ID1: 208560086
Name1: Naor Cohen

ID2: 301613501
Name2: Or Trabelsi

Deep Learning

2022-2023

TEL AVIV UNIVERSITY

EX2 - RNN

Theory

1.

a. Handling variable-length input sequences:

Padding: Add special tokens to shorter input sequences to make them the same length as the longest sequence.

Bucketing: Group input sequences by their length and pad them within each group, reducing the amount of padding needed overall.

Dynamic RNNs: Use dynamic RNNs that can process sequences of different lengths within the same batch by processing only valid timesteps.

b. Handling variable-length output sequences:

Dynamic Output Length with End-of-Sequence Token: Instead of padding the sequences, you can generate the output dynamically until the end-of-sequence token is predicted by the model. This way, the model can produce sequences of variable lengths, terminating each sequence when the end-of-sequence token is generated. This approach is often used in tasks like text generation or machine translation, where the length of the output sequence can vary.

Padding: Add special tokens to shorter input sequences to make them the same length as the longest sequence.

Bonus: Use Transformers! 😎

2.

Simpler architecture: GRU has a simpler structure compared to LSTM. It combines the input and forgets gates of LSTM into a single update gate, reducing the number of gating units. As a result, GRU has fewer parameters and is computationally more efficient.

Faster training and inference: Due to its simpler architecture, GRU typically requires fewer computations during training and inference compared to LSTM. This can lead to faster convergence during training and faster predictions at runtime.

Less prone to overfitting: The simpler architecture of GRU makes it less prone to overfitting on smaller datasets compared to LSTM. When the available training data is limited, GRU can often achieve comparable or better performance than LSTM.

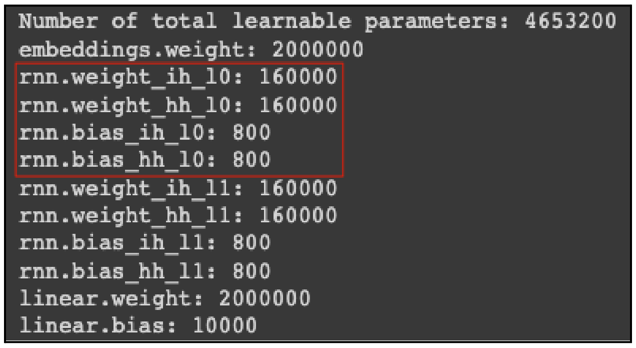
3.

In the given scenario, the LSTM cell consists of **8 fully connected layers**, each having 40,000 weight parameters (resulting from multiplying the input size of 200 by the output size of 200). Additionally, there are **4 bias vectors** with 200 parameters each. Overall, the parameter count for the LSTM cell is calculated as 8 multiplied by 40,000, plus 4 multiplied by 200, resulting in a total of 320,800 parameters.

$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma(\overset{1}{\mathbf{W}}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \overset{2}{\mathbf{W}}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \overset{1}{\mathbf{b}}_{i}) \\
\mathbf{f}_{(t)} &= \sigma(\overset{3}{\mathbf{W}}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \overset{4}{\mathbf{W}}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \overset{2}{\mathbf{b}}_{f}) \\
\mathbf{o}_{(t)} &= \sigma(\overset{5}{\mathbf{W}}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \overset{6}{\mathbf{W}}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \overset{3}{\mathbf{b}}_{o}) \\
\mathbf{g}_{(t)} &= \tanh(\overset{7}{\mathbf{W}}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \overset{8}{\mathbf{W}}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \overset{4}{\mathbf{b}}_{g}) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})
\end{aligned}
$$

PyTorch's LSTM implementation employs two bias terms (b_ih and b_hh) for each gate in the LSTM cell, in contrast to the single bias term used in the provided assignment implementation.

Therefore, in the notebook, you will probably see different numbers of weights:



```
Number of total learnable parameters: 4653200
embeddings.weight: 2000000
rnn.weight_ih_l0: 160000
rnn.weight_hh_l0: 160000
rnn.bias_ih_l0: 800
rnn.bias_hh_l0: 800
rnn.weight_ih_l1: 160000
rnn.weight_hh_l1: 160000
rnn.bias_ih_l1: 800
rnn.bias_hh_l1: 800
linear.weight: 2000000
linear.bias: 10000
```

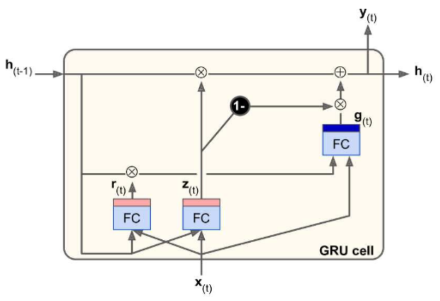
Since we are ignoring the bias terms, let's write the new equations based on the given GRU cell.

$$z_{(t)} = \sigma\left(W_{xz}{}^{T} \cdot x_{(t)} + W_{hz}{}^{T} \cdot h_{(t-1)}\right)$$

$$r_{(t)} = \sigma\left(W_{xr}{}^{T} \cdot x_{(t)} + W_{hr}{}^{T} \cdot h_{(t-1)}\right)$$

$$g_{(t)} = tanh\left(W_{xg}{}^{T} \cdot x_{(t)} + W_{hg}{}^{T} \cdot \left(r_{(t)} \otimes h_{(t-1)}\right)\right)$$

$$h_{(t)} = z_{(t)} \otimes h_{(t-1)} + \left(1 - z_{(t)}\right) \otimes g_{(t)}$$



Denote the following definitions,

$$\delta_{(2)} \equiv \frac{\partial \epsilon_{(2)}}{\partial h_{(2)}}$$

$$\epsilon_{(2)} \equiv \frac{1}{2} \cdot \left(h_{(2)} - y_{2}\right)^{2}$$

$$\sigma(x) \equiv \frac{1}{1+e^{-x}}$$

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

$$tanh'(x) \equiv 1 - tanh^{2}(x)$$

c. $\frac{\partial \epsilon_{(2)}}{\partial W_{xz}} = \frac{\partial \epsilon_{(2)}}{\partial h_{(2)}} \cdot \frac{\partial h_{(2)}}{\partial z_{(2)}} \cdot \frac{\partial z_{(2)}}{\partial W_{xz}} =$

$$= \delta_{(2)} \cdot \left(h_{(1)} - g_{(2)}\right) \cdot \sigma'\left(W_{xz}{}^{T} \cdot x_{(2)} + W_{hz}{}^{T} \cdot h_{(1)}\right) \cdot x_{(2)}$$

d. $\frac{\partial \epsilon_{(2)}}{\partial W_{hz}} = \frac{\partial \epsilon_{(2)}}{\partial h_{(2)}} \cdot \frac{\partial h_{(2)}}{\partial z_{(2)}} \cdot \frac{\partial z_{(2)}}{\partial W_{xz}} =$

$$= \delta_{(2)} \cdot \left(h_{(1)} - g_{(2)}\right) \cdot \sigma'\left(W_{xz}{}^{T} \cdot x_{(2)} + W_{hz}{}^{T} \cdot h_{(1)}\right) \cdot h_{(2)}$$

e. $\frac{\partial \epsilon_{(2)}}{\partial W_{xg}} = \frac{\partial \epsilon_{(2)}}{\partial h_{(2)}} \cdot \frac{\partial h_{(2)}}{\partial g_{(2)}} \cdot \frac{\partial g_{(2)}}{\partial W_{xg}} =$

$$= \delta_{(2)} \cdot \left(1 - z_{(2)}\right) \cdot tanh'\left(W_{xg}{}^{T} \cdot x_{(2)} + W_{hg}{}^{T} \cdot \left(r_{(2)} \otimes h_{(1)}\right)\right) \cdot x_{(2)}$$

f. $\frac{\partial \epsilon_{(2)}}{\partial W_{hg}} = \frac{\partial \epsilon_{(2)}}{\partial h_{(2)}} \cdot \frac{\partial h_{(2)}}{\partial g_{(2)}} \cdot \frac{\partial g_{(2)}}{\partial W_{hg}} =$

$$= \delta_{(2)} \cdot \left(1 - z_{(2)}\right) \cdot tanh'\left(W_{xg}{}^{T} \cdot x_{(2)} + W_{hg}{}^{T} \cdot \left(r_{(2)} \otimes h_{(1)}\right)\right) \cdot r_{(2)} \otimes h_{(1)}$$

g. $\frac{\partial \epsilon_{(2)}}{\partial W_{xr}} = \frac{\partial \epsilon_{(2)}}{\partial h_{(2)}} \cdot \frac{\partial h_{(2)}}{\partial g_{(2)}} \cdot \frac{\partial g_{(2)}}{\partial r_{(2)}} \cdot \frac{\partial r_{(2)}}{\partial W_{xr}} =$

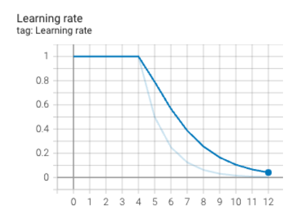
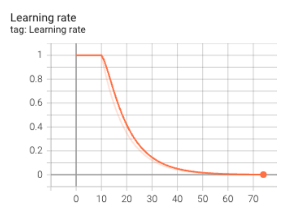
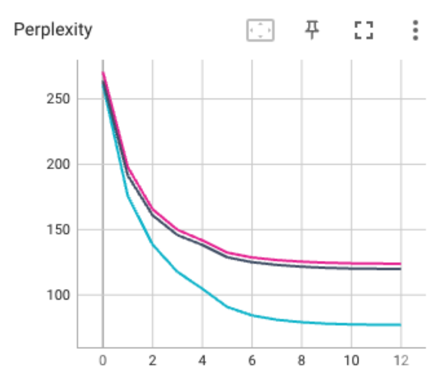
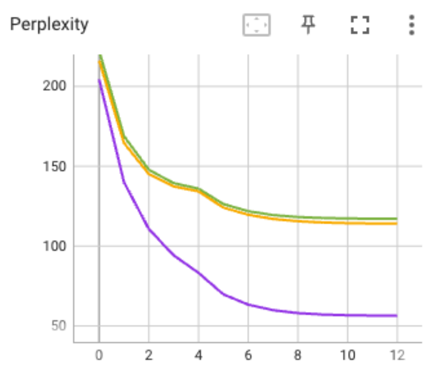
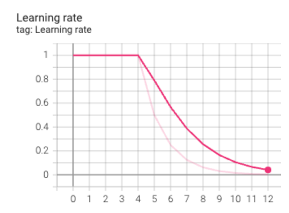
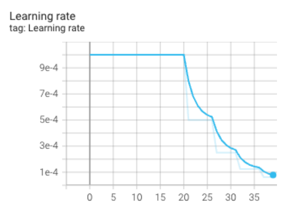
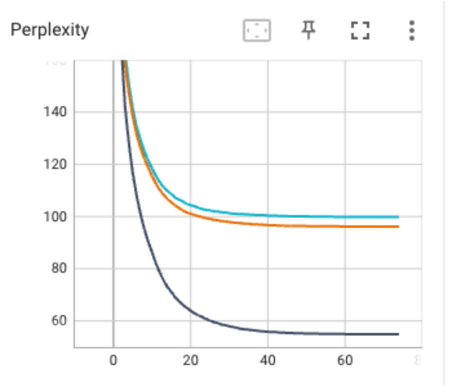
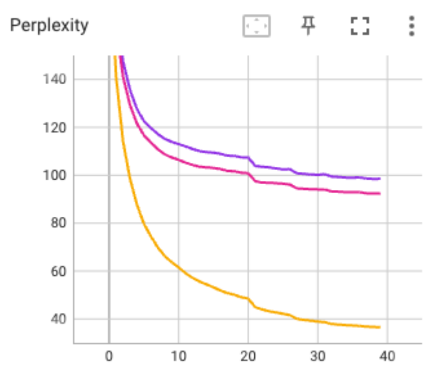
$$= \delta_{(2)} \cdot \left(1 - z_{(2)}\right) \cdot tanh'\left(W_{xg}{}^{T} \cdot x_{(2)} + W_{hg}{}^{T} \cdot \left(r_{(2)} \otimes h_{(1)}\right)\right) \cdot W_{hg}{}^{T} \cdot h_{(1)} \cdot \sigma'\left(W_{xr}{}^{T} \cdot x_{(t)} + W_{hr}{}^{T} \cdot h_{(t-1)}\right) \cdot x_{(2)}$$

h. $\frac{\partial \epsilon_{(2)}}{\partial W_{hr}} = \frac{\partial \epsilon_{(2)}}{\partial h_{(2)}} \cdot \frac{\partial h_{(2)}}{\partial g_{(2)}} \cdot \frac{\partial g_{(2)}}{\partial r_{(2)}} \cdot \frac{\partial r_{(2)}}{\partial W_{hr}} =$

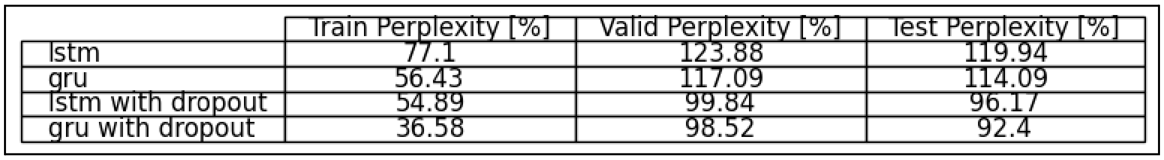
$$= \delta_{(2)} \cdot \left(1 - z_{(2)}\right) \cdot tanh'\left(W_{xg}{}^{T} \cdot x_{(2)} + W_{hg}{}^{T} \cdot \left(r_{(2)} \otimes h_{(1)}\right)\right) \cdot W_{hg}{}^{T} \cdot h_{(1)} \cdot \sigma'\left(W_{xr}{}^{T} \cdot x_{(t)} + W_{hr}{}^{T} \cdot h_{(t-1)}\right) \cdot h_{(1)}$$

Practical

Below is a table displaying the convergence graphs for test versus train perplexity for each technique utilized in the training of the LSTM and GRU networks.

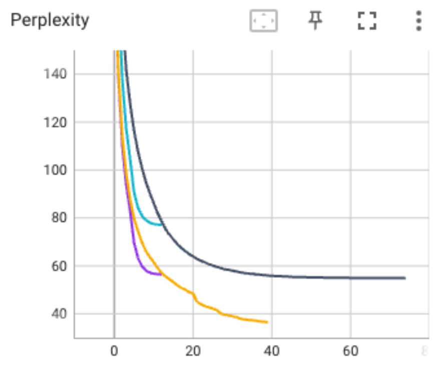
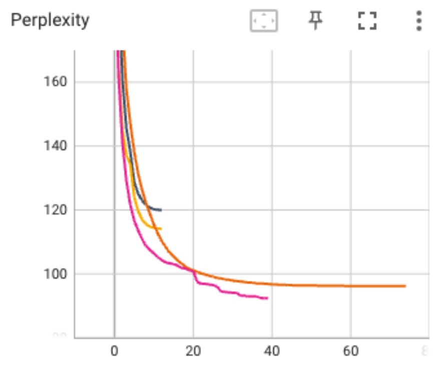
LSTM		**LSTM - Dropout**	
Optimizer: SGD **Learning Rate:** 1↓		**Dropout:** 0.5 **Optimizer:** SGD **Learning Rate:** 1↓	
 **Train:** train/RNN_lstm_dpFalse **Test:** test/train/RNN_lstm_dpFalse **Val:** val/train/RNN_lstm_dpFalse		 **Train:** train/RNN_lstm_dpTrue **Test:** test/train/RNN_lstm_dpTrue **Val:** val/train/RNN_lstm_dpTrue	
GRU		**GRU - Dropout**	
Optimizer: SGD **Learning Rate:** 1↓		**Dropout:** 0.5 **Optimizer:** Adam **Learning Rate:** 0.001↓	
 **Train:** train/RNN_gru_dpFalse **Test:** test/train/RNN_gru_dpFalse **Val:** val/train/RNN_gru_dpFalse		 **Train:** train/RNN_gru_dpFalse **Test:** test/train/RNN_gru_dpFalse **Val:** val/train/RNN_gru_dpFalse	

Summarized Table



	Train Perplexity [%]	Valid Perplexity [%]	Test Perplexity [%]
lstm	77.1	123.88	119.94
gru	56.43	117.09	114.09
lstm with dropout	54.89	99.84	96.17
gru with dropout	36.58	98.52	92.4

Conclusions

Train Set Perplexity	**Validation Set Perplexity**
 1. LSTM 2. GRU 3. LSTM + Dropout 4. GRU + Dropout	 1. LSTM 2. GRU 3. LSTM + Dropout 4. GRU + Dropout

Based on the results of the experiment comparing the performance of different Recurrent Neural Network (RNN) models, we can conclude the following:

- **Model Comparison**: The GRU models consistently outperformed the LSTM models across all three dataset partitions: training, validation, and testing. This might be due to the GRU's simpler architecture, which can often learn similar representations to LSTM with less risk of overfitting and in less time. However, this does not necessarily mean that GRUs are always superior to LSTMs, as the performance can vary depending on the specific task and dataset.

- **Effect of Dropout**: Adding dropout significantly improved the performance of both LSTM and GRU models. This suggests that the original models without dropout may have been overfitting to the training data. By using dropout, the models become more robust and generalize better to unseen data, as evidenced by the lower perplexity scores on the validation and test sets.

- **Training Time**: The LSTM with dropout took significantly more epochs to train (75) compared to the other models. This could be due to the additional complexity introduced by the dropout layers, which may slow down the convergence of the model. However, despite the longer training time, this model achieved better performance than the LSTM without dropout. The GRU with dropout, using the Adam optimizer, took fewer epochs (40) to train, suggesting that this combination might be more efficient.

- **Optimizer Impact**: The GRU model with dropout and Adam optimizer achieved the best results across all perplexity measures, suggesting that the Adam optimizer might be better suited for GRU in this task compared to SGD. Adam, which incorporates the benefits of other optimization methods with an adaptive learning rate, might provide better and quicker convergence in this case.

In summary, these results suggest that for this particular task and dataset, a GRU architecture with dropout and the Adam optimizer provides the best performance in terms of perplexity. Furthermore, the use of dropout appears to be an effective technique for improving the generalization of both LSTM and GRU models.

References

[1] LSTM — PyTorch 2.0 documentation

[2] Speed up your RNN with Sequence Bucketing | Kaggle

[3] Dynamic Recurrent Neural Network (LSTM) · TensorFlow Examples (aymericdamien) (gitbooks.io)

[4] What is Teacher Forcing for Recurrent Neural Networks? - MachineLearningMastery.com

[5] The Complete Guide To Moving EOS Tokens Off Of Exchanges | by Kevin Rose | EOS New York | Medium

[6] The Ultimate Showdown: RNN vs LSTM vs GRU - Which is the Best? - Shiksha Online