Deep Learning With Images

Background

The MNIST (Modified National Institute of Standards and Technology) database is a database full of handwritten digits, most commonly used for training image processing systems. This database has 60 thousand training and 10 thousand testing images of grayscale 28x28 pixel boxes. It served as a benchmark for neural networks and other models for decades even up until recently when University of Virginia's Department of System and Information Engineering had the best performance of a single convolutional neural network with and error rate of .25%. This prompted researchers to look for other challenges and their search lead to Kuzushiji (cursive japanese), for its obscurity since Japan's modernization of its writing ~150 years ago, and to promote community learning of classical Japanese literature. In this project I'll be using the Kuzushiji (cursive japanese) MNIST data set to examine what deep learning looks like with images.

(Parameter, Parameter, Parameters)

For this section I looked at a few ways to change the models by changing the parameters like how many epochs it ran for, the number of hidden dence layers and neurons inside, and the optimizer. My work time was somewhat limited and gpu cores changed since I was working on Rosie and lower time/core requests went through much easier.

I wasn't able to play around with this as much as I wanted to, but from the simplest mindset. More epochs, layers, neurons = better. 1k epochs vs 100k is debatable, what's more layers if you just change the number of neurons instead, but deeper networks work better for non-linear interactions etc. there is a diminishing returns horizon for each one. The optimizer was probably the quickest way to see what worked and what didn't across all models since increasing any of the other parameters usually resulted in a more accurate model.

The changes to the layers changing the number of parameters is a bit confusing, but it's basically multiplying each time. Filter size * dimensions * feature maps and for multiple convolutions it keeps going, which adds up all at the end.

There were definitely times when the network really freaked out and miscalculated whole rows, but that only happened 2 or 3 times of the multiple runs I did. More commonly I saw one or two rows/columns being misplaced consistently. Yet they still correctly placed them at least half the time.

The CNN performed better overall, especially adding a third convolutional layer. This is probably because cnn are better at reducing parameters but not losing the quality since it can auto detect important features.

Epochs: 10, # Hidden Dense Layers: 3, # of Neurons by Layer: 8,8,10, Optimizer: ‘adam’

Change	ORIGINAL SETUP	Optimizer: ‘adagrad’	Epoch: 100	# Neurons: 16, 16, 10	Accuracy Sparse categorical	Layers: 4 8, 8, 8, 10
Avg Loss	2.68884	3.14627	2.216439	8.69856	4.85400	3.70904
Avg Acc	0.10376	0.10870	.163764	0.44528	4.85400	0.13216

Epochs	# Hidden Dense Layers	# of Neurons by Layer	Optimizer	Accuracy	Avg Loss	Avg Acc
100,000	4	16, 16, 16, 10	‘adam’	‘accuracy’	0.480376	0.849648

It turns out there is a point of diminishing returns, the amount of epochs if I had to guess would start somewhere around 3000. It started strong at an accuracy of 60% within the first 1000 only to hit 80% by epoch 2080 and from there the increase was minimal. From what I saw the last epoch landed at loss: 0.4618 - acc: 0.8542 and the lowest loss/highest accuracy was 0.43755632638931274/0.86325.

Epochs: 5000, MaxPooling2D((2, 2), Dense layers: 16, 16, 10,
Loss: Sparse_categorical_crossentropy

Models	Changes	avg_loss	avg_acc	min loss	max acc
Optimizer: Adam	Convolutional layers: 2	1.003835991 2633896	0.677628148 9804387	0.656713366 5084839	0.80128336
	Convolutional layers: 3	0.875942811 9659424	0.714310183 4580302	0.558588683 6051941	0.82671666
Optimizer: adagrad	Convolutional layers: 2	1.510885224 8191833	0.523429433 234036	1.036683678 6270142	0.63336664
	Convolutional layers: 3	1.216259418 3683395	0.615079033 3934129	0.832511305 809021	0.75315

When it comes to comparing classical ML and this project I’d say I liked the classical ML only because I’m more comfortable with it. Being able to visualize the data with graphs is much easier and overall it’s a little more straightforward to understand. Comparatively DNN and CNN were just a bit harder to wrap my head around. The math didn’t math with my perception of math and it took longer to understand how they worked. However, this project was super interesting and informational too. There’s something about using DNN and CNN that makes the machine learning feel a little more real or close to the stereotype even though it’s a little simpler to set up (sort of). I like both and just like their usages, it’s dependent on the situation.

My little mistake:

Set up:
Hidden Dense Layers: 4
of Neurons by Layer: 16, 16, 16, 10
Optimizer: ‘adam’
Accuracy : ‘accuracy’

Epochs	100	1,000	100	1,000	10,000	10,000	100,000
Avg Loss	2.2357	1.93822	1.47217	1.18281	0.76603	0.713266	0.69721
Avg Acc	0.17879	0.25107	0.42739	0.57949	0.75727	0.77300	0.77905

A mistake I had made during this process was not remaking the model each time, so the second table changing the number of epochs were run consecutively. The original reason why I wanted to do this was because I wanted to know when the point of diminishing returns would happen. However, my rosie time clocked out before I realized it and I had to rerun my notebook which let me find out my mistake. My curiosity and frustration left me to then run the fit for 100,000 epochs again on a fresh start to see where it would end up which is detailed in the table describe in the original paper above.