**Chapter 10: Introduction to Artificial Neural Network with Keras**

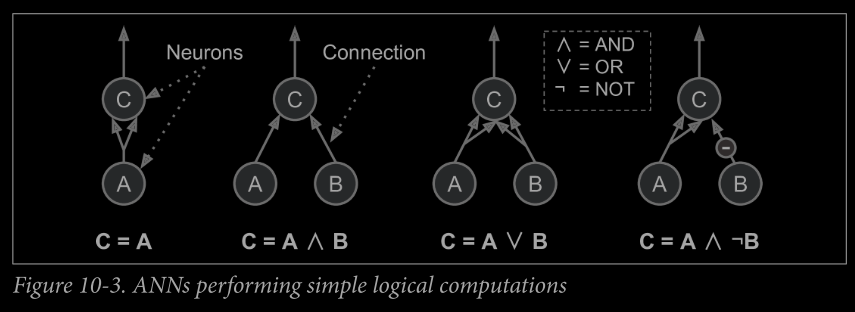
**@** [**https://www.analyticsvidhya.com/blog/2021/05/beginners-guide-to-artificial-neural-network/**](https://www.analyticsvidhya.com/blog/2021/05/beginners-guide-to-artificial-neural-network/)

**@** [**https://www.analyticsvidhya.com/blog/2021/11/artificial-neural-network-and-its-implementation-from-scratch/**](https://www.analyticsvidhya.com/blog/2021/11/artificial-neural-network-and-its-implementation-from-scratch/)

**Logical Computations with Neurons:**

**@** [**https://towardsdatascience.com/an-illustrated-guide-to-artificial-neural-networks-f149a549ba74**](https://towardsdatascience.com/an-illustrated-guide-to-artificial-neural-networks-f149a549ba74)

**Artificial Neuron:** It is a unit similar to a biological neuron, it has one or more binary inputs and one binary output. The artificial neuron simply activates its output when more than a certain no. of its input are active.

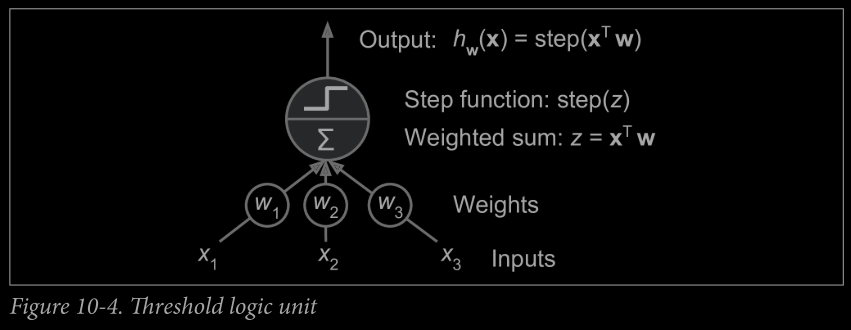


**The Perceptron:** It is based on the different artificial neuron called a **threshold logic unit (TLU)**, or sometimes a linear threshold unit(LTU)

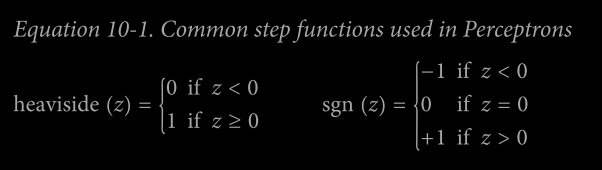
@ <https://medium.com/mlearning-ai/what-is-perceptron-a-beginnerstutorial-for-perceptron-632539884146>

@ <https://medium.com/@jorgesleonel/perceptrons-a22fb29facc4>

The inputs and outputs are now numbers (instead of binary on/off values) and each input connection is associated with a weight. The **TLU** computes a weighted sum of its inputs(z), then applies a step function to that sum outputs the result **hw(x)=step(z), where z= xTw** = w1x1 + w2x2 + w3x3 + . . . . . . . . . . . . . . . . . . wnxn.



# Commonly used step function used in perceptron is **Heaviside Step Function** and sometimes **Sign Function.**

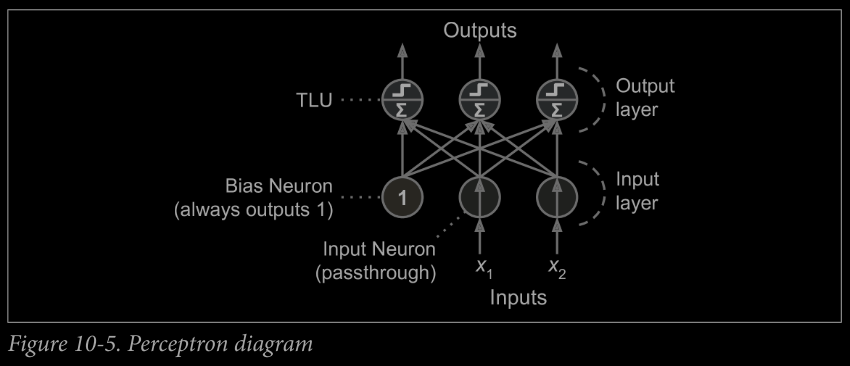


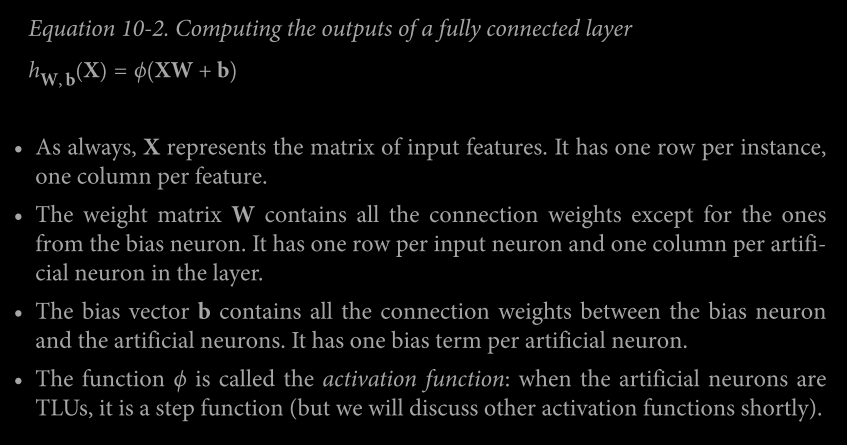
# A single TLU can be used for simple linear binary classification. It computes a linear combination of the inputs and if the result exceeds a threshold, it outputs the positive class or else outputs the negative class (just like a Logistic Regression classifier or a linear SVM).

# Training a TLU means finding appropriate values of w vector.

# **Perceptron is composed of a single layer of TLUs, with each TLU connected to all the inputs**. When all the neurons in a layer are connected to every neuron in the previous layer, it is called **fully connected layer or a dense layer.**

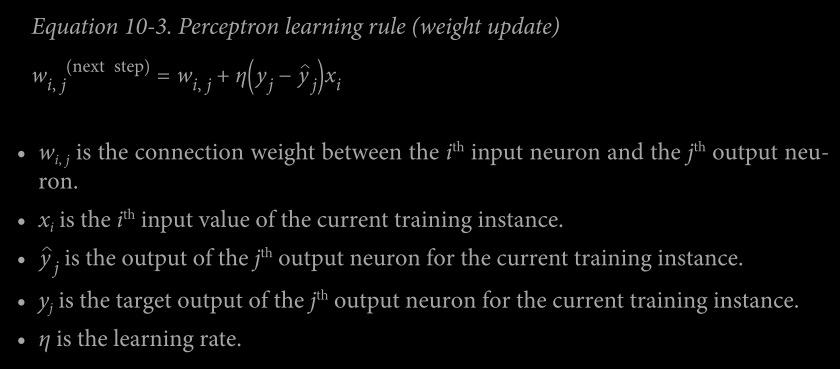
# **Bias Neuron (x0=1):** This neuron gives 1 output all the time and it is added to input layer.





**# Hebb’s Rule (or Hebbian Learning):** The connection weight between two neutrons is increased whenever they have same output.

# **Perceptron are trained** using the variant of this rule that takes into account the error made by the network, it reinforces connections that help reduce the rule.



# **Perceptron Convergence Theorem:** Like Logistic Regression Classifiers, the decision boundary of the perceptron is linear and is incapable of learning complex patterns. If the training instances are linearly separable then this algorithm would converge to a solution.

# **Scikit-Learn’s Perceptron class is equivalent to using an SGDClassifier** with the following hyperparameters: loss="perceptron", learning\_rate="constant", eta0=1 (the learning rate), and penalty=None (no regularization).

# Unlike Logistic Regression Classifier, Perceptron do not output a class probability (they are based on hard threshold) and hence Logistic Regression is more preferred.

**Multi-Layer Perceptron and Backpropagation**

@ <https://medium.com/@jorgesleonel/backpropagation-cc81e9c772fd>

@ <https://medium.com/@tiago.tmleite/neural-networks-multilayer-perceptron-and-the-backpropagation-algorithm-a5cd5b904fde>

@ <https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>

Activation Functions:

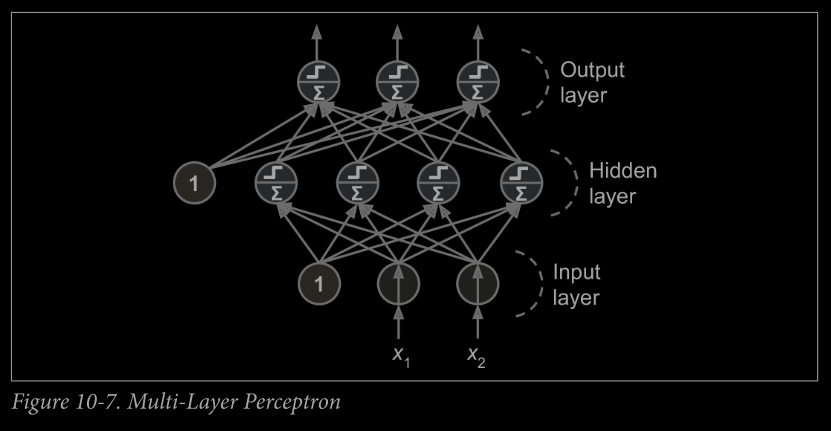
@ <https://medium.com/analytics-vidhya/activation-functions-all-you-need-to-know-355a850d025e>

@ <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

@ <https://prateekvishnu.medium.com/activation-functions-in-neural-networks-bf5c542d5fec>

@ <https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/>

An MLP is composed of one (passthrough) input layer, one or more layers of TLUs, called hidden layers, and one final layer of TLUs called the output layer.

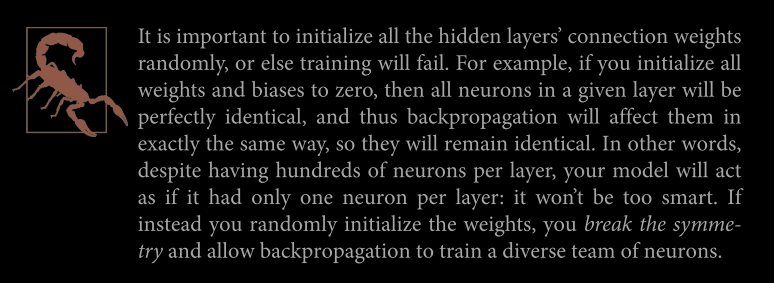


# Feedforward Neutral Network (FNN): The architecture in which the signal flows only in one direction (from inputs to the outputs).

# Automatically computing gradients is called automatic differentiation, or autodiff. There are various autodiff techniques, with different pros and cons. The one used by backpropagation is called reverse-mode autodiff. It

**# Backpropagation Training Algorithm:** This algorithm is based on Gradient Descent and it computes the gradient descent of network’s error with regards to every single parameter.

1. It handles one mini-batch at a time (for example containing 32 instances each), and it goes through the full training set multiple times. Each pass is called an epoch.
2. Each mini-batch is passed to the network’s input layer, which just sends it to the first hidden layer. The algorithm then computes the output of all the neurons in this layer (for every instance in the mini-batch). The result is passed on to the next layer, its output is computed and passed to the next layer, and so on until we get the output of the last layer, the output layer. This is the forward pass: it is exactly like making predictions, except all intermediate results are preserved since they are needed for the backward pass.
3. Next, the algorithm measures the network’s output error (i.e., it uses a loss function that compares the desired output and the actual output of the network, and returns some measure of the error).
4. Then it computes how much each output connection contributed to the error. This is done analytically by simply applying the chain rule (perhaps the most fundamental rule in calculus), which makes this step fast and precise.
5. The algorithm then measures how much of these error contributions came from each connection in the layer below, again using the chain rule—and so on until the algorithm reaches the input layer. As we explained earlier, this reverse pass efficiently measures the error gradient across all the connection weights in the network by propagating the error gradient backward through the network (hence the name of the algorithm).
6. Finally, the algorithm performs a Gradient Descent step to tweak all the connection weights in the network, using the error gradients it just computed.



# The authors made a key **change to the MLP’s architecture: they replaced the step function with the logistic function, σ(z) = 1 / (1 + exp(–z))**. This was essential because the step function contains only flat segments, so there is no gradient to work with (Gradient Descent cannot move on a flat surface), while the logistic function has a well-defined nonzero derivative everywhere, allowing Gradient Descent to make some progress at every step.

# **Other Activation Function:**

**1) Hyperbolic Tangent Function tanh(z) = 2** **σ(2z)-1**

**#** S-shaped graph, continuous, differentiable.

# Range [-1,1]: this makes each layer’s output more or less centered around 0 at the beginning of the training.

**2) Rectified Linear Unit Function: ReLU(z) = max(0,z)**

**#** Continuous but not differentiable at z=0

# Its derivative is zero for z<0; it does not have any maximum output value and helps to reduce some issues during the gradient descent.

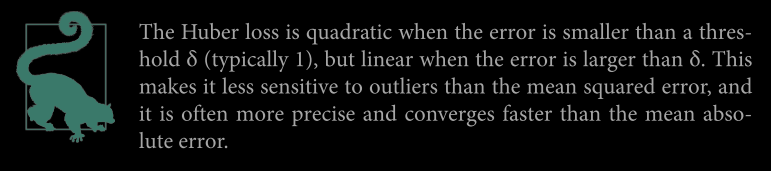
**Regression MLPs**

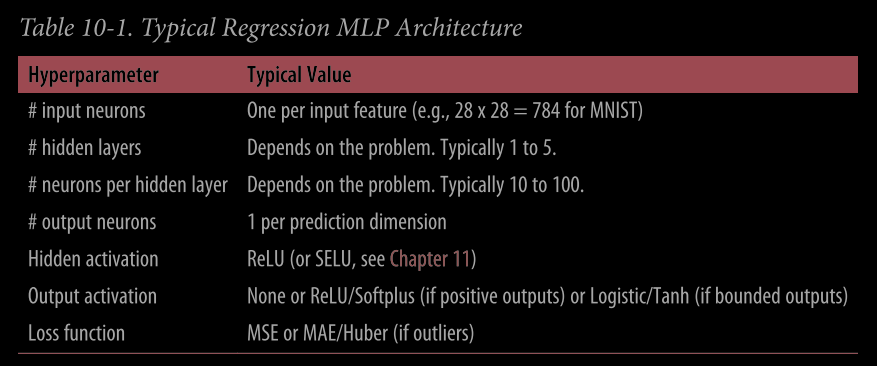
# In general, when building an MLP for regression, no activation function is needed so that the outputs are free to produce any range of values.

# If output will be always possible, then ReLU activation function, or the softplus activation function in the output layer.

# If output must be in a given range of values, then we use logistic function or the hyperbolic function provided that the labels are scaled to appropriate range: 0 to 1 for logistic function, or -1 to 1 for the hyperbolic tangent.

# **Loss function** to use during training is typically the **mean squared error**, but if you have **a lot of outliers in the training set, you may prefer to use the mean absolute error instead**. Alternatively, you can use the **Huber loss, which is a combination of both**.





**Classification MLPs**

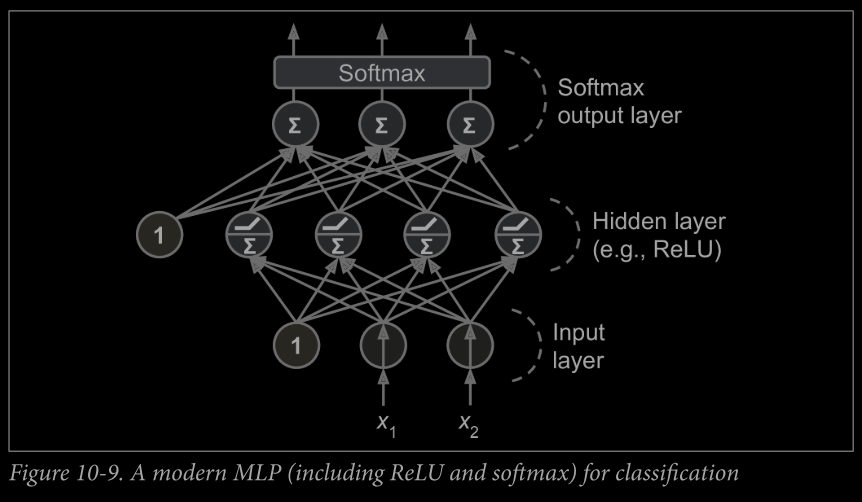
# **For Binary Classification,** only one output neuron is required along with logistic activation function. The output can be 0 or 1, interpreted as the estimated probability of the positive class.

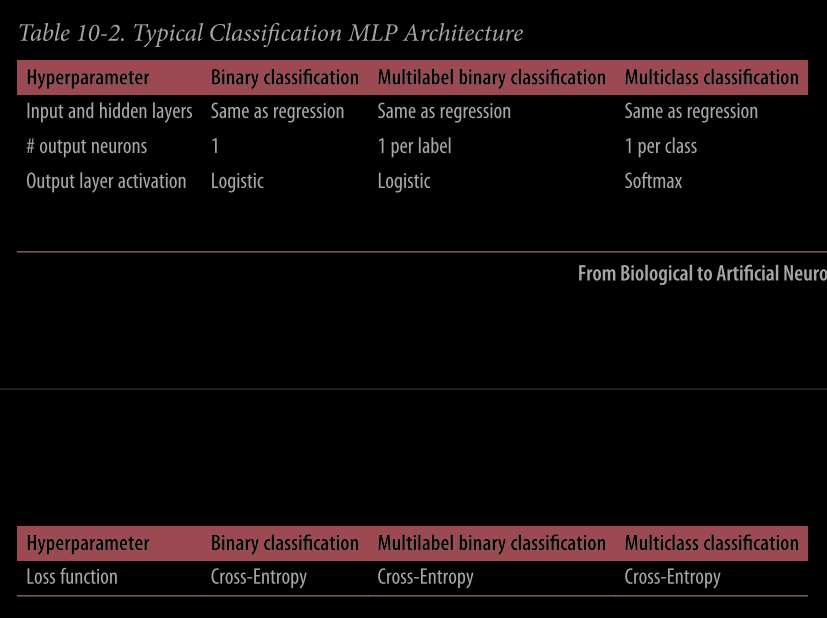
# **Multilabel Classification**

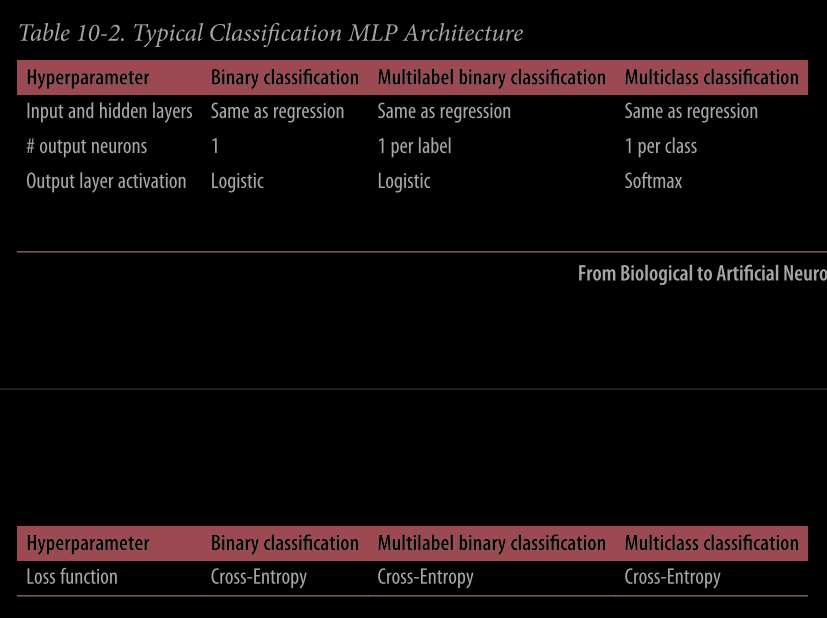
Example 1: an email classification system that predicts whether each incoming email is ham or spam, and simultaneously predicts whether it is an urgent or non-urgent email. In this case, you would need two output neurons, both using the logistic activation function: the first would output the probability that the email is spam and the second would output the probability that it is urgent. More generally, you would dedicate one output neuron for each positive class. Note that the output probabilities do not necessarily add up to one. This lets the model output any combination of labels: you can have non-urgent ham, urgent ham, non-urgent spam, and perhaps even urgent spam (although that would probably be an error).

Example 2: If each instance can belong only to a single class, out of 3 or more possible classes (e.g., classes 0 through 9 for digit image classification), then you need to have one output neuron per class, and you should use the softmax activation function for the whole output layer (see Figure 10-9). The softmax function (introduced in Chapter 4) will ensure that all the estimated probabilities are between 0 and 1 and that they add up to one (which is required if the classes are exclusive).

# **Loss Function: Cross-Entropy (log loss) is general choice as we are predicting probability distribution.**







**Implementing MLPs with Keras: READ FROM BOOK (Pg. 292)**