CPSC 663 Assignment 2

Author: Wenxin Xu

The `xu_wenxin_assignment_2.zip` file includes following files:

- `xu_wenxin_assignment_2.pdf` : A detailed report.
- `prob4.py` : a Python script contains neural network model and training functions.
- `xuw_assignment_2.ipynb`: A Jupyter Notebook contains all the code.

Problem 1 Gradient descent

1. Geometric interpretation

Provide a geometric interpretation of gradient descent in the one-dimensional case.

Suppose we have a cost function $C : \mathbb{R} \to \mathbb{R}$ which is continuous and differentiable,

to find the minimum of C, we randomly start at a point x_i (a value of our parameter x),

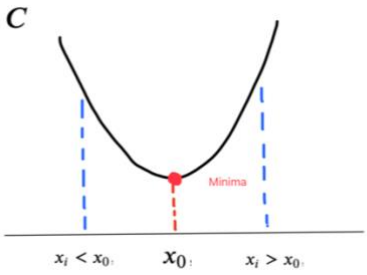
the formula of gradient descent gives that the updated value of parameter is $x_{i+1} \leftarrow x_i - \eta \nabla C_x$, where η is learning rate.

because we are in 1D, the gradient of C w.r.t x is just derivative of C w.r.t x, thus $x_{i+1} \leftarrow x_i - \eta C'(x_i)$.

when $x_i < x_0$, $C'(x_i) < 0$, x_{i+1} will increase

when $x_i > x_0$, $C'(x_i) > 0$, x_{i+1} will decrease

in each case, C will always decrease and reach the minimum.



2. Online learning

An extreme version of gradient descent is to use a mini-batch size of just 1. 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.

- Advantage of online learning:
  1. computationally fast because only one training sample is processed at a time.
  2. memory efficient because only one training sample is stored at a time.
  3. more easy to avoid local minimum because steps taken towards minima of loss function have oscillations such like shooting back and forth
- Disadvantage of online learning: unstable
  1. inaccurate learning because the steps taken towards the minima are noisy which leads to inaccurate direction
  2. slow convergence because the noisy steps
  3. computationally expensive because it doesn't take advantage of vectorized operations and parallel computing

Problem 2 Backpropagation

1 A single modified neuron

1. Backpropagation with a single modified neuron (Nielsen book, chapter 2)

   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?

- if the modified neuron is j-th neuron on l-th layer

1. input x: set the corresponding activation a^l for the input layer
2. feedforward: for each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and

$$a_i^l = \begin{cases} \sigma(z_i^l) & i \neq j \\ f(z_i^l) & i = j \end{cases}$$

3. output error $\delta^L = \nabla_{a^L} C \odot \sigma'(z^L)$
4. backpropagate the error: for each $l = L-1, L-2, \ldots, 2$ compute

$$\delta_i^l = \begin{cases} \left[(w_i^{l+1})^T \delta_i^{l+1}\right] \sigma'(z_i^l) & i \neq j \\ \left[(w_i^{l+1})^T \delta_i^{l+1}\right] f'(z_i^l) & i = j \end{cases}$$

5. output: gradient of cost function $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$

- if the modified neuron is j-th neuron on l-th layer

1. input x: set the corresponding activation a^l for the input layer
2. feedforward: for each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and

$$a_i^l = \begin{cases} \sigma(z_i^l) & i \neq j \\ f(z_i^l) & i = j \end{cases}$$

3. output error

$$\delta_i^L = \begin{cases} \frac{\partial C}{\partial a_i^L} \sigma'(z_i^L) & i \neq j \\ \frac{\partial C}{\partial a_i^L} f'(z_i^L) & i = j \end{cases}$$

4. backpropagate the error: for each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = \left[(w^{l+1})^T \delta^{l+1}\right] \odot \sigma'(z^l)$
5. output: gradient of cost function $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$

2 softmax and log-likelihood cost

Backpropagation with softmax and the log-likelihood cost (Nielsen book, chapter 3)

To apply the backpropagation algorithm for a network containing sigmoid layers to a network with a softmax layer,

we need to figure out an expression for the error $\delta_j^L = \partial C/\partial z_j^L$ in the final layer.

Show that a suitable expressions: $\delta_j^L = a_j^L - y_j$

$$\delta^L = \frac{\partial C}{\partial z_j^L} = \sum_k^n \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}$$

suppose y is true output class, $y \in \{1, 2, \cdots, n\}$

$$\begin{cases} \frac{\partial C}{\partial a_k^L} = \frac{\partial(-\ln a_y^L)}{\partial a_k^L} = \begin{cases} -\frac{1}{a_y^L} & (k=y) \\ 0 & (k \neq y) \end{cases} \\ \frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial\left(\frac{e^{z_k^L}}{\sum_j e^{z_k^L}}\right)}{\partial z_j^L} = \begin{cases} a_j^L(1-a_j^L) & (j=k) \\ -a_k^L a_j^L & (j \neq k) \end{cases} \end{cases}$$

$$\Rightarrow \sum_k^n \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} = \begin{cases} -\frac{1}{a_y^L}\left[a_j^L(1-a_j^L)\right] = a_j^L - 1 & (y=j) \\ -\frac{1}{a_y^L}(-a_k^L a_j^L) = a_j^L & (y \neq j) \end{cases}$$

$$\Rightarrow \delta^L = a_j^L - y_j$$

y_j is the jth element of one-hot encoding vector $y \in R^n$

3 Linear neuron

Backpropagation with linear neurons (Nielsen book, chapter 2)

Suppose we replace the usual non-linear activation function (sigmoid) with $\sigma(z) = z$ throughout the network.

Rewrite the backpropagation algorithm for this case.

$\sigma(z) = z \rightarrow \sigma'(z) = 1$

1. input x: set the corresponding activation a^l for the input layer
2. feedforward: for each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = z^l$
3. output error $\delta^L = \nabla_{a^L} C$
4. backpropagate the error: for each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = (w^{l+1})^T \delta^{l+1}$
5. output: gradient of cost function $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$

Problem 3 Cross Entropy Loss

1 Formula

1. It can be difficult at first to remember the respective roles of the ys and the as for cross-entropy. (Nielsen book, chapter 3)

   It 's easy to get confused about whether the right form is

$$-[y \ln a + (1-y)\ln(1-a)] \; or \; -[a \ln y + (1-a)\ln(1-y)]$$

- What happens to the second of these expressions when y = 0 or 1?

  because $\ln 0$ is undefined, the second expression is undefined when y = 0 or 1.

$$-[a \ln y + (1-a)\ln(1-y)] = \begin{cases} -[a \ln 0 + (1-a)\ln a] & y = 0 \\ -(1-a)\ln 0 & y = 1 \end{cases}$$

- Does this problem afflict (affect) the first expression? Why or why not?

  Not because the first expression is defined when y = 0 or 1.

$$-[y \ln a + (1-y)\ln(1-a)] = \begin{cases} -\ln(1-a) & y = 0 \\ -\ln a & y = 1 \end{cases}$$

2 $\sigma(z) = y$

. Show that the cross-entropy is still minimized when $\sigma(z) = y$ for all training inputs (i.e. even when $y \in (0, 1)$). (Nielsen book, chapter 3)

When this is the case the cross-entropy has the value:

$$C = -\frac{1}{n} \sum_x [y \ln y + (1 - y) \ln(1 - y)]$$

take derivative of cross-entropy loss $C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)]$ w.r.t activation a

$$\frac{\partial C}{\partial a} = -\frac{1}{n} \sum_x \left(\frac{y}{a} - \frac{1 - y}{1 - a} \right)$$

plug in $a = \sigma(z) = y$, we have

$$\frac{\partial C}{\partial a} = -\frac{1}{n} \sum_x (1 - 1) = 0$$

thus C is minimized when $a = y$

3 Backpropagation

. Given the network in Figure 1, calculate the derivatives of the cost with respect to the weights and the biases and the backpropagation error equations (i.e. δ^l for each layer l) for the first iteration using the cross-entropy cost function.

Please use sigmoid activation function on hl, h2, o1, and o2.

Initial weights are colored in red, initial biases are colored in orange, the training inputs and desired outputs are in blue.

This problem aims to optimize the weights and biases through backpropagation to make the network output the desired results.

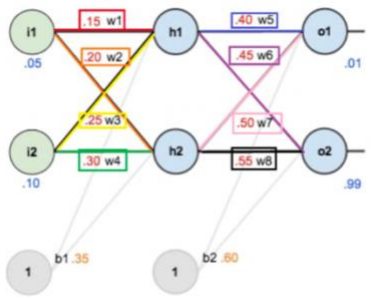


Figure 1: Simple neural network with initial weights and biases.

$\sigma'(x) = \sigma(x)(1 - \sigma(x))$

1. input $x = \begin{bmatrix} 0.05 \\ 0.10 \end{bmatrix}$

2. feedforward: for each $l = 1, 2(L = 2)$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$

$$z^1 = \begin{bmatrix} 0.3825 \\ 0.39 \end{bmatrix} \quad a^1 = \begin{bmatrix} 0.59447593 \\ 0.5962827 \end{bmatrix}$$

$$z^2 = \begin{bmatrix} 1.13593172 \\ 1.19546965 \end{bmatrix} \quad a^2 = \begin{bmatrix} 0.75693192 \\ 0.76771788 \end{bmatrix}$$

3. output error at final layer $\delta^2 = \nabla_{a^2} C \odot \sigma'(z^2) = \begin{bmatrix} 0.74693192 \\ -0.22228212 \end{bmatrix}$

$$\delta_i^L = -\left(\frac{y_i}{a_i^L} - \frac{1-y_i}{1-a_i^L}\right)\left[\sigma(z_i^L)(1 - \sigma(z_i^L))\right] = -\left(\frac{y_i}{a_i^L} - \frac{1-y_i}{1-a_i^L}\right)\left[a_i^L(1 - a_i^L)\right]$$
$$= a_i^L(1 - y_i) - (1 - a_i^L)y_i$$

4. backpropagate the error: $\delta^1 = \left[(w^L)^T \delta^2\right] \odot \sigma'(z^2) = \begin{bmatrix} 0.03452161 \\ 0.03813779 \end{bmatrix}$

5. output: gradient of cost function

$$\frac{\partial C}{\partial w_{jk}^2} = a_k^1 \delta_j^2 \Rightarrow \frac{\partial C}{\partial w^2} = \begin{bmatrix} 0.44403305 & 0.44538258 \\ -0.13214137 & -0.13254298 \end{bmatrix}$$

$$\frac{\partial C}{\partial b_j^2} = \delta_j^2 \Rightarrow \frac{\partial C}{\partial b^2} = \begin{bmatrix} 0.74693192 \\ -0.22228212 \end{bmatrix}$$

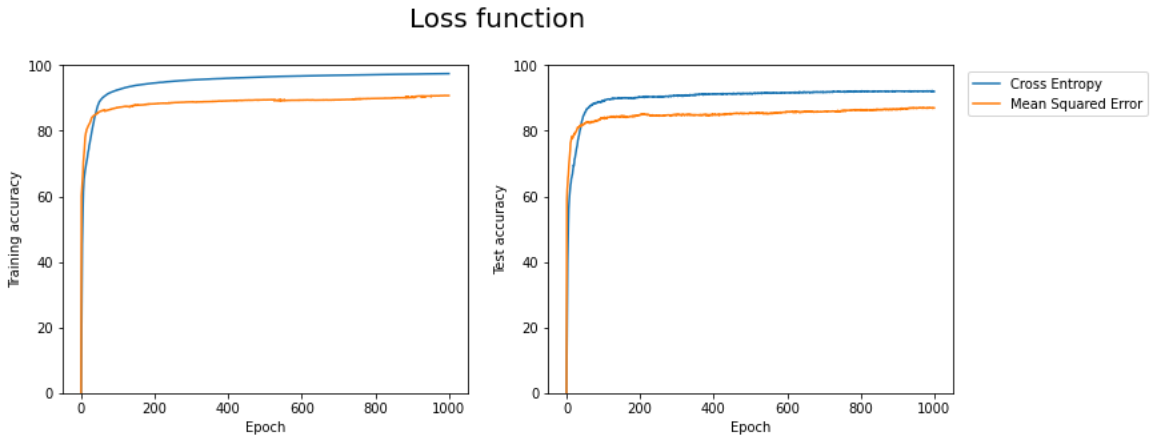
$$\frac{\partial C}{\partial w_{jk}^1} = a_k^0 \delta_j^1 \Rightarrow \frac{\partial C}{\partial w^1} = \begin{bmatrix} 0.00172608 & 0.00345216 \\ 0.00190689 & 0.00381378 \end{bmatrix}$$

$$\frac{\partial C}{\partial b_j^1} = \delta_j^1 \Rightarrow \frac{\partial C}{\partial b^1} = \begin{bmatrix} 0.03452161 \\ 0.03813779 \end{bmatrix}$$

Problem 4 MNIST

1 Loss function: Cross Entropy VS. Mean Squared Error

Create a plot of the training accuracy vs epoch for each loss function (2 lines, 1 plot)
Create a plot of the test accuracy vs epoch for each loss function (2 lines, 1 plot)



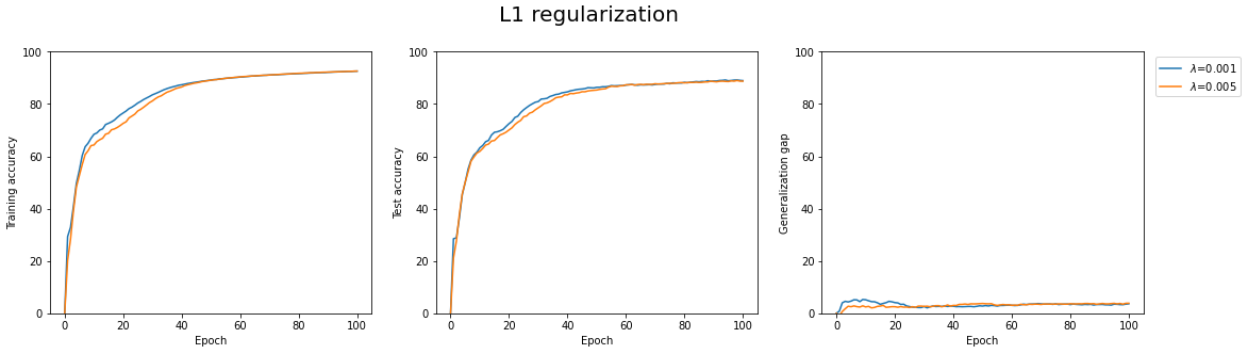
- Which loss function converges fastest? Which achieves the highest test accuracy? Provide some rational as to the observed differences.

  Cross entropy loss converges fastest and achieves the highest test accuracy, because it takes labels as discrete variables rather than MSE loss takes labels as continuous variables which is too strict.

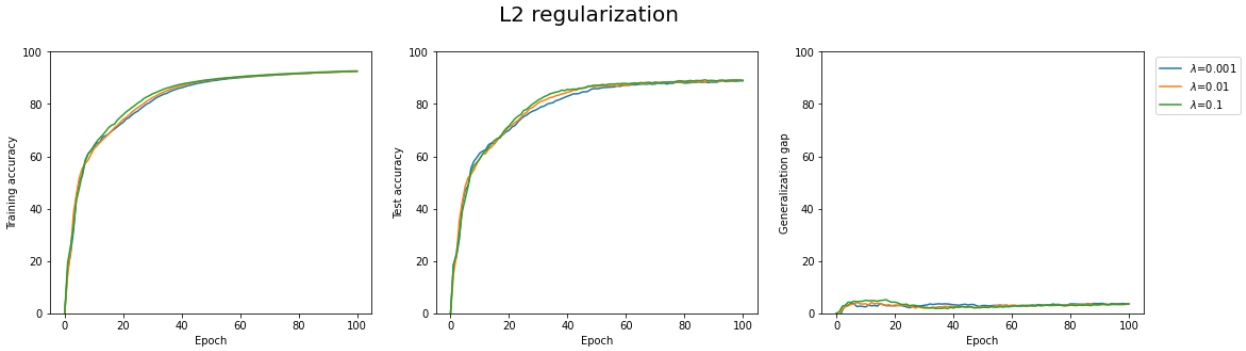
2 Regularization

Results below choosing Cross entropy loss as loss function.

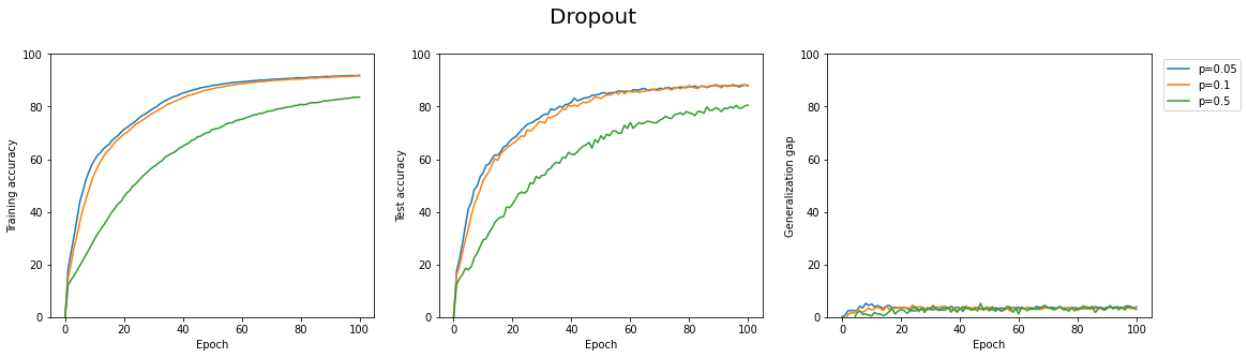
- Implement L1 regularization and train the model using $\lambda \in \{0.001, 0.005\}$. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each λ. (3 plots, 2 lines each).



- Implement L2 regularization and train the model using $\lambda \in \{0.001, 0.01, 0.1\}$. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each λ. (3 plots, 3 lines each).



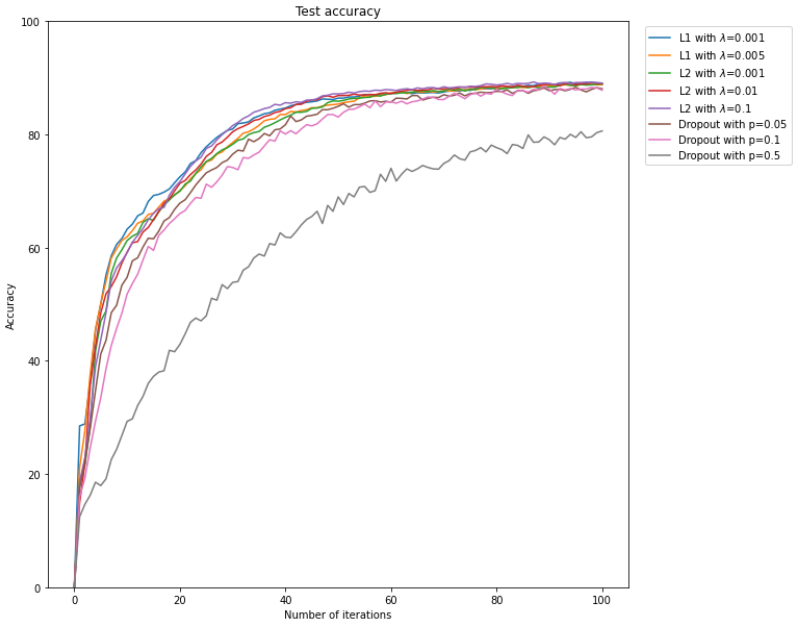
- Apply dropout to both hidden layers and train the model using $p \in \{0.05, 0.1, 0.5\}$. Create a plot of the train accuracy, test accuracy, and generalization gap vs epoch for each p value (3 plots, 3 lines each)



- Using the loss data you've collected so far, create a plot of the test accuracy vs epoch for each of the experiments performed for prob 4.2. (8 lines, 1 plot)

- Are the final results sensitive to each parameter? Is there any regularization method which performs best?

  Except from the performance of dropout with p=0.5 falls very behind the majority, other final results aren't very sensitive to which Lp norm regularization and λ we chose, both L1 and L2 performs good, L2 regularization with λ=0.1 performs best



Bonus

- Where does the softmax name come from? (Nielsen book, chapter 3)

  In classification problems, where softmax is used, typically there is one outcome having the maximum value, i.e. its probability is bigger than the others. So the output vector is a one-hot encoding vector with one element of 1 and all the others of 0. softmax is a function that maximize the maximum value by format of probability, the biggest possible value is 1. It is "soft" because it is differentiable at all points for all elements of the input vector compared to the max function $max(0,x)$ which is continuous but isn't differentiable at x=0.

Optional: Alternate equations of backpropagation

1. Error at final layer

- Show that error in final layer $\delta^L = \nabla_{a^L} C \odot \sigma'(z^L)$ can be written as $\delta^L = \Sigma'(z^L)\nabla_{a^L} C$

  where $\Sigma'(z^L)$ is a square matrix whose diagonal entries are the values $\sigma'(z^L)$ and whose off-diagonal entries are 0.

$$\nabla_a C \odot \sigma'(z^L) = \begin{pmatrix} \frac{\partial C}{\partial a_1^L} \times \sigma'(z_1^L) \\ \frac{\partial C}{\partial a_2^L} \times \sigma'(z_2^L) \\ \vdots \\ \frac{\partial C}{\partial a_n^L} \times \sigma'(z_n^L) \end{pmatrix}$$

$$\Sigma'(z^L)\nabla_a C = \begin{bmatrix} \sigma'(z_1^L) & & 0 \\ & \sigma'(z_2^L) & \\ & & \ddots \\ 0 & & \sigma'(z_n^L) \end{bmatrix} \begin{pmatrix} \frac{\partial C}{\partial a_1^L} \\ \frac{\partial C}{\partial a_2^L} \\ \vdots \\ \frac{\partial C}{\partial a_n^L} \end{pmatrix} = \begin{pmatrix} \sigma'(z_1^L)\frac{\partial C}{\partial a_1^L} \\ \sigma'(z_2^L)\frac{\partial C}{\partial a_2^L} \\ \vdots \\ \sigma'(z_n^L)\frac{\partial C}{\partial a_n^L} \end{pmatrix} = \nabla_a^L C \odot \sigma'(z^L)$$

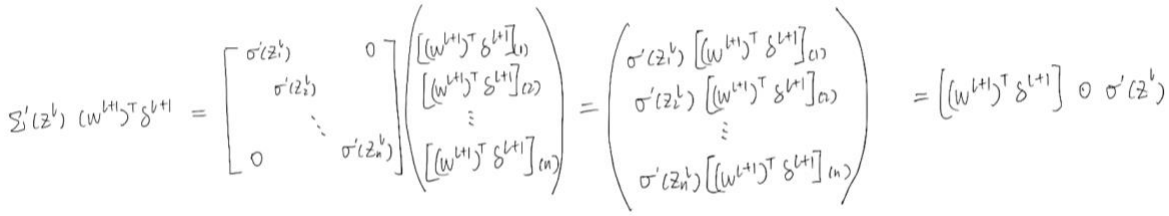
2. Error at layer l

show that error in layer l

$$\delta^l = \left[(w^{l+1})^T \delta^{l+1}\right] \odot \sigma'(z^l)$$

can be written as

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



3. Error at layer l: combine 1 and 2

- By combining the results from 1 and 2, show that

$$\delta^l = \Sigma'(z^l)(w^{l+1})^T \dots \Sigma'(z^{L-1})(w^L)^T \Sigma'(z^L) \nabla_{a^L} C$$