CMPE 258, Deep Learning

Deep Neural Network

March 1, 2018

DMH 149A

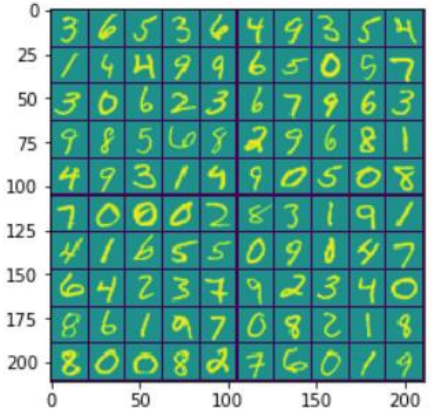
Taehee Jeong

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

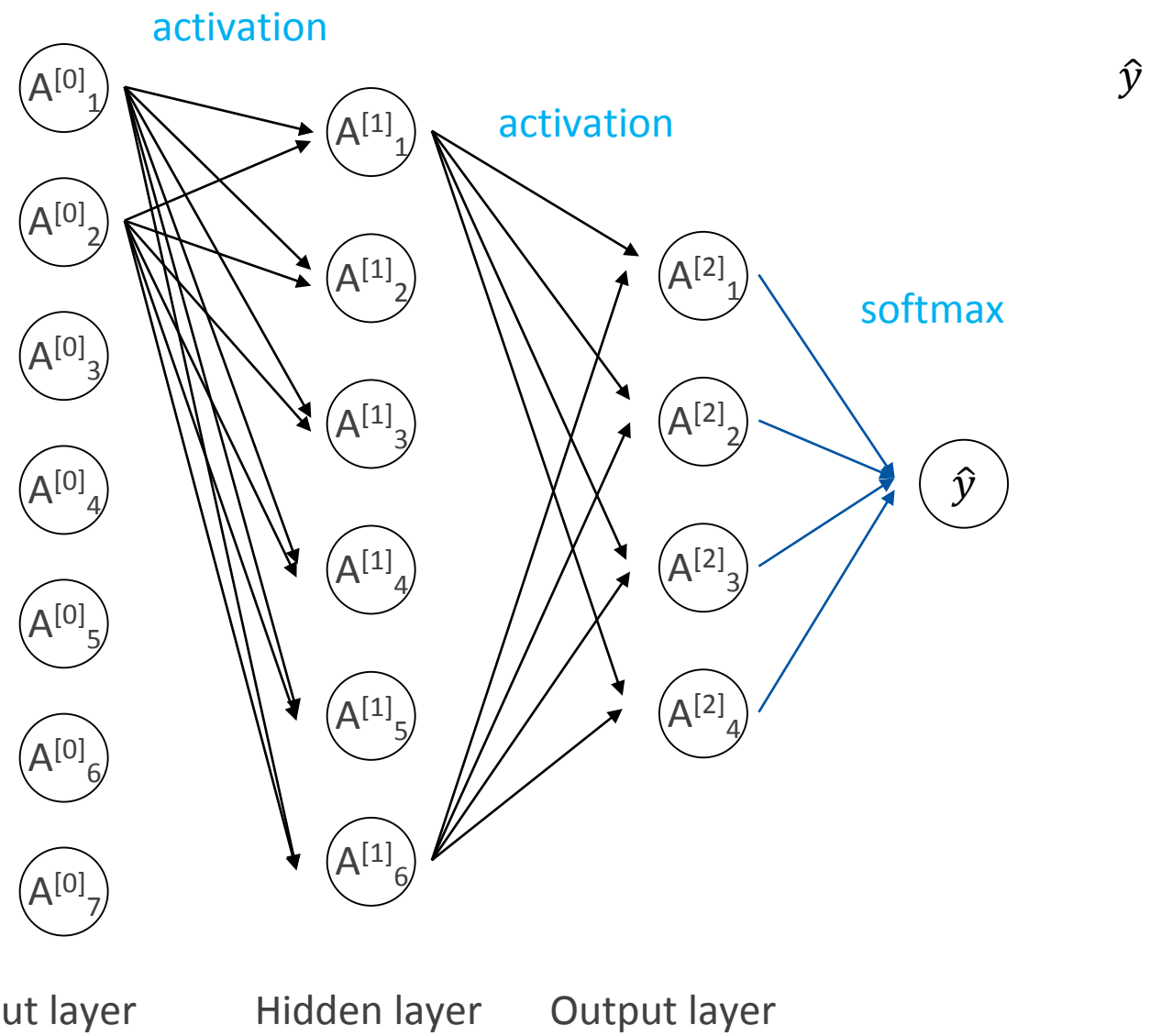
Please build neural network model using input layer (400 neurons), 1 hidden layer (25 neurons), and output layer (10 neurons) using training data set.

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)



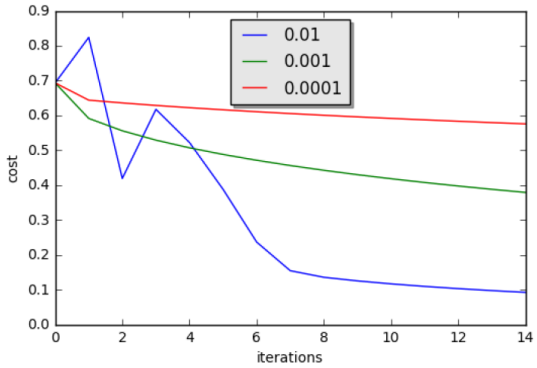
Assignment_3

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

- Initialize parameters
- Run the optimization loop
  - Forward propagation
  - Calculate cost function
  - Backward propagation
  - Gradient descent
- Return the learned parameters

<deep learning, Andrew Ng>

Exam_1 (mid term)

March 6th? March 8th?

You decide your score!

It will be based on accuracy on test data

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

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>

Parameter matrix dimension

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

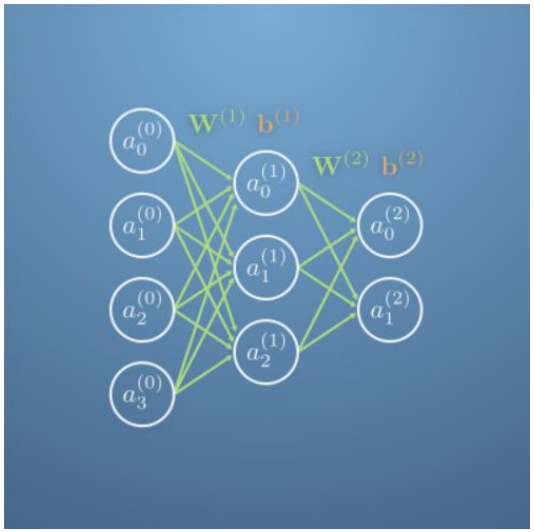
$$a^{[l]} = g^{[l]}(Z^{[l]})$$

$W^{[l]}$: $[n^{[l]}, n^{[l-1]}]$

$b^{[l]}$: $[n^{[l]}, 1]$ or $[1, n^{[l]}]$

Weight Matrix

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



<Mathematics for Machine Learning>

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

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

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

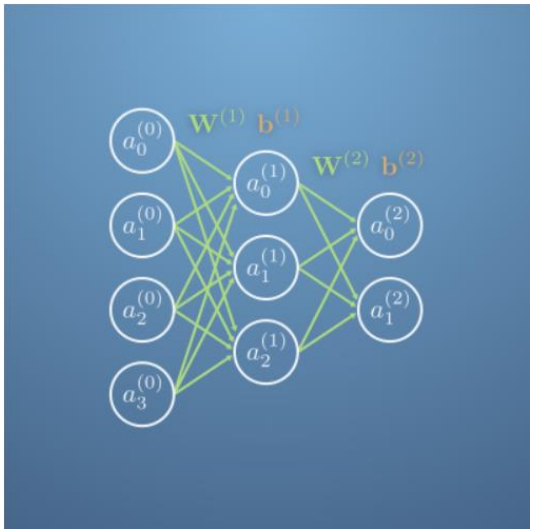
$W^{[1]}$: [3,4]

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]

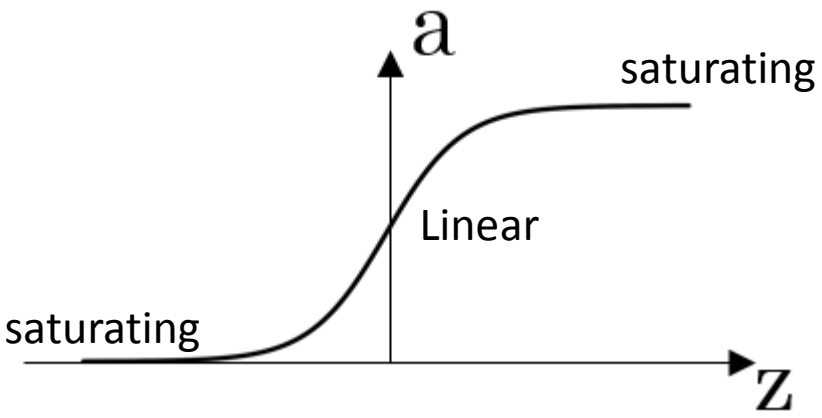
W_2[n_y,n_h]

b_2[1, n_y]

Activation functions in Neural Network

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- ELU

Sigmoid

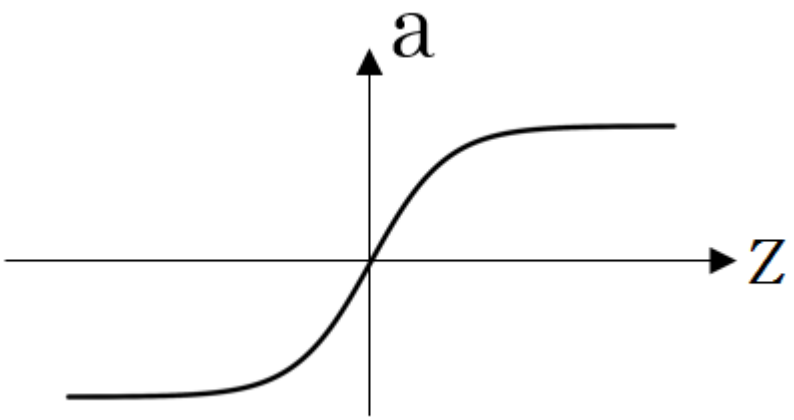


<Deep learning, Andrew Ng>

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

- It can be saturated(0 or 1) when z is large
- Gradients will be almost zero.
- The output is not zero-centered.

tanh



<Deep learning, Andrew Ng>

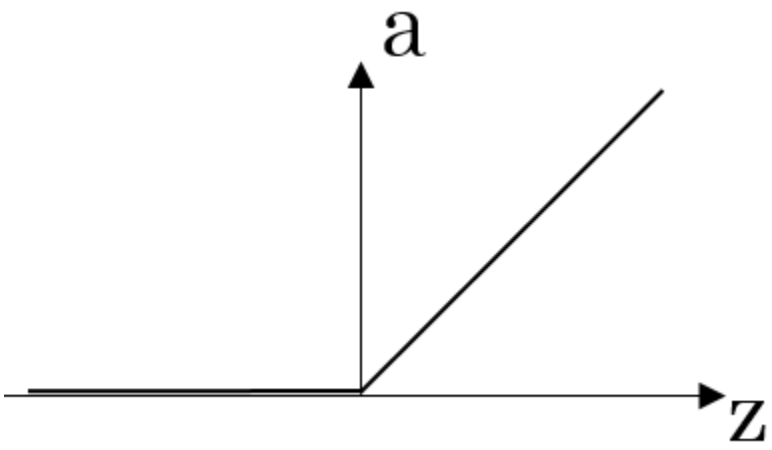
$$a = tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

- The output is zero-centered.
- It is simply a scaled sigmoid function.

$$tanh(z) = 2\sigma(2z) - 1$$

ReLU

Rectified Linear Unit

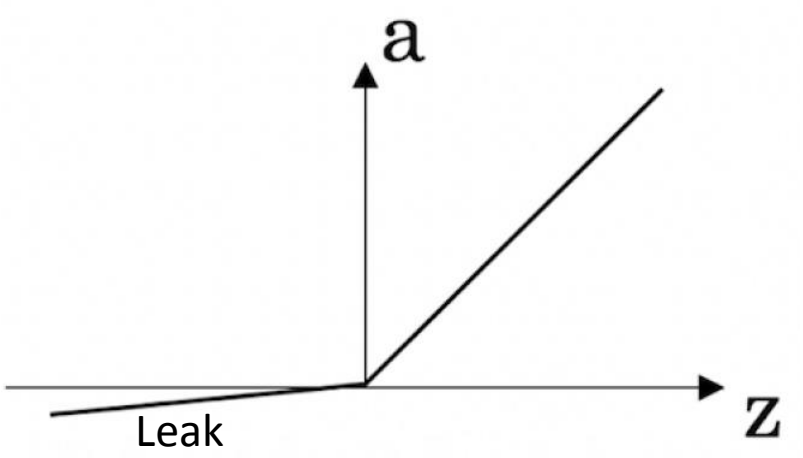


<Deep learning, Andrew Ng>

$$a = ReLU(z) = \max(0, z)$$

- It is zero when z is <=0
- It is linear with slope 1 when z is >0.
- It can learn fast
- It is most commonly used in CNN.
- Some network can be dead since neuron will not activate across entire dataset.

Leaky ReLU



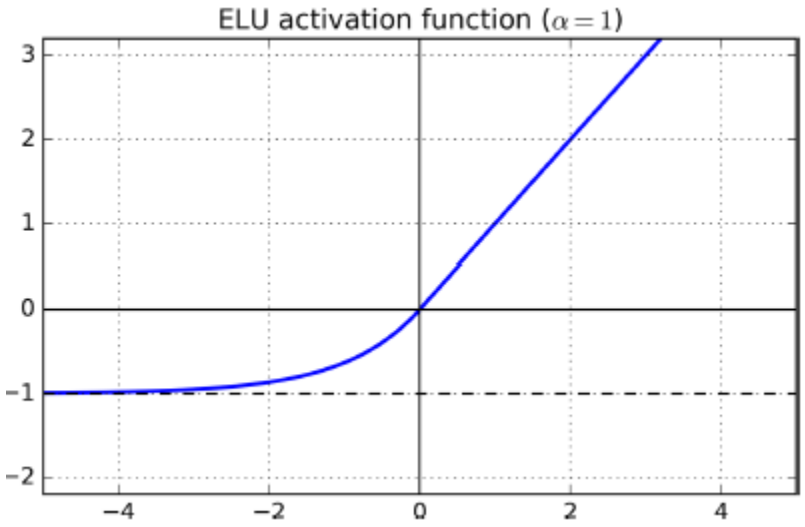
<Deep learning, Andrew Ng>

$$a = Leaky\ ReLU(z) = \max(0.01z, z)$$

- It is linear with small negative slope when z is <=0
- It is linear with slope 1 when z is >0.

ELU

Exponential Linear Unit



<Hands-On ML, Aurelien Geron>

$$a = ELU(z) = \begin{cases} \alpha(\exp(z) - 1) & if\ z < 0 \\ z & if\ z \geq 0 \end{cases}$$

- Negative values when z<0 → average output closer to 0
- Nonzero gradient for z<0 → avoid dying units issue
- It is smooth everywhere including z=0

Derivatives of activation functions

Sigmoid function

$$g(z) = a = \frac{1}{1 + \exp(-z)}$$

$$\frac{d}{dz} g(z) = \frac{1}{1 + \exp(-z)} \left(1 - \frac{1}{1 + \exp(-z)}\right)$$

$$g'(z) = g(z)(1 - g(z)$$

$$g'(z) = a(1 - a)$$

Derivatives of activation functions

Tanh function

$$g(z) = a = tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

$$\frac{d}{dz} g(z) = 1 - (\tanh(z))^2$$

$$g'(z) = 1 - g(z)^{\,2}$$

$$g'(z) = 1 - a^2$$

Derivatives of activation functions

ReLU and Leaky ReLU

ReLU

$$a = g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$$

Leaky ReLU

$$a = g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$$

Derivatives of activation functions

ELU

$$a = ELU(z) = \begin{cases} \alpha(\exp(z) - 1) & if\ z < 0 \\ z & if\ z \geq 0 \end{cases}$$

$$g'(z) = \begin{cases} \alpha \exp(z) & if\ z < 0 \\ 1 & if\ z \geq 0 \end{cases}$$

Regularization in Neural Network

- L2 regularization
- Elastic net regularization
- Early stopping
- Max norm constraints
- Dropout

<cs231n, Stanford university>

L2 regularization

Cost function

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(a^{[L](i)}) + (1-y^i)\log(1-a^{[L](i)})] + \frac{\lambda}{2m}\sum_{l=1}^{L}\|W^{[l]}\|^2$$

Gradient

$$dW^{[l]} = (from\ backpropagation) + \frac{\lambda}{m}W^{[l]}$$

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

$$W^{[l]} = W^{[l]} - \alpha[(from\ backprop) + \frac{\lambda}{m}W^{[l]}]$$

$$W^{[l]} = W^{[l]} - \frac{\alpha\lambda}{m}W^{[l]} - \alpha(from\ backprop)$$

Per-layer regularization:
It is not common to regularize different layers to different amount.

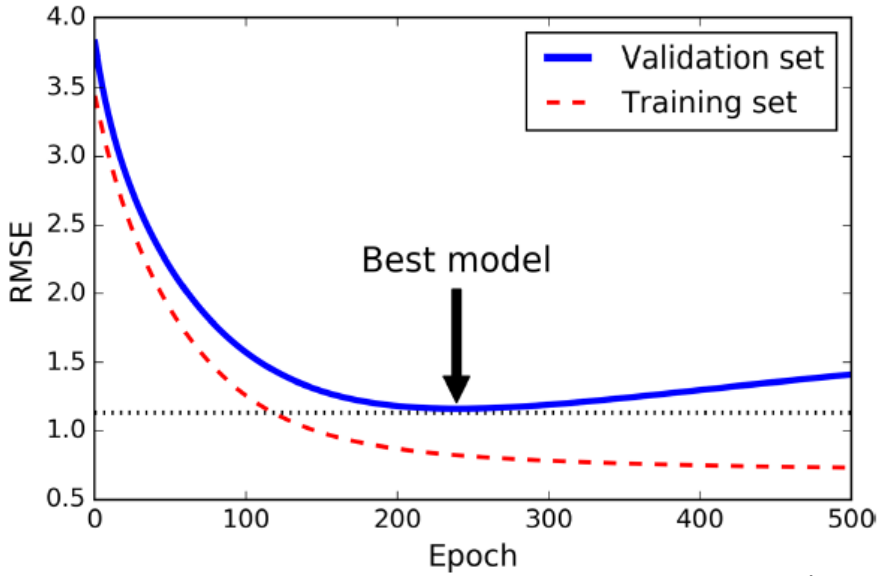
Elastic net regularization

Combine L1 regularization with L2 regularization

$$\lambda_1|W| + \lambda_2 W^2$$

Early Stopping

Stop training as the validation error reaches a minimum.



<Hands-On ML, Aurelien Geron>

Max norm constraints

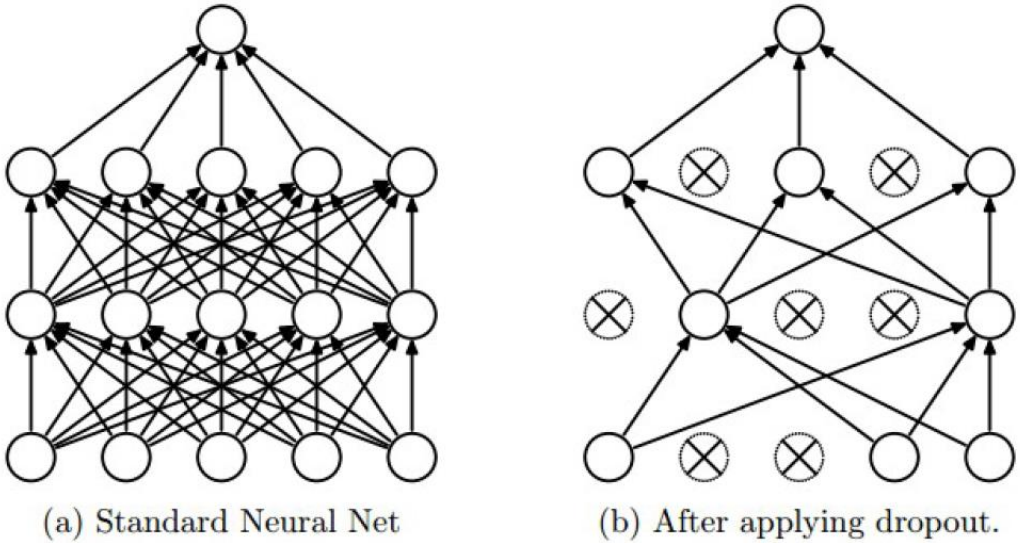
Enforce an absolute upper bound on the magnitude of the weight vector for every neuron.

$$\|W\|_2 < c$$

Drop out

$$px + (1 - p)0$$

Keep a neuron active with some probability p and set to zero others

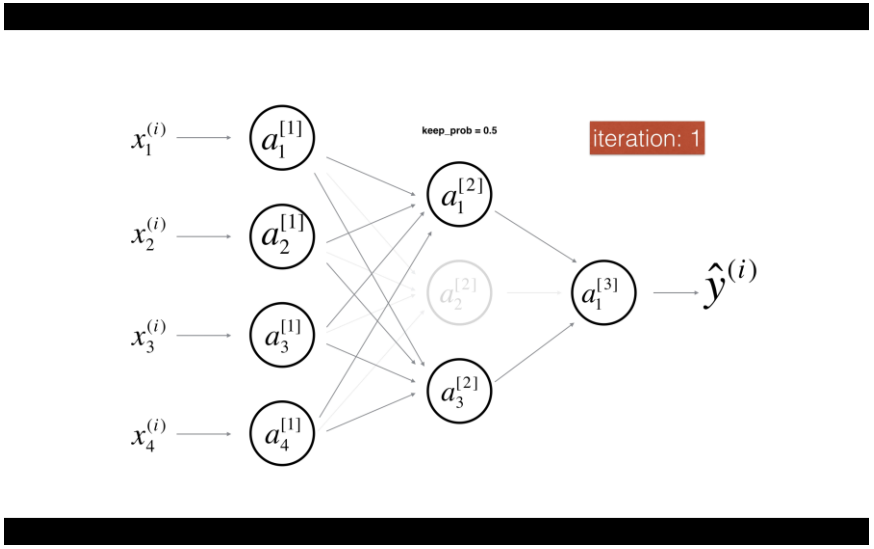


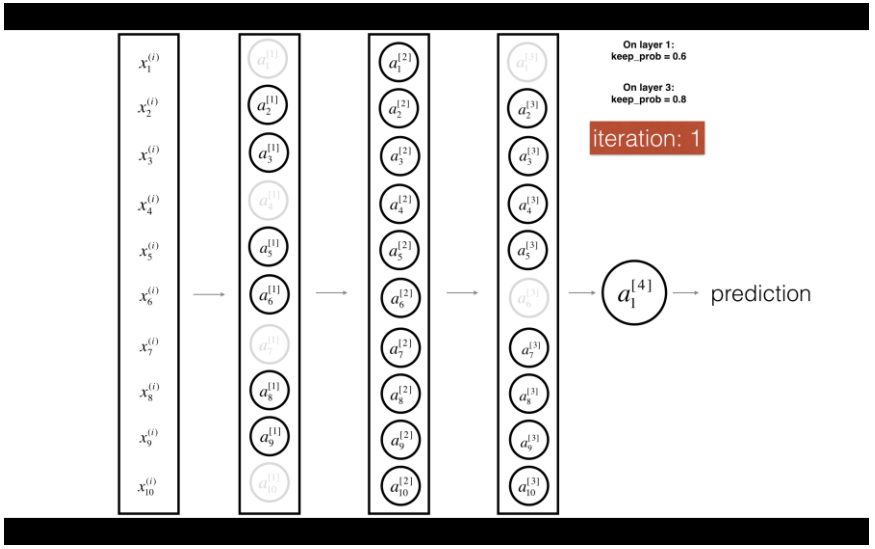
(a) Standard Neural Net

(b) After applying dropout.

<cs231n, Stanford university>

Drop out





<Deep learning, Andrew Ng>

Forward propagation with Drop out

1. Initialize matrix D1
2. Convert entries of D1 to 0 or 1(using keep prob as threshold)
3. Shut down some neurons of A1
4. Scale the value of neurons that haven't been shut down

```
keep_prob = 0.5
D1 = np.random.rand(A1.shape[0],A1.shape[1])
D1 = np.where(D1<keep_prob,1,0)
A1 = np.multiply(D1,A1)
A1 = A1/keep_prob
```

Backward propagation with dropout

1. Apply mask D1 to shut down the same neurons as during the forward propagation
2. Scale the value of neurons that haven't been shut down

```
dA1 = np.multiply(D1,dA1)
dA1 = dA1/keep_prob
dZ1 = np.multiply(dA1, np.int64(A1 > 0))
```

Summary

- Activation functions in Neural Network
- Derivatives of activation functions
- Regularization in Neural Network