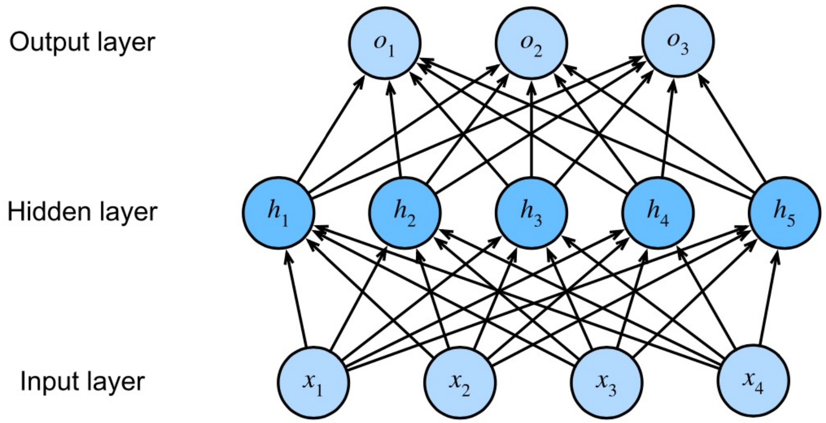
Chapter 4.7 & 4.8: Forward & Backward Prop, Vanishing & Exploding Gradient, Parameter Initialization

Yang Sun

Outline

- Forward & Backward Propagation
- Training Neural Networks
- Numerical Stability and Initialization
  - Vanishing and Exploding Gradients
  - Parameter Initialization

Forward Propagation



$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$$

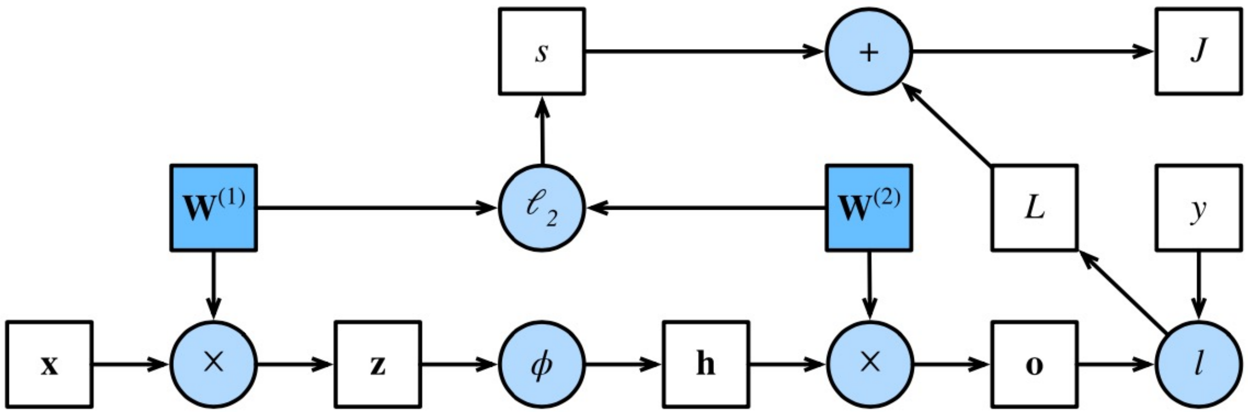
$$\mathbf{h} = \phi(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$$

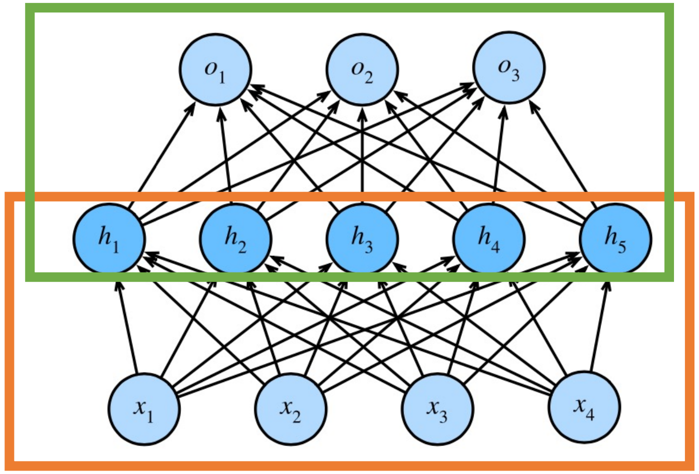
$\mathbf{x} \in \mathbb{R}^d$ $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ $\mathbf{z} \in \mathbb{R}^h$

$\mathbf{h} \in \mathbb{R}^h$ $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$ $\mathbf{o} \in \mathbb{R}^q$

Computational graph



Forward Propagation



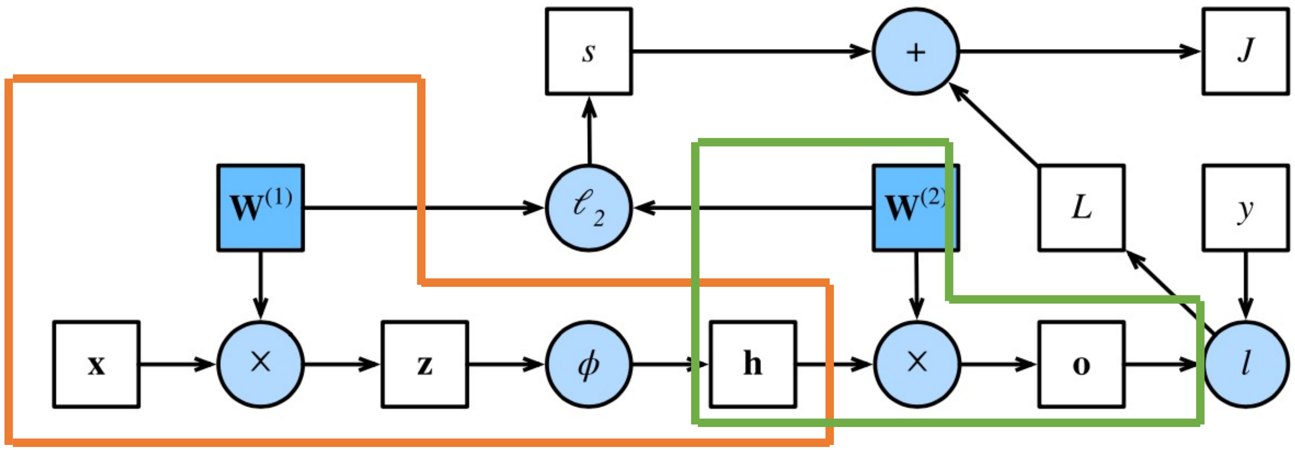
$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$$

$$\mathbf{h} = \phi(\mathbf{z})$$

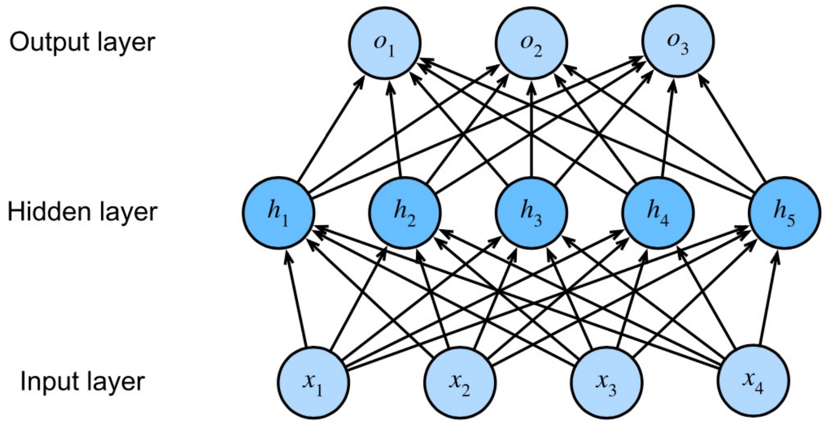
$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$$

$\mathbf{x} \in \mathbb{R}^d$ $\mathbf{W}^{(1)} \in \mathbb{R}^{h\times d}$ $\mathbf{z} \in \mathbb{R}^h$

$\mathbf{h} \in \mathbb{R}^h$ $\mathbf{W}^{(2)} \in \mathbb{R}^{q\times h}$ $\mathbf{o} \in \mathbb{R}^q$



Forward Propagation



$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$$

$$\mathbf{h} = \phi(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$$

$$\mathbf{x} \in \mathbb{R}^{d} \quad \mathbf{W}^{(1)} \in \mathbb{R}^{h\times d} \quad \mathbf{z} \in \mathbb{R}^{h}$$

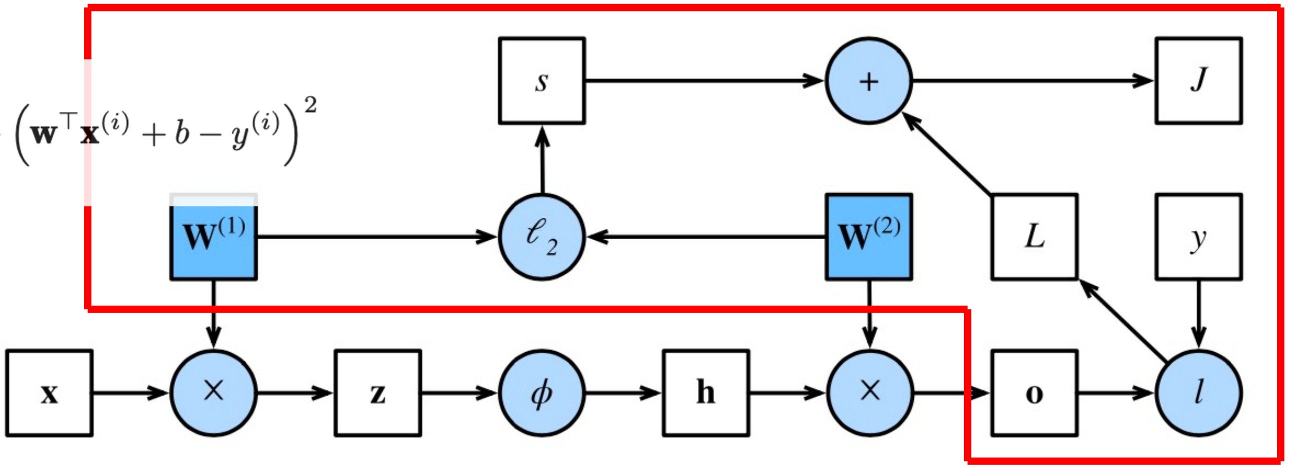
$$\mathbf{h} \in \mathbb{R}^{h} \quad \mathbf{W}^{(2)} \in \mathbb{R}^{q\times h} \quad \mathbf{o} \in \mathbb{R}^{q}$$

Loss Function: $L = l(\mathbf{o}, y) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\left(\mathbf{w}^\top\mathbf{x}^{(i)} + b - y^{(i)}\right)^2$

The Regularization Term:

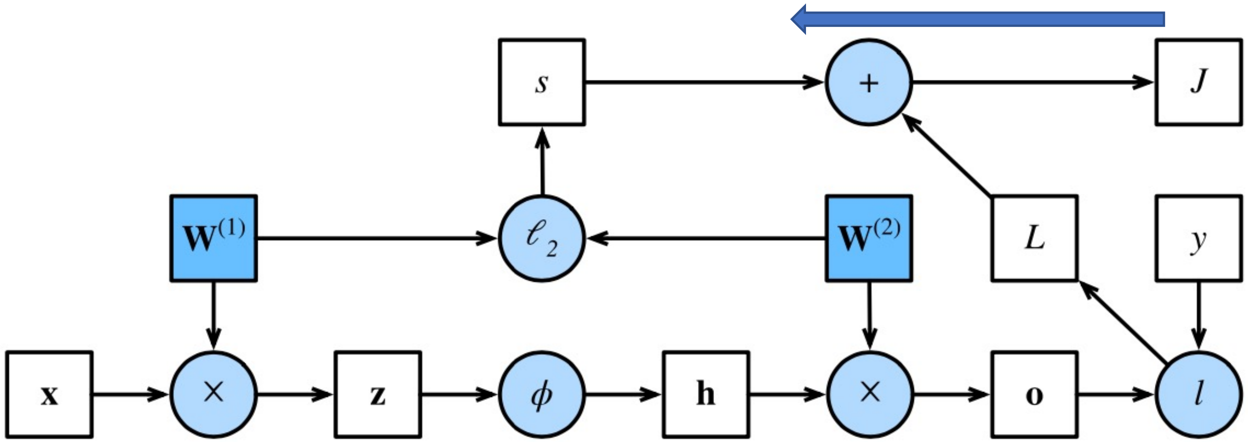
$$s = \frac{\lambda}{2}\left(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2\right)$$

Objective Function $J = L + s$



Backward Propagation

- Calculate the gradient of neural network parameters



$$J = L + s$$

$$L = l(\mathbf{o}, y)$$

$$s = \frac{\lambda}{2}\left(\|\mathbf{W}^{(1)}\|_F^2 + \|\mathbf{W}^{(2)}\|_F^2\right)$$

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$$

$$\mathbf{h} = \phi(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$$

Example:

Calculate gradients based on the chain rule

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \frac{\partial J}{\partial L}\frac{\partial L}{\partial \mathbf{o}}\frac{\partial \mathbf{o}}{\partial \mathbf{W}^{(2)}} + \frac{\partial J}{\partial s}\frac{\partial s}{\partial \mathbf{W}^{(2)}} = \frac{\partial J}{\partial \mathbf{o}}\mathbf{h}^\top + \lambda \mathbf{W}^{(2)}$$

The function of updating parameters

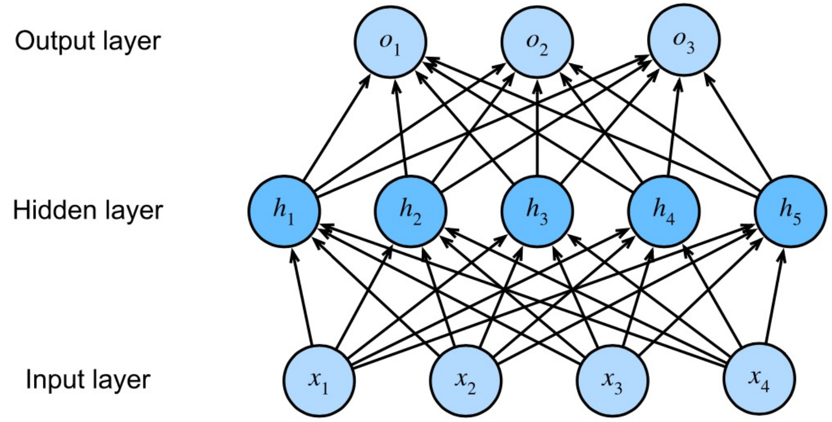
$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \frac{\partial\, L(\mathbf{w}, \mathbf{b})}{\partial(\mathbf{w}, \mathbf{b})}$$

Training Neural Network

$\mathbf{x} \in \mathbb{R}^d$ $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ $\mathbf{z} \in \mathbb{R}^h$

$\mathbf{h} \in \mathbb{R}^h$ $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$ $\mathbf{o} \in \mathbb{R}^q$

Use current parameters ($W^{(1)}$, $W^{(2)}$) to get output and calculate objective function



Calculate the gradients to update parameters

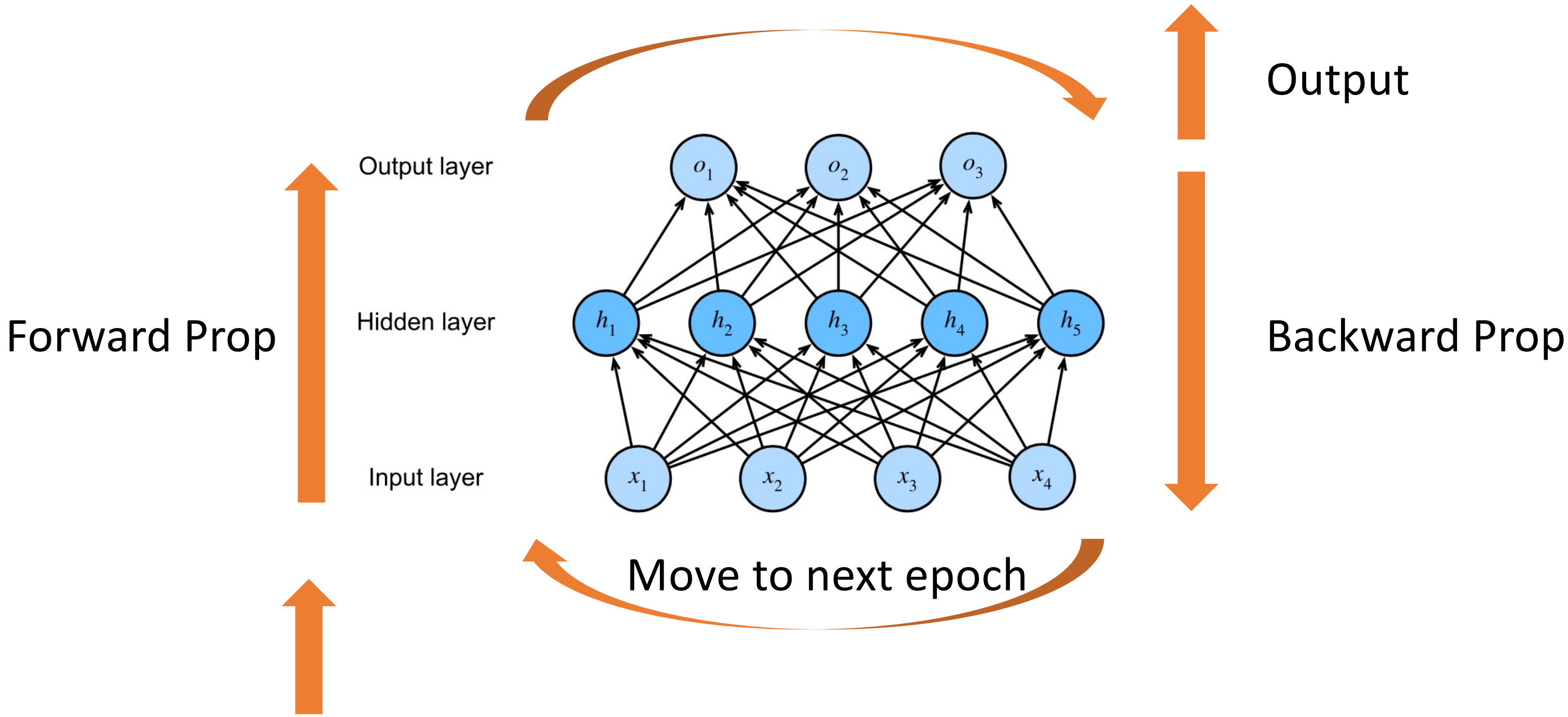
Move to next epoch

During training, all intermediate variables (i.e., **z** and **h**) from forward propagation should be retained until backpropagation is complete, since backpropagation need to reuse them to calculate gradients.

If the **batch size** or **the number of layers** is too large,

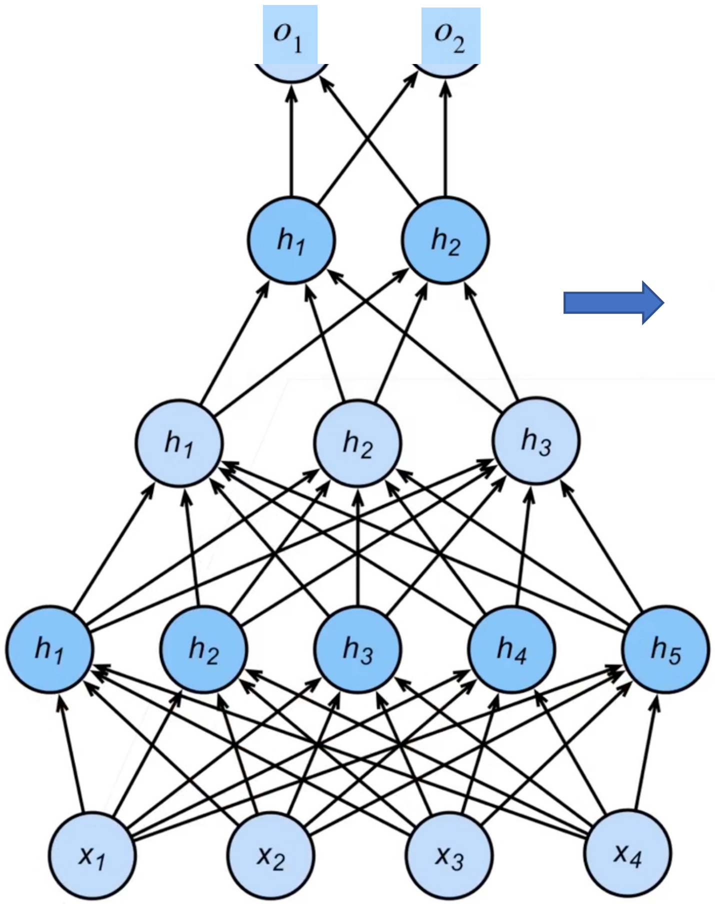
Out of Memory Errors!

Training Neural Network



Input **initial parameters** ($W^{(1)}$, $W^{(2)}$) and training dataset

Numerical Stability and Initialization



Consider a deep network with L layers, input x and output o.

Outputs of l-th layer= f_l(Inputs of l-th layer)

$$\mathbf{h}^{(l)} = f_l(\mathbf{h}^{(l-1)}) \quad \begin{cases} \mathbf{h}^{(l)} = \sigma(\mathbf{z}) \\ \mathbf{z} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} \end{cases}$$

$$\mathbf{o} = f_L \circ \ldots \circ f_1(\mathbf{x}).$$

$$\partial_{\mathbf{W}^{(l)}} \mathbf{o} = \underbrace{\partial_{\mathbf{h}^{(L-1)}} \mathbf{h}^{(L)} \cdot \ldots \cdot \partial_{\mathbf{h}^{(l)}} \mathbf{h}^{(l+1)}}_{\text{L – l matrices}} \underbrace{\partial_{\mathbf{W}^{(l)}} \mathbf{h}^{(l)}}_{\text{gradient vector}}.$$

Numerical Stability and Initialization

- Vanishing and Exploding Gradients

$$\partial_{\mathbf{W}^{(l)}} \mathbf{o} = \partial_{\mathbf{h}^{(L-1)}} \underbrace{\mathbf{h}^{(L)} \cdot \ldots \cdot \partial_{\mathbf{h}^{(l)}}}_{\text{L − l matrices}} \mathbf{h}^{(l+1)} \, \partial_{\mathbf{W}^{(l)}} \mathbf{h}^{(l)}.$$

An extreme case, there are 100 layers in network:

$$0.8^{100} \approx 2\times 10^{-10} \text{ Vanishing}$$

$$1.5^{100} \approx 4\times 10^{17} \quad \text{Exploding}$$

Numerical Stability and Initialization

- Vanishing and Exploding Gradients

$$\partial_{\mathbf{W}^{(l)}}\mathbf{o} = \underbrace{\partial_{\mathbf{h}^{(L-1)}}\mathbf{h}^{(L)} \cdot \ldots \cdot \partial_{\mathbf{h}^{(l)}}\mathbf{h}^{(l+1)}}_{\text{L − l matrices}}\, \partial_{\mathbf{W}^{(l)}}\mathbf{h}^{(l)}.$$

Vanishing

Exploding

- Value will be 0 or inf if it is out of range (for 16-bit floating-point format, the range is 6e-5 to 6e4)

Vanishing:

- Regardless of the learning rate, there is no progress in learning

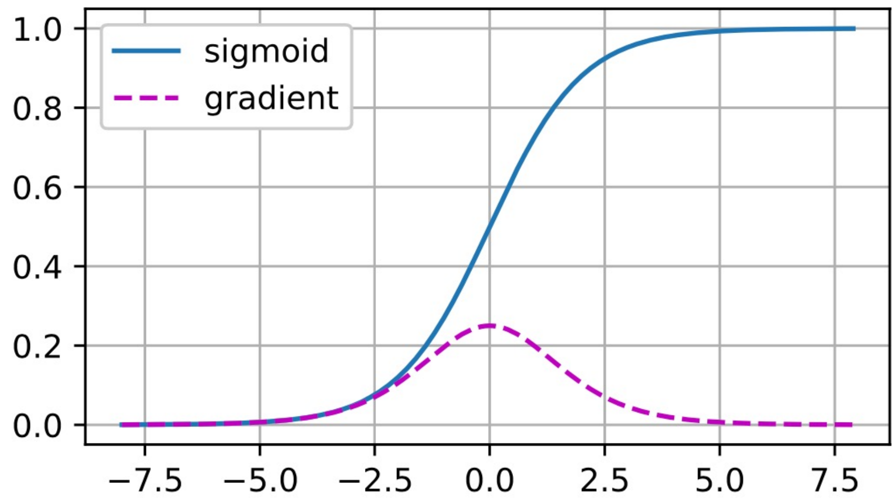
Exploding:

- the learning rate should be very small, otherwise the gradient descent cannot converge.

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \frac{\partial\, L(\mathbf{w}, \mathbf{b})}{\partial(\mathbf{w}, \mathbf{b})}$$

Vanishing Gradients

- Example: Sigmoid Function =1/(1 + exp(−x))



$$\partial_{\mathbf{W}^{(l)}}\mathbf{o} = \boxed{\underbrace{\partial_{\mathbf{h}^{(L-1)}}\mathbf{h}^{(L)} \cdot \ldots \cdot \partial_{\mathbf{h}^{(l)}}\mathbf{h}^{(l+1)}}_{\text{L - l matrices}}} \partial_{\mathbf{W}^{(l)}}\mathbf{h}^{(l)}$$

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t\mathbf{h}^{t-1})$$

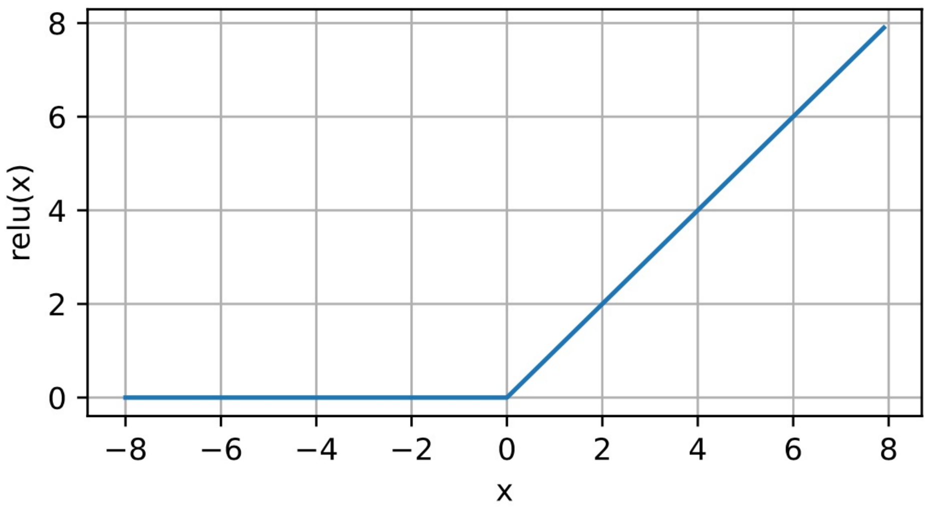
$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t\mathbf{h}^{t-1})\right)(W^t)^T$$

$$\boxed{\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i\mathbf{h}^{i-1})\right)(W^i)^T$$

If $\mathbf{W}^t\mathbf{h}^{t-1}$ is far away from 0, the gradient will be 0 to cause vanishing gradient problem.

Vanishing Gradients

- Example: ReLU = max(x, 0)



$$\partial_{\mathbf{W}^{(l)}}\mathbf{o} = \boxed{\underbrace{\partial_{\mathbf{h}^{(L-1)}}\mathbf{h}^{(L)} \cdot \ldots \cdot \partial_{\mathbf{h}^{(l)}}\mathbf{h}^{(l+1)}}_{\text{L – l matrices}}}\, \partial_{\mathbf{W}^{(l)}}\mathbf{h}^{(l)}$$

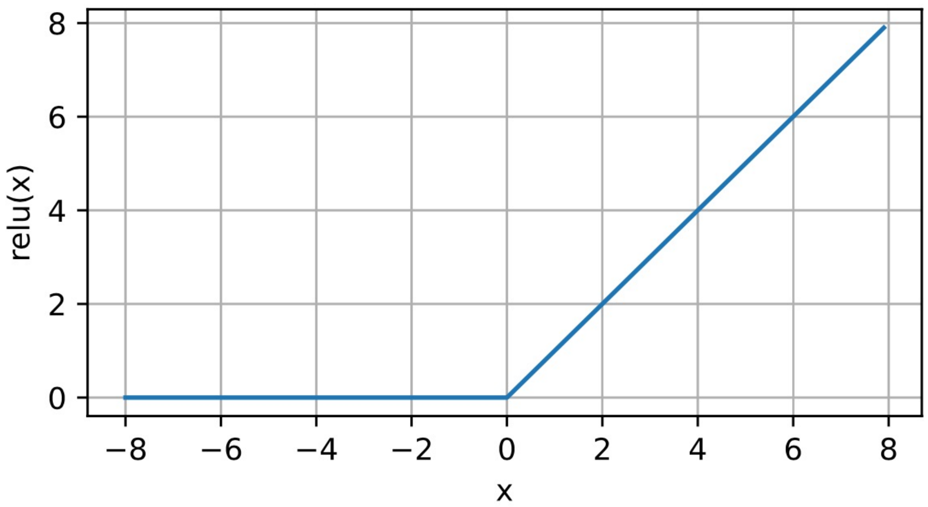
$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t\mathbf{h}^{t-1})$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t\mathbf{h}^{t-1})\right)(W^t)^T$$

$$\boxed{\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i\mathbf{h}^{i-1})\right)(W^i)^T$$

Vanishing Gradients

- Example: ReLU = max(x, 0)



$$\partial_{\mathbf{W}^{(l)}}\mathbf{o} = \underbrace{\boxed{\partial_{\mathbf{h}^{(L-1)}}\mathbf{h}^{(L)} \cdot \ldots \cdot \partial_{\mathbf{h}^{(l)}}\mathbf{h}^{(l+1)}}}_{\text{L - l matrices}} \partial_{\mathbf{W}^{(l)}}\mathbf{h}^{(l)}$$

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t\mathbf{h}^{t-1})$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t\mathbf{h}^{t-1})\right)(W^t)^T$$

$$\boxed{\prod_{i=t}^{d-1}\frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}} = \prod_{i=t}^{d-1}\text{diag}\left(\sigma'(\mathbf{W}^i\mathbf{h}^{i-1})\right)(W^i)^T$$

It solves the vanishing gradient problem. However, some elements will be $\prod_{i=t}^{d-1}(W^i)^T$

If W^i is always larger than 1, it will also explode when the network is very deep!

Exploding Gradients

- Example: Multiply 100 Gaussian Random Matrices

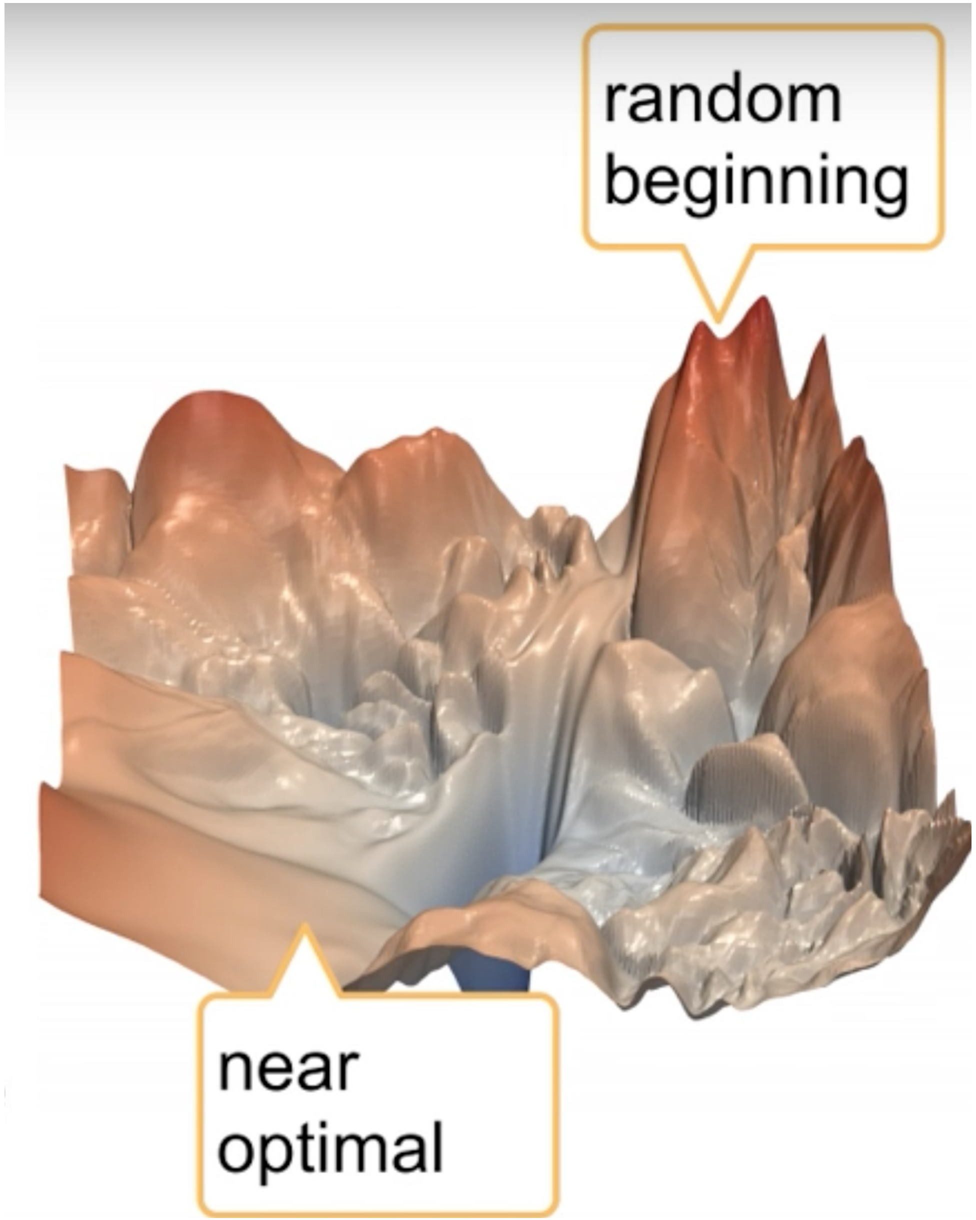
```
M = np.random.normal(size=(4, 4))
print('a single matrix', M)
for i in range(100):
    M = np.dot(M, np.random.normal(size=(4, 4)))

print('after multiplying 100 matrices', M)
```

```
a single matrix [[ 2.2122064   1.1630787   0.7740038   0.4838046 ]
 [ 1.0434405   0.29956347  1.1839255   0.15302546]
 [ 1.8917114  -1.1688148  -1.2347414   1.5580711 ]
 [-1.771029   -0.5459446  -0.45138445 -2.3556297 ]]
after multiplying 100 matrices [[ 3.4459714e+23 -7.8040680e+23  5.9973287e+23  4.5229990e+23]
 [ 2.5275089e+23 -5.7240326e+23  4.3988473e+23  3.3174740e+23]
 [ 1.3731286e+24 -3.1097155e+24  2.3897773e+24  1.8022959e+24]
 [-4.4951040e+23  1.0180033e+24 -7.8232281e+23 -5.9000354e+23]]
```

Parameter Initialization

Careful initialization helps us set parameters in a proper range

- Parameter surface far away from the optimal maybe very complex
- Near the optimal maybe very flatter that we can not train efficiently.



Parameter Initialization

- Can we set all parameters with 0 or a constant initially?

Parameter Initialization

- Can we set all parameter with 0 or a constant initially?

No!

If set as 0, all gradients when doing back propagation will be 0. Therefore, parameters cannot be updated.

$$\partial_{\mathbf{W}^{(l)}}\mathbf{o} = \boxed{\underbrace{\partial_{\mathbf{h}^{(L-1)}}\mathbf{h}^{(L)} \cdot \ldots \cdot \partial_{\mathbf{h}^{(l)}}\mathbf{h}^{(l+1)}}_{\text{L – l matrices}}} \partial_{\mathbf{W}^{(l)}}\mathbf{h}^{(l)}$$

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t\mathbf{h}^{t-1})$$

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t\mathbf{h}^{t-1})\right)(W^t)^T$$

$$\boxed{\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i}} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i\mathbf{h}^{i-1})\right)(W^i)^T$$

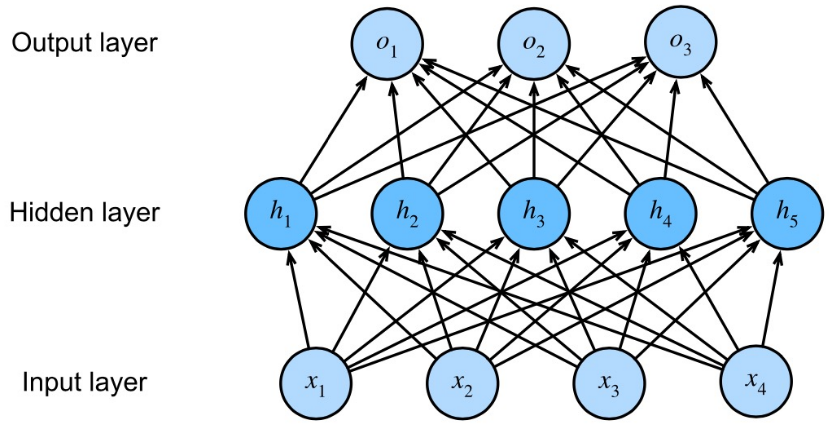
$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \frac{\partial\, L(\mathbf{w}, \mathbf{b})}{\partial(\mathbf{w}, \mathbf{b})}$$

Parameter Initialization

- Can we set all parameter with 0 or a constant initially?

No!

If set as a constant, the symmetry cannot be broken. The hidden layer would behave as if it had only a single unit.



$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$$

$$\mathbf{h} = \phi(\mathbf{z})$$

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$$

Parameter Initialization

- Default Initialization

Use **a normal distribution $N(0, \sigma^2)$** to initialize the values of the weights

It performs well in practice for moderate problem sizes.

Parameter Initialization

- Xavier Initialization

named after the first author of its creators (Glorot & Bengio, 2010).

Main idea: Keep variances of outputs (forward prop) and gradients (backward prop) of all layers as a constant.

$$\forall i, t \qquad \mathbb{E}[h_i^t] = 0 \qquad \mathrm{Var}[h_i^t] = a \qquad\qquad \mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \quad \mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b$$

Parameter Initialization

- Xavier Initialization

Assume the active function is linear function $$o_i = \sum_{j=1}^{n_{\text{in}}} w_{ij} x_j.$$

w_{ij} are all drawn independently from the same distribution with variance σ^2

$$\mathbb{E}[h_i^t] = 0 \qquad \text{Var}[h_i^t] = a \quad \Rightarrow \quad n_{\text{in}}\sigma^2 = 1$$

$$\Rightarrow \quad n_{\text{in}} = n_{\text{out}}$$

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \qquad \text{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b \quad \Rightarrow \quad n_{\text{out}}\sigma^2 = 1$$

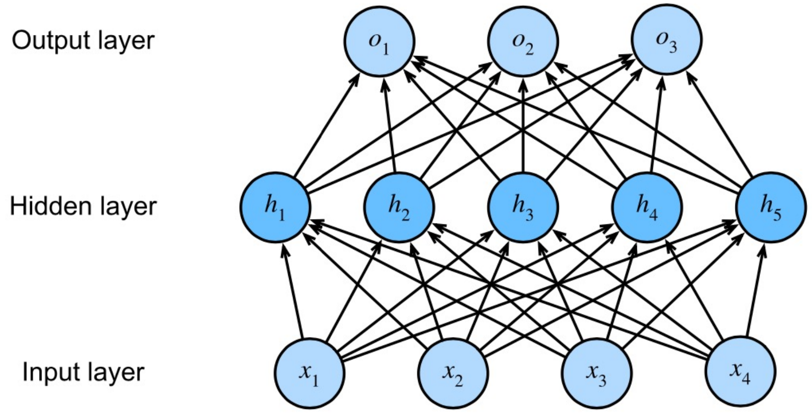
Parameter Initialization

- Xavier Initialization

$$\mathbb{E}[h_i^t] = 0 \qquad \mathrm{Var}[h_i^t] = a \quad \Rightarrow \quad n_{\mathrm{in}}\sigma^2 = 1$$

$$\Rightarrow \quad n_{\mathrm{in}} = n_{\mathrm{out}}$$

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \qquad \mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b \quad \Rightarrow \quad n_{\mathrm{out}}\sigma^2 = 1$$



we cannot possibly satisfy both conditions simultaneously…

Parameter Initialization

- Xavier Initialization

$$n_{\text{in}}\sigma^2 = 1$$

$$n_{\text{out}}\sigma^2 = 1$$

$$\Rightarrow \quad \frac{1}{2}(n_{\text{in}} + n_{\text{out}})\sigma^2 = 1 \textbf{ or equivalently } \sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}$$

Two choices:

- A Gaussian distribution with zero mean and variance $\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}$
- A Uniform distribution: $U\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right)$

  the uniform distribution U (−a, a) has variance $\frac{a^2}{3}$

Summary

- Forward propagation and Back propagation work together to update parameters in the gradient descent optimizer.
- Since we need to store intermediate variables and gradients during training, memory we have limits the batch size of the training datasets we can use and the depth of the network.
- Parameter Initialization is very important to keep numerical stability and training efficiency. Xavier is a good way for initialization in practice.

Reference: D2L online courses

https://www.youtube.com/watch?v=OWQNTURBdxw&list=PLZSO_6-bSqHQHBCoGaObUljoXAyyqhpFW&index=37