CS6320: Assignment 3

https://github.com/cs6320-501-fall22/assignment3-group18/tree/master

Group 18 Kevin Tak Hay Cheung (kxc220019) Humza Salman (mhs180007)

Introduction and Data

The project aims to implement the feed forward functionalities of a Feed Forward Neural Network and a Recurrent Neural Network. The purpose of these networks is to evaluate reviews and perform sentiment analysis to predict the 5-star rating of the review. We were provided preprocessed Yelp reviews that were already split into training, validation, and testing data. Additionally, we were provided the word embedding file for our RNN. We did not perform any further preprocessing steps to the data.

In our analysis of the data we learned the training data consisted of all output possibilities, however the same did not hold for validation or testing output possibilities. We decided to leave the validation and testing data as they were given to us and assumed it was meant to be as such. The breakdown of data is shown below.

Stars	Training Sample Count	Validation Sample Count	Testing Sample Count
1	3200	320	0
2	3200	320	0
3	3200	160	160
4	3200	0	320
5	3200	0	320
Totals	16000	800	800

FFNN

A feed forward neural network with one hidden layer is implemented in the FFNN.py. The script takes several arguments for the computation, including the number of neurons in the hidden layer, number of epochs, path to training and validation datasets. After taking in the arguments, the following steps will be executed in order:

Data Preprocessing

After loading the data, the script will create the vocabulary based on the training data - the vocabulary will be used to create the vocabulary vector for the document in the datasets. On top of that, two mapping dictionaries, *word2index* and *index2word*, will be created for mapping between words and their corresponding unique indices.

Next, both training and validation datasets will be passed to the function *convert_to_vector_representation*. For each document in the datasets, the function will count the number of occurrences of each word and store the count in the vocabulary vector. In the end, each document will be converted to a vector with dimension of 1 X vocabulary size. These vectors will be used as input for the neural network.

Model Training

In each epoch, the training data will be passed to the neural network in batches with a size of 16 documents. Each document in the batch will be passed to the network separately and the pass is achieved in the function *forward*. The function will return the softmax probabilities of each class.

We were tasked with implementing the feed forward functionalities. First, the function will receive the vocabulary vector (the bag-of-word representation of one document) as input argument. Next, we will compute the inner product between the weights of each neuron in the first hidden layer and the vocabulary vector. Thus, we will have a vector of the size of the hidden layer, which gives us the input of the hidden layer. Then, we use the activation function that was predefined to utilize the ReLU function to compute the first hidden layer representation. After that, we compute the output layer representation by applying a linear transformation to the hidden layer representation. Once we achieve this we pass the output to the softmax layer to compute the probability distribution for the predicted reviews.

```python
def forward(self, input_vector):

    # [to fill] obtain first hidden layer representation
    hidden_output = self.activation(self.W1(input_vector))
    # [to fill] obtain output layer representation
    output_output = self.W2(hidden_output)
    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(output_output.float())

    return predicted_vector
```

With the predicted_vector, which stores the predicted probability of each class for each document, we will compute the cross entropy loss by comparing the vector to the one-hot encoding of the gold label. After forward passing the entire batch (16 documents), the average cross entropy loss will be computed. The gradients of each weight will be computed in the *backward* function and the weight updates will be completed in the optimizer.step().

Model Validation

The vocabulary vectors in the validation datasets will be passed to the trained neural network. A predicted class will be chosen based on the predicted probability distribution from the model output, which will be used to compare with the gold label for accuracy calculation.

Debugging in FFNN

To better follow the feed forwarding in the network, we added the following codes into the script:

```python
if self.debug:
    print("===Input Vector===")
    print(input_vector)
    print("===Net Factor===")
    print(self.W1(input_vector))
    print("===Output Factor===")
    print(hidden_output)
    x = input("Press Anyway to continue...")
```

This code snippet allows us to keep track of how the vocabulary vectors (the input) get transformed to the output layer representation as well as the softmax predicted probability. Unfortunately, since the initialization of the weights are completed in a black box, we cannot verify the calculations manually.

RNN

A recurrent neural network is implemented in the rnn.py. After providing several modeling argument, including the number of neurons in the hidden layer, number of epochs, path to training and validation datasets, the following steps will be executed in order:

Data Preprocessing

The main difference between the RNN and FFNN implementations is that the RNN would take each word in the document as input, compared to a vocabulary vector as an overall representation of the entire document in FFNN. To convert the document to the input of the neural network, each word in the dataset will be converted to the corresponding word embedding. For example, a document with k word will be transformed to a vector of k subvectors, which has a dimension of d, the dimension of the word embedding.

Model Training

Similar to the feed forward neural network, the training data will be processed in a batch of 16 documents. We were tasked with implementing the feed forward functionalities.

First, we pass the input vector, which contains the word embeddings of the entire document, to the function *forward*. The *rnn* from Pytorch would compute the hidden state representation at each position of the document

and return two variables - *out*, *hidden*. *Out* contains the hidden state representation at each position and *hidden* contains the hidden layer representation of the final position. According to the assignment requirement, we need to pass each position's hidden layer representation to the final layer and sum up all representations as the score of each rating. We achieve this by applying the *self.W* linear transformation to the *out* variable. The linear transformation would return a vector with length of 5 for each position's hidden state representation. At the end, we sum up these representations and use softmax activation function to compute the probability distribution for the predicted reviews.

```
def forward(self, inputs):
    # [to fill] obtain hidden layer representation (https://pytorch.org/docs/stable/generated/torch.nn.RNN.html)
    out,  hidden = self.rnn(inputs)
    # [to fill] obtain output layer representations
    output = self.W(out)
    # [to fill] sum over output
    output = output.sum(dim = 0)
    # [to fill] obtain probability dist.
    predicted_vector = self.softmax(output)

    return predicted_vector
```

After forward passing the entire batch, the average loss is calculated and will be used for back propagation, during which the weight updates will be completed.

Model Validation

The vocabulary vectors in the validation datasets will be passed to the trained neural network. The predicted probability vector from the model output will be compared with the gold labels to compute the accuracy.

Debugging in RNN

We added the codes similar to the ones in the FFNN section for debugging purposes. The code allows us to keep track of the dimension and content of the vectors or matrices used in the process and make sure they are transformed correctly. It is shown below:

```
def forward(self, inputs):

    if self.debug:
        print(inputs)
        print(inputs.size())
        x = input("CHECK INPUT")

    # [to fill] obtain hidden layer representation (https://pytorch.org/docs/stable/generated/torch.nn.RNN.html)
    out,  hidden = self.rnn(inputs)
    if self.debug:
        print(out)
        print(hidden)
        print(out.size(), hidden.size())
        x = input("CHECK FORWARD OUTPUT")

    # [to fill] obtain output layer representations
    output = self.W(out)
    if self.debug:
        print(output)
        print(output.size())
        x = input("CHECK OUTPUT LAYER")

    # [to fill] sum over output
    output = output.sum(dim = 0)
    if self.debug:
        print(output)
        print(output.size())
        x = input("CHECK OUTPUT SUM")

    # [to fill] obtain probability dist.

    predicted_vector = self.softmax(output)
    if self.debug:
        print(predicted_vector)
        print(predicted_vector.size())
        x = input("CHECK SOFTMAX")

    return predicted_vector
```

Experiments and Results

Evaluations

After training the models, we will pass the testing dataset into each model and measure their performances. The evaluation will be based on overall accuracy - the ratio of correct prediction to testing set size.

Results

We have tried different hyperparameters for both FFNN and RNN and their performances are summarized in the following table.

Model	Epoch	Hidden Layer dimension	Training Accuracy	Validation Accuracy	Test Accuracy
FFNN	1	10	39%	32%	53%
FFNN	25	32	91%	49%	52%
FFNN	25	64	96%	57%	48%
RNN	10	32	36%	36%	34%
RNN	25	64	37%	38%	35%
RNN	25	128	28%	22%	38%

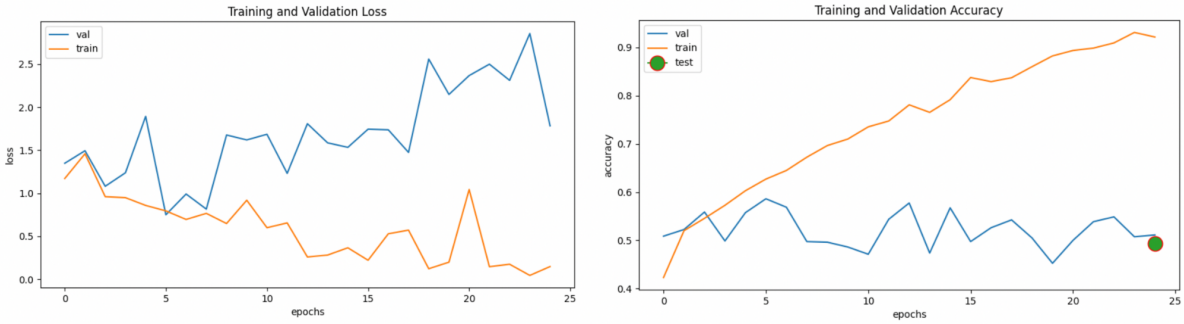
Our best performing model is the FFNN with 32 neurons in the hidden layer. The model has a 91% training accuracy, 49% validation accuracy, and 52% test accuracy. It seems that the model overfitted a lot since there is a huge difference between the training and validation accuracy. Another explanation could be the odd distribution of data between training, validation, and test sets. We may want to decrease the number of epochs as well as the hidden layer dimension for better performance.

In general, the FFNN performs better than the RNN, which may seem counter-intuitive. However, a normal text classification task with RNN should only take the last hidden layer as the input of the softmax activation because the hidden layer should contain every information from the previous states. However, in our assignment, we include every hidden layer, while each hidden layer contains previous information. This architecture results in the duplicate knowledge of the earlier position of the document, which could be considered as noise in the prediction. Therefore, it makes sense that the performance is much worse than the simple FFNN.

Analysis

Learning Curve of best model - training loss and dev set accuracy by epoch

The following charts show the loss and accuracy for the training and development sets for our best performing model - Feed Forward Neural Network with one 32-neuron hidden layer.



As we can see in the graphs, the training and validation accuracies get wider as epoch increases. It is a strong signal that the model overfits. A failsafe conditional branch to prevent overfitting (like the idea used in the RNN script) should be implemented in the FFNN script, and we should be able to yield a better result.

Error Analysis - list error examples and provide analysis - how to improve the system?

Odd Data Distribution: One reason why validation and test set performance did not perform as well as training could be due to the validation and test set data containing only 1-3 star reviews and 3-5 star reviews respectively. This would cause the model to be validated on only 1-3 star reviews and not learn as well from 3-5 star reviews.

FFNN 32 hidden_dim and 25 epochs: Looking at the test predictions for our best model we see the following distribution given in the table. We know based on the data distribution that the dev set presented an accurate representation for 3-star reviews, however the same was not done for 4 or 5 star reviews. Still, we see that the 5-star reviews had a 63% accuracy rate, however 4-star reviews struggled at a 32% accuracy rate. If our dev set contained an accurate representation of the test set we would have seen better results for 4-star reviews.

True Label	Correct	Wrong	Accuracy
3	87	73	0.54375
4	104	216	0.325
5	204	116	0.6375

RNN 32 hidden_dim and 10 epochs: Considering this is our best RNN model, we find the following distribution of predictions in the data. Given that we saw the 3-star review data in our dev set our model performed really well on the test-set. However, it performed extremely poorly on 4 and 5 star reviews given that we achieved accuracies of 0% and 1%, respectively. Attempting to fix the flawed data distribution would be a good next step so we ensure that our dev set is representative of the test set.

True Label	Correct	Wrong	Accuracy
3	151	9	0.94375
4	0	320	0
5	5	315	0.015625

Misclassified examples:

- *{"text": "I want to give props to Josh at this location. He's very much invested in customer service and would like to thank him for making my experience a good one!", "stars": 4.0}* - Taking this example into account we see there are only positive remarks that are made, yet only a 4-star review is given. Since it is super positive the models may have thought of it to be a 5-star review instead.
- *{"text": "The food is good just wasn't happy that it took over an hour for us to not even get our delivery.", "stars": 3.0}* - We can see that there are two aspects to this example which indicate a positive note and a negative note. The FFNN model would consider all words on their own whereas the RNN model would have considered the sequence in the order it is read which would lead to better predictions.

Features to include for system improvement

It is known that the RNN should perform better than a simple Bag-of-words FFNN, which is not observed in our experiments. As a result, we identified several areas that can improve our current RNN model:

- As mentioned above, the text classification should only rely on the final hidden layer. The current structure would provide so much duplicate information to the classifier, which results in a poor model
- Vanishing gradient is an important issue when it comes to RNN. A LSTM structure is preferred over a simple RNN. It allows the model output to rely on a much longer span of the text.
- We should be training the word embedding on the fly, such as using the ELMo technique. The current method relies on the word embedding dictionaries. It would cause semantic problems when the word has more than one meaning.

Conclusion

In conclusion, we saw that our best model was the FFNN model which performed with an accuracy of 52% on the test set. Going forward we could involve more data from different review-based applications such as Google Reviews to better generalize our predictions for how people rate places. Additionally, we could have also adjusted

the distribution of data in our validation and test sets to be representative of all output possibilities. We saw that there are benefits to changing certain hyperparameters.

Contributions of Each Member

Kevin: FFNN (50%), RNN(60%), Evaluation(40%), Report(50%)
Humza: FFNN (50%), RNN(40%), Evaluation(60%), Report(50%)

Feedback of Project

- We like how the assignment was straightforward and simple to do, however it did not feel challenging and did not promote our growth. Rather, it felt as though it could be done with a tutorial for creating a neural network. A lot of design decisions were already pre-determined and following a cookie-cutter method to fill in the feed forward functionality was not ideal for developing our understanding of neural networks.
- We like the report template provided as we use it as a guideline. Overall, a good project.