CSCI4622 Machine Learning 2020 Spring

Math behind Neural Network Training

Geena Kim

*Some of the slide/diagram adopted from CMU deep learning course

review: Learning in Perceptron

Learning in perceptron

- Perceptron rule
- Delta rule (Gradient Descent)

review: Learning in Perceptron

Perceptron algorithm

- Cycle through the training instances
- Only update W on misclassified instances
- If instance misclassified:
  - If instance is positive class (positive misclassified as negative)

$$W = W + X_i$$

  - If instance is negative class (negative misclassified as positive)

$$W = W - X_i$$

$$w_j' \leftarrow w_j - 1(\hat{y}_i - y_i) X_{ij}$$

$\hat{y}_i = 1$

$y_i = -1$

$w_j = w_j - 2$

$\hat{y}_i = -1$

$y_i = +1$

w_j

$w_j + 2$

Perceptron

Delta rule (Gradient Descent)

$$\omega_j \leftarrow \omega_j - \alpha \frac{\partial \mathcal{L}}{\partial \omega_j}$$

$$\mathcal{L} = \frac{1}{2}(\hat{y}_i - y_i)^2$$

$$\hat{y}_i = \sum_j \omega_j X_{ij}$$

$$\omega_j \leftarrow \omega_j - \alpha(\hat{y}_i - y_i) X_{ij}$$

$\mathcal{L}(w_j, x_{ji})$

$MSE = \sum_i \frac{1}{2}(y_i - \hat{y}_i)^2$

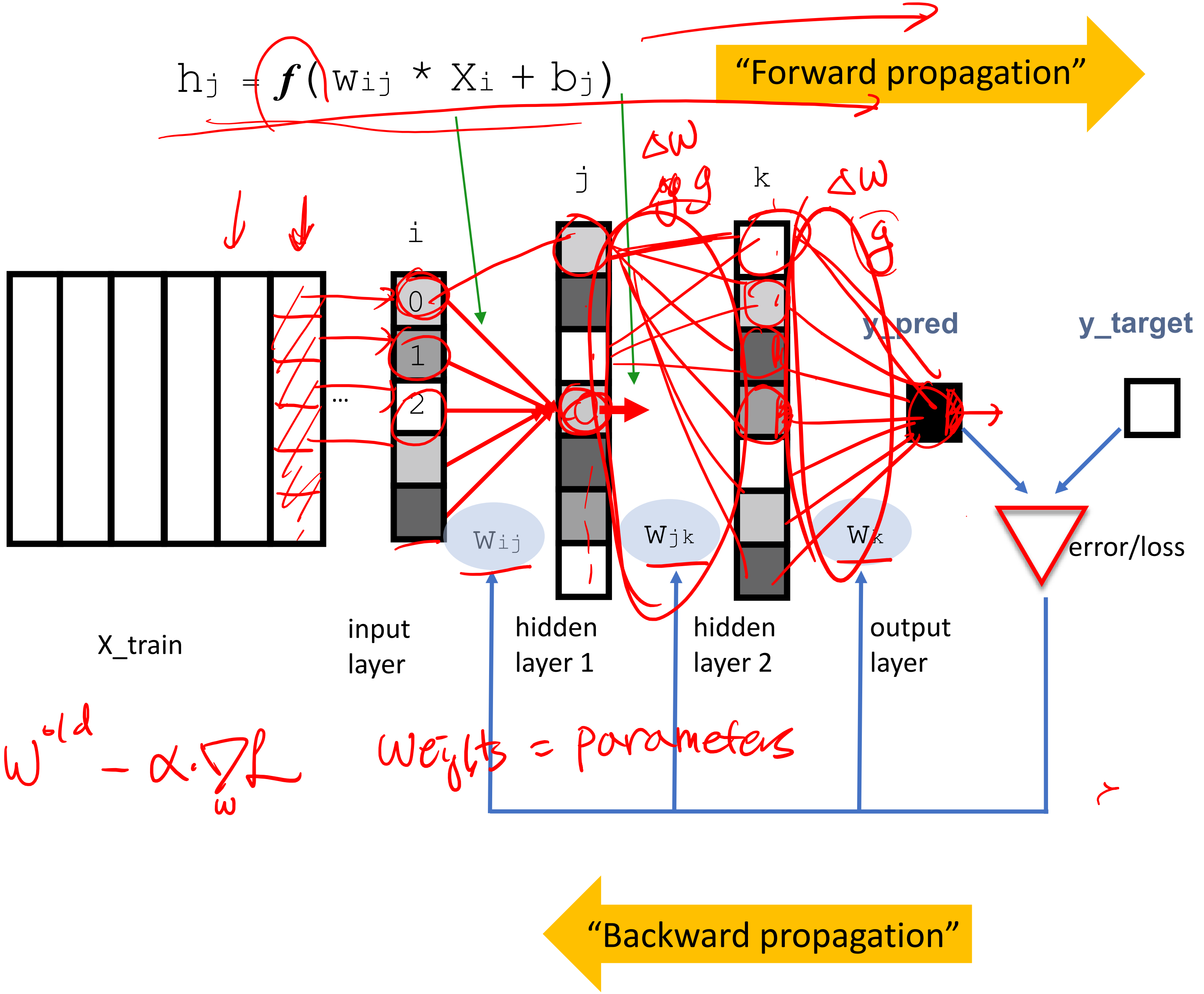
$\frac{\partial}{\partial w}(y - (wx+b))^2$

$(y - \underbrace{(wx+b)}_{\hat{y}}) \cdot (-x)$

$-\alpha(\hat{y} - y)x$

How Neural Network Training Works

$$h_j = f(w_{ij} * X_i + b_j)$$



MSE (reg)

BCE (B,C)

MCE (MC)

$$W^{new} \leftarrow W^{old} - \alpha \cdot \nabla_w L$$

Weights = Parameters

Weight Update in deep neural nets

$$W_{ij} \leftarrow W_{ij} - \alpha \boxed{\frac{\partial \mathcal{L}}{\partial W_{ij}}}$$

Weight Update Rule

Loss Function

Gradient (Chain Rule)

Back Propagation

Chain Rule Reminder

$\frac{\partial}{\partial x}\frac{1}{x} = -\frac{1}{x^2}$

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

$= g$

Example: gradient of sigmoid

$$\boxed{\sigma(z) = \frac{1}{1+e^{-z}}} = \frac{1}{g}$$

$z = \sum w_i x_i$

$g = 1 + e^{-z}$

$x_0 \; x_1 \; \cdots \; x_n$

$$\frac{\partial \sigma}{\partial w_i} = -\frac{1}{g^2} \cdot e^{-z} \cdot (-1) \cdot x_i$$

$$\frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} \cdot x_i$$

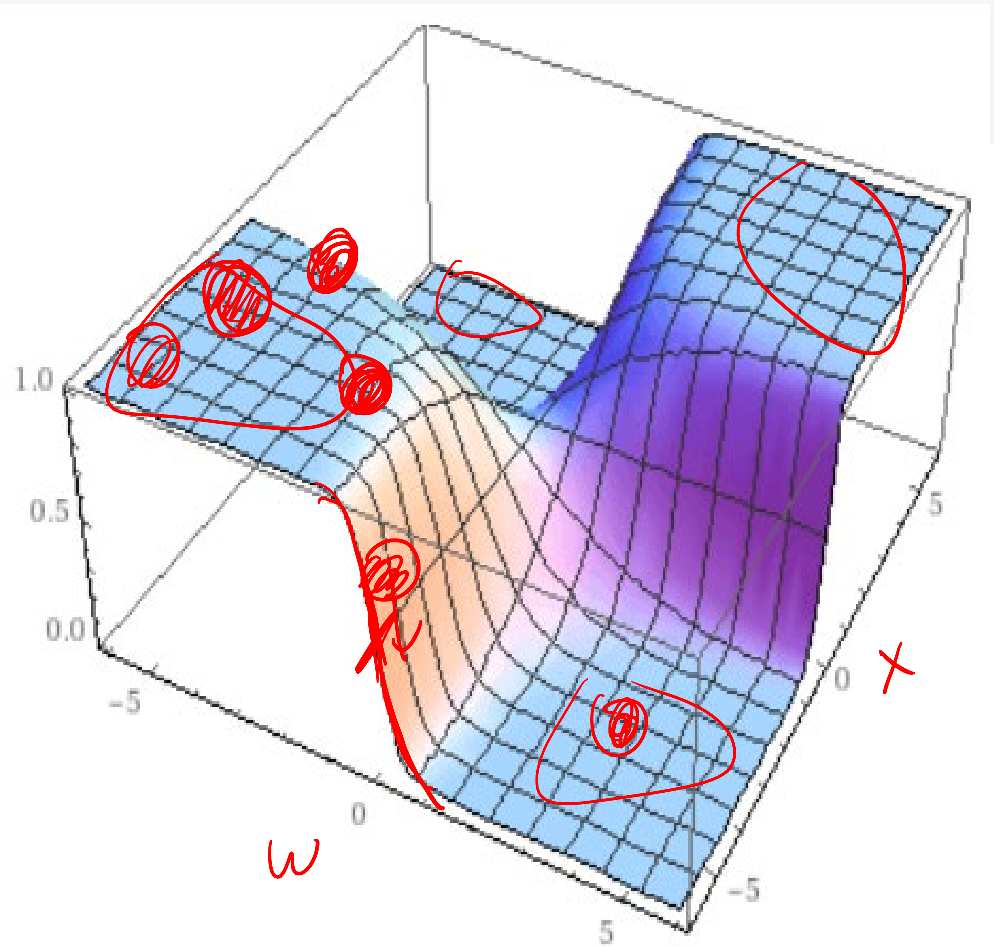
$\sigma \qquad 1-\sigma$

$\sigma \cdot (1-\sigma) \cdot x_i$

Chain Rule Reminder

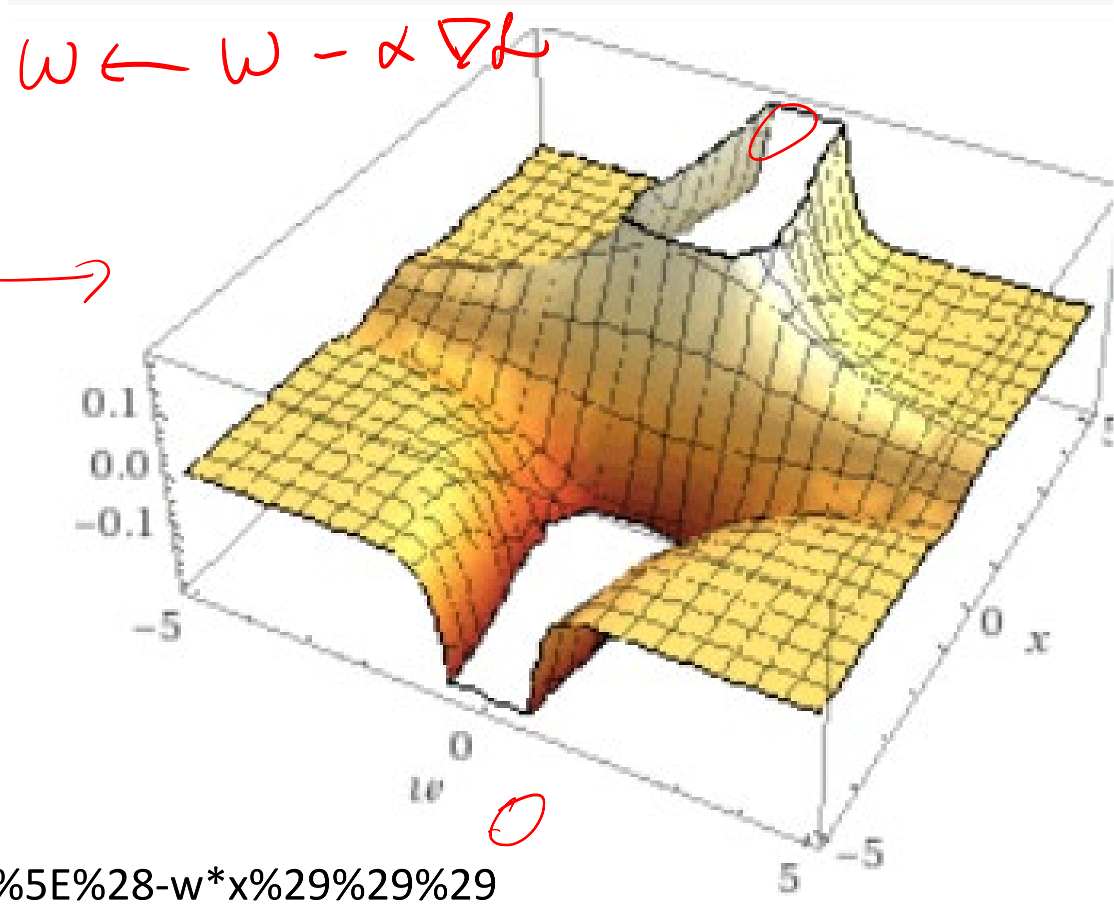
$$\frac{1}{1+e^{-w\,x}}$$

$w\,x$



$$\frac{\partial}{\partial w}\left(\frac{1}{1+e^{-w\,x}}\right) = \frac{x\,e^{-w\,x}}{(e^{-w\,x}+1)^2}$$

$w \leftarrow w - \alpha \nabla L$



https://www.wolframalpha.com/input/?i=d%2Fdw+%281%2F%281%2Be%5E%28-w*x%29%29%29

Calculating Gradient- Chain Rule

$$\mathcal{L} = -\frac{1}{m}\sum_{s}^{m} y_s \log a_s + (1-y_s)\log(1-a_s)$$

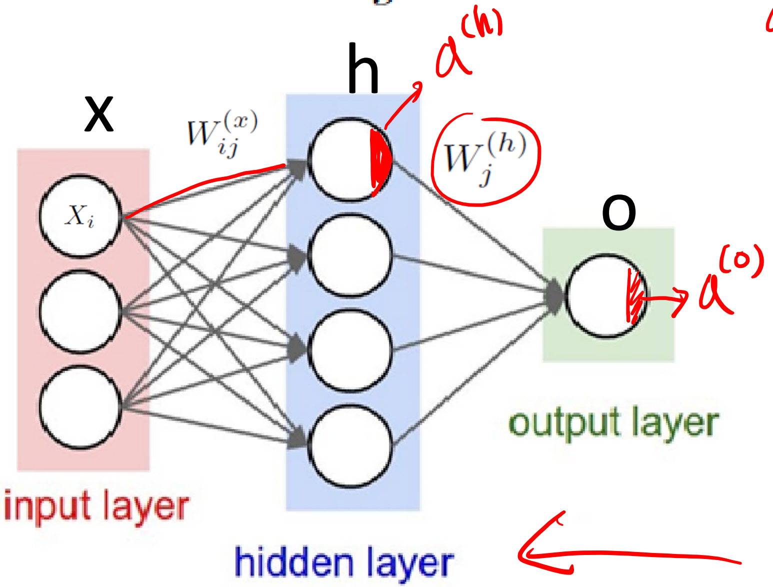
$\hat{y}_i$

$$W_{ij} \leftarrow W_{ij} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}}$$

$$\frac{\partial}{\partial x}\log x = \frac{1}{x}$$

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

$a =$



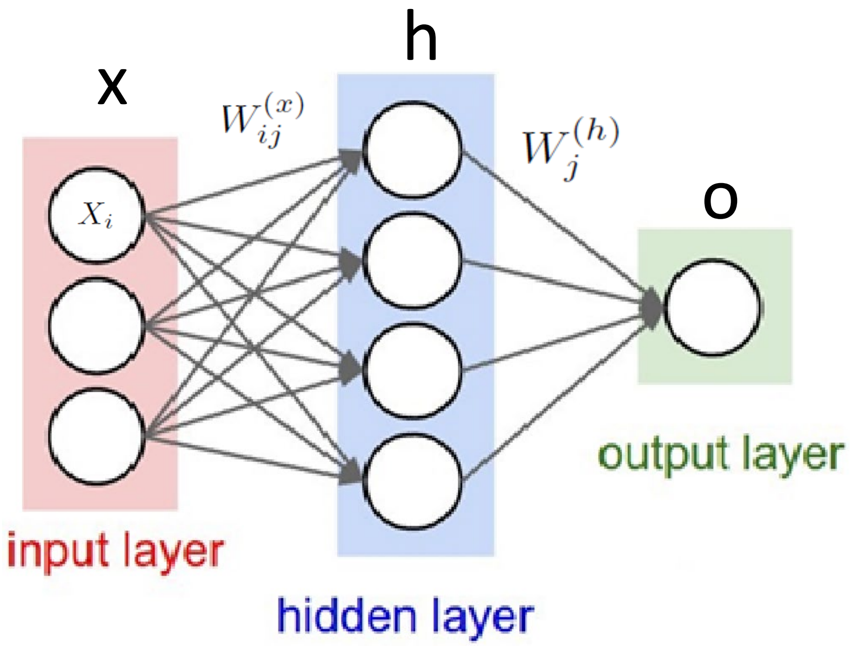
$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(x)}} = \frac{\partial \mathcal{L}(a^{(o)})}{\partial a^{(o)}}\frac{\partial a^{(o)}}{\partial W_{ij}^{(x)}}$$

$$= \left(\frac{y}{a^{(o)}} - \frac{1-y}{1-a^{(o)}}\right)\frac{\partial a^{(o)}}{\partial W_{ij}^{(x)}}$$

$$\frac{\partial a^{(o)}}{\partial W_{ij}^{(x)}} = \frac{\partial \sigma(z^{(o)})}{\partial z^{(o)}}\frac{\partial z^{(h)}}{\partial W_{ij}^{(x)}}$$

$$= \frac{\partial \sigma(z^{(o)})}{\partial z^{(o)}}\frac{\partial z^{(o)}}{\partial W_{ij}^{(x)}} = \sigma(z^{(o)})(1-\sigma(z^{(o)}))\frac{\partial z^{(o)}}{\partial W_{ij}^{(x)}}$$

Calculating Gradient- Chain Rule



$$\frac{\partial z^{(o)}}{\partial W_{ij}^{(x)}} = \sum_j^{n^{(h)}} W_j^{(h)} \frac{\partial a_j^{(h)}}{\partial W_{ij}^{(x)}}$$

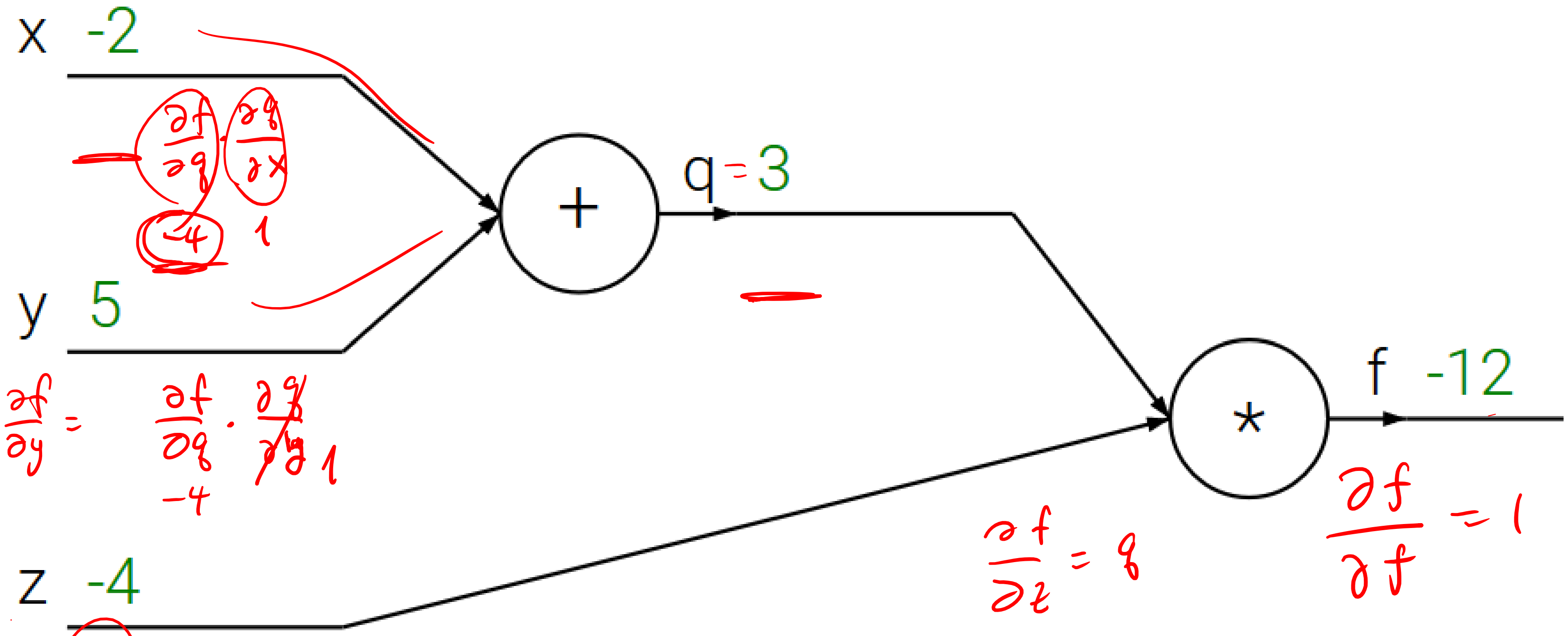
$$= \sum_j^{n^{(h)}} W_j^{(h)} \frac{\partial \sigma(z_j^{(h)})}{\partial W_{ij}^{(x)}} = \sum_j^{n^{(h)}} W_j^{(h)} \sigma(z_j^{(h)})(1-\sigma(z_j^{(h)})) \frac{\partial (z_j^{(h)})}{\partial W_{ij}^{(x)}} = \sum_j^{n^{(h)}} W_j^{(h)} \sigma(z_j^{(h)})(1-\sigma(z_j^{(h)}))X_i$$

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{(x)}} = \left(\frac{y}{a^{(o)}} - \frac{1-y}{1-a^{(o)}}\right) \sigma(z^{(o)})(1-\sigma(z^{(o)})) \sum_j^{n^{(h)}} W_j^{(h)} \sigma(z_j^{(h)})(1-\sigma(z_j^{(h)}))X_i$$

Back Propagation- Computation Graph

q = x+y

f = q*z



$\frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x}$

$-4 \quad 1$

$-4 = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y}$

$-4 \quad 1$

$\frac{\partial f}{\partial z} = q$

$\frac{\partial f}{\partial q} = z$

$\frac{\partial f}{\partial f} = 1$

Back Propagation- Computation Graph

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

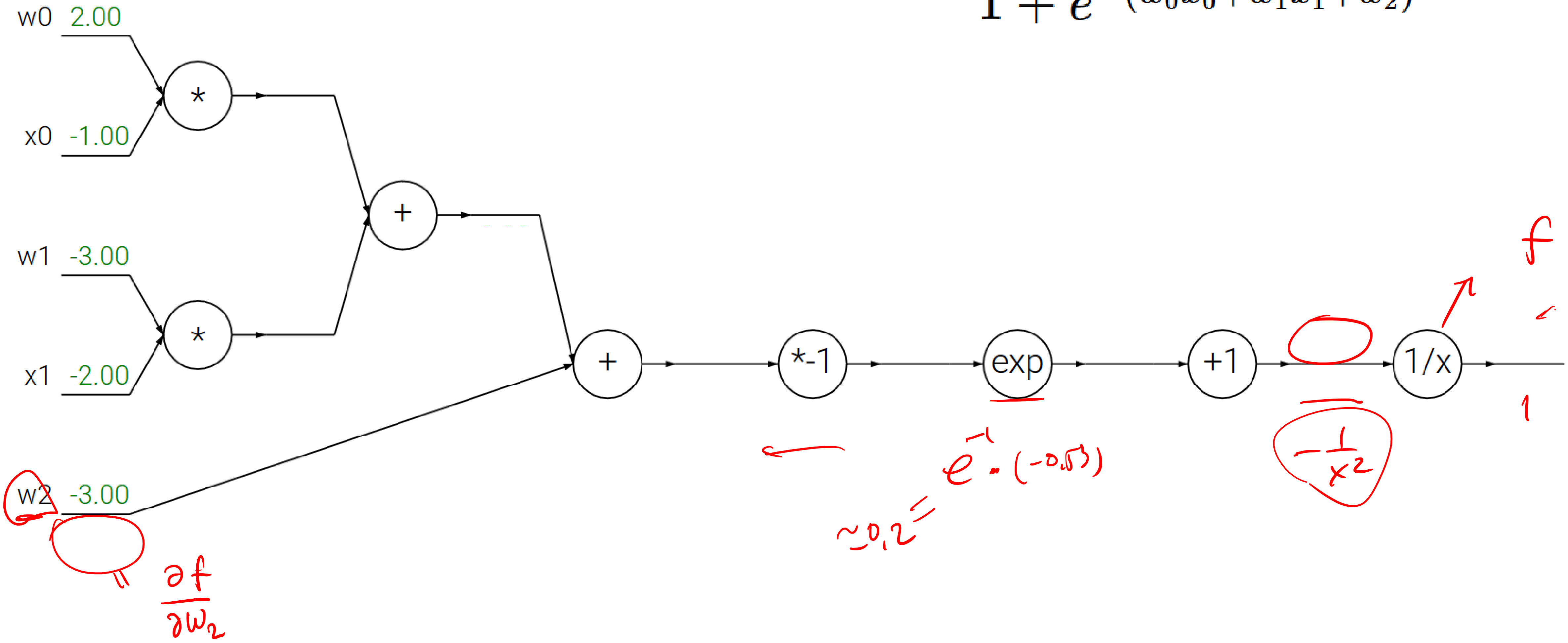


Image borrowed from Stanford cs231n

Back Propagation- Computation Graph

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

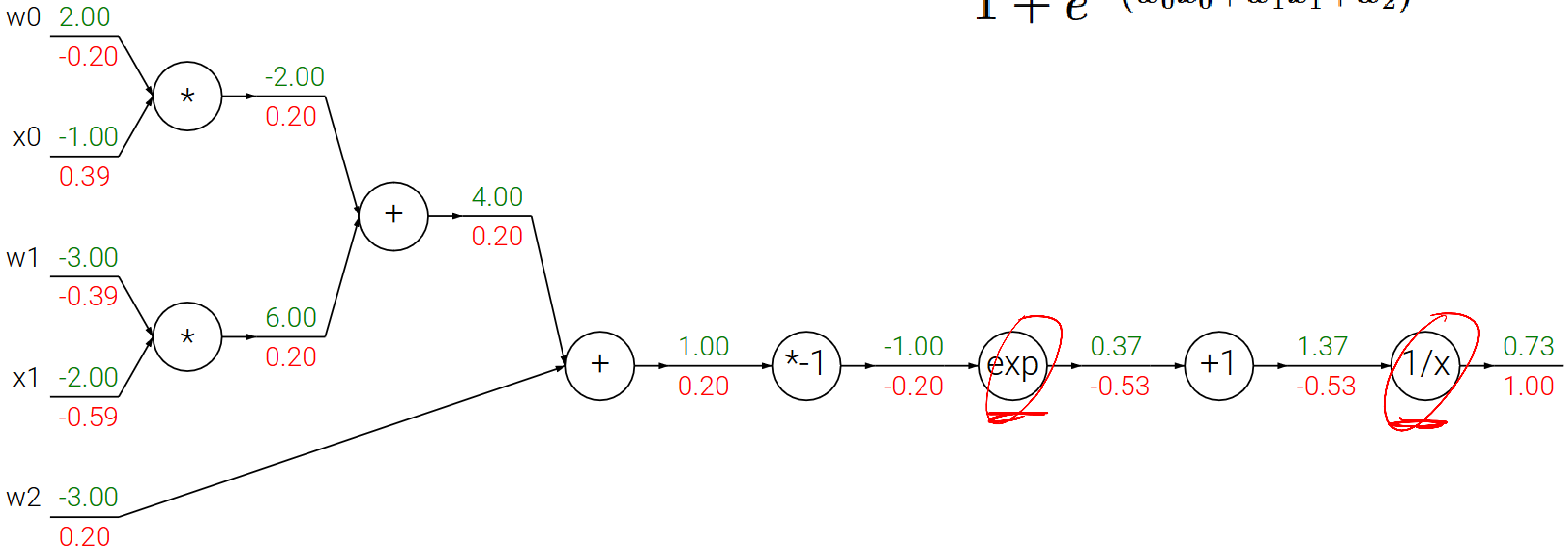


Image borrowed from Stanford cs231n

How does the computer perform differentiation?

Automatic Differentiation (Autodiff)

$f(x,y)\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$

```
defvjp(anp.add,         lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, g))
defvjp(anp.multiply,    lambda g, ans, x, y : unbroadcast(x, y * g),
                        lambda g, ans, x, y : unbroadcast(y, x * g))
defvjp(anp.subtract,    lambda g, ans, x, y : unbroadcast(x, g),
                        lambda g, ans, x, y : unbroadcast(y, -g))
defvjp(anp.divide,      lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
defvjp(anp.true_divide, lambda g, ans, x, y : unbroadcast(x,   g / y),
                        lambda g, ans, x, y : unbroadcast(y, - g * x / y**2))
defvjp(anp.power,
    lambda g, ans, x, y: unbroadcast(x, g * y * x ** anp.where(y, y - 1, 1.)),
    lambda g, ans, x, y: unbroadcast(y, g * anp.log(replace_zero(x, 1.)) * x ** y))
```

https://github.com/mattjj/autodidact/blob/master/autograd/numpy/numpy_vjps.py

https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec10.pdf

Gradient Descent

Optimization Goal

Find a set of (optimized) weights which minimize the error (or loss function) at the output

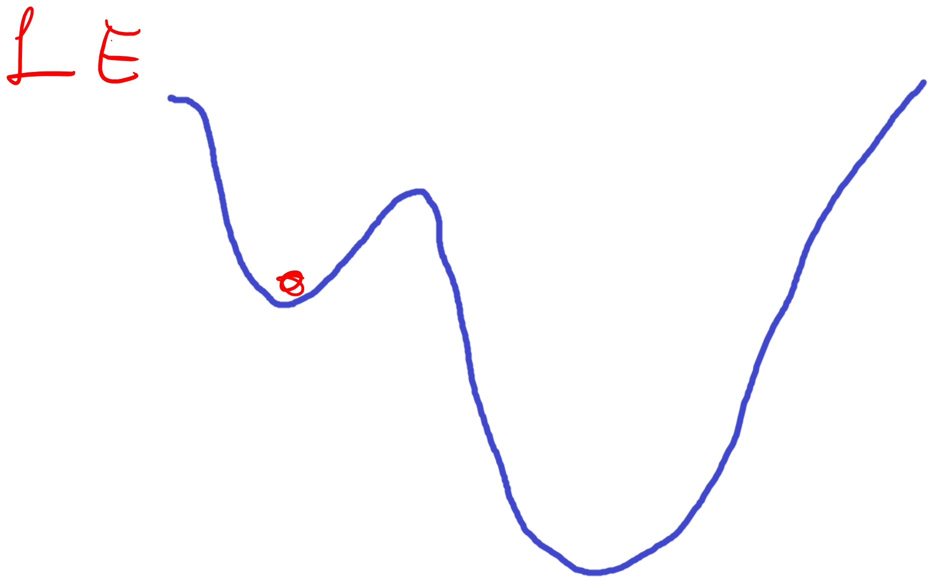
Weight update rule

$$w'^{L}_{nm} \leftarrow w^{L}_{nm} - \alpha * \delta w^{L}_{nm}$$

Δw

$$W_{ij} \leftarrow W_{ij} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}}$$

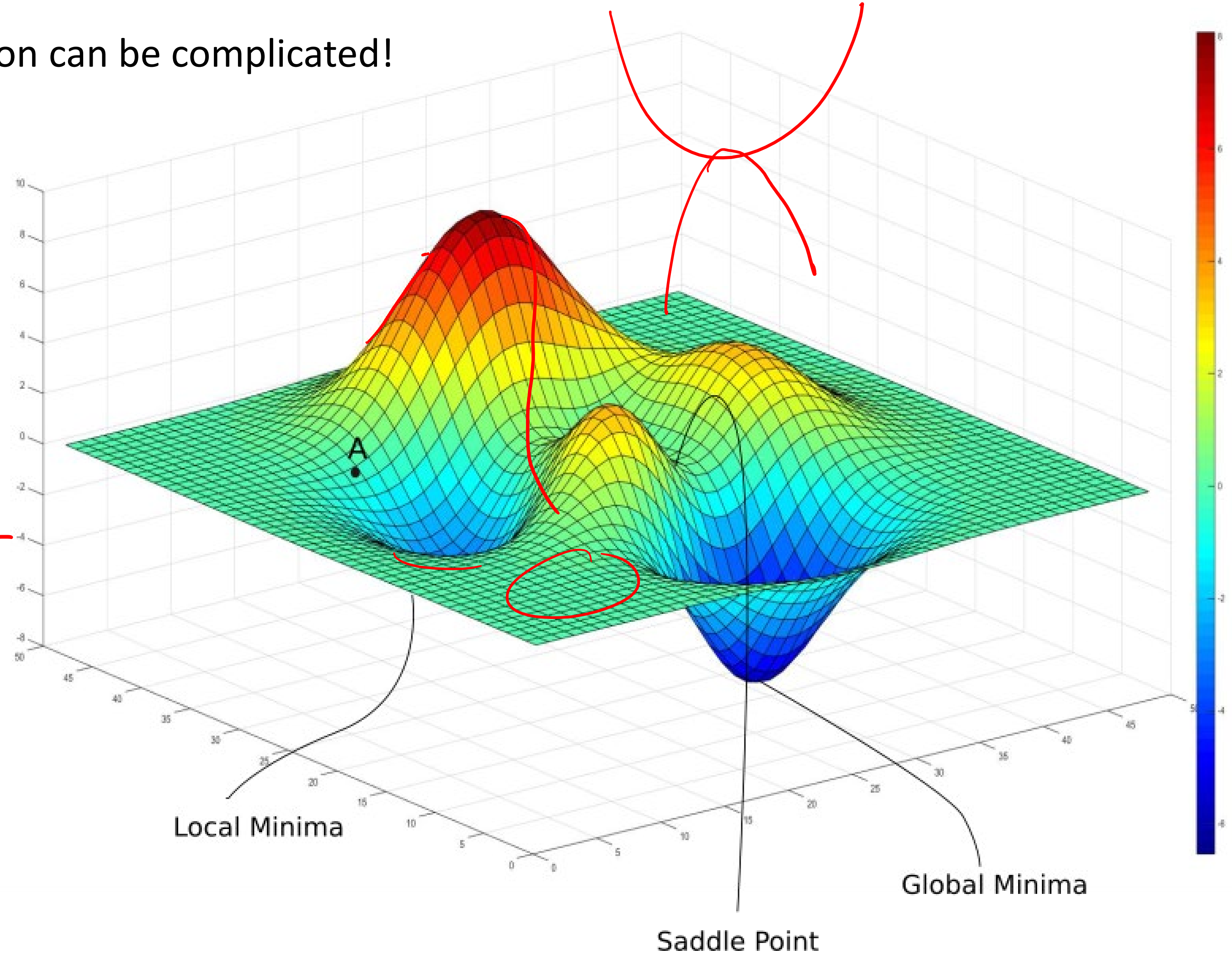
Global minimum vs. local minimum



Gradient Descent

Error surface in the multi dimension can be complicated!

$W \leftarrow W - \alpha \nabla L$



also… cliffs and plateaus.

Stochastic Gradient Descent

How many training samples at a time do we include to calculate the error?

W

$$W_{ij} \leftarrow W_{ij} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}}$$

Practically we use mini batches

1ep

1ep = show 10,000

minibatch bs= 10

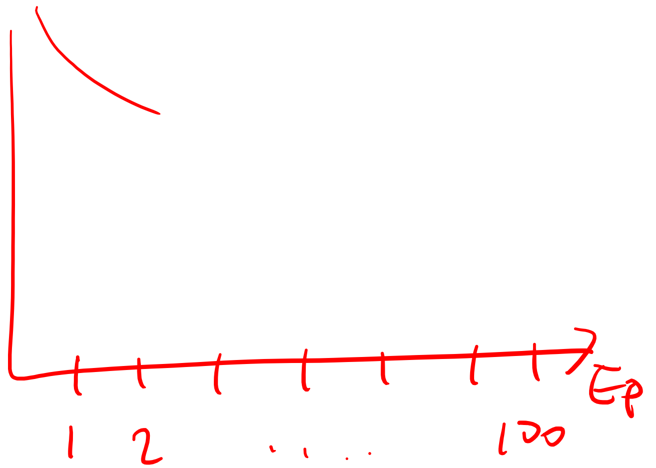
1000 batches

→ 1 step

1000

100

1step



Training speed and accuracy vs. minibatch size

Stochastic Gradient Descent

With decreasing learning rate (Learning rate scheduling)

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

warning- different notations used (from deeplearningbook.org)

Stochastic Gradient Descent with momentum

SGD with learning rate alone is slow to converge

Adding a momentum (moving average) can make it faster

$$v \leftarrow \alpha v - \epsilon \nabla_{\theta} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \right),$$

$$\theta \leftarrow \theta + v.$$

warning- different notations used (from deeplearningbook.org)

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name='SGD', **kwargs
)
```

Next time

- More optimization methods
- Regularization techniques
- Tips for training NN