Ryan George
CPSC 663 -- Deep learning
Problem set 1

Problem 1

1. Multilayer perceptron
Subtle decisions, such as determining someone's eye color from a photo.
2. Convolutional Neural Network
Where effects should only happen locally. This might be used to train an algorithm that predicts social sentiments among (sparsely-connected) social networks
3. Recurrent Neural Network
Tasks where the current data impacts the future data. This might include composing original music, where the quality/artistry of the next note depends on the previous notes
4. Autoencoder
Generating a musical score that is supposed to sound like a Beethoven symphony
5. Ultra deep learning
Extremely fine decisions, such as training an algorithm to play Pentago
6. Deep reinforcement learning
Teaching an action in which a clear outcome exists, such as shooting a basketball into a hoop

Problem 2

1. In linear regression, we seek $y = Ax^*$ that minimizes:

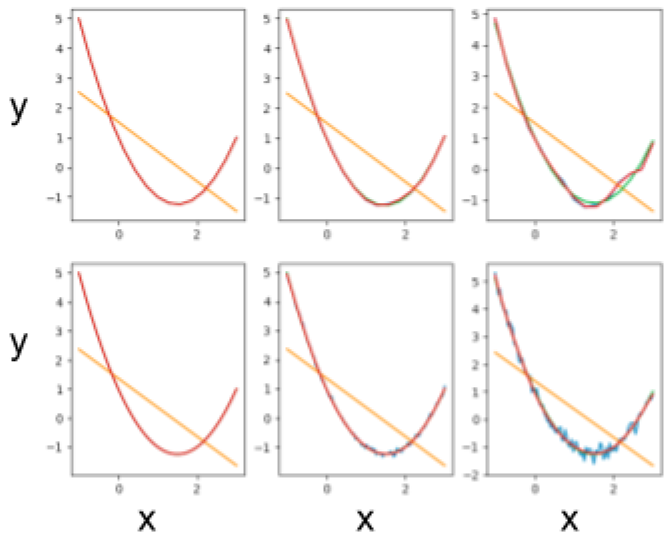
$$
\begin{aligned}
MSE = C(x) = |Ax - y|_2{}^2 = \sum_i (A_i x_i - y_i)^2 &= (Ax-y)^T(Ax-y) \\
&= (Ax)^T(Ax) - y^T Ax - (Ax)^T y + y^T y \\
&= (Ax)^T(Ax) - 2(Ax)^T y + y^T y \quad \text{(since } y^T Ax = (Ax)^T y\text{)} \\
&= x^T A^T (Ax) - 2(Ax)^T y + y^T y
\end{aligned}
$$

In order to minimize MSE, we differentiate by some perturbation to x:

$$
\begin{aligned}
\tfrac{\delta C}{\delta x} &= \tfrac{\delta}{\delta x}\left(x^T A^T(Ax) - 2(Ax)^T y + y^T y\right) \\
&= \tfrac{\delta}{\delta x}\left(x^T A^T (Ax)\right) + \tfrac{\delta}{\delta x}\left(-2(Ax)^T y\right) + \tfrac{\delta}{\delta x}(y^T y) \\
&= 2A^T(Ax) - 2A^T y \\
&= 0,
\end{aligned}
$$

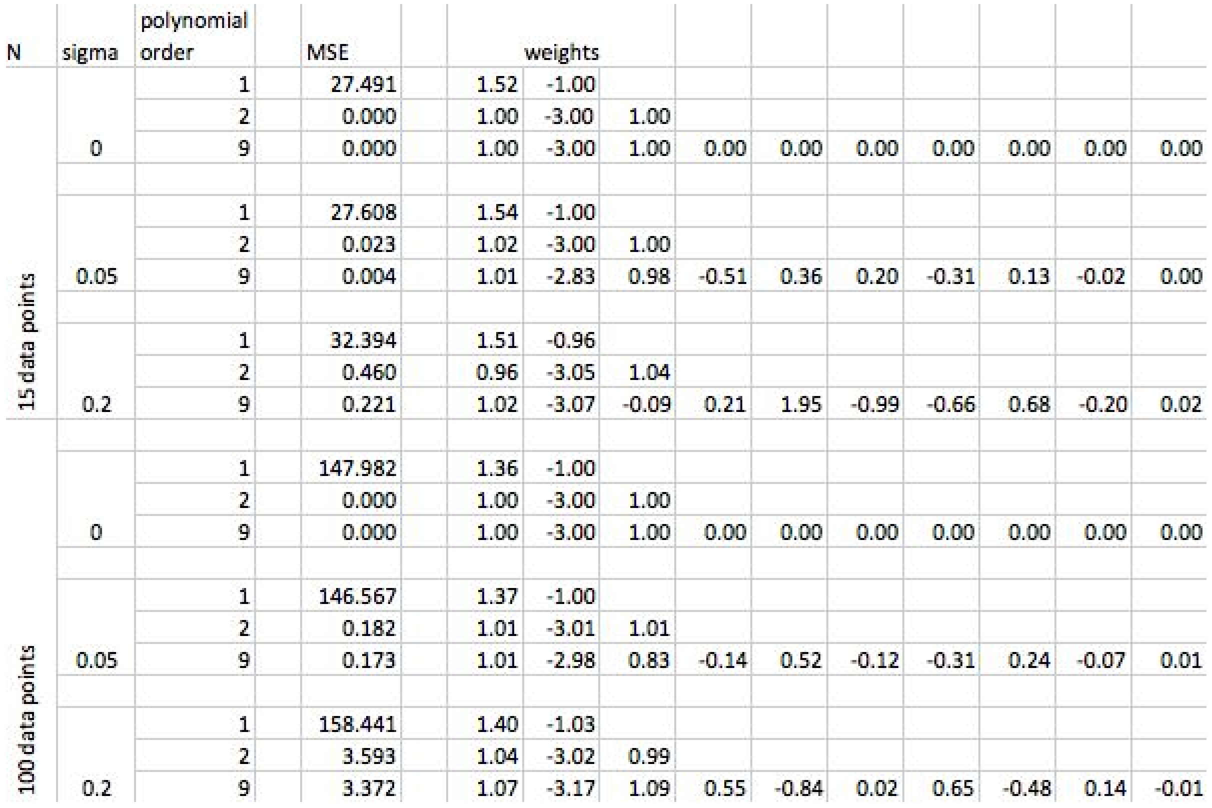
$$\Rightarrow\ 2A^T(Ax^*) = 2A^T y,$$
$$\Rightarrow\ (A^T A)x^* = A^T y$$
$$\Rightarrow\ x^* = (A^T A)^{-1} A^T y$$

Problem 2.
3. The fitted curves look like:



The green line doesn't *appear* to be overfitting the data too badly.

C. The MSE and weights are in this table:

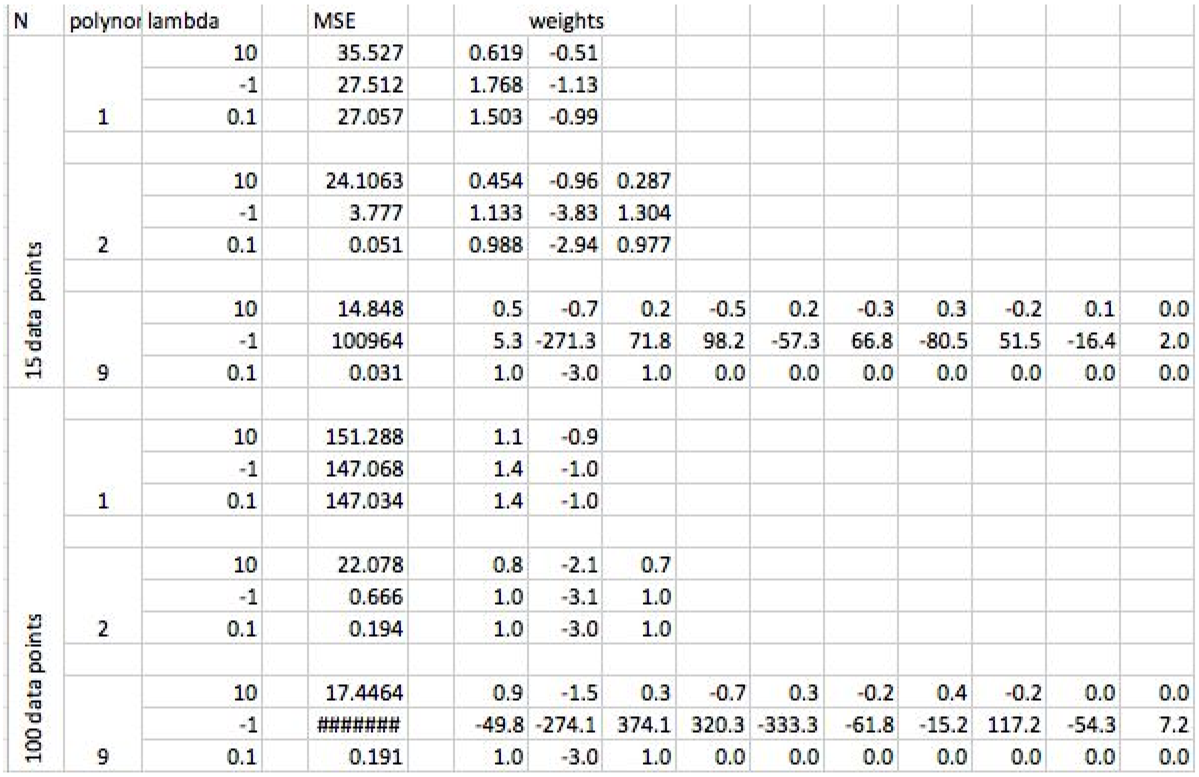


N	sigma	polynomial order	MSE	weights									
15 data points		1	27.491	1.52	-1.00								
		2	0.000	1.00	-3.00	1.00							
	0	9	0.000	1.00	-3.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		1	27.608	1.54	-1.00								
		2	0.023	1.02	-3.00	1.00							
	0.05	9	0.004	1.01	-2.83	0.98	-0.51	0.36	0.20	-0.31	0.13	-0.02	0.00
		1	32.394	1.51	-0.96								
		2	0.460	0.96	-3.05	1.04							
	0.2	9	0.221	1.02	-3.07	-0.09	0.21	1.95	-0.99	-0.66	0.68	-0.20	0.02
100 data points		1	147.982	1.36	-1.00								
		2	0.000	1.00	-3.00	1.00							
	0	9	0.000	1.00	-3.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		1	146.567	1.37	-1.00								
		2	0.182	1.01	-3.01	1.01							
	0.05	9	0.173	1.01	-2.98	0.83	-0.14	0.52	-0.12	-0.31	0.24	-0.07	0.01
		1	158.441	1.40	-1.03								
		2	3.593	1.04	-3.02	0.99							
	0.2	9	3.372	1.07	-3.17	1.09	0.55	-0.84	0.02	0.65	-0.48	0.14	-0.01

We do see that for N=100, sigma=.2, the **2nd and 9th order polynomials do overfit the data** (and get some higher MSE).

4. In order to regularize, we choose different lambda: [10, -1, .1] (too big; too small; just right).

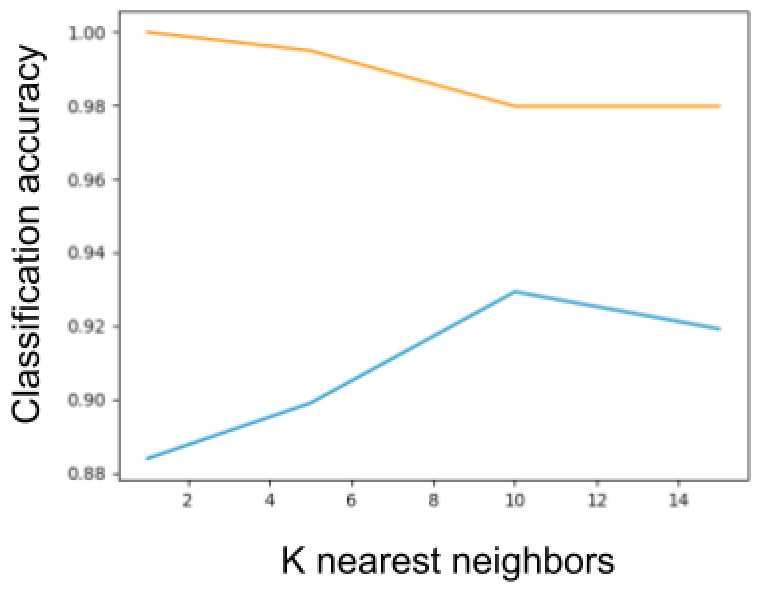
The results are:



N	polynor	lambda	MSE	weights									
		10	35.527	0.619	-0.51								
		-1	27.512	1.768	-1.13								
	1	0.1	27.057	1.503	-0.99								
		10	24.1063	0.454	-0.96	0.287							
		-1	3.777	1.133	-3.83	1.304							
	2	0.1	0.051	0.988	-2.94	0.977							
		10	14.848	0.5	-0.7	0.2	-0.5	0.2	-0.3	0.3	-0.2	0.1	0.0
		-1	100964	5.3	-271.3	71.8	98.2	-57.3	66.8	-80.5	51.5	-16.4	2.0
15 data points	9	0.1	0.031	1.0	-3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		10	151.288	1.1	-0.9								
		-1	147.068	1.4	-1.0								
	1	0.1	147.034	1.4	-1.0								
		10	22.078	0.8	-2.1	0.7							
		-1	0.666	1.0	-3.1	1.0							
	2	0.1	0.194	1.0	-3.0	1.0							
		10	17.4464	0.9	-1.5	0.3	-0.7	0.3	-0.2	0.4	-0.2	0.0	0.0
		-1	#######	-49.8	-274.1	374.1	320.3	-333.3	-61.8	-15.2	117.2	-54.3	7.2
100 data points	9	0.1	0.191	1.0	-3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Problem 3

2. The knn algorithm has the accuracy as follows:



where the orange is leave-one-out and blue is a K-fold with K=5. Clearly K=1 and K=5 overfit the data.

3. For part 3, I implemented a decision tree and support vector machine. Accuracy was 82.8% and 85.6%, respectively.

Problem 4

1. Before we multiply the network of perceptrons by c, at any given node the output is:

$$0,\ if\ w \cdot x + b \le 0$$
$$1,\ if\ w \cdot x + b > 0$$

After we multiply the network of perceptrons by c, the weights are represented by wc, and the biases are represented by wb. So at any given node the output is:

$$0,\ if\ cw \cdot x + cb = c(w \cdot x + b) \le 0$$
$$1,\ if\ cw \cdot x + cb = c(w \cdot x + b) > 0$$

Since $c > 0$, we can divide both inequalities by c without changing them. This gives the behavior to output:

$$0,\ if\ w \cdot x + b \leq 0$$
$$1,\ if\ w \cdot x + b > 0$$

Which is the same as before multiplying.

2. The output of a sigmoid neuron is:

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

If we multiply weights and biases by c, we get:

$$\frac{1}{1+exp\left(-\sum_j cw_j x_j - cb\right)} = \frac{1}{1+exp\left(-c\sum_j w_j x_j - cb\right)}$$

Imagine that $\sum_j w_j x_j - b > 0$. Then as $c \to \infty$, $-c\sum_j w_j x_j - cb \to -\infty$. Then the output of the sigmoid neuron behaves like:

$$\frac{1}{1+exp\left(-c\sum_j w_j x_j - b\right)} \to \frac{1}{1+very\ small\ number} \to 1$$

Similarly, if $\sum_j w_j x_j - b < 0$, then as $c \to \infty$, $-c\sum_j w_j x_j - cb \to \infty$. Then the output of the sigmoid neuron behaves like:

$$\frac{1}{1+exp\left(c\sum_j w_j x_j - b\right)} \to \frac{1}{1+very\ large\ number} \to 0$$

So the behavior is:

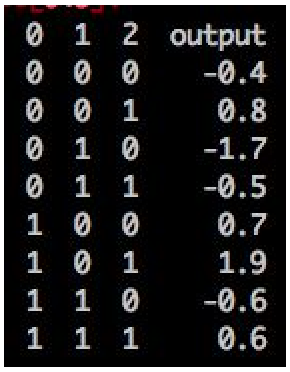
$$0,\ if\ \sum_j w_j x_j - b < 0$$

$$1,\ if\ \sum_j w_j x_j - b > 0$$

Which is exactly the behavior of the perceptron.

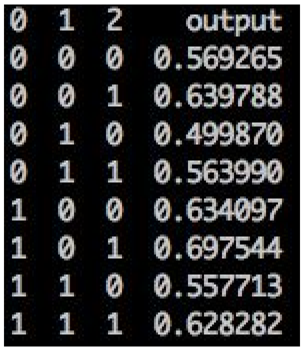
If $\sum_j w_j x_j - b = 0$ for a sigmoid neuron, then the output will be $\frac{1}{1+1} = .5$ regardless of c, which will cause divergence from the behavior of the perceptron network.

3. For the perceptron network, the output is:



```
0 1 2 output
0 0 0   -0.4
0 0 1    0.8
0 1 0   -1.7
0 1 1   -0.5
1 0 0    0.7
1 0 1    1.9
1 1 0   -0.6
1 1 1    0.6
```

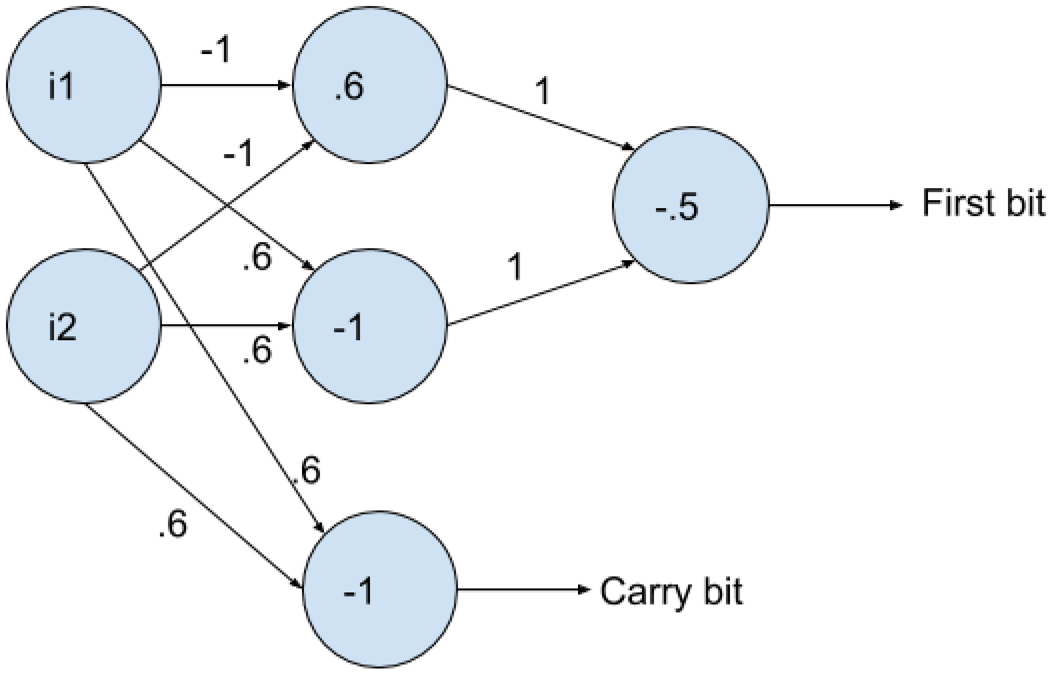
4. For the sigmoid neuron network, the output is:



```
0  1  2    output
0  0  0  0.569265
0  0  1  0.639788
0  1  0  0.499870
0  1  1  0.563990
1  0  0  0.634097
1  0  1  0.697544
1  1  0  0.557713
1  1  1  0.628282
```

Code is in the `prob4.py` file.

5. The adder looks like:



The neurons in the second layer have a very specific purpose. The top neuron fires “1” if both inputs are 0. The middle neuron fires “1” if both inputs are 1. The output from these two decides whether both neurons fired the same bit (in which case the output bit should be zero).
The bottom neuron tests the “AND” function, which determines whether there should be a carry bit.