Due 9 PM on February 3rd.

You are free to collaborate on problems 1, 2A, 2B and 3, subject to the collaboration policy stated in the syllabus. There is no collaboration allowed on problems 2C, 2D, 2E.

This set requires the installation of both Tensorflow and Keras. There will be a recitation and office hour dedicated to helping you install these packages if you have problems.

Submission Instructions
Please submit your assignment as a .zip archive with filename LastnameFirstname.zip (replacing Lastname with your last name and Firstname with your first name), containing a PDF of your assignment writeup and your code files. For the ConvNetJS part of problem 3, copy your code from the browser into a text file and include that in your submission.

1 Deep Learning Principles [35 Points]

TA: Siddharth Murching
Relevant Materials: Lectures on Deep Learning

Question A: Backpropagation and Weight Initialization [10 Points]
For this problem, we’ll be utilizing the Tensorflow Playground to visualize/fit a neural network.

i. 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 between the two 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 the ReLU function, for the difference in the performance of the two networks.

ii. 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.

Question B: [10 Points]
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.)

Question C: Approximating Functions [15 Points]

i. Construct (draw or describe) a fully-connected network with ReLU units that implements the OR function on two 0/1-valued inputs, $x_1$ and $x_2$. Your networks should contain the minimum number of hidden units possible. The OR function $\text{OR}(x_1, x_2)$ is defined as:
Your network need only produce the correct output when $x_1 \in \{0, 1\}$ and $x_2 \in \{0, 1\}$ (as described in the examples above).

ii. What is the minimum number of fully-connected layers (with ReLU units) needed to implement an XOR of two 0/1-valued inputs $x_1, x_2$? Recall that the XOR function is defined as:

- $\text{XOR}(1, 0) \geq 1$
- $\text{XOR}(0, 1) \geq 1$
- $\text{XOR}(1, 1) = 0$
- $\text{XOR}(0, 0) = \text{XOR}(1, 1) = 0$

For the purposes of this problem, we say that a network $f$ computes the XOR function if $f(x_1, x_2) = \text{XOR}(x_1, x_2)$ when $x_1 \in \{0, 1\}$ and $x_2 \in \{0, 1\}$ (as described in the examples above).

Explain why a network with fewer layers than the number you specified cannot compute XOR.
2 Depth vs Width on the MNIST Dataset [35 Points]

TA: Suraj Nair

Relevant Materials: Lectures on Deep Learning

Collaboration is not allowed on parts C, D, and E of this problem.

MNIST is a classic dataset in computer vision. It consists of images of handwritten digits (0 - 9) and the correct digit classification. In this problem you will implement a deep network using Keras to classify MNIST digits. Specifically, you will explore what it really means for a network to be “deep”, and how depth vs. width impacts the classification accuracy of a model. You will be allowed at most $N$ hidden units, and will be expected to design and implement a deep network that meets some performance baseline on the MNIST dataset.

Question A: Installation [2 Points]

Before any modeling can begin, Tensorflow and Keras must be installed. Tensorflow is a mathematical optimization framework that is widely used in machine learning. Keras is a python package that provides an easy programming interface for constructing neural networks, and uses Tensorflow under the hood.

To install Tensorflow, follow the steps on https://www.tensorflow.org/get_started/os_setup. We recommend using the Anaconda install instructions if you installed python via the Anaconda distribution, or using the Pip install instructions if you didn’t.

To install Keras, we recommend you simply use pip install keras. For further installation instructions look at https://keras.io/#installation

Once you have finished installing, write down the version numbers for both Tensorflow and Keras that you have installed.

Question B: The Data [3 Points]

Load the MNIST dataset using Keras; see the sample code on the course website for how.

i. Describe the input data. What are the dimensions of the images? What do the values in each array index represent? You can use the imshow function in matplotlib if you’d like to see the actual pictures (see the sample code).

ii. The labels in the data loaded via Keras are values between 0 and 9 corresponding to the digit represented by each image. This is nice and readable for we humans, but to a neural network it makes it seem like an image containing a 9 is somehow to be “bigger” than one containing a 0, and that an image containing an 8 is somehow “closer” to one containing a 9 than one containing a 1.

To convert our dataset to a form suitable for multiclass classification, we use one-hot encoding. This encoding scheme transforms each label $y$ into a length-10 binary vector $v$ where $v[y] = 1, v[i] = 0$ otherwise. For example, the the one-hot encoding of label 3 is the vector $[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$.

Use the keras function keras.np_utils.to_categorical to one-hot encode the the labels of the entire dataset. Also, reshape your inputs to be a single vector instead of a 2 Dimensional image using numpy.reshape. (To use these functions properly, read the keras and/or numpy documentation – reading documentation is the most important software engineering skill!)
What is the new shape of your training input?

**Question C: Modeling Part 1 [10 Points]**
Using Keras’s “Sequential” model class, build a deep network to classify the handwritten digits. You may only use the following layers: Dense (Equivalent to Fully connected), Activation (ReLU (Sets negative weights to 0), softmax (Sets highest weight to 1, rest to 0)), Dropout (Takes some fixed probability and sets weights to zero with that probability - effectively regularization) A sample network with 25 hidden units is in the sample code file.

Note: Activation, Dropout, and your last Dense(NB_CLASSES) layer do not count toward your hidden unit count.

Use categorical cross entropy as your loss function. There are also a number of optimizers you can use (an optimizer is just a fancier version of SGD), and feel free to play around with them, but RMSprop and Adam are the most popular and will probably work best. You also should find the batch size and number of epochs that gives you the best results (default is batch size = 32, nb_epochs=10.

Look at the sample code to see how to compile and train your model. Keras should make it very easy to tinker with your network architecture.

**Your task.** Using at most 100 hidden units, build a network using only the allowed layers that achieves test accuracy of at least 0.975. Turn in the code of your model as well as the best test accuracy that it achieved.

**Question D: Modeling Part 2 [10 Points]**
Repeat Question C, except now you may use 200 hidden units and must build a model with at least 2 hidden layers that achieves test accuracy of at least 0.98.

**Question E: Modeling Part 3 [10 Points]**
Repeat Question C, except now you may use 1000 hidden units and must build a model with at least 3 hidden layers that achieves test accuracy of at least 0.983.
3 Convolutional Neural Networks [30 Points]

TA: Suraj Nair, Siddharth Murching
Relevant Materials: Lecture on CNNs

**Question A: Zero Padding [5 Points]**
Consider a convolutional network in which we perform a convolution over each $8 \times 8$ patch of a $20 \times 20$ 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:

![Zero-Padding Example](image)

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

What are some benefits/drawbacks of this zero-padding scheme (in contrast to an approach in which we only perform convolutions over patches entirely contained within an image)?

**Question B: 5 x 5 Convolutions [5 Points]**
Consider a single convolutional layer, where your input is a $32 \times 32$ pixel, RGB image. In other words the input is a $32 \times 32 \times 3$ tensor. Your convolution has:

- Size: $5 \times 5 \times 3$
- Filters: 8
- Stride: 1

i. What is the number of parameters (weights) in this layer, including a bias term?

ii. What is the shape of the output tensor?

**Question C: Max/Average Pooling [10 Points]**
Pooling is a downsampling technique for reducing the dimensionality of a layer’s output. Pooling iterates across patches of an image similarly to a convolution, but pooling and convolutional layers compute
their outputs differently: given a pooling layer \( B \) with preceding layer \( A \), the output of \( B \) is some function (such as the max or average functions) applied to patches of \( A \)'s output.

Below is an example of max-pooling on a 2-D input space with a \( 2 \times 2 \) filter (the max function is applied to \( 2 \times 2 \) patches of the input) and a stride of 2 (so that the sampled patches do not overlap):

Average pooling is similar except you would take the average of each patch as its output instead of the max.

Consider the following 4 matrices:

\[
\begin{pmatrix}
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix},
\begin{pmatrix}
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0
\end{pmatrix},
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0
\end{pmatrix},
\begin{pmatrix}
0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0
\end{pmatrix}
\]

i. Apply \( 2 \times 2 \) average pooling with a stride of 2 to each of the above images.

ii. Apply \( 2 \times 2 \) max pooling with a stride of 2 to each of the above images.

iii. Consider a scenario in which we wish to classify a dataset of images of various animals, taken at various angles/locations and containing small amounts of noise (e.g. some pixels may be missing). Why might pooling be advantageous given these distortions in our dataset?

**Question D: ConvnetJS Question [10 Points]**

For this question, we'll be working with the ConvnetJS web demo for MNIST, which trains simple convolutional networks in-browser and lets you see what the activations of each layer are as the network classifies various images.

i. Follow the link above to load the demo with the default network, which consists of two convolutional/max pool layers (note that in the demo 'pool' always refers to max pooling).

Record the “Classification Loss” of the network after it has trained on around 10000 examples. Also record the total number of parameters in the network; you can find parameter counts in the layer descriptions in the “Network Visualization” section of the demo.
Now modify the network to use only fully-connected layers, with no convolutions or pooling. Do this by changing the code in the “Instantiate a Network and Trainer” section of the demo and clicking “change network”, which will restart training. Keep the optimizer/batch size the same. Fully connected layers can be created with the code snippet below (feel free to change the number of neurons, but stick to ReLU activations):

```javascript
layer_defs.push({type:'fc', num_neurons:20, activation:'relu'});
```

Note that the web demo can be finicky, so we recommend first adding a fully–connected layer at the end of the network and pressing the “change network” button before making whatever other changes you want to the network.

How many parameters do you need with a fully–connected network to achieve classification loss (within 0.1) comparable to that of the CNN after 10000 examples?

ii. Now modify the network to contain a single convolutional layer with pooling as follows:

```javascript
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'conv', sx:5, filters:8, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta',
batch_size:20, l2_decay:0.001});
```

In the “Network Visualization” section, look at the activations of the convolutional layer on a few images of the training set (whiter regions correspond to higher activations). Which regions of each image are highlighted? From this, what can you infer about which image attributes the convolutional layer is trying to detect? Pausing the network training will make it easier to inspect the visualizations.

iii. Extra Credit

Reload the demo. Make two modifications to the original CNN that result in an improved validation accuracy. Describe the motivation/impact of your modifications.