Neural Network Regression with TensorFlow

By [Your Name]

## A tutorial-style guide for intermediate Python developers building regression models using neural networks in TensorFlow.

There are many definitions for a [regression problem](https://en.wikipedia.org/wiki/Regression_analysis) but in our case, we're going to simplify it to be: predicting a number.

For example, you might want to:

* Predict the selling price of houses given information about them (such as number of rooms, size, number of bathrooms).
* Predict the coordinates of a bounding box of an item in an image.
* Predict the cost of medical insurance for an individual given their demographics (age, sex, gender, race).

In this notebook, we're going to set the foundations for how you can take a sample of inputs (this is your data), build a neural network to discover patterns in those inputs and then make a prediction (in the form of a number) based on those inputs.

What we're going to cover

Specifically, we're going to go through doing the following with TensorFlow:

* Architecture of a regression model
* Input shapes and output shapes
  + X: features/data (inputs)
  + y: labels (outputs)
* Creating custom data to view and fit
* Steps in modelling
  + Creating a model
  + Compiling a model
    - Defining a loss function
    - Setting up an optimizer
    - Creating evaluation metrics
  + Fitting a model (getting it to find patterns in our data)
* Evaluating a model
  + Visualizng the model ("visualize, visualize, visualize")
  + Looking at training curves
  + Compare predictions to ground truth (using our evaluation metrics)
* Saving a model (so we can use it later)
* Loading a model

Don't worry if none of these make sense now, we're going to go through each.

How you can use this notebook

You can read through the descriptions and the code (it should all run), but there's a better option.

Write all of the code yourself.

Yes. I'm serious. Create a new notebook, and rewrite each line by yourself. Investigate it, see if you can break it, why does it break?

You don't have to write the text descriptions but writing the code yourself is a great way to get hands-on experience.

Don't worry if you make mistakes, we all do. The way to get better and make less mistakes is to **write more code**.

**Typical Architecture Of A Regresison Neural Network**

The word *typical* is on purpose.

Why?

Because there are many different ways (actually, there's almost an infinite number of ways) to write neural networks.

But the following is a generic setup for ingesting a collection of numbers, finding patterns in them and then outputting some kind of target number.

Yes, the previous sentence is vague but we'll see this in action shortly.

| **Hyperparameter** | **Typical value** |
| --- | --- |
| Input layer shape | Same shape as number of features (e.g. 3 for # bedrooms, # bathrooms, # car spaces in housing price prediction) |
| Hidden layer(s) | Problem specific, minimum = 1, maximum = unlimited |
| Neurons per hidden layer | Problem specific, generally 10 to 100 |
| Output layer shape | Same shape as desired prediction shape (e.g. 1 for house price) |
| Hidden activation | Usually [ReLU](https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning) (rectified linear unit) |
| Output activation | None, ReLU, logistic/tanh |
| Loss function | [MSE](https://en.wikipedia.org/wiki/Mean_squared_error) (mean square error) or [MAE](https://en.wikipedia.org/wiki/Mean_absolute_error) (mean absolute error)/Huber (combination of MAE/MSE) if outliers |
| 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) |

*Table 1: Typical architecture of a regression network. Source: Adapted from page 293 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/)

Again, if you're new to neural networks and deep learning in general, much of the above table won't make sense. But don't worry, we'll be getting hands-on with all of it soon.

**Note:** A **hyperparameter** in machine learning is something a data analyst or developer can set themselves, where as a **parameter** usually describes something a model learns on its own (a value not explicitly set by an analyst).

## Introduction

## In this tutorial, we’ll build a neural network to perform regression using TensorFlow and Keras. Regression problems involve predicting continuous values (like price, temperature, etc.). We’ll create synthetic data, build a model, train it, and visualize predictions.

## 1. Import Libraries

We begin by importing the necessary libraries: TensorFlow for modeling and matplotlib for visualization..

**import tensorflow as tf**

**import matplotlib.pyplot as plt**

## 2. Create Data

We create synthetic data using a linear relationship with added noise. This mimics real-world regression problems.

**X = tf.range(-100, 100, 4)**

**y = X + 10**

3. Split the Data

We divide the data into training and testing sets. This allows us to evaluate how well our model generalizes to unseen data.

**X\_train = X[:40]**

**y\_train = y[:40]**

**X\_test = X[40:]**

**y\_test = y[40:]**

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

**plt.show()**

4. Visualize the Data

It’s good practice to visualize the data before modeling. Here, we use a scatter plot to observe the relationship.

**plt.figure(figsize=(10, 6))**

**plt.scatter(X\_train, y\_train, c='b', label='Training data')**

**plt.scatter(X\_test, y\_test, c='g', label='Testing data')**

**plt.legend()**

**plt.show()**

**Steps in modelling with TensorFlow**

Now we know what data we have as well as the input and output shapes, let's see how we'd build a neural network to model it.

In TensorFlow, there are typically 3 fundamental steps to creating and training a model.

1. **Creating a model** - piece together the layers of a neural network yourself (using the [Functional](https://www.tensorflow.org/guide/keras/functional) or [Sequential API](https://www.tensorflow.org/api_docs/python/tf/keras/Sequential)) or import a previously built model (known as transfer learning).
2. **Compiling a model** - defining how a models performance should be measured (loss/metrics) as well as defining how it should improve (optimizer).
3. **Fitting a model** - letting the model try to find patterns in the data (how does X get to y).

Steps in modelling with TensorFlow

Now we know what data we have as well as the input and output shapes, let's see how we'd build a neural network to model it.

In TensorFlow, there are typically 3 fundamental steps to creating and training a model.

1. **Creating a model** - piece together the layers of a neural network yourself (using the [Functional](https://www.tensorflow.org/guide/keras/functional) or [Sequential API](https://www.tensorflow.org/api_docs/python/tf/keras/Sequential)) or import a previously built model (known as transfer learning).
2. **Compiling a model** - defining how a models performance should be measured (loss/metrics) as well as defining how it should improve (optimizer).
3. **Fitting a model** - letting the model try to find patterns in the data (how does X get to y).

5. Build the Model

We use the Keras Sequential API to define a neural network model. This model contains two hidden layers with ReLU activation and one output layer for regression

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

**tf.keras.layers.Dense(10, input\_shape=[1], activation='relu'),**

**tf.keras.layers.Dense(10, activation='relu'),**

**tf.keras.layers.Dense(1)**

**])**

6. Compile the Model

Before training, we compile the model with a loss function and optimizer. We use Mean Absolute Error (MAE) as our loss metric and Adam as the optimizer.

**model.compile(loss='mae',**

**optimizer=tf.keras.optimizers.Adam(learning\_rate=0.01))**

7. Train the Model

We now train the model on the training data. We run for a specified number of epochs to allow the model to learn the data patterns.

**model.fit(X\_train, y\_train, epochs=100)**

8. Visualize 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.

**y\_pred = model.predict(X\_test)**

**plt.figure(figsize=(10, 6))**

**plt.scatter(X\_train, y\_train, c='b', label='Training data')**

**plt.scatter(X\_test, y\_test, c='g', label='Testing data')**

**plt.scatter(X\_test, y\_pred, c='r', label='Predictions')**

**plt.legend()**

**plt.show()**

## 9. Evaluate the Model

We evaluate the model performance by comparing predicted and actual values using the loss function.

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

## Conclusion

You’ve now built a basic neural network for regression using TensorFlow. This model learned to predict continuous values from synthetic data. In real-world projects, you'd use larger datasets, regularization, and model tuning for better accuracy