Neural Network Training Basics

Stephen Baek

Gradient Descent

Motivation

- Typical machine learning tasks:

$$\min_{\boldsymbol{\theta}} \sum_i \left\| y^{(i)} - f\left(\mathbf{x}^{(i)} \middle| \boldsymbol{\theta}\right) \right\|^2$$

  - where…
    - $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$: i-th data point in a dataset.
    - $f(\cdot\,|\boldsymbol{\theta})$: machine learning model with parameters $\boldsymbol{\theta}$.

- Examples:
  - x=[particle size, void fraction, porosity, fluid viscosity, …], y=pearmeability (regression)
  - x=drone image, y=crack/no crack (classification)

So, how do we find $\boldsymbol{\theta}$?

Linear Models

- $\min_{\boldsymbol{\theta}} \sum_i \left\| y^{(i)} - f\left(\mathbf{x}^{(i)} \middle| \boldsymbol{\theta}\right) \right\|^2$ where $f(\mathbf{x}|\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_{d-1} x_{d-1} = \mathbf{x}^{\mathrm{T}} \boldsymbol{\theta}$
- Let $\mathbf{X} \coloneqq \left[\left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}\right]$, and $\mathbf{y} \coloneqq \left[y^{(i)}\right]$, then:

$$\mathcal{L}(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2$$

Linear Models

- Solution:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \\
&= \left(\mathbf{y}^{\mathrm{T}} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\right)(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{y} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta} \\
&= \mathbf{y}^{\mathrm{T}}\mathbf{y} - 2\mathbf{y}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta}
\end{aligned}
$$

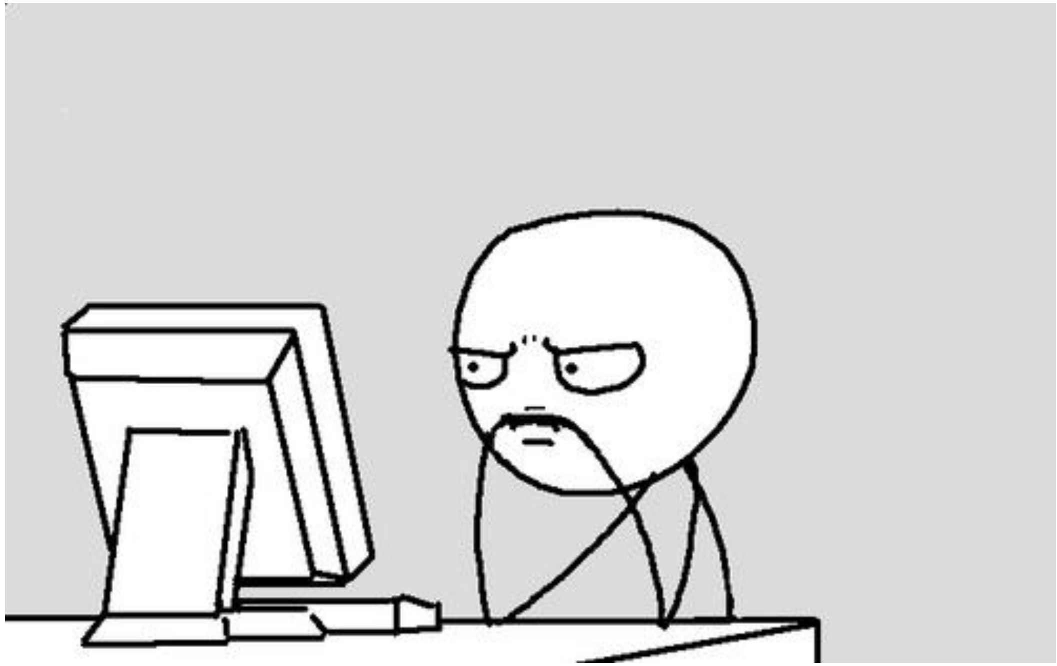
First order necessary condition: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}) = -2\mathbf{y}^{\mathrm{T}}\mathbf{X} + 2\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X} \equiv 0$

$$
\begin{aligned}
&\Leftrightarrow \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{y}^{\mathrm{T}}\mathbf{X} \\
&\Leftrightarrow \mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^{\mathrm{T}}\mathbf{y} \\
&\Leftrightarrow \boldsymbol{\theta} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}
\end{aligned}
$$

A bit of nonlinearity…

- Minimize $f(x) = (\cos x + \tan x)^2$, w.r.t. $x \in (-1,1)$
  - First order necessary condition:

$$\frac{\partial f}{\partial x} = 2(\cos x + \tan x)(-\sin x + \sec^2 x) \equiv 0$$

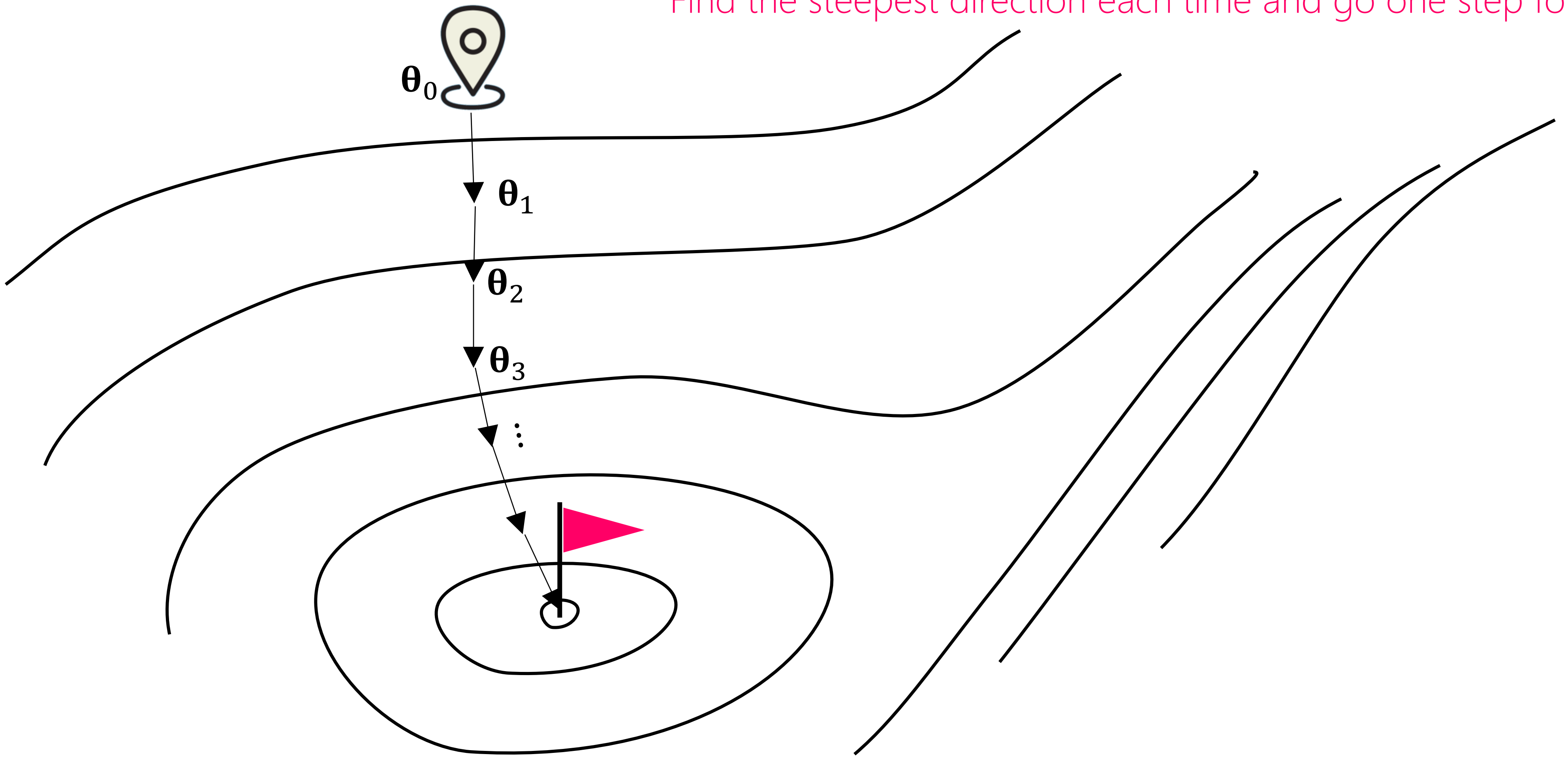


Climbing up a Mountain

- Q. Suppose you're an ***extremely*** near-sighted person (can only see things within, say 6 ft., of your periphery). What would be the best strategy to get to the peak?

A Strategy: Steepest Ascent

"Find the steepest direction each time and go one step forward."

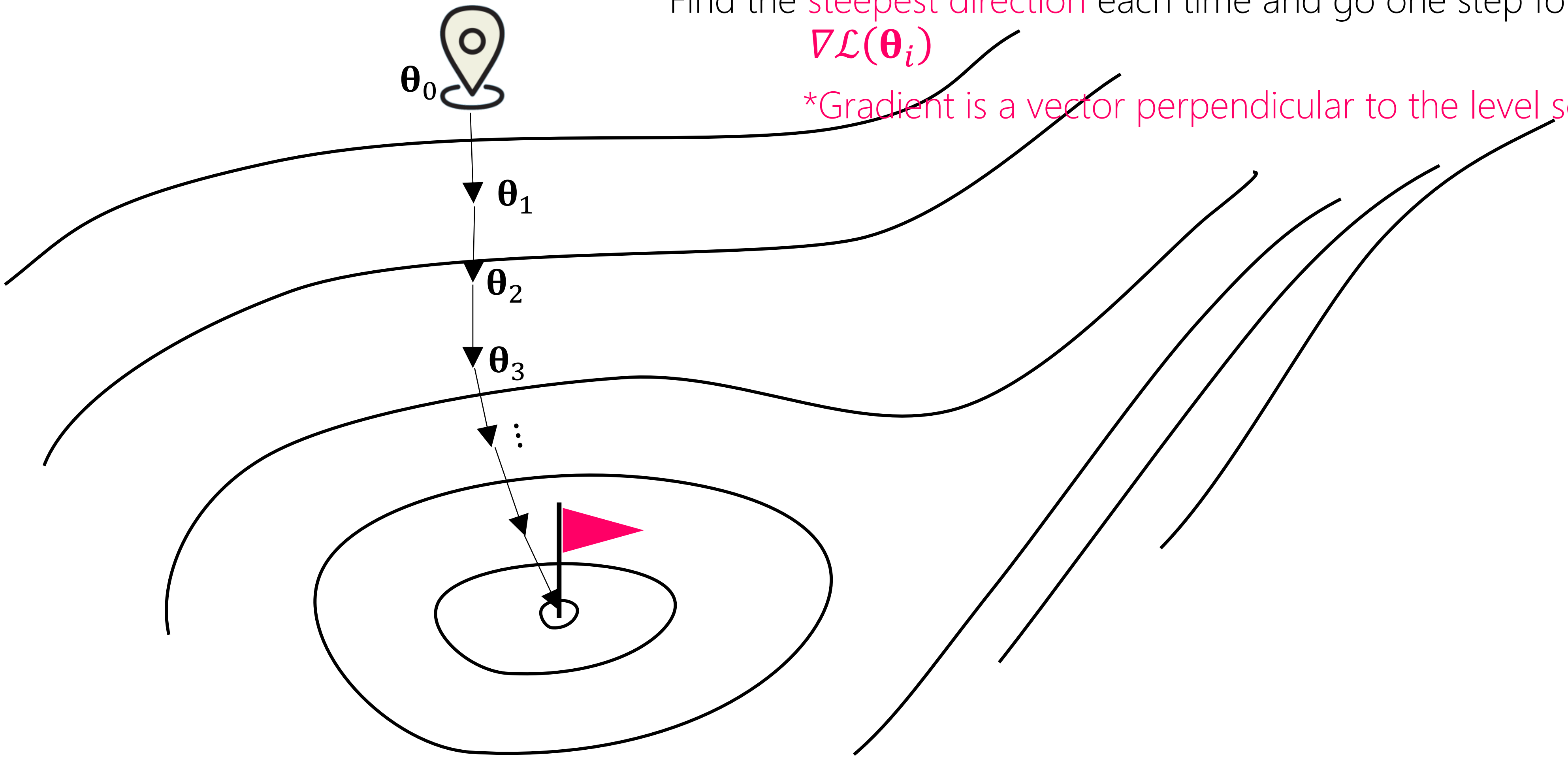


A Strategy: Steepest Ascent

"Find the steepest direction each time and go one step forward."

$\nabla\mathcal{L}(\boldsymbol{\theta}_i)$

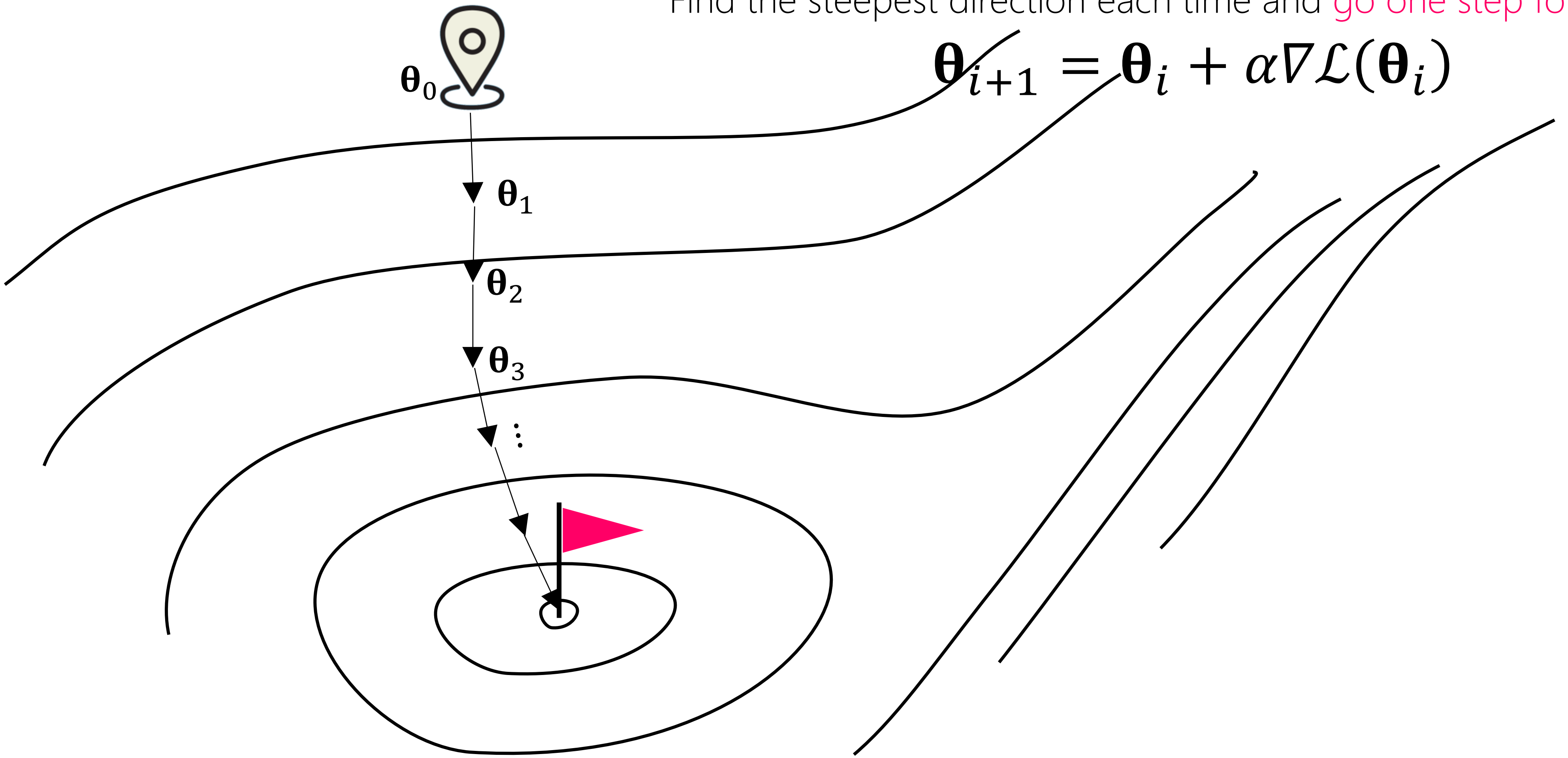
*Gradient is a vector perpendicular to the level set.



A Strategy: Steepest Ascent

"Find the steepest direction each time and go one step forward."

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha \nabla \mathcal{L}(\boldsymbol{\theta}_i)$$



Steepest Descent Algorithm (a.k.a. Gradient Descent)

- Given a differentiable function $\mathcal{L}$, the function value decreases the fastest if one goes in the direction of the negative gradient of $\mathcal{L}$.

- It follows that, for small enough scalar value α, if

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}_i)$$

  then $\mathcal{L}(\boldsymbol{\theta}_{i+1}) \leq \mathcal{L}(\boldsymbol{\theta}_i)$. (Proof: 1st order Taylor approximation)

Machine Learning with Gradient Descent

1. Design your model $f(\mathbf{x}|\boldsymbol{\theta})$, and the learning objective $\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}, y)$.
2. Initialize the model parameters $\boldsymbol{\theta}$ (usually with random numbers).
3. Evaluate the gradient $\nabla\mathcal{L}$ with respect to the current model parameters $\boldsymbol{\theta}$ and training dataset $\{\mathbf{x}^{(i)}, y^{(i)}\}$.
4. Improve the model parameters with a given learning rate α and the update strategy: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\nabla\mathcal{L}$.
5. Repeat 3~4 until converges

Thought Experiment on Gradient Descent

Q. What does it take to evaluate loss?

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \mathbf{y}^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) \right\|^2$$

Thought Experiment on Gradient Descent

Q. What does it take to evaluate loss?

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \mathbf{y}^{(i)} - f\left(\mathbf{x}^{(i)} | \, \boldsymbol{\theta}\right) \right\|^2$$

A. The function value $f\left(\mathbf{x}^{(i)} | \, \boldsymbol{\theta}\right)$ need to be evaluated for all $\mathbf{x}^{(i)}$ in the dataset.

Thought Experiment on Gradient Descent

Q. What does it take to evaluate loss?

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \mathbf{y}^{(i)} - f\left(\mathbf{x}^{(i)} \,|\, \boldsymbol{\theta}\right) \right\|^2$$

A. The function value $f\left(\mathbf{x}^{(i)} \,|\, \boldsymbol{\theta}\right)$ need to be evaluated for all $\mathbf{x}^{(i)}$ in the dataset.

Q. What about the gradient?

Thought Experiment on Gradient Descent

Q. What does it take to evaluate loss?

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \mathbf{y}^{(i)} - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) \right\|^2$$

A. The function value $f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)$ need to be evaluated for all $\mathbf{x}^{(i)}$ in the dataset.

Q. What about the gradient?

A. Same! Backpropagation needs to happen for all and each $\mathbf{x}^{(i)}$

Thought Experiment on Gradient Descent

Q. What does it take to evaluate loss?

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \mathbf{y}^{(i)} - f\left(\mathbf{x}^{(i)} | \, \boldsymbol{\theta}\right) \right\|^2$$

A. The function value $f\left(\mathbf{x}^{(i)} | \, \boldsymbol{\theta}\right)$ need to be evaluated for all $\mathbf{x}^{(i)}$ in the dataset.

Q. What about the gradient?

A. Same! Backpropagation needs to happen for all and each $\mathbf{x}^{(i)}$

Q. Then what happens to the gradient descent algorithm?

Thought Experiment on Gradient Descent

Q. What does it take to evaluate loss?

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| \mathbf{y}^{(i)} - f\left(\mathbf{x}^{(i)} | \, \boldsymbol{\theta}\right) \right\|^2$$

A. The function value $f\left(\mathbf{x}^{(i)} | \, \boldsymbol{\theta}\right)$ need to be evaluated for all $\mathbf{x}^{(i)}$ in the dataset.

Q. What about the gradient?

A. Same! Backpropagation needs to happen for all and each $\mathbf{x}^{(i)}$

Q. Then what happens to the gradient descent algorithm?

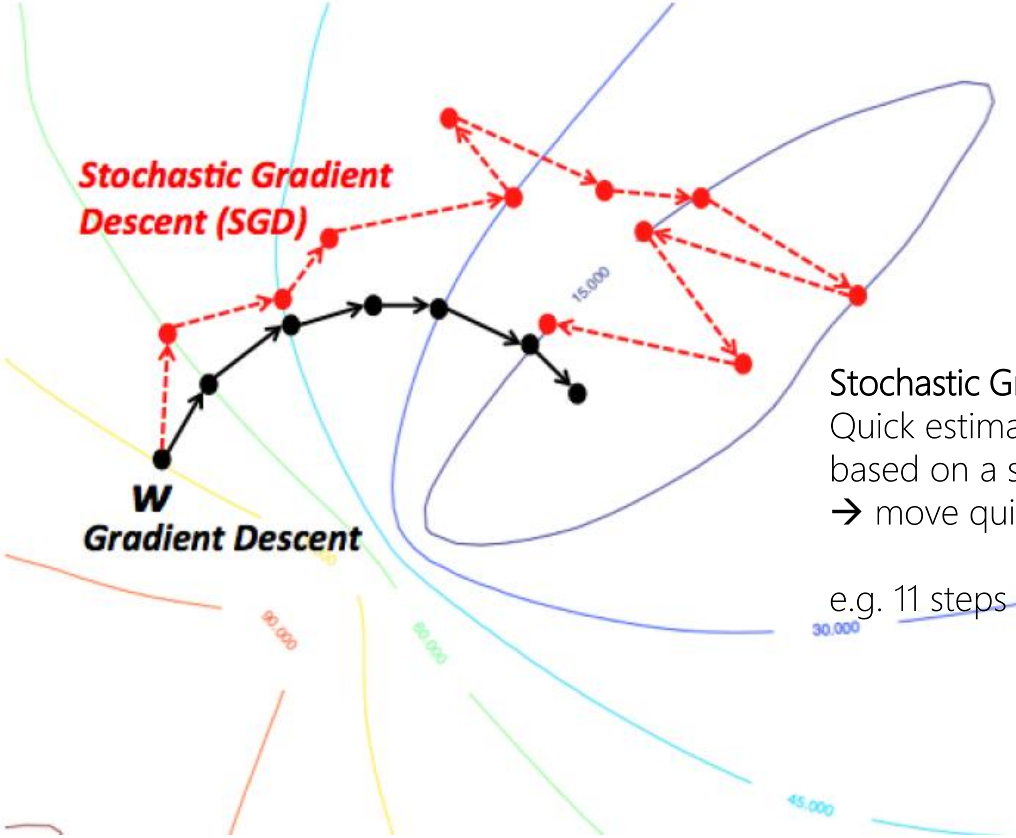
A. Computational time increases exponentially as N goes up.

Stochastic Gradient Descent (SGD)

- Idea:

Gradient Descent
Compute everything
→ make the optimal one step

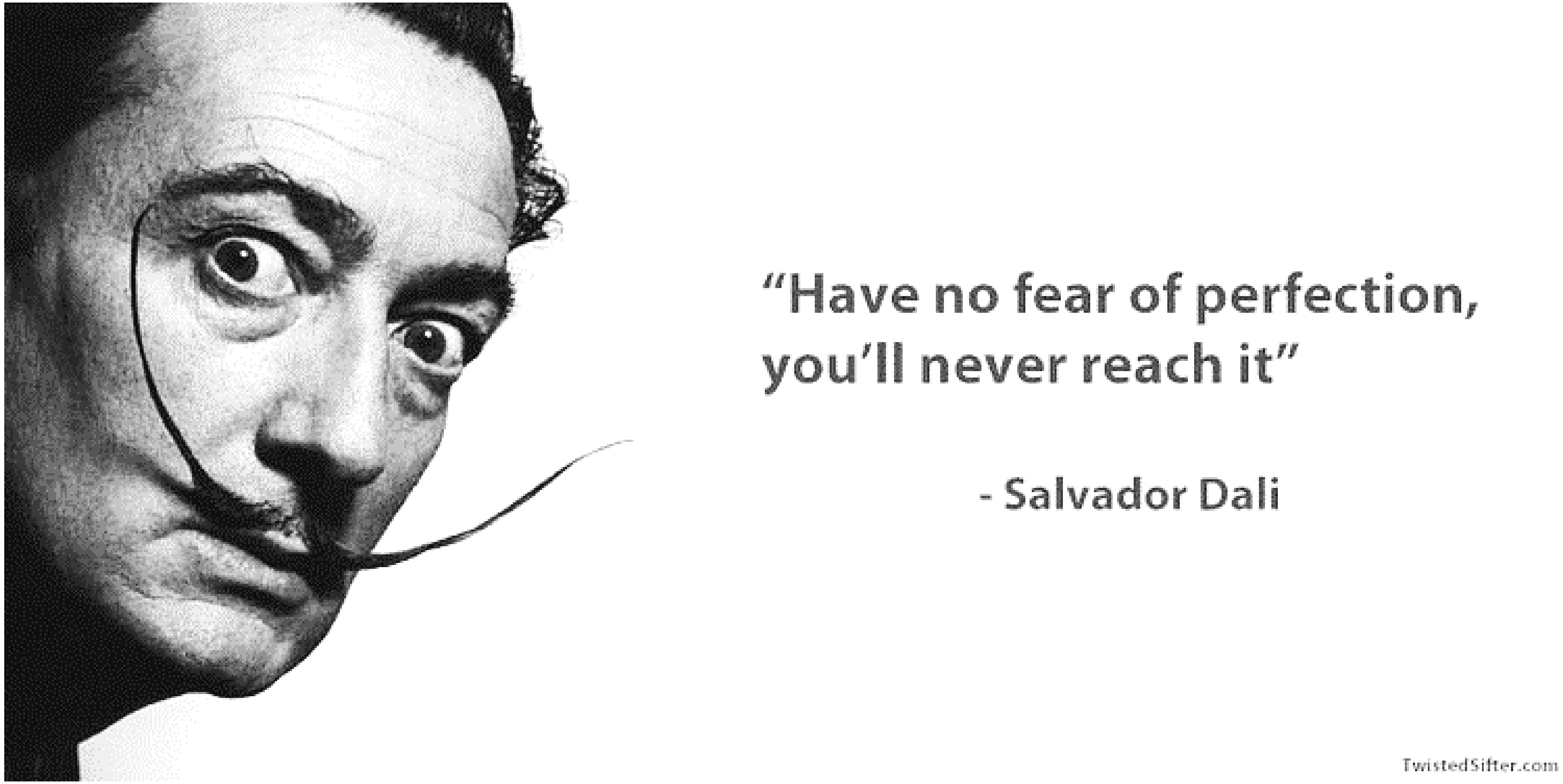
e.g. 6 steps * 1 hr/step = 6 hrs



Stochastic Gradient Descent
Quick estimation of the gradient based on a small batch
→ move quickly even if it's not the optimal

e.g. 11 steps * 5 min/step = 55 min

Stochastic Gradient Descent (SGD)



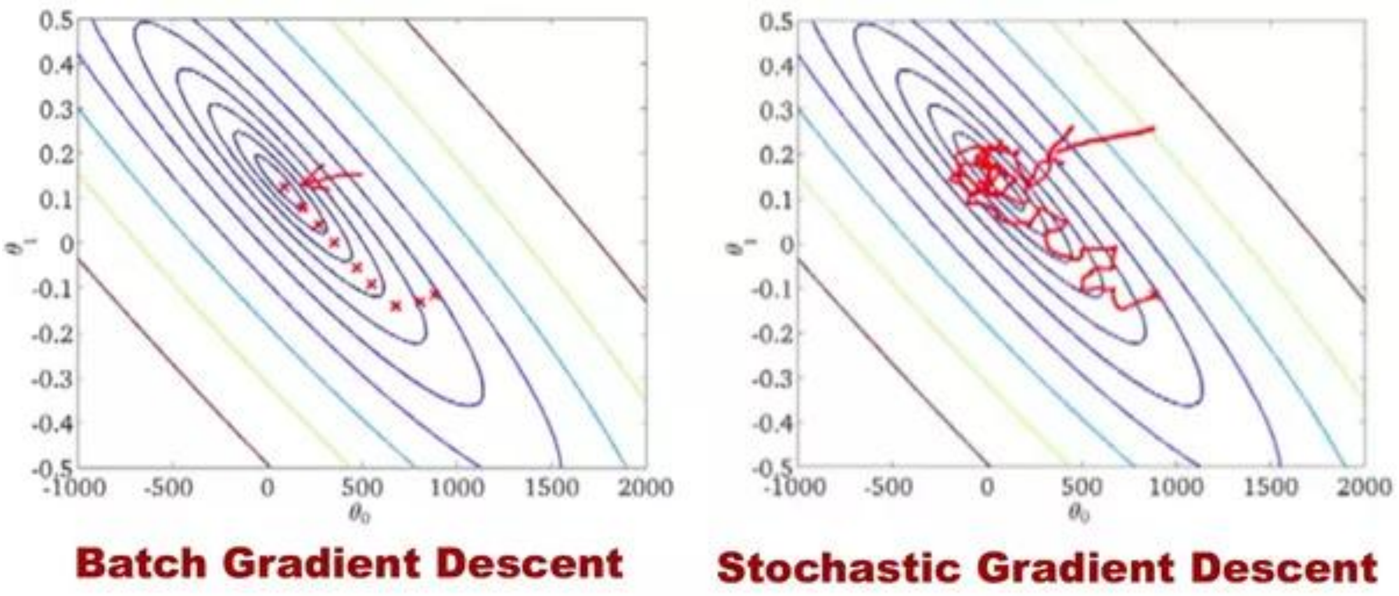
Stochastic Gradient Descent (SGD)

1. Randomly shuffle dataset
2. Repeat until converge {

```
    for a mini-batch {
            compute gradient only with the mini-batch
            update weights
    }
}
```

Stochastic Gradient Descent (SGD)

- Gradients come from mini-batches, so they can be noisy and inaccurate!

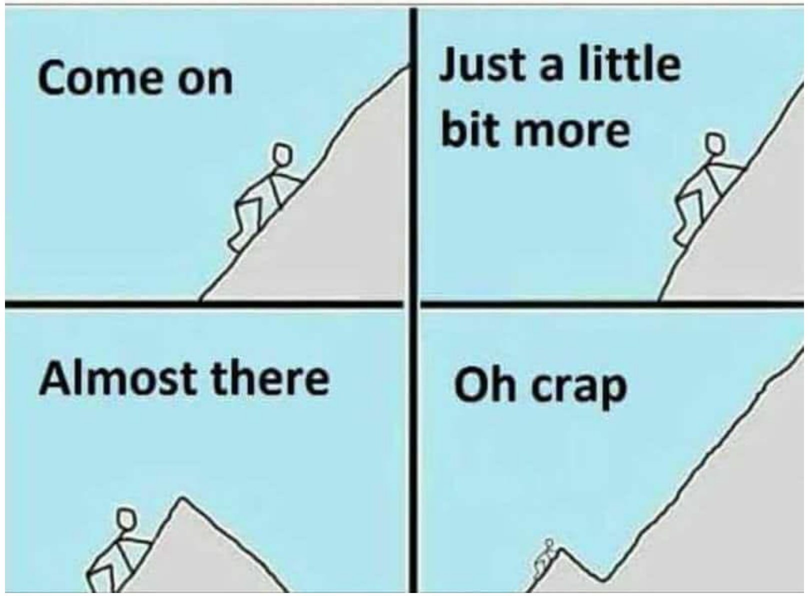


ML with Stochastic Gradient Descent

1. Design your model $f(\mathbf{x}|\boldsymbol{\theta})$, and the learning objective $\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}, y)$.
2. Initialize the model parameters $\boldsymbol{\theta}$ (usually with random numbers).
3. Randomly sample a batch $B = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N_b}$ with the batch size $N_b \ll N$.
4. Approximate the gradient $\nabla\mathcal{L}$ with respect to the current model parameters $\boldsymbol{\theta}$ and the current batch B.
5. Improve the model parameters with a given learning rate α and the update strategy: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\nabla\mathcal{L}$.
6. Repeat 3~5 until all samples in the training data set is consumed. ("Epoch")
7. Repeat 6 until converges

Problems of Vanilla (S)GD

- Local minima or Saddle points → zero gradient! → No update (gets stuck)



Momentum

- Idea: let’s build up a velocity (momentum)!
- SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

- SGD + Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

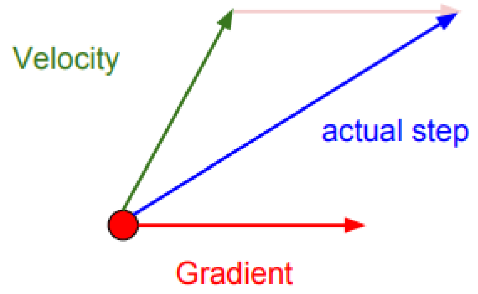
  - ρ: “friction” or “drag”. Causes decrease of velocity. Typically 0.9 or 0.99

SGD + Momentum

- Discuss:
  - High condition number (long-narrow valley)
  - Local minima and saddle points
  - Noisy gradient

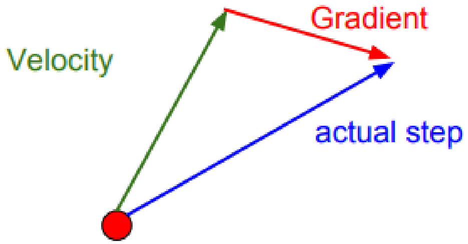
Nesterov Momentum

- Vanilla momentum method: Current gradient + Current velocity.



$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

- Nesterov Version oint where the current velocity would take us. Take the gradient there and perform the update.



☹

$$v_{t+1} = \rho v_t - \alpha \nabla f(\boxed{x_t + \rho v_t})$$
$$x_{t+1} = x_t + v_{t+1}$$

Nesterov Momentum

- $v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t),\ x_{t+1} = x_t + v_{t+1}$

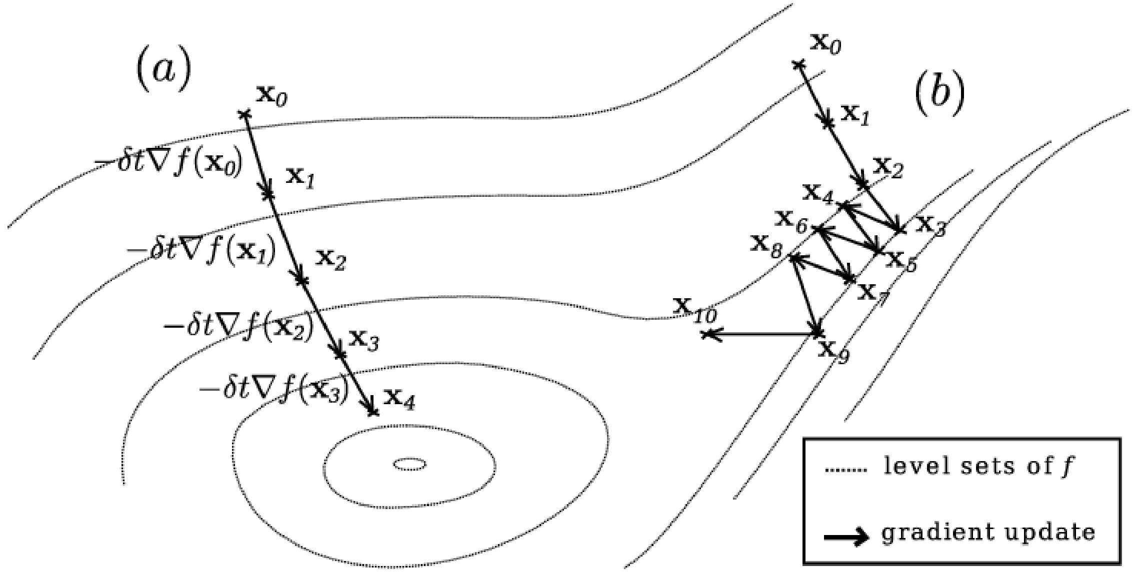
- We want to update in terms of x_t and $\nabla f(x_t)$, NOT $\nabla f(x_t + \rho v_t)$.

- Luckily, this can be rearranged by the change of variables: $\tilde{x}_t = x_t + \rho v_t$

$$
\begin{aligned}
v_{t+1} &= \rho v_t - \alpha \nabla f(\tilde{x}_t) \\
\tilde{x}_{t+1} &= \tilde{x}_t - \rho v_t + v_{t+1} + \rho v_{t+1} \\
&= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)
\end{aligned}
$$

Another issue with GD: Long Narrow Valley

- What if f happens to be steep in one direction but "flat" in the other directions? (Long narrow valley)
  - Condition number: ratio of largest to smallest singular value of the Hessian.
  - Large condition number → long-narrow valley



AdaGrad

- Perform element-wise scaling of the gradient
  - Scale factors determined based on the historical sum of squares…

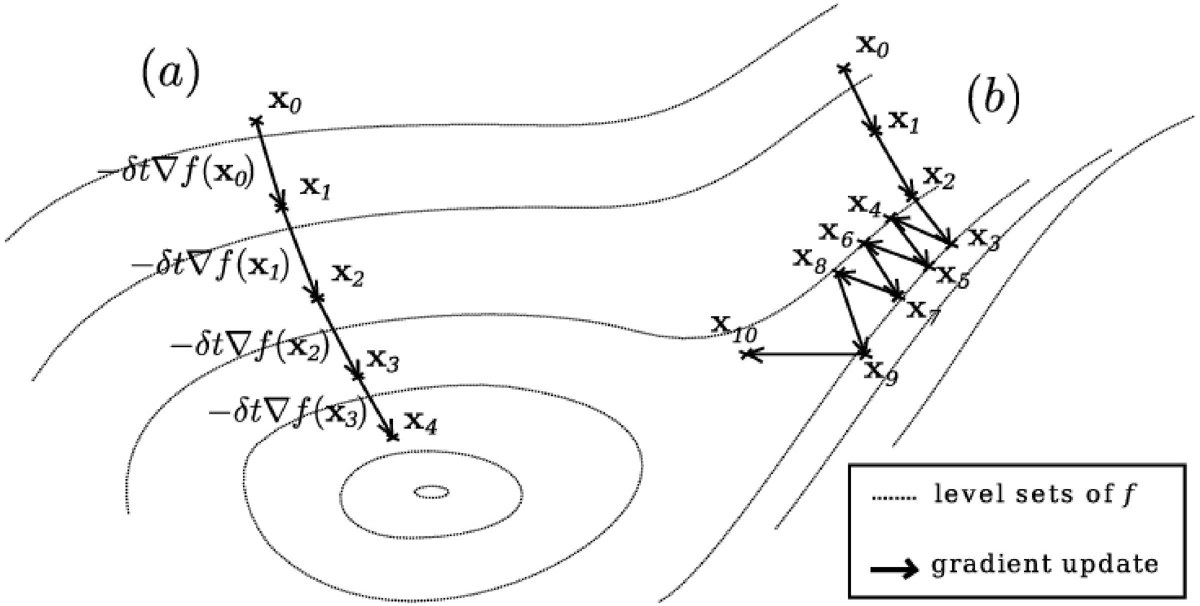
```
scale_factor = 0

for iter in range(0, MAX_ITER):
    dx = backpropagate(x)   # compute gradient
    scale_factor += dx*dx
    x -= learning_rate * dx / (np.sqrt(scale_factor) + epsilon)
```

- The element-wise scaling has an effect of "per-parameter learning rates" or "adaptive learning rates," thus, the name Adaptive Gradient.

AdaGrad

- Long narrow valley: what happens with AdaGrad?
  - Step size along steep directions will be damped.
  - Step size along flat directions will be accelerated.



AdaGrad

- Historical sum: what happens with AdaGrad after many iterations?
  - Step size decays to zero… ☹

```
scale_factor = 0

for iter in range(0, MAX_ITER):
    dx = backpropagate(x)   # compute gradient
    scale_factor += dx*dx
    x -= learning_rate * dx / (np.sqrt(scale_factor) + epsilon)
```

RMSProp

- AdaGrad (step size decays to zero):

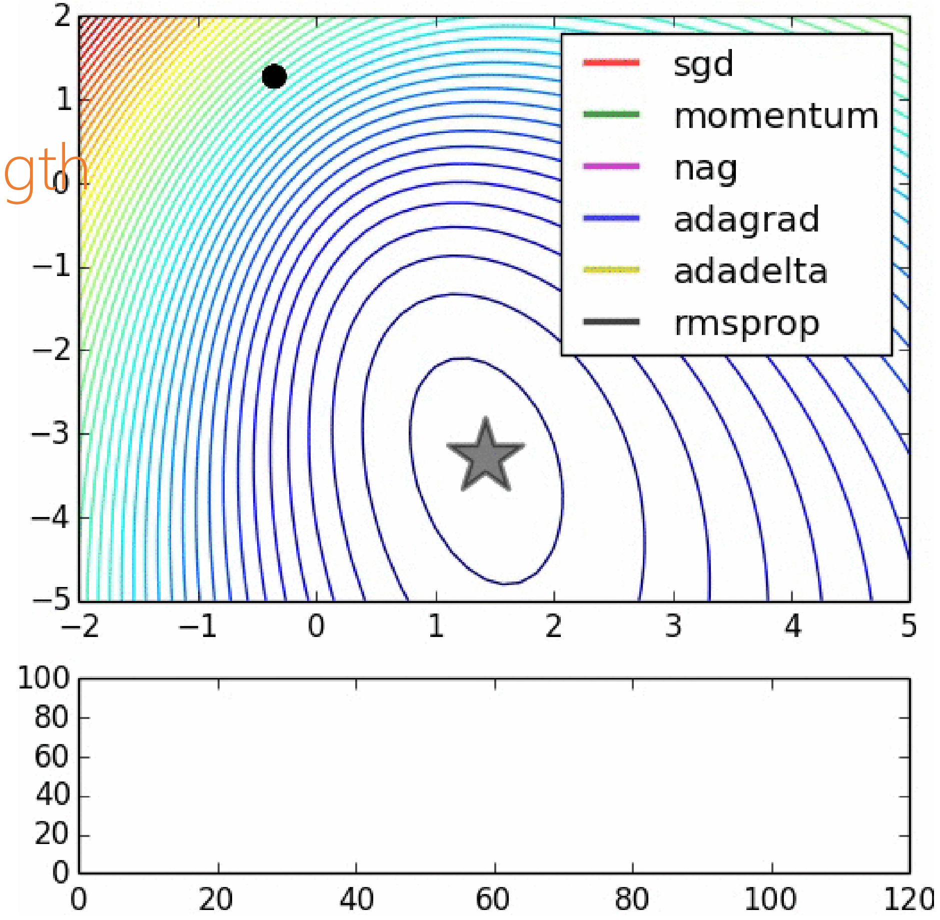
```
scale_factor = 0
for iter in range(0, MAX_ITER):
    dx = backpropagate(x)   # compute gradient
    scale_factor += dx*dx
    x -= learning_rate * dx / (np.sqrt(scale_factor) + epsilon)
```

- RMSProp (problem solved ☺)

```
scale_factor = 0
for iter in range(0, MAX_ITER):
    dx = backpropagate(x)   # compute gradient
    scale_factor = decay_rate*scale_factor + (1-decay_rate)*dx*dx
    x -= learning_rate * dx / (np.sqrt(scale_factor) + epsilon)
```

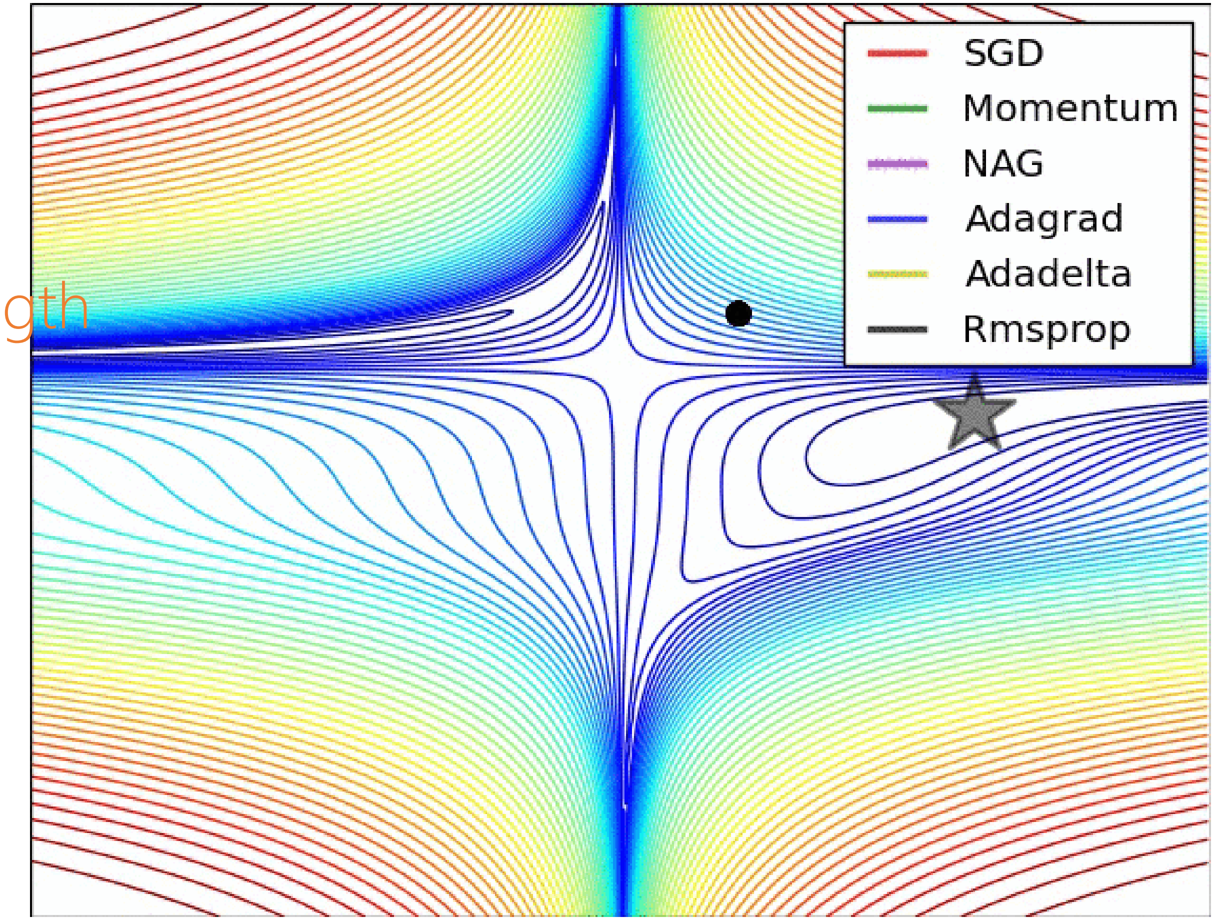
Comparison of Optimization Methods

*Fixed step length



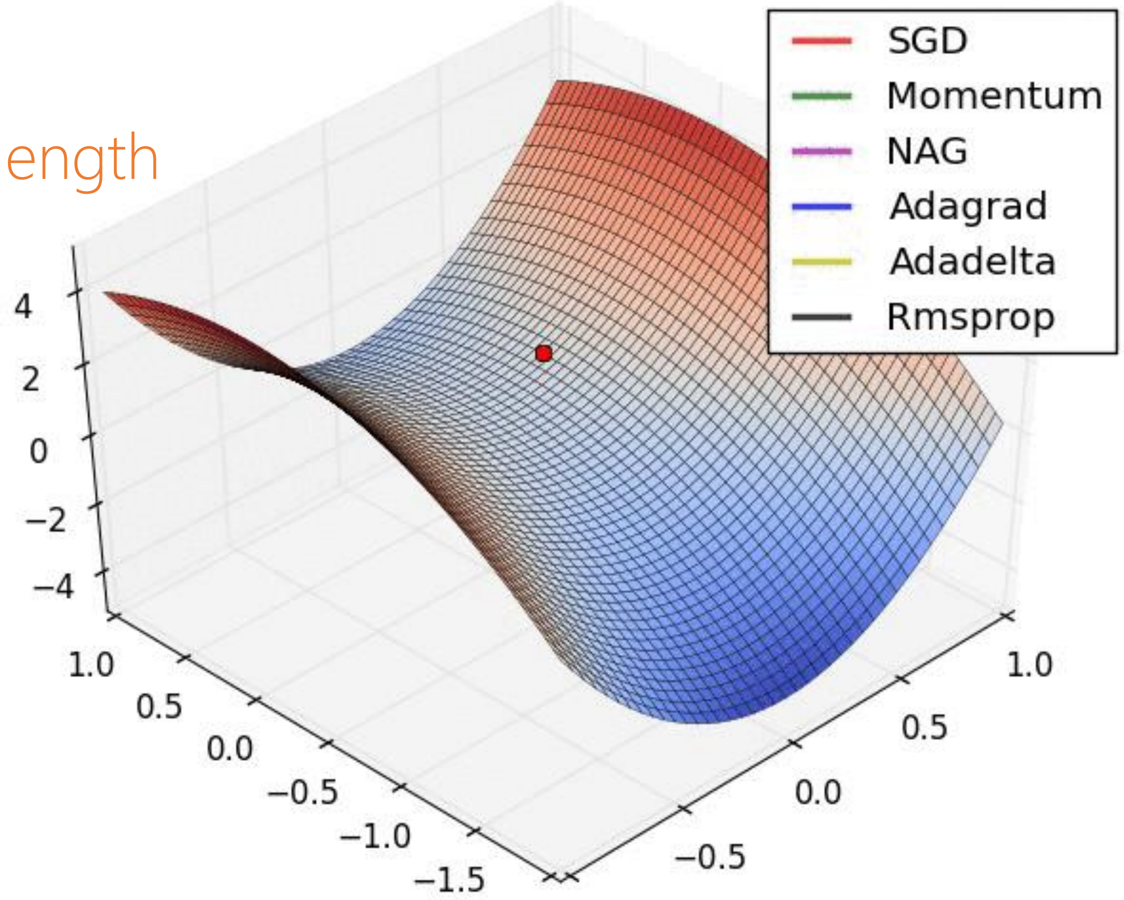
Comparison of Optimization Methods

*Fixed step length



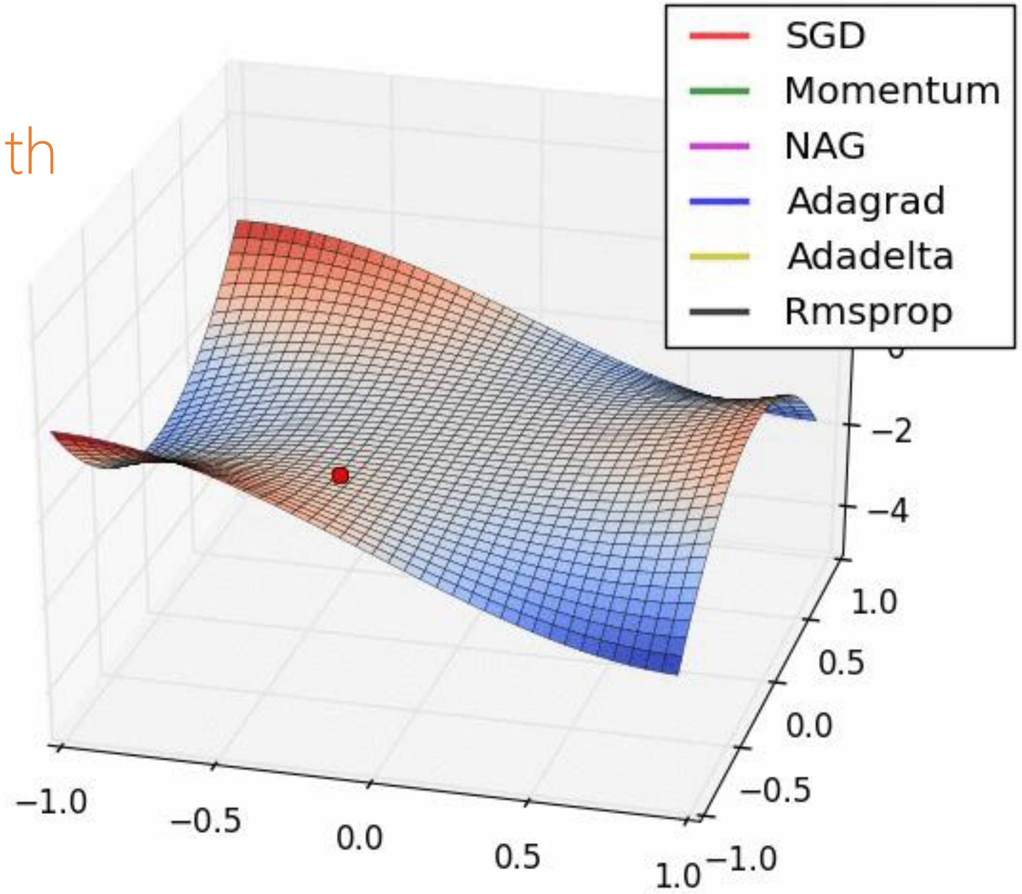
Comparison of Optimization Methods

*Fixed step length



Comparison of Optimization Methods

*Fixed step length



Adam (All of the above!)

- Why not take the advantage of both momentum and adaptive gradient methods?

```
moment = [0, 0]
for iter in range(0, MAX_ITER):
    dx = backpropagate(x)   # compute gradient
    moment[0] = beta[0]*moment[0] + (1-beta[0])*dx                # momentum
    moment[1] = beta[1]*moment[1] + (1-beta[1])*dx*dx             # RMSProp
    x -= learning_rate * moment[0] / (np.sqrt(moment[1]) + epsilon)
```

- Problem with the idea: what happens when iter = 0?
  - moments = 0 → bias!

Adam (All of the above!)

```
moment = [0, 0]
for iter in range(0, MAX_ITER):
    dx = backpropagate(x)   # compute gradient
    moment[0] = beta[0]*moment[0] + (1-beta[0])*dx              # momentum
    moment[1] = beta[1]*moment[1] + (1-beta[1])*dx*dx           # RMSProp
    x -= learning_rate * moment[0] / (np.sqrt(moment[1]) + epsilon)
```

- Modified version:

In typical scenarios, a good starting point:

- beta = some big number close to 1 (e.g. 0.9, 0.99)
- learning_rate = 1e-3

```
moment = [0, 0]
for iter in range(0, MAX_ITER):
    dx = backpropagate(x)   # compute gradient
    moment[0] = beta[0]*moment[0] + (1-beta[0])*dx              # momentum
    moment[1] = beta[1]*moment[1] + (1-beta[1])*dx*dx           # RMSProp
    unbiased[0] = moment[0] / (1 – beta[0]**iter)               # bias correction
    unbiased[1] = moment[1] / (1 – beta[1]**iter)
    x -= learning_rate * unbiased[0] / (np.sqrt(unbiased[1]) + epsilon)
```

History of Gradient Descent Optimizers

GD
Use all the data to evaluate the gradient
and make the optimal step for every iteration

SGD
Approximate the gradient
only with a small portion of data
and move more in a given amount of time

Momentum
Move a step forward and then
go a little further following the momentum

Nesterov Accelerated Gradient (NAG)
It is faster to move toward the momentum
and to compute the step on a new location

Adagrad
Make large steps at places already visited,
make smaller steps near new places

RMSProp
Make the step length decision
depending on the context

AdaDelta
Prevent "stop" because of too small steps

NADAM
RMSProp + NAG

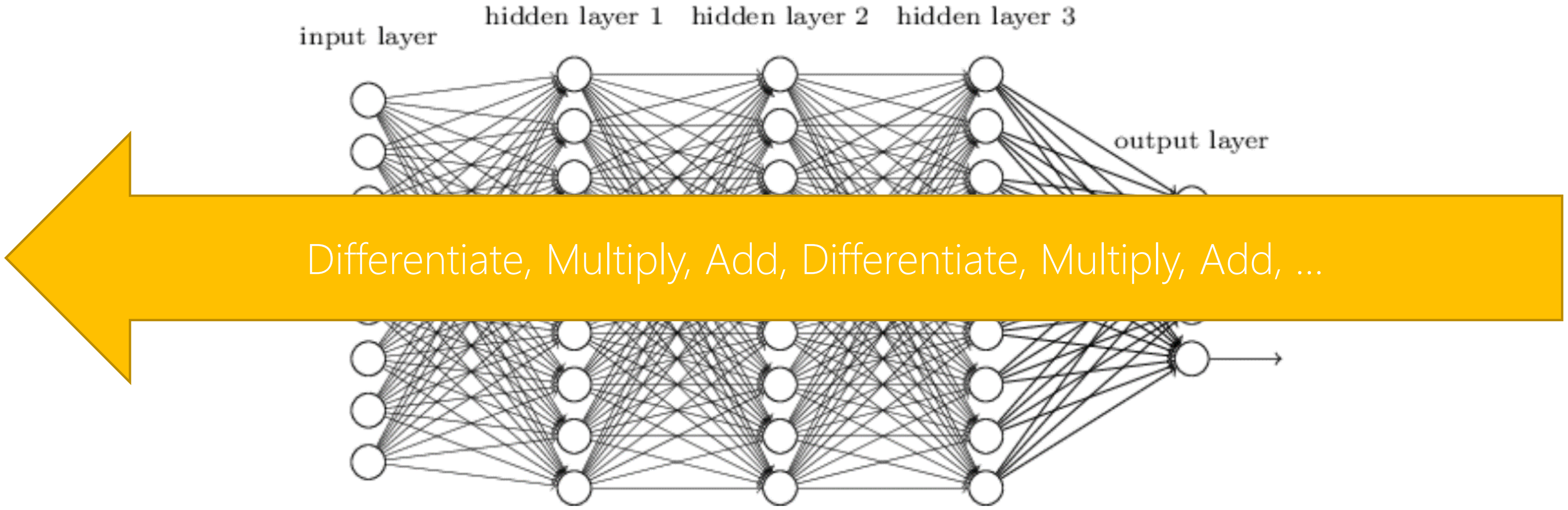
ADAM
RMSProp + Momentum

If you have no idea: ADAM!

Backpropagation

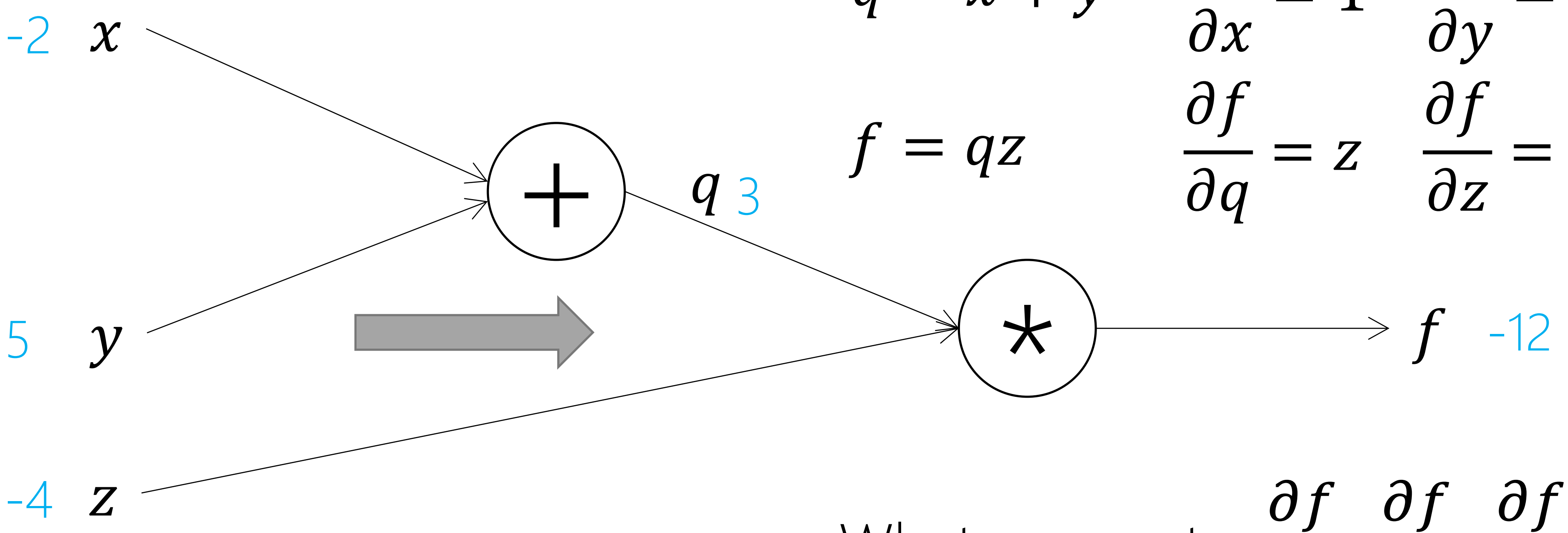
Backpropagation

- Rumelhart, Hinton, and Williams. (1986).
- A popular training method for neural nets
- Propagate what? "the gradient of the current error"



Backpropagation

- A simple example: $f(x, y, z) = (x + y)z$
  - Computational graph:



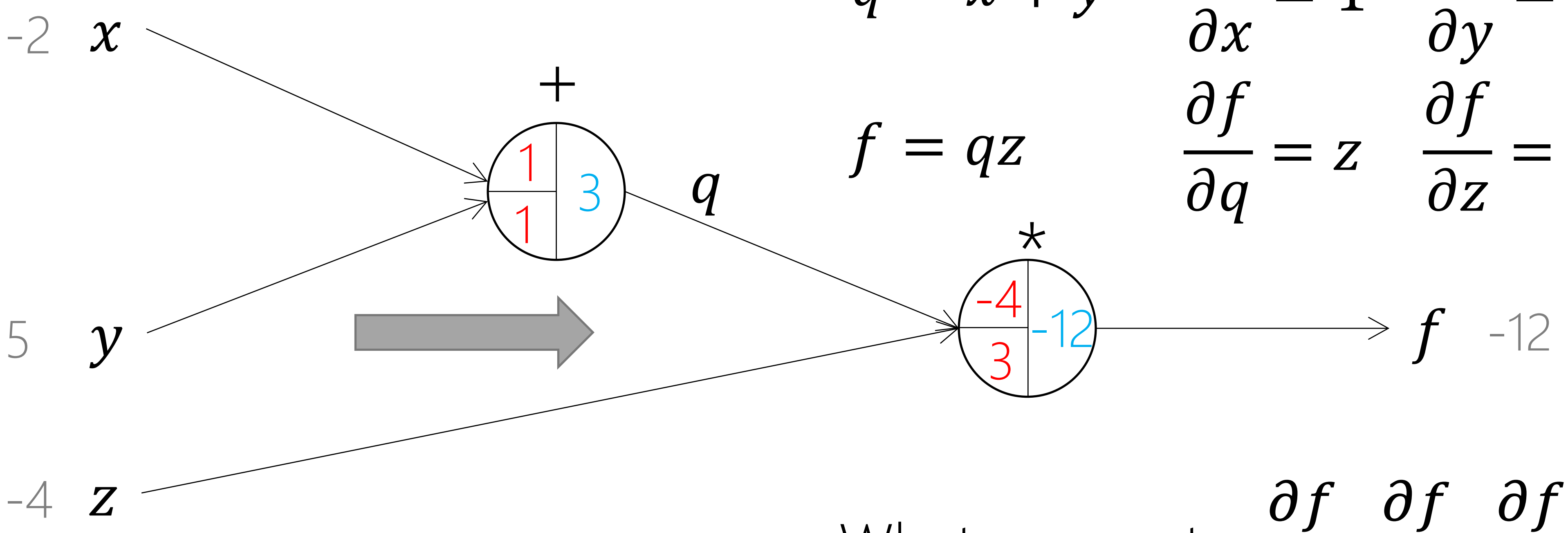
$q = x + y \quad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$

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

What we want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation

- A simple example: $f(x, y, z) = (x + y)z$
  - Computational graph:



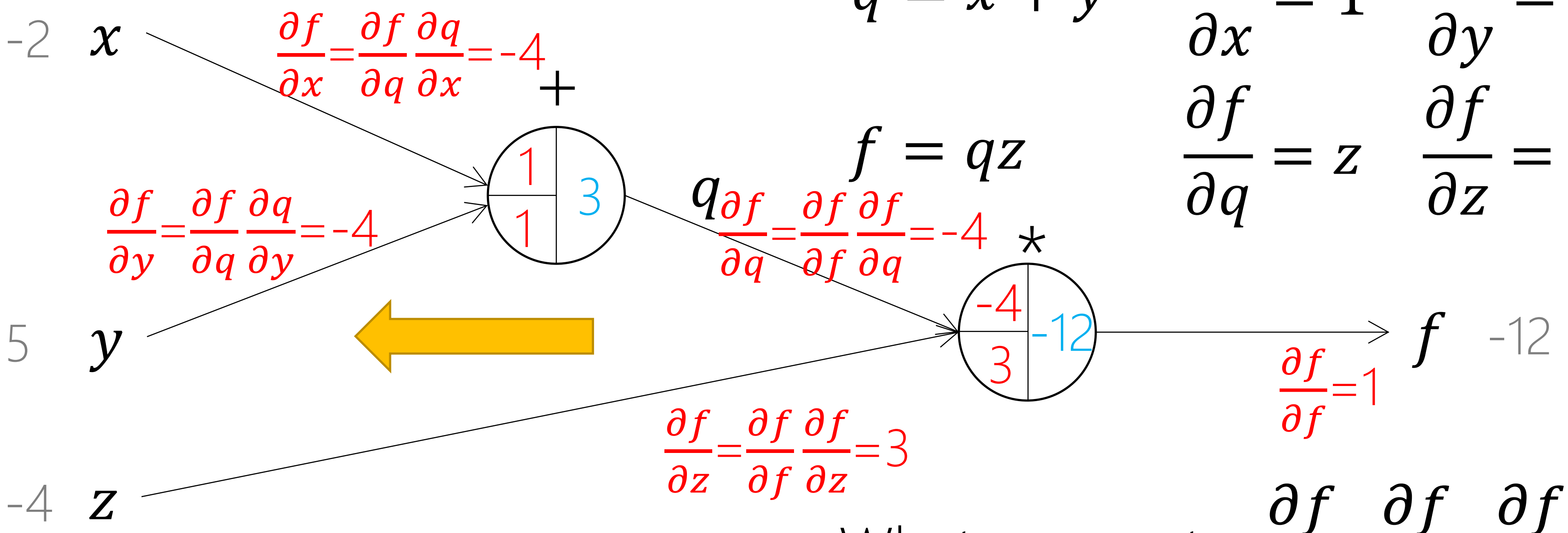
$q = x + y \quad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$

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

What we want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation

- A simple example: $f(x,y,z) = (x+y)z$
  - Computational graph:

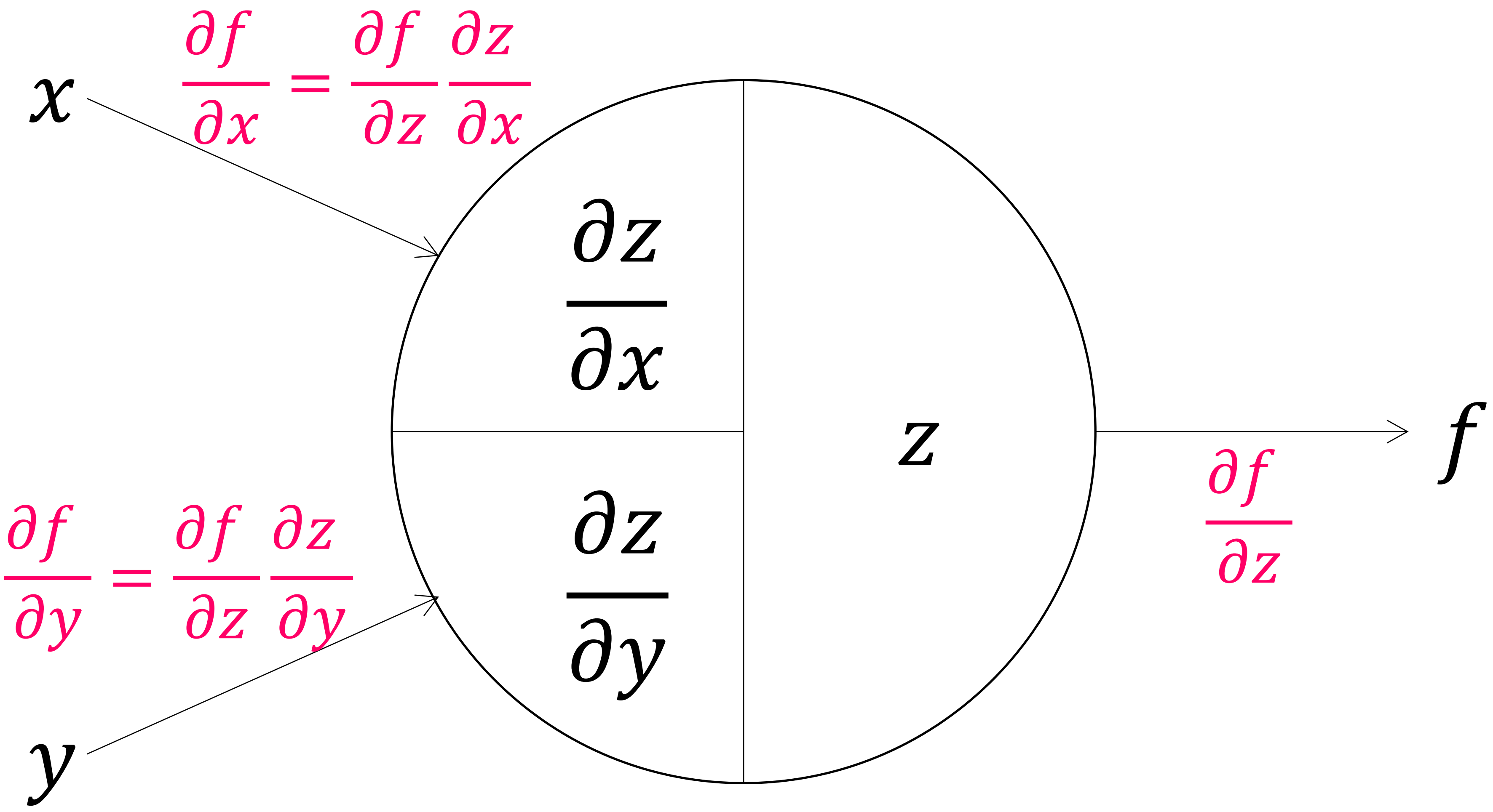


$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$

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

What we want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

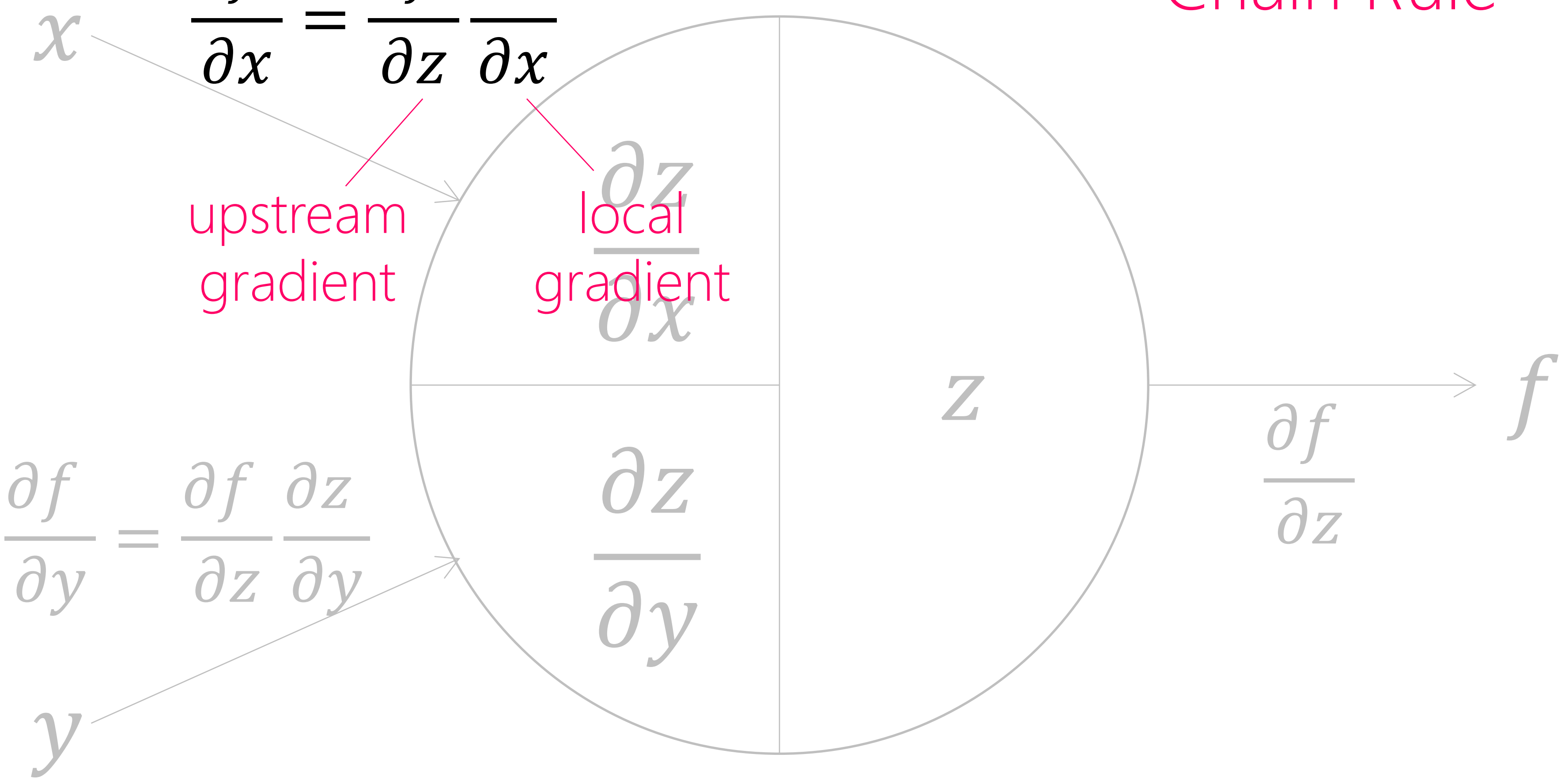
Backpropagation



Backpropagation

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

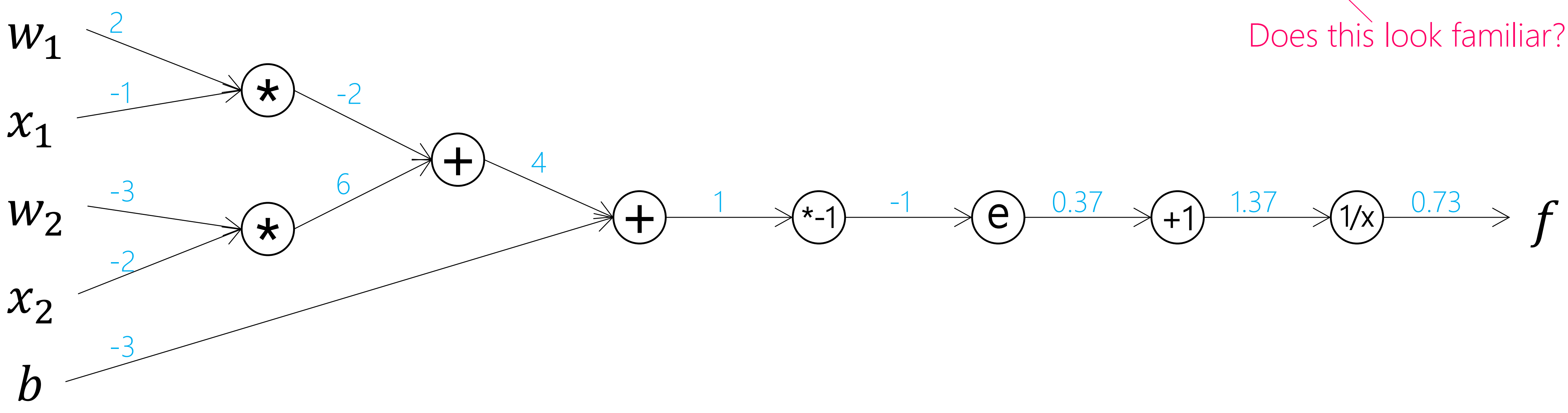
“Chain Rule”



Another Example

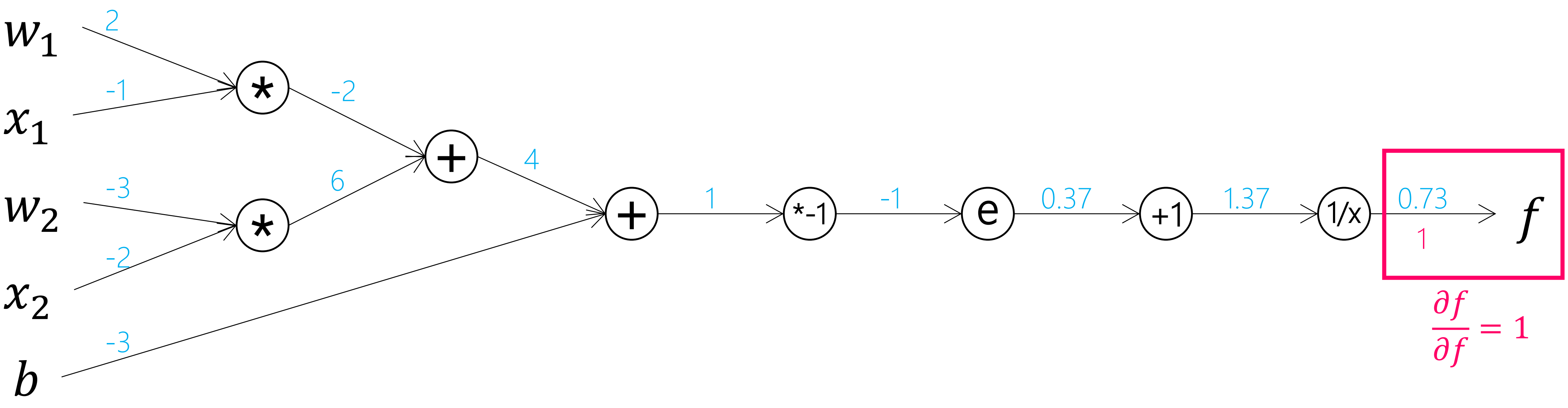
$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$

Does this look familiar?



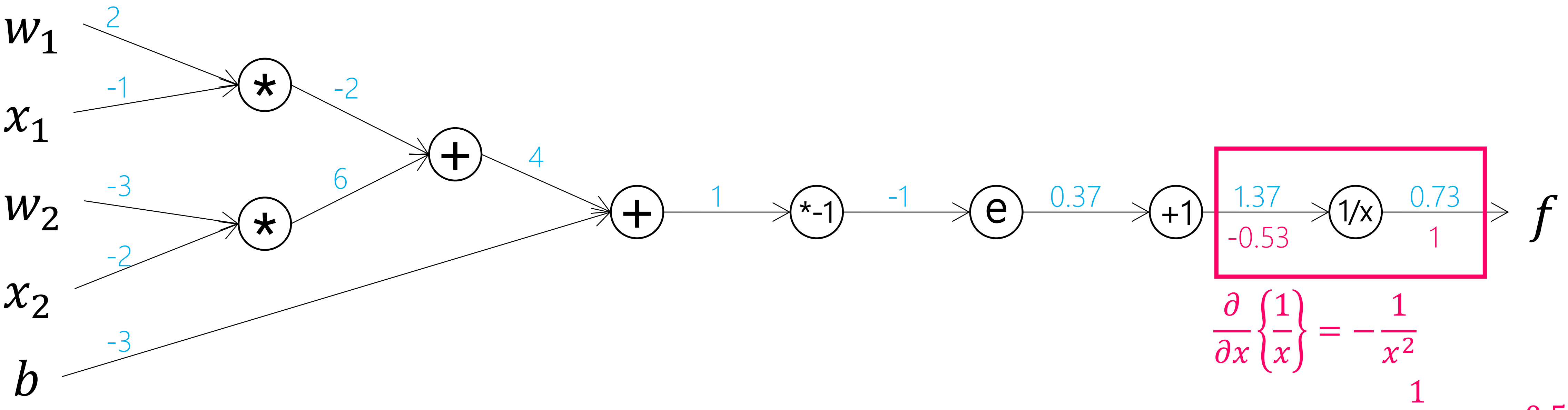
Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$



Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



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

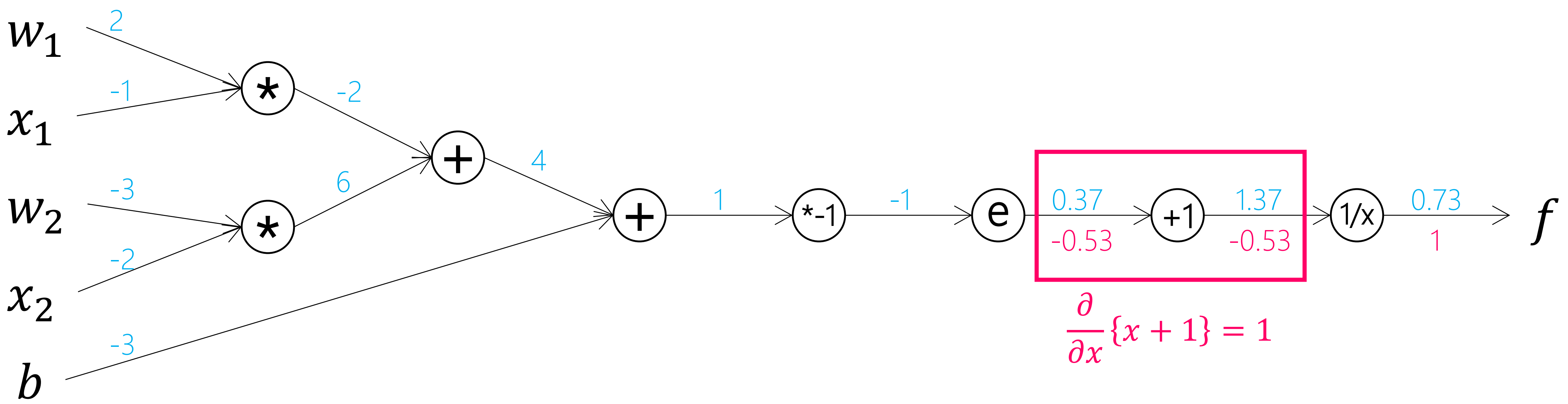
$$= -\frac{1}{1.37^2} = -0.53$$

Cheat Sheet:

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

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$



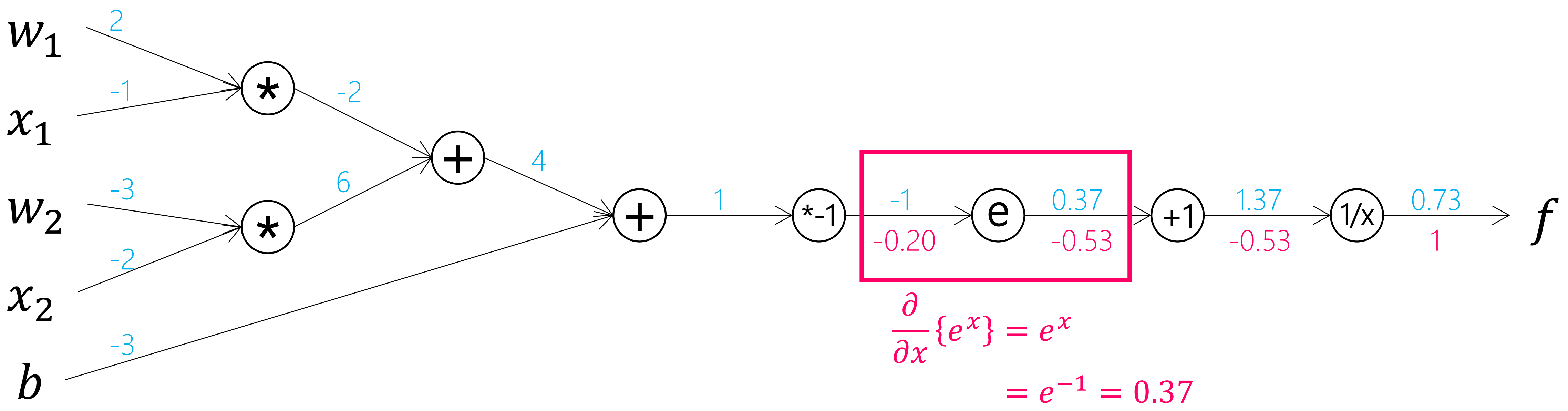
$$\frac{\partial}{\partial x}\{x+1\} = 1$$

Cheat Sheet:

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

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$



$$\frac{\partial}{\partial x}\{e^x\} = e^x$$

$$= e^{-1} = 0.37$$

$$0.37 * (-0.53) \approx -0.20$$

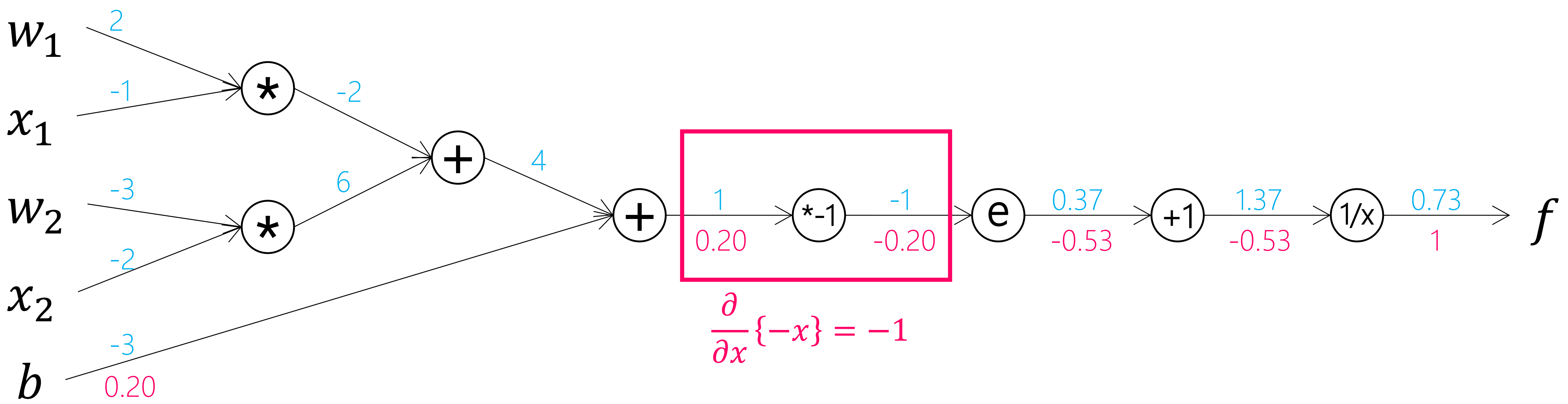
Cheat Sheet:

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

$$\frac{\partial}{\partial x}e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



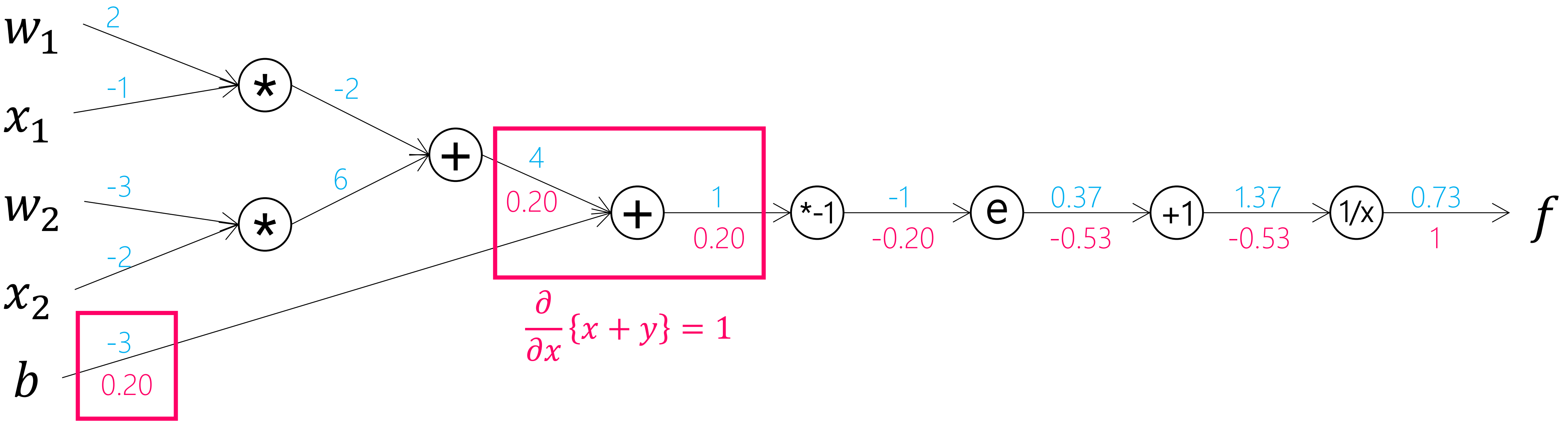
Cheat Sheet:

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

$$\frac{\partial}{\partial x} e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1x_1+w_2x_2+b)}}$$



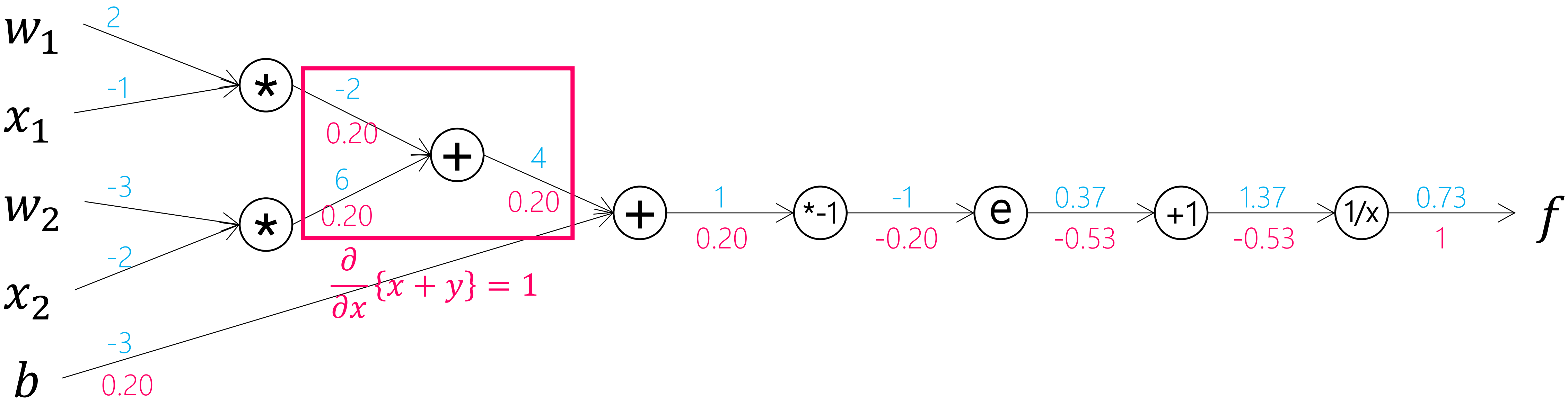
Cheat Sheet:

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

$$\frac{\partial}{\partial x} e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$



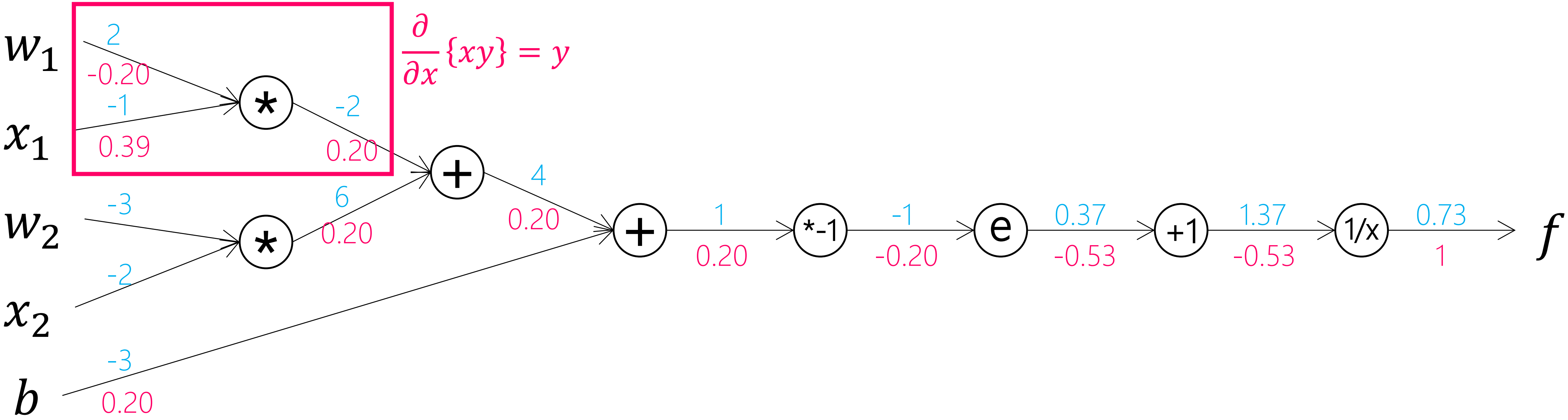
Cheat Sheet:

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

$$\frac{\partial}{\partial x}e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$



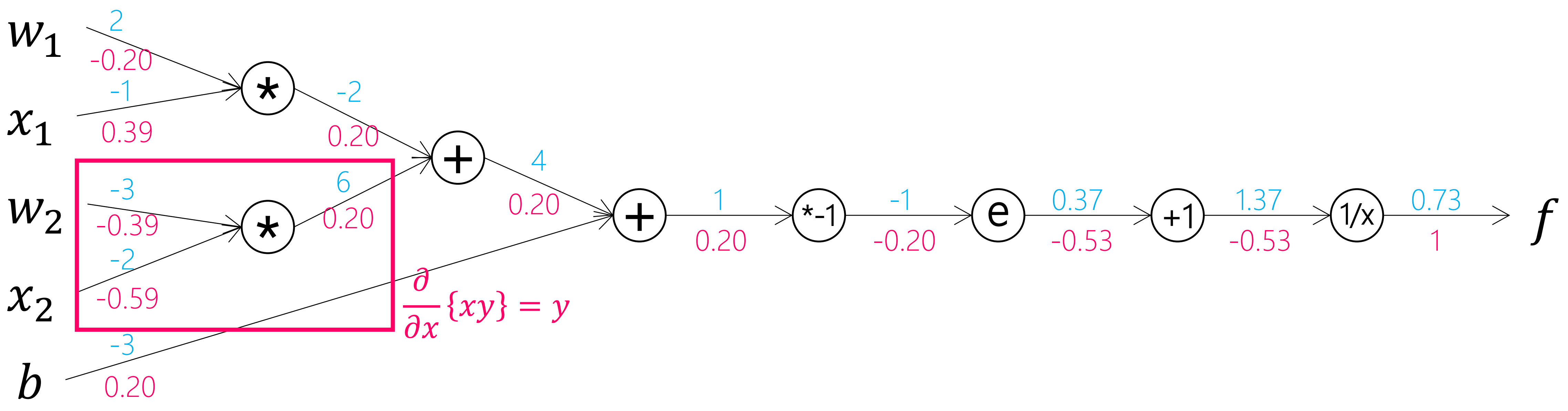
Cheat Sheet:

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

$$\frac{\partial}{\partial x}e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$



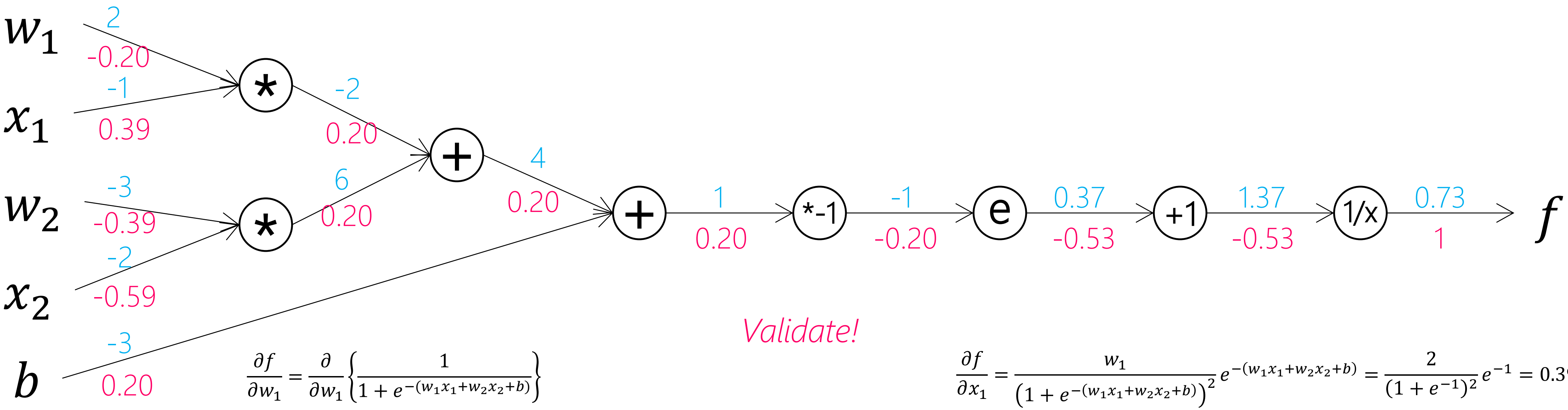
Cheat Sheet:

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

$$\frac{\partial}{\partial x}e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}$$



Validate!

$$\frac{\partial f}{\partial w_1} = \frac{\partial}{\partial w_1}\left\{\frac{1}{1+e^{-(w_1x_1+w_2x_2+b)}}\right\}$$

$$= -\frac{1}{\left(1+e^{-(w_1x_1+w_2x_2+b)}\right)^2}\frac{\partial}{\partial w_1}\left\{1+e^{-(w_1x_1+w_2x_2+b)}\right\}$$

$$= -\frac{1}{\left(1+e^{-(w_1x_1+w_2x_2+b)}\right)^2}e^{-(w_1x_1+w_2x_2+b)}\frac{\partial}{\partial w_1}\{-(w_1x_1+w_2x_2+b)\}$$

$$= \frac{x_1}{\left(1+e^{-(w_1x_1+w_2x_2+b)}\right)^2}e^{-(w_1x_1+w_2x_2+b)}$$

$$= \frac{-1}{\left(1+e^{-(-2+6-3)}\right)^2}e^{-(-2+6-3)} = -0.1966$$

$$\frac{\partial f}{\partial x_1} = \frac{w_1}{\left(1+e^{-(w_1x_1+w_2x_2+b)}\right)^2}e^{-(w_1x_1+w_2x_2+b)} = \frac{2}{(1+e^{-1})^2}e^{-1} = 0.3932$$

$$\frac{\partial f}{\partial w_2} = \frac{x_2}{\left(1+e^{-(w_1x_1+w_2x_2+b)}\right)^2}e^{-(w_1x_1+w_2x_2+b)} = \frac{-2}{(1+e^{-1})^2}e^{-1} = -0.3932$$

$$\frac{\partial f}{\partial x_2} = \frac{w_2}{\left(1+e^{-(w_1x_1+w_2x_2+b)}\right)^2}e^{-(w_1x_1+w_2x_2+b)} = \frac{-3}{(1+e^{-1})^2}e^{-1} = -0.5898$$

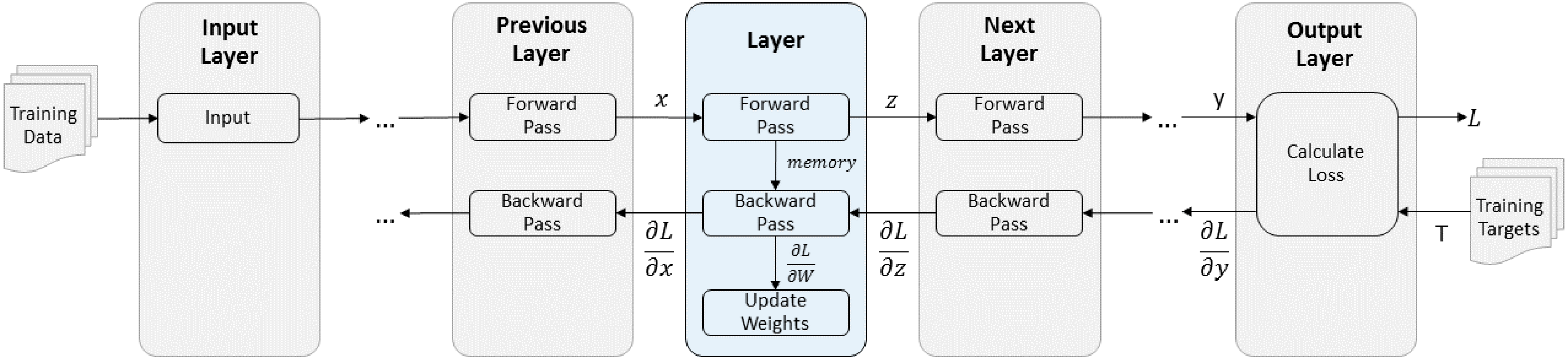
$$\frac{\partial f}{\partial b} = \frac{1}{\left(1+e^{-(w_1x_1+w_2x_2+b)}\right)^2}e^{-(w_1x_1+w_2x_2+b)} = \frac{1}{(1+e^{-1})^2}e^{-1} = 0.1966$$

Cheat Sheet:

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

$$\frac{\partial}{\partial x}e^x = e^x$$

Putting them all together



https://www.mathworks.com/help/nnet/ug/define-custom-deep-learning-layers.html

Loss Functions

Lots of Puzzling Terms!

- Mean Squared Error/L2 (MSE/L2): `torch.nn.MSELoss`
- Mean Absolute Error/L1 (MAE/L1): `torch.nn.L1Loss`
- Huber Loss: `torch.nn.HuberLoss`
- Smooth L1: `torch.nn.SmoothL1Loss`
- Cross Entropy: `torch.nn.CrossEntropyLoss`
- Binary Cross Entropy: `torch.nn.BCELoss`
- Kullback-Leibler Divergence: `torch.nn.KLDivLoss`
- Hinge Embedding: `torch.nn.HingeEmbeddingLoss`
- Connectionist Temporal Classification: `torch.nn.CTCLoss`
- Negative Log Likelihood: `torch.nn.NLLLoss`
- Cosine Embedding: `torch.nn.CosineEmbeddingLoss`
- Margin Ranking: `torch.nn.MarginRankingLoss`
- Soft Margin: `torch.nn.SoftMarginLoss; torch.nn.MultiLabelSoftMarginLoss`
- Triplet Margin: `torch.nn.TripletMarginLoss`

Information Theory (Claude Shannon, 1948)

- Digital information: series of bits (either 0 or 1)
- Sending a single bit (useful) of information = reducing receiver's uncertainty into half
- For example, who would win the Yonsei-Korea University game, assuming the both team have the equal chance of winning (50%)?
  - With no information, the uncertainty is 50-50.
  - One bit of information (Yonsei won!) → resolves the uncertainty.
- Another example, imagine a league of 8 teams. Who is the winner?
  - With no information, the uncertainty is 12.5% each.
  - How many bits of information do you need?
  - 3 bits! ($2^3 = 8$, or $\log_2(8)=3$)

Information Theory (Claude Shannon, 1948)

- What if the chance of winning is not equal?
  - Say, Yonsei (75% of winning) and Korea U (25% of winning).
  - If a sender says, Korea U won the game, the uncertainty drops by the factor of 4.
    - Uncertainty reduction = $-\log_2(1/4) = 2$
  - If a sender says, Yonsei won the game,
    - Uncertainty reduction = $-\log_2(3/4) = 0.42$
  - Therefore, the expected number of bits to resolve the uncertainty:
    - $-0.75*\log_2(3/4) - 0.25*\log_2(1/4) = 0.75*0.42 + 0.25*2 = 0.82$ bits

- Entropy: $H(p) = -\sum_i p_i \log_2(p_i)$
  - Average amount of information that can be derived from one sample drawn from a given probability distribution
  - Indicator of how unpredictable the probability distribution is.
  - More variation in the data → larger entropy.

Cross Entropy

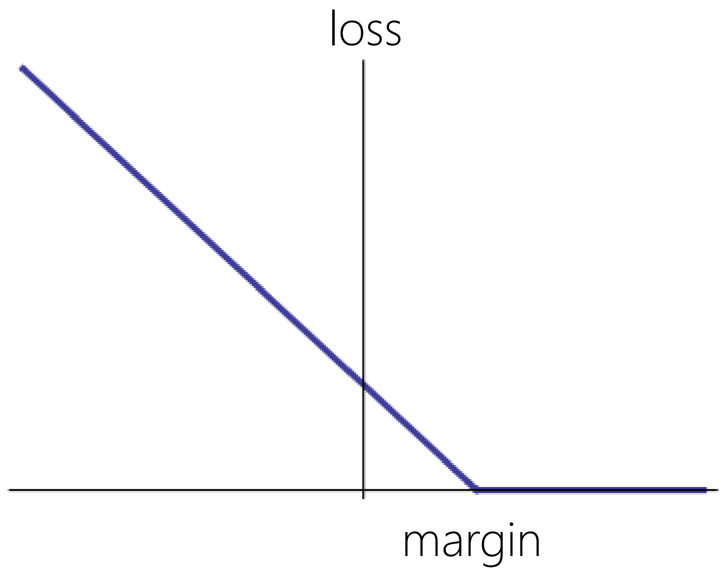
- Entropy: $H(p) = -\sum_i p_i \log_2(p_i)$
- Cross Entropy: $H(p, q) = -\sum_i p_i \log_2(q_i)$
  - Think of $\boldsymbol{p}$ as a true distribution and $\boldsymbol{q}$ as a predicted distribution.
  - If $\boldsymbol{q} = \boldsymbol{p}$ (correct prediction), cross entropy equals to entropy.
  - If $\boldsymbol{q} \neq \boldsymbol{p}$ (erroneous prediction), cross entropy gets greater (why?) than entropy by some number of bits

https://en.wikipedia.org/wiki/Gibbs'_inequality

- Kullback-Leibler Divergence: $D_{KL}(p, q) = H(p, q) - H(p)$
  - The amount of difference between cross entropy and entropy.
  - a.k.a, relative entropy

Hinge Loss

- Penalizes incorrectly classified examples + correctly classified examples that lie within the margin
- Hinge loss is generally faster than cross entropy but less accurate.



Regression: L_p Distances

- L_p norm or Minkowski distance:

$$L_p(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

- For $p = 1$, Manhattan (or city-block) distance:

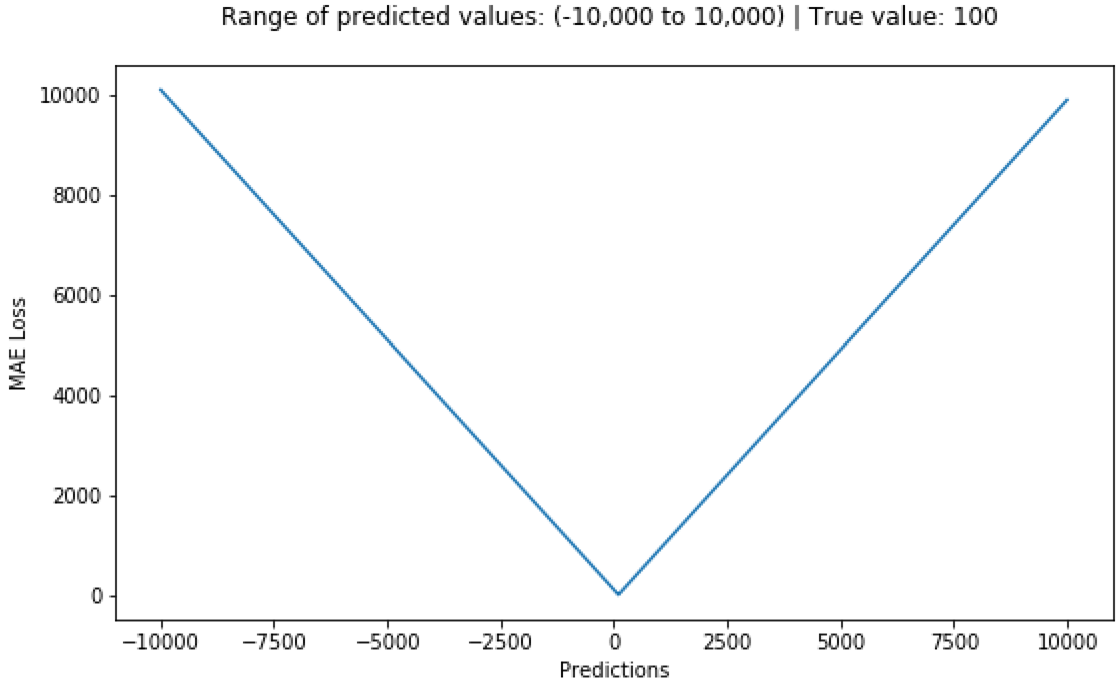
$$L_1(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

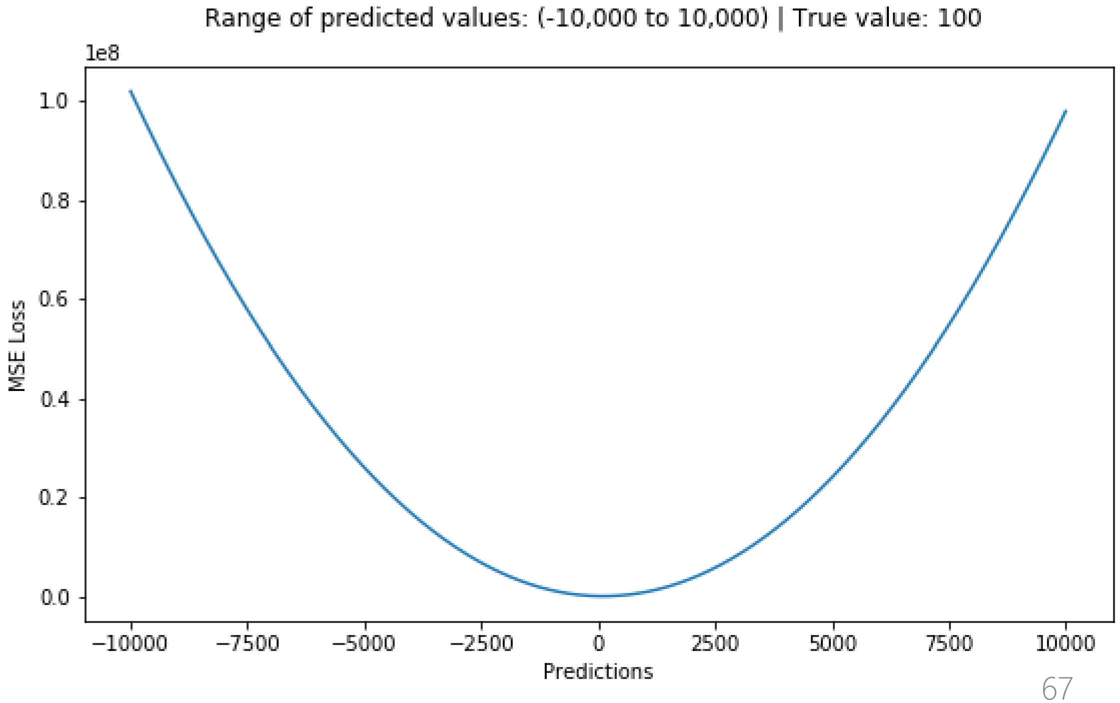
- For $p = 2$, Euclidean distance:

$$L_2(x, y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

L1 and L2 Loss

- Mean Absolute Error (MAE): $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- Mean Squared Error (MSE): $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2$

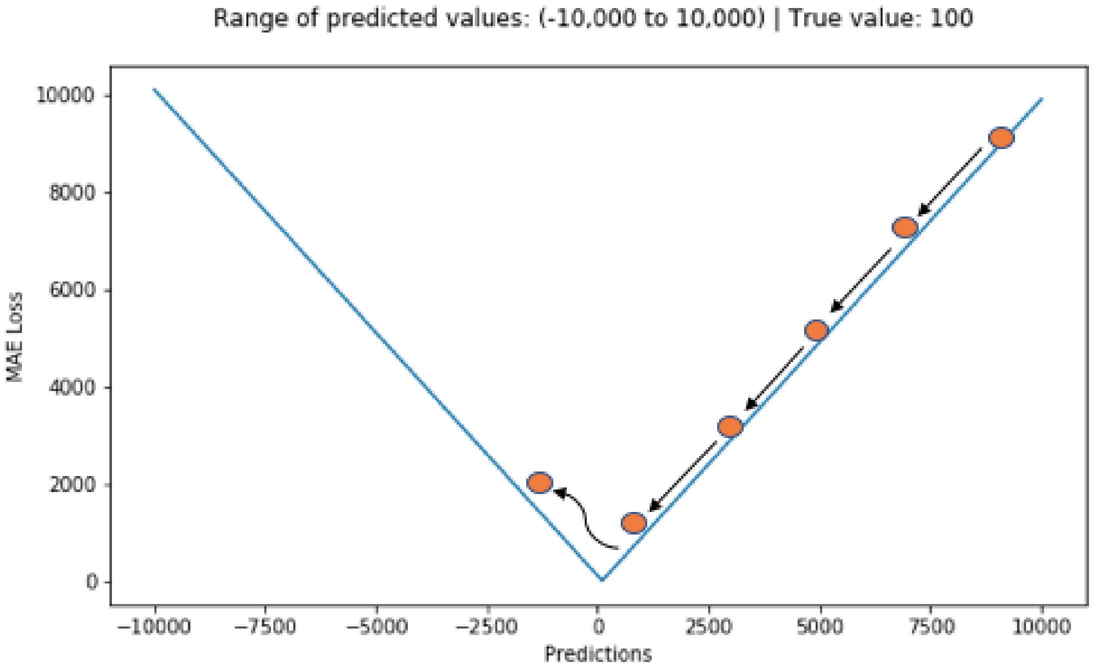


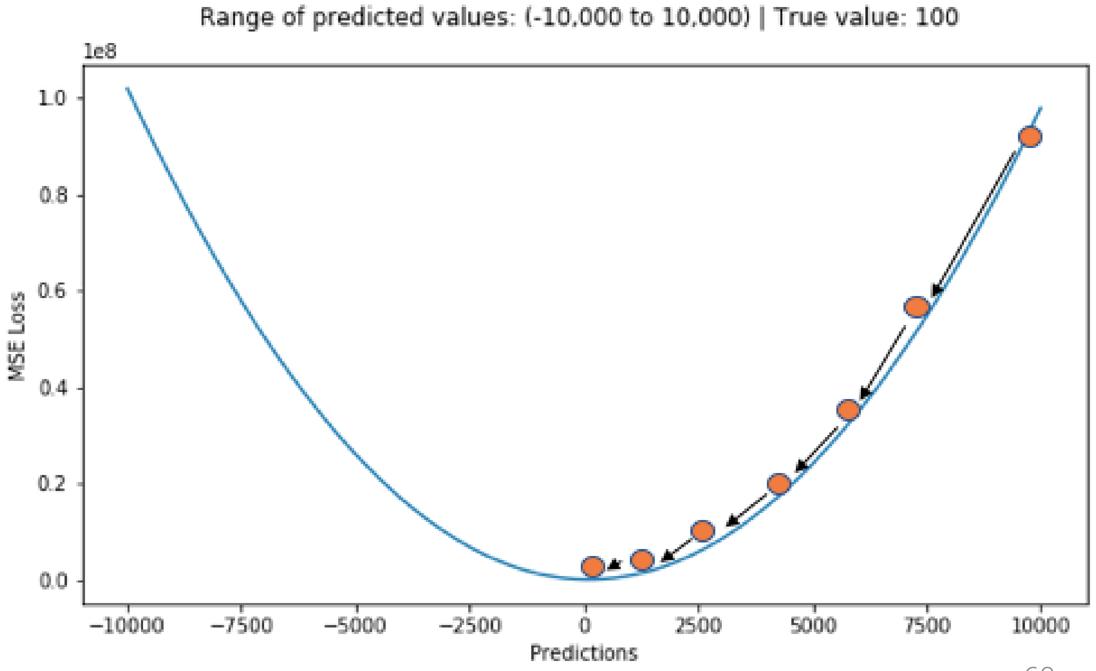


https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

L1 and L2 Loss

- Mean Absolute Error (MAE): $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- Mean Squared Error (MSE): $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|^2$



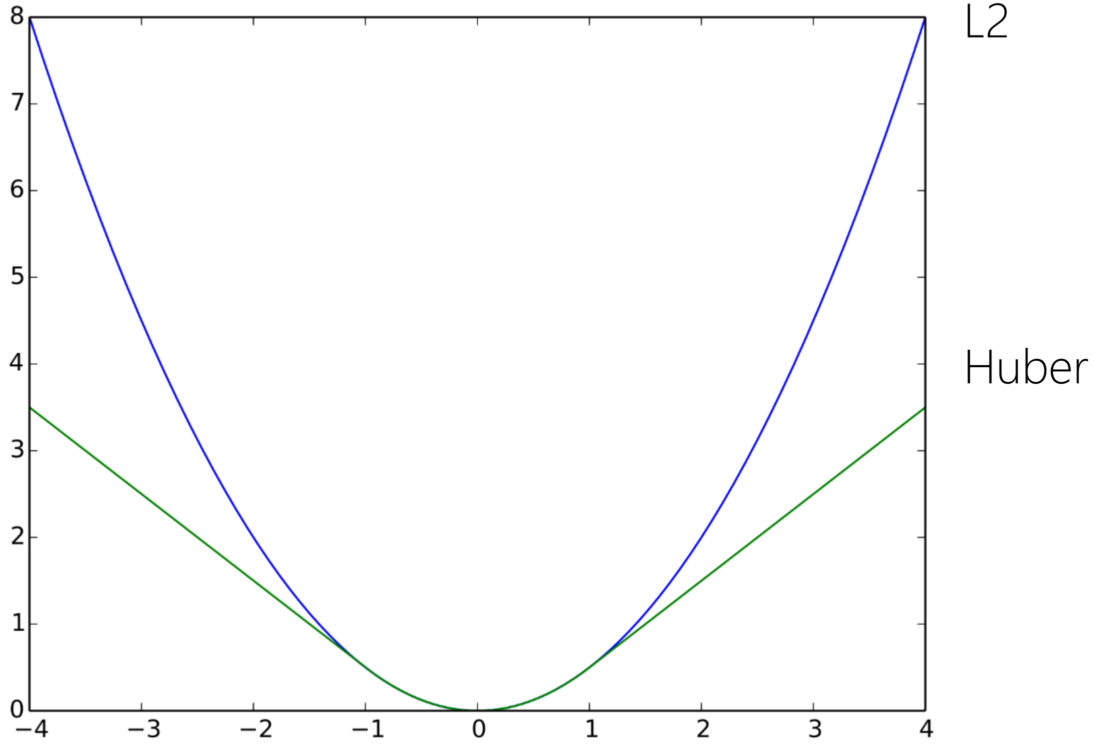


https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

Huber & Smoothed L1

- Huber:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta, \\ \delta \cdot \left(|a| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases}$$



https://en.wikipedia.org/wiki/Huber_loss#/media/File:Huber_loss.svg

Still puzzling?

- Mean Squared Error/L2 (MSE/L2): `torch.nn.MSELoss`
- Mean Absolute Error/L1 (MAE/L1): `torch.nn.L1Loss`
- Huber Loss: `torch.nn.HuberLoss`
- Smooth L1: `torch.nn.SmoothL1Loss`
- Cross Entropy: `torch.nn.CrossEntropyLoss`
- Binary Cross Entropy: `torch.nn.BCELoss`
- Kullback-Leibler Divergence: `torch.nn.KLDivLoss`
- Hinge Embedding: `torch.nn.HingeEmbeddingLoss`
- Connectionist Temporal Classification: `torch.nn.CTCLoss`
- Negative Log Likelihood: `torch.nn.NLLLoss`
- Cosine Embedding: `torch.nn.CosineEmbeddingLoss`
- Margin Ranking: `torch.nn.MarginRankingLoss`
- Soft Margin: `torch.nn.SoftMarginLoss; torch.nn.MultiLabelSoftMarginLoss`
- Triplet Margin: `torch.nn.TripletMarginLoss`

Some generically useful ones that are applicable to the majority of the PADL problems

More specific loss functions for advanced learners