CMPE 258, Deep Learning

Deep Neural Network

Feb 27, 2018

DMH 149A

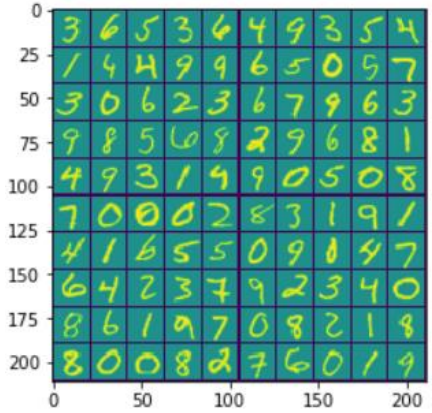
Taehee Jeong

Ph.D., Data Scientist

SJSU SAN JOSÉ STATE UNIVERSITY

Assignment_3

The due day is 3/5 (Monday).



Neural Network with one hidden layer

Data

Subset of MNIST handwritten digit

Each row is a 20 pixel by 20 pixel grayscale image of the digit.

The 20 by 20 pixels is unrolled into a 400-dimensional vector.

The last column ‘y’ is the label for the row.

	0	1	2	3	4	5	6	7	8	9	...	391	392	393	394	395	396	397	398	399	y
0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	5
1	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	9
2	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	7
3	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	6
4	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	5

Assignment_3

1. (40pts) Define functions
2. (5pts) Split data
3. (5pts) Initialize parameters
4. (20pts) Neural Network model with 1 hidden layer
5. (10pts) Predictions
6. (20pts) Optimization

Assignment_3

1. **(40pts) Define functions**

One-hot encoding

Sigmoid

Forward propagation

Backward propagation

Gradient descent

Softmax

2. (5pts) Split data

Split each data (Train & Test) set as input (x) and output (y) set.

Input set is the columns starting 0 to 399.

Output set is the column of ‘y’.

Assignment_3

3. (5pts) Initialize parameters

Please use np.random.seed(1) when weight coefficients is initialized.

Please set as zeros for bias terms.

4. (20pts) Neural Network model with 1 hidden layer

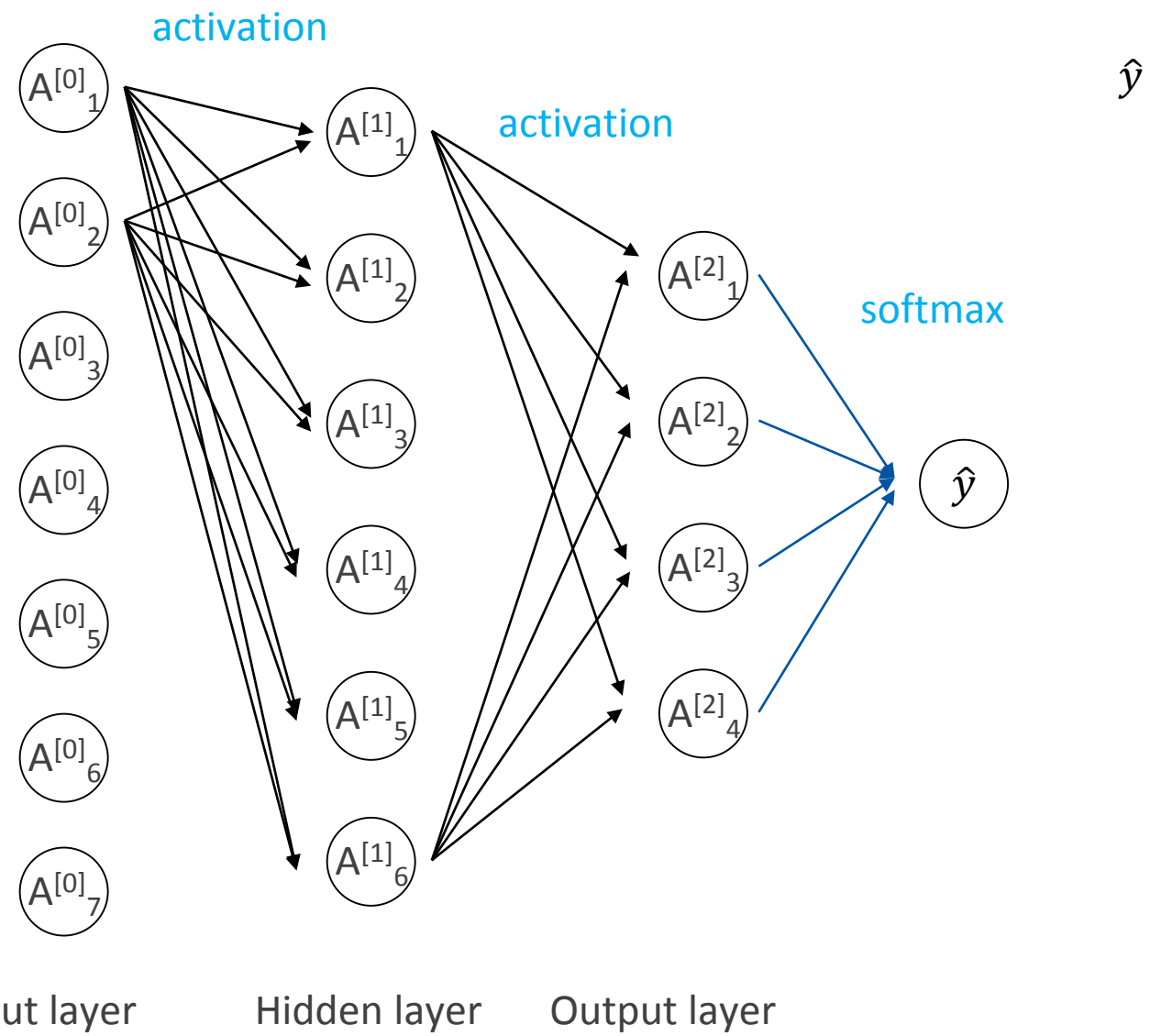
Please build neural network model using input layer (400 neurons), 1 hidden layer (25 neurons), and output layer (10 neurons) using training data set.

5. (10pts) Predictions

Please predict digit using softmax function.

Please calculate accuracy for the prediction using training data set and testing data set.

Model(Neural network)



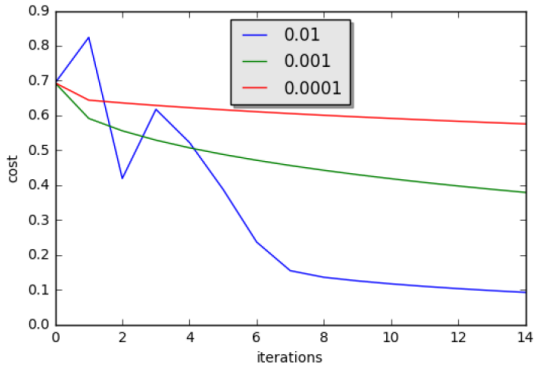
Assignment_3

6. (20pts) Optimization

Please optimize your model using various learning rate and number of iteration.

Please plot cost versus number of iteration with different learning rate for training data set.

Please print out the optimized accuracy for testing data set.



<Deep learning, Andrew Ng>

Model optimizing procedure

- Initialize parameters
- Run the optimization loop
  - Forward propagation to compute the loss function
  - Backward propagation to compute the gradients with respect to the loss function
  - Using the gradients, update your parameter with the gradient descent update rule
- Return the learned parameters

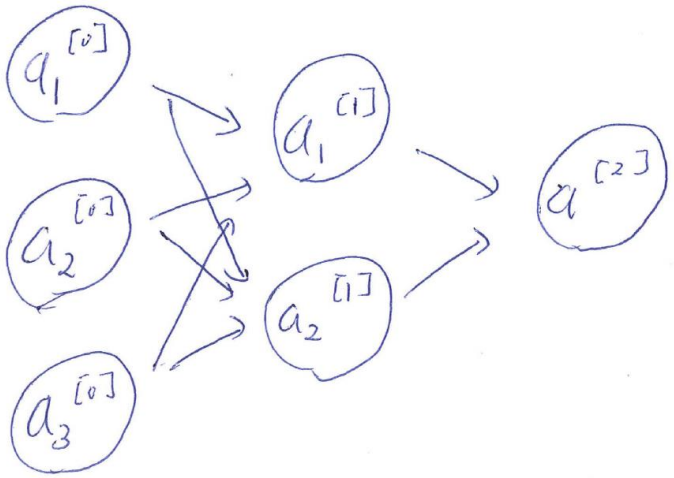
<deep learning, Andrew Ng>

Optimization loop

1. forward pass
2. cost computation
3. backward pass
4. parameter update

<deep learning, Andrew Ng>

Neural Network: 1 hidden layer



Input layer Hidden layer Output layer

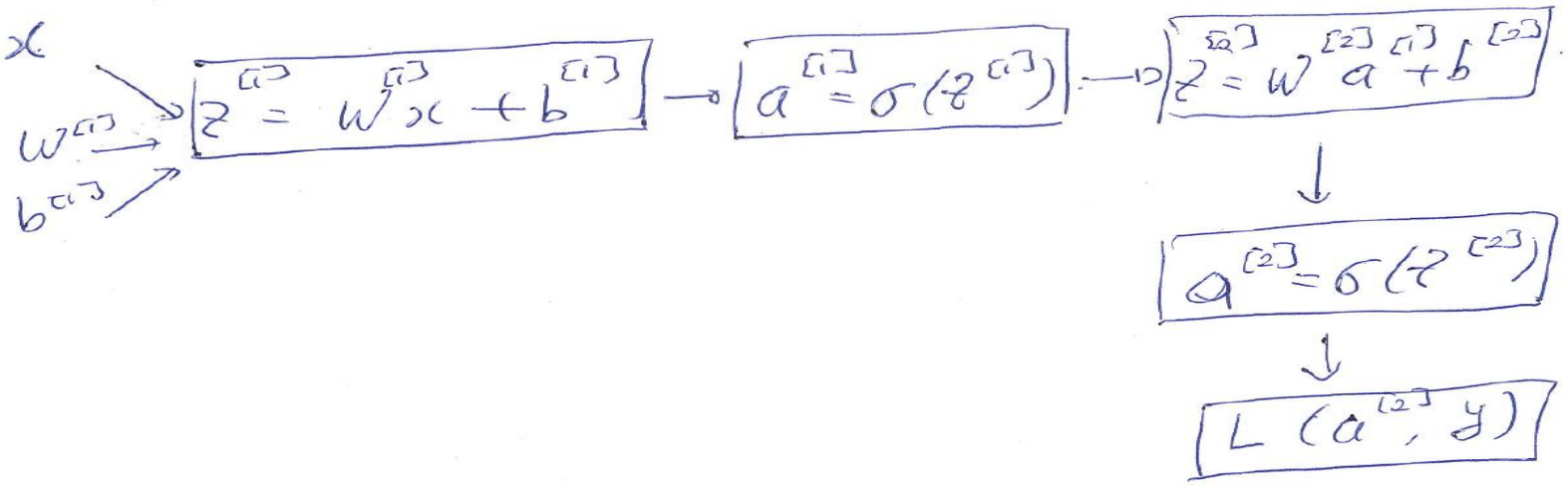
Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$

Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$

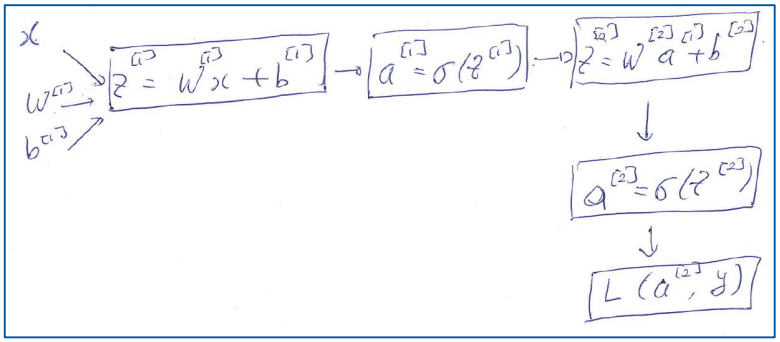
$$= \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$$

$\hat{y} = a^{[2]}$

Neural Network: 1 hidden layer



Derivative of $dL/dW^{[2]}$



$$L(\hat{y}, y) = -[y * \log(a^{[2]}) + (1-y) * \log(1-a^{[2]})]$$

$$da^{[2]} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$$

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = (a^{[2]} - y)\, a^{[1]} = dz^{[2]} \cdot a^{[1]}$$

$$db^{[2]} = dz^{[2]} = a^{[2]} - y$$

*Element wise multiplication

Derivative of $dL/dZ^{[1]}$

$$da^{[1]} = \frac{\partial L}{\partial a^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}}$$

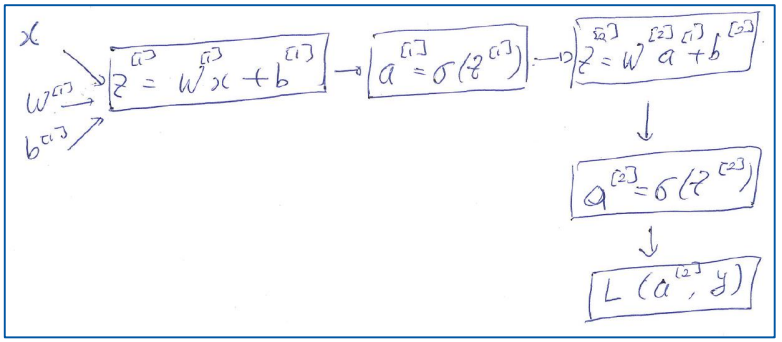
$$= dz^{[2]} \cdot w^{[2]}$$

$$= (a^{[2]} - y) \cdot w^{[2]}$$

$$dz^{[1]} = \frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}}$$

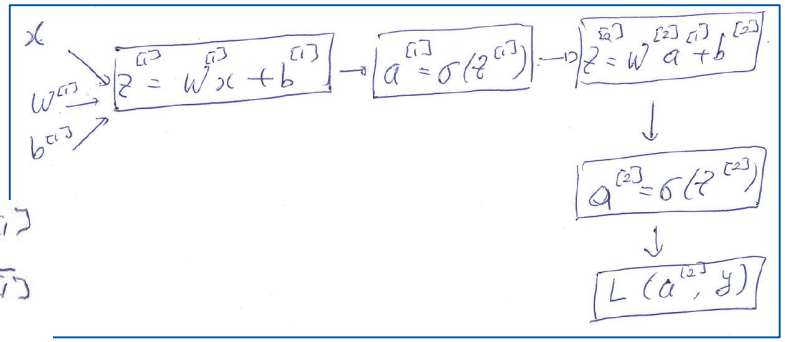
$$= da^{[1]} * a^{[1]} * (1 - a^{[1]})$$

$$= (a^{[2]} - y) \cdot w^{[2]} * a^{[1]} * (1 - a^{[1]})$$



*Element wise multiplication

Derivative of dL/dW[1]



$$dW^{[1]} = \frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

$$= dz^{[1]} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

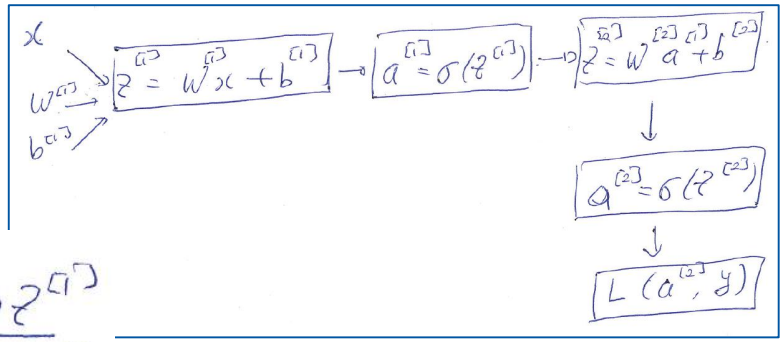
$$= (a^{[2]} - y) \cdot W^{[2]} * a^{[1]} * (1 - a^{[1]}) \cdot x$$

$$= dz^{[1]} \cdot x$$

$$= dz^{[1]} \cdot a^{[0]}$$

*Element wise multiplication

Derivative of $dL/db^{[1]}$



$$db^{[1]} = \frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

$$= dz^{[1]} \cdot \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

$$= dz^{[1]}$$

Forward propagation

$$z^{[1]} = w^{[1]} \cdot x + b^{[1]}$$

$$= w^{[1]} \cdot A^{[0]} + b^{[1]}$$

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

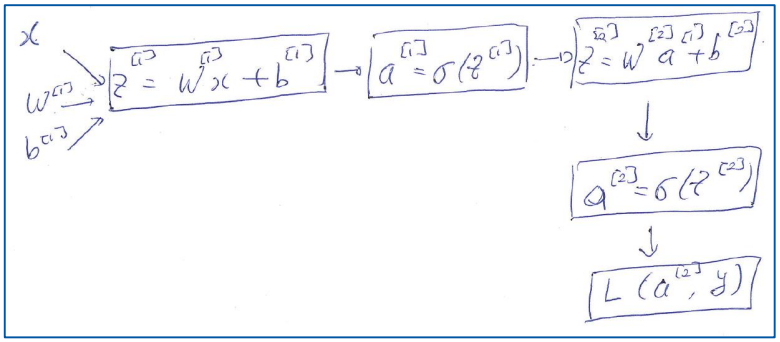
$$= \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]} \cdot A^{[1]} + b^{[2]}$$

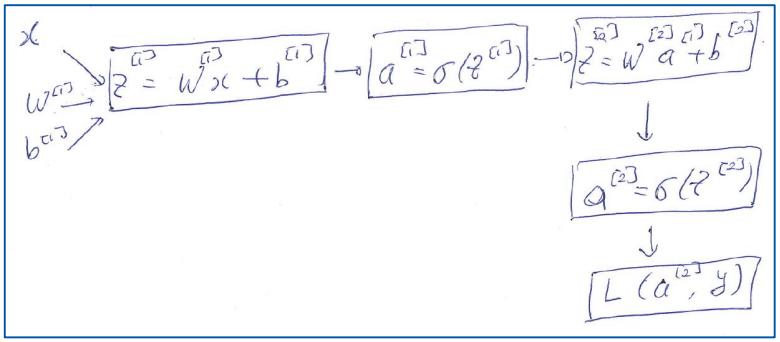
$$A^{[2]} = g^{[2]}(z^{[2]})$$

$$= \sigma(z^{[2]})$$

$$= \hat{Y}$$



Backward propagation



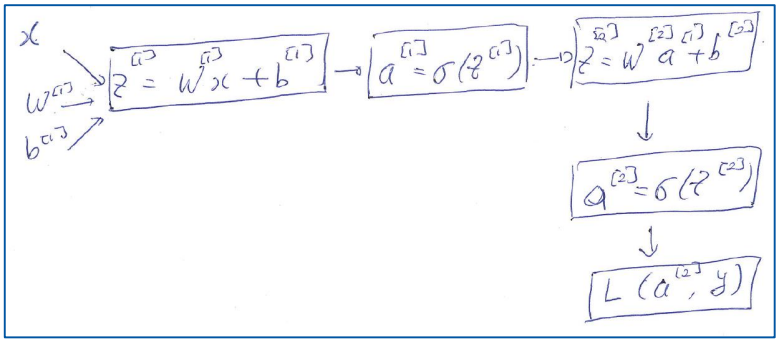
$$dA^{[2]} = -\frac{Y}{A^{[2]}} + \frac{1-Y}{1-A^{[2]}} \rightarrow \text{skip}$$

$$dz^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]}$$

$$db^{[2]} = \frac{1}{m} \text{np.sum}(dz^{[2]}, \text{axis}=1, \text{keepdims}=\text{True})$$

Backward propagation



$$dA^{[1]} = dz^{[2]} \cdot W^{[2]}$$

$$dz^{[1]} = dA^{[1]} * A^{[1]} * (1 - A^{[1]})$$

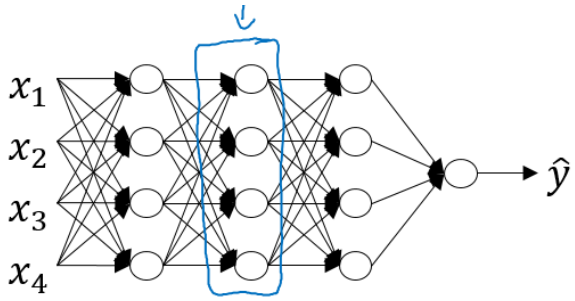
$$dW^{[1]} = \frac{1}{m} dz^{[1]} \cdot A^{[0]}$$

$$= \frac{1}{m} dz^{[1]} \cdot X$$

$$db^{[1]} = \frac{1}{m} np.sum(dz^{[1]}, axis=1, keepdims = True)$$

*Element wise multiplication

Forward and backward functions



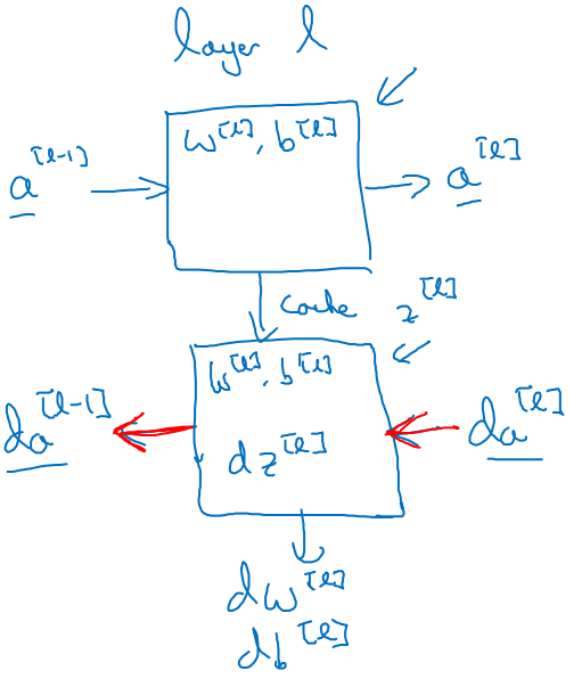
Layer l: $W^{[l]}, b^{[l]}$

→ Forward: Input $a^{[l-1]}$, output $a^{[l]}$

$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ cache $z^{[l]}$

$a^{[l]} = g^{[l]}(z^{[l]})$

→ Backward: Input $da^{[l]}$, cache($z^{[l]}$), output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$



<deep learning, Andrew Ng>

Forward and backward propagation

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y}
\end{aligned}
$$

$$
\begin{aligned}
dZ^{[L]} &= A^{[L]} - Y \\
dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]^T} \\
db^{[L]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True) \\
dZ^{[L-1]} &= dW^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
dZ^{[1]} &= dW^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]^T} \\
db^{[1]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)
\end{aligned}
$$

<deep learning, Andrew Ng>

Initialize weights

Neural network with hidden layers

Weights should be initialized randomly rather than to all zeros.

If weights are initialized to be zero, then, each neuron in the first hidden layer will perform the same computation. So even after multiple iterations of gradient descent, each neuron in the layer will be computing the same thing as other neurons.

<deep learning, Andrew Ng>

Initialize weights

Pitfall: all zero initialization

Assume that approximately half of the weights will be positive and half of them will be negative.

Set all the initial weights to zero.

If every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates.

In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same.

<cs231n, Stanford university>

Initialize weights

Small random numbers

Want the weights to be very close to zero, but not identically zero.

Assume that the neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the full network.

Initialize the weights of the neurons to small numbers and refer to doing so as symmetry breaking.

W = 0.01* np.random.randn(D,H)

randn samples from a zero mean, unit standard deviation Gaussian.

<cs231n, Stanford university>

Initialize weights

Logistic regression

Logistic regression does not have a hidden layer.

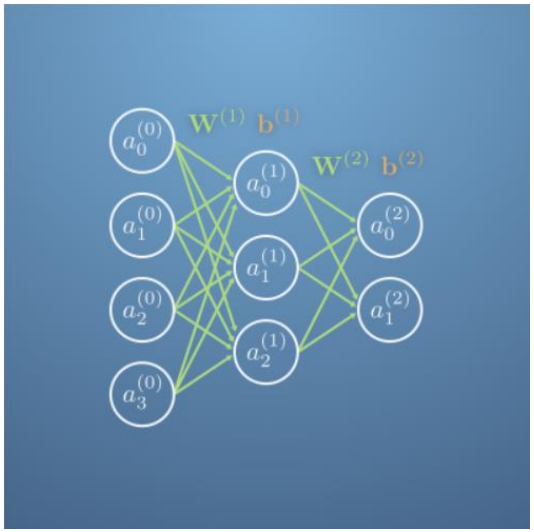
If you initialize the weights to zeros, the first example x fed in the logistic regression will output zero but the derivatives of the logistic regression depend on the input x (because there is no hidden layer) which is not zero.

So at the second iteration, the weights values follow x's distribution and are different from each other if x is not a constant vector.

<deep learning, Andrew Ng>

Weight Matrix

- Input layer : 4 neurons
- Hidden layer : 3 neurons



<Mathematics for Machine Learning>

$$a_1^{[1]} = \sigma\left(Z_1^{[1]}\right)$$
$$a_2^{[1]} = \sigma\left(Z_2^{[1]}\right)$$
$$a_3^{[1]} = \sigma\left(Z_3^{[1]}\right)$$

$$\begin{bmatrix} W_{11} W_{12} W_{13} W_{14} \\ W_{21} W_{22} W_{23} W_{24} \\ W_{31} W_{32} W_{33} W_{34} \end{bmatrix}$$

$$Z_1^{[1]} = b_1^{[1]} + W_{11}^{[1]} \cdot a_1^{[0]} + W_{12}^{[1]} \cdot a_2^{[0]} + W_{13}^{[1]} \cdot a_3^{[0]} + W_{14}^{[1]} \cdot a_4^{[0]}$$
$$Z_2^{[1]} = b_2^{[1]} + W_{21}^{[1]} \cdot a_1^{[0]} + W_{22}^{[1]} \cdot a_2^{[0]} + W_{23}^{[1]} \cdot a_3^{[0]} + W_{24}^{[1]} \cdot a_4^{[0]}$$
$$Z_3^{[1]} = b_3^{[1]} + W_{31}^{[1]} \cdot a_1^{[0]} + W_{32}^{[1]} \cdot a_2^{[0]} + W_{33}^{[1]} \cdot a_3^{[0]} + W_{34}^{[1]} \cdot a_4^{[0]}$$

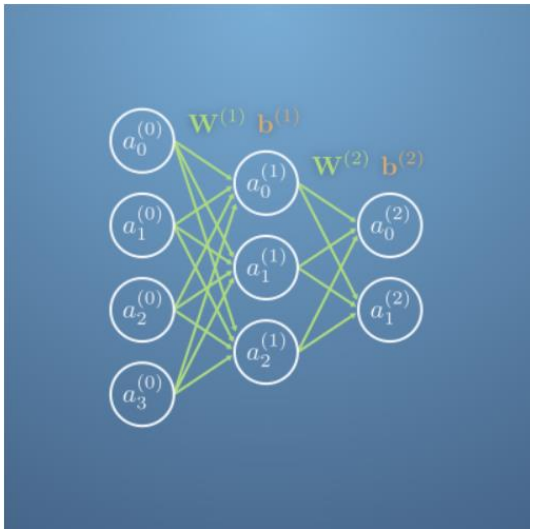
$W^{[1]}$: [3,4]

W_1[n_h,n_x]

b_1[1, n_h]

Weight Matrix

- Hidden layer : 3 neurons
- output layer : 2 neurons



<Mathematics for Machine Learning>

$$a_1^{[2]} = \sigma\left(Z_1^{[2]}\right)$$
$$a_2^{[2]} = \sigma\left(Z_2^{[2]}\right)$$
$$Z_1^{[2]} = b_1^{[2]} + W_{11}^{[2]} \cdot a_1^{[1]} + W_{12}^{[2]} \cdot a_2^{[1]} + W_{13}^{[2]} \cdot a_3^{[1]}$$
$$Z_2^{[2]} = b_2^{[2]} + W_{21}^{[2]} \cdot a_1^{[1]} + W_{22}^{[2]} \cdot a_2^{[1]} + W_{23}^{[2]} \cdot a_3^{[1]}$$

$$\begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix}$$

$W^{[2]}$: [2,3]

W_2[n_y,n_h]

b_2[1, n_y]

L2 Regularization

Most common form of regularization

Penalize the squared magnitude of all parameters directly.

Heavily penalize peaky weight vectors and prefer diffuse weight vectors.

Network to use all of its inputs a little rather that some of its inputs a lot.

During gradient descent parameter update, every weight is decayed

Linearly towards zero.

<cs231n, Stanford university>

Logistic regression with L2 Regularization

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1-y^i)\log(1-\widehat{y^i})] + \frac{\lambda}{m}\sum_{j=1}^{n} W_j^2$$

Gradient

$$\frac{\partial J}{\partial W_0} = -\frac{1}{m}\sum_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] \quad \text{for j=0}$$

$$\frac{\partial J}{\partial W_j} = -\frac{1}{m}\sum_{i=1}^{m}[(\widehat{y^i} - y^i)x_j^i] + \frac{2\lambda}{m}W_j \quad \text{for j} \geq 1$$

Neural network with L2 regularization

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)$$

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

Summary

- Initialize weights in neural network
- Weight matrix in neural network
- L2 regularization in neural network