[2]} = 1.4$$



Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

Gradients for $W^{[2]}$ and $b^{[2]}$:

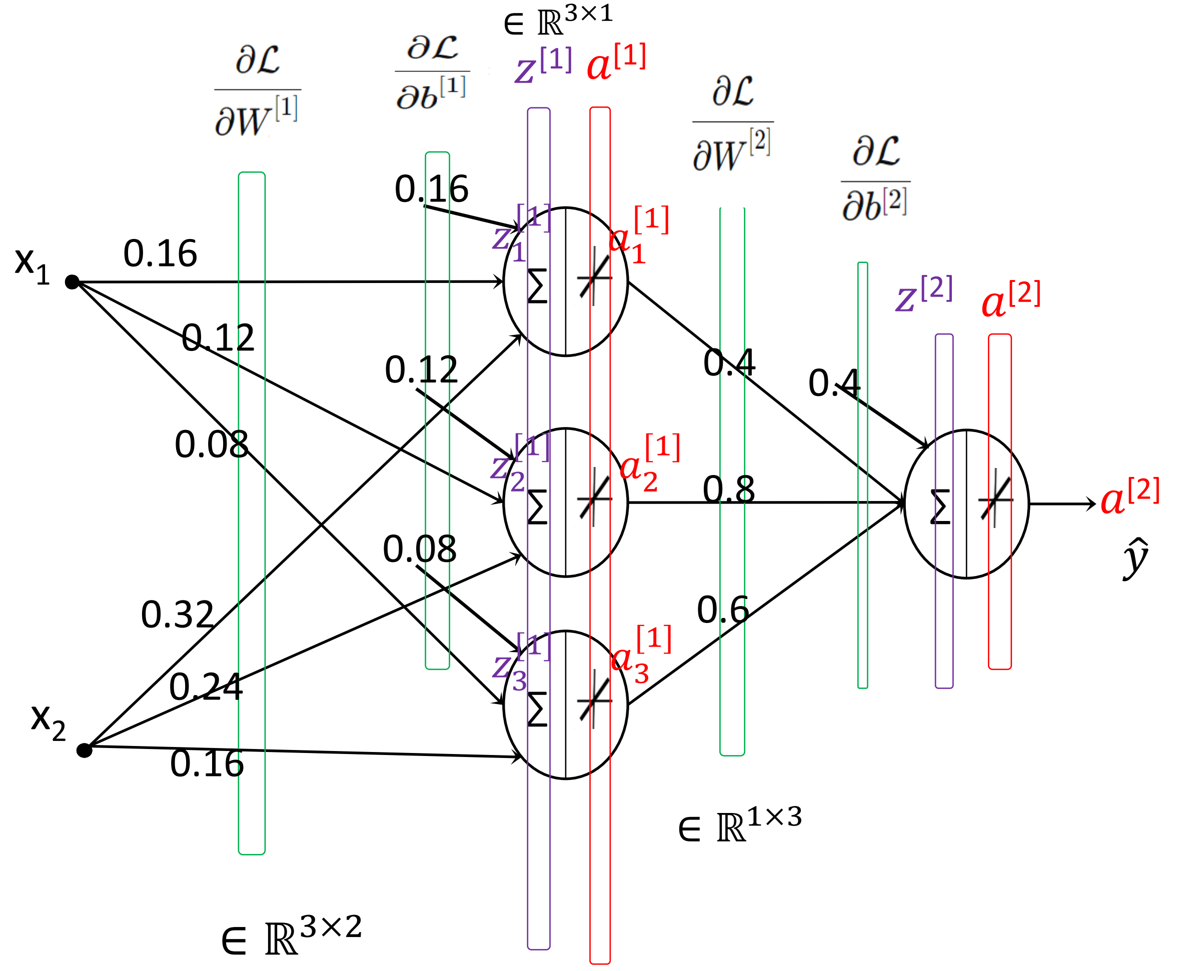
$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = (a^{[2]} - y)a^{[1]T} = (1.4 - 1)\begin{bmatrix}1.0\\2.0\\1.5\end{bmatrix}^T = \begin{bmatrix}0.4 & 0.8 & 0.6\end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = a^{[2]} - y = 0.4$$

Gradients for $W^{[1]}$ and $b^{[1]}$:

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = (a^{[2]} - y)W^{[2]\mathrm{T}}\, x^T = 0.4.\begin{bmatrix}0.4 & 0.3 & 0.2\end{bmatrix}^T \begin{bmatrix}1\\2\end{bmatrix}^T$$

$$\frac{\partial \mathcal{L}}{\partial W^{[1]}} = 0.4 \cdot \begin{bmatrix}0.4\\0.3\\0.2\end{bmatrix}\begin{bmatrix}1 & 2\end{bmatrix} = \begin{bmatrix}0.16 & 0.32\\0.12 & 0.24\\0.08 & 0.16\end{bmatrix}$$



$\frac{\partial \mathcal{L}}{\partial W^{[1]}}$
$\frac{\partial \mathcal{L}}{\partial b^{[1]}}$
$\in \mathbb{R}^{3\times1}$
$z^{[1]}$
$a^{[1]}$
$\frac{\partial \mathcal{L}}{\partial W^{[2]}}$
$\frac{\partial \mathcal{L}}{\partial b^{[2]}}$
$z^{[2]}$
$a^{[2]}$
x_1
x_2
0.16
0.12
0.08
0.32
0.24
0.16
0.16
0.12
0.08
$z_1^{[1]}$
$a_1^{[1]}$
$z_2^{[1]}$
$a_2^{[1]}$
$z_3^{[1]}$
$a_3^{[1]}$
0.4
0.8
0.6
0.4
$a^{[2]}$
$\hat{y}$
$\in \mathbb{R}^{1\times3}$
$\in \mathbb{R}^{3\times2}$

Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = (a^{[2]} - y)W^{[2]\mathrm{T}} = 0.4.\,[0.4 \quad 0.3 \quad 0.2]^T$$

$$\frac{\partial \mathcal{L}}{\partial b^{[1]}} = 0.4 \cdot \begin{bmatrix} 0.4 \\ 0.3 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.16 \\ 0.12 \\ 0.08 \end{bmatrix}$$

Update step for $W^{[2]}$ and $b^{[2]}$:

$$W^{[2](k+1)} = W^{[2](k)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial W^{[2]}}$$

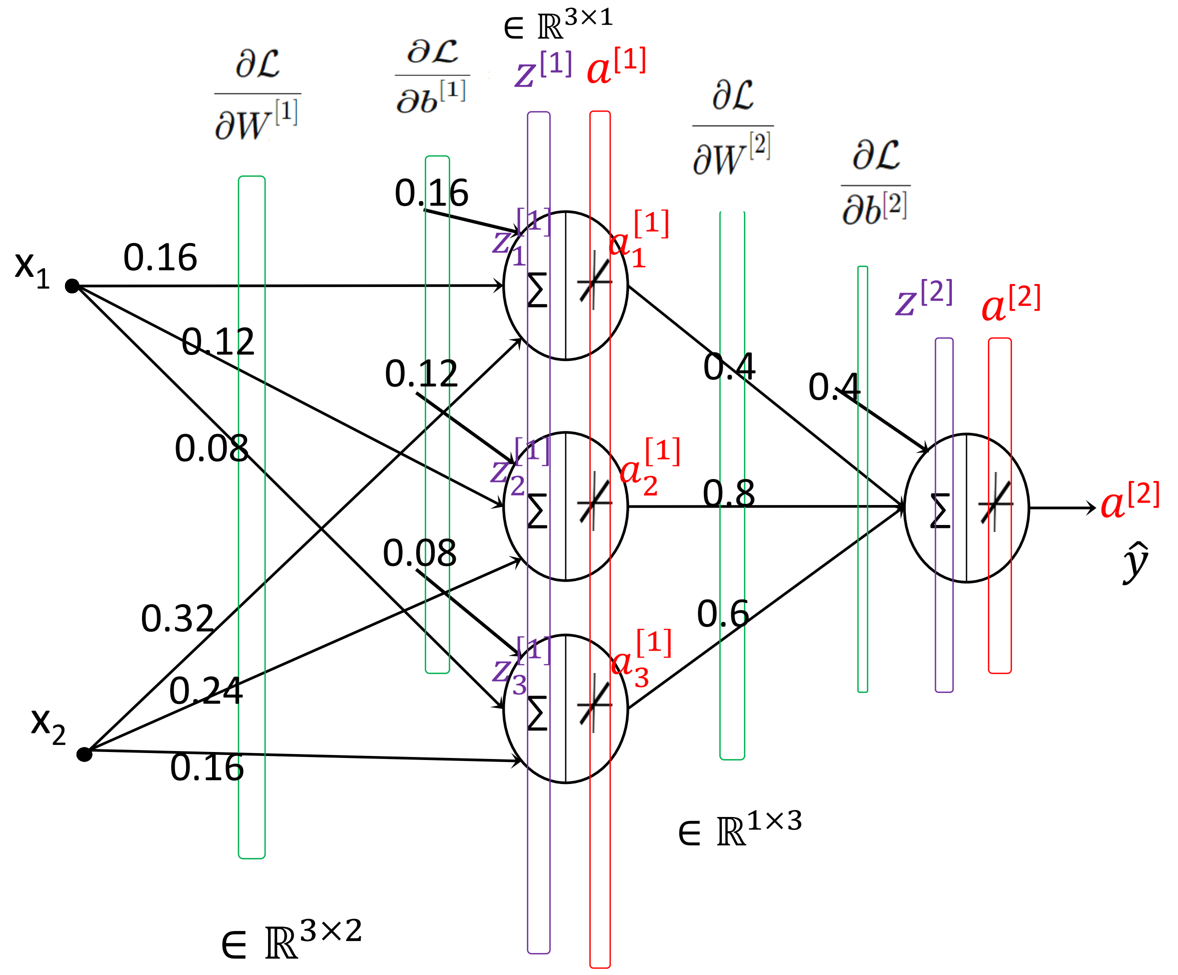
$W^{[2]}$ after one update step:

$$W^{[2]} = [0.4 \quad 0.3 \quad 0.2] - 0.1.\,[0.4 \quad 0.8 \quad 0.6]$$
$$= [0.36 \quad 0.22 \quad 0.14]$$

$$b^{[2](k+1)} = b^{[2](k)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial b^{[2]}}$$

$b^{[2]}$ after one update step:

$$b^{[2]} = 0.1 - 0.1 * 0.4 = 0.06$$



$\frac{\partial \mathcal{L}}{\partial W^{[1]}}$
$\frac{\partial \mathcal{L}}{\partial b^{[1]}}$
$\in \mathbb{R}^{3\times1}$
$z^{[1]}$ $a^{[1]}$
$\frac{\partial \mathcal{L}}{\partial W^{[2]}}$
$\frac{\partial \mathcal{L}}{\partial b^{[2]}}$
$z^{[2]}$ $a^{[2]}$
x_1
x_2
0.16
0.12
0.08
0.32
0.24
0.16
0.16
0.12
0.08
$z_1^{[1]}$ $a_1^{[1]}$
$z_2^{[1]}$ $a_2^{[1]}$
$z_3^{[1]}$ $a_3^{[1]}$
0.4
0.8
0.6
0.4
$a^{[2]}$
$\hat{y}$
$\in \mathbb{R}^{1\times3}$
$\in \mathbb{R}^{3\times2}$

Artificial Neural Networks (ANN)

Multilayer Perceptron — One Hidden Layer Network

Update step for $W^{[1]}$ and $b^{[1]}$:

$$W^{[1](k+1)} = W^{[1](k)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial W^{[1]}}$$

$W^{[1]}$ after one update step:

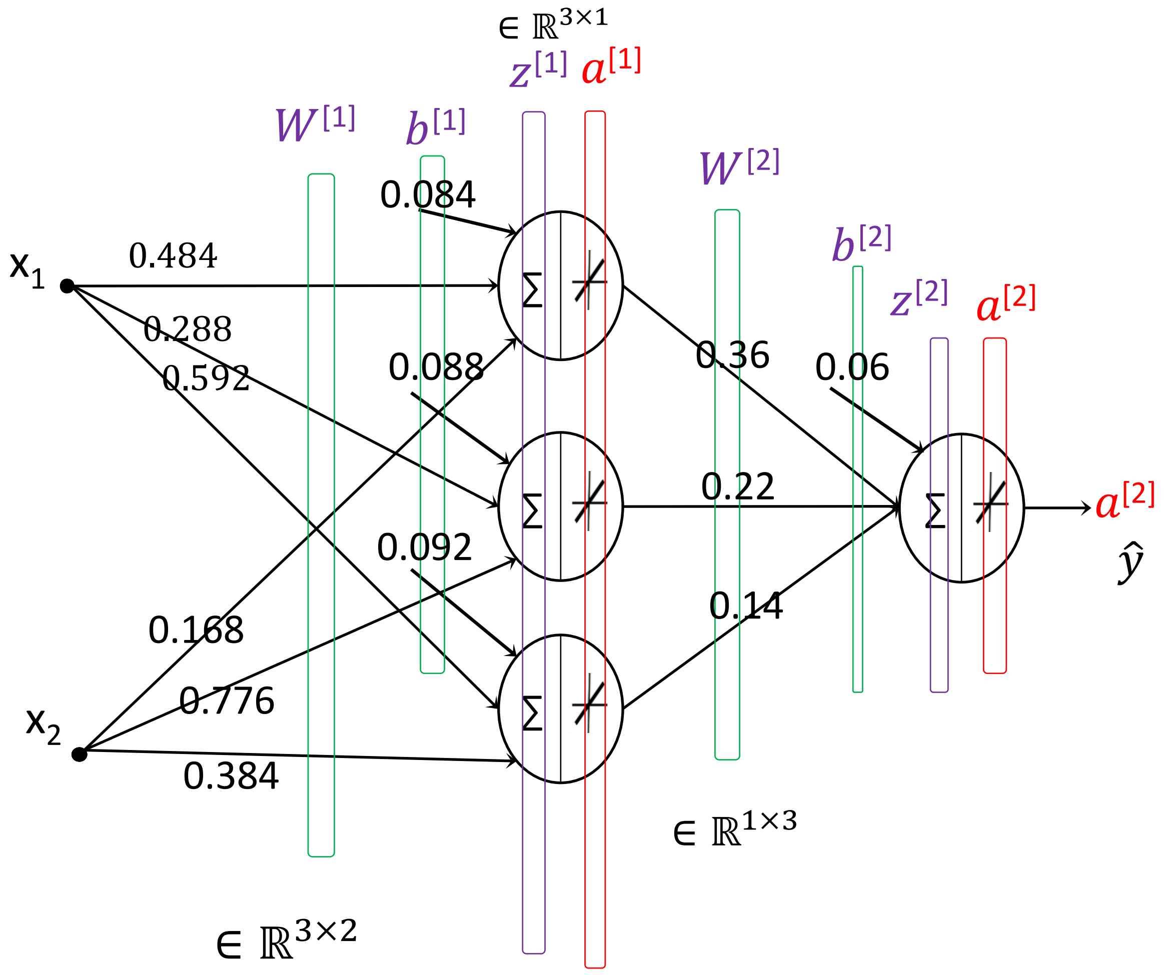
$$W^{[1]} = \begin{bmatrix} 0.5 & 0.2 \\ 0.3 & 0.8 \\ 0.6 & 0.4 \end{bmatrix} - 0.1. \begin{bmatrix} 0.16 & 0.32 \\ 0.12 & 0.24 \\ 0.08 & 0.16 \end{bmatrix}$$

$$= \begin{bmatrix} 0.484 & 0.168 \\ 0.288 & 0.776 \\ 0.592 & 0.384 \end{bmatrix}$$

$$b^{[1](k+1)} = b^{[1](k)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial b^{[1]}}$$

$b^{[1]}$ after one update step:

$$b^{[1]} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} - 0.1. \begin{bmatrix} 0.16 \\ 0.12 \\ 0.08 \end{bmatrix} = \begin{bmatrix} 0.084 \\ 0.088 \\ 0.092 \end{bmatrix}$$

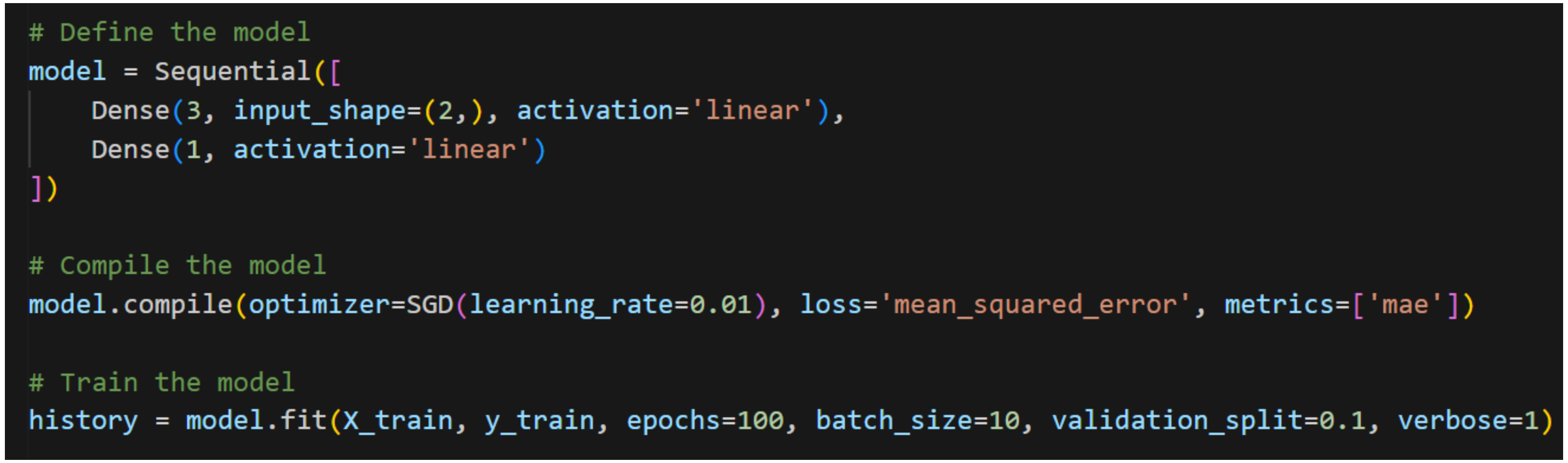


The parameters after one update step

Artificial Neural Networks (ANN)

TensorFlow Implementation of a One-Hidden-Layer Neural Network

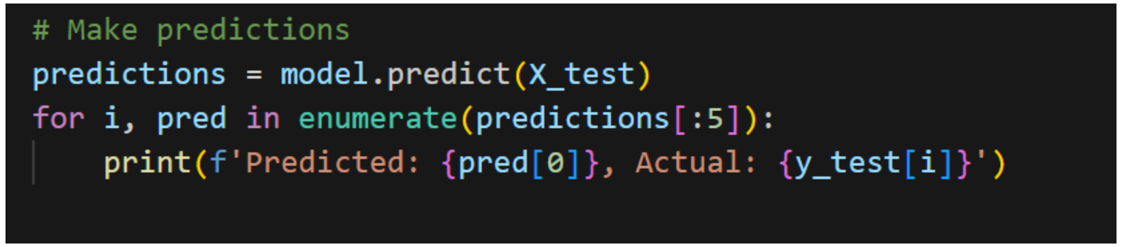
For a regression task:



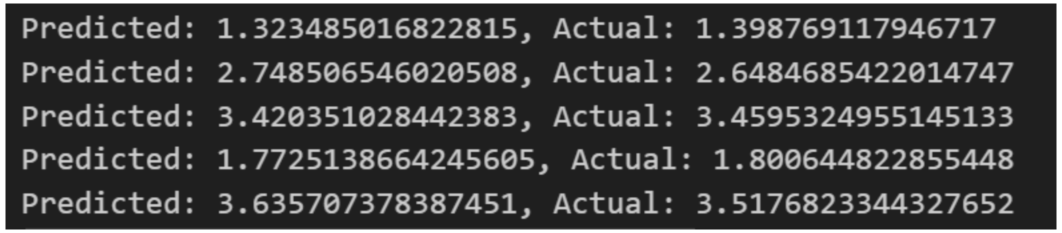
```python
# Define the model
model = Sequential([
    Dense(3, input_shape=(2,), activation='linear'),
    Dense(1, activation='linear')
])

# Compile the model
model.compile(optimizer=SGD(learning_rate=0.01), loss='mean_squared_error', metrics=['mae'])

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.1, verbose=1)
```



```python
# Make predictions
predictions = model.predict(X_test)
for i, pred in enumerate(predictions[:5]):
    print(f'Predicted: {pred[0]}, Actual: {y_test[i]}')
```

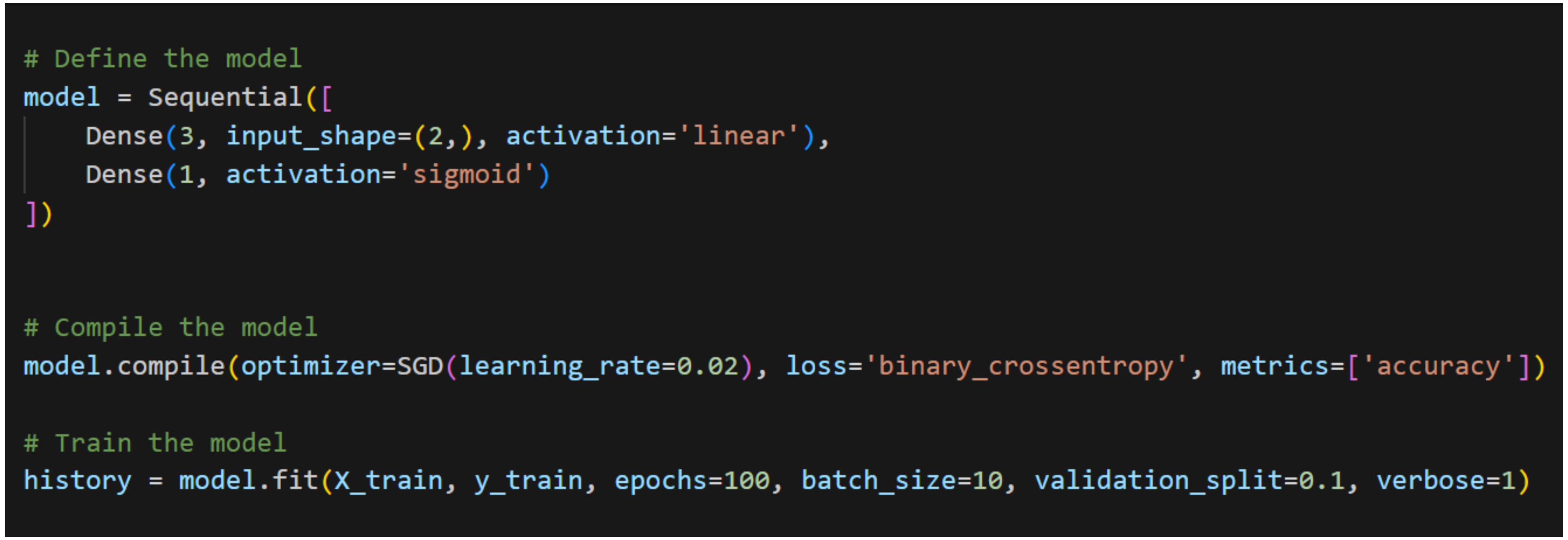


```
Predicted: 1.323485016822815, Actual: 1.398769117946717
Predicted: 2.748506546020508, Actual: 2.6484685422014747
Predicted: 3.420351028442383, Actual: 3.4595324955145133
Predicted: 1.7725138664245605, Actual: 1.800644822855448
Predicted: 3.635707378387451, Actual: 3.5176823344327652
```

Artificial Neural Networks (ANN)

TensorFlow Implementation of a One-Hidden-Layer Neural Network

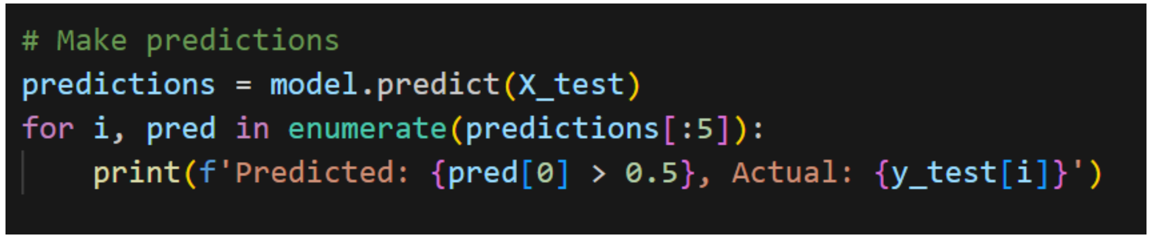
For a binary classification task:



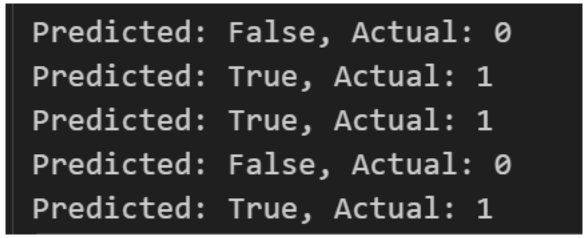
```
# Define the model
model = Sequential([
    Dense(3, input_shape=(2,), activation='linear'),
    Dense(1, activation='sigmoid')
])


# Compile the model
model.compile(optimizer=SGD(learning_rate=0.02), loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.1, verbose=1)
```



```
# Make predictions
predictions = model.predict(X_test)
for i, pred in enumerate(predictions[:5]):
    print(f'Predicted: {pred[0] > 0.5}, Actual: {y_test[i]}')
```

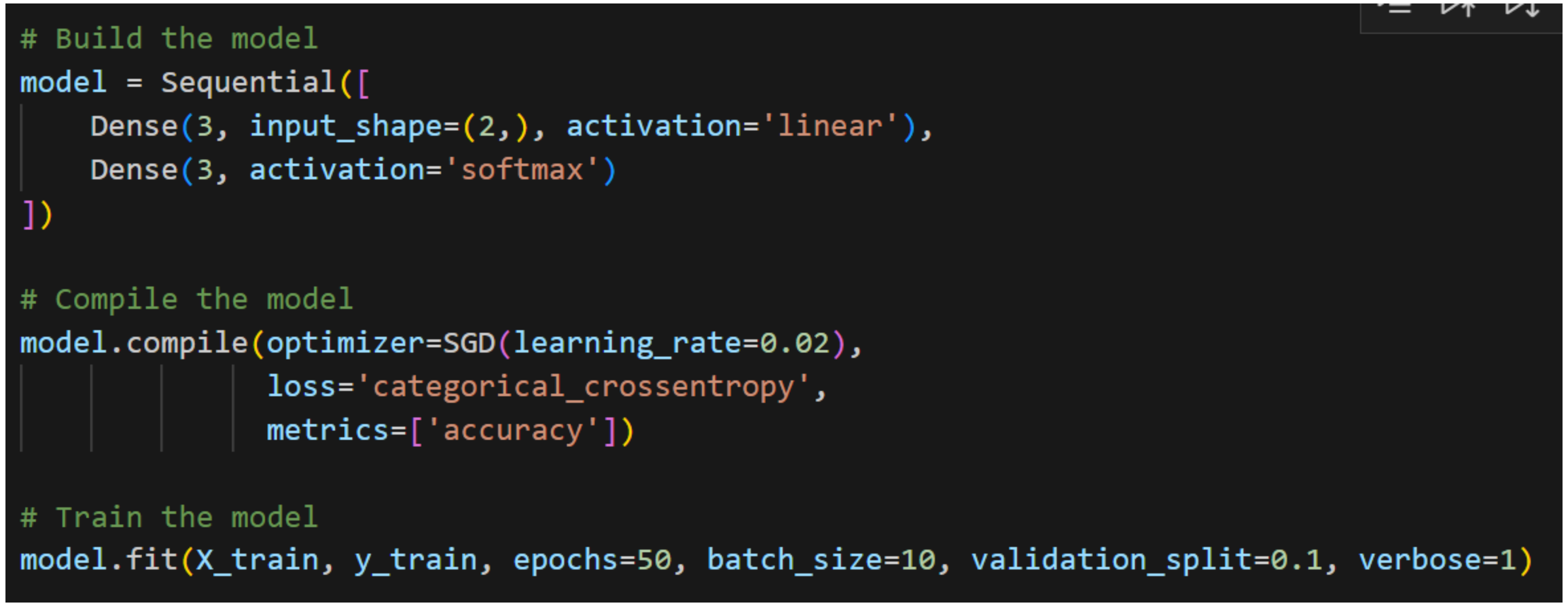


```
Predicted: False, Actual: 0
Predicted: True, Actual: 1
Predicted: True, Actual: 1
Predicted: False, Actual: 0
Predicted: True, Actual: 1
```

Artificial Neural Networks (ANN)

TensorFlow Implementation of a One-Hidden-Layer Neural Network

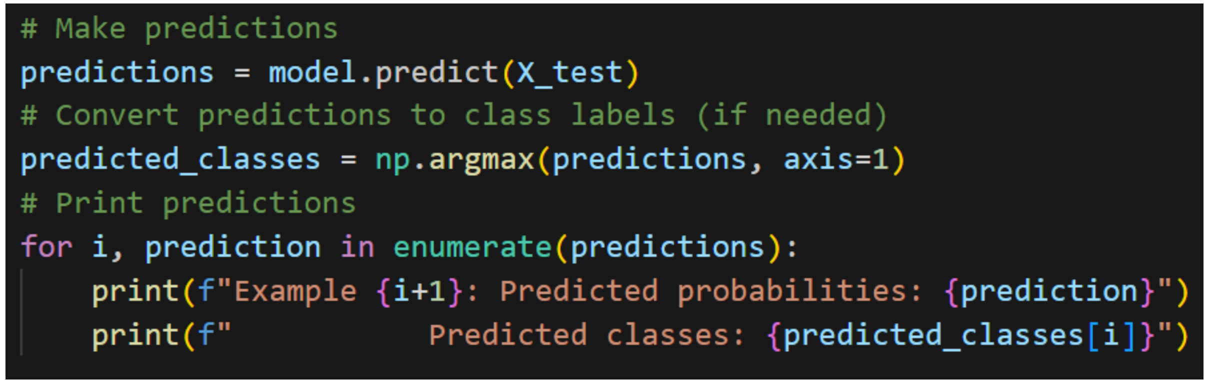
For a multiclass classification task



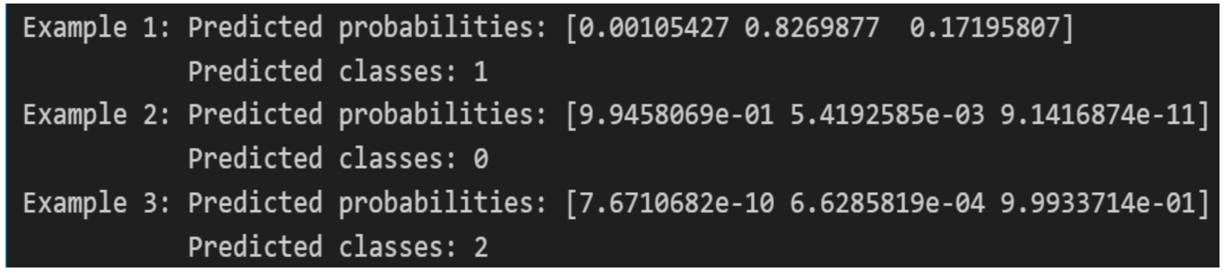
```python
# Build the model
model = Sequential([
    Dense(3, input_shape=(2,), activation='linear'),
    Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer=SGD(learning_rate=0.02),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=10, validation_split=0.1, verbose=1)
```



```python
# Make predictions
predictions = model.predict(X_test)
# Convert predictions to class labels (if needed)
predicted_classes = np.argmax(predictions, axis=1)
# Print predictions
for i, prediction in enumerate(predictions):
    print(f"Example {i+1}: Predicted probabilities: {prediction}")
    print(f"           Predicted classes: {predicted_classes[i]}")
```



```
Example 1: Predicted probabilities: [0.00105427 0.8269877  0.17195807]
           Predicted classes: 1
Example 2: Predicted probabilities: [9.9458069e-01 5.4192585e-03 9.1416874e-11]
           Predicted classes: 0
Example 3: Predicted probabilities: [7.6710682e-10 6.6285819e-04 9.9933714e-01]
           Predicted classes: 2
```

Artificial Neural Networks (ANN)

Shallow And Deep Neural Networks

+ Shallow Neural Networks:
  + They consist of an input layer, a single hidden layer, and an output layer
  + They can be effective in simpler classification or regression tasks, but they may struggle with learning complex patterns in data that require more hierarchical representations
+ Deep Neural Networks:
  + They have multiple hidden layers between the input and output layers
  + They are more computationally expensive to train and evaluate compared to shallow networks, mainly due to their increased number of parameters and computational demands per layer.
  + They can handle large amounts of data and are often used in tasks such as image and speech recognition, and natural language processing

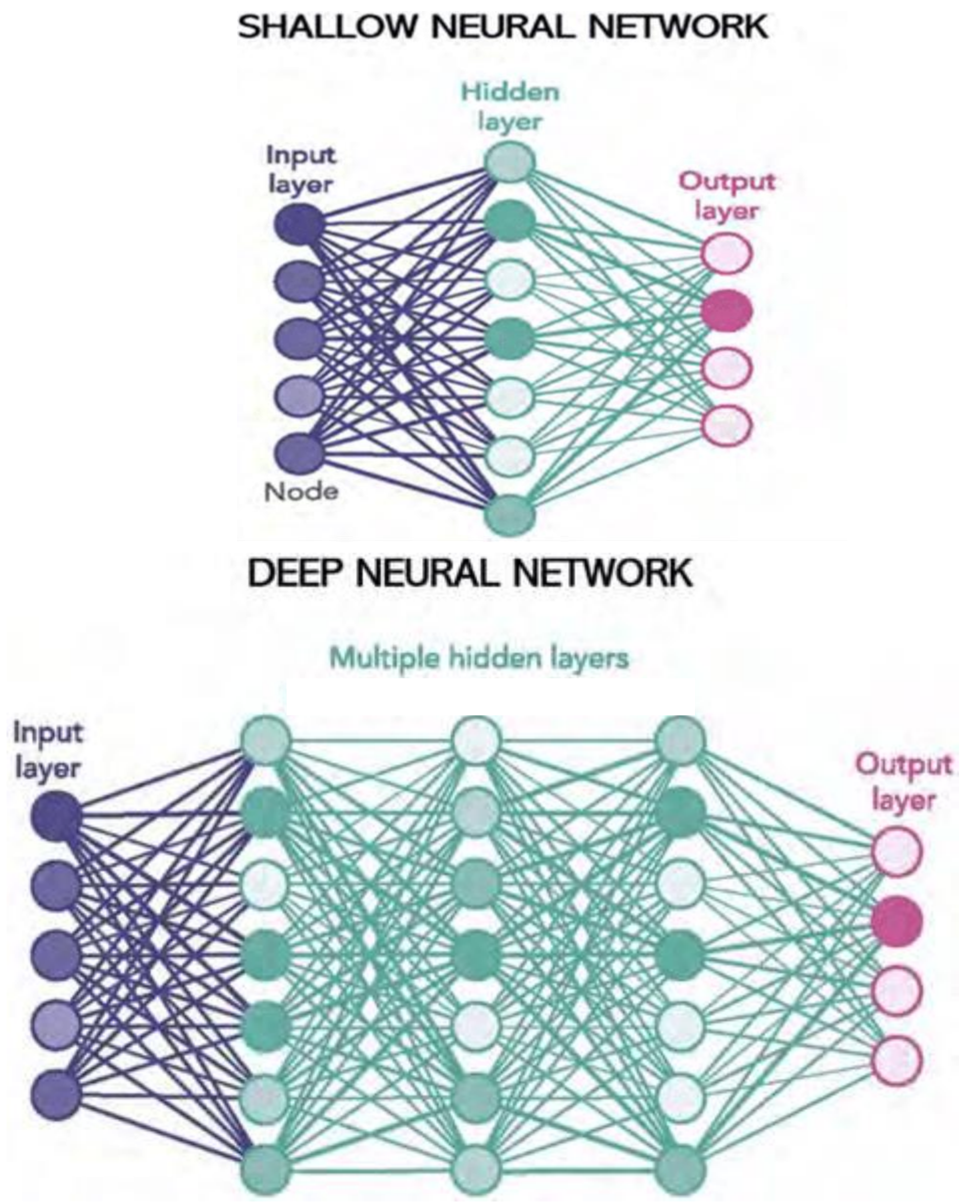


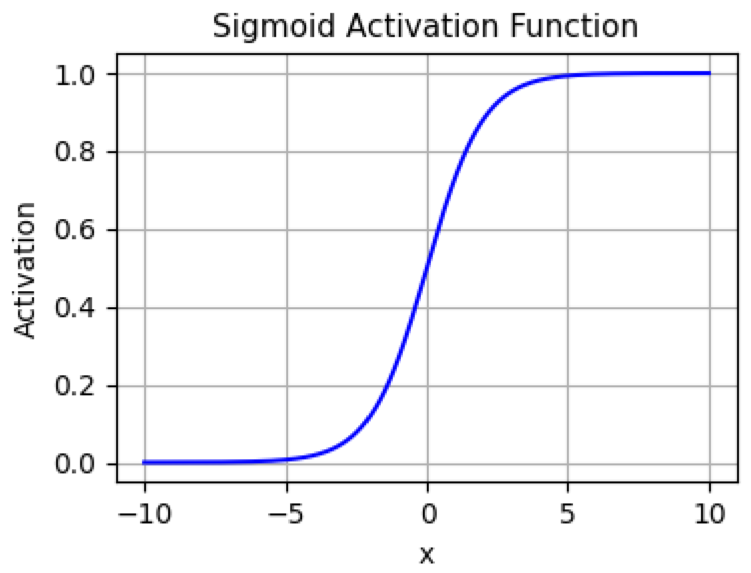
Image: https://www.researchgate.net/figure/Schematic-of-shallow-neural-network-and-deep-neural-network_fig1_346219836

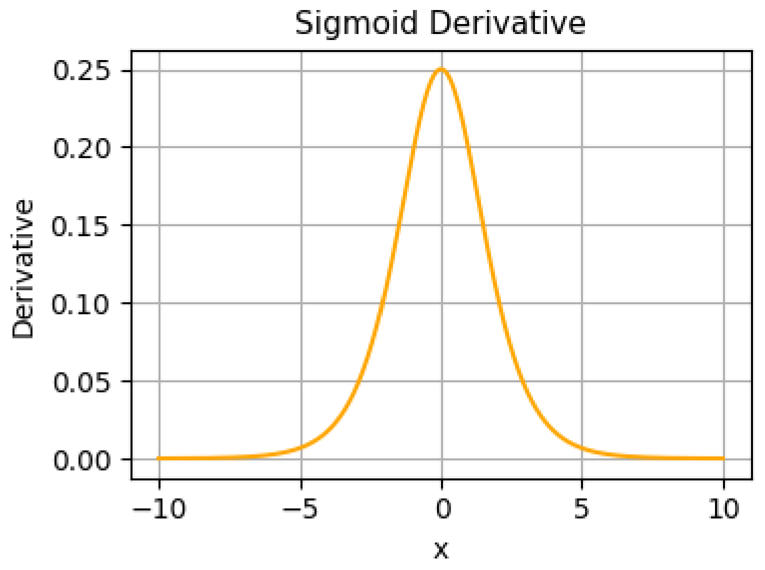
Artificial Neural Networks (ANN)

Activation Functions And Their Derivatives

1. **Sigmoid Function:**
   - Formula: $\sigma(x) = \frac{1}{1+e^{-x}}$
   - Derivative: $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$

- Smooth gradient: The function is differentiable
- Output values bound between 0 and 1: Useful for probability estimation
- Not zero-centered: The outputs are all positive





Artificial Neural Networks (ANN)

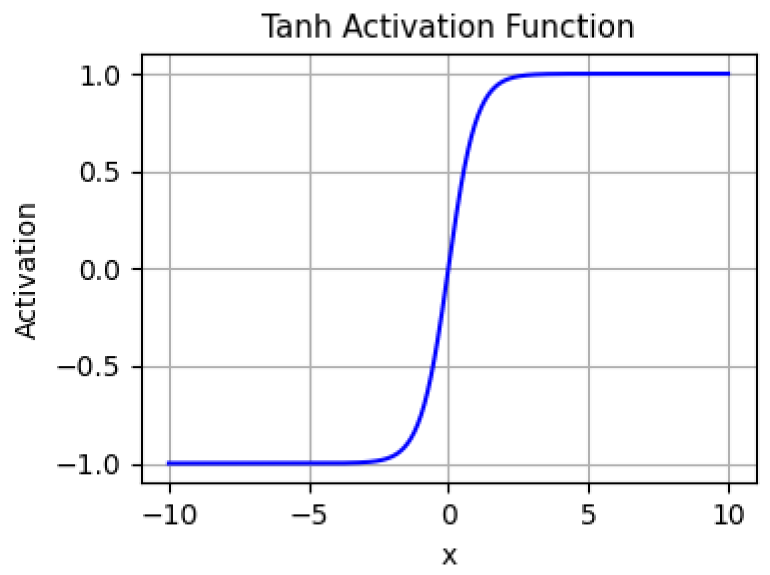
Activation Functions And Their Derivatives

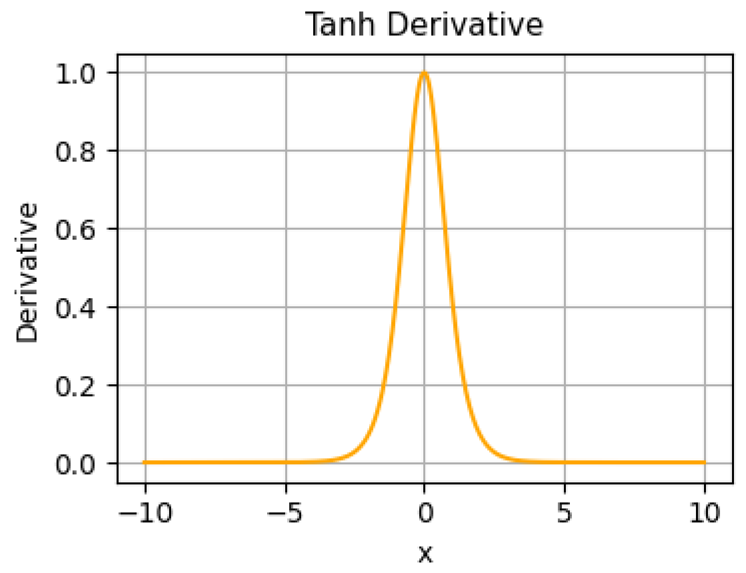
2. **Hyperbolic Tangent (Tanh)**:

   + Formula: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

   + Derivative: $\tanh'(x) = 1 - \tanh^2(x)$

+ Smooth gradient: The function is differentiable
+ Zero-centered: Outputs range between -1 and 1, making the optimization process more efficient





Artificial Neural Networks (ANN)

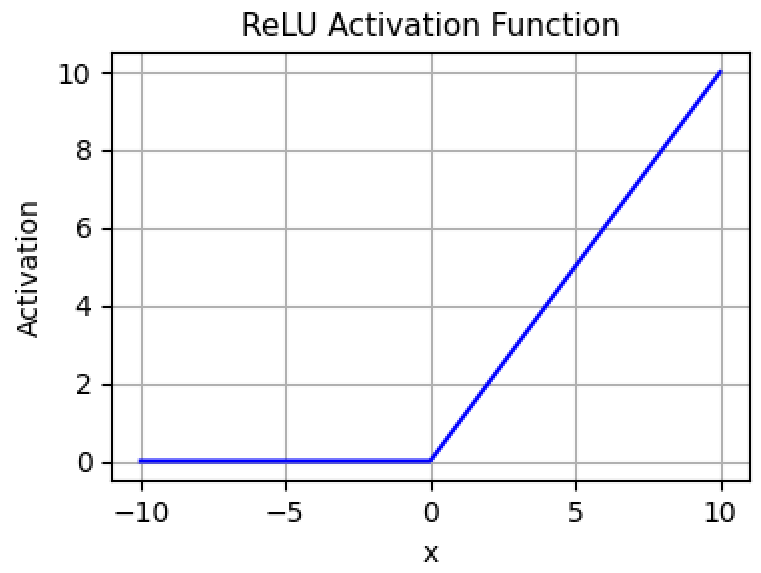
Activation Functions And Their Derivatives

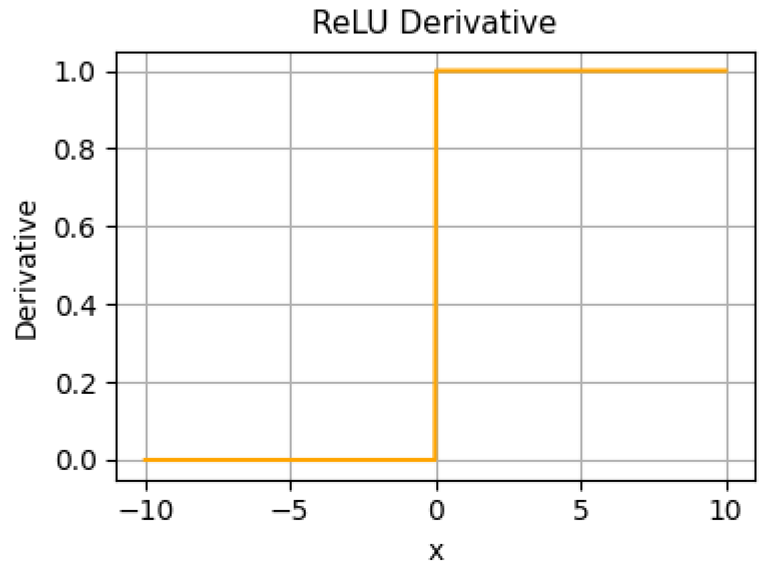
3. **ReLU (Rectified Linear Unit):**

   + Formula: $f(x) = \max(0, x)$

   + Derivative: $$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

+ Computationally efficient: Simple max(0, x) operation.
+ Allows gradients to flow when the input is positive.
+ Sparse activation: Produces a lot of zeros, making the network lightweight.
+ Unbounded output: Can produce very large values, which might lead to numerical instability.





Artificial Neural Networks (ANN)

Activation Functions And Their Derivatives

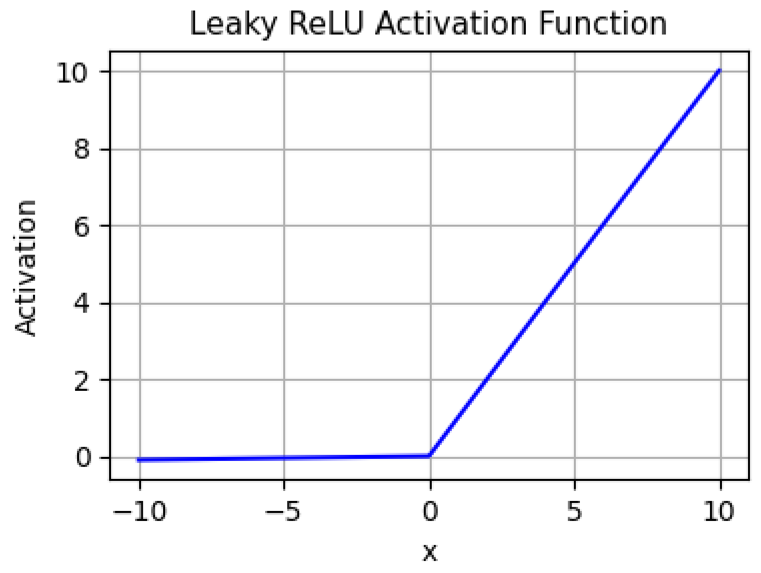
4. **Leaky ReLU:**

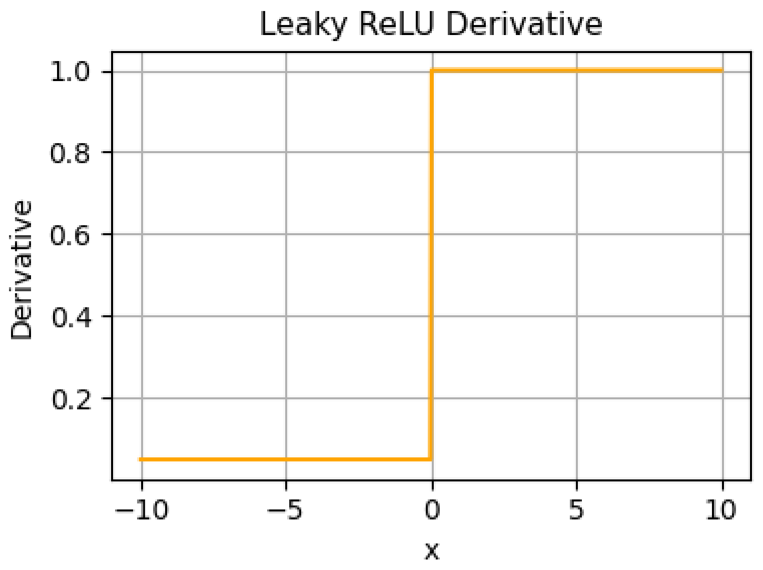
   - Formula: $$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$
   - Derivative:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases}$$

The derivative of the Leaky ReLU function is 1 for positive input and a small constant α for non-positive input. α=0.05 for this graph

- Solves the dying ReLU problem: Allows a small gradient when the input is negative.
- Choice of α: The small constant α needs to be tuned





Artificial Neural Networks (ANN)

Activation Functions And Their Derivatives

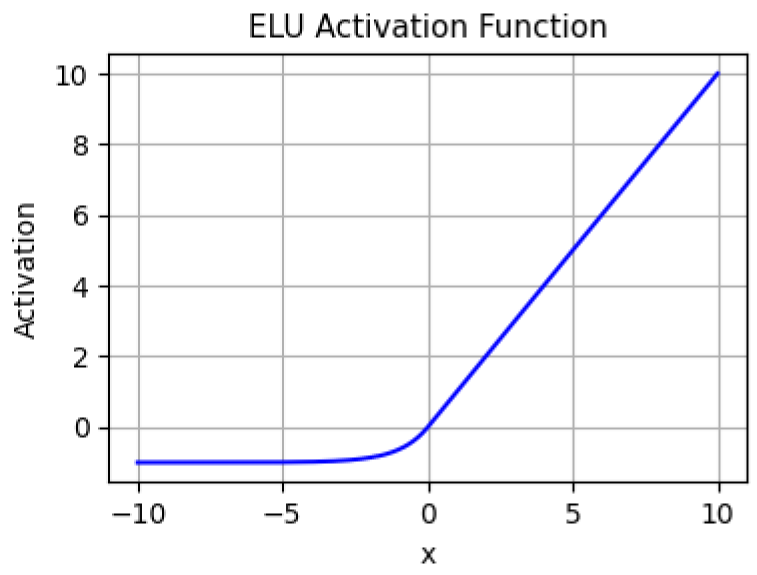
5. ELU (Exponential Linear Unit):

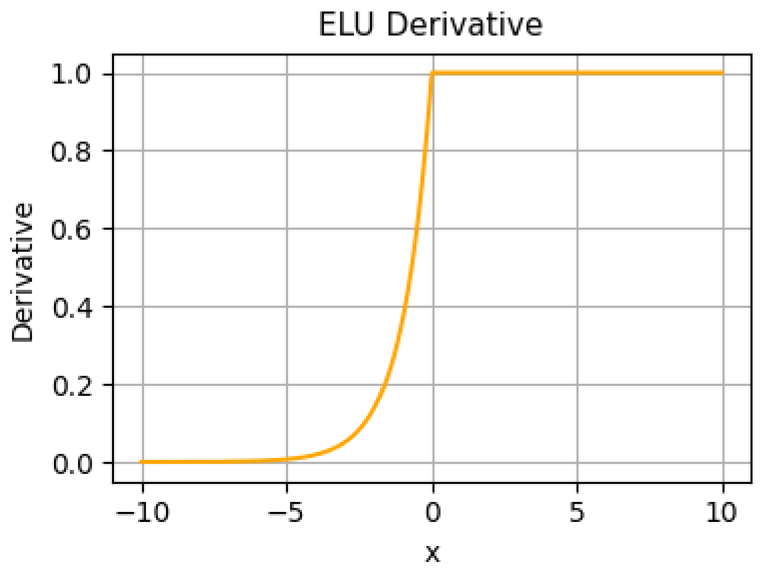
- Formula: $$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$
- Derivative:

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}$$

Where α is a hyperparameter that controls the value to which an ELU saturates for negative net inputs. Typically, α=1

- Smooth transitions: Provides smooth and continuous transitions which are beneficial for gradient-based optimization.
- Potential for slower computation: More complex than ReLU, which can slow down training and inference in practice.

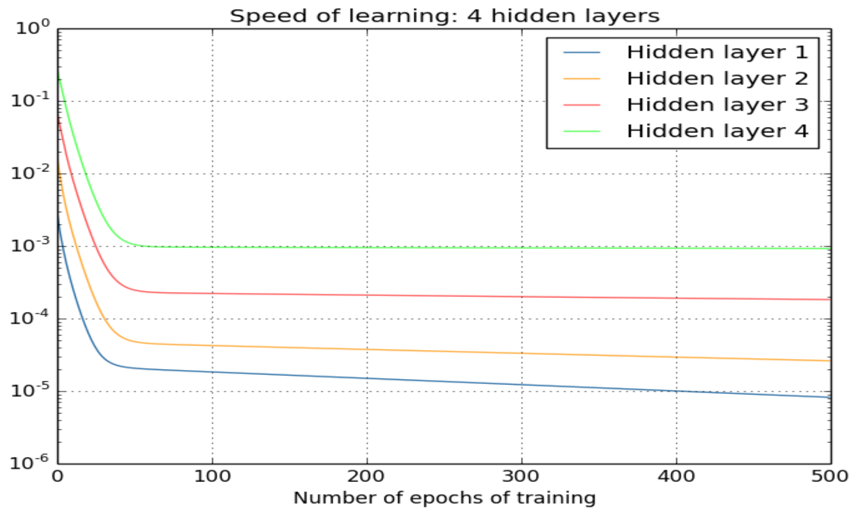




Artificial Neural Networks (ANN)

Challenges in Training Deep Neural Networks: Vanishing Gradients problem

+ Vanishing Gradients problem:
  + It occurs when the gradients of the loss function with respect to the weights become very small during backpropagation
  + This makes the early layers of the network to learn very slowly or not at all and training deep networks becomes very difficult, requiring many epochs and tuning.
  + That is can be because:
  - Activation Functions: Activation functions like sigmoid or tanh squash their input into a small range, resulting in small gradients. For example, the derivative of the sigmoid function has a maximum value of 0.25, which can worse the problem.
  - Weight Initialization: Poor initialization can result in the activations being in the saturation region of the activation function, where gradients are very small.



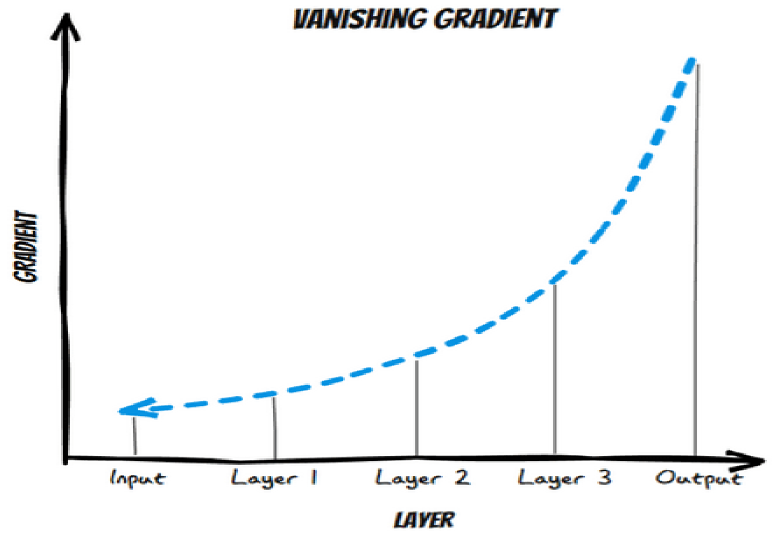
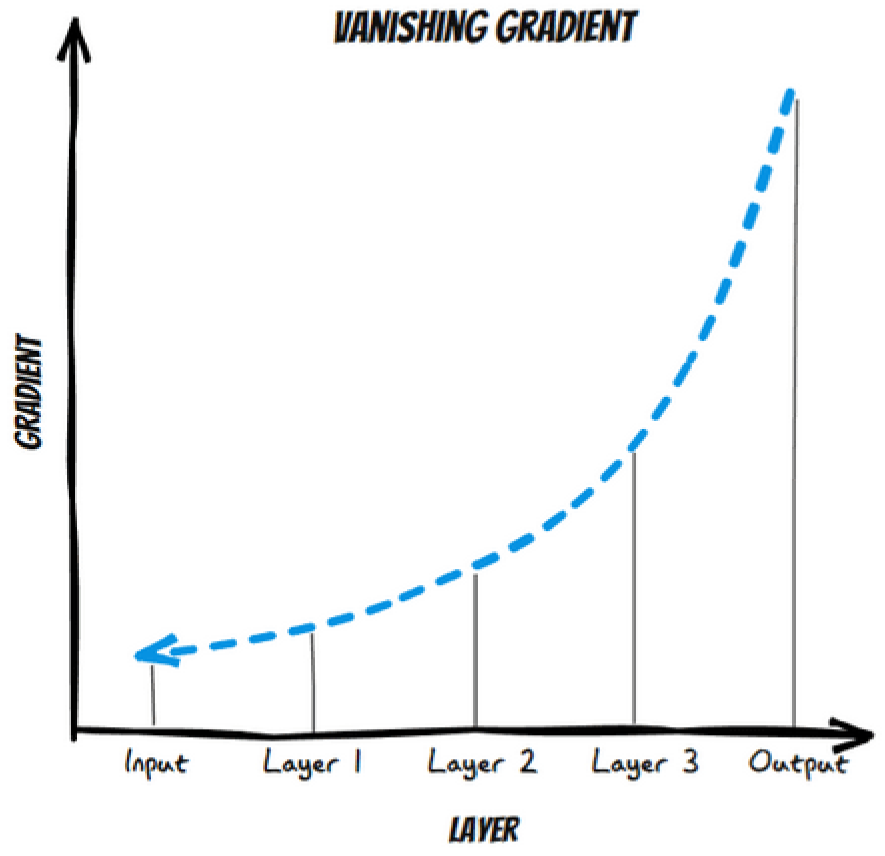


Image: https://www.linkedin.com/posts/gajanan-bhoomkar-551824115_deeplearning-gradientproblems-neuralnetworks-activity-7122187985374056448-XFkS/ in June2024 and http://neuralnetworksanddeeplearning.com/chap5.html

Artificial Neural Networks (ANN)

Challenges in Training Deep Neural Networks: Vanishing Gradients problem

+ To reduce Vanishing Gradients problem:
  + Use ReLU (Rectified Linear Unit) or its variants (Leaky ReLU, Parametric ReLU) instead of sigmoid or tanh, as ReLU does not saturate in the positive domain and has a gradient of 1 for positive values.
  + Batch Normalization: Normalizes the inputs to each layer, helping maintain a more stable distribution of activations throughout the network.

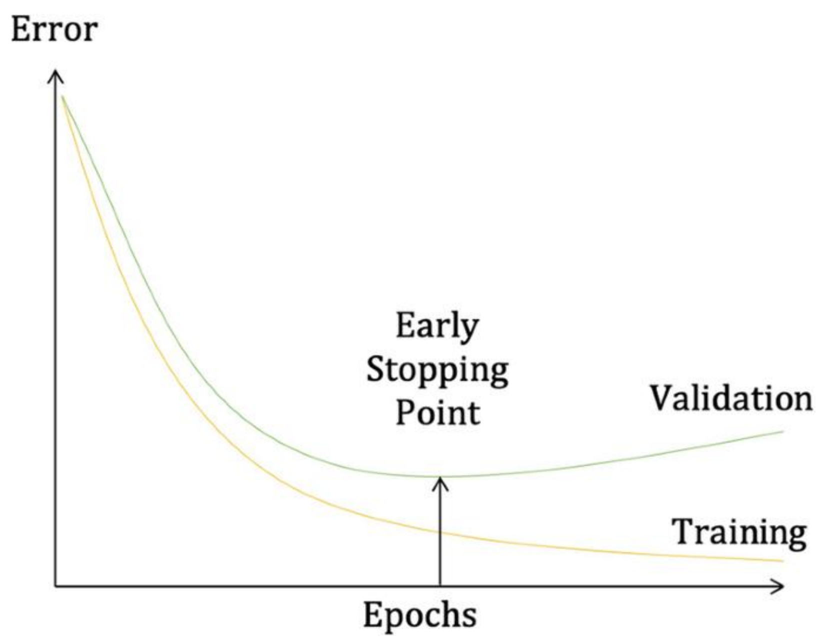


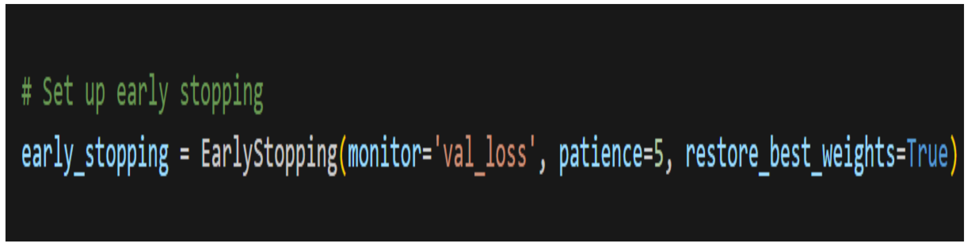
[Image: https://www.linkedin.com/posts/gajanan-bhoomkar-551824115_deeplearning-gradientproblems-neuralnetworks-activity-7122187985374056448-XFkS/](https://www.linkedin.com/posts/gajanan-bhoomkar-551824115_deeplearning-gradientproblems-neuralnetworks-activity-7122187985374056448-XFkS/)

Artificial Neural Networks (ANN)

Challenges in Training Deep Neural Networks: Overfitting

- Techniques to avoid overfitting:
  - Simplify the Model: Reduce the complexity of the neural network by decreasing the number of layers or the number of neurons per layer
  - More Training Data: Increase the size of the training dataset to help the model generalize better by collect more data examples or Artificially by creating modified versions of the data (Data Augmentation)
  - Early Stopping: Stop training when the performance on the validation set starts to degrade
  - Regularization:
  - L1 Regularization (Lasso): Adds the absolute value of weights to the loss function
  - L2 Regularization (Ridge): Adds the squared value of weights to the loss function





```
# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

Image: https://link.springer.com/chapter/10.1007/978-3-030-14596-5_4 https://tahera-firdose.medium.com/dropouts-enhancing-training-stability-and-generalization-badd5069a744

Artificial Neural Networks (ANN)

Challenges in Training Deep Neural Networks: Overfitting

+ Techniques to avoid overfitting:
  + Regularization:
  - Dropout:
    - Randomly drops units (along with their connections) from the neural network during training
    - During Training: For each training step, each neuron (except the output neurons) is kept with a probability p and dropped (set to zero) with a probability 1−p (dropout rate)
    - During Testing/Inference: All units are kept with corresponding weights multiplied by the probability p of keeping the given unit during the training phase.

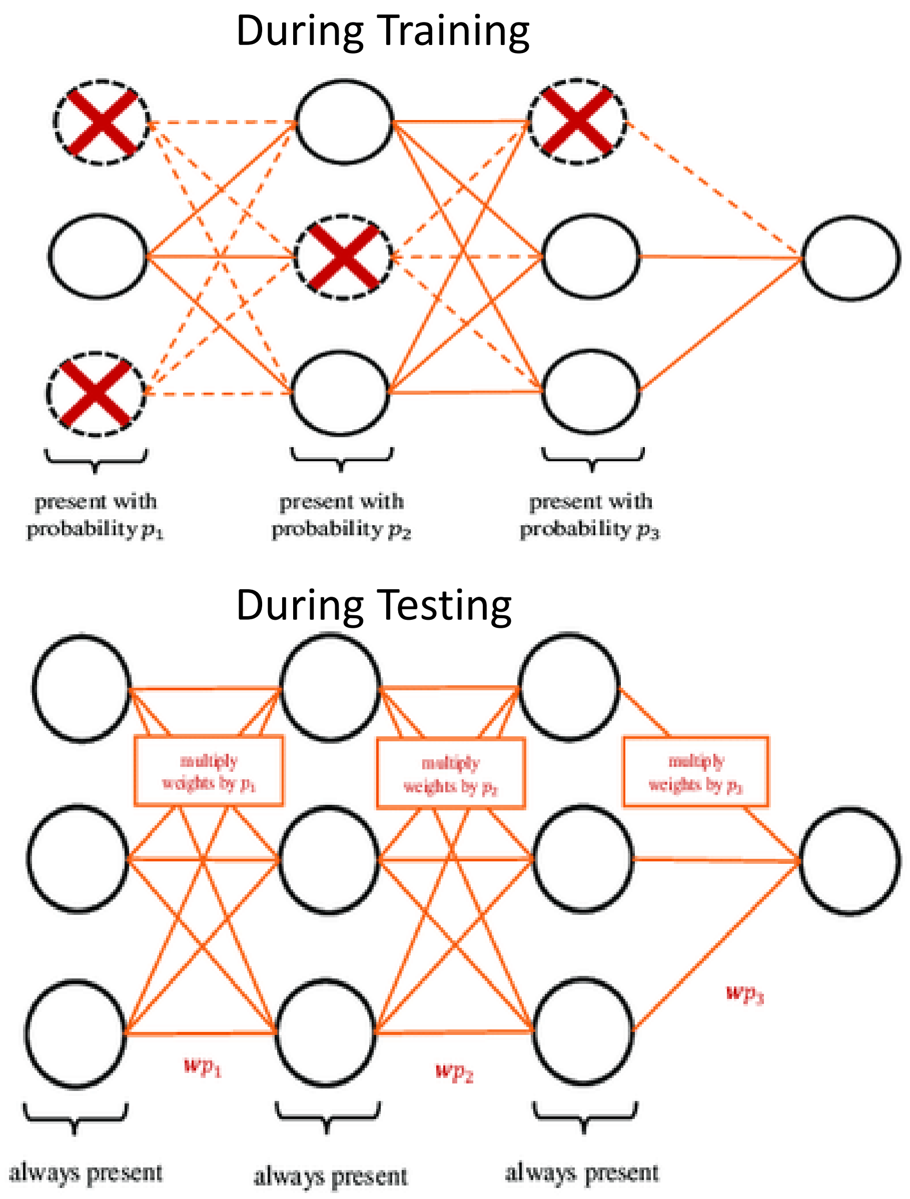

During Training
present with probability p_1
present with probability p_2
present with probability p_3
During Testing
multiply weights by p_1
multiply weights by p_2
multiply weights by p_3
wp_1
wp_2
wp_3
always present
always present
always present

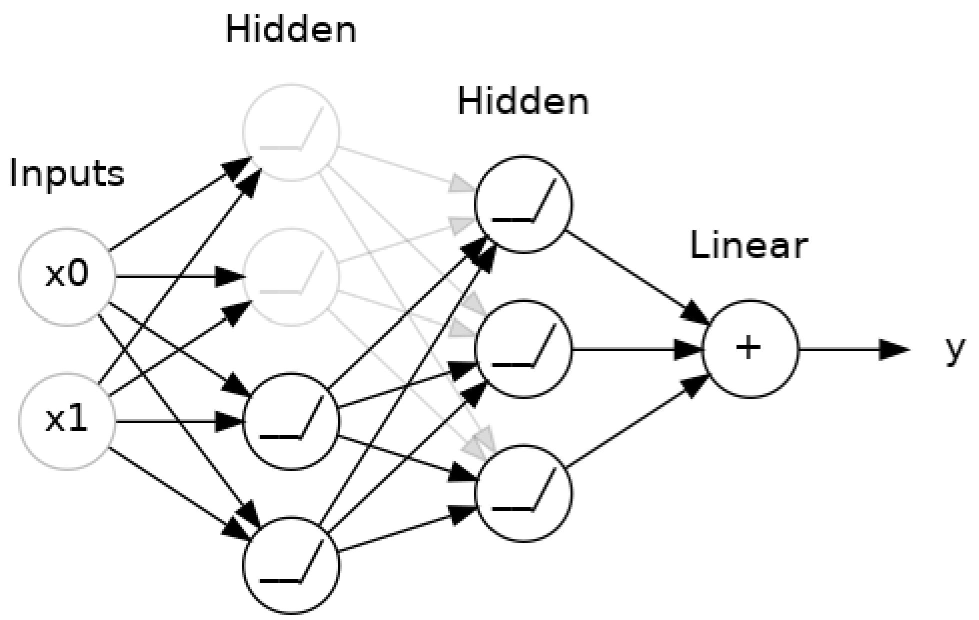
Image: https://www.researchgate.net/figure/Dropout-during-the-training-left-and-testing-phase-right_fig1_346691466

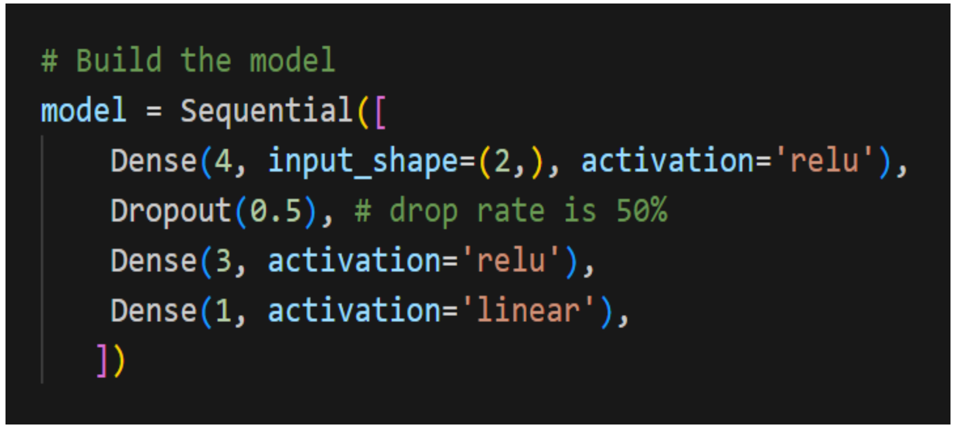
Artificial Neural Networks (ANN)

Challenges in Training Deep Neural Networks: Overfitting

+ Techniques to avoid overfitting:
  + Regularization:
  - Dropout: Example
    - 50% Dropout is applied on the first hidden layer
    - During Training Phase, 50% of the neurons in the layer to which dropout is applied are randomly selected and ignored (dropped out)

During Training





```
# Build the model
model = Sequential([
    Dense(4, input_shape=(2,), activation='relu'),
    Dropout(0.5), # drop rate is 50%
    Dense(3, activation='relu'),
    Dense(1, activation='linear'),
  ])
```

Image: https://tahera-firdose.medium.com/dropouts-enhancing-training-stability-and-generalization-badd5069a744

Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Elektro- und
Informationstechnik

www.h-ka.de

Notation Style

Concept	Symbol	Case	Meaning
Weight vector	w	Lowercase	Vector of weights for a linear model (1D)
Weight matrix	$W^{[l]}$	Uppercase	Weight matrix at layer l
Bias scalar	b	Lowercase	Bias for a neuron
Bias vector	$b^{[l]}$	Lowercase	Bias vector at layer l
Input feature vector	x	Lowercase	One example's feature vector
Activation	$a^{[l]}$	Lowercase	Activation at layer l