CS5787: Exercises 2

https://drive.google.com/drive/folders/1PtYo7eLchQffaVdnFCkWi82rg3qyaWvz?usp=drive_link

Full Name	**Dahwi Kim**	**Yoo Choi**
Net ID	dk865	yc2822

1 Theory: Question 1 [10 pts]

(a) One way to handle variable-length input sequences is to use padding to make all input sequences the same length. We want to find the longest input sequence in the batch, then pad all shorter sequences with padding tokens (either a unique token or 0's or token with no meaning) to match the length of the longest input sequence.

(b) For output sequences, we can handle varying lengths similarly through padding. However, during training, we want to ensure the loss is only computed for the actual tokens, excluding the padding tokens.
The loss computation needs to change in this case: after obtaining the raw loss, a binary mask is applied(multiplied) to ignore padding positions, ensuring that only the actual tokens contribute to the final loss. This prevents the padding tokens from affecting the training process.

2 Theory: Question 2 [10 pts]

Two advantages of GRUs (Gated Recurrent Units) over LSTMs (Long Short-Term Memory networks) are:

1. Simpler Architecture: GRUs have a simpler structure compared to LSTMs and thus fewer parameters to train. GRUs combine the forget and input gates in LSTMs into a single update gate, giving GRUs two gates: an update gate and a reset gate. LSTMs have three gates: input, output, and forget gates. Along with the removal of the separate cell state, this simpler architecture can result in faster training and inference times, as GRUs require fewer computations.

2. Less Prone to Overfitting: Due to having fewer parameters than LSTMs, GRUs are less likely to overfit, especially when the dataset is smaller or the model is relatively shallow. The reduced complexity makes GRUs more efficient and less memory-intensive.

3 Theory: Question 3 [10 pts]

In an LSTM cell, we have four dense (fully-connected) layers, denoted by $i_{(t)}$, $f_{(t)}$, $o_{(t)}$, and $g_{(t)}$ in the diagram. Therefore, we need to calculate the number of trainable parameters in one of those gates and multiply by 4 as all 4 gates take the same inputs.

The inputs to each gate are h_{t-1} and $x_{(t)}$, both of size 200. We also know that the output vector is size 200 because LSTM cell uses a vector of size 200 to describe the current state as mentioned in the problem.

Each gate's fully connected layer has:

- Weights for h_{t-1}: $200 \times 200 = 40,000$
- Weights for x_t: $200 \times 200 = 40,000$

- Biases: 200

Therefore, we have 80,200 (40,000 + 40,000 + 200) parameters for each gate.
So in total, the entire LSTM cell has

$$4 \cdot 80,200 = 320,800 \text{ parameters.}$$

4 Theory: Question 4 [20 pts]

Note that $\odot$ represents element-wise multiplication.

a.

$$\frac{\partial \epsilon_2}{\partial W_{xz}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_{xz}}$$

Taking a derivative of h with respect to z would obtain,

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

Knowing that the derivative of sigmoid,

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

We obtain the derivative of z with respect to W_{xz},

$$\frac{\partial z_2}{\partial W_{xz}} = z_2(1 - z_2) \cdot \frac{\partial (W_{xz}^T \cdot x_2)}{\partial W_{xz}} = z_2 \cdot (1 - z_2) \cdot x_2$$

Therefore, we get

$$\frac{\partial \epsilon_2}{\partial W_{xz}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot (h_1 - g_2) \cdot (z_2 \cdot (1 - z_2) \cdot x_2)$$

b.

$$\frac{\partial \epsilon_2}{\partial W_{hz}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_{hz}}$$

Taking a derivative of h with respect to z would obtain,

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

Taking derivative of z with respect to W_{hz},

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

Therefore, we get

$$\frac{\partial \epsilon_2}{\partial W_{xz}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot (h_1 - g_2) \cdot (z_2 \cdot (1 - z_2) \cdot h_1)$$

c.

$$\frac{\partial \epsilon_2}{\partial W_{xg}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial g_2} \cdot \frac{\partial g_2}{\partial W_{xg}}$$

Taking a derivative of h with respect to g would obtain,

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

Knowing that the derivative of tanh below,

$$\tanh'(x) = 1 - \tanh^2(x)$$

We obtain the derivative of g with respect to W_{xg},

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

Therefore, we get

$$\frac{\partial \epsilon_2}{\partial W_{xg}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot (1 - z_2) \cdot (1 - g_2^2) \cdot x_2$$

d.

$$\frac{\partial \epsilon_2}{\partial W_{hg}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial g_2} \cdot \frac{\partial g_2}{\partial W_{hg}}$$

Taking a derivative of h with respect to g would obtain,

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

Knowing that the derivative of tanh,

$$\tanh'(x) = 1 - \tanh^2(x)$$

We obtain the derivative of g with respect to W_{hg},

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

Therefore, we get

$$\frac{\partial \epsilon_2}{\partial W_{hg}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot (1 - z_2) \cdot (1 - g_2^2) \cdot (r_2 \odot h_1)$$

e.

$$\frac{\partial \epsilon_2}{\partial W_{xr}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial g_2} \cdot \frac{\partial g_2}{\partial r_2} \cdot \frac{\partial r_2}{\partial W_{xr}}$$

Taking a derivative of h with respect to g would obtain,

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

Knowing that the derivative of tanh,

$$\tanh'(x) = 1 - \tanh^2(x)$$

We obtain the derivative of g with respect to r,

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

Taking derivative of r with respect to W_{xr},

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

Therefore, we get

$$\frac{\partial \epsilon_2}{\partial W_{xr}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot (1 - z_2) \cdot (1 - g_2^2) \cdot (W_{hg}^T \cdot h_1) \cdot r_2 \cdot (1 - r_2) \cdot x_2$$

f.

$$\frac{\partial \epsilon_2}{\partial W_{hr}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial g_2} \cdot \frac{\partial g_2}{\partial r_2} \cdot \frac{\partial r_2}{\partial W_{hr}}$$

Taking a derivative of h with respect to g would obtain,

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

Knowing that the derivative of tanh,

$$\tanh'(x) = 1 - \tanh^2(x)$$

We obtain the derivative of g with respect to r,

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

Taking derivative of r with respect to W_{hzr},

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

Therefore, we get

$$\frac{\partial \epsilon_2}{\partial W_{hr}} = \frac{\partial \epsilon_2}{\partial h_2} \cdot (1 - z_2) \cdot (1 - g_2^2) \cdot (W_{hg}^T \cdot h_1) \cdot r_2 \cdot (1 - r_2) \cdot h_1$$

5 Practical [50 pts]

5.1 Architecture Description

The architecture follows the approach described in "Recurrent Neural Network Regularization", by Zaremba et al for "small" language models. It was tailored for word-level prediction on the Penn Tree Bank dataset. The model's key components are as follows:

- **Data processing:**
  - After we convert the words into tokens, we map the words to their first index of occurrence in the dataset. Once we have a 1-D tensor of integers for each dataset, we batchify the array into a shape (batchSize, lengthPerBatch). Then, we traverse through each batch by increments and extract a tuple (x, y), where x is an input sequence and y is its target sequence, offset by one time sequence. Any remainder tokens at the end of each batch are trimmed off. We referred to Ahmet Durmus's Python replication of Zaremba's paper for this step.

  - Resembling Zaremba's approach, we used a batch size of 20, where each minibatch is split into 20 chunks of sequences of length 20. Each batch therefore has 400 tokens.

- **Embedding Layer:** The model first starts with an embedding layer that takes in the input tokens (from a vocabulary of fixed size) into a dense vector representation. These embeddings have a size of 200 dimensions, which matches the dimensions of the hidden states in our model. The embedded layer is trained as well, aiming to capture semantic meaning in each word.
- **Recurrent layer:**
  - The core of the model is the RNN layer, which can configured to be either an LSTM or a GRU. These RNNs are designed to capture long-range dependencies in sequential data.
  - To follow the "small" model in the paper, both the LSTM and GRU have hidden state sizes of 200 and 2 recurrent layers.
  - For the LSTM model, both the hidden state and cell state are maintained. For the GRU, only the hidden state is preserved since it does not have a long-term cell state. In both cases, the hidden state of the prior minibatch is used for the next minibatch.
- **Dropout:** To prevent overfitting, dropout is applied both after the embedding layer and after the recurrent layer. Our results show that applying dropout as a regularization method in RNNs does help our model with generalization.
- **Fully Connected Output Layer:** After processing our embedded tokens through the RNN layers, we apply a fully connected layer to map the hidden state output back to the vector of vocab size, providing logits for the next predicted token in the sequence.
- **Parameter Initialization:** All model parameters, including the weights in the embedding, the LSTM/GRU cells, and the fully connected layer are initialized with uniform random values between $U(-0.1, 0.1)$. This was Zaremba's approach to help the model more consistently converge during training.
- **Hidden State Initialization:** The hidden states (and cell states in LSTM) are initialized to zeros at the beginning of every epoch. It allows the model to build memory from scratch at the start, but it preserves memory throughout the entire dataset.
- **State Detachment Between Batches:** During training, hidden states are detached from the computation graph between batches to prevent backpropagating through the entire dataset. Since the hidden states are preserved between batches, not detaching would cause the model to have to backpropagate through all previous batches for each gradient update, resulting in memory inefficiencies.

5.2 Training Hyperparameters & Experiments

- **Learning rates and decay:** Zaremba's "small" model trained the first four epochs at a learning rate of 1, then they decreased the learning rate by a factor of 2 after each epoch, for a total of 13 training epochs. We observed that hyperparameters used in the paper did not work achieve the same result. Therefore, we attempted to find the best initial learning rate and decay scheduling through repeated experimentation. The learning rate and decay scheduling for each model derived from many experiments are described in the Results section.
- **Dropout:** We found that a dropout probability of $p = 0.25$ achieved best results. Lower values caused too much overfitting and lack of generalization, and higher values could not find convergence.

5.3 Results

We were able to achieve < 125 validation perplexity for our models with no dropout, and < 105 validation perplexity for our models with dropout. During Omer's office hours, it was confirmed that a validation perplexity below 105 is acceptable for the regularized models.

- **LSTM No Dropout**

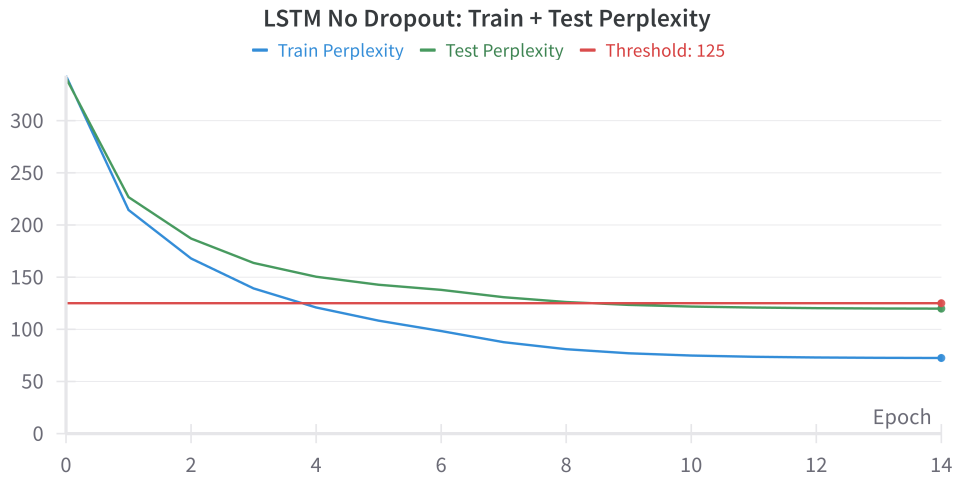

Figure 1: LSTM No Dropout: Train + Test Perplexity

For the non-regularized LSTM model, We ran the first 7 epochs with a learning rate of 2, then we decayed the learning rate by a factor of 2 every epoch thereafter for a total of 15 epochs. We achieved a validation perplexity of 124.26 at epoch 12 (`https://wandb.ai/cornell-tech-dl/dl-ex2/runs/yircprty?nw=8uedtjkja9v`).

- **GRU No Dropout**

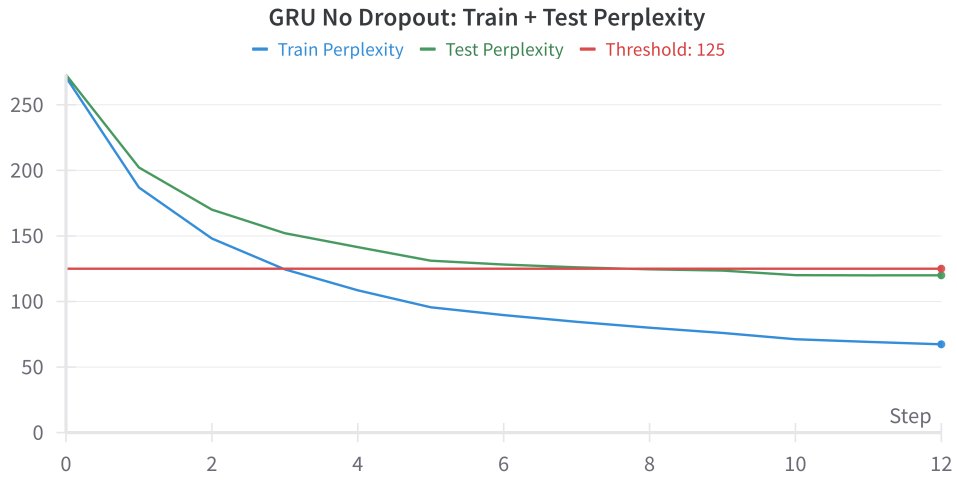

Figure 2: GRU No Dropout: Train + Test Perplexity

For the non-regularized GRU, we started with a learning rate of 1, then we decayed the learning rate by a factor of 2 every 5 epochs for a total of 13 epochs. We achieved a validation perplexity of 124.4 at epoch 11 (`https://wandb.ai/cornell-tech-dl/dl-ex2/runs/zc3gz8jg?nw=mrvr2c65z1`).

- **LSTM 25% Dropout**

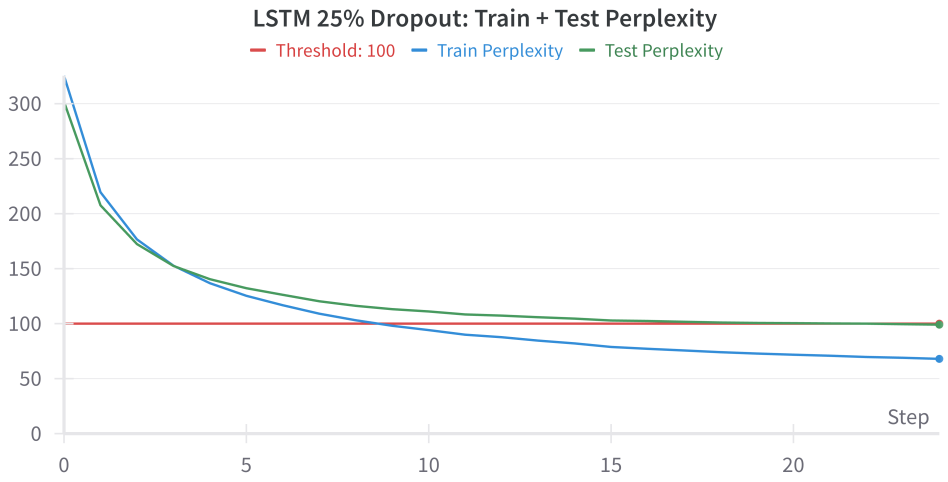

Figure 3: LSTM 25% Dropout: Train + Test Perplexity

For the LSTM with dropout, we chose a dropout probability of 0.25 because it obtained the best perplexity compared to other dropout rates (0.1, 0.3, and 0.4). We started with a learning rate of 4, and we decayed it by a factor of 1.15 every 15 epochs, for a total of 25 epochs. We achieved a validation perplexity of 102.66 at epoch 25 (`https://wandb.ai/cornell-tech-dl/dl-ex2/runs/k4noefui?nw=vihx2evwtl`).

- **GRU 25% Dropout**

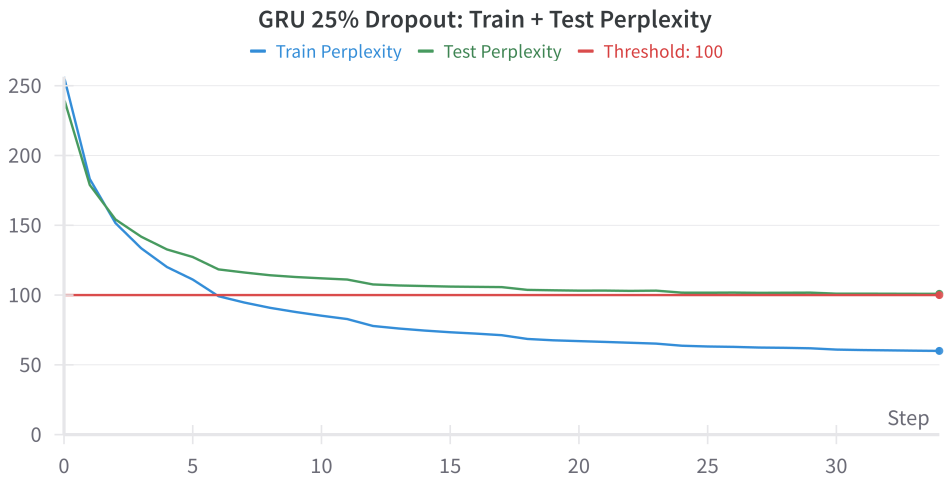

Figure 4: GRU 25% Dropout: Train + Test Perplexity

For the GRU with dropout, we also chose a dropout probability of 0.25 because it also obtained the best perplexity compared to other dropout rates (0.1, 0.3, and 0.4). We started with a learning rate of 2, and we decayed it by a factor of 1.65 every 6 epochs, for a total of 35 epochs. We achieved a validation perplexity of 104.49 at epoch 35 (`https://wandb.ai/cornell-tech-dl/dl-ex2/runs/mqvjd78v?nw=9hwg9okpmoj`).

For the model parameters that provided the best validation perplexity, we summarized their train set, validation, and test perplexities into the table below.

Perplexities of Various Models

Model	Training Perplexity	Validation Perplexity	Test Perplexity
LSTM No Dropout	72.49	123.75	119.82
GRU No Dropout	67.35	124.4	119.98
LSTM 25% Dropout	67.97	102.66	99.08
GRU 25% Dropout	59.98	104.49	100.88

Figure 5: Perplexities across all 4 models

5.4 Model Evaluation

5.4.1 Overfitting Across Models

The plots indicate that overfitting occurs in all four models, in both LSTM/GRU and with and without dropout. We see, for example, in the GRU No Dropout plot, the training perplexity steadily decreases over every epoch down to below 70 while the test perplexity plateaus at around epoch 6 and decreases at a much slower rate.

Models with dropout, such as the LSTM 25% Dropout and GRU 25% Dropout, exhibit better control over overfitting. The inclusion of dropout acts as our regularization technique, preventing the model from becoming too reliant on specific features. As a result, the gap between training and test perplexity is smaller for these models, especially for the LSTM with dropout.

5.4.2 Impact of Learning Rate

Contrary to Zaremba's results in his paper, we needed a much higher learning for the non-regularized LSTM to achieve desirable validation perplexity. We also decayed the learning rate much later in the training process. We found that both LSTM models, with and without dropout, required a higher learning rate than their GRU counterparts.

5.4.3 Learning Rate Experiments

After building our models following the approach in the paper regarding data processing and network setup, we first tested Zaremba's "small" model: LSTM with no dropout on a learning rate of 1 for the first 4 epochs, then decaying by a factor of 2 for every epoch after. However, for this setup, our perplexity was converging around 135-140. Given that we were using the SGD optimizer, we decided to try a higher initial learning rate and start decaying at a later epoch. This gave us better results and a validation perplexity of 124.8 within 12 epochs for our replication of Zaremba's "small" model.

Under the philosophy of trying out different learning rate scheduling, we tried out schedulers like step decay, cosine annealing, and cyclical learning rates. We found that step decay gave us the best results for the rest of our models. The challenge was to find the

optimal initial learning rate, number of steps between each decay, and the decay factor. The exact parameters we used for the learning rates in our four best models are specified above each graph.

Overall, models with dropout between recurrent layers required a higher initial learning rate and longer steps before decaying in order to converge down to the threshold perplexity values. The hardest challenge was finding the optimal decay factor for the GRU-based model with dropout. In the paper, Zaremba uses decay factors of 1.2 for his medium size model and 1.15 for his large model. Through experimentation, we found that a decay factor of 1.65 gave the optimal validation perplexity for the GRU with dropout.

5.5 Conclusion

Learning Rate Decay: As we described in detail above, we experimented with a wide range of initial learning rates and learning rate decays across all four models. We found that the best-performing model for each type of setup all used different learning rates and schedulers, showing that there does not exist a single best learning rate and decay.

Dropout: Though our models with dropout achieved better perplexities, we trained those models with more epochs. This was due to the fact that with dropout, we prevented our model from overfitting to specific patterns in the training data, giving it a better chance to perform well on unseen data over more epochs.

A potential experiment for further optimization could involve ensemble averaging, which is actually what Zaremba implements in his paper. Ensemble learning not only reduces variance where individual models could be overfitting to specific patterns in the training data, but it also exploits the fact that every model learns different aspects of the dataset, thus reducing overfitting.