گزارش تمرین ۱ درس هوش محاسباتی **علی صداقی ۹۷۵۲۱۳۷۸**

مدل استفاده شده در سوال ۱ و ۲ دقیقا همانند همدیگر هستند و تنها تفاوت این دو بخش در فراخوانی کردن تابع perceptron با ورودی متفاوت می‌باشند.

در ادامه همین مدل تعمیم می‌آید و قادر خواهد بود به تعداد دلخواه لایه که هر لایه می‌تواند تعداد دلخواهی نورون داشته باشد.

همچنین در مدل چند لایه، لایه های میانی دارای تابع فعال‌سازی ReLU و لایه‌ی آخر (خروجی) بر اساس این که مسئله Binary-Classification یا Multi-Classification هست دارای تابع فعال‌سازی Sigmoid یا Softmax می‌باشد.

ابتدا مدل را برای پرسپترون یک لایه با یک مثال در نظر می‌گیریم سپس آن‌را برای m مثال تعمیم می‌دهیم سپس مدل را چند لایه برای یک ورودی و در نهایت یک مدل چند لایه با m مثال می‌سازیم.

ویژگی کلی مدل ها:

- ✓ Batch Gradient Descent
- ✓ ReLU as activation function of hidden layers
- ✓ Sigmoid as activation function of output layer for binary classification problem
- ✓ Softmax as activation function of output layer for multi classification problem
- ✓ Cross entropy error for computing loss and costs

❖ مدل یک لایه برای یک مثال:

$x: one\ training\ example$

$y: label\ of\ the\ training\ example\ 0 \leq y \leq 1$

$n: number\ of\ input\ features$

$x_i: i-th\ feature\ of\ x\ example\ 1 \leq i \leq n$

$w_i: weight\ corresponding\ with\ i-th\ feature$

$b: bias\ term$

$\alpha: learning\ rate$

Forward Propagation

z: *linear activation*

$$z = \sum_{i=1}^{n} w_i x_i + b$$

a: *sigmoid activation*: *predicted label for* x

$$a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$l(a, y)$: *cross entropy loss of our prediction*

$$l(a, y) = -(y \log a + (1 - y) \log(1 - a))$$

Backward Propagation

da: *derivative of loss with respect of sigmoid activation*

$$da = \frac{\partial l(a, y)}{\partial a} = -\left(\frac{y}{a} - \frac{1 - y}{1 - a}\right) \Rightarrow da = \frac{1 - y}{1 - a} - \frac{y}{a}$$

$\acute{\sigma}(z)$: *derivative of sigmoid function*

$$\acute{\sigma}(z) = \frac{d\sigma(z)}{dz} = \frac{-e^{-z}}{(1 + e^{-z})^2} = \frac{1}{(1 + e^{-z})} \frac{1 - (1 + e^{-z})}{(1 + e^{-z})} \Longrightarrow \acute{\sigma}(z) = \sigma(z)(1 - \sigma(z))$$

dz: *derivative of loss with respect of linear activation*

$$dz = \frac{\partial l(a, y)}{\partial z} = \frac{\partial l(a, y)}{\partial a} \frac{\partial a}{\partial z} = \left(\frac{1 - y}{1 - a} - \frac{y}{a}\right) a(1 - a) = a(1 - y) - (1 - a)y$$

$$= a - ay - y + ay \Longrightarrow dz = a - y$$

dw_i: *derivative of loss with respect of* $i - th$ *feature weight*

$$dw_i = \frac{\partial l(a, y)}{\partial w_i} = \frac{\partial l(a, y)}{\partial z} \frac{\partial z}{\partial w_i} = (a - y)x_i$$

db: *derivative of loss with respect of bias term*

$$db = \frac{\partial l(a, y)}{\partial b} = \frac{\partial l(a, y)}{\partial z} \frac{\partial z}{\partial b} = a - y$$

Updating parameters

$$w_i := w_i - \alpha dw_i$$

$$b := b - \alpha db$$

❖ مدل یک لایه برای m مثال: از نسخه Vectorize شده این مدل در سوالات ۱ و ۲ استفاده شده است. پیاده سازی Vectorize در نوتبوک موجود است.

$m: number\ of\ training\ examples$

$x^{(j)}: j-th\ training\ example\ 1 \leq j \leq m$

$y^{(j)}: j-th\ training\ example\ label\ 0 \leq y^{(j)} \leq 1$

$n: number\ of\ input\ features$

$x_i: i-th\ feature\ of\ x\ example\ 1 \leq i \leq n$

$w_i: weight\ corresponding\ with\ i-th\ feature$

$b: bias\ term$

$\alpha: learning\ rate$

Forward Propagation

$$z^{(j)} = \sum_{i=1}^{n} w_i x_i^{(j)} + b$$

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

$$l\left(a^{(j)}, y^{(j)}\right) = -\left(y^{(j)} \log a^{(j)} + \left(1 - y^{(j)}\right) \log\left(1 - a^{(j)}\right)\right)$$

J: *cross entropy cost function for m training examples*

$$J = \frac{1}{m} \sum_{j=1}^{m} l(a^{(j)}, y^{(j)})$$

Backward Propagation

$$da^{(j)} = \frac{\partial l(a^{(j)}, y^{(j)})}{\partial a^{(j)}} = -\left(\frac{y^{(j)}}{a^{(j)}} - \frac{1 - y^{(j)}}{1 - a^{(j)}}\right) \Rightarrow da^{(j)} = \frac{1 - y^{(j)}}{1 - a^{(j)}} - \frac{y^{(j)}}{a^{(j)}}$$

$$dz^{(j)} = \frac{\partial l\left(a^{(j)}, y^{(j)}\right)}{\partial z^{(j)}} = \frac{\partial l\left(a^{(j)}, y^{(j)}\right)}{\partial a^{(j)}} \frac{\partial a^{(j)}}{\partial z^{(j)}} = \left(\frac{1 - y^{(j)}}{1 - a^{(j)}} - \frac{y^{(j)}}{a^{(j)}}\right) a^{(j)}\left(1 - a^{(j)}\right)$$

$$= a^{(j)}\left(1 - y^{(j)}\right) - \left(1 - a^{(j)}\right) y^{(j)} \Rightarrow dz^{(j)} = a^{(j)} - y^{(j)}$$

$$dw_i^{(j)} = \frac{\partial l\left(a^{(j)}, y^{(j)}\right)}{\partial w_i} = \frac{\partial l\left(a^{(j)}, y^{(j)}\right)}{\partial z^{(j)}} \frac{\partial z^{(j)}}{\partial w_i} = \left(a^{(j)} - y^{(j)}\right) x_i^{(j)}$$

$$db^{(j)} = \frac{\partial l\left(a^{(j)}, y^{(j)}\right)}{\partial b^{(j)}} = \frac{\partial l\left(a^{(j)}, y^{(j)}\right)}{\partial z^{(j)}} \frac{\partial z^{(j)}}{\partial b^{(j)}} = a^{(j)} - y^{(j)}$$

$$dW_i = \frac{1}{m} \sum_{j=1}^{m} dw_i^{(j)}$$

$$dB = \frac{1}{m} \sum_{j=1}^{m} db^{(j)}$$

Updating parameters

$$w_i := w_i - \alpha dW_i$$

$$b := b - \alpha dB$$

❖ مدل L لایه برای m مثال

$m: number\ of\ training\ examples$

$l: number\ of\ layers$

- ✓ $l = 0: input\ layer$
- ✓ $l = L: output\ layer$

$n^{[l]}: number\ of\ units(neurons)\ in\ layer\ l$

- ✓ $n^{[0]} = n_x: number\ of\ input\ features$
- ✓ $n^{[L]} = n_y: number\ of\ output\ units: number\ of\ classes$

$g^{[l]}: activation\ function\ in\ layer\ l$

- ✓ $1 \leq l \leq L-1: ReLU$
- ✓ $L: sigmoid\ or\ softmax$

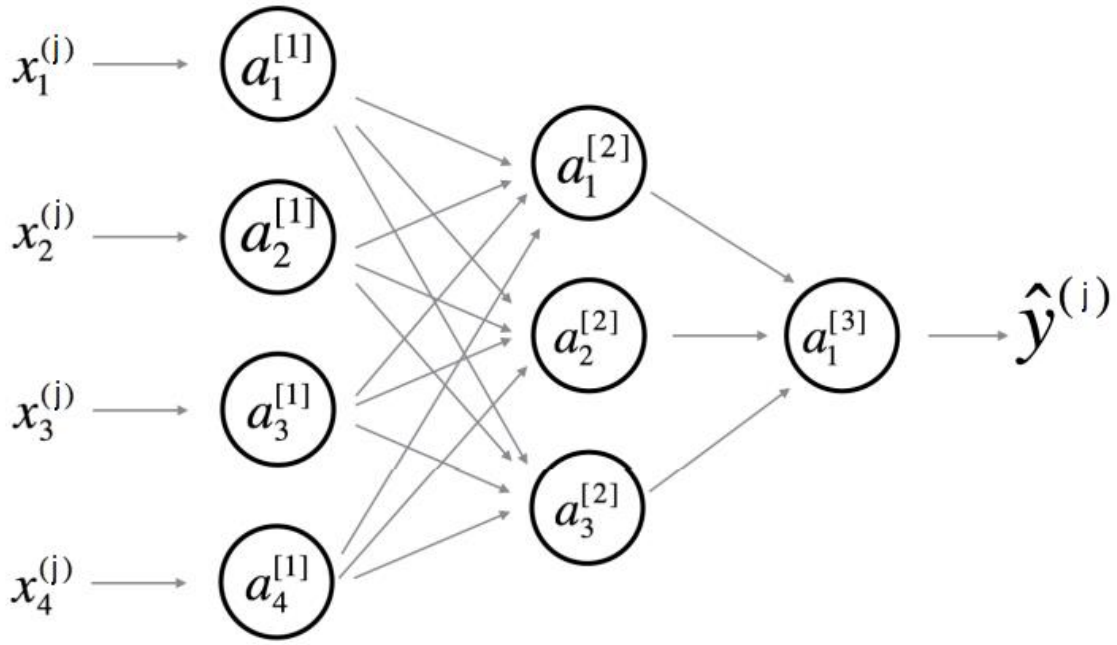
$a_i^{[l]}: output\ of\ neuron\ i\ in\ layer\ l$

- ✓ $a_i^{[0]}: x_i$

$w_{ik}^{[l]}: weight\ of\ neuron\ i\ in\ layer\ l$

$b_i^{[l]}: bias\ term\ of\ neuron\ i\ in\ layer\ l$

$\alpha: learning\ rate$



j: training example j

Forward Propagation

$$Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$$

$$A^{[1]} = g^{[1]}(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = g^{[2]}(Z^{[2]})$$

...

$$Z^{[L]} = W^{[L]}A^{[L-1]} + b^{[L]}$$

$$A^{[L]} = g^{L}(Z^{[L]})$$

$$J = \frac{1}{m}\sum_{j=1}^{m} l(A^{[L]}, Y)$$

Backward Propagation

ابتدا برای دو لایه ***Backward propagate*** می‌کنیم و رابطه بازگشتی به‌دست آمده را برای حالت چندین لایه تعمیم می‌دهیم. از روابط به دست آمده در حالت های تک لایه نیز استفاده می‌کنیم و فقط فعالسازی و مشتق فعال سازی را مستقیما تعریف نمی‌کنیم.

$$dZ^{[L]} = dA^{[L]} * \acute{g}^{[L]}(Z^{[L]})$$

$$dW^{[L]} = \frac{1}{m} dZ^{[L]}.A^{[L]}$$

$$db^{[L]} = \frac{1}{m} dZ^{[L]}$$

$$dA^{[L-1]} = W^{[L-1]}dZ^{[L]}$$

...

$$dZ^{[l]} = dA^{[l]} * \acute{g}^{[l]}(Z^{[l]})$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]}.A^{[l]}$$

$$db^{[l]} = \frac{1}{m} dZ^{[l]}$$

$$dA^{[l-1]} = W^{[l]}dZ^{[l]}$$

Updating parameters

$$W^{[l]} := W^{[l]} - \alpha dW^{[l]}$$

$$b^{[l]} := b^{[l]} - \alpha db^{[l]}$$

توابع ReLU و Softmax

همانند تابع Sigmoid اکنون توابع ReLU و Softmax را تعریف کرده و مشتق آن ها را محاسبه می‌کنیم و هر جا که لازم بود به جای $g^{[l]}$ قرار می‌دهیم.

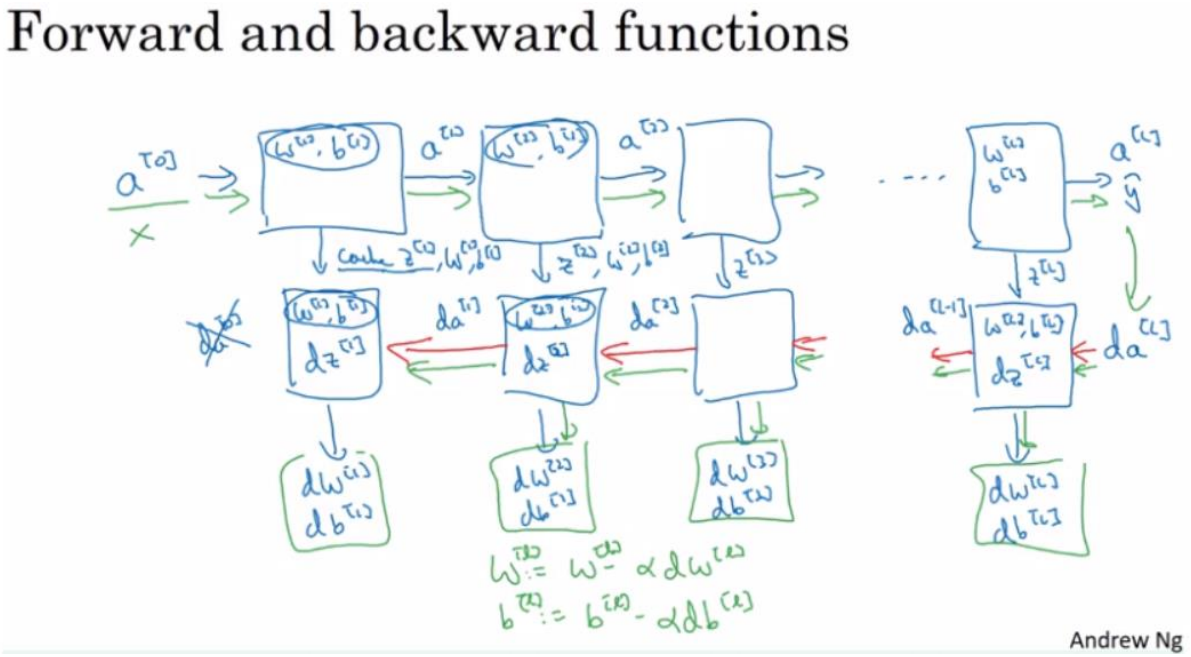
$$ReLU: g(z) = \max(0, z)$$
$$\acute{g}(z) =$$
$$0 \; if \; z < 0$$
$$1 \; if \; z \geq 0$$

$$Softmax: g(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$
$$g(\acute{z_\imath}) =$$
$$g(z_\imath)\big(1 - g(z_\imath)\big) \; \imath f \; \imath = \jmath$$
$$-g(z_\imath)g\big(z_\jmath\big) \; \imath f \; \imath \neq \jmath$$

چگونگی انتشار Forward و Backward در شکل زیر نشان داده شده است که از لایه 0 به لایه Lجلو می‌رویم و در مقدار Cost را محاسبه می‌کنیم، سپس از لایه آخر شروع به مشتق گیری می‌کنیم و در هر مرحله از مقادیر Cache تولید شده از Forward pass استفاده می‌کنیم. به هنگام Backward pass مقادیر گرادیان مربوط به همان لایه نیز محاسبه میکنیم و در پایان هر Epoch پارامتر ها را آپدیت می‌کنیم.

در Forward pass به مقادیر w و b دسترسی داریم و مقدار Z را Cache می‌کنیم و مقدار a را نیز به لایه بعدی Pass می‌دهیم.

در Backward pass با استفاده از مقادیر z و w و b که از Cache به ما رسیده مقادیر dw و db مربوط به آن لایه را حساب کرده و مقدار daرا نیز به لایه قبلی Pass می‌دهیم.



شکل از کورس دیپ لرنینگ Andrew Ng در کورسرا برداشته شده است.

سوال ۲)

از مدل یک لایه به دست آمده در صفحات قبلی استفاده کردیم تنها ورودی دادن آن متفاوت می‌باشد.

داده های ورودی مقادیر بزرگی دارند و این باعث می‌شود مقدار وزن ها و مقدار تابع Sigmoid برابر یک شود و مدل ارور دهد بنابراین لازم است داده ها نرمال و استاندارد شوند. نحوه نرمال سازی از فرم numpy فرمول زیر به‌دست آمده است.

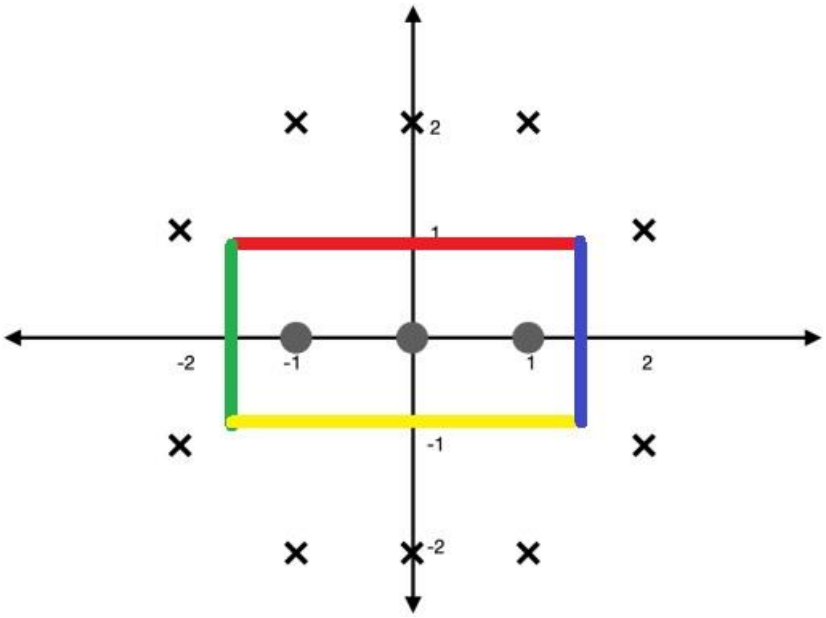
```
x_train = x - np.mean(x, axis=1, keepdims=True)
x_train /= np.std(x_train, axis=1, keepdims=True)
```

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}$$

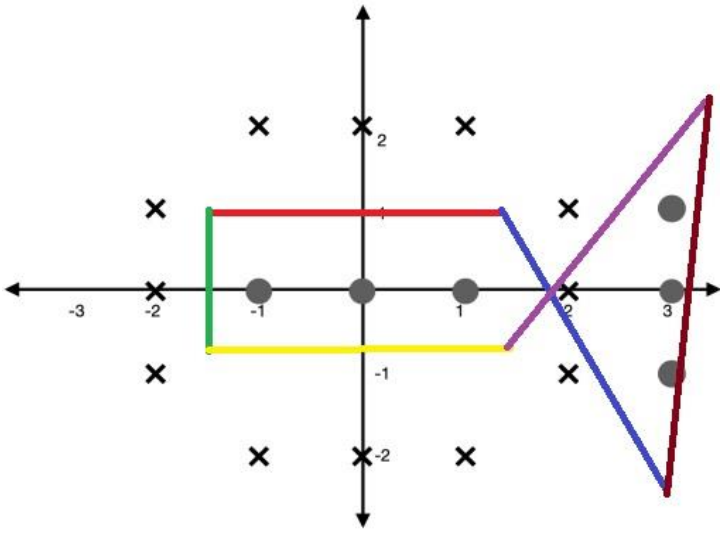
سوال ۳)

بله در مدل های Madaline هر Adaline (به صورت Parallel) جهت تولید یک خط می‌باشد و در نهایت می‌توان با تقاطع و اشتراک گیری، یک ناحیه غیر خطی را یاد گرفت و دسته‌بندی غیرخطی داشت. اما یک شرط وجود دارد که نواحی ما جداپذیر باشند (Separable Regions) (با توجه به Convex بودن ناحیه). توضیحات و مثال ها در موارد الف و ب آورده شده است.

الف) می‌توان با چهار خط قرمز، سبز، زرد، آبی یک ناحیه Convex تولید کرد و تفکیک پذیر است. شبکه Madaline نیز باید از ۴ Adaline که بیانگر هر رنگ می‌باشند و به صورت Parallel هستند و همچنین یک واحد AND کننده جهت پیدا کردن ناحیه درونی این ۴ خط.



ب) این مسئله را نمی‌توان با Madaline تفکیک کرد زیرا هیچ حالت Convexیی برای یافتن یک ناحیه وجود ندارد مثلا شکل زیر یکی از حالت ها با استفاده از ۶ خط است که واضح است که Convex نیست. همچنین یکی از نقاط X همواره در ناحیه O ها قرار می‌گیرد و نمی‌توان تفکیک کامل با استفاده از خطوط داشت.



سوال ۴)

با توجه به اینکه دیتاست MNIST یک مسئله چندکلاسی هست لایه آخر ما باید دارای 10 نورون باشد چون ۱۰ عدد داریم.

همچنین لایه 0 و ورودی باید به تعداد Input feature ها باشد یعنی: 784 = 28*28

در حل این سوال ابتدا داده ها با تقسیم بر ۲۵۵ شدن نرمالیزه شده اند.

ورودی Y نیز با استفاده از تابع to categorical و روش one hot کد شده است تا قابل استفاده در شبکه مولتی کلاس باشد.

ساختار شبکه به صورت زیر می‌باشد:

تمامی لایه ها به صورت Dense و Fully connected می‌بشاند.

Input Layer (layer 0): 28*28 units

Layer 1 (Hidden): 128 neurons with ReLU as activation function

Layer 2 (Hidden): 128 neurons with ReLU as activation function

Layer 3 (Output): 10 neurons with Softmax as activation function

با توجه به اینکه تعداد داده ها در این دیتا ست ۶۰هزار می‌باشد از حالت mini-batch استفاده کردیم تا زمان هر ایپاک طولانی نشود.

Batch size: 128

Epoch: 20

همچنین با توجه به اینکه مسئله مولتی کلاس می‌باشد از تابع خطا Categorical Cross Entropy استفاده کردیم.

برای جلوگیری از Over-fit کردن مدل نیز مقدار Validation Split را 0.2 گذاشتیم تا از بخشی از داده Train جهت Hold out کردن استفاده کرده باشیم.

از تابع evaluate نیز برای ارزیابی داده های تست استفاده کردیم.