CS5787: Exercises 2

https://github.com/shl225/CS5787_HW2

Sean Hardesty Lewis
shl225

1 Theory: Question 1 [10 pts]

One problem when working on sequence to sequence tasks, regardless of whether the model is an RNN or a Transformer, is varying sequence lengths. Assume you are given a minibatch with variable-length input and output sequences.

a. **Describe one way to deal with variable-length input sequences.**
One way to handle variable-length input sequences is through padding. This involves extending shorter sequences in a minibatch by appending a specific pad token (usually zero) until they match the length of the longest sequence. To make sure that the model doesn't process these padding values as meaningful data, a masking technique is used. A typical approach is to use a binary mask, where 1 represents real data and 0 indicates padding. For example, if the actual sequence is $[5, 8, 2]$ and the padded sequence is $[5, 8, 2, 0, 0]$, the corresponding mask would be $[1, 1, 1, 0, 0]$. This mask is then used during training to ensure that computations and backpropagation ignore the padded values.

b. **Similarly describe one way to deal with variable-length output sequences. Does your loss computation change at all in this case?**
Similarly one method to address variable-length output sequences involves padding the shorter sequences in the minibatch. But when it comes to loss computation, it is important to adjust for the padded outputs. Using a mask (like the one I described for inputs) makes sure that the loss is calculated only on the lengths of the actual sequences. This means applying the mask to the loss function so that it ignores the contributions from the padded tokens. For example, in cross-entropy loss, the loss for the padded positions would be multiplied by 0 in the mask, effectively removing them from the loss calculation.
Yes, the loss computation changes when dealing with variable-length output sequences. By using a mask to exclude the padded areas from the loss calculation, you ensure that the model's performance metrics reflect its ability to predict only the meaningful parts of the sequence and are not adversely affected by the padding. This adjustment is important for training accurate models in sequence-to-sequence tasks.

2 Theory: Question 2 [10 pts]

Name two advantages of GRUs over LSTMs.

Simpler Architecture
GRUs have a simpler structural design than LSTMs because they use fewer gating mechanisms. While LSTMs have three gates (input, output, and forget gates), GRUs combine the input and forget gates into a single "update gate" and use a "reset gate." This reduction in gates and therefore complexity can make GRUs faster to train than LSTMs, requiring fewer computational resources. This simplification also makes GRUs slightly easier to modify and experiment with in different neural network configurations.

Faster Training
Due to the reduced number of gates and parameters, GRUs tend to train faster than LSTMs. This can be useful when working with very large datasets or when computational resources

are a limiting factor. The reduced complexity usually results in lower memory requirements and faster execution, which can be good for applications requiring real-time analysis and processing.

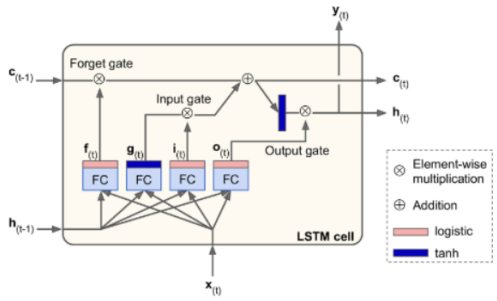
Note
On reading the paper referenced in Dr. Elor's referenced slidedeck "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", the authors Chung et. al find that in practice, the LSTM unit and GRU performed much better than the traditional tanh unit, but "could not make concrete conclusion on which of the two gating units [LSTM, GRU] was better." For this reason, my two advantages are not performance-related, but rather architecture. It is important to note that Cahuantzi et. al find that "Generally, GRUs outperform LSTM networks on low-complexity sequences while on high-complexity sequences LSTMs perform better."

3 Theory: Question 3 [10 pts]

The LSTM cell equations are as follows:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}^T x_t + W_{hi}^T h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}^T x_t + W_{hf}^T h_{t-1} + b_f) \\
o_t &= \sigma(W_{xo}^T x_t + W_{ho}^T h_{t-1} + b_o) \\
g_t &= \tanh(W_{xg}^T x_t + W_{hg}^T h_{t-1} + b_g) \\
c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t \\
y_t &= h_t = o_t \otimes \tanh(c_t)
\end{aligned}
$$

An illustration of an LSTM cell:



Assume that we have a network based on a single LSTM cell that uses a vector of size 200 to describe the current state. Also assume that its inputs are vectors of size 200. Considering only parameters related to the cell, how many parameters does it have?

Consider a CNN composed of three convolutional layers, each with 3×3 kernels, a stride of 2, and SAME padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200×300 pixels. What is the total number of parameters in the CNN? Explain your answer.

Given:

$$
\begin{aligned}
\text{Input vector size (input\_size)} &= 200 \\
\text{Hidden state size (hidden\_size)} &= 200
\end{aligned}
$$

Parameters in the LSTM cell:

An LSTM cell consists of four main gates (input, forget, output, and gate), each of which requires weight matrices for both the input and the hidden state, as well as bias vectors. These components determine how data flows through the LSTM cell.

Key Variables:

- **Input size (input_size)**: The size of the input vector, which is given as 200.
- **Hidden size (hidden_size)**: The size of the hidden state, also given as 200.

Calculation of Parameters:

1. **Weight Matrices for Input** ($W_{xi}, W_{xf}, W_{xo}, W_{xg}$):

   These matrices handle the transformation of the input vector at each gate.

- There are 4 gates in an LSTM cell, and each gate has its own input weight matrix.
- Each matrix has dimensions input_size $\times$hidden_size (200×200).
  The number of parameters in each matrix is $200 \times 200 = 40,000$.
- Since there are 4 such matrices, the total number of input weight parameters is $4 \times 40,000 = 160,000$.

$$\text{Number of parameters per matrix: } 200 \times 200 = 40,000$$
$$\text{Total for all input weights: } 4 \times 40,000 = 160,000$$

2. **Weight Matrices for Hidden State** ($W_{hi}, W_{hf}, W_{ho}, W_{hg}$):

   These matrices handle the transformation of the hidden state vector at each gate.

- Each gate has its own hidden state weight matrix, with dimensions hidden_size $\times$hidden_size (200×200). The number of parameters in each matrix is $200 \times 200 = 40,000$.
- Since there are 4 such matrices, the total number of hidden state weight parameters is $4 \times 40,000 = 160,000$.

$$\text{Number of parameters per matrix: } 200 \times 200 = 40,000$$
$$\text{Total for all hidden weights: } 4 \times 40,000 = 160,000$$

3. **Bias Vectors** (b_i, b_f, b_o, b_g):

   Each gate also has a bias vector with a length equal to the hidden state size.

- Each bias vector has hidden_size parameters (200).
- There are 4 bias vectors, so the total number of bias parameters is $4 \times 200 = 800$.

$$\text{Number of parameters per bias vector: } 200$$
$$\text{Total for all biases: } 4 \times 200 = 800$$

Total Parameters:

$$\text{Total parameters} = \text{Input weights} + \text{Hidden weights} + \text{Biases}$$
$$\text{Total parameters} = 160,000 + 160,000 + 800 = 320,800$$
$$\boxed{320,800 \text{ parameters}}$$

So the total number of parameters in the LSTM cell is **320,800**.

Overview:

- 160,000 parameters came from input weight matrices.
- 160,000 parameters come from hidden state weight matrices.
- 800 parameters come from bias vectors.
- So then LSTM cell must have 320,800 parameters.

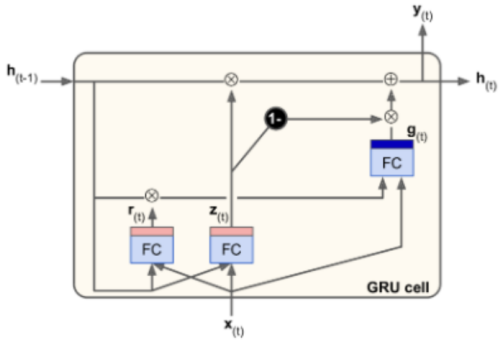
4 Theory: Question 3 [10 pts]

The GRU equations are as follows:

$$\mathbf{z}_t = \sigma(\mathbf{W}_{xz}^T \cdot \mathbf{x}_t + \mathbf{W}_{hz}^T \cdot \mathbf{h}_{t-1} + \mathbf{b}_z)$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_{xr}^T \cdot \mathbf{x}_t + \mathbf{W}_{hr}^T \cdot \mathbf{h}_{t-1} + \mathbf{b}_r)$$
$$\mathbf{g}_t = \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_t + \mathbf{W}_{hg}^T \cdot (\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + \mathbf{b}_g)$$
$$\mathbf{h}_t = \mathbf{z}_t \otimes \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \otimes \mathbf{g}_t$$

$$\text{Where } \sigma(x) = \frac{1}{1 + e^{-x}}$$

An illustration of an GRU cell:



Consider a GRU network with two timestamps (i.e. two iterations of the GRU cell), with loss $\epsilon_{(t)}$ (e.g., the L_2 loss: $\frac{1}{2}\left(h_{(t)} - y_t\right)^2$).

Assume the gradient $\frac{\partial \epsilon_{(2)}}{\partial h_{(2)}}$ is given.

Calculate the gradients of GRU for back propagation. For simplicity, you may ignore the bias, treat all variables as scalars and calculate the gradients of the second time stamp only. Using the chain rule, Calculate:

(a) $\frac{\partial \epsilon_{(2)}}{\partial W_{xz}}$

Step 1: Solving for $\frac{\partial h_2}{\partial W_{xz}}$

- We can see that h_2 depends on z_2, which depends on W_{xz}.
- Computing $\frac{\partial h_2}{\partial z_2}$:

$$h_2 = z_2 h_1 + (1 - z_2) g_2 \implies \frac{\partial h_2}{\partial z_2} = h_1 - g_2$$

- Computing $\frac{\partial z_2}{\partial W_{xz}}$:

$$z_2 = \sigma(a), \quad a = W_{xz} x_2 + W_{hz} h_1 \implies \frac{\partial z_2}{\partial W_{xz}} = \sigma'(a) x_2 = z_2(1 - z_2) x_2$$

- Applying chain rule:

$$\frac{\partial h_2}{\partial W_{xz}} = \frac{\partial h_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_{xz}} = (h_1 - g_2) \cdot z_2(1 - z_2) x_2$$

Step 2: Solving for $\frac{\partial \epsilon_{(2)}}{\partial W_{xz}}$

– Applying chain rule:

$$\frac{\partial \epsilon_{(2)}}{\partial W_{xz}} = \frac{\partial \epsilon_{(2)}}{\partial h_2} \cdot \frac{\partial h_2}{\partial W_{xz}} = \delta \cdot (h_1 - g_2) \cdot z_2(1 - z_2)x_2$$

Answer:

$$\boxed{\frac{\partial \epsilon_{(2)}}{\partial W_{xz}} = \delta \times (h_1 - g_2) \times z_2(1 - z_2)x_2}$$

(b) $\frac{\partial \epsilon_{(2)}}{\partial W_{hz}}$

Step 1: Solving for $\frac{\partial h_2}{\partial W_{hz}}$

– This is similar to part (a), but with respect to W_{hz}:

$$\frac{\partial z_2}{\partial W_{hz}} = z_2(1 - z_2)h_1$$

$$\frac{\partial h_2}{\partial W_{hz}} = (h_1 - g_2) \cdot z_2(1 - z_2)h_1$$

Step 2: Solving for $\frac{\partial \epsilon_{(2)}}{\partial W_{hz}}$

$$\frac{\partial \epsilon_{(2)}}{\partial W_{hz}} = \delta \cdot (h_1 - g_2) \cdot z_2(1 - z_2)h_1$$

Answer:

$$\boxed{\frac{\partial \epsilon_{(2)}}{\partial W_{hz}} = \delta \times (h_1 - g_2) \times z_2(1 - z_2)h_1}$$

(c) $\frac{\partial \epsilon_{(2)}}{\partial W_{xg}}$

Step 1: Solving for $\frac{\partial h_2}{\partial W_{xg}}$

– We can see that h_2 depends on g_2, which depends on W_{xg}.

– Computing $\frac{\partial h_2}{\partial g_2}$:

$$\frac{\partial h_2}{\partial g_2} = 1 - z_2$$

– Computing $\frac{\partial g_2}{\partial W_{xg}}$:

$$g_2 = \tanh(b), \quad b = W_{xg}x_2 + W_{hg}(r_2 h_1) \implies \frac{\partial g_2}{\partial W_{xg}} = (1 - g_2^2)x_2$$

– Applying chain rule:

$$\frac{\partial h_2}{\partial W_{xg}} = (1 - z_2) \cdot (1 - g_2^2)x_2$$

Step 2: Solving for $\frac{\partial \epsilon_{(2)}}{\partial W_{xg}}$

$$\frac{\partial \epsilon_{(2)}}{\partial W_{xg}} = \delta \cdot (1 - z_2) \cdot (1 - g_2^2)x_2$$

Answer:

$$\boxed{\frac{\partial \epsilon_{(2)}}{\partial W_{xg}} = \delta \times (1 - z_2) \times (1 - g_2^2)x_2}$$

(d) $\frac{\partial \epsilon_{(2)}}{\partial W_{hg}}$

Step 1: Solving for $\frac{\partial h_2}{\partial W_{hg}}$

- Computing $\frac{\partial g_2}{\partial W_{hg}}$:

$$\frac{\partial g_2}{\partial W_{hg}} = (1 - g_2^2) \cdot (r_2 h_1)$$

- Applying chain rule:

$$\frac{\partial h_2}{\partial W_{hg}} = (1 - z_2) \cdot (1 - g_2^2) \cdot r_2 h_1$$

Step 2: Solving for $\frac{\partial \epsilon_{(2)}}{\partial W_{hg}}$

$$\frac{\partial \epsilon_{(2)}}{\partial W_{hg}} = \delta \cdot (1 - z_2) \cdot (1 - g_2^2) \cdot r_2 h_1$$

Answer:

$$\boxed{\frac{\partial \epsilon_{(2)}}{\partial W_{hg}} = \delta \times (1 - z_2) \times (1 - g_2^2) r_2 h_1}$$

(e) $\frac{\partial \epsilon_{(2)}}{\partial W_{xr}}$

Step 1: Solving for $\frac{\partial h_2}{\partial W_{xr}}$

- We can see h_2 depends on r_2 via g_2.
- Computing $\frac{\partial h_2}{\partial r_2}$:

$$\frac{\partial h_2}{\partial r_2} = (1 - z_2) \cdot \frac{\partial g_2}{\partial r_2}$$

- Computing $\frac{\partial g_2}{\partial r_2}$:

$$\frac{\partial g_2}{\partial r_2} = (1 - g_2^2) \cdot W_{hg} h_1$$

- Computing $\frac{\partial r_2}{\partial W_{xr}}$:

$$\frac{\partial r_2}{\partial W_{xr}} = r_2(1 - r_2)x_2$$

- Applying chain rule:

$$\frac{\partial h_2}{\partial W_{xr}} = (1 - z_2) \cdot (1 - g_2^2) W_{hg} h_1 \cdot r_2(1 - r_2)x_2$$

Step 2: Solving for $\frac{\partial \epsilon_{(2)}}{\partial W_{xr}}$

$$\frac{\partial \epsilon_{(2)}}{\partial W_{xr}} = \delta \cdot \frac{\partial h_2}{\partial W_{xr}}$$

Answer:

$$\boxed{\frac{\partial \epsilon_{(2)}}{\partial W_{xr}} = \delta \times (1 - z_2)(1 - g_2^2) W_{hg} h_1 \times r_2(1 - r_2)x_2}$$

(f) $\frac{\partial \epsilon_{(2)}}{\partial W_{hr}}$

Step 1: Solving for $\frac{\partial h_2}{\partial W_{hr}}$

- Computing $\frac{\partial r_2}{\partial W_{hr}}$:

$$\frac{\partial r_2}{\partial W_{hr}} = r_2(1 - r_2)h_1$$

– Applying the chain rule:

$$\frac{\partial h_2}{\partial W_{hr}} = (1 - z_2) \cdot (1 - g_2^2) W_{hg} h_1 \cdot r_2 (1 - r_2) h_1$$

Step 2: Solving for $\frac{\partial \epsilon_{(2)}}{\partial W_{hr}}$

$$\frac{\partial \epsilon_{(2)}}{\partial W_{hr}} = \delta \cdot \frac{\partial h_2}{\partial W_{hr}}$$

Answer:

$$\boxed{\frac{\partial \epsilon_{(2)}}{\partial W_{hr}} = \delta \times (1 - z_2)(1 - g_2^2) W_{hg} h_1 \times r_2 (1 - r_2) h_1}$$

5 Practical [50 pts]

This project implements several variants of the "small" model as described in "Recurrent Neural Network Regularization", by Zaremba et al. for token prediction from the Penn Tree Bank dataset. The variants include:

- LSTM based network without dropout
- LSTM based network with dropout
- GRU based network without dropout
- GRU based network with dropout

The goal is to compare the perplexity of these techniques and achieve below 125 validation perplexity without dropout, and below 100 with it.

5.1 **Part (a): Convergence Graphs**

Below are the convergence graphs for each of the four experimental settings. Each graph shows both the training and validation perplexities over 200 epochs. The learning rate and dropout probabilities are specified for each setting.

5.1.1 **LSTM-based Network without Dropout**

Learning Rate: Starting at 5.0, adjusted using a scheduler based on validation perplexity.

Dropout Probability: 0.0

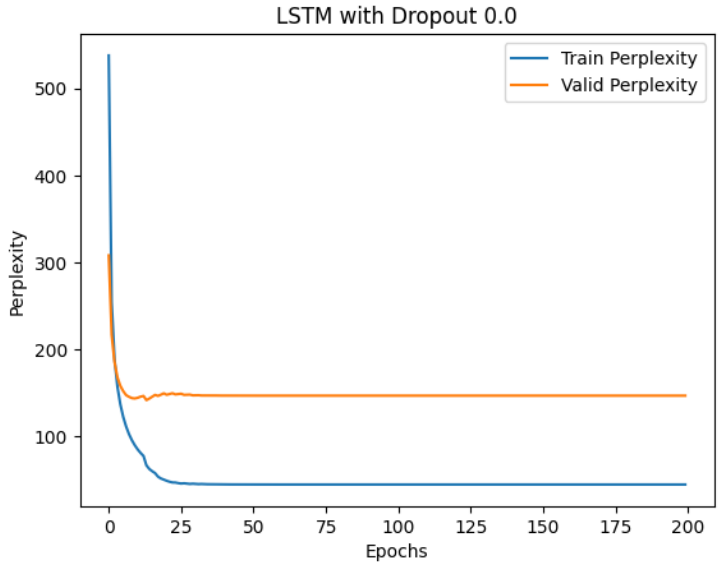


Figure 1: LSTM without Dropout: Train and Validation Perplexity over 200 Epochs

This graph shows how the training and validation perplexities decrease over 200 epochs for the LSTM model without dropout. The training perplexity decreases significantly, reaching around 44.93, while the validation perplexity decreases to approximately 141.88 but does not go below 125.

5.1.2 **LSTM-based Network with Dropout**

Learning Rate: Starting at 5.0, adjusted using a scheduler based on validation perplexity.

Dropout Probability: 0.25

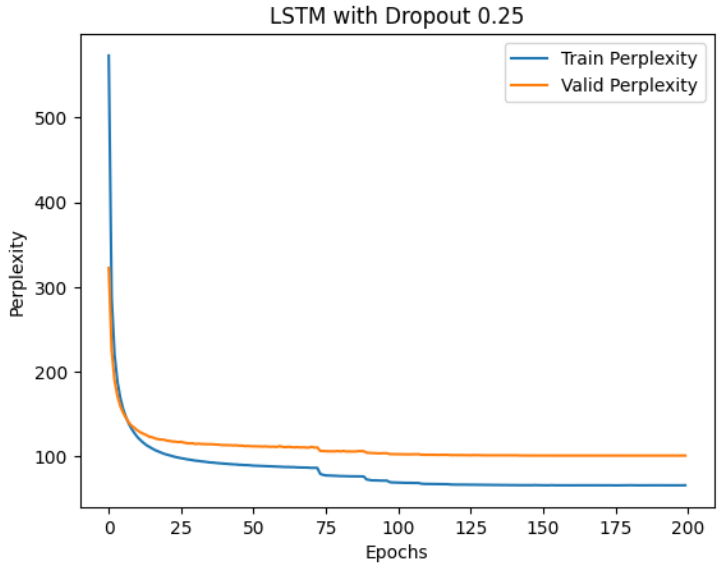


Figure 2: LSTM with Dropout (0.25): Train and Validation Perplexity over 200 Epochs

This graph shows the training and validation perplexities for the LSTM model with a dropout of 0.25. The training perplexity decreases to about 65.99, and the validation perplexity reaches around 100.97 but does not go below 100, even after my extensive training and hyperparameter adjustments.

5.1.3 GRU-based Network without Dropout

Learning Rate: Starting at 5.0, adjusted using a scheduler based on validation perplexity.

Dropout Probability: 0.0

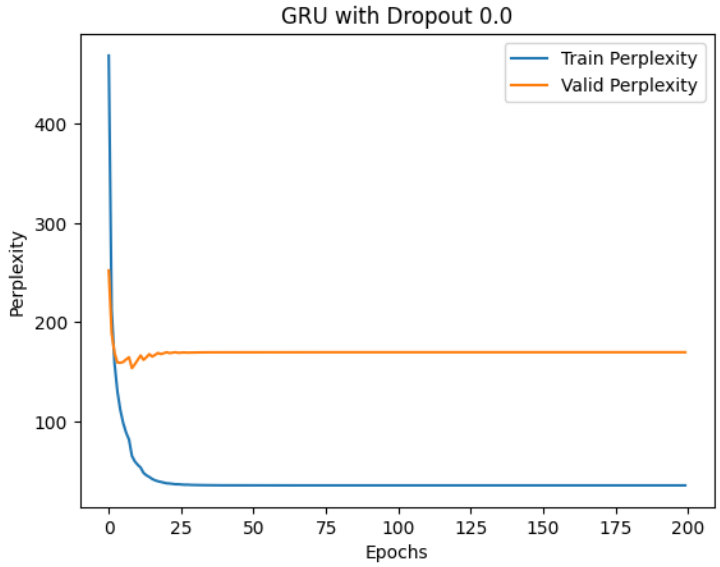


Figure 3: GRU without Dropout: Train and Validation Perplexity over 200 Epochs

This graph shows perplexities for the GRU model without dropout. The training perplexity decreases to approximately 35.62, but the validation perplexity flatlines around 153.75, remaining above 125 despite extended training.

5.1.4 GRU-based Network with Dropout

Learning Rate: Starting at 5.0, adjusted using a scheduler based on validation perplexity.

Dropout Probability: 0.28

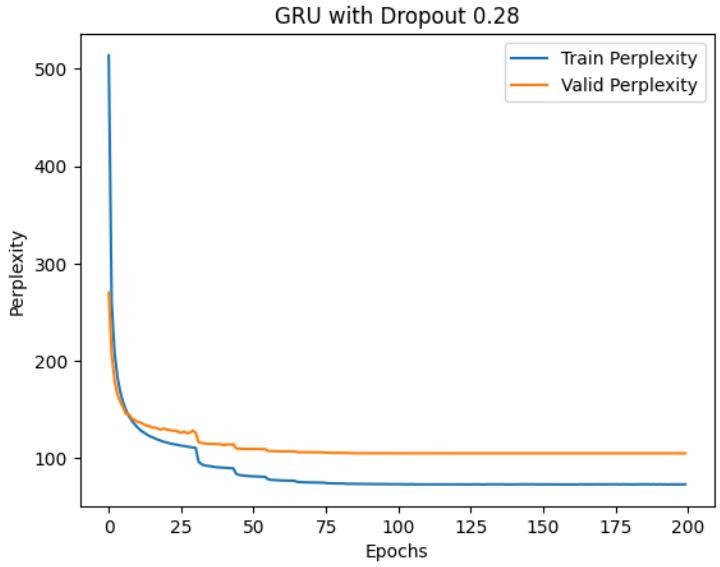


Figure 4: GRU with Dropout (0.28): Train and Validation Perplexity over 200 Epochs

This graph shows the perplexities for the GRU model with a dropout of 0.28. The training perplexity reduces to about 73.28, and the validation perplexity reaches approximately 105.10. Even after adjusting hyperparameters, the validation perplexity did not drop below 100.

5.2 Part (b): Summary of Results

Below is a table summarizing the final training perplexities, the minimum validation perplexities, and the test perplexity achieved at the minimum validation achieved by each model. The models are selected based on their lowest validation perplexity.

Model Type	Dropout	Final Train Perplexity	Min Validation Perplexity	Test Perplexity at Min
LSTM	0.0	44.93	141.88	139.78
GRU	0.0	35.62	153.75	96.92
LSTM	0.25	65.99	100.97	157.20
GRU	0.28	73.28	105.10	101.85

Table 1: Summary of Results for Each Model

5.3 Part (c): Conclusions

From the experiments conducted, several observations can be made:

1. **Impact of Dropout**: Introducing dropout improved the validation perplexity for both LSTM and GRU models. For the LSTM, the validation perplexity decreased from approximately 141.88 (without dropout) to 100.97 (with dropout). Similarly, for the GRU, it decreased from about 153.75 to 105.10.
2. **Training vs. Validation Perplexity**: While the training perplexities continued to decrease significantly over the epochs, the validation perplexities essentially flatlined after a certain point. This indicates that the models were overfitting to the training data, especially noticeable in the models without dropout (see graphs).
3. **Difficulty Achieving Target Perplexities**: Despite extensive training (200 epochs compared to the suggested 13) and adjusting hyperparameters such as dropout

rates and learning rates, I was unable to reduce the validation perplexity to below 100 for models with dropout and below 125 for models without dropout. This suggests that either with my current architecture and settings, there might be limitations in the model's capacity to generalize better on the validation set.

4. **GRU vs. LSTM Performance**: The LSTM models performed better on the validation set compared to the GRU models in terms of achieving lower perplexities. This could be due to the LSTM's ability to capture longer dependencies more effectively than GRUs in this context. See Note on Q2.
5. **Hyperparameter Adjustments**: I experimented with various dropout rates and learning rate schedules in an attempt to achieve lower validation perplexities. While these adjustments led to some improvements, they were not good enough to meet the target perplexities specified in the assignment guidelines.

Explanation for Validation Perplexity:

The inability to reach the desired validation perplexities could be due to several factors:

- **Model Complexity**: A hidden size of 200 units might not be good enough to capture the complexities of the Penn Tree Bank dataset.
- **Optimization Challenges**: Even with using learning rate schedulers and gradient clipping, the models might be getting stuck in local minima.
- **Data Preprocessing**: The way data is tokenized and preprocessed could impact the model's performance.

Possible Fixes:

- **Increase Model Capacity**: Experimenting with larger hidden sizes (e.g., 650 or 1500 units as in the "medium" and "large" models) might help achieve lower perplexities.
- **Advanced Regularization Techniques**: Incorporating techniques like weight decay, variational dropout, or layer normalization could improve generalization.
- **Hyperparameter Tuning**: Performing more exhaustive search over hyperparameters such as learning rates, dropout probabilities, and batch sizes might give better results as well.

Overall, our LSTM with tuned hyperparameters and dropout was the best performing, giving us back the lowest validation perplexity. The GRU model showed the most improvement when introducing dropout, but was unable to achieve lower validation perplexity than the LSTM dropout model. This project highlights the critical role of appropriate model selection and hyperparameter tuning in achieving effective language modeling, with specific attention to the balance between model capacity and overfitting.

5.4 **Training Instructions**

To train the models with my hyperparameters, use the following commands:

5.4.1 LSTM without Dropout:

```
config = TrainConfig(cell_type='LSTM', dropout=0.0, epochs=200)
```

5.4.2 LSTM with Dropout:

```
config = TrainConfig(cell_type='LSTM', dropout=0.25, epochs=200)
```

5.4.3 GRU without Dropout:

```
config = TrainConfig(cell_type='GRU', dropout=0.0, epochs=200)
```

5.4.4 GRU with Dropout:

```
config = TrainConfig(cell_type='GRU', dropout=0.28, epochs=200)
```

5.5 **Saving the Weights**

To save the weights of the trained models, use the following commands:

5.5.1 All Models:

```
if valid_perplexity < best_valid_perplexity:
    best_valid_perplexity = valid_perplexity
    torch.save(model.state_dict(),
        f'best_model_{config.cell_type}_dropout{config.dropout}.pth')
```

5.6 **Testing with Saved Weights**

To test the models with saved weights, use the following commands:

5.6.1 LSTM without Dropout:

```
model = RNNModel(vocab_size=vocab_size, hidden_size=200,
                    num_layers=2, dropout=0.0, model_type='LSTM')
model.load_state_dict(torch.load('best_model_LSTM_dropout0.0.pth'))
model.to(device)
```

5.6.2 LSTM with Dropout:

```
model = RNNModel(vocab_size=vocab_size, hidden_size=200,
                    num_layers=2, dropout=0.25, model_type='LSTM')
model.load_state_dict(torch.load('best_model_LSTM_dropout0.25.pth'))
model.to(device)
```

5.6.3 GRU without Dropout:

```
model = RNNModel(vocab_size=vocab_size, hidden_size=200,
                    num_layers=2, dropout=0.0, model_type='GRU')
model.load_state_dict(torch.load('best_model_GRU_dropout0.0.pth'))
model.to(device)
```

5.6.4 GRU with Dropout:

```
model = RNNModel(vocab_size=vocab_size, hidden_size=200,
                    num_layers=2, dropout=0.28, model_type='GRU')
model.load_state_dict(torch.load('best_model_GRU_dropout0.28.pth'))
model.to(device)
```

For more details, code, and relevant citations, please see the Github.