Machine Learning with TensorFlow

Rpoject: https://samsclass.info/129S/proj/ML100.htm

Project Analysis: ML 100 - Machine Learning with TensorFlow

This project is designed to introduce **basic machine learning concepts using TensorFlow** in Python, with hands-on practice through **Google Colab**. It starts from simple linear models and progresses to more complex neural network architectures for curve fitting and performance optimization.

1 🔑 Key Learning Objectives

- **Understanding Linear Relationships**
- **Building Neural Networks with TensorFlow**
- **Training Models with Data (Supervised Learning)**
- **Evaluating Model Performance (Loss Functions)**
- **Experimenting with Hyperparameters (Units, Layers, Noise)**

📊 Breakdown of Each Section

✅ ML 100.0: Introduction to TensorFlow (Basic Linear Model)

- **Code:**

```
import tensorflow as tf
print(tf.__version__)
```

- **What You Learn:**

- Setting up TensorFlow in Google Colab.
- Confirming TensorFlow installation and version.

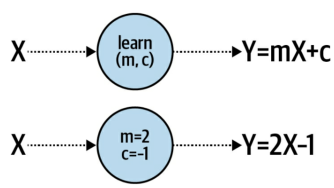


Figure 2-4. A single neuron learning a linear relationship

✅ ML 100.1: Learning a Linear Relationship

- **Mathematical Relationship:**

- **Complete Code to get flag:**

```python
import tensorflow as tf
import numpy as np
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([Dense(units=1, input_shape=[1])])
model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.1, -0.95, 1.07, 3.03, 4.91, 6.98], dtype=float)

model.fit(xs, ys, epochs=500)
print(model.predict(np.array([10.0])))
```

- **Result:**

```
Epoch 497/500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - loss: 0.0044
Epoch 498/500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 0.0044
Epoch 499/500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - loss: 0.0044
Epoch 500/500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - loss: 0.0044
```

- **What You Learn:**

- **Creating a simple neural network** with one input and one output node.

- **Stochastic Gradient Descent (SGD):** Optimizer that updates model weights.

- **Mean Squared Error (MSE):** Loss function to measure the difference between predicted and actual values.

• **Model Prediction:** Making predictions after training.

• **Key Concept:**

The model adjusts weights to minimize the error, learning the linear relationship between X and Y.

✅ **ML 100.2: Adding Noise to Data (Learning with Errors)**

• **Objective:**

Test the model's ability to handle **imperfect, noisy data**.

• **Updated Code:**

```
ys = np.array([-3.1, -0.95, 1.07, 3.03, 4.91, 6.98], dtype=float)
```

• **Result:**

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 4.3178
Epoch 498/500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - loss: 4.3178
Epoch 499/500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - loss: 4.3178
Epoch 500/500
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step - loss: 4.3178
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
[[-12.290493]]
```

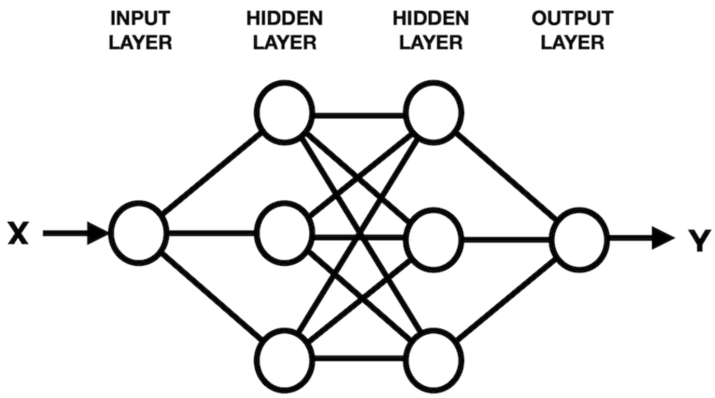
• **What You Learn:**

• **Real-world data is noisy**; models must generalize despite imperfections.

• The model still learns the **underlying trend** even if the data isn't perfect.

• The model failed to get 55 as the correct answer.

✅ ML 100.3: Fitting a Complex Curve



- **Complex Function with Noise:**

```
y_data = 0.1 * x_data * np.cos(x_data) + 0.1 * np.random.normal(size=1000)
```

- **Neural Network Architecture:**

```
model.add(Dense(1, activation='linear', input_shape=[1]))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='linear'))
```

- **Result:**

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 1)	2
dense_9 (Dense)	(None, 64)	128
dense_10 (Dense)	(None, 64)	4,160
dense_11 (Dense)	(None, 1)	65

Total params: 4,355 (17.01 KB)
Trainable params: 4,355 (17.01 KB)
Non-trainable params: 0 (0.00 B)

- **What You Learn:**

- **Using multiple hidden layers** and **ReLU activations** to capture complex patterns.

• **Overfitting vs. Generalization:** More layers can overfit, but also learn complex functions.

• **Visualization:** Plotting the model's predictions vs. the noisy data to see how well it fits.

✅ ML 100.4: Reducing Model Complexity

• **Change:**

Comment out one hidden layer:

```
# model.add(Dense(64, activation='relu'))
```

• **Result:**

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 1)	2
dense_13 (Dense)	(None, 64)	128
dense_14 (Dense)	(None, 1)	65

• **What You Learn:**

• **Model Simplification:** Removing layers reduces complexity but may still perform well.

• The model's performance depends on the **data complexity**, not just the architecture.

✅ ML 100.5: Hyperparameter Tuning (Units & Layers)

• **Experiments:**

• 32 units per layer, 1 hidden layer

• 16 units per layer, 2 hidden layers

• 4 units per layer, 3 hidden layers

• 4 units per layer, 4 hidden layers

• **What You Learn:**

• **Hyperparameter Tuning:** Adjusting units and layers to find the best model.

• **Loss Evaluation:** Tracking the loss function to compare model performance.

• **Bias-Variance Trade-off:** Finding the balance between underfitting and overfitting.

✅ ML 100.6: Data Variability (Impact of Input Size & Noise)

• **Experiments with Data Size & Noise:**

• 100 points, noise = 0.1

• 1000 points, noise = 0.5

• 300 points, noise = 0.01

• **What You Learn:**

• **Impact of Noise:** High noise levels make it harder for the model to learn patterns.

• **Data Quantity:** More data helps with generalization, but diminishing returns may occur.

• **Robustness:** Testing the model's performance under different conditions.

🚀 Core Machine Learning Concepts Covered

1. **Supervised Learning:**

Training models on labeled data (X and Y pairs).

2. **Neural Network Basics:**

Using **Dense layers** to learn linear and non-linear relationships.

3. **Model Training:**

• **Epochs:** Number of training iterations.

• **Loss Function (MSE):** Measures prediction error.

• **Optimizer (SGD/Adam):** Adjusts weights to reduce error.

4. **Overfitting vs. Underfitting:**

Experimenting with model complexity (units, layers, data size).

5. **Noise Tolerance:**

Testing models on noisy data to see how well they generalize.

6. **Hyperparameter Tuning:**

Adjusting layers, units, and data parameters for optimal performance.

📈 Skills You Gain

• **Practical TensorFlow Knowledge:**

Building models from scratch using TensorFlow and Keras.

• **Data Preprocessing & Visualization:**

Using NumPy and Matplotlib to handle data and visualize results.

• **Model Evaluation:**

Interpreting loss values, predictions, and model weights.

• **Problem-Solving:**

Adjusting model architecture to fit different types of data.