CS 577

HW #3

Khalid Saifullah

ID : A20423546
Semester : Spring 2021
Date : 25th March 2021

Theoretical Questions

Loss

Answer to the Question No. 1

L_1 : $L_i(\theta) = \sum_{j=1}^{k} \left| \hat{y}_j^{(i)} - y_j^{(i)} \right|$

L_2 : $L_i(\theta) = \sum_{j=1}^{k} \left(\hat{y}_j^{(i)} - y_j^{(i)} \right)^2$

Huber Loss : $L_i(\theta) = \sum_{j=1}^{k} \rho_\delta \left[\underbrace{\hat{y}^{(i)} - y^{(i)}}_{d} \right]$ where,

$$\rho_\delta(d) = \begin{cases} \frac{1}{2} d^2, & \text{if } |d| \le \delta \\ \delta(d - \delta/2), & \text{otherwise} \end{cases}$$

Log-Cosh-Loss : $L_i(\theta) = \sum_{j=1}^{k} \log\left[\cosh\left(\hat{y}^{(i)} - y^{(i)}\right)\right]$ where,

$$\log[\cosh(d)] \approx \begin{cases} d^2/2, & \text{if } |d| \text{ is small} \\ |d| - \log(2), & \text{otherwise} \end{cases}$$

→ L_1 and L_2 are sensitive to outliers.

→ Huber and log-cosh-loss functions are robust estimators that compensate for the outliers.

Answer to the Question No. 2

Predicted value, $\hat{y}_j^{(i)} = P(y = j \mid x^{(i)})$

→ Likelihood : (maximize) $L(\theta) = \prod_{i=1}^{m} \prod_{j=1}^{k} P(y = j \mid x^{(i)})^{y_j^{(i)}}$

→ Log-likelihood : (minimize) $L(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{k} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$

→ Sample loss for cross entropy,

$$L_i(\theta) = -\sum_{j=1}^{k} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$$

For random class assignment, worst loss value would be, $L(\theta) = k$

whereas, $P(y = j \mid x^{(i)}) = \frac{1}{k}$

Answer to the Question No. 3

Softmax loss, $L(\theta) = -\sum_{i=1}^{m}\sum_{j=1}^{k} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$

→ It is used in multi-class classification (on the output node).

Answer to the Question No. 4

Kullback-liebler loss : It measures similarity between distributions.

$$L(\theta) = -\sum_{i=1}^{n}\sum_{j=1}^{k} y_j^{(i)} \log\left(\frac{y_j^{(i)}}{\hat{y}_j^{(i)}}\right)$$

It is essentially the difference between entropy and cross-entropy. It is the same as cross-entropy when class labels don't change.

Answer to the Question No. 5

Hinge loss, $L_i(\theta) = \sum_{j=1}^{k} \max\left(0, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1\right)$

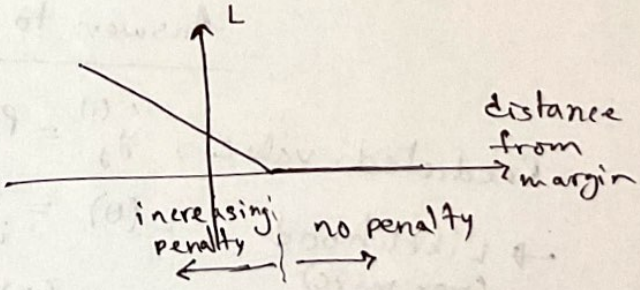
Squared hinge loss, $L_i(\theta) = \frac{1}{2}\sum_{j=1}^{k} \max\left(0, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1\right)^2$

These losses only penalize the misclassified samples and those close to the margins of the decision boundary.

Worst Value

Hinge loss = k

Squared hinge loss = $\frac{k}{2}$



Answer to the Question No. 6

$L_1(\theta) = \max(0, 0.5 - 0.5 + 1) + \max(0, 0.4 - 0.5 + 1) + \max(0, 0.3 - 0.5 + 1) = 2.7$

$L_2(\theta) = \max(0, 1.3 - 0.8 + 1) + \max(0, 0.8 - 0.8 + 1) + \max(0, -0.6 - 0.8 + 1) = 2.5$

$L_3(\theta) = \max(0, 1.4 - 2.7 + 1) + \max(0, -0.4 - 2.7 + 1) + \max(0, 2.7 - 2.7 + 1) = 1$

Answer to the Question No. 7

The objective of regularization is to obtain a simpler, more stable model that may perform better on unseen data by limiting the potential to overfit on the training data.

→ L_1 regularisation tends to make the weights sparser

→ L_2 regularisation tends to distribute the weights across the coefficients.

→ Since regularisation adds more hyperparameters, they will need to be fine tuned in order to obtain the best results.

Answer to the Question No. 8

Inspecting derivatives wrt the predicted outputs.

$$L_1 : \frac{L_i(\theta)}{\partial \hat{y}_j^{(i)}} = \begin{cases} -1 & \text{if } \hat{y}_j^{(i)} < y_j^{(i)} \\ 1 & \text{if } \hat{y}_j^{(i)} > y_j^{(i)} \\ 0 & \text{if } \hat{y}_j^{(i)} = y_j^{(i)} \end{cases}$$

$$L_2 : \frac{L_i(\theta)}{\partial \hat{y}_j^{(i)}} = 2 \left[\hat{y}_j^{(i)} - y_j^{(i)} \right] \hat{y}_j^{(i)}$$

$\therefore$ L_1 binarises the gradients whereas L_2 generates smoother derivatives.

Answer to the Question No. 9

Kernel regularization → applied to coefficients

Bias regularization → applied to bias term

Activity regularization → applied to the output of layers

Optimization

Answer to the Question No. 1

Backpropagation is the concatenation of simple chain rule, and allows for a fast scalable implementation of the gradient descent algorithm for the deep models used in deep learning. Direct numerical computation can be used for verification purpose.

Answer to the Question No. 2

Gradient descent (GD) updates the weights taking the whole dataset to compute the gradients, whereas stochastic gradient descent (SGD) updates weights after every example and more accurate Although gradient descent ~~gentr~~ produces better gradients, it is slower in updating the weights compared to SGD.

Answer to the Question No. 3

The batch size defines the tradeoff between faster weight updates and noisier gradients.

There are four main problems with SGD:

1) Specification of learning rate.
2) Some parameters may be more sensitive to loss than others (i.e. too fast in some directions and too slow in others).
3) Converging to a local minimum.
4) Mini batch gradient estimates are noisy.

Answer to the Question No. 4

poor conditioning → smoothed out by averaging with previous gradients

local minimum / saddle points → there is a ~~vecity~~ velocity vector even at those conditions and SGD with momentum utilizes that property

noisy gradients → smoothed out by moving average

Answer to the Question No. 5

NAG corrects the corrects for the location ~~at~~ of gradient at a future step and hence it gives better accuracy than the simple momentum.

Its derivation (or algorithm) produces a new term that evaluates the difference of velocity terms between iterations, which is in other words is the acceleration. Thus, the term "accelerated" is used.

Answer to the Question No. 6

Strategies for learning rate decay are:

1) step decay $\Rightarrow \eta \leftarrow \eta/2$ every iteration
2) Exponential decay $\Rightarrow \eta \leftarrow \eta_0 e^{-kt}$ every iteration
3) Fractional decay $\Rightarrow \eta \leftarrow \frac{\eta_0}{1+kt}$ every iteration

where k is the decay rate and t is the iteration index.

Answer to the Question No. 7

Newton's method attempts to ~~estimate the opti~~ compute the optimum learning rate (instead of specifying it) using Newton's decomposition ~~using~~ approximation which requires the derivative of a function at a specific point.

In order to compute the gradient, the hessian matrix has to be inverted, which is computationally expensive.

~~Answer to the Question No. 8~~

$$H = \begin{bmatrix} \frac{\partial J^2}{\partial \theta_1 \partial \theta_1} & \cdots & \frac{\partial J^2}{\partial \theta_1 \partial \theta_n} \\ \vdots & & \\ \frac{\partial J^2}{\partial \theta_n \partial \theta_1} & \cdots & \frac{\partial J^2}{\partial \theta_n \partial \theta_n} \end{bmatrix}$$

H ← 2nd order derivative matrix

Answer to the Question No. 8

Condition number generally defines the sensitivity using the singular value of the Hessian matrix.

$$\text{Condition number} = \frac{sv_1}{sv_m} \quad \begin{matrix} \leftarrow \text{largest sv} \\ \leftarrow \text{smallest sv} \end{matrix}$$

When the condition number is high, matrix is particularly sensitive to error in input which makes it harder to optimize

Answer to the Question No. 9

Adagrad approximates hessian matrix with a diagonal matrix by the sum of the vector product of the computed gradient and its transpose. This method, however, does not account for the variation of parameters, but the diagonal matrix is computationally inexpensive to invert.

$$B^{(i)} = \text{diag}\left(\sum_{j=1}^{i} \nabla J(\theta^{(i)}) \nabla J(\theta^{(j)})^T \right)^{1/2}$$

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta B^{-1} \nabla J(\theta)$$

Answer to the Question No. 10

~~Add~~ AdaGrad adds all the gradients from the start of the learning process, thus progressively reducing the contribution of newer gradients, because we normalise by elementwise sum of square gradients. ~~RMSProp solves this issue by adding a decay term to~~ ~~and~~

RMSProp solves this issue by adding a decay term to the ~~sum of~~ gradient sum.

Answer to the Question No. 11

In Adam algorithm,

$$m_1^{(i+1)} = \beta_1 \cdot m_1^{(i)} + (1-\beta_1)\nabla L(\theta^{(i)})$$ → first moment : velocity with momentum

$$m_2^{(i+1)} = \beta_2 \cdot m_2^{(i)} + (1-\beta_2)\left(\nabla L(\theta^{(i)}) \bullet \nabla L(\theta^{(i)})\right)$$ → second moment: elementwise step size scale

$$\theta^{(i+1)} = \theta^{(i)} - \eta m_1^{(i+1)} \bullet \frac{1}{\sqrt{m_2^{(i+1)} + \varepsilon}}$$

Because the moments are initiated to zero, when dividing by the second moment, we will get a large step. To compensate for this, we use bias correction term by dividing the momentum by a number depending on iteration number. ~~so that~~

Answer to the Question No. 12

Gradient descent with line search is essentially another method to approximate the optimum search step ~~after computing~~ (instead of a fixed step size), given a gradient direction.

- Given direction : $u = \nabla f(x)$
- Best step size : $\eta^* = \underset{\eta}{\operatorname{argmin}}\, f(x + \eta u)$
- Gradient descent : $\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta^{(i)} \nabla f(\theta^{(i)})$ ← η^*

Bracketing is a line search algorithm that iteratively performs a binary search given a range, to check if the optimum value lies in the one half of the bracket or the other half.

Given bracket $[a, b, c]$: $x = (b+c)/2$

If $f(x) \leq f(b) \Rightarrow [b, x, c]$

If $f(x) > f(b) \Rightarrow [a, b, x]$

Continue with smaller and smaller brackets until the bracket is small enough.

Alternative to bracketing: coordinate decent algorithm

It is more accurate but involves greater computation.

Answer to the Question No. 13

Quasi-Newton method involves approximation of Hessian inverse using gradient evaluations.

Algorithm:

1) Compute quasi-Newton direction:
$$\Delta\theta = -(H^{(i-1)})^{-1} \nabla J(\theta^{(i-1)})$$
2) Determine step size η^*
e.g. - bracketing line search
3) compute parameter update
$$\theta^{(i)} \leftarrow \theta^{(i-1)} + \eta^* \Delta\theta$$
4) Compute updated Hessian approximate $H^{(k)}$

Advantage of BFGS algorithm over the Newton is that it is less computationally expensive.

Compared to Adam, BFGS computes a more accurate approximation of Hessian matrix. However, in doing so, BFGS requires a large number of examples (e.g. entire dataset) which is computation consuming.

Regularization

Answer to the Question No. 1

Multiply each coefficient by $p \in [0, 1]$. As iterations progress, weights that are not reinforced, decay to 0. This is equivalent to adding regularisation term to the loss function.

Answer to the Question No. 2

Early stopping consists of splitting the training set into training and validation in order to see when the model overfits the training data. We stop when the validation error increases instead of when training error stops decreasing.

Strategy #1: Retrain on all the data using the number of iterations found from validation.

Strategy #2: Continue training from previous weight with entire data when validation loss is bigger than training loss.

Answer to the Question No. 3

Data augmentation is performed by the following ways:

1) Augment in feature or data domain
2) Augment by interpolating between examples or by adding noise (in data or feature domain)
3) Augment by transforming data (e.g. crop / rotate / scale / change intensity in images)

This increases the variability of samples and prevents the model from overfitting.

Answer to the Question No. 4

At each traing stage, ~~drop out~~ dropout units in fully connected layers with probability $(1-p)$ where p is the hyperparameter. ~~Weights~~ ~~w~~ Nodes which ~~are removed~~ were removed temporarily, are ~~reinstein~~ reinstated with their original weights in the subsequent stage.

It reduces dependency on a single node and hence distributes features across multiple nodes. It also reduces overfitting.

The main disadvantage is that it increases convergence time.

Answer to the Question No. 5

Since dropout is not performed during testing, thus. the output of each unit during testing has to be multiplied by the dropout chance.

Answer to the Question No. 6

Batch normalisation is the process of making the outputs of a given layer or node of the network to have a distribution with zero mean and unit variance. ~~Thus~~ ~~p~~

During training:

Input : Values of x over a mini-batch $B = \{x_{1\ldots m}\}$

Parameters to be learned : γ, β

Output : $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \longrightarrow \text{mini-batch mean}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \longrightarrow \text{min-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \longrightarrow \text{normalization}$$

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$ ⟶ scaling & shifting

During testing : we use the average normalisation values computed during training.

The randomness is introduced by the input batch.

Answer to the Question No. 7

Scale and shift is necessary during batch normalisation because some saturation is essential in order to terminate learning.

~~Scale = σ_j~~ $\gamma_j = \sigma_j$ and $\beta_j = \mu_j$ causes the normalisation to be cancelled.

We calculate the gradient wrt γ & β to learn.

A good initial value is, scale (γ) = 1 and shift (β) = 0.

Answer to the Question No. 8

Ensemble classifier means to train a set of ~~cl~~ independent classifiers whose combined response can produce an improved performance compared to each other, as our prediction is an average which is less sensitive to bias of the one single model.

~~Ca~~ Dropout, bagging and boosting are some methods of generating ensemble classifiers.

Programming Question Part

1. Datasets

Classification

The iris dataset [1] from UCI repository is perhaps the most popular known database found in the pattern recognition literature. The dataset is balanced and contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The samples have 4 attributes. The target labels consist of three classes, where each class corresponds to a particular type of iris plant, thereby making this a multi-class classification problem.

Regression

The wine quality dataset [2] from UCI repository has been used for the regression problem. The dataset contains 4898 samples each having 12 features. The objective is to predict the alcohol content of the wine.

2. Pre-processing

Classification

The dataset was loaded using a pandas dataframe. Then, we have dropped the ‘Id’ column as it is just an index having no meaning for our problem. The input features and labels were then isolated. The labels for our problem have text variables, e.g. ‘Iris-setosa’, ‘Iris-versicolor’ and ‘Iris-virginica’. Then the target variables were transformed into integers using the label encoder function from the scikit learn library. Afterwards, the examples were splitted into training and testing sets and the input features were normalized.

Regression

The dataset was loaded using a pandas dataframe. The ‘quality’ attribute was dropped because it had discrete values and is not relevant for determining the percentage of alcohol. Then, the input and the output were isolated. Note that the labels are continuous values here. We have splitted the examples into training and testing sets. Lastly, the input features were normalized.

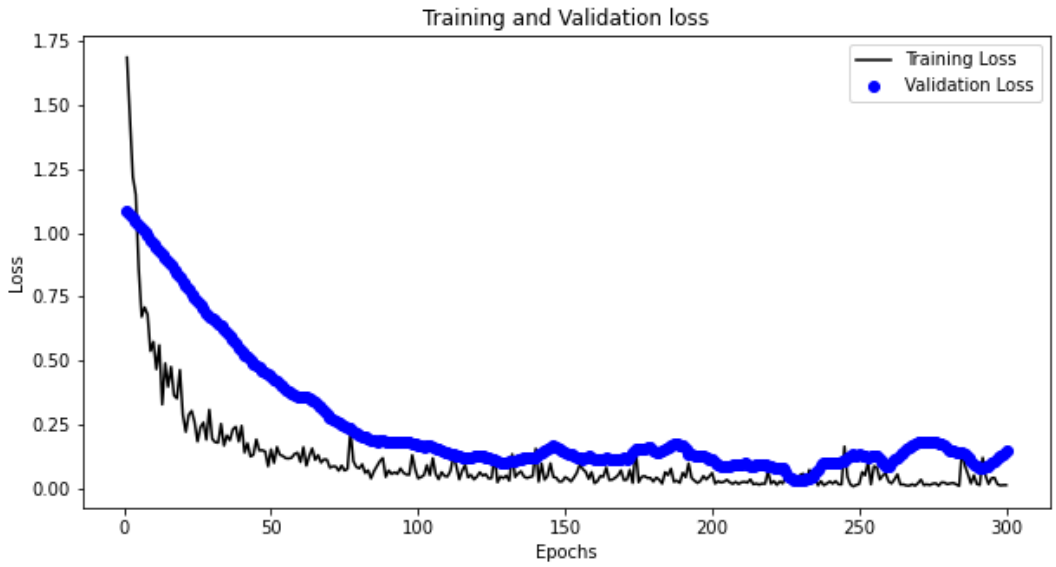
3. Evaluation of loss functions

Classification

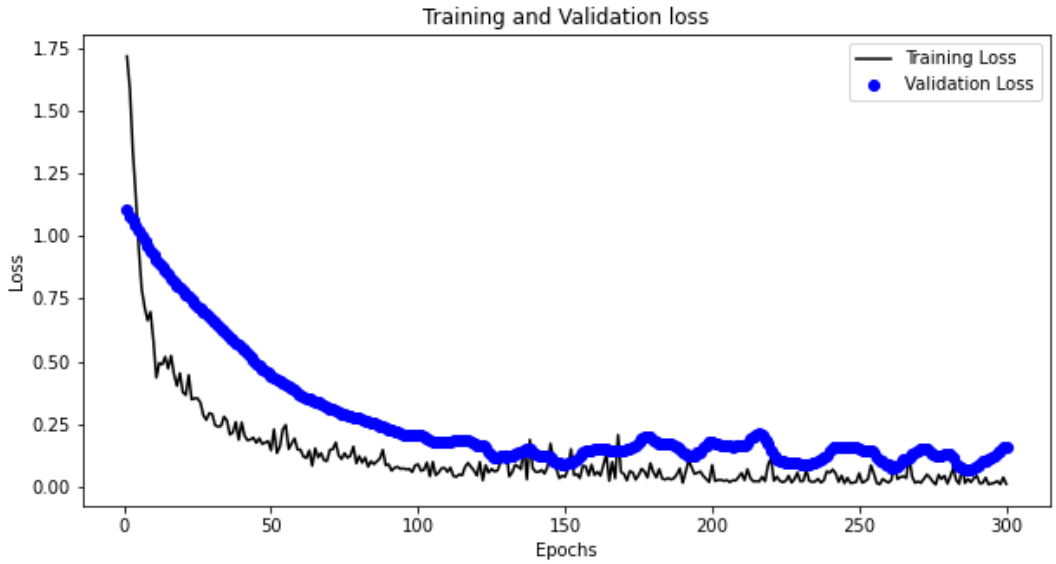
The following table shows the summary of hyper parameters tuning and their corresponding evaluations of various loss functions

Items	Parameters
Input Layer	64 units, ReLu activation
Hidden Layer	32 units, ReLu activation
Hidden Layer	16 units, ReLu activation
Output Layer	3 units, Softmax activation
Batch size	32
Epoch	300 (only for visual representation) 150 (for final model)
Metrics	Categorical accuracy
Optimizer	SGD

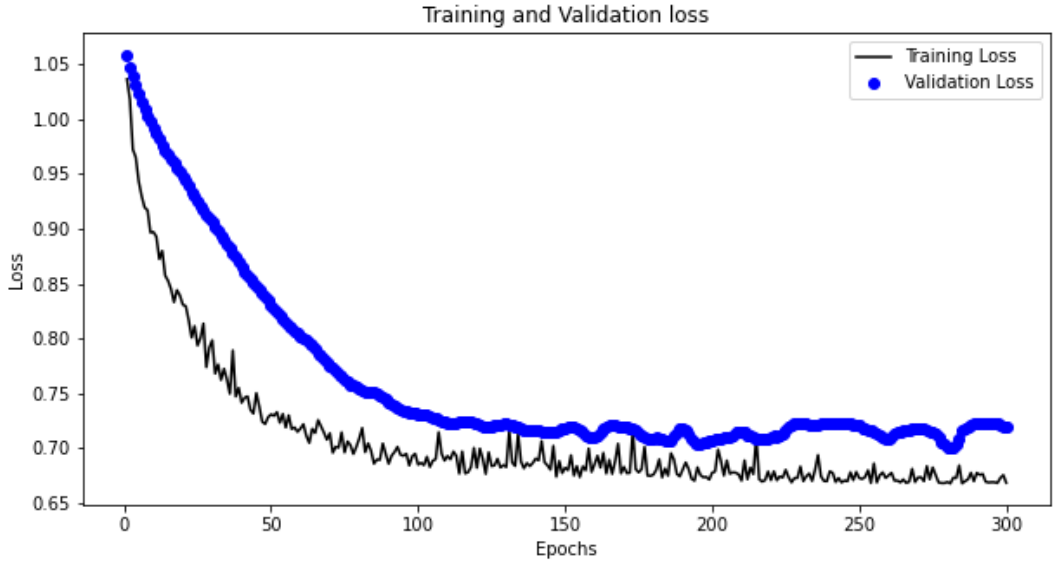
Categorical Cross Entropy Loss:



Kullback - Leibler Divergence Loss:



Hinge Loss:



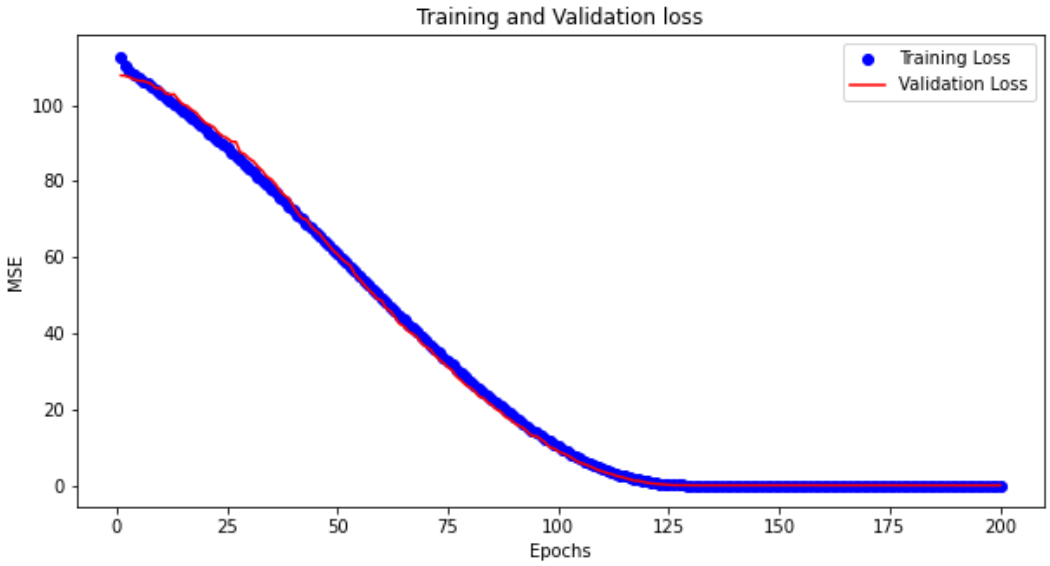
In conclusion, the categorical cross entropy loss was selected as the loss function for the network since it's performance is similar to that of the kullback-leibler divergence loss function. Besides, it also converged faster than the hinge loss.

Regression

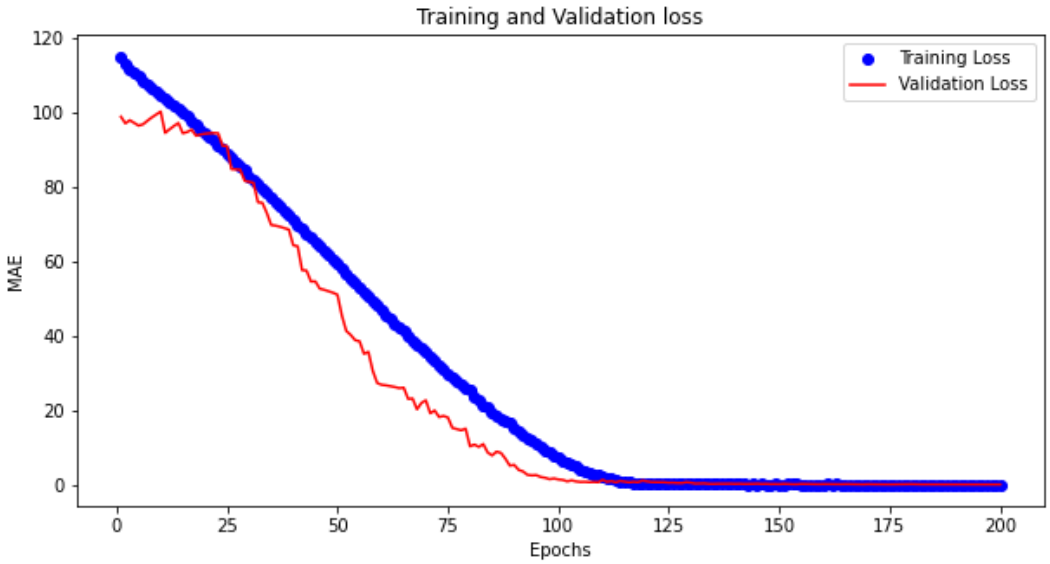
The following table shows the summary of hyper parameters tuning and their corresponding evaluations of various loss functions

Items	Parameters
Input Layer	512 units, ReLu activation
Hidden Layer	128 units, ReLu activation
Hidden Layer	32 units, ReLu activation
Hidden Layer	16 units, ReLu activation
Output Layer	1 units, Linear activation
Batch size	512
Epoch	100 (only for visual representation)
Metrics	Mean absolute error (MAE) or Mean squared error (MSE)
Optimizer	RMSprop

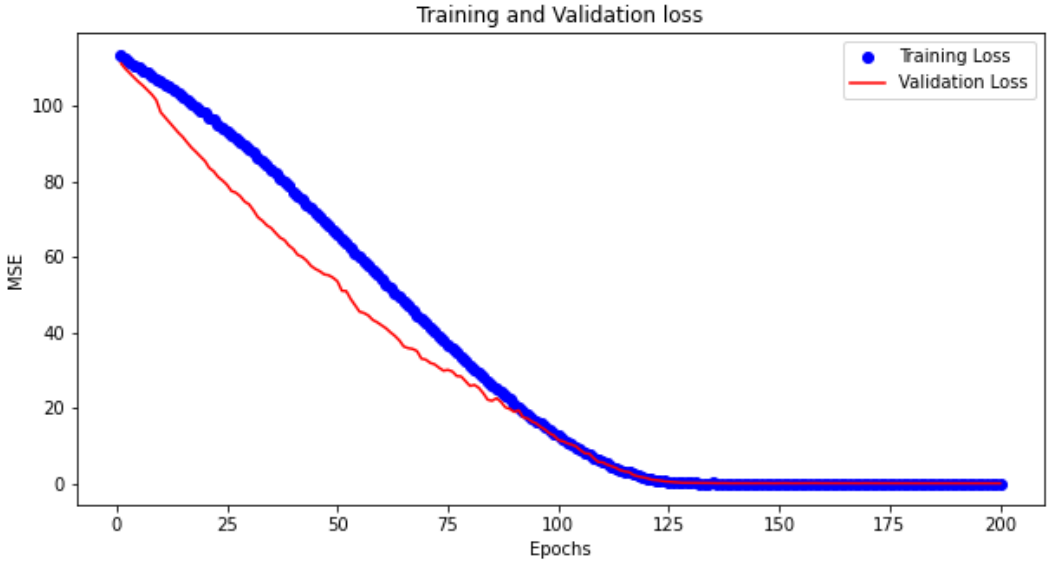
L2 Loss:



L1 Loss:



Log-cosh Loss:

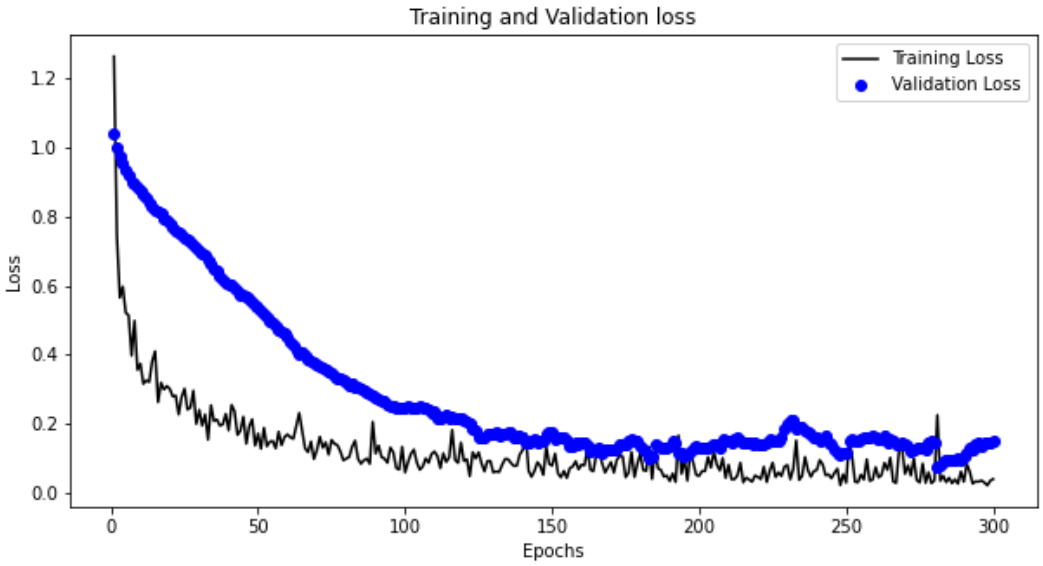


In conclusion, L2 loss function results in the best MSE profile compared to that of L1 and log-cosh loss functions. So, it is selected for the network.

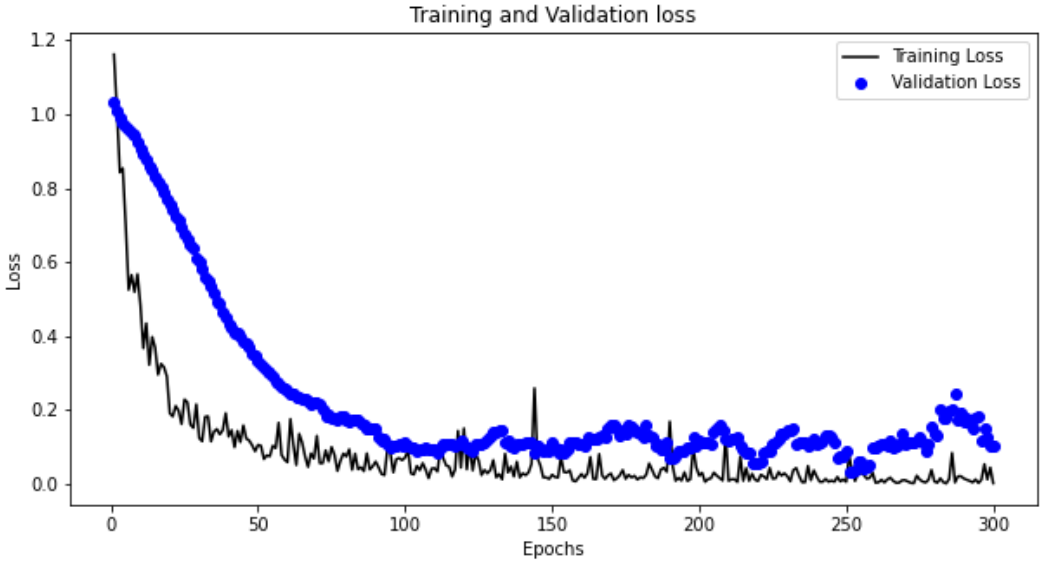
4. Evaluation of optimizer algorithms

Classification

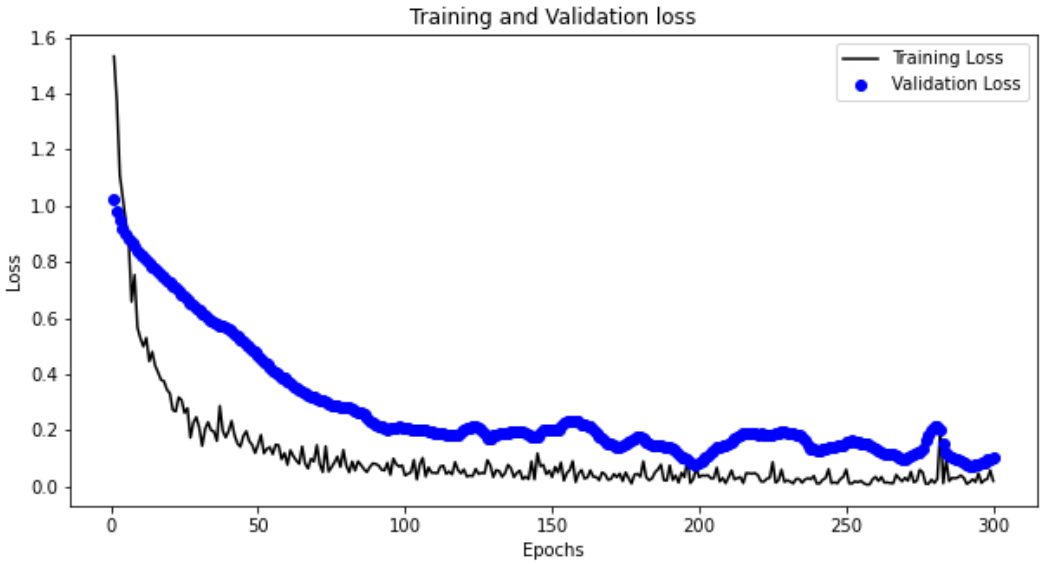
Stochastic Gradient Descent (SGD):



RMSprop:



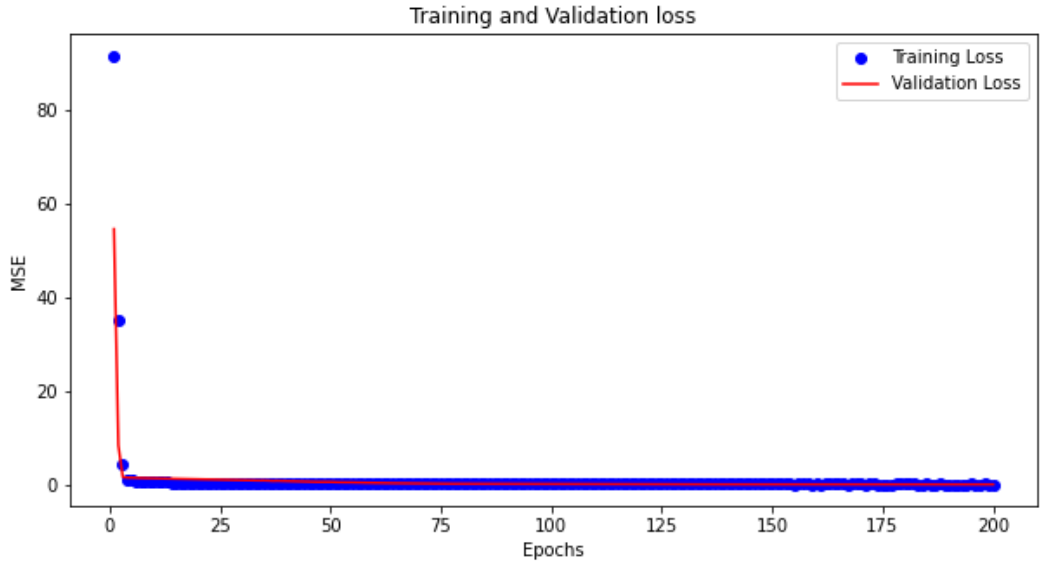
Adam:



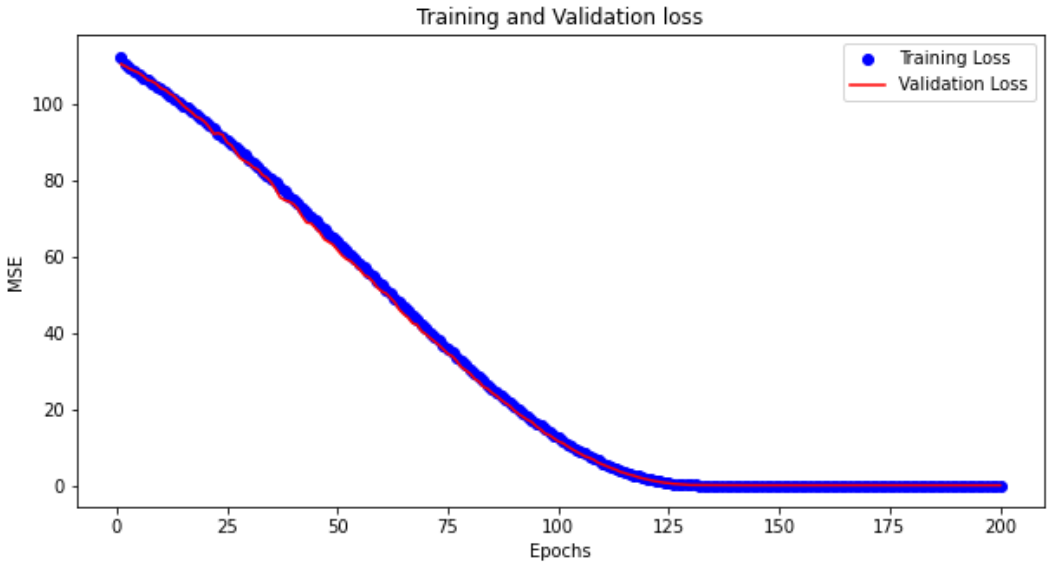
In conclusion, the adam optimizer is chosen to minimize our loss function as it converges the earliest compared to that of stochastic gradient descent and rmsprop optimizers.

Regression

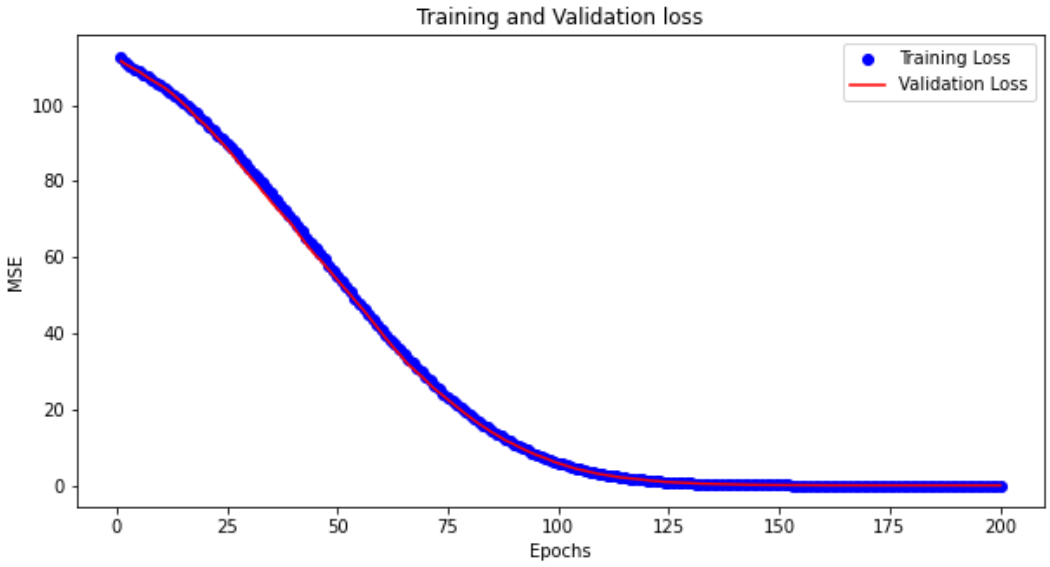
Stochastic Gradient Descent (SGD):



RMSprop:



Adam:

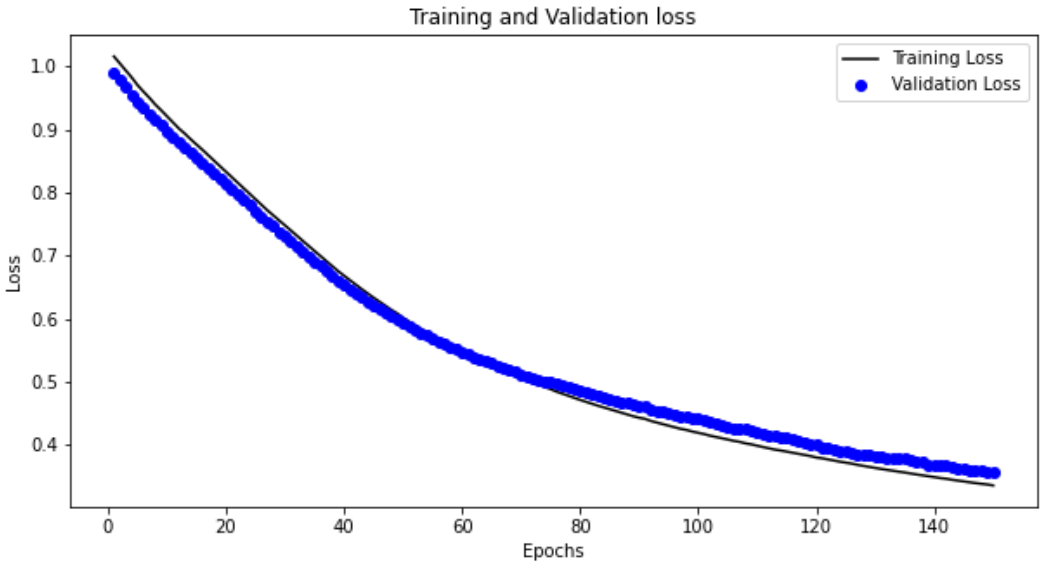


In conclusion, the stochastic gradient descent optimizer has been chosen to minimize our loss function as it converges the fastest compared to that of adam and rmsprop optimizers.

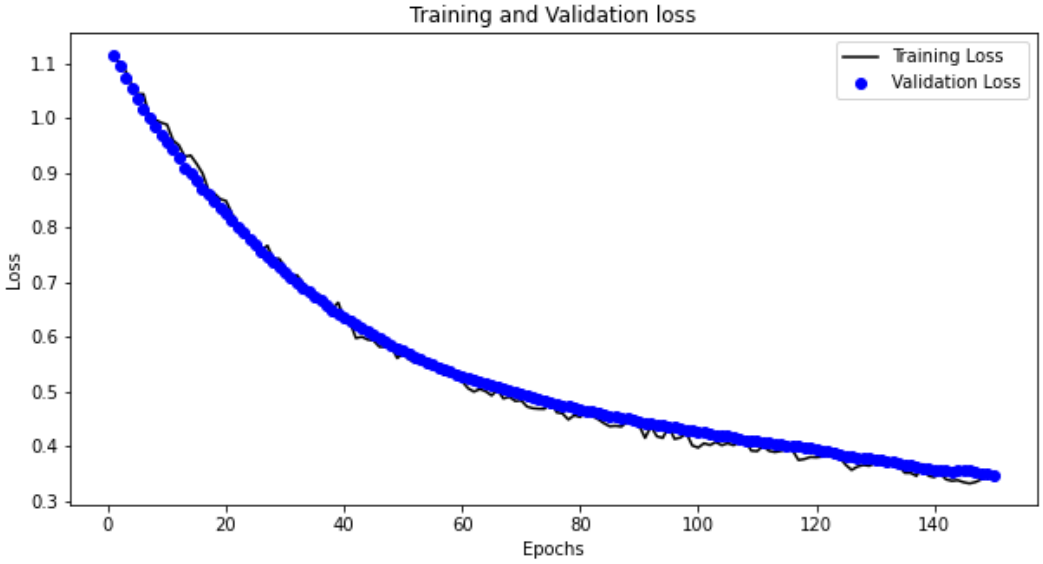
5. Effects of Regularization

Classification

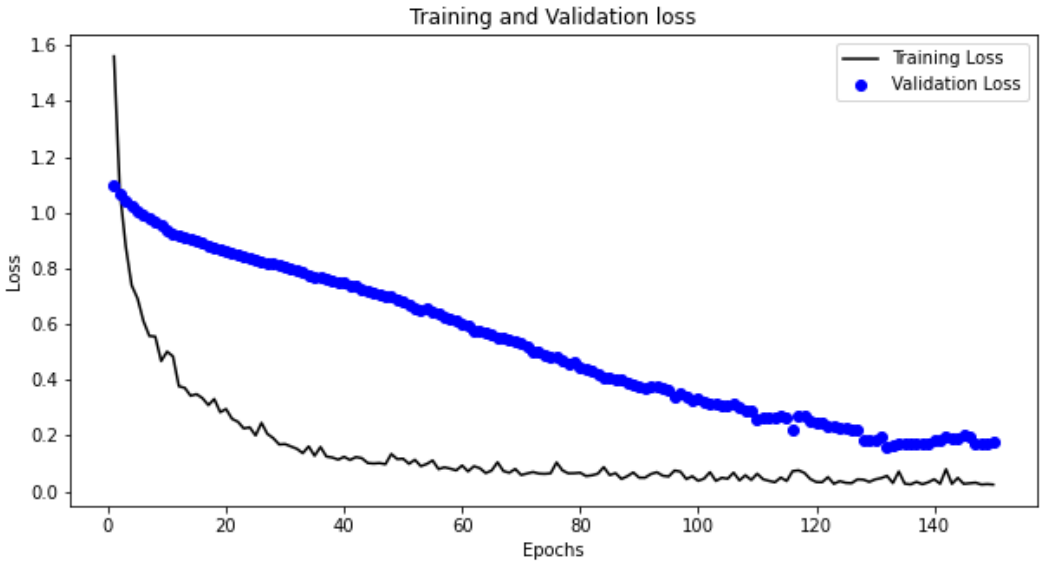
SGD optimizer with lr=0.1, decay=1e-6, momentum=0.9, nesterov=True



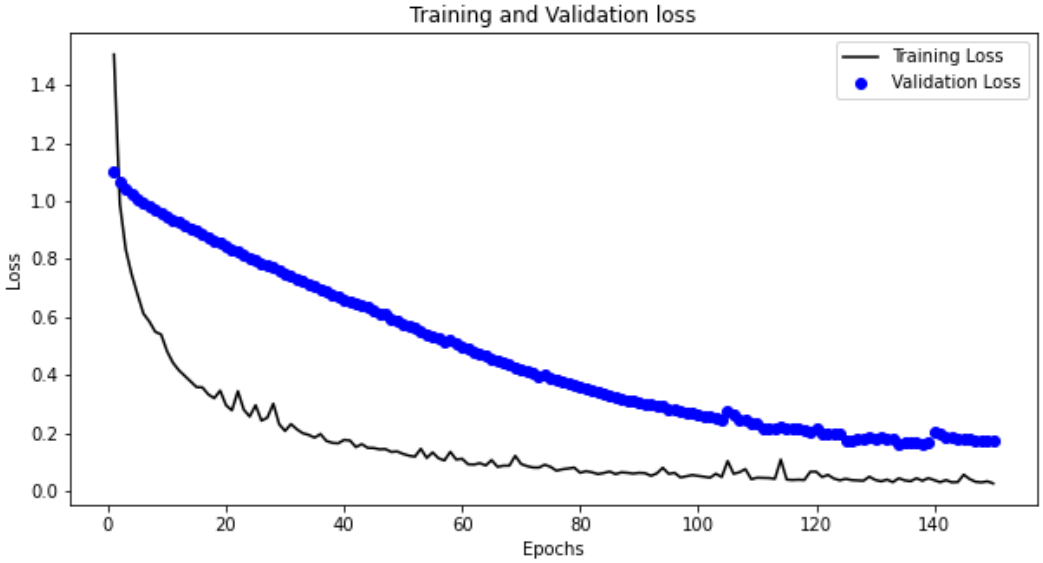
Adding dropout = 0.2 after every hidden layer:



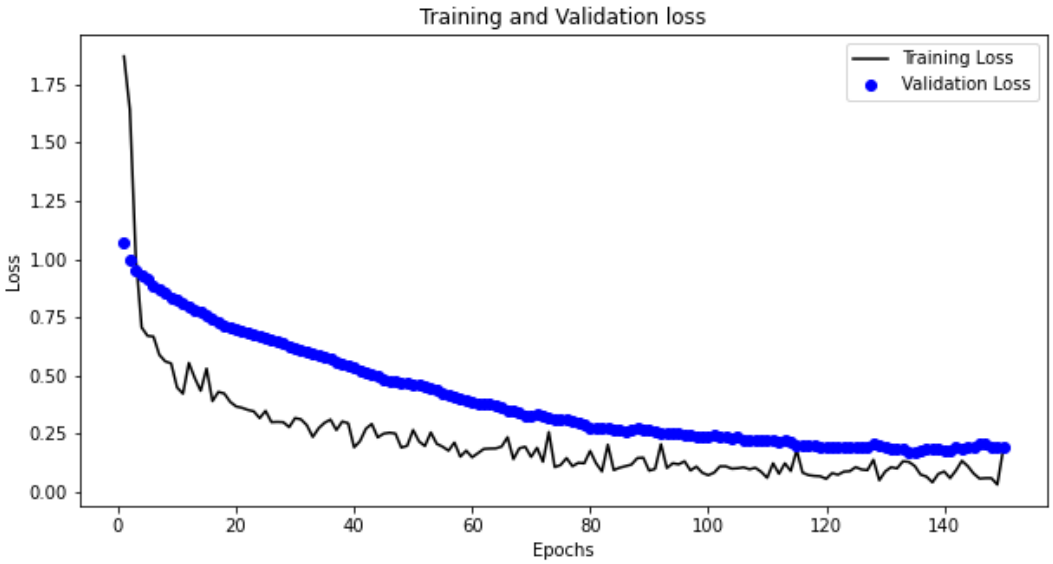
Removing all dropouts and adding batch normalization after every hidden layer:



Adding batch normalization and a drop out of 0.2 only after the first hidden layer:

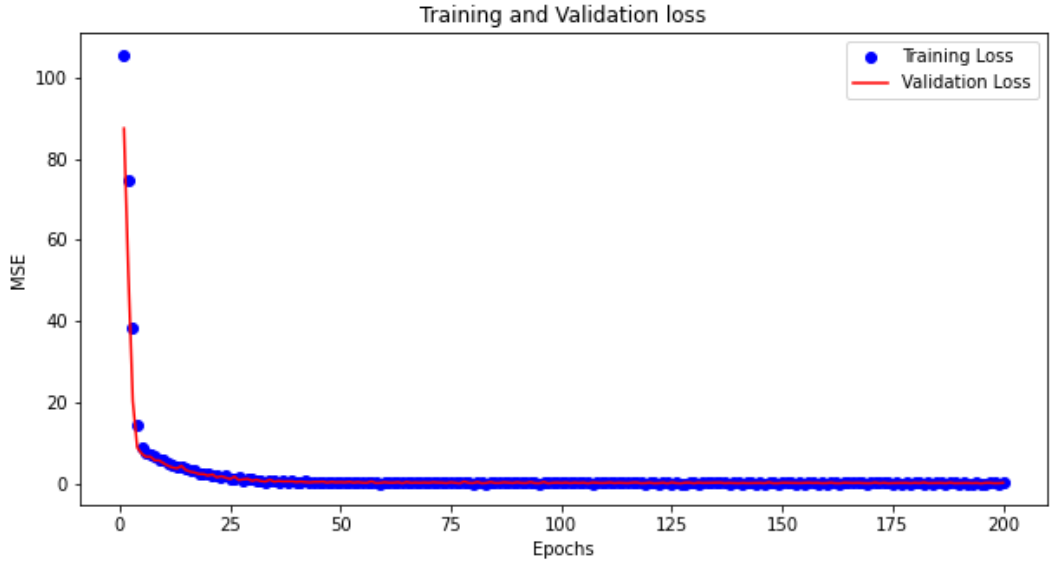


Adding dropout=0.2 after the first hidden layer and batch normalization after the second and third hidden layers:

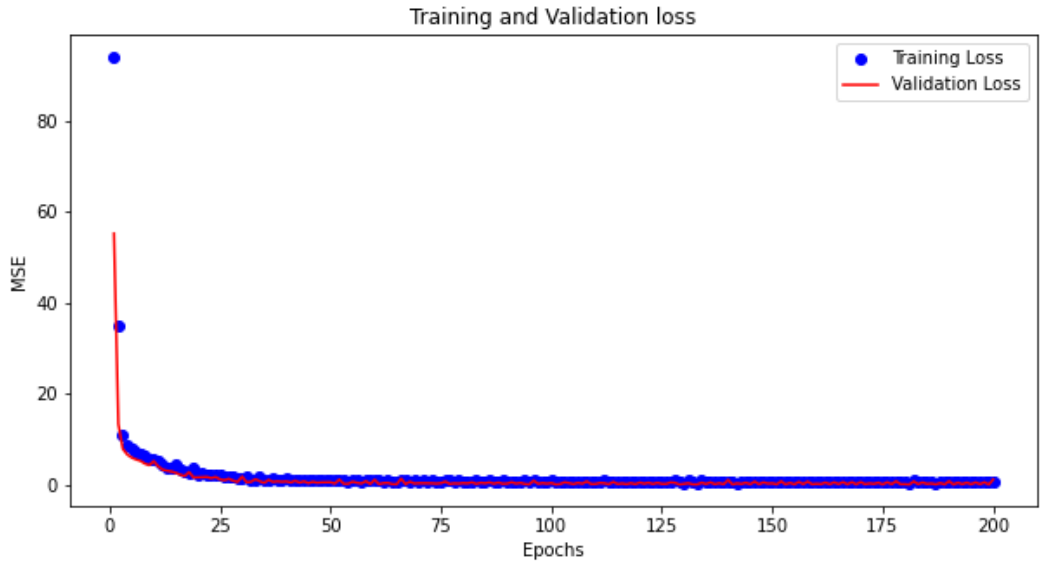


Regression

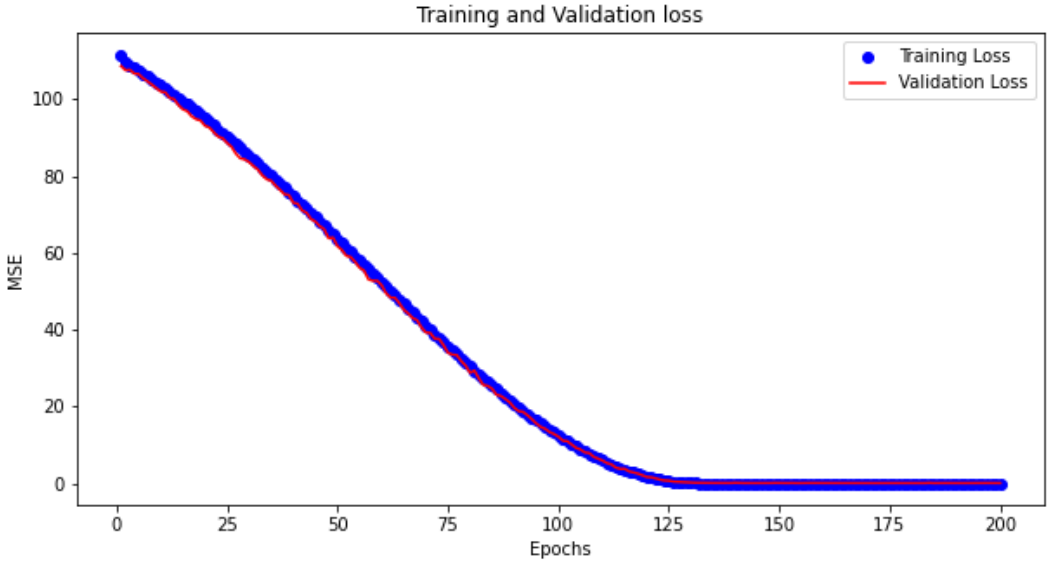
RMSprop optimizer with (lr=0.001, rho=0.9, epsilon=1e-07, decay=0.0):



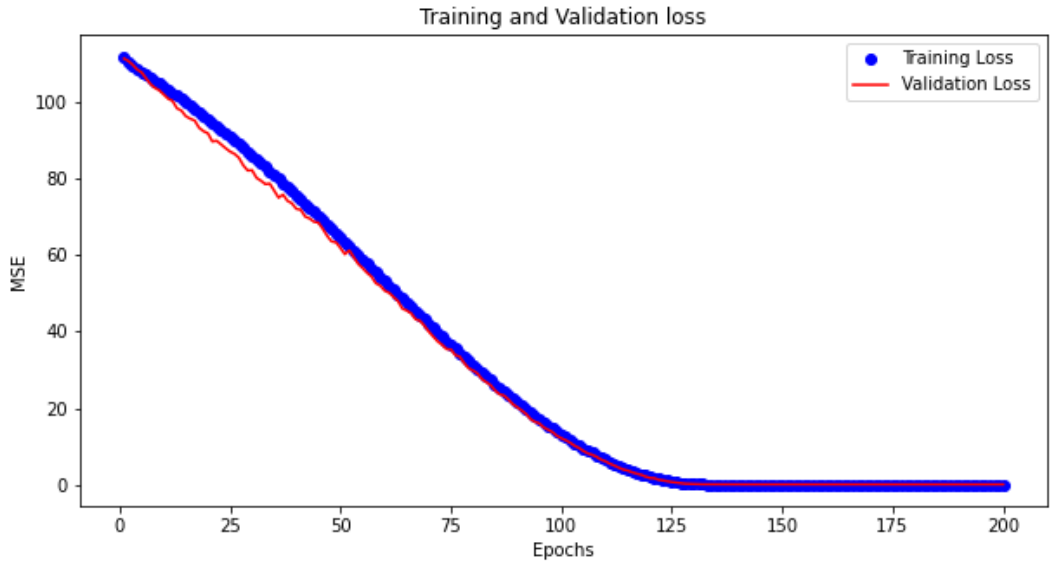
Adding dropout = 0.2 after every hidden layer:



Removing all dropouts and adding batch normalization after every hidden layer:



Adding drop=0.2 after first two hidden layers and batch normalization after every other subsequent hidden layers:



In conclusion, RMSprop optimizer with (lr=0.001, rho=0.9, epsilon=1e-07, decay=0.0) has been selected and dropout=0.2 after the first two hidden layers and batch normalization after each of the subsequent hidden layers should be added for smoother validation loss. The model is saved with the name ‘regression.h5’.

6. Summary of optimized final model parameters

Classification

Items	Parameters
Input Layer	64 units, ReLu activation
Hidden Layer	32 units, ReLu activation
Hidden Layer	16 units, ReLu activation
Output Layer	3 units, Softmax activation
Batch size	32
Epoch	150
Metrics	Categorical accuracy
Optimizer	SGD(lr=0.1, decay=1e-6, momentum=0.9,

	nesterov=True)
Loss function	Categorical cross entropy
Drop out	After first hidden layer
Batch normalisation	After the second and third hidden layer

Final Model gives:
Accuracy = 84.44

Regression

Items	Parameters
Input Layer	512 units, ReLu activation
Hidden Layer	128 units, ReLu activation
Hidden Layer	32 units, ReLu activation
Hidden Layer	16 units, ReLu activation
Output Layer	1 units, Linear activation
Batch size	512
Epoch	130
Metrics	Mean squared error (MSE)
Optimizer	RMSprop(lr=0.001, rho=0.9, epsilon=1e-07, decay=0.0)
Loss function	Mean squared error
Drop out	After first and second hidden layers only
Batch Normalisation	After third and fourth hidden layers only

Final Model gives:
Mean Squared Error = 0.17

References:

[1] Iris Dataset (UCI Repository)
[2] Wine Quality (UCI Repository)