DL-Cheat-Codes (/github/nikitaprasad21/DL-Cheat-Codes/tree/main)
/ ANN-Models (/github/nikitaprasad21/DL-Cheat-Codes/tree/main/ANN-Models)

Dropout

Dropout is a regularization technique commonly used in classification tasks with neural networks, but it can also be adapted for regression tasks.

In regression, dropout is applied to the hidden layers of the neural network to prevent overfitting and improve the generalization performance of the model.

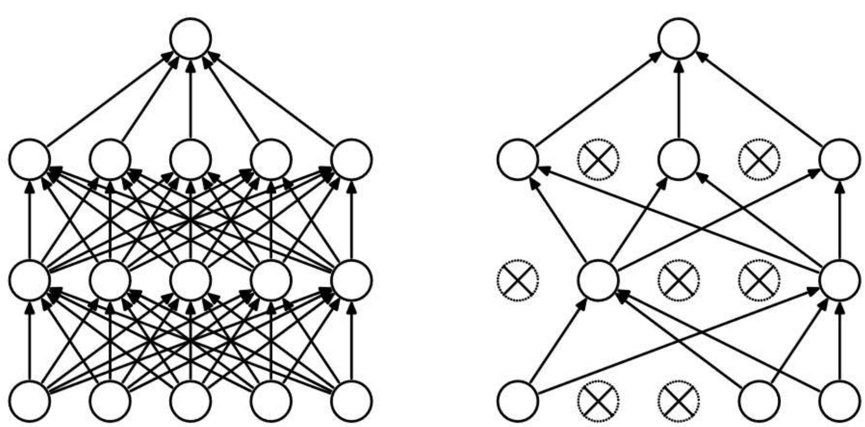
How you can apply dropout in regression using a neural network?

- **Model Architecture**: Define a neural network architecture suitable for regression. This typically includes an input layer, one or more hidden layers, and an output layer with a single neuron (since regression predicts a continuous value).

- **Dropout Layer**: Add dropout layers to the hidden layers of the neural network. Dropout layers randomly set a fraction of the input units to zero during training, which helps prevent overfitting by reducing co-adaptation of neurons.

- **Training**: Train the neural network with dropout enabled during training. Dropout is applied only during training, not during inference (prediction).

- **Prediction**: During prediction (inference), disable dropout to allow all units to contribute to the prediction.



(a) Standard Neural Net (b) After applying dropout.

Let's implement this!

In [2]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pyt
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all fil

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preser
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside
```

In [3]:
```
from sklearn.datasets import make_regression
```

In [70]:
```
input_,target_ = make_regression(n_samples= 100, n_features=1, n_targets=1, noise=20)
```

In [48]:
```
from sklearn.model_selection import train_test_split
```

In [71]:
```
train_input, test_input, train_target, test_target = train_test_split(input_, target_,
```

In [68]:
```
import matplotlib.pyplot as plt
```

In [72]:
```
# Plot the training data
plt.scatter(train_input, train_target, color='blue', label='Training Data')

# Plot the testing data
plt.scatter(test_input, test_target, color='red', label='Testing Data')

# Add labels and title
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Training and Testing Data')
plt.legend()

# Show the plot
plt.show()
```

Regression Model

In [18]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import mean_squared_error
```

```
2024-04-19 07:39:12.087122: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:
9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDN
N when one has already been registered
2024-04-19 07:39:12.087214: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:
607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT
when one has already been registered
2024-04-19 07:39:12.089056: E external/local_xla/xla/stream_executor/cuda/cuda_blas.c
c:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin c
uBLAS when one has already been registered
```

In [73]:
```
model_1 = Sequential()

model_1.add(Dense(128, activation = "relu", input_dim = 1))
model_1.add(Dense(128, activation="relu"))
model_1.add(Dense(1, activation="linear"))
adam = Adam(learning_rate=0.01)
model_1.compile(loss='mse', optimizer=adam, metrics=['mse'])
history = model_1.fit(train_input, train_target, epochs=500,
                    validation_data = (test_input, test_target))
```

```
Epoch 495/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 403.3142 - mse: 416.1519 - val_loss: 31
8.6132 - val_mse: 318.6132
Epoch 496/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 406.1562 - mse: 400.5760 - val_loss: 31
4.8477 - val_mse: 314.8477
Epoch 497/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 331.2830 - mse: 336.0630 - val_loss: 31
7.9986 - val_mse: 317.9986
Epoch 498/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 369.5007 - mse: 363.7740 - val_loss: 28
0.4391 - val_mse: 280.4391
Epoch 499/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 381.0677 - mse: 367.2274 - val_loss: 27
3.1143 - val_mse: 273.1143
Epoch 500/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 381.8665 - mse: 394.3039 - val_loss: 31
0.1675 - val_mse: 310.1675
```

Evaluate the model

In [74]:
```
_, train_mse = model_1.evaluate(train_input, train_target, verbose=0)
_, test_mse = model_1.evaluate(test_input, test_target, verbose=0)
print(f'Train Loss: {train_mse}, Test Loss: {test_mse}')
```

```
Train Loss: 381.5201721191406, Test Loss: 310.16754150390625
```

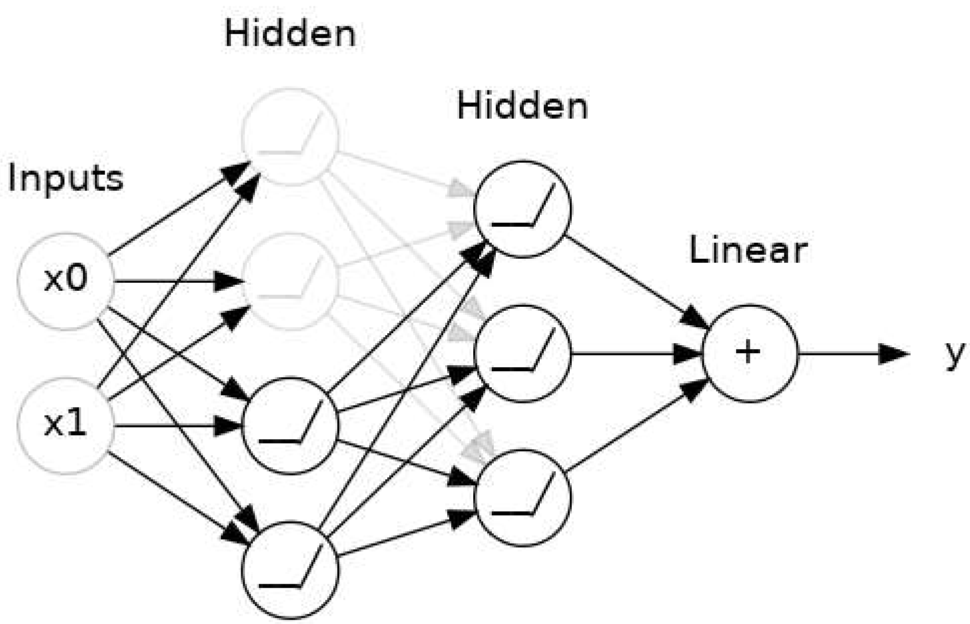
Prediction

In [75]:
```
y_pred_1 = model_1.predict(test_input)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
```

In [76]:
```
# Plot the training data
plt.scatter(train_input, train_target, color='blue', label='Training Data')

# Plot the testing data
plt.scatter(test_input, test_target, color='red', label='Testing Data')

plt.plot(test_input, y_pred_1)

# Add labels and title
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Training and Testing Data')
plt.legend()

# Show the plot
plt.show()
```

Dropout Model



In [77]:
```
model_2 = Sequential()

model_2.add(Dense(128, activation = "relu", input_dim = 1))
model_2.add(Dropout(0.2))

model_2.add(Dense(128, activation="relu"))
model_2.add(Dropout(0.2))

model_2.add(Dense(1, activation="linear"))
adam = Adam(learning_rate=0.01)
model_2.compile(loss='mse', optimizer=adam, metrics=['mse'])
history = model_2.fit(train_input, train_target, epochs=500,
                    validation_data = (test_input, test_target))
```

```
Epoch 495/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 419.7179 - mse: 430.2668 - val_loss: 31
2.1452 - val_mse: 312.1452
Epoch 496/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 546.6140 - mse: 554.1671 - val_loss: 31
2.8166 - val_mse: 312.8166
Epoch 497/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 430.4329 - mse: 378.1682 - val_loss: 35
6.7010 - val_mse: 356.7010
Epoch 498/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 426.0599 - mse: 440.0181 - val_loss: 30
9.0195 - val_mse: 309.0195
Epoch 499/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 468.7289 - mse: 471.6624 - val_loss: 31
7.1669 - val_mse: 317.1669
Epoch 500/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 435.6664 - mse: 431.6718 - val_loss: 35
6.4373 - val_mse: 356.4373
```

In [78]:

```
_, train_mse = model_2.evaluate(train_input, train_target, verbose=0)
_, test_mse = model_2.evaluate(test_input, test_target, verbose=0)
print(f'Train Loss: {train_mse}, Test Loss: {test_mse}')
```

```
Train Loss: 369.89385986328125, Test Loss: 356.4372863769531
```

In [79]:

```
y_pred_2 = model_2.predict(test_input)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
```

In [80]:

```
# Plot the training data
plt.scatter(train_input, train_target, color='blue', label='Training Data')

# Plot the testing data
plt.scatter(test_input, test_target, color='red', label='Testing Data')

plt.plot(test_input, y_pred_2)

# Add labels and title
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Training and Testing Data')
plt.legend()

# Show the plot
plt.show()
```

Dropout Model 2

In [81]:
```
model_3 = Sequential()

model_3.add(Dense(128, activation = "relu", input_dim = 1))
model_3.add(Dropout(0.5))

model_3.add(Dense(128, activation="relu"))
model_3.add(Dropout(0.5))

model_3.add(Dense(1, activation="linear"))
adam = Adam(learning_rate=0.01)
model_3.compile(loss='mse', optimizer=adam, metrics=['mse'])
history = model_3.fit(train_input, train_target, epochs=500,
                    validation_data = (test_input, test_target))
```

```
Epoch 495/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 886.4557 - mse: 917.0006 - val_loss: 27
8.6650 - val_mse: 278.6650
Epoch 496/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 542.0009 - mse: 547.2210 - val_loss: 31
0.6877 - val_mse: 310.6877
Epoch 497/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 711.6757 - mse: 718.6219 - val_loss: 32
1.6295 - val_mse: 321.6295
Epoch 498/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - loss: 521.7783 - mse: 534.4952 - val_loss: 30
6.3682 - val_mse: 306.3682
Epoch 499/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 483.9997 - mse: 487.3920 - val_loss: 28
1.5263 - val_mse: 281.5263
Epoch 500/500
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 621.3098 - mse: 634.3007 - val_loss: 29
4.1079 - val_mse: 294.1079
```

In [82]:

```
_, train_mse = model_3.evaluate(train_input, train_target, verbose=0)
_, test_mse = model_3.evaluate(test_input, test_target, verbose=0)
print(f'Train Loss: {train_mse}, Test Loss: {test_mse}')
```

```
Train Loss: 393.39910888671875, Test Loss: 294.1079406738281
```

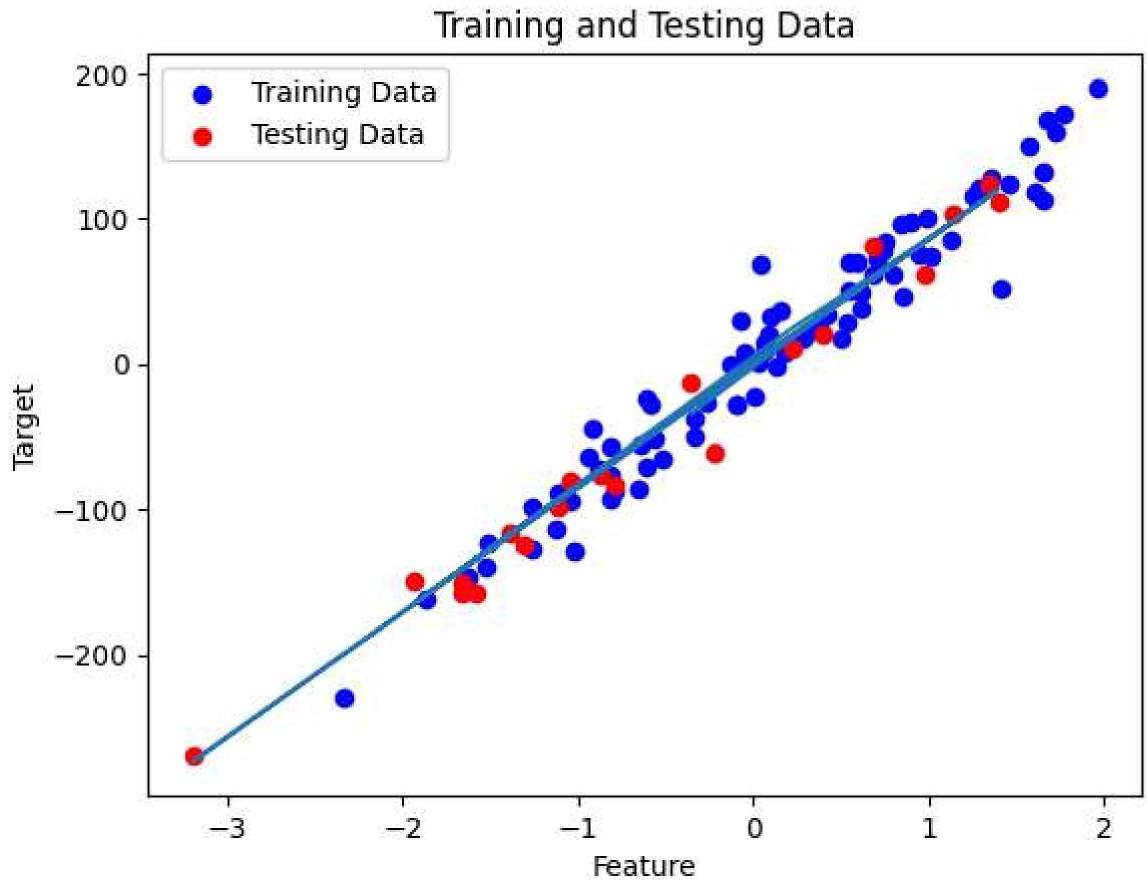
In [83]:

```
y_pred_3 = model_3.predict(test_input)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step
```

In [84]:

```
# Plot the training data
plt.scatter(train_input, train_target, color='blue', label='Training Data')

# Plot the testing data
plt.scatter(test_input, test_target, color='red', label='Testing Data')

plt.plot(test_input, y_pred_3)

# Add labels and title
plt.xlabel('Feature')
plt.ylabel('Target')
plt.title('Training and Testing Data')
plt.legend()

# Show the plot
plt.show()
```



After applying dropout regularization to the neural network for regression, we observed the following results:

- **Train Loss**: The train loss, as measured by the mean squared error, was found to be 393.40.
- **Test Loss**: The test loss, also measured by the mean squared error, was found to be 294.11.

Conclusion

These results suggest that the dropout regularization technique helped in reducing overfitting and improving the generalization performance of the neural network. The test loss is lower than the train loss, indicating that the model has learned to generalize well to unseen data.

> Stay tuned for more and Don't forget to **Star** this Github Repository for more such contents.

```
In [ ]:
```