CMPE 258, Deep Learning

Weights initialization

March 6, 2018

DMH 149A

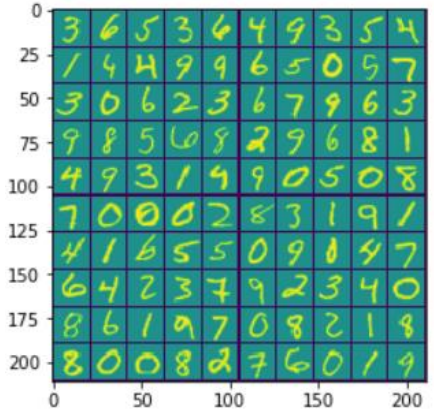
Taehee Jeong

Ph.D., Data Scientist

SJSU SAN JOSÉ STATE UNIVERSITY

Assignment_3

The due day was 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

Exam_1 (mid term)

Start March 8th 9am, End March 10th,11:59pm

You decide your score!

It will be based on accuracy on test data

We have Q&A session for Exam_1 during regular class hour on Thursday (March 8th)

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.

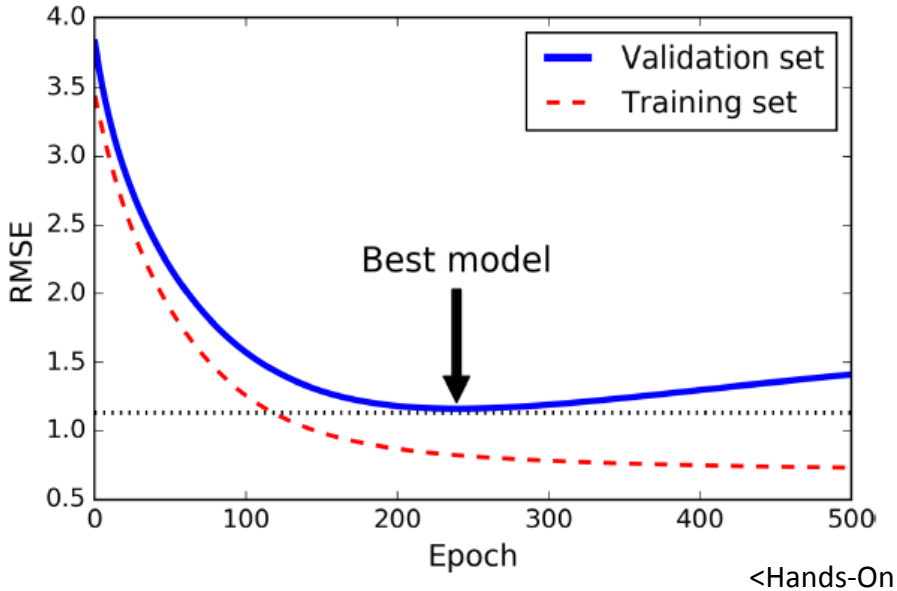
Elastic net regularization

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>

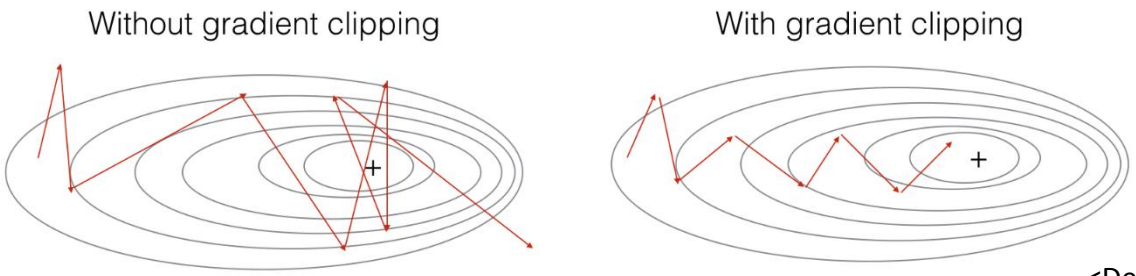
Max norm constraints

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

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

Gradient is clipping to prevent exploding gradients.

Every element of the gradient vector is clipped to lie between some range [-N,N].

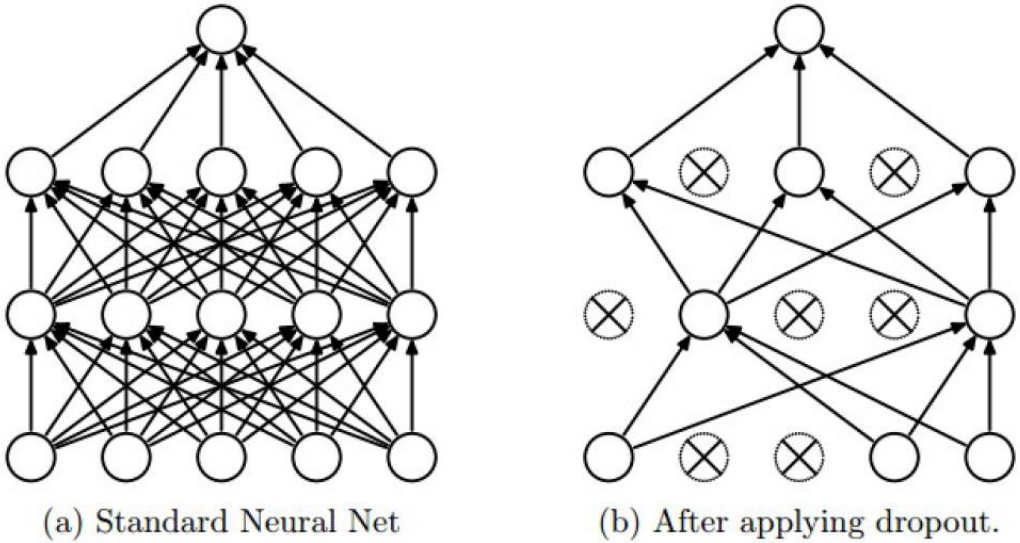


<Deep learning, Andrew Ng>

Drop out

$$p \cdot x + (1 - p) \cdot 0$$

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

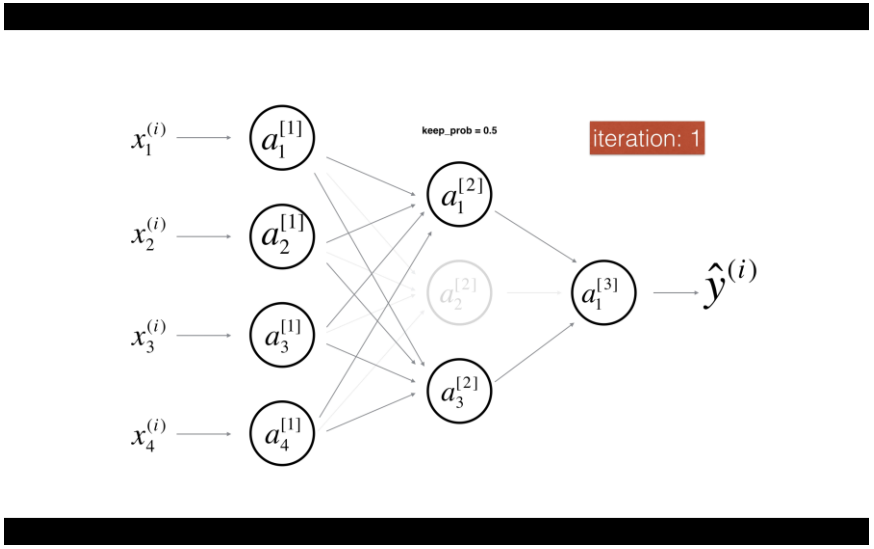


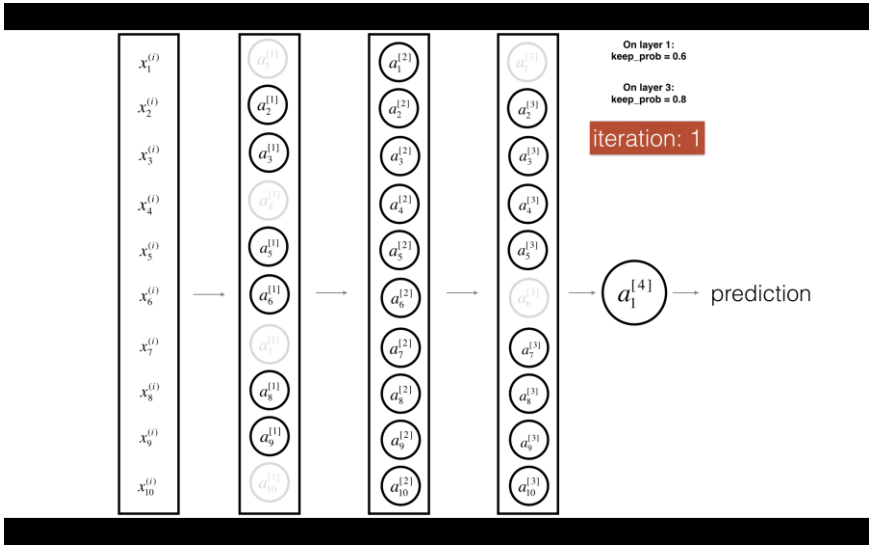
(a) Standard Neural Net

(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

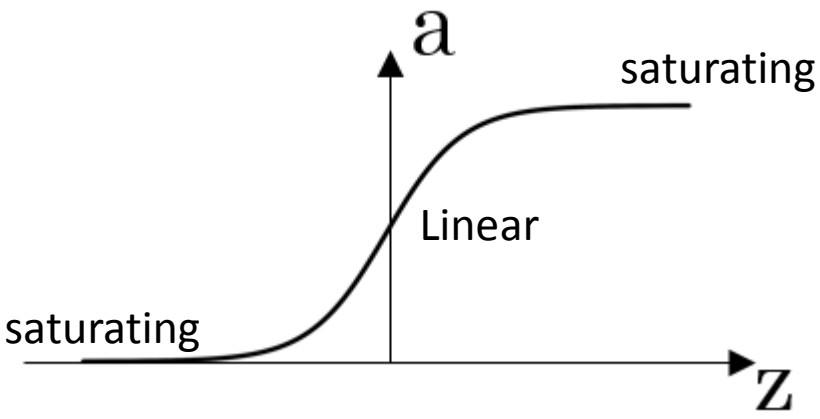
1. Apply mask D1 to shut down the same neurons as during the forward propagation
2. Scale the value of neurons that haven't been shut down

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

Activation functions in Neural Network

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU
- ELU

Sigmoid

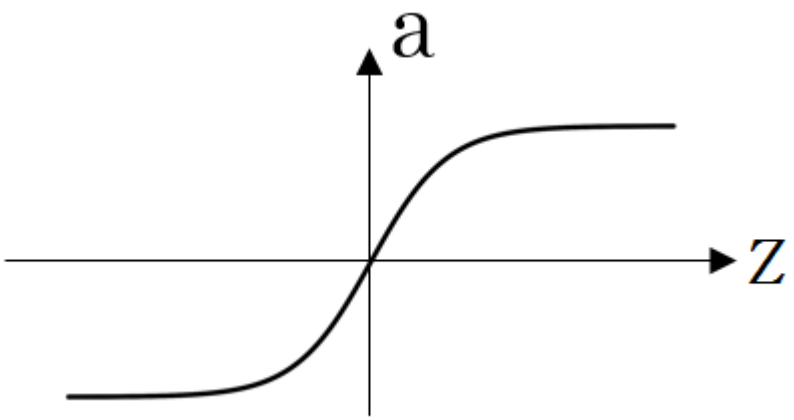


<Deep learning, Andrew Ng>

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

- Sigmoid neurons can be saturated when z is very large or very small.
- The saturation leads vanishing gradients.
- The output is not zero-centered (0~1).
- Exp is computationally expensive.

tanh



<Deep learning, Andrew Ng>

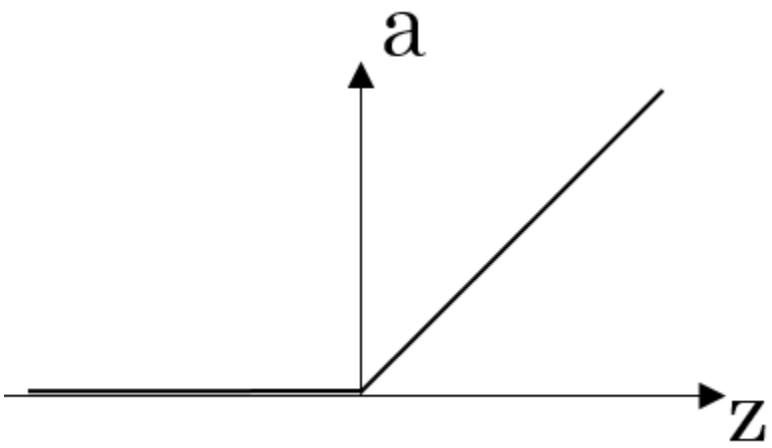
$$a = tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

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

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

ReLU

Rectified Linear Unit

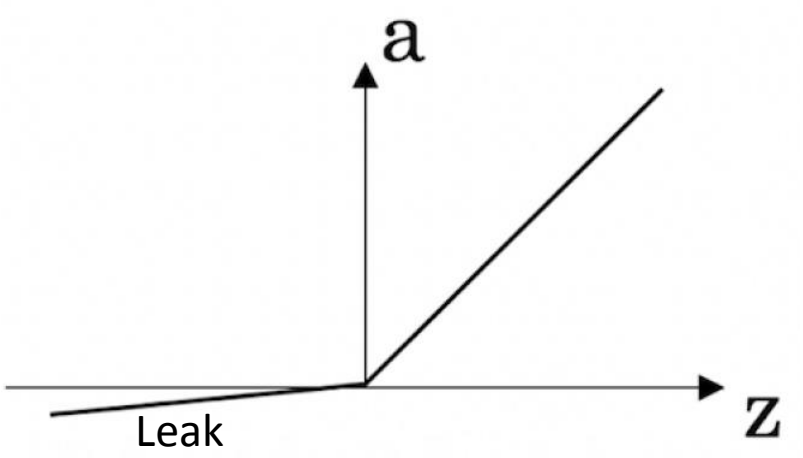


<Deep learning, Andrew Ng>

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

- It is zero when z is <=0
- It is linear with slope 1 when z is >0.
- Fast to compute
- Gradients do not vanish for z > 0.
- Faster convergence in practice.
- Not zero-centered.
- It is most commonly used in CNN.
- Some neurons can die if it is not activated, never updated.

Leaky ReLU



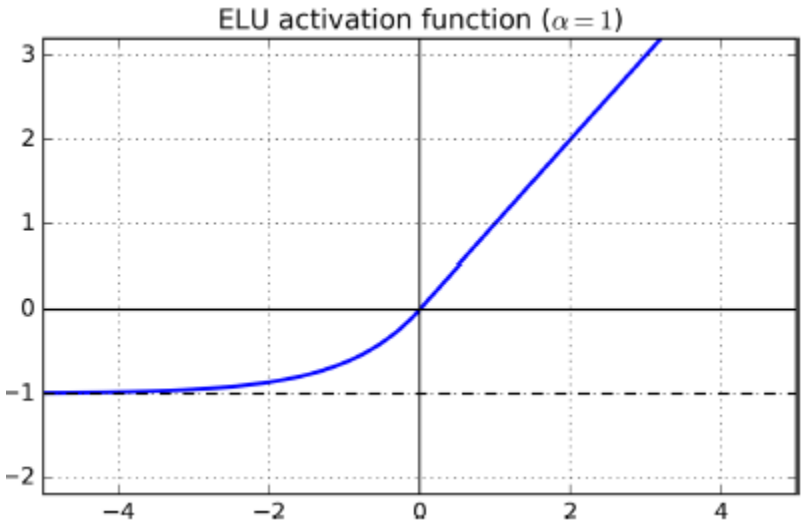
<Deep learning, Andrew Ng>

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

- It is linear with small negative slope when z is <=0
- It is linear with slope 1 when z is >0.
- It will not die

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

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

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

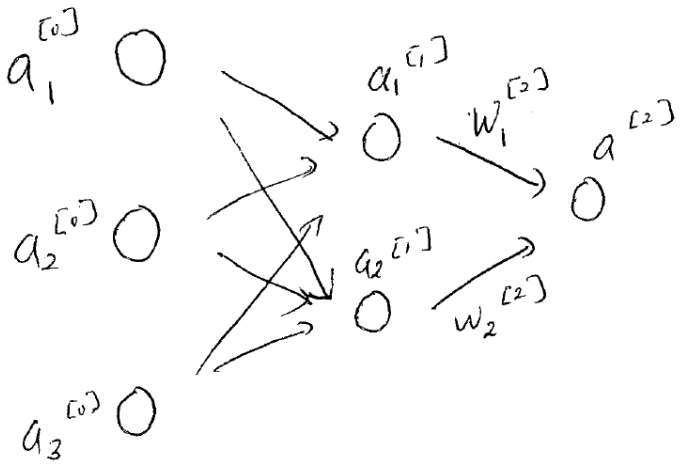
Derivatives of activation functions

ELU

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

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

Forward propagation in Neural Network



$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_1^{[1]} = w_{11}^{[1]} \cdot a_1^{[0]} + w_{12}^{[1]} \cdot a_2^{[0]} + w_{13}^{[1]} \cdot a_3^{[0]} + b_1^{[1]}$$

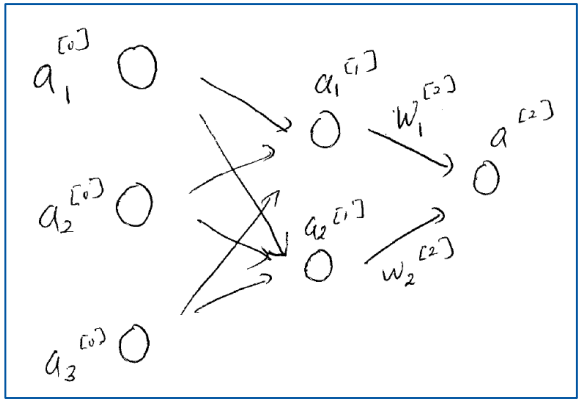
$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_2^{[1]} = w_{21}^{[1]} \cdot a_1^{[0]} + w_{22}^{[1]} \cdot a_2^{[0]} + w_{23}^{[1]} \cdot a_3^{[0]} + b_2^{[1]}$$

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

$$z^{[2]} = w_1^{[2]} \cdot a_1^{[1]} + w_2^{[2]} \cdot a_2^{[1]} + b^{[2]}$$

Forward propagation



$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_1^{[1]} = w_{11}^{[1]} \cdot a_1^{[0]} + w_{12}^{[1]} \cdot a_2^{[0]} + w_{13}^{[1]} \cdot a_3^{[0]} + b_1^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

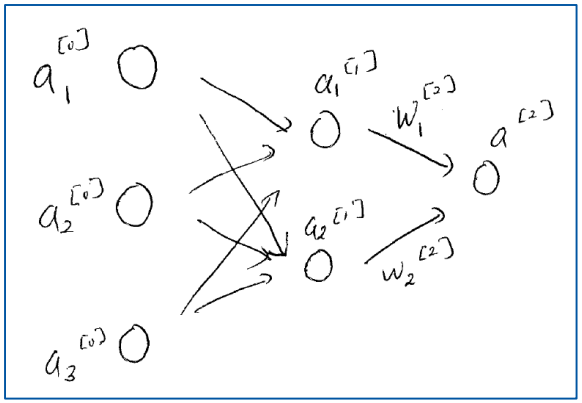
$$z_2^{[1]} = w_{21}^{[1]} \cdot a_1^{[0]} + w_{22}^{[1]} \cdot a_2^{[0]} + w_{23}^{[1]} \cdot a_3^{[0]} + b_2^{[1]}$$

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

$$z^{[2]} = w_1^{[2]} \cdot a_1^{[1]} + w_2^{[2]} \cdot a_2^{[1]} + b^{[2]}$$

If $W^{[1]}$,$b^{[1]}$, $W^{[2]}$,$b^{[2]}$ are started with zero,
Then, $Z^{[1]}$,$Z^{[2]}$ become zero.
$a^{[1]}$,$a^{[2]}$ become 0.5.

Backward



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

$$z^{[2]} = w_1^{[2]} \cdot a_1^{[1]} + w_2^{[2]} \cdot a_2^{[1]} + b^{[2]}$$

$$\frac{\partial L}{\partial w_1^{[2]}} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w_1^{[2]}}$$

$$= dz^{[2]} \cdot \frac{\partial z^{[2]}}{\partial w_1^{[2]}}$$

$$= dz^{[2]} \cdot a_1^{[1]}$$

$$\frac{\partial L}{\partial w_2^{[2]}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial w_2^{[2]}}$$

$$= dz^{[2]} \cdot \frac{\partial z^{[2]}}{\partial w_2^{[2]}}$$

$$= dz^{[2]} \cdot a_2^{[1]}$$

$dW^{[2]}$ is same

$a^{[1]}$,$a^{[2]}$ will always get same updated!

Weights initialization

Need to break symmetry.

- If $W^{[1]}$,$b^{[1]}$, $W^{[2]}$,$b^{[2]}$ are started with zero,
- Then, $Z^{[1]}$,$Z^{[2]}$ become zero.
- $a^{[1]}$,$a^{[2]}$ become 0.5.
- $dW^{[2]}$ is same
- $a^{[1]}$,$a^{[2]}$ will always get same updated!

Small random numbers

- W = 0.01* np.random.randn(D,H)
- samples from a zero mean, unit standard deviation Gaussian

Xavier initialization

When using sigmoid activation function

Multiplies weights by

$$\frac{\sqrt{2}}{\sqrt{n_{inputs} + n_{outputs}}}$$

<Hands-On ML, Aurelien Geron>

He initialization

When using ReLU activation function (and its variants, including the ELU activation)

Multiplies weights by

$$\sqrt{2}\frac{\sqrt{2}}{\sqrt{n_{inputs}+n_{outputs}}}$$

<Hands-On ML, Aurelien Geron>

Initialization for Hyperbolic tangent

When using tanh activation function

Multiplies weights by

$$4\frac{\sqrt{2}}{\sqrt{n_{inputs}+n_{outputs}}}$$

Weights initialization

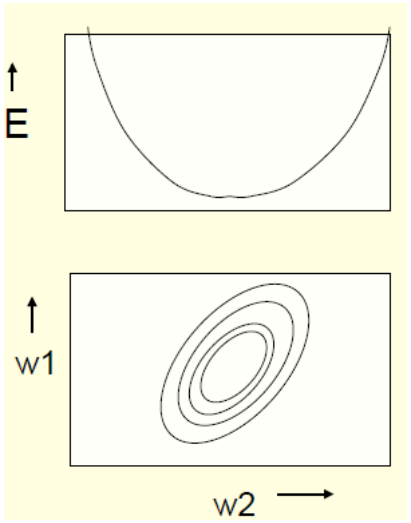
Activation function	Uniform distribution [–r, r]	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

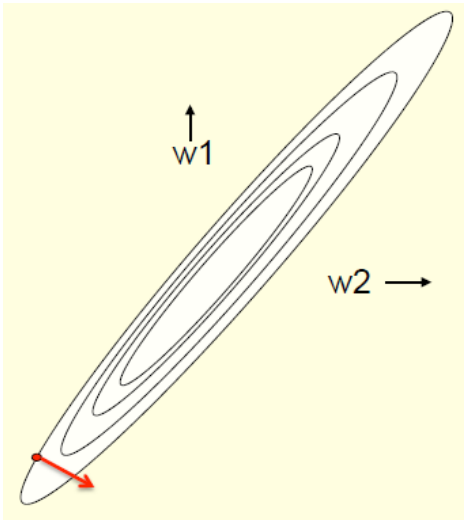
<Hands-On ML, Aurelien Geron>

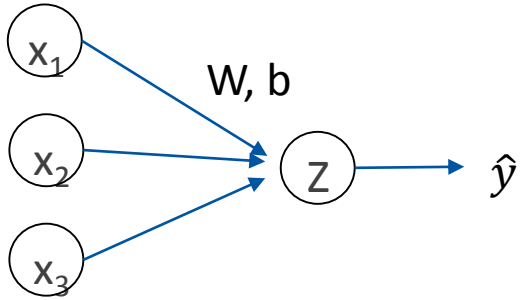
Normalizing inputs to speed up learning

In linear and logistic regression, we normalize input X to speed up learning.

$X = (X - \mu)/\sigma$





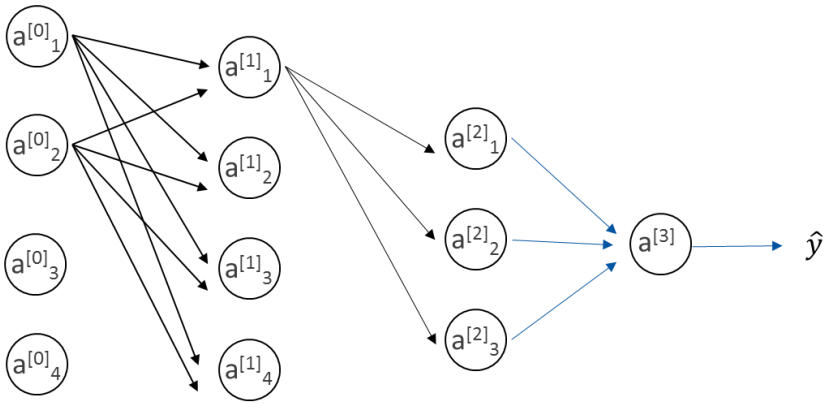


<Neural Networks, Geoffrey Hinton>

Normalizing activations in a network

In Neural Network, we may normalize $a^{[l-1]}$ to train $W^{[l]}$,$b^{[l]}$ faster.

In actual, we normalize $Z^{[l-1]}$ instead of $a^{[l-1]}$.



Batch Normalization

Address the vanishing / exploding gradients problems

Address the problem that the distribution of each layer's inputs changes during training.

Adding an operation in the model just before the activation function of each layer, zero-centering and normalizing the inputs, then scaling and shifting the result using two new parameters per layer

(one for scaling, the other for shifting)

Implementing Batch Norm

Given some intermediate values in NN,

$$\mu = \frac{1}{m}\sum_i Z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (Z^{(i)} - \mu)^2$$

If $\gamma = \sqrt{\sigma^2 + \epsilon}$

$\beta = \mu$

then

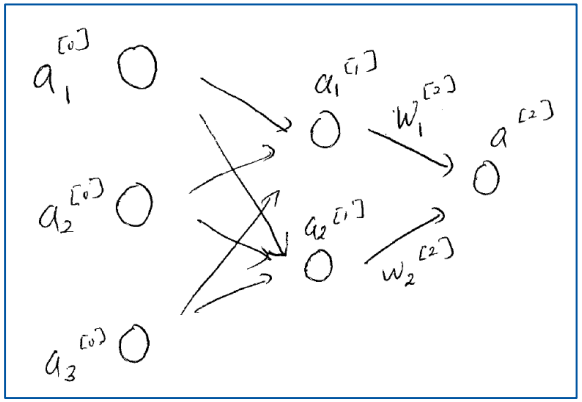
$\tilde{Z}^{(i)} = Z^{(i)}_{norm}$

$$Z^{(i)}_{norm} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{Z}^{(i)} = \gamma Z^{(i)}_{norm} + \beta$$

ε is a smoothing term to avoid division by zero, a tiny number
γ is scaling parameter for a layer
β is shifting parameter for a layer
γ and β *are learnable parameters for model*

Adding Batch Norm to a network



$a^{[0]} \xrightarrow{W^{[1]}, b^{[1]}} z^{[1]} \xrightarrow[\text{Batch Norm}]{\beta^{[1]}, \gamma^{[1]}} \tilde{z}^{[1]} \rightarrow a^{[1]} \xrightarrow{W^{[2]}, b^{[2]}} z^{[2]} \xrightarrow[\text{Batch Norm}]{\beta^{[2]}, \gamma^{[2]}} \tilde{z}^{[2]} \rightarrow a^{[2]}$

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

$\{ \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \ldots, \beta^{[L]}, \gamma^{[L]} \}$

$\beta^{[l]} := \beta^{[l]} - \alpha d\beta^{[l]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$

$z_1^{[1]} = w_{11}^{[1]} \cdot a_1^{[0]} + w_{12}^{[1]} \cdot a_2^{[0]} + w_{13}^{[1]} \cdot a_3^{[0]} + b_1^{[1]}$

$a_2^{[1]} = \sigma(z_2^{[1]})$

$z_2^{[1]} = w_{21}^{[1]} \cdot a_1^{[0]} + w_{22}^{[1]} \cdot a_2^{[0]} + w_{23}^{[1]} \cdot a_3^{[0]} + b_2^{[1]}$

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

$z^{[2]} = w_1^{[2]} \cdot a_1^{[1]} + w_2^{[2]} \cdot a_2^{[1]} + b^{[2]}$

Pros & Cons for Batch Normalization

Vanishing gradients problems are strongly reduced

Less sensitive to the weight initialization

Be able to use much larger learning rates

Play as a regularization

Add complexity to the model

Require Extra computation

Summary

- Regularization in Neural Network
- Activation functions in Neural Network
- Derivatives of activation functions
- Weight initialization
- Batch Normalization