HW3P1 Bonus

GRU Language model with CTC Loss

11-785: Introduction to Deep Learning (Spring 2025)

Out: **Not Implemented PM EDT**

Due: **Not Implemented EST**

Start Here

Collaboration policy

- You are expected to comply with the University Policy on Academic Integrity and Plagiarism
- You are allowed to talk with / work with other students on homework assignments
- You can share ideas but not code, you must submit your own code. All submitted code will be compared against all code submitted this semester and in previous semesters using MOSS.

Overview

This bonus assignment focuses on training a GRU-based language model using CTC loss. Students will implement a GRU module from scratch and integrate it into the MyTorch framework. The final implementation will be evaluated using an autograder on synthetic data.

Directions

- You are required to do this assignment in the Python (version 3) programming language. Do not use any auto-differentiation toolboxes (PyTorch, TensorFlow, Keras, etc) - you are only permitted and recommended to vectorize your computation using the Numpy library.
- If you haven't done so, use pdb to debug your code effectively.

1 MyTorch

The culmination of all of the Homework Part 1's will be your own custom deep learning library, which we are calling MyTorch. It will act similar to other deep learning libraries like PyTorch or Tensorflow. The files in your homework are structured in such a way that you can easily import and reuse modules of code for your subsequent homework.

For Homework 3 Bonus, MyTorch will have the following structure:

- CTC
  - CTC.py (Copy from HW3P1)
- mytorch
  - nn
    - activation.py (copy from previous HWs)
    - linear.py (copy from previous HWs)
    - loss.py (copy from previous HWs)
  - gru_cell.py (copy from HW3P1)
- models
  - **GRU.py**
- **train_end2end.py**

- **Install** Python3, NumPy and PyTorch in order to run the local autograder on your machine:
  - pip3 install numpy
  - pip3 install torch
- **Autograde** your code by running the following command from the top level directory:
  - NotImplemented
- **Hand-in** your code by running the following command from the top level directory, then SUBMIT the created handin.tar file to autolab:
  - NotImplemented
- **DO**
  - We recommend that you review the lectures covering GRU and CTC Loss.
- **DO NOT**
  - Import any other external libraries other than numpy, as extra packages that do not exist in autolab will cause submission failures. Also do not add, move, or remove any files or change any file names.

2 GRU [10 points]

Overview You've implemented GRUCell class in HW3P1 and now we look at integrating it in a way that it becomes trainable as a configurable GRU model, similar to the GRU module from PyTorch. As part of GRU.py file, you will implement the GRU class with configurable input and hidden sizes as well as number of layers.

GRU class strings together multiple GRUCell instances together as per the number of layers specified. The overarching GRU class, to perform its forward and backward functions, orchestrates passage of appropriate values from the individual GRUCell instances and in the right order.

Through the

2.1 Initialization

As part of the *__init__* method in the GRU class, we store the configuration parameter of the GRU model; this includes input_size, hidden_size and the num_layers parameter. Each GRUCell represents one layer and these layers are stacked on top of each other. We create a list to store these layers, in the right order. Each GRUCell gets instatiated with the right sizes and appended to this list.

2.2 Forward Implementation

Forward function, similar to the GRUCell, takes in an input list and the previous activation state as its inputs. It then creates a zero array for the initial previous activation state and then runs for each layer. Within each layer, it forwards each batch through the layer and gathers all gradients in a cache and prepares the output to be used as the input for the next layer. Finally, the last output becomes the final output of the model.

2.3 Backward Implementation

Backward function will now go through the layers in the opposite order of forward i.e. top to bottom. Each layer gave out two outputs, one to the next layer and one to the next cell in the same layer. We again initialize the gradient for the last timestep cells as zeros and use the gradient from the parts of model after the final layer as inputs for the top layer's last timestep and backpropogate across all timesteps in the top layer and then go downwards across all layers as well.

3 End-to-End Training Pipeline (train_end2end.py) [10 points]

After implementing the GRU model and its integration with CTC loss, the next step is to construct an end-to-end training pipeline that brings everything together.

3.1 Initialize the GRU-CTC Model

- The model consists of a stacked GRU network followed by a linear layer that maps hidden representations to output symbols.

- It is trained using Connectionist Temporal Classification (CTC) loss, which was implemented in HW3P1.

3.2 Load & Preprocess Data

- Input sequences are fed as batched sequences of shape `(seq_len, batch_size, input_size)`, where:
  - `seq_len` = Number of time steps
  - `batch_size` = Number of parallel sequences
  - `input_size` = Feature dimension per time step

3.3 Training Loop

- The pipeline iterates over multiple epochs, performing:
  - Forward pass
  - CTC Loss computation
  - Backward pass
- The gradients are propagated back through time.

Performance Monitoring

- Loss values are printed to track model convergence.
- The autograder will test your loss with random synthetic data, requiring it to be at or below a threshold of _ in order to achieve full points.
- Feel free to experiment with other hyperparameters, although this may not be needed.

4 Next steps

Equipped with this overview of the two tasks, you can now refer to the code comments to complete this challenge, best of luck!