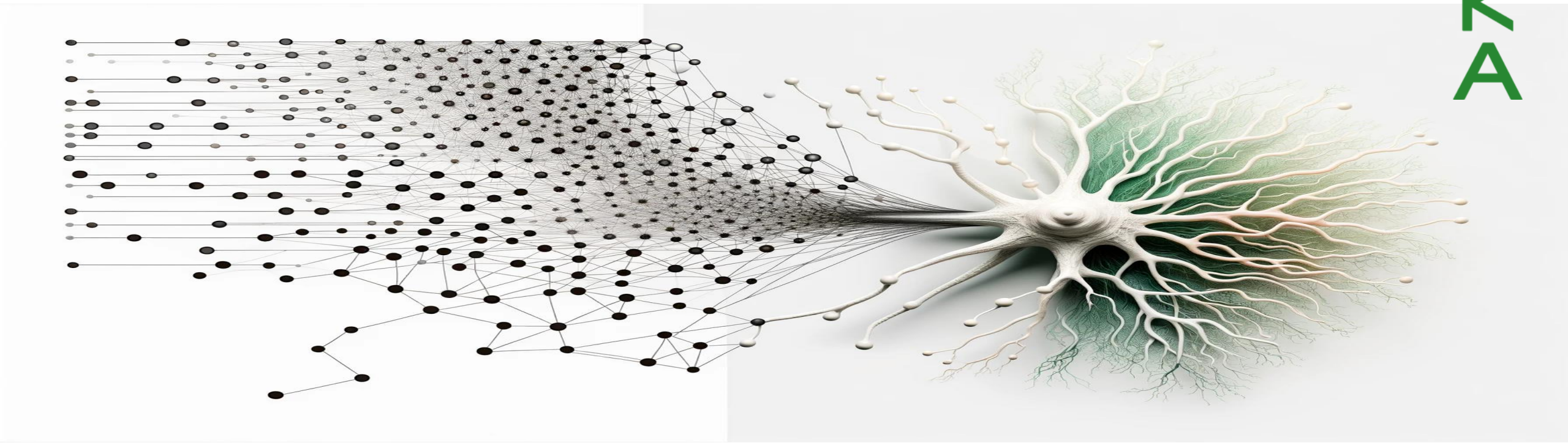
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Elektro- und
Informationstechnik

Artificial Neural Networks (ANN)

K
A



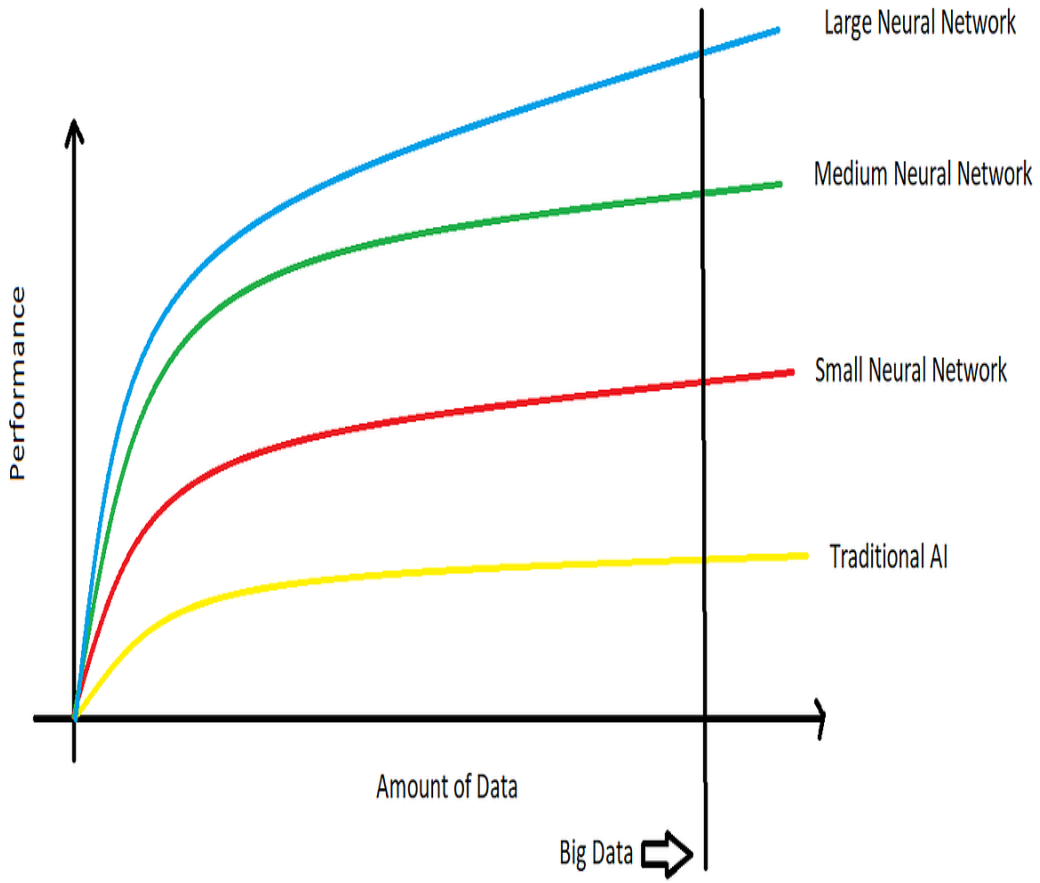
Source: DALL.E

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.

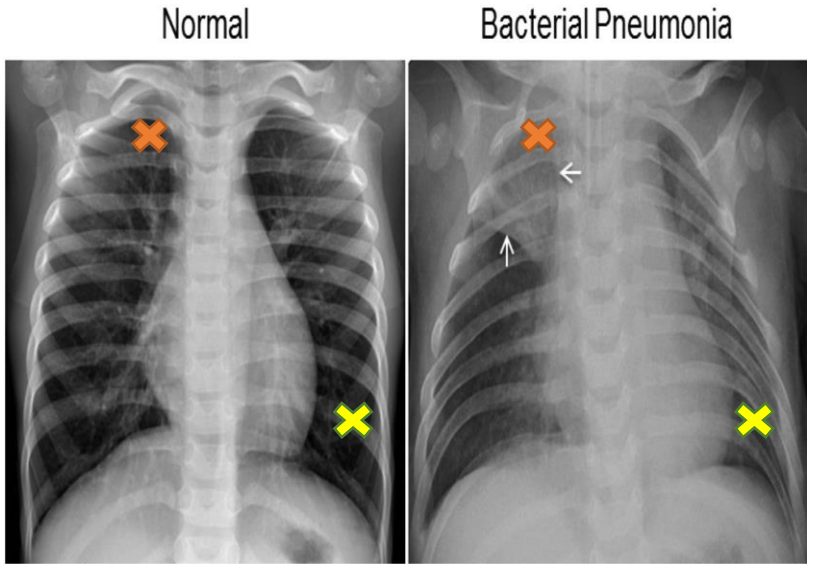


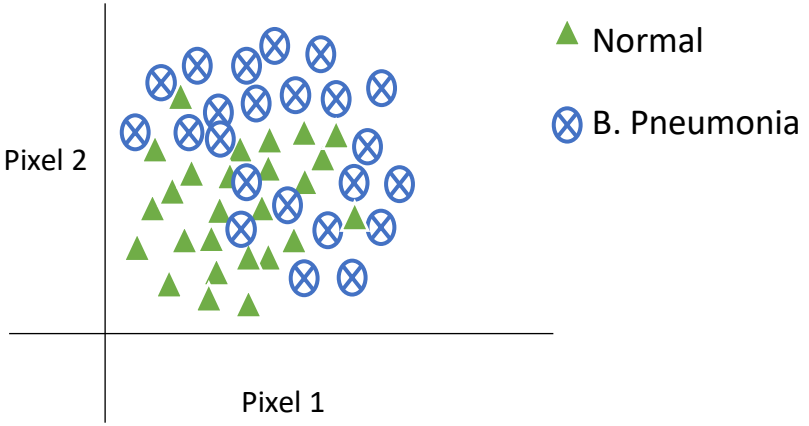
Image_source: https://medium.com/@didrichsen./demystifying-artificial-intelligence-a-basic-introduction-9f6386cb5439

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.

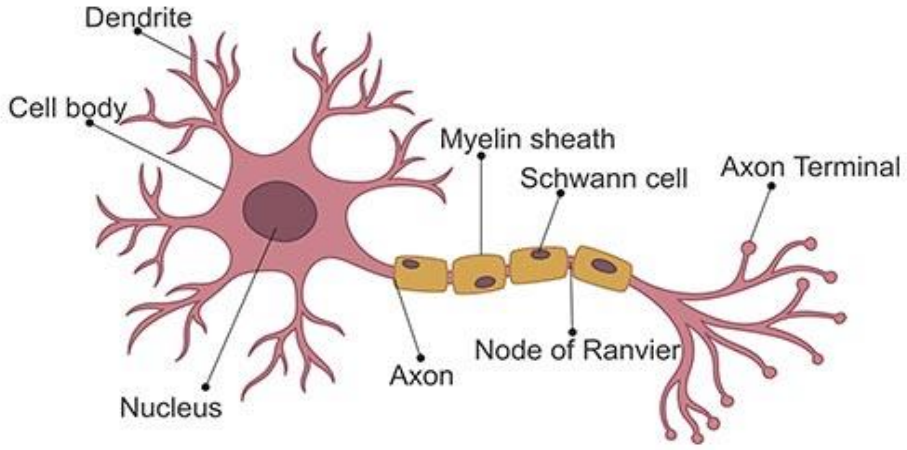




Artificial Neural Networks (ANN)

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



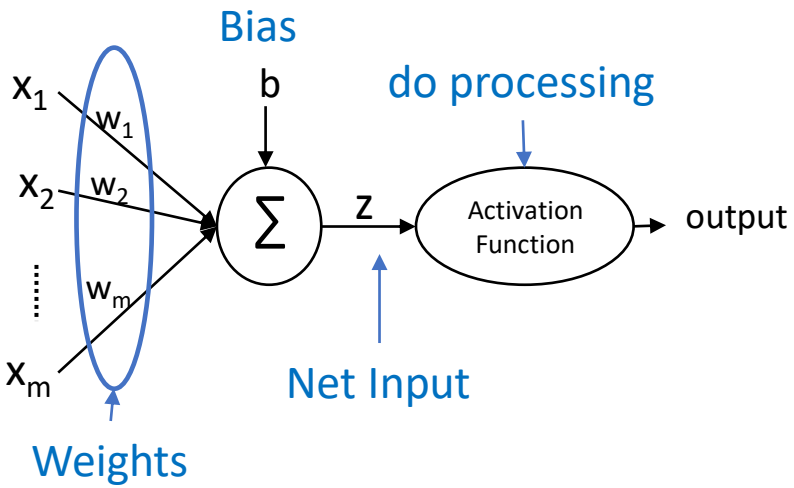


Image source: https://www.cusabio.com/Cell-Marker/Neuron-Cell.html

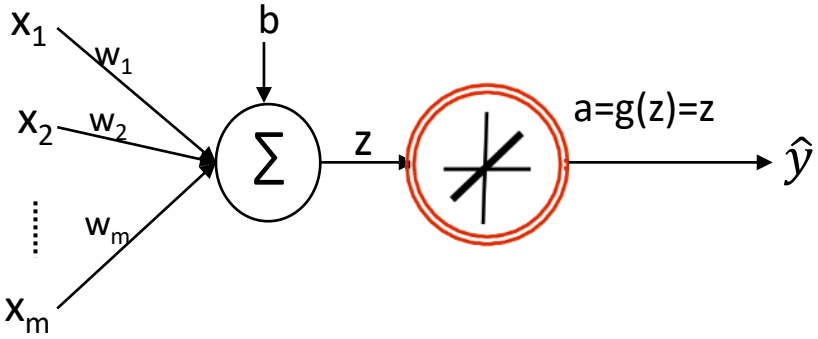
Artificial Neural Networks (ANN)

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 $= \hat{y} = w_1x_1 + w_2x_2 + \cdots + w_mx_m + b = w^T x + b$



Linear Regression Unit

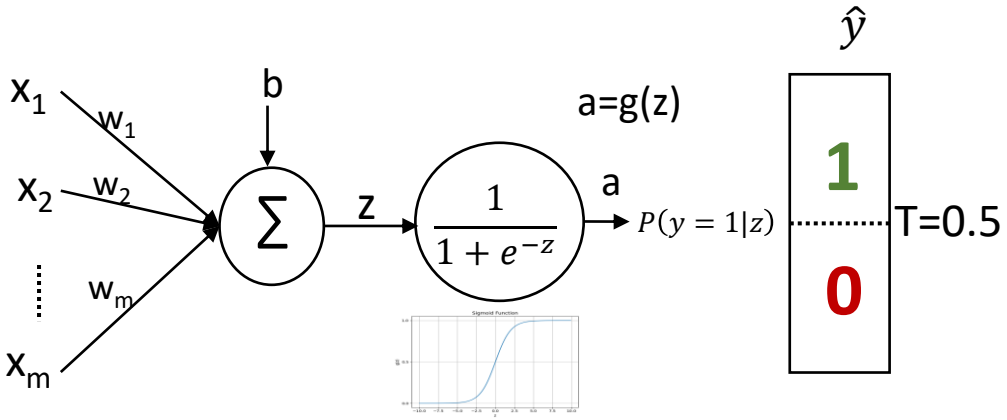
+ If the activation function is a sigmoid function , where

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

  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:

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

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

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

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

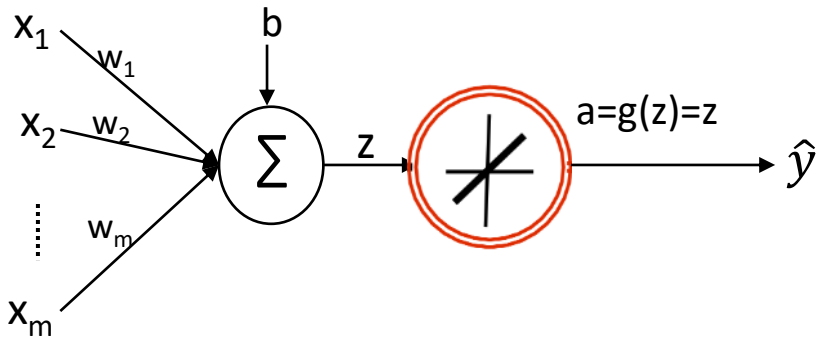
  - Overall loss in the case of linear regression Unit,

$$L(w,b) = \frac{1}{n}\sum_{i=1}^{n}(a^{(i)} - y^{(i)})^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{n}\sum_{i=1}^{n}(g(z^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}((w_1 x_1{}^{(i)} + \cdots + w_m x_m{}^{(i)} + b) - y^{(i)})^2$$

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

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

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

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

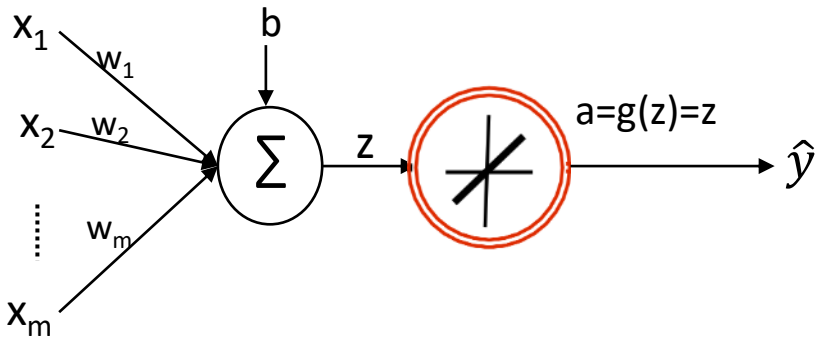
$$\frac{\partial L(w,b)}{\partial w_m} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_m}$$

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

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

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

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



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:

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

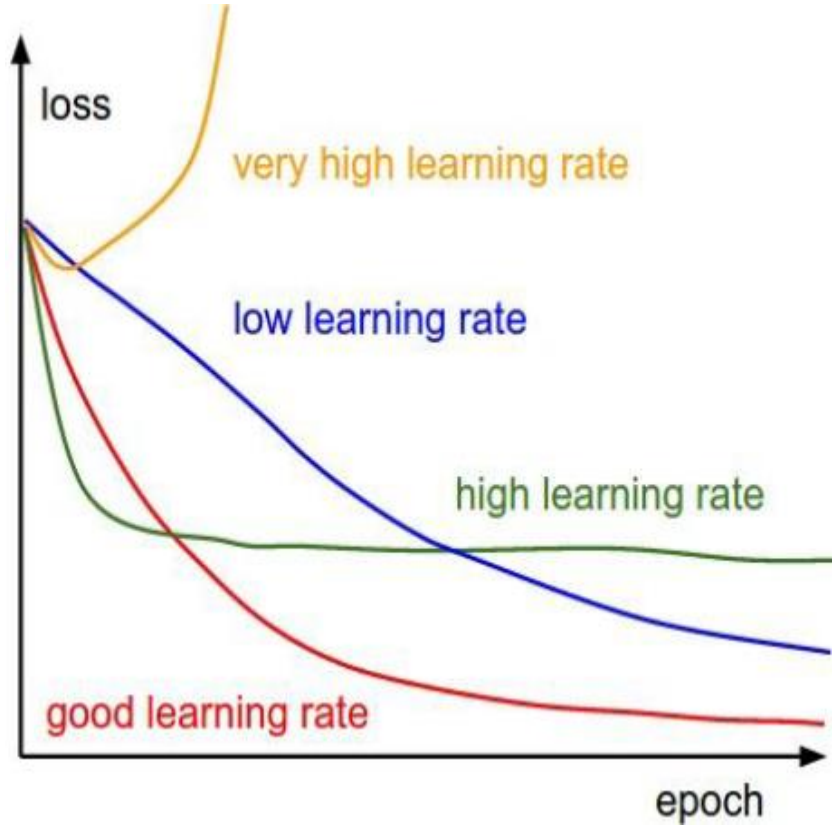
$$= w_1^{[k]} - \eta \cdot \frac{2}{n} \sum_{i=1}^{n} (g(z^{(i)}) - y^{(i)})\, x_1{}^{(i)}$$

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

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

  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,

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

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

  It is the binary cross entropy

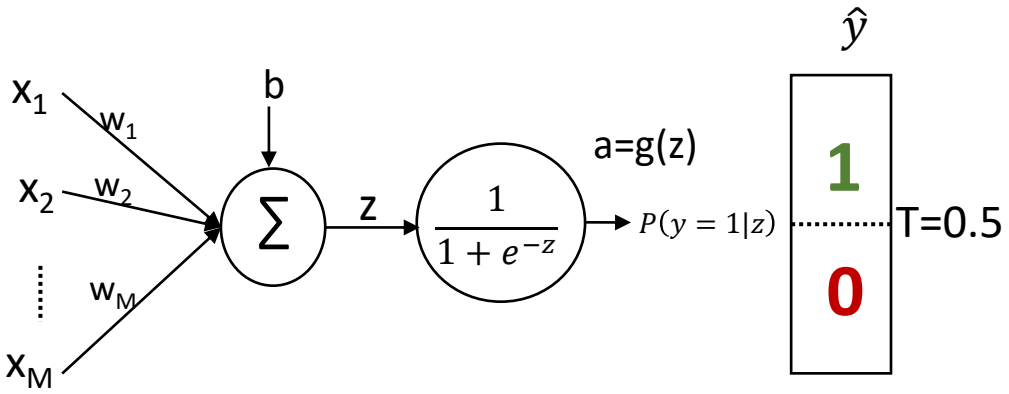
- The steps involved in batch Gradient Descent:
  - Initialize the weights and the bias with small random values to get started with the iteration process
  - Keep on iterating for k = 0, 1, 2,……. using the following update

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

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

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

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



Logistic Regression Unit

Artificial Neural Networks (ANN)

Backpropagation And Gradient Descent For A Single Neuron

+ Gradient calculations for the logistic regression unit

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

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

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

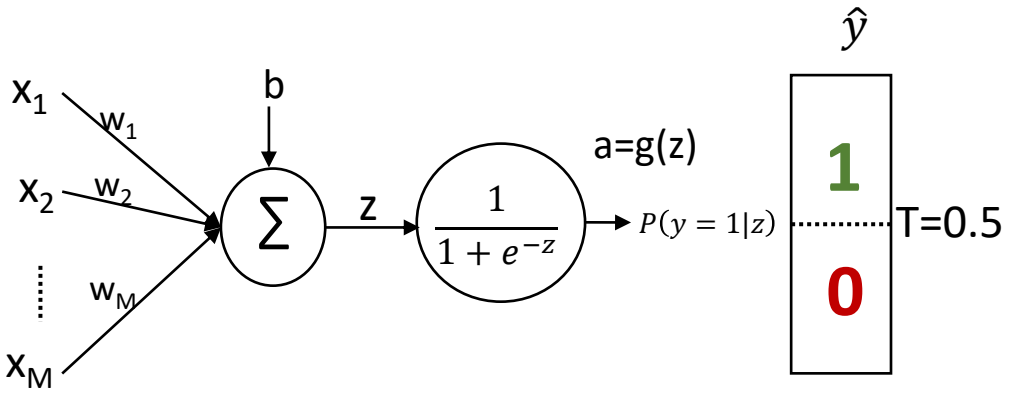
$$\frac{\partial L(w,b)}{\partial w_m} = \frac{\partial L(w,b)}{\partial g(z^{(i)})} \cdot \frac{\partial g(z^{(i)})}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_m}$$

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

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

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

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



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:

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

$$= \frac{1}{n}\sum_{i=1}^{n}(-y^{(i)} + y^{(i)} \cdot g\left(z^{(i)}\right) + g\left(z^{(i)}\right) - y^{(i)} \cdot g\left(z^{(i)}\right)) \cdot x_1{}^{(i)} = \frac{1}{n}\sum_{i=1}^{n}(g\left(z^{(i)}\right) - y^{(i)}) \cdot x_1{}^{(i)}$$

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

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

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

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

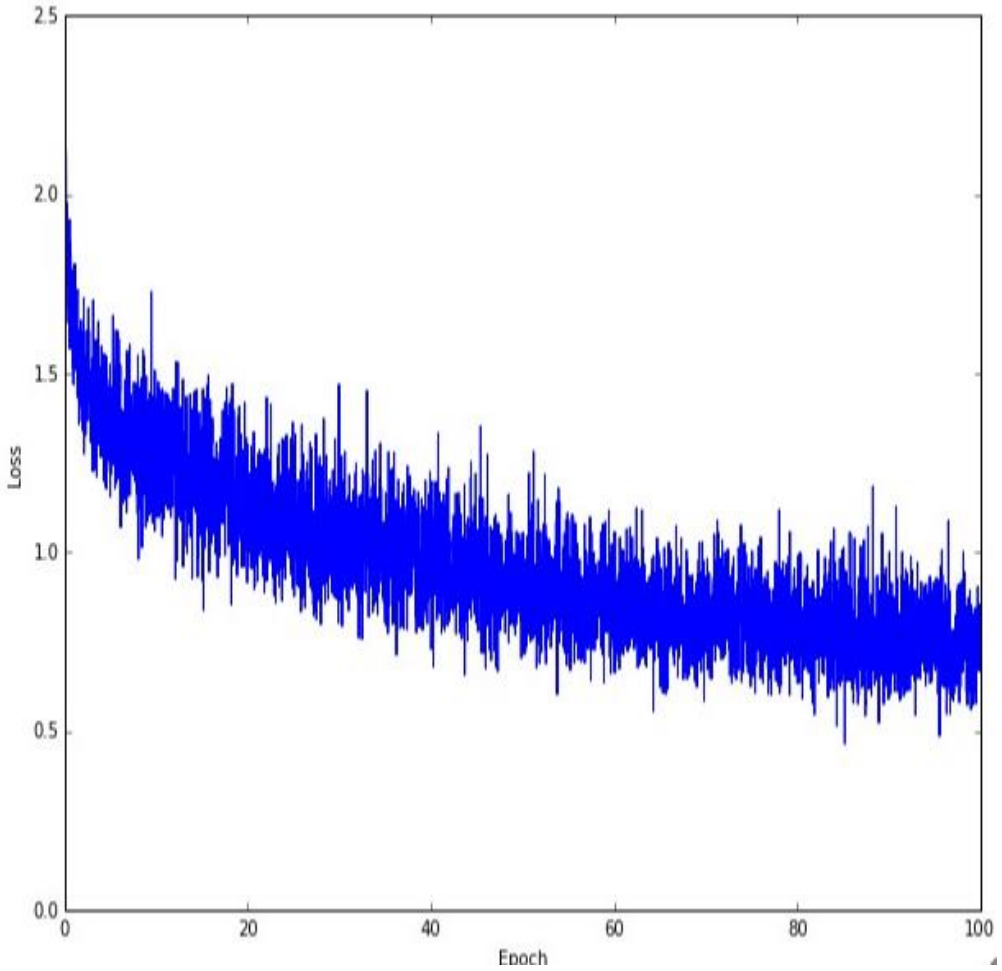
Artificial Neural Networks (ANN)

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

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

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

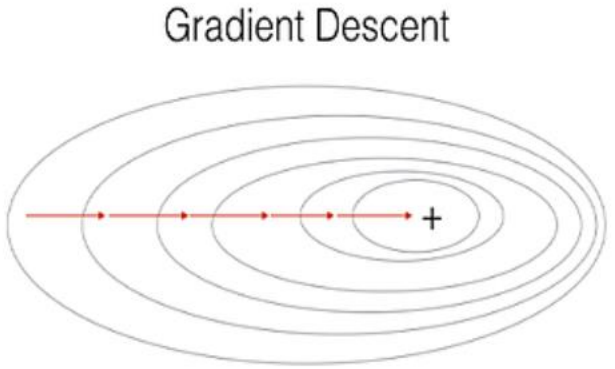


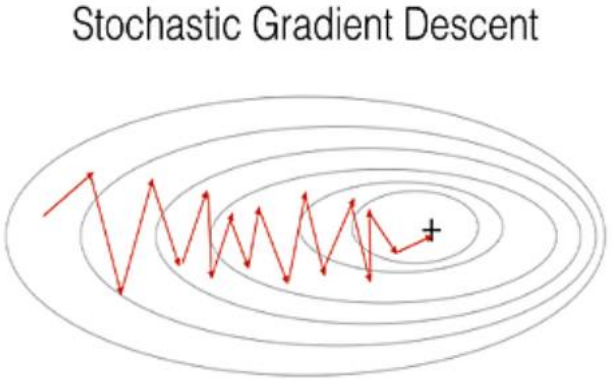
SGD fluctuation.

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





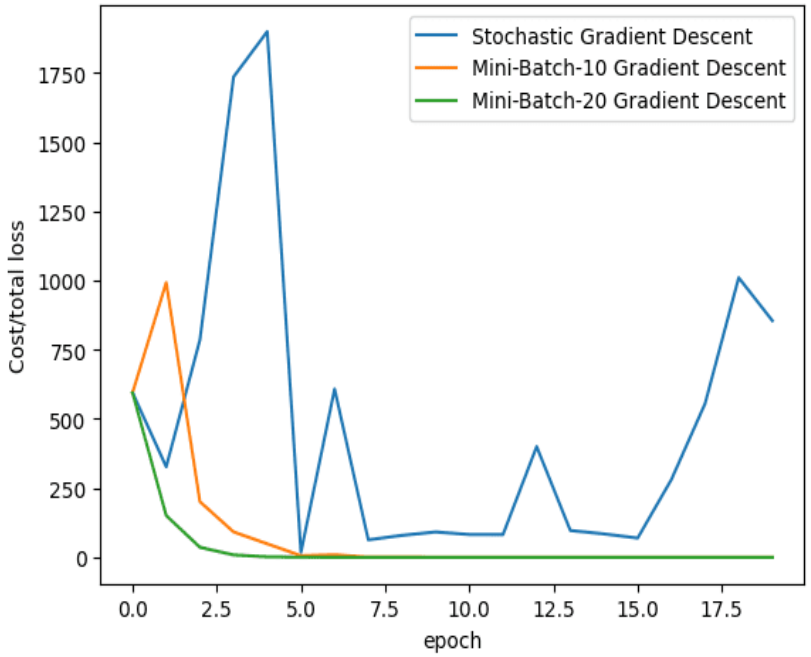
Artificial Neural Networks (ANN)

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

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

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



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

Artificial Neural Networks (ANN)

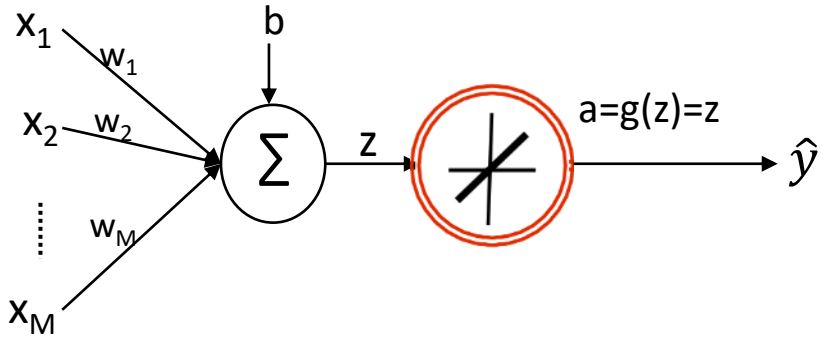
TensorFlow Implementation For A Single Neuron

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)

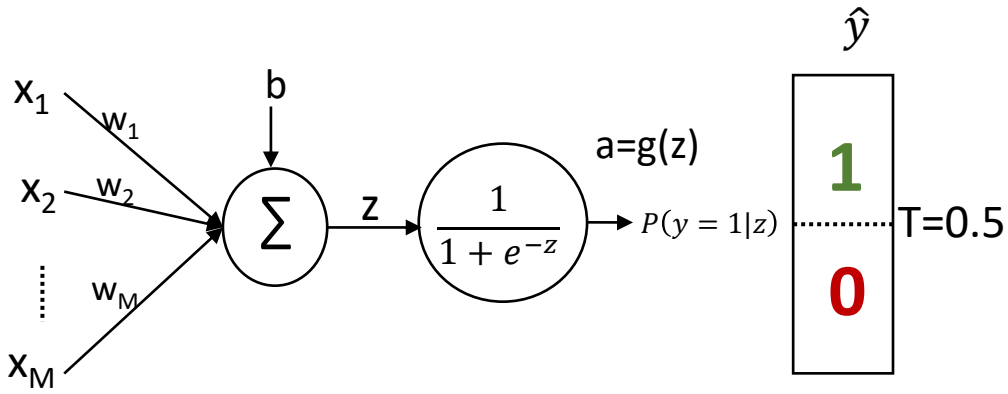
TensorFlow Implementation For A Single Neuron

For a binary classification task:

```
# 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)
```



Logistic Regression Unit

```
# 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)
```

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