# Neural Network Classification with TensorFlow

By [Your Name]

A tutorial-style guide for intermediate Python developers building neural networks with TensorFlow.

## Introduction

In this tutorial, we'll build and train a simple neural network to perform classification using TensorFlow and Keras. We'll work through loading data, building the model, compiling it, training, evaluating, and making predictions.

**TYPICAL ARCHITECTURE OF A CLASSIFICATION NEURAL NETWORK**

The word *typical* is on purpose.

Because the architecture of a classification neural network can widely vary depending on the problem you're working on.

However, there are some fundamentals all deep neural networks contain:

* An input layer.
* Some hidden layers.
* An output layer.

Much of the rest is up to the data analyst creating the model.

The following are some standard values you'll often use in your classification neural networks.

| **Hyperparameter** | **Binary Classification** | **Multiclass classification** |
| --- | --- | --- |
| Input layer shape | Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction) | Same as binary classification |
| Hidden layer(s) | Problem specific, minimum = 1, maximum = unlimited | Same as binary classification |
| Neurons per hidden layer | Problem specific, generally 10 to 100 | Same as binary classification |
| Output layer shape | 1 (one class or the other) | 1 per class (e.g. 3 for food, person or dog photo) |
| Hidden activation | Usually [ReLU](https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning) (rectified linear unit) | Same as binary classification |
| Output activation | [Sigmoid](https://en.wikipedia.org/wiki/Sigmoid_function) | [Softmax](https://en.wikipedia.org/wiki/Softmax_function) |
| Loss function | [Cross entropy](https://en.wikipedia.org/wiki/Cross_entropy#Cross-entropy_loss_function_and_logistic_regression) ([tf.keras.losses.BinaryCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy) in TensorFlow) | Cross entropy ([tf.keras.losses.CategoricalCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy) in TensorFlow) |
| Optimizer | [SGD](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD) (stochastic gradient descent), [Adam](https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam) | Same as binary classification |

Table 1: Typical architecture of a classification network. Source: Adapted from page 295 of [Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow Book by Aurélien Géron](https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/)

Don't worry if not much of the above makes sense right now, we'll get plenty of experience as we go through this notebook.

Let's start by importing TensorFlow as the common alias tf. For this notebook, make sure you're using version 2.x+.

## 1. Import Libraries

Let's start by importing the libraries we'll need, including TensorFlow and its high-level Keras API.

**import tensorflow as tf**

**import matplotlib.pyplot as plt**

## 2. Create Data

We'll create synthetic data using `sklearn.datasets.make\_circles`. This will help us visualize how well our neural network can learn non-linear patterns.

**from sklearn.datasets import make\_circles**

**n\_samples = 1000**

**X, y = make\_circles(n\_samples, noise=0.03, random\_state=42)**

## 3. Visualize the Data

Plotting the dataset helps us understand the classification boundary the neural network needs to learn.

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**circles = pd.DataFrame({"X0": X[:, 0], "X1": X[:, 1], "label": y})**

**plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.RdYlBu)**

**plt.xlabel("X0")**

**plt.ylabel("X1")**

**plt.title("Two Circles Classification Dataset")**

**plt.show()**

## 4. Build a Simple Neural Network

We use `tf.keras.Sequential` to stack layers. Our model consists of two hidden layers with ReLU activation and one output layer with sigmoid (binary classification).

**model = tf.keras.Sequential([**

**tf.keras.layers.Dense(4, activation="relu"),**

**tf.keras.layers.Dense(4, activation="relu"),**

**tf.keras.layers.Dense(1, activation="sigmoid")**

**])**

## 5. Compile the Model

Before training, we need to compile the model. We choose binary crossentropy as the loss function, Adam as the optimizer, and accuracy as a metric.

**model.compile(loss=tf.keras.losses.BinaryCrossentropy(),**

**optimizer=tf.keras.optimizers.Adam(),**

**metrics=["accuracy"])**

## 6. Fit the Model

We now train the model on our synthetic dataset. We'll use a validation split to monitor overfitting.

**history = model.fit(X, y, epochs=100, verbose=1)**

## 7. Evaluate the Model

After training, we can evaluate the performance using the final loss and accuracy metrics.

**model.evaluate(X, y)**

## 8. Make Predictions

We can now use our trained model to make predictions on new data. Here, we'll check predictions on a few samples from our dataset.

**preds = model.predict(X[:10])**

**print(preds)**

## 9. Visualize Predictions

To understand how well our model separates the two classes, we can visualize the decision boundary.

**def plot\_decision\_boundary(model, X, y):**

**x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1**

**y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1**

**xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 100),**

**np.linspace(y\_min, y\_max, 100))**

**grid = np.c\_[xx.ravel(), yy.ravel()]**

**preds = model.predict(grid)**

**preds = tf.round(preds)**

**preds = preds.numpy().reshape(xx.shape)**

**plt.contourf(xx, yy, preds, cmap=plt.cm.RdYlBu, alpha=0.7)**

**plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.RdYlBu)**

**plt.title("Decision Boundary")**

**plt.show()**

**plot\_decision\_boundary(model, X, y)**

## Conclusion

In this tutorial, we successfully built and trained a neural network for binary classification. You learned how to create synthetic data, visualize it, define a model architecture, compile, train, and evaluate it, and finally visualize the decision boundary. This is a great first step into using neural networks for classification tasks.