

# Codelet 5: Classification with Recurrent Neural Networking using PyTorch

The due date for this codelet is **Wednesday, Mar 5 at 11:59PM**.

## Introduction

The aim of this codelet is to expand on your PyTorch skills and to give you practice using an RNN as an encoder for text classification. You should draw on Chapters 9.5 and 9.6 of the textbook, the PyTorch documentation, and all of your prior codelets.

**Important: Only use PyTorch, numpy, pandas, matplotlib, and in-built Python for this codelet. The use of any other libraries, including the books d2l library results in an automatic unsatisfactory grade for this assignment.** A core goal of this class is to build your competencies as a machine learning engineer. I want to minimize abstractions from other libraries so that you build these skills.

## Outline

- Grading
- Classification with RNNs
  - Data
  - Model
  - Training and Evaluation

## Your assignment

Your task is to:

1. Download `codelet5.zip` from the course website and open it. You will find these instructions, a folder `data` which contains training and development data, `utils.py`, which includes a data class, and `codelet5.py`, which has scaffolding for you.
2. Complete each of the 3 broad tasks below in `codelet5.py` and include a file called `codelet5.pdf` with your answers to the written questions.

## Grading

When assessing your work, satisfactory achievement is demonstrated, in part, by:

- Simple, minimal, and clear code
- Code utilizes PyTorch methods in a way that simplifies your code
- Code passes relevant tests
- Explanation extends beyond repeating the textbook/material online
- Response are concrete and clearly demonstrate an understanding of the concepts

## Classification with RNNs

You will complete an implementation of a recurrent neural network trained using gradient descent on a text classification task. You will build on abstractions from PyTorch, demonstrating you understand the high-level algorithm for tasks like this. You have three tasks, of varying difficulty:

1. Orient yourself to the task and the data
2. Implement a model class
3. Train and evaluate your model

## Data and task

The problem we are trying to solve with this codelet is Standard American English grammatical judgments. At its core the task is to label a sentence in English as grammatical or ungrammatical. Our data consists of sentences with their label. Two examples are provided below.

- (1) One more pseudo generalization and I'm giving up.
- (2) Bill seems to be obnoxious, but I don't think that Sam turns out.

I suspect you share the judgment that (1) is a grammatical English sentence and (2) is not, whether or not you can articulate exactly why. I also hope you can see how this task is translatable to a classification task using an RNN as an encoder. Our aim is to build a representation of the whole sentence and then label it as 1 (grammatical) or 0 (ungrammatical).

Some initial preprocessing has been handled, including lowercasing the text, separating punctuation, and handling contractions. Additionally, I have applied a simple tokenizer to this text with the output in the final column in the text. Additionally, I provide a function that returns PyTorch dataloaders for training and validation data. Dataloaders handle the shuffling and batching.

Answer the following questions in the pdf accompanying your code:

1. What is the numerical representation of “professor” and “flowers”?
2. Change the batch size to 2 at the top of the codelet and print the first batch of the valid data. What shape does the input have? What is the meaning of each dimension in the input? What shape does the output have? What does 0 represent in the input and why has it been added?

## Recurrent Neural Network Encoder Model

Some scaffolding for your class is provided. This is the more challenging part of this codelet, so leave time to dig into this. Make use of discord, office hours, and TA hours if you get stuck (e.g., you spend more than 1 hours on a single issue without making any progress). Your task is to complete the `__init__` and `forward` methods. Notice that a variable `VOCAB_SIZE` is provided at the top of the script. Your network should meet the following specifications:

- The model should use `torch.nn.Embedding` to map the tokenized input to embeddings which are then the input to the network.
- The model should use `torch.nn.RNN` with the ReLU activation function. Notice that you should be flexible to different embedding sizes, number of hidden nodes, and number of layers. **Important: You should add `batch_first=True`.**
- You should map the representation of the whole sentence to a single output which you apply a Sigmoid activation function to.

Note you can use the default initialization strategies for the embeddings, the RNN, and the mapping to classification label. I've provided a method `init_hidden` which you can use to create the initial hidden representation you need to call `forward`.

Evidence of your completion of this task comes from successfully training your model in a later task.

## Training and Evaluation

No train, validation loss, or evaluation function is provided. Your aim is to train a 2 layer RNN with an embedding size of 100 and 50 hidden nodes per layer on this task using mini-batch gradient descent. It is helpful to begin with evaluating a untrained model's **accuracy** on this task. You should assume that an output greater than 0.5 corresponds to a prediction of grammatical (1), otherwise the prediction is ungrammatical (0). This function should avoid calculating the gradients.

To evidence completion of this task, and to check your own progress, you should report the accuracy of your initial model before training. You should add to the pdf accompanying your code a screenshot of the output of your code. Mine, for example, looks as follows (notice different random seeds greatly affect this number):

Initial Accuracy: 42.69%

This task is a hard one for an RNN to learn. In real settings, we would first pre-train a model on a language modeling task, and then finetune it on this classification task. In the context of this assignment, you should train your model for 5 epochs, reporting the average loss over the batches in an epoch, as well as the validation loss after each epoch. You should also report your final accuracy. You should identify the cost function that is appropriate for this task. You can and should draw heavily on earlier codelets.

To evidence completion of this task, and to check your own progress, you should report the by-epoch average training and validation loss and the final accuracy of your model. You should add to the pdf accompanying your code a screenshot of the output of your code. Mine, for example, looks as follows:

```
Epoch 1 Avg Train Loss: 0.6121307084726733 Valid Loss: 0.6300654411315918
Epoch 2 Avg Train Loss: 0.605087642752847 Valid Loss: 0.6465974946816763
Epoch 3 Avg Train Loss: 0.6024497300386429 Valid Loss: 0.627836654583613
Epoch 4 Avg Train Loss: 0.5953284900548846 Valid Loss: 0.6327693462371826
Epoch 5 Avg Train Loss: 0.5889056172481802 Valid Loss: 0.6408858895301819
Final Accuracy: 66.64%
```