PyTorch

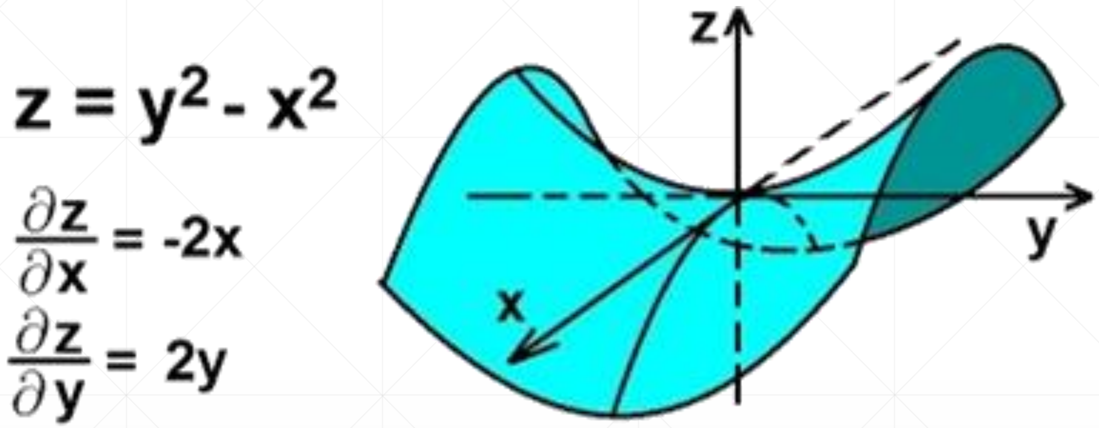
什么是梯度

主讲人：龙良曲

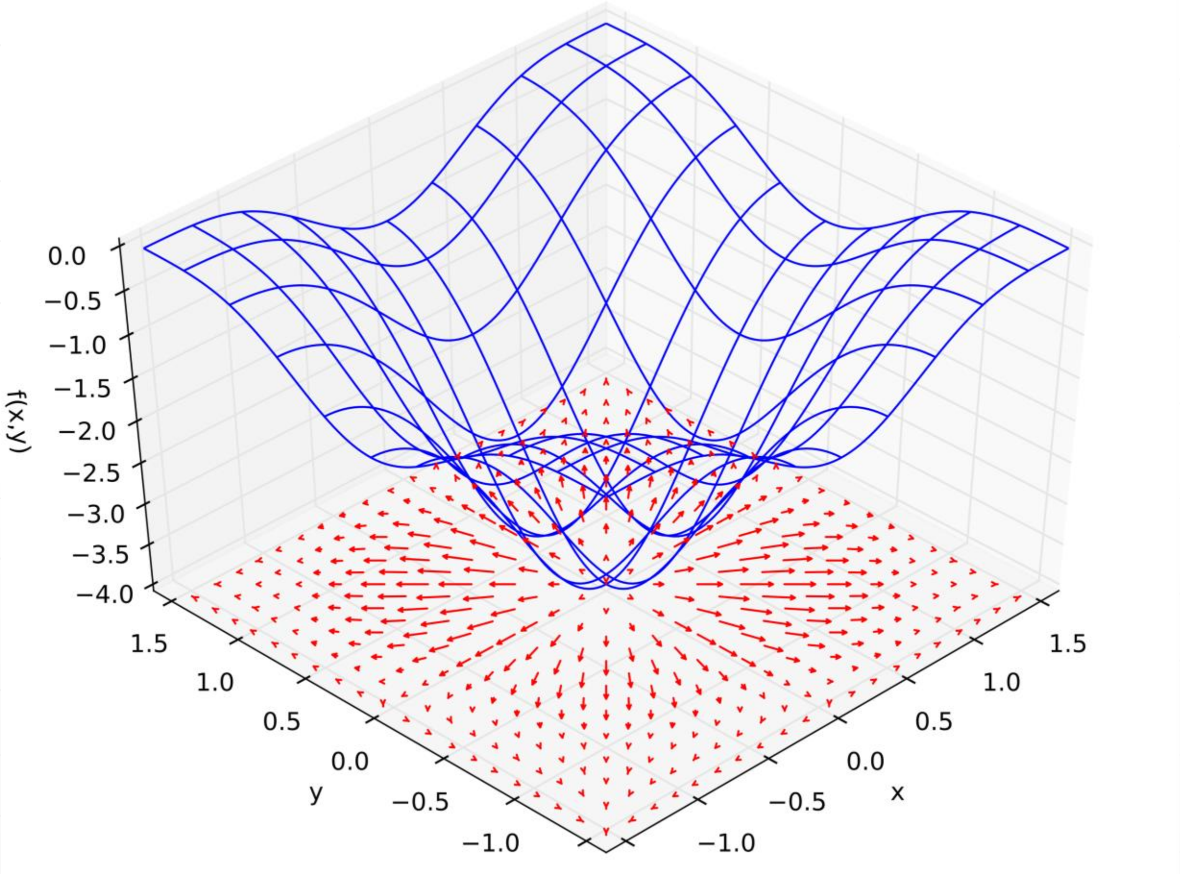
Clarification

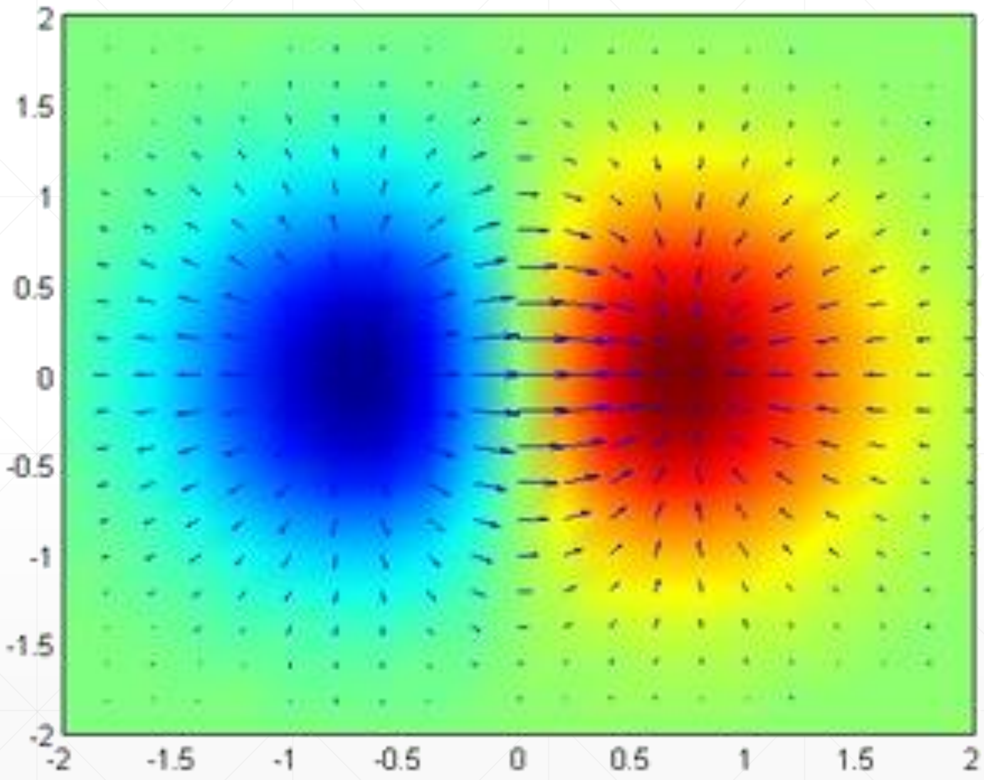
- 导数, derivate
- 偏微分, partial derivate
- 梯度, gradient

$$\nabla f = \left(\frac{\partial f}{\partial x_1}; \frac{\partial f}{\partial x_2}; \ldots; \frac{\partial f}{\partial x_n} \right)$$



What does grad mean?





How to search for minima?

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t) \,.$$

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

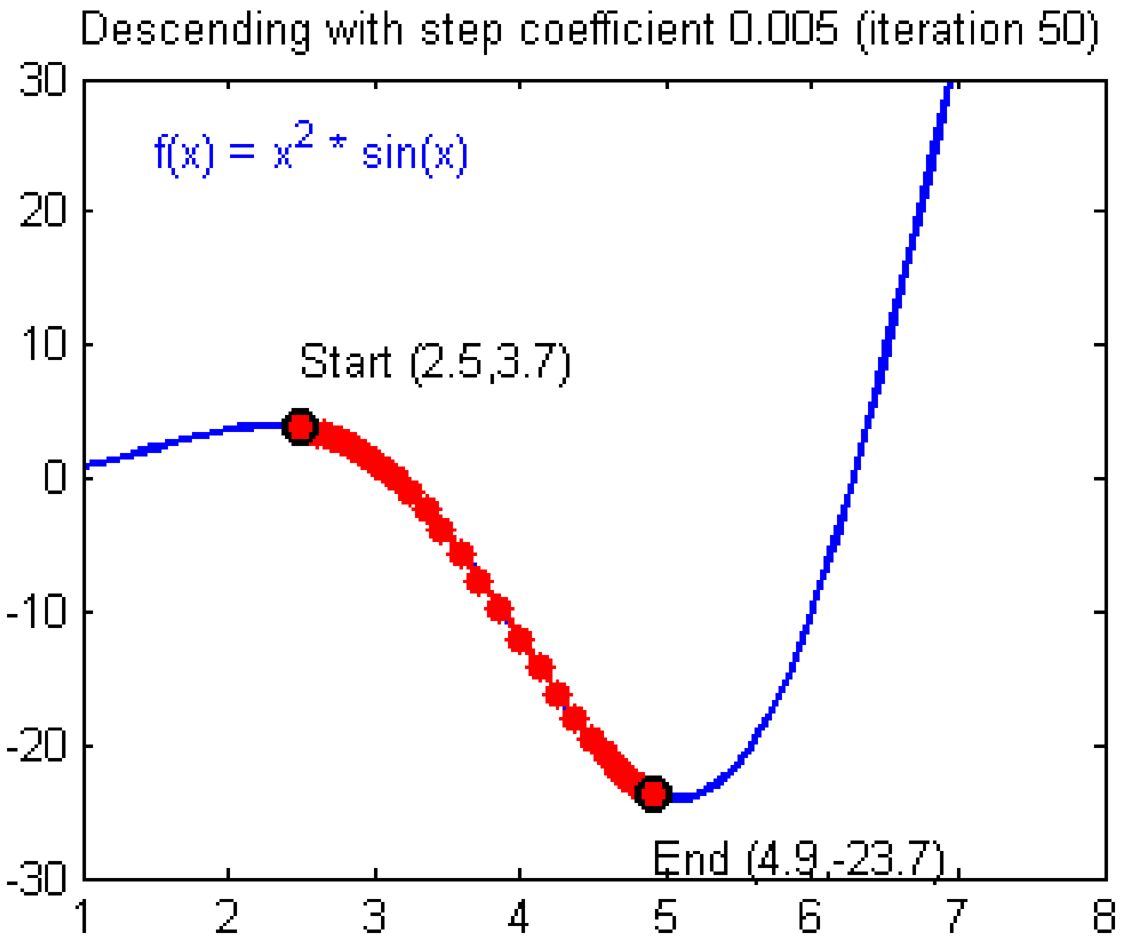
Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

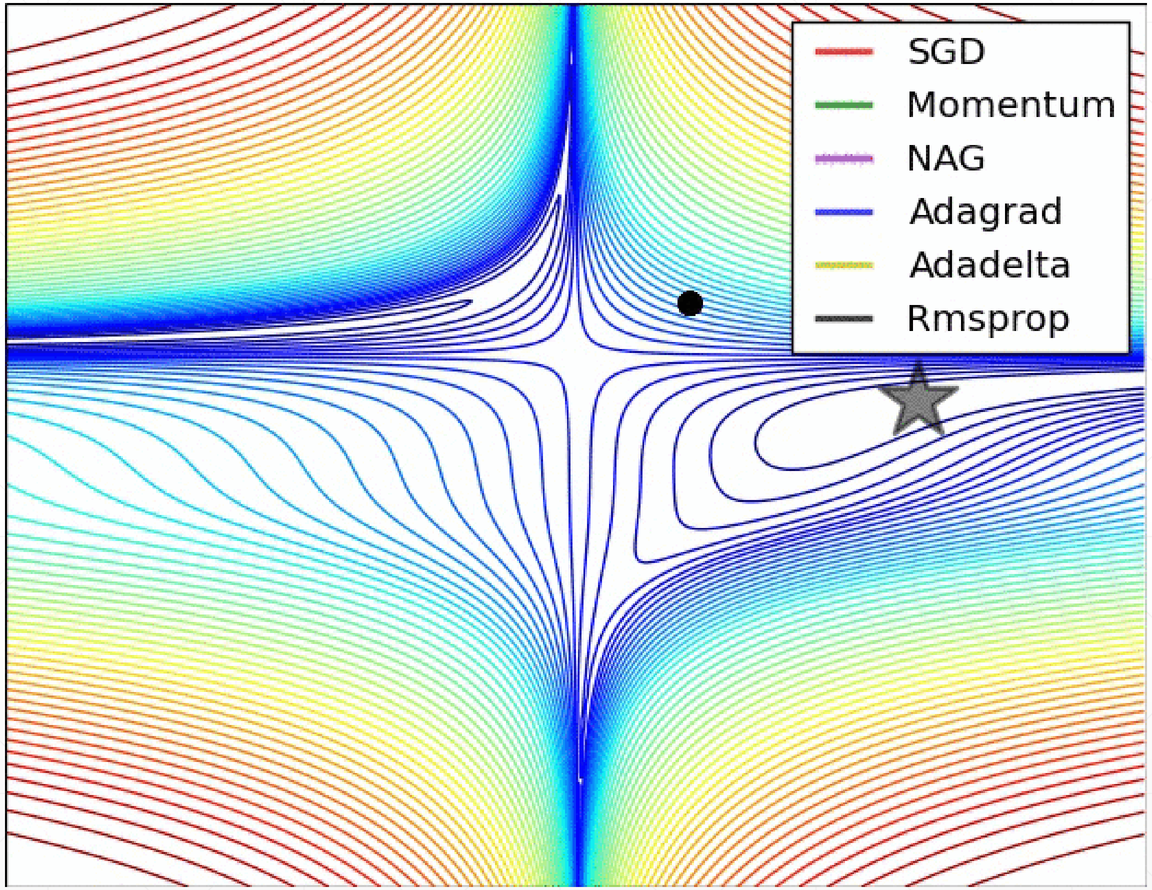
$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

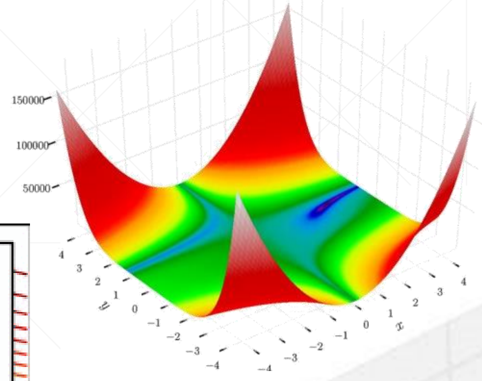
http://mccormickml.com/2014/03/04/gradient-descent-derivation/

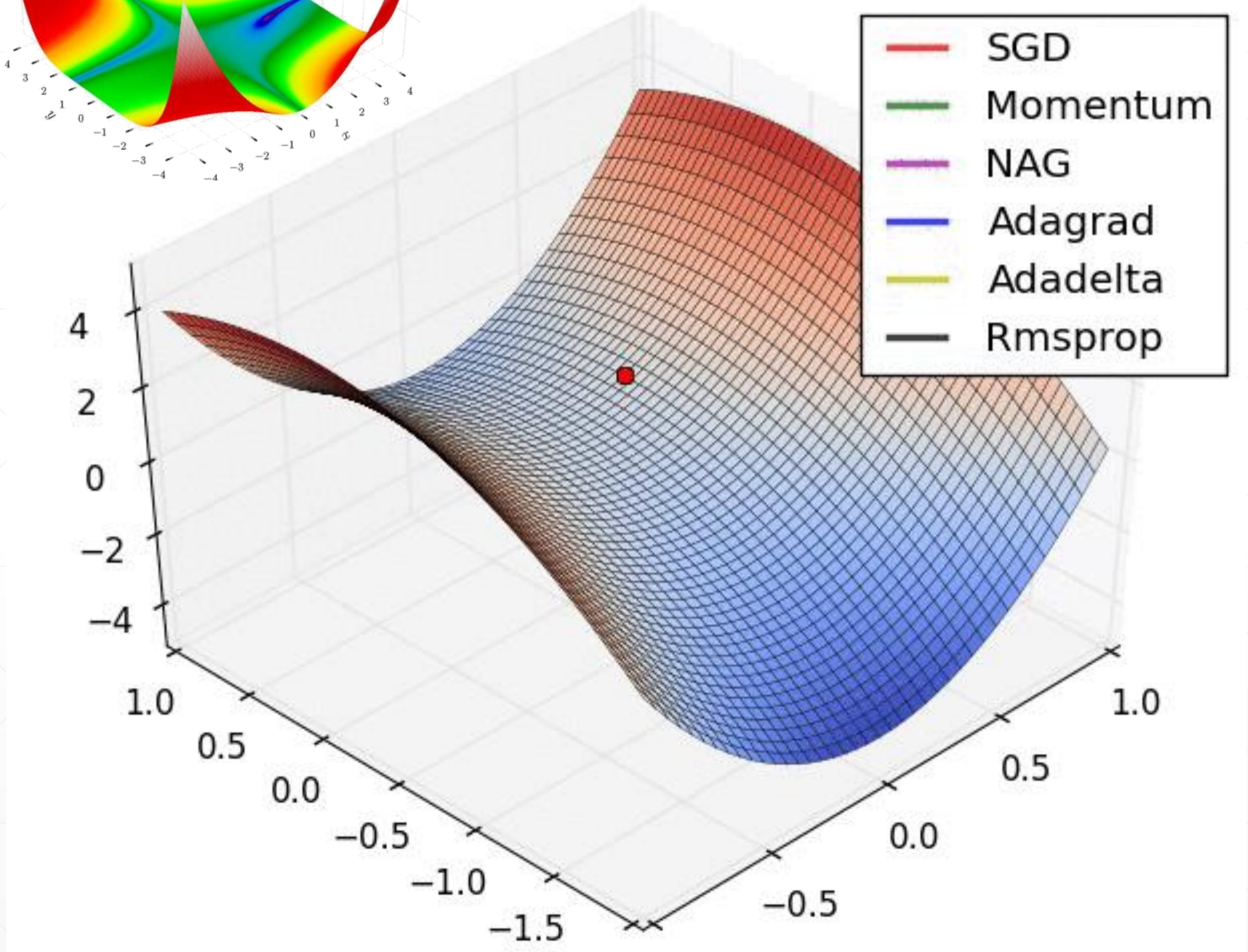
Learning process-1



Learning process-2

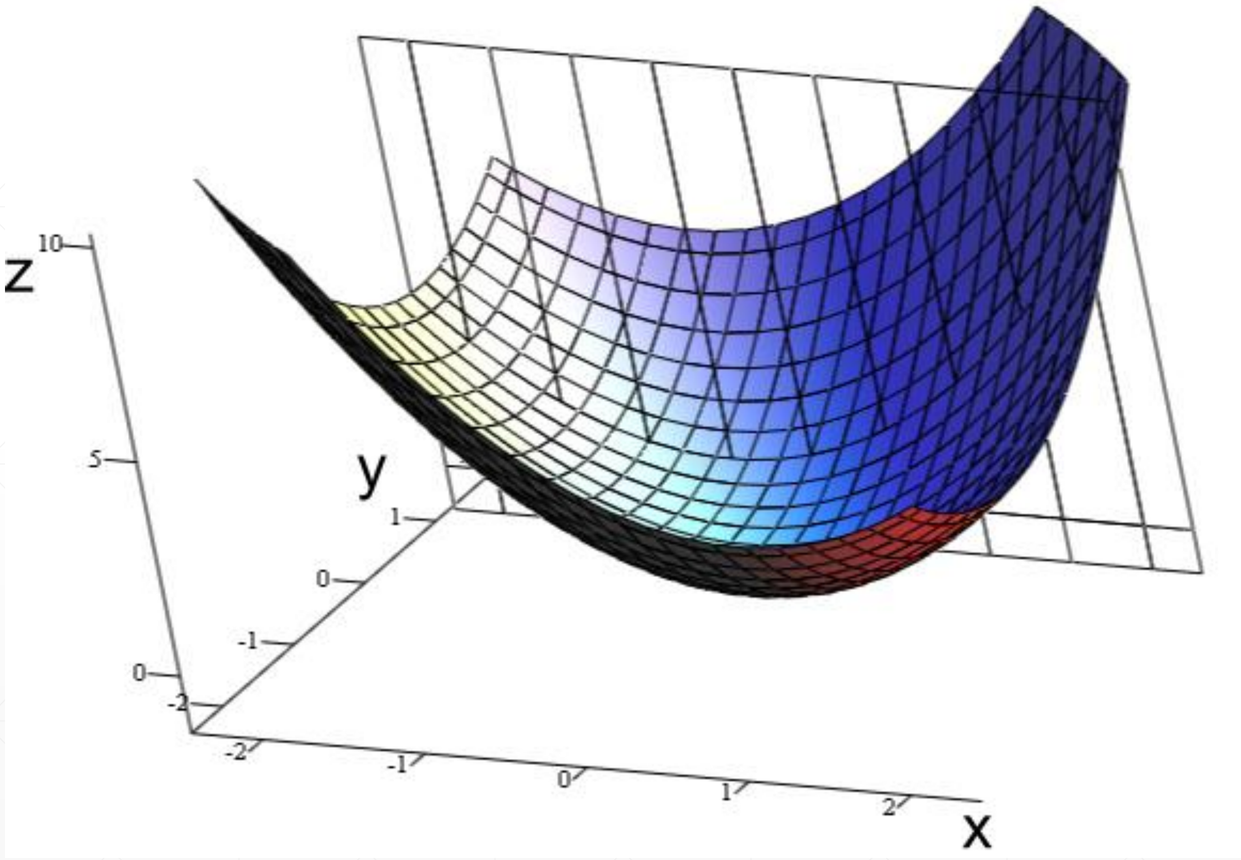




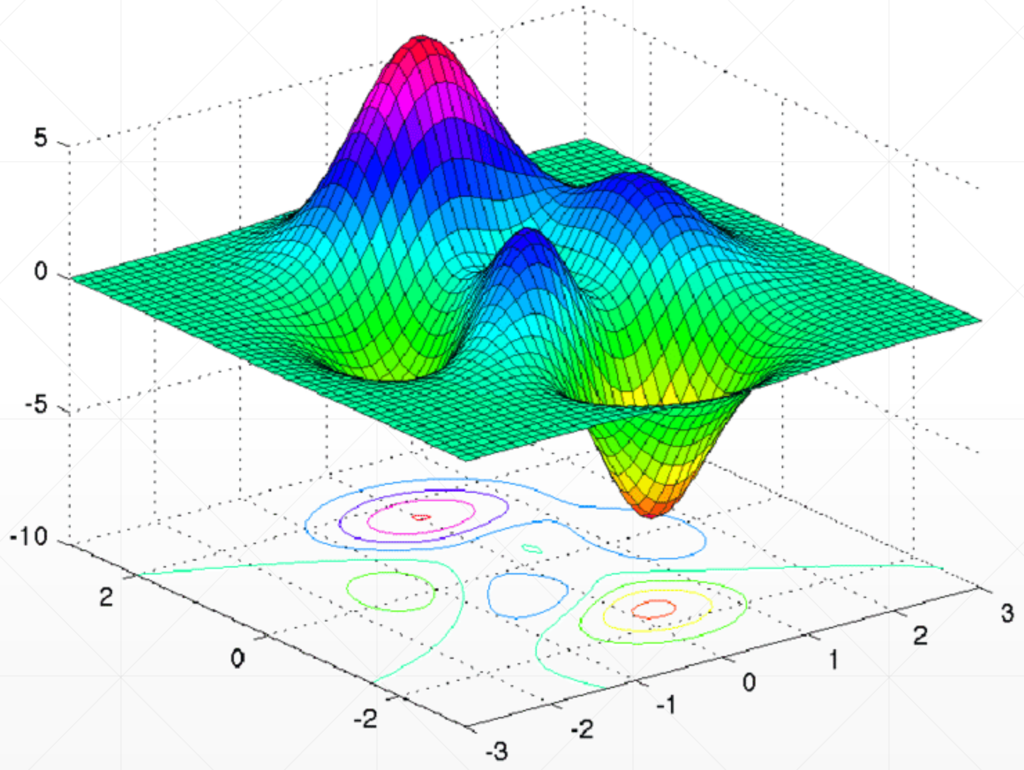


http://ruder.io/optimizing-gradient-descent/

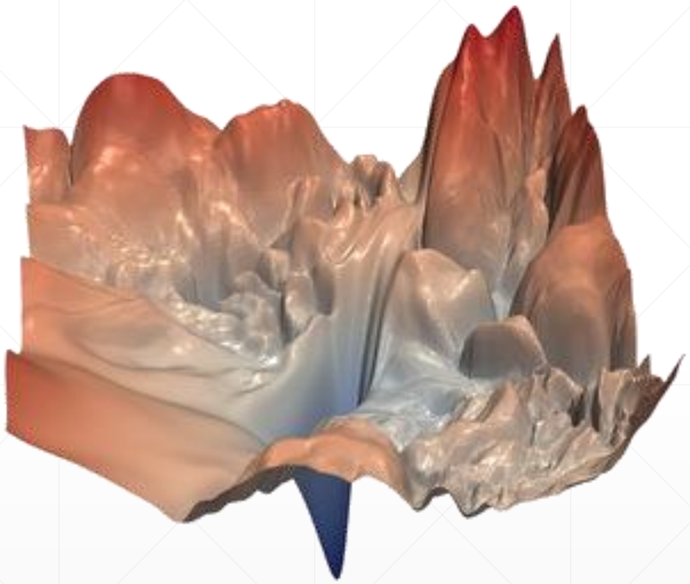
Convex function

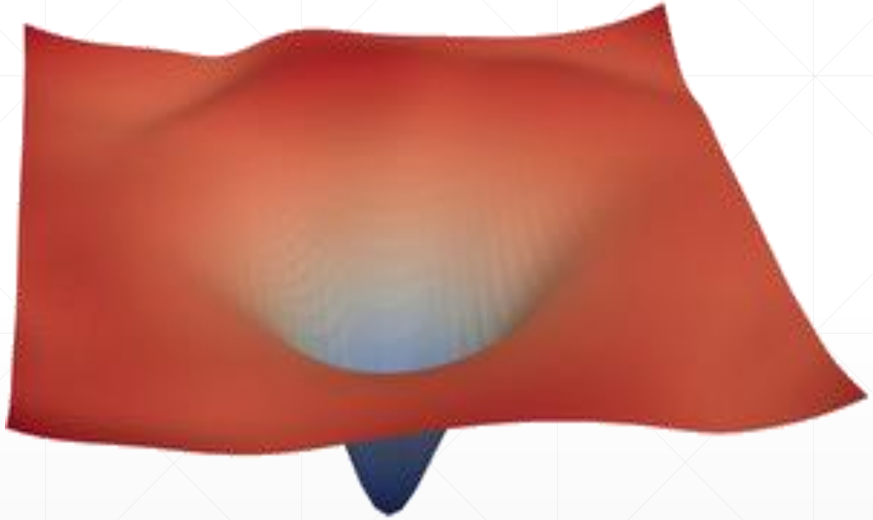


Local Minima



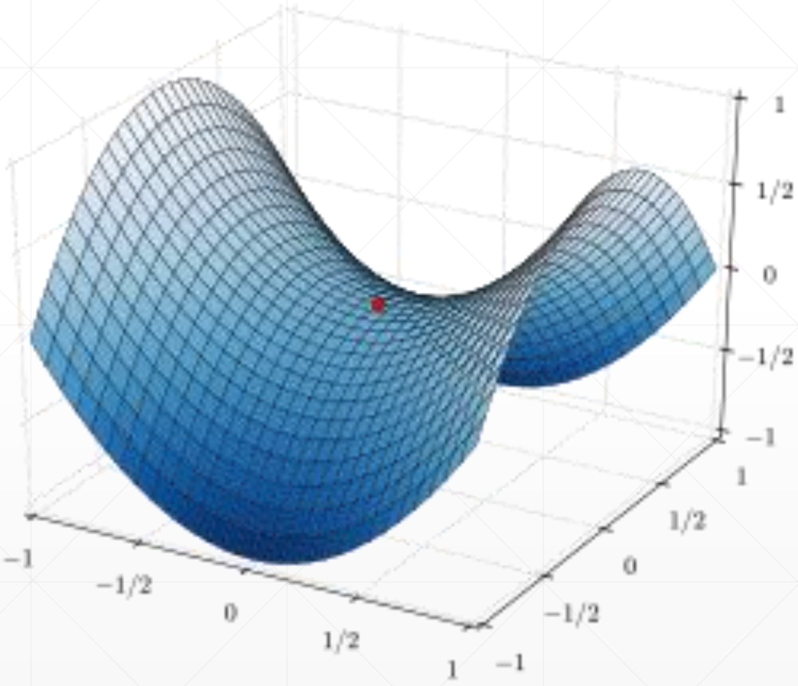
ResNet-56

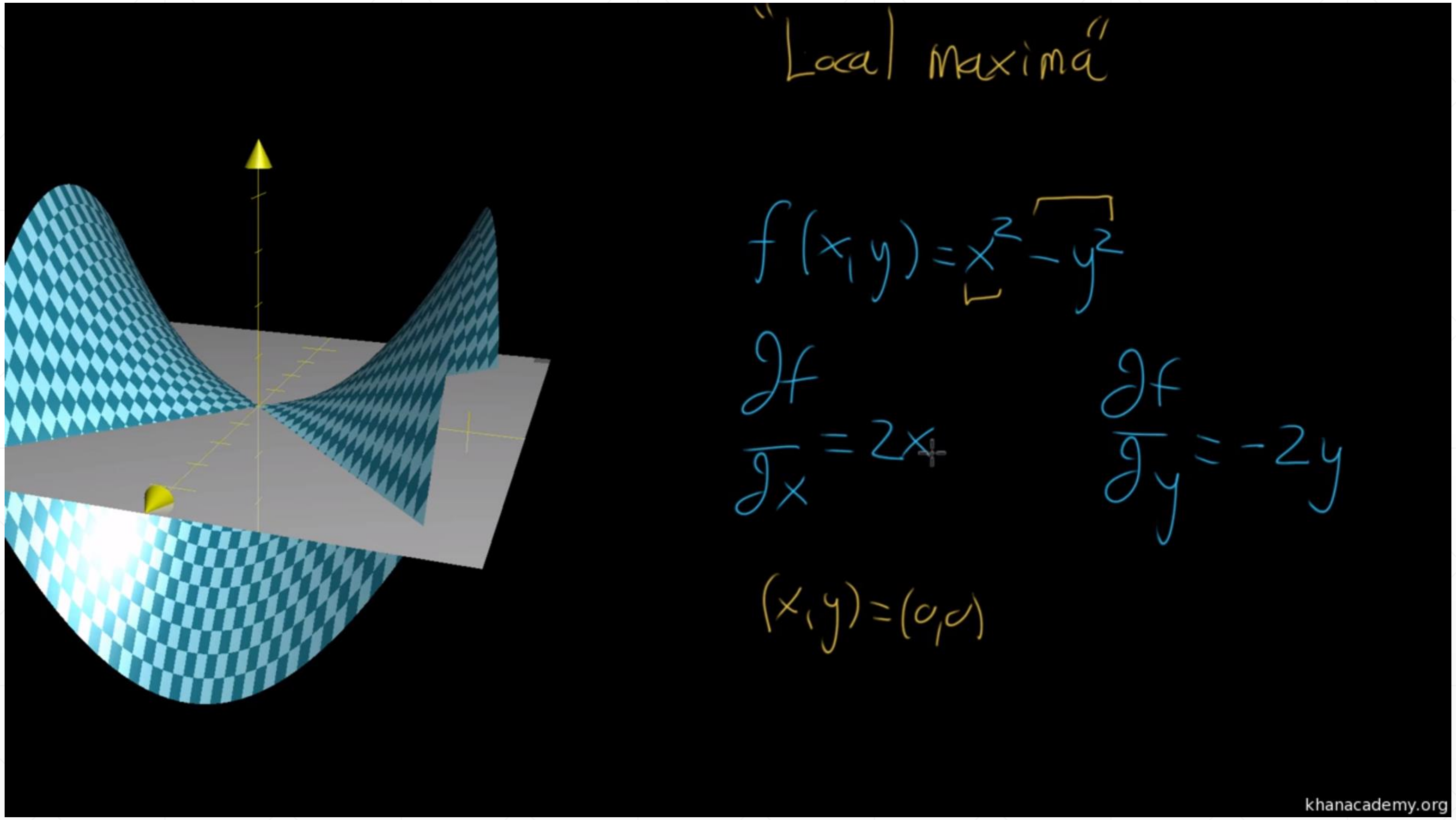




https://github.com/tomgoldstein/loss-landscape

Saddle point



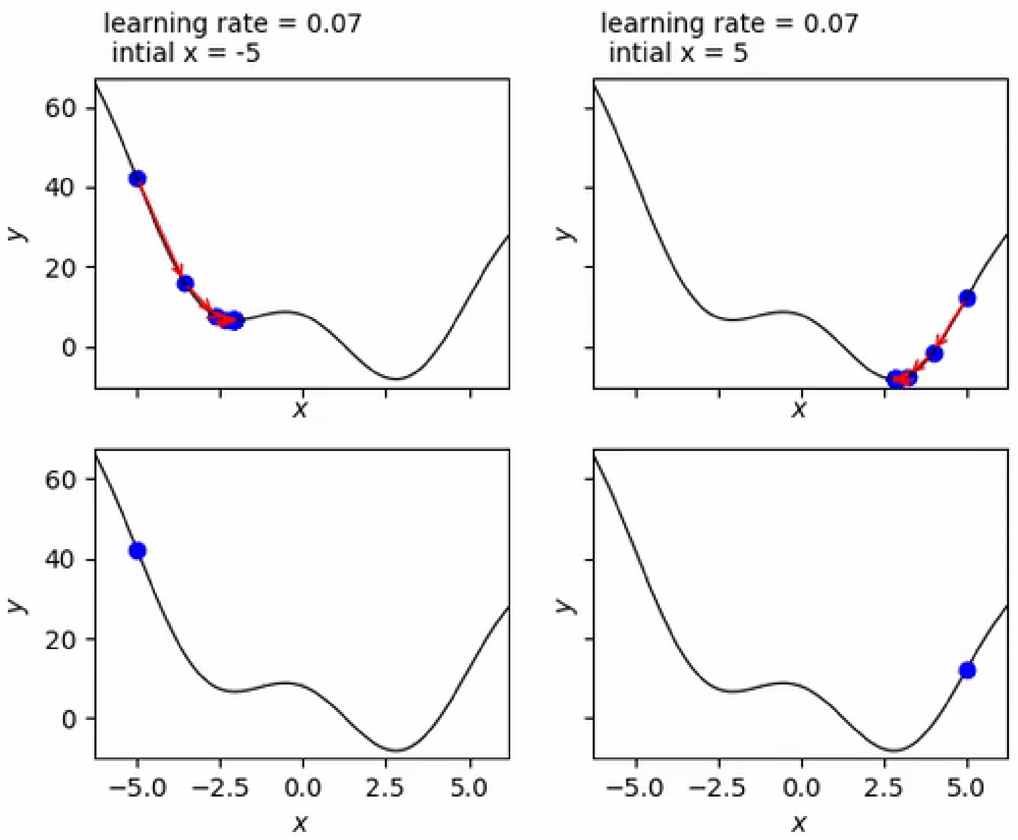


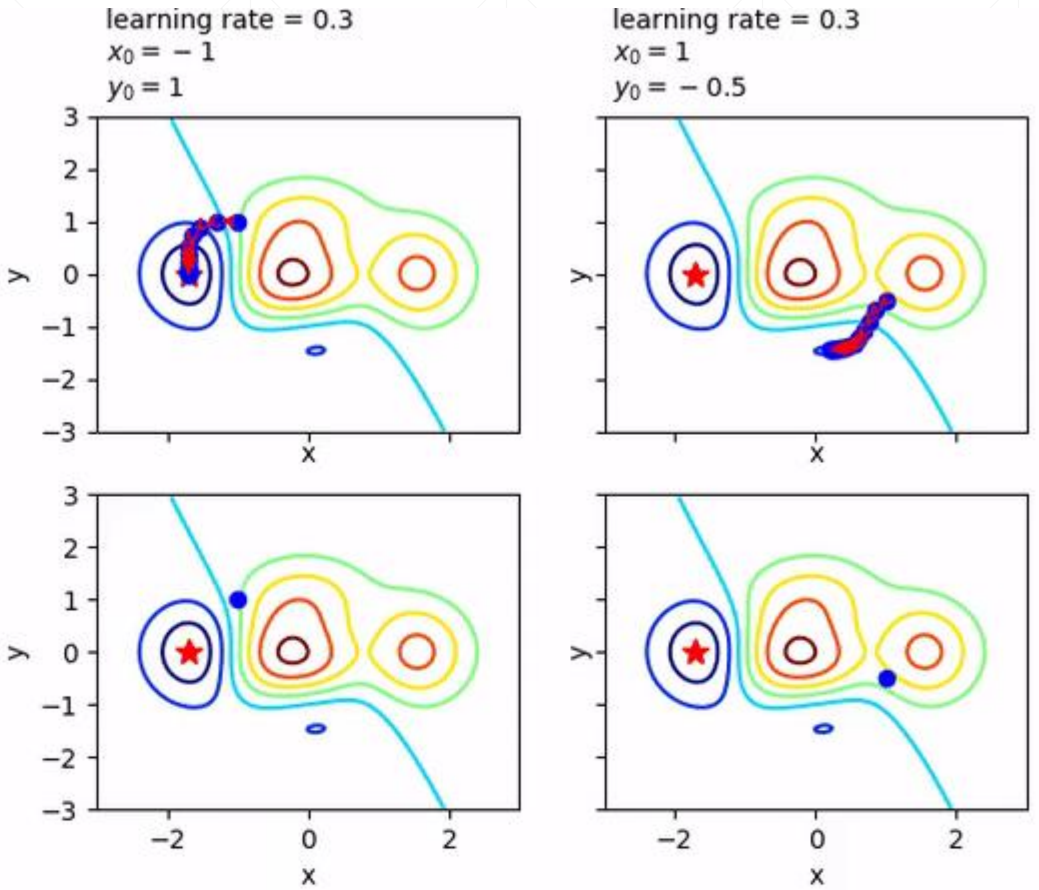
https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions-videos/v/saddle-points

Optimizer Performance

- initialization status
- learning rate
- momentum
- etc.

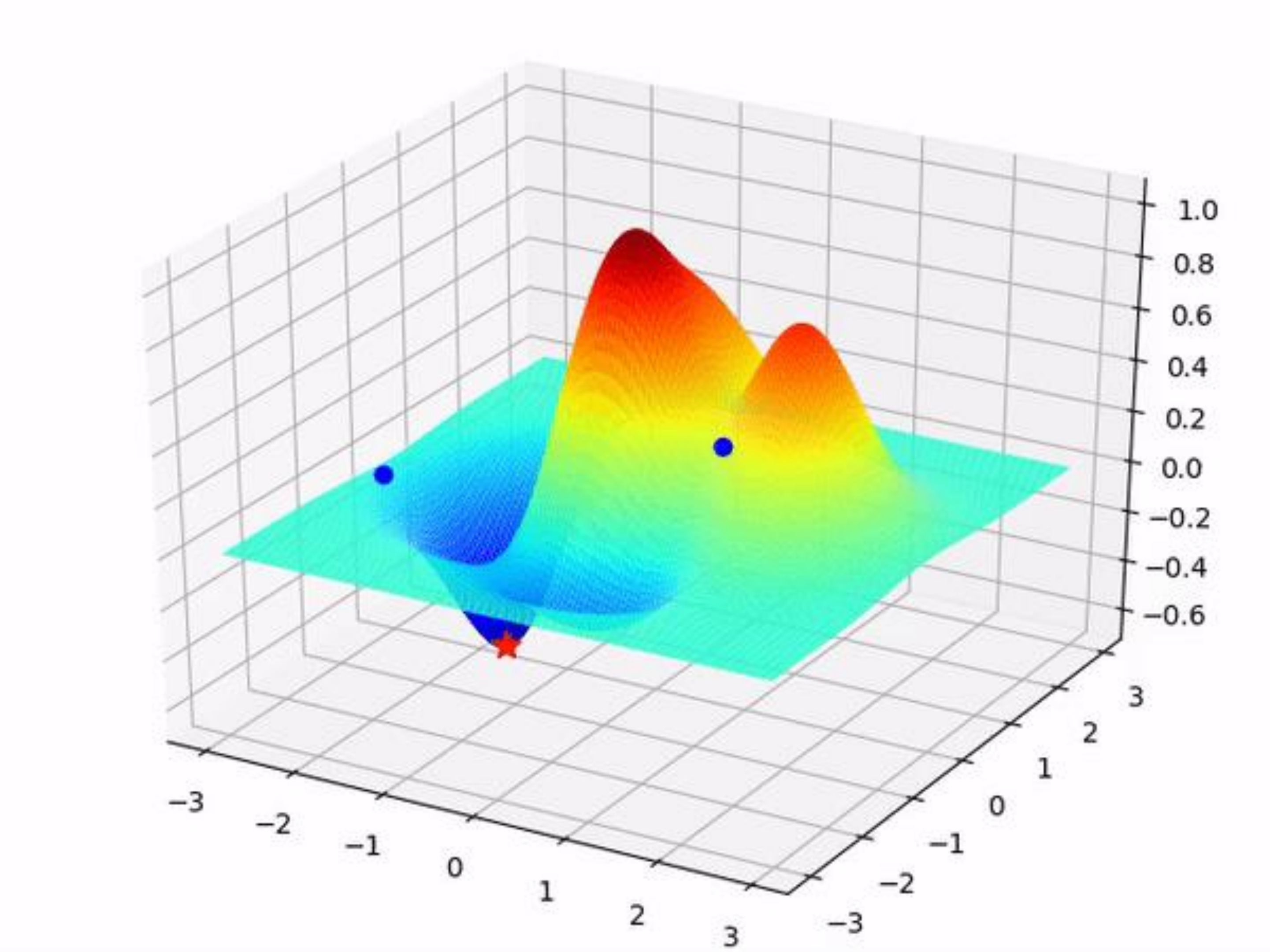
Initialization



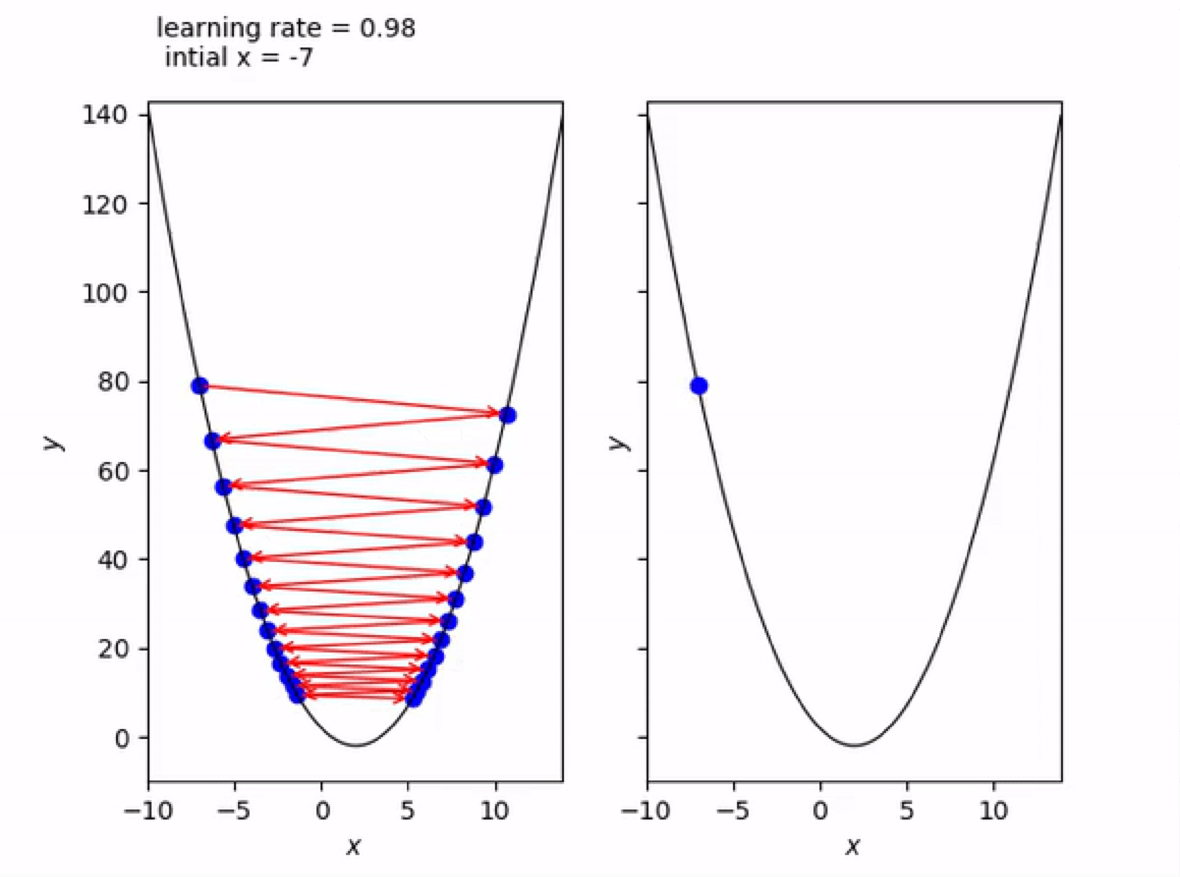


https://jed-ai.github.io/opt2_gradient_descent_1/

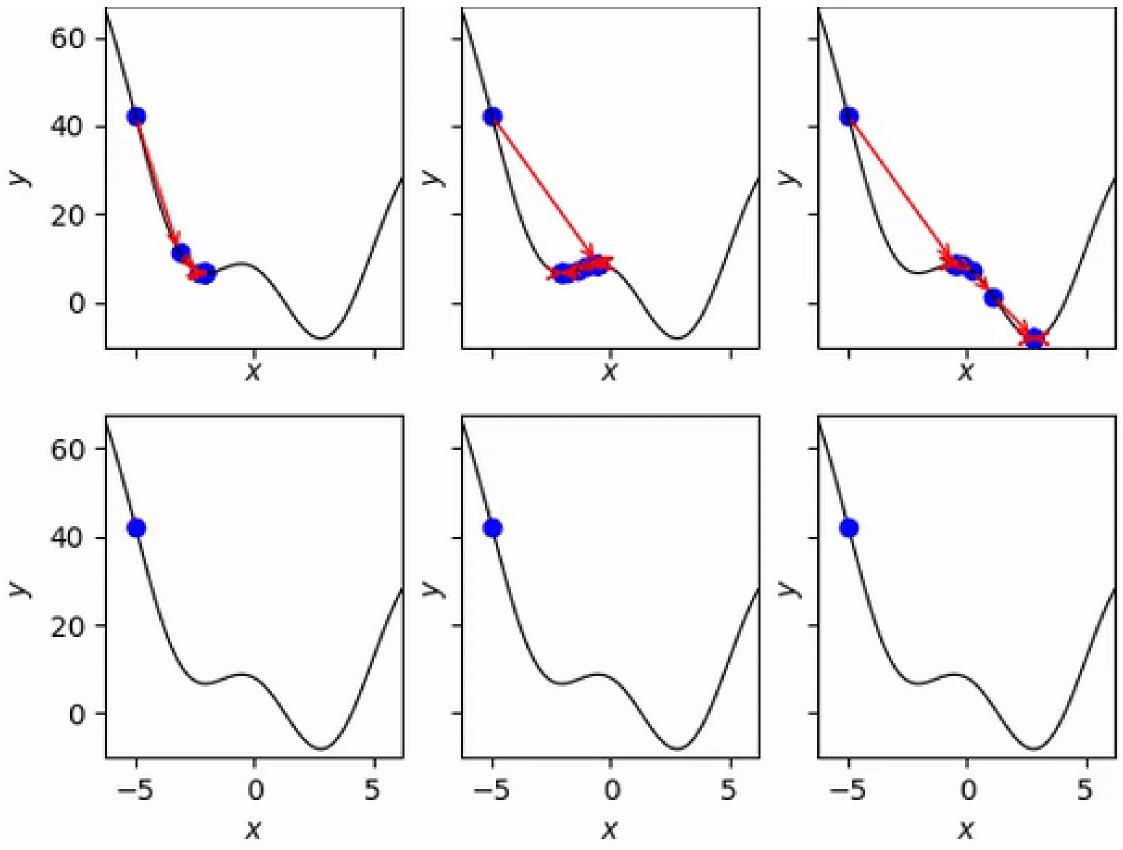
Initialization



Learning rate



Escape minima



下一课时

常见函数梯度

Thank You.

PyTorch

常见函数梯度

主讲人：龙良曲

Common Functions

Common Functions	Function	Derivative
Constant	c	0
Line	x	1
	ax	a
Square	x^2	$2x$
Square Root	$\sqrt{x}$	$(½)x^{-½}$
Exponential	e^x	e^x
	a^x	$\ln(a)\, a^x$
Logarithms	$\ln(x)$	$1/x$
	$\log_a(x)$	$1 / (x \ln(a))$
Trigonometry (x is in radians)	$\sin(x)$	$\cos(x)$
	$\cos(x)$	$-\sin(x)$
	$\tan(x)$	$\sec^2(x)$

$$xw + b$$

$$xw^2 + b^2$$

$$xe^{w} + e^{b}$$

$$[\boldsymbol{y} - (\boldsymbol{xw} + \boldsymbol{b})]^2$$

$$\mathrm{y}\log(\boldsymbol{x}\boldsymbol{w} + \boldsymbol{b})$$

下一课时

什么是激活函数

Thank You.

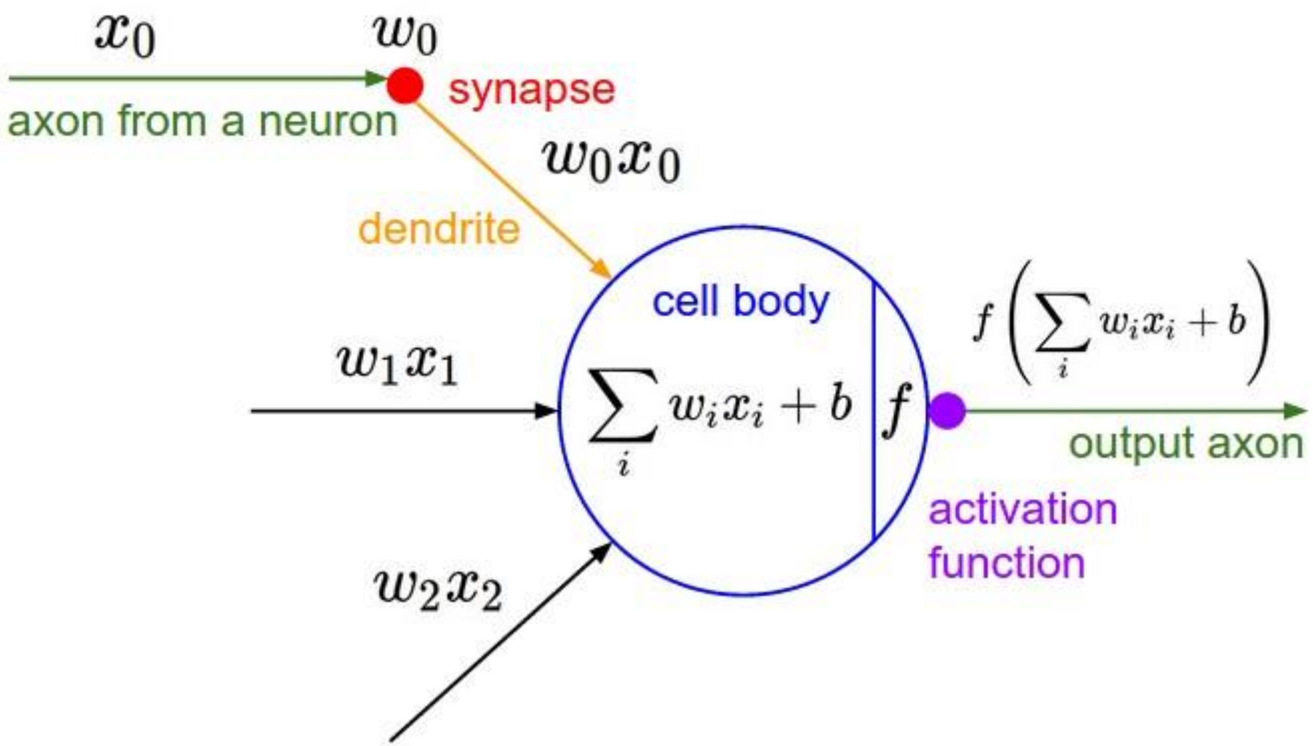
PyTorch

激活函数及其梯度

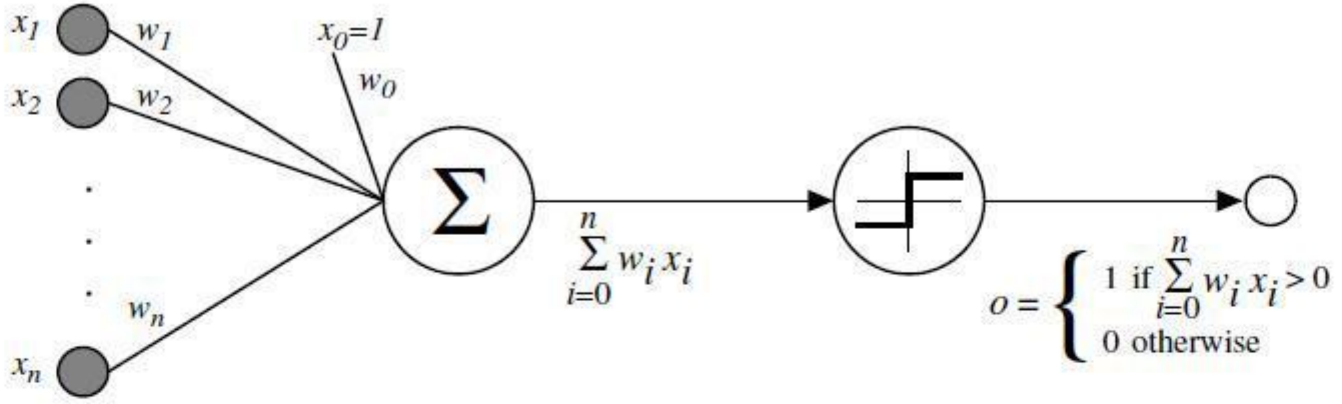
主讲人：龙良曲

Activation Functions

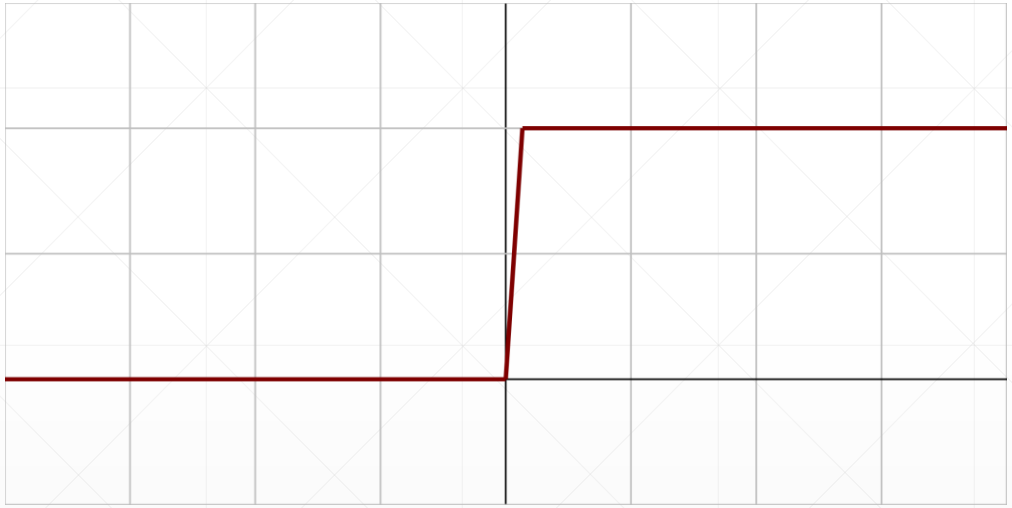
PITTS WITH LETTVIN: Pitts with Jerome Lettvin and one subject of their experiments on visual perception (1959). Wikipedia



Derivative

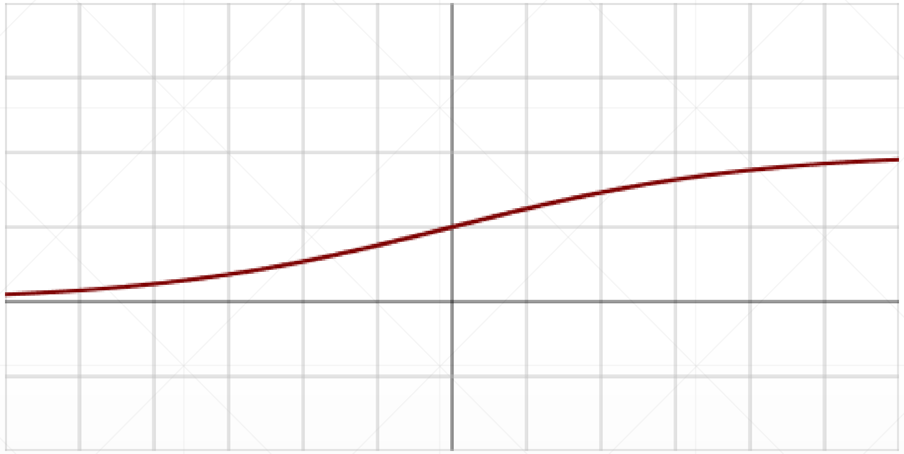


x_1
w_1
$x_0=1$
w_0
x_2
w_2
Σ
$\sum_{i=0}^{n} w_i x_i$
w_n
x_n
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ 0 \text{ otherwise} \end{cases}$



Sigmoid / Logistic

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$



Derivative

$$
\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{(1+e^{-x})-1}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}}{(1+e^{-x})^2} - \left(\frac{1}{1+e^{-x}}\right)^2 \\
&= \sigma(x) - \sigma(x)^2 \\
\sigma' &= \sigma(1-\sigma)
\end{aligned}
$$

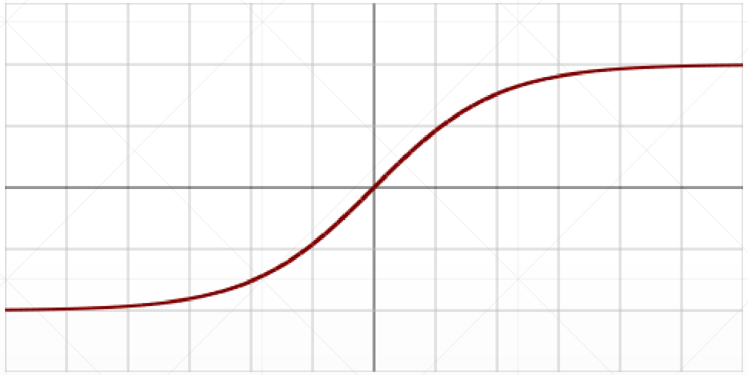
torch.sigmoid

```
In [5]: a=torch.linspace(-100,100,10)

In [6]: a
Out[6]:
tensor([-100.0000,  -77.7778,  -55.5556,  -33.3333,  -11.1111,   11.1111,
          33.3333,   55.5555,   77.7778,  100.0000])

In [7]: torch.sigmoid(a)
Out[7]:
tensor([0.0000e+00, 1.6655e-34, 7.4564e-25, 3.3382e-15, 1.4945e-05, 9.9999e-01,
        1.0000e+00, 1.0000e+00, 1.0000e+00, 1.0000e+00])
```

Tanh

$$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

$$= 2sigmoid(2x) - 1$$



Derivative

$$\frac{d}{dx}\tanh(x) = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2}$$

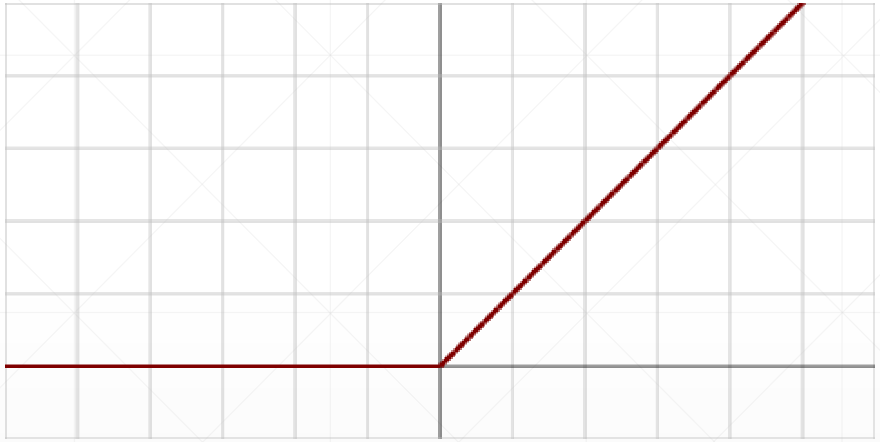
$$= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - \tanh^2(x)$$

torch.tanh

```
In [9]: a=torch.linspace(-1,1,10)

In [10]: torch.tanh(a)
Out[10]:
tensor([-0.7616, -0.6514, -0.5047, -0.3215, -0.1107,  0.1107,  0.3215,  0.5047,
         0.6514,  0.7616])
```

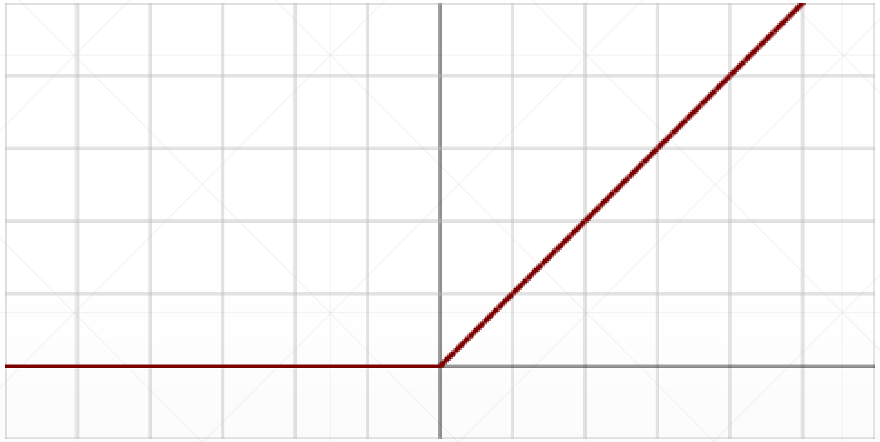
Rectified Linear Unit

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$



Derivative

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$



F.relu

```
In [11]: from torch.nn import functional as F

In [12]: a=torch.linspace(-1,1,10)

In [13]: torch.relu(a)
Out[13]:
tensor([0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1111, 0.3333, 0.5556, 0.7778,
        1.0000])

In [14]: F.relu(a)
Out[14]:
tensor([0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1111, 0.3333, 0.5556, 0.7778,
        1.0000])
```

下一课时

Loss及其梯度

Thank You.

PyTorch

LOSS及其梯度

主讲人：龙良曲

Typical Loss

- Mean Squared Error

- Cross Entropy Loss
  - binary
  - multi-class
  - +softmax
  - Leave it to Logistic Regression Part

MSE

- $\mathrm{loss} = \sum[y - (xw + b)]^2$
- $L2 - norm = \left\| y - (xw + b) \right\|_2$
- $loss = norm\left(y - (xw + b)\right)^2$

Derivative

- $\mathrm{loss} = \sum[y - f_\theta(x)]^2$

- $\frac{\nabla loss}{\nabla \theta} = 2\sum[y - f_\theta(x)] * \frac{\nabla f_\theta(x)}{\nabla \theta}$

autograd.grad

```
In [15]: x=torch.ones(1)
In [17]: w=torch.full([1],2)
In [19]: mse=F.mse_loss(torch.ones(1), x*w)
Out[20]: tensor(1.)

In [21]: torch.autograd.grad(mse,[w])
#RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn

In [22]: w.requires_grad_()
Out[22]: tensor([2.], requires_grad=True)

In [23]: torch.autograd.grad(mse,[w])
#RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn

In [24]: mse=F.mse_loss(torch.ones(1), x*w)

In [25]: torch.autograd.grad(mse,[w])
Out[25]: (tensor([2.]),)
```

loss.backward

```
In [15]: x=torch.ones(1)
In [17]: w=torch.full([1],2)
In [19]: mse=F.mse_loss(torch.ones(1), x*w)
Out[20]: tensor(1.)

In [21]: torch.autograd.grad(mse,[w])
#RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn

In [22]: w.requires_grad_()
Out[22]: tensor([2.], requires_grad=True)

In [23]: torch.autograd.grad(mse,[w])
#RuntimeError: element 0 of tensors does not require grad and does not have a grad_fn

In [24]: mse=F.mse_loss(torch.ones(1), x*w)
In [27]: mse.backward()

In [28]: w.grad
Out[28]: tensor([2.])
```

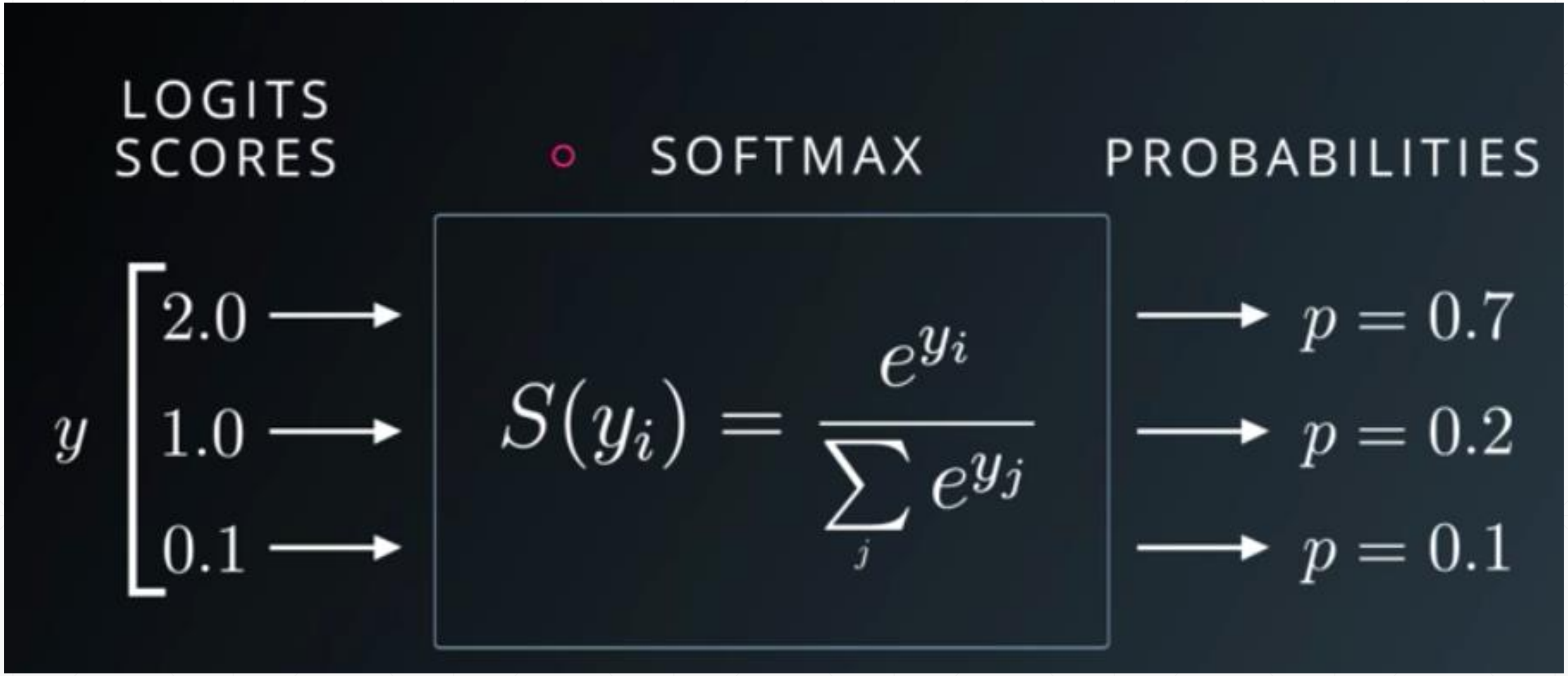
Gradient API

- torch.autograd.grad(loss, [w1, w2,…])
  - [w1 grad, w2 grad…]

- loss.backward()
  - w1.grad
  - w2.grad

Softmax

- soft version of max



Derivative

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

$$f(x) = \frac{g(x)}{h(x)}$$

$$f'(x) = \frac{g\prime(x)h(x) - h\prime(x)g(x)}{h(x)^2}$$

$$g(x) = e^{a_i}$$

$$h(x) = \sum_{k=1}^{N} e^{a_k}$$

when $i = j$

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} = \frac{e^{a_i} \sum_{k=1}^{N} e^{a_k} - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{e^{a_i} \left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\sum_{k=1}^{N} e^{a_k}}$$

$$= p_i(1 - p_j)$$

Derivative

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$\frac{\partial p_i}{\partial a_j} = \frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j}$$

$$f(x) = \frac{g(x)}{h(x)}$$

$$f'(x) = \frac{g\prime(x)h(x) - h\prime(x)g(x)}{h(x)^2}$$

$$g(x) = e^{a_i}$$

$$h(x) = \sum_{k=1}^{N} e^{a_k}$$

when $i \neq j$

$$\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} = \frac{0 - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2}$$

$$= \frac{-e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$$

$$= -p_j \cdot p_i$$

Derivative

$$\frac{\partial p_i}{\partial a_j} = \begin{cases} p_i(1-p_j) & if \quad i=j \\ -p_j.p_i & if \quad i \neq j \end{cases}$$

Or using Kronecker delta $\delta ij = \begin{cases} 1 & if \quad i=j \\ 0 & if \quad i \neq j \end{cases}$

$$\frac{\partial p_i}{\partial a_j} = p_i(\delta_{ij} - p_j)$$

F.softmax

```
In [29]: a=torch.rand(3) # tensor([0.1440, 0.5349, 0.7022])
In [33]: a.requires_grad_()
Out[33]: tensor([0.1440, 0.5349, 0.7022], requires_grad=True)

In [34]: p=F.softmax(a,dim=0)

In [35]: p.backward()
RuntimeError: Trying to backward through the graph a second time, but the buffers have already been
freed. Specify retain_graph=True when calling backward the first time.

In [38]: p=F.softmax(a,dim=0)

In [39]: torch.autograd.grad(p[1],[a],retain_graph=True)
Out[39]: (tensor([-0.0828,  0.2274, -0.1447]),)

In [40]: torch.autograd.grad(p[2],[a])
Out[40]: (tensor([-0.0979, -0.1447,  0.2425]),)
```

下一课时

链式法则

Thank You.