Assignment 5: RNNs and Transformers/Self-Attention

Concepts, intuitions and big picture

Each question might have multiple correct answers. (Check all that apply)

1. Which of these types of models would you use for completing prompts with generated text?

   a. An encoder model

   b. A decoder model

   Answer: b

2. Which of these types of models would you use for classifying text inputs according to certain labels?

   a. An encoder model

   b. A decoder model

   Answer: a

3. To which of these tasks would you apply a many-to-one RNN architecture?

   a. Speech recognition (input an audio clip and output a transcript)

   b. Sentiment classification (input a piece of text and output a 0/1 to denote positive or negative sentiment)

   c. Gender recognition from speech (input an audio clip and output a label indicating the speaker's gender)

   Answer: b and c

4. What possible source can the bias observed in a model have?

   a. The model is a fine-tuned version of a pretrained model and it picked up its bias from it.

   b. The data the model was trained on is biased.

   c. The metric the model was optimizing for is biased.

   Answer: a, b, and c

5. How many dimensions does the tensor output by a decoder Transformer model have, and what are they?

   a. 2: The sequence length and the batch size

   b. 2: The sequence length and the hidden size

   c. 3: The sequence length, the batch size, and the hidden size

   Answer: c

6. You are training an RNN language model. At the tth time step, what is the RNN doing? Choose the best answer.

   a. Estimating $P(y_1, y_2, \ldots, y_{t-1})$

   b. Estimating $P(y_1)$

   c. Estimating $P(y_t|y_1, y_2, \ldots, y_{t-1})$

   d. Estimating $P(y_t|y_1, y_2, \ldots, y_t)$

   Answer: c

7. You are training an RNN, and find that your weights and activations are all taking on the value of NaN (“Not a Number”). Which of these is the most likely cause of this problem?

   a. Vanishing gradient problem.

   b. Exploding gradient problem.

   c. ReLU activation function g(.) used to compute g(z), where z is too large.

   d. Sigmoid activation function g(.) used to compute g(z), where z is too large.

   Answer: b

8. You're done training an RNN language model. You're using it to sample random sentences as follows. What are you doing at each time step t?

   a. (i) Use the probabilities output by the RNN to pick the highest probability word for that time-step as y_t. (ii) Then pass the ground-truth word from the training set to the next time-step.

   b. (i) Use the probabilities output by the RNN to randomly sample a chosen word for that time-step as y_t. (ii) Then pass the ground-truth word from the training set to the next time-step.

   c. (i) Use the probabilities output by the RNN to pick the highest probability word for that time-step as y_t. (ii) Then pass this selected word to the next time-step.

   d. (i) Use the probabilities output by the RNN to randomly sample a chosen word for that time-step as y_t. (ii) Then pass this selected word to the next time-step.

   Answer: c and d (greedy selection and random sampling)

9. You have a pet crab whose mood is heavily dependent on the current and past few days' weather. You've collected data for the past 42 days on the weather, which you represent as a sequence as $x_1, \ldots, x_{42}$. You've also collected data on your crab's mood, which you represent as $y_1, \ldots, y_{42}$. You'd like to build a model to map from $x \rightarrow y$. Should you use a Unidirectional RNN or Bidirectional RNN for this problem?

   a. Bidirectional RNN, because this allows the prediction of mood on day t to consider more information.

   b. Bidirectional RNN, because this allows backpropagation to compute more accurate gradients.

   c. Unidirectional RNN, because the value of y_t depends only on $x_1, \ldots, x_t$ but not on $x_{t+1}, \ldots, x_{42}$

   d. Unidirectional RNN, because the value of y_t depends only on x, and not other days' weather.

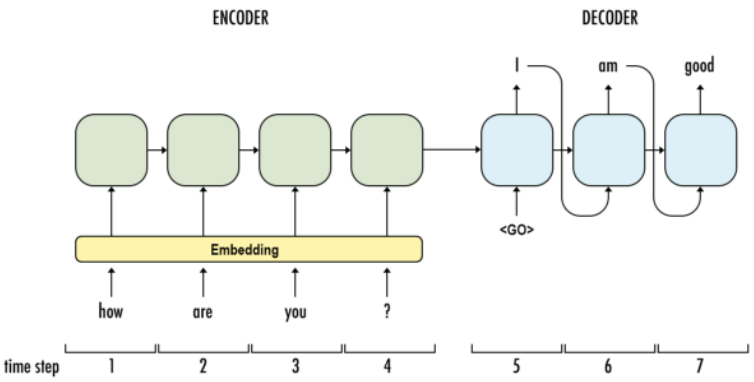
   Answer: c

10. Consider this encoder-decoder model. This model is a "conditional language model" in the sense that the encoder portion (shown in green) is modeling the probability of the input sentence x.

    a. True

    b. False

    Answer: b



11. Compared to the RNN-LMs and fixed-window-LMs, we expect the attention-based LMs to have the greatest advantage when:

    a. The input sequence length is large.

    b. The input sequence length is small.

    Answer: a

12. What is a model head?

    a. A component of the base Transformer network that redirects tensors to their correct layers

    b. Also known as the self-attention mechanism, it adapts the representation of a token according to the other tokens of the sequence

    c. An additional component, usually made up of one or a few layers, to convert the transformer predictions to a task-specific output

    Answer: c

13. What are the techniques to be aware of when batching sequences of different lengths together?

    a. Truncating

    b. Returning tensors

    c. Padding

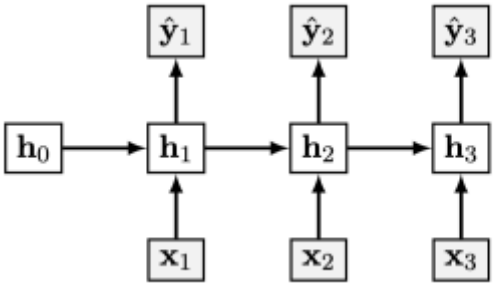
    d. Attention masking

    Answer: a, c, d

Training a Recurrent Neural Net

Consider the following simple RNN architecture: Where the layers and their

corresponding weights are given below:



$x_t \in R^3$

$W_{hx} \in R^{4\times 3}$

$h_t \in R^4$

$W_{yh} \in R^{2\times 4}$

$y_t, \hat{y}_t \in R^2$

$W_{hh} \in R^{4\times 4}$

$J = -\sum_{t=1}^{3}\sum_{i=1}^{2} y_{t,i}\log(\hat{y}_{t,i})$

$\hat{y}_t = \sigma(o_t)$

$o_t = W_{yh}h_t$

$h_t = \psi(z_t)$

$z_t = W_{hh}h_{t-1} + W_{hx}x_t$

Where σ is the softmax activation and ψ is the identity activation (i.e. no activation).

3.1 Computation Graph

Draw the computational graph for the given model.

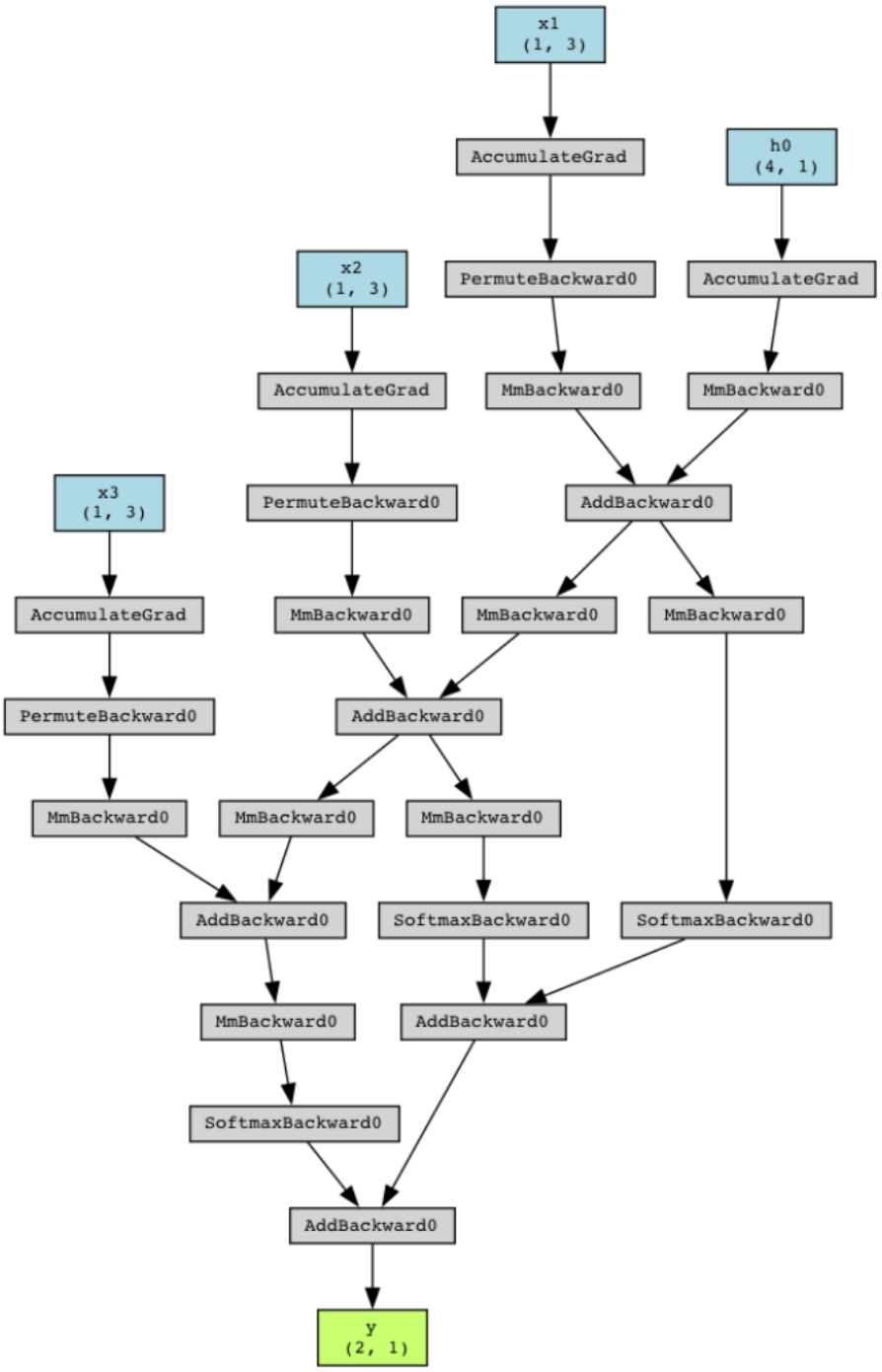


Figure1: Q2.1 Computation graph

3.2 Backprop for RNN

Now you will derive the steps of the backpropagation algorithm that lead to the computation of $\frac{\partial J}{\partial W_{hh}}$. For all parts of this question, please write your answer in terms of W_{hh}, W_{yh}, y, $\hat{y}$, h, and any additional terms specified in the question (note: this does not mean that every term listed shows up in every answer, but rather that you should simplify terms into these as much as possible when you can).

1. Let's define a cross-entropy for our RNN at time t: $J_t = -\sum_{i=1}^{2} y_{t,i} \log(\hat{y}_{t,i})$. What is $\frac{\partial J_t}{\partial o_t}$? Write your answer and show your work. Hint: First derive the partial

derivatives with respective to everything after h_t. After that, try to break up the objective into a term for each timestep.

Answer:

$J_t = -\sum_{i=1}^{2} y_{t,i} \log(\hat{y}_{t,i})$ and $\hat{y}_t = \sigma(o_t)$, where σ is the softmax function.

The derivative of the cross-entropy loss with respect to the output of the softmax is $\frac{\partial J_t}{\partial \hat{y}_{t,i}} = -\frac{y_{t,i}}{\hat{y}_{t,i}}$.

The derivative of the softmax output $\hat{y}_{t,i}$ with respect to the input $o_{t,j}$ is $\frac{\partial \hat{y}_{t,i}}{\partial o_{t,j}} = \hat{y}_{t,i}(\delta_{ij} - \hat{y}_{t,j})$, where δ_{ij} is the Kronecker delta (1 if i=j, 0 otherwise).

Using the chain rule, the derivative of J_t with respect to $o_{t,j}$ is:

$$\frac{\partial J_t}{\partial o_{t,j}} = \sum_{i=1}^{2} \frac{\partial J_t}{\partial \hat{y}_{t,i}} \frac{\partial \hat{y}_{t,i}}{\partial o_{t,j}}$$

$$\frac{\partial J_t}{\partial o_{t,j}} = \sum_{i=1}^{2} \left(-\frac{y_{t,i}}{\hat{y}_{t,i}} \right) \left(\hat{y}_{t,i}(\delta_{ij} - \hat{y}_{t,j}) \right)$$

$$\frac{\partial J_t}{\partial o_{t,j}} = -\sum_{i=1}^{2} y_{t,i}(\delta_{ij} - \hat{y}_{t,j})$$

$$\frac{\partial J_t}{\partial o_{t,j}} = -\left(y_{t,j}(\delta_{jj} - \hat{y}_{t,j}) + y_{t,k}(\delta_{kj} - \hat{y}_{t,j}) \right) \quad \text{where } k \neq j$$

$$\frac{\partial J_t}{\partial o_{t,j}} = -\left(y_{t,j}(1 - \hat{y}_{t,j}) + y_{t,k}(0 - \hat{y}_{t,j}) \right)$$

$$\frac{\partial J_t}{\partial o_{t,j}} = -(y_{t,j} - y_{t,j}\hat{y}_{t,j} - y_{t,k}\hat{y}_{t,j})$$

Since $\sum y_{t,i} = 1$ (one-hot encoding), $y_{t,k} = 1 - y_{t,j}$.

$$\frac{\partial J_t}{\partial o_{t,j}} = -(y_{t,j} - y_{t,j}\hat{y}_{t,j} - (1 - y_{t,j})\hat{y}_{t,j})$$

$$\frac{\partial J_t}{\partial o_{t,j}} = -(y_{t,j} - y_{t,j}\hat{y}_{t,j} - \hat{y}_{t,j} + y_{t,j}\hat{y}_{t,j})$$

$$\frac{\partial J_t}{\partial o_{t,j}} = -(y_{t,j} - \hat{y}_{t,j})$$

In vector form:

$$\frac{\partial J_t}{\partial o_t} = \hat{y}_t - y_t$$

2. Suppose you have a variable g_{ot} that stores the value of $\frac{\partial J_t}{\partial o_t}$. What is $\frac{\partial J_t}{\partial h_i}$ for an

arbitrary $i \in [1, 3]$ ($i \leq t$)? Write your solution in terms of the g_{ot} and the aforementioned variables and show your work.

Answer:

From the computational graph and the equations, $o_t = W_{yh} h_t$.

Using the chain rule, the derivative of J_t with respect to h_t is:

$$\frac{\partial J_t}{\partial h_t} = \frac{\partial J_t}{\partial o_t} \frac{\partial o_t}{\partial h_t}$$

We are given that $g_{ot} = \frac{\partial J_t}{\partial o_t}$. The derivative of o_t with respect to h_t is $\frac{\partial o_t}{\partial h_t} = W_{yh}^T$.

Substituting these into the chain rule equation:

$$\frac{\partial J_t}{\partial h_t} = g_{ot} \cdot W_{yh}^T$$

For $i < t$, the derivative $\frac{\partial J_t}{\partial h_i} = 0$ because J_t only directly depends on h_t.

Thus, for $i \leq t$: If $i = t$, $\frac{\partial J_t}{\partial h_t} = g_{ot} \cdot W_{yh}^T$ If $i < t$, $\frac{\partial J_t}{\partial h_i} = 0$

3. Suppose you have a variable g_{h_i} that stores the value of $\frac{\partial J_t}{\partial h_i}$. What is $\frac{\partial J_t}{\partial W_{hh}}$? Write your solution in terms of the g_{h_i} and the aforementioned variables. Show your work in the second.

Answer:

From the computational graph and the equations, $z_t = W_{hh} h_{t-1} + W_{hx} x_t$ and $h_t = \psi(z_t)$. Since ψ is the identity activation, $h_t = z_t$.

Therefore, $h_t = W_{hh} h_{t-1} + W_{hx} x_t$.

Using the chain rule, $\frac{\partial J_t}{\partial W_{hh}}$ can be written as:

$$\frac{\partial J_t}{\partial W_{hh}} = \frac{\partial J_t}{\partial h_t} \frac{\partial h_t}{\partial z_t} \frac{\partial z_t}{\partial W_{hh}}$$

From Question 3.2.2, for $i = t$, $\frac{\partial J_t}{\partial h_t} = g_{h_t}$.

Also, since ψ is the identity function, $\frac{\partial h_t}{\partial z_t} = \psi'(z_t) = 1$.

To find $\frac{\partial z_t}{\partial W_{hh}}$, the equation for z_t is $z_t = W_{hh} h_{t-1} + W_{hx} x_t$. The derivative of the term $W_{hh} h_{t-1}$ with respect to W_{hh} is h_{t-1}^T. The derivative of the term $W_{hx} x_t$ with respect to W_{hh} is 0 since W_{hh} is not in this term. So, $\frac{\partial z_t}{\partial W_{hh}} = h_{t-1}^T$.

Substituting these into the chain rule:

$$\frac{\partial J_t}{\partial W_{hh}} = g_{h_t} \cdot 1 \cdot h_{t-1}^T$$

$$\frac{\partial J_t}{\partial W_{hh}} = g_{h_t} h_{t-1}^T$$

This is the partial derivative of the loss at time t with respect to W_{hh}.

4. Suppose you have a variable $g_{Whh,t}$ that stores the value of $\frac{\partial J_t}{\partial W_{hh}}$. What is $\frac{\partial J}{\partial W_{hh}}$? Write your solution in terms of the $g_{Whh,t}$ and the aforementioned variables. Show your work.

   Answer:

   The total loss J is the sum of the losses at each time step: $J = \sum_{t=1}^{3} J_t$.

   To find the total derivative of J with respect to W_{hh}, we need to sum the partial derivatives of J_t with respect to W_{hh} for each time step t, since W_{hh} is used in the calculation of h_t at each time step.

   Per the question $g_{Whh,t} = \frac{\partial J_t}{\partial W_{hh}}$.

   Therefore, the total derivative $\frac{\partial J}{\partial W_{hh}}$ is the sum of $g_{Whh,t}$ over all time steps from $t = 1$ to $t = 3$:

$$\frac{\partial J}{\partial W_{hh}} = \sum_{t=1}^{3} \frac{\partial J_t}{\partial W_{hh}}$$

$$\frac{\partial J}{\partial W_{hh}} = \sum_{t=1}^{3} g_{Whh,t}$$

   This represents the total gradient of the loss J with respect to the recurrent weight matrix W_{hh}, which is calculated by summing the gradients backpropagated through each time step.

Self-Attention and Transformers

Recall that the transformer architecture uses scaled dot-product attention to compute attention weights:

$$\alpha^{(t)} = \text{softmax}\left(\frac{q_t K^\top}{\sqrt{h}}\right) \in [0,1]^n$$

The resulting embedding in the output of attention at position t are:

$$\text{Attention}(Q,K,V)_t = \sum_{t'=1}^{n} \alpha_{t'}^{(t)} v_{t'} \in \mathbb{R}^{1\times h}$$

where $\alpha^{(t)} = [\alpha_0^{(t)}, \ldots, \alpha_n^{(t)}]$. The same idea can be stated in a matrix form:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{h}}\right) V \in \mathbb{R}^{n \times h}$$

In the above equations:

- h is the hidden state dimension and n is the input sequence length;
- $X \in \mathbb{R}^{n \times h}$ is the input to the attention;
- $x_t \in \mathbb{R}^{1 \times h}$ is the slice of X at position t, i.e. vector representation (embedding) of the input token at position t;
- $W_q, W_k, W_v \in \mathbb{R}^{h \times h}$ are the projection matrices to build query, key and value representations;
- $Q = XW_q \in \mathbb{R}^{n \times h}$, $K = XW_k \in \mathbb{R}^{n \times h}$, $V = XW_v \in \mathbb{R}^{n \times h}$ are the query, key and value representations;
- $q_t = x_t W_q \in \mathbb{R}^{1 \times h}$ is the slice of Q at position t. Similarly, $k_t = x_t W_k \in \mathbb{R}^{1 \times h}$ and $v_t = x_t W_v \in \mathbb{R}^{1 \times h}$.

Now answer the following questions:

4.1 Complexity

What is the computational complexity of a self-attention layer in terms of n and h? In particular, show that the complexity of a self-attention layer at test-time scales quadratically with the sequence length n. Lastly, explain (no more than 5 sentences) why this can be computed efficiently on GPUs, despite being a quadratic function of sequence length n.

Answer:

The self-attention layer's main calculations involve comparing every word in the sequence to every other word therein to determine their relevance to understanding the current word's context. This information is the combined for all the words. If you have n words, this involves about $n \times n$ comparisons and combinations for each feature dimension h. This makes the main computational cost scale with $n^2 \times h$, or $O(n^2 h)$. This shows the complexity grows quadratically with the sequence length n.

Although the computation of self-attention has a quadratic complexity (i.e. n^2), GPU's can compute this efficiently because these calculation utilize matrixes, and GPUs are designed to use their multiple core to perform matrix computations in parallel. As such, they can handle the quadratic growth much better than CPUs.

4.2 Masking in Self-Attention

Suppose we are using a token-making objective to create a language generation model that uses self-attention. For example, suppose we want to mask $t = 3$ position. Then, it does not make sense for $q_t : \forall t \in [0 \ldots n]$ to look at k_3 and v_3. Describe one way we can go about implementing such masking.

Answer:

A common method for masking is using an **attention mask** applied before the softmax on the attention scores. To do this:

1. Calculate the attention scores using $\frac{QK^T}{\sqrt{h}}$.
2. Create an $n \times n$ mask. For language generation, this typically ends up being a lower triangular matrix. To mask specific positions (like column 3), set those mask values to a very large negative number.
3. To apply the mask, add the mask to the attention scores, i.e. $AttentionScores_{masked} = AttentionScores + Mask$
4. Apply softmax. The large negative values in the masked scores result in near-zero attention weights for the masked positions. This prevents the model from attending to masked tokens, which is crucial for causal language generation where tokens should not see future information.

Extra Credit: Linear Attention with SVD

It has been empirically shown in Transformer models that the context mapping matrix $P = \mathbf{softmax}\left(\frac{QK^\top}{\sqrt{h}}\right)$ often has a low rank. Show that if the rank of P is k and we already have access to the SVD of P, then it is possible to compute self-attention in $O(nkh)$ time.

Answer:

The Self-Attention output is PV, where P is the attention matrix. If P has rank k, its SVD is $P = U_k \Sigma_k V_k^\top$, where $U_k \in \mathbb{R}^{n \times k}$, $\Sigma_k \in \mathbb{R}^{k \times k}$, and $V_k \in \mathbb{R}^{n \times k}$.

PV can be computed efficiently by re-associating the matrix multiplication:

$$PV = (U_k \Sigma_k V_k^\top) V_{original} = U_k (\Sigma_k (V_k^\top V_{original}))$$

where $V_{original} \in \mathbb{R}^{n \times h}$ is the Value matrix.

To analyze the complexity complexity:

1. $V_k^\top V_{original}$: $(k \times n) \times (n \times h) \to O(nkh)$
2. $\Sigma_k (V_k^\top V_{original})$: $(k \times k) \times (k \times h) \to O(kh)$ (diagonal matrix multiplication)
3. $U_k (\Sigma_k (V_k^\top V_{original}))$: $(n \times k) \times (k \times h) \to O(nkh)$

The dominant complexity is $O(nkh)$. Thus, if P is low-rank k and its SVD is available, self-attention can be computed in $O(nkh)$ time.

4.4 Extra Credit: Linear Attention by Projection

Suppose we ignore the Softmax and scaling and let $P = \frac{QK^\top}{h} \in \mathbb{R}^{n \times n}$. Assume P is rank k. Show that there exist two linear projection matrices $C, D \in \mathbb{R}^{k \times n}$ such that $PV = Q(CK)^\top DV$ and the right hand side can be computed in $O(nkh)$ time. Hint: Consider using SVD in your proof.

4.4 Extra Credit: Linear Attention by Projection

Suppose we ignore the Softmax and scaling and let $P = \frac{QK^\top}{h} \in \mathbb{R}^{n \times n}$. Assume P is rank k. Show that there exist two linear projection matrices $C, D \in \mathbb{R}^{k \times n}$ such that $PV = Q(CK)^\top DV$ and the right hand side can be computed in $O(nkh)$ time. Hint: Consider using SVD in your proof.

Answer:

Given $PV = \frac{QK^\top}{h}V$, the goals is to show that $PV = Q(CK)^\top DV$ and the complexity of the right hand side is $O(nkh)$.

To analyze the computational complexity of $Q(CK)^\top DV$ given $Q \in \mathbb{R}^{n \times d}$, $K \in \mathbb{R}^{n \times d}$, $V \in \mathbb{R}^{n \times h}$, $C \in \mathbb{R}^{k \times n}$, $D \in \mathbb{R}^{k \times n}$.

- $CK \in \mathbb{R}^{k \times d}$ ($O(nkd)$)
- $Q(CK)^\top \in \mathbb{R}^{n \times k}$ ($O(ndk)$)
- $DV \in \mathbb{R}^{k \times h}$ ($O(nkh)$)
- $Q(CK)^\top DV \in \mathbb{R}^{n \times h}$ ($O(nkh)$)

The dominant complexity is $O(nkh)$.

To show the existence of $C, D \in \mathbb{R}^{k \times n}$ such that $\frac{1}{h}QK^\top V = QK^\top C^\top DV$, we need $\frac{1}{h}K^\top \approx K^\top C^\top D$.

The low rank k of $P = \frac{QK^\top}{h}$, as hinted in the problem's statement, implies a low-rank structure that allows for the existence of $C, D \in \mathbb{R}^{k \times n}$ such that the equality holds.

Assuming the existence of such C and D as stated, the computational complexity of $Q(CK)^\top DV$ is $O(nkh)$.

5 Programming

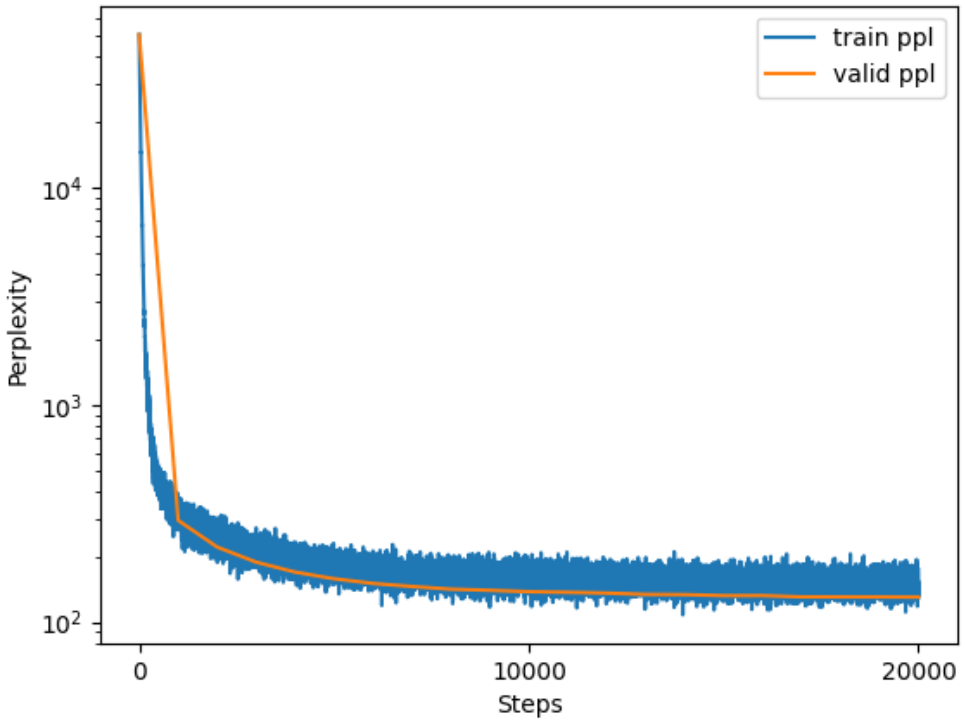
5.1.9 Train and Evaluate GPT LM on GPUs

Printout of cpu_gpu_comparison

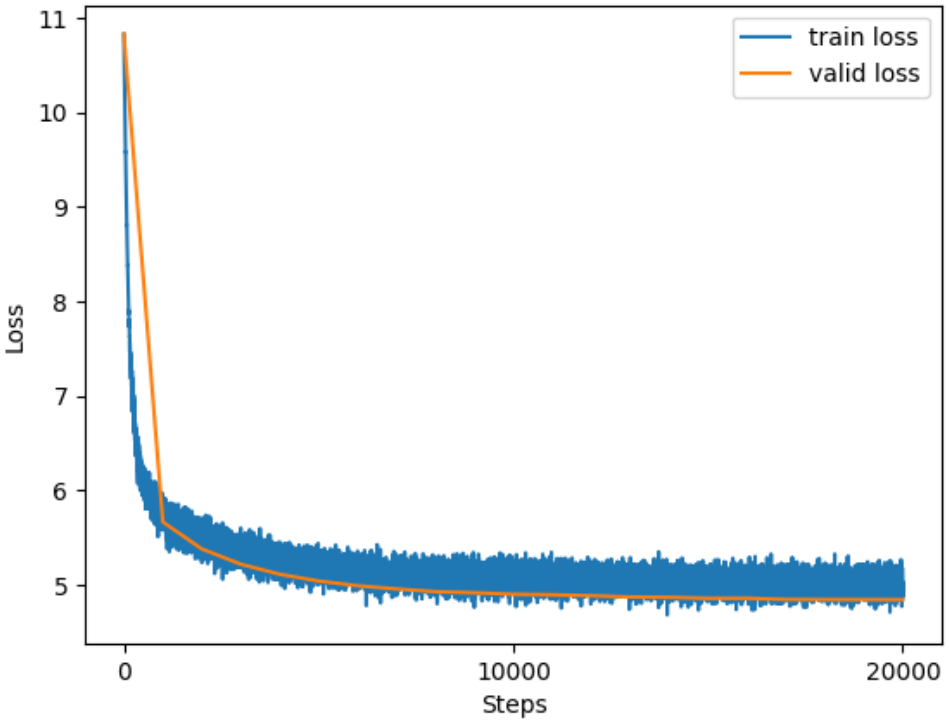
```
---------- Trainer Time Elapsed: 61.11s ----------
number of parameters: 2.50M
running on device cuda
iter 0: validation loss 10.83610 validation ppl 50823.046
best model so far, saving...
iter_dt 0.00s; iter 0: train loss 10.83670 train ppl
50853.211
---------- Trainer Time Elapsed: 3.65s ----------
```

Output of gpu_full_run

Perplexity


train ppl
valid ppl
Perplexity
10^4
10^3
10^2
0
10000
20000
Steps

Loss


train loss
valid loss
Loss
11
10
9
8
7
6
5
0
10000
20000
Steps