CS589: Machine Learning - Fall 2025

Homework 3: AutoDiff, Deep Learning and LLMs

Assigned: Friday, October 10th, 2025

Getting Started: This assignment consists of coding problems. Download the assignment archive from Canvas and unzip the file. Starter code for select coding problems is provided in the code directory. For general clarification questions, please submit public posts to Campuswire. For questions related to your specific solutions, please submit private posts on Campuswire. In-person help is available through regularly scheduled office hours.

Due Date and Late Work: The assignment is due at 8:00pm ET on October 16th, 2025. Students can submit up to 11:59pm on the due date with no penalty. Work submitted up to 11:59pm on one day after the assignment is due is subject to a penalty of 10%. Work submitted up to 11:59pm two days after the assignment is due is subject to a penalty of 20%. Gradescope will close at 11:59pm two days after the assignment is due and work can not be submitted for credit after this point.

How to Submit: Your written report must be submitted to Gradescope as a PDF file. You must select the page on which each answer appears. You are encouraged to typeset your PDF solutions using LaTeX. The source of this assignment is provided to help you get started. You may also submit a PDF containing scans of *clear* hand-written solutions. Work that is illegible will not count for credit. For this assignment, code should be submitted to Gradescope as Python 3.10+ Jupyter Notebooks. Autograding will not be used for this assignment. You may submit both the code and the report as many times as you like before Gradescope closes. Only your final submission will be graded. Any late penalties will be based on the timestamp of your final submission as determined by Gradescope.

Academic Honesty Reminder: Homework assignments are individual work. Being in possession of another student's solutions, code, code output, or plots/graphs for any reason is considered cheating. Sharing your solutions, code, code output, or plots/graphs with other students for any reason is considered cheating. Copying solutions from external sources (books, web pages, etc.) is considered cheating. Collaboration indistinguishable from copying is considered cheating. Posting your code to public repositories like GitHub (during or after the course) is not allowed. Manual and algorithmic cheating detection are used in this class. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

Generative AI Use Reminder: The use of generative AI tools to help with coding is permitted for programming problems in this course. The use of generative AI for all other work is considered cheating.

1. (*20 points*) **Learning Probability Mass Functions with Symbolic Differentiation.** Recall the problem of learning probability mass functions from HW02. In this problem we have a binary random variable X taking values in $\mathcal{X} = \{0, 1\}$. Let the probability mass function (PMF) for X be given by $P(X = 0) = \pi_0$ and $P(X = 1) = \pi_1$. The negative average log likelihood function for this problem is given below. We will explore deriving the learning rule for this model using SymPy in this question (see the *Symbolic Differentiation with SymPy* demo notebook for example use of SymPy).

$$nall(\pi, \mathcal{D}) = -\frac{1}{N_{tr}} \sum_{n=1}^{N_{tr}} (\mathbb{I}(x_n = 1) \log \pi + \mathbb{I}(x_n = 0) \log(1 - \pi))$$

a. (*5 pts*) Define N_0 to be the number of 0's in the data set and N_1 to be the number of 1's. Show that the objective function can be re-written as:

$$nall(\pi, \mathcal{D}) = -\frac{1}{N_{tr}} (N_1 \log \pi + N_0 \log(1 - \pi))$$

b. (*5 pts*)Use SymPy to implement the form of the nall function in terms of N_1 and N_0. Include your SymPy code in your report along with SymPy's latexed representation of the function as the answer to this question.

c. (*5 pts*) Now use SymPy to compute and simplify the derivative of the nall with respect to the parameters π. Include your SymPy code in your report along with SymPy's latexed representation of the function as the answer to this question.

d. (*5 pts*) Next, use SymPy to solve the stationary equation. Include your SymPy code in your report along with SymPy's latexed representation of the solution to the stationary equation.

2. (*15 points*) **Implementing Numerical Differentiation in PyTorch.** Implement a PyTorch function `approx_fprime` for performing numerical differentiation. Your function should take two arguments: a PyTorch function f that accepts a tensor of shape (D,) as inputs, and a vector x of shape (D,). The function should implement the numerical approximation to the gradient of f evaluated at x as described in Handout 8, slide 9. Your function should work for any value of D.

a. (*5 pts*) As your answer to this question, provide the implementation of your `approx_fprime` function in your report.

b. (*5 pts*) To validate your `approx_fprime` implementation, implement any differentiable function you like in PyTorch that takes at least a two dimensional input x to use as a test function. As your answer to this question, provide the code for your test function and explain why its a good choice to test your `approx_fprime` implementation.

c. (*5 pts*) Now, pick three values of x. For each value of x, report the value of x, the value of the numerical approximation to the gradient at x that you find using your `approx_fprime` implementation, the value of gradient at x that you find using PyTorch's autodiff implementation applied to your function, and the absolute different between the numerical and autodiff gradients vectors.

3. (*15 points*) **Optimizing Logistic Regression.** In this question you will experiment with learning the logistic regression model. The starter code for this question is in `lr.ipynb` and the data are in `data.pt`. The task is binary classification. This data set is linearly separable so the minimum training error is 0. However, standard steepest descent struggles to learn the model parameters.

a. (*5 pts*) To begin, try learning the model with the learning rate `1e-5, 1e-4, 1e-3, 1e-2` for 100 iterations each. When the model is fit, it will return the negative average log likelihood objective function value for each learning iteration. A plot of the objective function value vs iteration is called a "learning curve" and is an important tool for understanding how learning is converging. Make a single plot showing the learning curve for each learning rate. Make sure to include a legend. Include the plot in your report along with a table showing the training error rate for each of the learned models. Show the training error rate values to four decimal places. Include the code for this experiment in the `lr.ipynb` notebook.

b. (*5 pts*) Now, explore different methods to improve learning for this model. You may try any of the enhancements described in Handout 7, including different optimizers. As your answer to this question, describe the approach you found that worked the best. Show a learning curve for that approach and report the final training error found in your report. Include the code needed to reproduce your results in the `lr.ipynb` notebook (if you change the model definition, make a separate copy of the model so your notebook will still produce the correct output for you answer to part (a)).

c. (*5 pts*) Explain what problem you think the data have that make the model difficult to learn and what experiments you conducted to try to verify your hypothesis. Include any results you produced in your report and any code you wrote to help answer this question in the `lr.ipynb` notebook.

4. (*50 points*) **Exploring LLMs**. In this question, you will experiment with some of the internal structures of the GPT-2 model and perform fine-tuning. This model is a precursor to ChatGPT. The model version we will use is the "small" variant with 124M parameters. This model should be runnable on a system with 8gb of ram. If you have difficulty running it locally, you can run it on Google Colab.[1] This model is a decoder only transformer. It has 12 layers of multi-head attention transformer blocks. The model provides several outputs of interest. The first output is the attention scores. The attention scores are given as a list of PyTorch tensors. There is one tensor per transformer block. The attention tensor at index 11 holds the multi-head attention scores for the last transformer block. There are 12 attention heads. When given a single sentence of length L as input, each multi-head attention tensor **A** has shape $(1, 12, L, L)$. The value $\mathbf{A}[0, h, i, j]$ represents the attention from the word in position i to the word in position j for attention head h. The second output of interest is a tensor of unnormalized log space scores (called "logits") **S** of shape $(1, L, V)$ when L is the input sentence length and V is the vocabulary size. To produce actual next token probabilities for the last word in the sentence, the scores need to be passed through the softmax transform as shown below. Note that this model is trained on next word prediction only. It has had no finetuning applied. Use this information to answer the following questions. The starter code for this question is contained in the file `gpt2.ipynb`.

$$P(Y = w|\mathbf{S}) = \frac{\exp(\mathbf{S}[0, L-1, w])}{\sum_{w'=0}^{V-1} \exp(\mathbf{S}[0, L-1, w'])}$$

[1]A GPU can be used, but is not needed. Colab my enforce time limits on free use. The total compute time needed to run all experiments for this question is less than 1 minute. If using Colab, limiting total use time to 1 hour is advisable.

a. (*7 pts*) Consider the two prompts given in the notebook. For each prompt, write code to apply the model to obtain the next token prediction scores (use the provided `predict` function), apply the softmax transform to convert to token probabilities, and find the 10 highest probability tokens. Make a bar chart with one bar per token showing the token probability. Label the bars with the text form of the token (decode the token id to text using the provided `token_ids_to_text` function). Sort the bars from highest to lowest probability. Provide the two bar charts in your report.

b. (*5 pts*) Consider the results from part (a). Based on the prompts, is the model providing sensible outputs?

c. (*8 pts*) Consider again the two prompts given in the notebook. Write code to apply the model to obtain the attention scores (use the provided `predict` function), extract the attention scores between the last token and all other tokens in the last transformer block (layer index 11). Average the attention scores over the 12 attention heads. Now create a bar chart with one word per input token (you can convert each input sentence to a list of tokens using the provided `tokenize` function. Include the bar charts in your report.

d. (*5 pts*) Consider the results from part (c). Based on these results. What does the model seems to be attending to? What differences do you see in the average attention scores for the two prompts?

e. (*10 pts*) Using the provided `predict`, `token_ids_to_text` functions, implement autoregressive generation for the model. To do this, you need to write a loop that will obtain the next word predictions for the current sentence, find the most likely token, decode the token index into text form, append it to the sentence to obtain a new, extended sentence, and continue to the next iteration of the loop. Your generation function should return both the final sentence and a list of probabilities of each word generated. As your answer to this question, run the generation process on the two input sentences until the first period token ('.') is output, or 100 words are generated.

f. (*10 pts*) Lastly, pick the statement of a fact that the GPT-2 model currently completes incorrectly. The statement needs to have a single token answer. For example, the statement "The capital of Paris is" should be completed with the token "Paris" (token id 40313), but the most likely completion is "the." Determine the token id for the completion of your fact using `tokenizer.encode`. We'll call this the target id. You are now going to fine tune the model to correctly complete your fact by assigning higher probability to your target id than to any other token. To do this, use the prompt for your fact as the input to `predict`. Compute the probability assigned to your target token under the next word prediction. As the objective function to train the model, use the negative log likelihood of your target token id. To update the model weights, follow the PyTorch optimization approach shown for the logistic regression model in `lr.ipynb`. Run a sufficient number of iterations of training to make your target token the most likely output for your prompt. Once the model is learned, run most likely next word prediction to verify that the correct completion of your fact is returned. As your output for this question, make a graph of the learning curve for this training task.

g. (*5 pts*) With the fine-tuned model in hand, check other prompts. How has fine tuning the model affected the model's output?

5. (*0 points*) If you used generative AI tools to help complete any programming questions on this assignment, please briefly describe which tools you used, how you used them, and for which problems you used them. If you did not use generative AI tools, please indicate that as your response to this question.