Course: DATS 6203 – Machine Learning II

Malaria cell classification using MLP

Exam Number: 1

Instructor: Dr. Amir Jafari

Name: Rehapriadarsini Manikandasamy

RM
Initials

07.03.2022
Date

INTRODUCTION

The exam focused on developing a Multi-Layer Perceptron (MLP) network to distinguish the cells infected due to Malaria. The MLP network primarily focused on the classification of four different types of cell categories: “red blood cell”, “ring”, “schizont” and “trophozoite”. Since the dataset turned out to be an imbalanced one, accuracy is not considered as the best metric for finding the model. The metrics: macro-averaged F1-score and the Cohen’s Kappa score were used to measure the performance of the models.

During the seven-day challenge, the best model developed had an average score (F1-score and Kappa score) of 0.85.

DATASET

The dataset provided had 7315 cell images and the labels of these images were provided in an excel sheet. The provided dataset had been divided based on train and test labels. The ‘train’ split had 6024 images files, out of which ‘red blood cell’ had 4899 images while others had 776 (trophozoite), 256 (ring) and 93 (schizont). The dataset was unbalanced, and the images were having different image dimensions (i.e.: 100x103, 110x117, etc.,)

Preprocessing:

1. **Image Resizing –** The images in the dataset didn’t have unified sizes but the neural networks needed an input vector of same size. Hence, the images were resized to shape of 50 X 50 X 3.
2. **Label encoding –** The targets provided were ‘strings. A label encoding step was implemented to generate a target class: 1,0,0,0 – red blood cell, 0,0,0,1 – trophozoite, 0,1,0,0 – ring and 0,0,1,0- schizont. The model developed generates a target as integers (0,1,2,3) accordingly later.

MODELLING

Network Architecture:

The best model is a MLP network with 4 hidden, 1 input and 1 output layer. The input dimension is 7500 (50X50X3) and the input layer has an activation function of Rectified Linear Unit (ReLU). The hidden layers had activations of ReLU (first two) and SELU - Scaled Exponential Linear Units (last two) while the output layer uses Softmax activation which is good for predicting multinomial probability distribution. The number of neurons in each layer was 128,32,32,32,32. The optimizer used was ‘Adam’ and loss was ‘Categorical crossentropy’.

Tuning of the network:

The following tuning operations were performed throughout the seven-day challenge. The performance metrics on test data and held out (validation) datasets were provided in tables below after the discussion

Day 1 [i]**–** A simple three-layer network with layers for dropout and batch normalization was developed and trained with 10 epochs. Unfortunately, the test script had the model definition which ended up training the model again and resulted in having different values from original test script.

Day 2 – A three-layer model with 'relu' activation on input and hidden layers were generated with dropout and batch normalization layers. The optimizer was 'Adam' and loss was calculated based on 'Categorical crossentropy' with kernel initializers as GlorotUniform on input and hidden layers. The batch size and epochs were increased to 64 and 100 accordingly.

Day 3 – Two more hidden layers (relu activations) were added to the existing model with learning rate of 0.001 to check whether a complex network would produce an improvement. As expected, there was a good improvement on the model's performance

Day 4 – The existing model was replaced by using 'SELU' activations on last to hidden layers based on the inspiration from the following concept: *'Normalized outputs seem to be helpful in stabilizing the training process. That's the main reason behind the popularity of Batch Normalization. SELU is a way to output the normalized activations to the next layer.'*[ii] This helped in reducing the usage of batch normalization. A kernel-initializer of 'Orthogonal' was used.

Day 5 - Adding few more layers, experimenting on activation function and weight initializers didn't significantly improve the model. So, the neurons on first two hidden layers were increased to 128 and last two were replaced with 'elu' activation.

Day 6 – The model from day 4 was used again with kernel constraints[v] and momentums[iii] on optimizers. Inspiration: *'A large learning rate can result in very large network weights. Imposing a constraint on the size of network weights such as max-norm regularization with a size of 4 or 5 has been shown to improve results.'*[iv]

Day 7 [vi,vii,viii]**–** For the same model with max_norm kernel constraint, a bias initializer was set to Truncated Normal with mean and standard deviation of 0 and 1 respectively. This time the model was trained for 200 epochs and as expected to be the best model amongst all the submissions.

Final model:

```
model = tf.keras.Sequential()

# Define the first dense layer
model.add(tf.keras.layers.Dense(128, activation='relu',
input_shape=(INPUTS_r,),

kernel_initializer=tf.keras.initializers.Orthogonal(),
kernel_constraint=max_norm(4),bias_initializer=tf.keras.initializers.Truncate
dNormal(mean=0., stddev=1)))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(32, activation='relu',
kernel_initializer=tf.keras.initializers.Orthogonal(),kernel_constraint=max_n
orm(4),bias_initializer=tf.keras.initializers.TruncatedNormal(mean=0.,
stddev=1)))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(32, activation='relu',
kernel_initializer=tf.keras.initializers.Orthogonal(),kernel_constraint=max_n
orm(4),bias_initializer=tf.keras.initializers.TruncatedNormal(mean=0.,
```

```
stddev=1)))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(32, activation='selu',
kernel_initializer=tf.keras.initializers.Orthogonal(),kernel_constraint=max_n
orm(4),bias_initializer=tf.keras.initializers.TruncatedNormal(mean=0.,
stddev=1)))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.Dense(32, activation='selu',
kernel_initializer=tf.keras.initializers.Orthogonal(),kernel_constraint=max_n
orm(4),bias_initializer=tf.keras.initializers.TruncatedNormal(mean=0.,
stddev=1)))
model.add(tf.keras.layers.Dropout(0.1))
model.add(tf.keras.layers.Dense(OUTPUTS_a, activation='softmax'))  # final
layer , outputs_a is the number of targets
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001,beta_1=
0.99,beta_2=0.999,epsilon=1e-07,amsgrad=True),
loss='categorical_crossentropy', metrics=['accuracy'])
```

Performance measures based on test set

Metric/Day	Day1	Day2	Day3	Day4	Day5	Day6	Day7
F1_macro	~0.25	~0.45	0.65	0.66	0.63	0.71	0.75
Cohen	~0.33	~0.37	0.72	0.75	0.67	0.79	0.79
Average	0.27	0.37	0.45	0.47	0.43	0.50	0.516

Performance measures based on validation set

Metric/Day	Day1	Day2	Day3	Day4	Day5	Day6	Day7
F1_macro	0	0.736839	0.79955	0.80451	0.80451	0.8477	~0.8501
Accuracy	0	0.887771	0.89551	0.90557	0.90557	0.9272	~0.94
Cohen	0	0.585908	0.70360	0.70345	0.70345	0.7681	~0.76
Average	0	0.736839	0.79955	0.80451	0.80451	0.8477	0.8501
Sum	0	1.473678	1.59911	1.60902	1.60902	1.6954	~1.70

The above table shows how the model has been gradually improved from the basic to a well-developed MLP network.

Failed techniques:

The following preprocessing techniques were attempted in this process, but they were not implemented due to the bugs and conceptual errors faced during the development process.

1. Since the dataset was imbalanced a technique of oversampling was attempted to have a balanced data using the method of data augmentation.
2. To find the best model parameters a randomized search cv or grid search cv can be performed.

CONCLUSION/ FUTURE WORK

This work helped in understanding the idea behind the development of a multi-layer perceptron network and optimizing its parameters for a better trained model. The best model (model from day 7) was picked based on the average of all the provided metrics. The best model has an average score of 0.85 on the validation set and 0.51 on the test set. The model in the future can be enhanced using the discussed preprocessing technique like augmentation and parameter estimator technique like cross validation to find a better model.

REFERENCES:

i. https://keras.io/api/layers/initializers/#glorotuniform-lasshttps://arxiv.org/abs/2108.08186
ii. https://www.hardikp.com/2017/07/24/SELU-vs-RELU/
iii. https://keras.io/api/optimizers/adam/
iv. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
v. https://keras.io/api/layers/constraints/
vi. https://subscription.packtpub.com/book/programming/9781838821654/1/ch01lvl1sec04/3-multilayer-perceptron-mlp
vii. https://androidkt.com/initialize-get-biases-keras-model/
viii. https://stackoverflow.com/questions/40708169/how-to-initialize-biases-in-a-keras-model