Improving Learning in Neural Networks

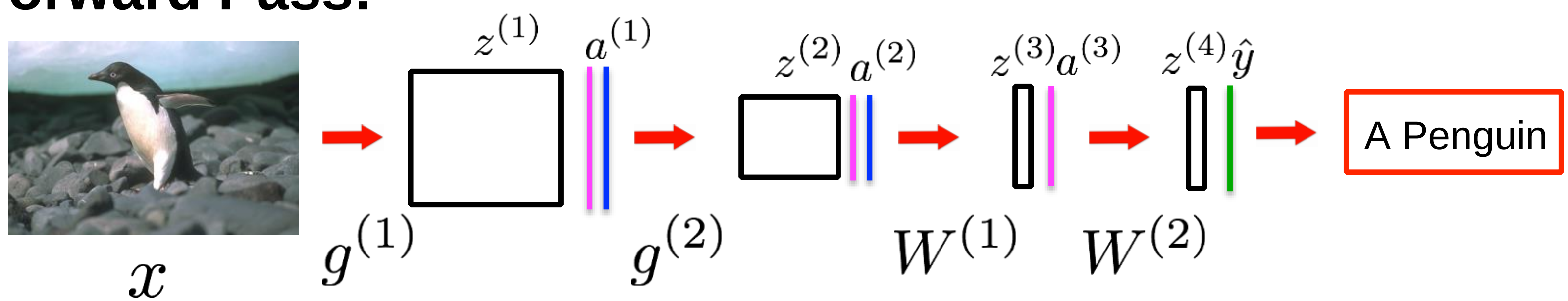
CIS 680

Convolutional Networks

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



Convolutional Networks

Notation:

- convolutional layer output

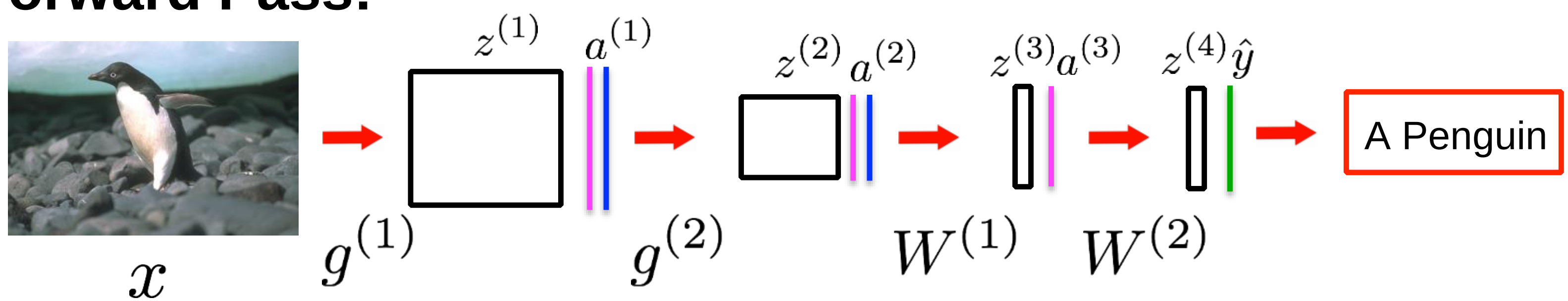
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:



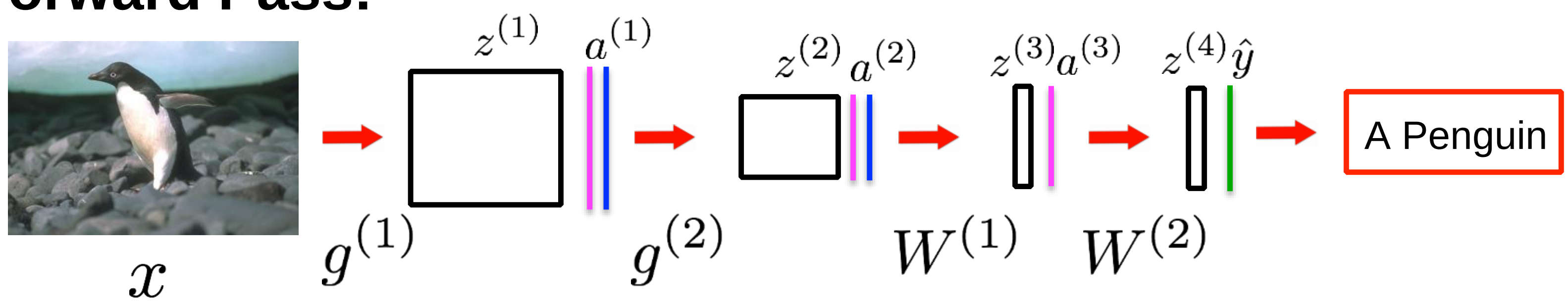
1. $a^{(1)} = pool(f(g^{(1)} * x))$

Convolutional Networks

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



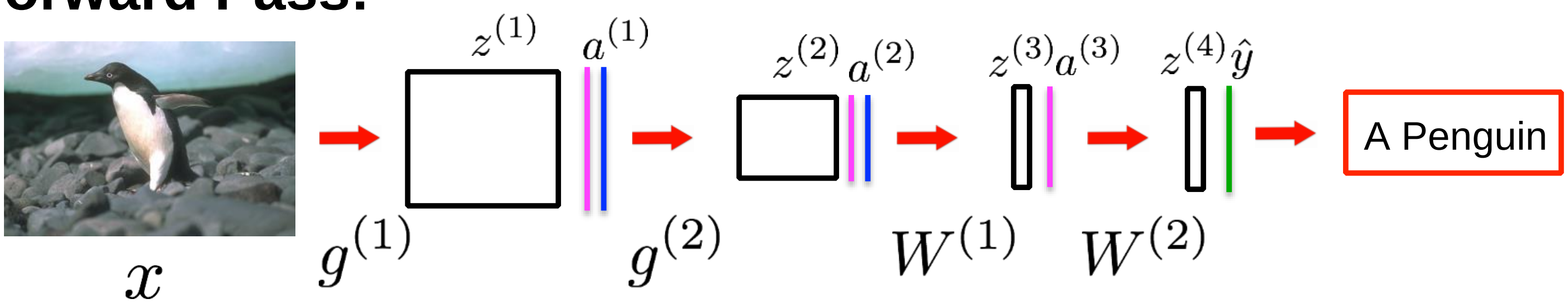
1. $a^{(1)} = pool(f(g^{(1)} * x))$
2. $a^{(2)} = pool(f(g^{(2)} * a^{(1)}))$

Convolutional Networks

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



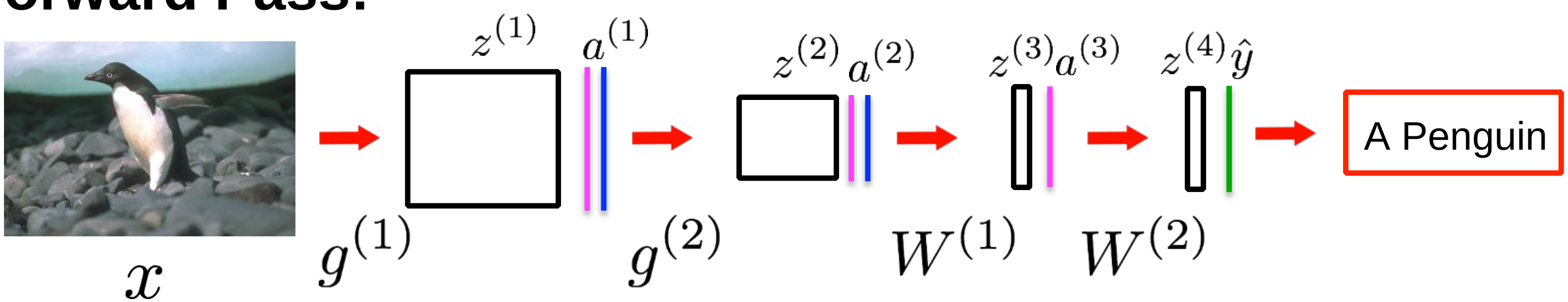
1. $a^{(1)} = pool(f(g^{(1)} * x))$
2. $a^{(2)} = pool(f(g^{(2)} * a^{(1)}))$
3. $a^{(3)} = f(W^{(1)} a^{(2)})$

Convolutional Networks

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



1. $a^{(1)} = pool(f(g^{(1)} * x))$
2. $a^{(2)} = pool(f(g^{(2)} * a^{(1)}))$
3. $a^{(3)} = f(W^{(1)} a^{(2)})$
4. $\hat{y} = softmax(W^{(2)} a^{(3)})$

Convolutional Networks

Notation:

- convolutional layer output

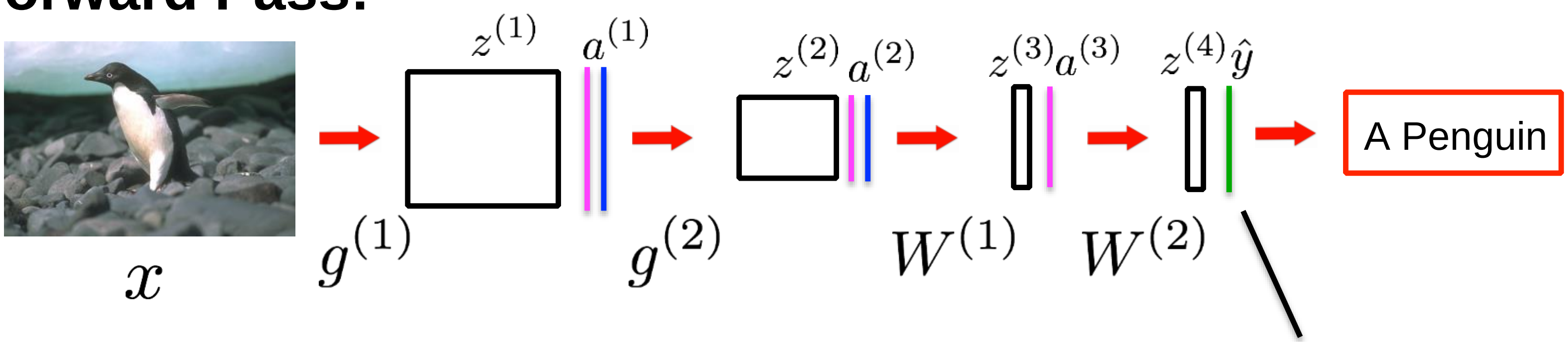
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:



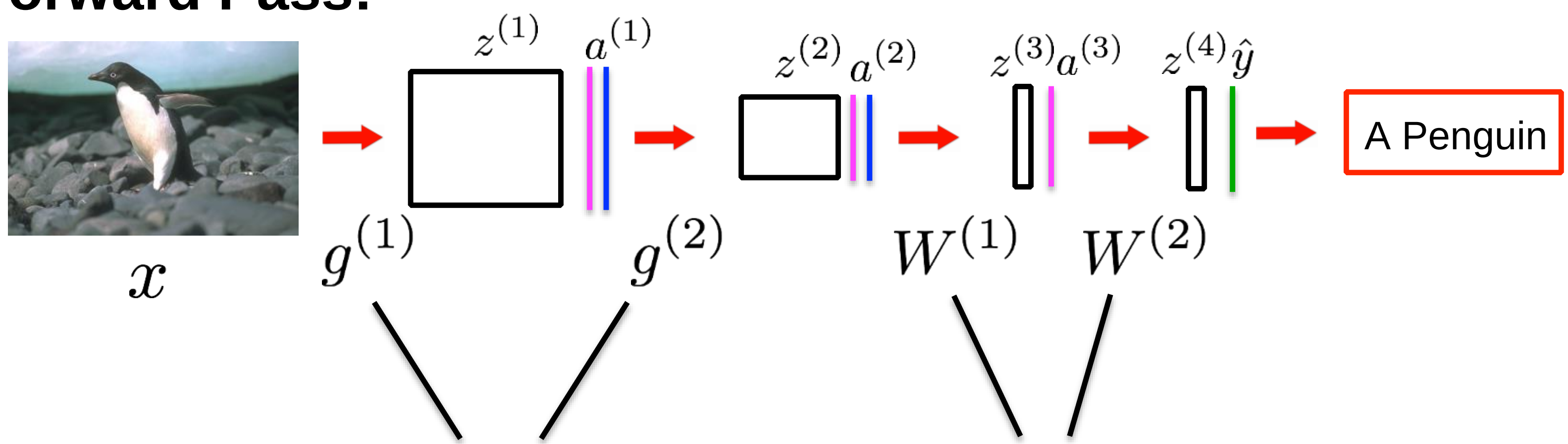
Predicted probabilities, which class this image belongs to

Convolutional Networks

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

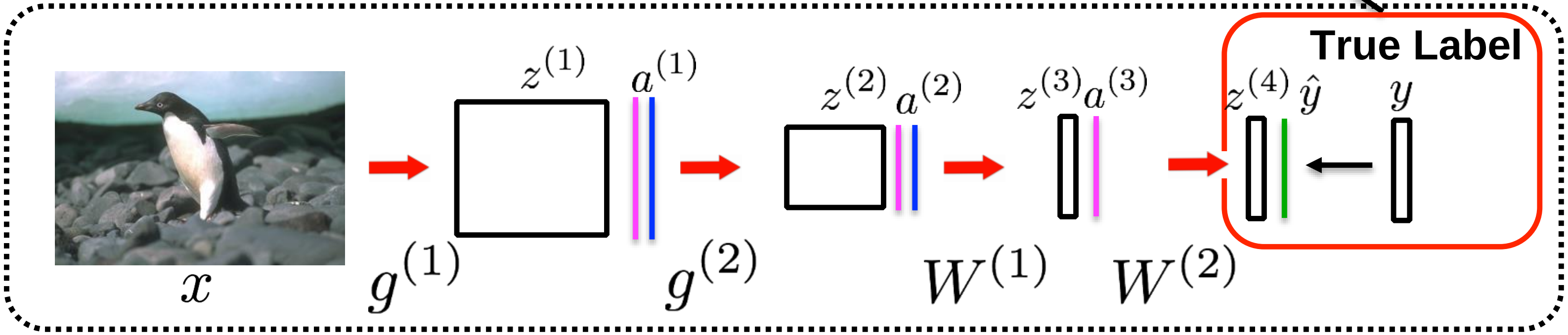
Forward Pass:



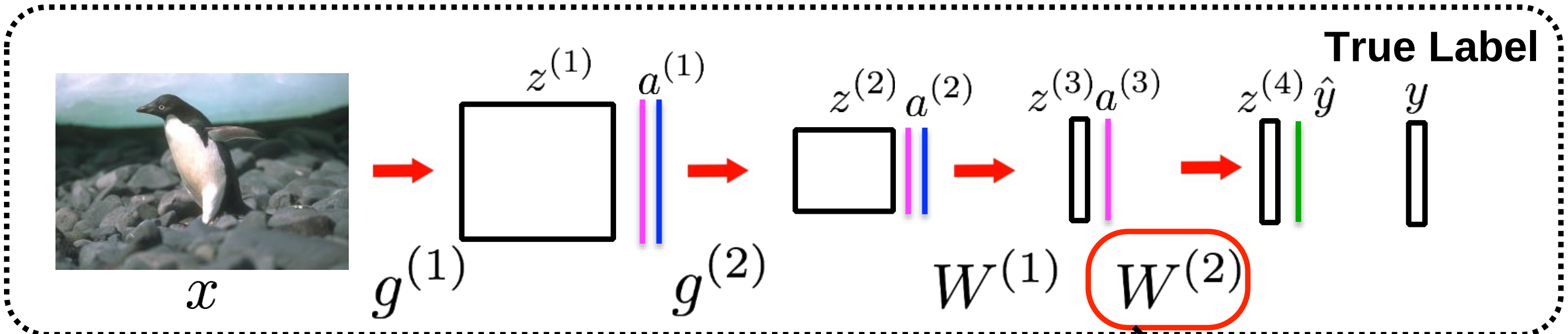
How to learn the parameters from the data?

Backpropagation

1. Compute the gradients of the overall loss and propagate it back: $\frac{\partial L}{\partial z^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(4)}}$



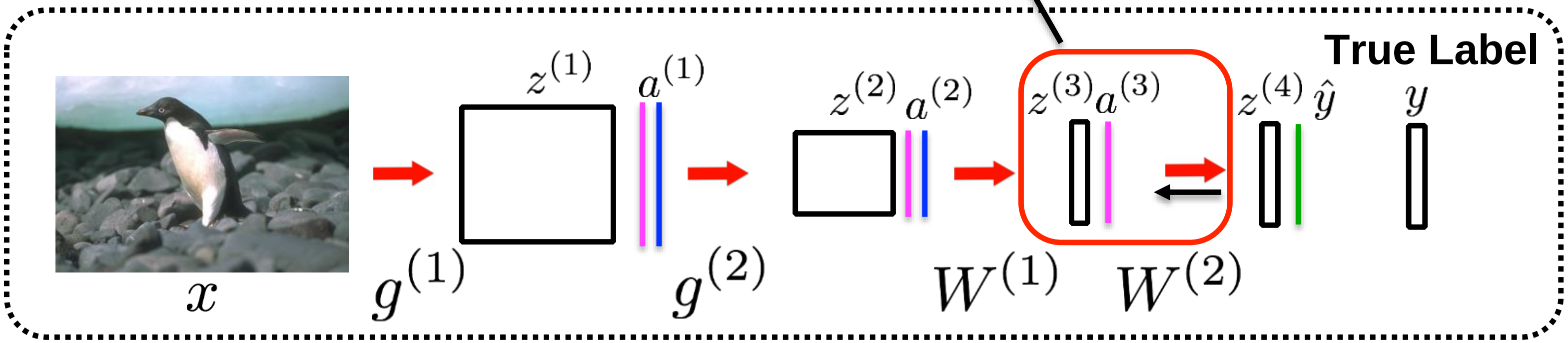
Backpropagation



2. Compute the gradients to adjust the weights: $\dfrac{\partial L}{\partial W^{(2)}} = \dfrac{\partial L}{\partial z^{(4)}} \dfrac{\partial z^{(4)}}{\partial W^{(2)}}$ where $z^{(4)} = W^{(2)} a^{(3)}$

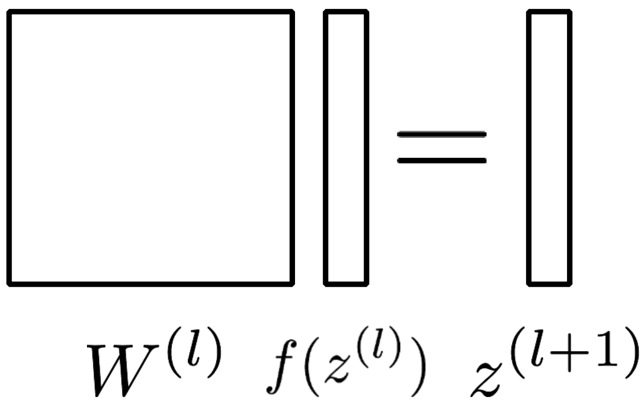
Backpropagation

3. Backpropagate the gradients to previous layers: $\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial f(z^{(3)})} \frac{\partial f(z^{(3)})}{\partial z^{(3)}}$ where $z^{(4)} = W^{(2)} a^{(3)}$



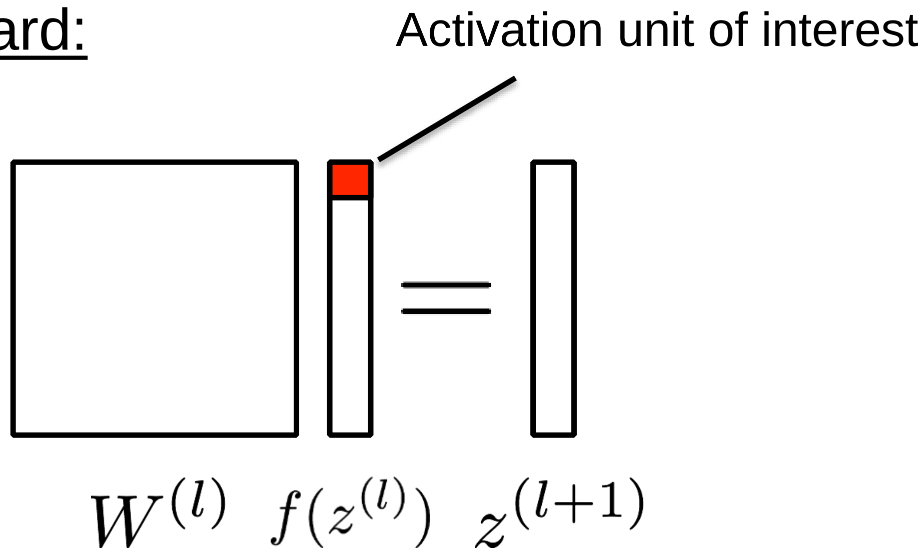
Fully Connected Layers:

Forward:



Fully Connected Layers:

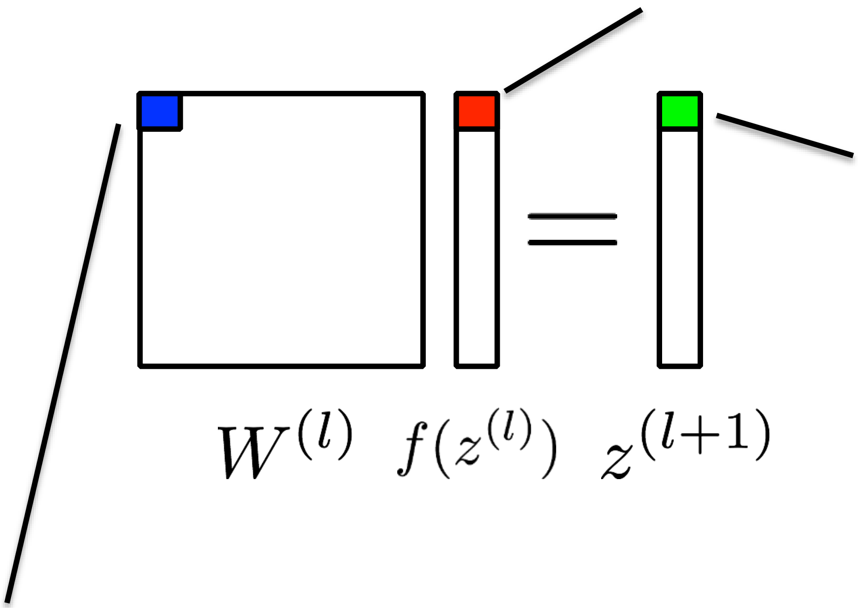
Forward:



<u>Fully Connected Layers:</u>

<u>Forward:</u>

Activation unit of interest

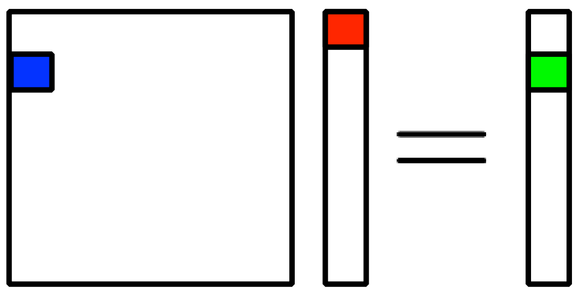


The output

The weight that is used in conjunction with the activation unit of interest

Fully Connected Layers:

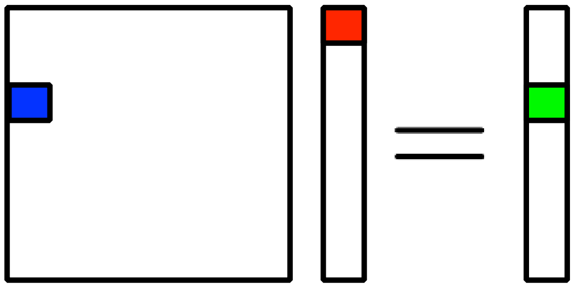
Forward:



$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$

<u>Fully Connected Layers:</u>

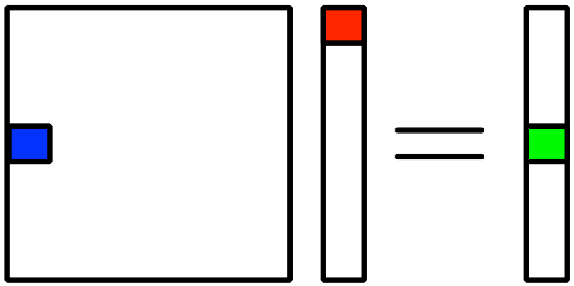
<u>Forward:</u>



$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

Fully Connected Layers:

Forward:

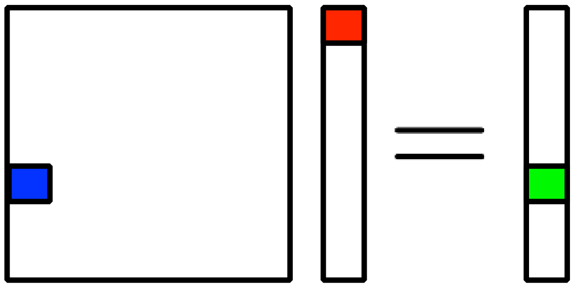


$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

Backpropagation

Fully Connected Layers:

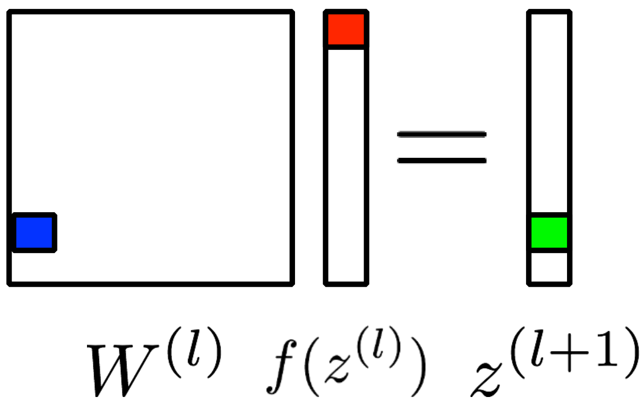
Forward:



$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

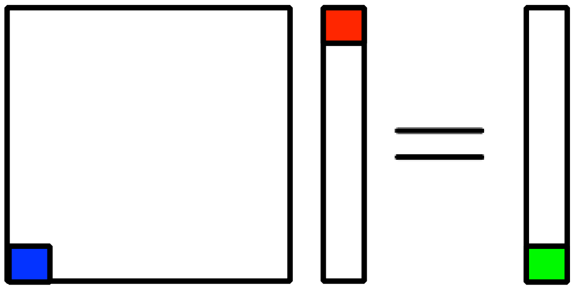
Fully Connected Layers:

Forward:



Fully Connected Layers:

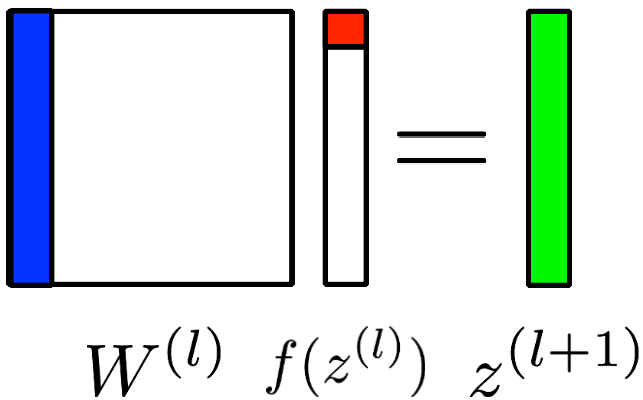
Forward:



$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

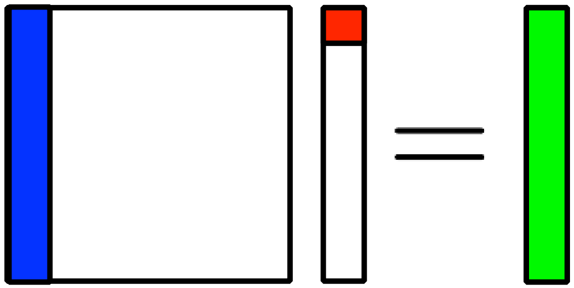
Fully Connected Layers:

Forward:



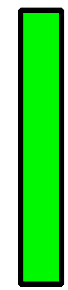
Fully Connected Layers:

Forward:



$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$

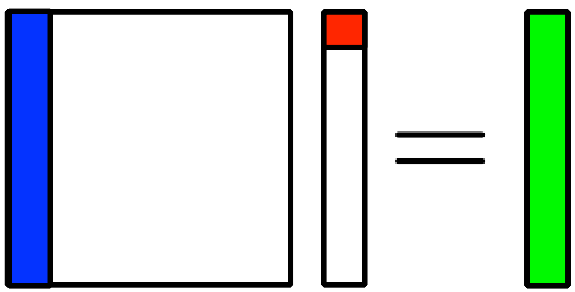
Backward:



$$\frac{\partial L}{\partial z^{(l+1)}}$$

Fully Connected Layers:

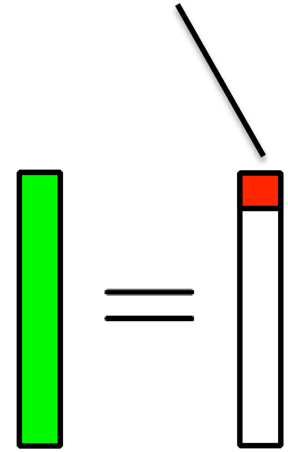
Forward:



$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$

A measure how much an activation unit contributed to the loss

Backward:

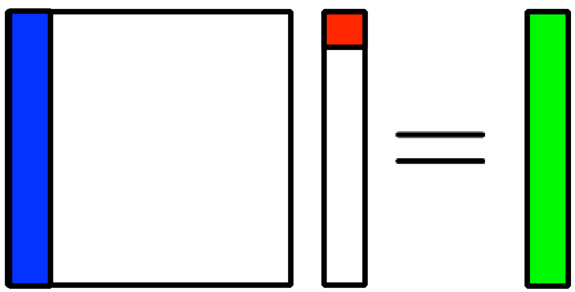


$\frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})}$

Fully Connected Layers:

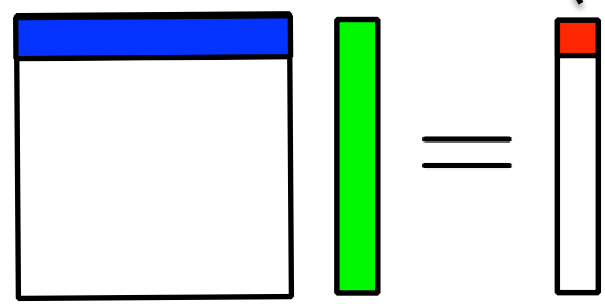
A measure how much an activation unit contributed to the loss

Forward:



$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

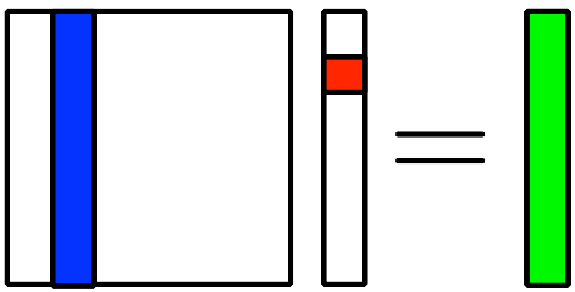
Backward:



$$(W^{(l)})^T \quad \frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})}$$

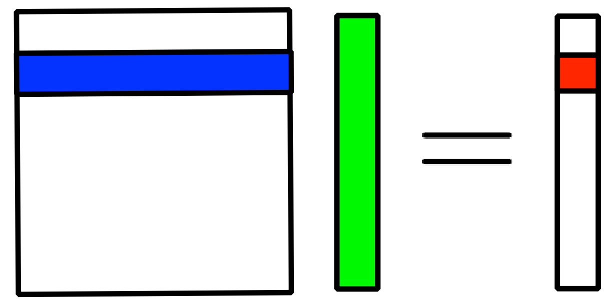
Fully Connected Layers:

Forward:



$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$

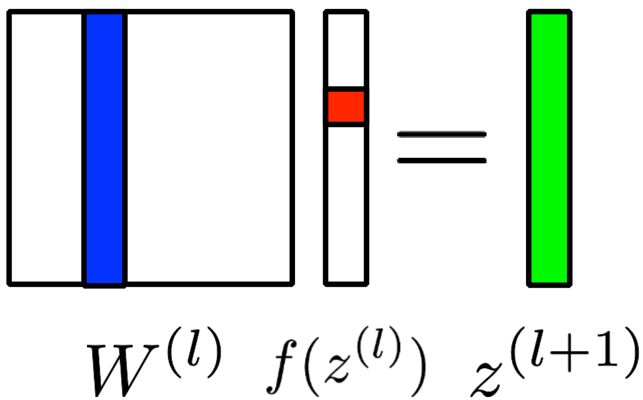
Backward:



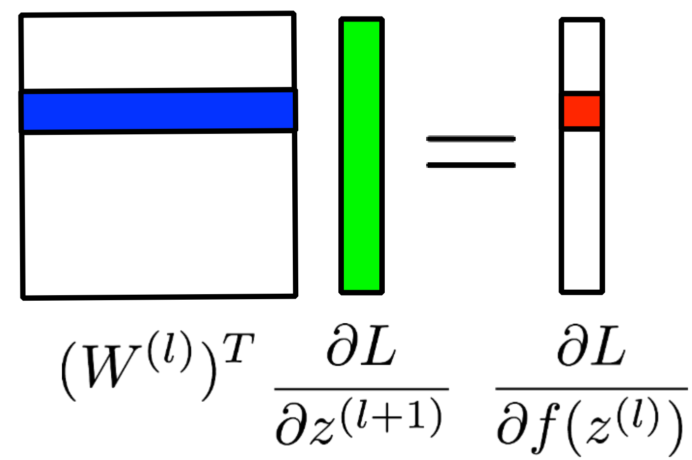
$(W^{(l)})^T \quad \frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})}$

Fully Connected Layers:

Forward:

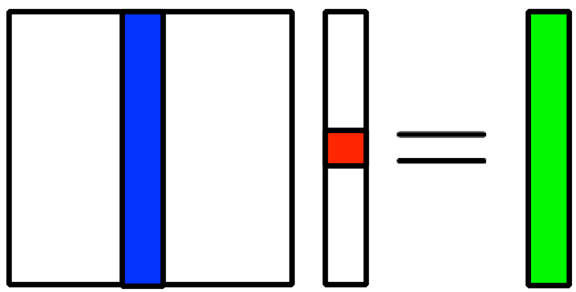


Backward:



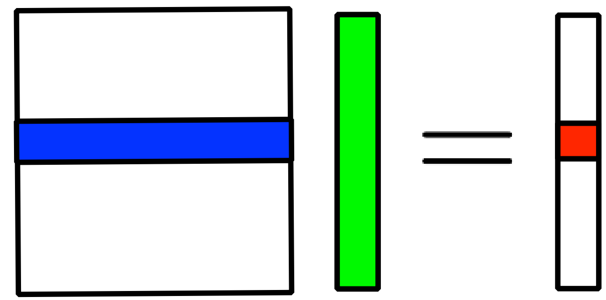
Fully Connected Layers:

Forward:



$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

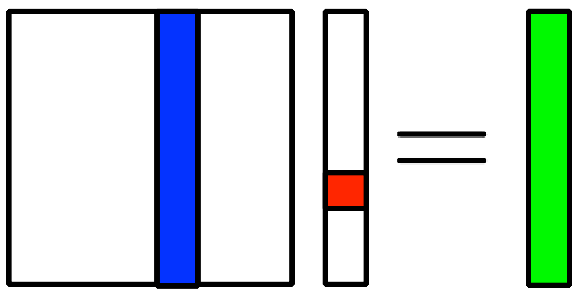
Backward:



$$(W^{(l)})^T \quad \frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})}$$

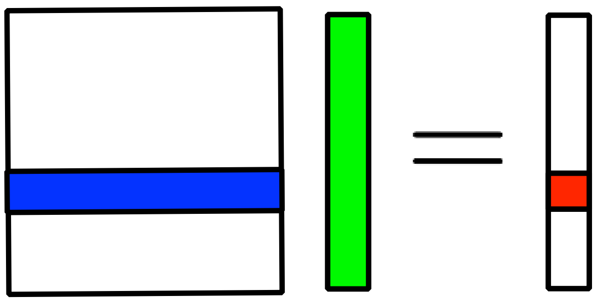
Fully Connected Layers:

Forward:



$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

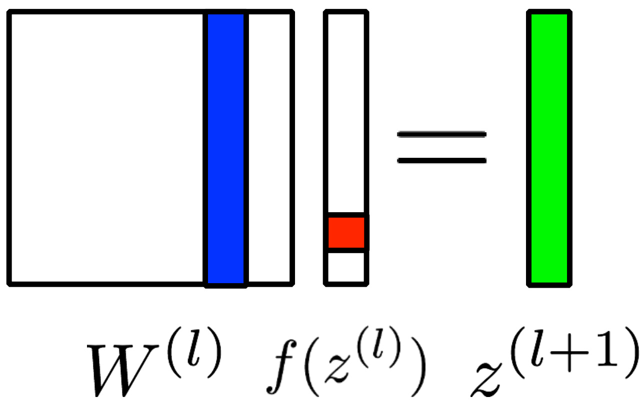
Backward:



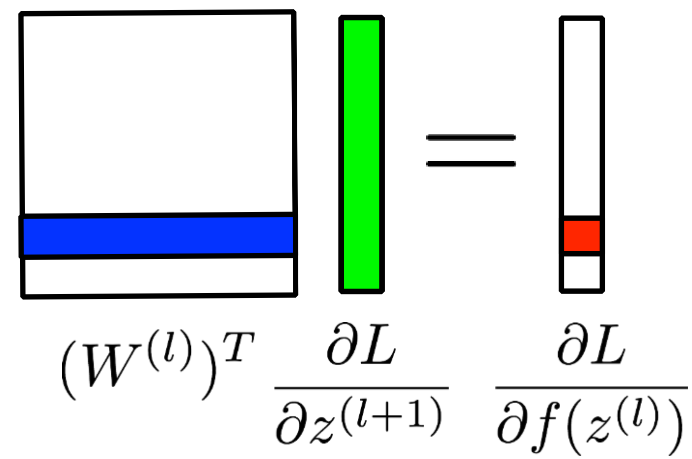
$$(W^{(l)})^T \quad \frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})}$$

Fully Connected Layers:

Forward:

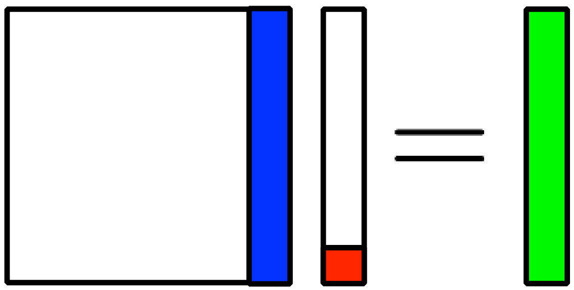


Backward:



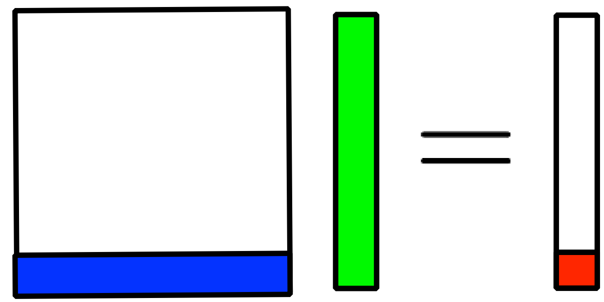
Fully Connected Layers:

Forward:



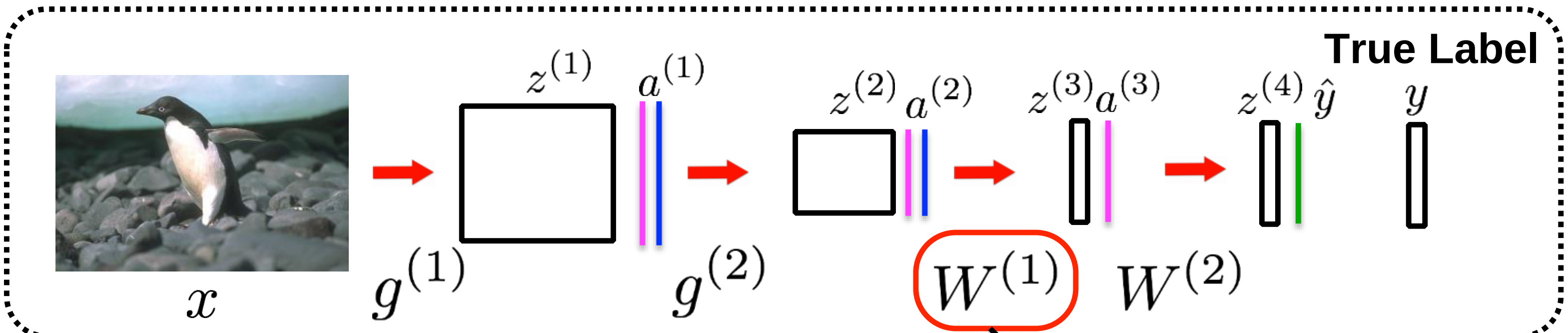
$$W^{(l)} \quad f(z^{(l)}) \quad z^{(l+1)}$$

Backward:



$$(W^{(l)})^T \quad \frac{\partial L}{\partial z^{(l+1)}} \quad \frac{\partial L}{\partial f(z^{(l)})}$$

Backpropagation

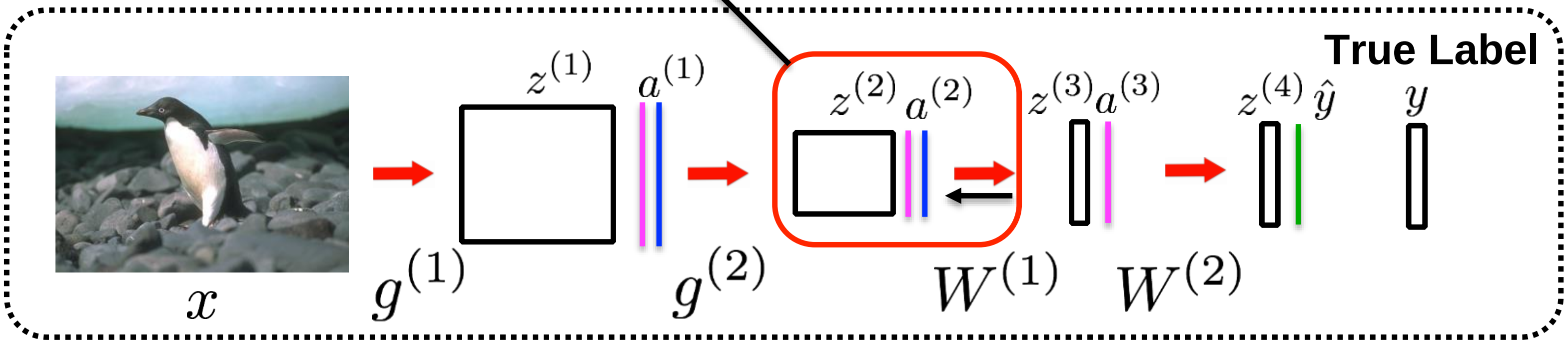


4. Compute the gradients to adjust the weights:
$$\frac{\partial L}{\partial W^{(1)}} = \frac{\partial L}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial W^{(1)}}$$
where $z^{(3)} = W^{(1)} a^{(2)}$

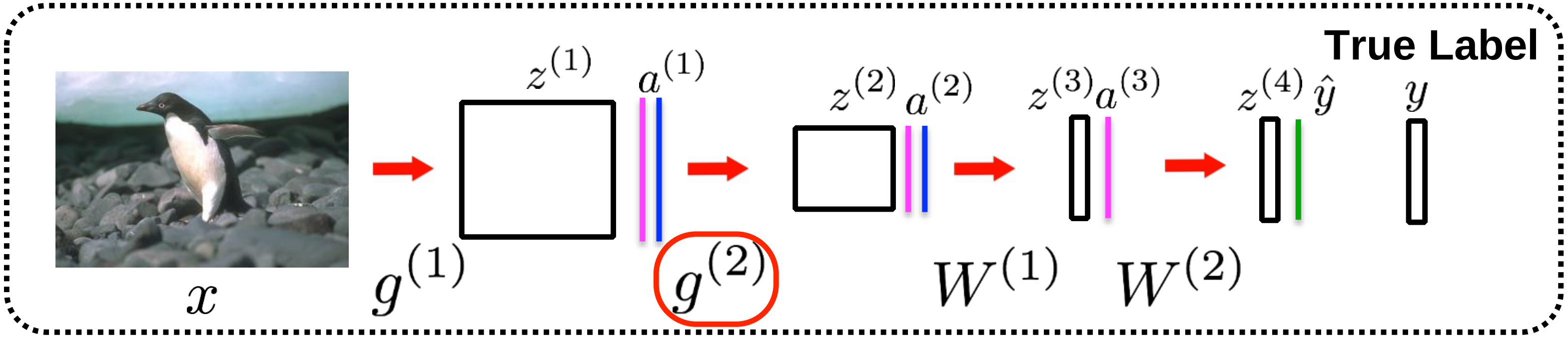
Backpropagation

5. Backpropagate the gradients to previous layers:

$$\frac{\partial L}{\partial z^{(2)}} = \frac{\partial L}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial f(z^{(2)})} \frac{\partial f(z^{(2)})}{\partial z^{(2)}} \quad \text{where } z^{(3)} = W^{(1)} a^{(2)}$$



Backpropagation

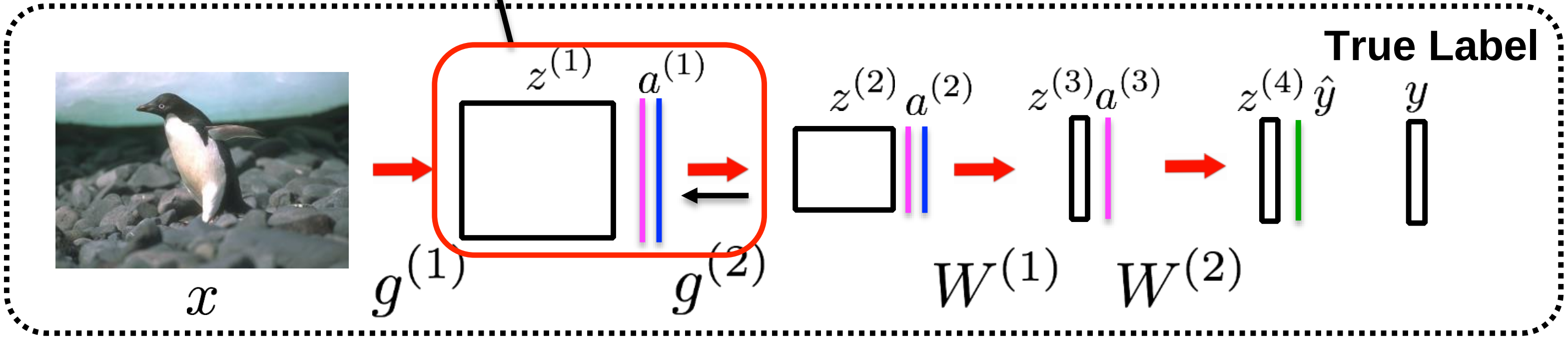


6. Compute the gradients to adjust the weights: $\dfrac{\partial L}{\partial g^{(2)}} = \dfrac{\partial L}{\partial z^{(2)}} \dfrac{\partial z^{(2)}}{\partial g^{(2)}}$ where $z^{(2)} = g^{(2)} * a^{(1)}$

Backpropagation

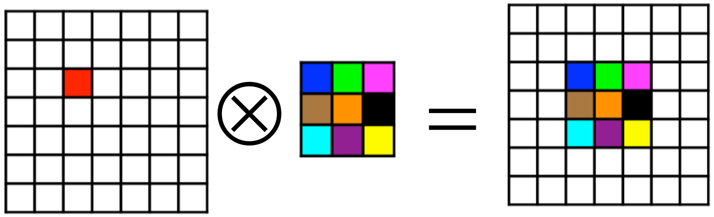
7. Backpropagate the gradients to previous layers:

$$\frac{\partial L}{\partial z^{(1)}} = \frac{\partial L}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial f(z^{(1)})} \frac{\partial f(z^{(1)})}{\partial z^{(1)}}$$ where $z^{(2)} = g^{(2)} * a^{(1)}$



Convolutional Layers:

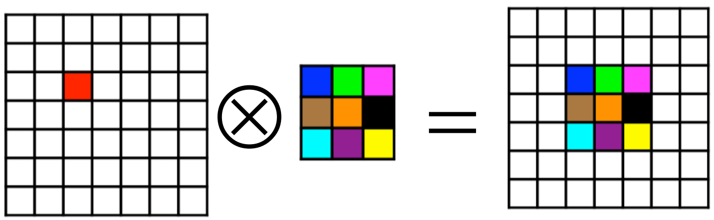
Forward:



$$a^{(l)} \otimes g^{(l)} = z^{(l+1)}$$

Convolutional Layers:

Forward:



$$a^{(l)} \otimes g^{(l)} = z^{(l+1)}$$

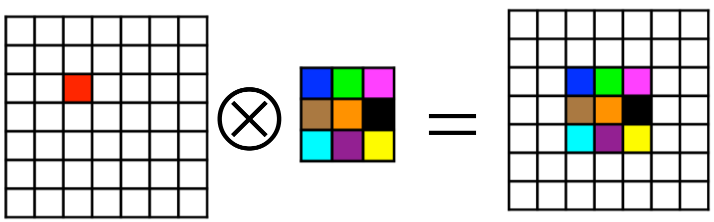
A measure how much an activation unit contributed to the loss

Backward:

$$\frac{\partial L}{\partial z^{(l+1)}} \qquad \frac{\partial L}{\partial f(z^{(l)})}$$

Convolutional Layers:

Forward:

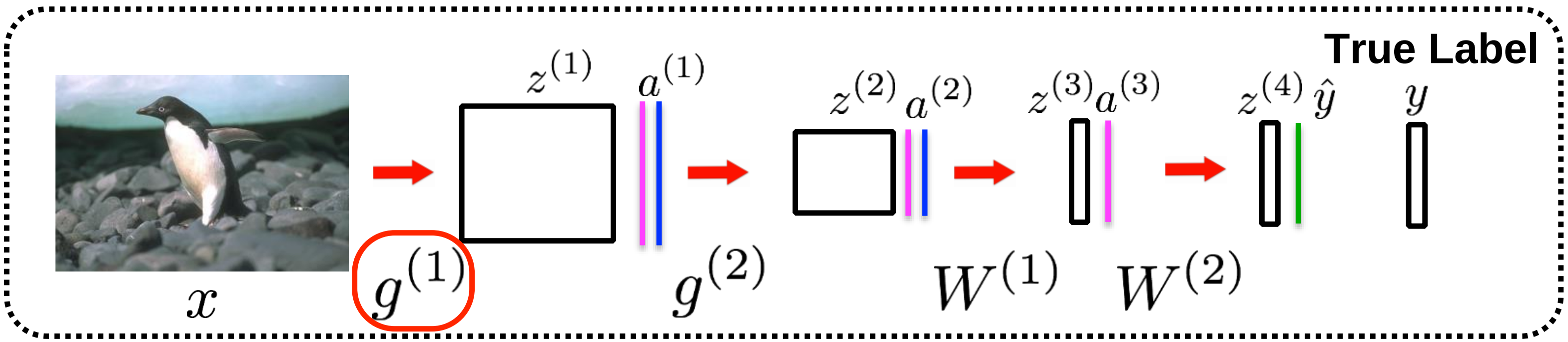


$$a^{(l)} \otimes g^{(l)} = z^{(l+1)}$$

Backward:

$$sum\left(g^{(l)} \odot \frac{\partial L}{\partial z^{(l+1)}} \right) = \frac{\partial L}{\partial f\left(z^{(l)}\right)}$$

Backpropagation



8. Compute the gradients to adjust the weights: $$\frac{\partial L}{\partial g^{(1)}} = \frac{\partial L}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial g^{(1)}}$$ where $z^{(1)} = g^{(1)} * x$

Neural Network Learning

Neural Network Learning

How do we better understanding various properties behind the learning process in neural nets?

How to Improve NN Learning?

Loss / Cost function:

- Loss function defines what we want the neural network to learn.

How to Improve NN Learning?

Loss / Cost function:

- Loss function defines what we want the neural network to learn.

How do we select a good loss function?

How to Improve NN Learning?

Loss / Cost function:

- Loss function defines what we want the neural network to learn.

- Humans learn best when they get feedback after being very wrong (e.g. a person learns to avoid scams after the first time he/she was scammed).

How to Improve NN Learning?

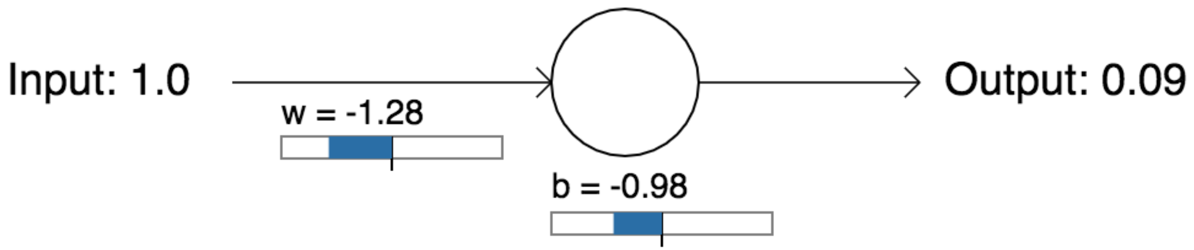
Loss / Cost function:

- Loss function defines what we want the neural network to learn.
- Humans learn best when they get feedback after being very wrong (e.g. a person learns to avoid scams after the first time he/she was scammed).
- Does the same learning trend apply to neural networks? If not we want to design a loss function with such learning characteristics.

How to Improve NN Learning?

Loss / Cost function:

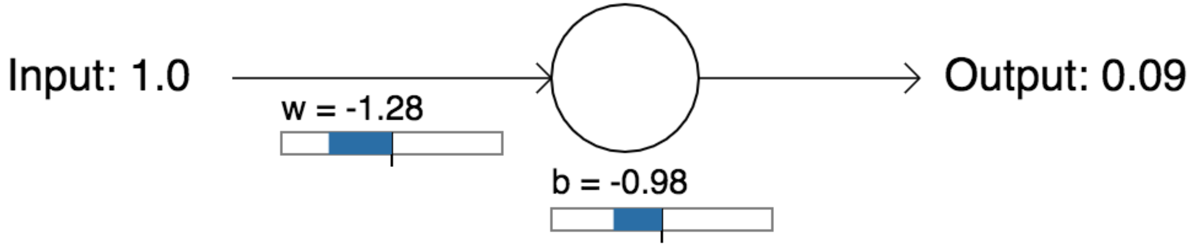
- Consider a neural network consisting of a single hidden neuron:



How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



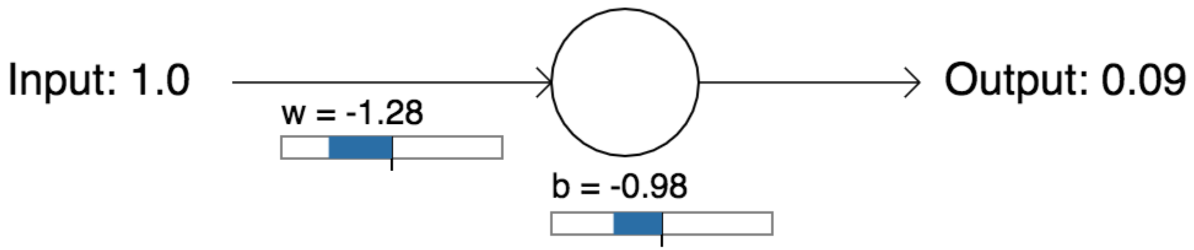
- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

How to Improve NN Learning?

<u>Loss / Cost function:</u>

- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

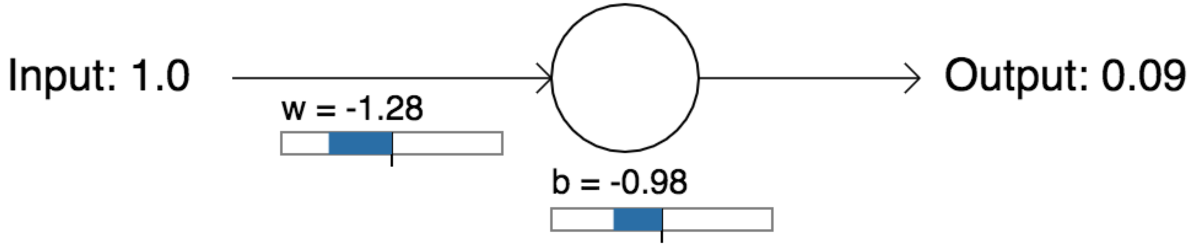
$$L = \frac{1}{2}(\sigma(z) - y)^2$$

NN prediction **ground truth**

How to Improve NN Learning?

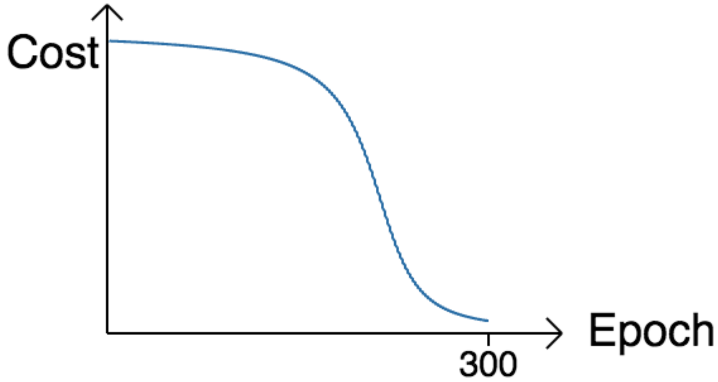
Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

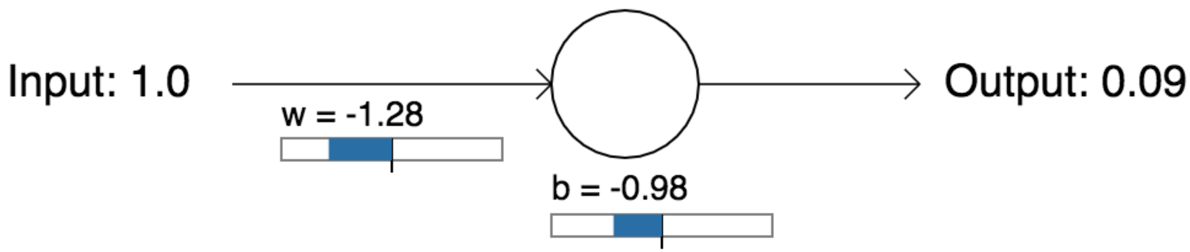
$$L = \frac{1}{2}(\sigma(z) - y)^2$$



How to Improve NN Learning?

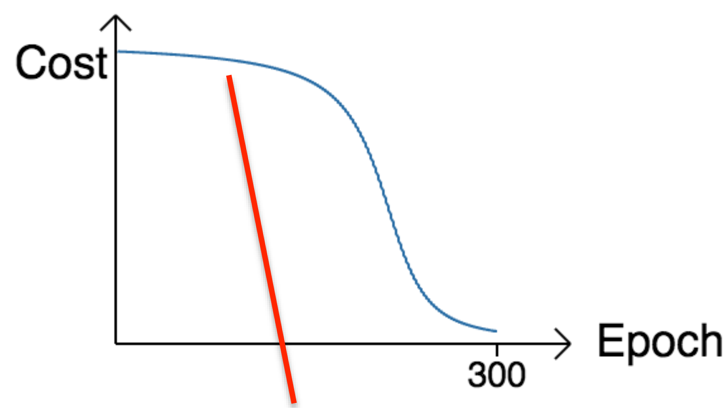
<u>Loss / Cost function:</u>

- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

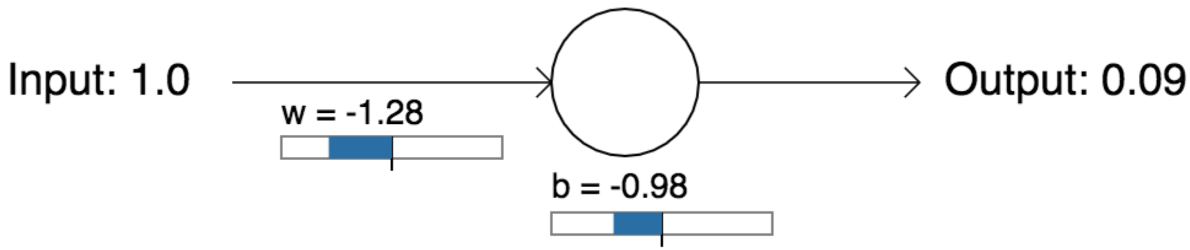


why is the learning slow initially?

How to Improve NN Learning?

Loss / Cost function:

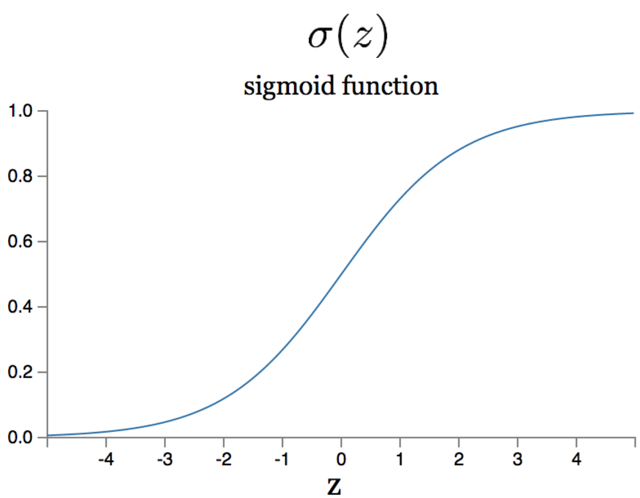
- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

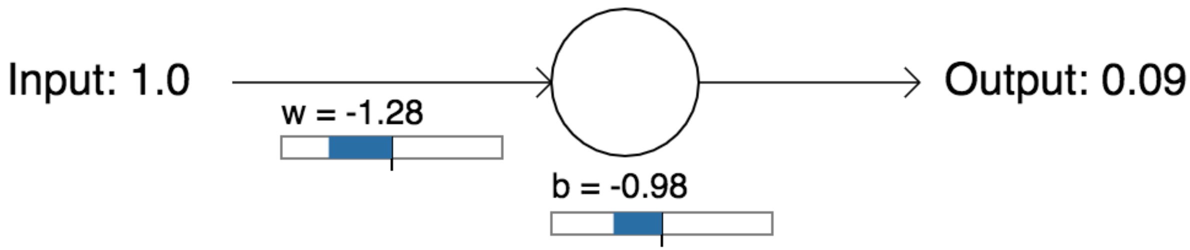
$$\boxed{\frac{\partial L}{\partial w} = (\sigma(z) - y)\sigma'(z)x}$$



How to Improve NN Learning?

Loss / Cost function:

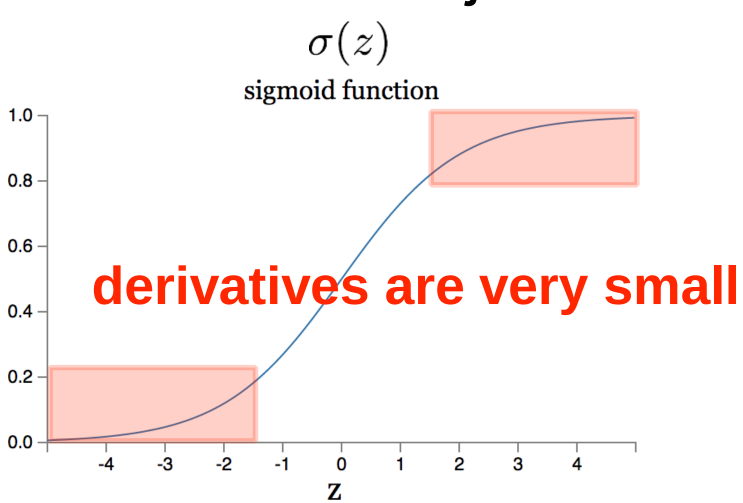
- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

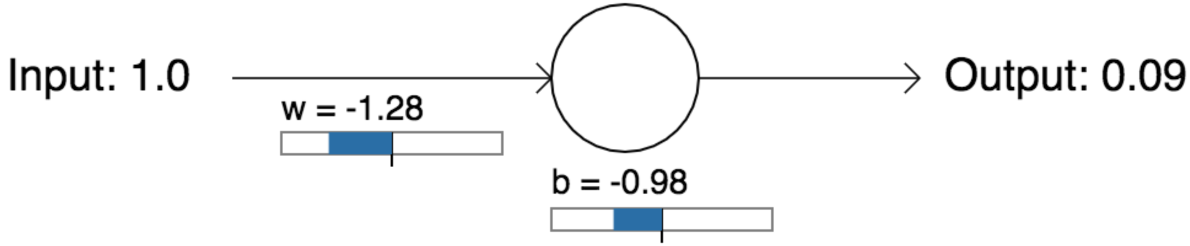
$$\frac{\partial L}{\partial w} = (\sigma(z) - y)\sigma'(z)x$$



How to Improve NN Learning?

Loss / Cost function:

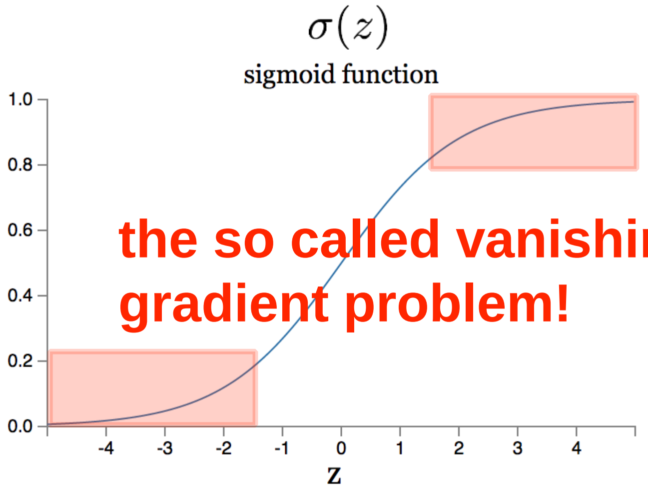
- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

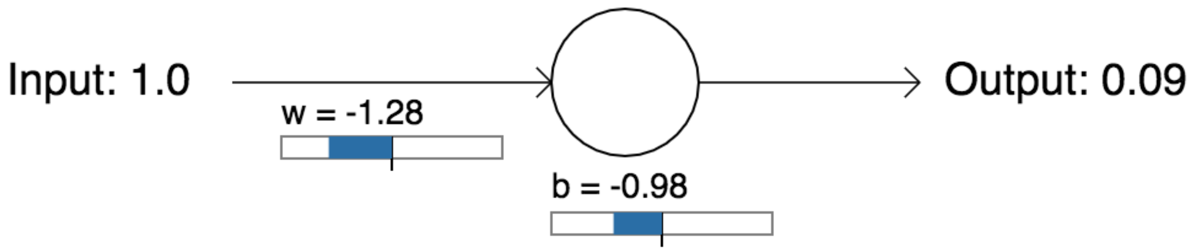
$$\frac{\partial L}{\partial w} = (\sigma(z) - y)\sigma'(z)x$$



How to Improve NN Learning?

Loss / Cost function:

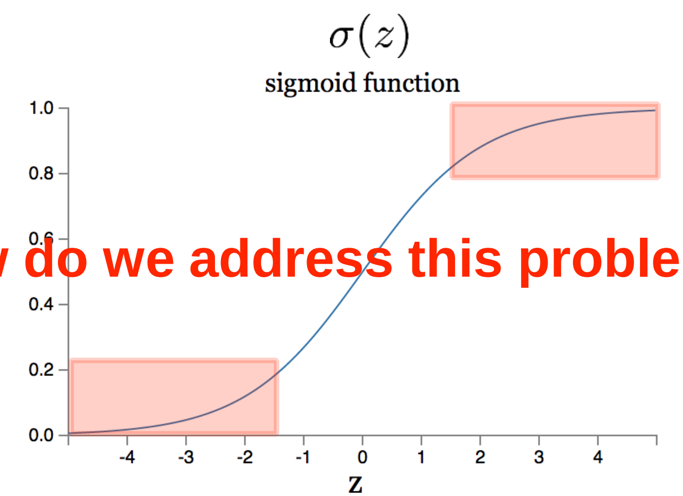
- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

$$\frac{\partial L}{\partial w} = (\sigma(z) - y)\sigma'(z)x$$

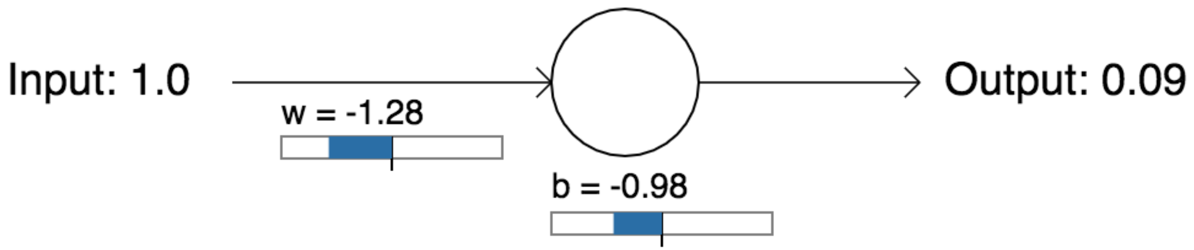


How do we address this problem?

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

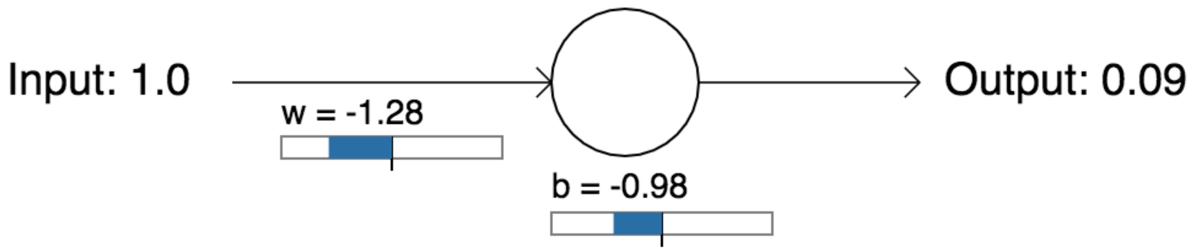
$$\frac{\partial L}{\partial w} = (\sigma(z) - y)\underline{\sigma'(z)}x$$

We need a learning objective that wouldn't have a sigmoid derivative in the gradient.

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



- First, let's examine a commonly used L2 loss objective:

$$L = \frac{1}{2}(\sigma(z) - y)^2$$

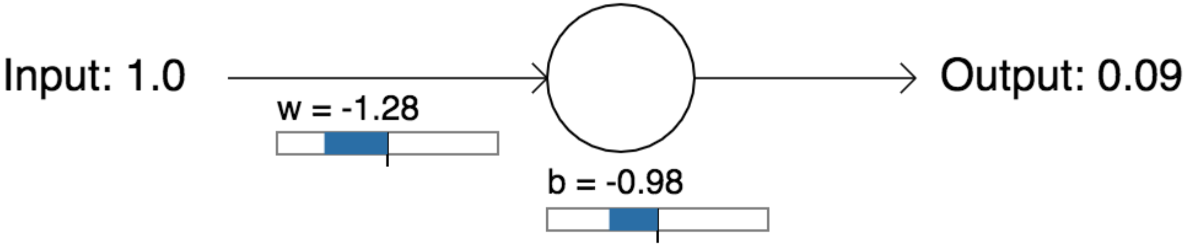
We want the following gradient:

$$\frac{\partial L}{\partial w} = (\sigma(z) - y)\sigma'(z)x \rightarrow \boxed{\frac{\partial L}{\partial w} = (\sigma(z) - y)x}$$

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:

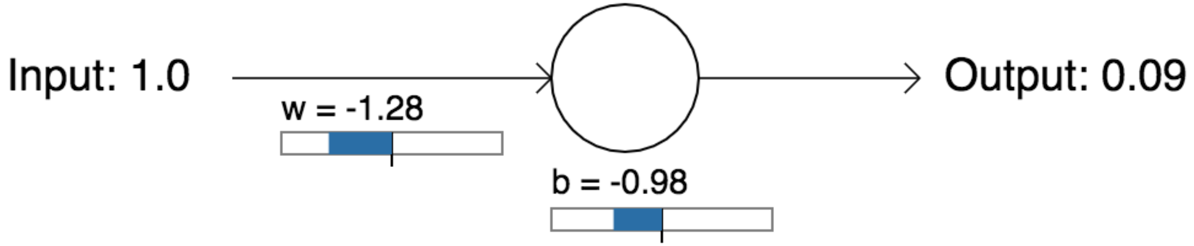


$$\frac{\partial L}{\partial w} = (\sigma(z) - y)x$$ Let $a = \sigma(z)$

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



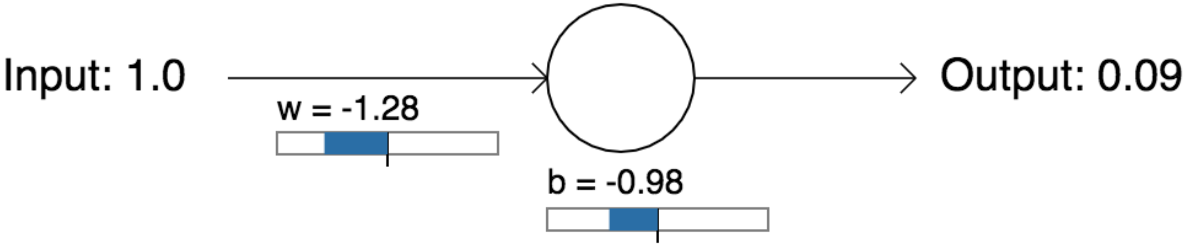
$$\frac{\partial L}{\partial w} = (\sigma(z) - y)x$$ Let $a = \sigma(z)$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w}$$ where $\frac{\partial a}{\partial z} = a(1-a)$ & $\frac{\partial z}{\partial w} = x$

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



$$\frac{\partial L}{\partial w} = (\sigma(z) - y)x \qquad \text{Let} \quad a = \sigma(z)$$

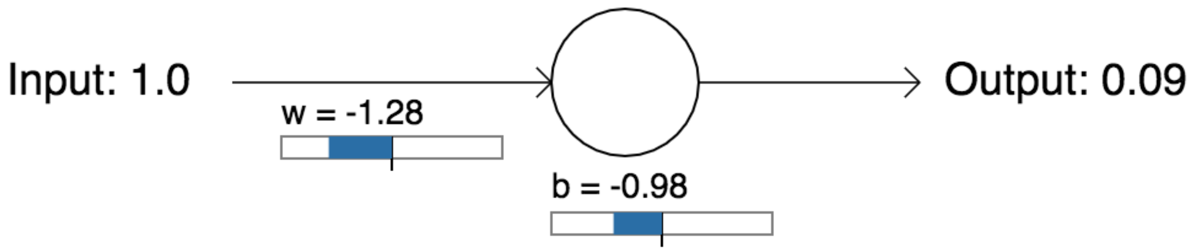
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} \quad \text{where} \quad \frac{\partial a}{\partial z} = a(1-a) \quad \& \quad \frac{\partial z}{\partial w} = x$$

$$(a - y)x = \frac{\partial L}{\partial a}a(1-a)x$$

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



$$\frac{\partial L}{\partial w} = (\sigma(z) - y)x \qquad \text{Let} \quad a = \sigma(z)$$

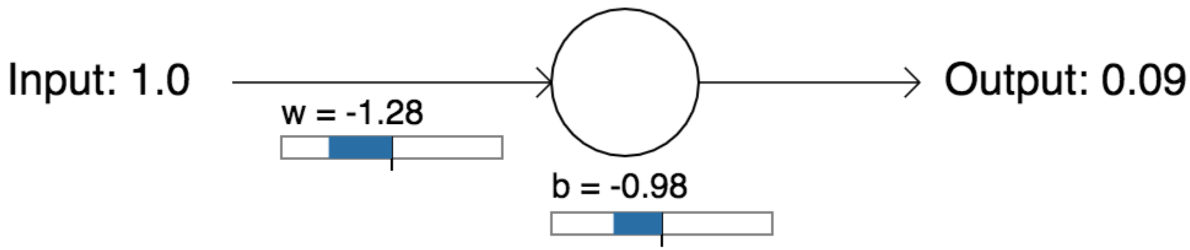
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} \quad \text{where} \quad \frac{\partial a}{\partial z} = a(1-a) \quad \& \quad \frac{\partial z}{\partial w} = x$$

$$(a-y)x = \frac{\partial L}{\partial a}a(1-a)x \longrightarrow \boxed{\frac{\partial L}{\partial a} = \frac{(a-y)}{a(1-a)}}$$

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



$$\frac{\partial L}{\partial w} = (\sigma(z) - y)x \qquad \text{Let} \quad a = \sigma(z)$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} \quad \text{where} \quad \frac{\partial a}{\partial z} = a(1-a) \quad \& \quad \frac{\partial z}{\partial w} = x$$

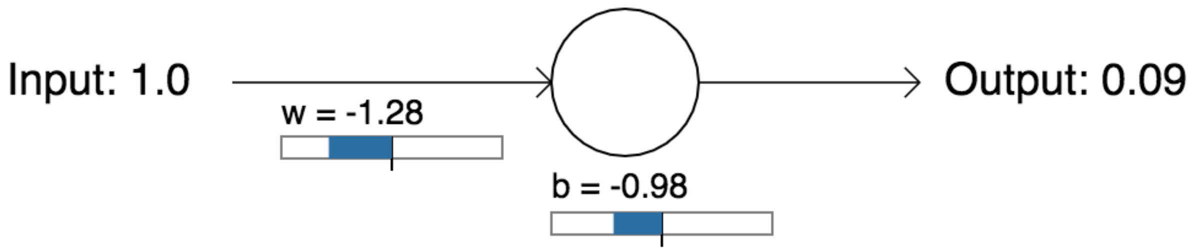
$$(a-y)x = \frac{\partial L}{\partial a}a(1-a)x \longrightarrow \frac{\partial L}{\partial a} = \frac{(a-y)}{a(1-a)}$$

$$\boxed{L = -(y \log a + (1-y)\log(1-a))}$$

How to Improve NN Learning?

<u>Loss / Cost function:</u>

- Consider a neural network consisting of a single hidden neuron:



$$\frac{\partial L}{\partial w} = (\sigma(z) - y)x \qquad \text{Let} \quad a = \sigma(z)$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} \qquad \text{where} \quad \frac{\partial a}{\partial z} = a(1-a) \qquad \& \quad \frac{\partial z}{\partial w} = x$$

$$(a-y)x = \frac{\partial L}{\partial a}a(1-a)x \longrightarrow \frac{\partial L}{\partial a} = \frac{(a-y)}{a(1-a)}$$

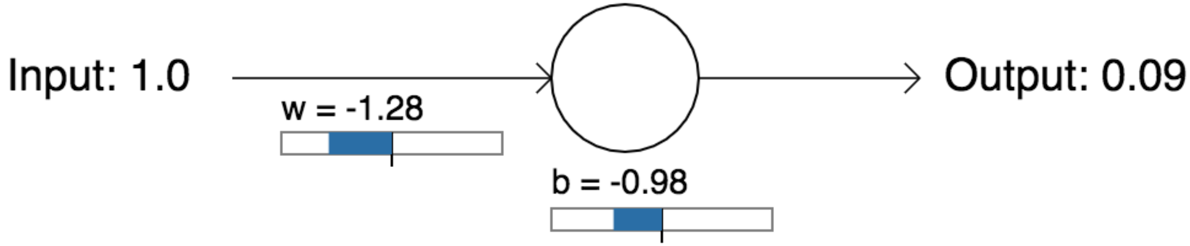
$$\boxed{L = -(y\log a + (1-y)\log(1-a))}$$

We just derived a cross-entropy loss function!

How to Improve NN Learning?

Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



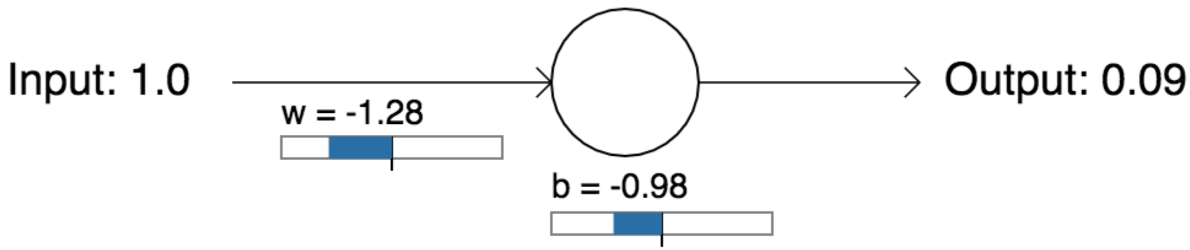
- Now let's examine a cross-entropy loss objective:

$$L = -(y \log a + (1 - y) \log (1 - a))$$

How to Improve NN Learning?

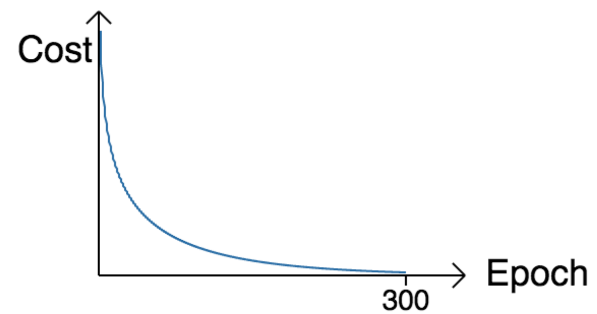
Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



- Now let's examine a cross-entropy loss objective:

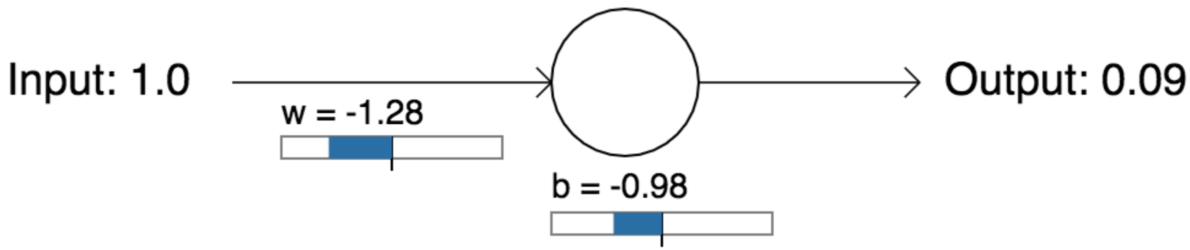
$$L = -(y \log a + (1-y)\log(1-a))$$



How to Improve NN Learning?

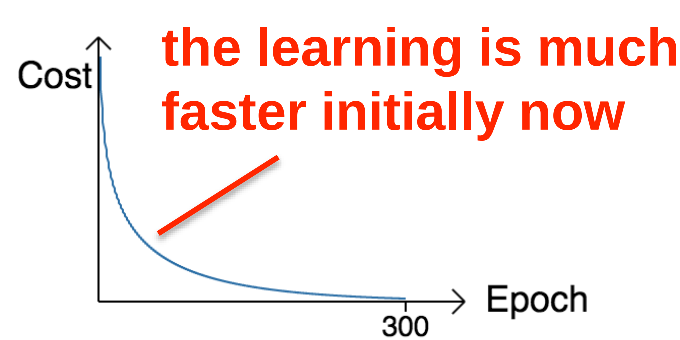
Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



- Now let's examine a cross-entropy loss objective:

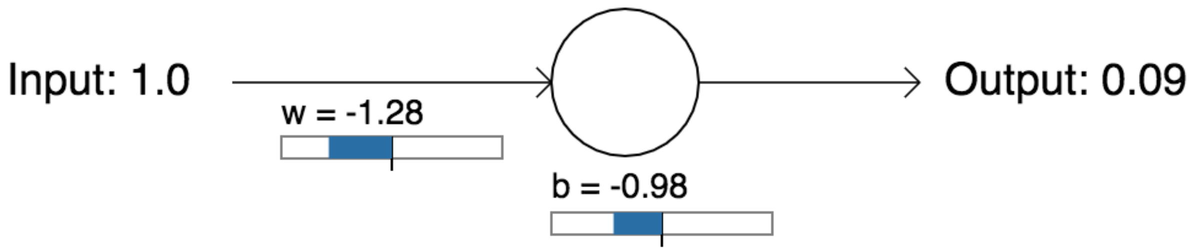
$$L = -(y \log a + (1 - y) \log (1 - a))$$



How to Improve NN Learning?

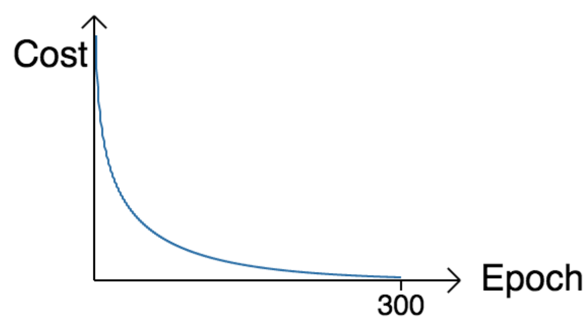
Loss / Cost function:

- Consider a neural network consisting of a single hidden neuron:



- Now let’s examine a cross-entropy loss objective:

$$L = -(y \log a + (1-y) \log (1-a))$$



$$\boxed{\frac{\partial L}{\partial w} = x(a-y)}$$

no sigmoid derivatives in the gradient equation

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$L = -\sum_{i=1}^{K} y_i \log\left(\hat{y}_i\right) \quad \text{where} \quad \hat{y}_i = \frac{\exp\left(z_i^{(4)}\right)}{\sum_{j=1}^{K} \exp\left(z_j^{(4)}\right)}$$

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$L = -\sum_{i=1}^{K} y_i \log\left(\hat{y}_i\right) \quad \text{where} \quad \hat{y}_i = \frac{\exp\left(z_i^{(4)}\right)}{\sum_{j=1}^{K} \exp\left(z_j^{(4)}\right)}$$

$$\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}$$

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$L = -\sum_{i=1}^{K} y_i \log\left(\hat{y}_i\right) \quad \text{where} \quad \hat{y}_i = \frac{\exp\left(z_i^{(4)}\right)}{\sum_{j=1}^{K} \exp\left(z_j^{(4)}\right)}$$

$$\frac{\partial L}{\partial z_i^{(4)}} = \boxed{\frac{\partial L}{\partial \hat{y}}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}$$

$$\boxed{\frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}}$$

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$L = -\sum_{i=1}^{K} y_i \log(\hat{y}_i) \quad \text{where} \quad \hat{y}_i = \frac{\exp(z_i^{(4)})}{\sum_{j=1}^{K} \exp(z_j^{(4)})}$$

$$\frac{\partial L}{\partial z_i^{(4)}} = \boxed{\frac{\partial L}{\partial \hat{y}}} \boxed{\frac{\partial \hat{y}}{\partial z_i^{(4)}}}$$

$$\boxed{\frac{\partial L}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i}}$$

$$\boxed{\frac{\partial \hat{y}_i}{\partial z_j^{(4)}} = \begin{cases} \hat{y}_i(1-\hat{y}_i), & \text{if} \quad i = j \\ -\hat{y}_i \hat{y}_j, & \text{if} \quad i \neq j \end{cases}}$$

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}$$

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}$$

$$= \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^{(4)}} + \sum_{i \neq j} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^{(4)}}$$

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}$$

$$= \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^{(4)}} + \sum_{i \neq j} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^{(4)}}$$

$$\boxed{= \hat{y}_i - y_i}$$

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}$$

$$= \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^{(4)}} + \sum_{i \neq j} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^{(4)}}$$

$$\boxed{= \hat{y}_i - y_i}$$

- **No sigmoid derivatives in the gradient!**

How to Improve NN Learning?

Loss / Cost function:

- How about the softmax loss function?

$$\frac{\partial L}{\partial z_i^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_i^{(4)}}$$

$$= \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^{(4)}} + \sum_{i \neq j} \frac{\partial L}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^{(4)}}$$

$$\boxed{= \hat{y}_i - y_i}$$

- **No sigmoid derivatives in the gradient!**
- **Therefore, learning shouldn't be slowed down.**

Vanishing Gradients

Notation:

- convolutional layer output

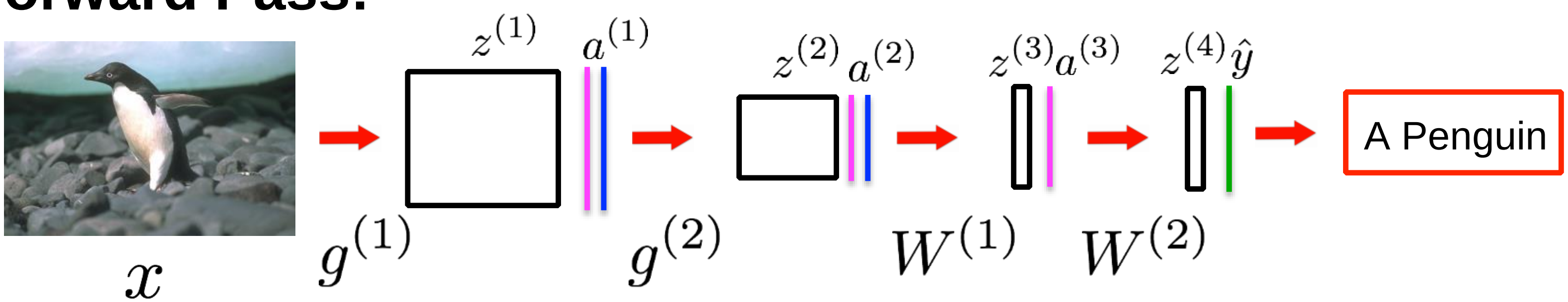
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:



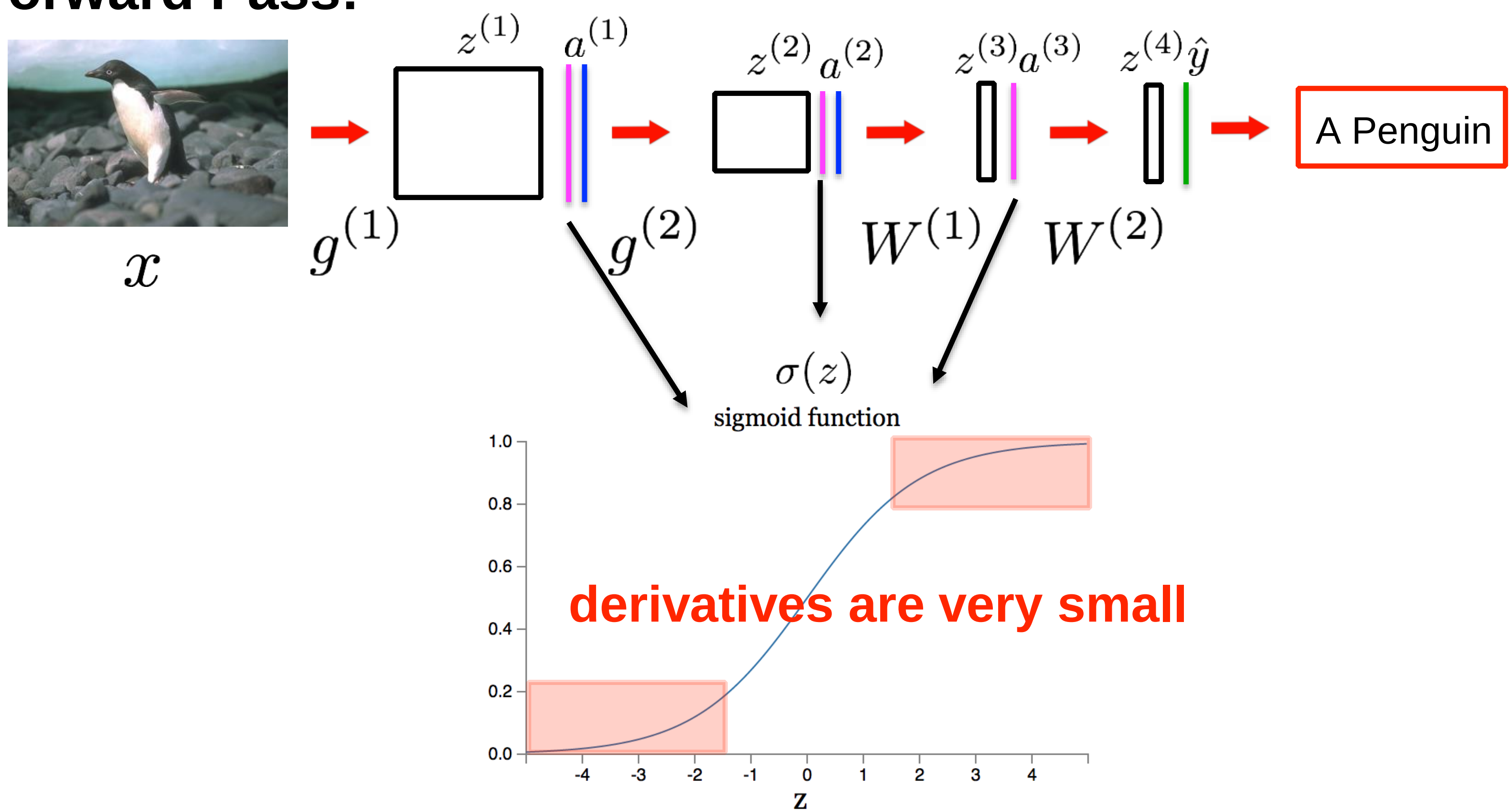
- **Did we solve the vanishing gradient problem?**

Vanishing Gradients

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



Vanishing Gradients

Notation:

- convolutional layer output

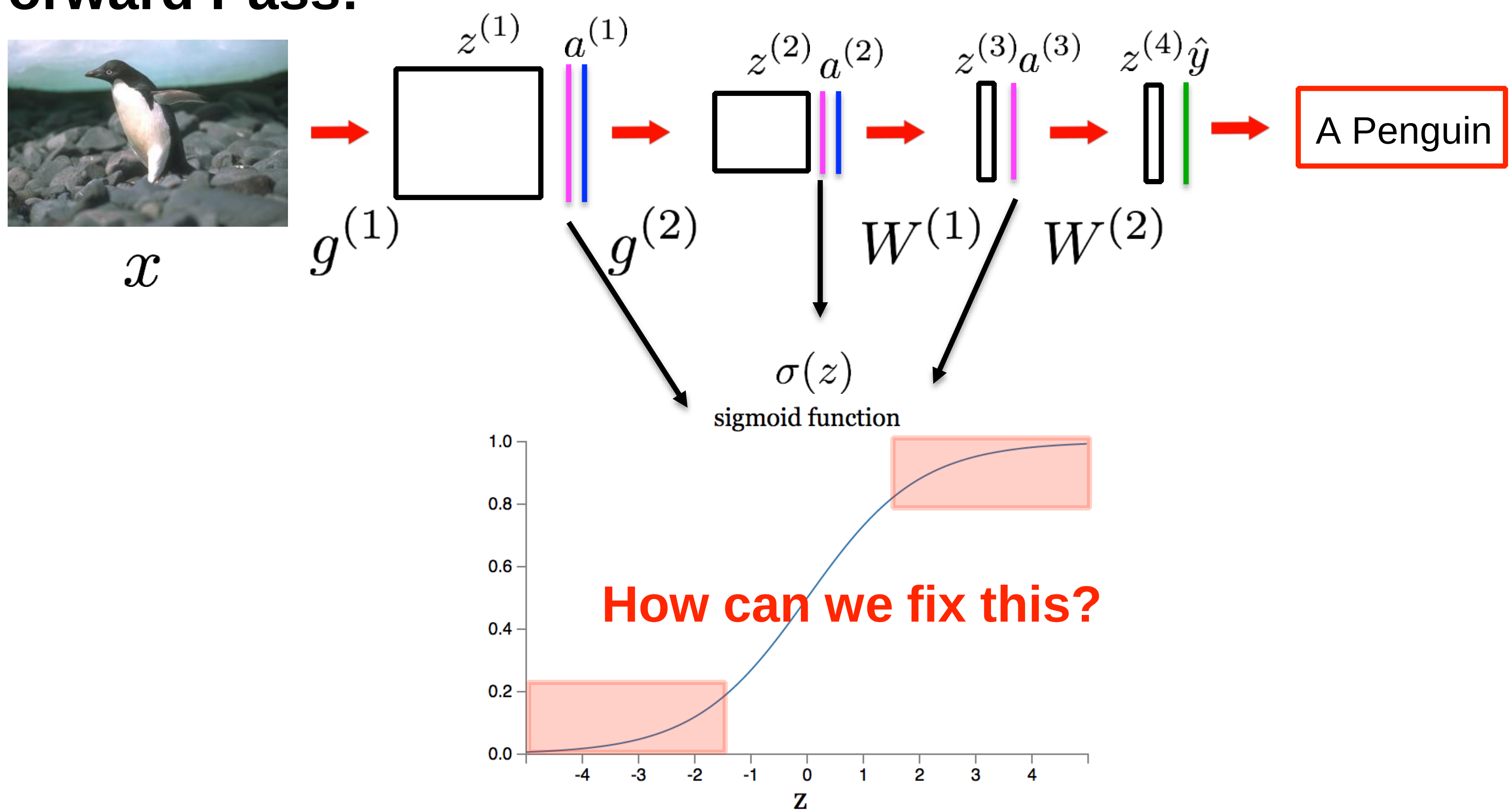
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:

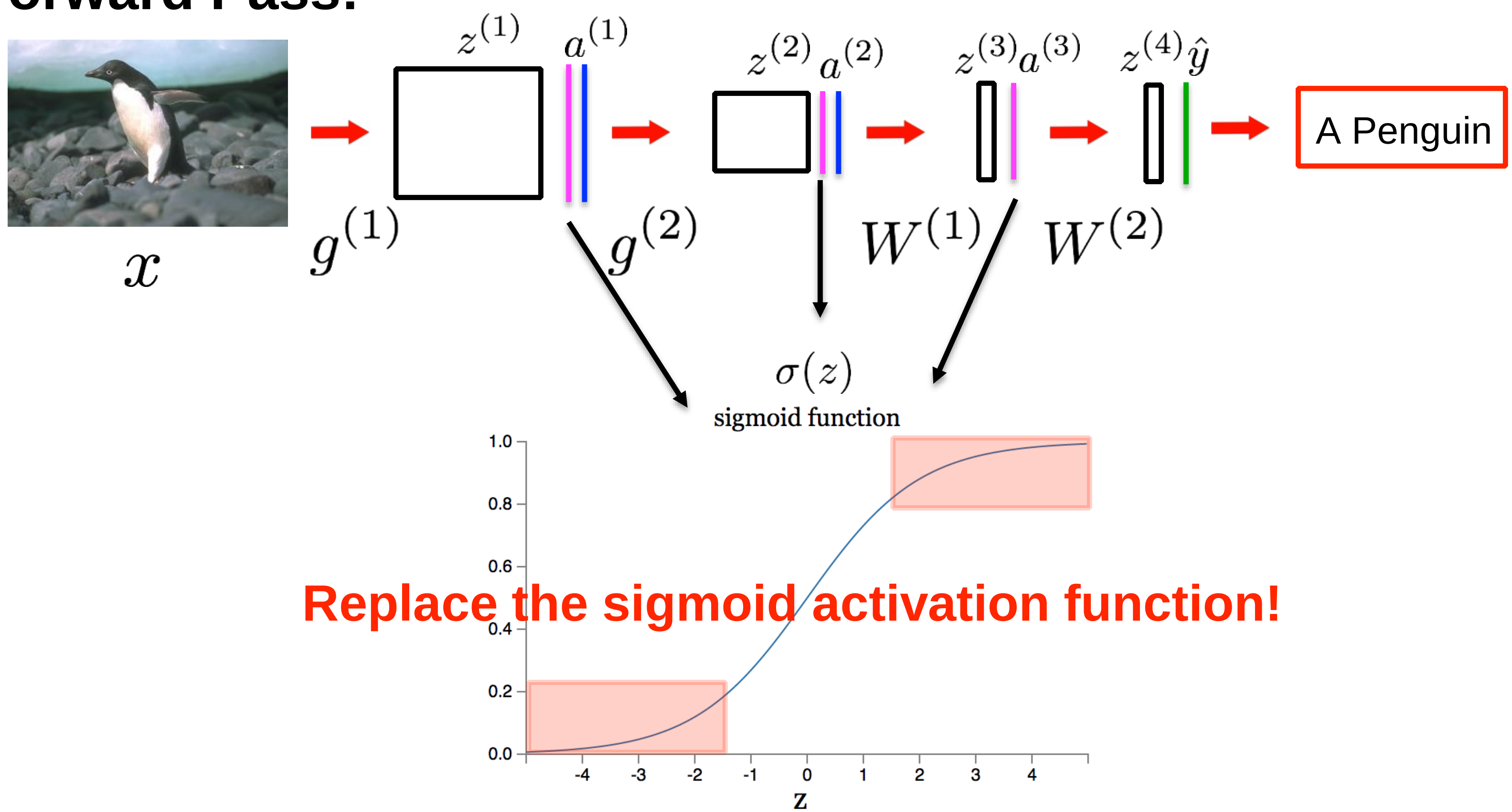


Vanishing Gradients

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



Replace the sigmoid activation function!

Vanishing Gradients

Activation function requirements:

- We want the activation function to be non-linear.
- We want the activation function to be differentiable.
- We want an activation function that eliminates the vanishing gradient problem.

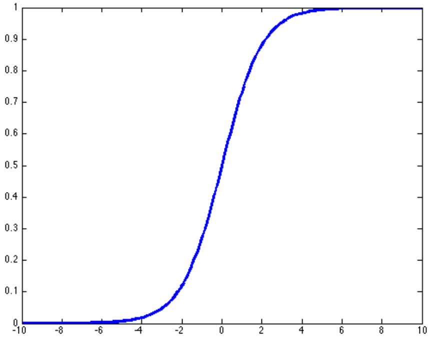
Vanishing Gradients

<u>Activation function requirements:</u>

- We want the activation function to be non-linear.
- We want the activation function to be differentiable.
- We want an activation function that eliminates the vanishing gradient problem.

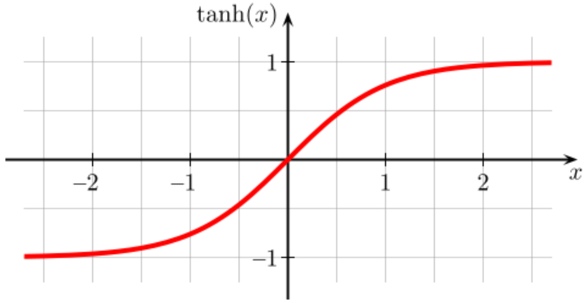
Sigmoid Function

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



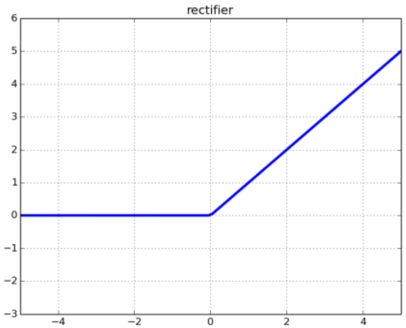
Hyperbolic Tangent

$$f(z) = tanh(z)$$



RELU

$$f(z) = max(0, z)$$



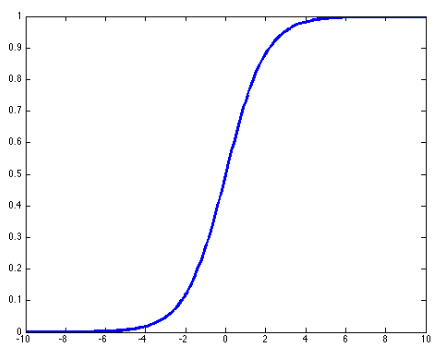
Vanishing Gradients

Activation function requirements:

- We want the activation function to be non-linear.
- We want the activation function to be differentiable.
- We want an activation function that eliminates the vanishing gradient problem.

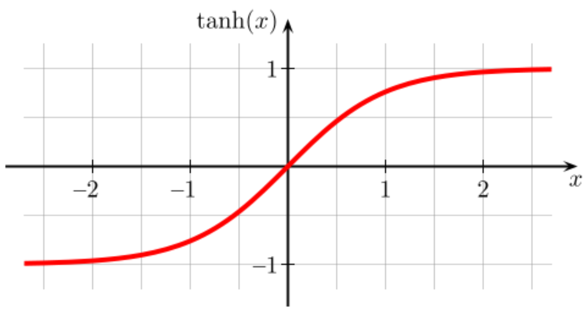
Sigmoid Function

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



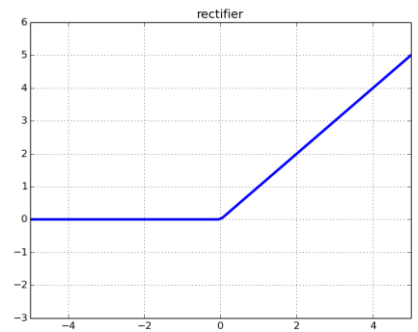
Hyperbolic Tangent

$$f(z) = tanh(z)$$



RELU

$$f(z) = max(0, z)$$



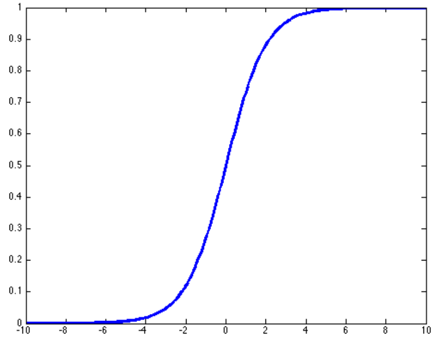
Vanishing Gradients

<u>Activation function requirements:</u>

- We want the activation function to be non-linear.
- We want the activation function to be differentiable.
- We want an activation function that eliminates the vanishing gradient problem.

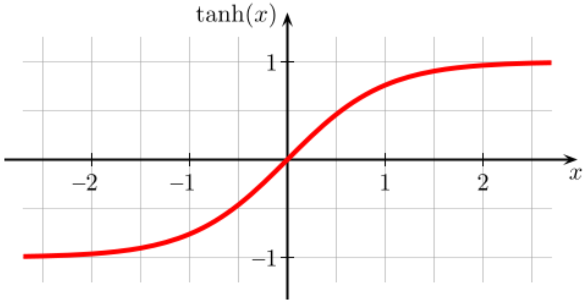
Sigmoid Function

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



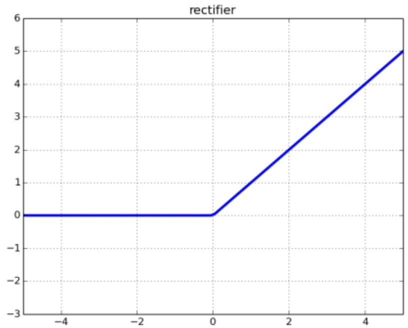
Hyperbolic Tangent

$$f(z) = tanh(z)$$



RELU

$$f(z) = max(0, z)$$



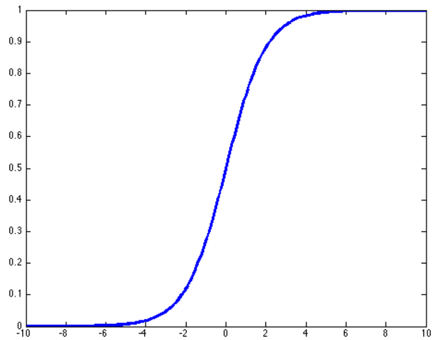
Vanishing Gradients

Activation function requirements:

- We want the activation function to be non-linear.
- We want the activation function to be differentiable.
- We want an activation function that eliminates the vanishing gradient problem.

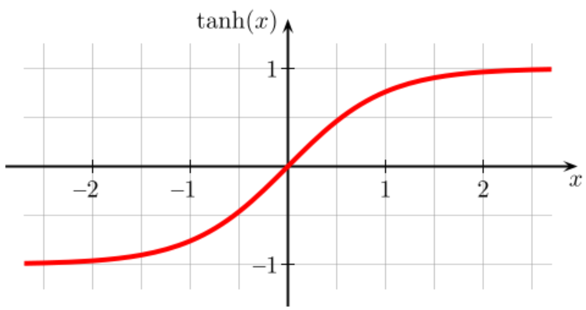
Sigmoid Function

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



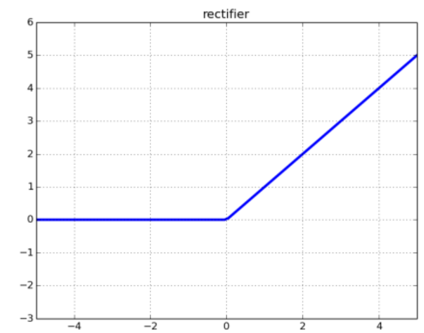
Hyperbolic Tangent

$$f(z) = tanh(z)$$



RELU

$$f(z) = max(0, z)$$



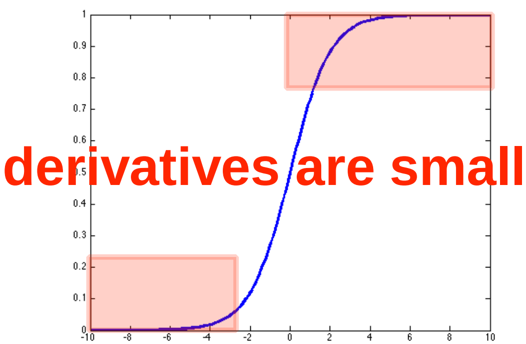
Vanishing Gradients

Activation function requirements:

- We want the activation function to be non-linear.
- We want the activation function to be differentiable.
- We want an activation function that eliminates the vanishing gradient problem.

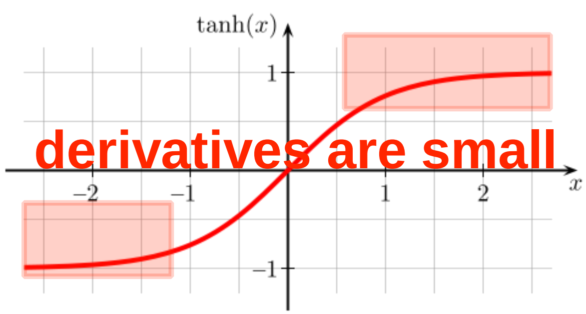
Sigmoid Function

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



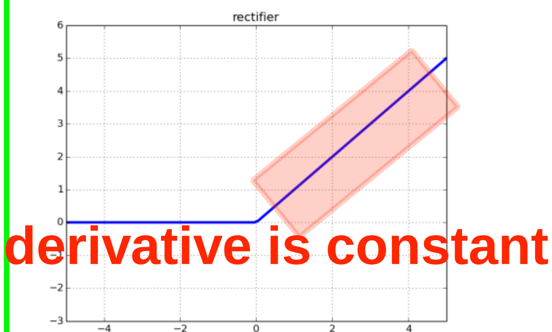
Hyperbolic Tangent

$$f(z) = tanh(z)$$



RELU

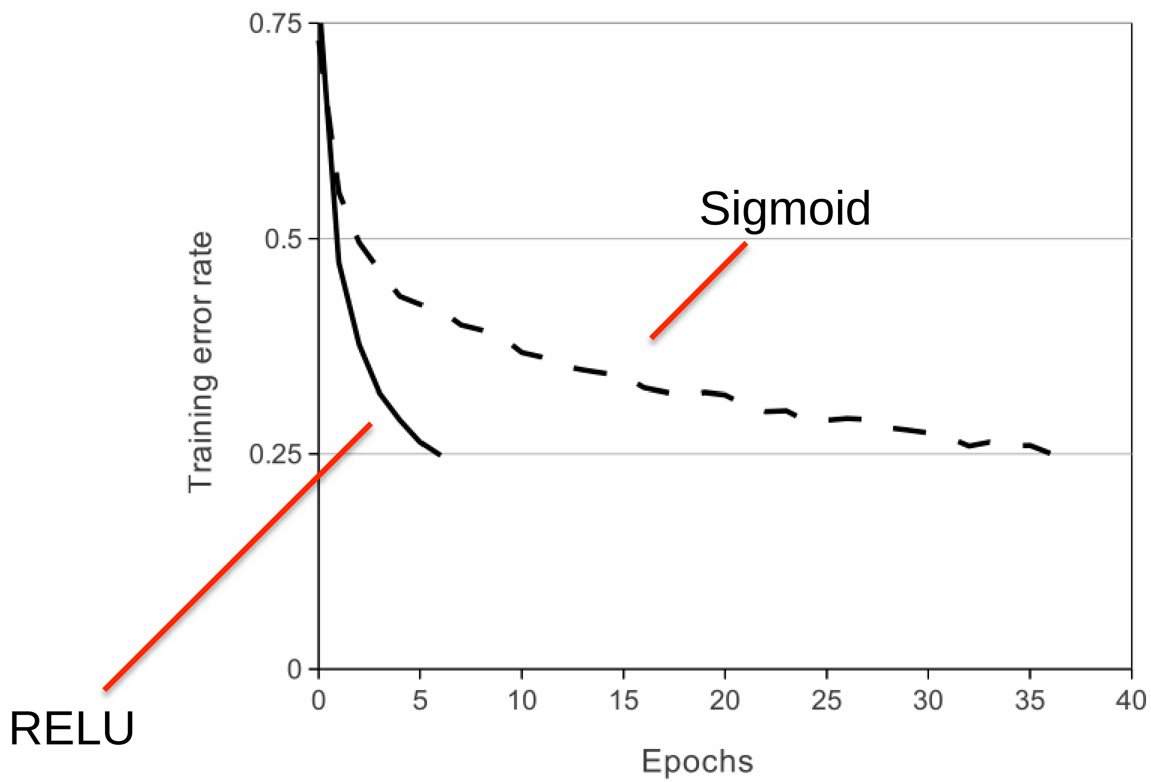
$$f(z) = max(0, z)$$



Vanishing Gradients

<u>Learning Speed:</u>

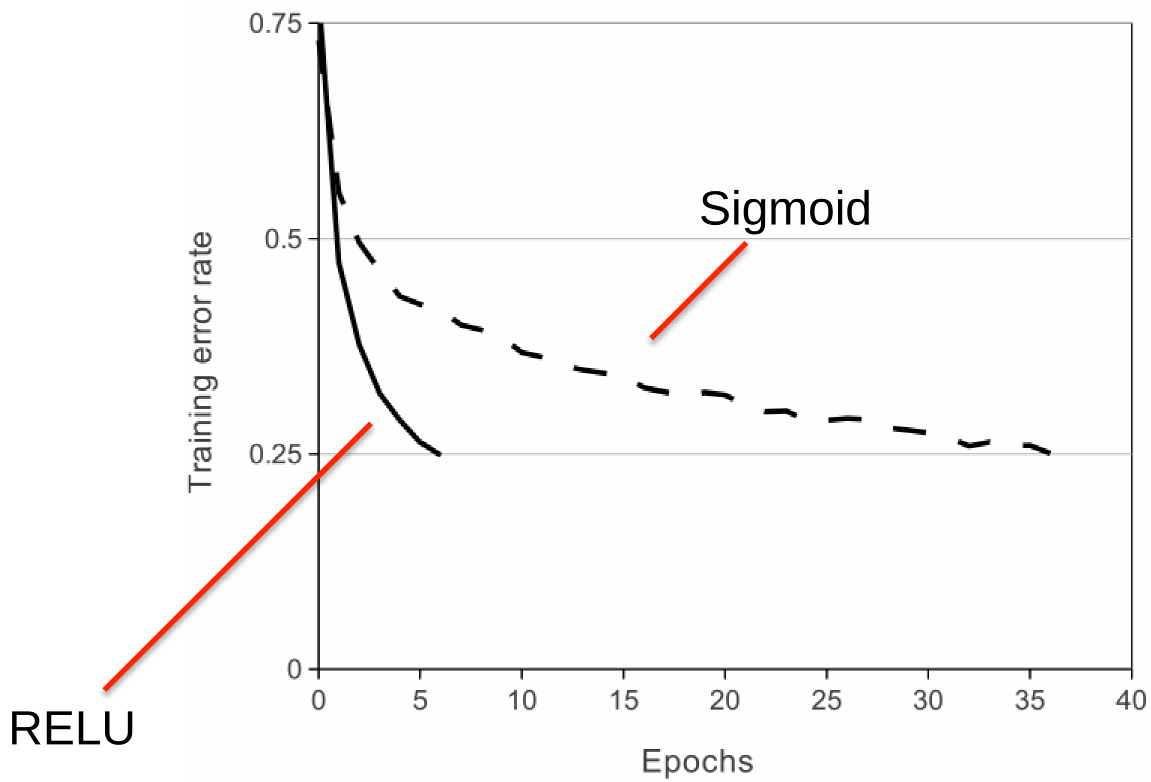
- The network that uses a RELU activation function learns significantly faster.



Vanishing Gradients

Learning Speed:

- The network that uses a RELU activation function learns significantly faster.

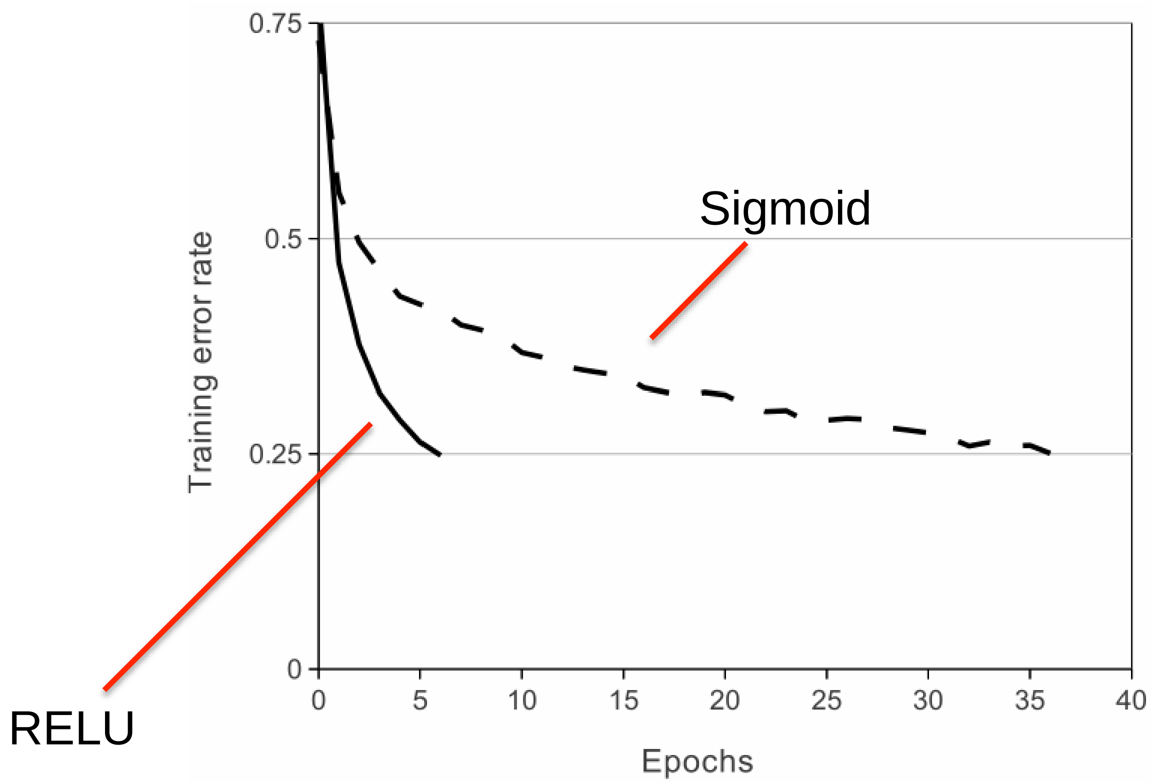


Training is significantly faster when using the RELU function

Vanishing Gradients

Learning Speed:

- The network that uses a RELU activation function learns significantly faster.



Does this mean the problem of vanishing gradients is completely solved?

Vanishing Gradients

Learning Speed:

- It turns out that even using RELU activation function doesn't completely eliminate the vanishing gradient problem.

- The key question is why the vanishing gradient problem still persists.

- To understand this we need to revisit the original back propagation algorithm.

Backpropagation

1. Let $\frac{\partial L}{\partial z_i^{(n)}} = \hat{y}_i - y_i$, where n denotes the number of layers in the network.

2. For each fully connected layer l:

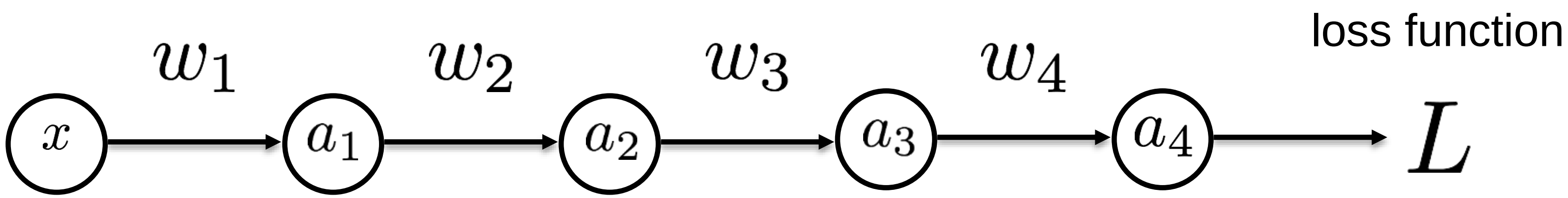
   - For each node i in layer l set:

$$\frac{\partial L}{\partial z_i^{(l)}} = (\sum_{j=1}^{s^{l+1}} W_{ji}^{(l)} \frac{\partial L}{\partial z_j^{(l+1)}}) \frac{\partial f(z_i^{(l)})}{\partial z_i^{(l)}}$$

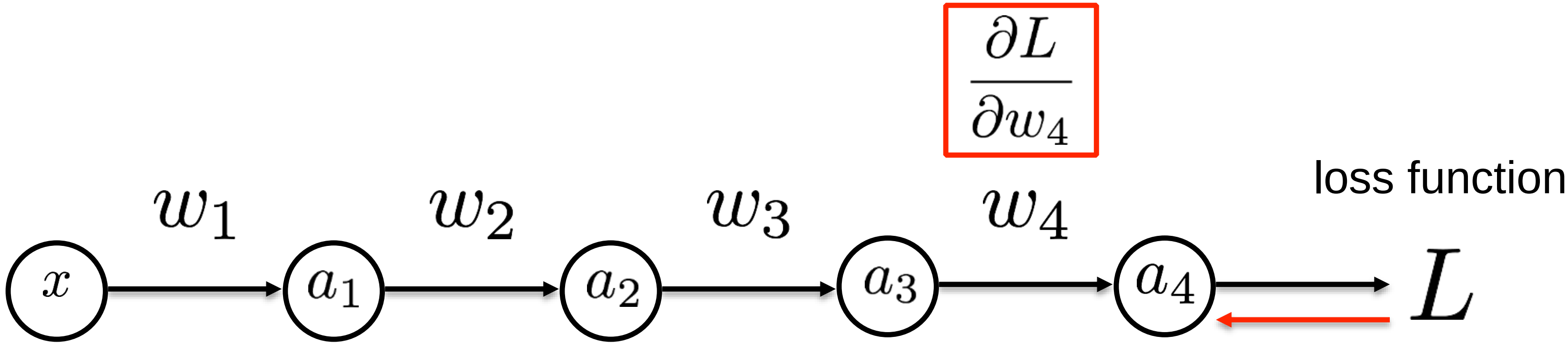
   - Compute partial derivatives: $\frac{\partial L}{\partial W_{ij}^{(l)}} = f(z_j^{(l)}) \frac{\partial L}{\partial z_i^{(l+1)}}$

   - Update the parameters: $W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial L}{\partial W_{ij}^{(l)}}$

Vanishing Gradients



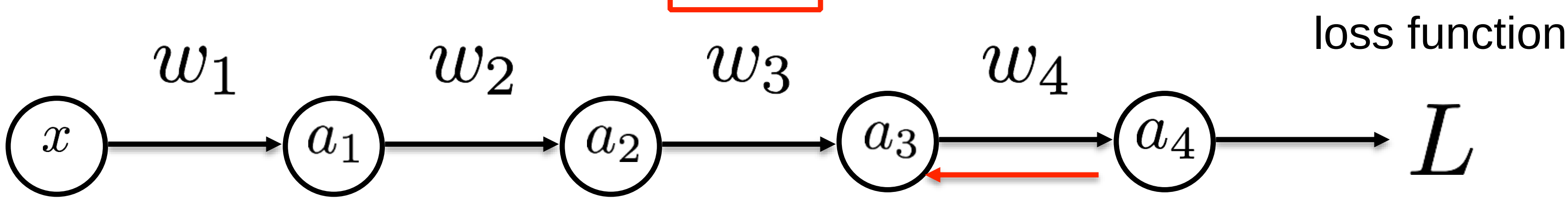
Vanishing Gradients



$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial a_4} \sigma'(z_4) a_3$$

Vanishing Gradients

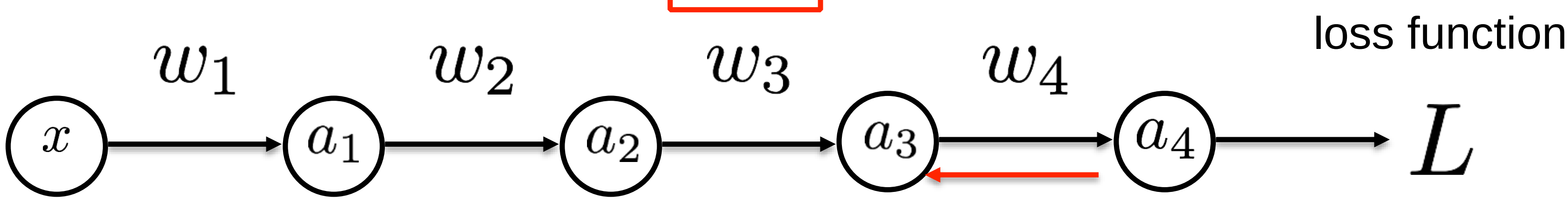
$$\boxed{\frac{\partial L}{\partial w_3}}$$



$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial a_3}\sigma'(z_3)a_2$$

Vanishing Gradients

$$\boxed{\frac{\partial L}{\partial w_3}}$$

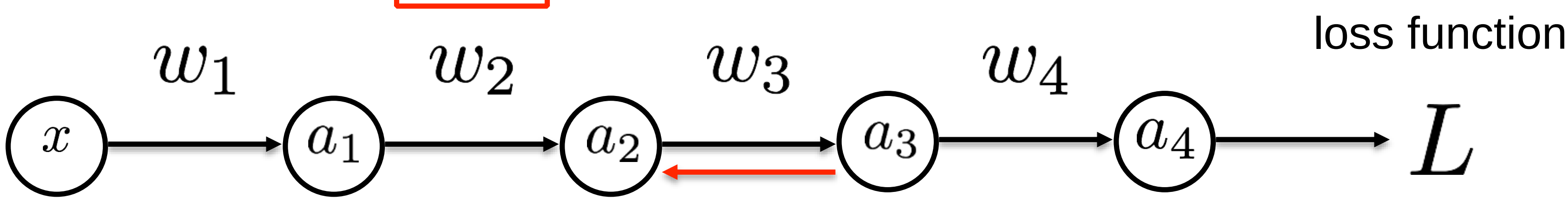


$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial a_3}\sigma'(z_3)a_2$$

$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)w_4\sigma'(z_3)a_2$$

Vanishing Gradients

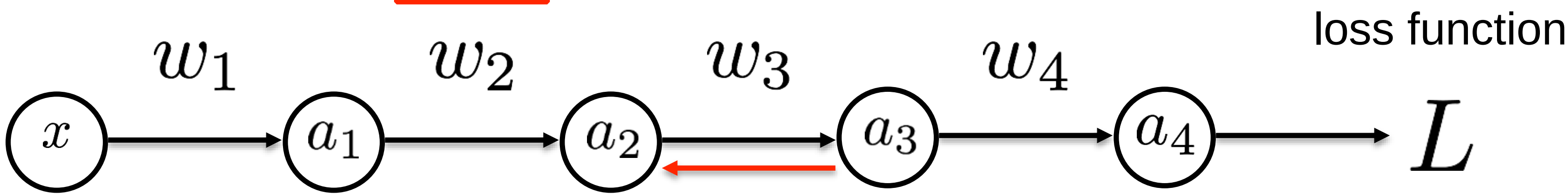
$$\boxed{\frac{\partial L}{\partial w_2}}$$



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_2}\sigma'(z_2)a_1$$

Vanishing Gradients

$$\boxed{\frac{\partial L}{\partial w_2}}$$

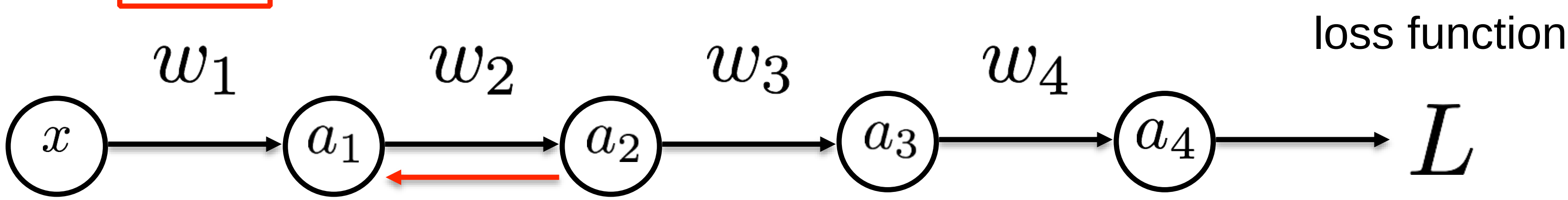


$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_2}\sigma'(z_2)a_1$$

$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)w_4\sigma'(z_3)w_3\sigma'(z_2)a_1$$

Vanishing Gradients

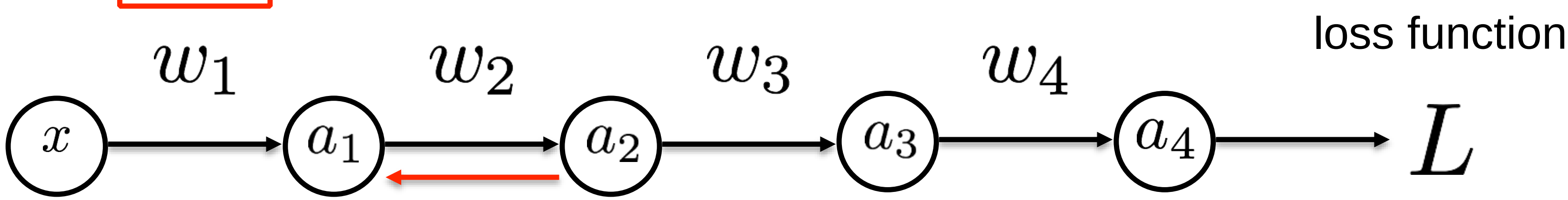
$$\boxed{\frac{\partial L}{\partial w_1}}$$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\sigma'(z_1)x$$

Vanishing Gradients

$$\boxed{\frac{\partial L}{\partial w_1}}$$

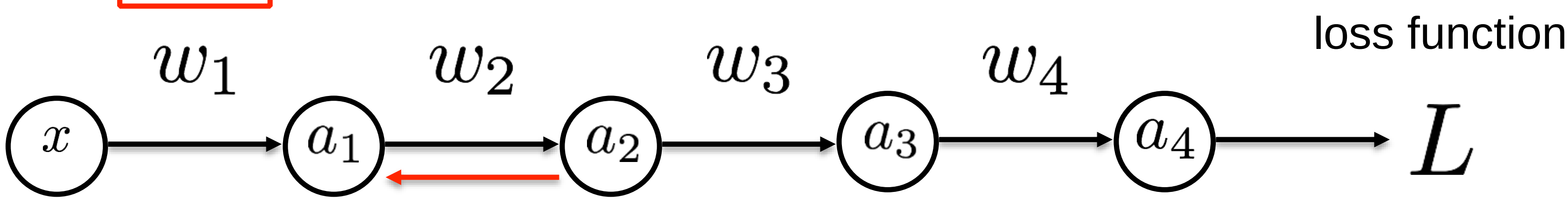


$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\sigma'(z_1)x$$

$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)w_4\sigma'(z_3)w_3\sigma'(z_2)w_2\sigma'(z_1)x$$

Vanishing Gradients

$$\boxed{\frac{\partial L}{\partial w_1}}$$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\sigma'(z_1)x$$

$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)\underbrace{w_4}_{\sim|0.01|}\sigma'(z_3)\underbrace{w_3}_{\sim|0.01|}\sigma'(z_2)\underbrace{w_2}_{\sim|0.01|}\sigma'(z_1)x$$

- **Weights are typically initialized to small values (e.g. gaussian distribution with 0 mean and 0.01 std dev)**

Vanishing Gradients

$$\boxed{\frac{\partial L}{\partial w_1}}$$

loss function

$$x \xrightarrow{w_1} a_1 \xrightarrow{w_2} a_2 \xrightarrow{w_3} a_3 \xrightarrow{w_4} a_4 \longrightarrow L$$

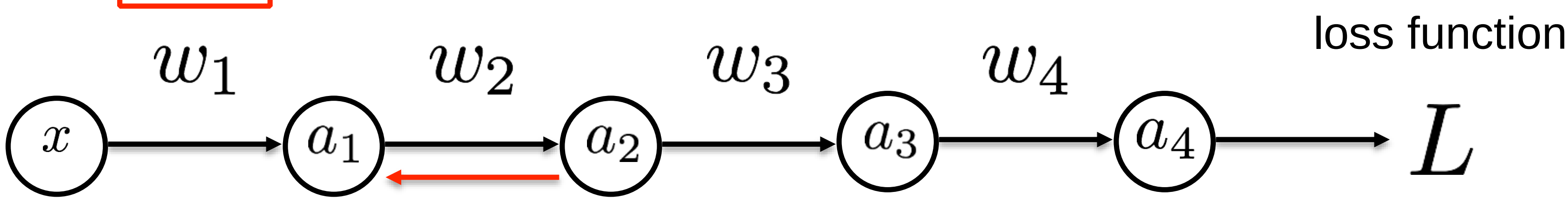
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\sigma'(z_1)x$$

$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)\underbrace{w_4}_{\sim|0.01|}\sigma'(z_3)\underbrace{w_3}_{\sim|0.01|}\sigma'(z_2)\underbrace{w_2}_{\sim|0.01|}\sigma'(z_1)x$$

- **Exponential decrease in the gradient as we move towards the early hidden layers.**

Vanishing Gradients

$$\boxed{\frac{\partial L}{\partial w_1}}$$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\sigma'(z_1)x$$

$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)w_4\sigma'(z_3)w_3\sigma'(z_2)w_2\sigma'(z_1)x$$

~|0.01| ~|0.01| ~|0.01|

- **As a result, the deepest hidden layers learn significantly faster, relative to the early layers that may not learn much at all.**

Deep Supervision

- We can reduce the vanishing gradient problem and make learning more effective via deep supervision.

Deep Supervision

- We can reduce the vanishing gradient problem and make learning more effective via deep supervision.
- Deep supervision refers to a concept of adding learning objectives / loss functions to the intermediate hidden layers.

Deep Supervision

- We can reduce the vanishing gradient problem and make learning more effective via deep supervision.

- Deep supervision refers to a concept of adding learning objectives / loss functions to the intermediate hidden layers.

- Backpropagation proceeds as usual, but now the gradients are propagated not from one but from multiple loss layers.

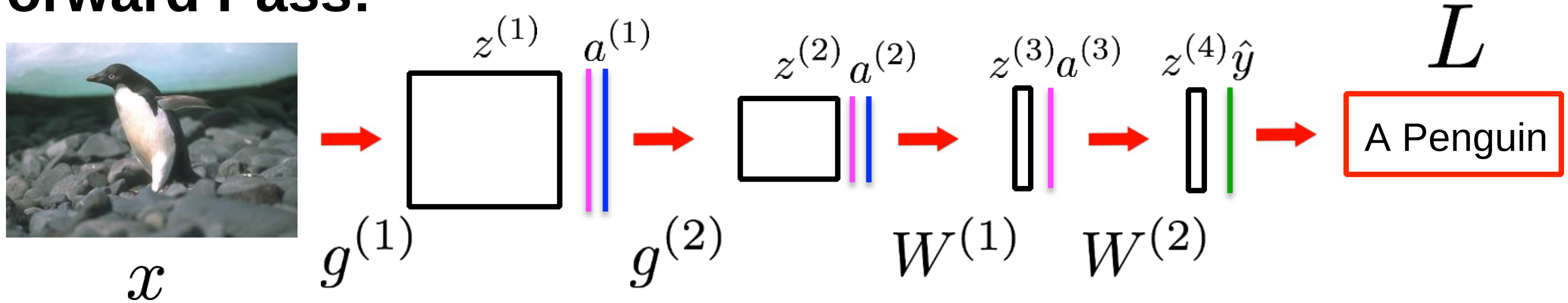
Deep Supervision

Standard CNN:

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



Deep Supervision

Standard CNN:

Notation:

- convolutional layer output

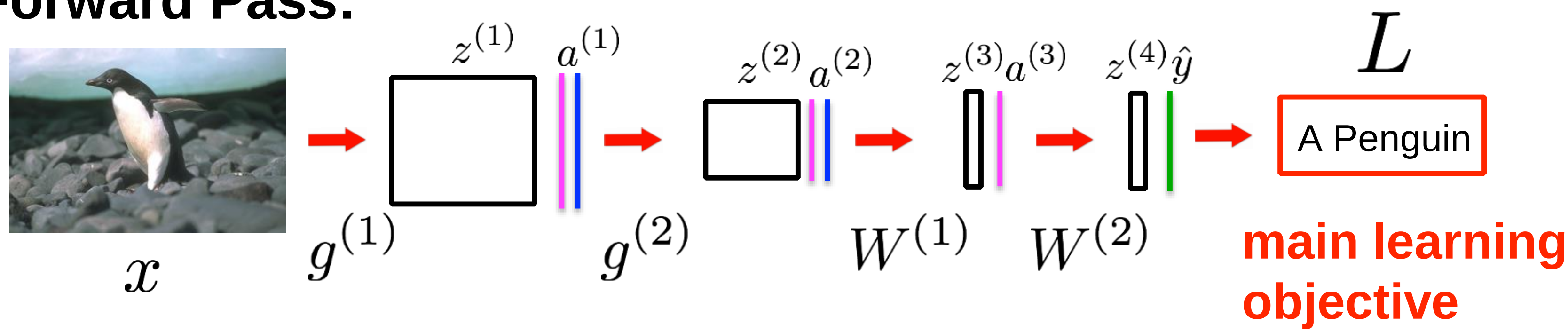
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:



Deep Supervision

Deeply Supervised CNN:

Notation:

- convolutional layer output

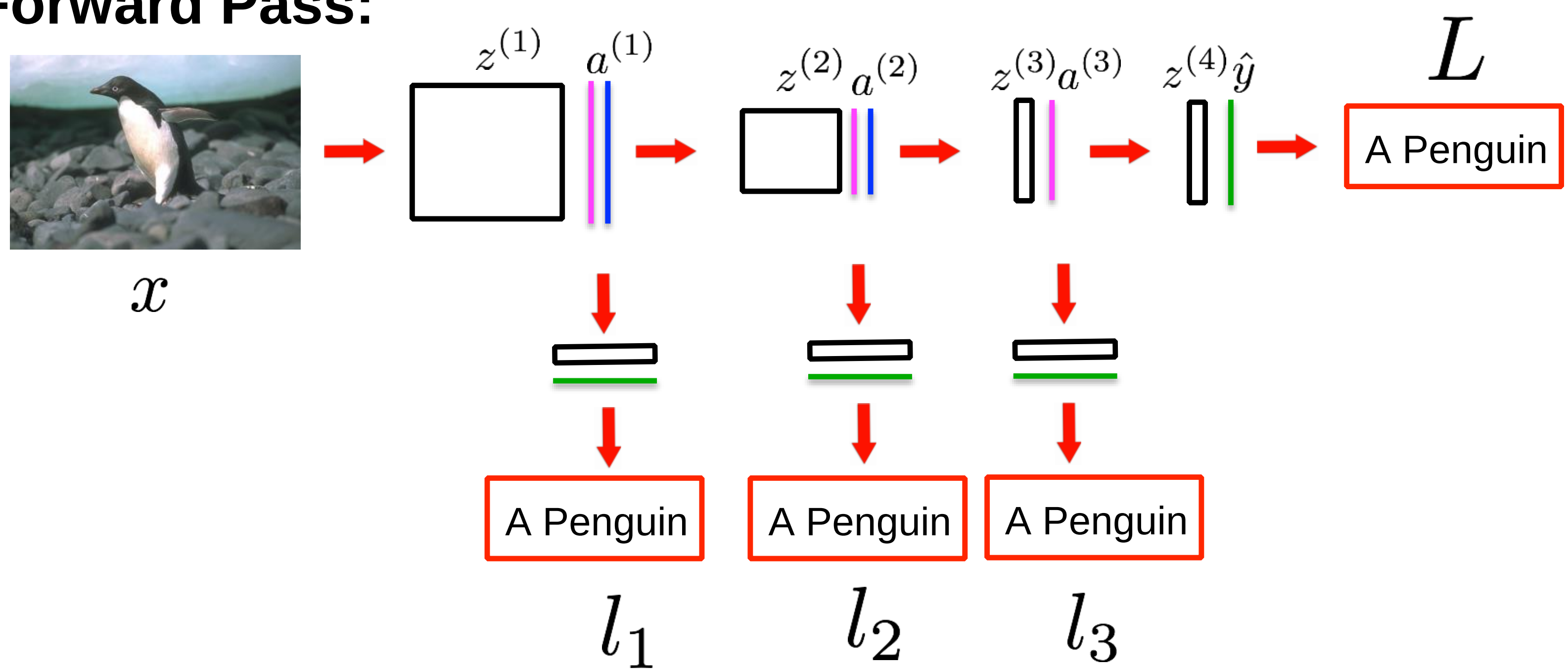
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:



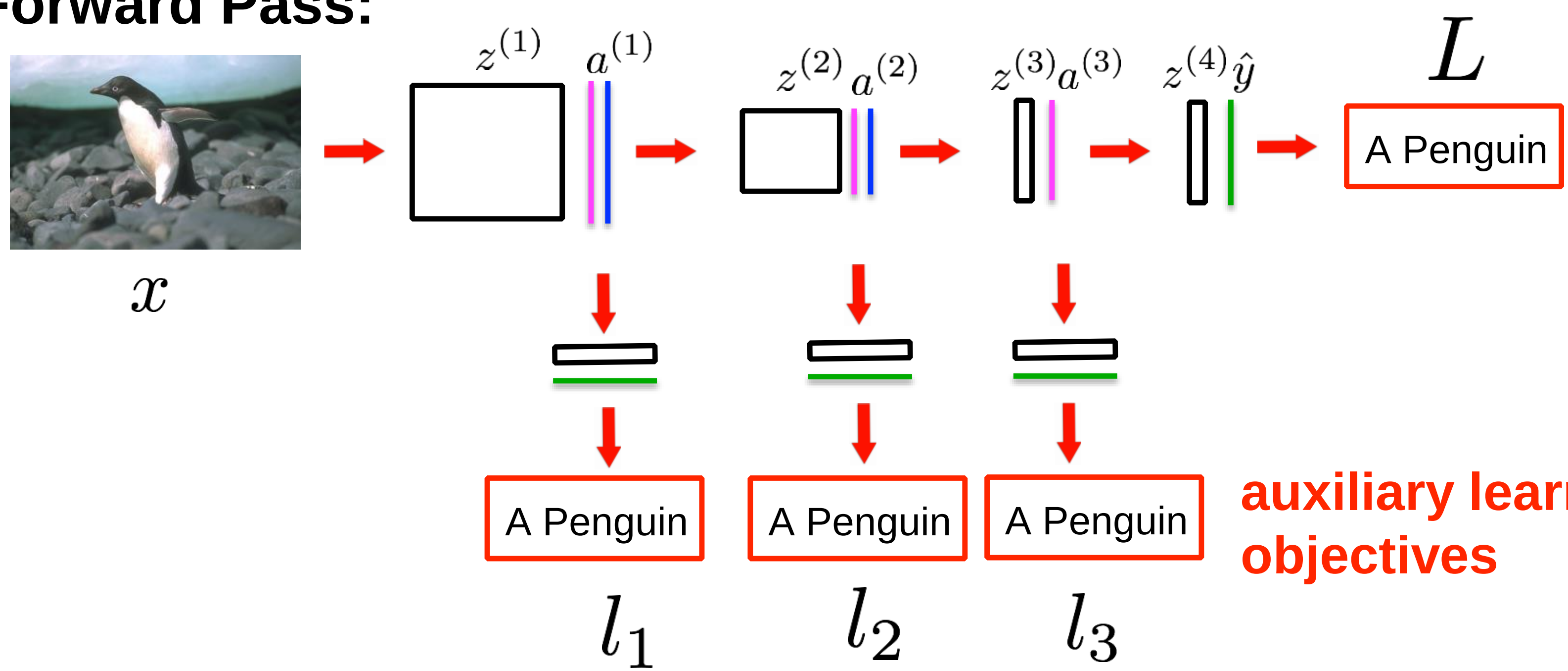
Deep Supervision

<u>Deeply Supervised CNN:</u>

Notation:

- convolutional layer output
- fully connected layer output
- max pooling layer
- sigmoid function f
- softmax function

Forward Pass:



auxiliary learning objectives

Deep Supervision

<u>Deeply Supervised CNN:</u>

Notation:

- convolutional layer output

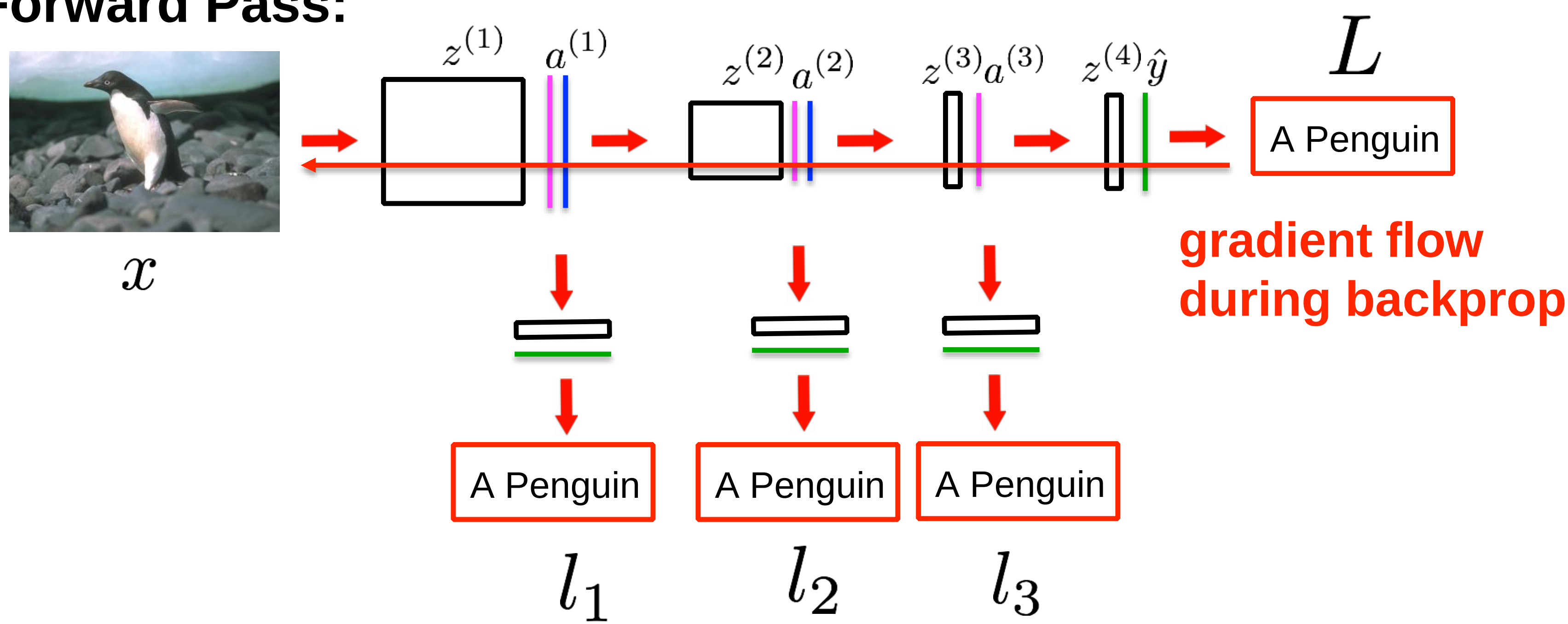
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:



Deep Supervision

Deeply Supervised CNN:

Notation:

- convolutional layer output

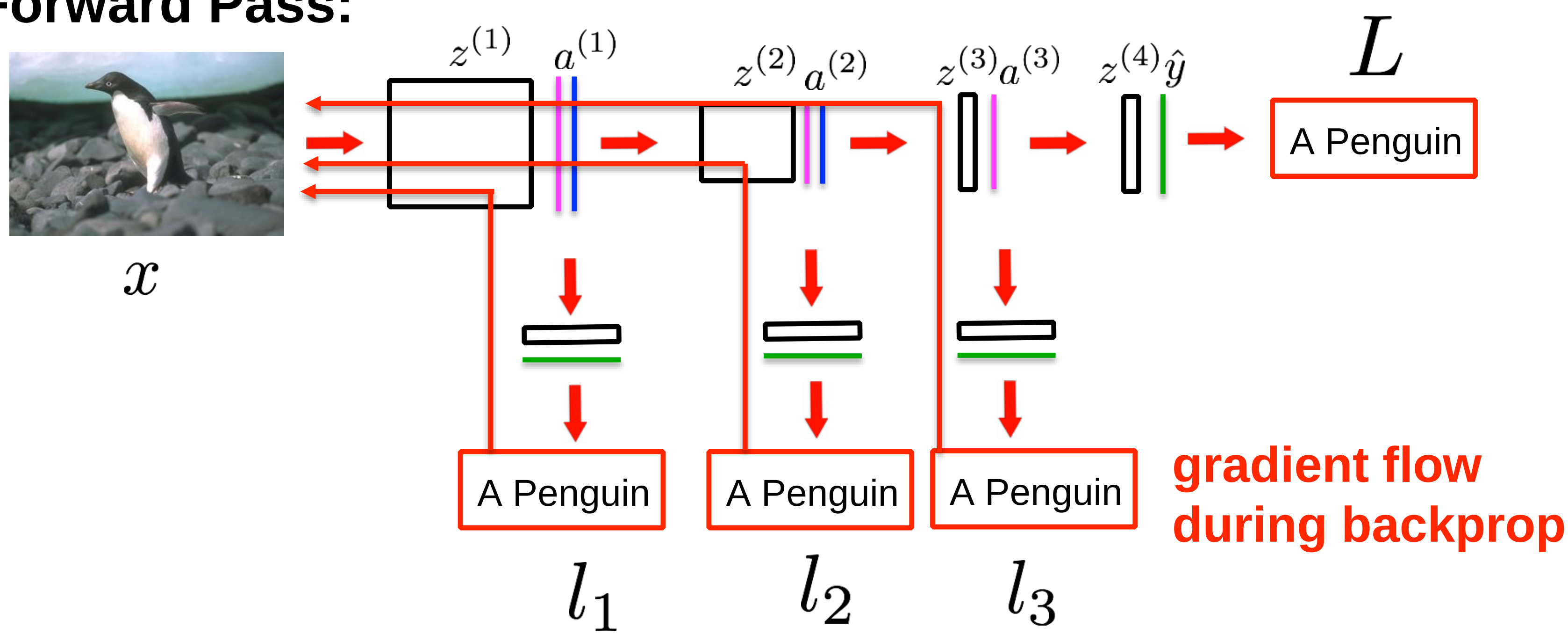
- fully connected layer output

- max pooling layer

- sigmoid function f

- softmax function

Forward Pass:



Deep Supervision

- Assume that we are a given a **labeled** training dataset

$$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$$

- Typically we would employ the following loss function:

$$L = -(y \log f(x) + (1 - y) \log (1 - f(x)))$$

Deep Supervision

- Assume that we are a given a **labeled** training dataset

$$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$$

- Typically we would employ the following loss function:

$$L = -(y \log f(x) + (1 - y) \log (1 - f(x)))$$

- Under deeply supervised networks, we will use:

$$L_{new} = L + \sum_j \alpha_j l_j$$

Deep Supervision

- Assume that we are a given a **labeled** training dataset

$$\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$$

- Typically we would employ the following loss function:

$$L = -(y \log f(x) + (1 - y) \log (1 - f(x)))$$

- Under deeply supervised networks, we will use:

$$L_{new} = L + \boxed{\sum_j \alpha_j l_j}$$

auxiliary learning objectives

Deep Supervision

- Each auxiliary learning objective can be written as:

$$l_j = -(y \log f_j(x) + (1 - y) \log (1 - f_j(x)))$$

- f_j refers to the network's output after a certain hidden layer j.

Deep Supervision

- Each auxiliary learning objective can be written as:

$$l_j = -(y \log f_j(x) + (1 - y) \log (1 - f_j(x)))$$

- f_j refers to the network's output after a certain hidden layer j.
- The differentiation can be done just as it's done with the main learning objective.

Deep Supervision

- Each auxiliary learning objective can be written as:

$$l_j = -(y \log f_j(x) + (1 - y) \log (1 - f_j(x)))$$

- f_j refers to the network's output after a certain hidden layer j.
- The differentiation can be done just as it's done with the main learning objective.
- The gradients from different learning objectives are summed in the hidden layers during the back propagation.

Deep Supervision

- Each auxiliary learning objective can be written as:

$$l_j = -(y \log f_j(x) + (1 - y) \log (1 - f_j(x)))$$

- f_j refers to the network's output after a certain hidden layer j.
- The differentiation can be done just as it's done with the main learning objective
- The gradients from different learning objectives are summed in the hidden layers during the back propagation.
    - **Helps to learn more discriminative features**
    - **Alleviates the vanishing gradient problem**

Deep Supervision

- Standard CNN

```
name: "LeNet"
layers {
  name: "mnist"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "mnist-train-leveldb"
    scale: 0.00390625
    batch_size: 64
  }
}
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "pool1"
  top: "conv2"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 50
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool2"
  top: "ip1"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
```

Deep Supervision

- Standard CNN

```
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool2"
  top: "ip1"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "relu1"
  type: RELU
  bottom: "ip1"
  top: "ip1"
}
layers {
  name: "ip2"
  type: INNER_PRODUCT
  bottom: "ip1"
  top: "ip2"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip2"
  bottom: "label"
}
```

Deep Supervision

- Standard CNN

Layer that produces predictions

```
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool2"
  top: "ip1"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "relu1"
  type: RELU
  bottom: "ip1"
  top: "ip1"
}
layers {
  name: "ip2"
  type: INNER_PRODUCT
  bottom: "ip1"
  top: "ip2"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip2"
  bottom: "label"
}
```

Deep Supervision

- Standard CNN

```
layers {
  name: "ip1"
  type: INNER_PRODUCT
  bottom: "pool2"
  top: "ip1"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "relu1"
  type: RELU
  bottom: "ip1"
  top: "ip1"
}
layers {
  name: "ip2"
  type: INNER_PRODUCT
  bottom: "ip1"
  top: "ip2"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip2"
  bottom: "label"
}
```

A softmax loss layer

Deep Supervision

- Deeply Supervised CNN

```
name: "LeNet"
layers {
  name: "mnist"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "mnist-train-leveldb"
    scale: 0.00390625
    batch_size: 64
  }
}
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
layers {
  name: "ip1_side"
  type: INNER_PRODUCT
  bottom: "pool1"
  top: "ip1_side"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "l1"
  type: SOFTMAX_LOSS
  bottom: "ip1_side"
  bottom: "label"
}
```

Deep Supervision

- Deeply Supervised CNN

```
name: "LeNet"
layers {
  name: "mnist"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "mnist-train-leveldb"
    scale: 0.00390625
    batch_size: 64
  }
}
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

A side layer that produces predictions

```
layers {
  name: "ip1_side"
  type: INNER_PRODUCT
  bottom: "pool1"
  top: "ip1_side"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "l1"
  type: SOFTMAX_LOSS
  bottom: "ip1_side"
  bottom: "label"
}
```

Deep Supervision

- Deeply Supervised CNN

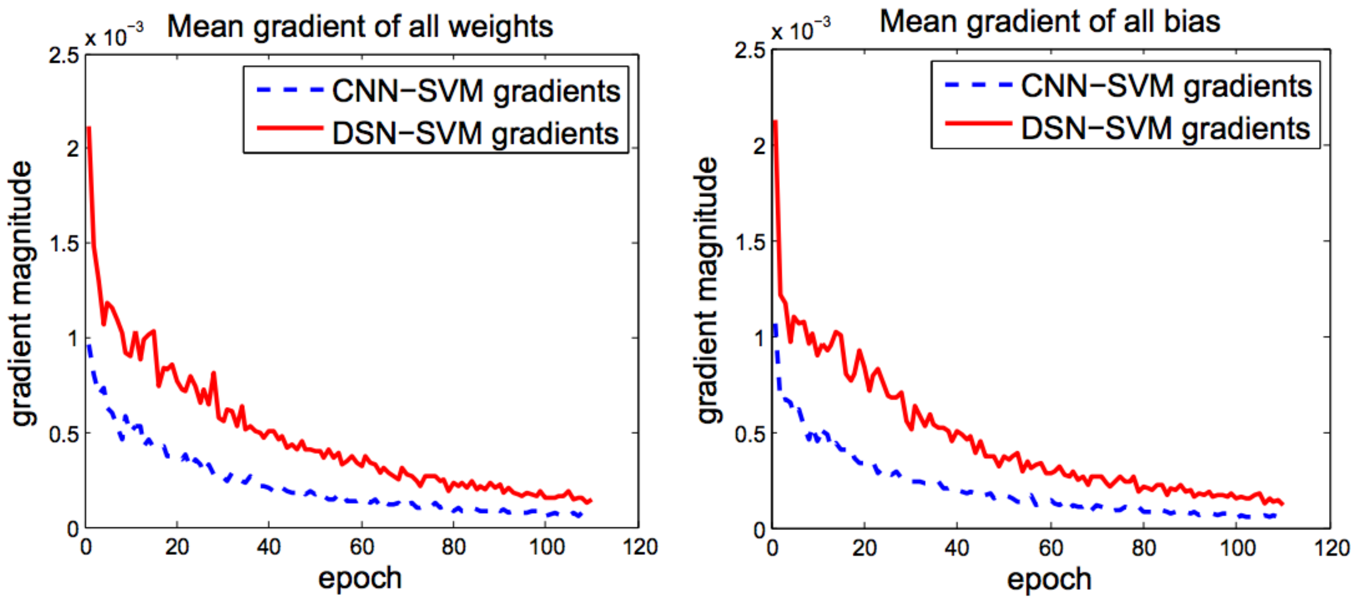
```
name: "LeNet"
layers {
  name: "mnist"
  type: DATA
  top: "data"
  top: "label"
  data_param {
    source: "mnist-train-leveldb"
    scale: 0.00390625
    batch_size: 64
  }
}
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
```

```
layers {
  name: "ip1_side"
  type: INNER_PRODUCT
  bottom: "pool1"
  top: "ip1_side"
  blobs_lr: 1
  blobs_lr: 2
  inner_product_param {
    num_output: 10
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layers {
  name: "l1"
  type: SOFTMAX_LOSS
  bottom: "ip1_side"
  bottom: "label"
}
```

An auxiliary loss function

Deep Supervision

Some interesting results:

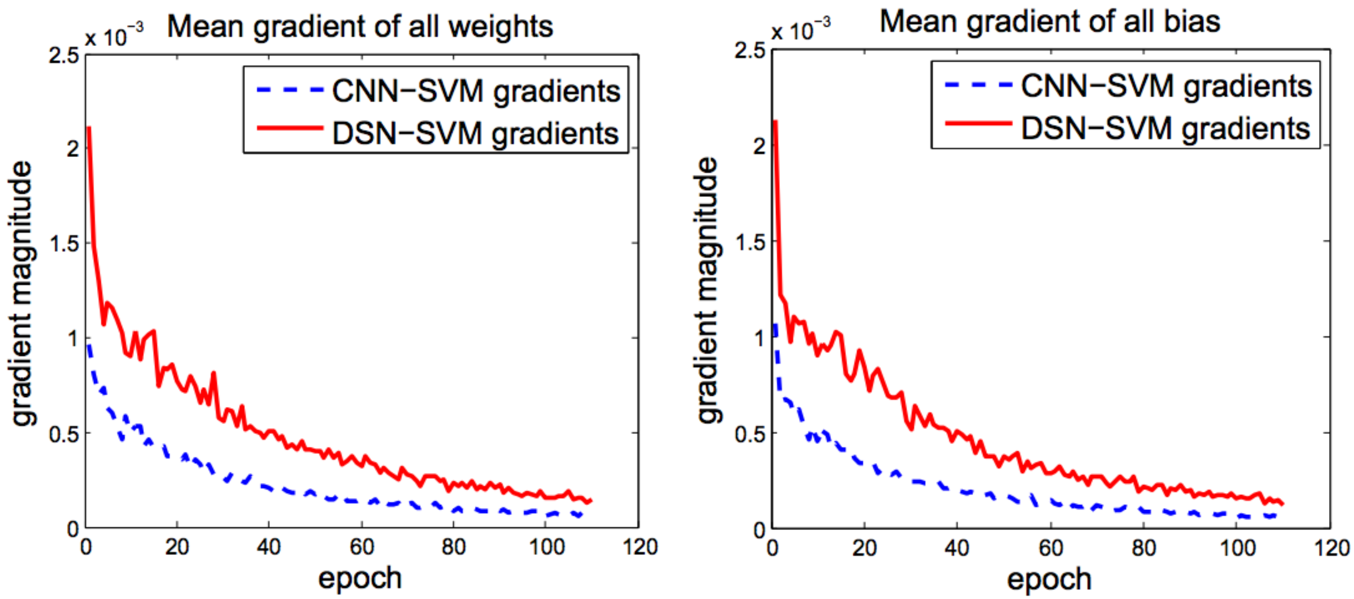


(a) (b)

Figure 3: Average absolute value of gradient matrices during the training on MNIST for (a) weights; (b) biases.

Deep Supervision

Some interesting results:



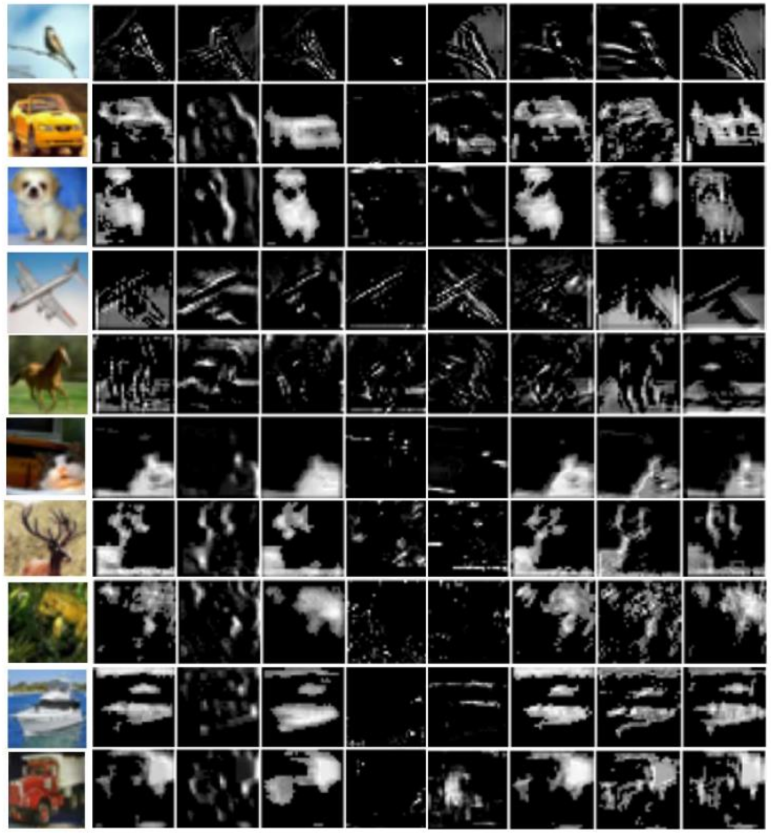
(a) (b)

Figure 3: Average absolute value of gradient matrices during the training on MNIST for (a) weights; (b) biases.

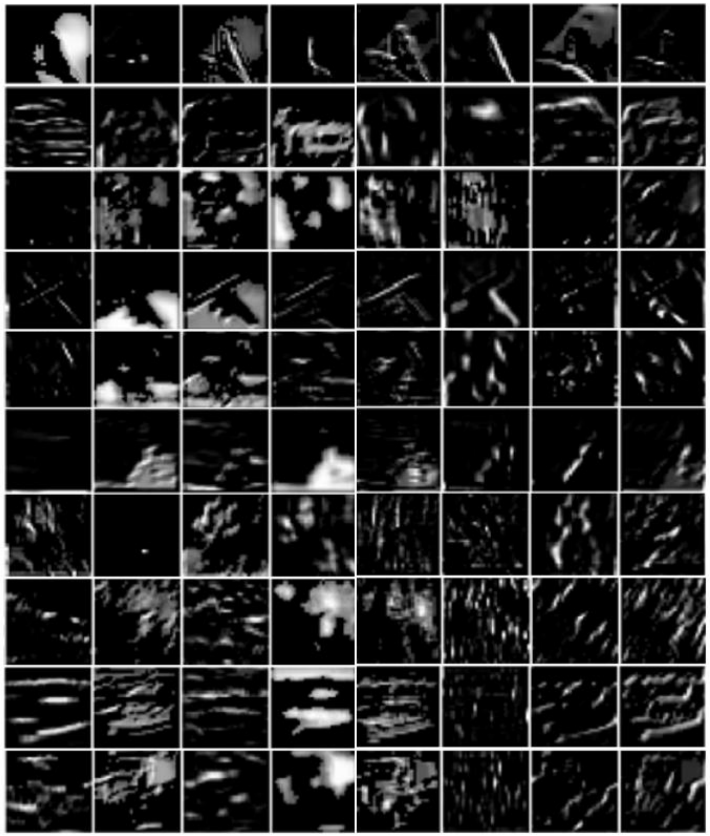
- **Deep supervision helps to reduce the vanishing gradient problem!**

Deep Supervision

Some interesting results:



(a) by DSN

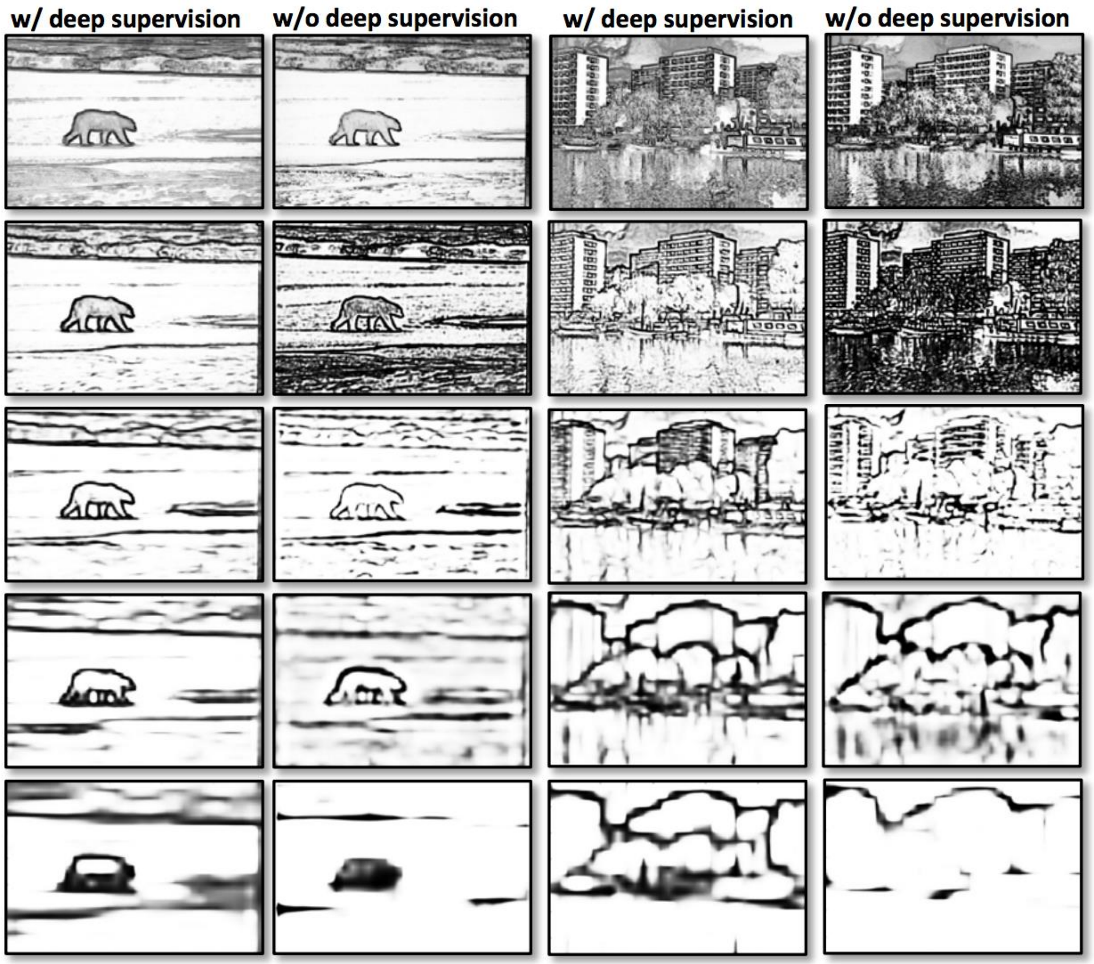


(b) by CNN

- **Learned features are more discriminative**

Deep Supervision

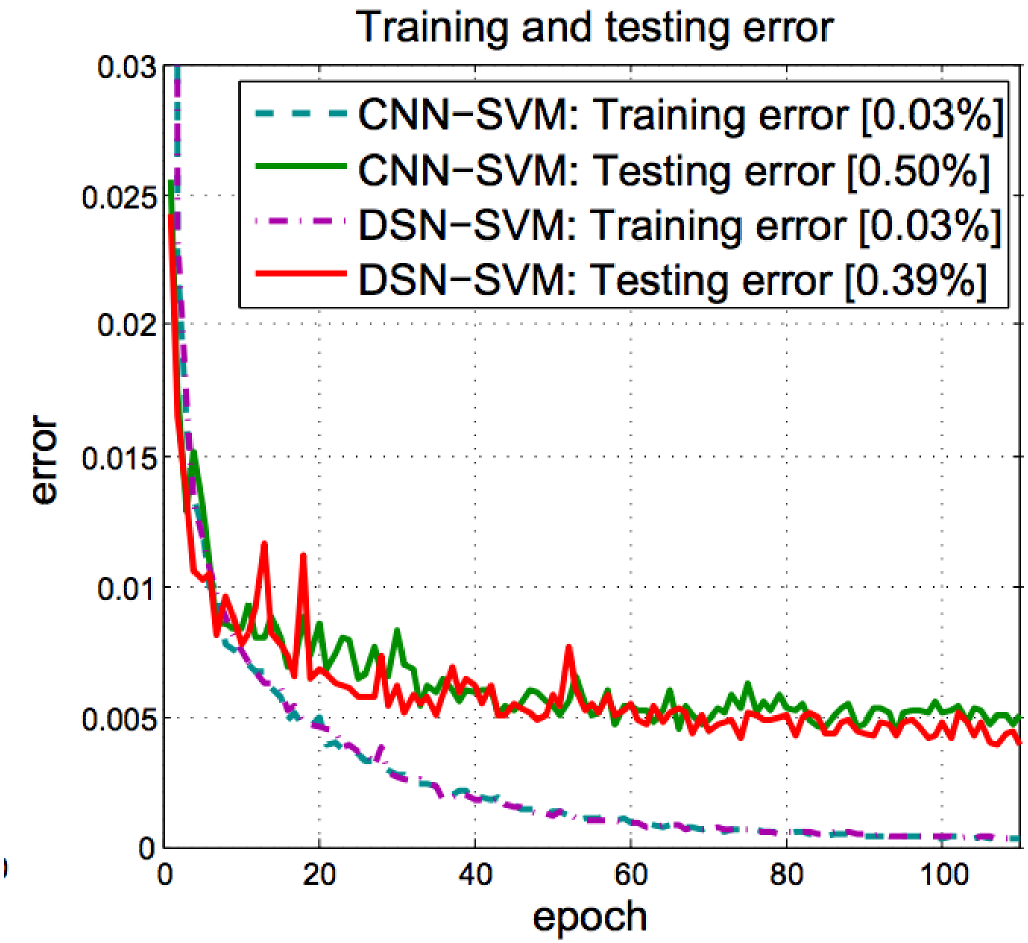
Some interesting results:



- **Learned features are more discriminative**

Deep Supervision

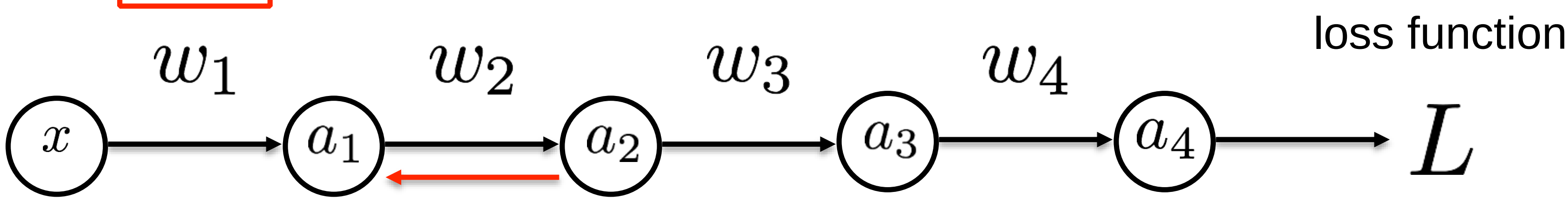
Some interesting results:



- **Deep supervision reduces testing error without overfitting the training data.**

Exploding Gradients

$$\boxed{\frac{\partial L}{\partial w_1}}$$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\sigma'(z_1)x$$

$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)w_4\sigma'(z_3)w_3\sigma'(z_2)w_2\sigma'(z_1)x$$

>|1| >|1| >|1|

Exploding Gradients

$$\boxed{\frac{\partial L}{\partial w_1}}$$

loss function

$$x \xrightarrow{w_1} a_1 \xrightarrow{w_2} a_2 \xrightarrow{w_3} a_3 \xrightarrow{w_4} a_4 \longrightarrow L$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\sigma'(z_1)x$$

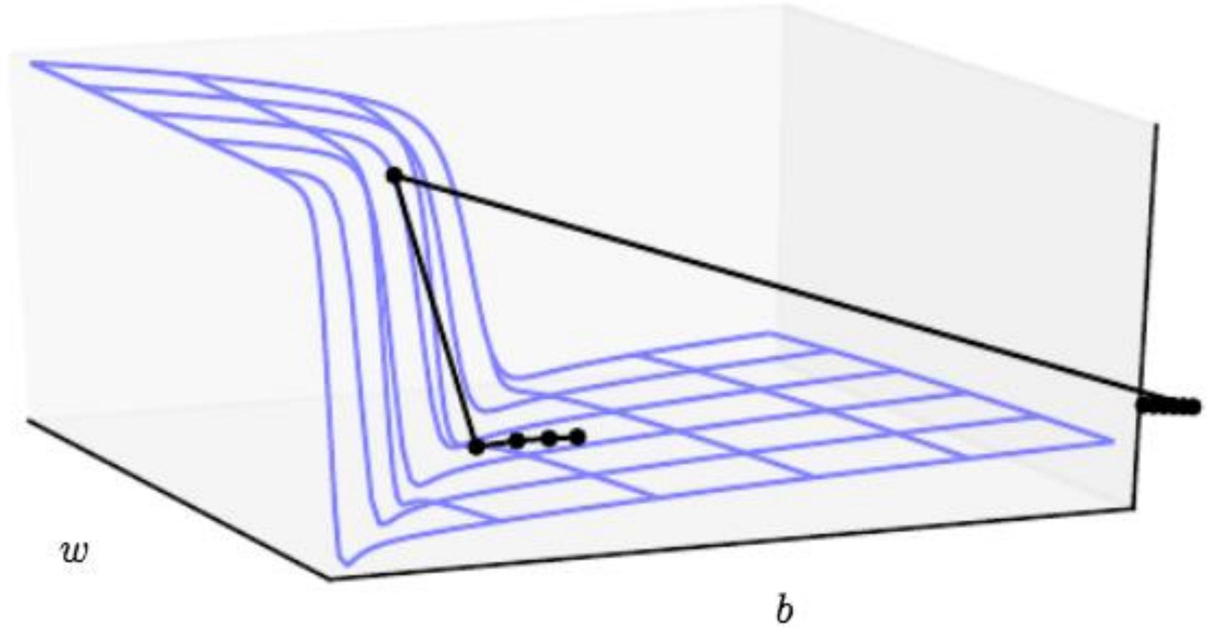
$$= \frac{\partial L}{\partial a_4}\sigma'(z_4)\underbrace{w_4}_{>|1|}\sigma'(z_3)\underbrace{w_3}_{>|1|}\sigma'(z_2)\underbrace{w_2}_{>|1|}\sigma'(z_1)x$$

- **An instance of exploding gradients**

Exploding Gradients

Exploding gradients:

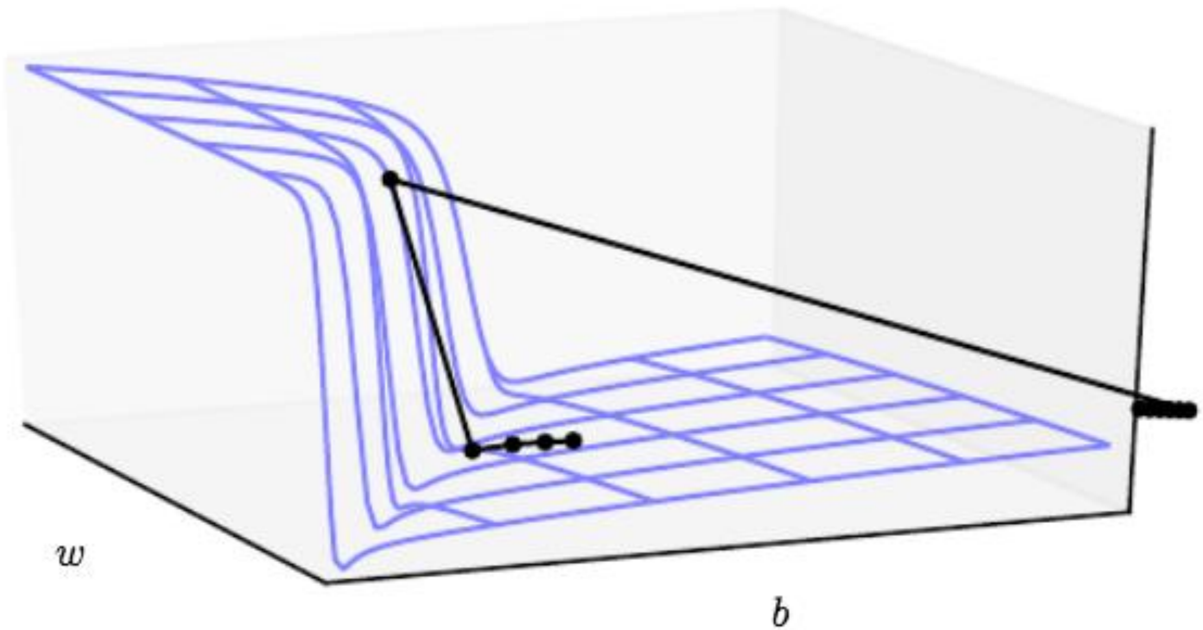
- One of the most often occurring learning problems.
- Due to large jumps parameter update becomes extremely unstable.



Exploding Gradients

Exploding gradients:

- One of the most often occurring learning problems.
- Due to large jumps parameter update becomes extremely unstable.

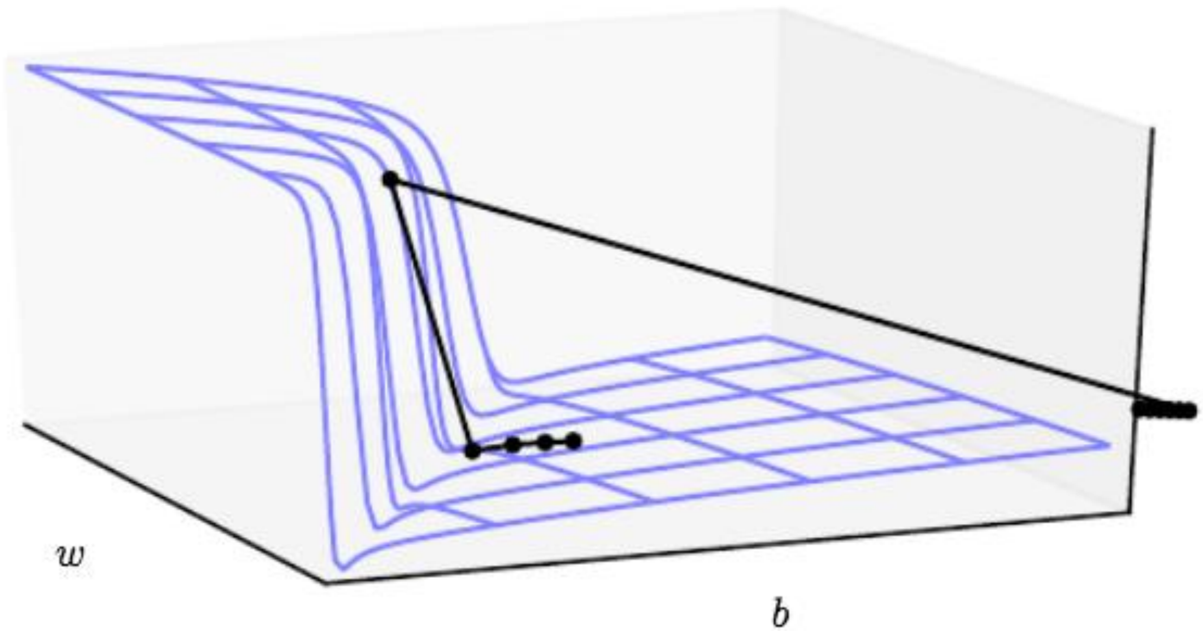


- **Solution #1: reduce the learning rate**

Exploding Gradients

Exploding gradients:

- One of the most often occurring learning problems.
- Due to large jumps parameter update becomes extremely unstable.



- **Solution #1: reduce the learning rate**
- **Solution #2: clip the gradients**

Momentum

Other ways to speed-up the training:

- Even if we address the vanishing gradient problem, the stochastic gradient descent (SGD) optimization is still quite slow.

Momentum

Other ways to speed-up the training:

- Even if we address the vanishing gradient problem, the stochastic gradient descent (SGD) optimization is still quite slow.

- We can accelerate the learning using the momentum method.

Momentum

<u>Other ways to speed-up the training:</u>

- Even if we address the vanishing gradient problem, the stochastic gradient descent (SGD) optimization is still quite slow.

- We can accelerate the learning using the momentum method.

- The momentum method introduces a speed variable, that keeps track of the direction and speed at which the parameters move through the parameter space.

Momentum

<u>Standard gradient descent:</u>

- Learning rule:

$$\theta = \theta - \epsilon \frac{\partial L}{\partial w}$$

Momentum

<u>Standard gradient descent:</u>

- Learning rule:

$$\theta = \theta - \epsilon \frac{\partial L}{\partial w}$$

<u>Gradient descent with momentum:</u>

- Learning rule:

$$v = \alpha v - \epsilon \frac{\partial L}{\partial w}$$

Momentum

Standard gradient descent:

- Learning rule:

$$\theta = \theta - \epsilon \frac{\partial L}{\partial w}$$

Gradient descent with momentum:

- Learning rule:

$$v = \alpha v - \epsilon \frac{\partial L}{\partial w}$$

$$\theta = \theta + v$$

Momentum

Standard gradient descent:

- Learning rule:

$$\theta = \theta - \epsilon \frac{\partial L}{\partial w}$$

Gradient descent with momentum:

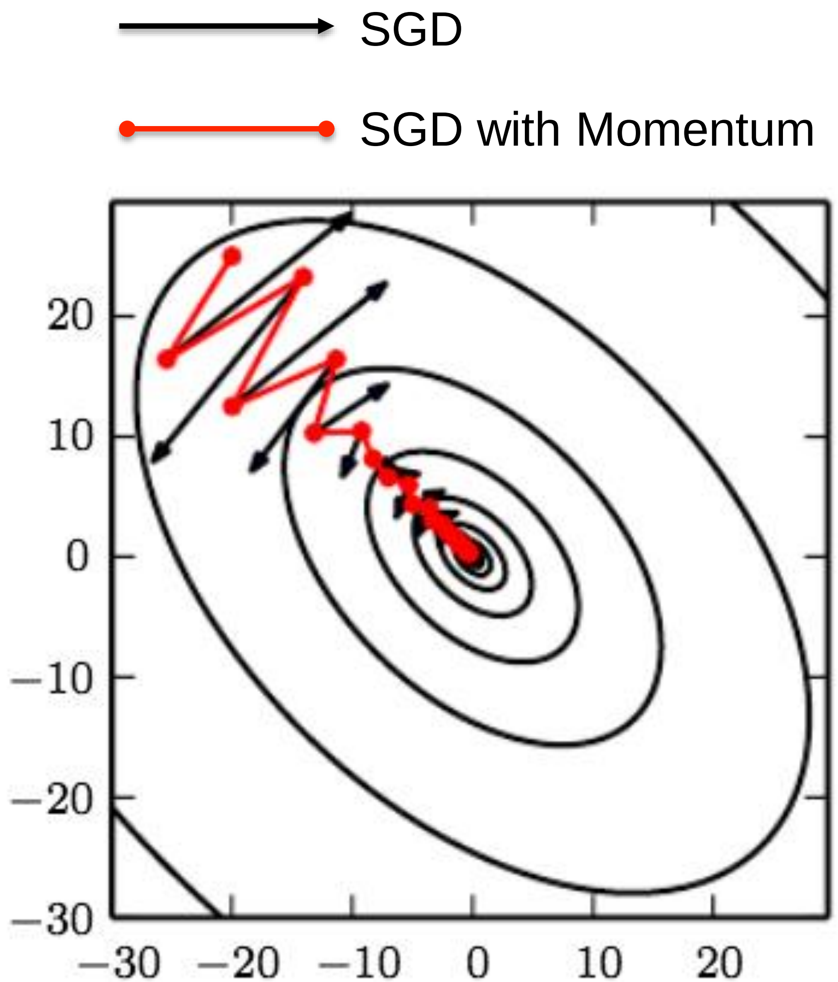
- Learning rule:

$$v = \alpha v - \epsilon \frac{\partial L}{\partial w}$$

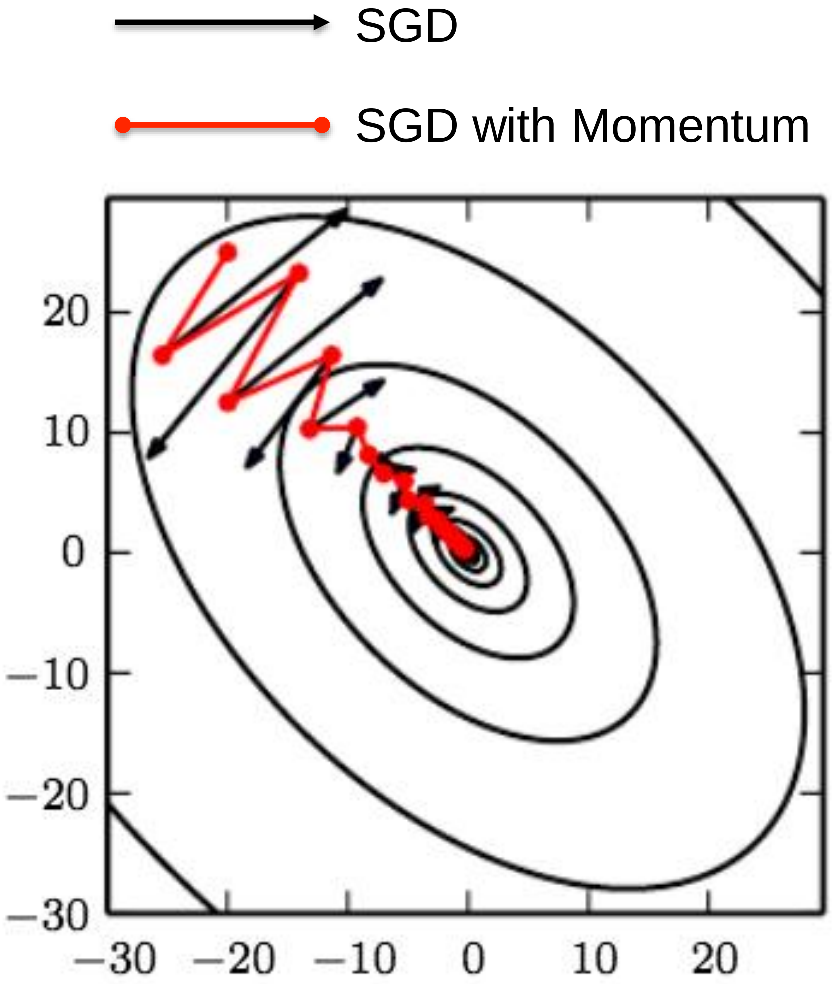
$$\theta = \theta + v$$

  - **Makes it more difficult for the parameters to fluctuate a lot, which makes learning more stable**

Momentum

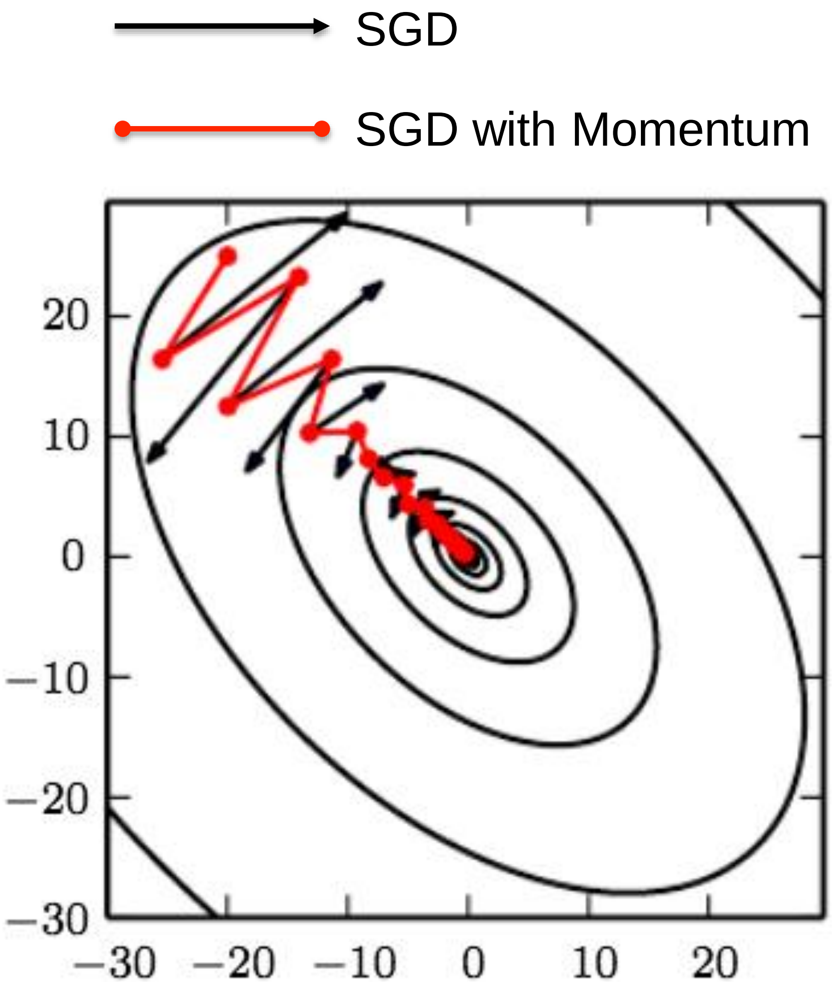


Momentum



- **Momentum helps to achieve more direct path towards local minimum**

Momentum



- **Momentum helps to achieve more direct path towards local minimum**
- **Therefore, learning becomes faster.**

Batch Normalization

Batch Training Mode:

- SGD training is typically done in batch mode (e.g. by randomly selecting N samples from the training dataset, and averaging the gradient across them during backprop).

Batch Normalization

Batch Training Mode:

- SGD training is typically done in batch mode (e.g. by randomly selecting N samples from the training dataset, and averaging the gradient across them during backprop).

Issues:

- Samples in different batches can be very different.
- Small changes to the network parameters amplify as the network becomes deeper
- This changes the internal node distribution in many layers.
- The layers need to continuously adapt to this new internal node distribution, which slows down the training.

Batch Normalization

Batch Normalization:

- In every layer, normalize each feature in the mini-batch to have zero-mean and the variance of 1.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

Batch Normalization

Batch Normalization:

- In every layer, normalize each feature in the mini-batch to have zero-mean and the variance of 1.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

What's wrong with this approach?

Batch Normalization

<u>Batch Normalization:</u>

- In every layer, normalize each feature in the mini-batch to have zero-mean and the variance of 1.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

What happens if we normalize the inputs to the sigmoid function?

Batch Normalization

Batch Normalization:

- In every layer, normalize each feature in the mini-batch to have zero-mean and the variance of 1.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

- **What happens if we normalize the inputs to the sigmoid function?**
- **We may lose representational power (e.g. ability to represent non-linear functions**

Batch Normalization

<u>Batch Normalization:</u>

- In every layer, normalize each feature in the mini-batch to have zero-mean and the variance of 1.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$\boxed{y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)} \qquad \text{// scale and shift}$$

- **Scaling and shifting restores the original representational power.**

Batch Normalization

Batch Normalization:

- In every layer, normalize each feature in the mini-batch to have zero-mean and the variance of 1.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$\boxed{y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)} \qquad \text{// scale and shift}$$

- **Scaling and shifting restores the original representational power.**
- **Two parameters gamma and beta learned during training.**

Batch Normalization

Backpropagation:

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

Backpropagation:

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Which gradients do we need to compute during a backward pass?

Batch Normalization

<u>Backpropagation:</u>

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.

Input: Values of x over a mini-batch: $\mathcal{B} = \boxed{\{x_{1\ldots m}\}}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Need to send backwards.

Batch Normalization

Backpropagation:

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.

Input: Values of x over a mini-batch: $\mathcal{B} = \boxed{\{x_{1\ldots m}\}}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\boxed{\mu_{\mathcal{B}}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\boxed{\sigma_{\mathcal{B}}^2} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\boxed{\widehat{x}_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Intermediate gradients

Batch Normalization

<u>Backpropagation:</u>

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.

Input: Values of x over a mini-batch: $\mathcal{B} = \boxed{\{x_{1\ldots m}\}}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\boxed{\mu_\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\boxed{\sigma_\mathcal{B}^2} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\boxed{\widehat{x}_i} \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \boxed{\gamma}\,\widehat{x}_i + \boxed{\beta} \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Parameter gradients

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$\boxed{y_i \leftarrow \gamma\widehat{x}_i + \beta} \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Intermediate gradients:

$$\boxed{\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\boxed{\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

<u>Intermediate gradients:</u>

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\boxed{\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1 \ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\boxed{\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2} \qquad \text{// mini-batch variance}$$

$$\boxed{\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

<u>Intermediate gradients:</u>

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_\mathcal{B}) \cdot \frac{-1}{2} (\sigma_\mathcal{B}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_\mathcal{B}} = \boxed{\left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \right)} + \boxed{\frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_\mathcal{B})}{m}}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\boxed{\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i} \qquad \text{// mini-batch mean}$$

$$\boxed{\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2} \qquad \text{// mini-batch variance}$$

$$\boxed{\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

<u>**Gradient to send backwards:**</u>

$$\frac{\partial \ell}{\partial x_i} = \boxed{\frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}} + \boxed{\frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} \cdot \frac{2(x_i - \mu_\mathcal{B})}{m}} + \boxed{\frac{\partial \ell}{\partial \mu_\mathcal{B}} \cdot \frac{1}{m}}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$\boxed{y_i \leftarrow \gamma\widehat{x}_i + \beta} \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

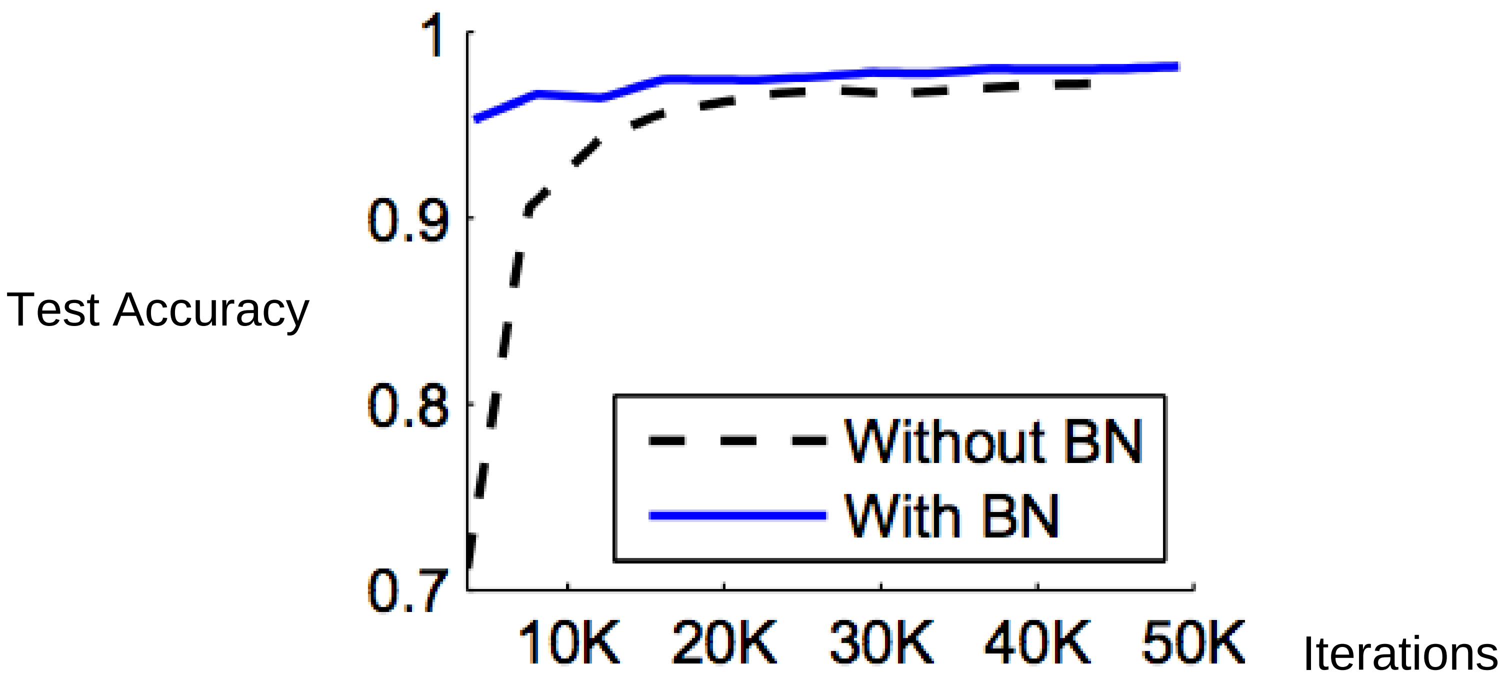
<u>Parameter gradients:</u>

$$\boxed{\begin{aligned} \frac{\partial \ell}{\partial \gamma} &= \sum\nolimits_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i \\ \frac{\partial \ell}{\partial \beta} &= \sum\nolimits_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \end{aligned}}$$

Batch Normalization

Batch Normalization:

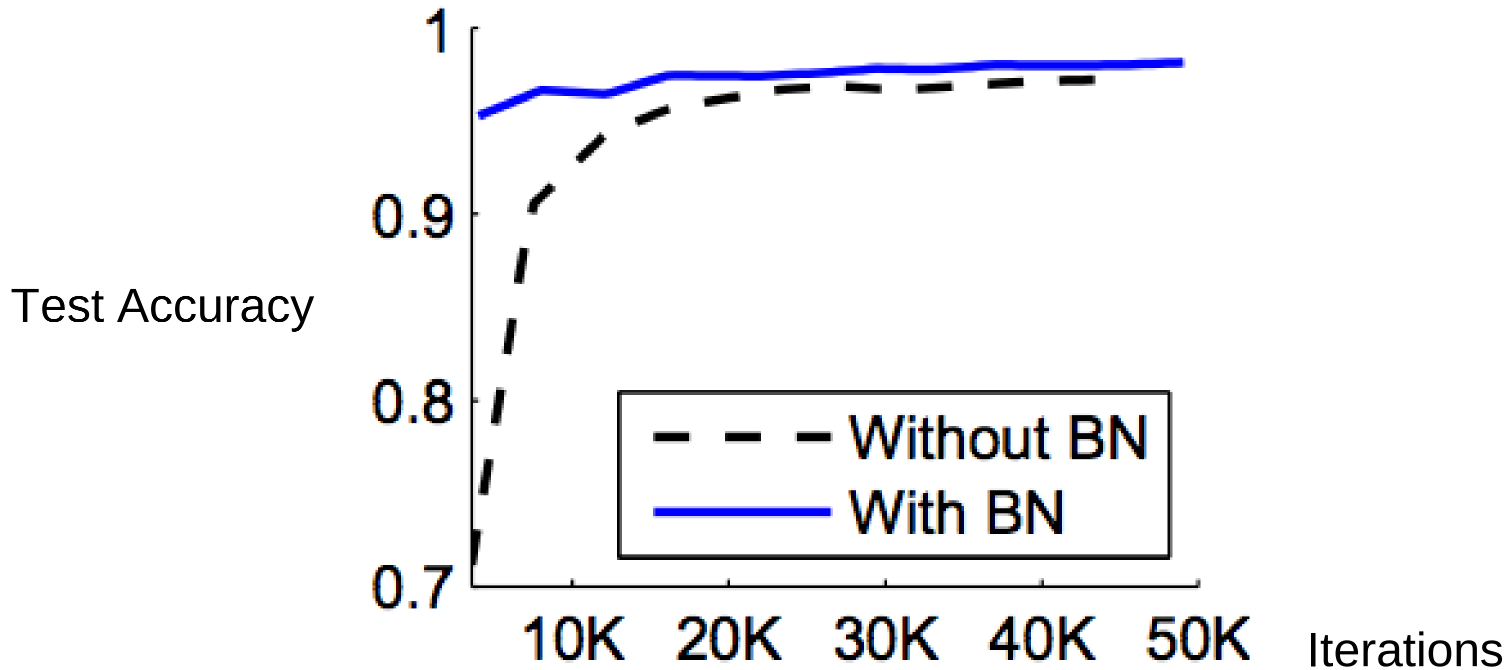
- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.



Batch Normalization

Batch Normalization:

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.



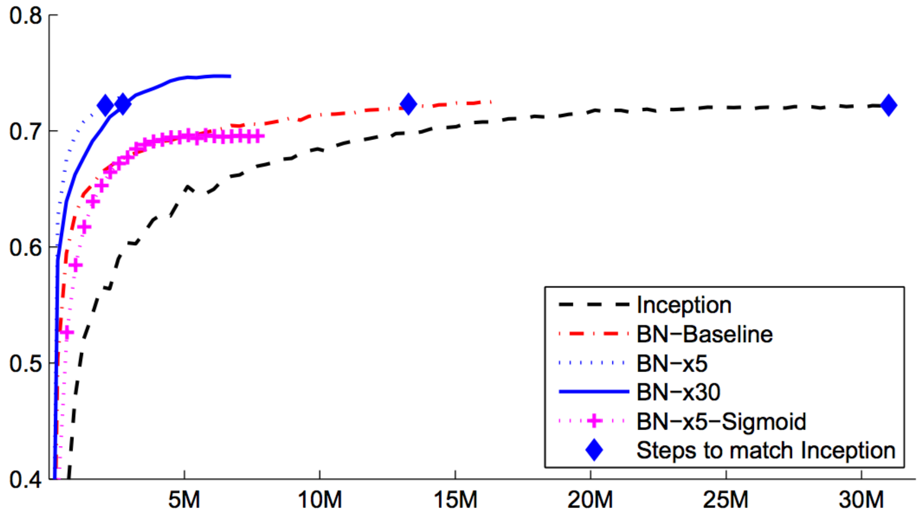
Batch normalization makes training much faster!

Batch Normalization

Batch Normalization:

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.

Test Accuracy



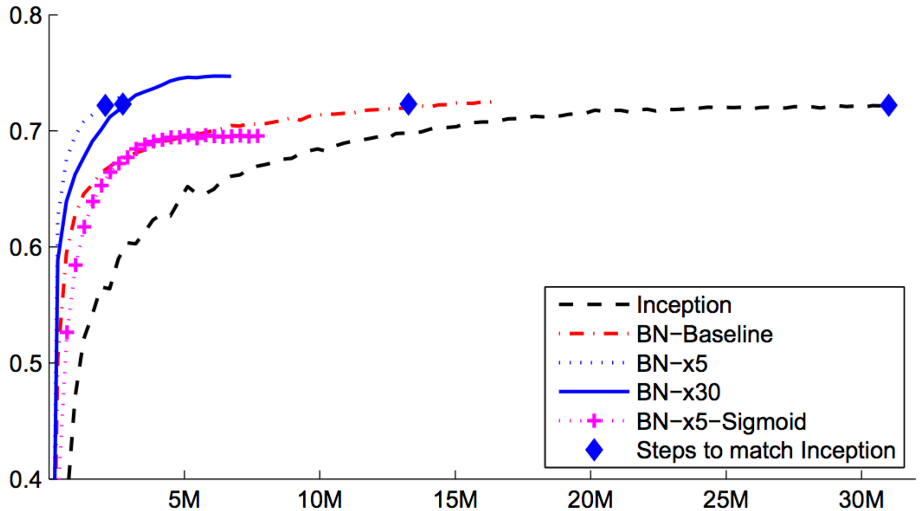
Iterations

Batch Normalization

Batch Normalization:

- Unlike many other normalization schemes, batch normalization can be easily incorporated into backprop.

Test Accuracy



Iterations

Batch normalization makes training much faster!

Summary

Key Take-Aways:

- Specifying the right loss (e.g. cross entropy) is important.
- Picking the right activation function (e.g. RELU) is also imperative.
- We can reduce vanishing gradient problem via deep supervision.
- Reducing the learning rate, and clipping gradients helps to prevent exploding gradient problem.
- Momentum methods allow faster and more stable learning.
- Batch normalization significantly speeds up the training.