Deep Learning / Spring 2025

Homework 1

Please upload your assignments on or before Feb 20, 2025.

- You are encouraged to discuss ideas with each other, or to consult online tools (such as LLMs). But you **must acknowledge** who you collaborated with or which tools you used, and you **must compose your own** writeup and code independently. Answers should be self-contained and must not appear to be generated by an LLM. We will not grade generic responses.

- We **require** answers to theory questions typeset. Handwritten homework submissions will not be graded.
- We **require** answers to coding questions in the form of a Jupyter notebook. Within each notebook, it is **necessary** to include brief, coherent explanations of both your code and your results to show us your understanding. Use the text block feature of Jupyter notebooks to include explanations.
- Upload both your theory and coding answers in the form of a **single PDF** on **Gradescope**.

---

1. **(4 points)** *Expressivity of neural networks*. Recall that the functional form for a single neuron is given by $y = \sigma(\langle w, x\rangle + b)$, where x is the input and y is the output. In this exercise, assume that x and y are 1-dimensional (i.e., they are both just real-valued scalars) and σ is the **step** activation: $\sigma(u) = 1$ for $u > 0$ and 0 otherwise. We will use multiple layers of such neurons to approximate pretty much any function f. There is no learning/training required for this problem; you should be able to guess/derive the weights and biases of the networks by hand.

    a. (1pt) A *box* function with height h and width δ is the function $f(x) = h$ for $0 < x < \delta$ and 0 otherwise. Show that a simple neural network with 2 hidden neurons with step activations can realize this function. Draw this network and identify all the weights and biases. Assume that the output neuron only sums up inputs and does not have a nonlinearity. Explain your reasoning clearly why your network recreates the box function, and list any assumptions you made in your reasoning.

    b. (2pts) Now suppose that f is *any arbitrary, smooth, bounded* function defined over an interval $[-B, B]$. (You can ignore what happens to the function outside this interval, or just assume it is zero). Use part a to show that this function can be closely approximated by a neural network with a hidden layer of neurons. You don't need a rigorous mathematical proof here; a handwavy argument or even a sketched figure is okay here, as long as you convey to us the right intuition and an explanation of your thinking.

    c. (1pt) Do you think the argument in part b can be extended to the case of d-dimensional inputs? (i.e., where the input x is a vector – think of it as an image, or text query, etc). If yes, comment on potential practical issues (for example, exponential growth in network size). If not, explain why not.

2. **(4 points)** *Choosing the right scale of initialization*. In this exercise, you will analyze a single fully-connected layer where each weight is drawn from a Gaussian distribution. Your goal is to derive constraints on the variance of these weights so that the signal neither vanishes nor explodes as it propagates through layers.

    a. Consider the forward pass through a fully-connected linear layer with n_{in} input inputs and n_{out} outputs. Let the weights be initialized as $N(0, \sigma^2)$ and assume that the inputs x_i to the layer are independent, zero-mean random variables with variance $\text{var}(x_i) = v^2$. Biases are set to zero. If a_j is the activation of the j^{th} neuron, provide a step-by-step derivation to calculate $\text{var}(a_j)$ in terms of σ^2, v^2, and n_{in}.

    b. Explain why, if σ^2 is not chosen appropriately, the variance of a_j could either vanish (becoming too small) or explode (becoming too large) as a function of n_{in}. Use this explanation to get a rough thumb rule on how σ^2 should be chosen depending on the number of inputs (also called the *fan-in*.)

    c. Now consider the alternate situation where each output neuron is equipped with the ReLU activation function. Your answer to the thumb rule question in part b would be essentially the same, but scaled by a constant factor. What is this factor? Give a short argument.

d. As we discuss in class, during the backward pass the gradients flowing back to the input neurons can be calculated as a matrix-vector multiplication with the transpose of the weight matrix W times the gradients at the output neurons. Assuming the output gradients are independent and identically distributed with variance g^2, what is the thumb rule to make gradients neither vanish nor explode?

e. So for *both* forward and backward passes to not vanish or explode, what would be a good strategy to pick the initialization variance? (*If you did your calculations correctly, you will arrive at a rule which is sometimes called the Xavier/Glorot initialization.*)

3. **(3 points)** *Improving the FashionMNIST classifier.* In the first demo (link), we trained a simple logistic regression model to classify MNIST digits. Repeat the same experiment, but now use a (dense) neural network with three (3) hidden layers with 256, 128, and 64 neurons respectively, all with ReLU activations. Display train- and test-loss curves, and report test accuracies of your final model. You may have to tweak the total number of training epochs to get reasonable accuracy. Finally, draw any 3 random image samples from the test dataset, visualize the predicted class probabilities for each sample, and comment on what you can observe from these plots.

4. **(4 points)** *Implementing back-propagation in Python from scratch.* Open the (incomplete) Jupyter notebook provided as an attachment to this homework and complete the missing items. In the second demo, we worked with autodiff. Autodiff enables us to implicitly store how to calculate the gradient when we call backward. We implemented some basic operations (addition, multiplication, power, and ReLU). In this homework problem, you will implement backprop for more complicated operations directly. Instead of using autodiff, you will manually compute the gradient of the loss function for each parameter.