RNN Review
& Hierarchical Attention Networks

SHANG GAO
Overview

- Review of Recurrent Neural Networks
- Advanced RNN Architectures
  - Long-Short-Term-Memory
  - Gated Recurrent Units
- RNNs for Natural Language Processing
  - Word Embeddings
  - NLP Applications
- Attention Mechanisms
- Hierarchical Attention Networks
Feedforward Neural Networks

In a regular feedforward network, each neuron takes in inputs from the neurons in the previous layer, and then pass its output to the neurons in the next layer.

The neurons at the end make a classification based only on the data from the current input.
Recurrent Neural Networks

In a recurrent neural network, each neuron takes in data from the previous layer AND its own output from the previous timestep.

The neurons at the end make a classification decision based on NOT ONLY the input at the current timestep BUT ALSO the input from all timesteps before it.

Recurrent neural networks can thus capture patterns over time (e.g. weather, stock market data, speech audio, natural language).
Recurrent Neural Networks

In the example below, the neuron at the first timestep takes in an input and generates an output.

The neuron at the second timestep takes in an input **AND ALSO** the output from the first timestep to make its decision.

The neuron at the third timestep takes in an input and also the output from the second timestep (**which accounted for data from the first timestep**), so its output is affected by data from both the first and second timestep.
Recurrent Neural Networks

Traditional Neuron:
output = sigmoid(weights * input + bias)

Recurrent Neuron:
output = sigmoid(weights1 * input + weights2 * previous_output + bias)
or
output = sigmoid(weights * concat(input, previous_output) + bias)
Toy RNN Example

Adding Binary

At each timestep, RNN takes in two values representing binary input.

At each timestep, RNN outputs the sum of the two binary values taking into account any carryover from previous timestep.
Problems with Basic RNNs

In a basic RNN, new data is written into each cell at every timestep.

Data from timesteps very early on get diluted because they are written over so many times.

In the example below, data from the first timestep is read into the RNN.

At each subsequent timestep, the RNN factors in data from the current timestep.

By the end of the RNN, the data from the first timestep has very little impact on the output of the RNN.
Problems with Basic RNNs

Basic RNN cells can’t retain information across a large number of timesteps.

Depending on the problem, RNNs can lose data in as few as 3-5 timesteps.

This is causes problems on tasks where information needs to be retained over a long time.

For example, in natural language processing, the meaning of a pronoun may depend on what was stated in a previous sentence.
Long Short Term Memory cells are advanced RNN cells that address the problem of long-term dependencies.

Instead of always writing to each cell at every time step, each unit has an internal ‘memory’ that can be written to selectively.
Long Short Term Memory

Input from the current timestep is written to the internal memory based on how relevant it is to the problem (relevance is learned during training through backpropagation)

If the input isn’t relevant, no data is written into the cell

This way data can be preserved over many timesteps and be retrieved when it is needed
Long Short Term Memory

Movement of data into and out of an LSTM cell is controlled by “gates”

The “forget gate” outputs a value between 0 (delete) and 1 (keep) and controls how much of the internal memory to keep from the previous timestep.

For example, at the end of a sentence, when a ‘.’ is encountered, we may want to reset the internal memory of the cell.

\[
    f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)
\]
Long Short Term Memory

The “candidate value” is the processed input value from the current timestep that may be added to memory

- Note that $\tanh$ activation is used for the “candidate value” to allow for negative values to subtract from memory

The “input gate” outputs a value between 0 (delete) and 1 (keep) and controls how much of the candidate value add to memory

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
Long Short Term Memory

Combined, the “input gate” and “candidate value” determine what new data gets written into memory.

The “forget gate” determines how much of the previous memory to retain.

The new memory of the LSTM cell is the “forget gate” $f_t$ * the previous memory state + the “input gate” $i_t$ * the “candidate value” from the current timestep.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$
Long Short Term Memory

The LSTM cell does not output the contents of its memory to the next layer

- Stored data in memory might not be relevant for current timestep, e.g., a cell can store a pronoun reference and only output when the pronoun appears

Instead, an “output” gate outputs a value between 0 and 1 that determines how much of the memory to output

The memory goes through a final $\text{tanh}$ activation before being passed to the next layer

$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \ast \text{tanh} (C_t)$$
Gated Recurrent Units are very similar to LSTMs but use two gates instead of three.

The “update gate” determines how much of the previous memory to keep.

The “reset gate” determines how to combine the new input with the previous memory.

The entire internal memory is output without an additional activation.

\[
\begin{align*}
    z_t &= \sigma \left( W_z \cdot [h_{t-1}, x_t] \right) \\
    r_t &= \sigma \left( W_r \cdot [h_{t-1}, x_t] \right) \\
    \tilde{h}_t &= \tanh \left( W \cdot [r_t \ast h_{t-1}, x_t] \right) \\
    h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
LSTMs vs GRUs

Greff, et al. (2015) compared LSTMs and GRUs and found they perform about the same.

Jozefowicz, et al. (2015) generated more than ten thousand variants of RNNs and determined that depending on the task, some may perform better than LSTMs.

GRUs train faster than LSTMs because they are less complex.

Generally speaking, tuning hyperparameters (e.g. number of units, size of weights) will probably affect performance more than picking between GRU and LSTM.
RNNs for Natural Language Processing

The natural input for a neural network is a vector of numeric values (e.g. pixel densities for imaging or audio frequency for speech recognition)

How do you feed language as input into a neural network?

The most basic solution is **one hot encoding**
- A long vector (equal to the length of your vocabulary) where each index represents one word in the vocabulary
- For each word, the index corresponding to that word is set to 1, and everything else is set to 0

$$V = \{\text{zebra, horse, school, summer}\}$$

\[
\begin{align*}
 v(\text{zebra}) &= [1, 0, 0, 0] \\
 v(\text{horse}) &= [0, 1, 0, 0] \\
 v(\text{school}) &= [0, 0, 1, 0] \\
 v(\text{summer}) &= [0, 0, 0, 1]
\end{align*}
\]
One Hot Encoding LSTM Example

Trained LSTM to predict the next character given a sequence of characters

Training corpus: All books in Hitchhiker’s Guide to the Galaxy series

One-hot encoding used to convert each character into a vector

72 possible characters – lowercase letters, uppercase letters, numbers, and punctuation

Input vector is fed into a layer of 256 LSTM nodes

LSTM output fed into a softmax layer that predicts the following character

The character with the highest softmax probability is chosen as the next character
Generated Samples

700 iterations: ae ae ae ae ae ae ae ae ae ae ae ae ae ae ae ae ae ae ae

4200 iterations: the sand and the said the sand and the said the sand and the said the sand and the said the sand and the said the sand

36000 iterations: seared to be a little was a small beach of the ship was a small beach of the ship was a small beach of the ship

100000 iterations: the second the stars is the stars to the stars in the stars that he had been so the ship had been so the ship had been

290000 iterations: started to run a computer to the computer to take a bit of a problem off the ship and the sun and the air was the sound

500000 iterations: "I think the Galaxy will be a lot of things that the second man who could not be continually and the sound of the stars"
One Hot Encoding Shortcomings

One-hot encoding is lacking because it fails to capture semantic similarity between words, i.e., the inherent meaning of word

For example, the words “happy”, “joyful”, and “pleased” all have similar meanings, but under one-hot encoding they are three distinct and unrelated entities

What if we could capture the meaning of words within a numerical context?
Word Embeddings

Word embeddings are vector representations of words that attempt to capture semantic meaning.

Each word is represented as a vector of numerical values.

Each index in the vector represents some abstract “concept”
  - These concepts are unlabeled and learned during training.

Words that are similar will have similar vectors:

<table>
<thead>
<tr>
<th></th>
<th>Masculinity</th>
<th>Royality</th>
<th>Youth</th>
<th>Intelligence</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>0.95</td>
<td>0.95</td>
<td>-0.1</td>
<td>0.6</td>
</tr>
<tr>
<td>Queen</td>
<td>-0.95</td>
<td>0.95</td>
<td>-0.1</td>
<td>0.6</td>
</tr>
<tr>
<td>Prince</td>
<td>0.8</td>
<td>0.8</td>
<td>0.7</td>
<td>0.4</td>
</tr>
<tr>
<td>Woman</td>
<td>-0.95</td>
<td>0.01</td>
<td>-0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>Peasant</td>
<td>0.1</td>
<td>-0.95</td>
<td>0.1</td>
<td>-0.3</td>
</tr>
<tr>
<td>Doctor</td>
<td>0.12</td>
<td>0.1</td>
<td>-0.2</td>
<td>0.95</td>
</tr>
</tbody>
</table>
Word2Vec

Words that appear in the same context are more likely to have the same meaning

- I am **excited** to see you today!
- I am **ecstatic** to see you today!

Word2Vec is an algorithm that uses a funnel-shaped single hidden layer neural network (similar to autoencoder) to create word embeddings

Given a word (in one-hot encoded format), it tries to predict the neighbors of that word (also in one-hot encoded format), or vice versa

Words that appear in the same context will have similar embeddings
Word2Vec

The model is trained on a large corpus of text using regular backpropagation.

For each word in the corpus, predict the 5 words to the left and right (or vice versa).

Once the model is trained, the embedding for a particular word is the row of the weight matrix associated with that word.

Many pretrained vectors (e.g. Google) can be downloaded online.
Word2Vec on 20 Newsgroups
Basic Deep Learning NLP Pipeline

Generate Word Embeddings
  ◦ Python gensim package

Feed word embeddings into LSTM or GRU layer

Feed output of LSTM or GRU layer into softmax classifier
NLP Applications for RNNs

Language Models
- Given a series of words, predict the next word
- Understand the inherent patterns in a given language
- Useful for autocompletion and machine translation

Sentiment Analysis
- Given a sentence or document, classify if it is positive or negative
- Useful for analyzing the success of a product launch or automated stock trading based off news

Other forms text classification
- Cancer pathology report classification
Advanced Applications

Question Answering
- Read a document and then answer questions
- Many models use RNNs as their foundation

Automated Image Captioning
- Given an image, automatically generate a caption
- Many models use both CNNs and RNNs

Machine Translation
- Automatically translate text from one language to another
- Many models (including Google Translate) use RNNs as their foundation
LSTM Improvements

Bi-directional LSTMs

Sometimes, important context for a word comes after the word (especially important translation)

◦ I saw a **crane** flying across the sky
◦ I saw a **crane** lifting a large boulder

Solution - use two LSTM layers, one that reads the input forward and one that reads the input backwards, and concatenate their outputs
LSTM Improvements
Attention Mechanisms

Sometimes only a few words in a sentence or document are important and the rest do not contribute as much meaning

- For example, when classifying cancer location from cancer pathology reports, we may only care about certain keywords like “right upper lung” or “ovarian”

In a traditional RNN, we usually take the output at the last timestep

By the last timestep, information from the important words may have been diluted, even with LSTMs and GRUs units

How can we capture the information at the most important words?
LSTM Improvements

Attention Mechanisms

Naïve solution: to prevent information loss, instead of using the LSTM output at the last timestep, take the LSTM output at every timestep and use the average.

Better solution: find the important timesteps, and weight the output at those timesteps much higher when doing the average.

Figure 5: Documents from Yelp 2013. Label 4 means star 5, label 0 means star 1.
LSTM Improvements
Attention Mechanisms

An attention mechanism calculates how important the LSTM output at each timestep is.

It’s a simple feedforward network with a single (tanh) hidden layer and a softmax output.

At each timestep, feed the output from the LSTM/GRU into the attention mechanism.
LSTM Improvements
Attention Mechanisms

Once the attention mechanism has all the timesteps, it calculates a softmax over all the timesteps:
- softmax always adds to 1

The softmax tells us how to weight the output at each timestep, i.e., how important each timestep is.

Multiply the output at each timestep with its corresponding softmax weight and add to create a weighted average.
LSTM Improvements

Attention Mechanisms

Attention mechanisms can take into account “context” to determine what’s important.

Remember dot product is a measure of similarity – two vectors that are similar will have larger dot product.

In normal softmax, dot product input with randomly initialized weights before applying softmax function.

\[
\alpha_{it} = \frac{\exp(u_{it}^T u_w)}{\sum_t \exp(u_{it}^T u_w)}
\]
LSTM Improvements
Attention Mechanisms

Instead, we can dot product with a vector that represents “context” to find words most similar/relevant to that context:

◦ For question answering, can represent a question being asked
◦ For machine translation, can represent the previous word
◦ For classification, can be initialized randomly and learned during training
LSTM Improvements
Attention Mechanisms

With attention, you can visualize how important each timestep is for a particular task.
LSTM Improvements
Attention Mechanisms

With attention, you can visualize how important each timestep is for a particular task.
CNNs for Text Classification

Start with Word Embeddings
- If you have 10 words, and your embedding size is 300, you’ll have a 10x300 matrix

3 Parallel Convolution Layers
- Take in word embeddings
- Sliding window that processes 3, 4, and 5 words at a time (1D conv)
- Filter sizes are 3x300x100, 4x300x100, and 5x300x100 (width, in-channels, out-channels)
- Each conv layer outputs 10x100 matrix
CNNs for Text Classification

Maxpool and Concatenate
- For each filter channel, maxpool across the entire width of sentence
- This is like picking the ‘most important’ word in the sentence for each channel
- Also ensures every sentence, no matter how long, is represented by same length vector
- For each of the three 10x100 matrices, returns 1x100 matrix
- Concatenate the three 1x100 matrices into a 1x300 matrix

Dense and Softmax
Hierarchical Attention Networks
Problem Overview

National Cancer Institute has asked Oak Ridge National Lab to develop a program that can automatically classify cancer pathology reports.

Pathology reports are what doctors write up when they diagnose cancer, and NCI uses them to calculate national statistics and track health trends.

Challenges:
- Different doctors use different terminology to label the same types of cancer
- Some diagnoses may reference other types of cancer or other organs that are not the actual cancer being diagnosed
- Typos

Task: given a pathology report, teach a program to find the type of cancer, location of cancer, histological grade, etc.
Approach

The performance of various different classifiers were tested:

- Traditional machine learning classifiers: Naive Bayes, Logistic Regression, Support Vector Machines, Random Forests, and XG-Boost
- Traditional machine learning classifiers require manually defined features, such as n-grams and tf-idf
- Given enough data, deep learning methods can learn their own features, such as which words or phrases are important

The Hierarchical Attention Network is a relatively new deep learning model that came out last year and is one of the top performers
The Hierarchical Attention Network (HAN) is a deep learning model for document classification.

Built from bidirectional RNNs composed of GRUs/LSTMs with attention mechanisms.

Composed of “hierarchies” where the outputs of the lower hierarchies become the inputs to the upper hierarchies.
**HAN Architecture**

Before feeding a document into the HAN, we first break it down into sentences (or in our case, lines)

The word hierarchy is responsible for creating sentence embeddings

- This hierarchy reads in one full sentence a time, in the form of word embeddings
- The attention mechanism selects the most important words
- The output is a sentence embedding that captures the semantic content of the sentence based on the most important words
HAN Architecture

The sentence hierarchy is responsible for creating the final document embedding
- Identical structure with the word hierarchy
- Reads in the sentence embeddings output from the word hierarchy
- The attention mechanism selects the most important sentence
- The output is a document embedding representing the meaning of the entire document

The final document embedding is used for classification
Experimental Setup

945 cancer pathology reports, all cases of breast and lung cancer

10 – fold cross validation used, 30 epochs per fold

Hyperparameter optimization applied on models to find optimal parameters

Two main tasks – primary site classification and histological grade classification

◦ Uneven class distribution, some with only ~10 occurrences in dataset
◦ F-score used for performance metric
◦ Micro F-score is weighted F-score average based on class size
◦ Macro F-score is unweighted F-score average across all classes
### HAN Performance

**Primary Site**

12 possible cancer subsite locations
- 5 lung subsites
- 7 breast subsites

Deep learning methods outperformed all traditional ML methods except for XGBoost

HAN had best performance, pretraining improved performance even further

#### Traditional Machine Learning Classifiers

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Primary Site Micro F-Score</th>
<th>Primary Site Macro F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>.554 (.521, .586)</td>
<td>.161 (.152, .170)</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>.621 (.589, .652)</td>
<td>.222 (.207, .237)</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>.616 (.585, .646)</td>
<td>.220 (.205, .234)</td>
</tr>
<tr>
<td>(C = 1, gamma = 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random Forest</td>
<td>.628 (.597, .661)</td>
<td>.258 (.236, .283)</td>
</tr>
<tr>
<td>(num trees = 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XGBoost</td>
<td>.709 (.681, .738)</td>
<td>.441 (.404, .474)</td>
</tr>
<tr>
<td>(max depth = 5, n estimators = 300)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Deep Learning Classifiers

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Primary Site Micro F-Score</th>
<th>Primary Site Macro F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recurrent Neural Network</td>
<td>.694 (.666, .722)</td>
<td>.468 (.432, .502)</td>
</tr>
<tr>
<td>(with attention mechanism)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convolutional Neural Network</td>
<td>.712 (.680, .736)</td>
<td>.398 (.359, .434)</td>
</tr>
<tr>
<td>Hierarchical Attention Network (no pretraining)</td>
<td>.784 (.759, .810)</td>
<td>.566 (.525, .607)</td>
</tr>
<tr>
<td>Hierarchical Attention Network (with pretraining)</td>
<td>.800 (.776, .825)</td>
<td>.594 (.553, .636)</td>
</tr>
</tbody>
</table>
HAN Performance
Histological Grade

4 possible histological grades
- 1-4, indicating how abnormal tumor cells and tumor tissues look under a microscope with 4 being most abnormal
- Indicates how quickly a tumor is likely to grow and spread

Other than RNNs, deep learning models generally outperform traditional ML models

HAN had best performance, but pretraining did not help performance

### Traditional Machine Learning Classifiers

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Histological Grade Micro F-Score</th>
<th>Histological Grade Macro F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>0.481 (0.442, 0.519)</td>
<td>0.264 (0.244, 0.283)</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.540 (0.499, 0.576)</td>
<td>0.340 (0.309, 0.371)</td>
</tr>
<tr>
<td>Support Vector Machine (C = 1, gamma = 1)</td>
<td>0.520 (0.482, 0.558)</td>
<td>0.330 (0.301, 0.357)</td>
</tr>
<tr>
<td>Random Forest (num trees = 100)</td>
<td>0.597 (0.558, 0.636)</td>
<td>0.412 (0.364, 0.476)</td>
</tr>
<tr>
<td>XGBoost (max depth = 5, n estimators = 300)</td>
<td>0.673 (0.634, 0.709)</td>
<td>0.593 (0.516, 0.662)</td>
</tr>
</tbody>
</table>

### Deep Learning Classifiers

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Histological Grade Micro F-Score</th>
<th>Histological Grade Macro F-Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recurrent Neural Network (with attention mechanism)</td>
<td>0.580 (0.541, 0.617)</td>
<td>0.474 (0.416, 0.536)</td>
</tr>
<tr>
<td>Convolutional Neural Network</td>
<td>0.716 (0.681, 0.750)</td>
<td>0.521 (0.493, 0.548)</td>
</tr>
<tr>
<td>Hierarchical Attention Network (no pretraining)</td>
<td>0.916 (0.895, 0.936)</td>
<td>0.841 (0.778, 0.895)</td>
</tr>
<tr>
<td>Hierarchical Attention Network (with pretraining)</td>
<td>0.904 (0.881, 0.927)</td>
<td>0.822 (0.744, 0.883)</td>
</tr>
</tbody>
</table>
TFIDF-weighted Word2Vec embeddings reduced to 2 dimensions via PCA for (A.) primary site train reports, (B.) histological grade train reports, (C.) primary site test reports, and (D.) histological grade train reports.
HAN document embeddings reduced to 2 dimensions via PCA for (A.) primary site train reports, (B.) histological grade train reports, (C.) primary site test reports, and (D.) histological grade train reports.
Pretraining

We have access to more unlabeled data than labeled data (approximately 1500 unlabeled, 1000 labeled)

To utilized unlabeled data, we trained our HAN to create document embeddings that matched the corresponding TF-IDF weighted word embeddings for that document

HAN training and validation accuracy with and without pretraining for (A.) primary site task and (B.) histological grade task
clinical information: left lung mass with calcification like consolidation history emphysema cirrhosis liver mass.

diagnosis:
- a. lung left lower lobe mass; needle biopsy:
  - small cell carcinoma

gross description:
- number of specimen containers: 1, labeled with name
- date of birth:
- a. container designation: needle biopsy number of tissue cores:
- multiple; length s: ranging from floatoken to floatoken cm; cassettes: entirely submitted
  - submitted in a
  - microscopic description: histologic examination performed.
  - pathologist:
  - electronically signed datetoken 12:07 pm

specimens submitted:
- a. left lung mass lower lobe
We can also use the HAN’s attention weights to find the words that contribute most towards the classification task at hand:

<table>
<thead>
<tr>
<th>Primary Site</th>
<th>Histological Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>mainstem</td>
<td>poorly</td>
</tr>
<tr>
<td>adenocarcinoma</td>
<td>g2</td>
</tr>
<tr>
<td>lower</td>
<td>high</td>
</tr>
<tr>
<td>breast</td>
<td>iii</td>
</tr>
<tr>
<td></td>
<td>dlr</td>
</tr>
<tr>
<td></td>
<td>Undifferentiated</td>
</tr>
<tr>
<td></td>
<td>g3</td>
</tr>
<tr>
<td></td>
<td>ii</td>
</tr>
<tr>
<td></td>
<td>i</td>
</tr>
<tr>
<td></td>
<td>g1</td>
</tr>
<tr>
<td></td>
<td>moderately</td>
</tr>
<tr>
<td></td>
<td>intermediate</td>
</tr>
<tr>
<td></td>
<td>well</td>
</tr>
<tr>
<td></td>
<td>arising</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
Scaling

Relative to other models, HAN is very slow to train
- On CPU, HAN takes approximately 4 hours to go through 30 epochs
- In comparison, CNN takes around 40 minutes to go through 30 epochs, and traditional machine learning classifiers take less than a minute
- The HAN is slow due to its complex architecture and use of RNNs, so gradients are very expensive to compute

We are current working to scale the HAN to run on multiple GPUs
- On Tensorflow, RNNs on GPU run slower than on CPU
- We are considering exploring a PyTorch implementation to get around this problem

We have successfully developed a distributed CPU-only HAN that runs on TITAN using MPI, with 4x speedup on 8 nodes
Attention is All You Need

New paper that came out June 2017 from Google Brain, in which they showed they could get competitive results in machine translation with only attention mechanisms and no RNNs.

We applied the same architecture to replace the RNNs in our HAN.

Because attention mechanisms are just matrix multiplications, it runs about 10x faster than the HAN with RNNs.

This new model performs almost as well as the HAN with RNNs — 0.77 micro-F on primary site (compared to 0.78 in original HAN), and 0.86 micro-F on histological grade (compared to 0.91 in original HAN).

Because no RNNs are utilized, this model is much easier to scale on the GPU.
Other Future Work

Multitask Learning
- Predict histological grade, primary site, and other tasks simultaneously within the same model
- Hopefully boost the performance of all tasks by sharing information across tasks

Semi-Supervised Learning
- Utilize unlabeled data during training rather than in pretraining with the goal of improving classification performance
- This task is challenging because in most semi-supervised tasks, we know all the labels within the dataset. In our case, we only have a subset of the labels.
Questions?