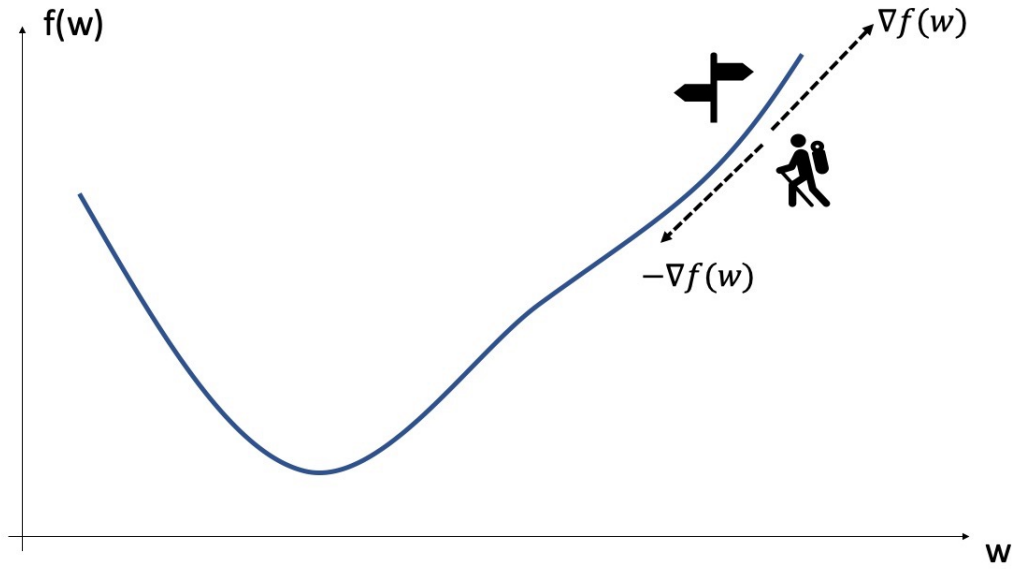
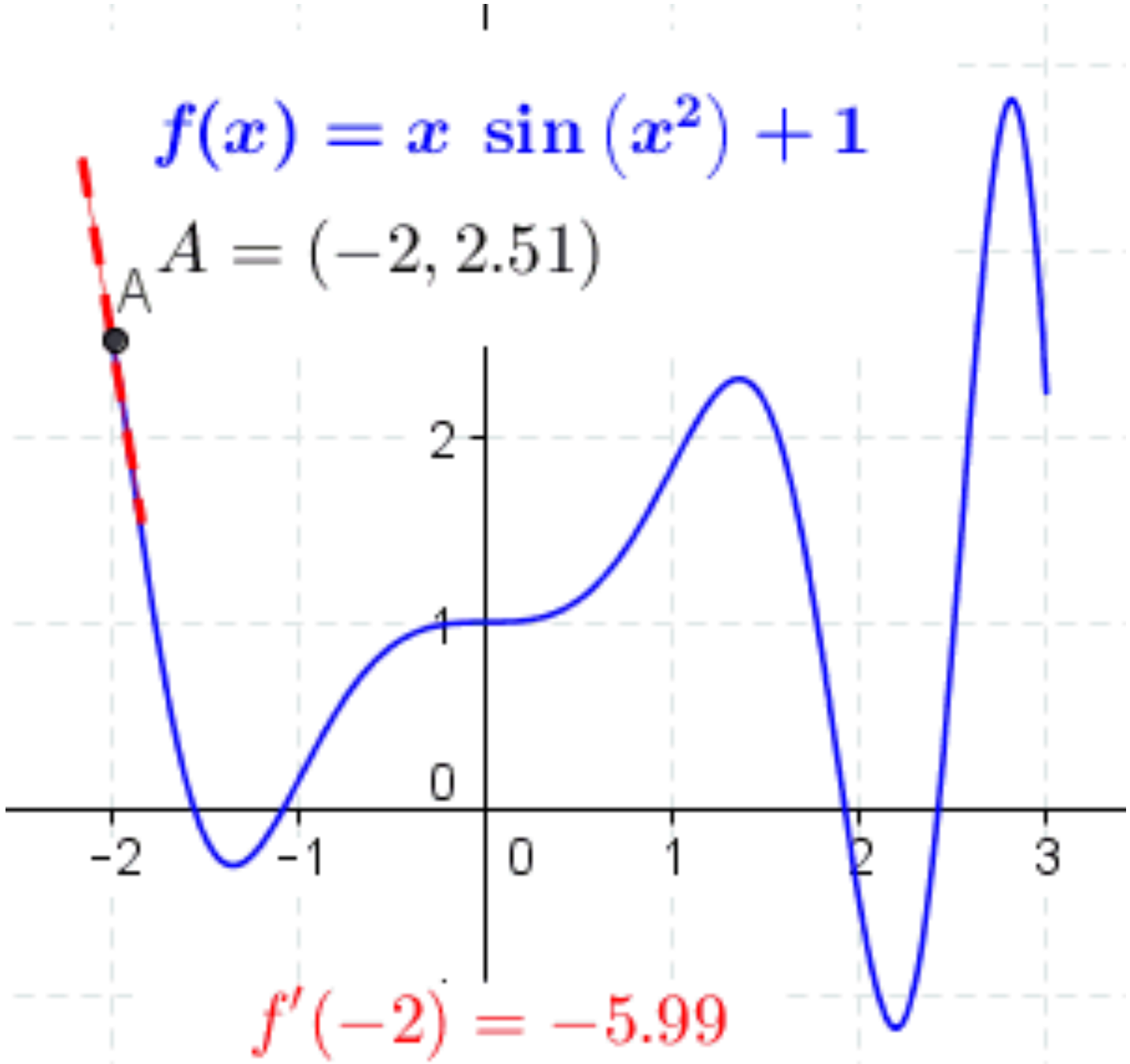
Gradient Descent



CS-EJ3311 - Deep Learning with Python
24.10.-11.12.2022
Aalto University & FiTech.io

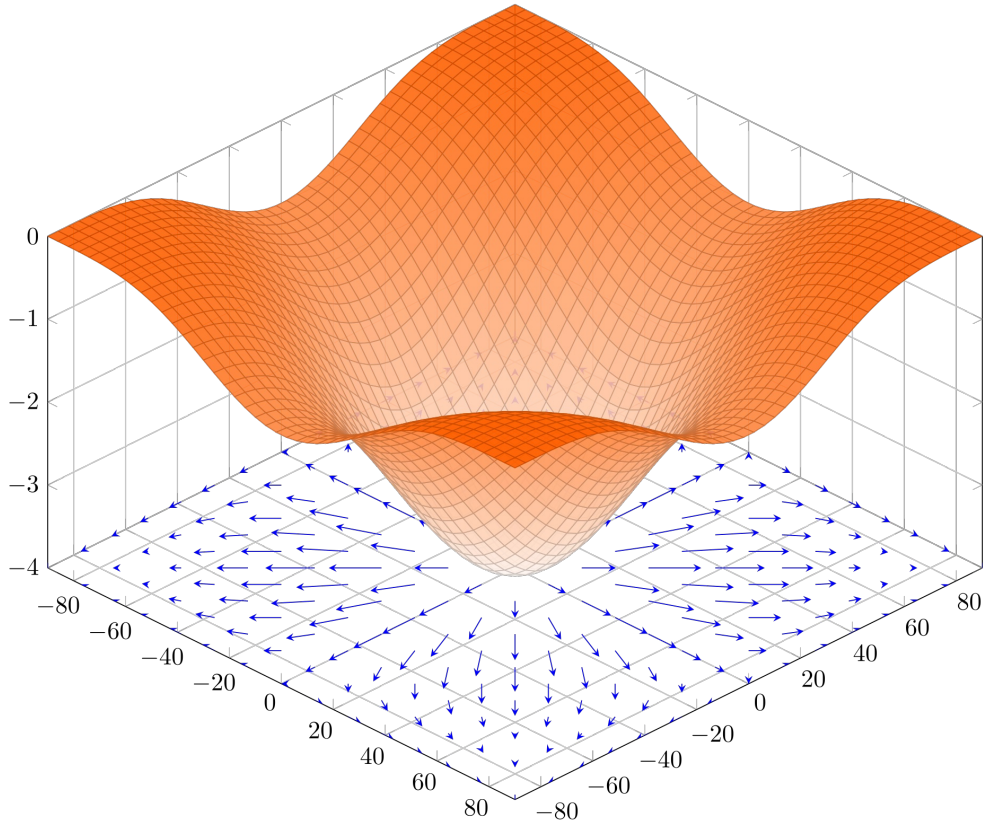
7.11.2022 Shamsi Abdurakhmanova



source

Gradient of multivariate function

$$\nabla \mathrm{f}(\mathbf{w}) = \begin{bmatrix} \frac{\partial f}{\partial w^{(1)}} \\ \frac{\partial f}{\partial w^{(2)}} \\ \vdots \\ \frac{\partial f}{\partial w^{(d)}} \end{bmatrix}$$

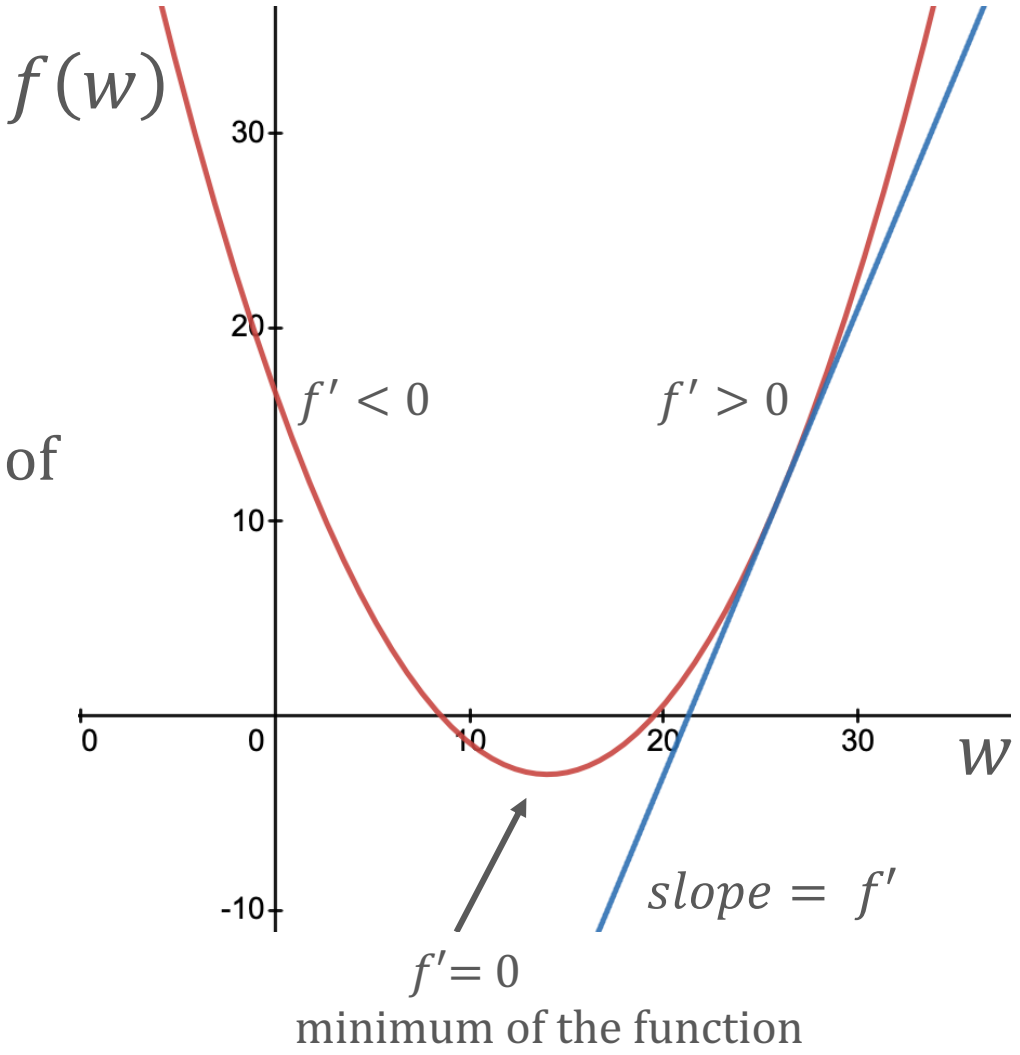


The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

source

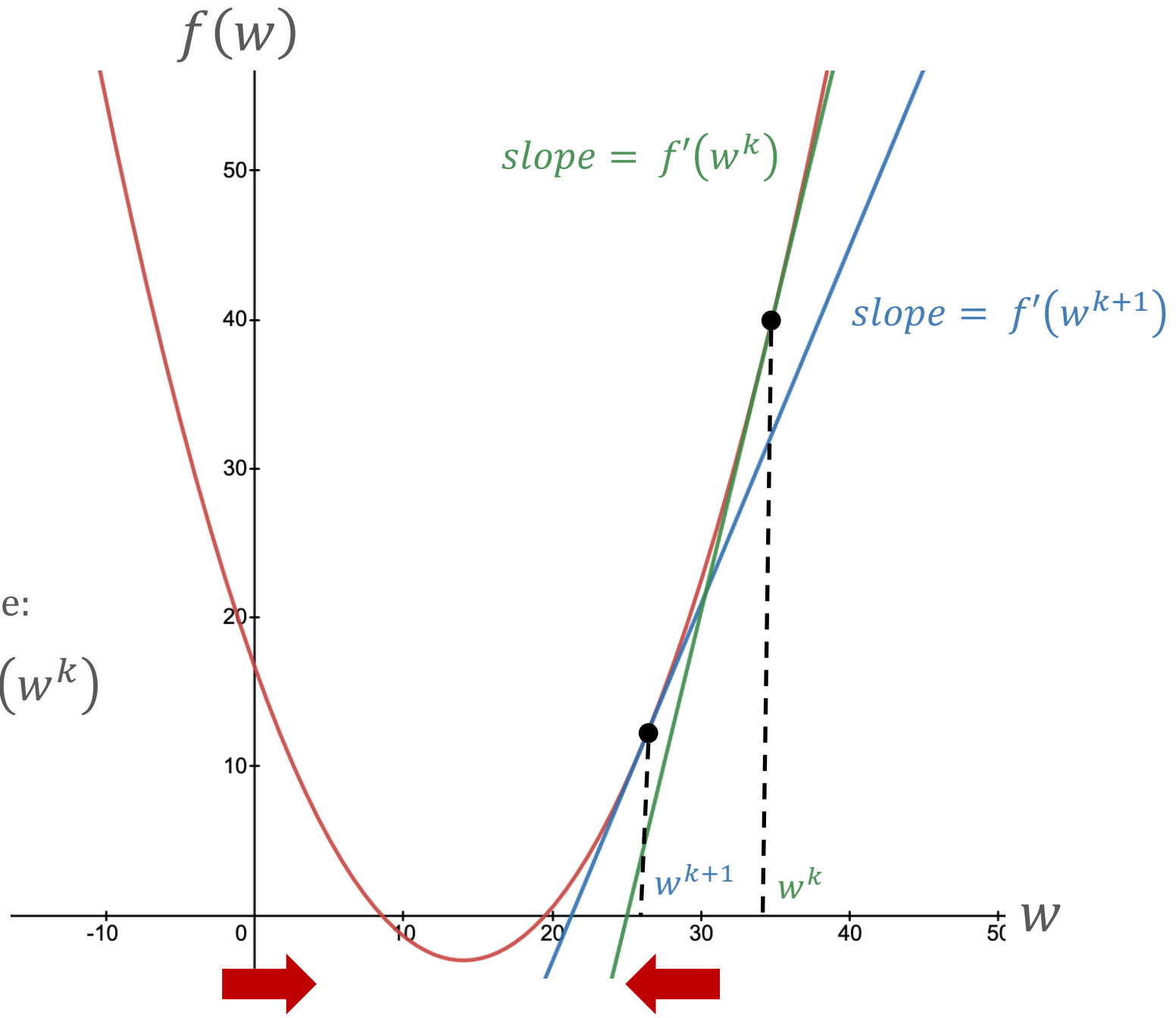
$$f(w) = (aw + b)^2 + c$$

The **derivative** of a function of a real variable measures the sensitivity to change of the function value (output value)—with respect to a change in its argument (input value).



weight update rule:

$$w^{k+1} = w^k - \alpha f'(w^k)$$



Derivation rules:

$F(w) = f(w) + g(w)$
$f(w) = cw$
$f(w) = w^2$

$F'(w) = f'(w) + g'(w)$
$f'(\mathrm{w}) = \mathrm{c}$
$f'(\mathrm{w}) = 2\mathrm{w}$

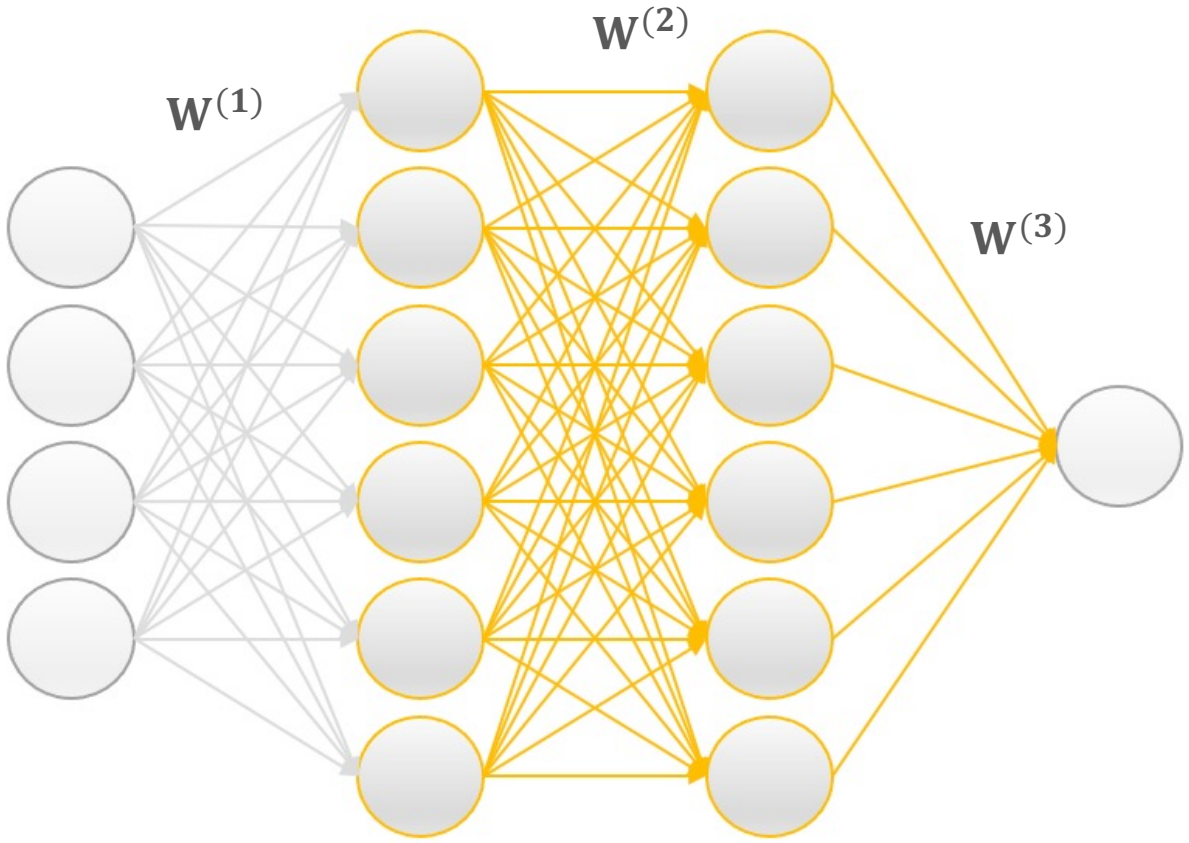
Chain rule:

$F(w) = f\big(g(w)\big)$

$F' = f'\big(g(w)\big)g'(w)$

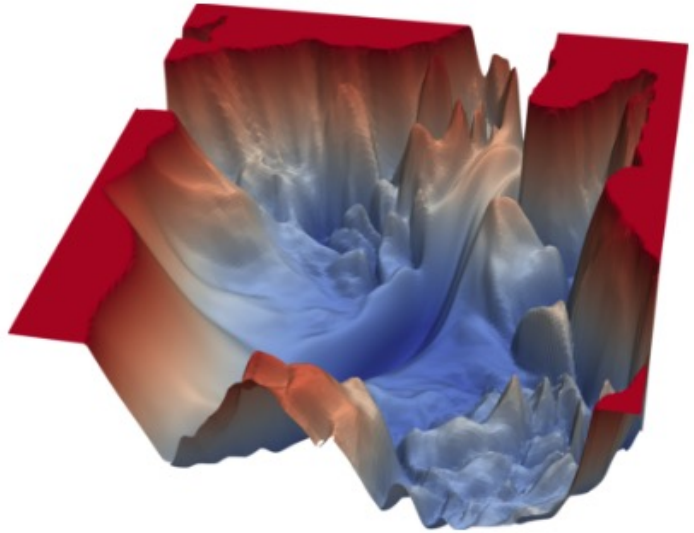
$f(w) = (aw + b)^2 + \mathrm{c}$  $f'(w) = 2a(aw + b)$

Min of $f(w)$ is at $f'(w) = 0$

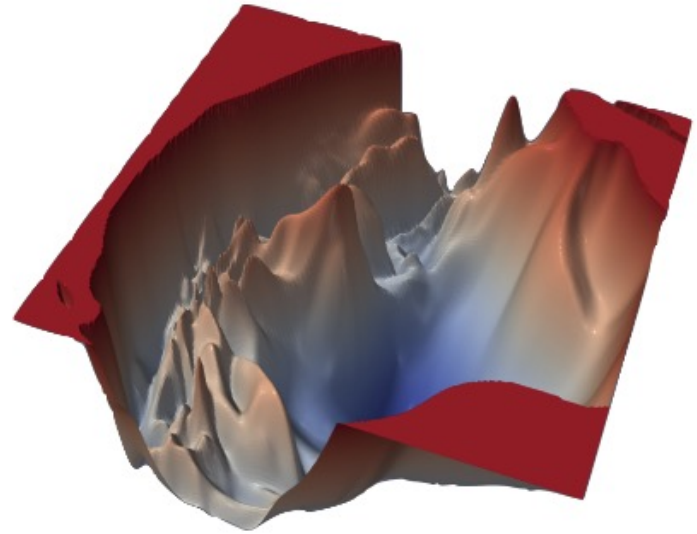


$$f(\mathbf{w}) = \text{Dense3}(\text{Dense2}(\text{Dense1}(x)))$$

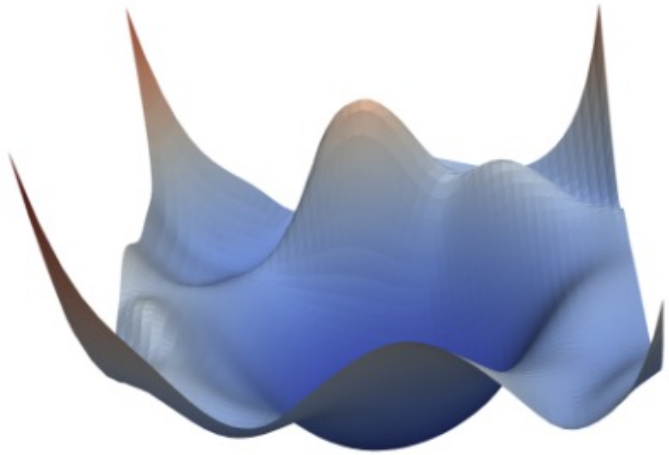
VGG-56



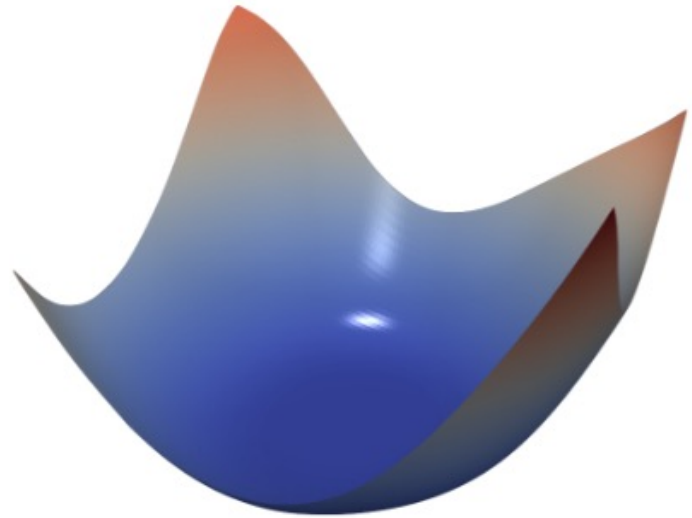
VGG-110



Renset-56



Densenet-121

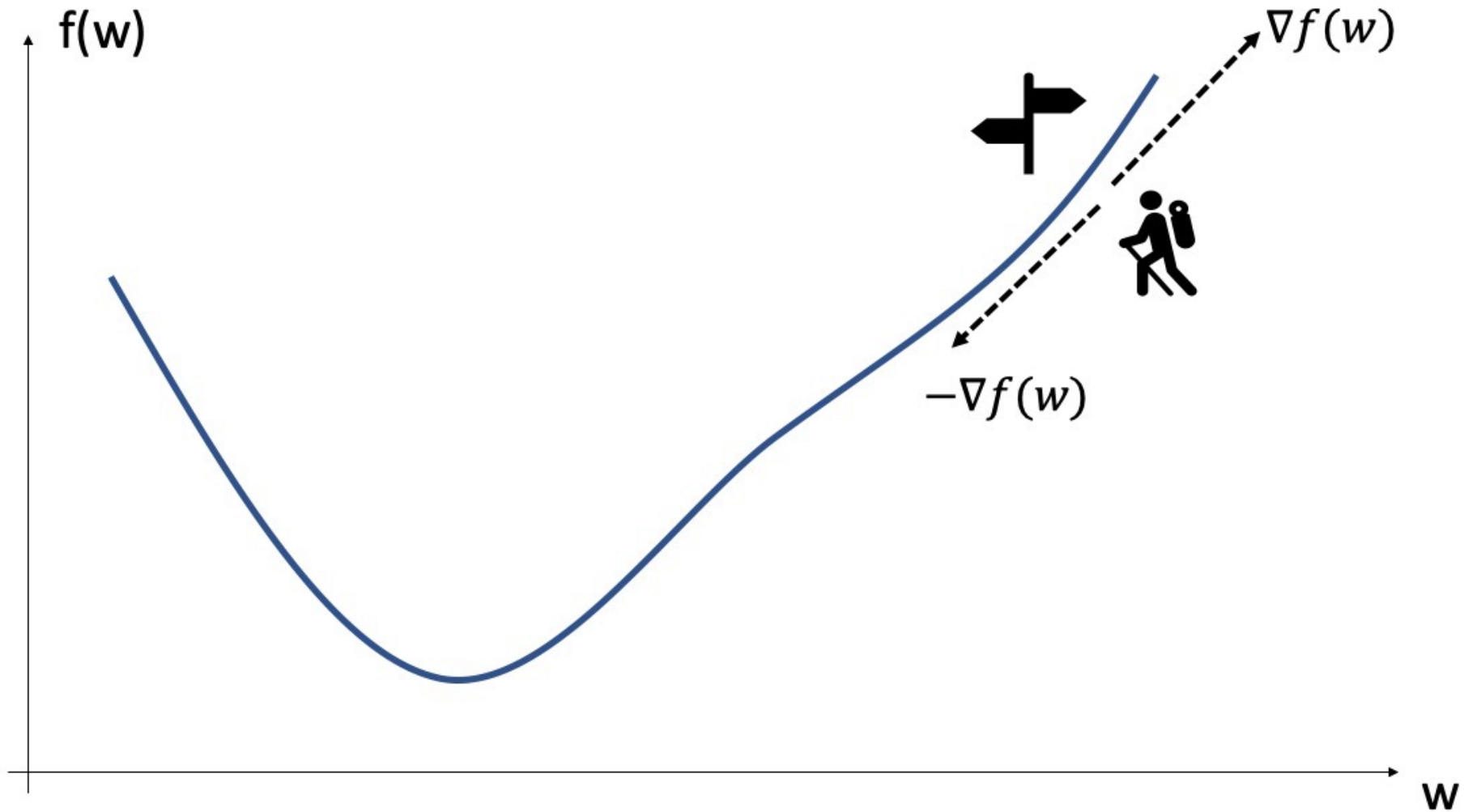


Gradient Descent (GD)

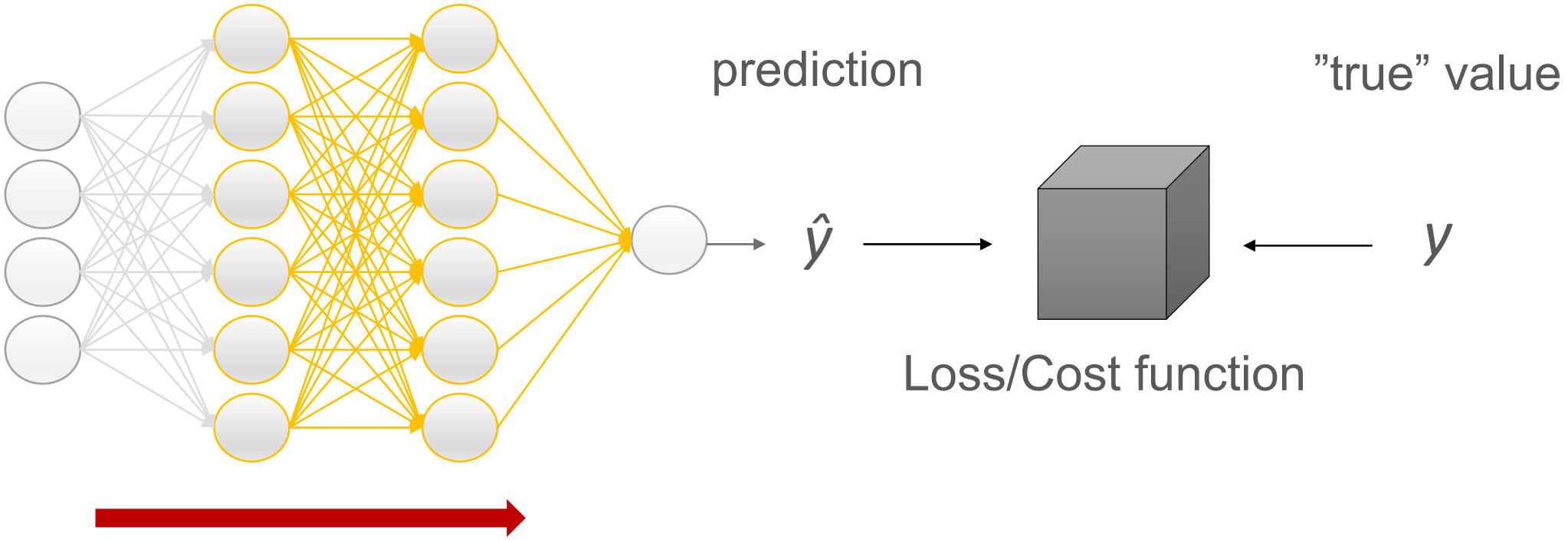
GD is a gradient based optimization algorithm:

uses a gradient of the loss function to tunes parameters

iteratively and to minimize a loss function

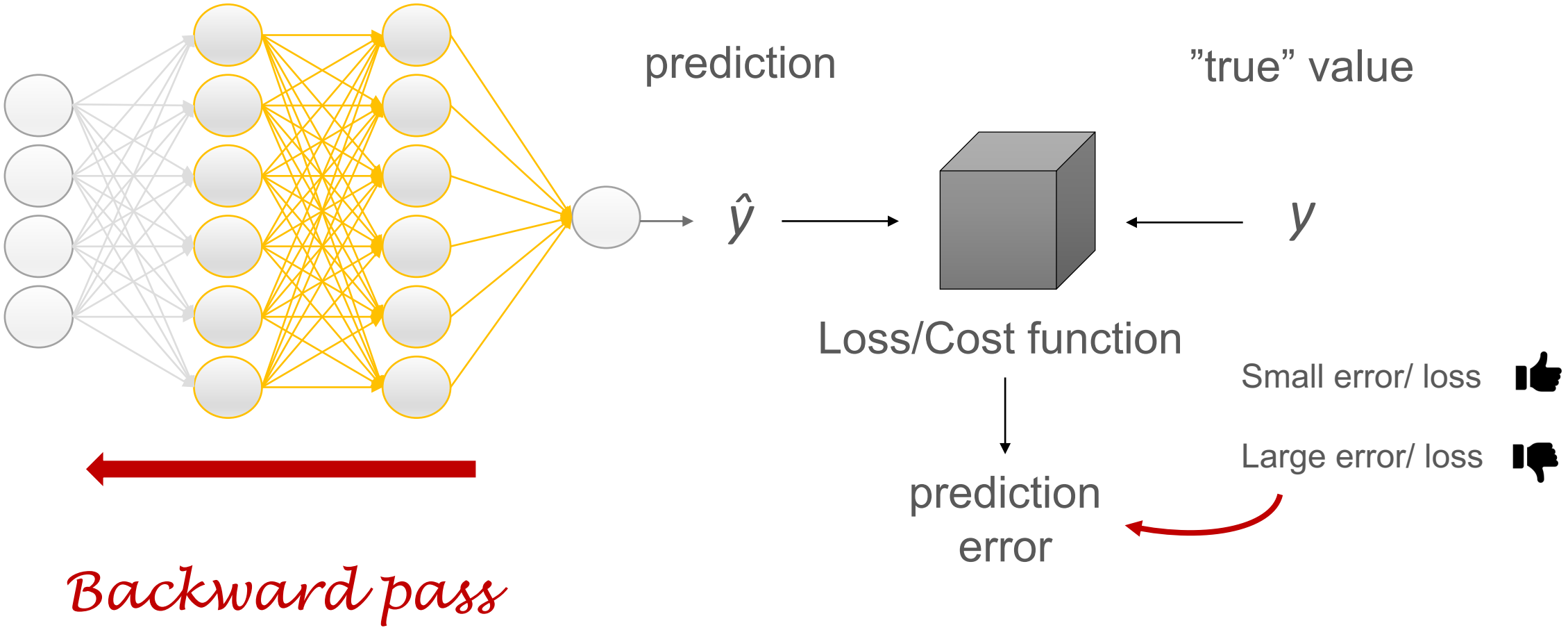

f(w)
$\nabla f(w)$
$-\nabla f(w)$
w

Gradient Descent Algorithm

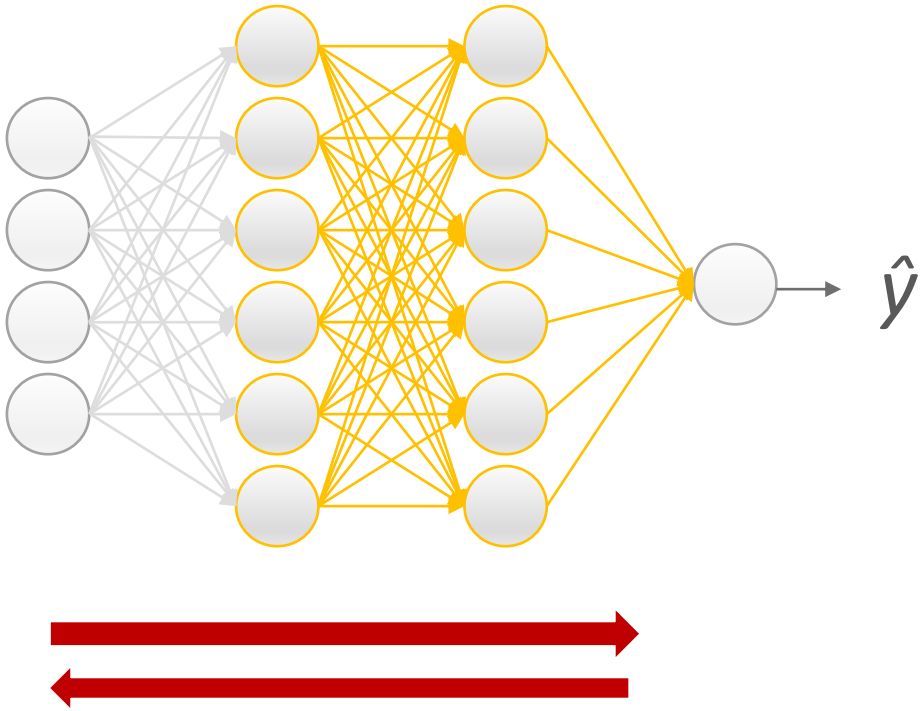


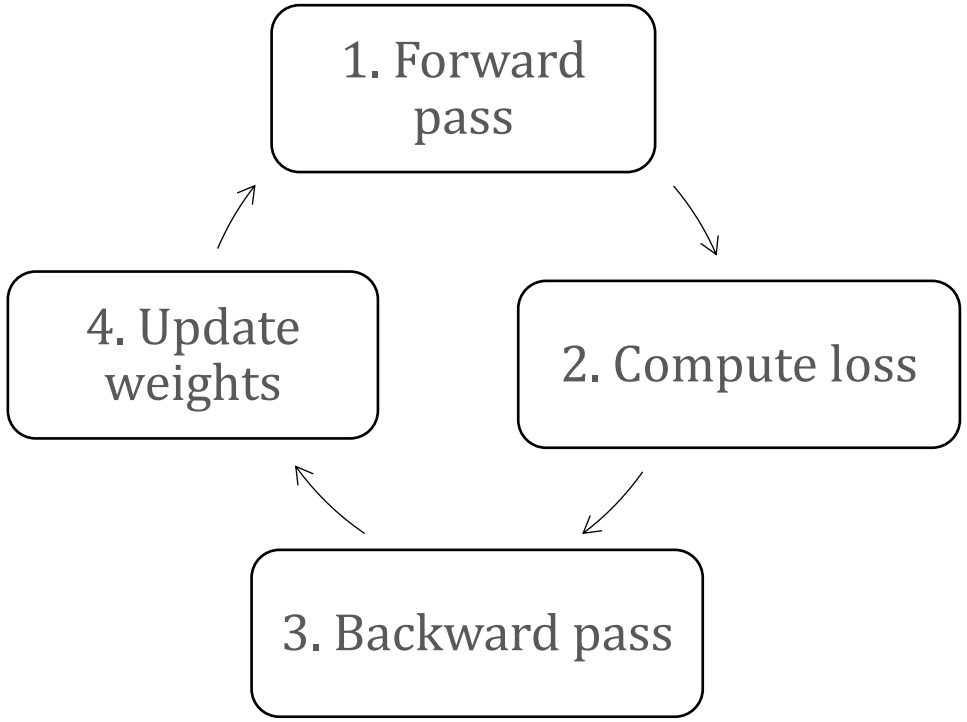
Forward pass

Gradient Descent Algorithm



Gradient Descent Algorithm





Linear regression:

- **one datapoint** i
- **one feature** $x^{(i)}$
- **one weight** w

Predictor:

$$h^{(w)}(x) = wx^{(i)} = \hat{y}^{(i)}$$

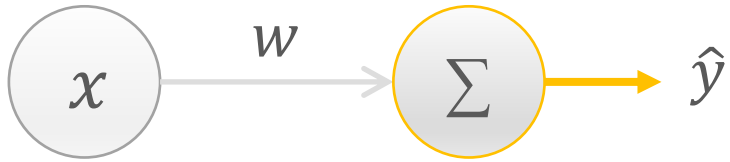
Loss:

$$f(w) = \left(y^{(i)} - \hat{y}^{(\mathrm{i})}\right)^2$$

1. Forward Pass - compute prediction given the current weight
2. Backward Pass - compute derivative of the loss function
3. Update the weight

Forward Pass

$$\hat{y} = wx$$



Loss

$$f(w) = \left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \left(y^{(i)} - wx^{(i)}\right)^2$$

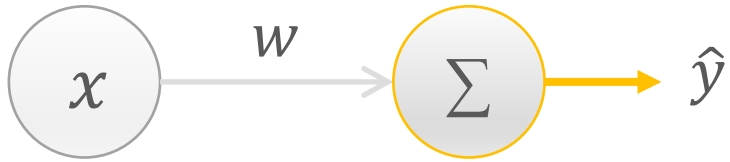
Local gradients

$$\frac{d\hat{y}}{dw} = x$$

$$\frac{df}{d\hat{y}} = -2\left(y^{(i)} - \hat{y}^{(i)}\right)$$

Forward Pass

$$\hat{y} = wx$$



Loss

$$f(w) = \left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \left(y^{(i)} - wx^{(i)}\right)^2$$

Backward Pass

Full gradient

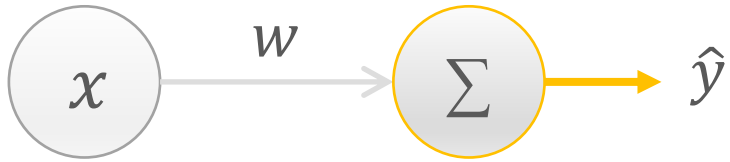
$$\nabla f = \frac{df}{dw} = \frac{df}{d\hat{y}}\frac{d\hat{y}}{dw} = -2x\left(y^{(i)} - \hat{y}^{(i)}\right)$$

Forward Pass

$\hat{y} = wx$

Loss

$f(w) = \left(y^{(i)} - wx^{(i)}\right)^2$



Backward Pass

Gradient Step $w^{(k+1)} = w^{(k)} - \alpha \nabla f\left(w^k\right)$

Logistic Loss:

- **one datapoint** i
- **one feature** $x^{(i)}$
- **one weight** w

Predictor:

$$p = \frac{1}{1 + e^{-wx}}$$

Loss:

$$f(w) = -y^{(i)}\, ln\left(p^{(i)}\right) - \left(1 - y^{(i)}\right)\ln(1 - p^{(i)})$$

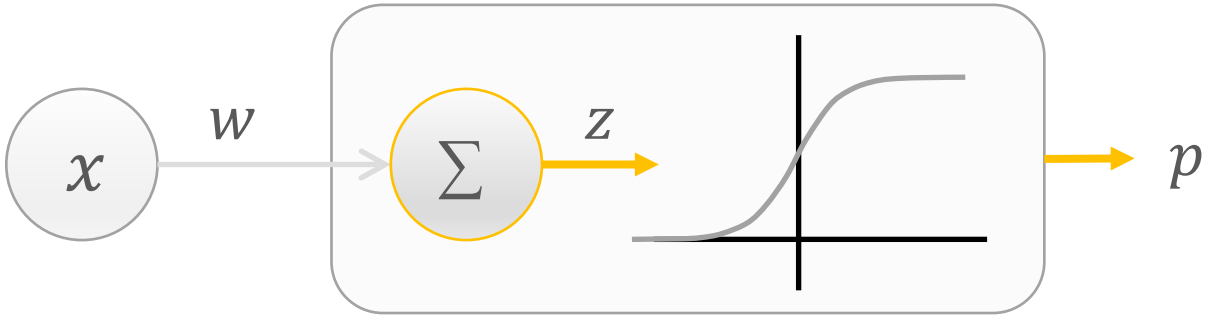
1. Forward Pass - compute prediction given the current weight
2. Backward Pass - compute derivative of the loss function
3. Update the weight

Forward Pass

$$z = wx \qquad p = \frac{1}{1 + e^{-z}}$$

Loss

$$f(w) = -y^{(i)}\, ln\left(p^{(i)}\right) - \left(1 - y^{(i)}\right)\ln\left(1 - p^{(i)}\right)$$

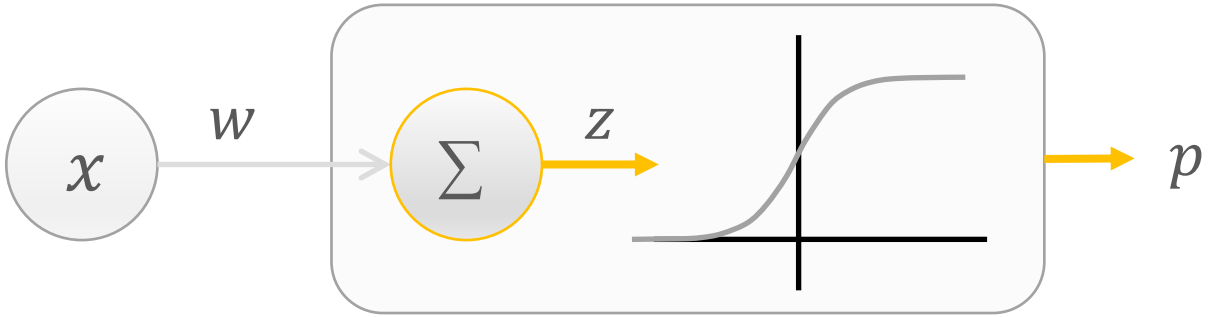


Forward Pass

$$z = wx \qquad p = \frac{1}{1 + e^{-z}}$$

Loss

$$f(w) = -y^{(i)}\, ln\left(p^{(i)}\right) - \left(1 - y^{(i)}\right)\ln\left(1 - p^{(i)}\right)$$



Local gradients

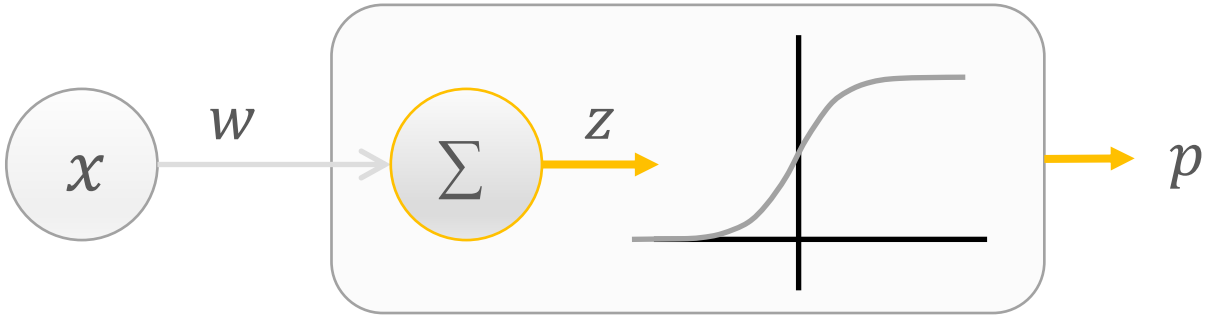
$$\frac{dz}{dw} = x \qquad \frac{dp}{dz} = p(1 - p) \qquad \frac{df}{dp} = -\frac{y}{p} + \frac{1 - y}{1 - p}$$

Forward Pass

$$z = wx \qquad p = \frac{1}{1 + e^{-z}}$$

Loss

$$f(w) = -y^{(i)}\, ln\left(p^{(i)}\right) - \left(1 - y^{(i)}\right)\ln\left(1 - p^{(i)}\right)$$



Backward Pass

Full gradient

$$\nabla f(w) = \frac{df}{dw} = \frac{df}{dp}\frac{dp}{dz}\frac{dz}{dw}$$

Forward Pass

$$z = wx$$

$$p = \frac{1}{1 + e^{-z}}$$

$$f(w) = -yln(p) - (1 - y)\ln(1 - p)$$

$$\frac{dz}{dw}\frac{dp}{dz}\frac{df}{dp}$$

$$\frac{\partial p}{\partial z}\frac{\partial f}{\partial p}$$

$$\frac{\partial f}{\partial p}$$

$$\nabla f(w) = \frac{\partial f}{\partial w} = \frac{\partial f}{\partial p}\frac{\partial p}{\partial z}\frac{\partial z}{\partial w}$$

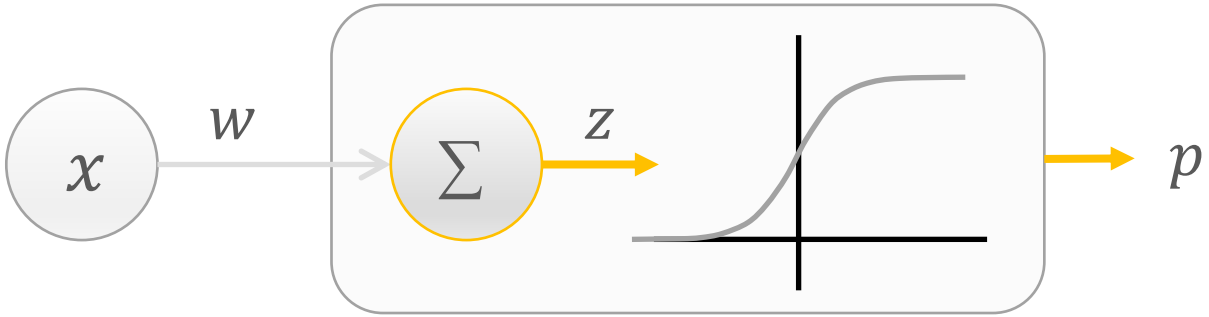
Backward Pass

Forward Pass

$$z = wx \qquad p = \frac{1}{1 + e^{-z}}$$

Loss

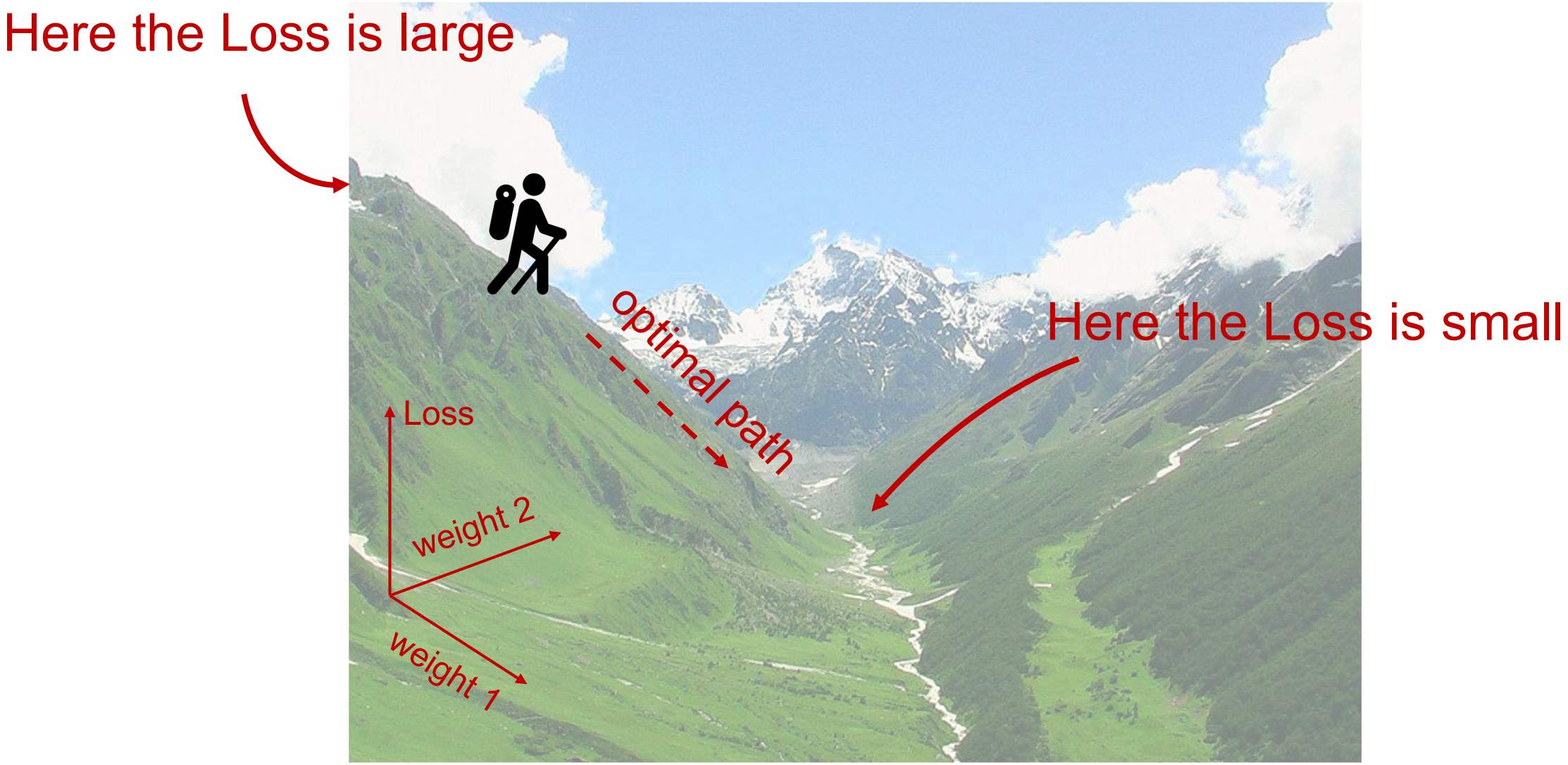
$$f(w) = -y^{(i)}\, ln\left(p^{(i)}\right) - \left(1 - y^{(i)}\right)\ln\left(1 - p^{(i)}\right)$$



Backward Pass

Gradient Step $\qquad w^{(k+1)} = w^{(k)} - \alpha \nabla f\left(w^{k}\right)$

Loss landscape hiker



Gradient Descent Algorithm

Most important hyper-parameters to set:

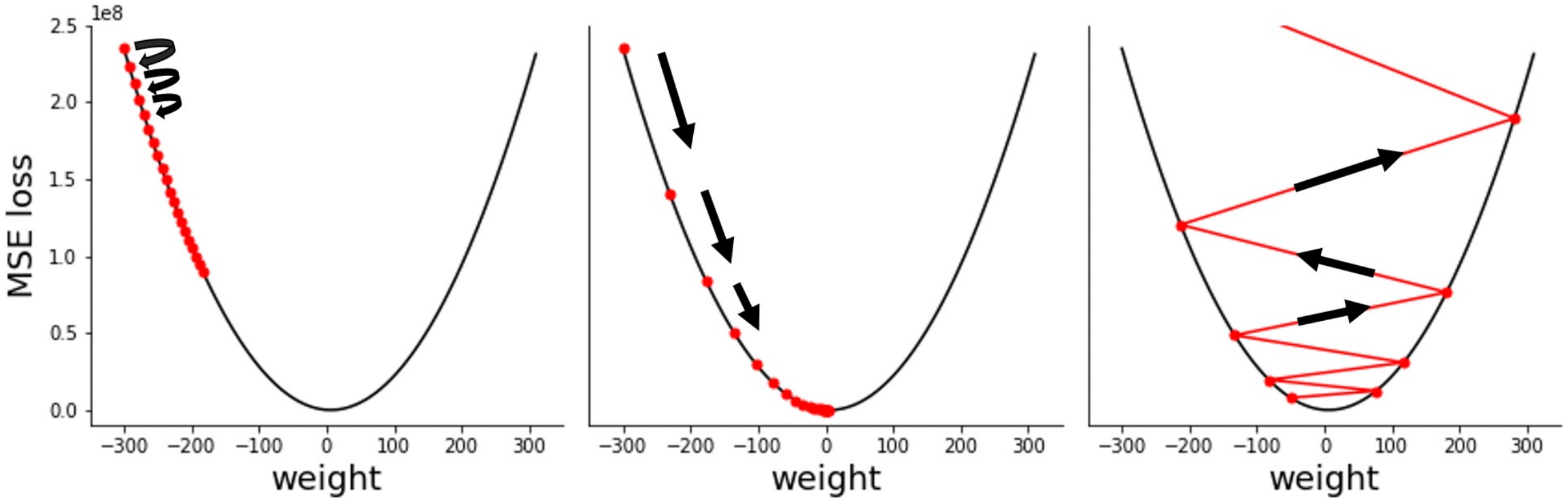
- Learning rate (step size) – by what amount we adjust/ tune/ update model's parameters (weights)

Learning rate

Large step size

Small step size

Learning rate



Gradient Descent Algorithm

Most important hyper-parameters to set:

- Learning rate (step size) – by what amount we adjust/ tune/ update model's parameters (weights)
- Number of itearations (epochs) - how many times we update model's weight

Number of itearations (epochs)

N.o. epochs is large

N.o. epochs is smal

Full (Batch) Gradient Descent

Linear regression:

- m datapoints
- feature matrix $\mathbf{X}$
- weight vector $\mathbf{w}$

Predictor:

$$h^{(\mathbf{w})}(\mathbf{x}) = \mathbf{X}\mathbf{w} = \hat{\mathbf{y}}$$

Loss:

$$f(\mathbf{w}) = \frac{1}{m}(\boldsymbol{y} - \hat{\mathbf{y}})^2$$

1. Forward Pass - compute prediction given the current weight
2. Backward Pass - compute derivative of the loss function
3. Update the weight

Full (Batch) Gradient Descent

$$\nabla f = -\frac{2}{m}\mathbf{X}^T(\boldsymbol{y} - \hat{\mathbf{y}})$$

Gradient Step

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha\nabla f\left(\mathbf{w}^{k}\right)$$

→ Slow when **X** is large

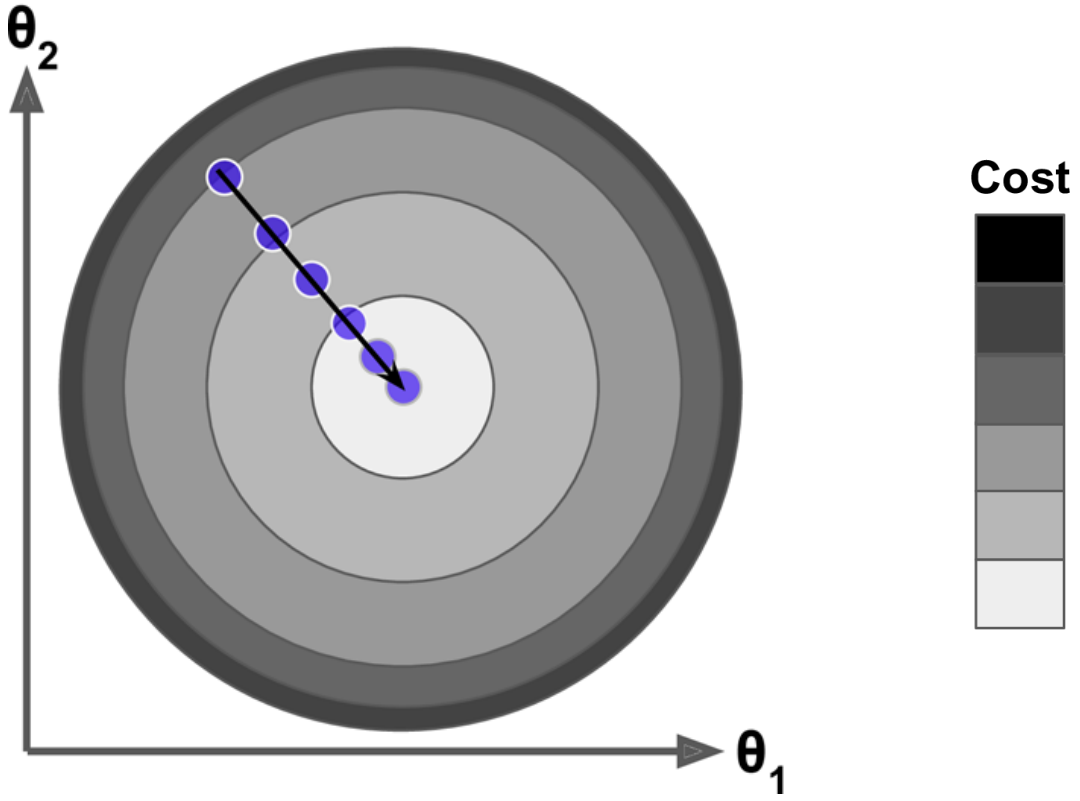


Figure 4-7. Gradient Descent with and without feature scaling
"Hands-On Machine Learning ..." A.Géron

Stochastic Gradient Descent

Noisy Gradient Step

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha g^{(k)}$$

$$g^{(k)} \approx \nabla f\left(\mathbf{w}^{k}\right)$$

$$g^{(k)} = -2\mathbf{x}\left(y^{(i)} - \hat{y}^{(\mathrm{i})}\right)$$

→ Fast, but very noisy

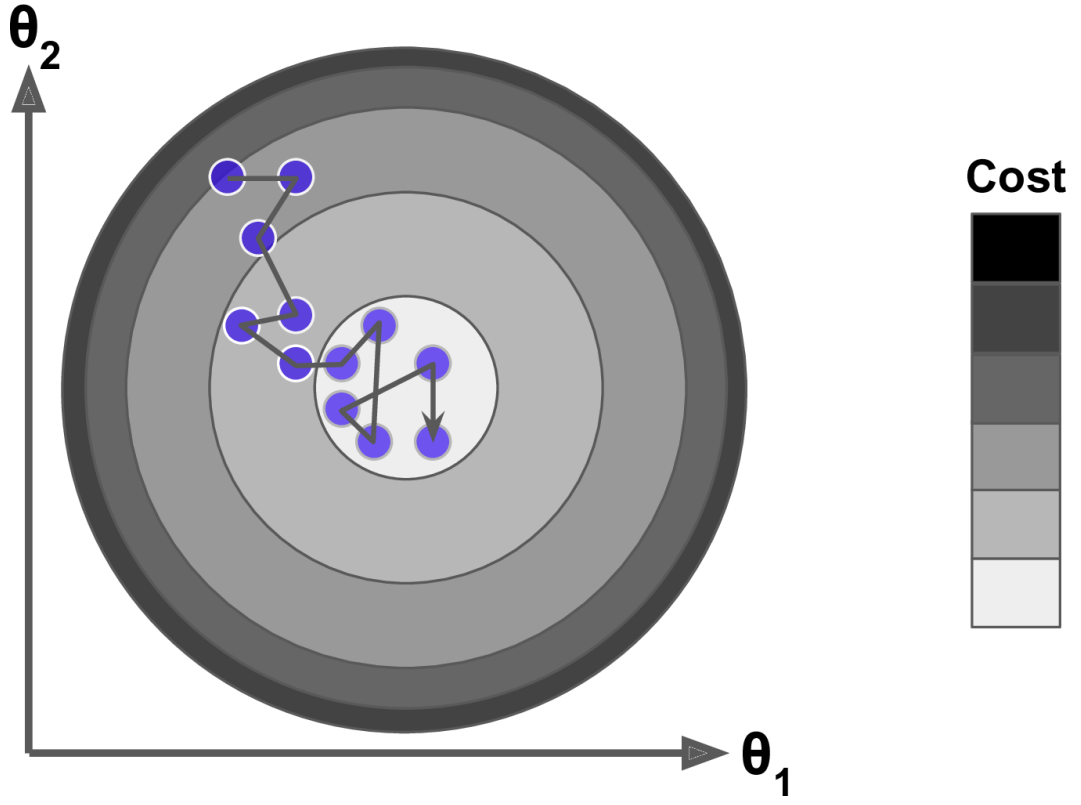


Figure 4-9. Stochastic Gradient Descent
"Hands-On Machine Learning ..." A.Géron

Mini-batch Gradient Descent

Noisy Gradient Step $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha g^{(k)}$

$$g^{(k)} \approx \nabla f\left(\mathbf{w}^{k}\right)$$

$$g^{(k)} = \frac{df}{d\mathbf{w}} = \frac{df}{d\hat{\mathbf{y}}}\frac{d\hat{\mathbf{y}}}{d\mathbf{w}} = -\frac{2}{s}\boldsymbol{S}^{T}(\boldsymbol{y} - \hat{\mathbf{y}})$$

$\mathbf{S}$ is a subset of $\boldsymbol{X}$ ($\mathbf{S} \in \boldsymbol{X}$)

Batch Gradient Descent

for epoch in epochs:

- Forward pass
- Compute loss
- Backward pass
- Update params

Mini-batch Gradient Descent

for epoch in epochs:
    for batch in batches:

- Forward pass
- Compute loss
- Backward pass
- Update params

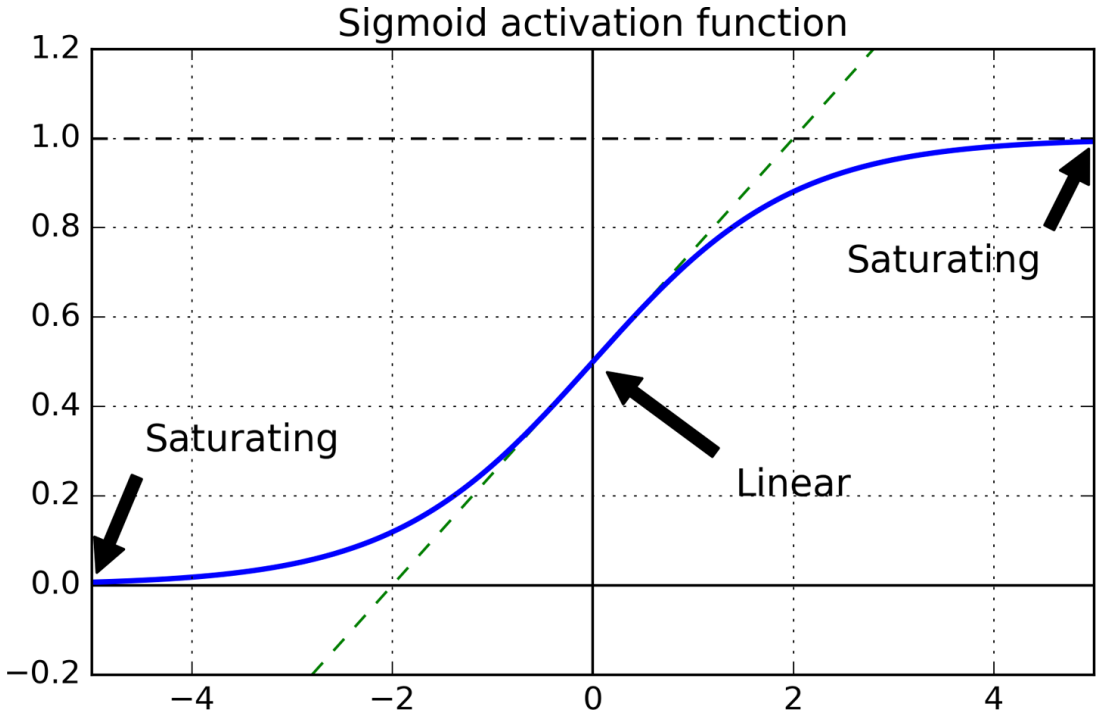
GD – unstable gradients

Early feedforward ANNs (hidden neurons with sigmoid activation + normal distribution for weights initialization):

1. Vanishing gradients – gradients getting smaller when flowing backwards → first layers are not trained

2. Exploding gradients – gradients get bigger and GD diverges

Activation functions

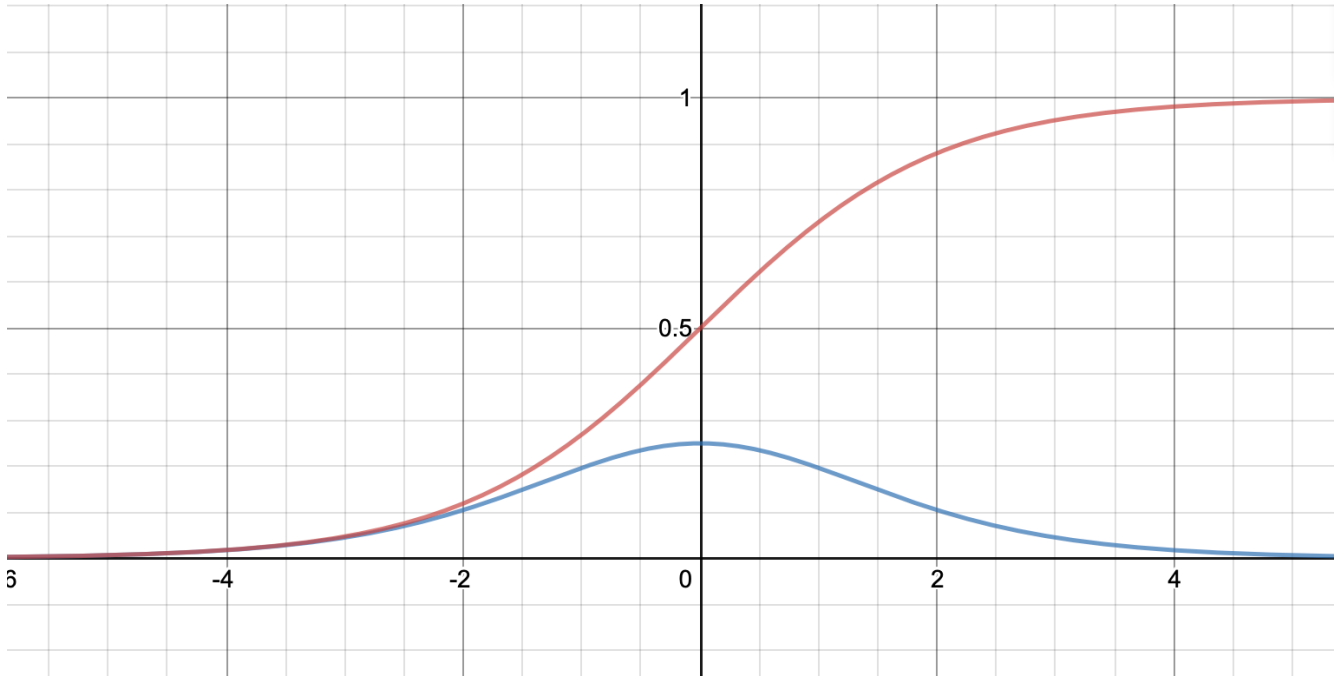


1. Output is in [0,1]
2. Saturating

Figure 11-1. Logistic activation function saturation
"Hands-On Machine Learning ..." A.Géron

sigmoid

$$p = \frac{1}{1 + e^{-z}}$$



local gradient

$$\frac{dp}{dz} = p(1 - p)$$

Full gradient

$$\nabla f(w) = \frac{df}{dw} = \frac{df}{dp}\frac{dp}{dz}\frac{dz}{dw}$$

Activation functions. Sigmoid.

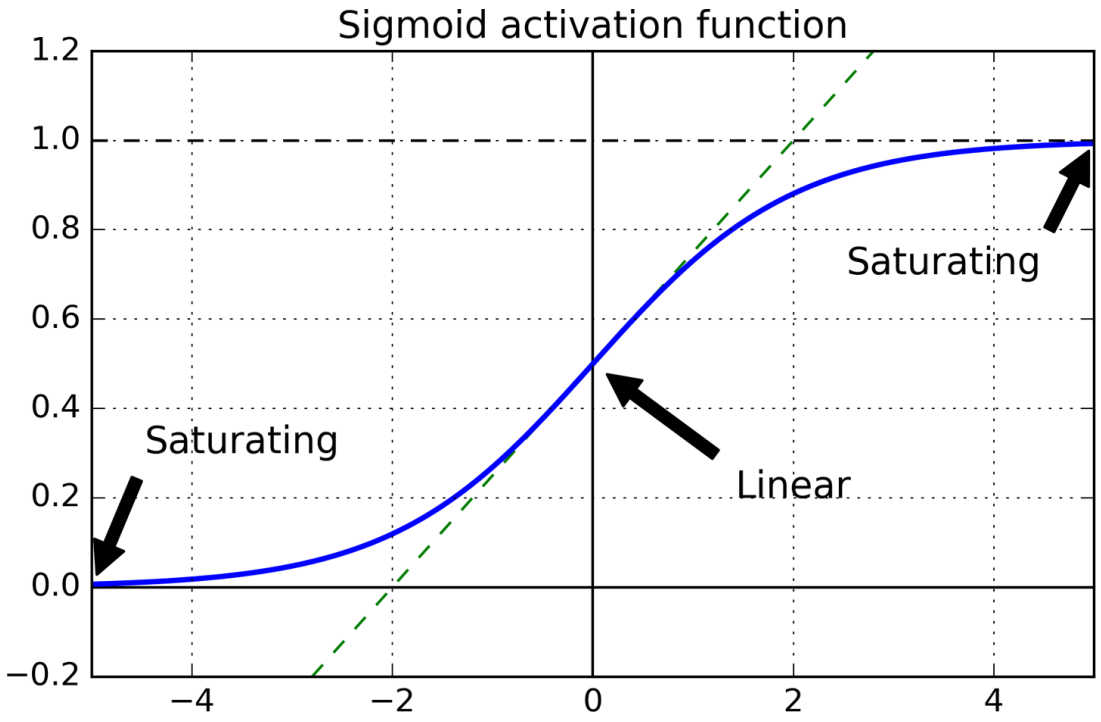


Figure 11-1. Logistic activation function saturation
"Hands-On Machine Learning ..." A.Géron

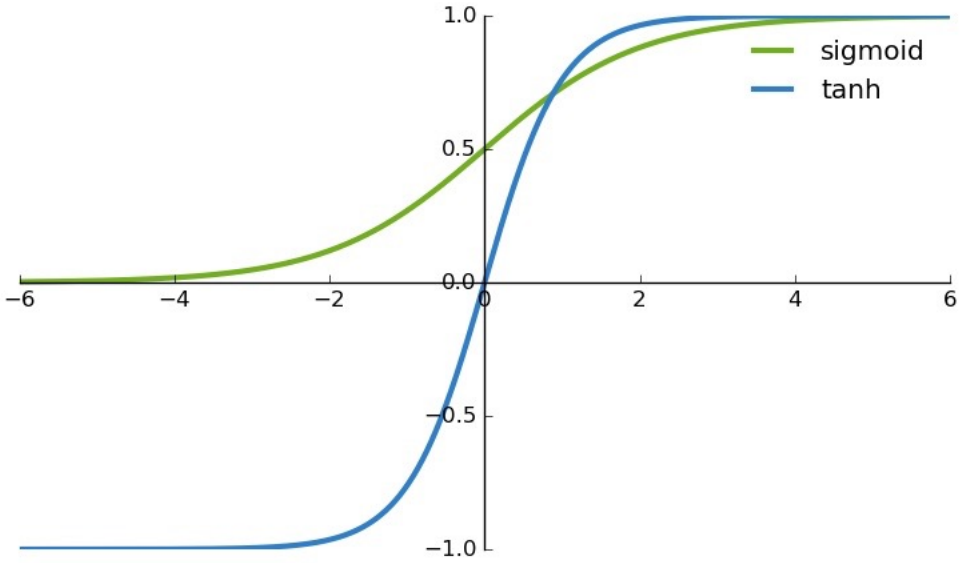
1. Output is in [0,1]

2. Saturating :(
→ can kill the gradient flow
→ first layers left untrained

3. Outputs are not zero-centered :(
→ can result in " zig-zagging" dynamics in the gradient updates

→ rarely used in hidden layers

Activation functions. Tanh.



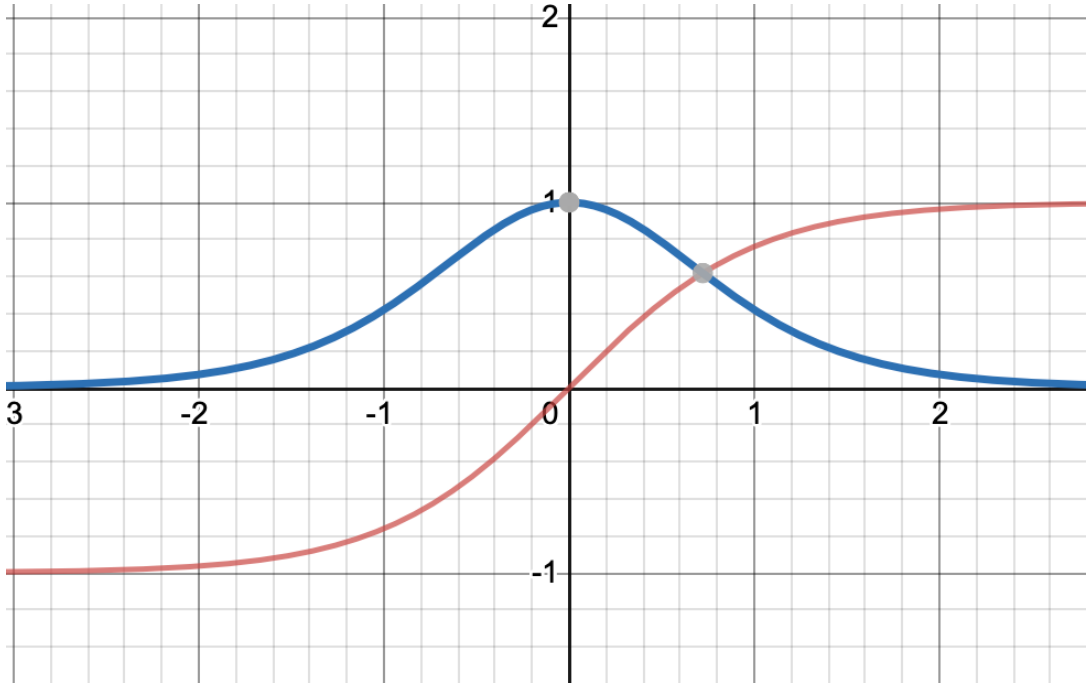
1. Output is in [-1,1]

2. Saturating :(

3. Outputs are zero-centered :)

tanh

$$g = \frac{2}{1 + e^{-2z}} - 1$$



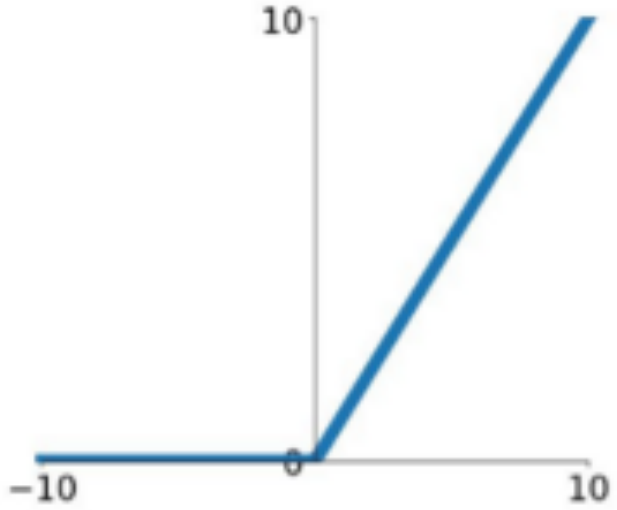
local gradient

$$\frac{dg}{dz} = 1 - g^2$$

Full gradient

$$\nabla f(w) = \frac{df}{dw} = \frac{df}{dg}\frac{dg}{dz}\frac{dz}{dw}$$

Activation functions. ReLU.

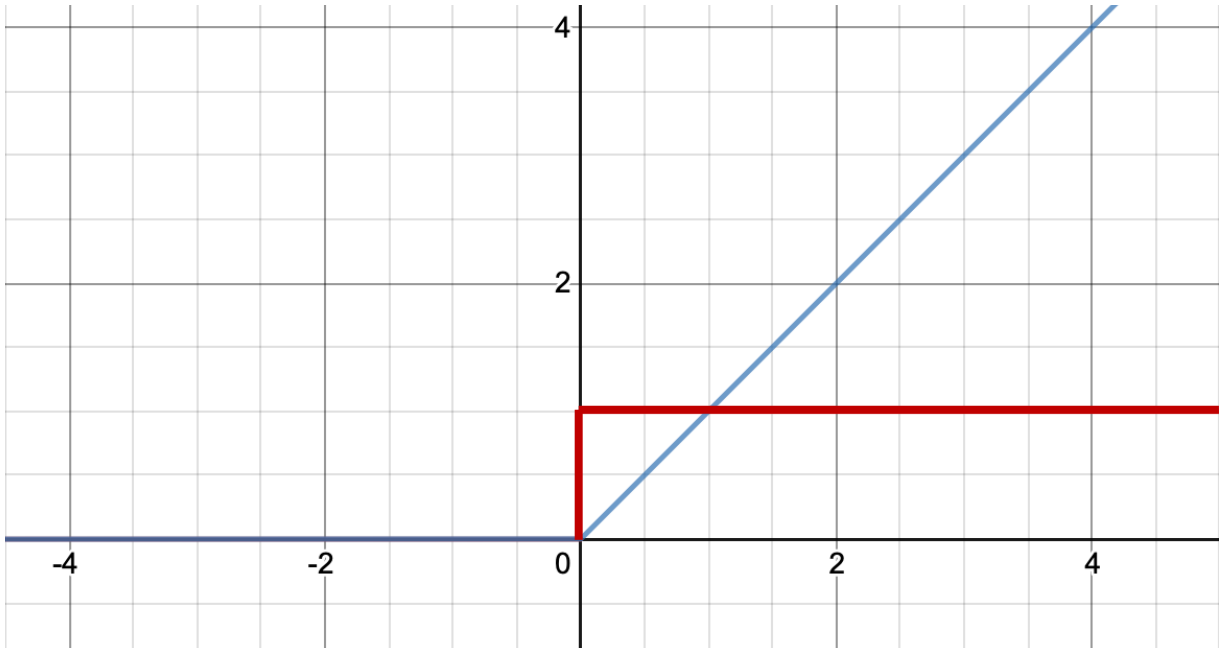


1. Output is in [-1,1]
2. Does not saturate :) (for x>0)
3. Easy to compute :)
4. Better in practice :) (faster convergence)

5. Outputs are not zero-centered :(
6. Saturation :((for x<0) (“dying” ReLUs)

ReLU

$$g = \max(0, z)$$



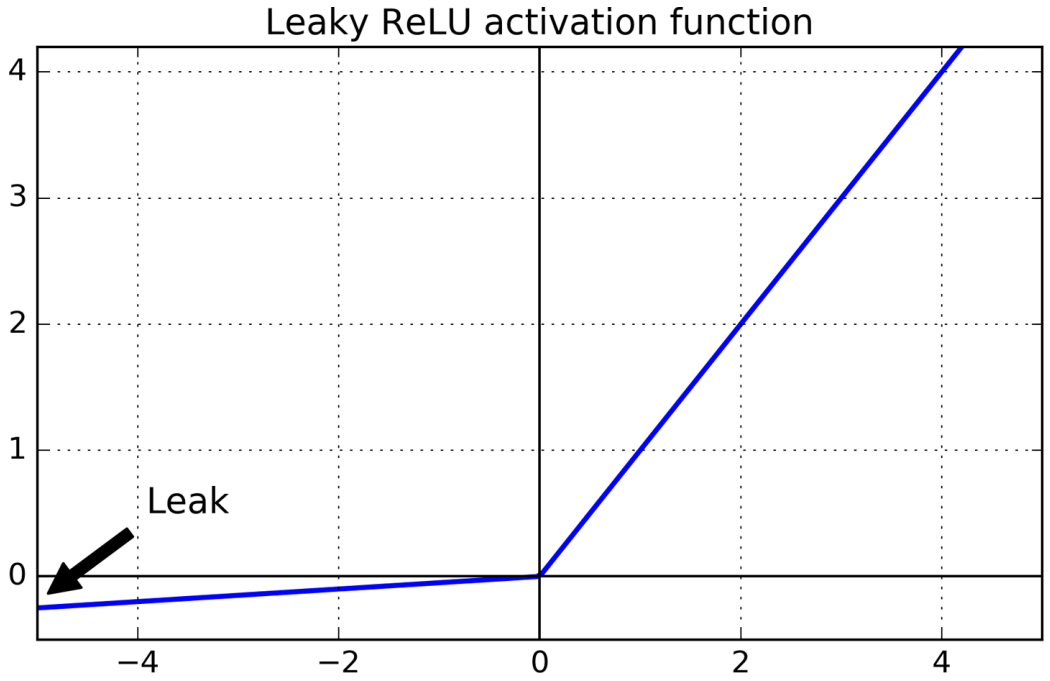
local gradient

$$\frac{dg}{dz} = \begin{cases} 0, z < 0 \\ 1, z \geq 0 \end{cases}$$

Full gradient

$$\nabla f(w) = \frac{df}{dw} = \frac{df}{dg}\frac{dg}{dz}\frac{dz}{dw}$$

Activation functions. Leaky ReLU.

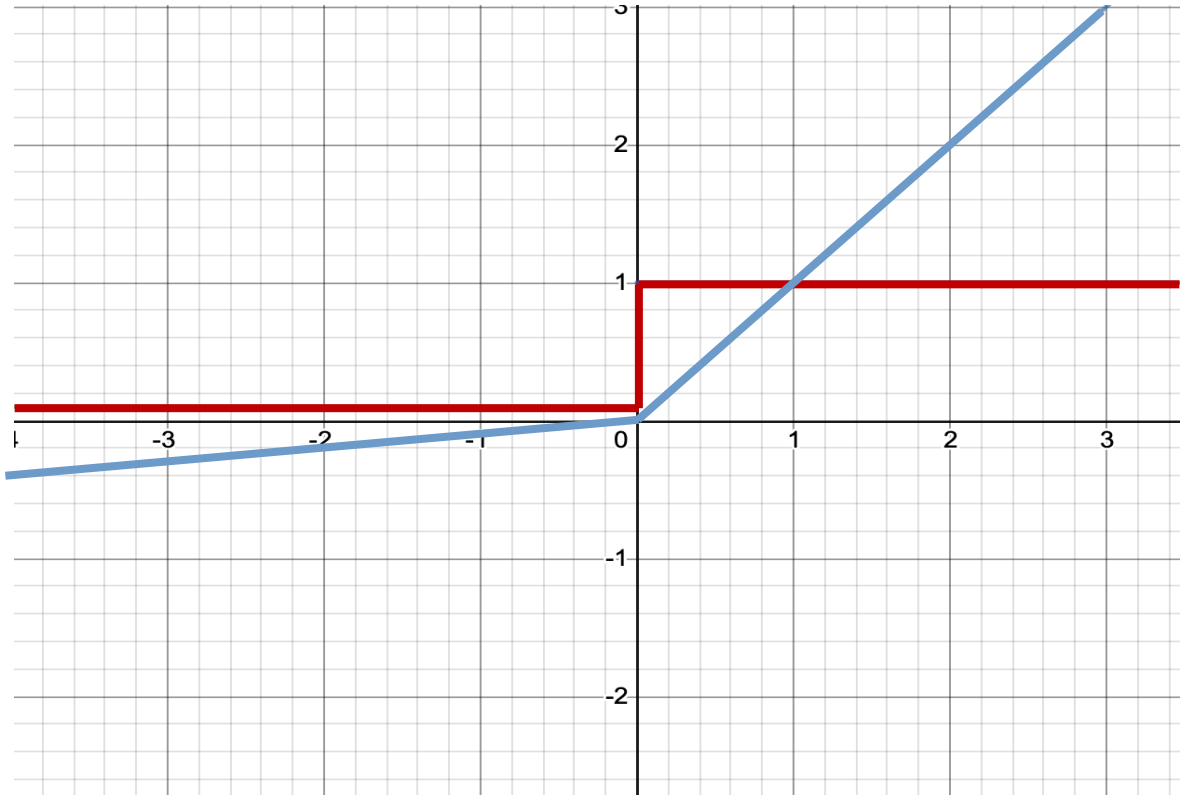


1. Does not saturate :)
2. Easy to compute :)
3. Better in practice :) (faster convergence)

Figure 11-2. Leaky ReLU
"Hands-On Machine Learning ..." A.Géron

Leaky ReLU

$$g = \max(\alpha z, z)$$



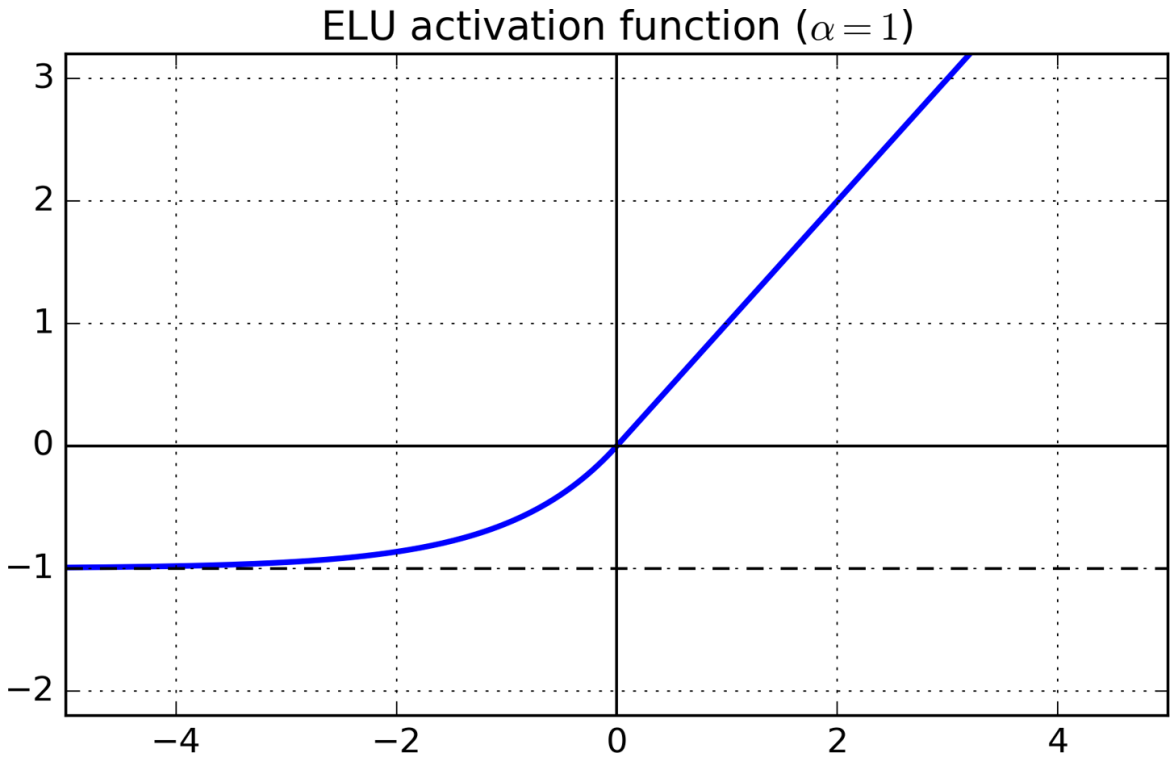
local gradient

$$\frac{dg}{dz} = \begin{cases} \alpha, z < 0 \\ 1, z \geq 0 \end{cases}$$

Full gradient

$$\nabla f(w) = \frac{df}{dw} = \frac{df}{dg}\frac{dg}{dz}\frac{dz}{dw}$$

Activation functions. ELU.



1. Does not saturate :)
2. Mean outputs closer to zero :)
3. Better in practice :) (faster convergence)
4. Slower to compute :(

Figure 11-3. ELU activation function
"Hands-On Machine Learning ..." A.Géron

Activation functions. TLTR.

1. Do not use sigmoid for hidden layers
2. Start with ReLU
3. Try out Leaky ReLU, ELU, etc.

Weight initialization

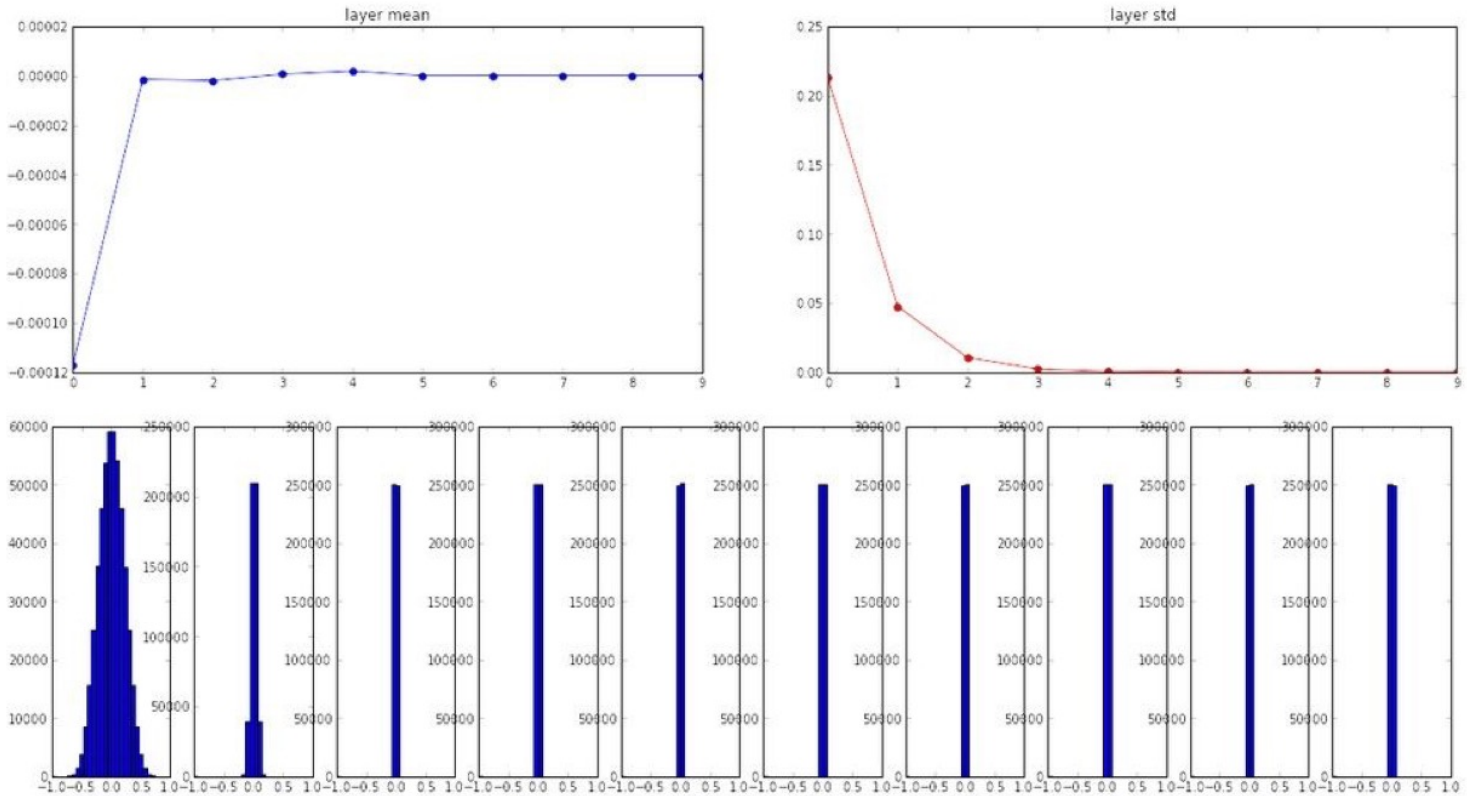
Poor weight initialization can also lead to unstable gradients:

1. All zero initialization → no source of asymmetry → same parameter updates

2. Symmetry breaking with:
   - small random numbers

Weight initialization with small random numbers

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```



http://cs231n.stanford.edu

Weight initialization

Poor weight initialization can also lead to unstable gradients:

1. All zero initialization → no source of asymmetry → same parameter updates

2. Symmetry breaking with:
   - small random numbers → can collapse to zero → no gradient signal
   - large random numbers → large output → can saturate (e.g. with tanh)

Weight initialization in Keras

Layer weight initializers

- Usage of initializers
- Available initializers
  - `RandomNormal` class
  - `RandomUniform` class
  - `TruncatedNormal` class
  - `Zeros` class
  - `Ones` class
  - `GlorotNormal` class
  - `GlorotUniform` class
  - `HeNormal` class
  - `HeUniform` class
  - `Identity` class
  - `Orthogonal` class
  - `Constant` class
  - `VarianceScaling` class
- Creating custom initializers
  - Simple callables
  - `Initializer` subclasses

`Dense` class

```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

https://keras.io/api/layers/initializers/

Feature scaling

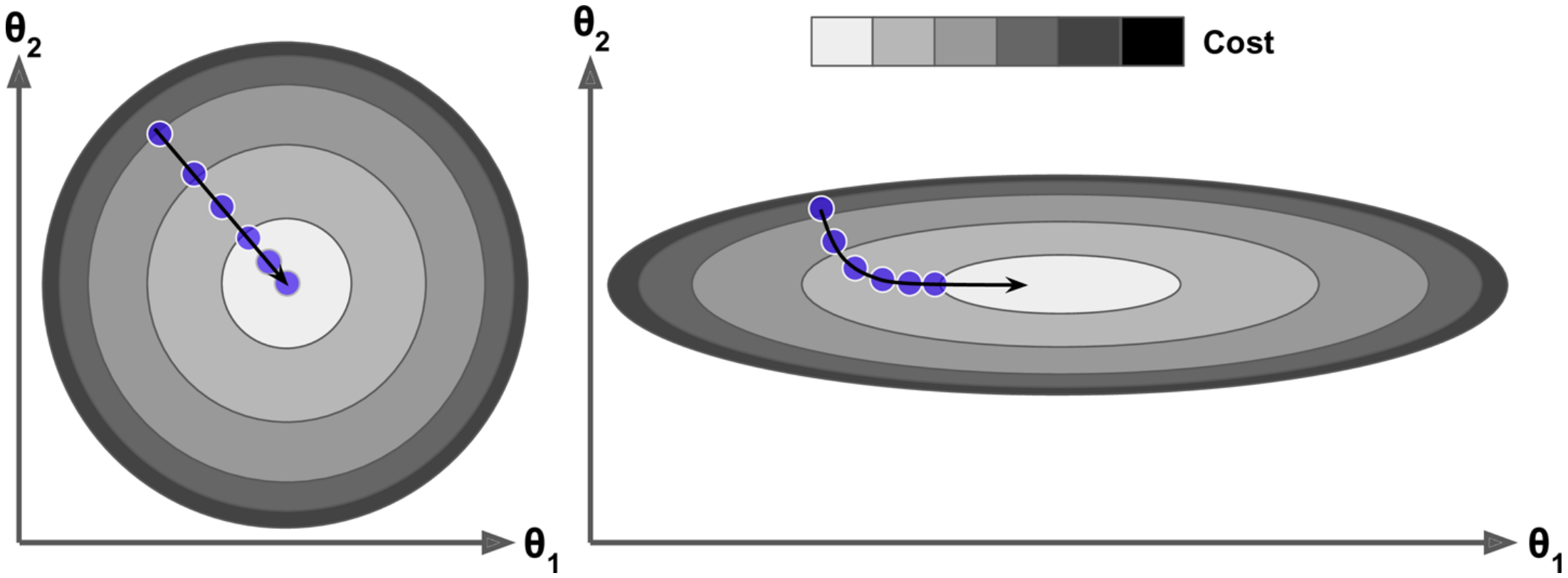


Figure 4-7. Gradient Descent with and without feature scaling
"Hands-On Machine Learning …" A.Géron

“We need the signal to flow properly in both directions: in the forward direction when making predictions, and in the reverse direction when backpropagating gradients. We don’t want the signal to die out, nor do we want it to explode and saturate.”

Ch.11“Hands-On Machine Learning …” A.Géron

Control the signal flow by choosing appropriate :

- activation function
- weight initialization
- learning rate
- data scaling/ normalization