

# HOMWORK 7: DEEP LEARNING

10-301/10-601 Introduction to Machine Learning (Summer 2024)  
<https://www.cs.cmu.edu/~hchai2/courses/10601/>

OUT: Tuesday, July 2nd  
DUE: Thursday, July 11th

## START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., “Jane explained to me what is asked in Question 2.1”). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: <https://www.cs.cmu.edu/~hchai2/courses/10601/#Syllabus>
- **Late Submission Policy:** See the late submission policy here: <https://www.cs.cmu.edu/~hchai2/courses/10601/#Syllabus>
- **Submitting your work:**
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (<https://gradescope.com>). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). When you are developing, check that the version number of the programming language environment (e.g. Python 3.9.12) and versions of permitted libraries (e.g. numpy 1.23.0) match those used on Gradescope. You have a **total of 10 Gradescope programming submissions**. Use them wisely. In order to not waste code submissions, we recommend debugging your implementation on your local machine (or the linux servers) and making sure your code is running correctly first before any Gradescope coding submission.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, we will be using Gradescope (<https://gradescope.com/>). Please use the provided template. Submissions must be written in LaTeX. Regrade requests can be made, however this gives the TA the opportunity to regrade your entire paper, meaning if additional mistakes are found then points will be deducted. Each derivation/proof should be completed on a separate page. For short answer questions you **should not** include your work in your solution. If you include your work in your solutions, your assignment may not be graded correctly by our AI assisted grader.
- **Materials:** The data that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

For multiple choice or select all that apply questions, shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions. For LaTeX users, replace `\choice` with

\CorrectChoice to obtain a shaded box/circle, and don't change anything else.

## Instructions for Specific Problem Types

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

**Select One:** Who taught this course?

- ☒ Henry Chai
- ☐ 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 scientists?

- ☒ Stephen Hawking
- ☒ Albert Einstein
- ☒ Isaac Newton
- ☐ 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 scientists?

- ☒ Stephen Hawking
- ☒ Albert Einstein
- ☒ Isaac Newton
- ☒ 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 (45 points)

### 1 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.

$$X = \begin{bmatrix} 1 & 0 & -2 & 3 & 4 & 1 \\ 2 & 9 & 5 & 6 & 0 & -1 \\ 0 & -3 & 1 & 3 & 4 & 4 \\ 6 & 5 & 2 & 0 & 6 & 8 \\ -5 & 4 & -3 & 1 & 3 & -2 \\ 4 & 1 & 2 & 8 & 9 & 7 \end{bmatrix} \quad F = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad Y = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

- (a) (1 point) Let an image  $X$  ( $6 \times 6$ ) be convolved with a filter  $F$  ( $3 \times 3$ ) using no padding and a stride of 1 to produce an output  $Y$  ( $4 \times 4$ ). 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  $6 \times 4$  (height  $\times$  width) and a filter of size  $2 \times 2$ , 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  $22 \times 22$ . Using one single  $4 \times 4$  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  $4 \times 4$  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  $40 \times 40$  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  $4 \times 4$  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.

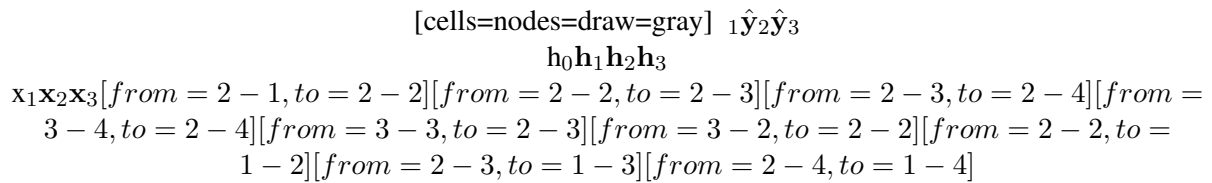
Your Answer

- (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

## 2 Recurrent Neural Network (31 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  $\mathbf{x}_{1:3}$ .

$$\begin{array}{ll}
 \mathbf{x}_t \in \mathbb{R}^3 & \mathbf{W}_{hx} \in \mathbb{R}^{4 \times 3} \\
 \mathbf{h}_t \in \mathbb{R}^4 & \mathbf{W}_{hy} \in \mathbb{R}^{2 \times 4} \\
 \mathbf{y}_t, \hat{\mathbf{y}}_t \in \mathbb{R}^2 & \mathbf{W}_{hh} \in \mathbb{R}^{4 \times 4}
 \end{array}
 \qquad
 \begin{array}{l}
 \mathbf{z}_t = \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t \\
 \mathbf{h}_t = \psi(\mathbf{z}_t) \\
 \mathbf{o}_t = \mathbf{W}_{hy}\mathbf{h}_t \\
 \hat{\mathbf{y}}_t = \text{soft}(\mathbf{o}_t) \\
 J_t = - \sum_{i=1}^2 y_{t,i} \log(\hat{y}_{t,i}) \\
 J = \sum_{t=1}^3 J_t
 \end{array}$$

Above  $\mathbf{y}_t$  is a one-hot vector representing the label for the  $t$ th timestep, *soft* is the **softmax** activation,  $\psi$  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.

$$[cells = nodes = draw = gray] J = \sum_{t=1}^3 J_t$$

$$\mathbf{W}_{hy}$$

$$\mathbf{y}_2 \mathbf{y}_3$$

$$\mathbf{y}_1 J_1 = CE(\hat{\mathbf{y}}_1, \mathbf{y}_1) \text{ part (c)} J_3 = CE(\hat{\mathbf{y}}_3, \mathbf{y}_3)$$

$$\mathbf{o}_1 = \text{soft}(\mathbf{o}_1) \hat{\mathbf{y}}_2 = \text{soft}(\mathbf{o}_2) \hat{\mathbf{y}}_3 = \text{soft}(\mathbf{o}_3)$$

$$\text{part (a)} \mathbf{o}_2 = \text{Lin}(\mathbf{W}_{hy}, \mathbf{h}_2) \mathbf{o}_3 = \text{Lin}(\mathbf{W}_{hy}, \mathbf{h}_3)$$

$$\mathbf{h}_1 = \psi(\mathbf{z}_1) \text{ part (b)} \mathbf{h}_3 = \psi(\mathbf{z}_3)$$

$$\mathbf{h}_0 \mathbf{z}_1 = \text{Lin}(\mathbf{W}_{hh}, \mathbf{W}_{hx}, \mathbf{h}_0, \mathbf{x}_1) \mathbf{z}_2 = \text{Lin}(\mathbf{W}_{hh}, \mathbf{W}_{hx}, \mathbf{h}_1, \mathbf{x}_2) \text{ part (d)}$$

$$\mathbf{x}_1 \mathbf{x}_2 \mathbf{x}_3$$

$$\begin{aligned} & \mathbf{W}_{hx} \mathbf{W}_{hh} [from = 8 - 1, to = 8 - 2] [from = 9 - 2, to = 8 - 2] [from = 8 - 2, to = \\ & 7 - 2] [from = 7 - 2, to = 8 - 3] [from = 7 - 2, to = 6 - 2] [from = 6 - 2, to = 5 - 2] [from = \\ & 5 - 2, to = 4 - 2] [from = 4 - 1, to = 4 - 2] [from = 9 - 3, to = 8 - 3] [from = 8 - 3, to = \\ & 7 - 3] [from = 7 - 3, to = 6 - 3] [from = 6 - 3, to = 5 - 3] [from = 5 - 3, to = 4 - 3] [from = \\ & 3 - 3, to = 4 - 3] [from = 3 - 4, to = 4 - 4] [from = 5 - 4, to = 4 - 4] [from = 6 - 4, to = \\ & 5 - 4] [from = 7 - 4, to = 6 - 4] [from = 8 - 4, to = 7 - 4] [from = 7 - 3, to = 8 - 4] [from = \\ & 9 - 4, to = 8 - 4] [shiftright = 4, from = 10 - 2, to = 8 - 2] [from = 10 - 2, to = 8 - 3] [from = \\ & 10 - 3, to = 8 - 2] [shiftright = 4, from = 10 - 3, to = 8 - 3] [from = 10 - 3, to = \\ & 8 - 4] [shiftright = 2, from = 2 - 3, to = 6 - 4] [from = 2 - 3, to = 6 - 2] [shiftright = \\ & 5, from = 2 - 3, to = 6 - 3] [from = 4 - 2, to = 1 - 3] [shiftright = 4, from = 4 - 3, to = \\ & 1 - 3] [shiftright, from = 4 - 4, to = 1 - 3] [shiftright = 3, from = 10 - 2, to = 8 - 4] \end{aligned}$$

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

2. 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: since calculating this derivative is a little involved. To solve this problem, you must use both the multivariate chain rule and the product rule for matrices. It might help you to remember that  $\mathbf{h}_i$  and  $\mathbf{h}_{i-1}$  are both functions of  $\mathbf{W}_{hh}$ . We highly recommend referencing these matrix calculus notes if you're confused when trying to calculate any of these derivatives and drawing out a computation graph if you're having trouble visualizing.)

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

Work

## Part I

### 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.

### Empirical Questions

The following questions should be completed as you work through the programming component of this assignment. **All plots must be computer-generated.** For all the questions below, unless otherwise specified, set `embed_dim` and `hidden_dim` to be 128, use the full train and validation data (i.e. `train_stories.json` and `valid_stories.json`), and use a learning rate of 0.001.

1. (8 points) For this question, create 2 plots, one with attention and the other without attention, after training your model for 10 epochs. The  $y$ -axis should show the loss function value and the  $x$ -axis should show the number of epochs. Each plot should have 4 lines:
  - the training loss using ReLU activations,
  - the validation loss using ReLU activations,
  - the training loss using `tanh` activations, and
  - the validation loss using `tanh` activations.

Please *title the two plots appropriately* so that we can determine which one is which and also *include a legend* that clearly indicates which curve corresponds to the training and validation losses under different activation functions.

Your Answer

2. (6 points) For this question, create a single plot for this question after training your model for 10 epochs using tanh activations and no attention. The  $y$ -axis should show the loss function value and the  $x$ -axis should show the number of epochs. The plot should have 6 lines:

- the training loss using `embed_dim = hidden_dim = 64`,
- the validation loss using `embed_dim = hidden_dim = 64`,
- the training loss using `embed_dim = hidden_dim = 128`,
- the validation loss using `embed_dim = hidden_dim = 128`,
- the training loss using `embed_dim = hidden_dim = 256`, and
- the validation loss using `embed_dim = hidden_dim = 256`.

Please *include a legend* that clearly indicates which curve corresponds to the training and validation losses under different embedding and hidden dimensionalities.

Your Answer

3. (4 points) For any hyperparameter configuration considered in the previous two questions, train your model for 10 epochs. Report which setting of the hyperparameters you chose (with or without attention, which activation function and what embedding/hidden dimensionality). Then, using the `complete` method provided in the starter code, generate 5 different completions for the prompt “Once upon a time there was a” for each value of `temperature` in the set  $\{0, 0.3, 0.7, 1\}$  (so 20 total completions). Include each of the completions below; make sure to clearly indicate which set was generated using each `temperature` value. Finally, describe the trend in the generated completions as `temperature` increases; which value generated the “best” completions in your opinion?

Your Answer



## 4 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

## 5 Programming: RNN (25 points)

Large language models (LLMs) like ChatGPT, Gemini, and LLaMA have achieved unprecedented levels of success (and hype) over the past few years. In this section, you will become familiar with the building blocks of these models by implementing your very own Recurrent Neural Network LLM<sup>1</sup> with self-attention.

You will be building your model using PyTorch, a widely-used open source deep learning library. **In this homework, you can and should call any built-in PyTorch module (e.g., `nn.Linear`) *except* `nn.RNNCell`, `nn.RNN` and `nn.MultiheadAttention` or any “functional equivalents” of these.**

### 5.1 Libraries (IMPORTANT)

We will be using the following libraries *only* for this assignment. Make sure that the versions in your local environment match the ones listed here.

```
torch==2.2.2
transformers==4.40.1
numpy==1.23.0
```

You should not use `transformers` for anything other than loading in the tokenizer.

### 5.2 The Task

Language modeling is the task of assigning probabilities to texts (i.e. sequences of tokens). Language modeling is most commonly done with *autoregressive* models, which use the chain rule of probability to decompose the probability of a text  $\mathbf{x} = [x_1, x_2, \dots, x_n]$  as follows:

$$P(\mathbf{x}) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \cdots P(x_n|x_1, \dots, x_{n-1})$$

In other words, an autoregressive language model is tasked with predicting the conditional probability of the next token given all the tokens that came before it,  $P(x_{t+1}|x_1, \dots, y_x)$ . In Section 5.5, we will see how these conditional probabilities will be computed by your RNN language model.

### 5.3 The Dataset

We will be training RNN-LMs on the [TinyStories](#) dataset, a collection of 2 million short stories with simple vocabularies generated by GPT-3.5 and GPT-4. Small language models trained on this data have been shown to have a high level of English fluency, strong storytelling abilities, and reasoning capabilities. We will be using a 528 example subset of this data, with 428 training examples and 100 validation examples.

All data files are provided in JSON format. You can load them in using the builtin `json` library as follows:

```
import json
with open(filename) as f:
    data = json.load(f)
```

#### 5.3.1 Tokenization

By their nature, ML models are unable to directly operate on text strings. As such, we have to first *tokenize* text into sequences of tokens by assigning numerical values to substrings (i.e. words, characters, punctuation, even whitespace). In this particular homework, we will be using *subword tokenization*, the kind of tokenization usually used by state-of-the-art language models like GPT-4 and LLaMA. This flavor of tokenization splits text up into subwords, which may be either full words or parts of words (for example, the “-ed” past tense suffix).

---

<sup>1</sup>Little language model

In the `my_tokenizer` directory of the handout, we have provided you with a pretrained tokenizer for the TinyStories dataset, which can be loaded using the `transformers` library:

```
>>> from transformers import AutoTokenizer
>>> tokenizer = AutoTokenizer.from_pretrained("my_tokenizer")
>>> tokenizer.encode("We like ML.")
[360, 192, 110, 1010, 1017, 103, 1]
>>> tokenizer.decode([360, 192, 110, 1010, 1017, 103, 1])
'Ve like ML.</s>'
```

Note that the training and validation data has already been tokenized for you. However, if you would like to convert tokens back to text, you can use the `tokenizer.decode` method, as shown above.

## 5.4 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>

## 5.5 Model Definition

In this homework, you will create a language model that uses an RNN backbone to score and generate text in `rnn.py` (starter code for this file is provided in the handout).

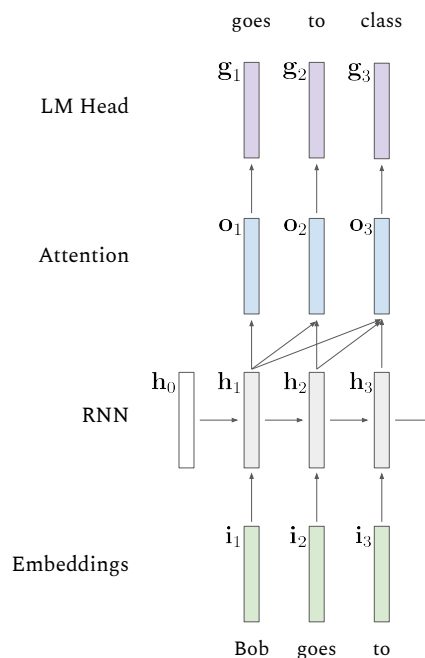


Figure 1: Computation graph of RNN with attention.

### 5.5.1 RNN Cell

This is the building block of the RNN, representing a single RNN step. This class has a single method, `forward`, which takes in the input for the current step  $i_t$  and the previous hidden state  $h_{t-1}$  and outputs

the next hidden state  $\mathbf{h}_t$ . Recall that the hidden state is defined as follows:

$$\mathbf{h}_t = \phi(\mathbf{W}_{i2h}\mathbf{i}_t + \mathbf{W}_{h2h}\mathbf{h}_{t-1})$$

where  $\phi$  is an activation function, either ReLU or tanh in this assignment. **Hint:** Here (and in the rest of the model definition, including in say Section 5.5.3), matrix products like  $\mathbf{W}_{i2h}\mathbf{i}_t$  correspond to `nn.Linear` layers in PyTorch.

### 5.5.2 RNN

This class represents the entire RNN and processes an entire sequence  $[\mathbf{i}_1, \mathbf{i}_2, \dots, \mathbf{i}_T]$ . Note the distinction between this class and `RNNCell`: `RNNCell`'s `forward` method runs a single step of the input sequence, while `RNN`'s `forward` runs for all steps of the input sequence. Hence, this class contains (up to) two modules, (1) the RNN cell and (2) self-attention, if it is applied (see Section 5.5.3). This class has two methods, `step` and `forward`.

`step()`: This method processes a single step of the input sequence. Specifically, it takes both in the current input  $\mathbf{i}_t$  and all preceding hidden states  $[\mathbf{h}_1, \dots, \mathbf{h}_{t-1}]$  and returns both the next hidden state  $\mathbf{h}_t$  and the next output state of the RNN  $\mathbf{o}_t$ . When attention is not used, both the hidden state and output state are just the hidden state yielded by the RNN cell. When attention is applied, the hidden state is still the RNN cell's hidden state, but the output state is the context vector outputted by the self-attention module (plus a residual). Note that this attention module is the reason why this method takes in all preceding hidden states, even though the `RNNCell` itself only needs the single previous hidden state.

`forward()`: This method processes an entire input sequence  $[\mathbf{i}_1, \mathbf{i}_2, \dots, \mathbf{i}_T]$ . It should iteratively call `step` on each vector in the input sequence, along with the appropriate hidden states argument. It should return the hidden states  $[\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T]$  and output states  $[\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T]$  over all steps.

### 5.5.3 Self-Attention

In lecture, you were introduced to attention for an encoder-decoder RNN, where states in the decoder attended to states from the encoder<sup>2</sup>. However, as our RNN has only a decoder and no encoder, we will make states in the decoder attend to previous states in the decoder itself – hence, *self*-attention. This is very similar to what you saw in lecture, only here, the query, keys, and values all come from the decoder hidden states.

The `SelfAttention.forward` method will take in a sequence of hidden states up to the current timestep  $t$ ,  $[\mathbf{h}_1, \dots, \mathbf{h}_t]$ . Note that in this case, the query, keys, and values are defined as follows.

$$\mathbf{q}_t = \mathbf{W}_q \mathbf{h}_t$$

$$\mathbf{k}_t = \mathbf{W}_k \mathbf{h}_t$$

$$\mathbf{v}_t = \mathbf{W}_v \mathbf{h}_t$$

Then, the output of the attention module will be the attention vector for the current timestep,  $\mathbf{a}_t$ , computed by a weighted average of the values. Note that we use scaled dot-product attention (as proposed in the original transformers paper), which divides the dot product of the key and query vectors by the dimension

---

<sup>2</sup>This version of attention is called *cross*-attention, as you're attending *across* to a different model.

of the keys,  $D_{\text{key}}$ .

$$\mathbf{s} = \left[ \frac{\mathbf{k}_1 \cdot \mathbf{q}_t}{\sqrt{D_{\text{key}}}}, \frac{\mathbf{k}_2 \cdot \mathbf{q}_t}{\sqrt{D_{\text{key}}}}, \dots, \frac{\mathbf{k}_t \cdot \mathbf{q}_t}{\sqrt{D_{\text{key}}}} \right]^T \quad (\text{Attention Scores})$$

$$\mathbf{w} = \text{Softmax}(\mathbf{s}) \quad (\text{Attention Weights})$$

$$\mathbf{a}_t = \mathbf{W}_o \left( \sum_{n=1}^t w_n \mathbf{v}_n \right)$$

Then, back in the `RNN.step` method, the attention output will be added with a residual of the current timestep's hidden state to get the RNN output state:  $\mathbf{o}_t = \mathbf{a}_t + \mathbf{h}_t$ . This line in the code has already been written for you.

### 5.5.4 RNN Language Model

Now, we can use the RNN backbone we have defined in the previous sections to create our very own LLM. However, we will need to add a couple modules on both ends of the RNN. Before the RNN, we need an embedding layer to convert integer token indices  $[x_1, \dots, x_T]$  to vector embeddings  $[\mathbf{i}_1, \dots, \mathbf{i}_T]$  that can be passed into the RNN as inputs. **Hint:** use `nn.Embedding` for this.

Meanwhile, after the RNN, we will need to add a language modeling head, a linear layer  $W_{\text{LM}}$  that projects the RNN output  $\mathbf{o}_t$  to the next-token logits  $\mathbf{g}_t = W_{\text{LM}}\mathbf{o}_t$ . These logits are not actually the next-token distribution; rather, they are “raw scores” that can be fed into a softmax to yield the distribution. Hence, we have that

$$P(x_{t+1}|x_1, \dots, x_t) = \text{Softmax}(\mathbf{g}_t)$$

## 5.6 Training and Evaluation

Now that you have created a working model, it's time to train and evaluate it! For this section, you will complete two functions: `train()` and `validate()`.

`train()`: This function will run a single epoch of training for the RNN language model. Note that we are using stochastic gradient descent, so we train on the examples one at a time.

We will use the same loss function as in HW5, cross entropy loss. Here, the cross entropy is computed between the predicted next-token distribution  $P(\hat{y}_{t+1}|y_1, \dots, y_t)$  and a target one-hot distribution with a 1 at the index corresponding to the true next token  $y_{t+1}$ . This loss should be computed for each token and then averaged over the sequence. Please see the `nn.CrossEntropyLoss` PyTorch documentation for instructions on how it should be used to compute the cross entropy loss.

Be careful to correctly shift the target tokens. For instance, the target for the first token is not the first token itself but the second (i.e., the next token). Also, as there is no token after the last token  $x_T$ , there can be no target for the last step's predicted next-token distribution  $P(x_{T+1}|x_1, \dots, x_T)$ . Thus, we can only compute loss for the first  $T - 1$  tokens in the sequence.

`validate()`: This function computes the average loss over a validation dataset. This should be implemented nearly the same as the `train` method, minus the gradient updates to the model.

## 5.7 Text Generation

While we have been training language models to compute probabilities over texts, they are more than probability estimators! Instead of using the next-token distribution to score existing tokens, we can also use it to predict new ones.

Suppose we are given a text prefix consisting of tokens  $\mathbf{x} = [x_1, \dots, x_m]$ . Then when we pass the final token  $x_m$  into our language model, we will get next token logits  $\mathbf{g}_m$  which imply a distribution  $P(x_{m+1}|x_1, \dots, x_m)$ . In this homework, you will implement two methods of picking a next token  $\hat{x}_{m+1}$  using this distribution:

1. **Greedy:** Pick the next token with the highest probability.

$$\hat{x}_{m+1} = \underset{x_{m+1}}{\operatorname{argmax}} P(x_{m+1}|x_1, \dots, x_m)$$

2. **Temperature Sampling:** Rather than deterministically picking the highest probability next token, we can also randomly sample a next token. However, we don't necessarily have to sample from the next-token distribution  $P(x_{m+1}|x_1, \dots, x_m)$  itself. Instead, we will sample from a slightly different *temperature-adjusted* distribution  $Q$ :

$$Q(x_{m+1}|x_1, \dots, x_m) = \operatorname{Softmax}(\mathbf{g}_m/\tau)$$

$$\hat{x}_{m+1} \sim Q(x_{m+1}|x_1, \dots, x_m)$$

where  $\mathbf{g}_m$  refers to the next-token logits (as defined in 5.5.4) and  $\tau$  is a sampling parameter referred to as the “temperature”. The value of  $\tau$  affects how “random” the samples are. Increasing the value of  $\tau$  increases “randomness,” while setting  $\tau = 0$  recovers greedy decoding.

After picking a token, this token can be fed back into the language model (i.e., get its embedding, feed that into the next step of the RNN, etc...) to yield another next-token distribution. This process can be repeated until some stopping criterion is met.

You will implement the `RNNLanguageModel.generate` method, which generates tokens given a prefix token sequence. Once you have completed it, you can use the `complete` function we provide to generate text from a string prefix.

## 5.8 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_data> <val_data>`

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. `<--embed_dim>`  
`<--hidden_dim> <--activation> <--use_attention> <--lr> <--num_epochs>`

These arguments are described below:

1. `<--train_data>`: string path to the training input `.txt` file
2. `<--val_data>`: string path to the validation input `.txt` file
3. `<--embed_dim>` positive integer specifying the size of the sentence embedding vector (**hyper-parameter**)
4. `<--hidden_dim>` positive integer specifying the number of hidden units to use in the model's hidden layer (**hyper-parameter**)

5. `<--activation>` string specifying activation layer to use in the RNN, either "tanh" or "relu" (**hyper-parameter**)
6. `<--use_attention>` integer flag specifying whether to use the attention head (**hyper-parameter**); if 1, use attention, if 0 don't.
7. `<--lr>` float value specifying the learning rate (**hyper-parameter**)
8. `<--num_epochs>` integer specifying the number of times the model will pass through the entire training dataset (**hyper-parameter**)
9. `<--metrics_out>` string path of output .txt file to which train and validation loss metrics should be written

Below is an example command to run using Tanh activation.

```
python3 rnn.py --train_data train_stories.json \  
--val_data valid_stories.json --embed_dim 256 \  
--hidden_dim 256 --activation tanh --use_attention 1 --lr 1e-3 \  
--num_epochs 10 --metrics_out metrics.txt
```

## 5.9 Outputs and Tests

Your code should write out a single output file (path given by the `--metrics_out` flag) containing the train and validation losses per epoch. Metrics writing is already taken care of for you in the starter code.

To help you debug your code, we've included a test file in your handout, `test_rnn.py`. This is a nonexhaustive set of tests which are meant to help you make sure your implementation is correct. Passing these tests does not guarantee a full score in your Gradescope submission, but it will help you identify functions which have errors. Do not edit these tests as we will not be able to guarantee correctness if you modify these tests. To run the test, run the following command line:

```
python3 test_rnn.py
```

In addition, for debugging purposes, we have included "tiny" versions of the train and validation datasets and a file `tiny_metrics.txt` which contains metrics for the following command:

```
python3 rnn.py --train_data tiny_train_stories.json \  
--val_data tiny_valid_stories.json --embed_dim 64 \  
--hidden_dim 64 --activation tanh --use_attention 1 \  
--lr 1e-3 --num_epochs 10 --metrics_out tiny_metrics.txt
```

Your metrics should match these to at least 3-4 decimal places. There may be some more deviation if you run your code locally/on Colab with a GPU, but all Gradescope submissions will be run on CPU so this should not be an issue.

## 5.10 Gradescope Submission

You should submit your `rnn.py` (**Note:** you must submit a `.py` file, `.ipynb` files will not be processed correctly). **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.

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