Training neural networks

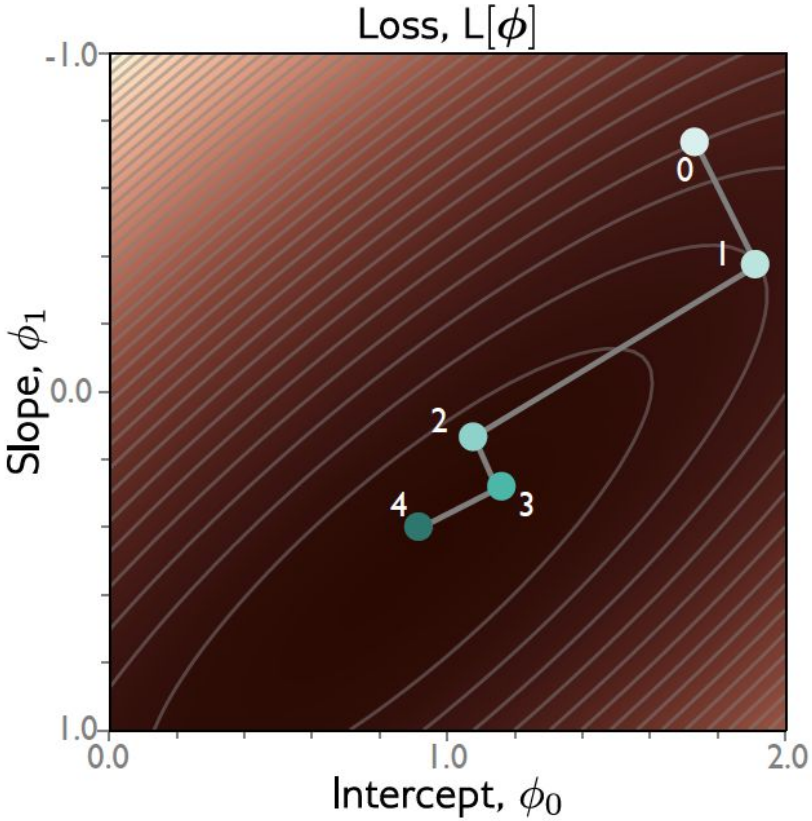
Florin Gogianu
Bitdefender ML team

* using slides, illustrations and ideas from prof. Simon Prince UDLB, Stanford CS231n, LeCun, Canziani, NYU Deep Learning

Reading

- Lecture 1: chapters 1, 2, 8.1-3, 9.1, Prince, *Understanding Deep Learning*
- Lecture 2: chapters 3 and 4, UDL. LeCun, Bengio, Hinton, *Deep Learning* (paper)
- Lecture 3:
  - chapters 5, 6 and 7, UDL.
  - Goh, *Why Momentum Really Works?*

Training a simple model



1. Define a loss function
2. Compute the change in parameters required to make the loss smaller
3. Apply the change and get new parameters
4. Repeat from (2)

Loss functions

Loss function

- Training dataset of I pairs of input/output examples:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}$$

- Loss function or cost function measures how bad model is:

$$L\Big[\underbrace{\phi, \mathrm{f}[\mathbf{x}, \phi]}_{\text{model}}, \underbrace{\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}}_{\text{train data}}\Big]$$

Loss function

- Training dataset of I pairs of input/output examples:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}$$

- Loss function or cost function measures how bad model is:

$$L[\phi]$$

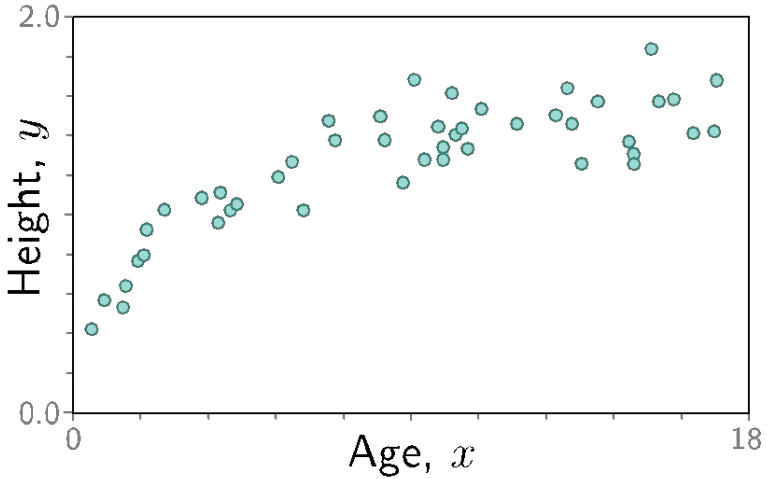
Returns a scalar that is smaller when model maps inputs to outputs better

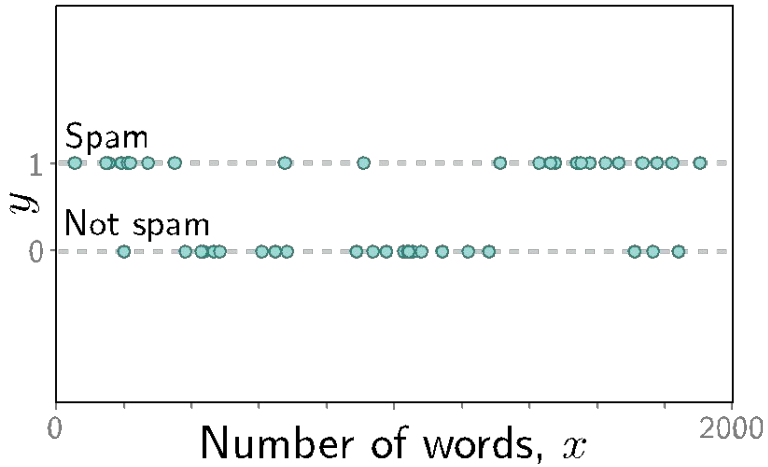
Loss function as an optimization objective

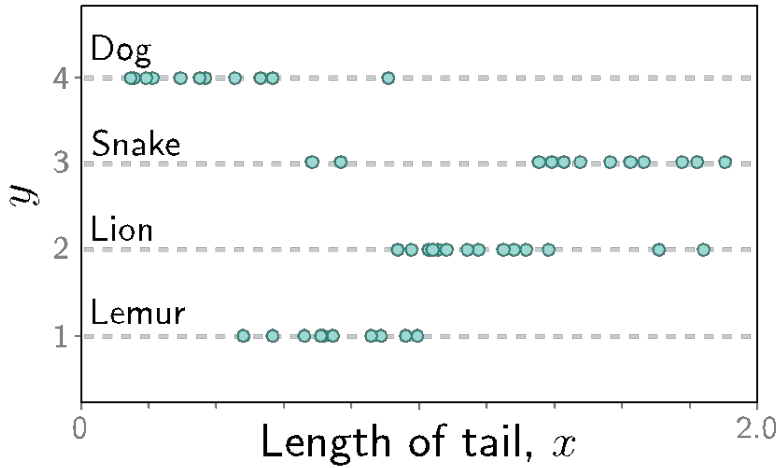
$$L\Big[\underbrace{\phi, \mathrm{f}[\mathbf{x}, \phi]}_{\text{model}}, \underbrace{\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}}_{\text{train data}}\Big]$$

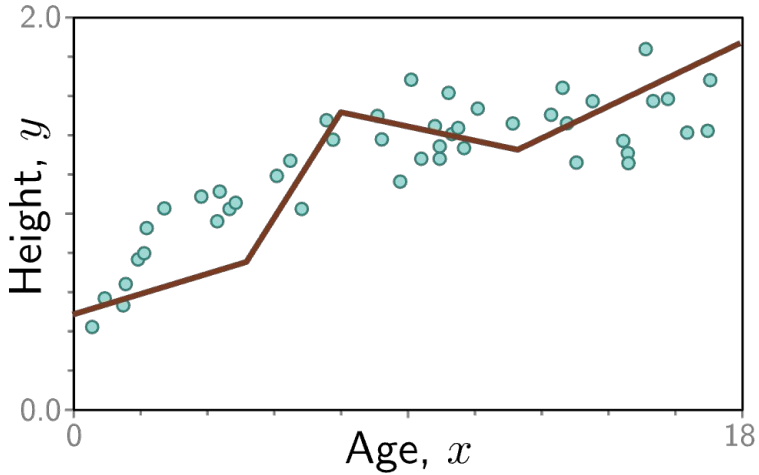
Find the parameters that minimize the loss:

$$\phi = \underset{\phi}{\operatorname{argmin}} \left[L(\phi) \right]$$


2.0
0.0
Height, y
0
18
Age, x

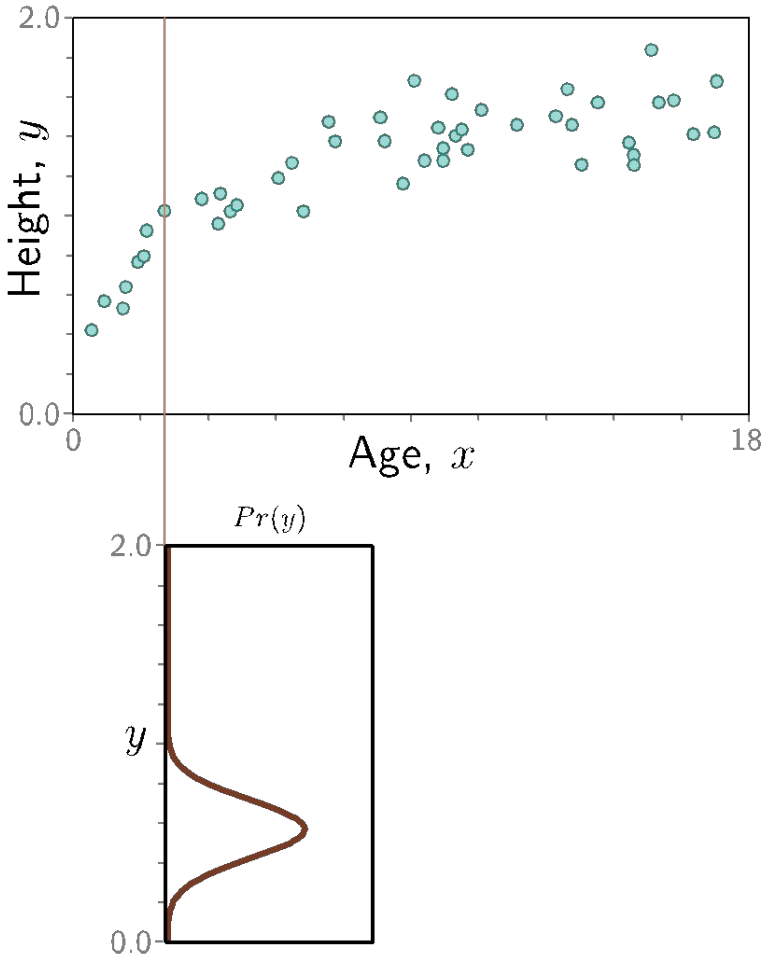

Spam
Not spam
1
0
y
0
2000
Number of words, x

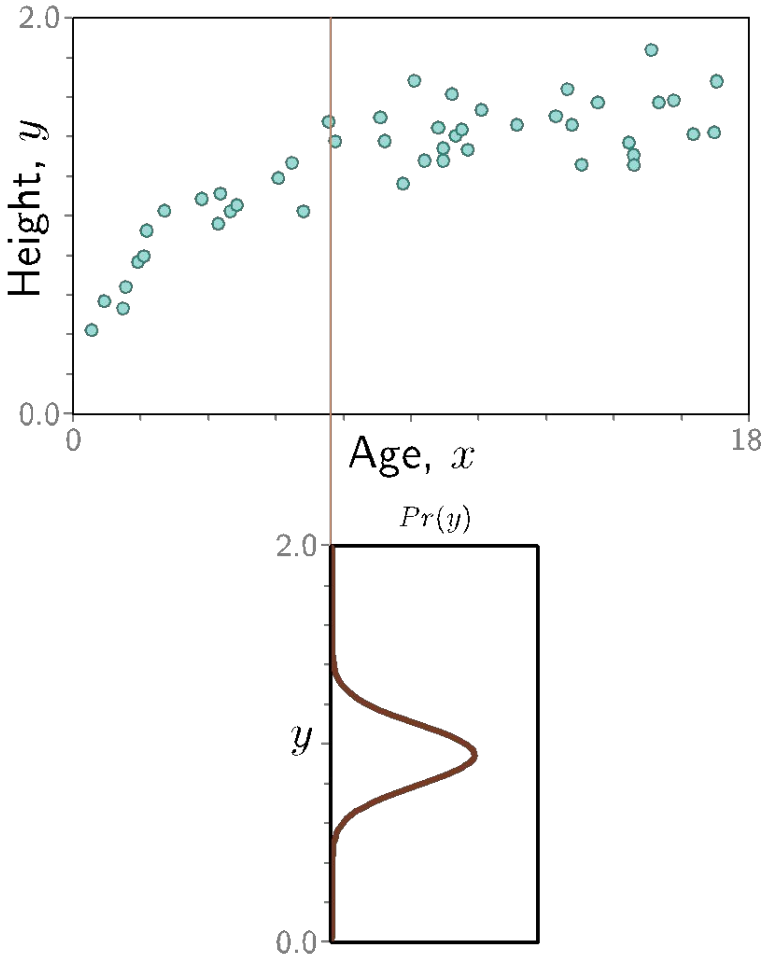

Dog
Snake
Lion
Lemur
4
3
2
1
y
0
2.0
Length of tail, x

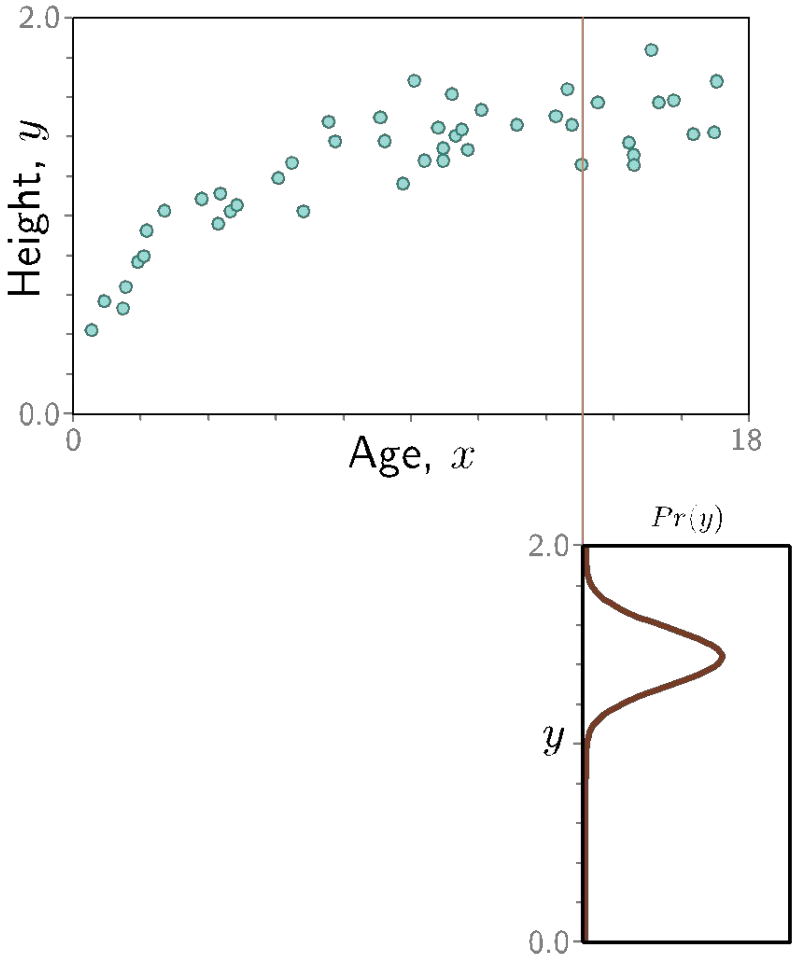


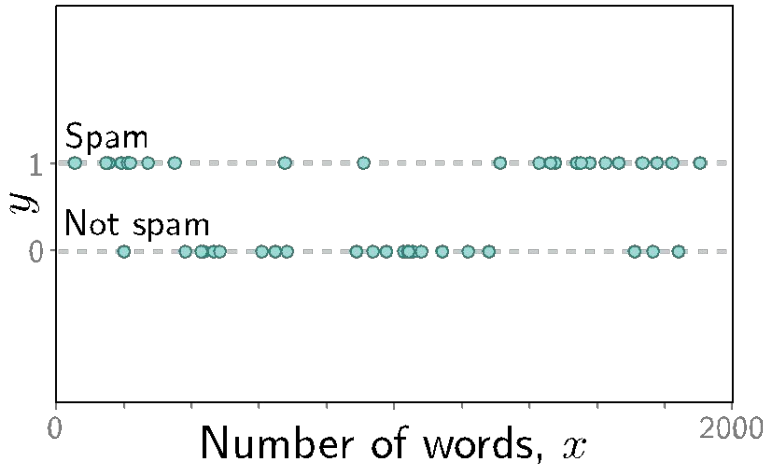
?

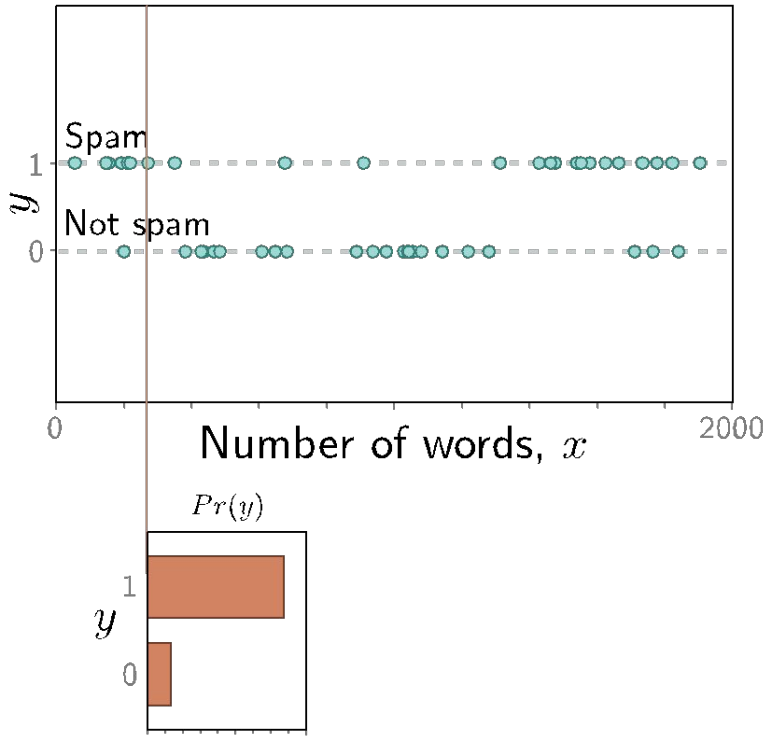
What do we mean,
probabilistically,
about fitting y = f(x)?

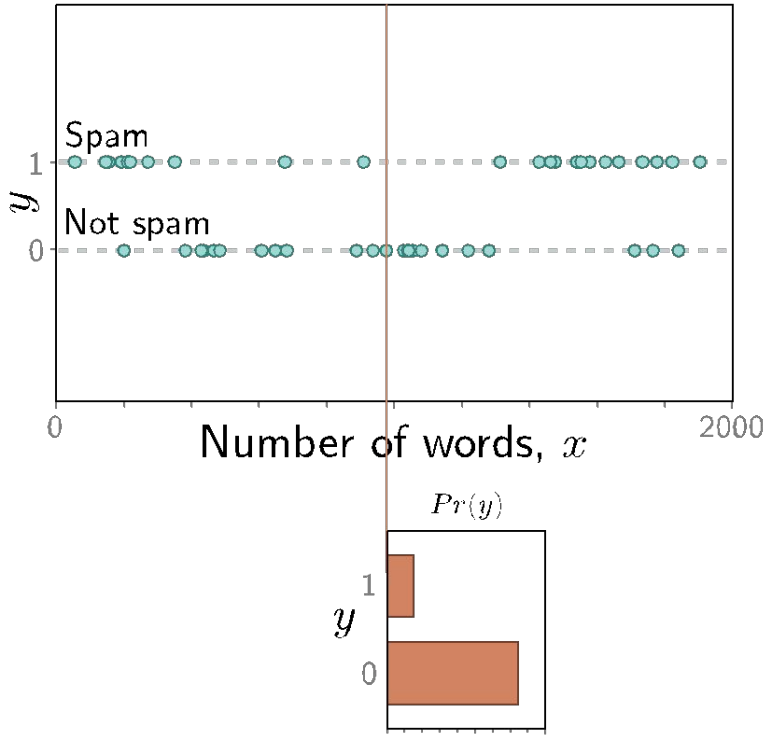

2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0

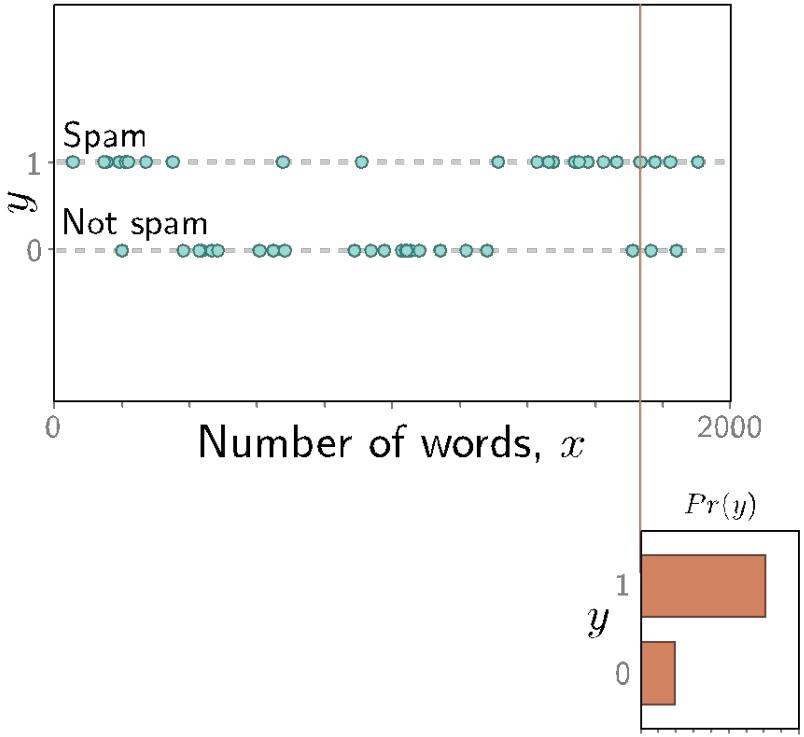

2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0

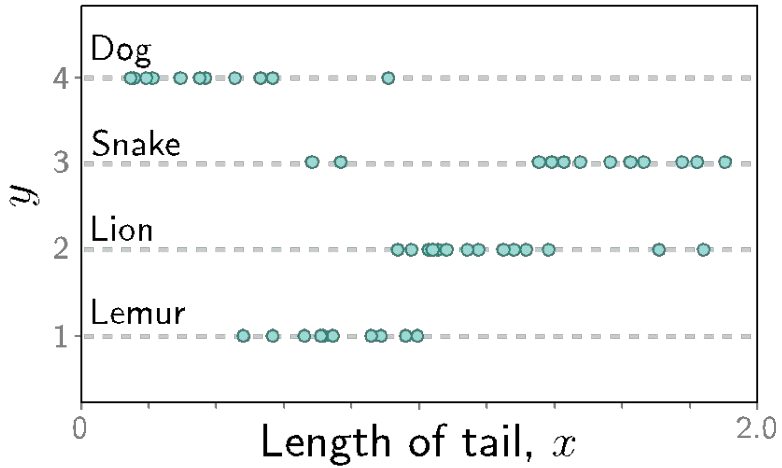

2.0
0.0
Height, y
0
18
Age, x
$Pr(y)$
2.0
y
0.0

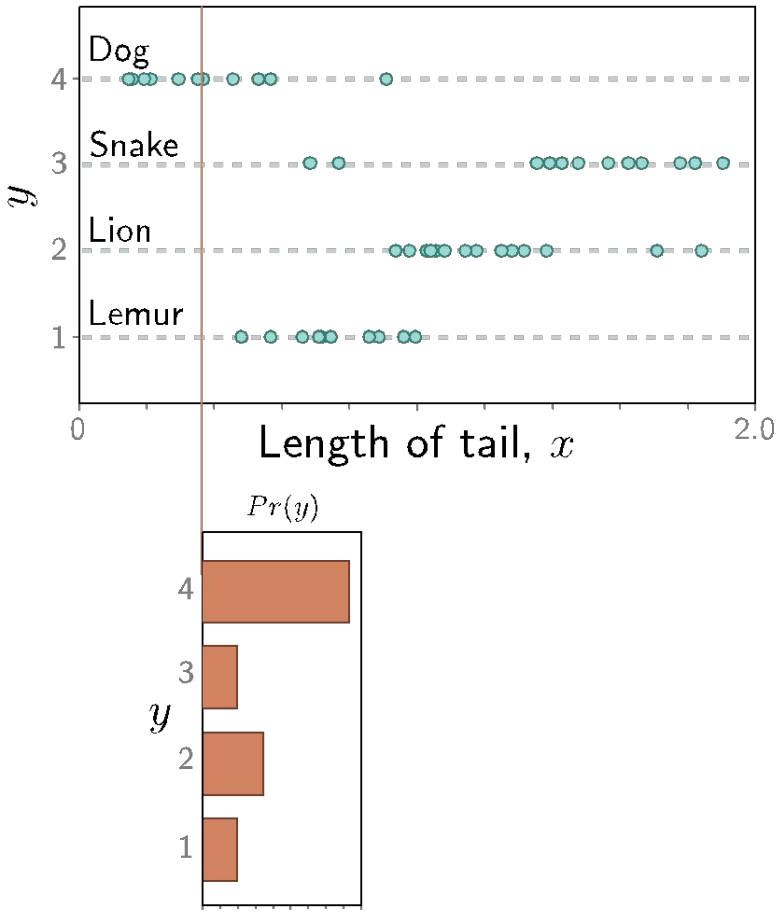

Spam
Not spam
1
0
y
0
2000
Number of words, x

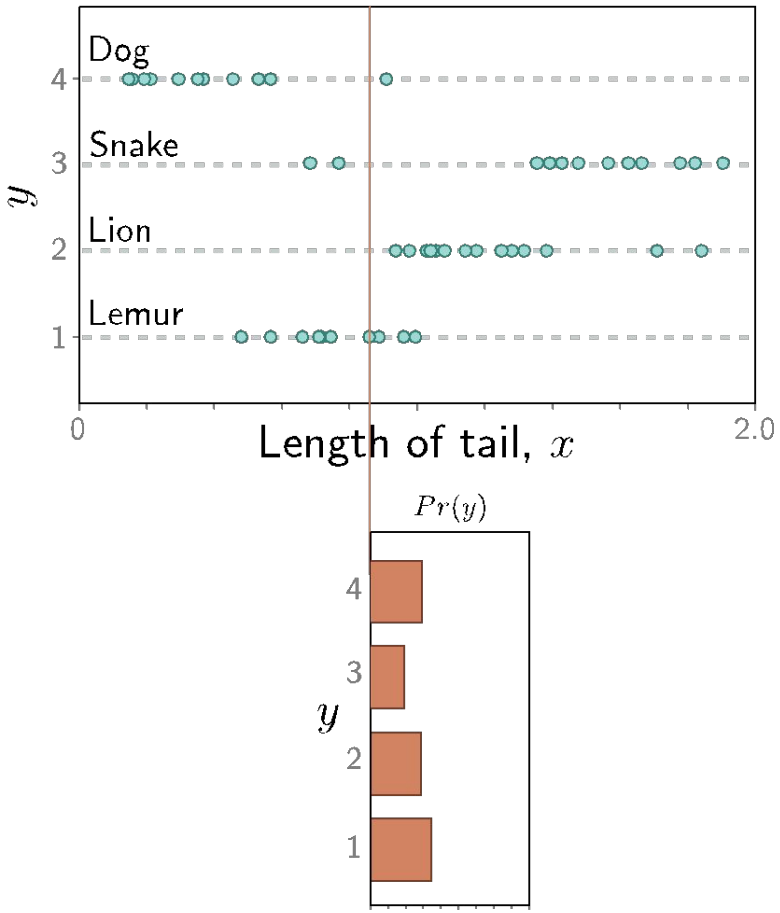

Spam
Not spam
1
0
y
0
2000
Number of words, x
$Pr(y)$
1
0
y

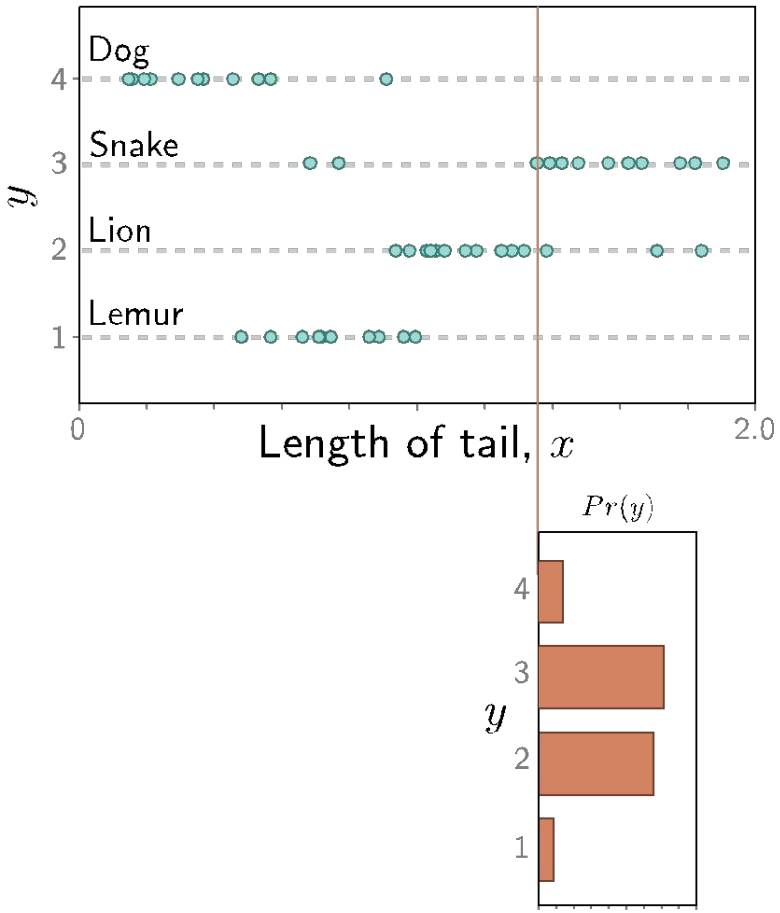

Spam
Not spam
1
0
y
0
2000
Number of words, x
$Pr(y)$
1
0
y

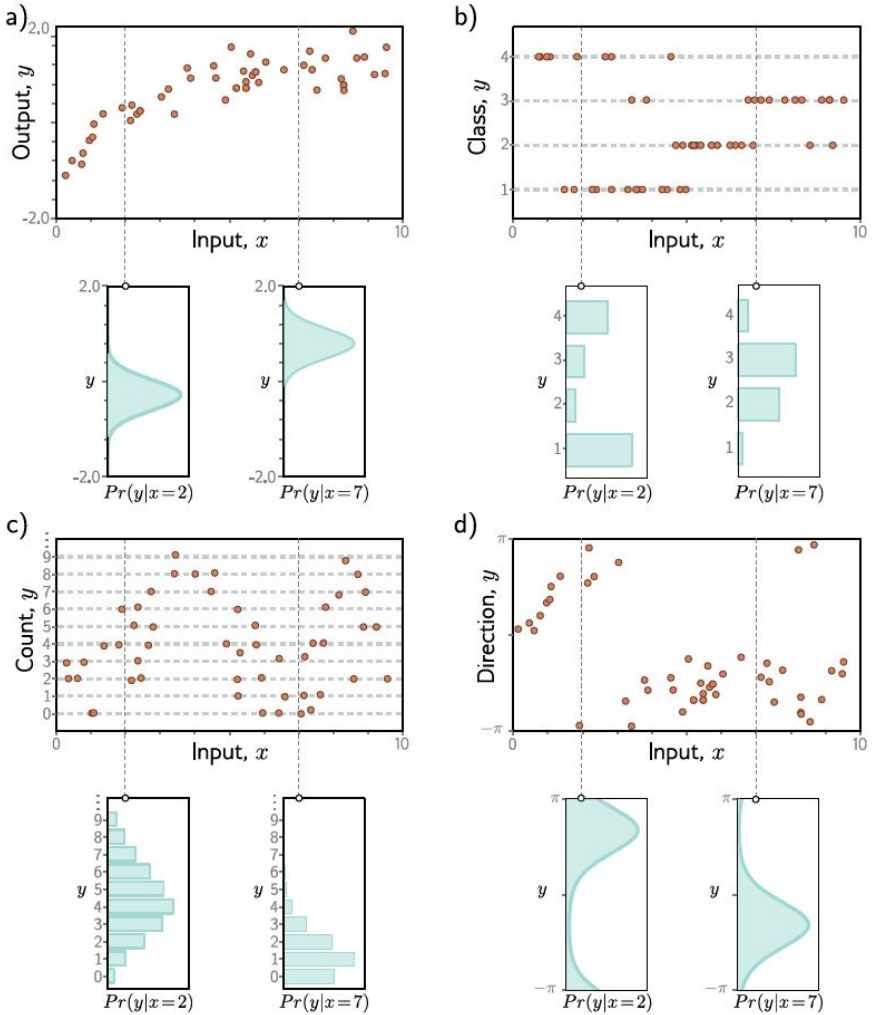

Spam
Not spam
1
0
y
0
2000
Number of words, x
$Pr(y)$
1
0
y


Dog
Snake
Lion
Lemur
4
3
2
1
y
0
2.0
Length of tail, x


Dog
Snake
Lion
Lemur
4
3
2
1
y
0
2.0
Length of tail, x
$Pr(y)$
4
3
2
1
y


Dog
Snake
Lion
Lemur
4
3
2
1
y
0
2.0
Length of tail, x
$Pr(y)$
4
3
2
1
y


Dog
Snake
Lion
Lemur
4
3
2
1
y
0
2.0
Length of tail, x
Pr(y)
4
3
2
1
y


a)
Output, y
Input, x
$Pr(y|x=2)$
$Pr(y|x=7)$
b)
Class, y
Input, x
$Pr(y|x=2)$
$Pr(y|x=7)$
c)
Count, y
Input, x
$Pr(y|x=2)$
$Pr(y|x=7)$
d)
Direction, y
Input, x
$Pr(y|x=2)$
$Pr(y|x=7)$

Encourage each prediction (model output) $\mathbf{y}_i$

to have high *probability* *conditioned* on (model input) $\mathbf{x}_i$

A general recipe for constructing loss functions

- ~~Model predicts output y given x~~
- Model predicts a *conditional probability* distribution:

$$Pr\left(\mathbf{y} \mid \mathbf{x}\right)$$

- Loss function aims to make the outputs have high probability unde the conditional distribution

From model output to probability distribution

- Pick a known distribution (eg.: Normal) to model output y with with parameters $\boldsymbol{\theta} = \{\mu, \sigma^2\}$

$$Pr\left(\mathbf{y} \mid \mathbf{x}\right)$$

- Fit the model so that it predicts the parameters μ and σ of the distribution.

Maximum likelihood criterion

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{x}_i)\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\boldsymbol{\theta}_i)\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\right].\end{aligned}$$

When we consider this probability as a function of the parameters ϕ, we call it a likelihood.

Problem:

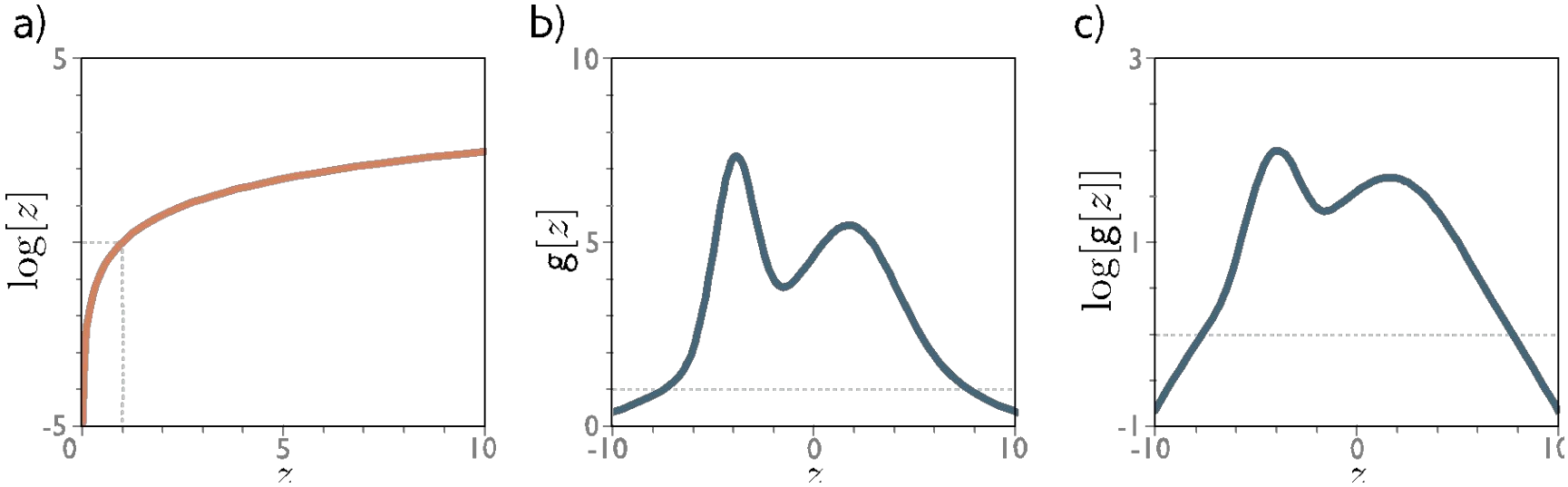
$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \left[\prod_{i=1}^{I} Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \right].$$

- The terms in this product might all be small
- The product might get so small that we can't easily represent it

Maximum log likelihood

$$\begin{aligned}\hat{\boldsymbol{\phi}} &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\log\left[\prod_{i=1}^{I} Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\right]\right] \\ &= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\left[\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\boldsymbol{\phi}])\Big]\right]\end{aligned}$$

The log function is monotonic



Minimizing negative log likelihood

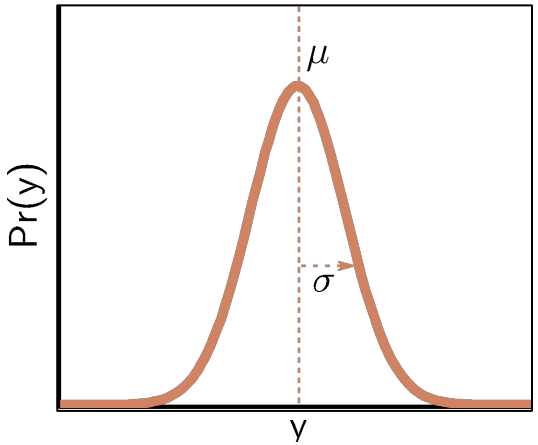
By convention, we minimize things (i.e., a loss)

$$\begin{aligned}\hat{\phi} &= \underset{\phi}{\text{argmax}}\left[\sum_{i=1}^{I}\log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\Big]\right] \\ &= \underset{\phi}{\text{argmin}}\left[-\sum_{i=1}^{I}\log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\Big]\right] \\ &= \underset{\phi}{\text{argmin}}\Big[\mathrm{L}[\phi]\Big]\end{aligned}$$

Inference (prediction)

- But now we predict a probability distribution
- We need an actual prediction (point estimate)
- Find the peak of the probability distribution (i.e., mean for normal)

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \left[Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]) \right]$$



From MLE to common loss functions

Go through 5.3 to 5.5 to convince yourselves that *minimizing the negative log likelihood* amounts to minimizing:

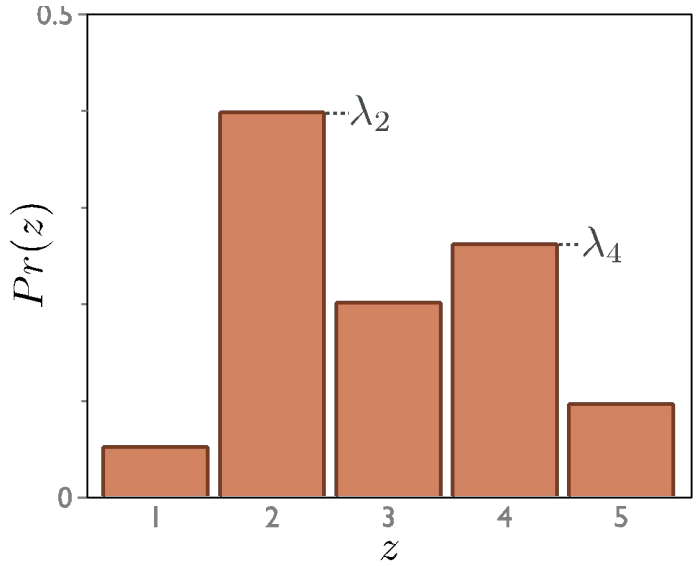
- Mean squared error (univariate regression / Gaussian)
- Binary cross-entropy (binary classification / Bernoulli)
- Cross-entropy (multiclass classification / Categorical)

Example: multiclass classification

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.

- Domain: $y \in \{1, 2, \ldots, K\}$
- Categorical distribution
- K parameters $\lambda_k \in [0,1]$
- Sum of all parameters = 1

$$Pr(y = k) = \lambda_k$$



Example: multiclass classification

2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}])$.

Problem:

- Output of neural network can be anything
- Parameters $\lambda_k \in [0,1]$, sum to one

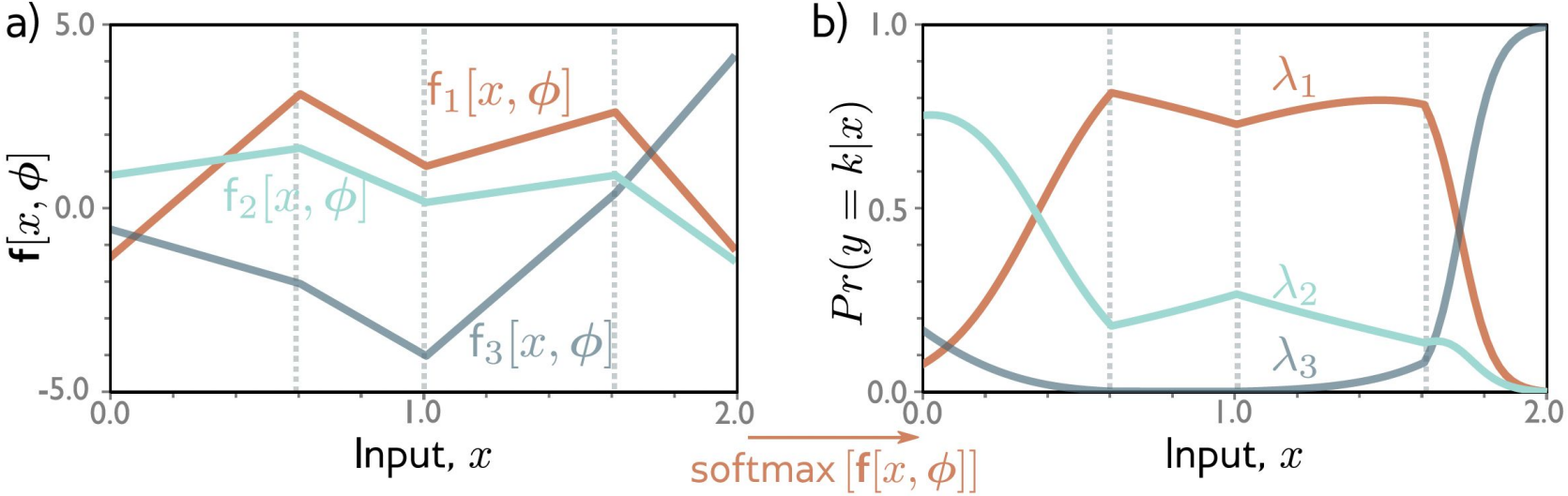
Solution:

- Pass through function that maps "anything" to [0,1], sum to one

$$\text{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K} \exp[z_{k'}]}$$

$$Pr(y = k|\mathbf{x}) = \text{softmax}_k[\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]]$$

Example: multiclass classification



$$Pr(y = k|\mathbf{x}) = \text{softmax}_k[\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]]$$

Example: multiclass classification

3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

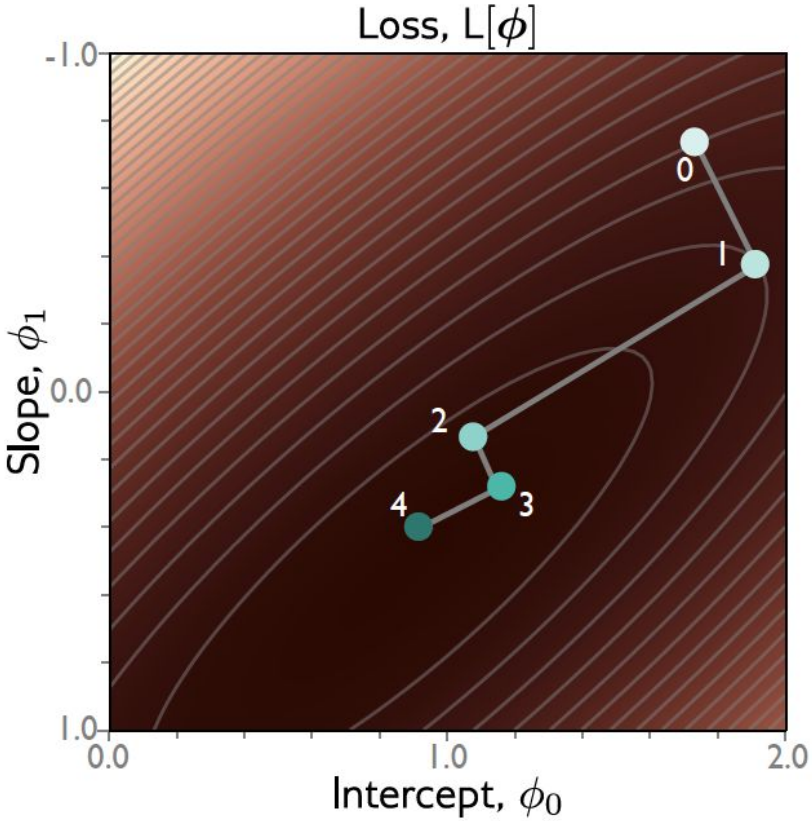
$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[L[\boldsymbol{\phi}]\right] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log\Big[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])\Big] \right]. \tag{5.7}$$

$$L[\boldsymbol{\phi}] = -\sum_{i=1}^{I} \log\left[\operatorname{softmax}_{y_i}\left[\mathbf{f}\left[\mathbf{x}_i, \boldsymbol{\phi}\right]\right]\right]$$

*Multiclass cross-entropy loss

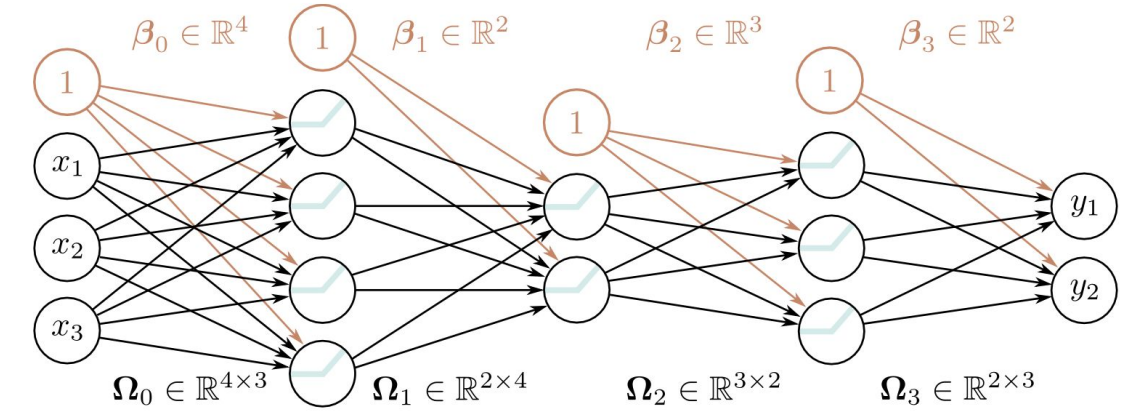
Computing gradients

Training a simple model



1. Define a loss function
2. Compute the change in parameters required to make the loss smaller
3. Apply the change and get new parameters
4. Repeat from (2)

Neural network



$$\mathbf{h}_1 = \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0\mathbf{x}]$$
$$\mathbf{h}_2 = \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1\mathbf{h}_1]$$
$$\mathbf{h}_3 = \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2\mathbf{h}_2]$$
$$\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}] = \boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3\mathbf{h}_3$$

Setup

Loss, sum of individual terms:

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \mathrm{l}[\mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}], y_i$$

SGD algorithm

$$\boldsymbol{\phi}_{t+1} \longleftarrow \boldsymbol{\phi}_t - \alpha \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

Parameters

$$\boldsymbol{\phi} = \{\boldsymbol{\beta}_0, \boldsymbol{\Omega}_0, \boldsymbol{\beta}_1, \boldsymbol{\Omega}_1, \boldsymbol{\beta}_2, \boldsymbol{\Omega}_2, \boldsymbol{\beta}_3, \boldsymbol{\Omega}_3\}$$

How to compute gradients?

$$\frac{\partial \ell_i}{\partial \boldsymbol{\beta}_k} \quad \text{and} \quad \frac{\partial \ell_i}{\partial \boldsymbol{\Omega}_k}$$

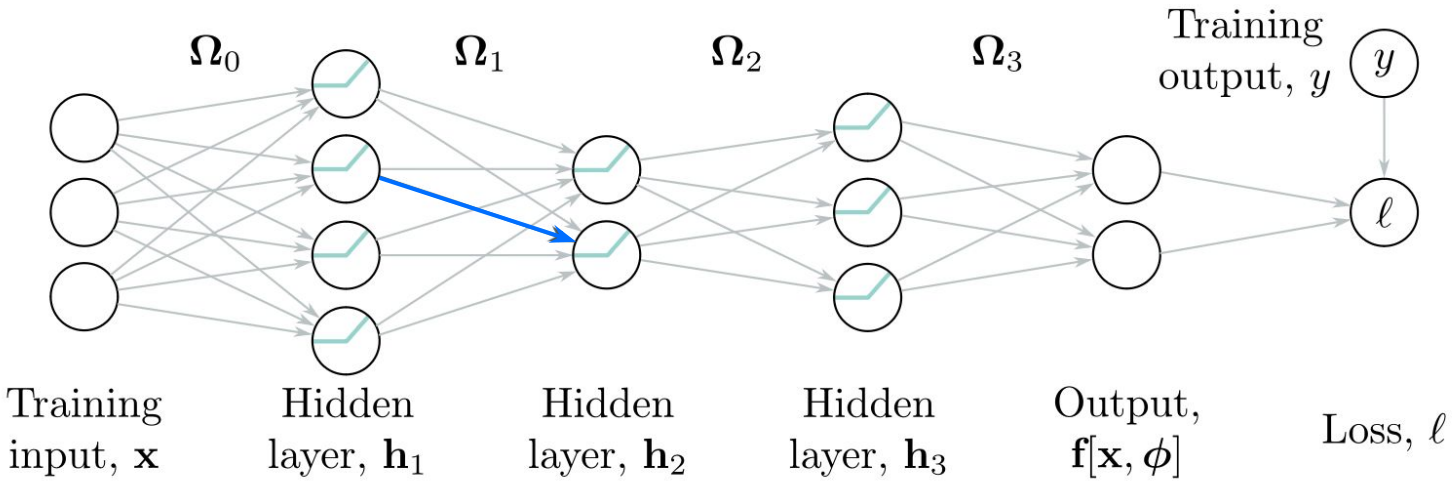
Big deal?

$$
\begin{aligned}
y' = \phi'_0 + \phi'_1 \mathrm{a}\left[\psi_{10} + \psi_{11}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{12}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{13}\mathrm{a}[\theta_{30} + \theta_{31}x]\right. \\
+ \phi'_2 \mathrm{a}[\psi_{20} + \psi_{21}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{22}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{23}\mathrm{a}[\theta_{30} + \theta_{31}x]] \\
\left. + \phi'_3 \mathrm{a}[\psi_{30} + \psi_{31}\mathrm{a}[\theta_{10} + \theta_{11}x] + \psi_{32}\mathrm{a}[\theta_{20} + \theta_{21}x] + \psi_{33}\mathrm{a}[\theta_{30} + \theta_{31}x]]\right]
\end{aligned}
$$

Huge equation, and we need to compute derivatives:

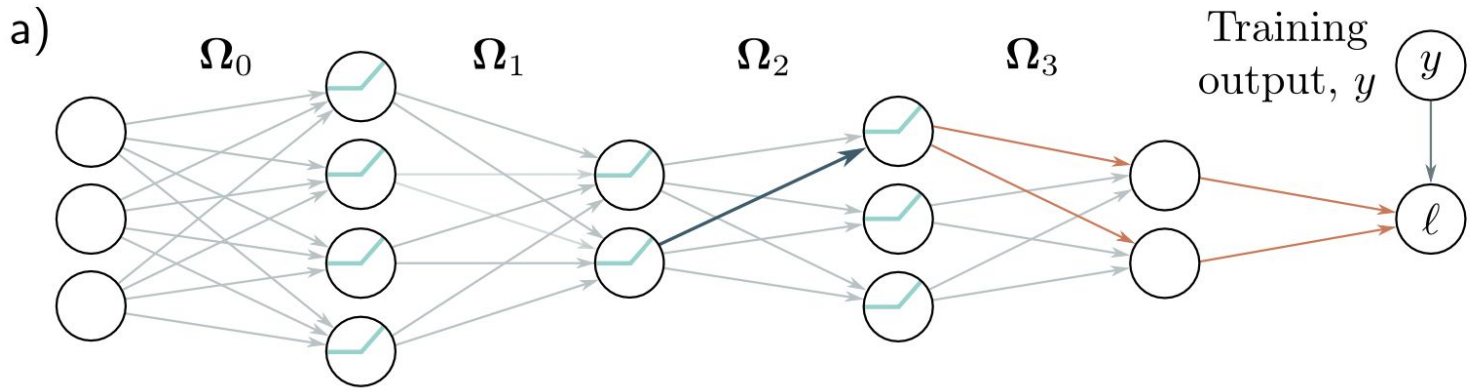
- for every parameter
- for every point in the batch
- for every iteration of SGD

Backpropagation algorithm. Forward pass



- Blue weight multiplies activation (ReLU output) in previous layer
- We want to know how change in blue weight affects loss
- If we double activation in previous layer, weight will have twice the effect
- Conclusion: we need to know the activations at each layer.

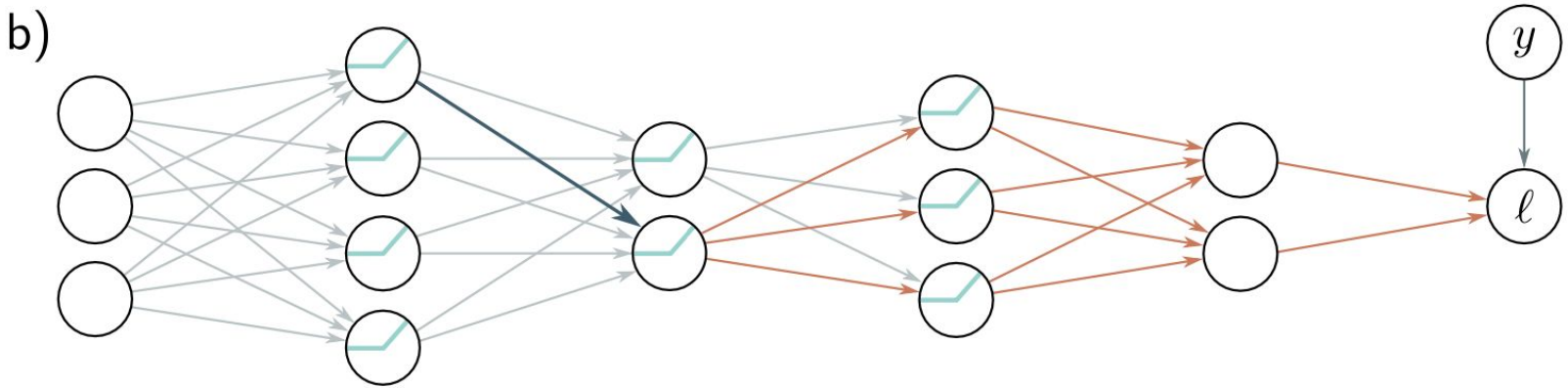
Backpropagation algorithm. Backward pass



To compute how a small change in a weight feeding into $\mathbf{h}_3$ modifies the loss, we need:

- How $\mathbf{h}_3$ changes the model output
- How the output changes the loss

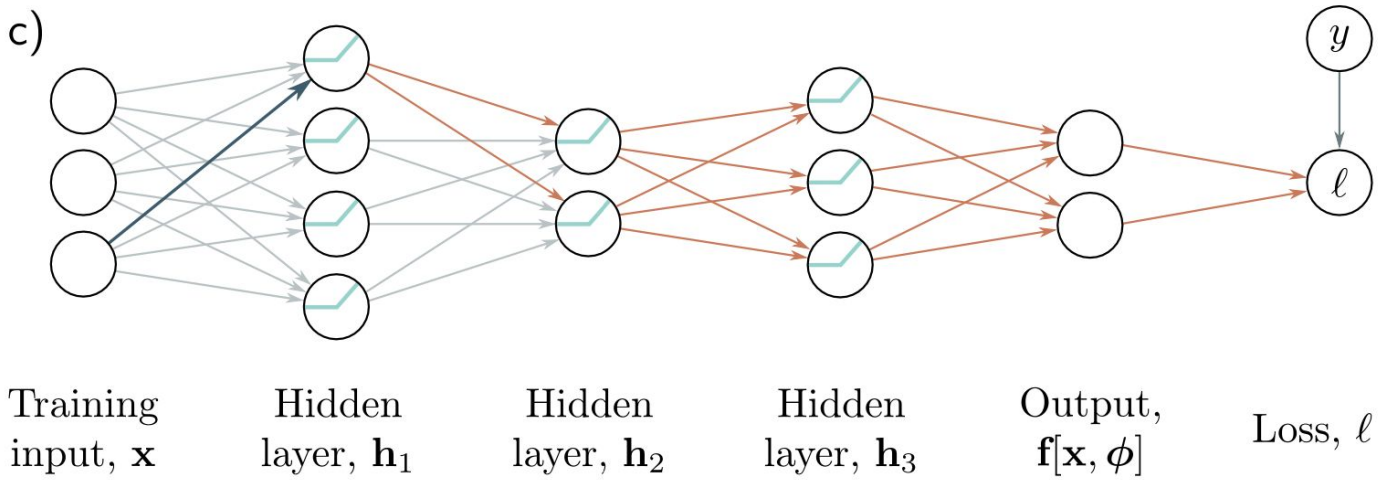
Backpropagation algorithm. Backward pass



To compute how a small change in a weight feeding into $\mathbf{h}_2$ modifies the loss, we need:

- How a change in layer $\mathbf{h}_2$ affects $\mathbf{h}_3$
- How $\mathbf{h}_3$ changes the model output
- How the output changes the loss

Backpropagation algorithm. Backward pass



To compute how a small change in a weight feeding into $\mathbf{h}_2$ modifies the loss, we need:

- How a change in layer $\mathbf{h}_1$ affects $\mathbf{h}_2$
- How a change in layer $\mathbf{h}_2$ affects $\mathbf{h}_3$
- How $\mathbf{h}_3$ changes the model output
- How the output changes the loss

Toy example

$$\mathrm{f}[x, \boldsymbol{\phi}] = \beta_3 + \omega_3 \cdot \cos\Big[\beta_2 + \omega_2 \cdot \exp\big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]\big]\Big]$$

$$\ell_i = (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

- A series of functions composed with each other
- Unlike in neural networks it consists of scalars and not vectors and matrices
- The “activation functions” are just $\sin$, $\exp$, $\cos$

Toy example

$$\mathrm{f}[x, \boldsymbol{\phi}] = \beta_3 + \omega_3 \cdot \cos\Big[\beta_2 + \omega_2 \cdot \exp\big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]\big]\Big]$$

$$\ell_i = (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

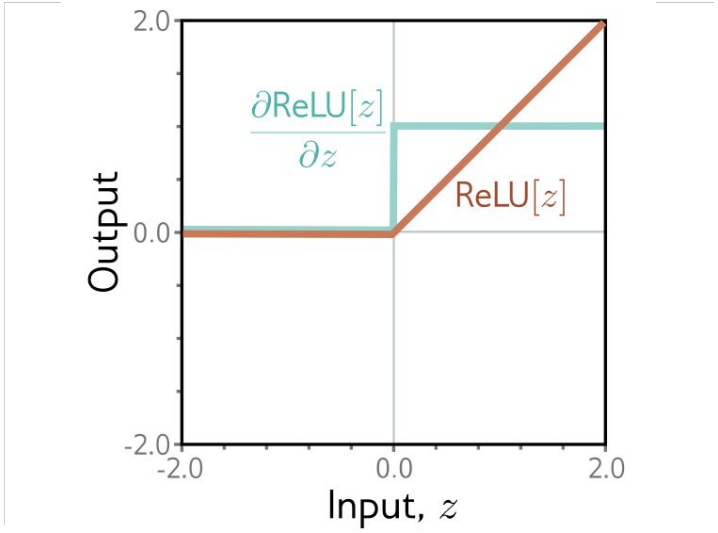
Derivatives of the activation functions

$$\frac{\partial \cos[z]}{\partial z} = -\sin[z] \qquad \frac{\partial \exp[z]}{\partial z} = \exp[z] \qquad \frac{\partial \sin[z]}{\partial z} = -\cos[z$$

Warmup! Derivative of ReLU

$$\mathrm{a}[z] = \mathrm{ReLU}[z] = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}.$$

Rectified Linear Unit



$$\mathbb{I}[z > 0]$$

Toy example

$$\mathrm{f}[x, \boldsymbol{\phi}] = \beta_3 + \omega_3 \cdot \cos\Big[\beta_2 + \omega_2 \cdot \exp\big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]\big]\Big]$$

$$\ell_i = (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

How does a small change in β_2 change the loss ℓ_i for the i'th example?

We want to compute:

$$\frac{\partial \ell_i}{\partial \beta_0}, \quad \frac{\partial \ell_i}{\partial \omega_0}, \quad \frac{\partial \ell_i}{\partial \beta_1}, \quad \frac{\partial \ell_i}{\partial \omega_1}, \quad \frac{\partial \ell_i}{\partial \beta_2}, \quad \frac{\partial \ell_i}{\partial \omega_2}, \quad \frac{\partial \ell_i}{\partial \beta_3}, \quad \text{and} \quad \frac{\partial \ell_i}{\partial \omega_3}$$

Gradients of composed functions

$$\mathrm{f}[x, \boldsymbol{\phi}] = \beta_3 + \omega_3 \cdot \cos\Big[\beta_2 + \omega_2 \cdot \exp\big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]\big]\Big]$$

$$\ell_i = (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

Calculating expressions by hand:

- Some expressions are very complicated
- There are some redundancies

$$\begin{aligned}\frac{\partial \ell_i}{\partial \omega_0} = \quad & -2\left(\beta_3 + \omega_3 \cdot \cos\Big[\beta_2 + \omega_2 \cdot \exp\big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x_i]\big]\Big] - y_i\right) \\ & \cdot \omega_1\omega_2\omega_3 \cdot x_i \cdot \cos[\beta_0 + \omega_0 \cdot x_i] \cdot \exp\Big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x_i]\Big] \\ & \cdot \sin\Big[\beta_2 + \omega_2 \cdot \exp\Big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x_i]\Big]\Big]\end{aligned}$$

Forward pass

$$\mathrm{f}[x, \boldsymbol{\phi}] = \beta_3 + \omega_3 \cdot \cos\Big[\beta_2 + \omega_2 \cdot \exp\big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]\big]\Big]$$

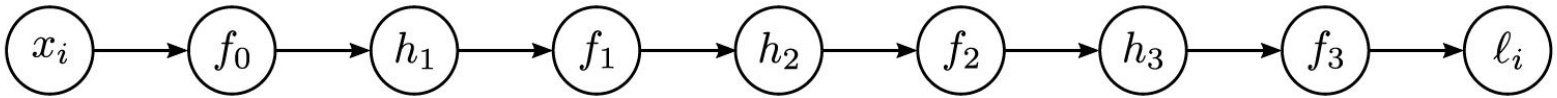
$$\ell_i = (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

1. Write it as a series of intermediate calculations

2. Compute these intermediate quantities

$$\begin{aligned} f_0 &= \beta_0 + \omega_0 \cdot x_i \\ h_1 &= \sin[f_0] \\ f_1 &= \beta_1 + \omega_1 \cdot h_1 \\ h_2 &= \exp[f_1] \end{aligned}$$

$$\begin{aligned} f_2 &= \beta_2 + \omega_2 \cdot h_2 \\ h_3 &= \cos[f_2] \\ f_3 &= \beta_3 + \omega_3 \cdot h_3 \\ \ell_i &= (f_3 - y_i)^2. \end{aligned}$$



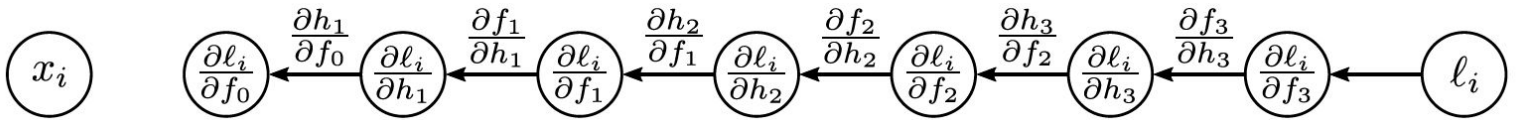
Backward pass

$$\mathrm{f}[x, \boldsymbol{\phi}] = \beta_3 + \omega_3 \cdot \cos\Big[\beta_2 + \omega_2 \cdot \exp\big[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]\big]\Big]$$

$$\ell_i = (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$\frac{\partial \ell_i}{\partial f_3}, \quad \frac{\partial \ell_i}{\partial h_3}, \quad \frac{\partial \ell_i}{\partial f_2}, \quad \frac{\partial \ell_i}{\partial h_2}, \quad \frac{\partial \ell_i}{\partial f_1}, \quad \frac{\partial \ell_i}{\partial h_1}, \quad \text{and} \quad \frac{\partial \ell_i}{\partial f_0}$$



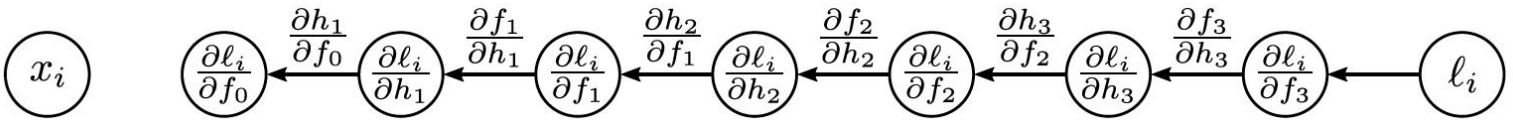
Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

The first are easy:

$$\frac{\partial \ell_i}{\partial f_3} = 2(f_3 - y_i)$$



Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

The rest are computed using the *chain rule*.

$$\frac{\partial \ell_i}{\partial h_3} = \frac{\partial f_3}{\partial h_3}\frac{\partial \ell_i}{\partial f_3}$$

How does a small change in $\mathbf{h}_3$ change ℓ_i?

How does a small change in $\mathbf{h}_3$ change $\mathbf{f}_3$?

How does a small change in $\mathbf{f}_3$ change ℓ_i?

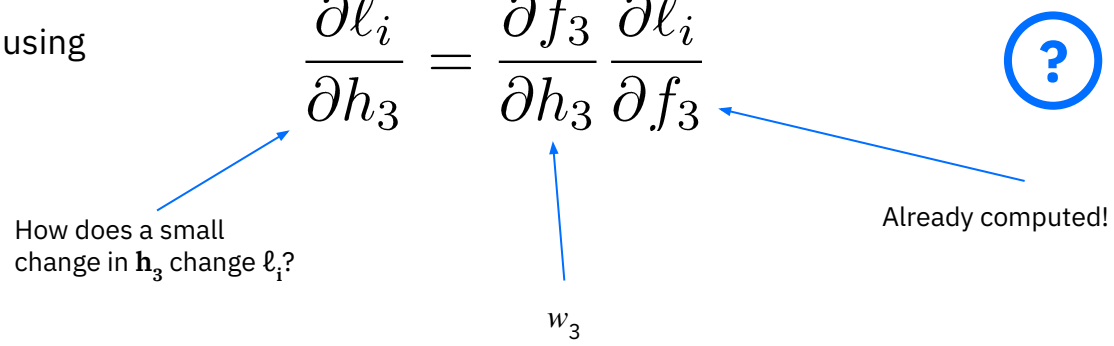
Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

The rest are computed using the *chain rule*.

$$\frac{\partial \ell_i}{\partial h_3} = \frac{\partial f_3}{\partial h_3}\frac{\partial \ell_i}{\partial f_3}$$



Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

The rest are computed using the *chain rule*.

$$
\frac{\partial \ell_i}{\partial f_2} = \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\right)\frac{\partial h_3}{\partial f_2}
$$

?

Already computed!

$-\sin(\mathbf{h}_2)$

Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

$$
\frac{\partial \ell_i}{\partial f_2} = \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\right)\frac{\partial h_3}{\partial f_2}
$$

The rest are computed using the *chain rule*.

Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

The rest are computed using the *chain rule*.

$$
\begin{aligned}
\frac{\partial \ell_i}{\partial f_2} &= \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\right)\frac{\partial h_3}{\partial f_2} \\
\frac{\partial \ell_i}{\partial h_2} &= \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\frac{\partial h_3}{\partial f_2}\right)\frac{\partial f_2}{\partial h_2}
\end{aligned}
$$

Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

The rest are computed using the *chain rule*.

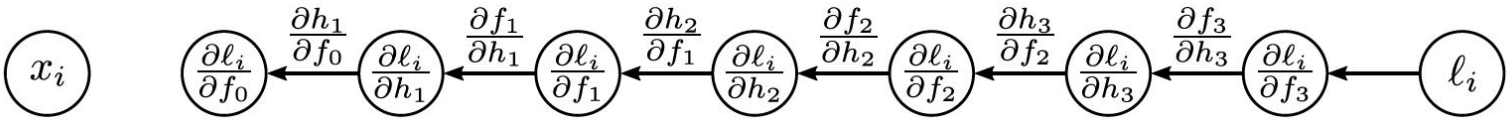
$$
\begin{aligned}
\frac{\partial \ell_i}{\partial f_2} &= \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\right)\frac{\partial h_3}{\partial f_2} \\
\frac{\partial \ell_i}{\partial h_2} &= \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\frac{\partial h_3}{\partial f_2}\right)\frac{\partial f_2}{\partial h_2} \\
\frac{\partial \ell_i}{\partial f_1} &= \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\frac{\partial h_3}{\partial f_2}\frac{\partial f_2}{\partial h_2}\right)\frac{\partial h_2}{\partial f_1} \\
\frac{\partial \ell_i}{\partial h_1} &= \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\frac{\partial h_3}{\partial f_2}\frac{\partial f_2}{\partial h_2}\frac{\partial h_2}{\partial f_1}\right)\frac{\partial f_1}{\partial h_1} \\
\frac{\partial \ell_i}{\partial f_0} &= \left(\frac{\partial \ell_i}{\partial f_3}\frac{\partial f_3}{\partial h_3}\frac{\partial h_3}{\partial f_2}\frac{\partial f_2}{\partial h_2}\frac{\partial h_2}{\partial f_1}\frac{\partial f_1}{\partial h_1}\right)\frac{\partial h_1}{\partial f_0}.
\end{aligned}
$$

Backward pass

1. Compute the derivatives of the loss with respect to these intermediate quantities, in *reverse order*.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

Chain rule all the way down!

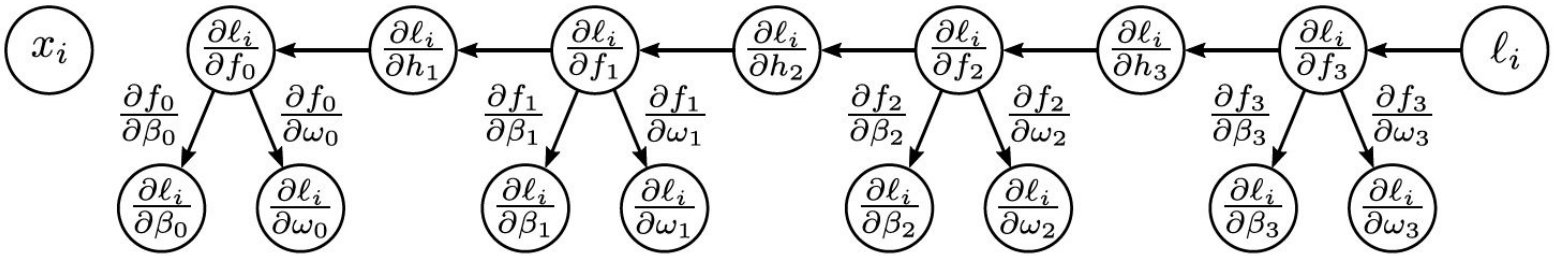


Backward pass

2. Find how the loss changes as a function of the parameters β and ω.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

Chain rule all the way down! Same recipe for weight and bias terms too!



Backward pass

2. Find how the loss changes as a function of the parameters β and ω.

$$
\begin{aligned}
f_0 &= \beta_0 + \omega_0 \cdot x_i \\
h_1 &= \sin[f_0] \\
f_1 &= \beta_1 + \omega_1 \cdot h_1 \\
h_2 &= \exp[f_1]
\end{aligned}
\qquad
\begin{aligned}
f_2 &= \beta_2 + \omega_2 \cdot h_2 \\
h_3 &= \cos[f_2] \\
f_3 &= \beta_3 + \omega_3 \cdot h_3 \\
\ell_i &= (f_3 - y_i)^2.
\end{aligned}
$$

Chain rule all the way down! Same recipe for weight and bias terms too!

$$
\frac{\partial \ell_i}{\partial \omega_k} = \frac{\partial f_k}{\partial \omega_k} \frac{\partial \ell_i}{\partial f_k}
$$

Reading suggestion

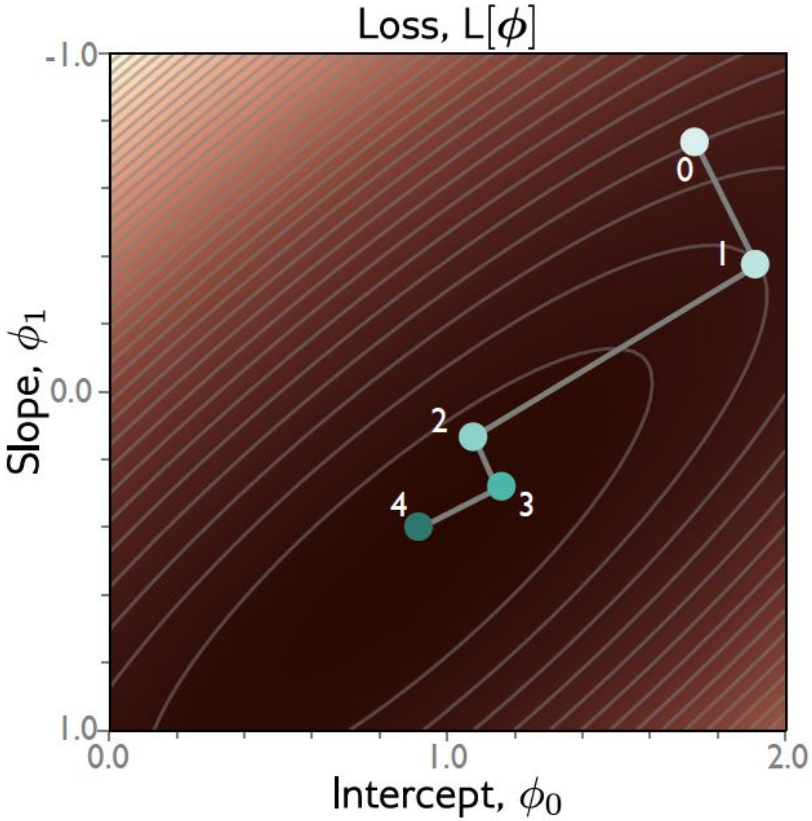
Check 7.4 in your textbook for a discussion on the extension to matrix calculus.

Automatic differentiation

- Modern deep learning frameworks compute derivatives automatically
- You just have to specify the model and the loss
- How?
  - Each component knows how to compute its own derivative
    - ReLU knows how to compute deriv of output w.r.t. input
    - Linear function knows how to compute deriv of output w.r.t. input
    - Linear function knows how to compute deriv of output w.r.t. parameter
  - You specify the order of the components
  - It can compute the chain of derivatives
- Works with branches as long as it’s still an acyclic graph

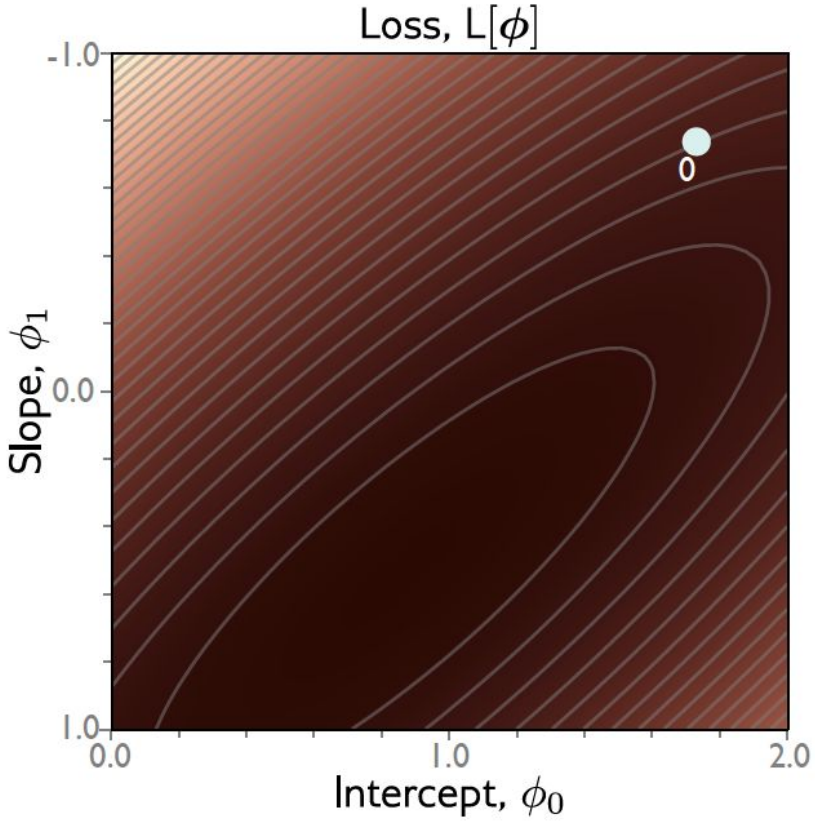
Optimisation

Training a simple model



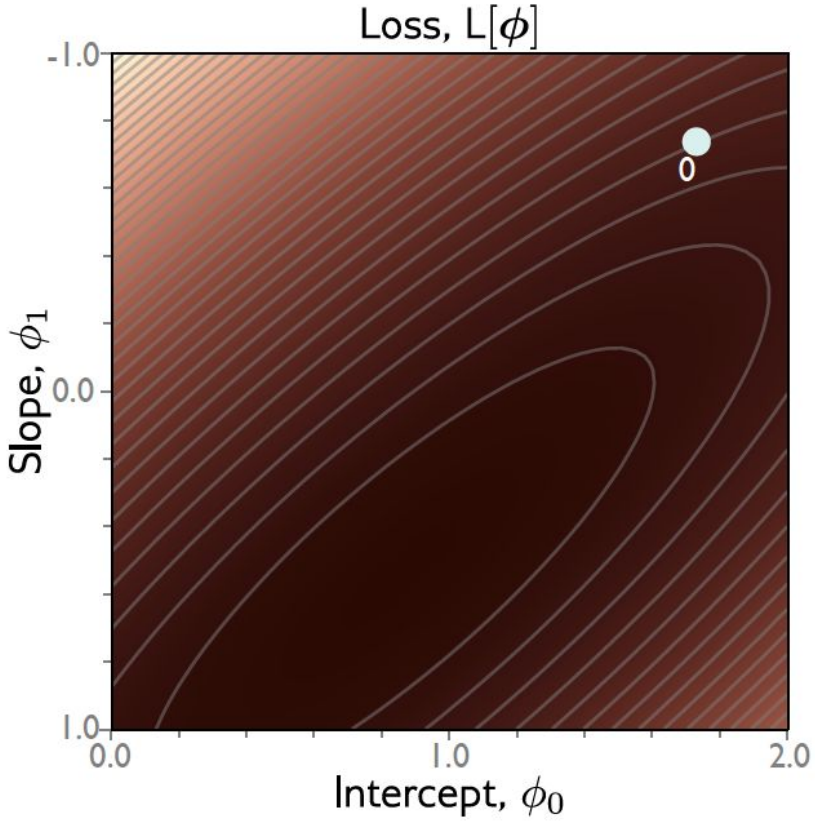
1. Define a loss function
2. Compute the change in parameters required to make the loss smaller
3. Apply the change and get new parameters
4. Repeat from (2)

Gradient descent



$$
\begin{aligned}
L[\boldsymbol{\phi}] &= \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \left(\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i\right)^2 \\
&= \sum_{i=1}^{I} \left(\phi_0 + \phi_1 x_i - y_i\right)^2
\end{aligned}
$$

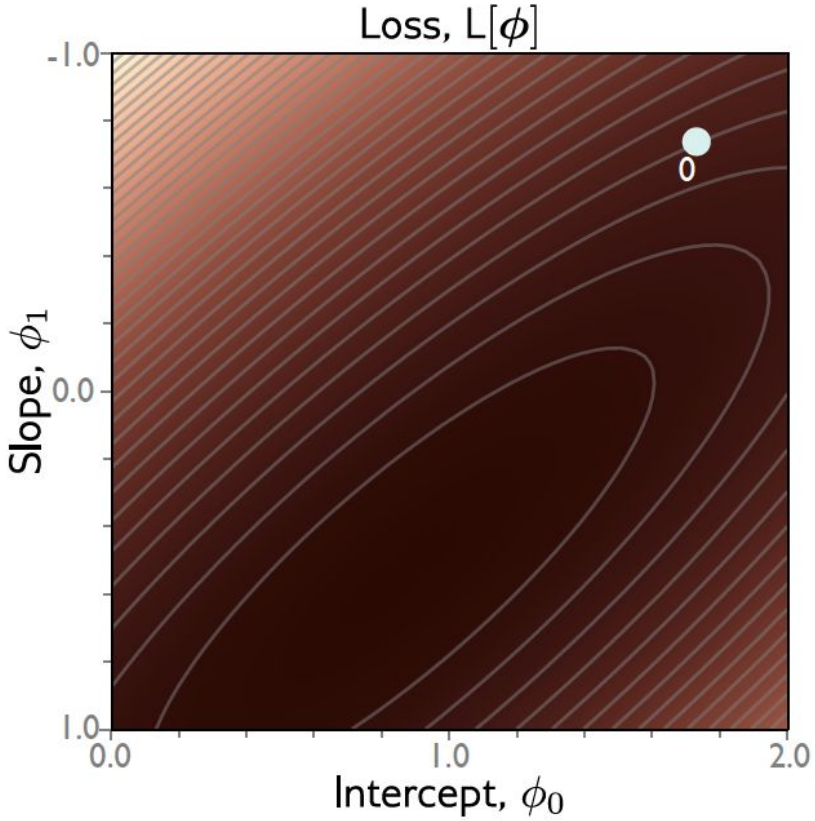
Gradient descent



$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

Gradient descent

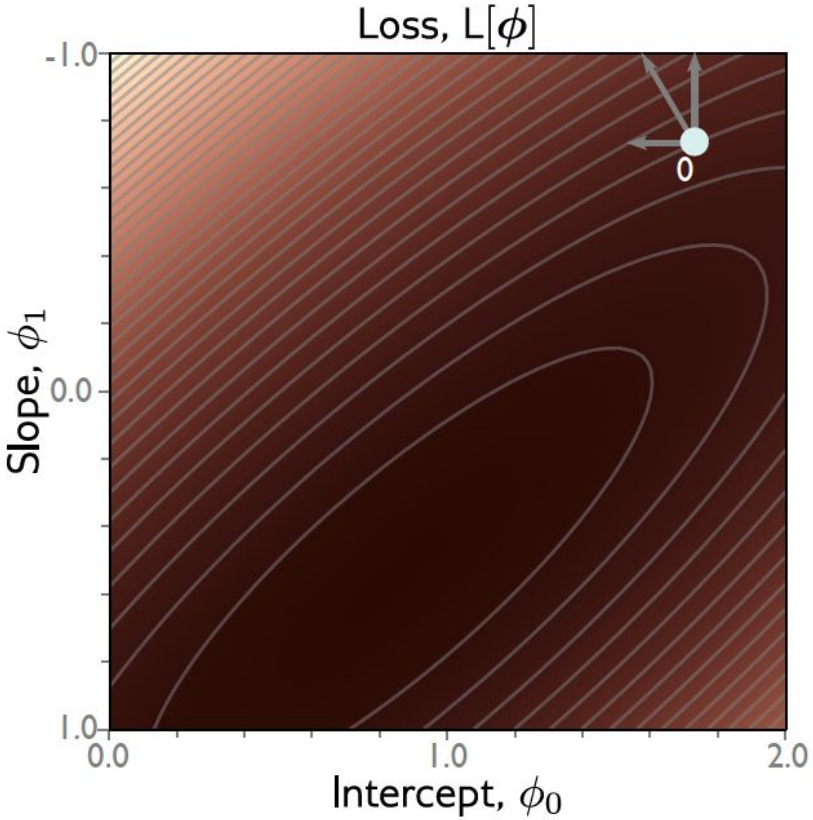


$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

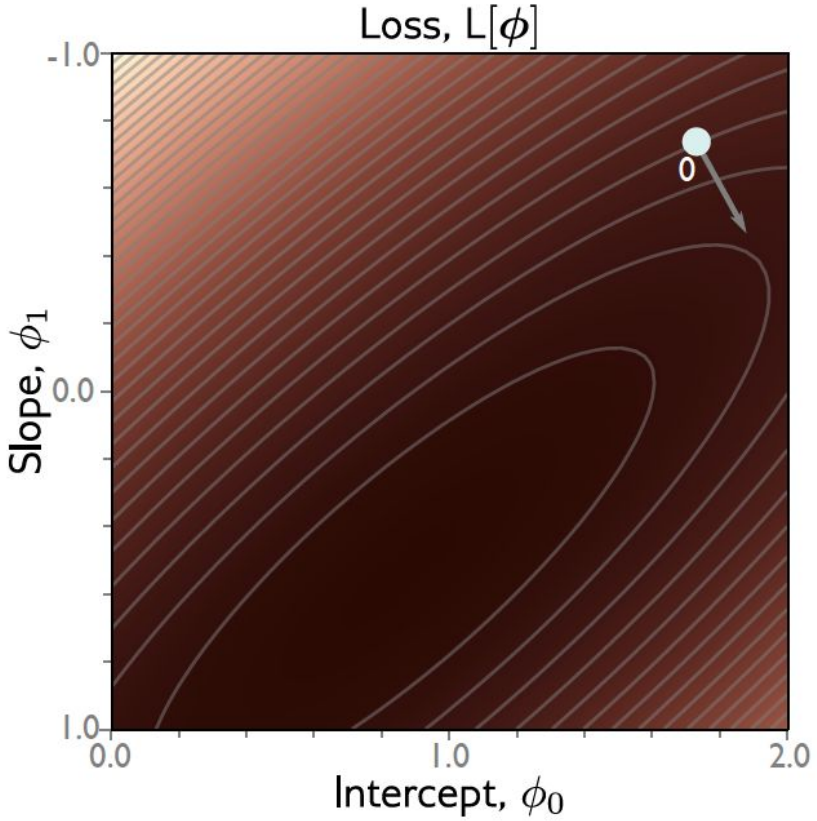
Gradient descent



$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Gradient descent



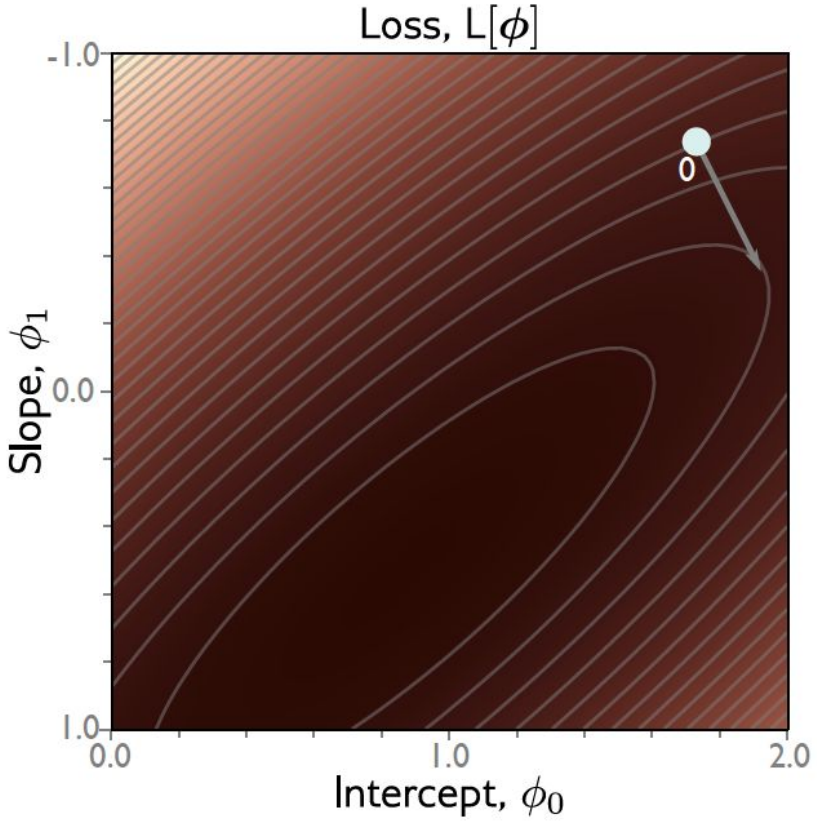
$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

α = step size or learning rate if fixed

Gradient descent



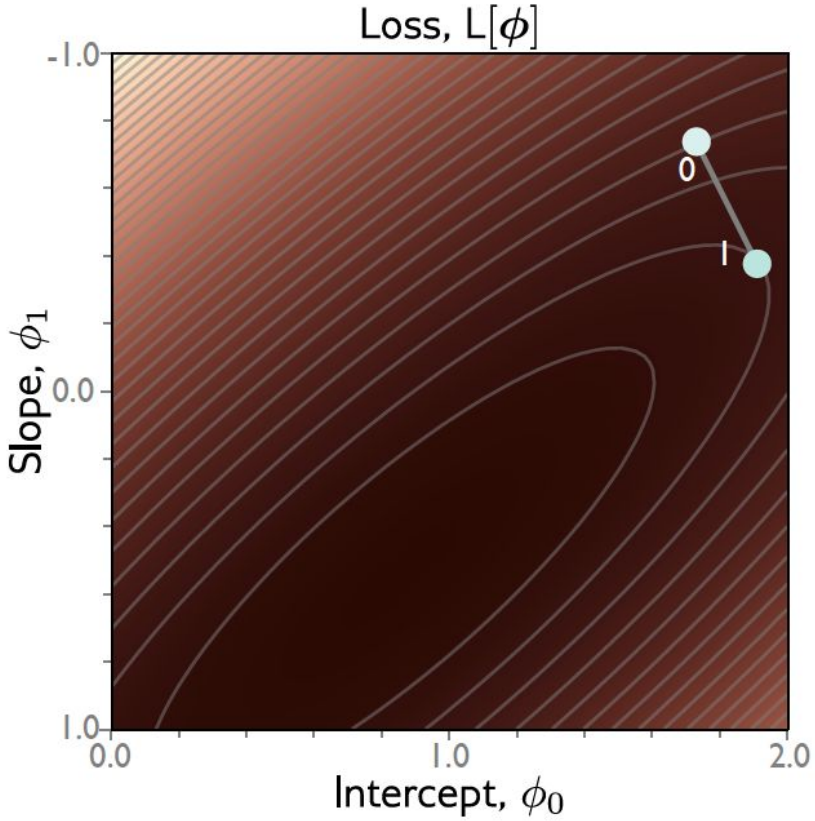
$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

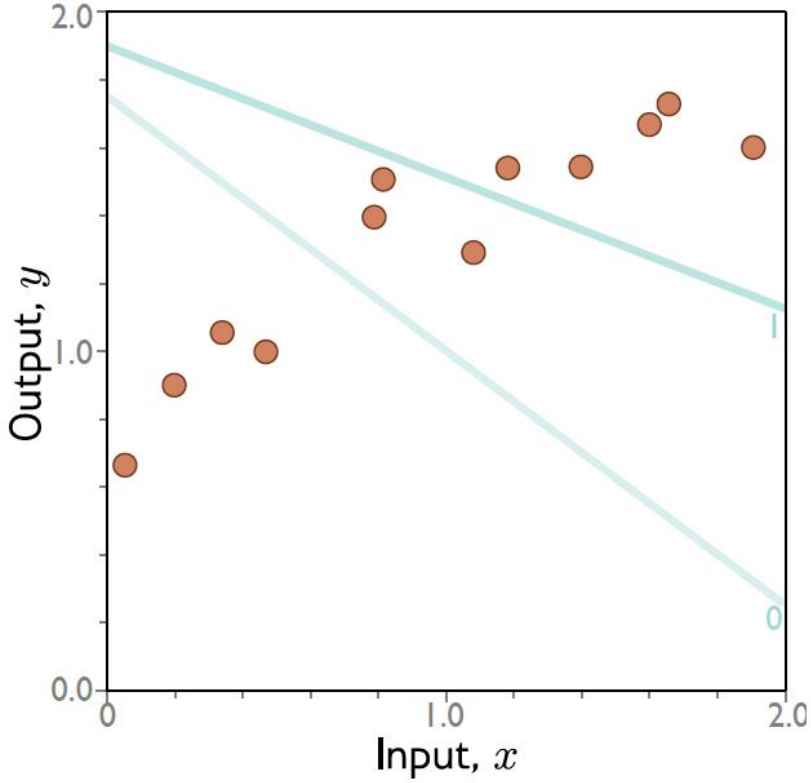
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

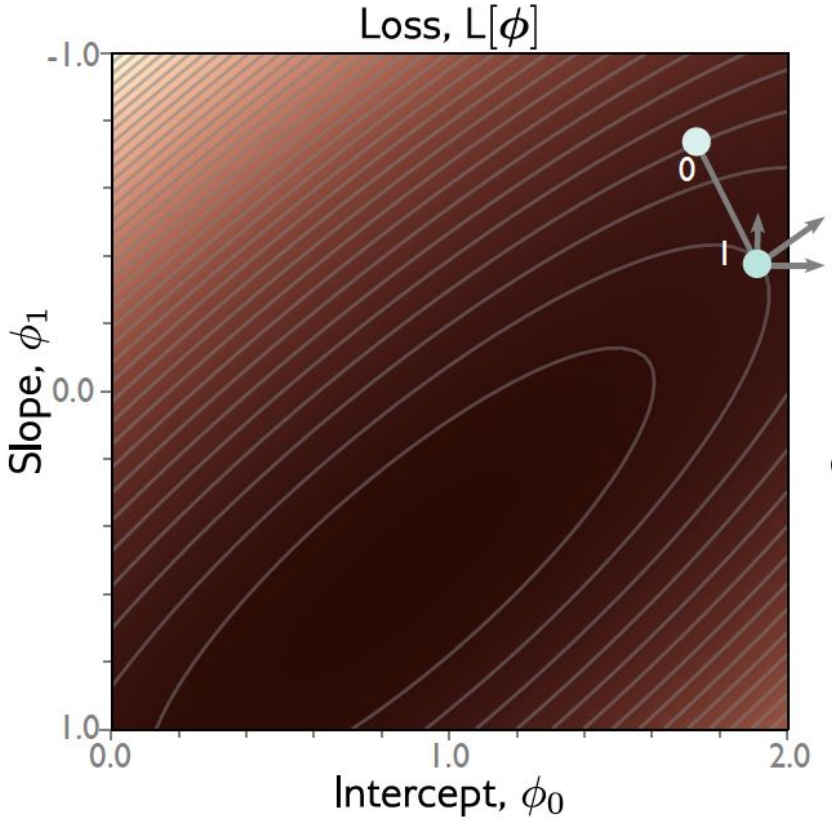
α = step size

Gradient descent





Gradient descent



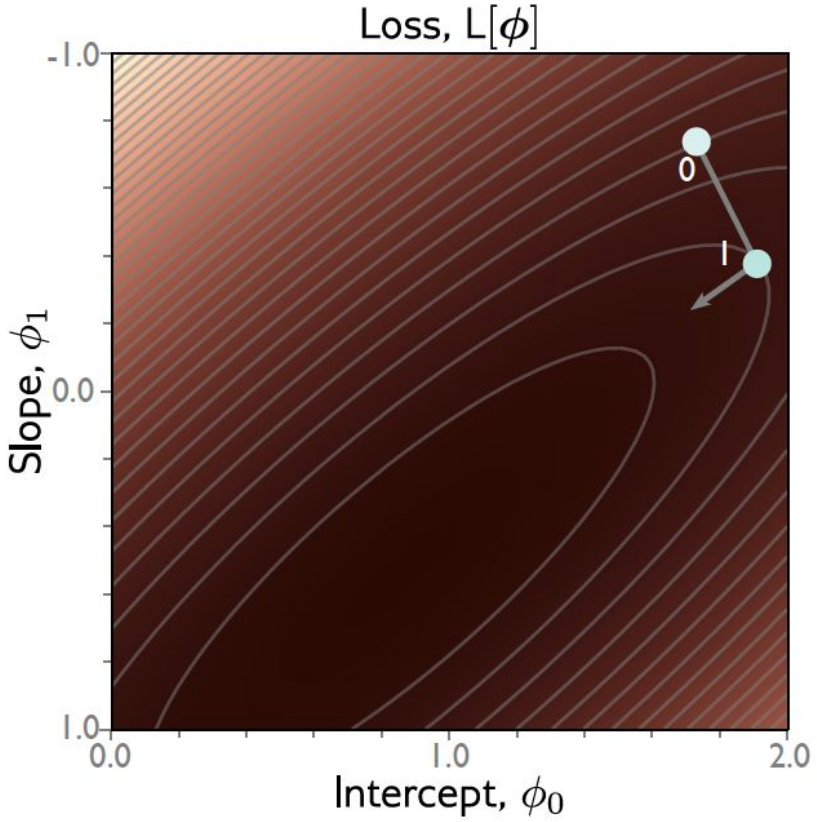
$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

α = step size

Gradient descent



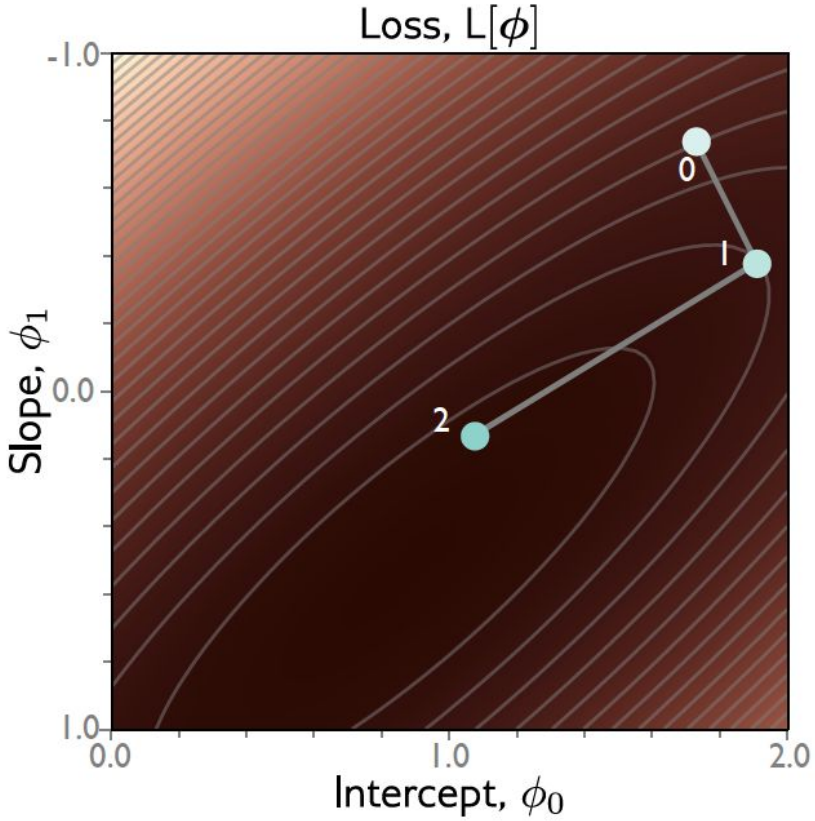
$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

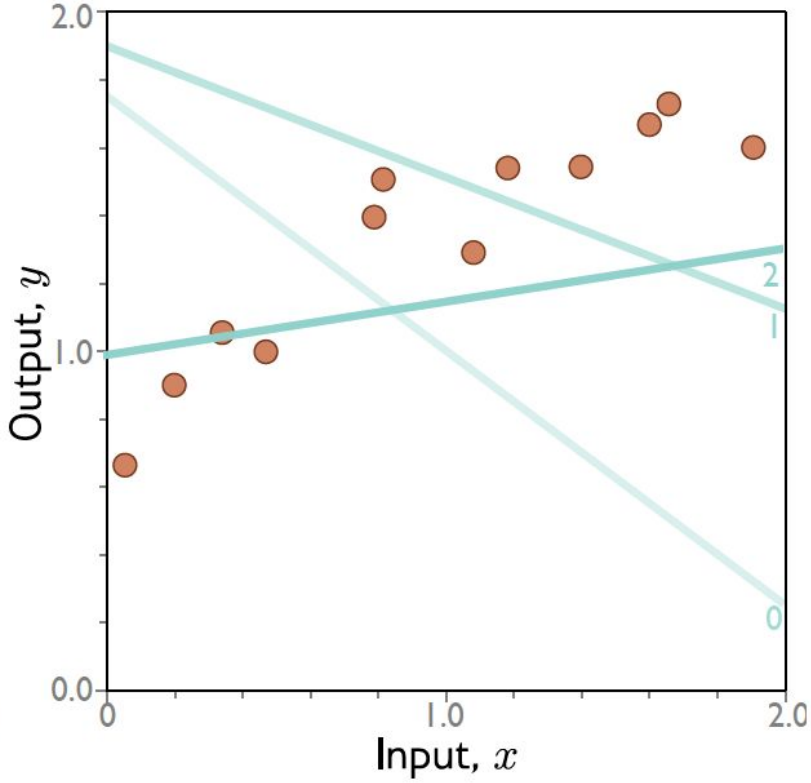
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

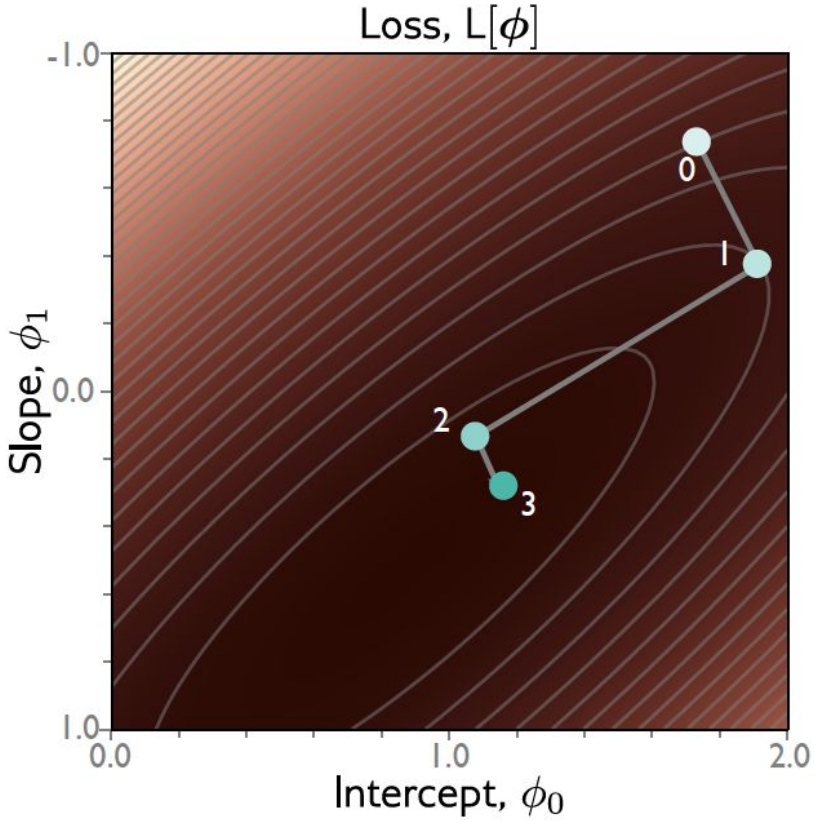
α = step size

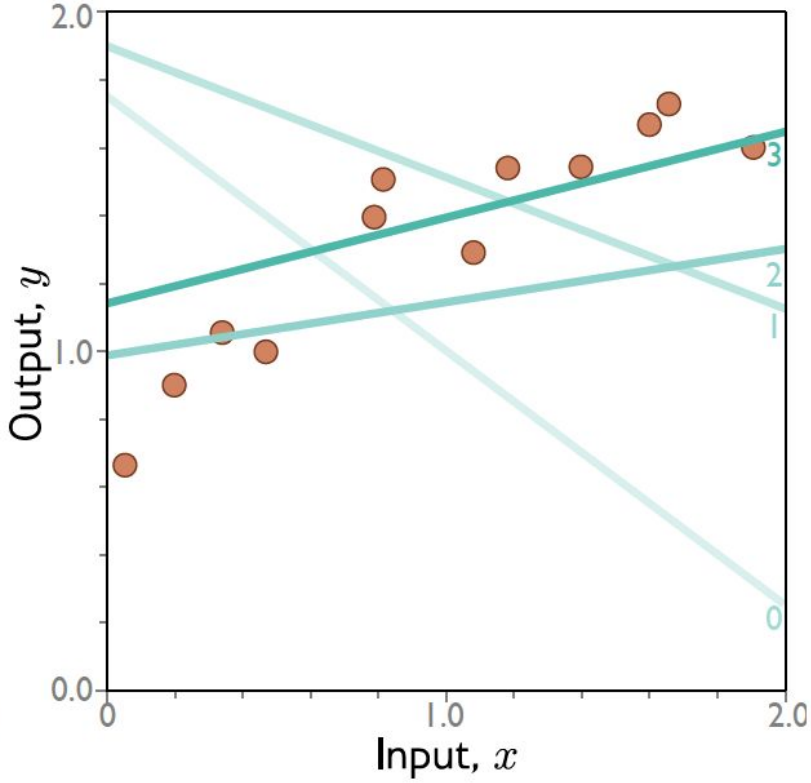
Gradient descent



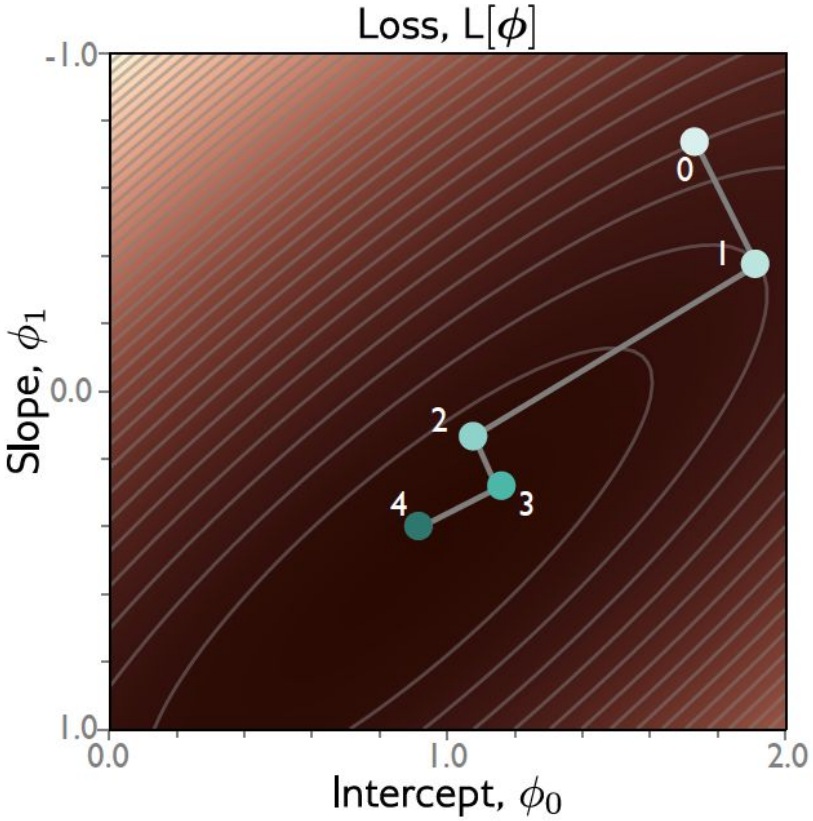


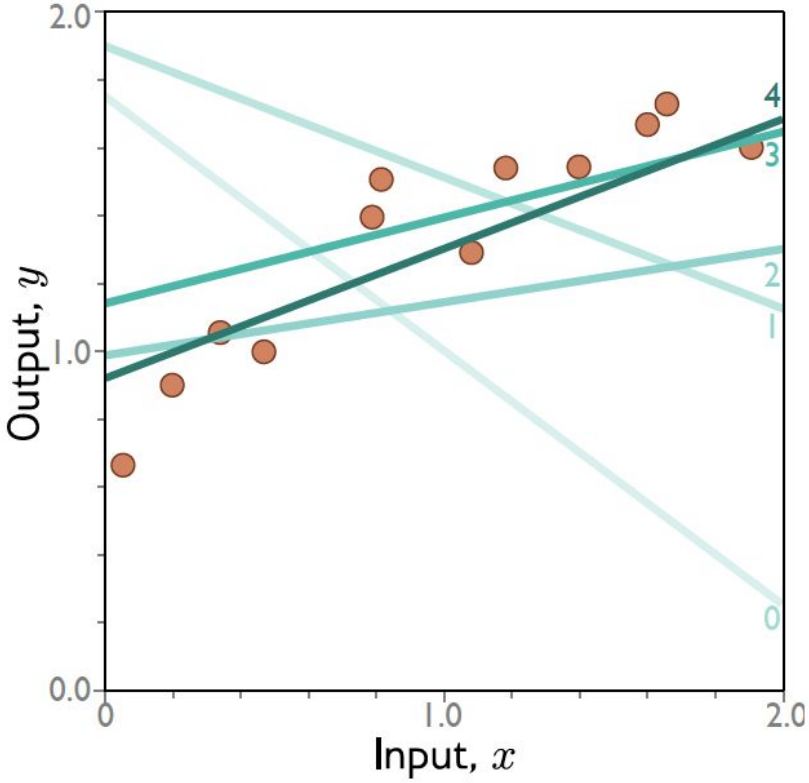
Gradient descent



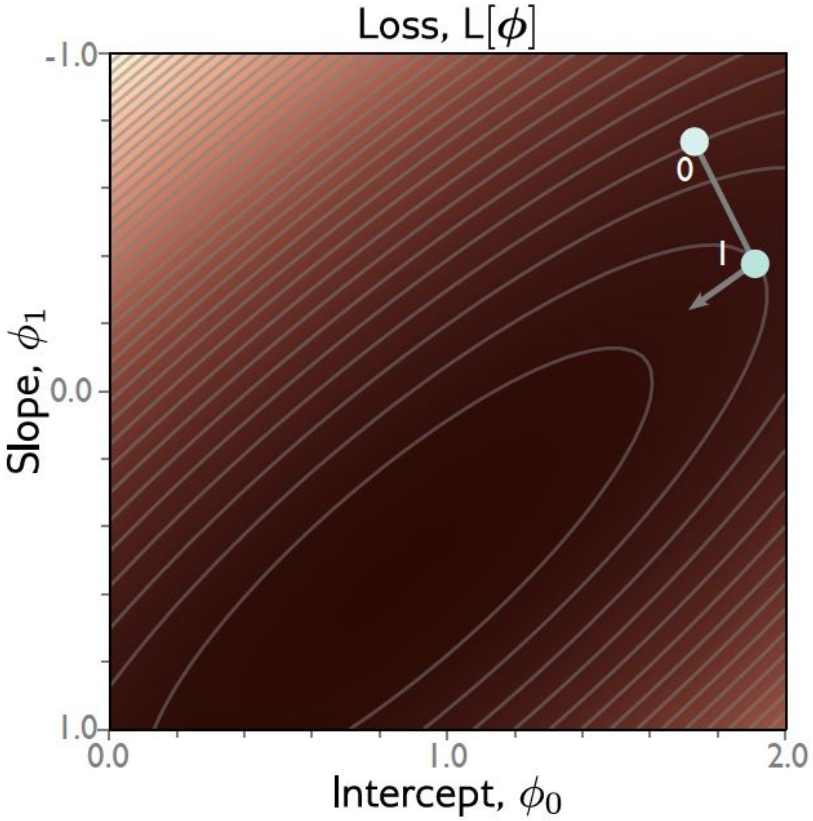


Gradient descent





Line search



$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

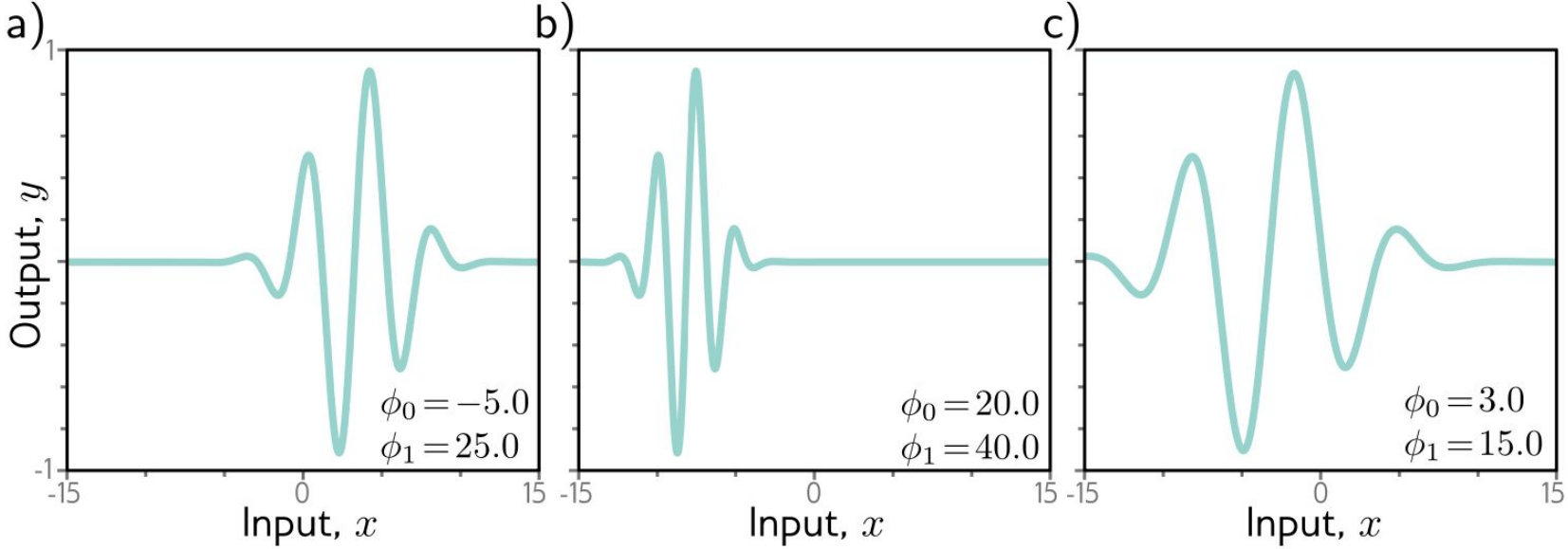
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

α = step size

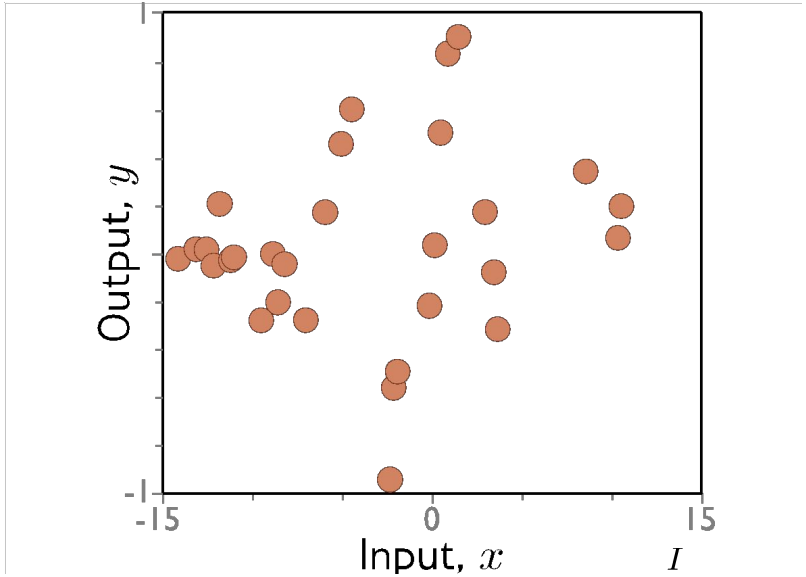
Non-convex case. Gabor model

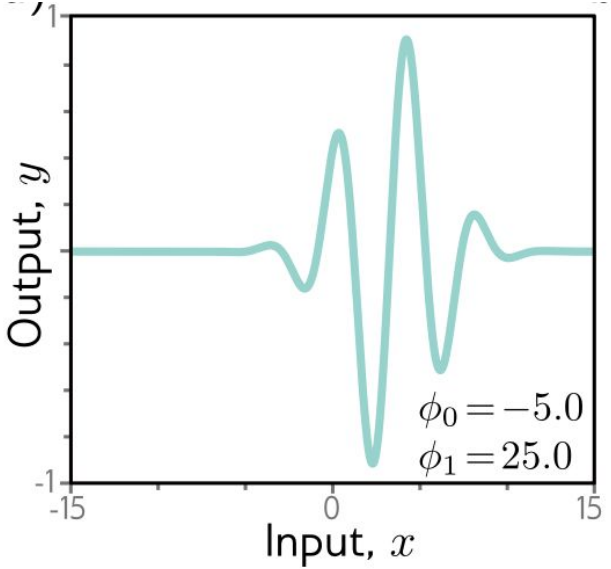
$$\mathrm{f}[x, \boldsymbol{\phi}] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{8.0}\right)$$



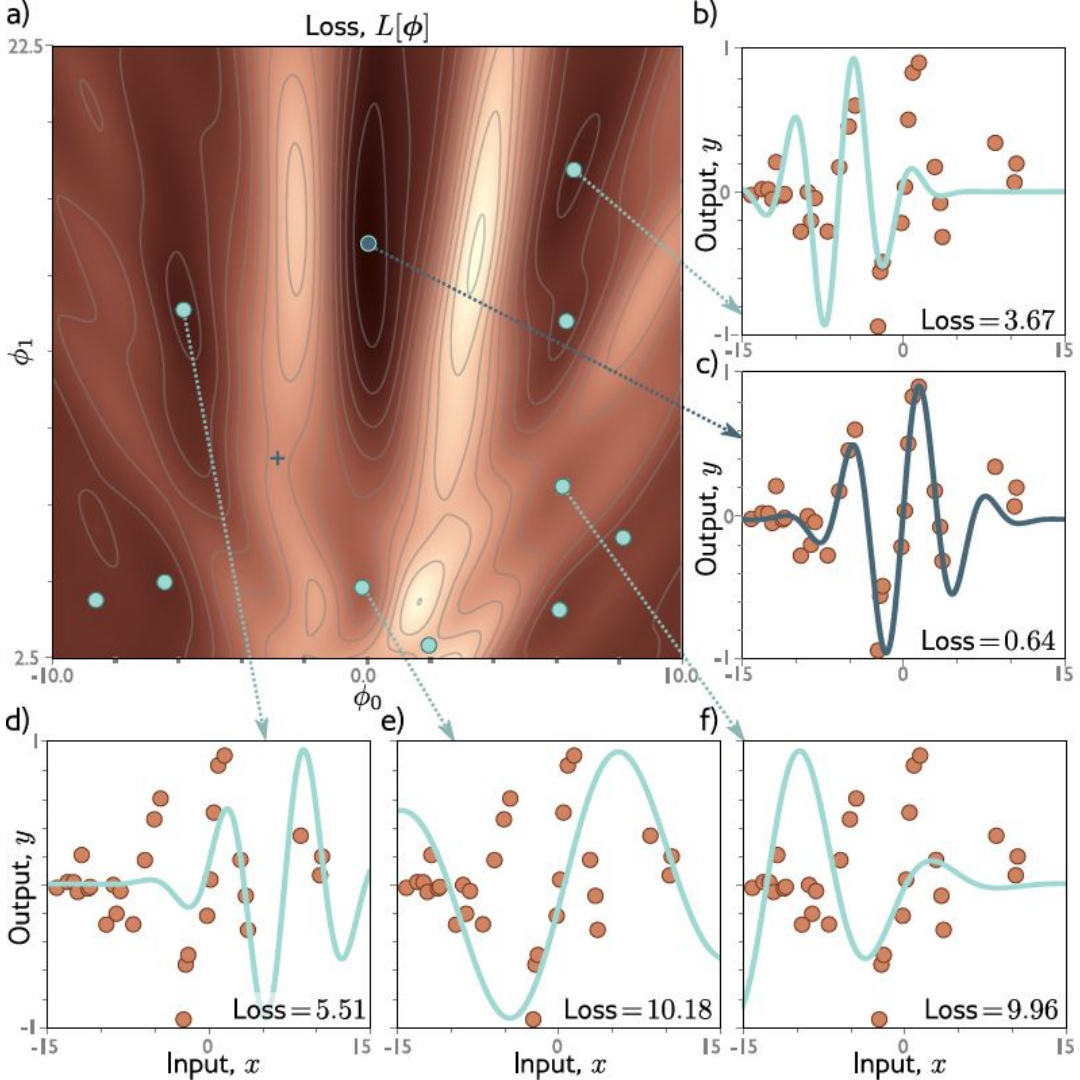
Gabor model

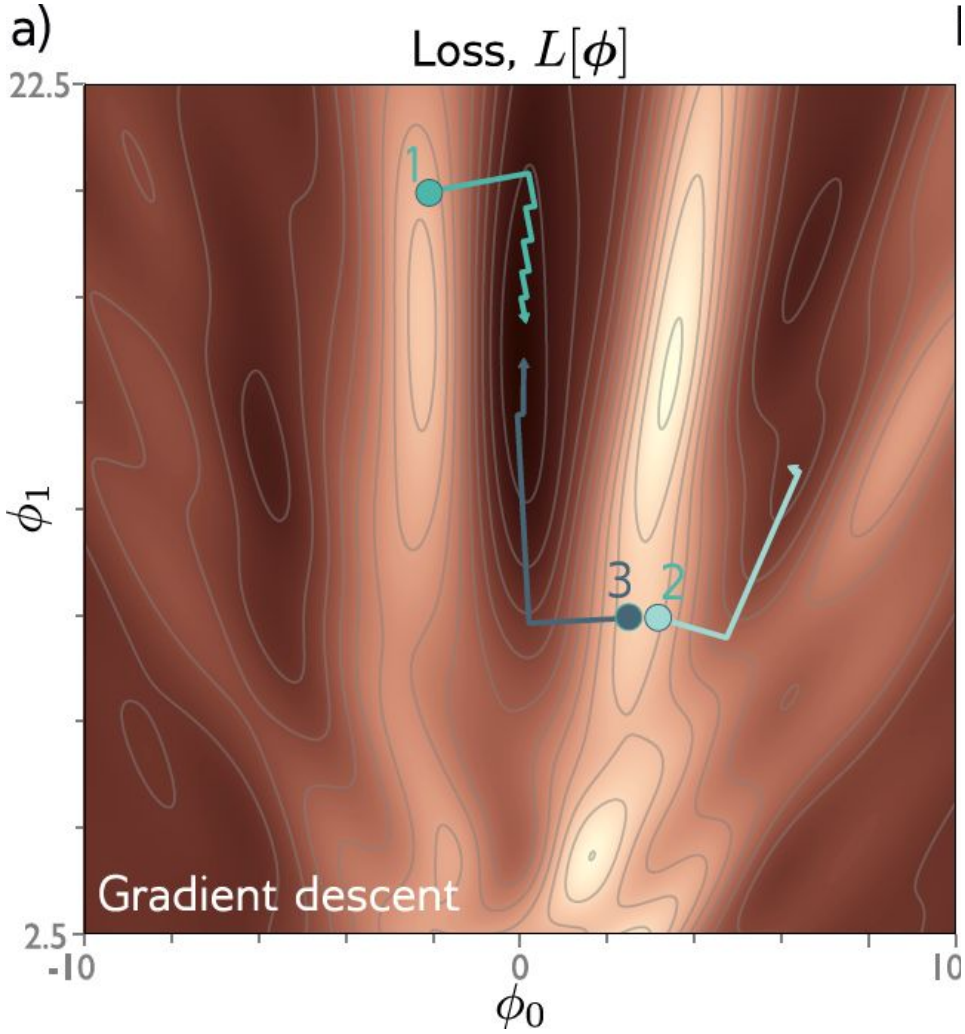
$$\mathrm{f}[x, \boldsymbol{\phi}] = \sin[\phi_0 + 0.06 \cdot \phi_1 x] \cdot \exp\left(-\frac{(\phi_0 + 0.06 \cdot \phi_1 x)^2}{8.0}\right)$$



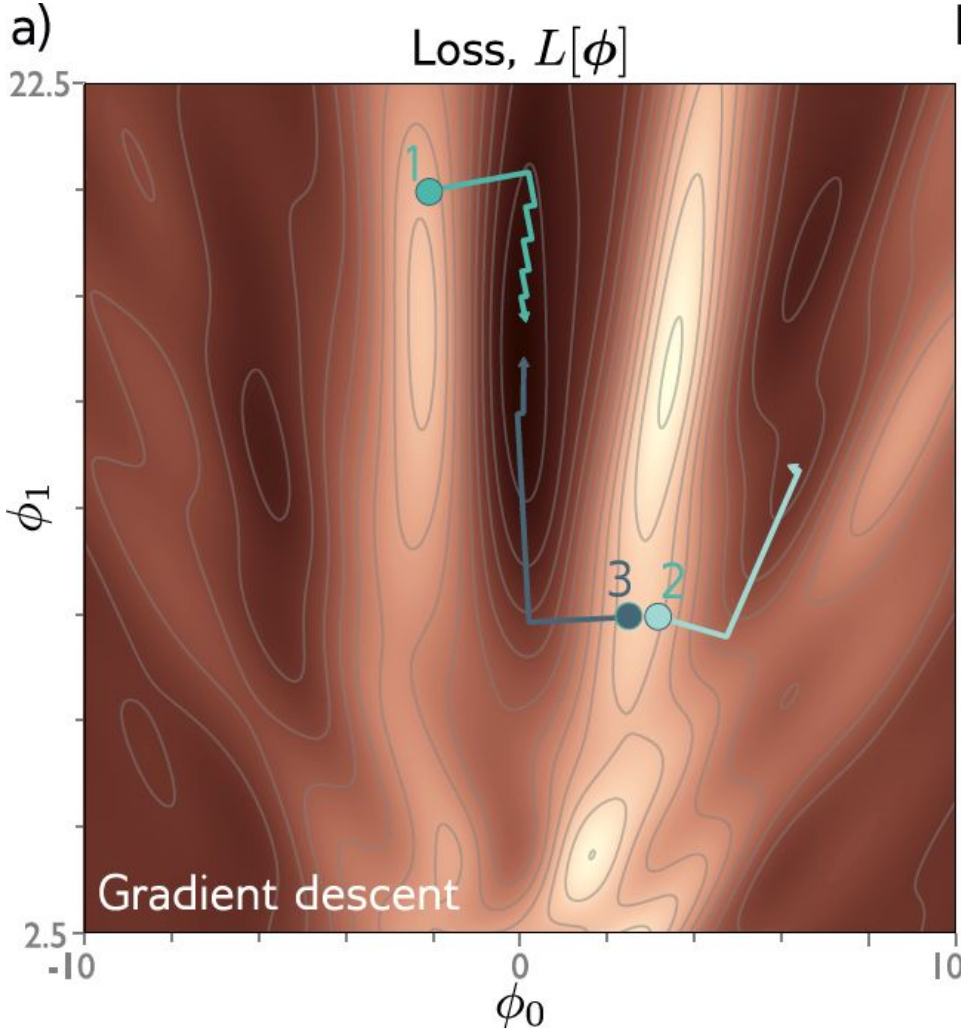


$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \left(\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i\right)^2$$


a)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
-10.0
0.0
10.0
ϕ_0
b)
Output, y
Loss = 3.67
c)
Output, y
Loss = 0.64
d)
Output, y
Loss = 5.51
Input, x
e)
Loss = 10.18
Input, x
f)
Loss = 9.96
Input, x

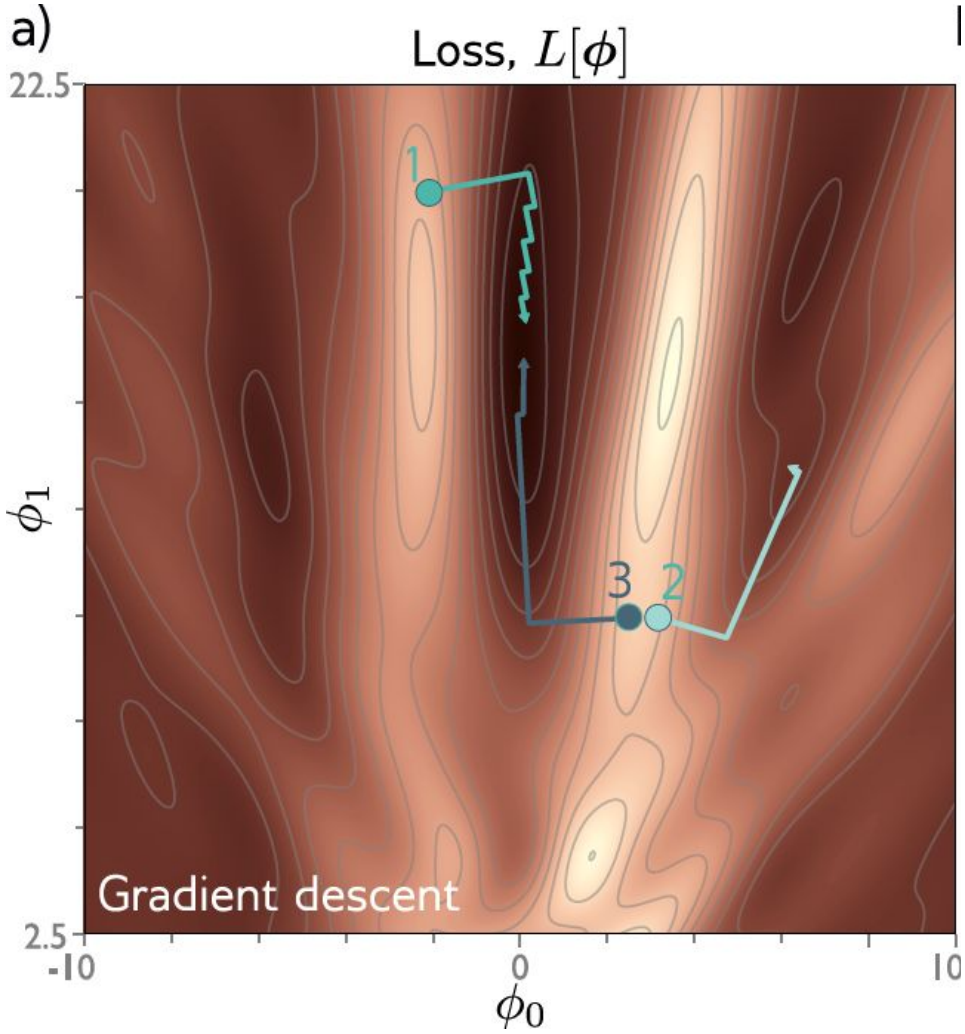


- Gradient descent gets to the global minimum if we start in the right “valley”
- Otherwise, descent to a local minimum
- Or get stuck near a saddle point



Solution: add noise!

- Stochastic gradient descent
- Compute gradient based on only a subset of points – a mini-batch
- Work through dataset sampling without replacement
- One pass though the data is called an epoch



Stochastic gradient descent

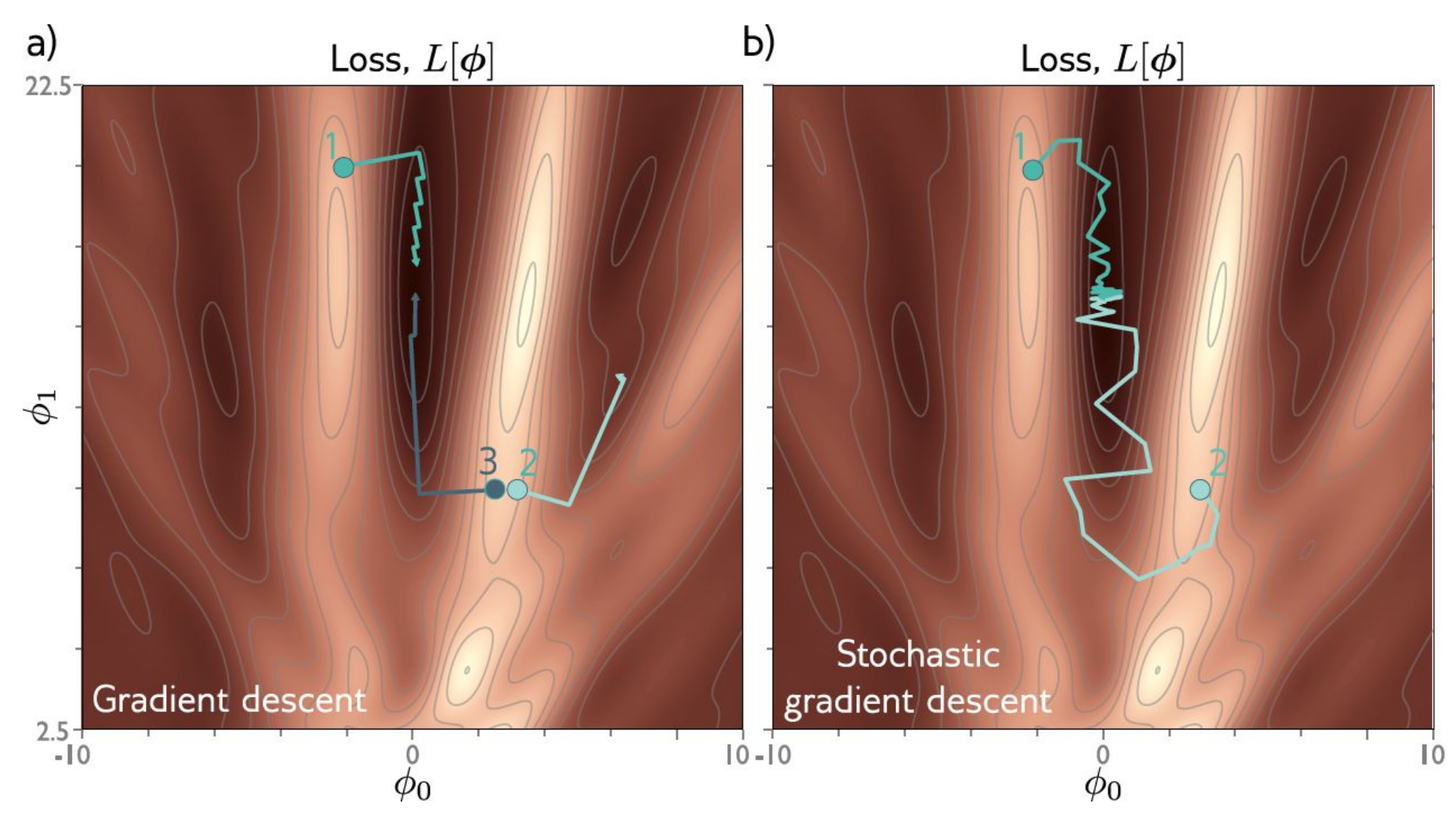
Before (full batch descent)

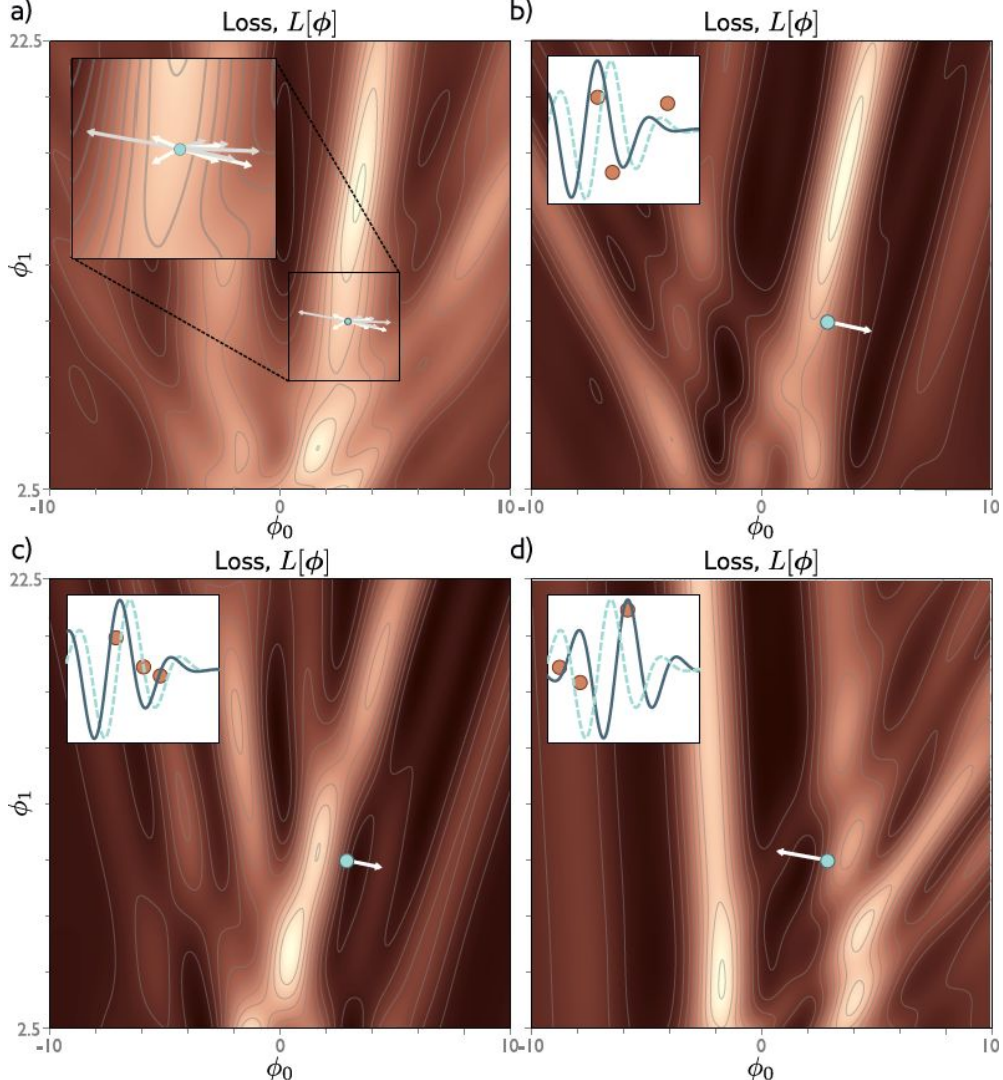
$$\phi_{t+1} \longleftarrow \phi_t - \alpha \sum_{i=1}^{I} \frac{\partial \ell_i[\phi_t]}{\partial \phi},$$

After (SGD)

$$\phi_{t+1} \longleftarrow \phi_t - \alpha \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\phi_t]}{\partial \phi},$$

Fixed learning rate α


a)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
1
2
3
Gradient descent
-10
0
10
ϕ_0
b)
Loss, $L[\phi]$
1
2
Stochastic
gradient descent
-10
0
10
ϕ_0

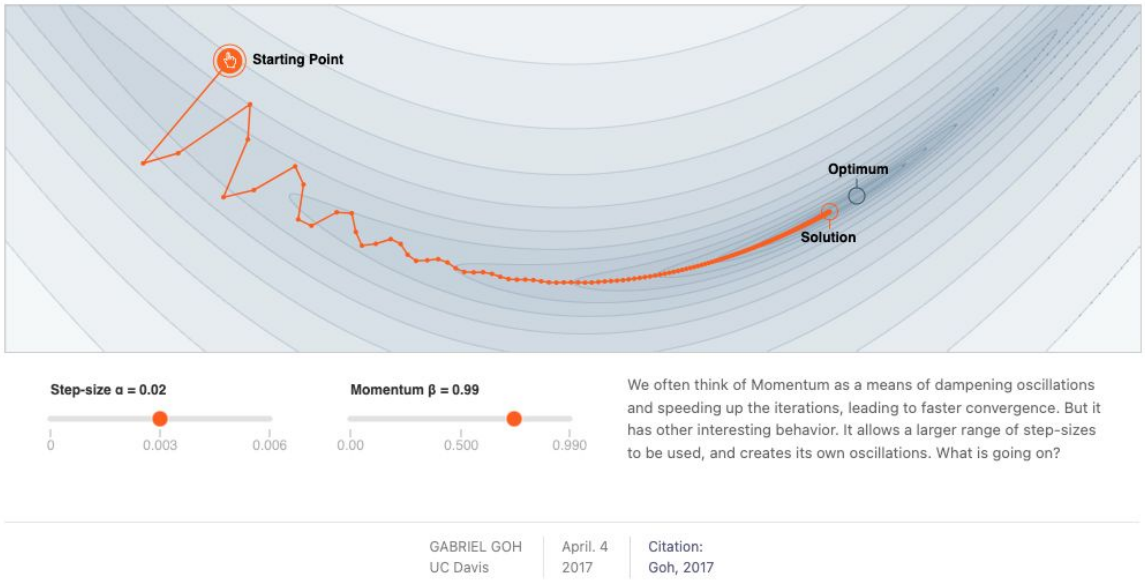

a)
Loss, $L[\phi]$
b)
Loss, $L[\phi]$
c)
Loss, $L[\phi]$
d)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
-10
0
10
ϕ_0

Properties of SGD

- Can escape from local minima
- Adds noise, but still sensible updates as based on part of data
- Uses all data equally
- Less computationally expensive
- Seems to find better solutions

- Doesn’t converge in traditional sense
- Learning rate schedule – decrease learning rate over time

Momentum



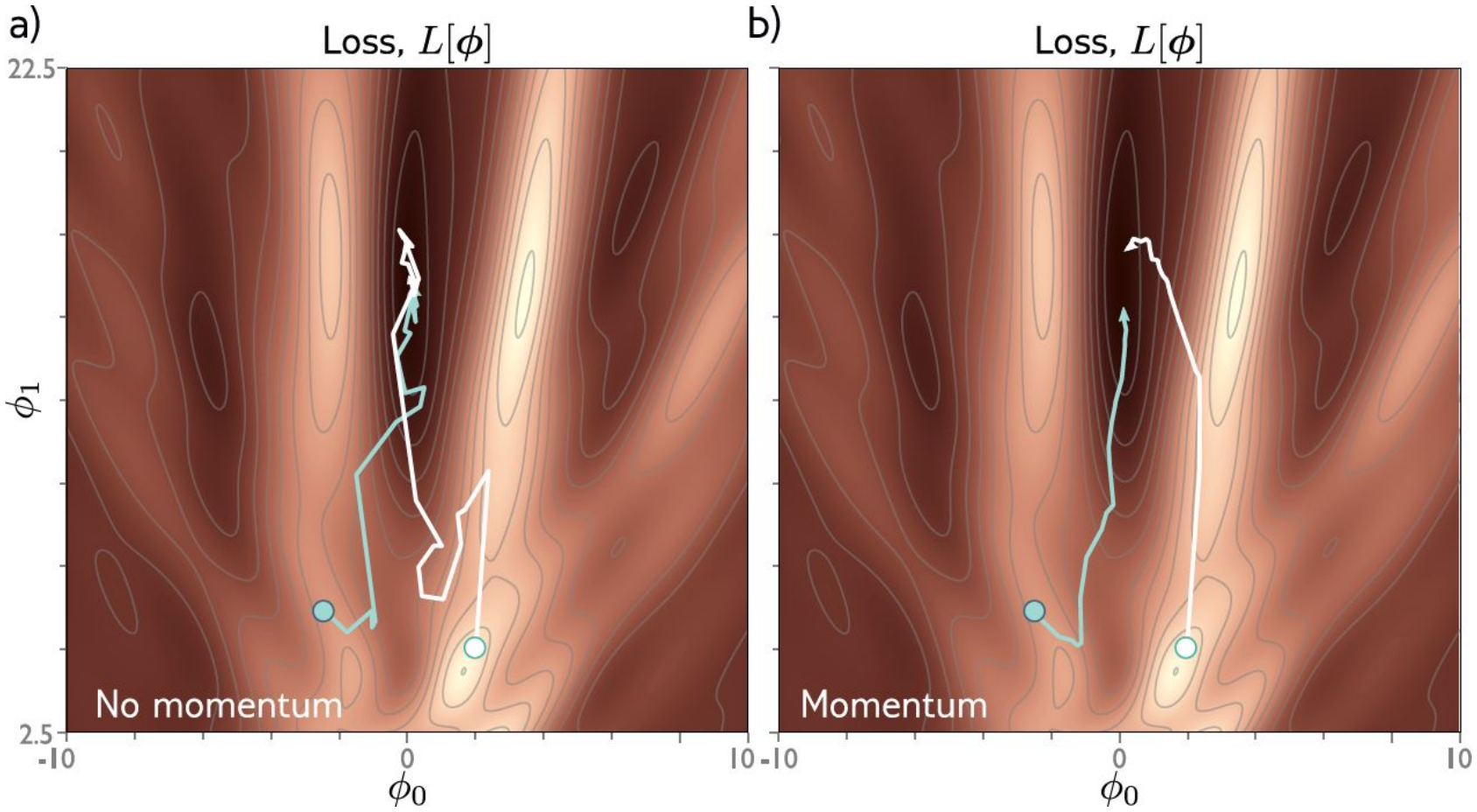
Goh, *Why momentum really works*

Momentum

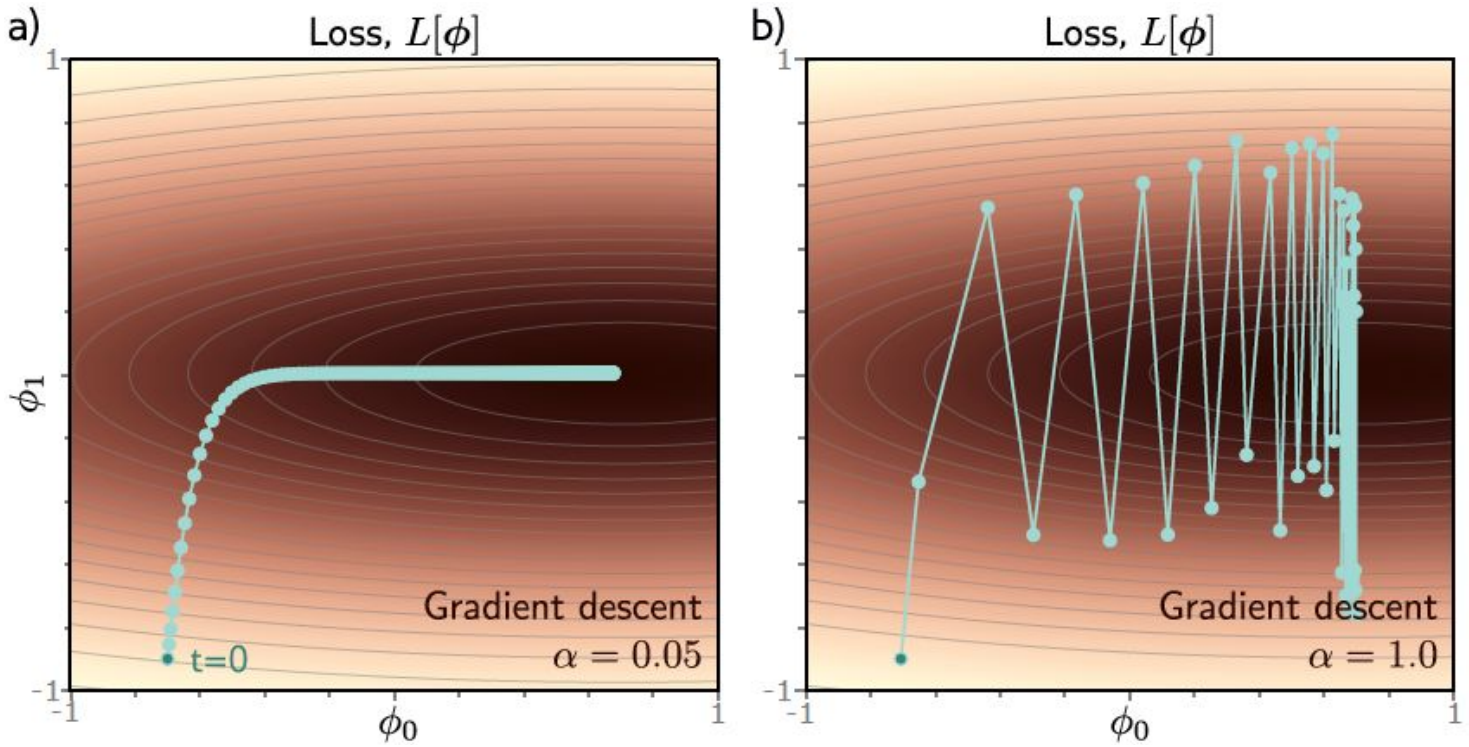
Weighted sum of this gradient and previous gradient

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta) \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \mathbf{m}_{t+1}$$


a)
Loss, $L[\phi]$
22.5
2.5
ϕ_1
No momentum
-10
0
10
ϕ_0
b)
Loss, $L[\phi]$
Momentum
-10
0
10
ϕ_0

Adaptive moment estimation. Adam



Normalized gradients

Measure mean and pointwise squared gradient

$$\mathbf{m}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\mathbf{v}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}^2$$

Normalize:

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}$$

Normalized gradients

Measure mean and pointwise squared gradient

$$\mathbf{m}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}} \qquad \mathbf{m}_{t+1} = \begin{bmatrix} 3.0 \\ -2.0 \\ 5.0 \end{bmatrix}$$

$$\mathbf{v}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}^2 \qquad \mathbf{v}_{t+1} = \begin{bmatrix} 9.0 \\ 4.0 \\ 25.0 \end{bmatrix}$$

Normalize:

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon} \qquad \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon} = \begin{bmatrix} 1.0 \\ -1.0 \\ 1.0 \end{bmatrix}$$

Normalized gradients

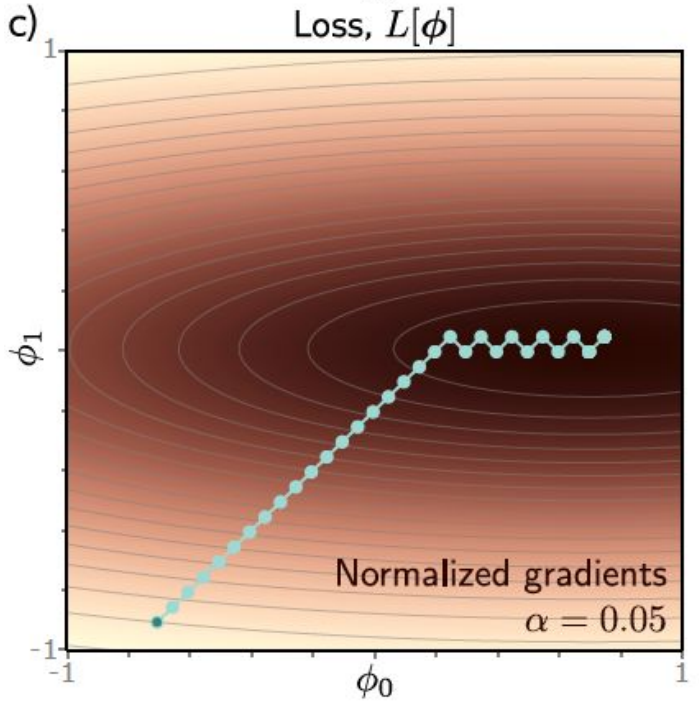
Measure mean and pointwise squared gradient

$$\mathbf{m}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\mathbf{v}_{t+1} \leftarrow \frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}^2$$

Normalize:

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}$$



Normalized gradients

Compute mean and pointwise squared gradients with momentum

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta)\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\mathbf{v}_{t+1} \leftarrow \gamma \cdot \mathbf{v}_t + (1-\gamma)\left(\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}\right)^2$$

Moderate near start of the sequence

to prevent small estimates

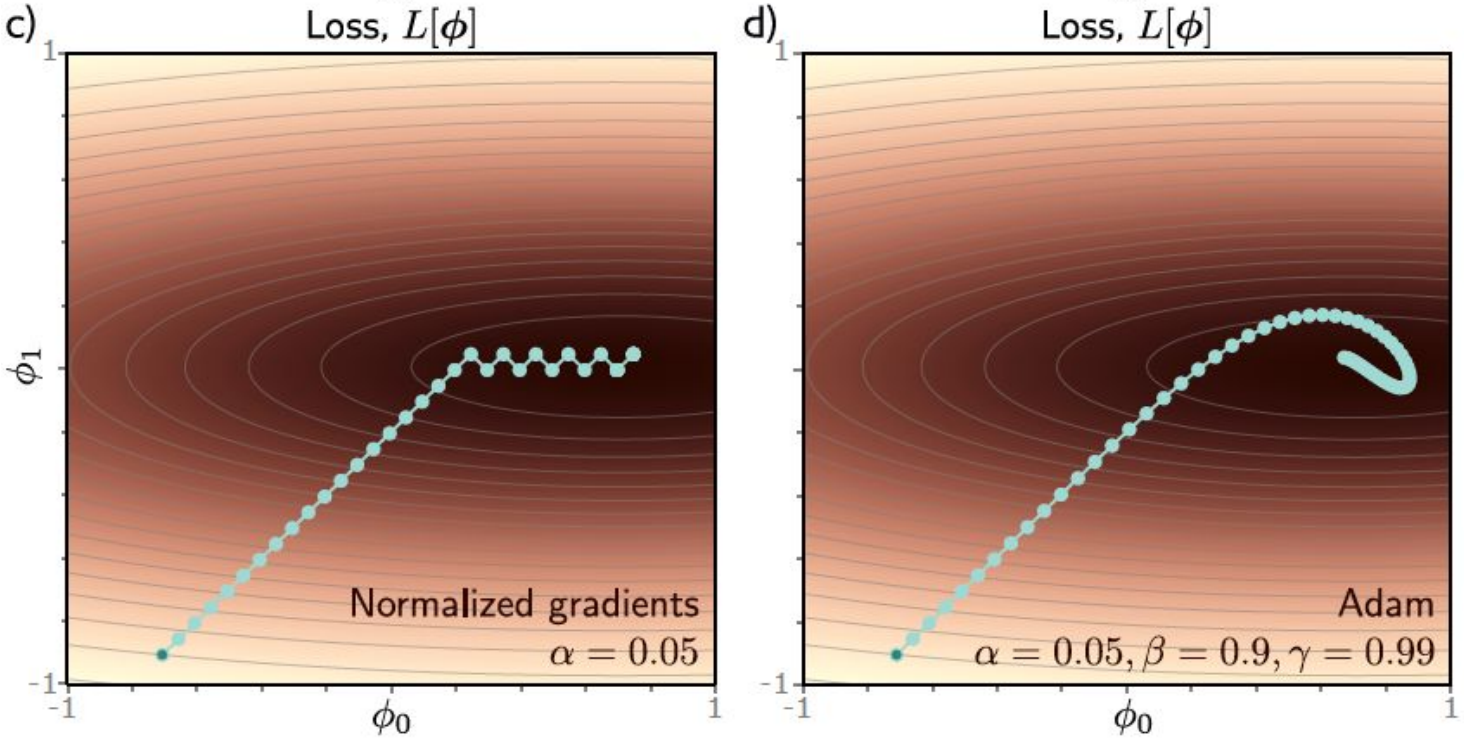
$$\tilde{\mathbf{m}}_{t+1} \leftarrow \frac{\mathbf{m}_{t+1}}{1-\beta^{t+1}}$$

$$\tilde{\mathbf{v}}_{t+1} \leftarrow \frac{\mathbf{v}_{t+1}}{1-\gamma^{t+1}}$$

Update the parameters

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1}} + \epsilon}$$

Adaptive moment estimation. Adam



RMSprop (precursor to Adam)

Compute ~~mean and~~ pointwise squared gradients with momentum

$$\mathbf{m}_{t+1} \leftarrow \beta \cdot \mathbf{m}_t + (1-\beta)\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

$$\mathbf{v}_{t+1} \leftarrow \gamma \cdot \mathbf{v}_t + (1-\gamma)\left(\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}\right)^2$$

Update the parameters

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1}} + \epsilon}\frac{\partial L[\boldsymbol{\phi}_t]}{\partial \boldsymbol{\phi}}$$

How to reach me:

florin.gogianu@gmail.com

Please send unstructured feedback, since this is a new version of the lecture!

Post-lecture reading suggestions

- Maximum Likelihood Estimation:
  - Prince, UDL book, ch. 5 to 5.5
  - Deisenroth, Mathematics for deep learning, ch. 8, 9
  - Bishop, Pattern recognition and machine learning, ch. 1.2.5, 1.2.6, 3.1, 3.3 and 4.3
- Gradient checkpointing:
  - I actually believe the first google result is a good resource :), especially if you check the papers linked at the bottom of the post.
- Ensembling in the weight space, some good starting points:
  - Izmailov, Averaging weights leads to wider optima and better generalization
  - Wortsman, Learning neural network subspaces