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

- $\text{loss} = \sum[y - (xw + b)]^2$

- $\boxed{L2 - norm = \|y - (xw + b)\|_2}$

- $loss = norm\left(y - (xw + b)\right)^2$

Derivative

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

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

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.])
```

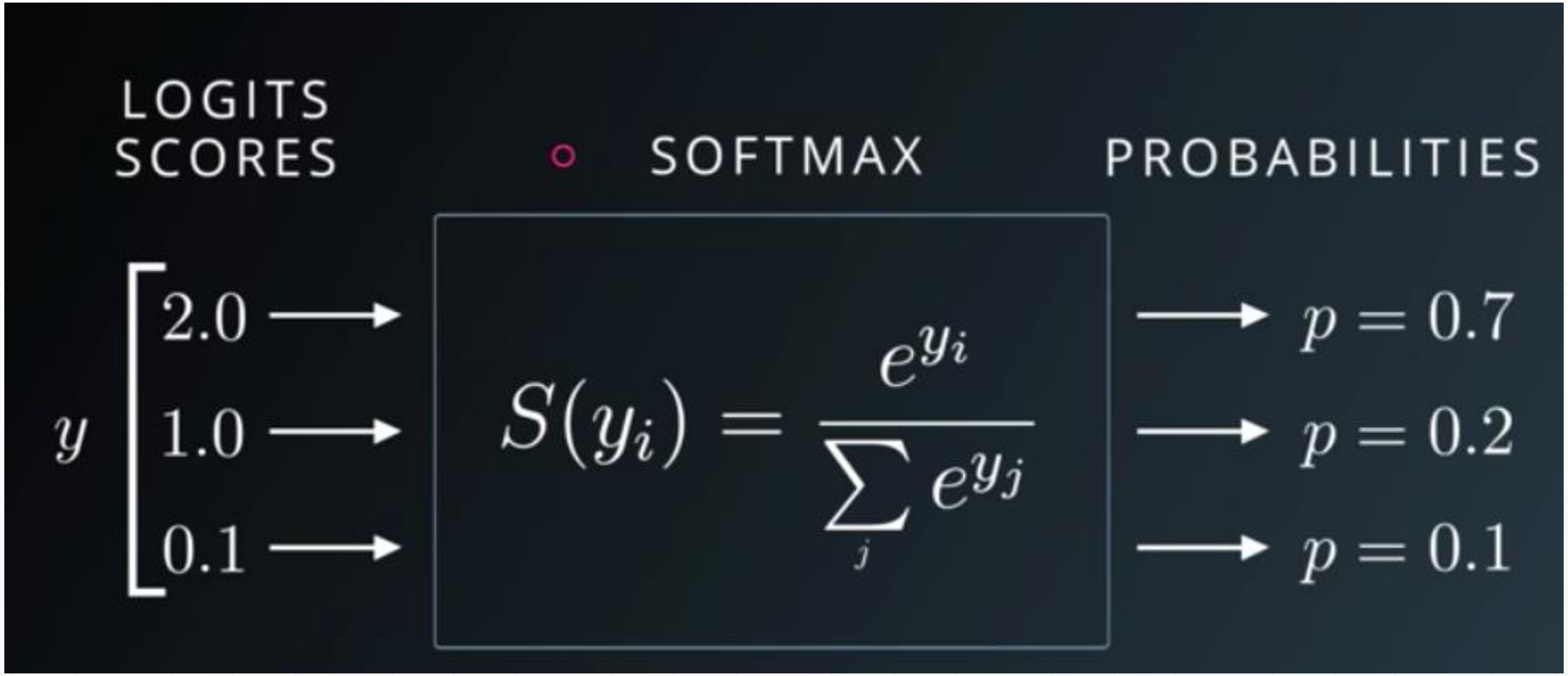
Gradient API

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

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

Softmax

- soft version of max



Derivative

导数

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

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

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

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

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

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

when $i = j$

$$\begin{aligned}
\frac{\partial \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}}{\partial a_j} &= \frac{e^{a_i} \sum_{k=1}^{N} e^{a_k} - e^{a_j} e^{a_i}}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_i} \left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\left(\sum_{k=1}^{N} e^{a_k}\right)^2} \\
&= \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}} \times \frac{\left(\sum_{k=1}^{N} e^{a_k} - e^{a_j}\right)}{\sum_{k=1}^{N} e^{a_k}} \\
&= p_i(1 - p_j)
\end{aligned}$$

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 $i \neq j$

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

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

$$= -p_j . p_i$$

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.