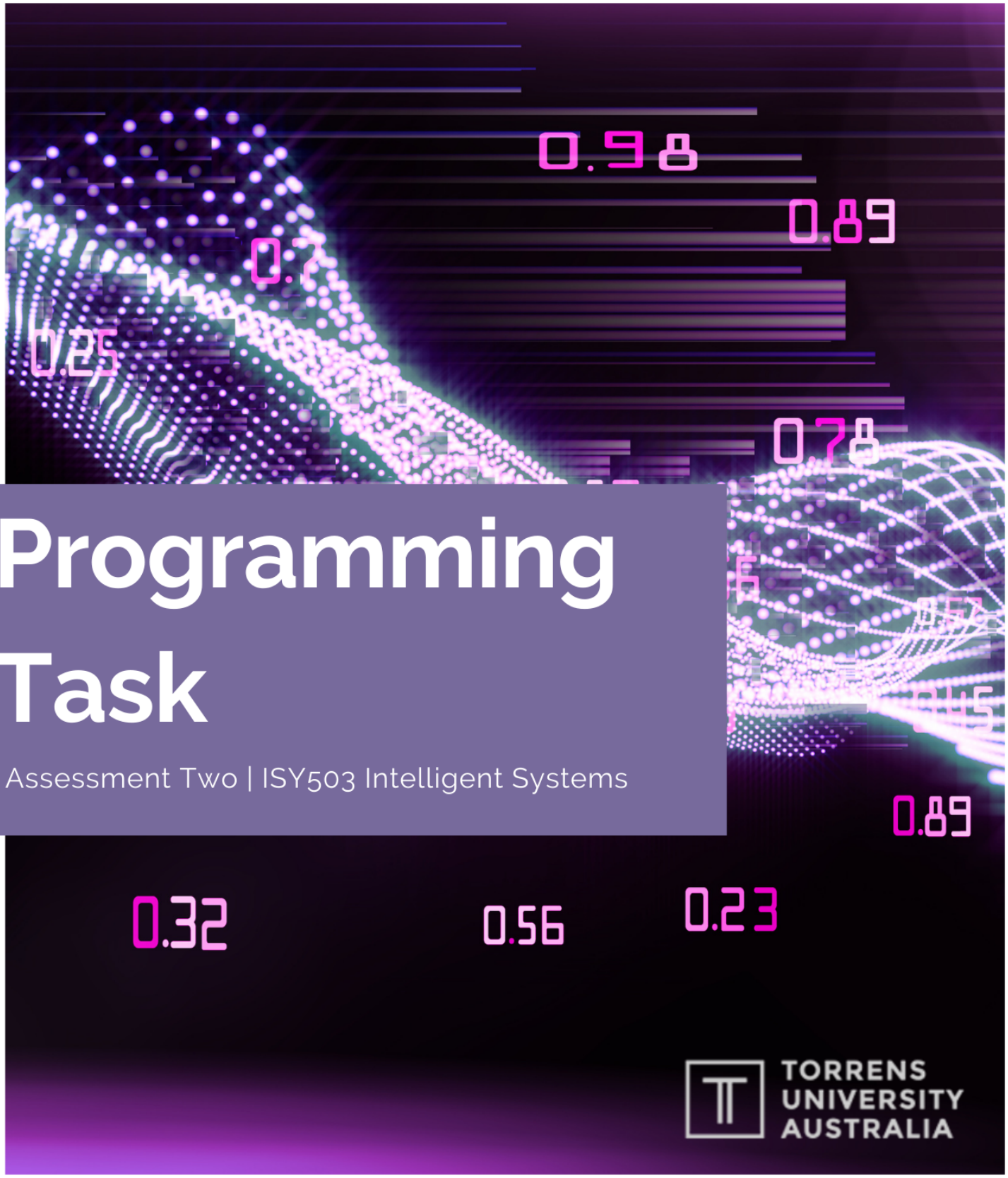



# Programming Task

Assessment Two | ISY503 Intelligent Systems

## Intro to modelling with Machine learning

Master of Software Engineering - Advanced Artificial Intelligence
July | 2021 | Australia

**Yolanda Carreno Gomez | ID: A00032641**

## Abstract

The present manual summarises programming tasks from Google focused on machine learning techniques; it is an introduction to modelling using ML, linear regression hyperparameters and normalisation techniques.

Hyperparameters are the variables that define the network structure, for instance the number of hidden units and the variables that determine how the network is trained, for example the Learning Rate. Hyperparameters are crucial to train ML models to obtain better AI or neural network results. Thus, for the purpose of this task, I found that there are four main hyperparameters: Batch size, num_epochs, hidden layer, and learning rate.

## Task 0: Use pandas to explore and prepare the data

The data is divided in two categories as listed below:

### numeric_feature_names

```
[ ] # Run to inspect categorical features.
    car_data[categorical_feature_names]
```

| | make | fuel-type | aspiration | num-doors | body-style | drive-wheels | engine-location | engine-type | num-cylinders | fuel-system |
|---|---|---|---|---|---|---|---|---|---|---|
| **183** | volkswagen | gas | std | two | sedan | fwd | front | ohc | four | mpfi |
| **154** | toyota | gas | std | four | wagon | 4wd | front | ohc | four | 2bbl |
| **116** | peugot | diesel | turbo | four | sedan | rwd | front | l | four | idi |
| **161** | toyota | gas | std | four | hatchback | fwd | front | ohc | four | 2bbl |
| **63** | mazda | diesel | std | ? | sedan | fwd | front | ohc | four | idi |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **55** | mazda | gas | std | two | hatchback | rwd | front | rotor | two | 4bbl |
| **80** | mitsubishi | gas | turbo | two | hatchback | fwd | front | ohc | four | spdi |
| **76** | mitsubishi | gas | std | two | hatchback | fwd | front | ohc | four | 2bbl |
| **33** | honda | gas | std | two | hatchback | fwd | front | ohc | four | 1bbl |
| **60** | mazda | gas | std | four | sedan | fwd | front | ohc | four | 2bbl |

205 rows × 10 columns

### categorical_feature_names

```
[ ] # Run to inspect numeric features.
    car_data[numeric_feature_names]
```

| | symboling | wheel-base | length | width | height | weight | engine-size | bore | stroke | compression-ratio | horsepower | peak-rpm | city-mpg | highway-mpg | price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **183** | 2 | 97.30 | 171.70 | 65.50 | 55.70 | 2209 | 109 | 3.19 | 3.40 | 9.00 | 85 | 5250 | 27 | 34 | 7975 |
| **154** | 0 | 95.70 | 169.70 | 63.60 | 59.10 | 2290 | 92 | 3.05 | 3.03 | 9.00 | 62 | 4800 | 27 | 32 | 7898 |
| **116** | 0 | 107.90 | 186.70 | 68.40 | 56.70 | 3252 | 152 | 3.70 | 3.52 | 21.00 | 95 | 4150 | 28 | 33 | 17950 |
| **161** | 0 | 95.70 | 166.30 | 64.40 | 52.80 | 2122 | 98 | 3.19 | 3.03 | 9.00 | 70 | 4800 | 28 | 34 | 8358 |
| **63** | 0 | 98.80 | 177.80 | 66.50 | 55.50 | 2443 | 122 | 3.39 | 3.39 | 22.70 | 64 | 4650 | 36 | 42 | 10795 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **55** | 3 | 95.30 | 169.00 | 65.70 | 49.60 | 2380 | 70 | ? | ? | 9.40 | 101 | 6000 | 17 | 23 | 10945 |
| **80** | 3 | 96.30 | 173.00 | 65.40 | 49.40 | 2370 | 110 | 3.17 | 3.46 | 7.50 | 116 | 5500 | 23 | 30 | 9959 |
| **76** | 2 | 93.70 | 157.30 | 64.40 | 50.80 | 1918 | 92 | 2.97 | 3.23 | 9.40 | 68 | 5500 | 37 | 41 | 5389 |
| **33** | 1 | 93.70 | 150.00 | 64.00 | 52.60 | 1940 | 92 | 2.91 | 3.41 | 9.20 | 76 | 6000 | 30 | 34 | 6529 |
| **60** | 0 | 98.80 | 177.80 | 66.50 | 55.50 | 2410 | 122 | 3.39 | 3.39 | 8.60 | 84 | 4800 | 26 | 32 | 8495 |

205 rows × 15 columns

## Task 1: Make your best model with numeric features. No normalization allowed.

Modify the model provided below to achieve the lowest eval loss. You may want to change various hyperparameters

- learning rate
- choice of optimizer
- hidden layer dimensions -- make sure your choice here makes sense given the number of training examples
- batch size
- num training steps
- (anything else you can think of changing)
- Do not use the `normalizer_fn` arg on `numeric_column`.

In this task, I used deep neural network which has more layers than MLP and the normal neural network as shown in the figure 1

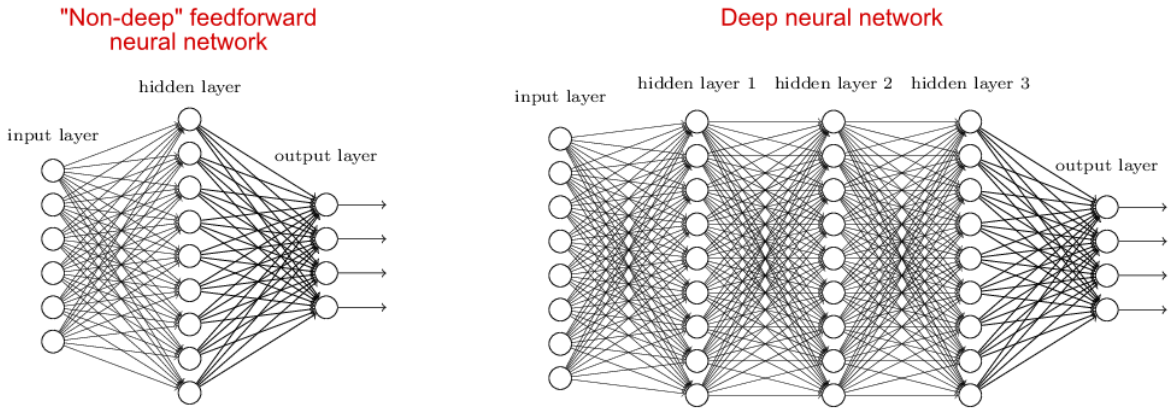



Figure 1. Deep neural network

The goal is to provide better hyperparameters to obtain lowest eva (evaluation process) loss (error rate). This model task trains, evaluates and then predicts.

To solve this task, I have changed the gradient descent for Adagrad Optimizer, the reason is because when I run the operation, the data overlaps, and the type of data I am using is failing. This task only uses the numeric features.

Adagrad is an algorithm for gradient-based optimization which adapts the learning rate to the parameters, excecuting low learning rates for parameters

associated with often occurring features, and greater updates like high learning rates for parameters related to infrequent features, (Ruder, 2016).

In other words, during the neural network training, its weights are arbitrarily initialised originally, subsequently they are updated in each epoch to improve the network accuracy. At each epoch, the training data output compares to actual data through the loss function to compute the error and then the weight is updated accordingly, the **optimizer** can be represented as bellow.

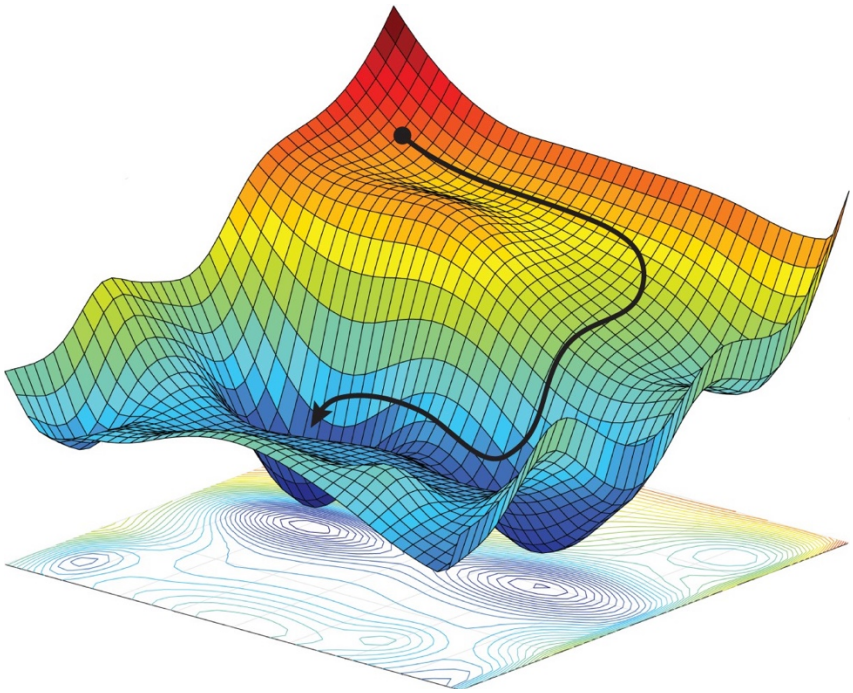


Figure 2. Optimizer representation

After implementing different gradient descent optimization algorithms, I found that Adagrad gets the best result compared to Adam, Adadelta, Momentum, and AMSGrad.

The changes are as bellow:

**From:**

```
est = tf.estimator.DNNRegressor(
    feature_columns=model_feature_columns,
    hidden_units=[64],
```

```
    optimizer=tf.train.GradientDescentOptimizer(learning_rate=0.01),
  )
```

**To:**

```
est = tf.estimator.DNNRegressor(
    feature_columns=model_feature_columns,
    hidden_units=[128],
    optimizer=tf.train.AdagradOptimizer(learning_rate=0.03),
  )
```

The normalization has not been used and hyperparameters have been changed

**From:**

```
16  - batch size
None – num_epochs
64 - hidden layer
0.01 -  learning rate
```

Original result:

```
scores {'average_loss': 202.7198, 'label/mean': 12949.43, 'loss':
3196.7354, 'prediction/mean': 12949.551, 'global_step': 10000}
```

**To:**

```
16  - batch size
6000 – num_epochs
128 - hidden layer
0.03 -  learning rate
```

My best solution:

```
scores {'average_loss': 30.111895, 'label/mean': 12949.43, 'loss':
30.111895, 'prediction/mean': 12948.418, 'global_step': 10000}
```

By defining DNNregressor with the above hyperparameters with 10000 times of training, I reduced the error rate from 3,196.7354 to 30.111895  and dropped

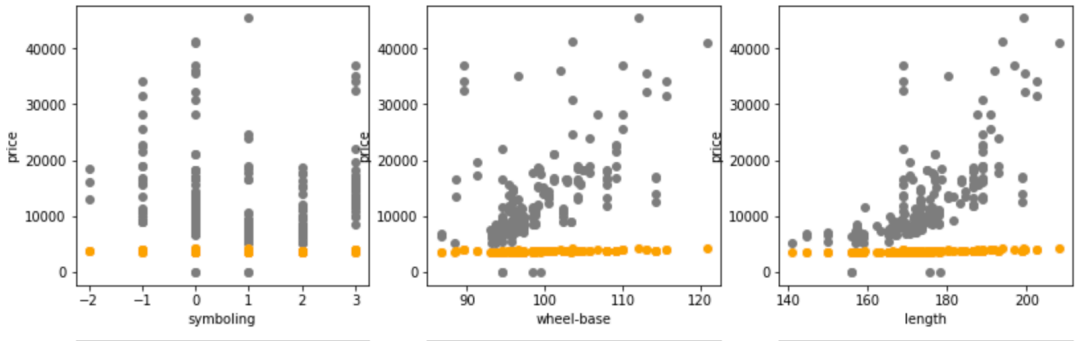
the average loss from 202.7198 to 30.111895. That was the best solution I could get.

## Task 2: Take your best numeric model from earlier. Add normalization.

To solve this task model, I used mean normalization function and the following hyperparameters:

```
16  - batch size
6000 – num_epochs
128 - hidden layer
0.03 -  learning rate
```

My result:




```
scores {'average_loss': 159447700.0, 'label/mean': 12949.43, 'loss':
2514367500.0,  'prediction/mean':  3077.7976,  'global_step':  10000}
```

Original result:

```
16  - batch size
None – num_epochs
64 -hidden layer
0.01 -  learning ratescores
```

```
scores {'average_loss': 148274740.0, 'label/mean': 12949.43, 'loss':
2338178600.0, 'prediction/mean': 3716.531, 'global_step': 10000}
```

**Does normalization improve model quality on this dataset? Why or why not?**

The normalization did not improve the model because I wanted to predict the real values, I was not getting any closer to what I expected. A good normalization should show you the closest to the predicted one, the normalization should get the data/cluster together to have less errors.

The main goal of the normalization is to reduce the standard deviation, the less spread the data is, the better. But that was not the case with this task model.

## Task 3: Make your best model using only categorical features

To solve this task, I used the same hyperparameters than the previous tasks.
Example Task 3

Original hyperparameters and result:

```
16  - batch size
None – num_epochs
64 -hidden layer
0.01 -  learning ratescores

scores {'average_loss': 158473800.0, 'label/mean': 12949.43, 'loss':
2499009800.0, 'prediction/mean': 3323.4685, 'global_step': 10000}
```

My hyperparameters and result:

```
16  - batch size
6000 – num_epochs
128 -hidden layer
0.03 -  learning rate

scores {'average_loss': 33032070.0, 'label/mean': 12949.43, 'loss':
520890340.0, 'prediction/mean': 12602.933, 'global_step': 10000}
```

By defining DNNregressor with the above hyperparameters with 10,000 times of training, I reduced dramatically the error rate from 2,499,009,800.0 to 520,890,340.0 and dropped efficiently the average loss from 158,473,800.0 to 33,032,070.0. That was the best solution I could get.

## Task 4: Using all categorical and numerical the features, make the best model that you can make

To solve this task I did two tests; one with mean normalization and another one without normalization.

My hyperparameters

```
16  - batch size
60000 – num_epochs
128 -hidden layer
0.03 -  learning rate
```

Results:

With mean normalization

```
scores {'average_loss': 21282280.0, 'label/mean': 12949.43, 'loss':
335605200.0, 'prediction/mean': 12944.759, 'global_step': 10000}
```

without normalization

```
scores {'average_loss': 18.920738, 'label/mean': 12949.43, 'loss':
298.3655, 'prediction/mean': 12948.648, 'global_step': 10000}
```

Normalization didn't get better results with the functions I have; it doesn't make any difference by using both categorical and numerical features and using normalization because the errors are so high.

In summary, this assignment helped me to understand the use of ML modelling techniques, I found that normalization was not possible to use in all the models because the clusters did not get better when training, on the other hand, changing the optimisation functions and the hyperparameters helped a lot to improve the results of this modelling tasks.

# REFERENCES

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. Retrieved from https://ruder.io/optimizing-gradient-descent/index.html#adagrad