**Name:** Volia Palisse

**Student ID number:** 20242911

# Q1. Linear Regression

#### Task 1

1. This line is a for loop that repeats for a specified number of iterations (tot\_iter). It determines the number of times the model parameters (weights and bias) will be updated according to gradient descent.
2. Calculate the predicted values (pred) by applying the weights (w) and bias (b) to the input data (x) at the current stage of iteration. The equation used for determining the predicted value is an hypothesis function corresponding to a linear regression model (linear equation).
3. Computes the Mean Square Error (mse) which corresponds to the average of the squared differences between predicted values (pred) and actual values (y). It acts as a loss function to measure the accuracy of the model.
4. It then computes the gradient of the loss function for the weights (w). This gradient (dw) corresponds to the average value of the product of the input data (x) and the derivative of the loss function (2 \* (pred - y)). It’s used to determine how much the weights need to be adjusted to minimize the previously computed loss.
5. This line works similarly but with regards to the bias. The gradient allows for determining the direction and magnitude at which weight and bias needs to be updated to minimize loss.
6. The weights are then updated by subtracting the product of the learning rate (lr) and their gradient (dw). This way the weights move in the direction that best minimizes the loss.
7. Same as the previous line but to update the bias. Those two steps allow for the ‘learning’ part of the algorithm.
8. //
9. When all the iterations are done, the program exits the for loop and computes the final prediction with the final learned weight and bias.
10. The final loss (Mean Square Error) is calculated.
11. We print the error.
12. And we print the final weight and bias.

#### Task 2

The general loss function for linear regression is as follows :

L_{mse} = \frac{1}{n} \sum_{i=1}^{n} (y_i - pred_i)^2 

%fb046b5b-2b5d-4241-8805-4698c674065c

with :

pred_i = w x_i + b 

%f1645231-fd81-4a5d-a9e5-071423bd756d

Thus :

L_{mse} = \frac{1}{n} \sum_{i=1}^{n} (y_i - (w x_i + b))^2 

%5c9730ec-3548-4ae7-93ad-19979fd7b85e

We are trying to find the partial derivatives regarding weight and bias, let’s start with weight :

L_{mse} = \frac{1}{n} \sum_{i=1}^{n} (y_i - w x_i - b)^2 

%d6b90791-d470-4b57-9d56-78f8fddf7402

\frac{\partial L_{mse}}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w} (y_i - w x_i - b)^2 

%12501b7f-da34-4a29-8cae-534daf165642

\frac{\partial L_{mse}}{\partial w} = -\frac{1}{n} \sum_{i=1}^{n} 2 x_i (y_i - w x_i - b)
%d0ecbe7f-df66-470b-99bd-c578a3772309

\frac{\partial L_{mse}}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} 2 x_i (w x_i + b - y_i) 

%1ba3ce45-e5c1-4e69-8753-eb154ecf4137

\frac{\partial L_{mse}}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} 2 x_i (pred_i - y_i) 

%9d546aca-d815-4a27-a375-69011a8e39de

As for the bias :

L_{mse} = \frac{1}{n} \sum_{i=1}^{n} (y_i - w x_i - b)^2
%8f484f6b-1dc6-43a2-8c20-6b7b09b7f0f1

\frac{\partial L_{mse}}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial b} (y_i - w x_i - b)^2 

%b73d7599-cc9d-459b-8e72-09f0de6bc1c8

\frac{\partial L_{mse}}{\partial b} = -\frac{1}{n} \sum_{i=1}^{n} 2 (y_i - w x_i - b) 

%1402d687-d682-406f-b8eb-ca18f3870fd6

\frac{\partial L_{mse}}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} 2 ((w x_i + b) - y_i) 

%6642e77c-2b25-4297-98c5-d8ae2bea2157

\frac{\partial L_{mse}}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (pred_i - y_i) 

%a64e248d-6bfa-4e1f-96fb-02e787869d9c

We have now obtained the partial derivatives corresponding to lines 4 and 5 of the code :

\frac{\partial L_{mse}}{\partial w} = \frac{1}{n} \sum_{i=1}^{n} 2 x_i (pred_i - y_i) 

%9d546aca-d815-4a27-a375-69011a8e39de

\frac{\partial L_{mse}}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (pred_i - y_i) 

%a64e248d-6bfa-4e1f-96fb-02e787869d9c

#### Task 3 : Your answer should be submitted with .ipynb

# Q2. Lasso Regression

#### Task 1

1. This line is a for loop that repeats for a specified number of iterations (tot\_iter). It determines the number of times the model parameters (weights and bias) will be updated according to gradient descent.
2. This line calculates the predicted values (pred) based on the current weights and biases. Where (x) is the input data matrix, (w) is the weight vector, and (b) is the bias vector. The @ symbol is used to multiply matrices.
3. Computes the Mean Square Error (mse) which corresponds to the average of the squared differences between predicted values (pred) and actual values (y). When using lasso regression we add (alpha \* np.mean(np.abs(w))) to the MSE to penalize large weights and promote sparsity. The MSE acts as a loss function to measure the accuracy of the model.
4. //
5. It then computes the gradient of the loss function for the weights (w). The term (np.dot(x.T, (2 \* (pred - y))) / x.shape[0]) computes the gradient of the MSE while (alpha \* np.sign(w)) is the gradient of the lasso regularization. (np.sign(w)) returns the sign of each corresponding (w) to ensure the L1 penalty is well applied. It’s used to determine how much the weights need to be adjusted to minimize the previously computed loss.
6. We then compute the loss regarding the bias (b) in a similar way we did for simple linear regression.
7. //
8. The weights are then updated by subtracting the product of the learning rate (lr) and their gradient (dw). This way the weights move in the direction that best minimizes the loss.
9. Same as the previous line but to update the bias. Those two steps allow for the ‘learning’ part of the algorithm.
10. //
11. The final prediction is calculated.
12. The final loss/error is calculated
13. We print the error.
14. And we print the final weight and bias.

#### Task 2

Simple linear regression and lasso regression both use MSE as a way to measure deviation between predicted and actual values. However lasso regression introduces an L1 penalty term proportional to the absolute value of the weights. This produces multiple benefits.

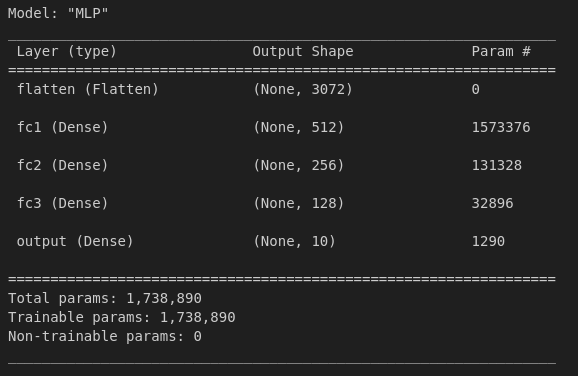
When using simple regression weights are determined by how much they minimize the error only, which can lead to very large weights, introducing the L1 penalty term causes the coefficients to trend towards zero. This helps with overfitting and allows the model to work better with new data. The trending towards zero also means some coefficients will actually be null, excluding features that might be irrelevant from the model which makes it sparser. Increasing the value of ⍺ will drive coefficients towards zero more effectively but a god equilibrium has to be determined.

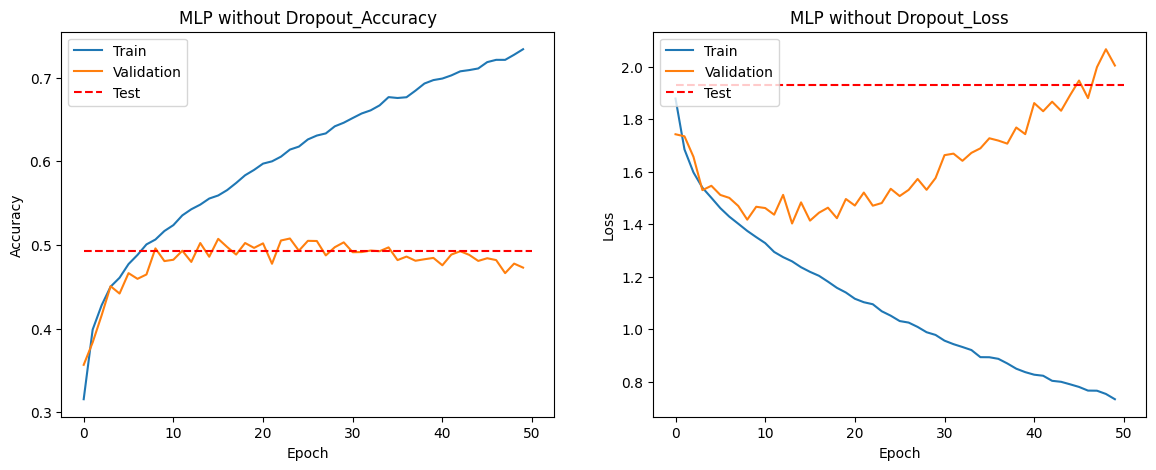
To conclude, lasso regression allows for reduced overfitting and simple models.

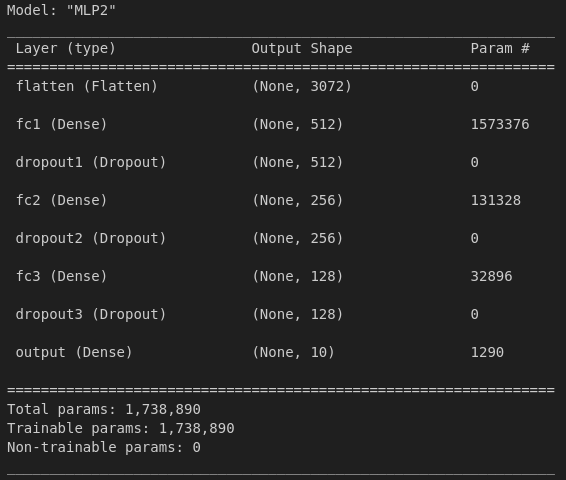
#### Task 3 : Your answer should be submitted with .ipynb

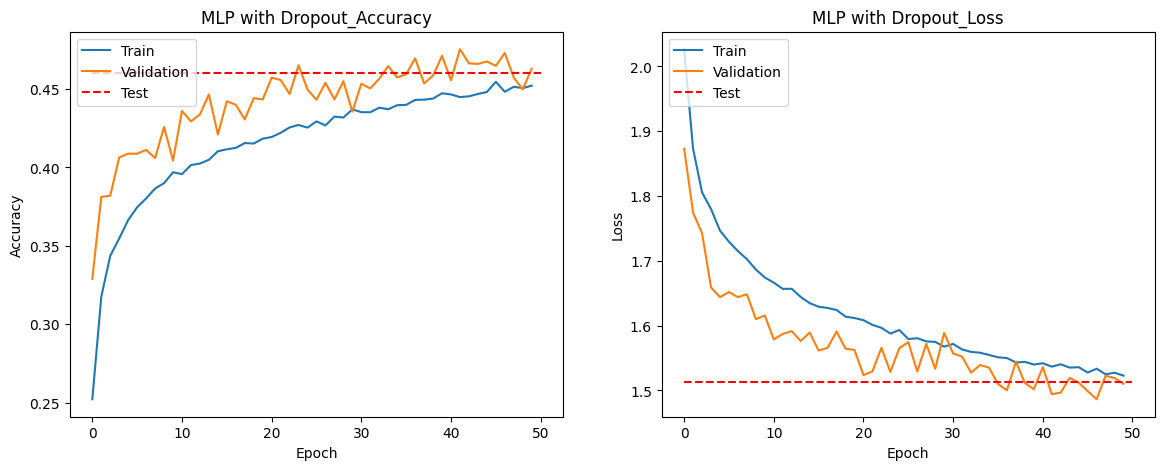
# Q3. MLPs

#### Output images from Task 1~4









#### Task 5

Dropout layers are used in neural networks to prevent overfitting. Overfitting occurs when a model learns the training data too well, including noise and outliers, resulting in poor performance on unseen data (validation/test data). Dropout helps mitigate this by randomly setting to zero a fraction of neurons during the training process, meaning that these neurons are temporarily removed from the network along with their connections. Because neurons are randomly dropped, the neural network is forced to learn more robust features which are less prone to be overfitted. The model is then more capable of generalization outside of training.

1. The training accuracy of model one is better than model two but they tend to be improving for each epoch in both cases. However the test and validation data of both models act differently, for the first one, the accuracy of the validation set plateaus after roughly ten epochs while for the second the validation accuracy curve follows the training accuracy curve. This behavior indicates that, while the first model became way better in training, it was very overfitted and struggled with generalization. The second model with dropout however did not suffer from overfitting, but got its accuracy in training reduced by the dropout layers. In the end both models got roughly similar accuracy after fifty epochs, however the first model was overfitted and wouldn’t have improved with more epochs, while the second could have.
2. The loss in training for both models seemed to be declining, though the model without dropout seemed better, achieving lower loss at the fifty epochs mark. However when looking at the validation curve, the first model loss started by declining and then quickly raised while the second had a validation loss very similar to its training loss. The first model seems to have adapted too much to the training set and lost its ability to generalize to outside data, which didn’t happen for the second model using dropout. Thus while the first model seemed better while training, the second model ended up having a lower loss than the first one after fifty epochs..

# Q4. CNNs

#### 2.1.2. Discussion on “modified\_cnn\_1”

The difference between base and modified are as follows :

* Base has two convolutional layers while modified has four.
* Base has two MaxPooling layers, one for each convolutional layer while modified doesn’t have any.
* The number of filters and the size of the kernel of the convolutional layers also vary between the two models.

The modified model has more convolutional layers which means it can learn more complex details and features which makes it potentially better for complex datasets. The larger Kernel size makes it able to capture more spatial information, however it also makes it more likely to be overfitted, the additional size also makes it more complex to run. Meanwhile the base model has pooling layers which reduces spatial dimensions, making the model less complex. However pooling layers tend to lose some information, the modified model has no pooling layers which makes it capable of capturing finer details but the higher dimension count it implies makes it more prone to overfitting. The modified model ends up having more parameters than the base one, making it again more flexible but also more likely to be overfitted.

The two models usefulness can vary depending on the dataset, for less complex images, the base model is better, but for complex ones the modified might be relevant, provided it isn't overfitted which is likely.

#### 2.2.2. Discussion on “modified\_cnn\_2”

The differences between base and modified 2 are as follows:

* Base has two convolutional layers while modified 2 has nine.
* Base model has two MaxPooling layers, one for each convolutional layer, while modified 2 has three, one every three convolutional layers.
* The number of filters and the size of the kernel of the convolutional layers also vary between the two models.

The modified model, with its increased number of convolutional layers, can learn more complex details and features, making it potentially better suited for complex datasets. The use of 1x1 kernels allows combination of information between RGB channels without changing spatial dimensions, though it may limit the model's ability to capture spatial relationships effectively. It’s also more prone to being overfitted due to its high number of parameters.

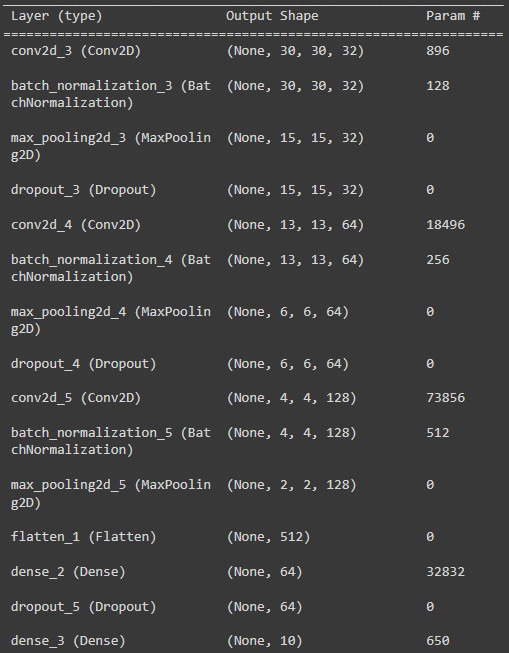
On the other hand, the larger kernels in the base model (9x9 and 3x3) are better at capturing spatial information. The MaxPooling layers in the base model help reduce spatial dimensions, making the model less complex and computationally demanding.

#### 2.3.2. Discussion on “modified\_cnn\_3”

The only difference between the two models is the addition of a dropout layer between the last two dense layers. Adding a dropout layer will help in preventing overfitting by randomly setting to zero a fraction of neurons during the training process, meaning they can’t be used. This forces the neural network to learn more new features, making it more likely to be able to adapt to data outside of its training dataset.

#### 3.2. Analyze your “my\_cnn”

My first idea was to add some complexity to the model, for this reason I added an additional convolutional layer followed by a MaxPooling layer, I then added batch normalization between each convolutional and pooling layers. However, upon testing my model It ended up being highly overfitted. For this reason I added multiple dropout layers interlocked between the others. My model ended up looking as follows :



The results were also way more satisfying, the dropout layers clearly helped reduce overfitting to a minimum, as confirmed by the graph. The final results on the test set were roughly 78% which I deemed satisfying. However, using dropout layers means the accuracy of the model might vary a lot when trained. Some training runs got as low as 72% accuracy.

# Q5. Augmentations

You don’t have to write a report for Q5. Instead, make sure your code works fine in submitted .ipynb.