Sentiment Analysis with Neural Networks

In this project, we will be training a neural network for sentiment analysis. Given a body of text, it will predict whether the sentiment of that text is positive or negative.

The Model

The model we will be using consists of a word embedding layer followed by a dense layer and a sigmoid activation function.

Assuming there are M_j words in document j, the feature vector for document j is given by:

$$f_j = \frac{1}{M_j} \sum_{m=1}^{M_j} \mathbf{x}_{\mathbf{w}(m,j)}$$

Meanwhile, our dense layer will be given by:

$$z_j = \mathbf{w} \cdot \mathbf{f_j} + b$$

Our final output will be $\sigma[z_j]$, representing our prediction for document j as a value between 0 and 1.

Loss Function

We will be using a binary cross-entropy loss function to evaluate our model's performance. This function is defined as follows:

$$\begin{aligned} L_j(Y, \sigma[z_j]) &= -y \log \sigma[z_j] - (1-y)\log(1-\sigma[z_j]) \\ &= -y \log \sigma[\mathbf{w} \cdot \mathbf{f}_j + b] - (1-y)\log(1-\sigma[\mathbf{w} \cdot \mathbf{f}_j + b]) \\ &= -y \log \sigma[\mathbf{w} \cdot (\frac{1}{M_j} \sum_{m=1}^{M_j} \mathbf{x}_{\mathbf{w}(m,j)}) + b] - (1-y)\log(1-\sigma[\mathbf{w} \cdot (\frac{1}{M_j} \sum_{m=1}^{M_j} \mathbf{x}_{\mathbf{w}(m,j)}) + b] \end{aligned}$$

Derivative of the Loss Function

We will be using Adam to train our model, which is an optimization algorithm built on stochastic gradient descent. While Tensorflow will calculate all the needed gradients on its own, it is still a worthwhile exercise to calculate the derivative of our loss function to understand what Tensorflow's underlying code will be doing.

To perform gradient descent, we will need to calculate the derivatives $\frac{dL}{d\mathbf{x}}$, $\frac{dL}{d\mathbf{w}}$, and $\frac{dL}{db}$ for each document (or, in the case of stochastic gradient descent, each batch). We can use the chain rule to expand them out as follows:

$$\frac{dL}{d\mathbf{x}} = \frac{dL}{d\sigma[z]}\frac{d\sigma[z]}{dz}\frac{dz}{d\mathbf{f}}\frac{d\mathbf{f}}{d\mathbf{x}}$$

$$\frac{dL}{d\mathbf{w}} = \frac{dL}{d\sigma[z]}\frac{d\sigma[z]}{dz}\frac{dz}{d\mathbf{w}}$$

$$\frac{dL}{db} = \frac{dL}{d\sigma[z]}\frac{d\sigma[z]}{dz}\frac{dz}{db}$$

As we can see, there are quite a few repeated terms in these expanded derivatives. To make the calculation easier, and to improve the performance of our training, we can calculate the repeated terms a single time and reuse them for each gradient calculation.

$$\begin{aligned}\frac{dL}{d\sigma[z]} &= \frac{d}{d\sigma[z]}[-y\log\sigma[z_j] - (1-y)\log(1-\sigma[z_j])]\\ &= -y(\frac{1}{\sigma[z]}) - (1-y)(\frac{1}{1-\sigma[z]})(-1)\\ &= \frac{-y}{\sigma[z]} + \frac{1-y}{1-\sigma[z]}\end{aligned}$$

$$\begin{aligned}\frac{d\sigma[z]}{dz} &= \frac{d}{dz}[(1+e^{-z})^{-1}]\\ &= -(1+e^{-z})^{-2}(-e^{-z})\\ &= \frac{e^{-z}}{(1+e^{-z})^2}\end{aligned}$$

The remaining derivatives are not reused, but still must be calculated.

$$\begin{aligned}\frac{dz}{d\mathbf{f}} &= \frac{d}{d\mathbf{f}}[\mathbf{w}\cdot\mathbf{f}+b]\\ &= \mathbf{w}\end{aligned}$$

$$\begin{aligned}\frac{d\mathbf{f}}{d\mathbf{x}} &= \frac{d}{d\mathbf{x}}[\frac{1}{M}\sum_{m=1}^{M}\mathbf{x}_{\mathbf{w}(m)}]\\ &= \frac{1}{M}\sum_{m=1}^{M}1\\ &= \frac{1}{M}M\\ &= 1\end{aligned}$$

$$\begin{aligned}\frac{dz}{d\mathbf{w}} &= \frac{d}{d\mathbf{w}}[\mathbf{w}\cdot\mathbf{f}+b]\\ &= \mathbf{f}\end{aligned}$$

$$\begin{aligned}\frac{dz}{db} &= \frac{d}{db}[\mathbf{w}\cdot\mathbf{f}+b]\\ &= 1\end{aligned}$$

With the components calculated, we can now rewrite our needed gradients:

$$\begin{aligned}\frac{dL}{d\mathbf{x}} &= \frac{dL}{d\sigma[z]}\frac{d\sigma[z]}{dz}\frac{dz}{d\mathbf{f}}\frac{d\mathbf{f}}{d\mathbf{x}} \\ &= (\frac{-y}{\sigma[z]} + \frac{1-y}{1-\sigma[z]})(\frac{e^{-z}}{(1+e^{-z})^2})(\mathbf{w})(1)\end{aligned}$$

$$\begin{aligned}\frac{dL}{d\mathbf{w}} &= \frac{dL}{d\sigma[z]}\frac{d\sigma[z]}{dz}\frac{dz}{d\mathbf{w}} \\ &= (\frac{-y}{\sigma[z]} + \frac{1-y}{1-\sigma[z]})(\frac{e^{-z}}{(1+e^{-z})^2})(\mathbf{f})\end{aligned}$$

$$\begin{aligned}\frac{dL}{db} &= \frac{dL}{d\sigma[z]}\frac{d\sigma[z]}{dz}\frac{dz}{db} \\ &= (\frac{-y}{\sigma[z]} + \frac{1-y}{1-\sigma[z]})(\frac{e^{-z}}{(1+e^{-z})^2})(1)\end{aligned}$$

Scalability with Adam

To train our model, we want to avoid doing gradient descent across our entire dataset. We will be training our model on K documents, where K could be a very large number, so this will require us to store a lot of information and will update the model very slowly. Instead, we will be doing gradient descent over batches of our data points. This allows us to update our weights quickly, leading to a pretty accurate model quite quickly. It also allows our model to be trained on a computer with less memory.

To implement this, we can simply update the batch_size parameter in our Tensorflow model.

Training, Testing, and Validation Sets

The database we are using (Yelp Polarity) comes prepackaged with a training and testing set. We will make sure to only use the test set when evaluating the final model.

We have also set aside 20% of our data to act as our validation set. This will allow us to test our model while we train it in order to avoid overfitting. We want it separate from our training data so that we don't lose our generalizability, and we want it separate from our testing data so we can avoid unnecessary bais in our model construction.

To create this validation set, we used Tensorflow's built-in validation_split parameter. We also added a callback to our model to stop the training early if the error in the validation set starts growing. We set the patience parameter to 3, which means that the model will allow up to 3 increases in validation-set loss before it stops training. As a fail-safe, we also limited our model to 50 epochs, but it always early-stopped before we reached this point.

See our implementation below:

```
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=3, verbose=2)
model.fit(texts, labels, batch_size=2048, epochs=50,
    validation_split=0.2, callbacks=[early_stopping], verbose=2)
```

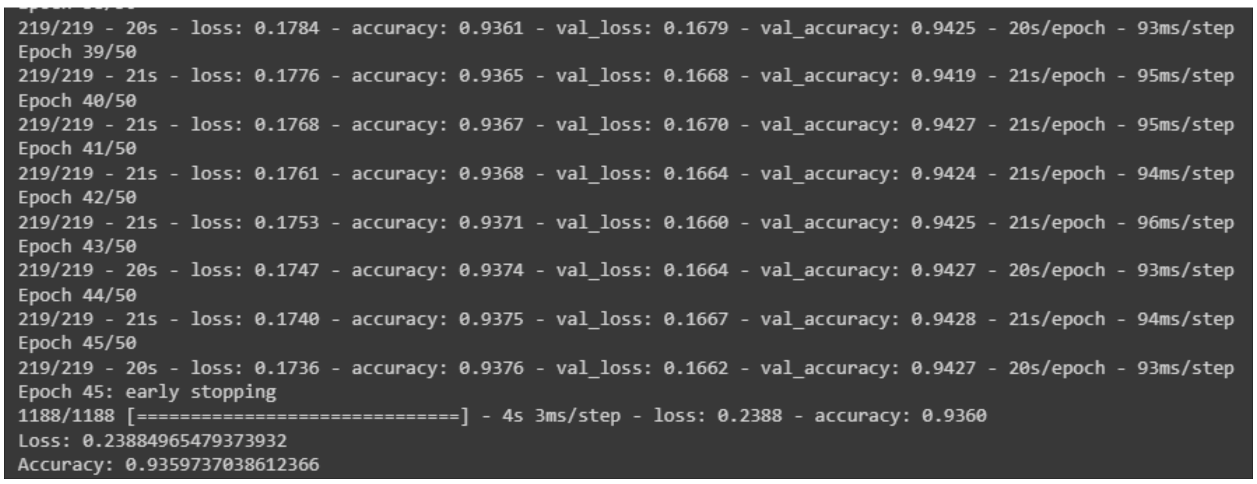
Evaluation of the Model

Our model ran for 45 epochs before stopping early due to the validation set callback. The training ended with a loss of 0.1662 and an accuracy of 0.9376, and the validation set had a loss of 0.1662 with an accuracy of 0.9427.

Now trained, we ran the model on our test data (which, I remind you, was not involved in the training process at all). On this data, our model showed a total loss of 0.2388 and an accuracy of 0.9360.

These results are quite strong. Our accuracy is quite high, even on the test set, and I feel confident saying that this project was a success.

I would expect that much of the remaining error is unavoidable given our model definition as we have not incorporated word order. One could easily think of two sentences with a very similar composition of words where the word order completely changes the perceived sentiment. Even still, our accuracy is quite high for such a simple model, and I would expect that our results are good enough for most practical applications of sentiment analysis.



```
219/219 - 20s - loss: 0.1784 - accuracy: 0.9361 - val_loss: 0.1679 - val_accuracy: 0.9425 - 20s/epoch - 93ms/step
Epoch 39/50
219/219 - 21s - loss: 0.1776 - accuracy: 0.9365 - val_loss: 0.1668 - val_accuracy: 0.9419 - 21s/epoch - 95ms/step
Epoch 40/50
219/219 - 21s - loss: 0.1768 - accuracy: 0.9367 - val_loss: 0.1670 - val_accuracy: 0.9427 - 21s/epoch - 95ms/step
Epoch 41/50
219/219 - 21s - loss: 0.1761 - accuracy: 0.9368 - val_loss: 0.1664 - val_accuracy: 0.9424 - 21s/epoch - 94ms/step
Epoch 42/50
219/219 - 21s - loss: 0.1753 - accuracy: 0.9371 - val_loss: 0.1660 - val_accuracy: 0.9425 - 21s/epoch - 96ms/step
Epoch 43/50
219/219 - 20s - loss: 0.1747 - accuracy: 0.9374 - val_loss: 0.1664 - val_accuracy: 0.9427 - 20s/epoch - 93ms/step
Epoch 44/50
219/219 - 21s - loss: 0.1740 - accuracy: 0.9375 - val_loss: 0.1667 - val_accuracy: 0.9428 - 21s/epoch - 94ms/step
Epoch 45/50
219/219 - 20s - loss: 0.1736 - accuracy: 0.9376 - val_loss: 0.1662 - val_accuracy: 0.9427 - 20s/epoch - 93ms/step
Epoch 45: early stopping
1188/1188 [==============================] - 4s 3ms/step - loss: 0.2388 - accuracy: 0.9360
Loss: 0.23884965479373932
Accuracy: 0.9359737038612366
```

Figure 1: Final training results and test data evaluation

Project Code

You can find our Google Colab notebook here.

```
import os

!pip install datasets
from datasets import load_dataset

from google.colab import drive

import tensorflow as tf
import tensorflow.keras as keras

import numpy as np

drive.mount('/content/drive/')
os.chdir('/content/drive/MyDrive/sentiment_analysis')

def get_dataset_split():

    dataset = load_dataset("yelp_polarity")

    training_texts = np.array(dataset['train']['text'])
    training_labels = np.array(dataset['train']['label'])

    testing_texts = np.array(dataset['test']['text'])
    testing_labels = np.array(dataset['test']['label'])

    all_text = np.concatenate([training_texts, testing_texts])

    return training_texts, training_labels, testing_texts,
       testing_labels, all_text

def build_model(all_text, model_path = 'empty_model.keras'):

    if os.path.exists(model_path):
        model = keras.models.load_model(model_path)
    else:
        text_vectorization = keras.layers.TextVectorization(
           max_tokens=20000)

        text_vectorization.adapt(all_text)

        vocabulary = text_vectorization.get_vocabulary()
        vocabulary_size = len(vocabulary)
        embedding_dimensions = 50

        model = keras.Sequential([
            text_vectorization,
            keras.layers.Embedding(input_dim=vocabulary_size,
               output_dim=embedding_dimensions),
            keras.layers.GlobalAveragePooling1D(),
            keras.layers.Dense(1, activation='sigmoid')
        ])

        model.save(model_path)

    model.compile(optimizer='adam', loss='binary_crossentropy',
       metrics=['accuracy'])
    return model

def train_model(model, texts, labels):
    early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss
       ', patience=3, verbose=2)
    model.fit(texts, labels, batch_size=2048, epochs=50,
       validation_split=0.2, callbacks=[early_stopping], verbose=2)

def evaluate_model(model, texts, labels):
    loss, accuracy = model.evaluate(texts, labels)

    print(f'Loss:␣{loss}')
    print(f'Accuracy:␣{accuracy}')

training_tests, training_labels, testing_texts, testing_labels,
   all_text = get_dataset_split()

model = build_model(all_text)

train_model(model, training_tests, training_labels)

model.save('sentiment_analysis_model.keras')

# model = keras.models.load_model('sentiment_analysis_model.keras')

evaluate_model(model, testing_texts, testing_labels)
```

Acknowledgements

I did this project (the code, the derivations, and the write-up) entirely by myself. With that said, Aditya Gaur and Kushal Mohta were valuable sources of discussion, for which I am grateful.