ID and Names

| | | | |
|---|---|---|---|
| 1. 何文劭<br>2. 林永璿<br>3. 王嘉澤<br>4. 何庭昀<br>5. 李承恩 | 6. 廖為謙<br>7. 蔡宇翔<br>8. 陳銘宏<br>9. 張繁可<br>10. 吳亭慧 | 11. 陳彥禎<br>12. 蘇冠禎<br>13. 張大衛<br>14. 何文劭<br>15. 陳宏慶 | 16. 陳宏慶<br>17. 何庭昀<br>18. 李承恩<br>19.<br>20. 蔡宇翔 |
| 21. 林澤佑<br>22. 張繁可<br>23. 楊喬諳<br>24. 楊喬諳<br>25. 林澤佑 | 26. 楊喬諳<br>27. 何文劭<br>28.<br>29.<br>30. 何庭昀 | 31. 李承恩<br>32.<br>33. 蔡宇翔<br>34. 袁佑緣<br>35. 袁佑緣 | 36. 袁佑緣<br>37. 郭俊廷<br>38. 郭俊廷<br>39. 郭俊廷<br>40. 林澤佑 |
| 41. 陳冠羽<br>42. 陳冠羽<br>43. 陳冠羽<br>44. 陳德禮<br>45. | 46.<br>47.<br>48. 陳德禮<br>49.<br>50. 陳德禮 | 51.<br>52.<br>53.<br>54.<br>55. | 56.<br>57. |

There are 57 questions in the book by Michael Nielsen.

---

Question 1.
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.

Solutions 1. (何文劭)

output = 0 if $w \cdot x + b \leq 0$ $\quad w \cdot x + b \leq 0 \iff c \cdot w \cdot x + c \cdot b \leq 0$ if $c>0$
1 if $w \cdot x + b > 0$ $\quad w \cdot x + b > 0 \iff c \cdot w \cdot x + c \cdot b > 0$ if $c>0$

---

Question 2.

**2.** • **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?

Solutions 2. (林永璿)

Output of sigmoid neurons=σ(z)=σ(w·x+b),
σ((cw)·x+(cb))=σ(c(w·x+b))=σ(cz)
when z>0 σ(cz) tends to 1 as c→∞
when z<0 σ(cz) tends to 0 as c→∞
thus when z≠0, c→∞, the behaviour of this network of sigmoid neurons is exactly the same as the network of perceptrons
However, σ(cz) =0.5 as z=0,independent to c, which is different from the output of an perceptron with z=0( output of an perceptron =0 as z≤0)

---

Question 3.
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 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.

Solutions 3. (王嘉澤)

---

Question 4.

**4.** • Prove the assertion of the last paragraph. *Hint:* If you're not already familiar with the Cauchy-Schwarz inequality, you may find it helpful to familiarize yourself with it.

Solutions 4. (何庭昀)

Question 4

Prove $\Delta V = -\frac{\varepsilon \nabla C}{\|\nabla C\|}$ minimize $\nabla C \cdot \Delta V$ when $\|\Delta V\| = \varepsilon$

Solutions 4

by Cauchy-Schwarz inequality $\|\nabla C\| \cdot \|\Delta V\| \geq \|\nabla C \cdot \Delta V\|$

equality holds if $\Delta V = k \nabla C$ for constant $k$

$\because \|\Delta V\| = \varepsilon$

to minimize $\nabla C \cdot \Delta V$

$\therefore$ pick $k = -\frac{\varepsilon}{\|\nabla C\|}$

$\Rightarrow \Delta V = k \nabla C = -\frac{\varepsilon \nabla C}{\|\nabla C\|}$ minimize $\nabla C \cdot \Delta V$ when $\|\Delta V\| = \varepsilon$

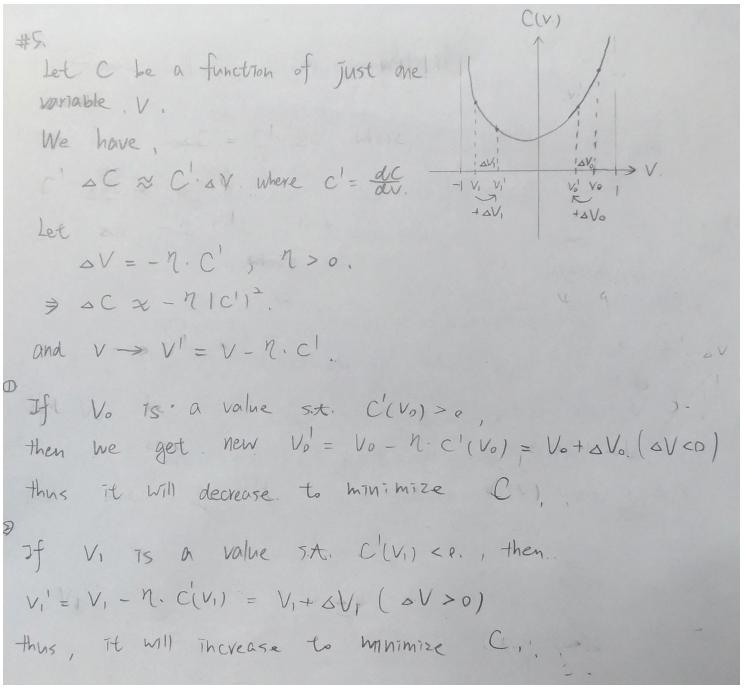
---

Question 5.:

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 ?

Solutions 5. (李承恩)




---

Question 6.

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

$$w_k \to w'_k = w_k - \eta \partial C_x / \partial w_k \text{ and } b_l \to b'_l = b_l - \eta \partial C_x / \partial b_l.$$

Then we choose another training input, and update the weights and biases again. And so on, repeatedly. This procedure is known as online, 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.

Solutions 6. (廖為謙)

- Advantage: It is better than mini-batch method when updating the learning model because online learning only uses one example in each iteration. This is useful when we need to update the model frequently since it use only one sample each time to train the model. This allows us to model problems where you have a continuous stream of data and you want an algorithm to learn from them. This algorithm can adapt to changing user preferences
- Disadvantage: Mini-batch is more stable than online learning on convergence issue because the training data used by mini-batch is fixed while the training data used by online learning updates frequently.

---

Question 7.
Write out Equation (22) in component form, and verify that it gives tha same result as the rule (4) for computing the output of a sigmoid neuron.
Solutions 7. (蔡宇翔)

$$\text{Equation 22: } a' = \sigma(wa + b)$$

$$\text{rule 4: } \frac{1}{1 + exp(-\sum_j w_j x_j - b)} = \sigma(\sum_j w_j x_j + b)$$

$$a' = \sigma(wa + b) = \{(\sigma(\sum_j w_{ij} a_j + b_j))_i\}$$

$w_{ij}$ is the weight from the j-th neuron in previous layer to i-th neuron in current layer.
The summation sums all weights times inputs, so it satisfies rules (4)

---

Question 8.

Try creating a network with just two layers  an input and an output layer, no hidden layer with 784 and 10 neurons,η = 100.0 respectively. Train the network using stochastic gradient descent. What classification accuracy can you achieve?

Solutions 8. (陳銘宏)

When η = 100.0, the best classification accuracy achieved so far is 58.35% when the mini-batch size is 20. When η is not constrained, the best classification accuracy found so far is 91.93% (η=0.875 and mini-batch size=20). Moreover, when the mini-batch size=10, the accuracy can also achieve 91.36% and 91.14% with η=6.25 and 13.125, respectively. However, due to the lack of the hidden layer, the trend of the results is quite unstable. Hence, I believe it should not be easy to reproduce the same results.

---

Question 9.

**Alternate presentation of the equations of backpropagation:** I've stated the equations of backpropagation (notably (BP1) and (BP2)) using the Hadamard product. This presentation may be disconcerting if you're unused to the Hadamard product. There's an alternative approach, based on conventional matrix multiplication, which some readers may find enlightening. (1) Show that (BP1) may be rewritten as

$$\delta^L = \Sigma'(z^L)\nabla_a C, \tag{33}$$

where $\Sigma'(z^L)$ is a square matrix whose diagonal entries are the values $\sigma'(z^L_j)$, and whose off-diagonal entries are zero. Note that this matrix acts on $\nabla_a C$ by conventional matrix multiplication. (2) Show that (BP2) may be rewritten as

$$\delta^l = \Sigma'(z^l)(w^{l+1})^T \delta^{l+1}. \tag{34}$$

(3) By combining observations (1) and (2) show that

$$\delta^l = \Sigma'(z^l)(w^{l+1})^T \ldots \Sigma'(z^{L-1})(w^L)^T \Sigma'(z^L)\nabla_a C \tag{35}$$

For readers comfortable with matrix multiplication this equation may be easier to understand than (BP1) and (BP2). The reason I've focused on (BP1) and (BP2) is because that approach turns out to be faster to implement numerically.

Solutions 9. (張繁可)

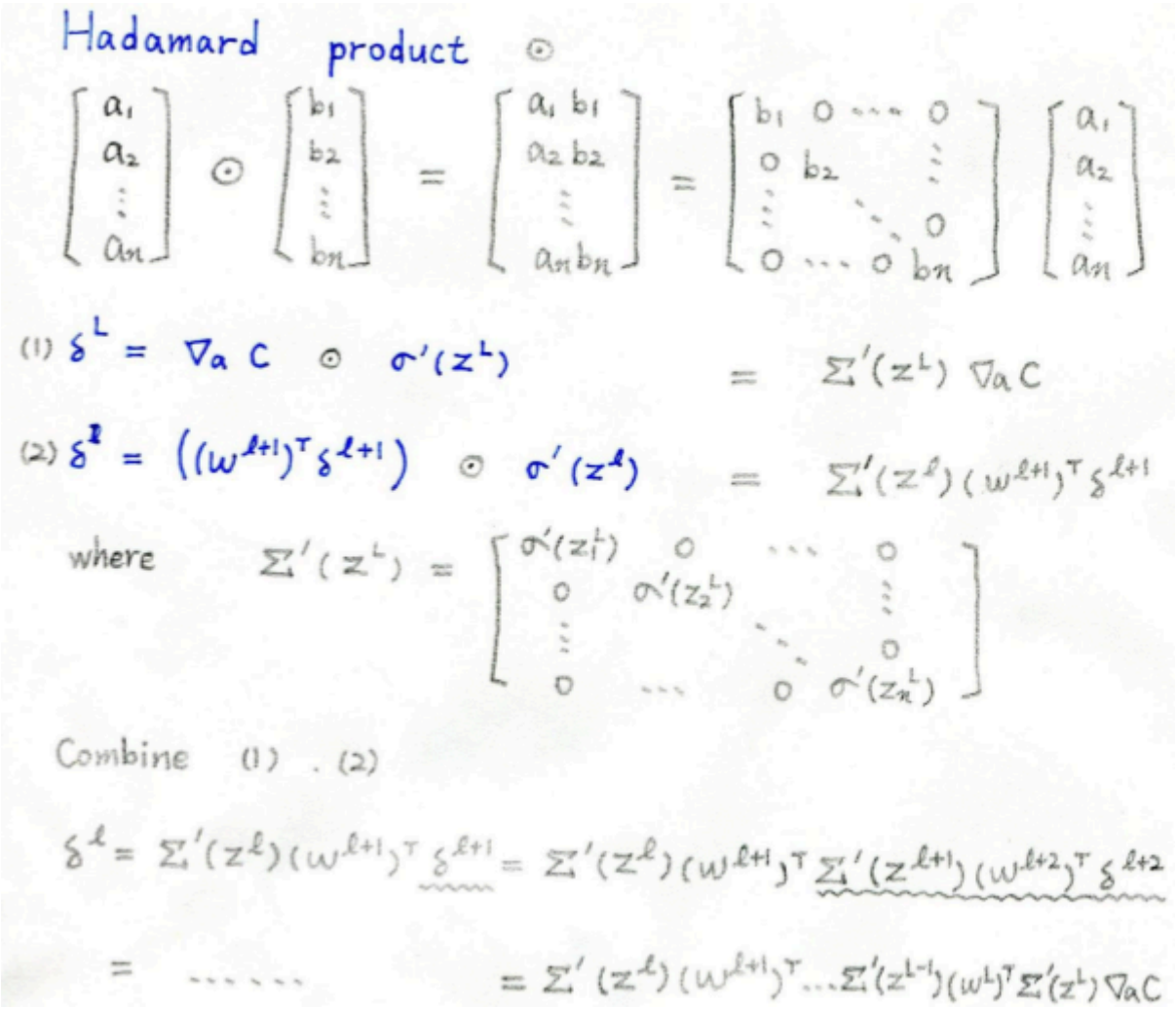
**Summary: the equations of backpropagation**

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

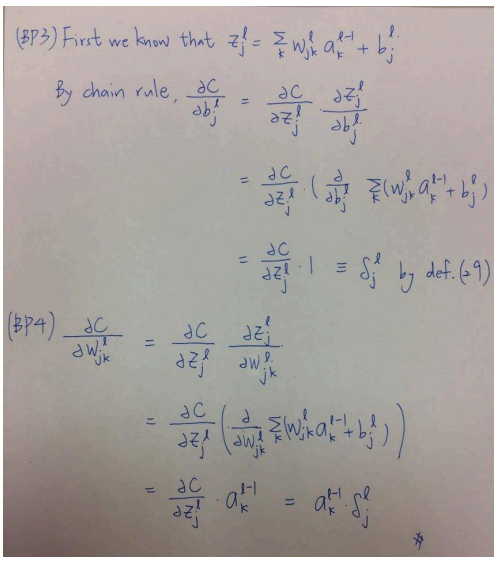
$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{BP4}$$




---

Question 10.
Prove Equations (BP3) and (BP4)

Solutions 10. (吳亭慧)




---

Question 11.

**Backpropagation with a single modified neuron**

Suppose we modify a single neuron in a feedforward network so that the output from the neuron is given by $f(\sum_j w_j x_j + b)$, where *f* is some function other than the sigmoid. How should we modify the backpropagation algorithm in this case?

Solutions 11. (陳彥禎)

The modified backpropagation algorithm will be like this:

1. Input x : Set the corresponding activation $a^1$ for the input layer.
2. Feedforward : For each $l = 2, 3, \ldots, L-1$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$ and $z^L = w^L a^{L-1} + b^L$ and $a^L = f(z^L)$.
3. Output error $\delta^L$ : Compute the vector $\delta^L = \nabla_a C \odot f'(z^L)$
4. Backpropagate the error : For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. Output : The gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

---

Question 12.

Solutions 12. (Name(s) of solution provider)

---

Question 13.

**Fully matrix based approach to backpropagation over a minibatch:** Our implementation of stochastic gradient descent loops over training examples in a minibatch. It's possible to modify the backpropagation algorithm so that it computes the gradients for all training examples in a mini batch simultaneously. The idea is that instead of beginning with a single input vector, , we can begin with a matrix X = [x_1, ... x_n] whose columns are the vectors in the mini batch. We forward propagate by multiplying by the weight matrices, adding a suitable matrix for the bias terms, and applying the sigmoid function everywhere. We backpropagate along similar lines. Explicitly write out pseudocode for this approach to the backpropagation algorithm. Modify `network.py` so that it uses this fully matrix based approach. The advantage of this approach is that it takes full advantage of modern libraries for linear algebra. As a result it can be quite a bit faster than looping over the minibatch. (On my laptop, for example, the speedup is about a factor of two when run on MNIST classification problems like those we considered in the last chapter.) In practice, all serious libraries for backpropagation use this fully matrix based approach or some variant.

Solutions 13. (張大衛)

Please see the zip file `ex13.zip' in the same directory.

---

Question 14.

Verify that σ'(z)=σ(z)(1-σ(z)).

Solutions 14. (何文劭)

$$\sigma(z)' = \left(\frac{1}{1+e^{-z}}\right)' = \frac{e^{-z}}{(1+e^{-z})^2} = (\frac{1}{1+e^{-z}})(\frac{e^{-z}}{(1+e^{-z})}) = \sigma(z)(1-\sigma(z))$$

---

Question 15.One gotcha with the cross-entropy is that it can be difficult at first to remember the respective roles of the s and the s. It's easy to get confused about whether the right form is or . What happens to the second of these expressions when or ? Does this problem afflict the first expression? Why or why not? y = y1, y2,… a L 1 , a L 2 ,… C = − [ ln + (1 − )ln(1 − )] . 1 n ∑x ∑ j yj a L j yj a L j (63) ∑j 1 0 y a −[y ln a + (1 − y)ln(1 − a)] −[a ln y + (1 − a)ln(1 − y)

Solutions 15. (Name(s) of solution provider)

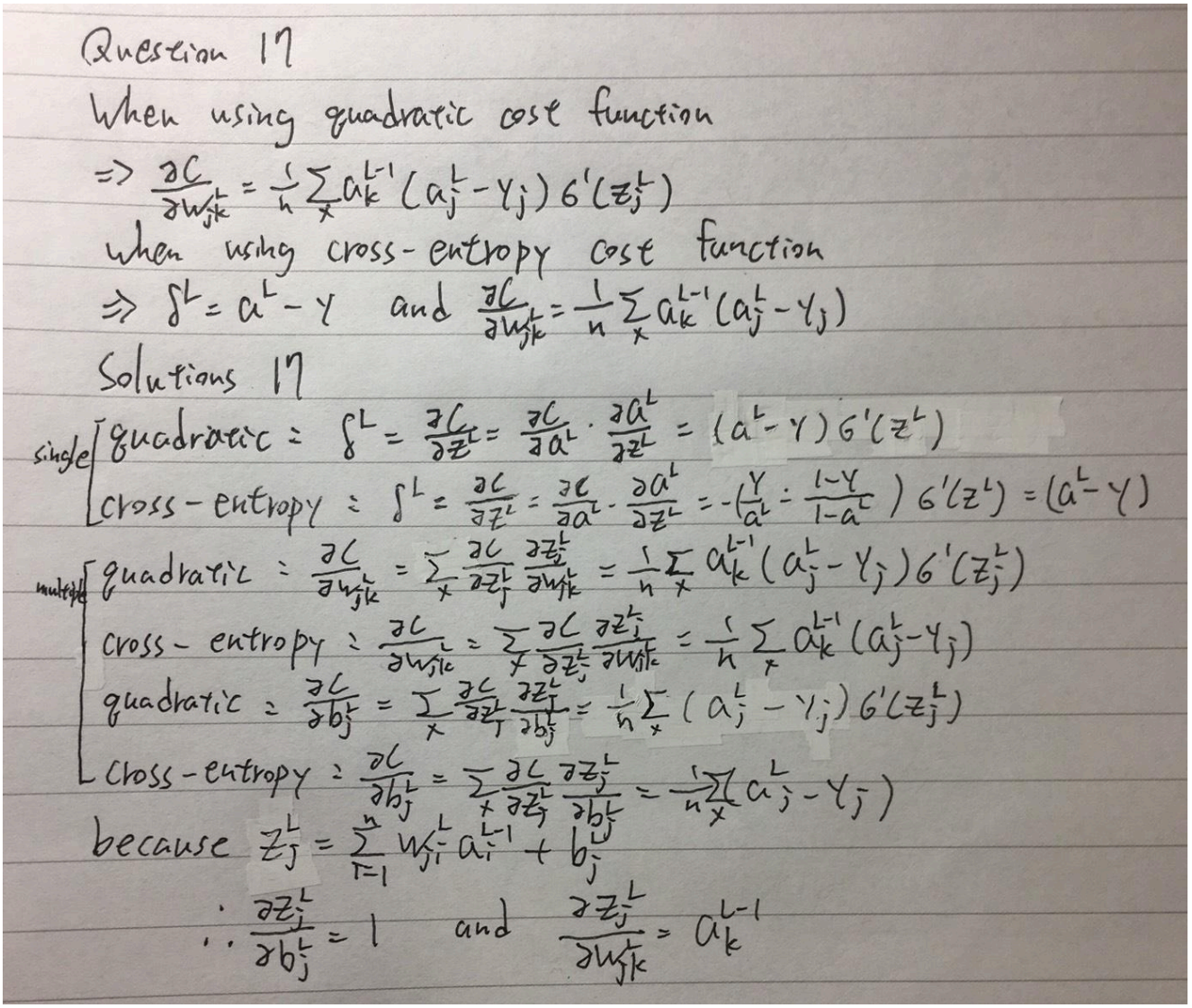
---

Question 16.

Solutions 16. (Name(s) of solution provider)

Question 17.

Solutions 17. (何庭昀)

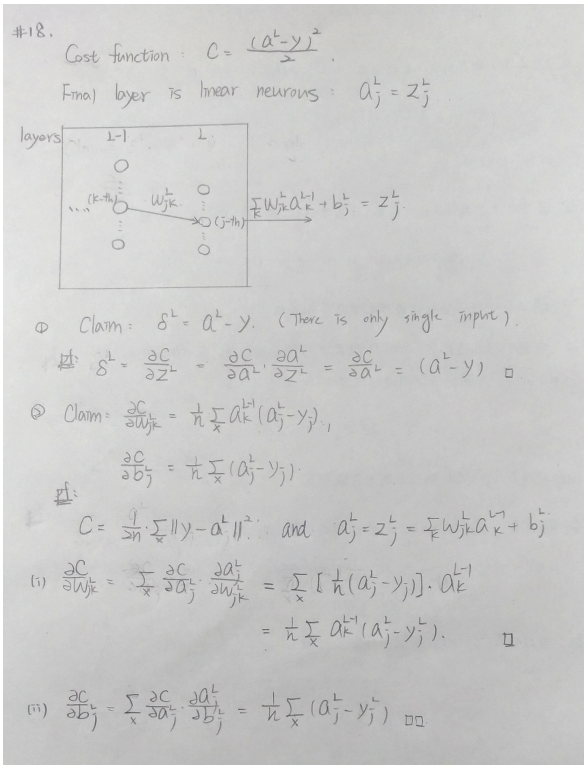



Question 18.

Show that if we use the quadratic cost function then the output error $\delta^L$ for a single training example x is given by $\delta^L = a^L - y$.

Show that the partial derivatives with respect to the weights and biases in the output layer are given by
$\partial C/\partial w^L_{jk} = (1/n)*\sum_x (a^{(L-1)}_k * (a^L_j - y_j)$ and
$\partial C/\partial b^L_j = (1/n)*\sum_x (a^L_j - y_j)$ .

Solutions 18. (李承恩)




---

Question 19.

Solutions 19. (Name(s) of solution provider)

---

Question 20.

**20.** • Construct an example showing explicitly that in a network with a sigmoid output layer, the output activations $a_j^L$ won't always sum to 1.

Solutions 20. (蔡宇翔)

$$a_j^L = \frac{1}{1 + e^{-z_j^L}}$$

Take $z_1^L = 0, z_2^L = 5$

$$a_1^L = \frac{1}{2}, a_2^L = \frac{1}{1 + e^{-5}} > \frac{1}{2}$$

$$a_1^L + a_2^L > \frac{1}{2} + \frac{1}{2} = 1$$

---

Question 21.

**21.** • **Monotonicity of softmax** Show that $\partial a^L_j / \partial z^L_k$ is positive if $j = k$ and negative if $j \neq k$. As a consequence, increasing $z^L_j$ is guaranteed to increase the corresponding output activation, $a^L_j$, and will decrease all the other output activations. We already saw this empirically with the sliders, but this is a rigorous

Solutions 21. (林澤佑)

Suppose there are $n$ neuron in the output layer with softmax function.

For $j = 1, 2, \cdots, n$, write $a^L_j = \dfrac{e^{z^L_j}}{\sum\limits_{k=1}^{n} e^{z^L_k}} = \dfrac{h(z^L_j)}{\sum\limits_{k=1}^{n} h(z^L_k)} = 1 - \dfrac{\sum\limits_{k\neq j} h(z^L_k)}{\sum\limits_{k=1}^{n} h(z^L_k)}$, where $h(x) = e^x$.

If $j \neq k$, then $\dfrac{\partial a^L_j}{\partial z^L_k} = \left[\dfrac{h(z^L_j)}{\sum\limits_{k=1}^{n} h(z^L_k)}\right]' = -\dfrac{h(z^L_j)}{(\sum\limits_{k=1}^{n} h(z^L_k))^2} \cdot h'(z^L_k) = -\dfrac{h(z^L_j)}{(\sum\limits_{k=1}^{n} h(z^L_k))^2} \cdot e^{z^L_k} < 0.$

If $j = k$, then $\dfrac{\partial a^L_j}{\partial z^L_j} = \left[1 - \dfrac{\sum\limits_{k\neq j} h(z^L_k)}{\sum\limits_{k=1}^{n} h(z^L_k)}\right]' = \dfrac{\sum\limits_{k\neq j} h(z^L_k)}{(\sum\limits_{k=1}^{n} h(z^L_k))^2} \cdot h'(z^L_j) = \dfrac{\sum\limits_{k\neq j} h(z^L_k)}{(\sum\limits_{k=1}^{n} h(z^L_k))^2} \cdot e^{z^L_j} > 0.$

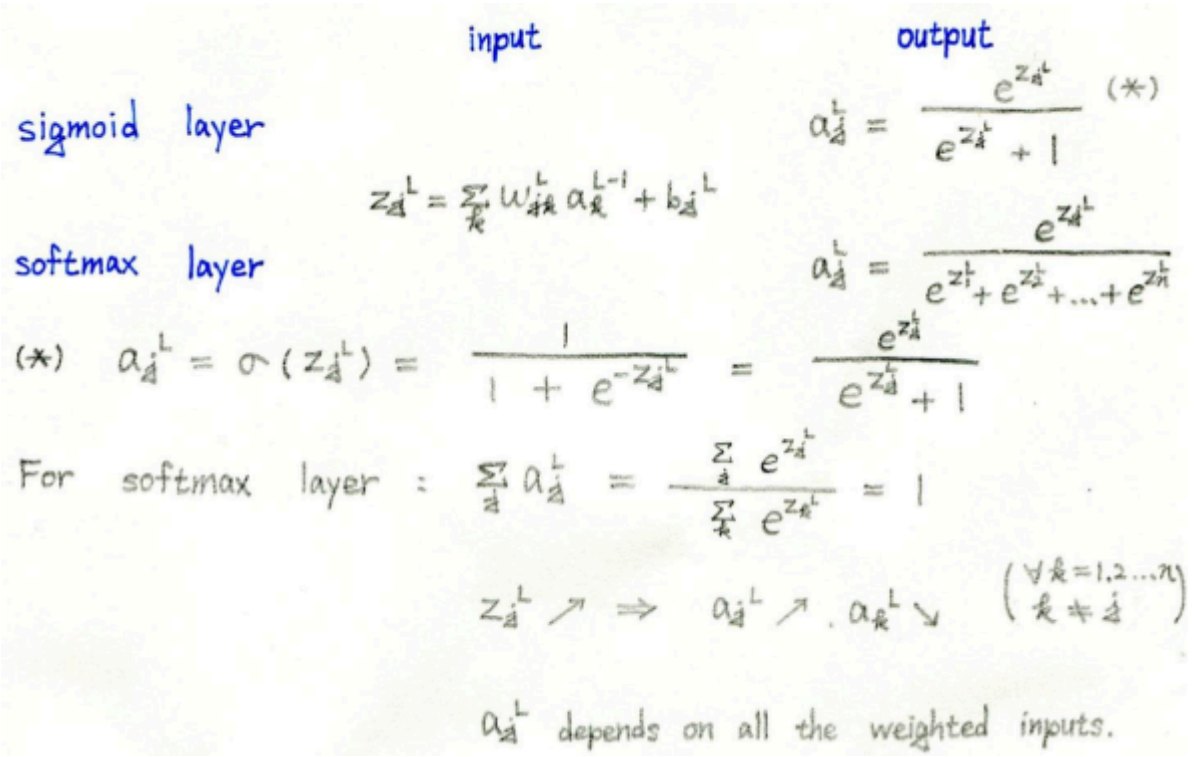
The desired result follows.

---

Question 22.

- **Non-locality of softmax** A nice thing about sigmoid layers is that the output $a^L_j$ is a function of the corresponding weighted input, $a^L_j = \sigma(z^L_j)$. Explain why this is not the case for a softmax layer: any particular output activation $a^L_j$ depends on *all* the weighted inputs.

Solutions 22. (張繁可)

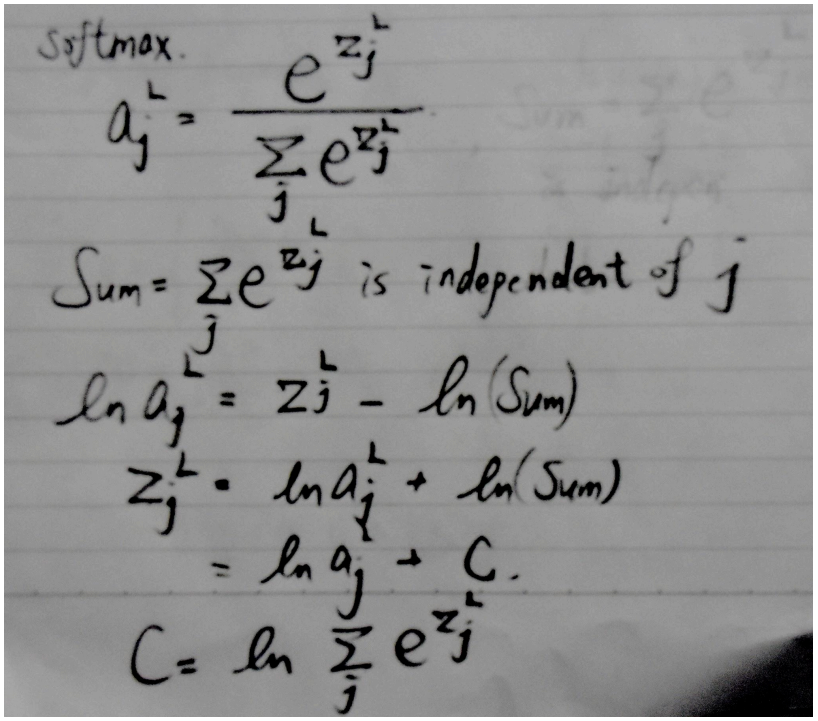



---

Question 23.

**23.** • **Inverting the softmax layer** Suppose we have a neural network with a softmax output layer, and the activations $a_j^L$ are known. Show that the corresponding weighted inputs have the form $z_j^L = \ln a_j^L + C$, for some constant $C$ that is independent of $j$.

Solutions 23. (楊喬諳)



Question 24.

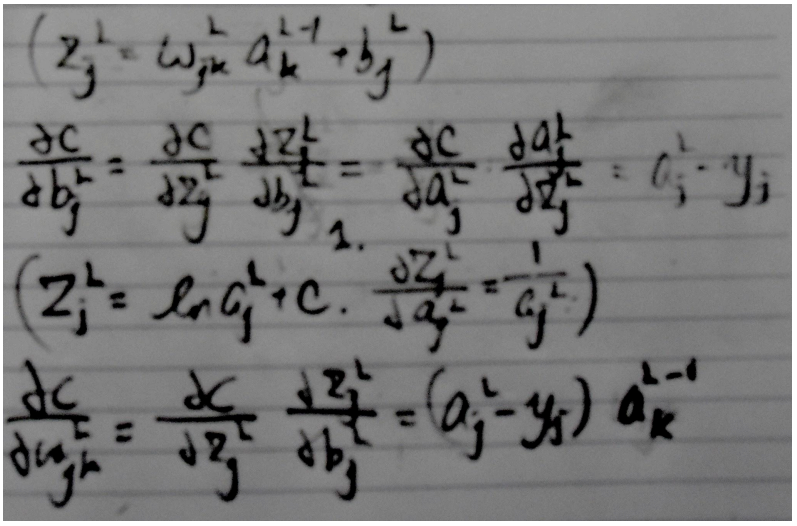
**24.** • Derive Equations (81) and (82).

$$C \equiv -\ln a_y^L. \tag{80}$$

$$\frac{\partial C}{\partial b_j^L} = a_j^L - y_j \tag{81}$$

$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1}(a_j^L - y_j) \tag{82}$$

Solutions 24. (楊喬諳)



---

Question 25.

**25.** • **Where does the "softmax" name come from?** Suppose we change the softmax function so the output activations are given by

$$a_j^L = \frac{e^{cz_j^L}}{\sum_k e^{cz_k^L}}, \tag{83}$$

where $c$ is a positive constant. Note that $c = 1$ corresponds to the standard softmax function. But if we use a different value of $c$ we get a different function, which is nonetheless qualitatively rather similar to the softmax. In particular, show that the output activations form a probability distribution, just as for the usual softmax. Suppose we allow $c$ to become large, i.e., $c \to \infty$. What is the limiting value for the output activations $a_j^L$? After solving this problem it should be clear to you why we think of the $c = 1$ function as a "softened" version of the maximum function. This is the origin of the term "softmax".

Solutions 25. (林澤佑)

And $\sum_j a_j^L = \dfrac{\sum_j e^{cz_j^L}}{\sum_k e^{cz_k^L}} = 1$. Hence, we show that $\{a_j^L\}_j$ forms a probability distribution.

Now, let $z_{j_0}^L = \max_j z_j^L$.

Assume that $z_j^L - z_{j_0}^L < 0$ for all $j \neq j_0$. i.e., $z_{j_0}$ is *the* only maximal.

By dividing numerator and denominator with $e^{cz_{j_0}^L}$ we can rewrite $a_j^L$ as follows:

$$a_j^L = \frac{e^{cz_j^L}}{\sum_k e^{cz_k^L}} = \frac{e^{c(z_j^L - z_{j_0}^L)}}{\sum_k e^{c(z_k^L - z_{j_0}^L)}} = \begin{cases} \dfrac{e^{c(z_j^L - z_{j_0}^L)}}{1 + \sum_{k \neq j_0} e^{c(z_k^L - z_{j_0}^L)}}, & \text{if } j \neq j_0 \\ \dfrac{1}{1 + \sum_{k \neq j_0} e^{c(z_k^L - z_{j_0}^L)}}, & \text{if } j = j_0 \end{cases}$$

Since $z_j^L - z_{j_0}^L < 0$, we have $c\left(z_j^L - z_{j_0}^L\right) \to -\infty$ as $c \to \infty$.

Hence, as $c \to \infty$, we have $a_j^L \to \begin{cases} 0, & \text{if } j \neq j_0 \\ 1, & \text{if } j = j_0 \end{cases} = \begin{cases} 0, & \text{if } a_j \neq a_{j_0} \\ 1, & \text{if } a_j = a_{j_0} \end{cases}$

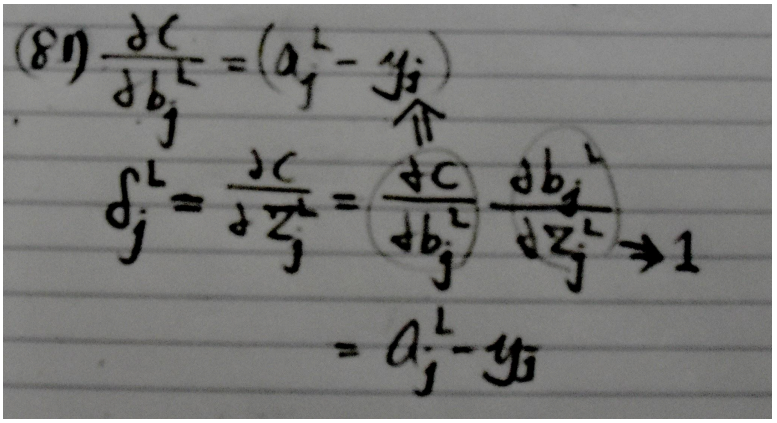
That is, $a_j^L$ is 1 when $z_j^L$ is the maximal among all $z_j^L$, and 0 if it is not the maximal one.

Further, if there are $m$ maximal among $z_j^L$, that is, $z_{j_k} \geq z_j^L$ for $k = 1, 2, \cdots, m$.
By the same argument as above, we have the following result:

As $c \to \infty$, we have $a_j^L \to \begin{cases} 0, & \text{if } j \neq j_k \text{ for all } k = 1, 2, \cdots m \text{ and for all } j \\ \dfrac{1}{m}, & \text{if } j = j_k \text{ for some } k = 1, 2, \cdots m \end{cases}$

---

Question 26.
Solutions 26. (楊喬諳)




---

Question 27.
As discussed above, one way of expanding the MNIST training data is to use small rotations of training images. What's a problem that might occur if we allow arbitrarily large rotations of training images?

Solutions 27. (何文劭)

---

Question 28.

Solutions 28. (Name(s) of solution provider)

---

Question 29.

**29.** • Verify that the standard deviation of $z = \sum_j w_j x_j + b$ in the paragraph above is $\sqrt{3/2}$. It may help to know that: (a) the variance of a sum of independent random variables is the sum of the variances of the individual random variables; and (b) the variance is the square of the standard deviation.

Solutions 29. (Name(s) of solution provider)

Suppose there are $n_{in}$ inputs $x_j$, and half of them are 1 and another half are 0.

Suppose $w_j$ $N(0, \frac{1}{n_{in}})$ and $b$ $N(0,1)$ which are all independent.

Then we have:

$$\begin{aligned} Var(z) &= Var(\textstyle\sum_j w_j x_j + b) \\ &= \textstyle\sum_j Var(w_j x_j) + Var(b) \text{ since they are independent.} \\ &= \textstyle\sum_j x_j^2 Var(w_j) + Var(b) \\ &= \frac{n_{in}}{2}\frac{1}{n_{in}} + 1 \text{ since half of } x_j \text{ are 1 and half of them are 0, so are their square.} \\ &= \frac{3}{2}. \end{aligned}$$

Hence, the standard deviation of $z$ is $\sqrt{\frac{3}{2}}$.

---

Question 30.

**30.** • **Connecting regularization and the improved method of weight initialization** L2 regularization sometimes automatically gives us something similar to the new approach to weight initialization. Suppose we are using the old approach to weight initialization. Sketch a heuristic argument that: (1) supposing $\lambda$ is not too small, the first epochs of training will be dominated almost entirely by weight decay; (2) provided $\eta\lambda \ll n$ the weights will decay by a factor of $\exp(-\eta\lambda/m)$ per epoch; and (3) supposing $\lambda$ is not too large, the weight decay will tail off when the weights are down to a size around $1/\sqrt{n}$, where $n$ is the total number of weights in the network. Argue that these conditions are all satisfied in the examples graphed in this section.

L2 regularization sometimes automatically gives us something similar to the new approach to weight initialization. Suppose we are using the old approach to weight initialization. Sketch a heuristic argument that: (1) supposing $\lambda$ is not too small, the first epochs of training will be dominated almost entirely by weight decay; (2) provided $\eta\lambda \ll n$ the weights will decay by a factor of $\exp(-\eta\lambda/m)$ per epoch; and (3) supposing $\lambda$ is not too large, the weight decay will tail off when the weights are down to a size around $1/\sqrt{n}$,where is the total number of weights in the network.

Solutions 30. (何庭昀)

(1)because when $\lambda$ is not too small, by the formula $w \to \left(1-\frac{\eta\lambda}{n}\right)w-\frac{\eta}{m}\sum_x \frac{\partial C_x}{\partial w}$,

the $-\frac{\eta\lambda}{n}$ part will near 1 and the first epochs of training will be dominated almost entirely by weight decay.

(2) $w$ part will be multiple (n/m) times by $\left(1-\frac{\eta\lambda}{n}\right)$ in one epoch, because $\eta\lambda \ll n$ ,

$\left(1-\frac{\eta\lambda}{n}\right)$ ^(n/m) roughly equal to $\exp(-\eta\lambda/m)$

(3)???

---

Question 31.

> **31.** • Modify the code above to implement L1 regularization, and use L1 regularization to classify MNIST digits using a 30 hidden neuron network. Can you find a regularization parameter that enables you to do better than running unregularized?

Modify the code above to implement L1 regularization, and use L1 regularization to classify MNIST digits using a 30 hidden neuron network.

Can you find a regularization parameter that enables you to do better than running unregularized ?

Solutions 31. (李承恩)

---

Question 32.

Solutions 32.

---

Question 33.

**33.** • Modify `network2.py` so that it implements early stopping using a no-improvement-in-$n$ epochs strategy, where $n$ is a parameter that can be set.

Solutions 33. (蔡宇翔)
The modified file is network2_ex33.py.
Change its name as network2.py.
net.SGD(training_data, 30, 10, 10.0, lmbda = 1000.0,
...evaluation_data=validation_data, monitor_evaluation_accuracy=True, no_improvement_num=10)
no_improvement_num's default value is zero, and it is not be executed without setting.

```
Epoch 14 training complete
Accuracy on evaluation data: 1030 / 10000
Highest accuracy on evaluation data: 1090 7/14

Epoch 15 training complete
Accuracy on evaluation data: 991 / 10000
Highest accuracy on evaluation data: 1090 7/15

Epoch 16 training complete
Accuracy on evaluation data: 1090 / 10000
Highest accuracy on evaluation data: 1090 7/16

Epoch 17 training complete
Accuracy on evaluation data: 967 / 10000
Highest accuracy on evaluation data: 1090 7/17
No Improvement in 10
```

In picture, the highest accuracy is 1090 in Epoch 7, so it terminated because no improvement during epoch 7~17

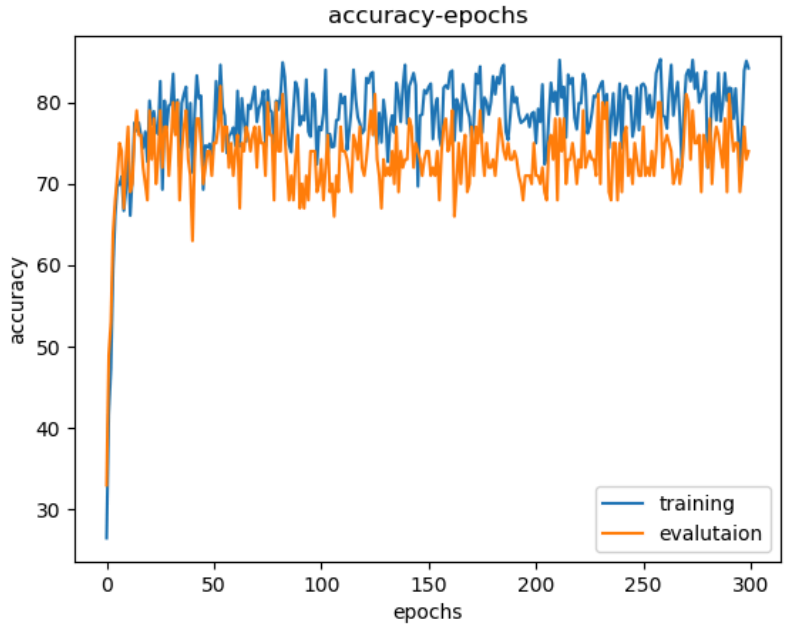
---

Question 34. Can you think of a rule for early stopping other than no-improvement-in n ? Ideally, the rule should compromise between getting high validation accuracies and not training too long. Add your rule to network2.py , and run three experiments comparing the validation accuracies and number of epochs of training to no-improvement-in 10 .

Solutions 34. (袁佑緣)

I add a early_stop_factor(< 1) to max accuracy comparison,  saying the program updates max accuracy(improves) if the current accuracy is bigger than the max accuracy times early_stop_factor. It would make the program stop later and try to improve more.

I use ex34.py to generate an experiment sample and save it in numpy data, the following figure is the whole training process (300 epochs).




And I call ex34-compare.py to compare the effect of the addition of early_stop_factor, the following table is a comparison of different early_stop_factor.

| early_stop_factor | early stop epoch (total 300) | best accuracy |
|---|---|---|
| 1 (standard no-improvement-in-10) | 41 | 73.0 |
| **0.98** | **63** | **75.0** |
| 0.96 | 134 | 70.0 |
| 0.95 | no early stop | none |

---

Question 35. Modify network2.py so that it implements a learning schedule that: halves the learning rate each time the validation accuracy satisfies the noimprovementin 10 rule; and terminates when the learning rate has dropped to 1/128 of its original value.

Solutions 35. (袁佑緣)

The test code is ex35.py, and the network is network_ex35.py, and the following figure is the result of the training process with decreasing learning rate.

```
Epoch 148 training complete
Cost on training data: 2.63434152297
Accuracy on training data: 825 / 1000
Best Accuracy:  825
No imporvement counter:  10
Cost on evaluation data: 8.01430983185
Accuracy on evaluation data: 80 / 100
==========Early stop==========
```

---

Question 36.

It's tempting to use gradient descent to try to learn good values for hyperparameters such as λ and η . Can you think of an obstacle to using gradient descent to determine λ ? Can you think of an obstacle to using gradient descent to determine η ?

Solutions 36. (袁佑緣)

Actually the final cost is a function of λ, η and weights, and since final weights depend on λ and η, so the final cost is really a function of two variables λ and η.
But there is no clear way to write the partial derivatives cost function respect to λ and η, for example, to compute final cost function c respect to λ

$$\frac{\partial}{\partial \lambda} C(\lambda, \eta) = \frac{\partial}{\partial \lambda} C_0(\lambda, \eta) + \frac{\lambda}{2n} \sum_i 2 w_i \frac{\partial w_i}{\partial \lambda} + \frac{1}{2n} \sum_i w_i^2,$$

but we don't know how to compute $\frac{\partial}{\partial \lambda} C_0(\lambda, \eta)$ and $\frac{\partial w_i}{\partial \lambda}$ .

---

Question 37
What would go wrong if we use **μ>1** in the momentum technique?
Solutions 37. (郭俊廷)

$$v \to v' = \mu v - \eta \nabla C \tag{107}$$
$$w \to w' = w + v'. \tag{108}$$

If **μ>1,** there is a possibility that the effect of momentum over-exceed the effect of gradient descent eq.(107).This will cause the gradient descent fail to move toward the minimum.

---

Qusetion 38
What would go wrong if we use **μ<0** in the momentum technique?
Solutions 38. (郭俊廷)

$$v \to v' = \mu v - \eta \nabla C \tag{107}$$
$$w \to w' = w + v'. \tag{108}$$

If **μ<0 ,** it will change the direction of velocity in the first term in eq.(107) every time. This may cause the gradient descent unable to move forward effectively or even move in the opposite direction.

---

Qusetion 39

Solutions 39. (郭俊廷)

1. Initialize self.velocities with zeros.

```
def default_weight_initializer(self):
    self.biases = [np.random.randn(y, 1) for y in self.sizes[1:]]
    self.weights = [np.random.randn(y, x)/np.sqrt(x)
                    for x, y in zip(self.sizes[:-1], self.sizes[1:])]
    self.velocities = [np.zeros(w.shape) for w in self.weights]
```

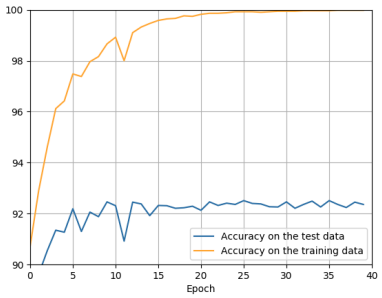
2. Mimicking eq(107) and eq(108)

```
def update_mini_batch(self, mini_batch, eta, lmbda, n, mu=0):
    """Update the network's weights and biases by applying gradient
    descent using backpropagation to a single mini batch.  The
    ``mini_batch`` is a list of tuples ``(x, y)``, ``eta`` is the
    learning rate, ``lmbda`` is the regularization parameter, and
    ``n`` is the total size of the training data set.

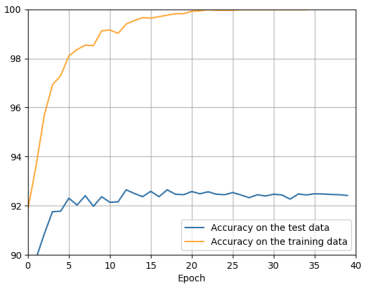
    """
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.velocities = [mu * v - (eta/len(mini_batch))*nw
                       for v, nw in zip(self.velocities, nabla_w)]
    self.weights = [v+(1-eta*(lmbda/n))*w
                    for v, w in zip(self.velocities, self.weights)]
    self.biases = [b-(eta/len(mini_batch))*nb
                   for b, nb in zip(self.biases, nabla_b)]
```

**3. Results**

**μ=0**




**μ=0.3 (Achieves the accuracy saturation faster!!!)**




---

Question 40

Prove the identity in Equation (111). i.e., derive the relation between sigmoid function and hyperbolic tangent.

Solutions 40 (林澤佑)

Let $\sigma(z) = \dfrac{1}{1+e^{-z}}$, be the sigmoid function. Then we have:

$$\begin{aligned} & 2\sigma(z) - 1 \\ = & \frac{2}{1+e^{-z}} - \frac{1+e^{-z}}{1+e^{-z}} \\ = & \frac{1-e^{-z}}{1+e^{-z}} \cdot \frac{e^{z/2}}{e^{z/2}} \\ = & \frac{e^{z/2} - e^{-z/2}}{e^{z/2} + e^{-z/2}} = \tanh(\frac{z}{2}). \end{aligned}$$

i.e., $\sigma(z) = \dfrac{1+\tanh(z/2)}{2}$ which is the equation (111).

---

Question 41

We've seen how to use networks with two hidden layers to approximate an arbitrary function. Can you find a proof showing that it's possible with just a single hidden layer? As a hint, try working in the case of just two input variables, and showing that: (a) it's possible to get step functions not just in the x or y directions, but in an arbitrary direction; (b) by adding up many of the constructions from part (a) it's possible to approximate a tower function which is circular in shape, rather than rectangular; (c) using these circular towers, it's possible to approximate an arbitrary function.

Solutions 41 (陳冠羽)

(a) 由第四章的討論我們知道，藉由縮放，sigmoid function $\sigma(z)$可以近似 step function $h(z)$。

現在考慮兩個輸入$z = (x, y)$，取$w = (w_1, w_2)$，則當$w$方向固定，長度足夠長時有

$$\sigma(w_1 x + w_2 y + b) \approx h_w(t + b/|w|)$$

，其中$t$為$(x, y)$方向軸上的座標，$h_w$即為$w$方向的$step\ function$。

(b) 仿照課本的作法，由(a)我們可以造出"wall function"，即w方向上的on-off function。接著我們要做出tower function。我們的策略是，把tower的高度(值)分散到足夠多通過此點的wall，使得這些wall各自的高度(值)微不足道，但總和正好為tower高度(值)，如此我們可以近似circular towers。

(c) 任何function都可以用數個circular towers來逼近。更高維度的情況可以類推。

---

Question 42

Earlier in the book we met another type of neuron known as a rectified linear unit. Explain why such neurons don't satisfy the conditions just given for universality. Find a proof of universality showing that rectified linear units are universal for computation.

Solutions 42 (陳冠羽)

因為rectified linear unit $z \to \infty$時不會飽和(saturated)，不能造出step function，因此不適用前段證明。
在下面這篇論文中，作者"construct a sparsely-connected depth-4 neural network and bound its error in approximating f"。
**Provable approximation properties for deep neural networks(2015),** Uri Shaham, Alexander Cloninger, Ronald R. Coifman
http://cpsc.yale.edu/sites/default/files/files/tr1513(1).pdf

---

Question 43
Suppose we consider linear neurons, i.e., neurons with the activation function $s(z)=z$. Explain why linear neurons don't satisfy the conditions just given for universality. Show that such neurons can't be used to do universal computation.

Solutions 43 (陳冠羽)
因為linear neurons $z \to \infty$時及 $z \to -\infty$均不會飽和(saturated)，不能造出step function，因此亦不適用前段證明。
另外，若僅採用linear neurons，則最終合成函數亦為linear function(or affine transformation)，無法近似任意函數(because the composite of translation, dilation, reflection is still linear)。

---

Question 44-45
Solutions 44-45

---

**Question 44:** In our discussion of vanishing gradient problem, we made use of the fact that sigma'(z) < ¼. Suppose we used a different activation function, one whose derivative could be much larger. Would that help us avoid the unstable gradient problem?
**Solution 44:** (陳德禮)
Unstable gradient problem arises when the gradient of the previous layer(s) is smaller than the following layers. This means that in earlier layers, neurons learn at a much slower rate than in later layers.
Assuming we deploy a different activation function, whose derivative is larger than ¼ (sigma'(z) >= 1/4), we still cannot anticipate the vanishing gradient problem. The reason is shown in Equation (122) (Niellsen, page 207). The partial derivative of the cost function is basically a multiplication of sigma'(z) function. Even if we pick a value of sigma'(z) > ¼, we will still end up with the vanishing gradient issue, as multiplication of fraction yields an even smaller fraction.

---

Question 46

**46.** • **Identity neuron:** Consider a neuron with a single input, $x$, a corresponding weight, $w_1$, a bias $b$, and a weight $w_2$ on the output. Show that by choosing the weights and bias appropriately, we can ensure $w_2\sigma(w_1 x + b) \approx x$ for $x \in [0,1]$. Such a neuron can thus be used as a kind of identity neuron, that is, a neuron whose output is the same (up to rescaling by a weight factor) as its input. *Hint: It helps to rewrite* $x = 1/2 + \Delta$, *to assume* $w_1$ *is small, and to use a Taylor series expansion in* $w_1\Delta$.

Solution 46 ()

**Question 48:** (a) What classification do you get if you omit the fully-connected layers, and just use the convolutional-pooling layer and softmax layer? (b) Does the inclusion of the fully connected layer help?

**Solution 48:** (陳德禮)

(a) Omission of the fully-connected layers still allow us to obtain reasonably accurate outputs from both convolutional and pooling layers (hence, a linear classification). The addition of the fully-connected layers improve the learning outcome as it is a cheap way to create non-linear combinations of the existing features.

(b) Thus, the inclusion of the fully connected layer does help and improve the learning outcome of the system as it enhances the (possibly non-linear) function in that space.

**Question 50:** The idea of convolutional layers is to behave in an invariant way across images. It may seem surprising, then, that our network can learn more when all we've done is translate the input data. *Can you explain why this is actually quite reasonable?*

**Solution 50:** (陳德禮)

Schematic Diagram:

Convolutional Layer ----------> Pooling Layer ----------> Fully Connected Layer

The output of **the convolutional layer**(s) generates meaningful, low-dimensional, and invariant information. The addition of **the pooling layer**(s) afterwards generates a robust output that does not change significantly despite the change in the input data. The statement above is quite reasonable because both convolutional and pooling layers can cause “underfitting.” [1] The application of pooling layer(s) post convolutional layer(s) do not necessarily apply on all channels to retain “highly invariant features” [1] and to prevent underfitting “when the translation invariance prior is incorrect” [1]

Source(s):

[1] Goodfellow, I. Bengio, Y. Courville, A. 2016. “*Deep Learning*”. page 347