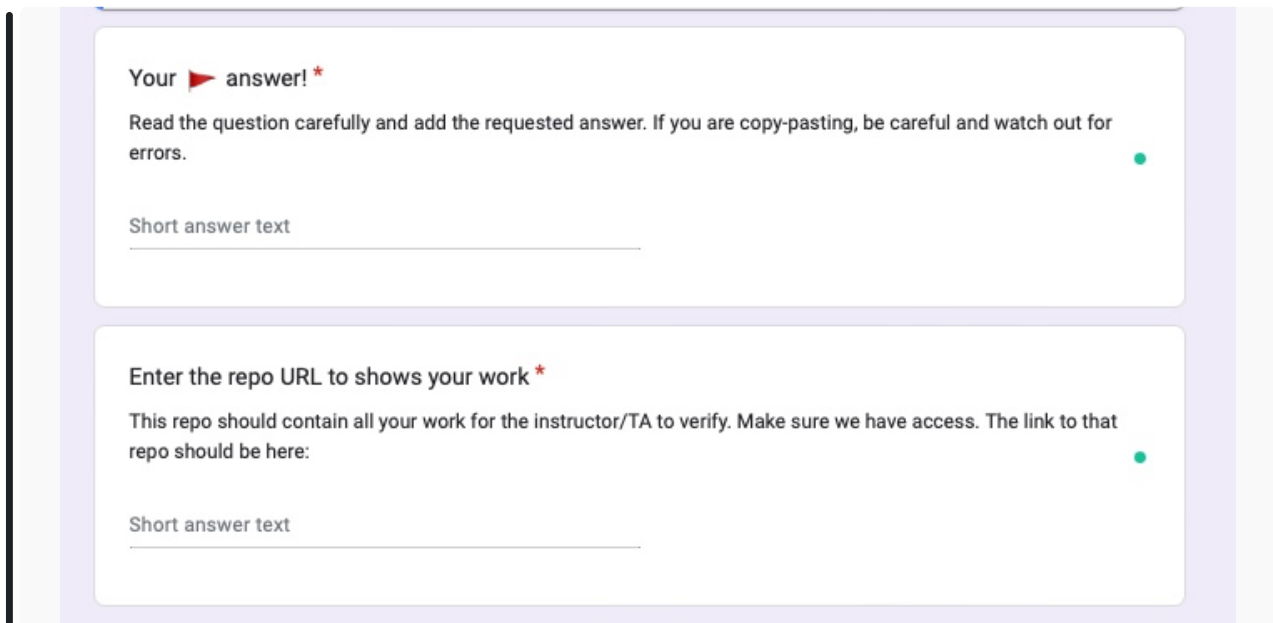


## QUEST 1 : Language Modeling and Transfer Learning

⚠ If you haven't done [QUEST 0 : PyTorch Warmup](#), now is a very good time to finish it. Don't let your quests pile up. If you are stuck at a quest, ask the TA for help or come to the section.

### Capture the Flags 🚩🚩

Starting with QUEST 1, we will be using a DEFCON style "capture the flag" method for evaluation. Every quest has one or more "flags" that you have to capture using the "[capture this flag]" link. Upon clicking the link you will see a Google form with something like this:



**Note:** Make sure you are logged in to Google Forms with your @ucsc account. Failing to do this will mess up your grade assignment!

Here's an example capture the flag:

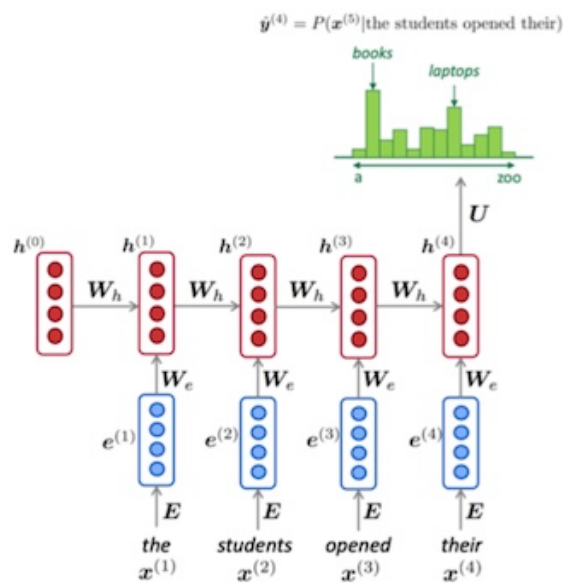
🚩 What is 2 + 2? [\[capture this flag\]](#) (for repo URL, just type `https://google.com`)

Go ahead and capture the flag now! It is very important that you capture the flags **as you progress** through your quest. The timestamps are recorded too ;-)

*In this and future QUESTS you might see some "Bonus 🚩". You don't need to capture them to complete the assignment, unless you want to score bonus points. Bonus points can offset any future point losses in grading. So you might want to collect them after all.*

## Language Modeling and Transfer Learning

In this QUEST, we will explore language modeling with traditional LMs like Elman RNNs, LSTMs and GRUs, and transfer learning with those LMs. This is probably the hardest quest you will embark upon (coding-wise) for the entire course, so get started right away!



A starter code for this QUEST is in this [Github repository](#). First, **copy** the contents of this repo to your own **private repository** with the same name. **You should invite Brendan (kingb12) and me (delip) to your Github repo with read-only access.** Your local workspace folder should look something like this:

```
cs244-quest1 tree .
.
├── data
│   └── prepare.py
├── data.py
├── main.py
├── requirements.txt
├── rnnlm.py
└── utils.py

2 directories, 6 files
```

Set it up by installing the requirements and downloading the dataset:

```
pip install -r requirements.txt
cd data
python prepare.py
ls ./
wikitext-2      wikitext-2-mini
```

This will download the Wikttext-2 dataset, a common dataset used for language modeling. In addition, there is a mini version of wikttext-2 to help you do rapid experiments and debugging etc. Once you have your code working the mini dataset, you can then train it on the larger dataset. Note: Wikttext2 itself is a "toy" dataset by industry standards.

► What is the % of <unk> tokens in the full training data? [[Capture this flag](#)] (for repo URL, use the URL of the repo you've created for the quest from now on)

You cannot run the code yet, because there are some TODOs that need to be implemented.

1) Study the class `RNNModel` in detail. It implements Elman RNNs. Your first task is to extend the code to implement LSTM and GRU.

```
def __init__(
    self,
    vocab_size,
    in_embedding_dim,
    n_hidden,
    n_layers,
    dropout=0.5,
    rnn_type="elman", # can be elman, lstm, gru
):
    super(RNNModel, self).__init__()

    if rnn_type == "elman":
        self.rnn = nn.RNN(
            in_embedding_dim,
            n_hidden,
            n_layers,
            nonlinearity="tanh",
            dropout=dropout,
        )
    else:
        # TODO: implement lstm and gru
        # self.rnn = ...
        raise NotImplementedError
```

2) Add another argument to the constructor of `RNNModel` to make `self.rnn` bidirectional and implement it. Now you should be able to instantiate uni/bidirectional RNNs, LSTMs, and GRUs.

3) Now study `main.py` carefully. Go ahead and implement the function to compute perplexity:

```
def compute_perplexity(loss):
    # TODO: implement this function
    raise NotImplementedError
```

Hint: To get this right, carefully look at the loss and the perplexity expression.

At this point, your code should be ready to run. By default it will run on `wikitext-2-mini` but you can change it to run on the full `wikitext-2` corpus. Beware, that will take some time depending on how much compute you have. There's a lot of optimization that can be done to this code to make it faster, but that's not our goal for this QUEST.

Bonus ► : If you speed up the training code by 100% or more, without just switching to new device or increasing batch size, send Brendan and Me an email with details for bonus points on this quest [[capture the flag](#)]

Bonus ► : Rewrite the code using Dataset, Dataloader. You might want to read up on [collate function](#). Show it in action in a Python notebook. [\[capture the flag\]](#)

Bonus ► : This code does manual optimization. Rewrite the code to use the PyTorch Adam optimizer. [\[capture the flag\]](#)

4) Once you do these steps, you should be able to run the code for RNNs, LSTMs, and GRUs.

```
cs244-quest1-completed python3 main.py
-----
| end of epoch   1 | time:   3.59s | valid loss 10.09 | valid ppl 24151.75
-----
| end of epoch   2 | time:   1.00s | valid loss   9.59 | valid ppl 14582.14
-----
...
```

However, everytime you run this you get a different value for loss and perplexity, even if your default seed is fixed (1111). Identify the sources of this variation. To capture the flags all related to peprlexity, to write a script to the experiment 10 times and report average over the 10 runs.

For the default dataset (wikitext-2-mini) and default arguments, report the following and capture the corresponding flags.

- What is the average final validation perplexity for RNN? [\[capture the flag\]](#)
- What is the average final validation perplexity for LSTM? [\[capture the flag\]](#)
- What is the average final validation perplexity for GRU? [\[capture the flag\]](#)

## Transfer Learning

One way to "transfer" representations here is to use pre-trained embeddings from another model. For this home work, we will use the GloVe 50d vectors. You can get them from [here](#). In rnnlm.py you locate this line:

```
self.in_embedder = nn.Embedding(vocab_size, in_embedding_dim)
```

This is creating a 1-hot encoding based embedding layer.

1) Write code to initialize this layer with the GloVe 50d vectors. After you initialize the Glove embeddings, you will freeze the embedding layers. To do this, you will need to do the following:

```
model.in_embedder.weight.requires_grad = False
```

And

```
params_to_update = [p for p in model.parameters() if param.requires_grad == True]
torch.nn.utils.clip_grad_norm_(params_to_update, args.clip)
for p in params_to_update:
    p.data.add_(p.grad, alpha=-1r)
```

After you implement this, capture the following flags (we are still using default arguments for everything else):

- ▶ What is the average final validation perplexity for LSTM + Glove? [\[capture the flag\]](#)
- ▶ Using the bidirectional argument and code change you made to the model, what is the average final validation perplexity for Bi-LSTM + Glove? [\[capture the flag\]](#)

2) Unfreeze the embedding layer in rnnlm.py

```
model.in_embedder.weight.requires_grad = True
```

After you implement this, capture the following flags (we are still using default arguments for everything else):

- ▶ What is the average final validation perplexity for LSTM + Glove? [\[capture the flag\]](#)
- ▶ Using the bidirectional argument and code change you made to the model, what is the average final validation perplexity for Bi-LSTM + Glove? [\[capture the flag\]](#)

Reflect on the results.

---

This officially ends this QUEST. While we don't plan to grade any of the following or offer bonuses, you can practice your modeling skills by trying out the following:

1. Extend this code to build an LSTM text classifier. You can use this clickbait detection [dataset](#).
2. Pretrain the LSTM on the full wiki-text-2 corpus we have supplied with the Glove embedding using the LM objective.
3. Use the pretrained LM to build a text classifier for clickbait detection. Measure changes in your validation and test performance for both cases.