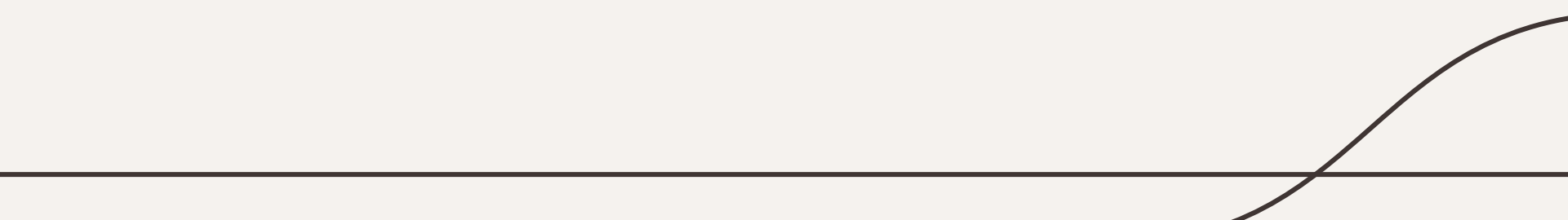




Sentence-Level Categorization: Question Classification

SC4002 Natural Language Processing

Aloysius Ang



Task

- Conduct sentence-level classification using the task of Question Classification
 - Categorize each question into one of the topics from a predefined label set
 - There are 5 classes in total

label-coarse	label-fine	text
4	40	How far is it from Denver to Aspen ?
5	21	What county is Modesto , California in ?
3	12	Who was Galileo ?
0	7	What is an atom ?
4	8	When did Hawaii become a state ?
4	40	How tall is the Sears Building ?
3	5	George Bush purchased a small interest in which baseball team ?
1	30	What is Australia 's national flower ?
0	9	Why does the moon turn orange ?
0	7	What is autism ?

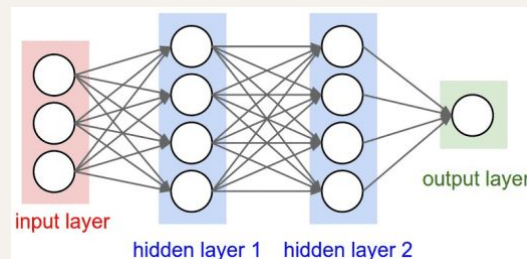
Data Preparation

1. Convert words to vectors using pre trained word2vec embeddings
 - a. Replace out of vocab words with 0 vectors. This means we will skip the words
2. Aggregate the word vectors to represent a sentence
 - a. Possible methods are max pooling, averaging over word representations (mean) and taking the representation of the last word
 - b. Decided to take the mean of the word vectors as taking the mean gives a better representation of the sentence as a whole
3. Padding sentences with 0 vectors so that each sentence will have the same number of words
 - a. This is to ensure equal dimensions to prevent errors during model training

```
array([[ -8.34960938e-02,  9.61914062e-02,  2.02148438e-01,
         1.32812500e-01,  1.92382812e-01, -9.46044922e-03,
         1.47460938e-01,  8.20312500e-02, -1.82617188e-01,
        -9.96093750e-02, -5.10253906e-02, -1.06445312e-01,
         1.40625000e-01, -6.39648438e-02, -2.24609375e-01,
         2.12890625e-01,  2.63671875e-02, -1.18164062e-01,
         8.49609375e-02, -1.13769531e-01,  1.38671875e-01,
         2.53906250e-01, -5.51757812e-02,  1.59179688e-01,
        -9.86328125e-02, -7.76367188e-02, -2.38281250e-01,
         1.53320312e-01, -5.51757812e-02, -3.56445312e-02,
        -2.15820312e-01, -2.67578125e-01, -4.17480469e-02,
        -1.51367188e-01,  2.89062500e-01, -2.58789062e-02,
        -4.10156250e-02, -2.89062500e-01,  2.13867188e-01,
         1.76239014e-03, -8.88671875e-02,  1.35498047e-02,
         5.73730469e-02,  7.22656250e-02, -5.00488281e-02,
        -1.61132812e-01, -3.10058594e-02, -2.39257812e-02,
        -1.14257812e-01,  1.23046875e-01, -8.34960938e-02,
        -1.55273438e-01, -1.14135742e-02,  6.98242188e-02,
         2.40234375e-01, -1.54296875e-01, -1.39648438e-01,
        -3.34472656e-02, -1.44042969e-02, -1.24511719e-01,
        -1.93023682e-03, -6.01196289e-03, -7.27539062e-02,
         7.81250000e-03,  9.52148438e-02, -2.07519531e-02,
         5.71289062e-02, -6.03027344e-02,  6.49414062e-02,
        -5.73730469e-02, -2.09960938e-01, -7.91015625e-02,
        -7.27539062e-02,  7.56835938e-02,  2.08007812e-01,
        ...,
        -2.19726562e-01,  6.39648438e-02,  1.66015625e-02,
         4.56542969e-02,  3.26171875e-01, -3.80859375e-01,
         1.70898438e-01,  5.66406250e-02, -1.04492188e-01,
         1.38671875e-01, -1.57226562e-01,  3.23486328e-03,
        -4.80957031e-02, -2.48046875e-01, -6.20117188e-02]])
```

Model Training: Feed Forward Neural Network

- Model Description:
 - Layers: Build models with 2 and 4 layers
 - Models with more than 4 layers requires a large amount of time to train
 - Batch Size: 128
 - Input Size: 300 (size of word embeddings)
 - Hidden layer size: 256
 - Number of classes: 5 (number of labels)
 - Number of epochs: 100
 - ReLu Activation Function with Softmax Activation
 - Adam Optimizer with learning rate 0.001
 - Early stopping when validation loss < 0.95
 - Mini Batch Gradient Descent
 - Model parameters are updated after each mini batch
 - Cross entropy loss: used for classification tasks
- Best model: The model with 4 hidden layers and the above parameters



```
class FeedForwardNet(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, hidden_size3, hidden_size4, num_classes):
        super(FeedForwardNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, hidden_size3)
        self.relu3 = nn.ReLU()
        self.fc4 = nn.Linear(hidden_size3, hidden_size4)
        self.relu4 = nn.ReLU()
        self.fc5 = nn.Linear(hidden_size4, num_classes)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        out = self.relu3(out)

        # Last hidden layer (layer 4)
        out = self.fc4(out)
        out = self.relu4(out)

        # Perform mean aggregation along the sequence dimension (dim=1)
        out = torch.mean(out, dim=1, keepdim=False)

        # Output layer
        out = self.fc5(out)
        out = self.softmax(out)
        return out
```

Explanation for the Model Parameters

4 Hidden Layers

- Hidden layers increase the network's capacity to learn complex representations
- A model with the right number of hidden layers can generalize well to data, leading to good validation and test accuracy
- However, too many layers can lead to vanishing and exploding gradients due to the repeated multiplication of gradients during back propagation
- Many layers also require more computational resources. This increases the time and space complexity

Hidden Layer Size 256

- More neurons in the hidden layers increase the ability to learn complex patterns, leading to higher accuracy. However, too much may lead to overfitting while too little may lead to underfitting
 - Larger networks can worsen the issue of vanishing or exploding gradients
 - Larger hidden layers can lead to slower trainings and require more computational resources
-

Explanation for the Model Parameters

Batch Size 128

- A small batch size may introduce more noise into the gradient estimates, which acts as a regularization that helps the model avoid overfitting. A large batch size however, may cause the model to converge too quickly to the sharper minima of the loss landscape, leading to worse generalization
- The noisy gradient estimates associated with small batch sizes also leads to more frequent updates and faster convergence in Mini-Batch Gradient Descent

Mini-Batch Gradient Descent

- Mini-batch gradient descent is used as it strikes a balance between batch gradient descent and stochastic gradient descent
- Batch gradient descent updates model parameters after computing the gradient on the entire training dataset. Very slow on large datasets, high memory requirements and inefficient use of resources
- Stochastic gradient descent (SGD) updates model parameters for each training data. Highly noisy updates can make convergence slower and training unstable
- Mini-Batch updates model parameters after computing the gradient on a subset of the training data. This strikes a balance between the noisy updates of SGD and the smooth updates of batch gradient descent

Explanation for the Model Parameters

ReLU Activation Function and Softmax

$$\text{ReLU}(x) = \max(0, x)$$

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- ReLu activation is used in the hidden layers to introduce non-linearity, which aids the network in learning complex relationships. ReLu helps to avoid vanishing gradient problems as well by avoiding saturation (where gradients become very small)
- Non linearity allows the neural network to model complex patterns that are not linearly separable by approximating any complex functions. This is essential in natural language processing
- Softmax is used in the output layer of classification models to produce a probability distribution over multiple classes. It transforms the scores into a probability distribution over multiple classes
- Softmax ensures that the sum of probabilities across all classes is equal to one, enabling the model to output probabilities for each class.

Cross Entropy Loss

- Measures the performance of a classification model whose output is a probability value between 0 and 1
- Specifically designed for classification tasks where the output is a probability distribution over classes

Explanation for the Model Parameters

Adam Optimizer with Learning Rate 0.001

- Adam optimizer uses adaptive learning rates for each parameter, leading to efficient training and better performance
- Uses moving average of the gradient and squared gradient to provide momentum, helping the optimizer to accelerate in directions with consistent gradients and dampen oscillations
- Learning Rate is a hyperparameter that controls the size of the steps the optimizer takes during gradient descent
 - Step Size: It determines how quickly or slowly a model learns. A larger learning rate might speed up training but can overshoot the optimal solution. A smaller learning rate may lead to more precise convergence but can be slow and potentially get stuck in local minima

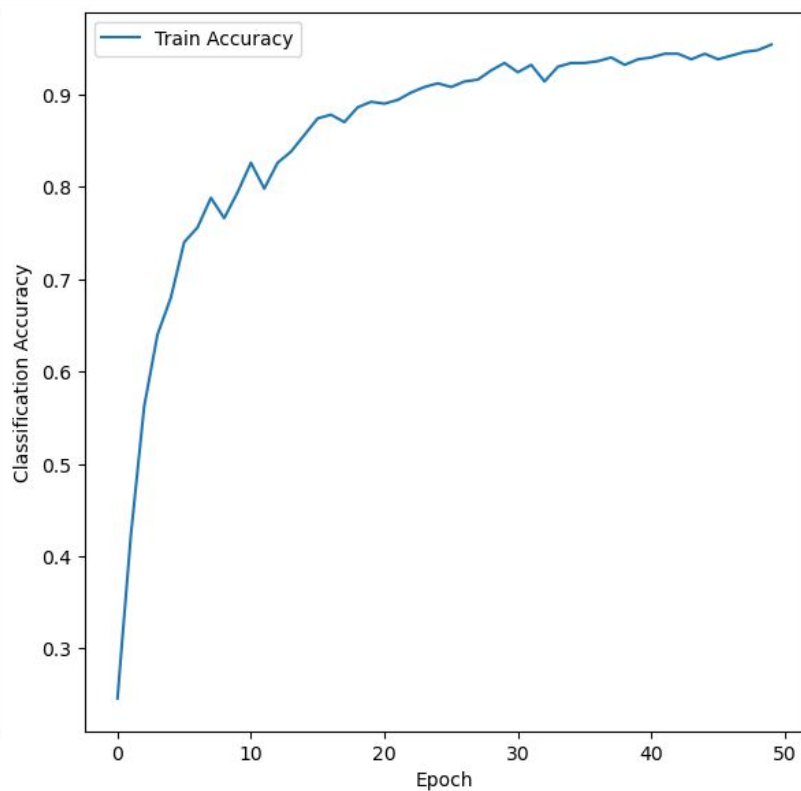
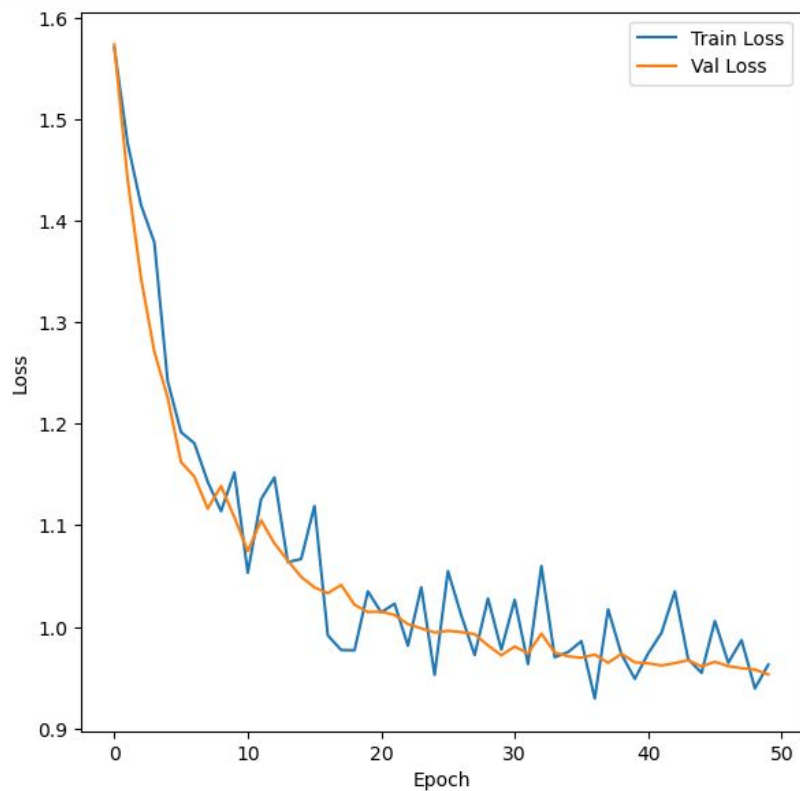
Early Stoppage with Validation Loss < 0.95 and patience 10

- Early stoppage is implemented to prevent overfitting
- Training will stop once validation loss is below 0.95 or if there is not much improvements for 10 epochs
- Validation loss is the measure of the model's error on the validation set. A lower validation loss indicates better model performance

Results

```
Epoch 1/100, Train Loss: 0.0000, Val Loss: 1.5738, Val Accuracy: 24.60%, F1 score: 0.10
Epoch 2/100, Train Loss: 1.5707, Val Loss: 1.4409, Val Accuracy: 42.20%, F1 score: 0.22
Epoch 3/100, Train Loss: 1.5236, Val Loss: 1.3438, Val Accuracy: 56.20%, F1 score: 0.34
Epoch 4/100, Train Loss: 1.4877, Val Loss: 1.2713, Val Accuracy: 64.00%, F1 score: 0.42
Epoch 5/100, Train Loss: 1.4605, Val Loss: 1.2259, Val Accuracy: 68.00%, F1 score: 0.48
Epoch 6/100, Train Loss: 1.4168, Val Loss: 1.1623, Val Accuracy: 74.00%, F1 score: 0.52
Epoch 7/100, Train Loss: 1.3793, Val Loss: 1.1481, Val Accuracy: 75.60%, F1 score: 0.56
Epoch 8/100, Train Loss: 1.3510, Val Loss: 1.1165, Val Accuracy: 78.80%, F1 score: 0.59
Epoch 9/100, Train Loss: 1.3250, Val Loss: 1.1387, Val Accuracy: 76.60%, F1 score: 0.61
Epoch 10/100, Train Loss: 1.3015, Val Loss: 1.1079, Val Accuracy: 79.40%, F1 score: 0.63
Epoch 11/100, Train Loss: 1.2866, Val Loss: 1.0746, Val Accuracy: 82.60%, F1 score: 0.65
Epoch 12/100, Train Loss: 1.2654, Val Loss: 1.1050, Val Accuracy: 79.80%, F1 score: 0.66
Epoch 13/100, Train Loss: 1.2538, Val Loss: 1.0822, Val Accuracy: 82.60%, F1 score: 0.68
Epoch 14/100, Train Loss: 1.2456, Val Loss: 1.0652, Val Accuracy: 83.80%, F1 score: 0.69
Epoch 15/100, Train Loss: 1.2326, Val Loss: 1.0493, Val Accuracy: 85.60%, F1 score: 0.70
Epoch 16/100, Train Loss: 1.2215, Val Loss: 1.0388, Val Accuracy: 87.40%, F1 score: 0.71
Epoch 17/100, Train Loss: 1.2151, Val Loss: 1.0332, Val Accuracy: 87.80%, F1 score: 0.72
Epoch 18/100, Train Loss: 1.2020, Val Loss: 1.0413, Val Accuracy: 87.00%, F1 score: 0.73
Epoch 19/100, Train Loss: 1.1895, Val Loss: 1.0218, Val Accuracy: 88.60%, F1 score: 0.74
Epoch 20/100, Train Loss: 1.1783, Val Loss: 1.0146, Val Accuracy: 89.20%, F1 score: 0.75
Epoch 21/100, Train Loss: 1.1711, Val Loss: 1.0150, Val Accuracy: 89.00%, F1 score: 0.75
Epoch 22/100, Train Loss: 1.1637, Val Loss: 1.0117, Val Accuracy: 89.40%, F1 score: 0.76
Epoch 23/100, Train Loss: 1.1573, Val Loss: 1.0027, Val Accuracy: 90.20%, F1 score: 0.77
Epoch 24/100, Train Loss: 1.1496, Val Loss: 0.9985, Val Accuracy: 90.80%, F1 score: 0.77
Epoch 25/100, Train Loss: 1.1450, Val Loss: 0.9945, Val Accuracy: 91.20%, F1 score: 0.78
...
Epoch 48/100, Train Loss: 1.0686, Val Loss: 0.9593, Val Accuracy: 94.60%, F1 score: 0.85
Epoch 49/100, Train Loss: 1.0669, Val Loss: 0.9579, Val Accuracy: 94.80%, F1 score: 0.85
Epoch 50/100, Train Loss: 1.0643, Val Loss: 0.9530, Val Accuracy: 95.40%, F1 score: 0.86
Early stopping at epoch 50 with val loss 0.9530
```

Results



Results

Test Accuracy: 73.8%

Model 2: Long Short-Term Memory (LSTM)

How LSTM Works

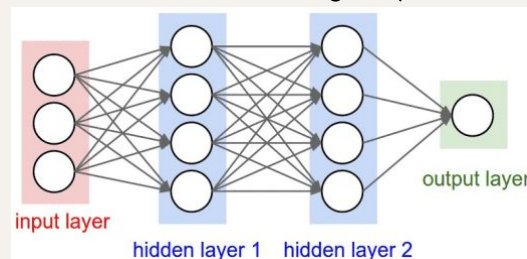
- Consists of a series of memory cells containing gates: Forget gate, Input gate and Output gate
 - Gates control the flow of information and help the LSTM maintain and update its memory over time
- Forget gate: decides which information to discard from the cell state
- Input gate: determine which new information to add to the cell state
- Output gate: decides what the next hidden state should be, based on the current cell state and input
- Cell state: the LSTM's memory
- Hidden states: captures short-term memory and is updated at each time step

Benefits in Natural Language Programming

- Capture long-term dependencies
 - Mitigating vanishing gradient problem
 - Flexibility with variable-length sequences
 - Context awareness
-

Model Training: Long Short Term Memory (LSTM)

- Model Description:
 - Layers: Build models with 1 and 2 layers
 - Models with more than 2 layers requires a large amount of time to train
 - Batch Size: 64
 - Input Size: 300 (size of word embeddings)
 - Hidden layer size: [34, 64, 128]
 - Number of classes: 5 (number of labels)
 - Number of epochs: 1000
 - Softmax Activation Function
 - Adam Optimizer with learning rate 0.001
 - Early stopping when validation loss < 0.0152
 - Mini Batch Gradient Descent
 - Model parameters are updated after each mini batch
 - Cross entropy loss: used for classification tasks
- Best model: The model with 1 hidden layers, 64 hidden layer size and the above parameters



```
class FeedForwardNet(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, hidden_size3, hidden_size4, num_classes):
        super(FeedForwardNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, hidden_size3)
        self.relu3 = nn.ReLU()
        self.fc4 = nn.Linear(hidden_size3, hidden_size4)
        self.relu4 = nn.ReLU()
        self.fc5 = nn.Linear(hidden_size4, num_classes)
        self.softmax = nn.Softmax(dim=1)

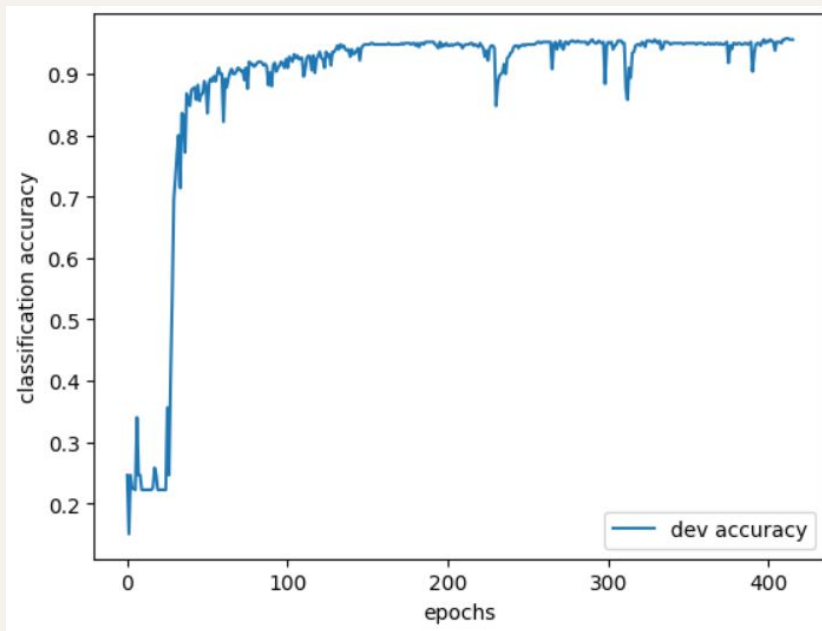
    def forward(self, x):
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        out = self.relu3(out)

        # Last hidden layer (layer 4)
        out = self.fc4(out)
        out = self.relu4(out)

        # Perform mean aggregation along the sequence dimension (dim=1)
        out = torch.mean(out, dim=1, keepdim=False)

        # Output layer
        out = self.fc5(out)
        out = self.softmax(out)
        return out
```

Results



Test Accuracy: 90.8%

Improvements

- LSTM is good for NLP because it is able to retain contexts in sentences. However, it may not perform well with long sentences. This is because the probability of keeping the context from a word that is far away from the current word being processed decreases exponentially with the distance from it. Furthermore, some words in a sentence are more important than others.
- To solve this transformers could be used
 - Transformers contain encoders and decoders, which contains a self-attention unit. These attention units enables the model to consider the entire sequence at once and weigh the importance of each word relative to others.
 - Process all elements of a sequence simultaneously, speeding up training and inference.



**Thank
You**