Recurrent Neural Networks (RNNs)

The task was to use Recurrent Neural Networks (RNNs) to classify online reviews into positive or negative categories. This involved understanding how RNNs process text data and apply this understanding to sentiment analysis.

The purpose is to outline the steps taken in this sentiment analysis project, from data preparation to model training and evaluation. It aims to highlight the challenges faced, the learning curve experienced, and the insights gained, making the technical process understandable even to those unfamiliar with the field.

Description of Experience

The project centered around processing and analyzing a dataset of reviews. The goal was to automatically determine the sentiment expressed in each review using RNNs, a type of artificial intelligence designed for working with sequences, like sentences.

Specific Details.

The first step was to load all the online reviews from a file into our workspace. This file contained lots of text written by people about different products they've used.

```
df = pd.read_csv("data/NLP-REVIEW-DATA-CLASSIFICATION-TRAINING.csv")
df.head()
```

	ID	reviewText	summary	verified	time	log_votes	isPositive
0	65886	Purchased as a quick fix for a needed Server 2...	Easy install, seamless migration	True	1458864000	0.000000	1
1	19822	So far so good. Installation was simple. And r...	Five Stars	True	1417478400	0.000000	1
2	14558	Microsoft keeps making Visual Studio better. I...	This is the best development tool I've ever used.	False	1252886400	0.000000	1
3	39708	Very good product.	Very good product.	True	1458604800	0.000000	1
4	8015	So very different from my last version and I a...	... from my last version and I am having a gre...	True	1454716800	2.197225	0

Sometimes, some reviews might not have any text (they're missing). So, we filled in those empty spots with the word "missing" to make sure every review had some text for the computer to look at.

```
df["reviewText"] = df["reviewText"].fillna("missing")
```

Divided the reviews into two groups. One group was for teaching the computer what positive and negative reviews look like (training set), and the other was for testing how well it learned (validation set).

```
# This separates 10% of the entire dataset into validation dataset.
train_text, val_text, train_label, val_label = train_test_split(
    df["reviewText"].tolist(),
    df["isPositive"].tolist(),
    test_size=0.10,
    shuffle=True,
    random_state=324,
)
```

Since computers understand numbers better than words, we made a list (vocabulary) that turned every unique word in the reviews into a specific number. This way, the computer could start to understand the text.

```
tokenizer = get_tokenizer("basic_english")
counter = Counter()
for line in train_text:
```

```
    counter.update(tokenizer(line))
vocab = vocab(counter, min_freq=2, specials=["<unk>"]) #min_freq>1 for skipping mis
vocab.set_default_index(vocab['<unk>'])
```

Next, converted all the reviews from words into sequences of numbers using the vocabulary we made. This process was like translating the reviews into a language the computer could understand.

```
# Let's create a mapper to transform our text data
text_transform_pipeline = lambda x: [vocab[token] for token in tokenizer(x)]
```

The model likes to have everything in a neat order, so we made sure every review was the same length. If a review was too short, we added extra numbers to make it longer, and if it was too long, we made cuts to shorten it. This way, all reviews were equal in size for the model.

```
def pad_features(reviews_split, seq_length):
    # Transform the text
    # use the dict to tokenize each review in reviews_split
    # store the tokenized reviews in reviews_ints
    reviews_ints = []
    for review in reviews_split:
        reviews_ints.append(text_transform_pipeline(review))

    # getting the correct rows x cols shape
    features = np.ones((len(reviews_ints), seq_length), dtype=int)

    # for each review, I grab that review
    for i, row in enumerate(reviews_ints):
        features[i, -len(row):] = np.array(row)[:seq_length]

    return torch.tensor(features, dtype=torch.int64)
```

With the reviews ready, we built the RNN model. This is a special kind of model designed to read sequences (like our reviews) and learn from them. It's particularly good at noticing patterns in text, which helps it decide if a review is positive or negative

```
class Net(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size, num_classes, num_layers
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size, padding_idx=1)
        self.rnn = nn.RNN(
            embed_size, hidden_size, num_layers=num_layers, batch_first=True
        )

        self.linear = nn.Linear(hidden_size, num_classes)

    def forward(self, inputs):
        embeddings = self.embedding(inputs)
        # Call the RNN layer
        outputs, _ = self.rnn(embeddings)

        # Output shape after RNN: (batch_size, max_len, hidden_size)
        # Get the output from the last time step with outputs[:, -1, :] below
        # The output shape becomes: (batch_size, 1, hidden_size)
        # Send it through the linear layer
        return self.linear(outputs[:, -1, :])

# Initialize the weights
def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.xavier_uniform_(m.weight)
    if type(m) == nn.RNN:
        for param in m._flat_weights_names:
            if "weight" in param:
                nn.init.xavier_uniform_(m._parameters[param])
```

Training is like teaching the model through practice. We showed it many reviews, ones we already knew were positive or negative, and let it makes its own guesses. Whenever it was guessed wrong, we corrected it. Over time, it got better at guessing on its own.

```
def accuracy(y_hat, y):
    """Compute the number of correct predictions."""
    pred = torch.argmax(y_hat, axis=1)
    return torch.sum(pred == y)

def eval_accuracy(net, data_loader):
    # Use accumulator to keep track of metrics: correct predictions, num of predict
    metric = d2l.Accumulator(2)

    net.eval()
    for X, y in data_loader:
        y_hat = net(X)
        metric.add(accuracy(y_hat, y), y.numel())

    return metric[0] / metric[1]

print("Classification Accuracy:", eval_accuracy(model, val_loader))
```

```
Classification Accuracy: 0.37320402298850575
```

After training, we checked how well the model learned by having it guess the sentiments of new reviews it hadn't seen before. This test showed us if the model could accurately tell positive from negative reviews based on what it learned.

```
print("Classification Accuracy on Validation set:", eval_accuracy(model, val_loader
```

Personal Reflection

Thoughts and Feelings: The complexity of translating human sentiment into something a computer could recognize was daunting yet intriguing. The initial skepticism gradually turned into curiosity and excitement as the project progressed.

Analysis and Interpretation: Observing the RNN model gradually learned and improved its accuracy in classifying sentiments was a tangible demonstration of AI's potential. The model's growing proficiency highlighted the intricate relationship between data preparation and model performance.

Connections to Theoretical Knowledge: Applying RNNs to real data bridged classroom learning with practical application. Concepts such as neural network architecture, sequence processing, and machine learning algorithms became more concrete and understandable.

Critical Thinking: The project underscored the importance of thorough data preprocessing and the impact of model parameters on learning outcomes. Reflecting on the model's occasional misclassifications prompted considerations about the complexity of human language and the challenges it poses to AI.

Discussion of Improvements and Learning

Personal Growth: This experience was transformative, fostering a deeper appreciation for the field of NLP and AI. It cultivated a more nuanced understanding of the challenges and possibilities inherent in teaching machines to interpret human language.

Skills Developed: Beyond technical skills like coding and model tuning, the project honed analytical thinking, problem-solving, and the ability to convey complex technical processes in accessible terms.

Future Application: The skills and insights gained hold promise for future projects, not only in sentiment analysis but in broader applications of NLP and AI. The project has opened avenues for further exploration and innovation.

Summary: The sentiment analysis lab using RNNs was a comprehensive learning experience, shedding light on the practical aspects of NLP. It was a journey that not only enhanced technical knowledge but also stimulated a deeper curiosity about the field.

Final Thoughts: Reflecting on the entire process, from conceptualization to execution and evaluation, this project underscored the transformative potential of machine learning in interpreting human language. It has laid a solid foundation for future exploration and application in the ever-evolving landscape of AI.

References:

- AWS Learning Module 2 Lab 4.
- https://www.youtube.com/watch?v=dSCFk168vmo
- https://www.youtube.com/watch?v=y9PLF2GsD-c