Lab -- Recurrent Neural Networks

Group information

Create Your Group

Once you and your partner(s) are in a breakout room, you need to create a group.

Instructions:

(1) **One** of you should click here to create a group.

(2) Everyone else should then enter the given new group name (including trailing numbers) into the box below (and press enter when done).
Name of group to join:

Reload this page anytime to refresh your group status (and to get a delete button, if you want to remove yourself from a group you've joined).

Groups are limited to 3 persons max each, so that checkoff discussions can be inclusive.

You are not currently in any group

For this lab:

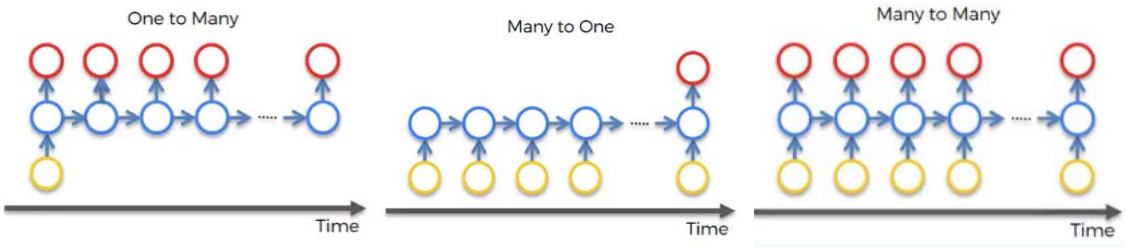
- You will need to understand the material in the notes on **recurrent neural networks**.
- We have prepared a Colab notebook that is essential for doing this lab that can be found **here**. We recommend using the colab but code files can be found **here**. You can download this to your computer if you prefer.

RNNs

1) Applications of RNNs

1.1) Examples

In this section, we'll consider examples of RNN applications with respect to the forms of inputs and outputs.



1.2) Choose a structure

Choose an RNN structure from one-to-many, many-to-one, and many-to-many, to address each of the problems below. For each, describe how you would structure the input to the RNN (which has to be a **fixed-length** vector at each time step), and what kind of output unit(s) you would use.

Note that yellow nodes stand for inputs, blue nodes stand for hidden units, and red nodes stand for outputs.

1.2.1)

Given a review for a new product on a shopping website, detect whether the text's sentiment is positive or negative, that is, whether the writer writes positively or negatively about the new product.

1.2.2)

Assign a part-of-speech tag to each word in an English sentence: "The old man will man the boat." -> ["determiner", "adjective", "noun", "modal", "verb", "determiner", "noun", "punctuation"]

1.2.3)

Given a picture (encoded as a vector of features computed by a CNN), generate a caption, for example: "A cat is sitting on a rock" or "Dogs playing at the park".

1.2.4)

Translate a sentence from Spanish to English: "Quiero aprender más álgebra lineal." -> "I want to learn more linear algebra." As you can see, here we have different sequence lengths and words might be in different order (álgebra lineal vs. linear algebra). What architecture (or combination of architectures), from the above described, would you use to address this problem? Be ready to discuss some challenges of these architectures.

Check this box and submit when you have finished all parts of this question.

Save | Submit | View Answer | Ask for Help

As staff, you are always allowed to submit. If you were a student, you would see the following:

You have infinitely many submissions remaining.

2) Generating sequences

In this question, we will look into using an RNN model to **predict the next element** in a sequence. We will focus on sequences of text characters (this is sometimes referred to as a "language" model). We will want to train on one or more sequences and then, given an initial character or sequence of characters, we want the RNN to predict what characters should come next. An example sequence might be the following:

```
c = ['m', 'i', 't']
```

This sequence will be used both as input to an RNN and as desired output (offset by one time step), since we are training the RNN to produce the next character in the sequence.

2.1) RNN structure

Since the input to an RNN at each time step is encoded as a fixed-length vector, we will first encode each character of the sequence using a one-hot encoding. Let $\phi(c_t)$ represent the one-hot encoding of character c_t. Recall that if we have V possible characters, then $\phi(c_t)$ will be a vector of size V where each element corresponds to one of the possible characters.

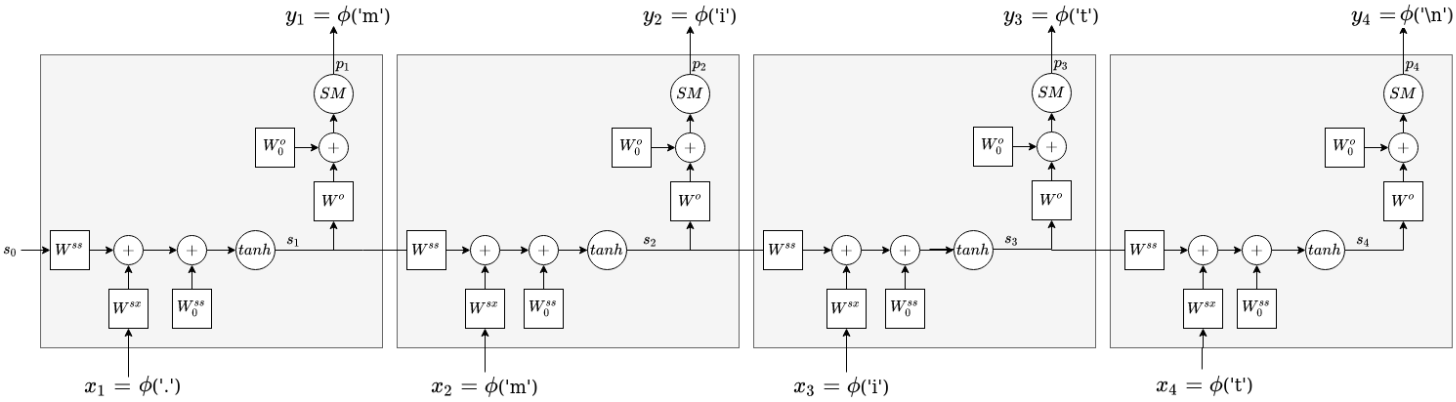
The inputs to the RNN, x, will consist of the encoded characters with a special start character. The output sequence of the RNN, y, will also consist of the encoded characters but have a special end character. In this lab, we will use '.' and '\n' as the start and end character respectively. This format will allow us to train an RNN to do character prediction (at time t, the RNN will have seen $c_1, \ldots, c_{t-1}$ and will try to predict c_t).

$$x = [<\text{start}>, \phi(c_1), \phi(c_2), \ldots, \phi(c_n)]$$

$$y = [\phi(c_1), \phi(c_2), \ldots, \phi(c_n), <\text{end}>]$$

Note that x and y are shifted by one time step.

The following diagram **unravels the RNN** and shows what x and y would be for the sequence `c=['m','i','t']`:



The particular form of the RNN that we will look at is:

$$\begin{aligned} x_t &= \phi(c_{t-1}) \\ s_t &= \tanh(W^{ss} s_{t-1} + W^{sx} x_t + W_0^{ss}) \\ p_t &= \text{softmax}(W^o s_t + W_0^o) \end{aligned}$$

Note that this in the same form as the Continuous State Machines in Lab 8.

2.1.1)

Be prepared to discuss these points during checkoff:

- When the symbols are all the lowercase letters in the English alphabet, what is the size of V?
- Which time step of x should the output y_t match?
- What is the role of s_t?
- How is p_t related to s_t?

Training: We will first describe how to train this RNN given a dataset of sequences. For each sequence in the dataset:

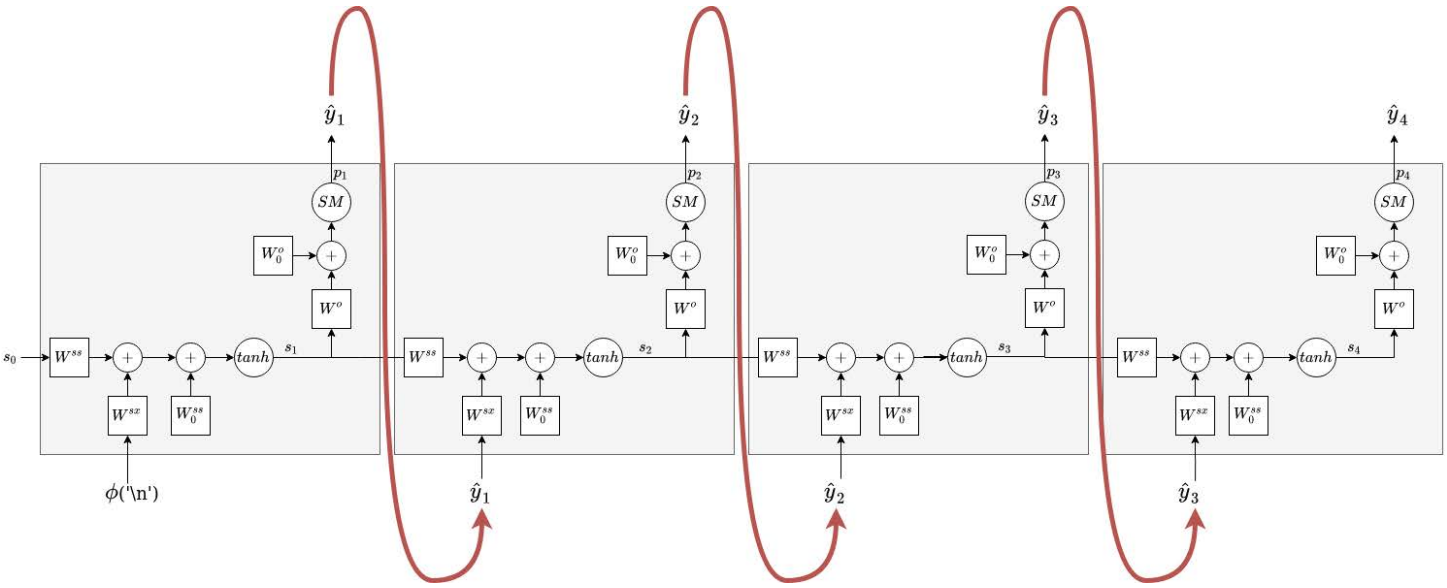
- Format the input and output for the RNN as described above (add start/end characters to the sequence for the input/output respectively).
- Feed each character of the input into the RNN (at time t, this will be c_{t-1} from the original sequence).

- Use the NLL between the predictions p_t and the true character encodings y_t and perform backpropagation to update the weights in the matrices.

Generation: Once the RNN is trained, we can use it to generate new text based on its own predictions.

- Starting with the *start* symbol ('.'), it predicts a next character by sampling it from the softmax distribution on the output p_t in the trained model, then it feeds that character as the next input into the model and repeats until an *end* symbol ('\n') is generated.
  An alternative generation approach picks the most likely character at p_t instead of sampling from the softmax distribution.

We visualize this process below where we use $\hat{y}_t$ to represent the encoding of the character the network predicted at time t. Notice how the network uses its own predictions as inputs. This differs from training where we always used the characters from the sequence in our dataset as inputs (see the training diagram above).



2.1.2)

Be prepared to discuss these points during checkoff:

- How are the training sequences used in the **training** phase?
- How are multiple sequences produces in the **generation** phase?

2.2) Memorizing a Sequence

We will first see how well these models can learn to produce a single sequence. Let's consider generating a sequence of 10 `a` characters. We will train the RNN and then call the generation method 11 times. The first time we give it `'.'` as input and then the following 10 times we give it the output of the RNN at the previous time step. We want the outputs to be:

```
['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', '\n']
```

2.2.1)

What does the RNN need to learn in order to perform this task? Mark all that are true.

- [ ] A linear classifier that predicts 'a' versus `\n' depending on the hidden state
- [ ] A state machine that encodes a count of the number of characters seen so far in the hidden state

Save | Submit | View Answer | Ask for Help

As staff, you are always allowed to submit. If you were a student, you would see the following:
You have infinitely many submissions remaining.

Next, assume that the one-hot encoding of 'a' is `[[1], [0]]` and the encoding of '\n' is `[[0], [1]]`. For now, assume that the encoding of '.' is also `[[0], [1]]`.

Picking the dimension of the hidden state, m, to be 1 and training an RNN with this sequence for 10000 iterations of SGD, we obtain the following RNN matrices:

```
Wss = [[2.66555941]] Wsx = [[-0.75865931,  2.91783285]] Wss0 = [[-0.37149935]]
Wo = [[ 9.63304408], [-9.63296282]]  Wo0 = [[ 2.64391274], [-2.64382701]]
```

If we now call the RNN generation function repeatedly, we get the following output (approximately). Note that `state` is s_{t-1}, `x` is x_t, `new_state` is s_t and `p` is p_t.

```
t= 1 state [[0.]]         x [[0] [1]] new_state [[0.98779174]] p [[1.0], [0.0]]
t= 2 state [[0.98779174]] x [[1] [0]] new_state [[0.90566354]] p [[1.0], [0.0]]
t= 3 state [[0.90566354]] x [[1] [0]] new_state [[0.85753147]] p [[1.0], [0.0]]
t= 4 state [[0.85753147]] x [[1] [0]] new_state [[0.81961469]] p [[1.0], [0.0]]
t= 5 state [[0.81961469]] x [[1] [0]] new_state [[0.78357789]] p [[1.0], [0.0]]
t= 6 state [[0.78357789]] x [[1] [0]] new_state [[0.74361363]] p [[1.0], [0.0]]
t= 7 state [[0.74361363]] x [[1] [0]] new_state [[0.69210644]] p [[1.0], [0.0]]
t= 8 state [[0.69210644]] x [[1] [0]] new_state [[0.61361073]] p [[1.0], [0.0]]
t= 9 state [[0.61361073]] x [[1] [0]] new_state [[0.46639812]] p [[1.0], [0.0]]
t=10 state [[0.46639812]] x [[1] [0]] new_state [[0.11257404]] p [[1.0], [0.0]]
t=11 state [[0.11257404]] x [[1] [0]] new_state [[-0.68052211]]  p [[0.0], [1.0]]
```

2.2.2)

For what values of s_t is the output character '\n'?

- ○ positive values
- ○ negative values

Save | Submit | View Answer | Ask for Help

As staff, you are always allowed to submit. If you were a student, you would see the following:
You have infinitely many submissions remaining.

2.2.3)

Be prepared to discuss these points during checkoff:

- Note that the state starts with value 0, how does it get be close to 1 at the first step?
- At what time step does the `new_state` (s_t) go below 0?
- How does the change in sign of s_t affect the output? (Think about how softmax works)

2.2.4)

The sequence above had a fixed length, but what if we didn't include a termination character and wanted to generate repeating strings?

- What if we wanted to train a language model to generate the sequence `['m', 'i', 't', 'm', 'i', 't', ....]` forever? What would the state need to encode?
- What about `['m', 'm', 'i', 'i', 't', 't', 'm', 'm', 'i', 'i', ...]` What would the state need to encode?
- What about `['m', 'm', 'm', 'm', ....]` What would the state need to encode?

2.3) Hard Sequences

Some sequences are harder to memorize than others. As a measure of difficulty of learning, let's consider two factors:

- `num_hidden`: The dimension of the hidden state (referred to as m above),
- `num_steps`: The number of steps to run the optimizer for.

We will focus on the first of these - the dimension of the hidden state.

The RNN will first be trained on a dataset of a single sequence. We will then use the trained RNN to generate a new sequence as described above (see the **generation** section). **Our goal is to have the RNN reproduce the training sequence reliably during generation (including stopping at the right place).**

We'll consider these sequences (note we visualize them as strings instead of list of characters):

- $c^{(1)}$ = "aaaaaaaaaa"
- $c^{(2)}$ = "aabaaabbaaaabababaaabaa"
- $c^{(3)}$ = "abcdefghijklmnopqrstuvwxyz"
- $c^{(4)}$ = "abcabcabcabcabc"

Which sequence do you think will be most difficult for the RNN to learn? Recall that the vocabulary includes all the letters in the input, but we're not using size of V as indicating difficulty. Here, "difficult" just consists of needing larger hidden dimension.

-- ▾

Save | Submit | View Answer | Ask for Help

As staff, you are always allowed to submit. If you were a student, you would see the following:
You have infinitely many submissions remaining.

Be prepared to explain your reasoning during the checkoff.

2.4) Running the code

We will now try training an RNN on these single-sequence datasets. We have provided code to do this for you. In the Colab you will find definitions of a procedure for training and using models of sequential data (you can also use `code_for_lab9.py` from the downloadable files).

```
test_word(word, interactive=False, num_hidden=1, num_steps=10000, step_size=0.005)
```

- `word` is a string representing the sequence of characters. These will be converted into a training set of sequence pairs for a language model as described above.
- `num_hidden` indicates the number of units in the hidden layer (the dimension of the states),
- `num_steps` indicates steps of (stochastic) gradient descent,
- `step_size` the magnitude of the gradient descent steps,
- `interactive` indicates, when False, to generate 100 random sequences from learned model, when True, it asks for a partial sequence and then completes it in the most likely way given the learned model. Note: input sequences can only include characters present in the word (or the start character `.`)

Note that `test_word(word="aaaaaaaaaa")` uses as both training and test set only the input `word` (i.e. the model is literally only trying to learn how to represent that sequence).

Try using different values of each of the following parameters:.

- `num_hidden`: The dimension of the hidden state,
- `num_steps`: The number of steps. Try `num_steps` in [1000, 5000, 10000, 15000, 20000].

Note that the initial weights are chosen randomly, so results will vary for each run. You can set the random seed if you can't get consistent results.

2.4.1)

Try learning each of the above sequences, using the `test_word` function for different values of `num_hidden` and `num_steps`; pay attention to the training error value printed by the code. The output of `test_word` is 100 sequences generated from a trained model, that is, sampling from the softmax distribution on the output p_t. Find the difficulty of each string (the minimum size of hidden layer and number of steps required to consistently reproduce the string, where we prioritize minimizing hidden layer size (i.e. `num_hidden`). Comment on your results.

Checkoff 1:
Have a check-off conversation with a staff member, to explain your results.

Ask for Help | Ask for Checkoff

3) Language Modeling

3.1) Larger datasets

We will now try training the RNN using larger datasets with more realistic sequences. We will specifically look at different strategies to generate sequences with a trained model.

- `interactive=True`: You will be prompted to enter the start of a sequence. We will use the RNN to complete the sequence (taking the most likely character at each prediction).
- `interactive_top5=True`: Similar to `interactive=True` but instead of choosing the most likely character at each time step, you will be shown the top 5 most likely characters and asked to choose one.

The `test_food` function uses the file food.txt of recipe names.

- Run `test_food` with `interactive=True`.
- Run `test_food` with `interactive_top5=True`.

3.1.1)

What is the mechanism by which these RNNs generate the top 5 characters? More specifically, what characters do the trained RNN seem to propose for you to choose at each location?

3.1.2)

Without having you manually choose the next character, what would be a mechanism by which these RNNs should generate their output? Would that produce diverse sequences (i.e., if you run the generation multiple times with the same start-of-sequence string, would you get different outputs)?

Note: If your installation of Python has trouble finding the data files, try setting the `dirname` in the `code_for_lab9.py` file to the pathname for the folder where the data can be found.

4) Word Embeddings

RNNs are often used to process language, for example to map from one sequence of words to another sequence of words.

RNNs, like other NNs, take vectors as input, so to get them to process words we need some way of turning words into vectors. One way to do this is with a one-hot encoding (as in the previous lab problems). But there are lots of lower dimensional but more informative representations of words called "word embeddings".

One popular technique for producing word embeddings is called word2vec. It assigns each word a vector embedding such that words that appear in similar sentence contexts have embeddings that are close in vector space. Typically, we create these embedding vectors through machine learning techniques (i.e. an embedding weight matrix of size [*number of words* x *embedding size*] is learned).

4.1) Exploring embeddings

Here, we investigate a phenomenon that will affect any RNN that uses these embeddings. We've trained embeddings for all the words that appeared in a large dataset of news articles. Instead of loading embeddings for every word in this set (together they take up about 3 GB!), we have pre-selected several words whose embeddings have interesting properties.

```
words = ["woman", "man", "boy", "girl", "doctor", "nurse", "programmer", "homemaker", "queen", "king", "receptionist", "librarian", "socialite", "hairdresser", "nanny", "bookkeeper", "stylist", "maestro", "protege", "philosopher", "captain", "architect", "surgeon", "brilliant", "mother", "father"]
```

The function below takes two lists and computes the cosine distance between corresponding elements of the lists:

$$d(w_1, w_2) = 1 - \frac{w_1 \cdot w_2}{||w_1||||w_2||}$$

[We can see that with cosine distance, the magnitude of the involved vectors is irrelevant; distance is dependent on the angle between the two vectors. Smaller angles correspond to closer vectors. Using cosine distance, we also know that distances will range between 0 and 1.]

Use the function below to find the distances between various pairs of words from the above list. Compare the distances to each other. What do the relative distances seem to reflect? What unexpected distances do you notice?

```
1 def run():
2     return t1(w1list=["woman", "man"], w2list=["mother", "father"])
3
```

Run Code | Save | Submit | View Answer | Ask for Help

As staff, you are always allowed to submit. If you were a student, you would see the following:

You have infinitely many submissions remaining.

4.2)

Now let's take a look at some of the problems that arise when we use these embeddings in the wild. Please go to Google Translate and translate the following into Hungarian: "She is an engineer. He is beautiful". What do you get?

Now take the Hungarian translation above and translate it back into English using Google Translate. Please note that Hungarian is a gender neutral language (so there are no gendered pronouns). What do you get? Why might this be happening? How does this relate to the unexpected distances we saw in the previous problem? And where do these unexpected distances come from?

4.3)

What are some applications where using biased word embeddings may have a negative impact?

Checkoff 2:

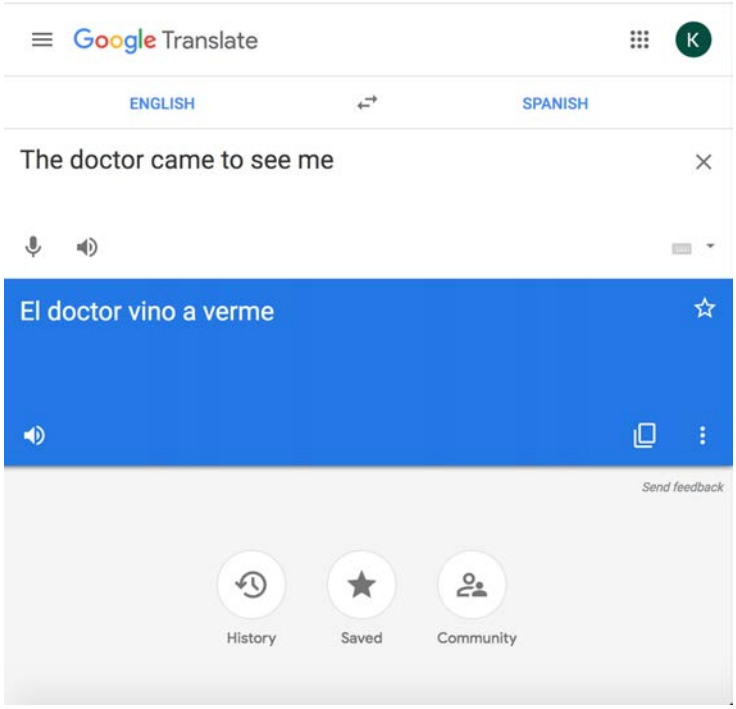
Have a check-off conversation with a staff member, to explain your results.

Ask for Help | Ask for Checkoff

While you wait for a checkoff...

De-biasing these translations is ongoing process. Here's an example from 2019. And here is their fix as of today, but as illustrated in the above examples, it is still far from perfect. If you are curious to read more on the Hungarian translation example, see this blog post. There is also a lot of research in this space. Here's a paper that works on debiasing such embeddings

As of November 20, 2019 vs. November 1, 2020



5) More Food for Thought

In the past few years, another model architecture, the transformer, has replaced the RNNs as the model of choice for language processing tasks. Generative Pre-trained transformer (GPT-3) is one such language model developed by Open-AI. Many developers have come up with many different applications that are often indistinugishable from human generated texts.

This is an application where GPT-3 is used to generate tweets given any word as the theme. For example, here's a tweet generated with `mit` as the theme word: "MIT slaps you across the face with reality and drags you to Mars. That's why it's the best." We'll look at the transformer model in next week's lab.

RES.TLL-008 Social and Ethical Responsibilities of Computing (SERC)

Fall 2021