CSE 151B Deep Learning
UCSD, Spring 2025

Homework 2

Due 11:59 p.m, May 6th, 2025

Please submit your written solutions as a single PDF file using the provided LaTeX template on the course website. (`https://sites.google.com/view/cse151b-251b`). You can use Overleaf (`https://www.overleaf.com/`) to compile LaTex files online. For problems requiring code, additionally submit all necessary code in one zip file. Code must run and produce the results reported in the PDF for full credit.

1 Automatic Differentiation [3 Points]

Problem A [2 points]: Backpropagation and Weight Initialization

We'll be utilizing the Neural Network Playground to visualize/fit a neural network.

i. [1 points]: Fit the neural network at this link for about 250 iterations, and then do the same for the neural network at this link. Both networks have the same architecture and use ReLU activations. The only difference is how the layer weights were initialized – you can examine the layer weights by hovering over the edges between neurons.

Give a mathematical justification, based on what you know about the backpropagation algorithm and ReLU, for the difference in the performance of the two networks.

ii. [1 points]: Reset the two demos from part i (there is a reset button to the left of the "Run" button), change the activation functions of the neurons to sigmoid instead of ReLU, and train each of them for 4000 iterations.

Explain the differences in the models learned, and the speed at which they were learned, from those of part i in terms of the backpropagation algorithm and the sigmoid function.

Problem B [1 points]: Data Shuffling

When training any model using SGD, it's important to shuffle your data to avoid correlated samples. To illustrate one reason for this that is particularly important for ReLU networks, consider a dataset of 1000 points, 500 of which have positive (+1) labels, and

500 of which have negative (-1) labels. What happens if we train a fully-connected network with ReLU activations using SGD, looping through all the negative examples before any of the positive examples? (Hint: this is called the "dying ReLU" problem.)

2 Convolutional Neural Networks [7 Points]

Problem A [1 points]: Convolution
Consider a single convolutional layer, where your input is a 32×32 pixel, RGB image. In other words, the input is a $32 \times 32 \times 3$ tensor. The convolution is designed to be:

- Size: $5 \times 5 \times 3$
- Filters: 16
- Stride: 1
- No zero-padding

i. [0.5 points]: What is the total number of parameters (weights) in this convolutional layer with a bias term?

ii. [0.5 points]: What is the shape of the output tensor?

Problem B [1 points]: Padding

Consider a convolutional network in which we perform a convolution over each 8×8 patch of a 20×20 input image. It is common to zero-pad input images to allow for convolutions past the edges of the images. An example of zero-padding is shown below: What is one benefit and one drawback to this zero-padding scheme (in contrast to an approach in which we only perform convolutions over patches entirely contained within an image)?

Problem C [2 points]: Pooling

Pooling is a downsampling technique for reducing the dimensionality of a layer's output. Below is an example of max-pooling on a 2-D input space with a 2×2 filter (the max function is applied to 2×2 patches of the input) and a stride of 2 (so that the sampled patches do not overlap):

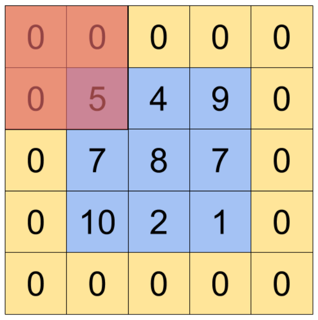
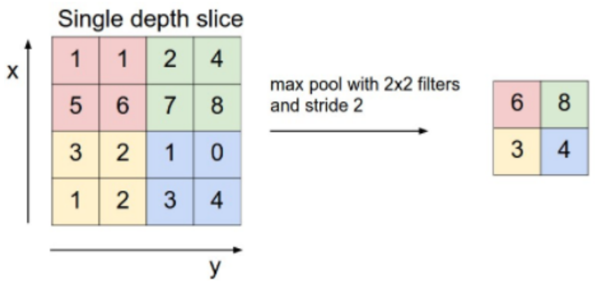


Figure 1: A convolution being applied to a 2×2 patch (the red square) of a 3×3 image that has been zero-padded to allow convolutions past the edges of the image



Average pooling is similar except that you would take the average of each patch as its output instead of the maximum. Consider the following 4 matrices:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

i. [1 points]:

Apply 2×2 average pooling with a stride of 2 to each of the above images.

ii. [1 points]:

Apply 2×2 max pooling with a stride of 2 to each of the above images.

Problem D [3 points]: Pytorch implementation

Using Pytorch's "Sequential" model class, build a deep *convolutional* network to classify the handwritten digits in MNIST. Implement the `CNN` model class in the notebook. You are only allowed to use the following in your model design:

- Linear Layers
- Conv2D
- MaxPool2D
- BatchNorm2D
- Dropout Layers
- ReLU and Softmax

Your task is to build a deep convolutional network that achieves **test accuracy of at least 0.985**. You are required to train the model for 10 epochs with a batch size of 32. Note: your model must have fewer than 1 million parameters, as measured by *'sum(p.numel() for p in model.parameters())'*.

In order to design your model, you should train your model for 1 epoch (batch size 32) and look at the final test accuracy after training. This should take no more than 10 minutes, and should give you an immediate sense for how fast your network converges and how good it is.

Set the probabilities of your dropout layers to 10 equally-spaced values $p = 0, 0.1, \ldots, 0.9$, train for 1 epoch, and report the final model accuracies for each. To guide you, feel free work with the provided Jupyter notebook, titled `HW2_notebook.ipynb`.

i. [1 points]: Implement the training and train your 10 models with different dropout probabilities, and also train the best model for 10 epochs. Turn in your source code and trained model weights to the respective Gradescope submission.

ii. [1 points]: Discuss what you found to be the most effective strategies in designing a convolutional network. Which regularization method was most effective (dropout, layerwise regularization, batch norm)?

iii. [1 points]: Do you foresee any problem with this way of validating our hyperparameters? If so, why?

Hints:

- Start with the Pytorch Tutorial notebook, modify some of the regularization parameters (layerwise regularization strength, dropout probabilities) to see if you can maximize the test accuracy. Add/modify layers so long as the total number of parameters remains under the cap of 1 million.

- You may want to read up on successful convolutional architectures, and emulate some of their design principles. Please cite any idea you use that is not your own.
- To understand the input and output tensor shapes of your model in order to develop better models, you can access each layer in the list *'model.layer'* or iterate over *model.named_modules()*.
- If your model is running slowly on your CPU, try making each layer narrower and stacking more layers so you can leverage deeper representations.