Due Date: March 10th 23:00, 2020

Instructions

- *For all questions, show your work!*
- *Submit your report (pdf) and your code (zip) electronically via the course Gradescope page.*
- *An outline of code will be provided in the course repo at this link. You must start from this outline and follow the instructions in it (even if you use different code, you must follow the overall outline and instructions).*
- *TAs for this assignment are Jessica Thompson, Jonathan Cornford and Lluis Castrejon.*

Summary:

In this assignment, you will implement and train **sequential language models** on the Penn Treebank dataset. Language models learn to assign a likelihood to sequences of text. The elements of the sequence (typically words or individual characters) are called tokens, and can be represented as one-hot vectors with length equal to the vocabulary size, e.g. 26 for a vocabulary of English letters with no punctuation or spaces, in the case of characters, or as indices in the vocabulary for words. In this representation an entire dataset (or a mini-batch of examples) can be represented by a 3-dimensional tensor, with axes corresponding to: (1) the example within the dataset/mini-batch, (2) the time-step within the sequence, and (3) the index of the token in the vocabulary. Sequential language models do **next-step prediction**, in other words, they predict tokens in a sequence one at a time, with each prediction based on all the previous elements of the sequence. A trained sequential language model can also be used to generate new sequences of text, by making each prediction conditioned on the past *predictions* (instead of the ground-truth input sequence).

As a starting point, you are provided with an implementation of a **simple ("vanilla") RNN** (recurrent neural network). Problem 1 is to implement an RNN with a gating mechanism on the hidden state, specifically with **gated recurrent units (GRUs)**. Problem 2 is to implement the **attention module of a transformer network** (we provide you with PyTorch code for the rest of the transformer). Problem 3 is to train these 3 models using a variety of different optimizers and hyperparameter settings and Problem 4 is to analyze the behaviour of the trained models. Each problem is worth 25 points.

The Penn Treebank Dataset This is a dataset of about 1 million words from about 2,500 stories from the Wall Street Journal. It has Part-of-Speech annotations and is sometimes used for training parsers, but it's also a very common benchmark dataset for training RNNs and other sequence models to do next-step prediction.

Preprocessing: The version of the dataset you will work with has been preprocessed: lower-cased, stripped of non-alphabetic characters, tokenized (broken up into words, with sentences separated by the <eos> (end of sequence) token), and cut down to a vocabulary of 10,000 words; any word not in this vocabulary is replaced by <unk>. For the transformer network, positional

information (an embedding of the position in the source sequence) for each token is also included in the input sequence. In both cases the preprocessing code is given to you.

Problem 1

Implementing an RNN with Gated Recurrent Units (GRU) (25pts) The implementation of your RNN must be able to process mini-batches. Implement the model **from scratch** using PyTorch Tensors, Variables, and associated operations (e.g. as found in the `torch.nn` module). Specifically, use appropriate matrix and tensor operations (e.g. dot, multiply, add, etc.) to implement the recurrent unit calculations; you **may not** use built-in Recurrent modules. You **may** subclass `nn.module`, use built-in Linear modules, and built-in implementations of nonlinearities (tanh, sigmoid, and softmax), initializations, loss functions, and optimization algorithms. Your code must start from the code scaffold and follow its structure and instructions.

The use of "gating" (i.e. element-wise multiplication, represented by the $\odot$ symbol) can significantly improve the performance of RNNs. The Long-Short Term Memory (LSTM) RNN is the best known example of gating in RNNs; GRU-RNNs are a slightly simpler variant (with fewer gates).

The equations for a GRU are:

$$\boldsymbol{r}_t = \sigma_r(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1} + \boldsymbol{b}_r) \tag{1}$$
$$\boldsymbol{z}_t = \sigma_z(\boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1} + \boldsymbol{b}_z) \tag{2}$$
$$\tilde{\boldsymbol{h}}_t = \sigma_h(\boldsymbol{W}_h \boldsymbol{x}_t + \boldsymbol{U}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h) \tag{3}$$
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \tag{4}$$
$$P(\boldsymbol{y}_t | \boldsymbol{x}_1, ..., \boldsymbol{x}_t) = \sigma_y(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y) \tag{5}$$

$\boldsymbol{r}_t$ is called the "reset gate" and $\boldsymbol{z}_t$ the "forget gate". The trainable parameters are $\boldsymbol{W}_r, \boldsymbol{W}_z, \boldsymbol{W}_h, \boldsymbol{W}_y$, $\boldsymbol{U}_r, \boldsymbol{U}_z, \boldsymbol{U}_h, \boldsymbol{b}_r, \boldsymbol{b}_z, \boldsymbol{b}_h$, and $\boldsymbol{b}_y$, as well as the initial hidden state parameter $\boldsymbol{h}_0$. GRUs use the sigmoid activation function for σ_r and σ_z, and tanh for σ_h.

See further instructions in the solution template.

Problem 2

Implementing the attention module of a transformer network (25pts) While prototypical RNNs "remember" past information by taking their previous hidden state as input at each step, recent years have seen a profusion of methodologies for making use of past information in different ways. The transformer [1] is one such fairly new architecture which uses several self-attention networks ("heads") in parallel, among other architectural specifics. The transformer is quite complicated to implement compared to the RNNs described so far; most of the code is provided and your task is only to implement the multi-head scaled dot-product attention. The attention vector for m heads

[1] See `https://arxiv.org/abs/1706.03762` for more details.

indexed by i is calculated as follows:

$$\boldsymbol{A}_i = \text{softmax}\left(\frac{\boldsymbol{Q}_i \boldsymbol{W}_{Q_i} (\boldsymbol{K}_i \boldsymbol{W}_{K_i})^\top}{\sqrt{d_k}}\right) \tag{6}$$

$$\boldsymbol{H}_i = \boldsymbol{A}_i \boldsymbol{V} \boldsymbol{W}_{V_i} \tag{7}$$

$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{concat}(\boldsymbol{H}_1, ..., \boldsymbol{H}_m)\boldsymbol{W}_O \tag{8}$$

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are queries, keys, and values respectively, $\boldsymbol{W}_{Q_i}, \boldsymbol{W}_{K_i}, \boldsymbol{W}_{V_i}$ are their corresponding embedding matrices, $\boldsymbol{W}_O$ is the output embedding, and d_k is the dimension of the keys. $\boldsymbol{Q}, \boldsymbol{K}$, and $\boldsymbol{V}$ are determined by the output of the feed-forward layer of the main network (given to you). $\boldsymbol{A}_i$ are the attention values, which specify which elements of the input sequence each attention head attends to.

Note that the implementation of multi-head attention requires binary masks, so that attention is computed only over the past, not the future. A mask value of 1 indicates an element which the model is allowed to attend to (i.e. from the past); a value of 0 indicates an element it is not allowed to attend to. This can be implemented by modifying the softmax function to account for the mask $\boldsymbol{s}$ as follows:

$$\tilde{\boldsymbol{x}} = \exp(\boldsymbol{x}) \odot \boldsymbol{s} \tag{9}$$

$$\text{softmax}(\boldsymbol{x}, \boldsymbol{s}) \doteq \frac{\tilde{\boldsymbol{x}}}{\sum_i \tilde{x}_i} \tag{10}$$

To avoid potential numerical stability issues, we recommend a different implementation:

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} \odot \boldsymbol{s} - 10^9(1 - \boldsymbol{s}) \tag{11}$$

$$\text{softmax}(\boldsymbol{x}, \boldsymbol{s}) \doteq \frac{\exp(\tilde{\boldsymbol{x}})}{\sum_i \exp(\tilde{x}_i)} \tag{12}$$

This second version is equivalent (up to numerical precision) as long as $\boldsymbol{x} >> -10^9$, which should be the case in practice.

Problem 3

Training language models and model comparison (25pts) Unlike in classification problems, where the performance metric is typically accuracy, in language modelling, the performance metric is typically based directly on the cross-entropy loss, i.e. the negative log-likelihood (NLL) the model assigns to the tokens. For word-level language modelling it is standard to report **perplexity (PPL)**, which is the exponentiated average per-token NLL (over all tokens):

$$\exp\left(\frac{1}{TN}\sum_{t=1}^{T}\sum_{n=1}^{N} -\log P(\boldsymbol{x}_t^{(n)} | \boldsymbol{x}_1^{(n)}, ..., \boldsymbol{x}_{t-1}^{(n)})\right),$$

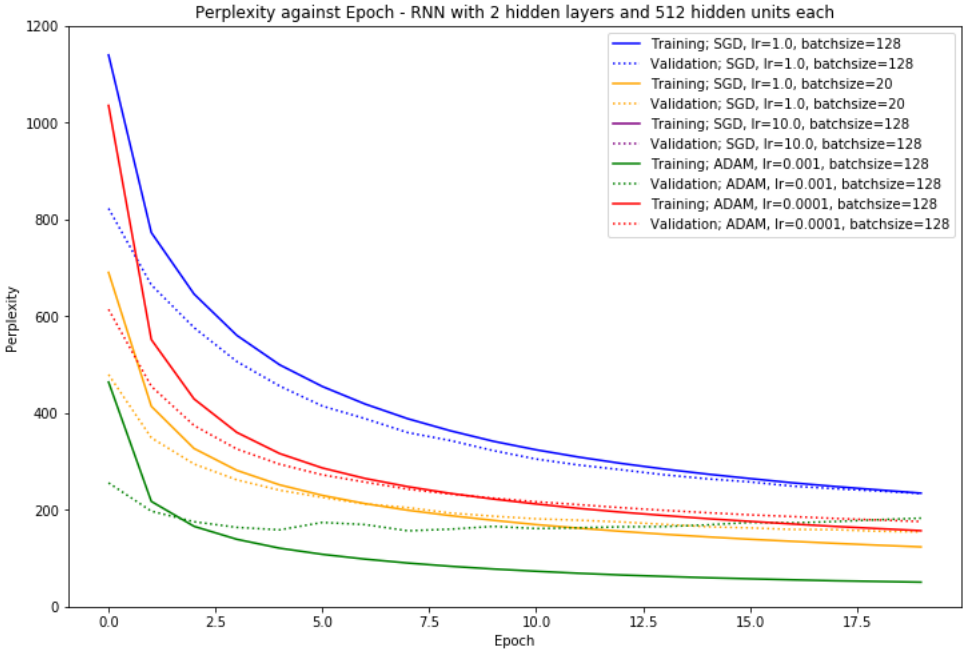
where t is the index with the sequence, and n indexes different sequences. For Penn Treebank in particular, the test set is treated as a single sequence (i.e. $N = 1$). The purpose of this assignment is to perform model exploration, which is done using a validation set. As such, we do not require you to run your models on the test set.

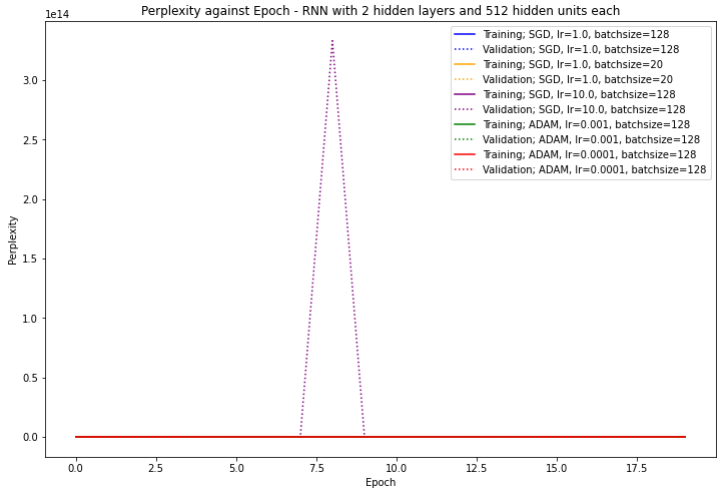
You will train each of the three architectures using either stochastic gradient descent or the ADAM optimizer. The training loop is provided in *run_exp.py*.

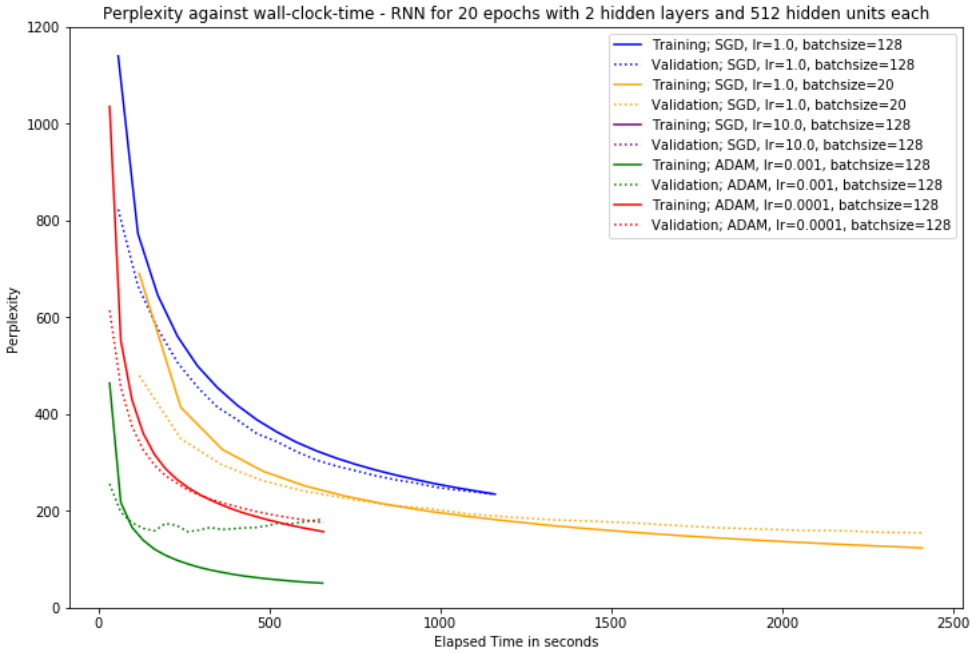
1. - 4. You are asked to run 4 experiments (3.1, 3.2, 3.3, 3.4) with different architectures, optimizers, and hyperparameters settings. These parameter settings are given to you in the code (*run_exp.py*). In total there are 12 settings for you to run ($4+3+3+4=14$). For each experiment (3.1, 3.2, 3.3, 3.4), plot learning curves (train and validation) of PPL over both **epochs** and **wall-clock-time**. Figures should have labeled axes and a legend and an explanatory caption.

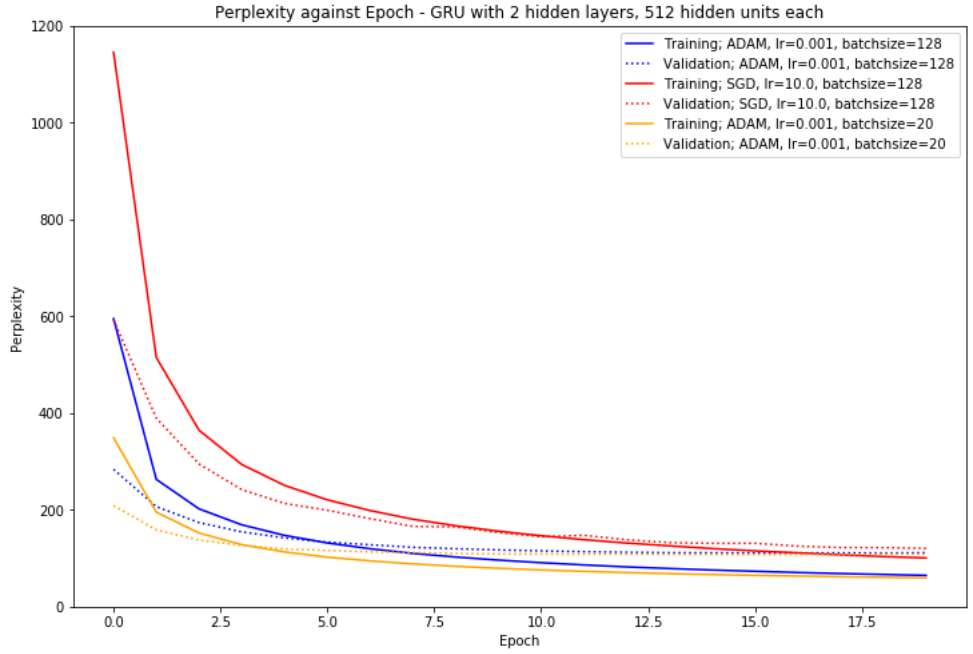
Answer

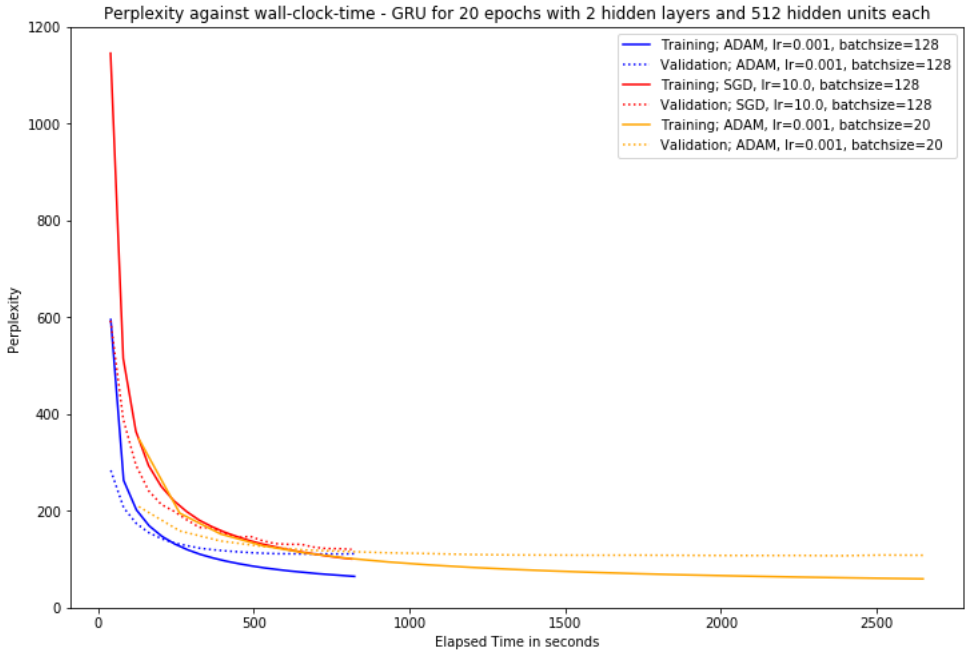
Note that the first and second plots are the same, just with a different y-axis scale.

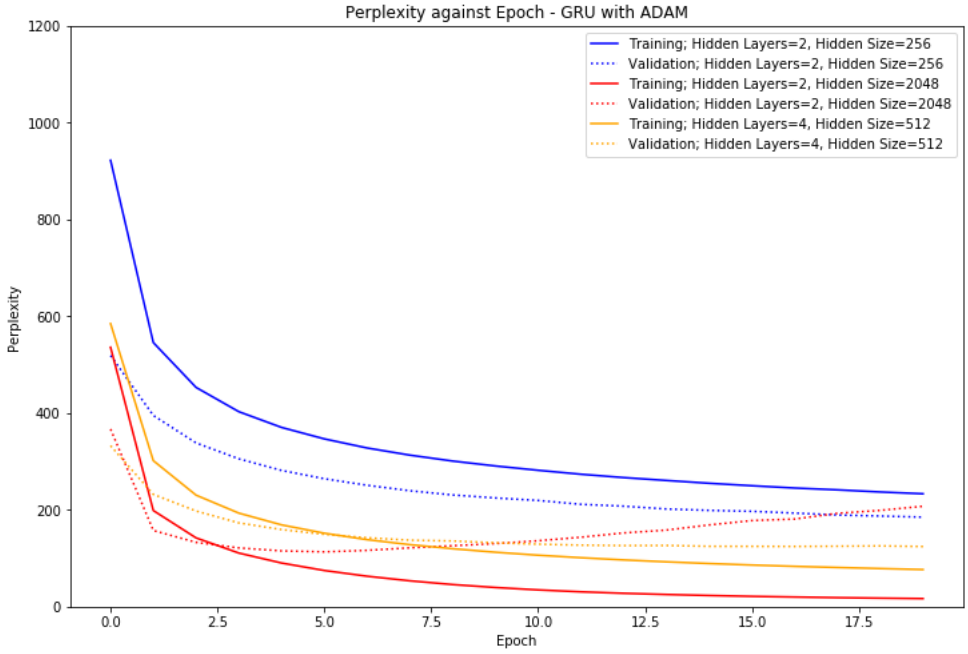


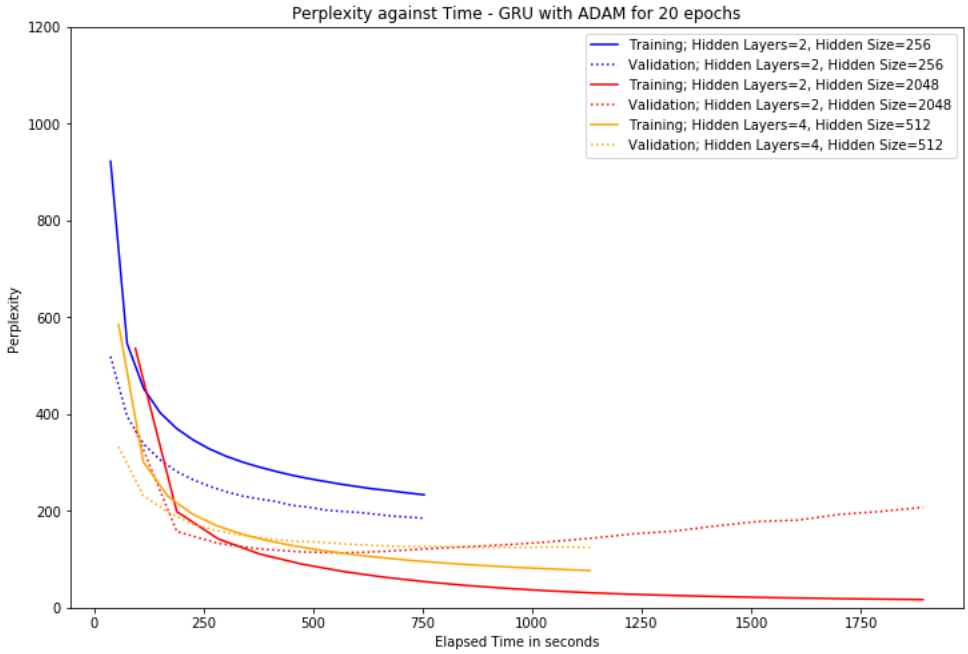


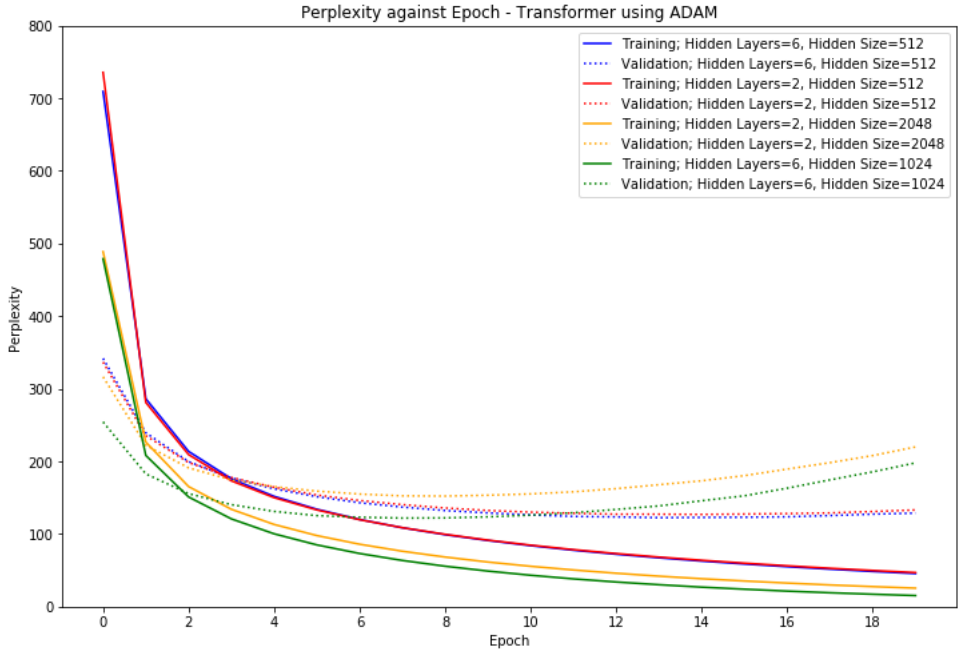

Perplexity against wall-clock-time - RNN for 20 epochs with 2 hidden layers and 512 hidden units each
Training; SGD, lr=1.0, batchsize=128
Validation; SGD, lr=1.0, batchsize=128
Training; SGD, lr=1.0, batchsize=20
Validation; SGD, lr=1.0, batchsize=20
Training; SGD, lr=10.0, batchsize=128
Validation; SGD, lr=10.0, batchsize=128
Training; ADAM, lr=0.001, batchsize=128
Validation; ADAM, lr=0.001, batchsize=128
Training; ADAM, lr=0.0001, batchsize=128
Validation; ADAM, lr=0.0001, batchsize=128
Perplexity
Elapsed Time in seconds
0
200
400
600
800
1000
1200
0
500
1000
1500
2000
2500

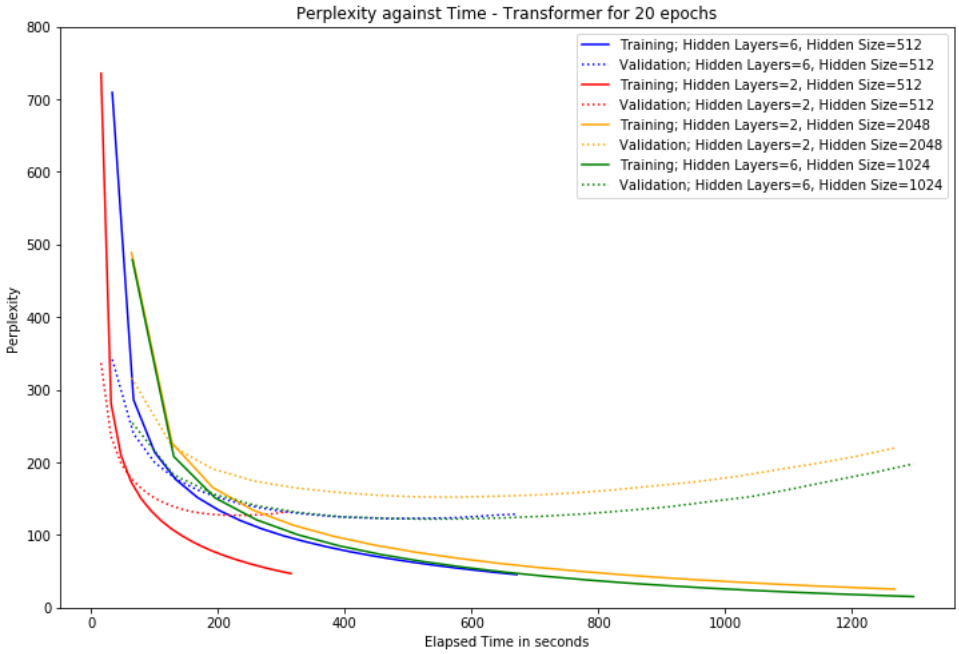

Perplexity against Epoch - GRU with 2 hidden layers, 512 hidden units each
Training; ADAM, lr=0.001, batchsize=128
Validation; ADAM, lr=0.001, batchsize=128
Training; SGD, lr=10.0, batchsize=128
Validation; SGD, lr=10.0, batchsize=128
Training; ADAM, lr=0.001, batchsize=20
Validation; ADAM, lr=0.001, batchsize=20
Perplexity
Epoch
0
200
400
600
800
1000
1200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Perplexity against wall-clock-time - GRU for 20 epochs with 2 hidden layers and 512 hidden units each
Training; ADAM, lr=0.001, batchsize=128
Validation; ADAM, lr=0.001, batchsize=128
Training; SGD, lr=10.0, batchsize=128
Validation; SGD, lr=10.0, batchsize=128
Training; ADAM, lr=0.001, batchsize=20
Validation; ADAM, lr=0.001, batchsize=20
Perplexity
Elapsed Time in seconds
0
200
400
600
800
1000
1200
0
500
1000
1500
2000
2500


Perplexity against Epoch - GRU with ADAM
Training; Hidden Layers=2, Hidden Size=256
Validation; Hidden Layers=2, Hidden Size=256
Training; Hidden Layers=2, Hidden Size=2048
Validation; Hidden Layers=2, Hidden Size=2048
Training; Hidden Layers=4, Hidden Size=512
Validation; Hidden Layers=4, Hidden Size=512
Perplexity
Epoch
0
200
400
600
800
1000
1200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Perplexity against Time - GRU with ADAM for 20 epochs
Training; Hidden Layers=2, Hidden Size=256
Validation; Hidden Layers=2, Hidden Size=256
Training; Hidden Layers=2, Hidden Size=2048
Validation; Hidden Layers=2, Hidden Size=2048
Training; Hidden Layers=4, Hidden Size=512
Validation; Hidden Layers=4, Hidden Size=512
Perplexity
Elapsed Time in seconds
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750


Perplexity against Epoch - Transformer using ADAM
Training; Hidden Layers=6, Hidden Size=512
Validation; Hidden Layers=6, Hidden Size=512
Training; Hidden Layers=2, Hidden Size=512
Validation; Hidden Layers=2, Hidden Size=512
Training; Hidden Layers=2, Hidden Size=2048
Validation; Hidden Layers=2, Hidden Size=2048
Training; Hidden Layers=6, Hidden Size=1024
Validation; Hidden Layers=6, Hidden Size=1024
Perplexity
Epoch
0
100
200
300
400
500
600
700
800
0
2
4
6
8
10
12
14
16
18



5. Make a table of results summarizing the train and validation performance for each experiment, indicating the architecture and optimizer. Sort by architecture, then optimizer, and number the experiments to refer to them easily later. Bold the best result for each architecture.[2] The table should have an explanatory caption, and appropriate column and/or row headers. Any shorthand or symbols in the table should be explained in the caption.

Answer

In the following table, Exp is the experiment number, LR is the learning rate, BS is the batch size, HS is the hidden size, layers is the number of hidden layers, KPROB is the keep probability, PPL is the Perplexity as defined in Problem 3, and train and valid PPL is the training and validation perplexity.

[2]You can also make the table in LaTeX, but you can also make it using Excel, Google Sheets, or a similar program, and include it as an image.

Architecture	Optimizer	Exp	LR	BS	HS	Layers	KPROB	Train PPL	Valid PPL
RNN	SGD	3.1	1.0	128	512	2	0.8	233.9796	233.3213
RNN	SGD	3.1	1.0	20	512	2	0.8	122.9286	**153.9474**
RNN	SGD	3.1	10.0	128	512	2	0.8	12413.12139	3099.0898
RNN	ADAM	3.1	0.001	128	512	2	0.8	50.0481	156.19749
RNN	ADAM	3.1	0.0001	128	512	2	0.8	156.3897	175.3674
GRU	SGD	3.2	10.0	128	512	2	0.5	100.0399	119.7932
GRU	ADAM	3.2	0.001	128	512	2	0.5	76.0617	123.6852
GRU	ADAM	3.2	0.001	20	512	2	0.5	58.9231	**106.7096**
GRU	ADAM	3.3	0.001	128	256	2	0.2	232.6856	184.3798
GRU	ADAM	3.3	0.001	128	2048	2	0.5	16.0886	112.6486
GRU	ADAM	3.3	0.001	128	512	4	0.5	76.0617	123.6852
Transformer	ADAM	3.4	0.0001	128	512	6	0.9	45.2376	122.2014
Transformer	ADAM	3.4	0.0001	128	512	2	0.9	46.5893	126.6517
Transformer	ADAM	3.4	0.0001	128	2048	2	0.6	25.1016	151.9283
Transformer	ADAM	3.4	0.0001	128	1024	6	0.9	14.8952	**121.6143**

Table 1:
Training and validation Perplexity for each experiment, architectures, optimizers, and hyperparameters

6. Which hyperparameters + optimizer would you use if you were most concerned with wall-clock time, with generalization performance.

   Answer

   I would use the ADAM optimizer since it accelerates training because of the adaptive learning rates and the exponential moving average of the gradients, with a batch size of around 128, and a learning rate of around 0.01. While a smaller learning rate allows the model to learn a more optimal or globally optimal set of weights, it takes longer to train since the learning rate controls the speed of learning. Also, a smaller batch size means more update per epoch and a higher number of total batches, which while it may give faster convergence it may also take more time as there are more updates to be made. Note that GPUs allow computational speedups because of their parallelism, meaning that they are able to take bigger batch size. As you can see, there is a trade-off between speed and generalization performance, especially for the learning rate and the batch size.

7. For exp 3.1 you trained an RNN with either SGD or ADAM. What did you notice about the optimizer's performance with different learning rates?

   Answer

   A crucial parameter for SGD is the learning rate. If it is too large, the learning curve will show violent oscillations (as shown in plot 2 with a learning rate of 10.0), with the cost function increasing significantly. Thus, with SGD, smaller learning rate achieved better generalization performance. This can be explained by the fact that with a large learning rate, we are taking big steps that may result in sub-optimal solution. For the ADAM optimizer, the suggested

learning rate of 0.001 (seen in the Deep learning book [3]) achieved better generalization performance after a few epochs, but then performed similarly to the model with ADAM optimizer and a learning rate of 0.0001. We can also see that the ADAM optimizer with a learning rate of 0.001 had a bit more volatility in the validation performance.

8. For exp 3.2 you trained a GRU. Was its performance as you expected and why?

Answer

Its performance is what I expected since the GRU gates can better capture long-term relationships of the input than the RNN. Thus, all of the different GRU models achieved better validation performance than the RNN models tried earlier.

9. In exp 3.3 you explored different hyperparameter settings in an attempt to improve the performance of the GRU. Were the validation/training curves as you expected for each setting? Comment on why. *Hint: For each hyperparameter setting, consider how the training and validation phases differ.*

Answer

The model with hidden size of 2048 units and drop out keep probability of 0.5 shows sign of overfitting, as expected with the high number of hidden units. Indeed, the plot shows a really low training perplexity while the validation perplexity starts to decrease and then increases as the number of epochs grows. For the model with 2 hidden layers, 256 hidden units and a dropout keep probability of 0.2, I expected better generalization performance than the model with 4 hidden layers and 512 hidden units because of its lower capacity, thus less overfitting. However, it is not what the plots show, which suggests that the problem at hand needed a more complex model. Nevertheless, the architecture with 2 hidden layers, 256 hidden units and a dropout keep probability of 0.2 is the only model with a better validation performance than its training perplexity. This can be explained by the low drop out keep probability. Indeed, during training we are disabling neurons, which means that some information is lost and the subsequent layers attempt to construct answers based on incomplete representations. However, during validation all of the units are available so the network has all of its computational power, which explains why it performs better than during training.

10. In exp 3.4 you trained a Transformer with various hyper-parameter settings. Given the recent high profile transformer based language models, are the results as you expected? Speculate as to why or why not.

Answer

All of the plots show signs of overfitting, with the training perplexity decreasing and the validation perplexity first decreasing, then increasing as the number of epochs gets bigger. It is what I expected since the Transformer architecture has a high capacity. Its capacity seems to be too high compared to our task and the number of training examples we have.

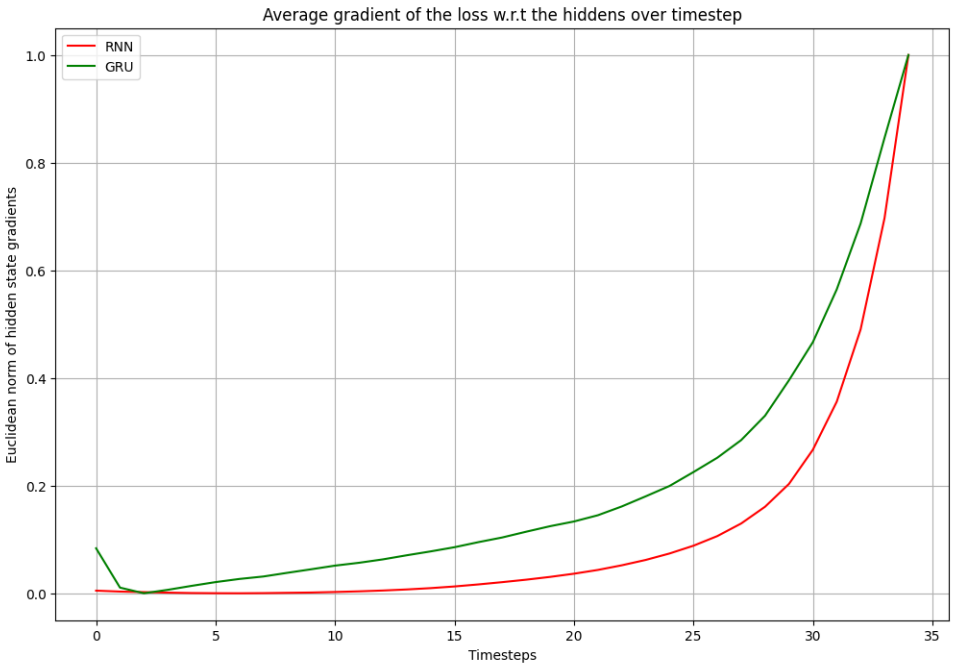
[3] https://www.deeplearningbook.org/contents/optimization.html

Problem 4

Detailed evaluation of trained models (25pts) For this problem, we will investigate properties of the trained models from Problem 3. Perform the following evaluations for the two models (one RNN and one GRU) for which the parameters were saved (indicated by the flag –save_best in the code).

1. For one minibatch of training data, compute the average gradient of the loss at the *final* time-step with respect to the hidden state at *each* time-step t: $\nabla_{\boldsymbol{h}_t}\mathcal{L}_T$. The norm of these gradients can be used to evaluate the propagation of gradients; a rapidly decreasing norm means that longer-term dependencies have less influence on the training signal, and can indicate **vanishing gradients**. Plot the Euclidian norm of $\nabla_{\boldsymbol{h}_t}\mathcal{L}_T$ as a function of t for the RNN and GRU architectures. Rescale the values of each curve to [0,1] so that you can compare both on one plot. Describe the results qualitatively, and provide an explanation for what you observe, discussing what the plots tell you about the gradient propagation in the different architectures.

Answer



From the above plot, we can observe that the gradients of previous hidden units are getting smaller and smaller for both architecture, meaning that longer-term dependencies have less influence on the training signal. However, we see that gradients from the RNN are decreasing faster than the ones from the GRU architecture, which means that the RNN has a less memory about inputs from previous time steps than the GRU. This is in fact what is expected from this architecture, as the gated recurrent units were designed to combat vanishing gradients. Indeed, the benefit of the GRU architecture comes from the update and reset gate. The update

gate helps the model determine how much information from previous time steps should be passed along, while the reset gate decide how much previous information should be forgotten.

2. Generate samples from both the RNN and GRU models, by recursively making $\hat{\boldsymbol{x}}_{t+1} = \arg\max P(\boldsymbol{x}_{t+1}|\hat{\boldsymbol{x}}_1, ..., \hat{\boldsymbol{x}}_t)$.[4] Make sure to condition on the sampled $\hat{\boldsymbol{x}}_t$, *not* the ground truth. Produce 20 samples from both the RNN and GRU: 10 sequences of the same length as the training sequences, and 10 sequences of *twice* the length of the training sequences. Do you think that the generated sequence quality correlates with model validation perplexity? Justify your answer.

   Choose 3 "best", 3 "worst", and 3 that are "interesting". Put all 40 samples in an appendix to your report.

Answer

3 best sentences

In these sentences, even though they are not what I would qualify good sentences, the models seem to make association between words such as billion and pesetas which is a spanish currency, million with $ sign and cents, and some conjugations of verb tense that makes sense, like in the sentence "the company is expected to be acquired by the end".

(a) (GRU) than N N of the total of the N billion pesetas <eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of <unk><unk><eos>the <unk><unk><unk><unk><unk>

(b) (GRU) 's <unk><unk><unk><unk><unk>and <unk><unk><eos>the company 's <unk>division is a <unk>of the company 's <unk>division <eos>the company is expected to be acquired by the end

(c) (RNN) 's net income of $ N million or N cents a share <eos>the company said it was a <unk>N N stake in the first quarter <eos>the company said it was a <unk>

3 worst sentences

In these sentences, we see that the models generated end of sentence token like <eos>and consecutive unknown tokens. This indicates that they were not able to predict well based on past predictions if the token was a stopword like "the" and "to".

(a) (RNN)
of the <unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>
<eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk>
<unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk><unk><unk>
<unk><unk><unk><unk><unk><unk><eos>the
<unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk><unk>
<unk><unk><unk><unk><unk><unk><unk>

[4] It is possible to generate samples in the same manner from the Transformer, but the implementation is more involved, so you are not required to do so.

(b) (GRU) applies to <unk>the <unk>of the <unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk>and
<unk><unk><eos>the <unk><unk><unk>

(c) (RNN)
as well as the <unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk>
<unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk>
<eos>the <unk><unk><unk><unk>

3 interesting sentences

In these sentences, we can see that the models made association with verb negation like "hasn't" , "wasn't", "i'm not", made somewhat plausible sentence such as "the rest of the past century" and seems to link "wasn't" with "interested" and "hasn't" with "been notified". Also, with the last sentence of the 3 interesting ones, the model seems to be repeating its predictions.

(a) (GRU)
's <unk><unk><unk>unit <eos>the company said it was n't interested in the company <eos>the company said it has n't been notified by the company <eos>the company said it was n't interested in the company <eos>the company said it has n't been notified by the filing <eos>the company said it was n't interested in the company <eos>the company said it has n't been

(b) (GRU)
the rest of the past century <eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk><unk>

(c) (GRU)
'm glad to be a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get

Overall, the two models that generated these sequences were the ones that we were told to save in the run_exp.py script. The quality of the sentences generated somewhat reflects the validation perplexity of those two models. In fact, the GRU model generated better sentences than the RNN. However, while both model have a big validation perplexity gap (the RNN model has a validation perplexity of 233,32 and the GRU has a validation perplexity of 123,68), I expected a bigger difference in the quality of the sentences. This may be explained by the fact that we are conditioning on the sampled $\hat{x}_t$ and not the ground truth.

Appendix, sequences generated by RNN and GRU

10 sequences of length 35 generated by RNN

Seq 1 : than N N <eos>the company said the company was n't <unk><eos>the company said it will be a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said

Seq 2 : income of the company 's stock <eos>the company said it is a<unk><unk><eos>the company said it will be a <unk><unk><eos>the company said it is a <unk><unk><eos>

Seq 3 : 's net income of $ N million or N cents a share <eos>the company said it was a <unk>N N stake in the first quarter <eos>the company said it was a <unk>

Seq 4 : 're n't know the <unk><eos>the<unk>is a <unk>

<unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>the <unk><unk><unk><unk><unk><unk><unk><unk><unk>

Seq 5 : 're n't know the <unk><eos>the<unk>is a <unk>

<unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>the <unk><unk><unk><unk><unk><unk><unk><unk><unk>

Seq 6 : they 're <unk>to be <unk><eos >the <unk>is n't <unk>
<eos>the <unk>is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a

Seq 7 : of the <unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>
<eos>the <unk><unk><unk>
<unk><unk><unk><unk><unk>
<unk><eos>the <unk><unk><unk><unk><unk><unk><unk>

Seq 8 : the same quarter <eos>the company said it is n't <unk>to the <unk><eos>the company said it is n't <unk><eos>the company said it will be a <unk><unk><eos>the company

Seq 9 : 's net income <eos>the company said it will be a <unk><unk><eos>the company said it will be a <unk><unk><eos>the company said it is a <unk><unk><eos>the company

Seq 10 : as well as the <unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk>
<unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the
<unk><unk><unk><unk>

10 sequences of length 70 generated by RNN

Seq 1 : than N N <eos>the company said the company was n't <unk><eos>the company said it will be a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company

Seq 2 : income of the company 's stock <eos>the company said it is a <unk><unk><eos>the company said it will be a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk>

Seq 3 : 's net income of $ N million or N cents a share <eos>the company said it was a <unk>N N stake in the first quarter <eos>the company said it was a <unk>N N stake in the first quarter <eos>the company said it was a <unk>N N stake in the first quarter <eos>the company said it was a <unk>N N stake in the

Seq 4 : 're n't know the <unk><eos>the <unk>is a <unk><unk><eos>the <unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk><unk><unk>
<unk><unk><unk><unk><unk><unk>
<eos>the <unk><unk><unk><unk><unk><unk><unk><unk>
<unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the
<unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the

Seq 5 : 're n't know the <unk><eos>the <unk>is a <unk><unk><eos>the <unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk><unk><unk>
<unk><unk><unk><unk><unk><unk>
<eos>the <unk><unk><unk><unk><unk><unk><unk><unk>
<unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the
<unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the

Seq 6 : they 're <unk>to be <unk><eos>the <unk>is n't <unk><eos>the <unk>is a
<unk><unk><eos>the company said it is a
<unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a
<unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a
<unk><unk><eos>the company said it is

Seq 7 : of the <unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>
<eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk>
<unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk><unk><unk>
<unk><unk><unk><unk><unk><unk><eos>the
<unk><unk><unk><unk><unk><unk><unk><unk><unk><eos>the <unk><unk>
<unk><unk><unk><unk><unk><unk><unk>

Seq 8 : the same quarter <eos>the company said it is n't <unk>to the <unk><eos>the company said it is n't <unk><eos>the company said it will be a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a

<unk><unk><eos>the company said it is a <unk><unk><eos>the

Seq 9 : 's net income <eos>the company said it will be a <unk><unk><eos>the company said it will be a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the company said it is a <unk><unk><eos>the

Seq 10 : as well as the <unk><unk><eos>the <unk><unk><unk><unk>
<unk><unk><unk><unk><unk><unk><eos>the <unk><unk><unk><unk><unk><unk>
<unk><unk><unk><eos>the <unk><unk><unk><unk><unk><unk>
<unk><unk><unk><eos>the <unk><unk><unk><unk>
<unk><unk><unk><unk><unk><eos>the <unk>
<unk><unk><unk><unk><unk><unk><unk>
<unk><eos>the <unk><unk><unk><unk><unk><unk>

10 sequences of length 35 generated by GRU

Seq 1 : than N N of the total of the N billion pesetas <eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of <unk><unk><eos>the <unk><unk><unk><unk><unk>

Seq 2 : lire <eos>the company 's <unk><unk><unk><unk><unk><unk><unk>and <unk><unk>and <unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>and <unk><unk><eos>the

Seq 3 : 's <unk><unk><unk>unit <eos>the company said it was n't interested in the company <eos>the company said it has n't been notified by the company <eos>the company said it was n't

Seq 4 : 'm glad to be a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going

Seq 5 : 'm glad to be a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going

Seq 6 : applies to <unk>the <unk>of the <unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk>and
<unk><unk><eos>the <unk><unk><unk>

Seq 7 : of the <unk>of the <unk><unk><eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk>

Seq 8 : the rest of the past century <eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk><unk>

Seq 9 : 's <unk><unk><unk><unk><unk>and <unk><unk><eos>the company 's <unk>division

is a <unk>of the company 's <unk>division <eos>the company is expected to be acquired by the end

Seq 10 : as <unk><unk>and <unk><unk><eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk>is a <unk>of the

10 sequences of length 70 generated by GRU

Seq 1 : than N N of the total of the N billion pesetas <eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of <unk><unk><eos>the
<unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>and <unk><unk><eos>the
<unk><unk><unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk><unk>and <unk><unk><eos>the <unk><unk>

Seq 2 : lire <eos>the company 's <unk><unk><unk><unk><unk><unk><unk>and <unk><unk>and
<unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>and
<unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk><unk><unk>and <unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk><unk>and <unk><unk><eos>the <unk><unk>

Seq 3 : 's <unk><unk><unk>unit <eos>the company said it was n't interested in the company <eos>the company said it has n't been notified by the company <eos>the company said it was n't interested in the company <eos>the company said it has n't been notified by the filing <eos>the company said it was n't interested in the company <eos>the company said it has n't been

Seq 4 : 'm glad to be a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get

Seq 5 : 'm glad to be a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get a lot of <unk><eos>i 'm not going to get out of the <unk><eos>i 'm not going to get

Seq 6 : applies to <unk>the <unk>of the <unk><unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk><unk>
<unk>and <unk><unk><eos>the <unk><unk><unk><unk><unk><unk><unk>
<unk><unk><unk><unk><unk><unk><unk><unk>and <unk><unk><eos>the
<unk><unk><unk><unk><unk><unk><unk><unk><unk>
<unk><unk><unk><unk>and <unk><unk><eos>the

Seq 7 : of the <unk>of the <unk><unk><eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk>is a <unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk>

Seq 8 : the rest of the past century <eos>the <unk>of the <unk><unk>is a <unk>of the

<unk><unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk><eos>the

Seq 9 : 's <unk><unk><unk><unk><unk>and <unk><unk><eos>the company 's <unk>division is a <unk>of the company 's <unk>division <eos>the company is expected to be acquired by the end of the year <eos>the company said it will sell its N N stake in the u.s. and <unk>of the u.s. <eos>the company said it will redeem N million shares of N N

Seq 10 : as <unk><unk>and <unk><unk><eos>the <unk>of the <unk><unk>is a <unk>of the <unk><unk>of the <unk><unk><eos>the <unk>of the <unk>is a <unk>of the <unk><unk>of <unk><unk><eos>the <unk><unk>is a <unk><unk>of <unk><unk>and <unk><unk><eos>the <unk><unk>is a <unk><unk>of <unk><unk>and <unk><unk><eos>the