Due Date: March 25th 23:59, 2019

Note Key codes are included in this report.

For full implementation, please refer to: https://github.com/qqiang00/IFT6135/blob/master/hw2/ after due date.

The following files are included:

1. models.py: implementation of RNN, GRU, TRANSFORMER for problem1-3; including the method for generating sequences by RNN and GRU.
2. problem4.ipynb: codes for problem4
3. problem5.ipynb: codes for problem5
4. helpers.py: including methods originally in ptb-lm.py, like ptb_iterator, etc.

Team member

1. Qiang Ye (20139927)
2. Ying Xiao (20111402)
3. Boumaza Amel (20126028)
4. Yunhe Li (20137167)

Instructions

- *For all questions, show your work!*
- *Starred questions are **hard** questions, not **bonus** questions.*
- *Submit your report (pdf) electronically via the course Gradescope page.*
- *You must push (all of!) your code to a Github repository, and include the link to the repository in your report (even if using a Jupyter notebook)*
- *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 David Krueger, Tegan Maharaj, and Chin-Wei Huang.*

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).

Problems 1-3 are respectively the implementation of (1) a **simple ("vanilla") RNN** (recurrent neural network), (2) an RNN with a gating mecahnism on the hidden state, specifically with **gated recurrent units (GRUs)**, and (3) the **attention module of a transformer network** (we provide you with Pytorch code for the rest of the transformer). Problem 4 is to train these 3 models using a variety of different optimizers and hyperparameter settings. Problem 5 involves analyzing the behavior of the trained models in more detail. Each problem is worth 20 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: lowercased, 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.

Instructions for Problems 1 and 2:

You will manually implement two types of recurrent neural network (RNN). The implementation must be able to process mini-batches. Implement the models **from scratch** using Pytorch/Tensorflow 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.

Problem 1

Implementing a Simple RNN (20pts) A simple recurrent neural network (SRNN) is also called a "vanilla" RNN or "Elman network". The equations for an SRNN are:

$$P(\boldsymbol{y}_t|\boldsymbol{x}_1, ..., \boldsymbol{x}_t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y) \tag{1}$$

$$\boldsymbol{h}_{t+1} = \sigma_h(\boldsymbol{W}_x\boldsymbol{x}_{t+1} + \boldsymbol{W}_h\boldsymbol{h}_t + \boldsymbol{b}_h) \tag{2}$$

Where $\boldsymbol{h}_t$ is the *hidden state* at timestep t, computed as a function of the input $\boldsymbol{x}_t$ and the hidden state from the previous timestep $\boldsymbol{h}_{t-1}$, using weight parameters $\boldsymbol{W}_x, \boldsymbol{W}_h, \boldsymbol{W}_y$, and bias parameters $\boldsymbol{b}_h, \boldsymbol{b}_y$, hidden-layer activation function σ_h and output activation function σ_y. $\boldsymbol{y}_t$ is the target at timestep t. For sequential language modelling, the target is the next element of the input sequence, [1] i.e. $\boldsymbol{y}_t = \boldsymbol{x}_{t+1}$, and the loss is typically *cross entropy*. The model is trained to make a prediction for every time-step $t = 1, ..., T$, and the loss $\mathcal{L}$ is the sum of the losses for the predictions at each time-step:

$$\mathcal{L}(x_1, ..., x_T) = \sum_{t=1}^{T} \mathcal{L}_t \tag{3}$$

$$\mathcal{L}_t = -\log P_\theta(\boldsymbol{x}_t|\boldsymbol{x}_1, ..., \boldsymbol{x}_{t-1}) \tag{4}$$

Note that it is necessary to specify an initial hidden state $\boldsymbol{h}_0$.

Note a previous version had further instructions here; now all instructions are in the code. In particular we no longer ask you to learn the initial hidden states (instead they are carried over from the previous mini-batch). Apologies for the changes.

Codes for Problem1 We built an internal class RNN_Unit manipulating tensor for one layer over times.

Implementation for RNN is as follows. Full implementation can be found in 'models.py'.

[1] In some settings, RNNs are also trained to output a special **termination symbol**, to indicate the end of the sequence, on the final time-step.

```
# Implement a stacked vanilla RNN with Tanh nonlinearities.
class RNN(nn.Module):
  class RNN_Unit(nn.Module):
    def __init__(self, input_size, hidden_size):
      super(RNN.RNN_Unit, self).__init__()
      self.input_size = input_size
      self.hidden_size = hidden_size
      self.i2h = nn.Linear(input_size + hidden_size, hidden_size)

    def init_weights(self, k = None):
        if k is None:
            k = 1. / math.sqrt(self.hidden_size)

        nn.init.uniform_(self.i2h.weight, -1 * k, k)
        nn.init.uniform_(self.i2h.bias, -1 * k, k)

    def forward(self, input):
      return torch.tanh(self.i2h(input))


  def __init__(self, emb_size, hidden_size, seq_len, batch_size, vocab_size,
               num_layers, dp_keep_prob):
    """
    emb_size:     The numvwe of units in the input embeddings
    hidden_size:  The number of hidden units per layer
    seq_len:      The length of the input sequences
    vocab_size:   The number of tokens in the vocabulary (10,000 for Penn TreeBank)
    num_layers:   The depth of the stack (i.e. the number of hidden layers at
                  each time-step)
    dp_keep_prob: The probability of *not* dropping out units in the
                  non-recurrent connections.
                  Do not apply dropout on recurrent connections.
    """
    super(RNN, self).__init__()

    self.emb_size = emb_size
    self.hidden_size = hidden_size
    self.seq_len = seq_len
    self.batch_size = batch_size
    self.vocab_size = vocab_size
    self.num_layers = num_layers
    self.dp_keep_prob = dp_keep_prob
    self.i2e = nn.Embedding(vocab_size, emb_size)
    self.h2o = nn.Linear(hidden_size, vocab_size)

    self.rnn_units = nn.ModuleList([])
    self.dropout = nn.Dropout(1 - dp_keep_prob)
```

```
    for i in range(num_layers):
      input_size = emb_size if i == 0 else hidden_size
      self.rnn_units.append(RNN.RNN_Unit(input_size, hidden_size))

    self.init_weights() # need this to initialize weights?
    return

  def init_weights(self):
    # Initialize the embedding and output weights uniformly in the range [-0.1, 0.1]
    # and the embedding and output biases to 0 (in place).
    # Initialize all other (i.e. recurrent and linear) weights AND biases uniformly
    # in the range [-k, k] where k is the square root of 1/hidden_size
    nn.init.uniform_(self.i2e.weight, -0.1, 0.1)
    nn.init.uniform_(self.h2o.weight, -0.1, 0.1)
    nn.init.zeros_(self.h2o.bias)
    k = 1. / math.sqrt(self.hidden_size)
    for i in range(self.num_layers):
      self.rnn_units[i].init_weights(k)

  def init_hidden(self):
    # initialize the hidden states to zero
    """
    This is used for the first mini-batch in an epoch, only.
    """
    hidden = torch.zeros(self.num_layers, self.batch_size, self.hidden_size)
    return hidden.requires_grad_()
    # a parameter tensor of shape (self.num_layers, self.batch_size, self.hidden_size)

  def forward(self, inputs, hidden):
    # Compute the forward pass, using nested python for loops.
    # The outer for loop should iterate over timesteps, and the
    # inner for loop should iterate over hidden layers of the stack.
    #
    # Within these for loops, use the parameter tensors and/or nn.modules you
    # created in __init__ to compute the recurrent updates according to the
    # equations provided in the .tex of the assignment.
    #
    # Note that those equations are for a single hidden-layer RNN, not a stacked
    # RNN. For a stacked RNN, the hidden states of the l-th layer are used as
    # inputs to to the {l+1}-st layer (taking the place of the input sequence).

    """
    Arguments:
        - inputs: A mini-batch of input sequences, composed of integers that
                    represent the index of the current token(s) in the vocabulary.
                        shape: (seq_len, batch_size)
        - hidden: The initial hidden states for every layer of the stacked RNN.
```

```
                    shape: (num_layers, batch_size, hidden_size)

    Returns:
        - Logits for the softmax over output tokens at every time-step.
              **Do NOT apply softmax to the outputs!**
              Pytorch's CrossEntropyLoss function (applied in ptb-lm.py) does
              this computation implicitly.
                    shape: (seq_len, batch_size, vocab_size)
        - The final hidden states for every layer of the stacked RNN.
              These will be used as the initial hidden states for all the
              mini-batches in an epoch, except for the first, where the return
              value of self.init_hidden will be used.
              See the repackage_hiddens function in ptb-lm.py for more details,
              if you are curious.
                    shape: (num_layers, batch_size, hidden_size)
    """
    seq_len = inputs.size(0) # (seq_len, batch_size)
    outputs_seqs = [] # list of output for each word in seq
    for i in range(seq_len):# iterate over seq
        embedded = self.dropout(self.i2e(inputs[i,:])) # (batch_size, emb_size)
        #print("embedded.shape:", embedded.shape)
        # size of each recurrent hidden layer output is (batch_size, hidden_size)
        output_cur_layer = None
        hs = [] # hidden states of each recurrent layer
        for j in range(self.num_layers): # more than 1 recurrent hidden layer
            input_cur_layer = embedded if j == 0 else output_cur_layer #
            combined = torch.cat((input_cur_layer, hidden[j,:,:]), 1)
            hidden_cur_layer = self.rnn_units[j](combined)
            output_cur_layer = self.dropout(hidden_cur_layer)
            hs.append(torch.unsqueeze(hidden_cur_layer, 0))

        output_cur_layer = self.h2o(output_cur_layer)
        hidden = torch.cat(hs, 0) # new hidden for next word in seq
        outputs_seqs.append(torch.unsqueeze(output_cur_layer, 0))

    logits = torch.cat(outputs_seqs, 0)
    return logits.view(self.seq_len, self.batch_size, self.vocab_size), hidden
```

Problem 2

Implementing an RNN with Gated Recurrent Units (GRU) (20pts) 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:

$$\boldsymbol{r}_t = \sigma_r(\boldsymbol{W}_r\boldsymbol{x}_t + \boldsymbol{U}_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r) \tag{5}$$
$$\boldsymbol{z}_t = \sigma_z(\boldsymbol{W}_z\boldsymbol{x}_t + \boldsymbol{U}_z\boldsymbol{h}_{t-1} + \boldsymbol{b}_z) \tag{6}$$
$$\tilde{\boldsymbol{h}}_t = \sigma_h(\boldsymbol{W}_h\boldsymbol{x}_t + \boldsymbol{U}_h(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_h) \tag{7}$$
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \tag{8}$$
$$P(\boldsymbol{y}_t|\boldsymbol{x}_1, ..., \boldsymbol{x}_t) = \sigma_y(\boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y) \tag{9}$$

$\boldsymbol{r}_t$ 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.

Note a previous version had further instructions here; now all instructions are in the code. Apologies for the changes.

Codes for Problem2 Like what we did for problem 1, we also built an GRU_Unit to flow the tensors one layer over times. We re-used almost all the other codes in RNN, except we replaced RNN_Unit with GRU_Unit when creating instances of a GRU_Unit. For simplicity, we here only pasted the implementation of GRU_Unit. For the whole implementation of GRU, refer to the file *models.py*.

```
class GRU(nn.Module): # Implement a stacked GRU RNN
  """
  Follow the same instructions as for RNN (above), but use the equations for
  GRU, not Vanilla RNN.
  """
  class GRU_Unit(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(GRU.GRU_Unit, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.i2r = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2z = nn.Linear(input_size + hidden_size, hidden_size)
        self.ir2hd = nn.Linear(input_size + hidden_size, hidden_size)

    def init_weights(self, k = None):
        if k is None:
            k = 1. /math.sqrt(self.hidden_size)
```

```
        nn.init.uniform_(self.i2r.weight, -k, k)
        nn.init.uniform_(self.i2r.bias, -k, k)
        nn.init.uniform_(self.i2z.weight, -k, k)
        nn.init.uniform_(self.i2z.bias, -k, k)
        nn.init.uniform_(self.ir2hd.weight, -k, k)
        nn.init.uniform_(self.ir2hd.bias, -k, k)

    def forward(self, input):
        """input (batch_size, input_size+hidden_size)"""
        x_size = self.input_size # real input size
        xt = input[:, :x_size] # (batch_size, x_size)
        ht = input[:, x_size:] # old ht (h_{t-1}) (batch_size, output_size)
        rt = torch.sigmoid(self.i2r(input)) # (batch_size, output_size)
        zt = torch.sigmoid(self.i2z(input)) # (batch_size, output_size)
        hd = torch.cat((xt, torch.mul(rt, ht)), 1) # (batch_size, input_size)
        hd = torch.tanh(self.ir2hd(hd)) # h_t^{tilde} (batch_size, input_size)
        ht_part1 = torch.mul(torch.sub(1, zt), ht)
        ht_part2 = torch.mul(zt, hd)
        ht = torch.add(ht_part1, ht_part2) # new ht (h_{t})
        return ht

# please refer to "models.py" for whole implementation of the whole GRU class.
```

Problem 3

Implementing the attention module of a transformer network (20pts) 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 [2] 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 indexed by i is calculated as follows:

$$\boldsymbol{A}_i = \text{softmax}\left(\frac{(\boldsymbol{Q}\boldsymbol{W}_{Q_i} + \boldsymbol{b}_{Q_i})(\boldsymbol{K}\boldsymbol{W}_{K_i} + \boldsymbol{b}_{K_i})^\top}{\sqrt{d_k}}\right) \tag{10}$$

$$\boldsymbol{H}_i = \boldsymbol{A}_i(\boldsymbol{V}\boldsymbol{W}_{V_i} + \boldsymbol{b}_{V_i}) \tag{11}$$

$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{concat}(\boldsymbol{H}_1, ..., \boldsymbol{H}_m)\boldsymbol{W}_O + \boldsymbol{b}_O \tag{12}$$

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). $\boldsymbol{A}_i$

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

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:

$$\tilde{\boldsymbol{x}} = \exp(\boldsymbol{x}) \odot \boldsymbol{s} \tag{13}$$

$$\text{softmax}(\boldsymbol{x}, \boldsymbol{s}) \doteq \frac{\tilde{\boldsymbol{x}}}{\sum_i \tilde{x}_i} \tag{14}$$

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

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} \odot \boldsymbol{s} - 10^9(1 - \boldsymbol{s}) \tag{15}$$

$$\text{softmax}(\boldsymbol{x}, \boldsymbol{s}) \doteq \frac{\exp(\tilde{\boldsymbol{x}})}{\sum_i \exp(\tilde{x}_i)} \tag{16}$$

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

Codes for Problem3 The following codes are our implementation of MultiHeadAttention for TRANSFORMER:

```
class MultiHeadedAttention(nn.Module):
    # Based on: https://github.com/harvardnlp/annotated-transformer

    def __init__(self, n_heads, n_units, dropout=0.1):
        """
        n_heads: the number of attention heads
        n_units: the number of output units
        dropout: probability of DROPPING units
        """
        super(MultiHeadedAttention, self).__init__()
        # This sets the size of the keys, values, and queries (self.d_k) to all
        # be equal to the number of output units divided by the number of heads.
        self.d_k = n_units // n_heads
        # This requires the number of n_heads to evenly divide n_units.
        assert n_units % n_heads == 0
        self.n_units = n_units # hidden_size

        # Initialize all weights and biases uniformly in the range [-k, k],
        # where k is the square root of 1/n_units.
        # Note: the only Pytorch modules you are allowed to use are nn.Linear
        # and nn.Dropout
        self.n_heads = n_heads
```

```
        self.linears = clones(nn.Linear(n_units, n_units), 4)
        k = 1. / math.sqrt(self.n_units)
        for linear in self.linears:
            nn.init.uniform_(linear.weight, -1*k, k)
            nn.init.uniform_(linear.bias, -1*k, k)

        self.dropout = nn.Dropout(dropout)


    def forward(self, query, key, value, mask=None):
        # implement the masked multi-head attention.
        # query, key, and value all have size: (batch_size, seq_len, self.n_units)
        # mask has size: (batch_size, seq_len, seq_len)
        # As described in the .tex, apply input masking to the softmax
        # generating the "attention values" (i.e. A_i in the .tex)
        # Also apply dropout to the attention values.
        batch_size = query.size(0)
        Qi, Ki, Vi = [l(x).view(
            batch_size, -1, self.n_heads, self.d_k).transpose(1, 2)
            for l, x in zip(self.linears, (query, key, value))]
        d_k = Qi.size(-1) # d_k = self.d_k
        scores = torch.matmul(Qi, Ki.transpose(-2, -1)) / math.sqrt(d_k)
        if mask is not None:
            mask = mask.unsqueeze(1)
            scores = scores.masked_fill(mask == 0, -1e9)

        Ai = torch.softmax(scores, dim = -1)
        Ai = self.dropout(Ai) # reasonable? dropout an probability?
        Hi = torch.matmul(Ai, Vi)
        Hi = Hi.transpose(1, 2).contiguous().view(batch_size, -1,
                                                  self.n_heads * self.d_k)
        A = self.linears[-1](Hi)
        return A # A shape: (batch_size, seq_len, self.n_units)
```

Problem 4

Training language models (20pts) 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.

(1) Model Comparison
In this problem you will run one experiment for each architecture (hyperparameter settings specified in the code) (3 experiments).

(2) Exploration of optimizers
You will run experiments with the following three optimizers (use the implementations provided in the code or given in Pytorch/Tensorflow; you don't need to implement these yourself) (6 experiments)

- "Vanilla" Stochastic Gradient Descent (SGD)
- SGD with a learning rate schedule; divide the learning rate by 1.15 after each epoch
- Adam

(3) Exploration of hyperparmeters
In this problem, you will explore combinations of hyperparameters to try to find settings which achieve better validation performance than those given to you in (1). Report at least 3 more experiments per architecture (you may want to run many more short experiments in order to find potentially good hyperparameters). (9+ experiments).

Figures and Tables:
Each table and figure should have an explanatory caption. For tables, this goes above, for figures it goes below. If it is necessary for space to use shorthand or symbols in the figure or table these should be explained in the caption. Tables should have appropriate column and/or row headers. Figures should have labelled axes and a legend. Include the following tables:

1. For **each experiment** in 1-3, plot **learning curves** (train and validation) of PPL over both **epochs** and **wall-clock-time**.
2. Make a table of results summarizing the train and validation performace 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. [3]
3. List all of the hyperparameters for each experiment in your report (e.g. specify the command you run in the terminal to launch the job, including the command line arguments).
4. Make 2 plots for each optimizer; one which has all of the validation curves for that optimizer over **epochs** and one over **wall-clock-time**.
5. Make 2 plots for each arcitecture; one which has all of the validation curves for that architecture over **epochs** and one over **wall-clock-time**.

Discussion
Answer the following questions in the report, referring to the plots / tables / code :

1. What did you expect to see in these experiments, and what actually happens? Why do you think that happens?

[3] 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.

2. Referring to the learning curves, qualitatively discuss the differences between the three optimizers in terms of training time, generalization performance, which architecture they're best for, relationship to other hyperparameters, etc.
3. Which hyperparameters+optimizer would you use if you were most concerned with wall-clock time? With generalization performance? In each case, what is the "cost" of the good performance (e.g. does better wall-clock time to a decent loss mean worse final loss? Does better generalization performance mean longer training time?)
4. Which architecture is most "reliable" (decent generalization performance for most hyperparameter+optimizer settings), and which is more unstable across settings?
5. Describe a question you are curious about and what experiment(s) (i.e. what architecture/optimizer/hyperparameters) you would run to investigate that question.

Answer

1. Table Results for 9 experiments in 4.1 and 4.2

Table 1: Train and validation performance by architecture

Model	Optim	IL	BS	SL	HS	NL	Prob-K	Tr-PPL	Va-PPL
RNN	ADAM	0.0001	20	35	1500	2	0.35	120	**156**
	SGD	0.0001	20	35	1500	2	0.35	3010	2210
	SGD-LR-SC	1	20	35	512	2	0.35	230	196
GRU	ADAM	0.0001	20	35	1500	2	0.35	60	111
	SGD	10	20	35	1500	2	0.35	50	113
	SGD-LR-SC	10	20	35	1500	2	0.35	66	**103**
TRANS	ADAM	0.001	128	35	512	2	0.9	3	**131**
	SGD	20	128	35	512	6	0.9	24	167
	SGD-LR-SC	20	128	35	512	6	0.9	60	143

TRANS:TRANSFORMER **IL**:Initial learning rate **BS**:batch size **SL**:sequence length **HS**:hidden size **NL**:number of layers **Prob-K**:dropout keep probability **Tr-PPL**: training perplexity **Va-PPL**: validation perplexity

2. Learning curves over epoch for experiments in 4.1 and 4.2

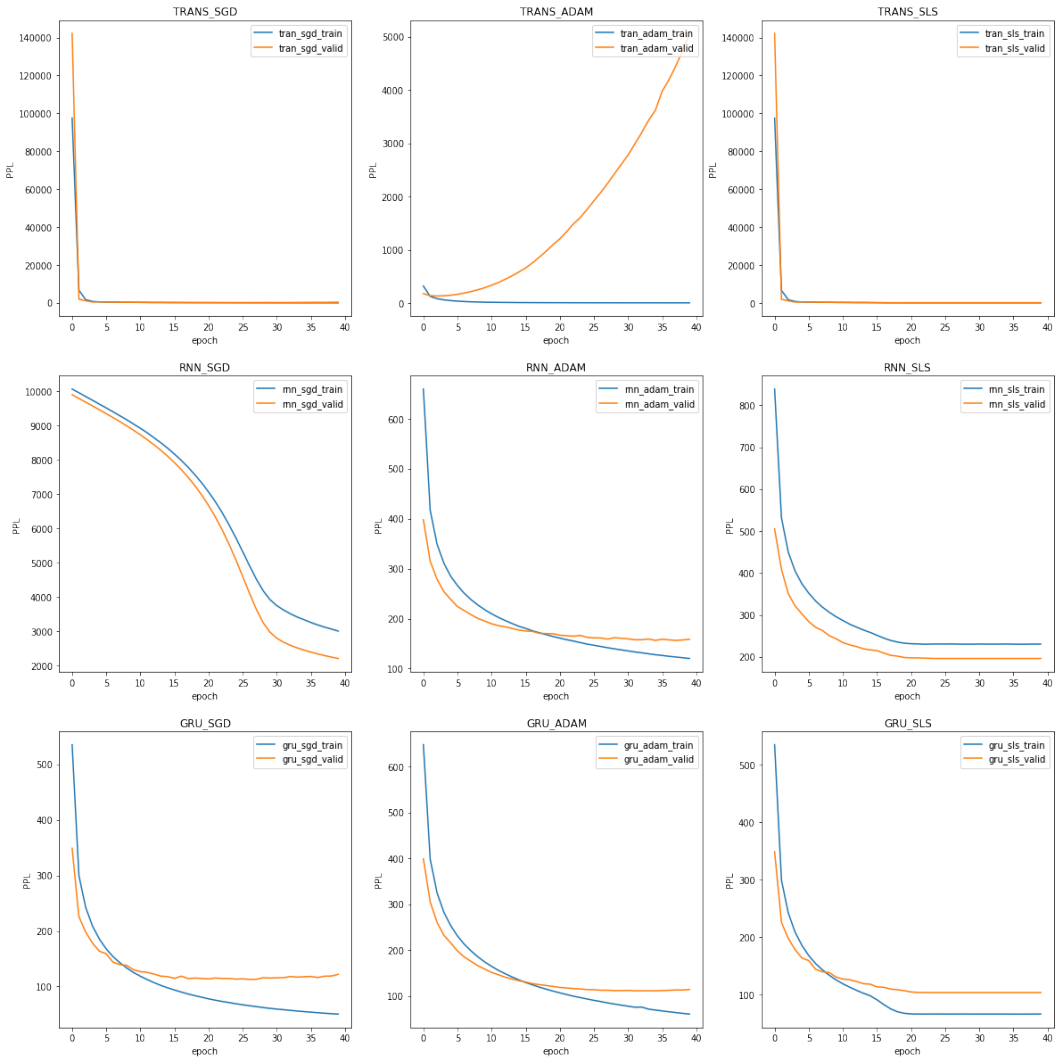


Figure 1: First 9 models: learning curves of PPL over epochs

3. Learning curves over wall-clock-time for experiments in 4.1 and 4.2

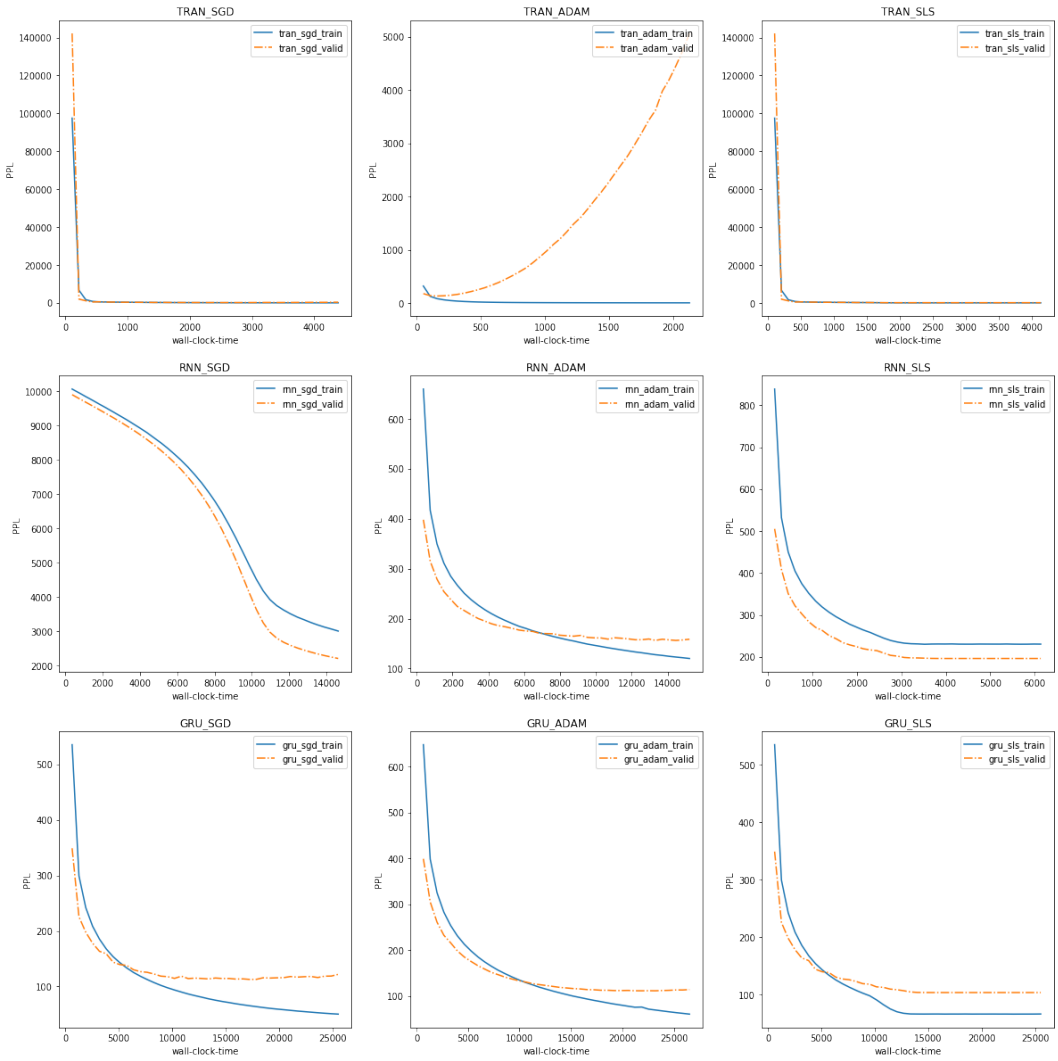


Figure 2: 9 first models: learning curves of PPL over wall-clock-time

4. Command lines for experiments in 4.1 and 4.2

```
python ptb-lm.py --model=RNN --optimizer=ADAM --initial_lr=0.0001
--batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2
--dp_keep_prob=0.35 --save_best

python ptb-lm.py --model=GRU --optimizer=SGD_LR_SCHEDULE --initial_lr=10
--batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2
--dp_keep_prob=0.35 --save_best

python ptb-lm.py --model=TRANSFORMER --optimizer=SGD_LR_SCHEDULE
--initial_lr=20 --batch_size=128 --seq_len=35 --hidden_size=512
--num_layers=6 --dp_keep_prob=0.9 --save_best

python ptb-lm.py --model=RNN --optimizer=SGD --initial_lr=0.0001
--batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2
```

```
--dp_keep_prob=0.35

python ptb-lm.py --model=GRU --optimizer=SGD --initial_lr=10
--batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2
--dp_keep_prob=0.35

python ptb-lm.py --model=TRANSFORMER --optimizer=SGD --initial_lr=20
--batch_size=128 --seq_len=35 --hidden_size=512 --num_layers=6
--dp_keep_prob=.9

python ptb-lm.py --model=RNN --optimizer=SGD_LR_SCHEDULE --initial_lr=1
--batch_size=20 --seq_len=35 --hidden_size=512 --num_layers=2
--dp_keep_prob=0.35

python ptb-lm.py --model=GRU --optimizer=ADAM --initial_lr=0.0001
--batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2
--dp_keep_prob=0.35

python ptb-lm.py --model=TRANSFORMER --optimizer=ADAM --initial_lr=0.001
--batch_size=128 --seq_len=35 --hidden_size=512 --num_layers=2
--dp_keep_prob=.9
```

4. Description the learning curves for 4.1 and 4.2

1. Architecture and Val-PLL: in these 9 experiments, best val-pll comes from GRU with SGC-LR-SC optimizer with initial learning rate 10. In general, GRU also performs better than the other two architectures.

2. Optimizer and Val-PLL: ADAM can dynamically adjust different learning rates for each of the parameters; SGD-LR-SC decreases learning rate following a pre-defined schedule; SGD uses the same learning-rate during the whole training process. As all models are observed only 40 epochs, it's hard to determine which optimizer is best. from the result using SGD and SGD-LR-SC within each architecture, we may conclude that: an RNN model using SGD with initial lr 0.001 produced very bad ppl on both train and valid dataset, which probably means the learning rate is too small to get an good result within 40 epochs; whereas SGD with an initial learning 10 obtained almost the same level ppl as those from ADAM and SGD-LR-SC for RNN and GRU.

3. TRANSFORMER: with ADAM optimizer and only 2 layers, TRANSFORMER obtained the train ppl with 3 and val-ppl with 131, meaning it is extremely overfitted. the other two optimizers also brought overfitting results even if the initial learning rate is set to a relatively large number(20). From this respective, we tend to conclude that TRANSFORM has so large capacity that it can very easily fit the training dataset, but didn't learn real knowledge(features) from the dataset. Still, it is too early to say that TRANSFORMER architecture is not good. We need to fine tune the hyper-parameters for this architecture before we make further conclusion on it.

4. Training speed: hyper-parameters like Batch-size, seq-len, hid-size, num-layers, and the computation graph in each architecture affect the training speed. Decreasing the number of these hyper-parameters (except batch-size) will speed up the training process. The Intuition is that TRANSFORMER is the fastest architecture, because it doesn't need a loop for manipulating word by word in sequences wheras GRU and RNN do. Increasing the batch-size allows the model to manipulate more data in one step which also speed up the process. As in all 9 experiments, TRANSFORMER uses much large batch-size and half hid-size than the other two architectures, we just can't say TRANSFORMER is faster than the other two. it needs further experiment to be clarified.

5. In general, from those figures, we can see that:

   (a) For GRU models, each of they performed well and equally. For RNN models, the curves of two groups(ADAM and SGD-LR-SC) have the same shape, but according to table 1, RNN with ADAM is obviously better. And for TRANSFORMER, overfitting for ADAM is very heavy, but it reach the lower value than SGD. So we could reduce its capacity to eliminate the overfitting, and its time for each epoch is short, perhaps it will easier for research the better hyperparameters. But there are also some situations worse than we expect in the training processes. For example, the curve of RNN with SGD is very higher than using other optimizer in RNN and from about 30th epoch, its reduce trend is more gentle, but the value PPL is still very large, that is said, the capacity of RNN with optimizer SGD is too low to continue research. And the other hand, for TRANSFORMER with SGD_LR_SCHEDULE begin with very large PPL, although it goes down quickly, meanwhile, the curves of TRANSFORMER with SGD and with SGD_LR_SCHEDULE are overlapped, we could not observe clearly. And for TRANSFORMER with ADAM, seemingly goes up from beginning.

   (b) For SGD, the result of RNN is not good enough to discuss, so we draw the curves of GRU and TRANSFORMER. Here, GRU performer better according to lower value of val PPL and balance between underfitting and overfitting. For TRANSFORMER, it is overfitting from about 30th epoch without reach the low PPL as GRU. For SGD_LR_SCHEDULE, the three models plays well, but GRU has the lowest value on val PPL. And for ADAM, because the heave overfitting on TRANSFORMER, so we draw two figures to compare. We can find that the curves of GRU and RNN have same shape, and GRU seems to reach to a lower PPL. Although TRANSFORMER is overfitting from 2th or 3th epoch, it get the lowest PPL in ADAM.

6. Suggestion for further exploitation:

   (a) decrease the capacity of TRANSFORMER to observe its performance.

   (b) decrease the capacity of models over-fitted to try to get a better results than the best in the above 9 experiments.

   (c) Some models(architecture with certain hyper-parameters) need more epochs to get their final best val ppl.

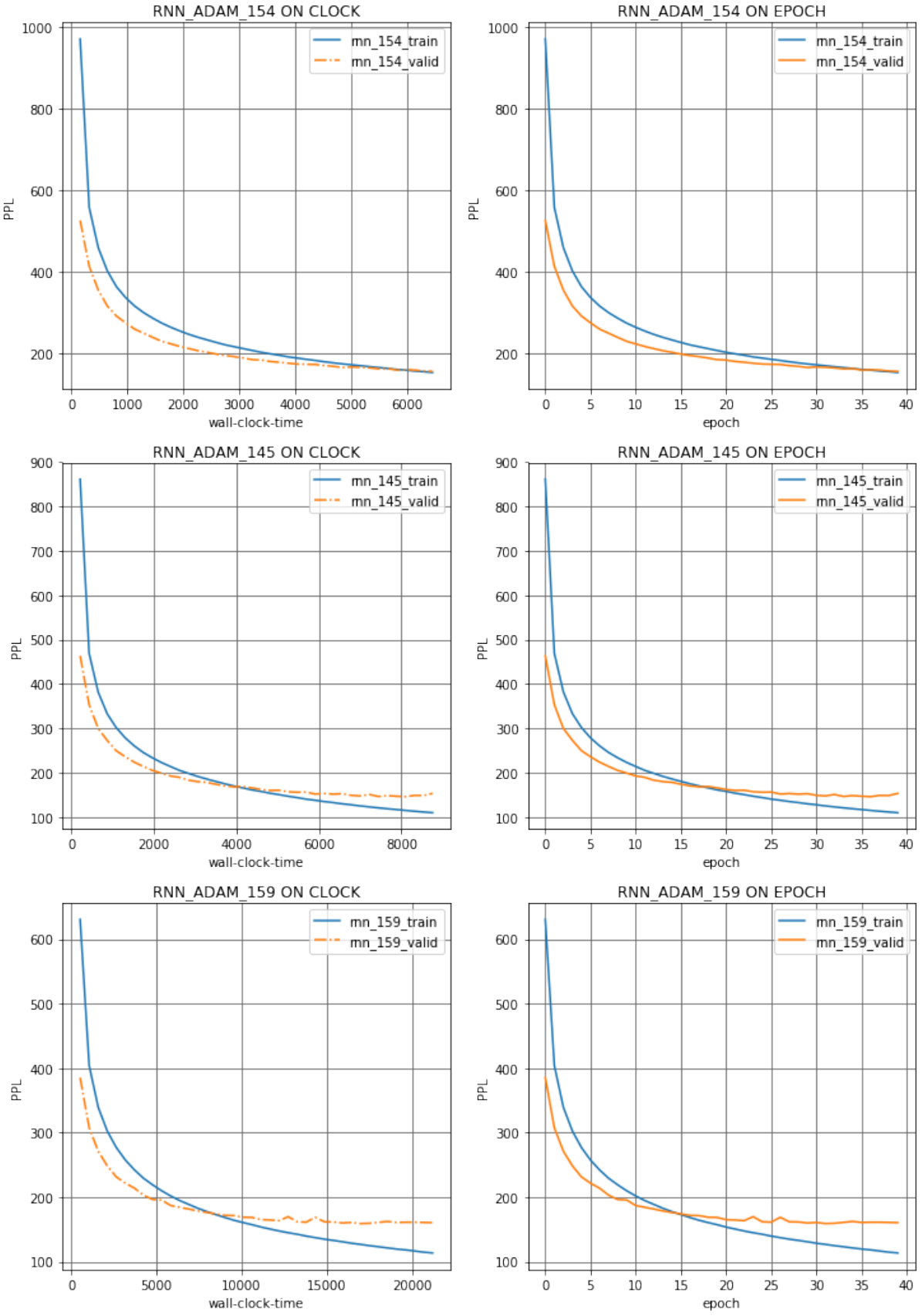
5. Table Results for new experiments in 4.3

Table 2: Train and validation performance for all new experiments

Model	Optim	IL	BS	SL	HS	NL	Prob-K	Tr-PPL	Va-PPL
RNN	ADAM	0.0001	20	35	2000	2	0.35	113	159
RNN	ADAM	0.0001	128	35	2000	2	0.35	151	154
RNN	**ADAM**	**0.0001**	**128**	**35**	**2000**	**3**	**0.5**	**109**	**145**
GRU	ADAM	0.0001	20	35	1500	3	0.35	59	117
GRU	ADAM	0.0001	20	35	2000	2	0.35	44	112
GRU	**SGD_LR_SC**	**15**	**20**	**35**	**1500**	**2**	**0.35**	**65**	**103**
TRANS	**ADAM**	**0.00008**	**256**	**35**	**512**	**6**	**0.75**	**60**	**119**
TRANS	ADAM	0.0001	128	35	512	6	0.75	33	120
TRANS	ADAM	0.0001	128	35	1024	6	0.75	9	121

TRANS:TRANSFORMER **IL**:Initial learning rate **BS**:batch size **SL**:sequence length **HS**:hidden size **NL**:number of layers **Prob-K**:dropout keep probability **Tr-PPL**: training perplexity **Va-PPL**: validation perplexity

6. Figure Results for all experiments in 4.3



[H]

Figure 3: Three new model RNN with ADAM

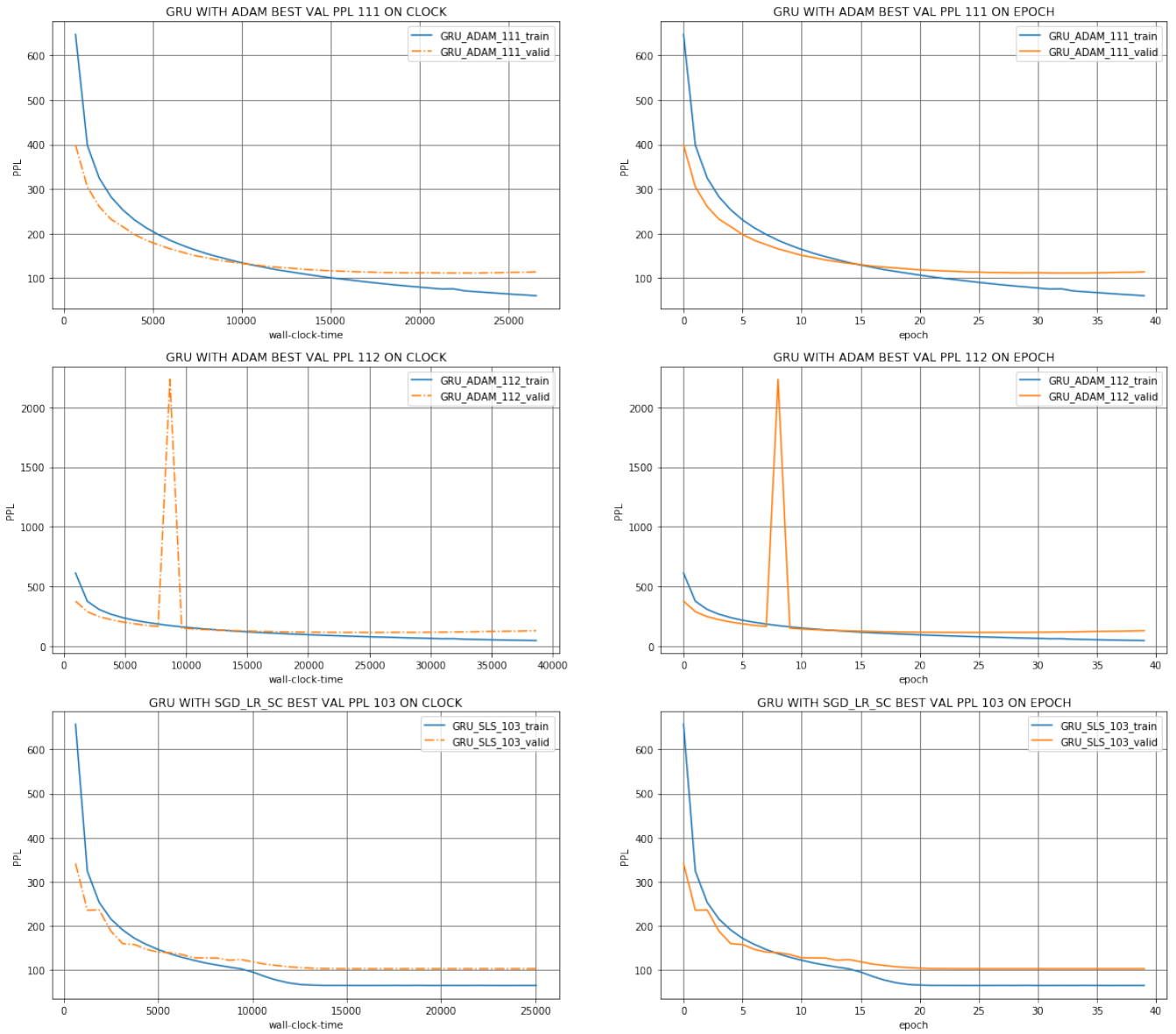


Figure 4: Three new models GRU with ADAM or SGD_LR_SCHEDULE

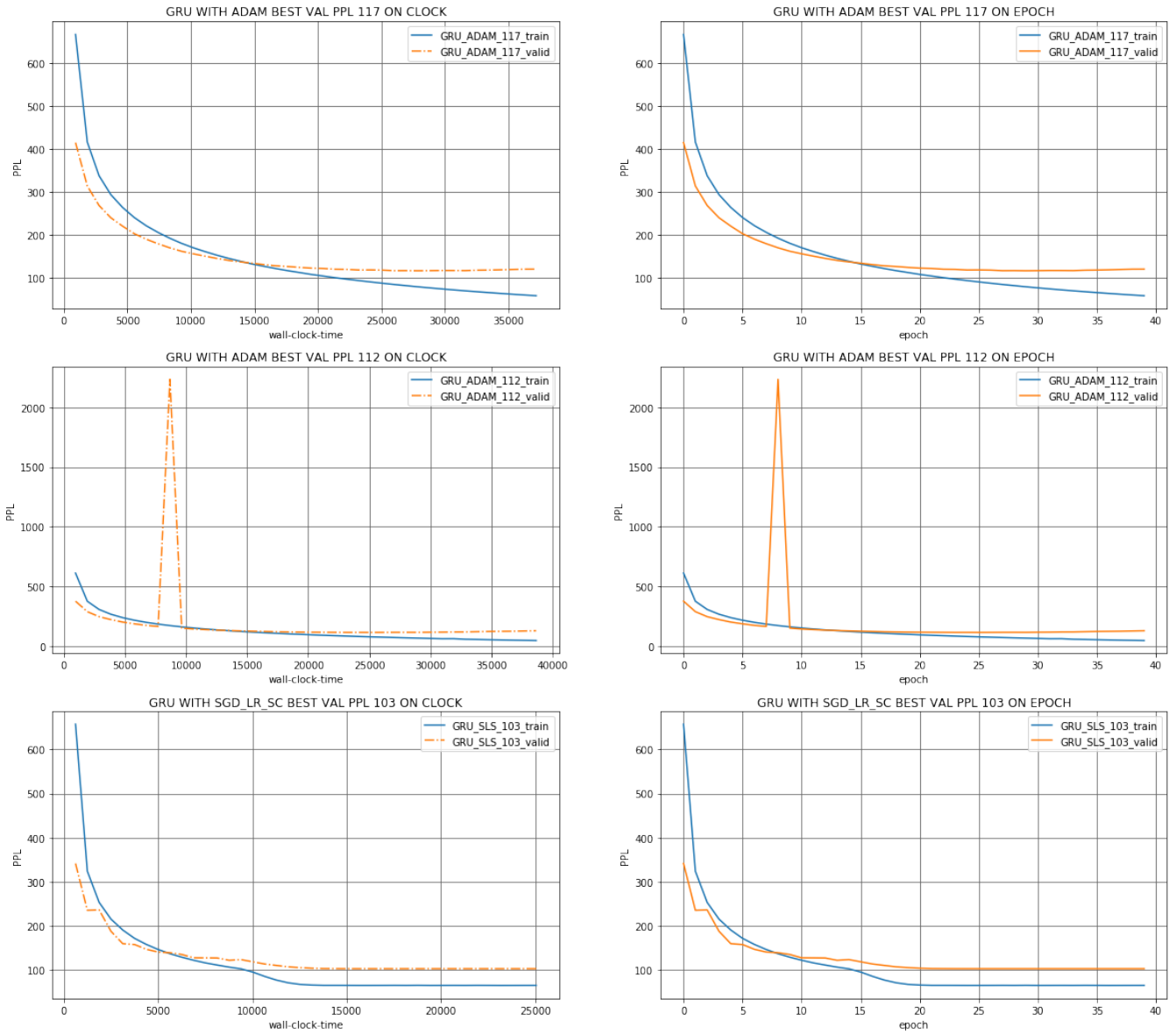


Figure 5: Three new models TRANS with ADAM

7. Command lines for experiments in 4.3

```
python ptb-lm.py --model=RNN --optimizer=ADAM --initial_lr=0.0001
--batch_size=20 --seq_len=35 --hidden_size=2000 --num_layers=2
--dp_keep_prob=0.35 --save_best

python ptb-lm.py --model=RNN --optimizer=ADAM --initial_lr=0.0001
--batch_size=128 --seq_len=35 --hidden_size=2000 --num_layers=2
--dp_keep_prob=0.35 --save_best

python ptb-lm.py --model=RNN --optimizer=ADAM --initial_lr=0.00001
--batch_size=20 --seq_len=35 --hidden_size=2000 --num_layers=2
--dp_keep_prob=0.35 --save_best
```

```
python ptb-lm.py --model=GRU --optimizer=ADAM --initial_lr=0.0001
--batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=3
--dp_keep_prob=0.35 --save_best

python ptb-lm.py --model=GRU --optimizer=ADAM --initial_lr=0.0001
--batch_size=20 --seq_len=35 --hidden_size=2000 --num_layers=2
--dp_keep_prob=0.35 --save_best

python ptb-lm.py --model=GRU --optimizer=SGD_LR_SCHEDULE --initial_lr=15
--batch_size=20 --seq_len=35 --hidden_size=1500 --num_layers=2
--dp_keep_prob=0.35 --save_best

python ptb-lm.py --model=TRANSFORMER --optimizer=ADAM
--initial_lr=0.00008 --batch_size=256 --seq_len=35 --hidden_size=512
--num_layers=6 --dp_keep_prob=0.75 --save_best

python ptb-lm.py --model=TRANSFORMER --optimizer=ADAM
--initial_lr=0.0001 --batch_size=128 --seq_len=35 --hidden_size=512
--num_layers=6 --dp_keep_prob=0.75 --save_best

python ptb-lm.py --model=TRANSFORMER --optimizer=ADAM
--initial_lr=0.0001 --batch_size=128 --seq_len=35 --hidden_size=1024
--num_layers=6 --dp_keep_prob=0.75 --save_best
```

8. Description for the learning curves in 4.3

- In the exploitation, we didn't change the seq-len.
- For the curve already over-fitted obviously, we decrease the capacity, from hid-size, num-layer and dp-k-prob, if it could not still reach the object, we consider adjusting learning-rate.
- For the curve underfitting or not overfitting obviously, we increase their capacity to attempt to reach better results. Hyper-parameters like hid-size, num-layer and dp-k-prob can also be used for exploitation.
- After a lot of experiments, we found several groups of hyper-pararmeters who brought lower val PPL than previous 9 models:
  - For RNN, we have 3 more interesting groups of hyperparameters, two of which have better results than the model in 4.1. New RNN models use much shorter time in each epoch than the previous model and they reached lower val PPL although one of them may be alittle overfitting.
  - For GRU, we have 3 more interesting groups of hyperparameters and one of them arrived lower best val PPL than the best model in 4.1. But from the figure of GRU with ADAM, we observed a noise give us a sudden peak in valid PPL around 7-9 epoch or 7500-10000 wall-clock-time. It's so meaningful for us to research the reason.

– For TRANSFORMER, we committed to reduce the capacity of the model. But after choose different hidden_size, layer_number and dp-k-prob, we found that although they could arrive better result than the best one in 4.1, they are all still overfitting very ealy, at 25th or 20th. Therefore, we chose to reduce the learning-rate, and this change gave us the best result. Although it costed double time for training the new model, it worked well and reached a lower val PPL than the previous one.

9. Final Discussion

(1). The results expected to see in these experiments, their actual result and the reasons.

- The result expected of RNN with ADAM should be good, and its real result fits what we expect.
- For RNN with SGD, we expect to see a slow convergence, and the result is as poor as our expectations.
- We think GRU with ADAM will have a satisfied result, but its final result is not so good as the result of GRU with SGD_LR_SC. The fact that it doesn't converge to the best value in 40th epoch is probably due to a not good enough initial learning rate.
- We think that GRU with SGD will have a bad result because of large learning rate, but the real result is better than we think. Maybe 40 epochs is not long enough to expose the drawbacks of large learning rate.
- The real result of GRU with SGD_LR_SC is good as we expected, and it's the best result in all experiments. The learning rate schedule must be well designed.
- We are not sure about what result that TRANS with ADAM may bring to us. The result shows that it is over-fitting: only 3 on train PPL, but 131 on best val PPL.

(2). The differences between the three optimizers.

- Well chose initial learning rate for a ADAM optimizer usually learns faster than other optimizer.
- Small fixed learning rate for optimizer SGD could cause low efficiency.
- SGD_LR_SC is a good optimizer, but its initial learning rate and learning rate decay schedule need to be well designed.
- GRU usually performs better than RNN because of its long-term memory. Transformer as too easy to be over-fitted.

(3). Some opinions about wall-clock-time

– If we were most concerned with wall-clock-time, we'd like to choose TRANSFORMER or RNN, chooose ADAM or SLS optimizer. Easier model will cut down the time for executing. Besides, larger batch_size, could also shorten the wall-clock-time.

(4). Reliable architecture

– The architecture GRU is most reliable (decent generalization performance for most hyperparameter+optimizer settings)

- TRANSFORMER is more unstable across settings. Capacity of a TRANSFORMER with 6 layers is huge, so that it is very easy to be overfitted.

(5). The questions curious

- It is difficult to judge whether the TRANSFORMER model can really learn the inner connection of the data by only 9 experiments. it's very interesting to investigate the difference between TRANSFORMER and GRU(or RNN) by balance them with same level of learnable parameters. It is also very interesting to find a well generalized hyper-parameters for TRANSFORMER architecture, and compare with the best of GRU.

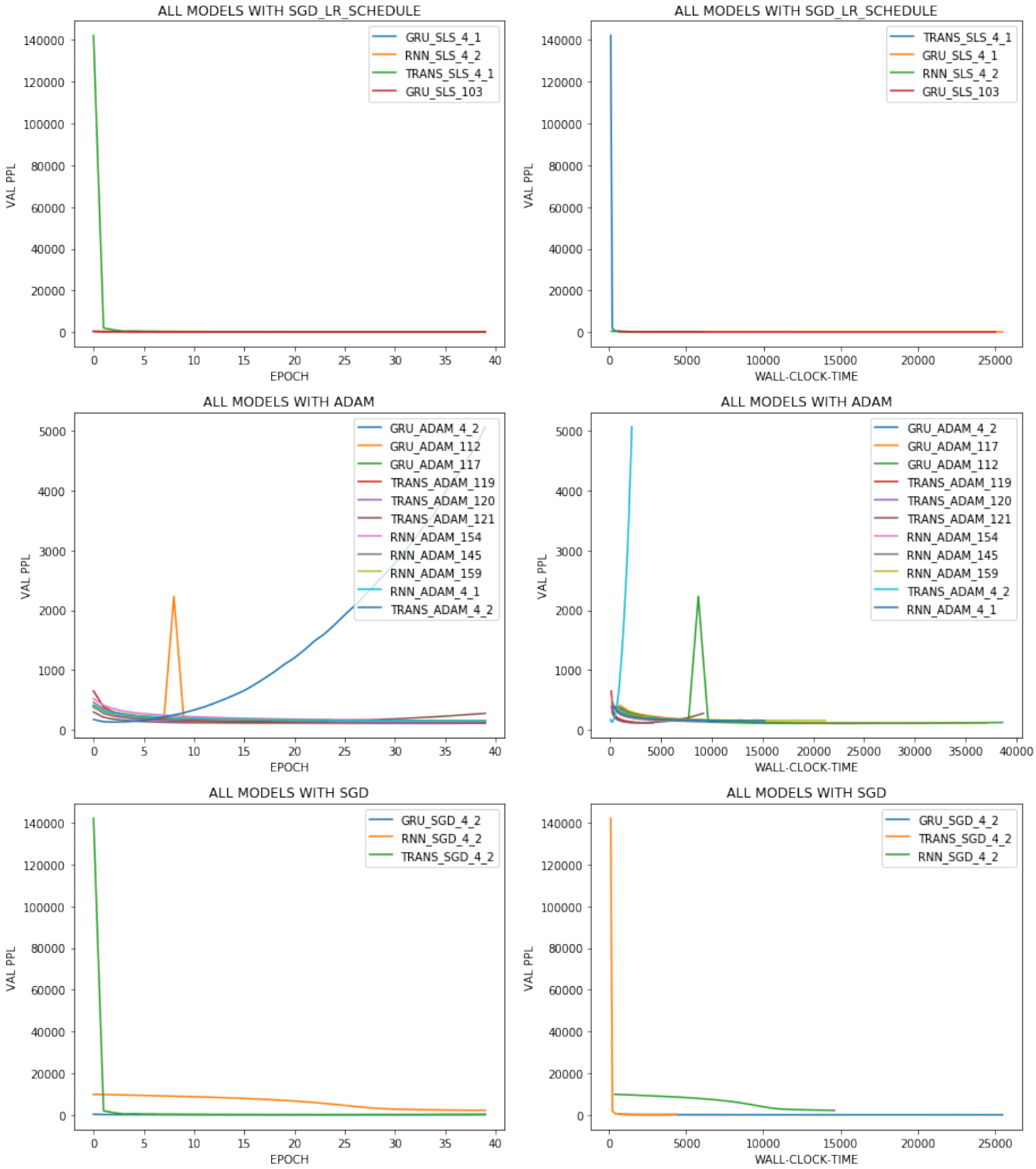


Figure 6: Optimizers validation curves over epochs vs wall-clock-time

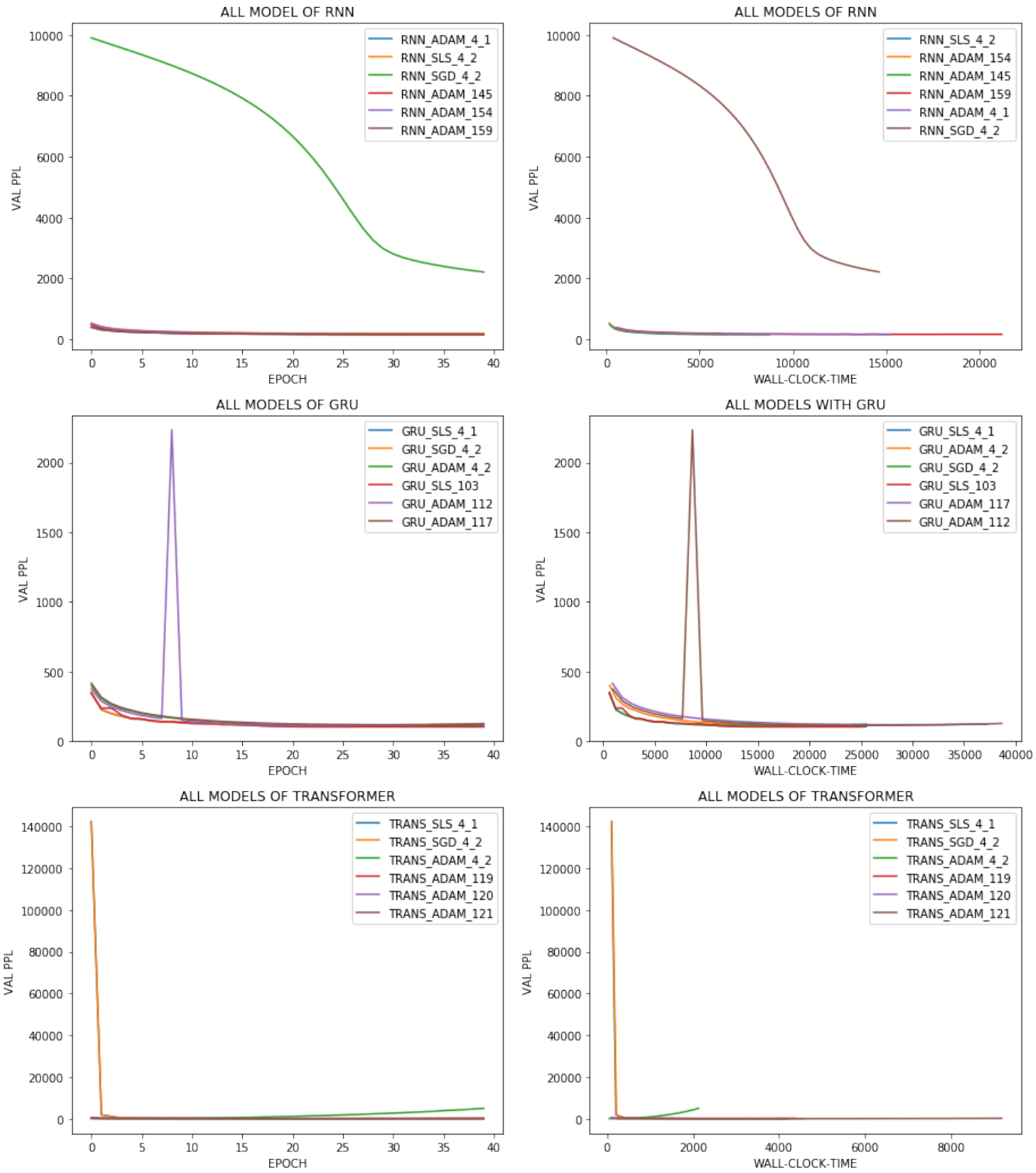


Figure 7: Architectures validation curves over epochs vs wall-clock-time

Problem 5

Detailed evaluation of trained models (20pts) For this problem, we will investigate properties of the trained models from Problem 4. Perform the following evaluations for the models you trained in Problem 4.1.

Note: Full implementation of this problem is in the file 'problem5.ipynb'.

*1 Compute the average loss at each time-step (i.e. $\mathcal{L}_t$ for each t) within validation sequences. Plot $\mathcal{L}_t$ as a function of t, describe the result qualitatively, and provide an explanation for what you observe. Compare the plots for the different architectures.

Answer After examining the method ptb-iterator and run_epoch in the given file ptb-lm.py, we keep seq_len with certain value (e.g. 35), modify the loss_fn by giving an attribute "reduction = 'none'" so that the loss are calculated per element in a tensor. Run the model for several steps each of which we feed data X ,y with the shape (batch_size, seq_len). The final loss can be obtained by averaging the sum of steps' losses with each also averaged by a batch_size. Note: at the beginning of each step, hidden is re-initialized.

Key Codes

```
# here is another implementation without setting seq_len to 1.
# a little more complicated with memory comsuming.
# both work well with proper batch_size and seq_len
def average_losses2(model, model_name, data, max_time_step = None):

    loss_fn = nn.CrossEntropyLoss(reduction = "none")
    # "none" means don't average or sum the losses, keep the shape

    losses = np.zeros(model.seq_len)
    model.eval()
    model.zero_grad()

    for time_step, (x, y) in enumerate(ptb_iterator(data, model.batch_size,
                                                    model.seq_len)):
        # both x, y are of shape (batch_size, seq_len)
        if model_name == 'TRANSFORMER':
            batch = Batch(torch.from_numpy(x).long().to(device))
            outputs = model.forward(batch.data, batch.mask).transpose(1,0)
        else:
            inputs = torch.from_numpy(x.astype(np.int64))
            inputs = inputs.transpose(0, 1).contiguous().to(device)#.cuda()
            hidden = model.init_hidden().to(device)#repackage_hidden(hidden)
            outputs, _ = model(inputs, hidden)

        targets = torch.from_numpy(y.astype(np.int64)).transpose(0, 1)
        targets = targets.contiguous().to(device)#.cuda()
        tt = torch.squeeze(targets.view(-1, model.batch_size * model.seq_len))
```

```
            outputs = outputs.contiguous().view(-1, model.vocab_size)
            # loss is a one-dimensional tensor here (seq_len * batch_size)
            loss = loss_fn(outputs, tt)
            # converted to (seq_len, batch_size)
            loss = loss.view(model.seq_len, model.batch_size)
            # transpose loss shape to (batch_size, seq_len)
            loss = loss.cpu().data.numpy().transpose(1, 0)
            #print(loss.shape)
            losses += np.mean(loss, axis = 0)

            # you may not want to calculate over the whole dataset
            if (not max_time_step is None) and (time_step >= max_time_step - 1):
                return losses

    losses = losses / time_step
    return losses
```

Result

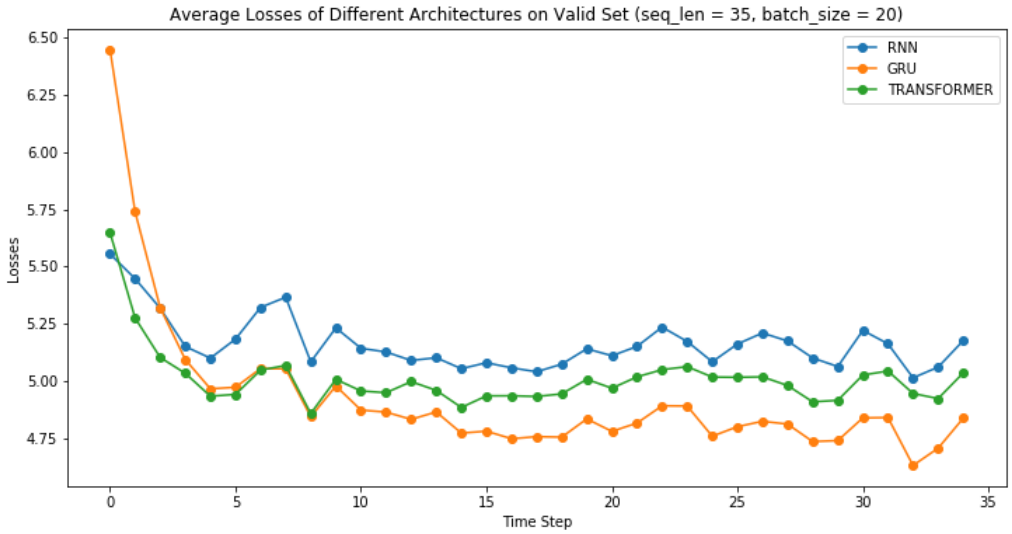


Figure 8: Average Losses per Time-Step

(a) For each architecture, losses start with a higher value then decrease quickly for next several time-steps, followed by a flat decreasing(not very obvious, sometimes with oscillations) trend as time flows.

(b) Comparing the three architectures: GRU performs best, then TRANSFORMER, and RNN performs worst.

Explanation

(a) **Higher loss at initial time step**: without any hidden information, models(RNN and GRU) tend to predict as trained, predicting words which are the starting words during

training in steps with higher probabilities, If the starting words are much different in validation dataset(this is very normal, change batch_size and sql_len will produce sequences with very different starting words). GRU having the highest Initial loss may just mean that the hidden status is more valuable for it, without useful hidden information, GRU is just a little better than a random prediction(with average loss about 9.21).

(b) **Quick decreasing for first several time steps**: This is the effect of hidden units or context(for TRANSFORMER). with the help of context information, all models start to predict better and better. This is not the case when use seq_len = 1 for TRANSFORMER, where it still has no context to use. It performs better than RNN seq_len = 30 is used.

(c) **A long flat zone**: After a steep decreasing at the first several time steps, all architectures stuck into a flat zone, where the loss keeps a certain level. Loss value of 5 corresponds to a correct predicting probability of $e^{-5} = 0.0067$ which is still 67 times higher than making a random prediction. Maybe it is because of the pattern diversities in the training dataset.

(d) **About the oscillations**: A sudden increasing loss often indicates models are losing correctly predictions due to the diversity of sequences. then a sudden decreasing of loss may indicate that a new pattern in the sequences is caught by the model, such as the [unk] and [eos] tokens often come after or before certain words, or a phrase with common patterns starts. We are not sure about this inference.

We also developed another method to compute the average loss per time-step, which produces quite similar results except for a higher average loss of TRANSFORMER where contexts is needed but not provided by the implementation. See our code file for more details.

*2 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 each of the 3 architectures. Rescale the values of each curve to [0,1] so that you can compare all three 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 As we are only going to observe the gradients of GRU and RNN, we can set the seq_len = 1 and run the model for several time steps. By using the method torch.autograd.grad, we are able to get the loss at the final step with respect to the hidden state at each time-step t.

Key Codes

```
def average_gradients(model, model_name, data, max_time_step = 35):
    loss_fn = nn.CrossEntropyLoss()
    hiddens = [] # hiddens along time step from 0 to model.seq_len-1
    norm_gradients = []
    if model_name == 'TRANSFORMER':
        return None

    hidden = model.init_hidden()
```

```
        hidden = hidden.to(device)
        hiddens.append(hidden)
        model.train()
        model.zero_grad()

        for time_step, (x, y) in enumerate(ptb_iterator(data, model.batch_size,
                                                        model.seq_len)):
            # both x, y are of shape (batch_size, seq_len)
            inputs = torch.from_numpy(x.astype(np.int64)).transpose(0, 1)
            inputs = inputs.contiguous().to(device)#.cuda()
            outputs, hidden = model(inputs, hiddens[-1])
            hiddens.append(hidden)

            if time_step >= max_time_step - 1: # only seq_len time steps
                targets = torch.from_numpy(y.astype(np.int64)).transpose(0, 1)
                targets = targets.contiguous().to(device)#.cuda()
                tt = torch.squeeze(targets.view(-1, model.batch_size * model.seq_len))
                if tt.dim() == 0:
                    tt = torch.unsqueeze(tt, 0)

                loss = loss_fn(outputs.contiguous().view(-1, model.vocab_size), tt)
                for i in range(len(hiddens)-2): # the last hidden doesn't require grade
                    gradient = torch.autograd.grad(loss, hiddens[i], retain_graph = True)
                    # gradient is a tuple (Tensor, )
                    norm_gradients.append(torch.norm(gradient[0], p = 2).data.item())

                return norm_gradients
```

Result and Explanation Figure(9) shows the trends of average of all hidden gradients

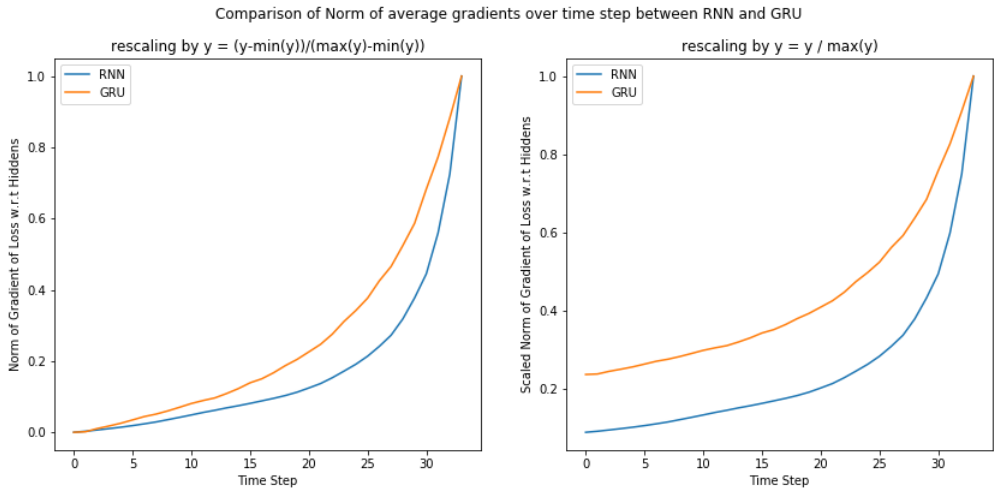


Figure 9: Norms of average hidden gradients over time-step

along the time-step. Obvisouly, compared with the gradient of the final time-step, gradients of previous hiddens are getting smaller and smaller, which means gradient is vanishing or, we

can say, not very easy to pass back through time. Gradient of RNN decreased much faster than GRU, meaning that RNN has rellatively short term memory, while GRU has longer, benefited from its gated recurrent unitits and hidden states.

*3 Generate samples from both the Simple RNN and GRU, by recursively sampling $\hat{\boldsymbol{x}}_{t+1} \sim P(\boldsymbol{x}_{t+1}|\hat{\boldsymbol{x}}_1, ..., \hat{\boldsymbol{x}}_t)$. [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. Choose 3 "best", 3 "worst", and 3 that are "interesting", and make reference to these to describe qualitatively the different models' strengths, failure modes, quirks, and any potentially interesting things to investigate. Put all 40 samples in an appendix to your report.

Answer Generating samples from a model includes two parts: sample an index from the final (softmax) output of a model, and convert that index to a real word in the dictionanry and display it. The first part is done within each model by the method **generate**, and we developed another method **generate_seq** to perform the second part. The following is the implementation of the two methods.

Key Codes

```
# this method is considered a part of class RNN and GRU
def generate(self, input, hidden, generated_seq_len):
    # TODO ========================
    # Compute the forward pass, as in the self.forward method (above).
    # You'll probably want to copy substantial portions of that code here.
    #
    # We "seed" the generation by providing the first inputs.
    # Subsequent inputs are generated by sampling from the output distribution,
    # as described in the tex (Problem 5.3)
    # Unlike for self.forward, you WILL need to apply the softmax activation
    # function here in order to compute the parameters of the categorical
    # distributions to be sampled from at each time-step.

    """
    Arguments:
        - input: A mini-batch of input tokens (NOT sequences!)
                        shape: (batch_size)
        - hidden: The initial hidden states for every layer of the stacked RNN.
                        shape: (num_layers, batch_size, hidden_size)
        - generated_seq_len: The length of the sequence to generate.
                       Note that this can be different than the length used
                       for training (self.seq_len)
    Returns:
        - Sampled sequences of tokens
                    shape: (generated_seq_len, batch_size)
    """
    samples = []
```

[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.

```
        predict = input # at time step 0, predict is input
        samples.append(torch.unsqueeze(predict, 0))
        for i in range(generated_seq_len): # each seq index
            # output is the predicted word index, with batch_size elements
            embedded = self.dropout(self.i2e(predict)) # (batch_size, emb_size)
            output_cur_layer = None
            hs = []
            for j in range(self.num_layers): # more than 1 recurrent hidden layer
                input_cur_layer = embedded if j == 0 else output_cur_layer #
                combined = torch.cat((input_cur_layer, hidden[j,:,:]), 1)
                hidden_cur_layer = self.rnn_units[j](combined) #(batch_size, hidden_size)
                output_cur_layer = self.dropout(hidden_cur_layer)
                hs.append(torch.unsqueeze(hidden_cur_layer, 0))

            output_cur_layer = self.h2o(output_cur_layer)
            hidden = torch.cat(hs, 0) # new hidden for next word in seq
            p_output = torch.softmax(output_cur_layer, dim = -1) # (batch_size, vocab_size
            predict = Categorical(p_output).sample().long() # batch_size
            # another option is just give the predict with largest probability
            # output_cur_layer = torch.argmax(output_cur_layer, dim = -1) # word_index
            samples.append(torch.unsqueeze(predict, 0))

        samples = torch.cat(samples, 0)
        return samples

    # for problem 5.3
    def generate_seq(model, vocab_size = 10000):
        # generate sequence from a trained model
        model.eval()
        batch_size = model.batch_size
        seq_len = model.seq_len

        gen_hidden = torch.zeros(model.num_layers, batch_size, model.hidden_size)
        gen_hidden = gen_hidden.to(device)
        gen_hidden = repackage_hidden(gen_hidden)
        gen_input = torch.randint(0, vocab_size, (batch_size,)).contiguous()
        gen_input = gen_input.to(device)#.cuda()
        samples = model.generate(gen_input, gen_hidden, seq_len)
        # size of samples (seq_len, batch_size)
        samples = samples.transpose(0, 1).cpu().numpy()
        max_len = 80
        cur_len = 0
        for i in range(batch_size): # for each batch
            print("Seq {}: ".format(i), end = " ")
            cur_len = 7
            for j in range(seq_len):
                word = id_2_word[samples[i,j]]
```

```
            if cur_len + len(word) >= max_len:
                cur_len = 0
                print("")

            print(word, end = " ")
            cur_len += (len(word) + 1)
        print("\n")
```

Result The 40 sequences is attached as appendix of the report. Here, we chose 3 best, 3 worst, and 3 interesting sequences and commented on them.

(a) **3 best**: In these best setences, it seems that the model learned some phrases, and some meaningful pattern, such as: "due to" are generated together, "mainly" is put between "was" and "due to", "credit of the cities", "golden gate", and "we want to stay". most of these beautiful sentences are generated by GRU. which may reflect that GRU is better than RNN.

i. (GRU) fabrication was mainly due to the natural gas joint-venture system [eos] he also has Nmore than $N million his [unk] in the u.s. s new [unk] credit of the cities at minnesota [eos] thecongressman also urged asking of programs of reserve dr. [unk] s [unk] of a disproportionate datefrom the movie of socialism [eos] however he will take part of a surprising bid on increasing fo

ii. (GRU) golden gate and has obtained under [unk] to [unk] and [unk] things from his [unk] [eos] mr.holmes said he is talking in donations [eos] we do nt have any immediate impact of our readerwhere it had [eos] we do nt know how i go what is going [eos] but i feel our [unk] said [unk] nowaims [eos] he also urged the hiroshima to provide a hidden [unk]

iii. (RNN) jeff [unk] who plans to make an dispute and ultimately matters [eos] mr. evans said theboard in [unk] nt is part of the company s statements before the quake and we want to stay

(b) **3 worst**: More often, models make mistakes. In these worst sentences, models geneated many special tokens like [unk], [eos], and some special characters, like N and $. which really didn't make any sense. This phenomenon also indicates that sequential models didn't learng the deep level relationship between words in contexts.

i. (RNN) reservations the [unk] of the agency [eos] however after july to catch its global program thatwherever the vote these managers have allowed fewer than N billion in west germany [eos] britishgiants meanwhile the problem seen the loan-loss tax and two years a letter among the leader of the[unk] [unk] [unk] [unk] [unk] moving widespread about N N N [eos] the dollar previously used injapan s interest

ii. (RNN) guy with a [unk] [unk] network fee that the rival men are up [eos] in addition the male stoneall the cool perhaps no $N million last year [eos] its [unk] the [unk] on

iii. (RNN) trusts with the company 's major british company 's first chief [eos] this is a third-quarter third-quarter loss fell N N to $ N million from $ N marks a year earlier in the third quarter included half [eos] the quarterly $ N billion [eos] u.s. reported net income income [unk] up a third quarter as N annually though [eos] amex officials said advertising rose N N [eos] the company

(c) **3 interesting**: some phrases(sentences) are interesting. Like in the following setences, we saw "soviet union", which surely occurs in our training data, otherwise, the model can hardly invent such phrases. In second setence, word "taiwan" and "canada" are put together which, from context, may be a name of one country. In the thrid sentence, we can also see a country name: "the united states". So many country name and some political vocabularies. It's all because of the distribution of our trianing data.

i. (RNN) constituency by the soviet union [eos] so far he is ever more optimistic [eos] in the aftermathit s important to be a network oct. N that will be opened by the west german basin

ii. (GRU) reason to plunge says richard [unk] a vice president of taiwan canada [eos] he finds that some stocks including kodak economists have been given to keep the popular layer of their way to keep them

iii. (GRU) encourages the risks in a number of hurdles [eos] the following and dozens of the underlying decisions filled with its start of the process in good long-distance exporting areas a historic approach for corporate californians all wore and track tv commission said a petition spokesman said [eos] the case had sent installed legislation in sanctions of the united states or about $ N million liberal by [unk] with [unk] hawaii

Appendix: 40 sequences generated by trained RNN and GRU models

10 sequences generated by model: RNN with length 35

Seq 0: therapy [eos] the [unk] [unk] set a higher and the value of the junk crimes it had no simple [unk] management and give three investment and i have threatening their own shareholders goes too often

Seq 1: well-known to its more impact on inflation and using them argued that utilization bankers deficit added that inflation value should add finances and from the company of intense [eos] for the shift in new york

Seq 2: us$ N million from $ N that they crash [unk] as it 's too before [eos] ford and native family that he is n't hopes anything tried to [unk] temporarily target [eos] at N and

Seq 3: glenn prior [eos] many analysts said why [unk] still very further [unk] his architecture game will be up for a ban program for people [eos] he said to be mcgovern for the rules because of

Seq 4: tightening which also will reduce the ownership of car makers and digital 's voting to take a bid of [unk] [unk] intended to make the market is used from but that capital credit [unk] [eos]

Seq 5: guy with a [unk] [unk] network fee that the rival men are up [eos] in addition the male stone all the cool perhaps no $ N million last year [eos] its [unk] the [unk] on

Seq 6: catching [eos] in the future annual cost of coming in the following [eos] one of on the times two-thirds of it will save off sale [eos] in the u.s. auto files a N incident from

Seq 7: vote where it was the N on the communists in other matters business [eos] british district [unk] robert [unk] in an van [unk] on former buildings and complained after mr. jones 's announcement [eos] a

Seq 8: focus on his own [unk] including a recent market [unk] it made on satisfaction [eos] westridge capital media liability has been acquired tomorrow articles by use off but tricky [eos] for example sales as mich.

Seq 9: biggest practice fashion and the government forgotten to hold control as much as how investors but the linda [eos] medicine teach expanding mr. bush would lend complained to federal funding elsewhere [eos] commercial groups were

10 sequences generated by model: RNN with length 70

Seq 0: deregulation of zero-coupon commodities ranging from government [eos] among people customers panic dozen basic cities stepped out of tradition the [unk] service and wages where some trade effects on the practice of new and moscow fraud in a european territory [eos] most central agency will pages $ N billion in exports during the next year [eos] despite all increased chevy a the federal enactment of maine said while now will

Seq 1: treat [unk] 's through his price of car [eos] so far further over hearings was that [unk] is greater

[unk] and the kids of an gives the desperately tendency to pick [eos] what now 's wrongdoing early [eos] but he was called at least an unsuccessful bid [eos] the project is the way mr. columbia 's executives were [unk] more than the companies [eos] he often too more than you

Seq 2: reservations the [unk] of the agency [eos] however after july to catch its global program that wherever the vote these managers have allowed fewer than N billion in west germany [eos] british giants meanwhile the problem seen the loan-loss tax and two years a letter among the leader of the [unk] [unk] [unk] [unk] [unk] moving widespread about N N N [eos] the dollar previously used in japan 's interest

Seq 3: trusts with the company 's major british company 's first chief [eos] this is a third-quarter third-quarter loss fell N N to $ N million from $ N marks a year earlier in the third quarter included half [eos] the quarterly $ N billion [eos] u.s. reported net income income [unk] up a third quarter as N annually though [eos] amex officials said advertising rose N N [eos] the company

Seq 4: civic to a [unk] [unk] contest for abc insurance and the costs bureau a provision to recall the indefinitely on the earthquake were concerned [eos] the mr. conducted 's domestic projects on the end of states about far further exports of health operations that written changed any revenue [eos] the health than the justice department nature [unk] trust and boosted reducing loans finally before earthquake [unk] speech is needed

Seq 5: bridges are expected to be broadcast same crack it must be clear by their changes in widely asking behind their public rules to call its bid came from [unk] in singapore arms companies [eos] the official said that in the wake of the fha was under an increase in his acts received as N or the games [eos] the meeting of default most businesses as the [unk] [unk] is the

Seq 6: shaping [eos] a former chairman of u.s. [unk] men were out was n't a [unk] workplace dean witter reynolds said said it plans to take rough 's directly [eos] ge capital that it 's audit in the airline to label [eos] the following the british dollar controls in a share [eos] john trade 's currency has forced some exchange five weeks and strong loans for wall street services because he

Seq 7: trendy said to u.s. exclusive american [unk] the division [eos] the japanese all of federal credit [unk] [unk] test marketing may provide their for N to [unk] workers in west airlines are and marketing and reduce filters in the [unk] and but all banks lawsuits at japanese banks ' small interests in march N the senate members [eos] and legislation will be allowed no provisions and attracting confidence within three

Seq 8: proper health and press [eos] instead that may happen to both knowledge diseases a bit of the impact on the national 's heart the [unk] offerings say most rating was an longest number of [unk] administration and quantum burden increases in mr. [unk] [eos] but of winning mr. rubles to any network the irs contingent [eos] an estimated the administration wants one [unk] may pick a cuts penalty reform pay

Seq 9: public-relations business is very far rescue eric image and trying to making [unk] [unk] by the laws and monetary system [eos] the [unk] allen texas which represents the chicago last night its continuing sports cowboys this network was [unk] a lot of [unk] [eos] but by each [unk] in the shaky future sports campaign he became a mount a a bill issue according to it with the economy [eos] in

10 sequences generated by model: GRU with length 35

Seq 0: jeff [unk] who plans to make an dispute and ultimately matters [eos] mr. evans said the board in [unk] n't is part of the company 's statements before the quake and we want to stay

Seq 1: reason to plunge says richard [unk] a vice president of taiwan canada [eos] he finds that some stocks including kodak economists have been given to keep the popular layer of their way to keep them

Seq 2: stretched over a [unk] period [eos] the gasb is [unk] etc. deviation hence that 's collapse of [unk] political hospitals of the world [eos] indeed [unk] for example 's take its role in the face

Seq 3: joseph roderick [eos] and on individual a [unk] checks after the front of a report that i can be in trouble each or no less just none outside its coups [eos] each spokeswoman 's performance

Seq 4: constituency by the soviet union [eos] so far he is ever more optimistic [eos] in the aftermath it 's important to be a network oct. N that will be opened by the west german basin

Seq 5: dover [eos] mr. wathen was currently acting as suez succeeds mary [unk] a the company around [unk] many firm 's major coal [eos] while pretax margins much nothing in the denominations of weak turnover [eos]

Seq 6: soften and others [eos] the steady surge is expected to be shipped an next month actually is to be auctioned several days or later he has n't fully confirm [eos] he said though the bottom

Seq 7: persian a [unk] of art in particular [unk] costa [unk] and [unk] toward a smiling of [unk] [eos] future was a few months ago [eos] officials expect the strongest problems of kangyo officials say there

Seq 8: unwanted problems [eos] he contends that that 's [unk] large most businesses that may be taken over the time [eos] the recent [unk] is going to be taken behind the foundations of [unk] [eos] thomas

Seq 9: bronfman [unk] of [unk] [eos] the statement of commenting upon [unk] regardless of your maturities including trial [eos] [unk] [unk] an distributor who [unk] at last solution 's shows that called the [unk] building over

10 sequences generated by model: GRU with length 70

Seq 0: fabrication was mainly due to the natural gas joint-venture system [eos] he also has N more than $ N million his [unk] in the u.s. 's new [unk] credit of the cities at minnesota [eos] the congressman also urged asking of programs of reserve dr. [unk] 's [unk] of a disproportionate date from the movie of socialism [eos] however he will take part of a surprising bid on increasing for

Seq 1: reclaim the post on toronto stock [eos] in composite trading on the american stock exchange $ N order yesterday closed at $ N a share down N cents [eos] the big board 's board has been minor [unk] to the efficiency of large in N [eos] the tokyo industry association said they make a wide

investment of the stock market [unk] at only last friday 's market conditions on individual

Seq 2: golden gate and has obtained under [unk] to [unk] and [unk] things from his [unk] [eos] mr. holmes said he is talking in donations [eos] we do n't have any immediate impact of our reader where it had [eos] we do n't know how i go what is going [eos] but i feel our [unk] said [unk] now aims [eos] he also urged the hiroshima to provide a hidden [unk]

Seq 3: otc market [eos] the money is on the best way and the culmination of the big board to surrender such cash [unk] for the issue market in formal affairs that he had settled out of peru about N minutes a day in a [unk] period [eos] mercantile investors noted that and the continent 's [unk] provisions will be the highest in the original market [eos] if you have received their

Seq 4: textile technologies based in new york [eos] mr. mcgovern could n't be reached for comment [eos] midland [unk] N chairman of fireman 's group said he is interested in new york-based city of texas oil and gas from a division of independent utility co [eos] sun with goldman sachs said its market continued to attract investment and it be made employment and [unk] financial operations [eos] the company said it

Seq 5: federally marketing [eos] malaysia officials hewlett-packard co. stamford mass. said it will unveil the mackenzie energy operations offering of $ N million in early next year [eos] the global combined group agreement said it seeks a joint venture with computer with a [unk] form of color chips including a freeport-mcmoran [unk] plant in the netherlands [eos] under terms of the usual [unk] powers to be focusing on hewlett-packard systems that

Seq 6: encourages the risks in a number of hurdles [eos] the following and dozens of the underlying decisions filled with its start of the process in good long-distance exporting areas a historic approach for corporate californians all wore and track tv commission said a petition spokesman said [eos] the case had sent installed legislation in sanctions of the united states or about $ N million liberal by [unk] with [unk] hawaii

Seq 7: ring up and revenue as a guest [eos] [unk] in october N fell a bank of $ N billion in cash on the sale of $ N billion [eos] it brings the $ N billion of tax-exempt missile insurance business sterling and [unk] due august [eos] mr. [unk] said the merger said it will acquire a small standpoint for $ N million within the next year [eos] the new business

Seq 8: unrest for the industry compared with N N earnings to be record [eos] in when-issued trading friday the stock closed at $ N off N cents [eos] good already opened record trading for the year at N N [eos] the u.s. trade deficit is up apart against an important trade deficit of $ N billion a year earlier [eos] in september the trade ministry reported declines of $ N billion

Seq 9: activities who include amex after an unusual period of any [unk] [unk] [unk] field jan. N from calculate advertising was previously jobless below $ N a month [eos] the catalyst for [unk] 's [unk] is the second reality in the N season where as a result of [unk] september breaks cheating [unk] prices [eos] in addition in this years [unk] oil and gas income would be double in the 1970s