

HOMework 7: DEEP LEARNING

10-301/10-601 Introduction to Machine Learning (Spring 2024)

<http://www.cs.cmu.edu/~mgormley/courses/10601/>

OUT: Thursday, March 28th

DUE: Monday, April 8th

TAs: Sebastian, Bhargav, Rohan, Kevin, Varsha, Haohui, Neural the Narwhal

Summary In this assignment you will implement an RNN and performance evaluation. You will begin by going through some conceptual questions about CNNs, RNNs, and transformers for intuition for deep learning models and then use that intuition to build your own models.

START HERE: Instructions

- **Collaboration Policy:** Please read the collaboration policy here: <http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html>
- **Late Submission Policy:** See the late submission policy here: <http://www.cs.cmu.edu/~mgormley/courses/10601/syllabus.html>
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
 - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in \LaTeX . Each derivation/proof should be completed in the boxes provided. You are responsible for ensuring that your submission contains exactly the same number of pages and the same alignment as our PDF template. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader and there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score).
 - **Programming:** You will submit your code for programming questions on the homework to [Gradescope](#). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). You are only permitted to use [the Python Standard Library modules](#) and `numpy`. Ensure that the version number of your programming language environment (i.e. Python 3.9.12) and versions of permitted libraries (i.e. `numpy` 1.23.0) match those used on Gradescope. You have 10 free Gradescope programming submissions, after which you will begin to lose points from your total programming score. We recommend debugging your implementation on your local machine (or the Linux servers) and making sure your code is running correctly first before submitting your code to Gradescope.
- **Materials:** The data and reference output that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Instructions for Specific Problem Types

For “Select One” questions, please fill in the appropriate bubble completely:

Select One: Who taught this course?

- ☒ Matt Gormley
- ☐ Marie Curie
- ☐ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who taught this course?

- ☒ Henry Chai
- ☐ Marie Curie
- ☒ Noam Chomsky

For “Select all that apply” questions, please fill in all appropriate squares completely:

Select all that apply: Which are instructors for this course?

- ☒ Matt Gormley
- ☒ Henry Chai
- ☒ Hoda Heidari
- ☐ I don't know

Again, if you need to change your answer, you may cross out the previous answer(s) and bubble in the new answer(s):

Select all that apply: Which are the instructors for this course?

- ☒ Matt Gormley
- ☒ Henry Chai
- ☒ Hoda Heidari
- ☒ I don't know

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

10-601

10-~~6~~301

Written Questions (48 points)

1 L^AT_EX Bonus Point and Template Alignment (1 points)

1. (1 point) **Select one:** Did you use L^AT_EX for the entire written portion of this homework?

☐ Yes

☐ No

2. (0 points) **Select one:** I have ensured that my final submission is aligned with the original template given to me in the handout file and that I haven't deleted or resized any items or made any other modifications which will result in a misaligned template. I understand that incorrectly responding yes to this question will result in a penalty equivalent to 2% of the points on this assignment.

Note: Failing to answer this question will not exempt you from the 2% misalignment penalty.

☐ Yes

2 Convolutional Neural Network (14 points)

1. In this problem, consider a convolutional layer from a standard implementation of a CNN as described in lecture, without any bias term.

$X =$

1	0	-2	3	4	1
2	9	5	6	0	-1
0	-3	1	3	4	4
6	5	2	0	6	8
-5	4	-3	1	3	-2
4	1	2	8	9	7

$F =$

-1	-1	-1
-1	8	-1
-1	-1	-1

$Y =$

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

- (a) (1 point) Let an image X (6×6) be convolved with a filter F (3×3) using no padding and a stride of 1 to produce an output Y (4×4). What is value of j in the output Y ?

Your Answer

- (b) (1 point) Suppose you instead had an input feature map (or image) of size 6×4 (height \times width) and a filter of size 2×2 , using no padding and a stride of 2, what would be the resulting output size? Write your answer in the format: height \times width.

Your Answer

2. Parameter sharing is a very important concept for CNN because it drastically reduces the complexity of the learning problem and consequently that of the model required to tackle it. The following questions will deal with parameter sharing. Assume that there is no bias term in our convolutional layer.

(a) (1 point) **Select all that apply:** Which of the following are parameters of a convolutional layer?

- ☐ Stride size
- ☐ Padding size
- ☐ Input size
- ☐ Filter size
- ☐ Weights in the filter
- ☐ None of the above

(b) (1 point) **Select all that apply:** Which of the following are hyperparameters of a convolutional layer?

- ☐ Stride size
- ☐ Padding size
- ☐ Input size
- ☐ Filter size
- ☐ Weights in the filter
- ☐ None of the above

(c) (1 point) Suppose for the convolutional layer, we are given grayscale images of size 22×22 . Using one single 4×4 filter with a stride of 2, no padding and a single output channel, what is the **number of parameters** you are learning in this layer?

Your Answer

(d) (1 point) Now suppose we do not do parameter sharing. That is, each output pixel of this layer is computed by a separate 4×4 filter. Again we use a stride of 2, no padding and a single output channel. What is the **number of parameters** you are learning in this layer?

Your Answer

- (e) (1 point) Now suppose you are given a 40×40 colored image, which consists of 3 channels, each representing the intensity of one primary color (so your input is a $40 \times 40 \times 3$ tensor). Once again, you attempt to produce an output map without parameter sharing, using a unique 4×4 filter per output pixel, with a stride of 2, no padding and a single output channel (so the number of channels in the filter are the same as the number of channels in the input image). What is the number of parameters you are learning in this layer?

Your Answer

- (f) (1 point) In *one concise sentence*, describe a reason why parameter sharing is a good idea for a convolutional layer applied to image data, besides the reduction in number of learned parameters.

Your Answer

3. Neural the Narwhal was expecting to implement a CNN for Homework 5, but he is disappointed that he only got to write a simple fully-connected neural network.

- (a) (2 points) Neural decides to implement a CNN himself and comes up with the following naive implementation:

```
# image X has shape (H_in, W_in), and filter F has shape (K, K)
# the output Y has shape (H_out, W_out)
Y = np.zeros((H_out, W_out))
for r in range(H_out):
    for c in range(W_out):
        for i in range(K):
            for j in range(K):
                Y[r, c] += X[____blank____] * F[i, j]
```

What should be in the *blank* above so that the output Y is correct? Assume that H_{out} and W_{out} are pre-computed correctly, the filter has a stride of 1 and there's no padding.

Your Answer

- (b) (2 points) Neural now wants to implement the backpropagation part of the network but is stuck. He decides to go to office hours to ask for help. One TA tells him that a CNN can actually be implemented using matrix multiplication. He receives the following 1D convolution example:

Suppose you have an input vector $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]^T$ and a 1D convolution filter $\mathbf{w} = [w_1, w_2, w_3]^T$. Then if the output is $\mathbf{y} = [y_1, y_2, y_3]^T$, $y_1 = w_1x_1 + w_2x_2 + w_3x_3$, $y_2 = \dots$, $y_3 = \dots$. If you look at this closely, this is equivalent to

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \mathbf{A} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

where the matrix \mathbf{A} is given as \dots

What is matrix \mathbf{A} for this \mathbf{x} , \mathbf{y} and \mathbf{w} ? Write only the final answer. Your work will *not* be graded.

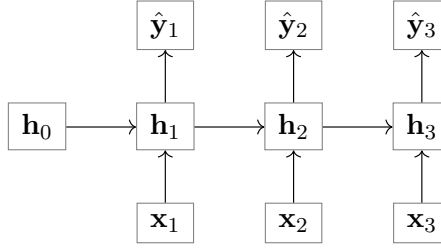
Your Answer

- (c) (2 points) Neural wonders why the TA told him about matrix multiplication when he wanted to write the backpropagation part. Then he notices that the gradient is extremely simple with this version of CNN. Explain in *one concise sentence (or one short mathematical expression)* how you can compute $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ once you obtain \mathbf{A} for some *arbitrary* input \mathbf{x} , filter \mathbf{w} , and the corresponding 1D convolution output \mathbf{y} (so \mathbf{A} is obtained following the same procedure as in part (b), but \mathbf{x} , \mathbf{y} and \mathbf{w} can be different from the example). Write only the final answer. Your work will *not* be graded.

Your Answer

3 Recurrent Neural Network (15 points)

1. Consider the following simple RNN architecture:



where we have inputs \mathbf{x}_t , hidden states \mathbf{h}_t , and outputs $\hat{\mathbf{y}}_t$ for each timestep t . The dimensions of these and the weights of the model are given below. On the right, we show the computation, performed by the RNN to obtain the outputs $\hat{\mathbf{y}}_t$ and subsequently the loss J for a single input $\mathbf{x}_{1:3}$.

$$\begin{array}{ll} \mathbf{x}_t \in \mathbb{R}^3 & \mathbf{W}_{hx} \in \mathbb{R}^{4 \times 3} \\ \mathbf{h}_t \in \mathbb{R}^4 & \mathbf{W}_{hy} \in \mathbb{R}^{2 \times 4} \\ \mathbf{y}_t, \hat{\mathbf{y}}_t \in \mathbb{R}^2 & \mathbf{W}_{hh} \in \mathbb{R}^{4 \times 4} \end{array}$$

$$\mathbf{z}_t = \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t$$

$$\mathbf{h}_t = \psi(\mathbf{z}_t)$$

$$\mathbf{o}_t = \mathbf{W}_{hy}\mathbf{h}_t$$

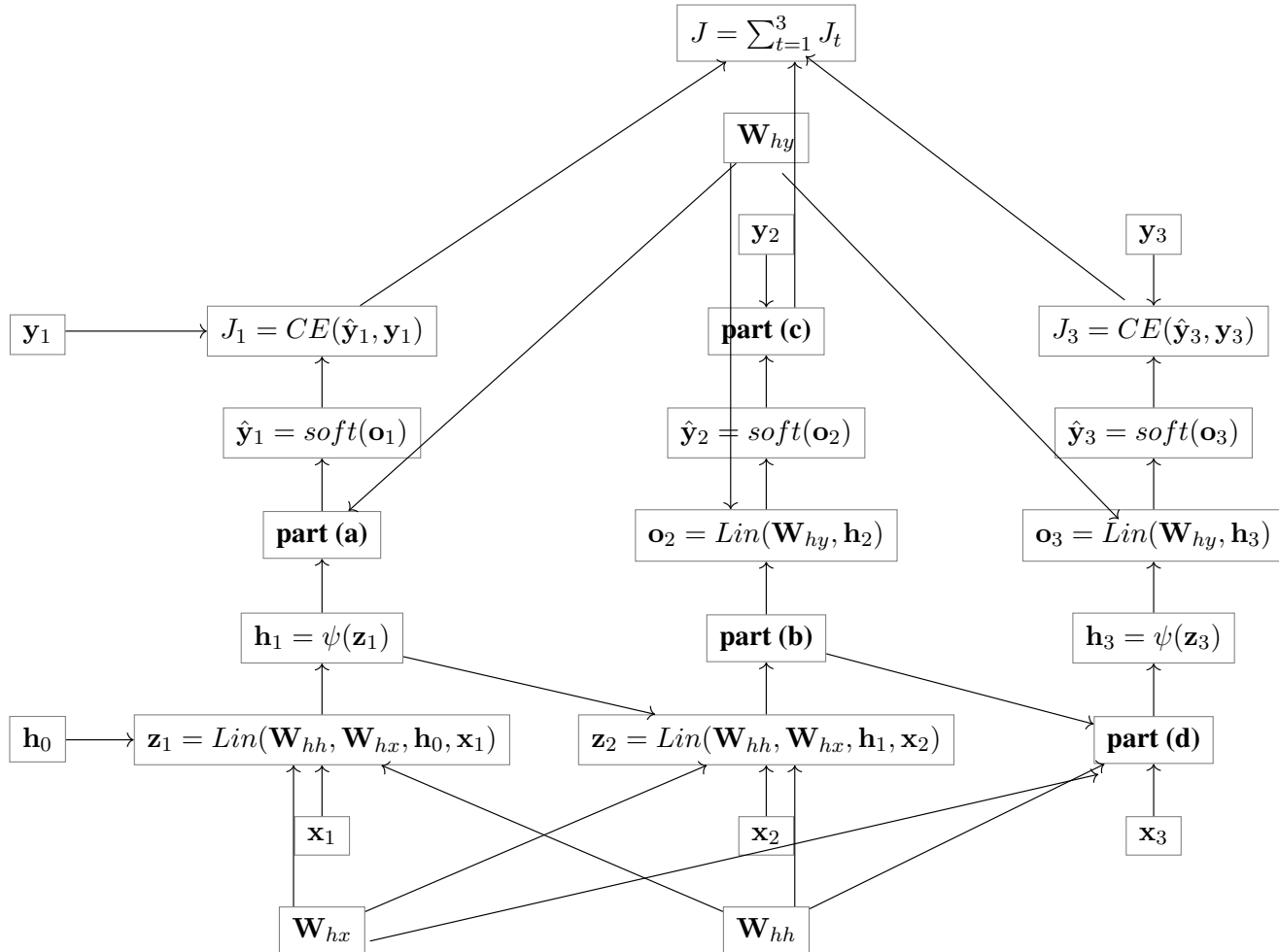
$$\hat{\mathbf{y}}_t = \text{soft}(\mathbf{o}_t)$$

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

$$J = \sum_{t=1}^3 J_t$$

Above \mathbf{y}_t is a one-hot vector representing the label for the t th timestep, *soft* is the **softmax** activation, ψ is the **identity** activation (i.e. no activation), J is the cross entropy loss computed by the function $CE()$. Note here that we assume that we have no intercept term.

- (a) (4 points) You will now construct the unrolled computational graph for the given model. Use input sequence \mathbf{x} , label \mathbf{y} , and the RNN equations presented above to complete the graph by filling in the solution boxes for the corresponding blanks.



(a)	(b)
(c)	(d)

- (b) Now you will derive the steps of the backpropagation algorithm that lead to the computation of $\frac{dJ}{d\mathbf{W}_{hh}}$. For all parts of this question, please write your answer in terms of $\mathbf{W}_{hh}, \mathbf{W}_{hy}, \mathbf{y}, \hat{\mathbf{y}}, \mathbf{h}$, 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).

- i. (2 points) What is $g_{J_t} = \frac{\partial J}{\partial J_t}$? Write your solution in the first box, and show your work in the second.

$$\frac{\partial J}{\partial J_t}$$

Work

- ii. (2 points) What is $g_{\mathbf{o}_t} = \frac{\partial J}{\partial \mathbf{o}_t}$ for an arbitrary $t \in [1, 3]$? Write your solution in the first box, and show your work in the second. Write your answer in terms of $\hat{\mathbf{y}}_t$, \mathbf{y}_t , and g_{J_t} . (Hint: Think about how you can write J_t in terms of \mathbf{o}_t , then use the chain rule. You may want to use a result from homework 5 to help here.)

$$\frac{\partial J}{\partial \mathbf{o}_t}$$

Work

- iii. (2 points) What is $g_{\mathbf{h}_i} = \frac{\partial J}{\partial \mathbf{h}_i}$ for an arbitrary $i \in [1, 3]$? Write your solution in terms of $\mathbf{g}_{\mathbf{o}_t}$, \mathbf{W}_{hh} , \mathbf{W}_{hy} in the first box, and show your work in the second. (Hint: Find $\frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_i}$, then use the chain rule. Also, for a given i , think about which \mathbf{o}_t 's \mathbf{h}_i affects)

$\frac{\partial J}{\partial \mathbf{h}_i}$

Work

- iv. (3 points) What is $g_{\mathbf{W}_{hh}} = \frac{\partial J}{\partial \mathbf{W}_{hh}}$? Write your solution in terms of $\mathbf{g}_{\mathbf{h}_i}$ and \mathbf{h} in the first box, and show your work in the second. (Hint: \mathbf{W}_{hh} is in every timestep, so you need to consider that in the derivative.)

$\frac{\partial J}{\partial \mathbf{W}_{hh}}$

Work

2. (2 points) **Select all that apply:** Which of the following are true about RNN and RNN-LM?

- ☐ An RNN cannot process sequential data, whereas an RNN-LM is designed for sequential data processing such as in natural language processing.
- ☐ An RNN-LM is only exclusively used as an encoder, which can process sequential data and encode it into a fixed-size state vector.
- ☐ An RNN-LM includes additional layers and structures specifically designed to predict the next token in a sequence, making it more suited for tasks like text generation than a standard RNN.
- ☐ The RNN-LM is trained to maximize the probability of a sequence of tokens, given a previous sequence, which is not a typical training objective of a standard RNN.
- ☐ None of the above.

4 Transformers and AutoDiff (5 points)

```
1 global tape = stack()
2
3 class Module:
4
5     method init()
6         out_tensor = null
7         out_gradient = 1
8
9     method apply_fwd(List in_modules)
10        in_tensors = [x.out_tensor for x in in_modules]
11        out_tensor = forward(in_tensors)
12        tape.push(self)
13        return self
14
15    method apply_bwd():
16        in_gradients = backward(in_tensors, out_tensor, out_gradient)
17        for i in 1,..., len(in_modules):
18            in_modules[i].out_gradient += in_gradients[i]
19        return self
20
21 function tape_bwd():
22     while len(tape) > 0
23         m = tape.pop()
24         m.apply_bwd()
```

1. (1 point) **Select one:** This is a code snippet from lecture 18 slide 16. In the context of the method `apply_fwd()` inside the `Module` class, what is the primary role of the `tape.push(self)` command that pushes the module onto the tape?

- ☐ It records the current module onto the stack along with its parameters and tensors to ensure that the output tensor is saved for the backward pass.
- ☐ It pushes the current computation's gradient onto the stack for immediate use in the forward pass.
- ☐ It duplicates the module to allow for parallel computations in subsequent layers of the neural network.
- ☐ It activates the module for the forward pass, making it the only active computation in the network.

2. (2 points) **True or False:** We can replace a stack with a queue in Module-based AutoDiff. Explain your reasoning in no more than 2 sentences in the box below.

- ☐ True
- ☐ False

Your Answer

3. Consider a Transformer model employing a multi-headed self-attention mechanism. Suppose the input consists of a sequence of T tokens, each token represented by a d_{model} -dimensional embedding vector. This model utilizes H attention heads. During the attention process, each head generates keys, queries, and values from the input embeddings. The dimensionality of the key and query vectors is d_k for each head, and the attention function produces an output vector of dimension d_v for each token and head.

- (a) (1 point) Which of the following represents the dimension of the key tensor for a single attention head?

- ☐ $T \times d_v$
- ☐ $H \times d_k \times d_{\text{model}}$
- ☐ $T \times d_k$
- ☐ $T \times d_{\text{model}} \times d_k$

- (b) (1 point) Which of the following represents the dimension of the output tensor of the multi-headed attention before any final linear transformation?

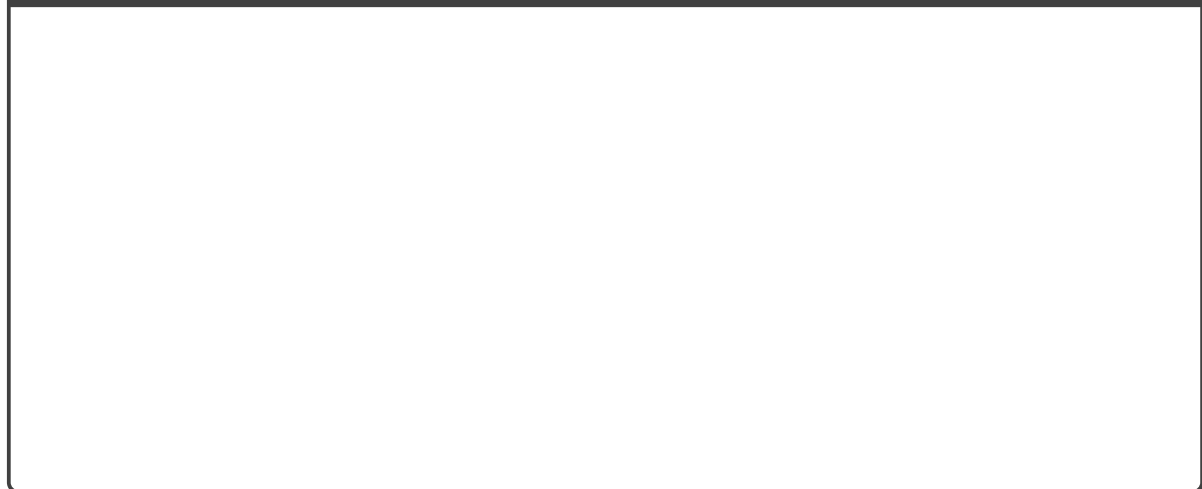
- ☐ $T \times H \times d_k$
- ☐ $T \times H \times d_v$
- ☐ $T \times d_{\text{model}}$
- ☐ $H \times d_k \times d_v$

5 Empirical Questions (13 points)

The following questions should be completed as you work through the programming component of this assignment. **Please ensure that all plots are computer-generated.** For all the questions below, unless otherwise specified, set `embedding_dim` and `hidden_dim` to be 128. Run on the `en.{train/val}_40.twocol.oov` files in the handout.

1. (4 points) Create a single plot for this question after running for 15 epochs. The y -axis should show the F1 score (as a decimal) and the x -axis should show the number of epochs. The graph should have 4 total lines showing the train and validation F1 scores of two settings: using `ReLU` activation and using `Tanh` activation.

Your Answer



2. (4 points) Create a single plot for this question after running for 5 epochs. The y -axis should show the F1 score (as a decimal) and the x -axis should show the number of epochs. The graph should have 6 total lines showing the train and validation F1 scores of three settings: equal embedding and hidden dimensions of 64, 128 and 512.

Your Answer



3. (5 points) In a maximum of 5 sentences, explain the results of the above experiments. In particular, do the training and validation F1 curves look the same, or different? How do the hyperparameters affect performance?

Your Answer

6 Collaboration Questions

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found [here](#).

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details.
3. Did you find or come across code that implements any part of this assignment? If so, include full details.

Your Answer

7 Programming (55 points)

In this section, you will implement a Recurrent Neural Network using only PyTorch primitives. You will write a custom DataLoader to read in the text data. You will even implement a custom activation layer functions without using built-in PyTorch functions like `nn`.

7.1 The Task

Named entity recognition (NER) is the task of classifying named entities, typically proper nouns, into pre-defined categories, such as person, location, or organization. Consider the example sequence below, where each word is appended with a tab and then its tag:

` `	O
Rhinestone	B-ORG
Cowboy	I-ORG
' '	O
(O
Larry	B-PER
Weiss	I-PER
)	O
-	O
3:15	O

Rhinestone and Cowboy are labeled as an organization (ORG), while Larry and Weiss is labeled as a person (PER). Words that are not named entities are assigned the O tag. The B- prefix indicates that a word is the beginning of an entity, while the I- prefix indicates that the word is inside the entity.

7.2 The Dataset

[CoNLL 2003](#) is a dataset comprised of various Reuters news stories between 1996 and 1997. It is designed for English and German Named Entity Recognition (NER) and Token Classification tasks. The CoNLL dataset provides labelled entity data in English with approximately 15K training examples, 3.5K validation examples, and 3.6K test examples. The German dataset contains 12.7K training examples, 3K validation samples, and 3.2K test examples. In this assignment we will strictly be using the English dataset.

7.3 File Formats

The contents and formatting of each of the files in the handout folder is explained below.

1. **en.train.twocol.oov** This file contains labeled text data that you will use in training your model. Specifically, the text contains one word per line that has already been preprocessed, cleaned and tokenized. Words that are Out of Vocabulary (OOV) in the original dataset are replaced with the special token `*OOV*`. Every sequence has the following format:

```
<Word0>\t<Tag0>\n<Word1>\t<Tag1>\n ... <WordN>\t<TagN>\n
```

where every `<WordK>\t<TagK>` unit token is separated by a newline. Between each sequence is an empty line. If we have two, three-word sequences in our data set, the data will look like so:

```
<Word0>\t<Tag0>\n
<Word1>\t<Tag1>\n
<Word2>\t<Tag2>\n
\n
<Word0>\t<Tag0>\n
<Word1>\t<Tag1>\n
```

```
<Word2>\t<Tag2>\n\n
```

Note: Word 2 of the second sequence does end with two newlines because it is the end of the data set.

2. **en.val.twocol.oov:** This file contains labeled validation data that you will use to evaluate your model. This file has the same format as **train.txt**.

7.4 Required Reading: PyTorch Tutorial

Before proceeding any further, you must complete the PyTorch Tutorial. Please read the full collection of the Introduction to PyTorch, i.e. Learn the Basics || Quickstart || Tensors || Datasets & DataLoaders || Transforms || Build Model || Autograd || Optimization || Save & Load Model.

<https://pytorch.org/tutorials/beginner/basics/intro.html>

7.5 Custom Dataset and DataLoader

One essential step when using PyTorch for any problem is to create a custom Dataset class for your model's DataLoader. A DataLoader is a class that PyTorch uses to supply your model with the data it needs for training. Typically, your model doesn't update parameters using the entire dataset at once, it uses batches of the data (though for this model we will be using a batch size of 1). The DataLoader is responsible for creating these batches under the hood, for the training functions you write later on to use. Each DataLoader relies on a custom Dataset class you must write.

Writing a custom Dataset class is one of the most essential steps you must perform when working on a deep learning problem. This class is what PyTorch uses to load your dataset into memory, in a format that will work well for your neural network. Neural networks require each data point to have numerical features, so if your data is categorical, or you are working with text data, you must convert these into numbers in your Dataset class. Each dataset is unique in terms of the types of preprocessing that are required, which is why you must typically write your own custom Dataset class for your problem. For more information on Datasets and DataLoaders, and to see an example of these two in action, please see the following [link](#).

You always want to make sure the training and test data are separate. To do this, it is sometimes common to see a custom TestDataset class as well. In our case, this will not be necessary, and we will simply be creating a separate Dataset object for the training and test data. Using separate objects ensures that the DataLoader doesn't using testing data to train the model, and vice versa.

In PyTorch, all custom Dataset classes inherit from the `torch.utils.data.Dataset` parent class and you must implement the following three inherited methods. These functions are what the DataLoader will call when it is creating the batches of data that the model will train on or evaluate.

1. `__init__(self, ...)`

This is the initialization function. This is where you pass in important information to the Dataset as arguments, like the name of the file or directory where the data you will to process is stored. For our text dataset, which is pretty small, we will use this function to load the input text file (**train.txt** or **validation.txt**) into memory by creating data structures that can be easily indexed into. This is also where we recommend converting text into numerical data.

2. `__len__(self)`

You must return the length, or size of dataset. In other words, how many examples are there in total.

3. `__getitem__(self, index)`

This function is used to get a single item from the dataset. You pass in the index of this item/datapoint as an argument, and the expected output is two `torch.Tensors`: one for the inputs/x values, and

one for the true outputs/ y values. In our case, we are dealing with sequences, so your function should return one `torch.Tensor` containing a sequences of words (as ints), and another `torch.Tensor` containing the corresponding sequence of tags (as ints). You will likely be working with lists or numpy arrays, so we recommend using the `torch.tensor()` [method](#) to perform conversions.

Custom TextDataset: For our problem, we are asking you to create a custom Dataset class called a TextDataset to process the text files **en.train.twocol.oov** and **en.val.twocol.oov**. As mentioned above, the input text files consist of several sequences of (word, tag) pairs. Within each pair, the word and tag are separated by a `\t` character. Within each sequence, the (word, tag) pairs are separated by `\n` characters. Within the overall text file, the different sequences are separated by an additional `\n` character. You must process these text files so that, as mentioned above, when the `getitem` function is called to retrieve item i , it returns the i th sequences of words (converted to ints) and the i th sequences of tags (also converted to ints).

For example, if the DataLoader calls `getitem` function for the fifth item, and this is the corresponding fifth sequence in the dataset:

```
<Word5>\t<Tag0>\n
<Word17>\t<Tag0>\n
<Word8>\t<Tag4>\n
<Word2>\t<Tag2>\n
<Word10>\t<Tag1>\n
\n
```

The TextDataset is expected to return the following Tensors:

```
torch.Tensor([5, 17, 8, 2, 10]), torch.Tensor([0, 0, 4, 2, 1])
```

Note: The left Tensor is for input data (words) and the right Tensor is for output data (tags).

Here are the important steps/TODOs to complete this TextDataset:

1. First, you must make sure to parse the text file to remove the white space characters like `\t` and `\n`. Functions like `strip()` and `split()` may be useful to you here. Using these functions correctly will make it easy to separate sequences from each other (hint: each sequence ends in the same combination of characters). This should be done in the `init` method.
2. Once you have determined how to remove the white space and separate sequences from each other, you must now process each individual sequence. You want to convert these sequences of (word, tag) pairs into two separate sequences of numbers: one sequence of numeric words and one sequence of numeric tags. To convert words or tags to numbers, one easy strategy is to treat each unique word or tag as an integer. You must therefore create a mapping of unique words to integers while parsing the text file, for both the words and tags. You should use these mappings to then convert the sequences of words and tags into sequences of ints, and store them in easily indexed data structures, like a numpy array or list. We recommend storing the words and tags in separate data structures, to make the `getitem` method easier. This should also be done in the `init` method.
3. One important consideration is that you need to make sure that the training TextDataset object and test TextDataset object use the same mappings. (You don't want your test TextDataset object to map the same word to a different int, since your model will then predict the wrong tag). To do this, we suggest initializing empty dictionaries outside the TextDataset class, and passing these dictionaries into the `init` function as additional arguments. The training TextDataset object will receive empty dictionaries and will fill them in while using them. Once filled in, the dictionaries will then be passed

to the test `TextDataset` object, which will just directly use the non-empty dictionaries it has received, without editing them.

(Hint: Because our datasets used an OOV token, as mentioned above, you don't need to worry about new words appearing in the test data that you've never seen in training. But in general it's good to check for this case.)

4. Additionally, make sure to create a dictionary of indices to tags. This will be important when evaluating the F1 score of your predictions. The evaluate function we provide needs to be given a list of ground truth tags and predicted tags. It cannot just use the ints outputted by your model, since it doesn't what tag each int corresponds to. Therefore, when creating your dictionary of tags to indices, simultaneously create an inverse mapping of indices to tags. You can do this by once again passing an empty dictionary into the `init` function and then filling it up as you parse the training data.
5. You must make sure to return the length of the dataset (the total number of sequences) in the `len` function. Either you can calculate this within the `len` function itself, or store it as a variable in the `init` function and return it here.
6. Finally, you must implement the `getitem` method. In this method, you should grab the `i`th sequences of words (converted to ints) and the `i`th sequences of tags (also converted to ints) and return them as `torch.Tensor` objects. Again, the `torch.tensor()` method will be useful here for converting to your final format.

Once the `TextDataset` is complete, you can create separate objects for the training and test data, and feed these `TextDatasets` into `DataLoaders`. Again, see the following [link](#) for an example. To ensure that all data is iterated in the same order and your results match the expected ones, we require using the `shuffle=False` option for the `DataLoaders`. Make sure to keep in mind that we are asking you to use a batch size of 1 for this problem, as batching with variable length sequences adds complexity.

7.6 Model Definition

In this assignment, you will create a sequence tagging model, that uses an RNN as the sequence model, with an embedding layer before the sequence layer. The RNN model you create should be able to support ReLU and Tanh activation functions. The RNN will be built as a composition of linear layers and activations, of which you will write your own functions, and leverage pytorch's autograd capabilities to perform backpropagation. For this assignment, you are not allowed to use `nn.RNN`, `nn.Linear`, `nn.ReLU`, nor `nn.Tanh`. We have given function stubs for each part, and will describe them in more detail below.

7.6.1 Activation Functions

You need to implement two activations: ReLU and Tanh. Each activation has two functions, which include `TanhFunction`, `Tanh` and `ReLUFunction`, `ReLU`. Using `Tanh` as an example, the `TanhFunction` class will extend pytorch's autograd functionality while the `Tanh` class acts as a wrapper for `TanhFunction`. The function specifications in the handout should guide you through the steps, but it is very similar to `LinearFunction` in that you will have to implement `forward` and `backward` for the activation function classes. In the activation module's `Tanh` function, use `.apply()` on the input for the `forward` method. NOTE: it is ok to use `torch.tanh` in the `tanhFunction`, but do not use `torch.nn.Tanh`. A backwards implementation is not necessary for this module function (Make sure you understand why). For additional discussion on the topic of extending PyTorch in this way, see [here](#).

7.6.2 Testing your Functions

In order to verify the correctness of the backward method of the three functions you implemented, you should use `torch.autograd.gradcheck`. See [here](#) for more information. More specifically, you can

use this function to test the correctness of `LienarFunction`, `ReLUFunction`, and `TanhFunction`. We strongly recommend testing all three of your functions before moving on to the next steps, because your bugs and errors will propagate and make debugging much harder. For office hours, we ask that you show us that you have tried using `gradcheck` to ensure these three functions work.

7.6.3 RNN

Now you will implement an RNN, using the existing linear layer and activation functions that you have created earlier. See below for implementation details.

The `init` method sets all the parameters for the model. This includes the `embedding_dim`, hyperparameters, Linear layers, and activations.

The `forward` pass takes in embeddings of size `embedding_dim`, and passes it through the RNN in order, returning a list of hidden states. The number of states should be equal to the length of the sequence. The first hidden state should be initialized to be all zeros and be updated with every word in the sequence. Again, make sure to keep in mind that the batch size is 1.

7.6.4 Tagging Model

Now we will put together all the parts to create a model. The tagging model will extend `pytorch`'s `module` class. The `init` method has 5 parameters.

1. `vocab_size`: Integer representing the number of unique words in the dictionary
2. `tagset_size`: Integer representing the number of unique outputs in the tag space
3. `embedding_dim`: Integer representing the size of the sentence embeddings (hyperparameter)
4. `hidden_dim`: Integer representing the size of the hidden state dimensions in the RNN (hyperparameter)
5. `activation`: String representing the activation function to be used in the RNN (hyperparameter)

The `init` method should initialize the embedding layer (Hint: Use `nn.Embedding`), the RNN class, as well as a final Linear layer for use on the outputs of the RNN. The `forward` method should return a distribution of the `tag_space` for each element in the sentence. **Please pay close attention to the shapes of the inputs and outputs as you do this.**

7.7 Training and Evaluation

Now that you have created a working model, it's time to train and evaluate it! For this section, you will write three functions: `train_one_epoch()`, `predict_and_evaluate()`, and `train()` (and optionally `calculate_metrics()`).

`calculate_metrics()`: This is an optional function you can choose to fill in that helps modularize the evaluation. Use your map of ints to tags, in addition to the `evaluate` function, to calculate the F1 score for your predictions. Please take a look at the documentation for the `evaluate` function in `metrics.py` to see what parameters it expects and what it will output. You only need to output the F1 score, but if you would like to keep track of precision and recall, you are welcome to. If you are unfamiliar with these terms, these are just different metrics to assess how well your model is doing, specifically when you don't have an equal balance of outputs in your data.

`train_one_epoch()`: Performs the necessary calls to model, optimizer, and loss function to train the model for one epoch, but does not return anything.

`predict_and_evaluate()`: Returns the loss, accuracy, `f1_score`, and predictions for the corresponding epoch. Please either call `calculate_metrics()` here or use the `evaluate` function here in addition to your map of indices to tags to calculate the F1 score.

`train()`: Returns the train loss/accuracy and `f1` score for each epoch. It should also return the final epoch's predictions on the training and test sets.

If you're unsure how to start, we highly recommend taking a look at [this tutorial](#) from the PyTorch official documentation. We also recommend taking a look at PyTorch's `reshape()` [method](#), `view()` [method](#), `squeeze()` [method](#), and `tolist()` [method](#) to help you out when dealing with the tensors your model outputs.

7.8 Command Line Arguments

The autograder runs and evaluates the output from the files generated, using the following command:

```
$ python3 rnn.py [args...]
```

Where `[args...]` is a placeholder for command-line arguments: `<train_input>` `<test_input>` `<train_out>` `<test_out>` `<metrics_out>`.

Additional hyper-parameters for the model utilize "double dashes". You should experiment with these arguments to improve the performance of the model in the empirical section. `<--activation>`
`<--embedding_dim>` `<--hidden_dim>` `<--num_epochs>`

These arguments are described below:

1. `<train_input>`: string path to the training input `.txt` file (see Section 7.2)
2. `<test_input>`: string path to the testing input `.txt` file (see Section 7.2)
3. `<train_out>` string path to output `.txt` file to which the prediction on the training data should be written
4. `<test_out>` string path to output `.txt` file to which the prediction on the training data should be written
5. `<metrics_out>` string path of the output `.txt` file to which metrics such as train and validation F1 score should be written
6. `<--activation>` string specifying activation layer to use in the model, either "tanh" or "relu" (**hyper-parameter**)
7. `<--embedding_dim>` positive integer specifying the size of the sentence embedding vector (**hyper-parameter**)
8. `<--hidden_dim>` positive integer specifying the number of hidden units to use in the model's hidden layer (**hyper-parameter**)

Below is an example command to run using Tanh activation.

```
python rnn.py data_conll03/en.train_10.twocol.oov \  
data_conll03/en.val_10.twocol.oov \  
train_out.txt val_out.txt metrics_out.txt \  
--activation="relu" --embedding_dim=50 --hidden_dim=50 \  
--num_epochs=10 \  

```

7.9 Output: Labels File

Your program should write two output `.txt` files containing the tag predictions of your model on training data (`<train_out>`) and validation data (`<validation_out>`). Each should contain the predicted labels for each example printed on a new line. Use `\n` to create a new line.

Your labels should exactly match those of a reference implementation – this will be checked by the autograder by running your program and evaluating your output file against the reference solution. An example of the labels is given below.

```
5
1
0
5
5
3
4
```

7.10 Output: Metrics File

Generate a file where you report the accuracy and F1 score for train and test validation on the final epoch. Round the values to 6 decimal places. Make sure to follow the newline format shown here.

Below is example of approximate metrics run on the 10% data for 10 epochs. We used the ReLU activation, embedding_dim and hidden_dim at 50.

```
accuracy(train): 0.98588
accuracy(test): 0.883971
f1(train): 0.909717
f1(test): 0.524194
```

7.11 Gradescope Submission

You should submit your `rnn.py` and `metrics.py`. **Any other files will be deleted.** Please do not use other file names. This will cause problems for the autograder to correctly detect and run your code. Please go through the appendix at the end for information on starter-code.

Some additional tips: Make sure to read the autograder output carefully. The autograder for Gradescope prints out some additional information about the tests that it ran. For this programming assignment we've specially designed some buggy implementations that you might implement and will try our best to detect those and give you some more useful feedback in Gradescope's autograder. Make wise use of autograder's output for debugging your code.

Note: For this assignment, you have 10 submissions to Gradescope before the deadline, but only your last submission will be graded.