IEEE CIS/SMC DEEP LEARNING CHALLENGE

Name: MOHIT SHARMA

Student ID: 110031631

Email ID: sharma88@uwindsor.ca

Introduction

- A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. There are different types of neural networks but they always consist of the same components: neurons, synapses, weights, biases, and functions. These components functioning similar to the human brains and can be trained like any other ML algorithm.
- DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network. For instance, it was proved that sparse multivariate polynomials are exponentially easier to approximate with DNNs than with shallow networks.
- DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network did not accurately recognize a particular pattern, an algorithm would adjust the weights.That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.

1. DESIGN

a. Motivation

- The architecture utilizes One vs. Rest classifier technique to predict the probabilities of belonging to a class by multiplying the probabilities of not belonging to other classes.
- Since, in a dataset, the number of samples of not belonging to a class would be more than the number of samples belonging to a class, it makes sense to train the model to predict the probability of not belonging to a class.
- Hence, a model trained for one vs. rest is good at predicting the probability of ‘not belonging to a class’.
- The probability of belonging to a class (assume there are 5 classes) can then be calculated as follows:

  P(‘Class_1’) = P(‘Class_1’) * P(‘not Class_2’) * P(‘not Class_3’) * P(‘not Class_4’) * P(‘not Class_5’)

  Where, P(‘Class_i’) = Probability of belonging to Class i

  P(‘not Class_i’) = Probability of not belonging to Class i

  = 1 – (Probability of belonging to Class i)

- This method could even be useful for tackling imbalanced classes because the one vs. rest classifier is being trained to predict probabilities of not belonging to a class and there are larger numbers of such samples. Therefore, it is easier to predict if a sample does not belong to a class i.

b. Algorithm

- The multi-label classification task is divided into two: multi-class classification with ten classes for Label1 and a binary classification task for Label2.
- The multi-label classification task is further transformed into a binary classification task by utilizing one vs. rest classification technique. For each class i where i ∈ $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, the Label1 is transformed as follows:

  New Label1 = $\begin{cases} 1 & if\ Label1\ \ != \ \ i \\ 0 & if\ Label1 == \ \ i \end{cases}$

  As a result, the model required to be trained is for binary classification task where it predicts the probability of not belonging to a class i.

- Both the Labels are classified using the same model for the binary classification task.
- For Label1, let the probabilities predicted by the model be defined by a dataframe named 'not_belongs'. Then,

  P[1] = P[1] * not_belongs[2] * not_belongs[3] * not_belongs[4] * not_belongs[5] * not_belongs[6] * not_belongs[7] * not_belongs[8] * not_belongs[9] * not_belongs[10]

c. Architecture

NETWORK ARCHITECTURE

- The model used for the binary classification task has the following architecture:

  INPUTS

  - The model consists of two inputs, one for categorical columns and the other for continuous columns of the feature vectors.
  - The shape of the categorical data is 17 and shape of the continuous data is 11
  - The categorical input layer is named as ‘cat_input’ and continuous input layer is named as ‘cnt_input’

  EMBEDDING LAYER:

  - An embedding layer is used to process the categorical data which is followed by a Flatten layer.
  - The embedding layer uses a vocabulary size of 2819
  - The embedding layer is named as ‘cat_embedding’ and the flatten layer is named as ‘cat_flatten’
  - As the name suggests, flatten layer flatten’s the inputs.

DENSE LAYER:

- Dense layer with 32 neurons and ‘relu’ (rectified linear unit) activation processes the continuous data from cnt_input layer.
- The dense layer is followed by the dropout layer which randomly sets input units to 0 with a frequency of 0.4 during training to prevent overfitting.

CONCATENATION LAYER:

- Concatenation layer combines the outputs of the embedding layer and the dense layer to feed to the final network of dense layers.

FINAL NETWORK OF DENSE LAYERS:

- This section has three dense layers with 128, 64 and 32 neurons resp. Each of these three dense layers has ‘relu’ activation.
- These layers are named as ‘dense1’, ‘dense2’ and ‘dense3’.

OUTPUT LAYER:

- The output layer is a dense layer with a single neuron and ‘sigmoid’ activation required for binary classification task.
- The output layer is named as ‘output’.

2. EXPERIMENTAL RESULTS

a. Experimental Setting

MODEL TRAINING AND FITTING PARAMETERS

- Loss: ‘binary_crossentropy’ is used as the loss function with the default parameters as follows:

```
from_logits = False
label_smoothing = 0
reduction = losses_utils.ReductionV2.AUTO
name = ‘binary_crossentropy’
```

- Metrics: ‘exact_match_ratio’ and ‘Hamming Loss’ are used as the metrics to be evaluated. Custom functions (emr_custom & hamming_loss_custom) are written to calculate the metrics.
- Optimizer: tensorflow.keras.optimizers.Adam or ‘adam’ is used as the optimizer with the default parameters:

```
learning_rate=0.001
beta_1=0.9
beta_2=0.999
epsilon=1e-07
amsgrad=False
name='Adam'
```

- While fitting the model using model.fit() function, following parameters are used:

  ```
  steps_per_epoch=1000
  batch_size=256
  epochs=50
  validation_split=0.2 (20% train data is used for validation)
  callbacks=[scheduler_cb, early_stopping_cb]
  ```

  The model is fitted with scheduler call back and early stopping callback.

  The scheduler callback utilizes the tensorflow API tf.keras.callbacks.ReduceLROnPlateau which reduces the learning rate by a factor once learning stagnates. This callback has been configured to monitor the validation_loss and if no improvement is seen for 2 epochs, the learning rate is reduced.

  The early stopping callback utilizes the tensorflow API tf.keras.callbacks.EarlyStopping which stops training when a monitored metric has stop improving. The metric configured here is validation_loss.

  The parameters for ReduceLROnPlateau are:

  ```
  monitor='val_loss',
  factor=0.5,
  patience=2,
  verbose=0,
  mode='auto',
  min_delta=0.0001,
  cooldown=0,
  min_lr=0
  ```

The parameters for EarlyStopping are:

```
monitor='val_loss',
min_delta=0,
patience=5,
verbose=1,
mode='auto',
baseline=None,
restore_best_weights=True
```

DATA PREPROCESSING

Following data preprocessing is done:

1. Feature columns with a Pearson correlation greater than 0.9 within themselves are filtered and only a single feature column of all the correlated columns are kept for feeding to the model.

   Following are the columns that are dropped due to the high correlation to other feature columns:

```
Dropped columns are:
['F7', 'F8', 'F14', 'F15', 'F23', 'F25', 'F26', 'F34', 'F35', 'F36', 'F38', '
F40', 'F41', 'F42']
```

2. After filtering columns based on correlation, the selected Feature columns are split into categorical features and continuous feature s based on number of unique values for a feature column. The thr eshold on the number of unique values for a column being selecte d as categorical feature is kept at 10,000.
   Based on this criteria, the categorical and continuous columns ar e:

```
   Categorical columns are:

['F2', 'F3', 'F4', 'F5', 'F6', 'F10', 'F11', 'F20', 'F27', 'F28', 'F29',
'F30', 'F31', 'F32', 'F33', 'F37', 'F39']

   Continuous columns are:

['F1', 'F9', 'F12', 'F13', 'F16', 'F17', 'F18', 'F19', 'F21', 'F22', 'F24']
```

3. The categorical columns are then encoded using LabelEncoder from sklearn.preprocessing package.
4. Similarly, the continuous columns are scaled using StandardScaler from the same package.
5. Target columns are created for the one vs. rest classifier training. These columns are:
   targetnot0, targetnot1, targetnot2, targetnot3, targetnot4, targetnot5, targetnot6, targetnot7, targetnot8, targetnot9

---

b. Analysis

EXPERIMENT1

Metric	Value
Exact Match Ratio	0.78312199
Hamming Loss	0.16688529

EXPERIMENT2

Metric	Value
Exact Match Ratio	0.78043774
Hamming Loss	0.16905941

EXPERIMENT3

Metric	Value
Exact Match Ratio	0.78353495
Hamming Loss	0.16615653

EXPERIMENT4

Metric	Value
Exact Match Ratio	0.78872127
Hamming Loss	0.16143176

EXPERIMENT5

Metric	Value
Exact Match Ratio	0.78136083
Hamming Loss	0.16811203

EXPERIMENT6

Metric	Value
Exact Match Ratio	0.78359568
Hamming Loss	0.16628407

EXPERIMENT7

Metric	Value
Exact Match Ratio	0.78397221
Hamming Loss	0.16601078

EXPERIMENT8

Metric	Value
Exact Match Ratio	0.78276976
Hamming Loss	0.16708570

EXPERIMENT9

Metric	Value
Exact Match Ratio	0.77951464
Hamming Loss	0.17004931

EXPERIMENT10

Metric	Value
Exact Match Ratio	0.78088713
Hamming Loss	0.16857358

EXACT MATCH RATIO:

Average for exact match ratio for the 10 runs = 0.78279162

Standard deviation for exact match ratio for the 10 runs = 0.00257804

HAMMING LOSS:

Average for Hamming Loss for the 10 runs = 0.16696484

Standard deviation for Hamming Loss for the 10 runs = 0.00236803

References

- Deep Neural Networks