CS6375 Assignment 1

<https://github.com/sahithireddy16/SXM220353_CS6375_assignment1.git>

SAHITHI REDDY MOPURU

SXM220353

**1. Introduction and Data**

In this project, we implemented two types of neural networks, a Feedforward Neural Network (FFNN) and a Recurrent Neural Network (RNN), to perform a 5-class sentiment analysis task using a dataset of Yelp reviews. The primary goal of this task is to predict the sentiment rating (on a scale from 1 to 5) of each review based on the review text. Each review is preprocessed and then passed into the models for sentiment classification.

**Feedforward Neural Network (FFNN):** The FFNN processes review text using a bag-of-words approach to convert the text into a fixed-length vector representation. The network consists of an input layer that receives the vectorized review, followed by a hidden layer that applies a non-linear activation function (ReLU). The final layer is an output layer that produces a probability distribution over the five sentiment classes, using the softmax function for sentiment classification.

**Recurrent Neural Network (RNN):** The RNN utilizes word embeddings to represent the review text as a sequence of vectors. It processes this sequence through GRU (Gated Recurrent Unit) layers, which capture temporal dependencies and relationships between words across the sequence. This allows the model to generate context-aware predictions for the sentiment by summing up the hidden representations across all time steps and using them in the final output layer for classification.

The dataset consists of three sets: training, validation, and test sets, with the following statistics:

| **Dataset** | **Number of Entries** |
| --- | --- |
| Training | 16,000 |
| Validation | 800 |
| Test | 800 |

* The **training set** is used to fit the model parameters.
* The **validation set** is used to fine-tune hyperparameters and prevent overfitting.
* The **test set** is used for evaluating the final performance of the models.

In this report, we describe the implementations of both the FFNN and RNN models, followed by an analysis of their performance on the sentiment analysis task. We experimented with multiple hyperparameter settings to optimize the models, and our evaluation is based on accuracy metrics for the validation and test sets.

**2. Implementations (45pt)**

**2.1 Feedforward Neural Network (FFNN) (20pt)**

* The **Feedforward Neural Network** (FFNN) is a simple neural model where the input is a fixed-length vector representing a review (using a bag-of-words approach), and the task is to classify the sentiment of the review.
* I implemented the **forward() function** by first computing the hidden layer representation, passing it through a non-linear activation function (ReLU), and then computing the output layer. The output is passed through a **softmax layer** to obtain a probability distribution over the five sentiment classes. Here is a snippet of the

implemented forward() function:

**A screen shot of a computer code

Description automatically generated**

**Explanation:**

* **Input Layer:** The input is a vector of size equal to the vocabulary, where each dimension represents the occurrence of a word in the review.
* **Hidden Layer:** The input vector is passed through a linear transformation (W1) followed by a ReLU activation. The hidden layer size is determined by the hidden\_dim parameter.
* **Output Layer:** The hidden representation is passed through another linear transformation (W2), which produces a vector of size 5 (corresponding to the five sentiment classes).
* **Softmax**: This is applied to the output to convert the raw scores into probabilities.

The training process uses the negative log-likelihood loss function (NLLLoss), and the SGD optimizer with a learning rate of 0.01 and momentum of 0.9. The training accuracy is calculated after each epoch, and we also validate the model using the validation set.

**2.2 Recurrent Neural Network (RNN) (25pt)**

The **RNN** is designed for sequential data, like a sequence of words in a review, processing the input one-time step at a time while updating its hidden state. For this task, pre-trained word embeddings are used to represent the review text as sequences of vectors, which are then fed into a single-layer RNN with a hidden state.

The **forward () function** processes the input sequence, applies the RNN layer, and computes a probability distribution over sentiment classes. Below is the implementation of the forward () function:

A screen shot of a computer code

Description automatically generated

**Explanation:**

* **Input Sequence:** The input to the RNN is a sequence of word embeddings, where each word in the review is mapped to a pre-trained embedding vector (e.g., GloVe, Word2Vec).
* **RNN Layer:** The input is processed through an RNN, which computes a hidden state at each time step. The rnn\_out contains the hidden states for all time steps, and hidden contains the final hidden state.
* **Output Layer:** We use the hidden state from the last time step to make the final prediction. This is passed through a fully connected layer (self.W) to transform it into a 5-dimensional output (one for each sentiment class).
* **SoftMax:** The output is passed through a SoftMax function to convert the raw scores into a probability distribution over the sentiment classes.

**Training Process:**

* The model uses negative log-likelihood loss (NLLLoss), and training is performed using the Adam optimizer with a learning rate of 0.01.
* The training loop processes mini-batches of 16 samples, and the word embeddings for each review are converted into sequences of vectors. The RNN updates its hidden state at each time step based on the word sequence, and the final hidden state is used to classify the sentiment.

The training stops early when the validation accuracy begins to decline, preventing overfitting.

The RNN model captures the sequential nature of the text, making it more suitable than FFNN for tasks where word order and context are important for prediction.

**3. Experiments and Results (45pt)**

**Evaluations (15pt)**

To evaluate the performance of the FFNN and RNN models on sentiment analysis, accuracy is used as the primary metric. It measures the proportion of correctly predicted sentiment classes out of the total predictions, calculated as:

A close up of a text

Description automatically generated

**Training Accuracy**

During each epoch, the model's performance on the training data is assessed. For each mini-batch, the predicted label is compared to the ground truth, and the number of correct predictions is counted. The training accuracy is then calculated as the ratio of correct predictions to total predictions, helping to ensure the model is effectively learning the task.

**Validation Accuracy**

After each training epoch, the model's performance is evaluated on the validation set to assess how well it generalizes to unseen data. High validation accuracy indicates that the model is not overfitting to the training data and is effectively generalizing to new examples.

**Test Accuracy (Final Evaluation)**

After training and validation, the best-performing model is evaluated on the test set, which consists of entirely unseen data. This step provides an unbiased estimate of the model's real-world performance. The test data is loaded using the `load\_data()` function, similar to the training and validation data. The selected model, based on the highest validation accuracy, is then used to compute the test accuracy, which serves as the final measure of the model's performance and is reported in the results section.

**Early Stopping Criterion**

Early stopping was implemented to prevent overfitting during training. If validation accuracy does not improve or starts to decline over several consecutive epochs, training is halted. This ensures the model does not over-optimize for the training data at the expense of generalization. The stopping condition is triggered when validation accuracy drops while training accuracy continues to rise.

**Cross-Entropy Loss**

Along with accuracy, the loss function is monitored during training and validation. Both models use negative log-likelihood loss (NLLLoss), suitable for multi-class classification tasks. The goal is to minimize this loss, as a lower loss generally indicates better classification accuracy.

Tracking both accuracy and loss across training, validation, and testing helps ensure that the models are learning effectively and not overfitting.

**3.2 Results (30pt)**

In this section, we summarize the performance of the different models we trained, including two versions of the Feedforward Neural Network (FFNN) and Recurrent Neural Network (RNN). We also describe the variations in the model architectures and the impact of these changes on model performance.

Feedforward Neural Network (FFNN)

**Model 1: Regular FFNN(FFNN)**

The first FFNN model follows a standard architecture, consisting of:

* A single hidden layer with a ReLU activation function.
* The output layer consists of 5 units, corresponding to the 5 sentiment classes.
* No dropout or additional regularization techniques were applied.
* The model was trained using the SGD optimizer with momentum and a learning rate of 0.01.

This model provided a decent baseline for sentiment classification, but there were signs of overfitting due to the lack of regularization techniques, as seen by the increasing difference between training and validation accuracy over epochs.

**FFNN With epochs 10, hidden dim : 64**

A screen shot of a computer

Description automatically generatedA screen shot of a graph

Description automatically generated

Test accuracy: 0.57625

**Model 2: FFNN with Dropout and L2 Regularization(FFNN1)**

The second FFNN model (Model 2) incorporated several important modifications designed to improve generalization and reduce overfitting:

**Additional Hidden Layer:**We added a second hidden layer (W2), allowing the model to capture more complex relationships in the data.

**Dropout Regularization:**A dropout layer was added after each hidden layer, with a dropout rate of 50%. This technique randomly drops neurons during training, preventing the model from relying too much on any specific neurons and thus helping with overfitting.

**L2 Regularization (Weight Decay):**L2 regularization was applied to penalize large weights, further promoting generalization. The regularization strength was set to 0..01.

These changes significantly improved the model's performance on the validation and test sets by reducing overfitting and improving generalization. Below is a comparison of the results between Model 1 and Model 2 for FFNN:

**FFNN With epochs 10, hidden dim : 64**

**A screen shot of a graph

Description automatically generatedA screen shot of a graph

Description automatically generated**

| **Epoch** | **Training Accuracy** | **Validation Accuracy** | **Training Loss** | **Validation Loss** |
| --- | --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **0.3618** | **0.4150** | **1.4671** | **1.5164** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **2** | **0.4529** | **0.4313** | **1.4653** | **1.2255** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **3** | **0.4914** | **0.4488** | **1.3842** | **1.6576** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **4** | **0.5169** | **0.5075** | **1.0925** | **1.2254** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5** | **0.5371** | **0.4925** | **1.0380** | **1.1889** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **6** | **0.5487** | **0.5400** | **1.0298** | **1.3831** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **7** | **0.5603** | **0.5088** | **1.0948** | **1.1810** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **8** | **0.5721** | **0.4988** | **1.1995** | **1.4539** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **9** | **0.5833** | **0.5538** | **0.9825** | **1.2768** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **10** | **0.5901** | **0.5775** | **1.1020** | **1.3086** |

**Observations**

**Model 1 (Regular FFNN):**

* The first FFNN model, while achieving reasonable training accuracy, showed signs of overfitting as the gap between training and validation accuracy increased after a few epochs.
* The test accuracy was also lower, indicating that the model struggled to generalize to unseen data, likely due to the lack of regularization techniques.

**Model 2 (FFNN with Dropout and L2 Regularization):**

* The second FFNN model outperformed the first one on all metrics (training, validation, and test accuracy).
* The introduction of dropout and L2 regularization helped to reduce overfitting significantly. The model's validation and test accuracy were more consistent with its training accuracy.

**Impact of Hyperparameters:**

* Increasing the hidden layers and batch size provided more stable gradient updates during training, while dropout and L2 regularization helped the model generalize better to the validation and test sets.
* These changes highlight the importance of regularization in neural networks, especially when dealing with relatively small datasets like this one.

**Recurrent Neural Network (RNN)**

**Model 1: Simple RNN**

The first RNN model used a basic architecture and followed these specifications:

**Architecture:** A single RNN layer with 64 hidden units and a tanh activation function.

**Input Representation:** The input to the model is a sequence of word embeddings, one for each word in the review.

**Output Layer**: The hidden state from the last time step is passed through a fully connected layer to produce logits for the five sentiment classes.

**Optimizer:** The model was trained using the Adam optimizer with a learning rate of 0.01.

**No Regularization:** No dropout or other regularization techniques were applied.

This model provided a reasonable baseline for sentiment classification, but it exhibited some overfitting as training accuracy continued to rise, while validation accuracy plateaued and began to decline.

**Rnn model 1 epchos:10, Hid dim: 64**

A graph of a graph

Description automatically generated with medium confidenceA graph with blue and orange lines

Description automatically generated

**Model 2: RNN with Dropout and L2 Regularization**

The second RNN model (Model 2) incorporated several improvements to reduce overfitting and enhance generalization:

**Increased Hidden Units:** The number of hidden units was increased to 64, which allowed the model to capture more complex patterns within the input text.

**Dropout Regularization:** A dropout layer with a 50% dropout rate was added after the RNN layer. This helped prevent the model from relying too heavily on specific neurons and improved generalization.

**L2 Regularization (Weight Decay):** L2 regularization was applied with a strength of 1e-4 to penalize large weights and improve model generalization.

**Optimizer:** The SGD optimizer was used again with the same learning rate of 0.01, but this time, the presence of regularization helped the model converge more stably.

**Early Stopping:** Early stopping was implemented to avoid overfitting by stopping training when validation accuracy ceased to improve for several consecutive epochs.

The introduction of regularization techniques and additional hidden units in Model 2 significantly improved the model’s performance on both the validation and test sets, reducing overfitting and improving generalization.

Below is a comparison of the results between Model 1 and Model 2 for RNN:

**Rnn model 2 epochs :10 , hid dim 64**

A graph with blue and orange lines

Description automatically generatedA graph with blue and orange lines

Description automatically generated

| **Epoch** | **Training Accuracy** | **Validation Accuracy** | **Training Loss** | **Validation Loss** |
| --- | --- | --- | --- | --- |
| 1 | 0.2726 | 0.3513 | 1.5587 | 1.4618 |
| 2 | 0.3481 | 0.4075 | 1.4502 | 1.3398 |
| 3 | 0.3661 | 0.4338 | 1.4287 | 1.3025 |
| 4 | 0.3801 | 0.4100 | 1.4077 | 1.3365 |
| 5 | 0.3887 | 0.4163 | 1.3948 | 1.3254 |
| 6 | 0.3942 | 0.4062 | 1.3859 | 1.3438 |
| 7 | 0.3942 | 0.4612 | 1.3895 | 1.2209 |
| 8 | 0.4042 | 0.4062 | 1.3733 | 1.3471 |
| 9 | 0.4039 | 0.4725 | 1.3762 | 1.2230 |
| 10 | 0.4032 | 0.4612 | 1.3696 | 1.2002 |

Test accuracy: 0.3638

**Observations**

**Model 1 (Simple RNN):**

The first RNN model, while showing decent training accuracy, began to overfit as the validation accuracy stagnated. The test accuracy was also lower, indicating that the model struggled to generalize to unseen data, likely due to the absence of regularization techniques.

**Model 2 (RNN with Dropout and L2 Regularization):**

The second RNN model outperformed the first one on all metrics. The introduction of dropout and L2 regularization helped reduce overfitting significantly. Both validation and test accuracy were more consistent with the training accuracy, indicating better generalization.

The increase in hidden units allowed the model to capture more complex relationships in the input data, and the regularization techniques provided stability during training.

**Impact of Hyperparameters:**

* Increasing the number of hidden units improved the model's capacity to learn from the text. The addition of dropout and L2 regularization helped prevent the model from overfitting, resulting in better performance on the validation and test sets.
* Switching to a larger batch size (32) helped stabilize the training process and led to smoother convergence.

**4. Analysis (Bonus: 10pt)**

**Learning Curve of the Best System (5pt)**

To demonstrate the performance of the best models (FFNN Model 2 and RNN Model 2 with dropout and L2 regularization), we plotted the learning curve showing training loss and validation accuracy across epochs. This illustrates how the models' performance evolves over time and helps assess overfitting or underfitting.

The plot includes:

Training Loss: Shows how the model's loss decreases during training, indicating parameter optimization.

Validation Accuracy: Tracks the model's generalization to unseen data, ideally improving or stabilizing over epochs.

**FFNN**

A screen shot of a graph

Description automatically generated

**RNN**

A graph on a computer screen

Description automatically generated

The plot confirms that the training loss consistently decreases while validation accuracy improves, indicating the models are not overfitting.

**Error Analysis (5pt)**

During error analysis, we examined several examples where the best-performing model (RNN Model 2) failed to correctly predict the sentiment.

Imbalanced Training Data: The model struggled with high-rated reviews due to a lack of such examples in the training set, leading to biased predictions towards lower ratings.

*Improvement:* Balance the dataset by adding more positive reviews or using data augmentation

**5. Conclusion and Others (5pt)**

**Individual Member Contribution**

As a solo contributor, I handled all aspects, including:

FFNN Implementation: Developed the forward pass, trained the model, and experimented with hyperparameters.

RNN Implementation: Implemented the forward pass for both simple and enhanced RNNs with dropout and L2 regularization, conducted training, testing, and performance analysis.

**Error Analysis & Learning Curves:**Analyzed errors in the best RNN model and plotted learning curves for training loss and validation accuracy.

**Report Writing:**Independently documented the implementation, experiments, results, and conclusions.

**Feedback for the Assignment**

**Time Spent:** About 20 hours were spent on coding, experiments, and writing the report.

**Difficulty:** Challenging but manageable, with hyperparameter tuning and controlling overfitting being the most demanding tasks. Working with PyTorch was insightful, especially in handling RNN sequences and using pre-trained embeddings. Regularization helped improve model generalization on unseen data.

**Suggestions for Improvement:**

**Advanced RNN Models:** Including tasks with more advanced architectures like LSTMs or bidirectional RNNs could offer deeper insights into sequence modeling.

**Error Analysis:** Encouraging detailed error analysis would aid in identifying prediction challenges and potential model improvements.

Overall, this assignment provided valuable hands-on experience in neural networks for sentiment analysis, enhancing practical skills in model implementation, experimentation, and evaluation in NLP.