QData Undergraduate Reading Group (**QURG**)

First Meeting
Friday, October 4th, 2019

Agenda

- Logistics & Planning
- General AI Stuff
- *Neural Networks and Deep Learning* Chapter 1
- Research project standup

Logistics

- Our availability:
  - https://www.when2meet.com/?8181497-hm55s
- Best times:
  - Sunday afternoon
  - Monday evenings (Jeffrey?)
  - Wednesday lunch
  - Friday morning
- Prof. Qi suggests meeting on Friday evenings
  - I’m not sure about this idea...

Logistics

- Rough plan:
  - Read a book chapter and/or paper every week
- What kind of workload are we looking for?
- ~~What about a chapter and a paper?~~
  - Never mind, we'll just do one per week
  - But how do you all feel – books vs. papers?
  - Chollet vs. Goodfellow?
- Anyone else who's interested is welcome to join
  - But we are still going to reserve time to talk about our QData research projects
  - More members also means we each have to present less often :)
- 🍕? 🥯? ☕?!

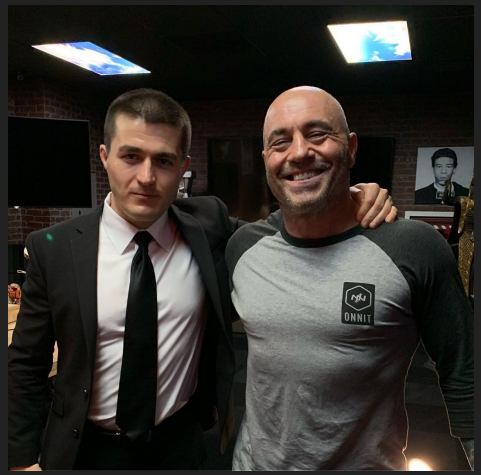
AI Resources

fast.ai

Making neural nets uncool again

arXiv.org

**these are just the things i use, pls suggest more



DLNN Chapter 1

- Brief review of each section
  - Ask questions!
- Work through selected exercises together
- If you didn’t actually write the code, that’s ok

Intro

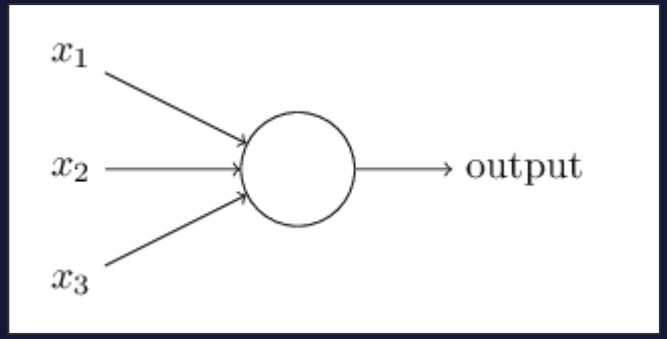
- Neural networks are different from conventional programming– we don’t teach them rules, we establish boundaries which enable them to learn from data
- Until 2006, we couldn’t really train them well
  - The early 2000s (pre-2006) were a dark time for research on neural networks
  - Big breakthrough: “A fast learning algorithm for deep belief nets” (Hinton, 2006)
  - Second big breakthrough: CNNs in 2012
- This book: provides an intuition about neural networks (hopefully)

Using neural nets to recognize handwritten digits

- This is easy to tell a human to do, but hard to teach a computer
- Our visual cortex has *140 million neurons*, interconnected in a complex way
- MNIST

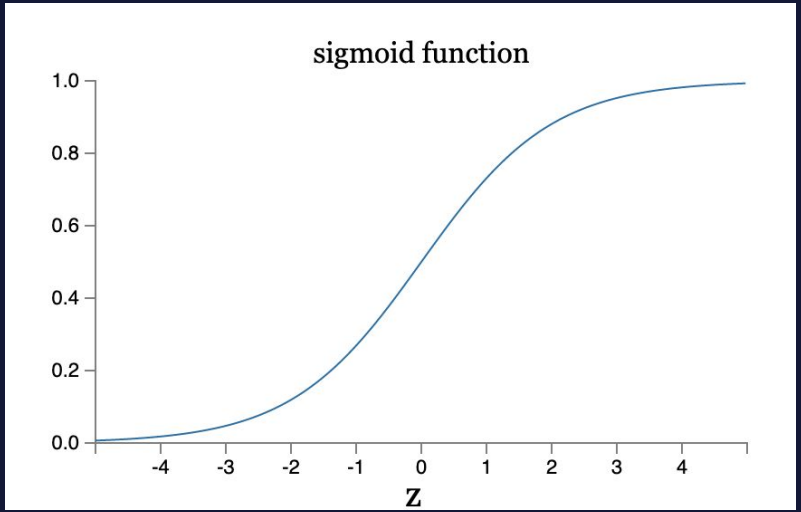
An aside: popular datasets

Perceptrons & Sigmoid neurons



$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$



Perceptrons & Sigmoid neurons

Exercise 1/7

Sigmoid neurons simulating perceptrons, part I
Suppose we take all the weights and biases in a network of perceptrons, and multiply them by a positive constant, c>0. Show that the behaviour of the network doesn't change.

Exercise 2/7

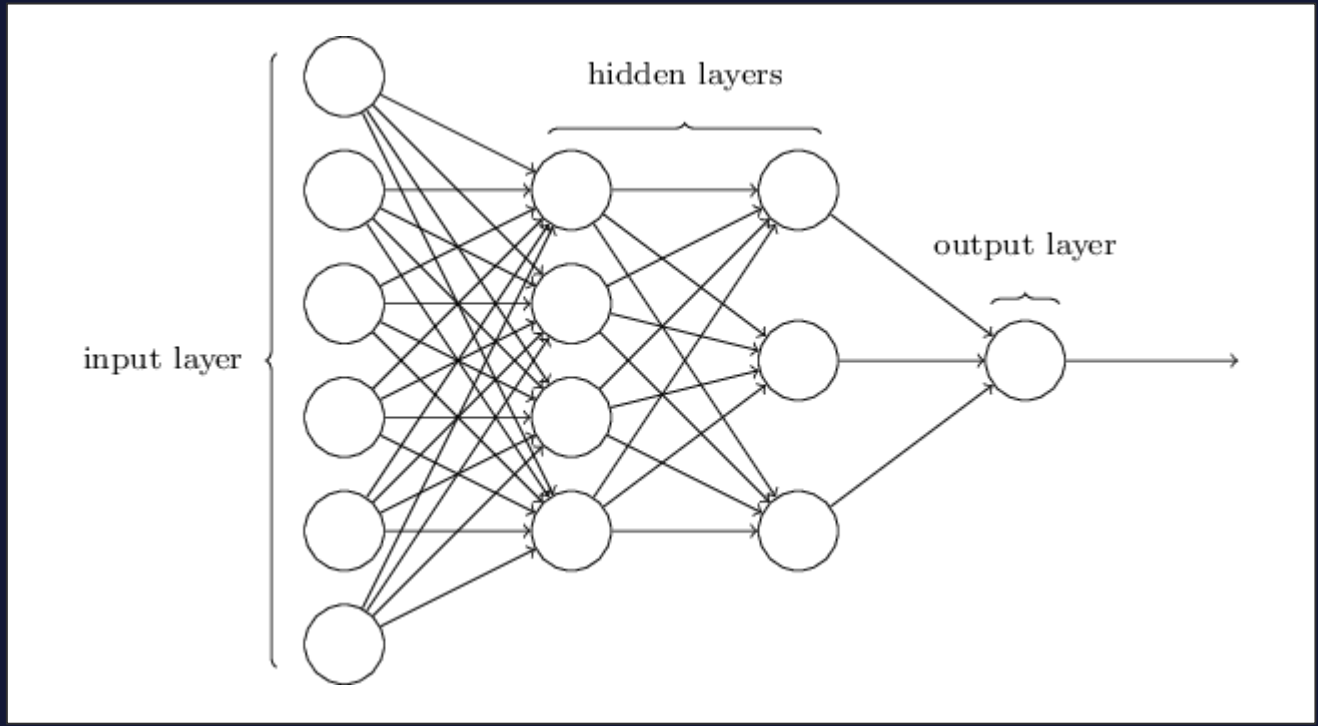
Sigmoid neurons simulating perceptrons, part II

Suppose we have the same setup as the last problem - a network of perceptrons. Suppose also that the overall input to the network of perceptrons has been chosen. We won't need the actual input value, we just need the input to have been fixed. Suppose the weights and biases are such that $w \cdot x+b \neq 0$ for the input x to any particular perceptron in the network.

Now replace all the perceptrons in the network by sigmoid neurons, and multiply the weights and biases by a positive constant $c>0$.

Show that in the limit as $c \to \infty$ the behaviour of this network of sigmoid neurons is exactly the same as the network of perceptrons. How can this fail when $w \cdot x+b=0$ for one of the perceptrons?

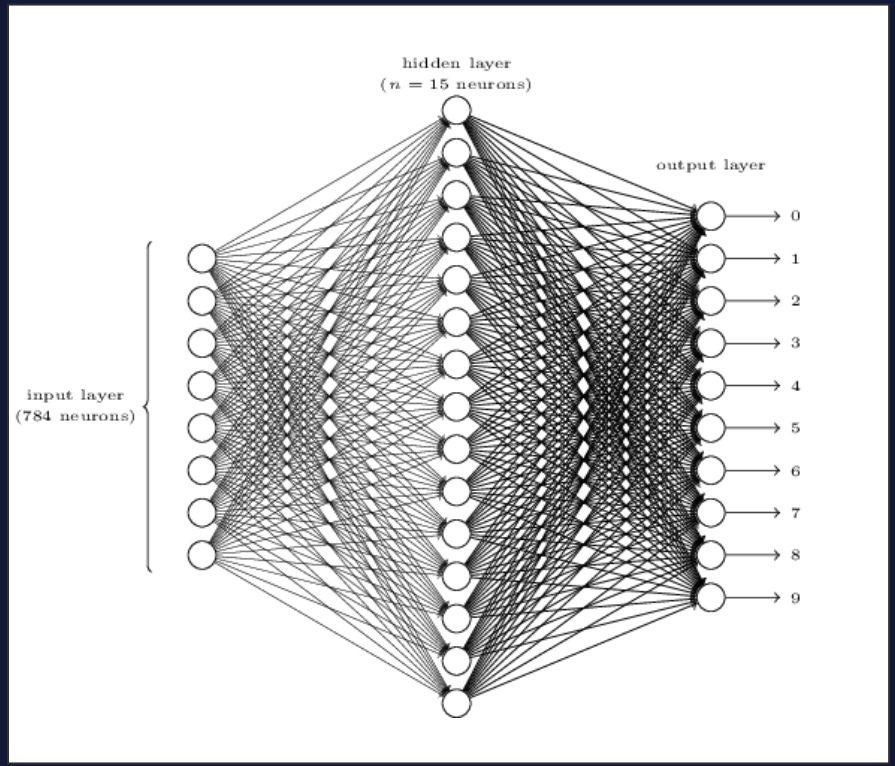
The architecture of neural networks



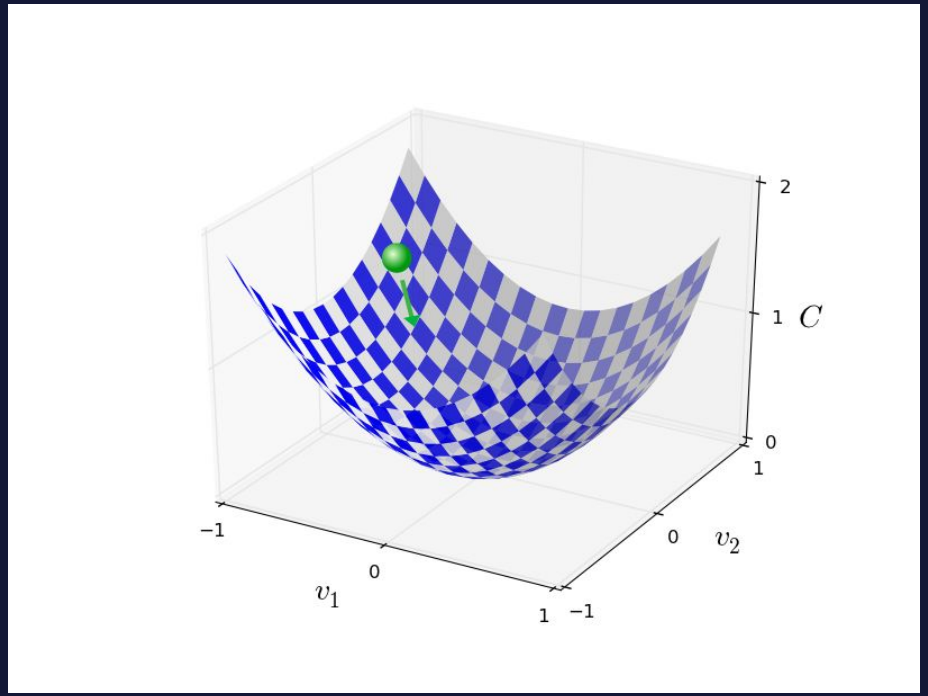
Exercise 3/7

There is a way of determining the bitwise representation of a digit by adding an extra layer to the three-layer network above.

The extra layer converts the output from the previous layer into a binary representation, as illustrated in the figure below. Find a set of weights and biases for the new output layer. Assume that the first 3 layers of neurons are such that the correct output in the third layer (i.e., the old output layer) has activation at least 0.99, and incorrect outputs have activation less than 0.01.



Learning with gradient descent



$$v \to v' = v - \eta \nabla C.$$

(The last paragraph)

Indeed, there's even a sense in which gradient descent is the optimal strategy for searching for a minimum. Let's suppose that we're trying to make a **move Δv in position so as to decrease C as much as possible**. This is equivalent to minimizing $\Delta C \approx \nabla C \cdot \Delta v$. We'll constrain the size of the move so that $\|\Delta v\| = \epsilon$ for some small fixed $\epsilon > 0$. In other words, we want a move that is a small step of a fixed size, and we're trying to find the movement direction which decreases C as much as possible.

It can be proved that the choice of Δv which minimizes $\nabla C \cdot \Delta v$ is $\Delta v = -\eta \nabla C$, where $\eta = \epsilon / \|\nabla C\|$ is determined by the size constraint $\|\Delta v\| = \epsilon$. So gradient descent can be viewed as a way of taking small steps in the direction which does the most to immediately decrease C.

Learning with gradient descent

Exercise 5/7

I explained gradient descent when C is a function of two variables, and when it's a function of more than two variables.

What happens when C is a function of just one variable? Can you provide a geometric interpretation of what gradient descent is doing in the one-dimensional case?

Learning with gradient descent

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Learning with gradient descent

Exercise 6/7

An extreme version of gradient descent is to use a mini-batch size of just 1. That is, given a training input, x, we update our weights and biases according to the rules

$$w_k \to w_k' = w_k - \eta \partial C_x / \partial w_k \quad (1)$$
$$b_l \to b_l' = b_l - \eta \partial C_x / \partial b_l \quad (2)$$

Then we choose another training input, and update the weights and biases again. And so on, repeatedly. This procedure is known as *online* or *incremental* learning. In online learning, a neural network learns from just one training input at a time (just as human beings do). Name one advantage and one disadvantage of online learning, compared to stochastic gradient descent with a mini-batch size of, say, 20.

Implementing the network to classify digits

Bonus Exercise

Try creating a network with just two layers - an input and an output layer, no hidden layer - with 784 and 10 neurons, respectively. Train the network using stochastic gradient descent. What classification accuracy can you achieve?

93.2%

Implementing the network to classify digits

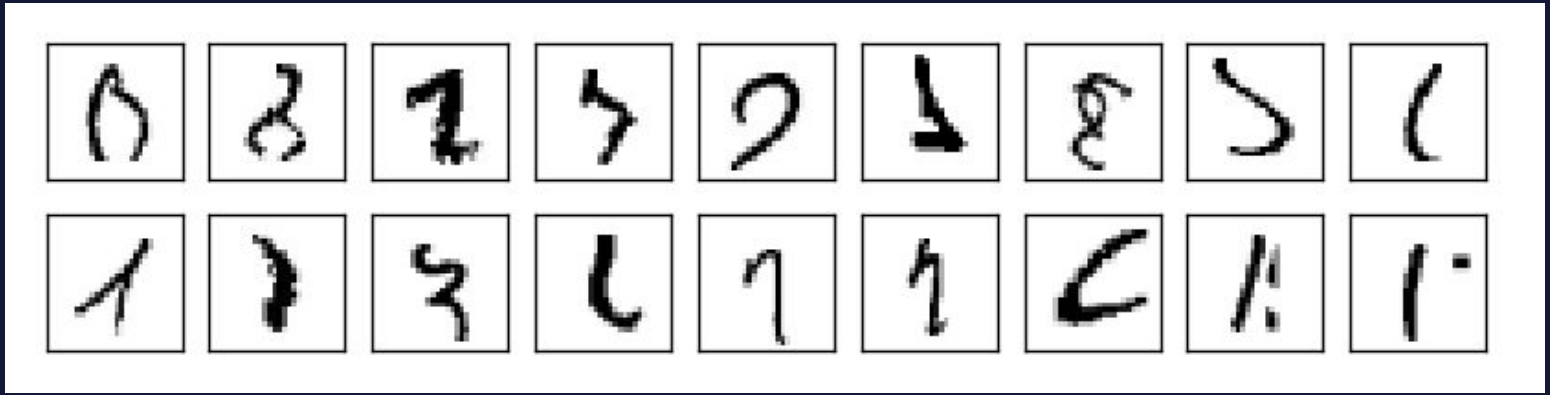
```
# jm8wx 10/4/19
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.model
  tf.keras.layers.Flat
  tf.keras.layers.Dense(1
])

model.compile(optimizer='sgd',
              loss='mean_squared_error',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

model.evaluate(x_test,  y_test, verbose=2)
```

93.2%

Exercise 7/7

Write out Equation (22) in component form, and verify that it gives the same result as the rule (4) for computing the output of a sigmoid neuron.

Implementing the network to classify digits



sophisticated algorithm $\leq$ simple learning algorithm + good training data.

Toward deep learning

- Since 2006, we've been able to train deeper networks on more data
- Training tricks & strategies, network architectures, etc. improve every year
- Advances in hardware match advances in software
- CPUs > GPUs > TPUs

A bit of inspiration

- Two of the most important innovations in deep learning have been 1-line changes to the training process
- **Dropout**: just randomly set 50% of the neurons to 0 every training step
- **Batch normalization**: normalize and shift the inputs to each layer to reduce variance

Project Talk Time