Assignment 3

Release Date: 2025-03-12

Deadline: Friday, April 4th, at 1:00pm.

Submission: You must submit two files through MarkUs: (1) a PDF file containing your writeup, titled `a3-writeup.pdf`, and (2) your code file `nmt.ipynb, bert.ipynb, clip.ipynb`. There will be sections in the notebook for you to write your responses. Your writeup may be typed or handwritten, but please ensure it is legible. Make sure that the relevant outputs (e.g. `print_gradients()` outputs, plots, etc.) are included and clearly visible.

See the syllabus on the course website for detailed policies. You may ask questions about the assignment on Piazza.

The assignment is marked out of 100 points. 10 points are allocated to neatness, so be sure that your answers and code are clear and readable!

You may notice that some questions are worth 0 points, which means we will not mark them in this assignment. However, we think these questions are educationally valuable and may help you prepare for the tests, so we recommend you spend some time thinking about them.

Important Instructions

Read the following before attempting the assignment.

Collaboration Policy

You are welcome to work together with other students on the homework. You are also welcome to use any resources you find (online tutorials, textbooks, papers, etc.) to help you complete the homework. However, you must write up your submission by yourself and not use any content generated by someone else or generative AI. You must cite all the collaboration and resources you used to complete each assignment. If you hand in homework that makes use of content that you did not create or you do not disclose the collaboration or resources, you will get a 0 for that homework. Note also that if you rely too much on outside resources, you will likely not learn the material and will do poorly on the exams, during which such resources will not be available.

Generative AI Policy

You may ask general questions about concepts related to the homework questions. However, you may not directly ask them for hints on homework assignment questions; e.g., you should not be copying and pasting directly from the assignment handout into the chat interface. Also, you may not directly use the outputs of AI chatbots in your homework solutions (even paraphrased) unless instructed to do so.

You must include any chat transcripts related to a homework assignment along with your submission for the assignment. We will interpret the above policy leniently when judging whether the GenAI use is appropriate, though we reserve the right to update the GenAI policy if we find that students are using it in a way that reduces the educational value of the homeworks.

If you use GenAI, then your transcripts should be submitted on MarkUs with filenames starting with `chat_transcript`.

Written Assignment

What you have to submit for this part

For reference, here is everything you need to hand in for the first half of the PDF report `a3-writeup.pdf`.

- **Problem 1:** 1.1.2, 1.2.1, 1.3.1, 1.3.2

1 RNNs and Attention [20pts]

For any successful deep learning system, choosing the right network architecture is as important as choosing a good learning algorithm. In this question, we will explore how various architectural choices can have a significant impact on learning. We will analyze the learning performance from the perspective of vanishing /exploding gradients as they are backpropagated from the final layer to the first.

1.1 Warmup: A Single Neuron RNN

Consider an n layered fully connected network that has scalar inputs and outputs. For now, assume that all the hidden layers have a single unit, and that the weight matrices are set to 1 (because each hidden layer has a single unit, the weight matrices have a dimensionality of $\mathbb{R}^{1\times 1}$).

1.1.1 Effect of Activation - ReLU [0pt]

Lets say we're using the ReLU activation. Let x be the input to the network and let $f : \mathbb{R}^1 \to \mathbb{R}^1$ be the function the network is computing. Do the gradients necessarily have to vanish or explode as they are backpropagated? Answer this by showing that $0 \leq |\frac{\partial f(x)}{\partial x}| \leq 1$.

1.1.2 Effect of Activation - Different weights [5pt]

Solve the problem in 1.1.1 by assuming now the weights are not 1. You can assume that the i-th hidden layer has weight w_i. Do the gradients necessarily have to vanish or explode as they are backpropagated? Answer this by deriving a similar bound as in Sec 1.1.1 for the magnitude of the gradient.

1.2 Matrices and RNN

We will now analyze the recurrent weight matrices under Singular Value Decomposition. SVD is one of the most important results in all of linear algebra. It says that any real matrix $M \in \mathbb{R}^{mxn}$ can be written as $M = U\Sigma V^T$ where $U \in \mathbb{R}^{mxm}$ and $V \in \mathbb{R}^{nxn}$ are square orthogonal matrices, and $\Sigma \in \mathbb{R}^{mxn}$ is a rectangular diagonal matrix with nonnegative entries on the diagonal (i.e. $\Sigma_{ii} \geq 0$ for $i \in \{1, \ldots, \min(m, n)\}$ and 0 otherwise). Geometrically, this means any linear transformation can be decomposed into a rotation/flip, followed by scaling along orthogonal directions, followed by another rotation/flip.

1.2.1 Gradient through RNN [5pt]

Let say we have a very simple RNN-like architecture that computes $x_{t+1} = \text{sigmoid}(Wx_t)$. You can view this architecture as a deep fully connected network that uses the same weight matrix at each layer. Suppose the largest singular value of the weight matrix is $\sigma_{max}(W) = \frac{1}{4}$. Show that the largest singular value of the input-output Jacobian has the following bound:

$$0 \leq \sigma_{max}(\frac{\partial x_n}{\partial x_1}) \leq (\frac{1}{16})^{n-1}$$

(Hint: if $C = AB$, then $\sigma_{max}(C) \leq \sigma_{max}(A)\sigma_{max}(B)$. Also, the input-output Jacobian is the multiplication of layerwise Jacobians).

1.3 Relative Attention

Relative attention We implement relative attention, as in traditional Softmax attention, however we now include a new term that allows us to reweight the attention scores based on the distance between inputs. Now assume that we are working with a single-headed local self-attention mechanism on a one-dimensional sequence $x \in \mathbb{R}^n$ s.t. $x_i \geq 0$ for all i and $W_Q, W_K, W_V \in \mathbb{R}$.

$$\alpha_{i,j}(Q, K, p) = \left(\frac{Q_i K_j}{\sqrt{d_k}} + p_{i-j} \right)$$

where (p_k) is a sequence of scalars and $Q, K \in \mathbb{R}^n$. Note that here we allow the index k to take on negative values.

$$\text{ReltativeAttention}(Q, K, V, p) = \text{softmax}(\alpha(Q, K, p))V$$

And our attention layer is simply:

$$\text{RelAttnLayer}(x; W_Q, W_K, W_V, p) = \text{ReltativeAttention}(W_Q x, W_K x, W_V x, p)$$

1.3.1 Implement a 1D Convolution [5pts]

Implement the following $1D$-convolution with kernel $w = [2, 0, 1]^T$,as using the RelAttnLayer. Specifically, you need to implement:

$$\text{Conv1D}(x; w)_i = \sum_{j=-1}^{1} x_{i+j} w_{j+2}$$

You may ignore the behavior of your implementation in the edge cases when $i > n - 1$ and $i < 1$. Please specify the p, W_Q, W_K, and W_V you used and argue why they closely approximate this function in the relevant range.

1.3.2 Implement Max Pooling [5pts]

Now we will use RelAttnLayer to approximate 1d max-pooling with a stride of 1 and a window size of $2k + 1$ around the current input. Recall that max pooling with a stride 1 and width of 2k + 1 is simply:

$$\text{MaxPool}(x)_i = \max_{-k \leq m \leq k} x_{m+i}$$

Again you may ignore the edge cases in your solutions $i > n - k$ and $i < k + 1$. Please specify the p, W_Q, W_K, and W_V you used and argue why they approximately implement this function in the relevant range.

Programming Assignment

What you have to submit for this part

For reference, here is everything you need to hand in:

- This is the second half of your PDF report `a3-writeup.pdf`. Please include the solutions to the following problems. You may choose to export `nmt.ipynb, bert.ipynb, clip.ipynb` as a PDF and attach it to the first half of `a3-writeup.pdf`.
- Your code file `nmt.ipynb, bert.ipynb, clip.ipynb`

Introduction

In this assignment, you will explore common tasks and model architectures in Natural Language Processing (NLP). Along the way, you will gain experience with important concepts like *attention* mechanisms (Section 2), *pretrained language models* (Section 3) and *multimodal* vision and language models (Section 4). Your code should make use of vectorization whenever possible.

Setting Up

We recommend that you use **Colab**(`https://colab.research.google.com/`) for the assignment. To setup the Colab environment, just open the notebooks for each part of the assignment and **make a copy** in your own Google Drive account.

Deliverables

Each section is followed by a checklist of deliverables to add in the assignment writeup. To also give a better sense of our expectations for the answers to the conceptual questions, we've put maximum sentence limits. You will not be graded for any additional sentences.

2 Neural machine translation (NMT) [40pt]

Neural machine translation (NMT) is a subfield of NLP that aims to translate between languages using neural networks. In this section, will we train a NMT model on the toy task of English → Pig Latin. Open notebook `https://colab.research.google.com/drive/1EasmqdV4sLHscn6-8aqtJPasFa6YHpKt?usp=sharing`, and complete all required parts. Please read the following background section carefully before attempting the questions.

Background

The task

Pig Latin is a simple transformation of English based on the following rules:

1. If the first letter of a word is a *consonant*, then the letter is moved to the end of the word, and the letters "ay" are added to the end: `team` → `eamtay`.
2. If the first letter is a *vowel*, then the word is left unchanged and the letters "way" are added to the end: `impress` → `impressway`.
3. In addition, some consonant pairs, such as "sh", are treated as a block and are moved to the end of the string together: `shopping` → `oppingshay`.

To translate a sentence from English to Pig-Latin, we apply these rules to each word independently:

`i went shopping` → `iway entway oppingshay`

Our goal is to build a NMT model that can learn the rules of Pig-Latin *implicitly* from (English, Pig-Latin) word pairs. Since the translation to Pig Latin involves moving characters around in a string, we will use *character-level* transformer model. Because English and Pig-Latin are similar in structure, the translation task is almost a copy task; the model must remember each character in the input and recall the characters in a specific order to produce the output. This makes it an ideal task for understanding the capacity of NMT models.

The data

The data for this task consists of pairs of words $\{(s^{(i)}, t^{(i)})\}_{i=1}^{N}$ where the *source* $s^{(i)}$ is an English word, and the *target* $t^{(i)}$ is its translation in Pig-Latin.[1] The dataset contains 3198 unique (English, Pig-Latin) pairs in total; the first few examples are:

`{ (the, ethay), (family, amilyfay), (of, ofway), ... }`

In this assignment, you will investigate the effect of dataset size on generalization ability. We provide a small and large dataset. The small dataset is composed of a subset of the unique words from the book "Sense and Sensibility" by Jane Austen. The vocabulary consists of 29 tokens: the 26 standard alphabet letters (all lowercase), the dash symbol `-`, and two special tokens `<SOS>` and `<EOS>` that denote the start and end of a sequence, respectively.[2] The second, larger dataset

[1] In order to simplify the processing of *mini-batches* of words, the word pairs are grouped based on the lengths of the source and target. Thus, in each mini-batch, the source words are all the same length, and the target words are all the same length. This simplifies the code, as we don't have to worry about batches of variable-length sequences.

[2] Note that for the English-to-Pig-Latin task, the input and output sequences share the same vocabulary; this is not always the case for other translation tasks (i.e., between languages that use different alphabets)

is obtained from Peter Norvig's natural language corpus.[3] It contains the top 20,000 most used English words, which is combined with the previous data set to obtain 22,402 unique words. This dataset contains the same vocabulary as the previous dataset.

The model

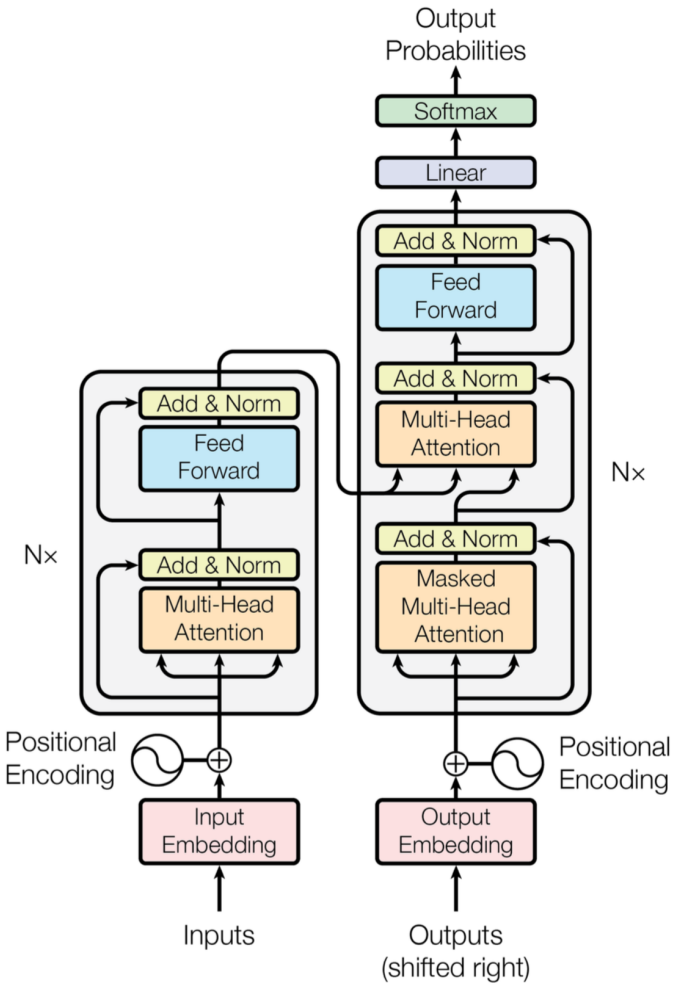

Figure 1: The transformer architecture. Vaswani et al. [2017]

Translation is a *sequence-to-sequence* (seq2seq) problem. The goal is to train a model to transform one sequence into another. A transformer model Vaswani et al. [2017] uses an encoder-decoder architecture and relies entirely on an attention mechanism to draw global dependencies between the input sequence and the output sequence. The encoder processes the input sequence in parallel using stacked self-attention and point-wise fully connected layers, as shown in Figure 1. Given the hidden representations of each input token processed through an encoder, the decoder then generates an output sequence one at a time. The model is auto-regressive when generating the output tokens.

Specifically, input characters are passed through an embedding layer before being fed into an encoder model. If H is the dimension of the encoder hidden state, we learn a $29 \times H$ embedding

[3] https://norvig.com/ngrams/

matrix, where each of the 29 characters in the vocabulary is assigned a H-dimensional embedding. At each time step, the decoder outputs a vector of *unnormalized log probabilities* given by a linear transformation of the decoder hidden state. When these probabilities are normalized (i.e. by passing them through a softmax), they define a distribution over the vocabulary, indicating the most probable characters for that time step. The model is trained via a cross-entropy loss between the decoder distribution and ground-truth at each time step.

2.1 Transformers for NMT (Attention Is All You Need) [14pt]

In order to answer the following questions correctly, please make sure that you have **run the code from `nmt.ipynb`, Part1, `Training and evaluation code` prior to answering the following questions.**

1. [4pt] In lecture, we learnt about Scaled Dot-product Attention used in the transformer models. The function f is a dot product between the linearly transformed query and keys using weight matrices W_q and W_k:

$$\tilde{\alpha}_i^{(t)} = f(Q_t, K_i) = \frac{(W_q Q_t)^T (W_k K_i)}{\sqrt{d}},$$
$$\alpha_i^{(t)} = \text{softmax}(\tilde{\alpha}^{(t)})_i,$$
$$c_t = \sum_{i=1}^{T} \alpha_i^{(t)} W_v V_i,$$

   where, d is the dimension of the query and the W_v denotes weight matrix project the value to produce the final context vectors.

   Implement the scaled dot-product attention mechanism. Fill in the `forward` methods of the `ScaledDotAttention` class. Use the PyTorch torch.bmm (or @) to compute the dot product between the batched queries and the batched keys in the forward pass of the `ScaledDotAttention` class for the unnormalized attention weights.

   The following functions are useful in implementing models like this. You might find it useful to get familiar with how they work. (click to jump to the PyTorch documentation):

   - squeeze
   - unsqueeze
   - expand_as
   - cat
   - view
   - bmm (or @)

   Your forward pass **needs to** work with both 2D query tensor (`batch_size x (1) x hidden_size`) **and** 3D query tensor (`batch_size x k x hidden_size`).

2. [4pt] **Implement the causal scaled dot-product attention mechanism.** Fill in the `forward` method in the `CausalScaledDotAttention` class. It will be mostly the same as the `ScaledDotAttention` class. The additional computation is to mask out the attention to the future time steps. You will need to add `self.neg_inf` to some of the entries in the unnormalized attention weights. You may find torch.tril or torch.triu handy for this part.

3. [2pt] We will now use `ScaledDotAttention` as the building blocks for a simplified transformer Vaswani et al. [2017] encoder.

   The encoder looks like the left half of Figure 1. The encoder consists of three components:

   - Positional encoding: To encode the position of each word, we add to its embedding a constant vector that depends on its position:

     $$\text{pth word embedding} = \text{input embedding} + \text{positional encoding}(p)$$

     We follow the same positional encoding methodology described in Vaswani et al. [2017]. That is we use sine and cosine functions:

     $$\text{PE}(\text{pos}, 2i) = \sin \frac{\text{pos}}{10000^{2i/d_{model}}} \tag{2.1}$$

     $$\text{PE}(\text{pos}, 2i+1) = \cos \frac{\text{pos}}{10000^{2i/d_{model}}} \tag{2.2}$$

     Since we always use the same positional encodings throughout the training, we pre-generate all those we'll need while constructing this class (before training) and keep reusing them throughout the training.
   - A `ScaledDotAttention` operation.
   - A following MLP.

   For this question, describe why we need to represent the position of each word through this positional encoding in one or two sentences. Additionally, describe the advantages of using this positional encoding method, as opposed to other positional encoding methods such as a one hot encoding in one or two sentences.

4. [4pt] In the code notebook, we have provided an experimental setup to evaluate the performance of the Transformer as a function of hidden size and data set size. Run the Transformer model using hidden size 32 versus 64, and using the small versus large dataset (in total, 4 runs). We suggest using the provided hyper-parameters for this experiment.

   Run these experiments, and report the effects of increasing model capacity via the hidden size, and the effects of increasing dataset size. In particular, report your observations on how loss as a function of gradient descent iterations is affected, and how changing model/dataset size affects the generalization of the model. Are these results what you would expect?

   In your report, include the two loss curves output by `save_loss_comparison_by_hidden` and `save_loss_comparison_by_dataset`, the lowest attained validation loss for each run, and your response to the above questions.

Deliverables

Create a section in your report called **Scaled Dot Product Attention**. Add the following:

- Screenshots of your `ScaledDotProduct`, `CausalScaledDotProduct` implementations. Highlight the lines you've added. [4pt]
- Your answer to question 3. [2pt]
- The two loss curves plots output by the experimental setup in question 4, and the lowest validation loss for each run. [4pt]
- Your response to the written component of question 4. Your analysis should not exceed **six** sentences. [4pt]

2.2 Decoder Only NMT [12pt]

In this subsection, we will train a decoder-only NMT model using the `CausalAttention` mechanism. The key difference between this approach and the previous encoder-decoder approach is that we do not encode a hidden state of the input sequence first using an encoder. Instead, we feed both the input sequence and the target sequence to a decoder simultaneously, as in Figure 2. The input sequence and the target sequence will be separated using an end-of-prompt token (`EOP`). The concatenated input to the decoder will have `SOS` token added at the beginning, and the concatenated target will have `EOS` token added at the end. In our provided notebook, the decoder will process this concatenated input using *causal attention*, but we compute the cross-entropy loss by using the output tokens from the output of `<EOP>` only.

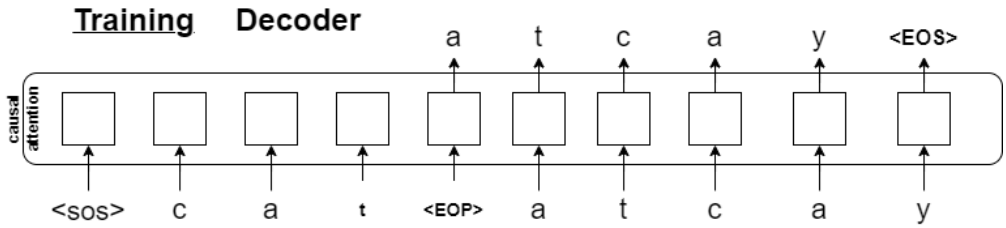

Figure 2: Training the decoder-only NMT model.

For test-time translations, we first feed the input sequence to a trained decoder, enclosed by a `SOS` token and a `EOP` token, as shown in Figure 3. We obtain the first translated token a in this case and concatenate the input sequence with the generated token. Then we feed the concatenated sequence to the decoder and obtain two tokens a and t. This procedure is repeated until reaching the maximum target length or generating a `<EOS>` token.

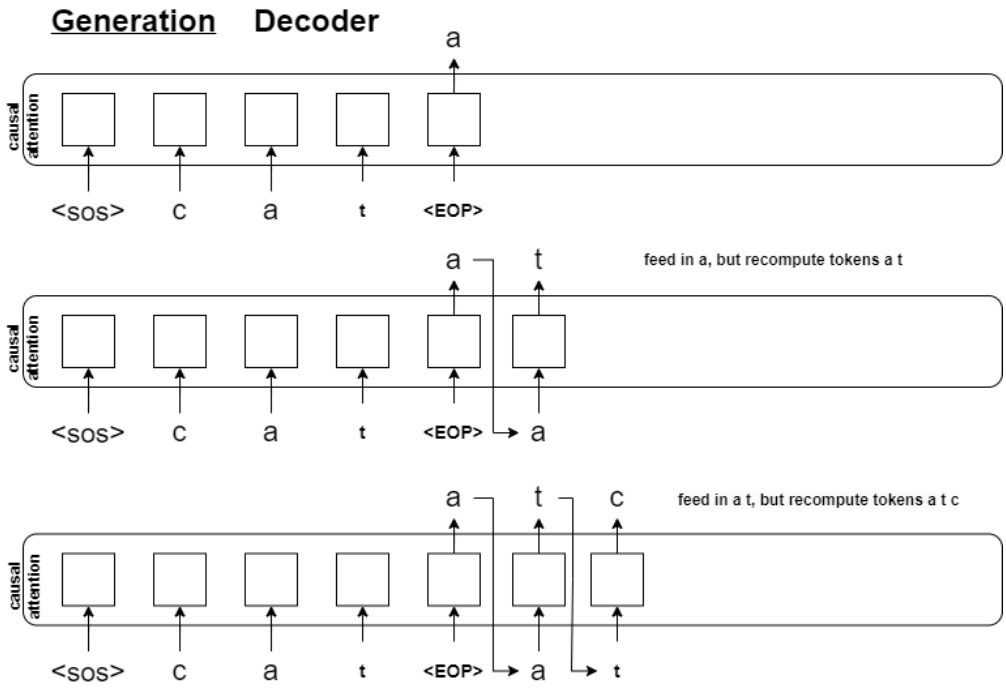

Figure 3: Translating a text using the decoder-only NMT model.

In order to answer the following questions correctly, please make sure that you have **run the code from `nmt.ipynb`, Part2, `Training and evaluation code` prior to answering the following questions.**

1. [4pt] *Construct the input tensors and the target tensors for training a decoder.* For this question, we ask you to implement the function `generate_tensors_for_training_decoder_nmt` that takes in an input sequence plus an end-of-prompt token and an output sequence plus an end-of-sentence token and returns two concatenated sequences. One has the form

   `<SOS> input sequence <EOP> output sequence`

   , as in the input to the decoder shown in Figure 2, and the other has the form

   `input sequence <EOP> output sequence <EOS>`

2. [4pt] *Implement the forward function in* `DecoderOnlyTransformer`.

3. [4pt] *Train the model.* Now, run the training and testing code block to see the generated translation using a decoder-only model. Comment on the pros and cons of the decoder-only approach. How is the quality of your generated results compared to the ones using the encoder-decoder model?

Deliverables

Create a section in your report called **Decoder Only NMT**. Add the following:

- Your answer to question 1. (Screenshots of your implementations) [4pt]
- Your answer to question 2. (Screenshots of your implementations) [4pt]
- Your written response to the question 3. [4pt]

2.3 Scaling Law and IsoFLOP Profiles [14pt]

This section will give you hands-on experience charting scaling law curves to forecast neural network performance. Scaling law is a fundamental concept that describes how the performance of a neural network changes with its size. Specifically, it relates the number of parameters or computations required by a neural network to achieve a certain level of performance, such as accuracy or loss. The scaling law provides a useful tool for predicting the performance of neural networks as they are scaled up or down.

IsoFLOP is a method proposed in the "Training Compute-Optimal Large Language Models" paper Hoffmann et al. [2022] to study the scaling law of large language models. The authors of the paper used IsoFLOP to study the effect of model size on the performance of large language models and to determine the optimal model size that maximizes performance for a given computational budget.

The motivation for using IsoFLOP to forecast neural network performance is twofold. Firstly, it provides a more accurate and efficient way to explore the scaling law of large language models than traditional methods, which involve training multiple models at different sizes. Secondly, IsoFLOP allows for a better understanding of the trade-off between model size and training cost, which is crucial for designing large-scale neural network architectures that are both efficient and effective. By

leveraging IsoFLOP, researchers can gain insights into the scaling properties of neural networks, such as their accuracy and computational efficiency, and optimize their performance for specific applications and computational resources.

In this question, we will plot the scaling law curve for the decoder-only translation models from the previous section. The notebook provided trains six translation models with different model sizes and varies the FLOP counts by training for different numbers of epochs. You are asked to complete the functions to make the final IsoFLOP curve consisting of models ranging from 0.08 TFLOPs to 1.28 TFLOPs.

1. [2pt] Train six decoder-only translation models using the code provided and plot the validation loss as the function of FLOPs. Comment on any interesting thing you observe. Does larger model always have a smaller validation loss? (Hint: See Question **??**)

2. [4pt] IsoFLOP Profiles. For a given FLOPs, fit a quadratic function to the validation loss and number of parameters in the log space. Find the optimal number of parameters using the quadratic function. Specifically, you need to fill the "find_optimal_params" function.

3. [4pt] Complete the Compute Optimal Model plot by fitting a linear line to the target FLOPs and the optimal model parameters. Based on the plot, estimate the optimal number of parameters when we have a compute budget of $1e15$.

4. [4pt] Plot Compute Optimal Token using the code provided. Now, given the Compute Optimal Model plot and Compute Optimal Token plot, is the training setup in Section 2.2.3 compute optimal? If not, how should we change it?

Deliverables

Create a section in your report called **Scaling Law and IsoFLOP Profiles**. Add the following:

- Your written response to the question 1. Your answer should not exceed 3 sentences. [2pt]
- Your answer to question 2. (Screenshots of your implementations) [4pt]
- Your answer to question 3. (Screenshots of your implementations). The optimal number of parameters given 1e15 FLOPs and the process of how you estimate it. [4pt]
- Your written response to the question 4. Your answer should not exceed 3 sentences. [4pt]

3 Fine-tuning Pretrained Language Models (LMs) [20pt]

The previous sections had you train models *from scratch*. However, similar to computer vision (CV), it is now very common in natural language processing (NLP) to *fine-tune* pretrained models. Indeed, this has been described as "NLP's ImageNet moment."[4] In this section, we will learn how to fine-tune pretrained *language models* (LMs) on a new task. We will use a simple classification task, where the goal is to determine whether a verbal numerical expression is *negative* (label 0), *zero* (label 1), or *positive* (label 2). For example, "eight minus ten" is negative, so our classifier should output label index 0. As our pretrained LM, we will use the popular BERT model, which uses a transformer encoder architecture. More specifically, we will explore two versions of BERT: **MathBERT** Shen et al. [2021], which has been pretrained on a large mathematical corpus ranging from pre-kindergarten to college graduate level mathematical content and **BERTweet** Nguyen et al. [2020], which has been pretrained on 100s of millions of tweets.

Most of the code is given to you in the notebook `https://colab.research.google.com/drive/12YA2bNMWwCaJqLYEtsUv3bVnrZ12x6eT?usp=sharing`. The starter code uses the *HuggingFace Transformers* library[5], which has more than 50k stars on GitHub due to its ease of use, and will be very useful for your NLP research or projects in the future. Your task is to adapt BERT so that it can be fine-tuned on our downstream task. Before starting this section, please carefully review the background for BERT and the verbal arithmetic dataset (below).

Background

BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers (BERT) Devlin et al. [2019] is a LM based on the Transformer Vaswani et al. [2017] encoder architecture that has been pretrained on a large dataset of unlabeled sentences from Wikipedia and BookCorpus Zhu et al. [2015]. Given a sequence of tokens, BERT outputs a "contextualized representation" vector for each token. Because BERT is pretrained on a large amount of text, these contextualized representations encode useful properties of the syntax and semantics of language.

BERT has 2 pretraining objectives: (1) Masked Language Modeling (MLM), and (2) Next Sentence Prediction (NSP). The input to the model is a sequence of tokens of the form:

```
[CLS] Sentence A [SEP] Sentence B
```

where `[CLS]` ("class") and `[SEP]` ("separator") are special tokens. In MLM, some percentage of the input tokens are randomly "masked" by replacing them with the `[MASK]` token, and the objective is to use the final layer representation for that masked token to predict the correct word that was masked out[6]. In NSP, the task is to use the contextualized representation of the `[CLS]` token to predict whether sentence A and sentence B are consecutive sentences in the unlabeled dataset. See Figure 4 for the conceptual picture of BERT pretraining and fine-tuning.

Once pretrained, we can fine-tune BERT on a downstream task of interest, such as sentiment analysis or question-answering, benefiting from its learned contextual representations. Typically, this is done by adding a simple classifier, which maps BERTs outputs to the class labels for our downstream task. Often, this classifier is a single linear layer + softmax. We can choose to train

[4] `https://ruder.io/nlp-imagenet/`

[5] `https://huggingface.co/docs/transformers`

[6] The actual training setup is slightly more complicated but conceptually similar. Notice, this is similar to one of the models in Programming Assignment 1!

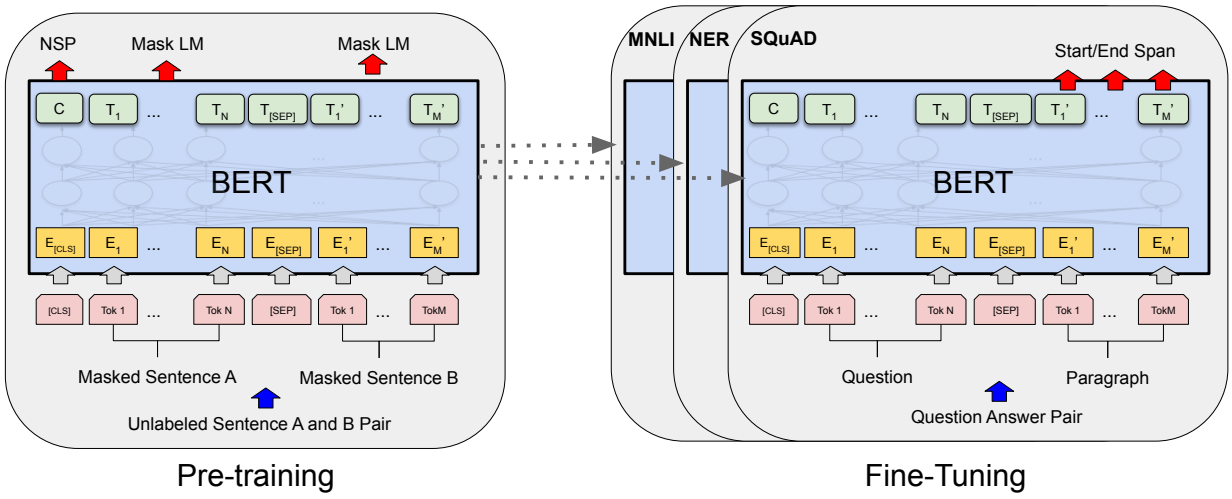

Figure 4: Overall pretraining and fine-tuning for BERT. Reproduced from BERT paper Devlin et al. [2019]

only the parameters of the classifier, or we can fine-tune both the classifier and BERT model jointly. Because BERT has been pretrained on a large amount of data, we can get good performance by fine-tuning for a few epochs with only a small amount of labelled data.

In this assignment, you will **fine-tune BERT** on a **single sentence classification task**. Figure 5 illustrates the basic setup for fine-tuning BERT on this task. We prepend the tokenized sentence with the `[CLS]` token, then feed the sequence into BERT. We then take the contextualized `[CLS]` token representation at the last layer of BERT as input to a simple classifier, which will learn to predict the probabilities for each of the possible output classes of our task. We will use the pretrained weights of MathBERT, which uses the same architecture as BERT, but has been pretrained on a large mathematical corpus, which more closely matches our task data (see below).

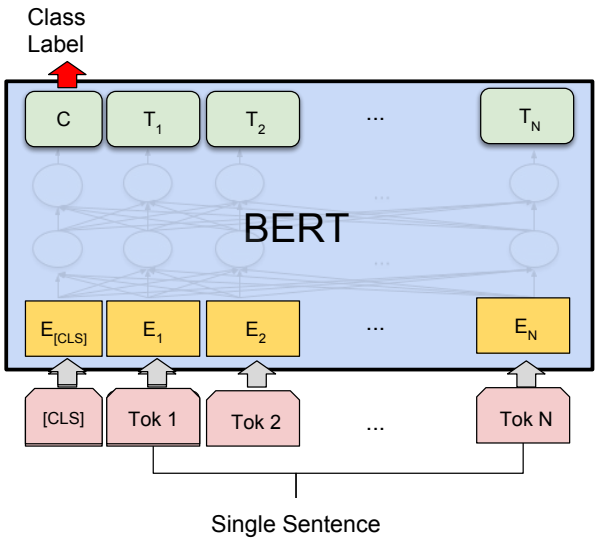

Figure 5: Fine-tuning BERT for single sentence classification by adding a layer on top of the contextualized `[CLS]` token representation. Reproduced from BERT paper Devlin et al. [2019]

Verbal Arithmetic Dataset

The verbal arithmetic dataset contains pairs of input sentences and labels. The input sentences express a simple addition or subtraction. Each input is labelled as 0, 1, or 2 if it evaluates to *negative*, *zero*, or *positive*, respectively. There are 640 examples in the train set and 160 in the test set. All inputs have only **three tokens** similar to the examples shown below:

Input expression	Label	Label meaning
four minus ten	0	"negative"
eighteen minus eighteen	1	"zero"
four plus seven	2	"positive"

Questions:

1. [10pt] *Add a classifier to BERT.* Open the notebook `https://colab.research.google.com/drive/1fGlrDODomFdZ8r_uzSSuH-57jvUaHB9x?usp=sharing` and complete Question 1 by filling in the missing lines of code in `BertForSentenceClassification`.

2. [0pt] *Fine-tune BERT.* Open the notebook and run the cells under Question 2 to fine-tune the BERT model on the verbal arithmetic dataset. If question 1 was completed correctly, the model should train, and a plot of train loss and validation accuracy will be displayed.

3. [5pt] *Freezing the pretrained weights.* Open the notebook and run the cells under Question 3 to fine-tune only the classifiers weights, leaving BERTs weights frozen. After training, answer the following questions (no more than **four** sentences total)

   - Compared to fine-tuning (see Question 2), what is the effect on train time when BERTs weights are frozen? Why? (1-2 sentences)
   - Compared to fine-tuning (see Question 2), what is the effect on performance (i.e. validation accuracy) when BERTs weights are frozen? Why? (1-2 sentences)

4. [5pt] *Effect of pretraining data.* Open the notebook and run the cells under Question 4 in order to repeat the fine-tuning process using the pretrained weights of BERTweet. After training, answer the following questions (no more than **three** sentences total).

   - Compared to fine-tuning BERT with the pretrained weights from MathBERT (see Question 2), what is the effect on performance (i.e. validation accuracy) when we fine-tune BERT with the pretrained weights from BERTweet? Why might this be the case? (2-3 sentences)

5. [0pt] *Inspect models predictions.* Open the notebook and run the cells under Question 5. We have provided a function that allows you to inspect a models predictions for a given input. Can you find examples where one model clearly outperforms the others? Can you find examples where all models perform poorly?

Deliverables:

- The completed `BertForSentenceClassification`. Either the code or a screenshot of the code. Make sure both the `__init__` and `forward` methods are clearly visible. [10pt]
- Answer to question 3. Your answer should not exceed **4 sentences**. [5pt]
- Answer to question 4. Your answer should not exceed **3 sentences**. [5pt]

4 Connecting Text and Images with CLIP [10pt]

Throughout this course, we have seen powerful image models and expressive language models. In this section, we will connect the two modalities by exploring CLIP, a model trained to predict an image's caption to learn better image representations.

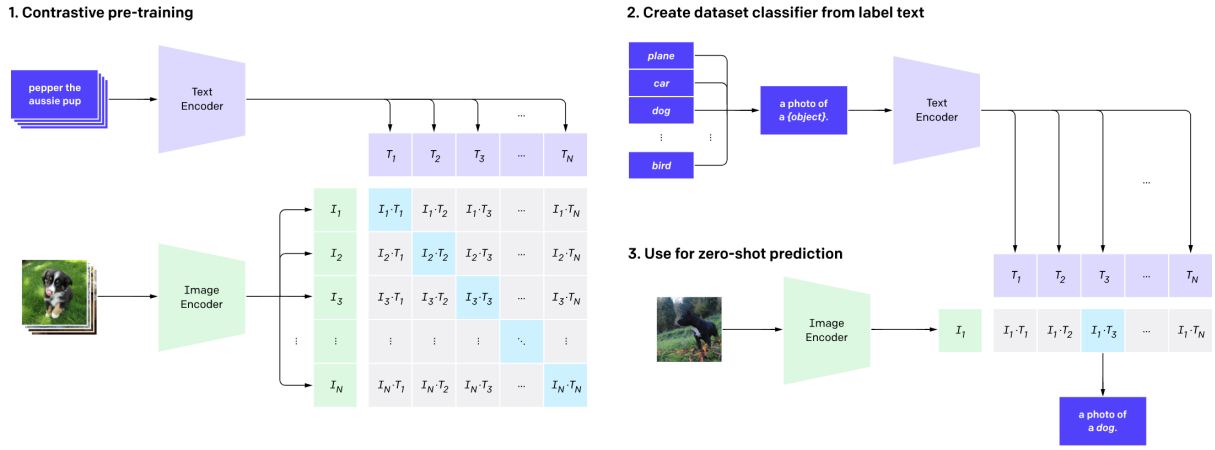

Figure 6: 1. Contrastive pre-training task that predicts the caption that corresponds to an image out of many possible captions. 2. At test time, each class is converted to a caption. This is used with 3. as a zero-shot classifier for a new image that predicts the best (image, caption) pair. Figure taken from [Radford et al., 2021a]

Background for CLIP:

The motivation behind **C**ontrastive **L**anguage-**I**mage **P**re-training (CLIP) [Radford et al., 2021b] was to leverage information from natural language to improve zero-shot classification of images. The model is pre-trained on 400 million (image, caption) pairs collected from the internet on the following task: given the image, predict which caption was paired with it out of 32,768 randomly sampled captions (Figure 6). This is done by first computing the feature embedding of the image and feature embeddings of possible captions. The cosine similarity of the embeddings is computed and converted into a probability distribution. The outcome is that the network learns many visual concepts and associates them with a name.

At test time, the model is turned into a zero-shot classifier: all possible classes are converted to a caption such as "a photo of a (class)" and CLIP estimates the best (image, caption) pair for a new image. Overall, CLIP offers many significant advantages: it does not require expensive hand-labelling while achieving competitive results and offers greater flexibility and generalizability over existing ImageNet models.

Questions:

1. [0pt] *Interacting with CLIP.* Open the notebook `https://colab.research.google.com/drive/1OOMGiyjGN6wELBhTnMo72aXYA5zLVPoJ?usp=sharing`. Read through Section I and run the code cells to get familiar with CLIP.

2. [10pt] *Prompting CLIP*. Complete Section II. Come up with a caption that will "prompt" CLIP to select the following target image:

Figure 7: Image that should be selected by CLIP.

Comment on the process of finding the caption: was it easy, or were there any difficulties? (no more than **one** sentence)

Deliverables:

- The caption you wrote that causes CLIP to select the image in Figure 7, as well as a brief (1 sentence) comment on the search process. [10pt]

What you need to submit

- The completed notebook files: `nmt.ipynb, bert.ipynb, clip.ipynb`.
- A PDF document titled `a3-writeup.pdf` containing your answers to the conceptual questions. You may directly append the PDF exports of the notebooks into the final `a3-writeup.pdf`.

References

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George E Dahl, Christopher J Shallue, and Roger Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *arXiv preprint arXiv:1907.04164*, 2019.

Christopher J Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E Dahl. Measuring the effects of data parallelism on neural network training. *arXiv preprint arXiv:1811.03600*, 2018.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Jia Tracy Shen, Michiharu Yamashita, Ethan Prihar, Neil Heffernan, Xintao Wu, Ben Graff, and Dongwon Lee. Mathbert: A pre-trained language model for general nlp tasks in mathematics education. *arXiv preprint arXiv:2106.07340*, 2021.

Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. Bertweet: A pre-trained language model for english tweets. *arXiv preprint arXiv:2005.10200*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://www.aclweb.org/anthology/N19-1423`.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

Alec Radford, Ilya Sutskever, Jong Wook Kim, Gretchen Krueger, and Sandhini Agarwal. Clip: Connecting text and images, Jan 2021a. URL `https://openai.com/blog/clip/`.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021b.