Universität Konstanz

02 | Supervised Learning

Max Pellert (https://mpellert.at)

Deep Learning for the Social Sciences

Github Classroom

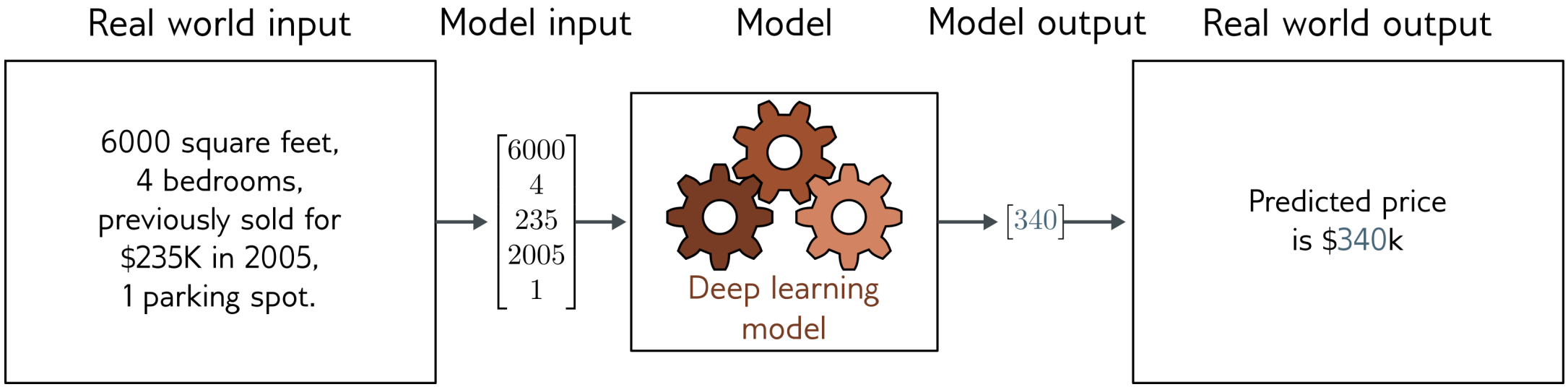
Universität Konstanz

https://github.com/DLSS-24

From now on, please find all class materials in this repo:
https://github.com/DLSS-24/DLSS-24

Supervised learning

Regression



Univariate regression problem (one output, real value)

Supervised learning overview

Supervised learning model = mapping from one or more inputs to one or more outputs

Computing the inputs from the outputs = **inference**

Example:

Input is age and mileage of secondhand Toyota Prius

Output is estimated price of car

Supervised learning overview

Model is a mathematical equation, but better and more generally: model is a *family of equations*

Model includes **parameters**

Parameters affect outcome of equations

Training a model = finding parameters that predict outputs "well" from inputs for a training dataset of input/output pairs

Notation

Check Appendix A of “Understanding Deep Learning”

Input $\mathbf{x}$

Output $\mathbf{y}$

Variables are always indicated with Roman letters

Normal = scalar

Bold = vector

CAPITAL BOLD = matrix

Notation

Model $\mathbf{y} = \mathbf{f}[\mathbf{x}]$

Functions are always indicated with square brackets

Normal = returns scalar

Bold = returns vector

CAPITAL BOLD = returns matrix

Notation examples

Input $\mathbf{x} = \begin{bmatrix} \text{age} \\ \text{mileage} \end{bmatrix}$ **Structured** or **tabular** data

Output $y = [\text{price}]$

Model $y = \mathbf{f}[\mathbf{x}]$

Parameters $\boldsymbol{\phi}$ Parameters are always Greek letters

Model $\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$

Loss function

We use a 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 the model is at relating input to output for the examples:

$$L[\boldsymbol{\phi}, \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}], \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{I}]$$

Or short:

$$L[\boldsymbol{\phi}]$$

Training

Loss function: $L[\phi]$ returns a scalar that is smaller when model maps inputs to outputs better

During training, we try to find the parameters that minimize the loss:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}}[L[\phi]]$$

Testing

To test the model, we evaluate it on a separate test dataset of input/output pairs

Crucially, it must not have seen that data during training (suspiciously high, almost perfect performance on the test set is an indicator that there may have been a spillover)

Testing allows us to see how well it generalizes to “new data”

Testing

Still, the test data is usually from the same domain and collected in the same way as the training data, so external validity can be low although test set performance is high

Always critically assess and try to assess performance "in the wild" to establish the model's limits

This is clearly where your ways to think from the social sciences can come in very handy!

Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W., & Müller, K.-R. (2019). Unmasking Clever Hans predictors and assessing what machines really learn. Nature Communications, 10(1), 1096. https://doi.org/10.1038/s41467-019-08987-4

Example: 1D Linear regression model

Model

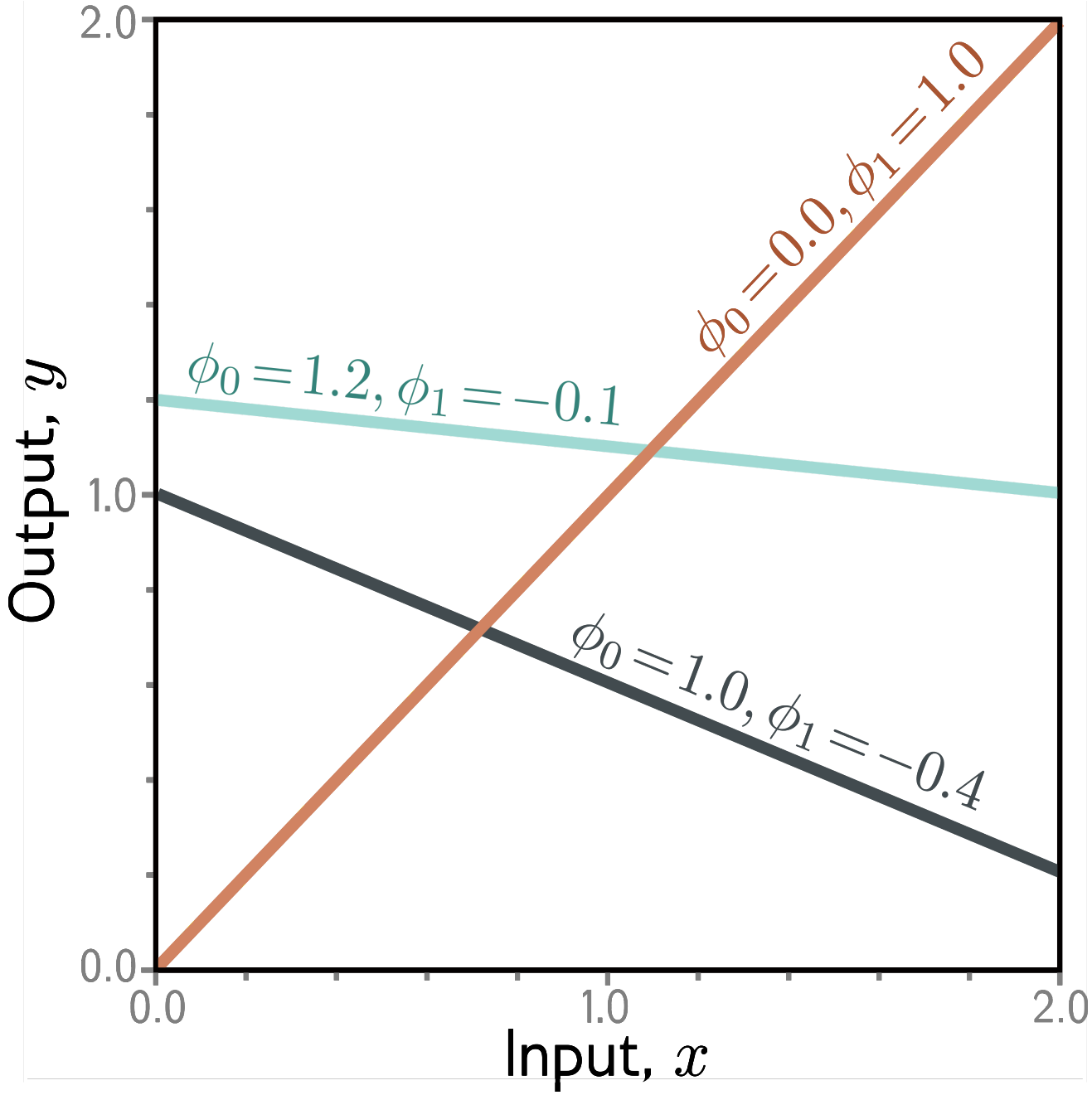
$$y = \mathrm{f}[x, \boldsymbol{\phi}]$$
$$= \phi_0 + \phi_1 x$$

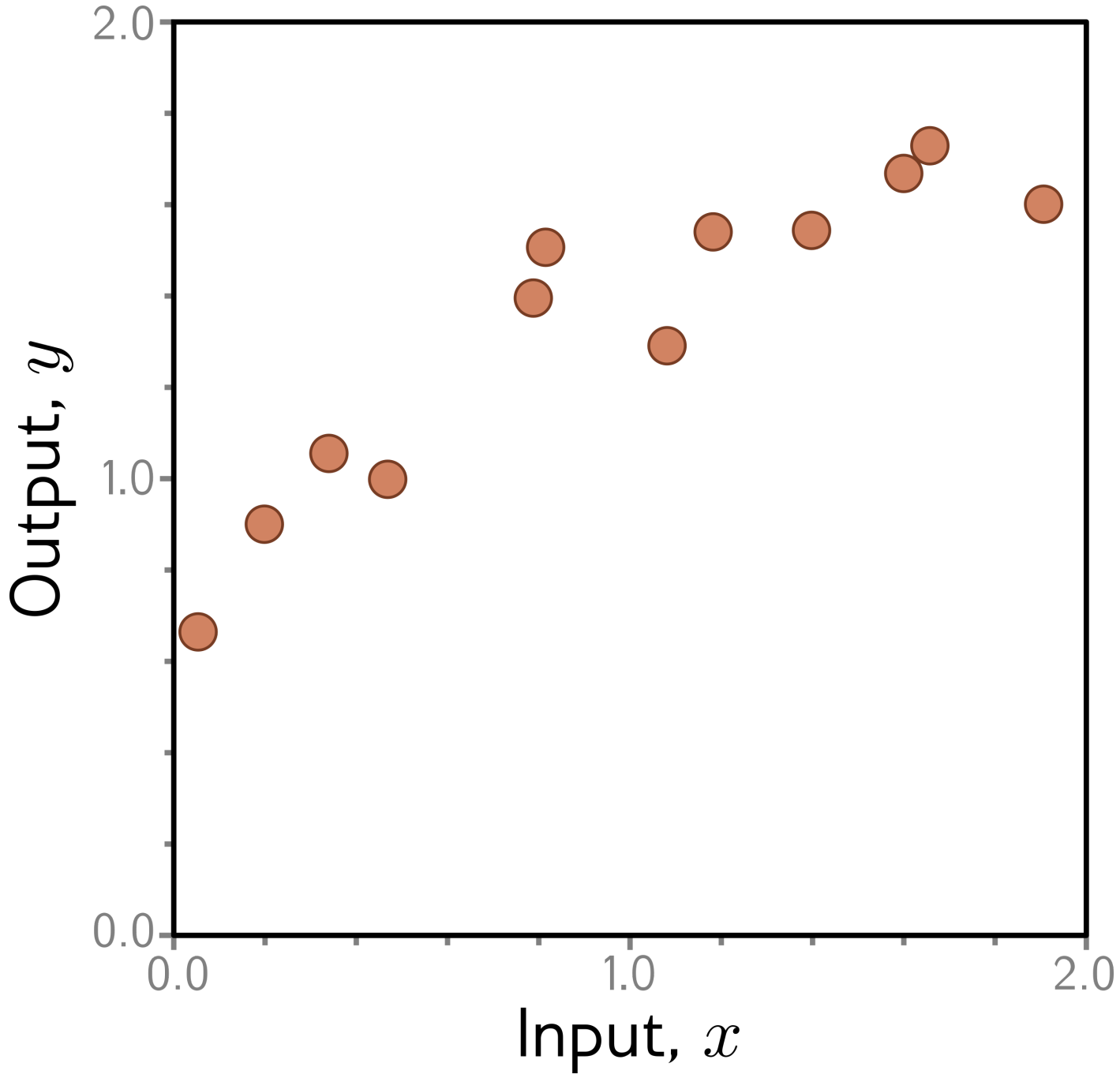
Parameters

$$\boldsymbol{\phi} = \begin{bmatrix} \phi_0 \\ \phi_1 \end{bmatrix}$$

← y-offset

← slope


2.0
1.0
0.0
0.0
1.0
2.0
Output, y
Input, x
$\phi_0 = 0.0, \phi_1 = 1.0$
$\phi_0 = 1.2, \phi_1 = -0.1$
$\phi_0 = 1.0, \phi_1 = -0.4$

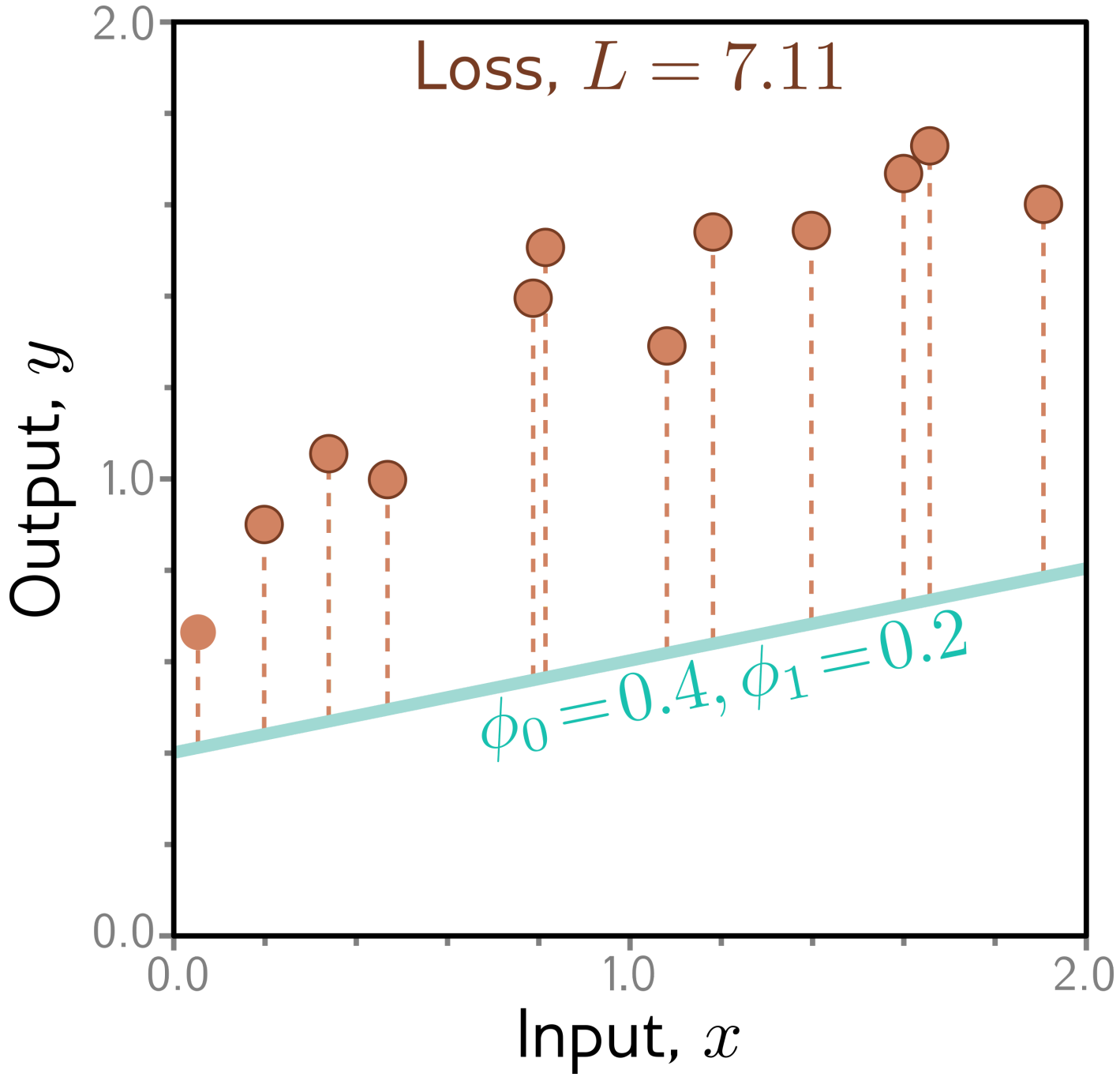


Loss function:

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

$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

Least squares loss function

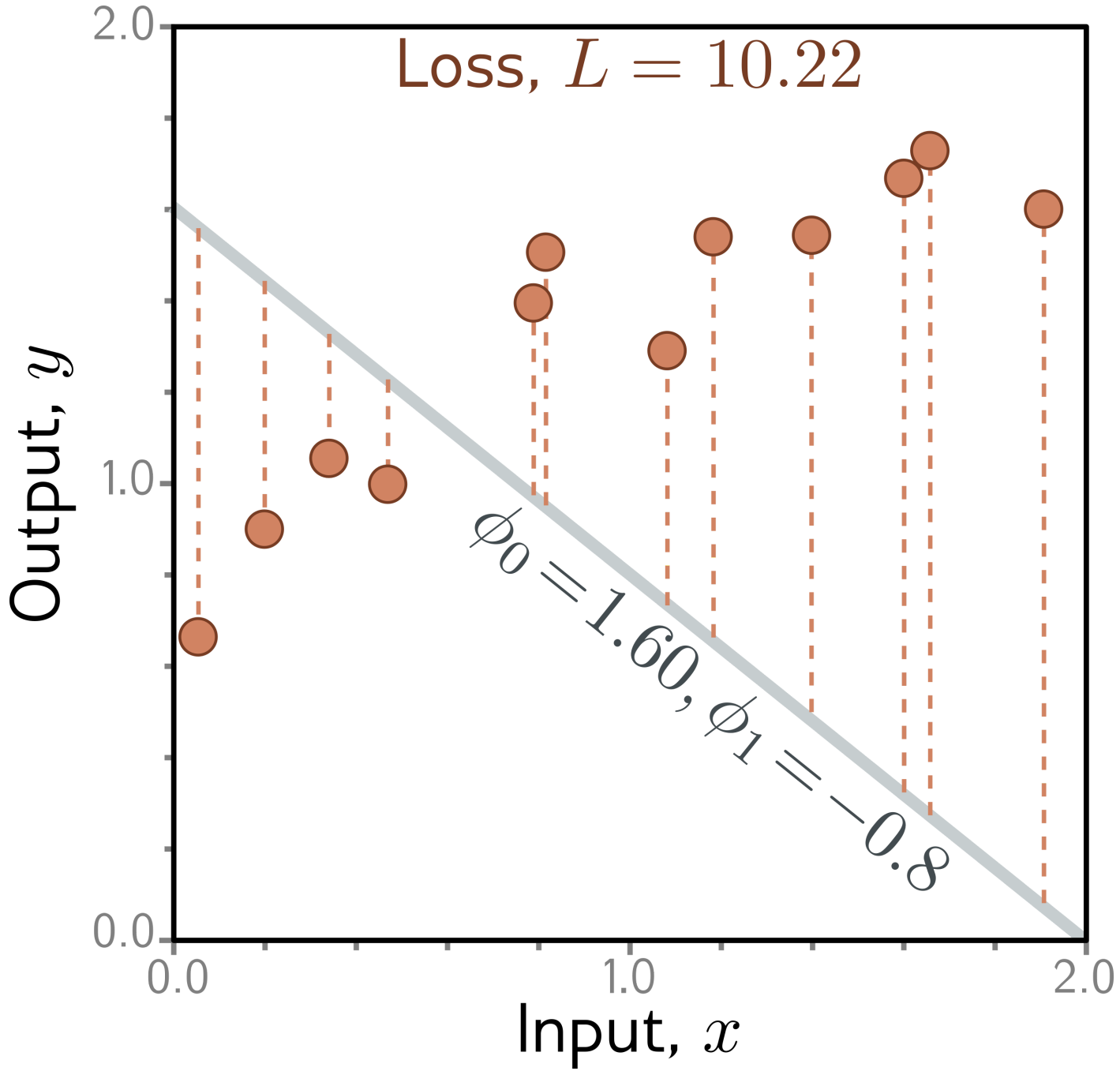


Loss function:

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

$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

Least squares loss function

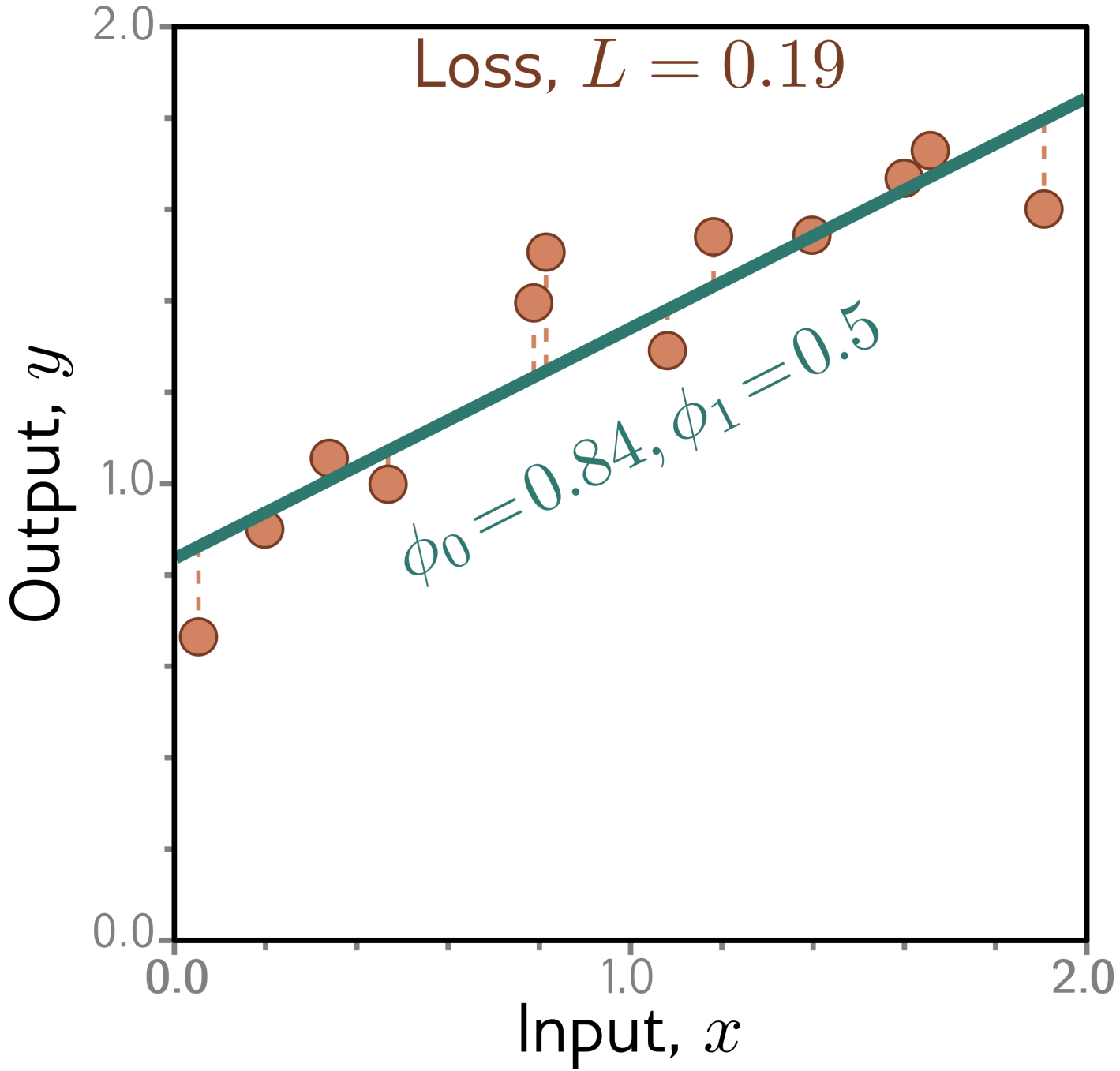


Loss function:

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

$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

Least squares loss function

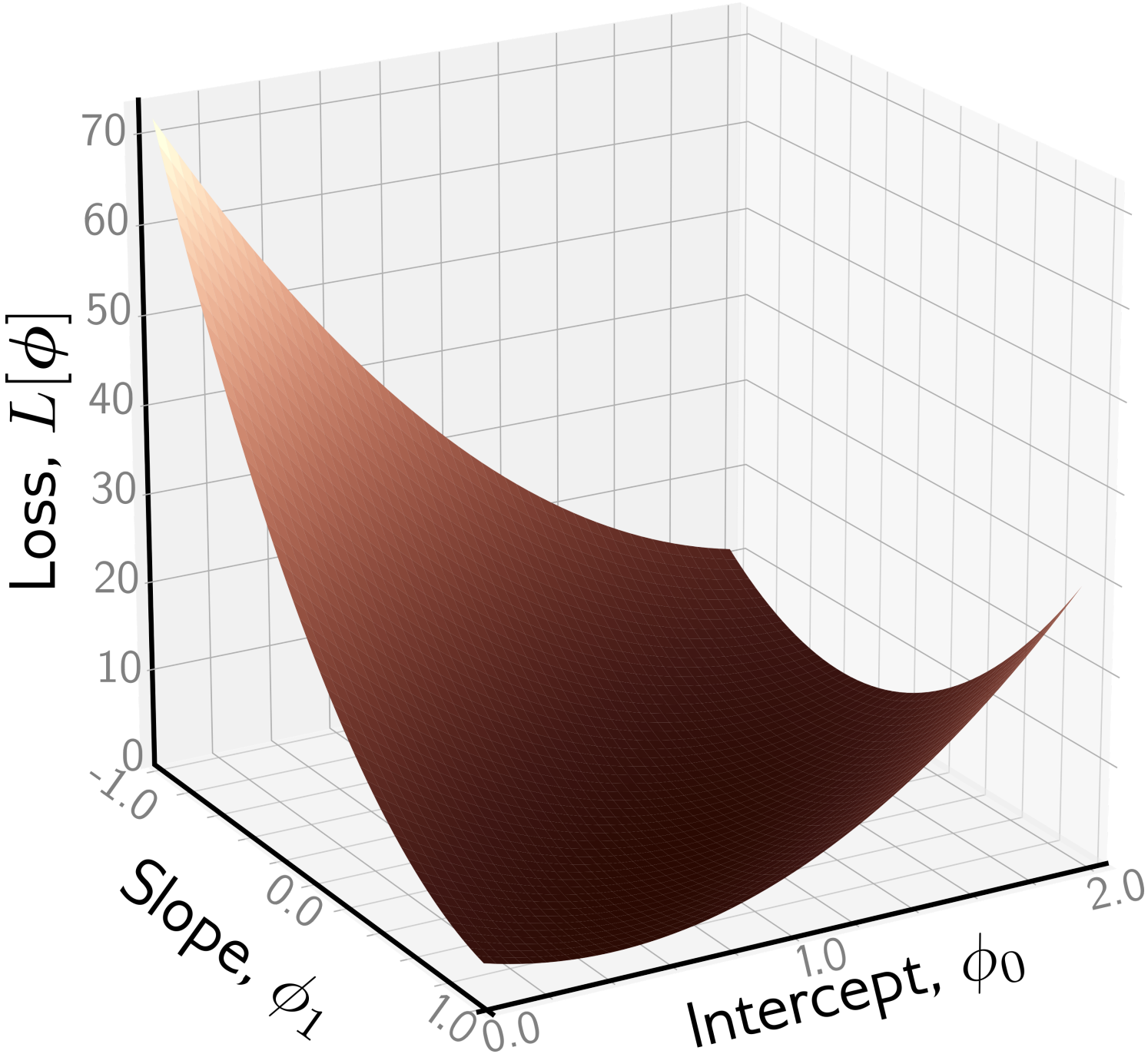


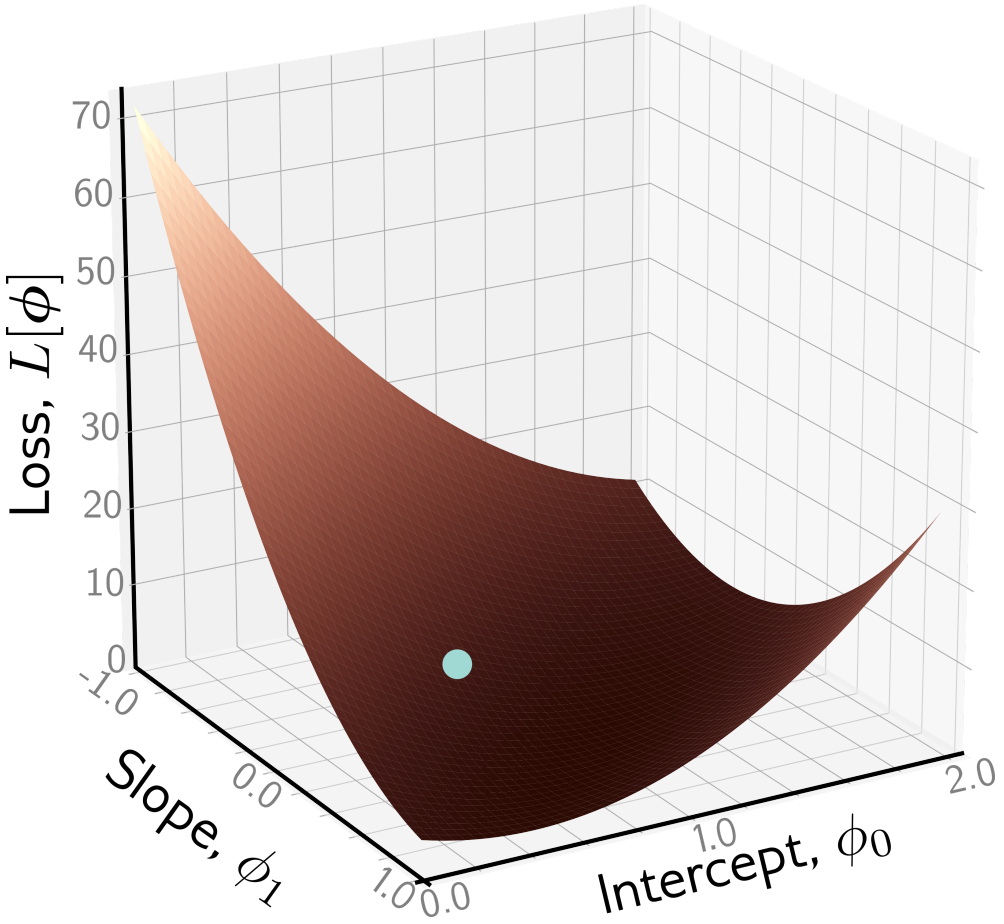
Loss function:

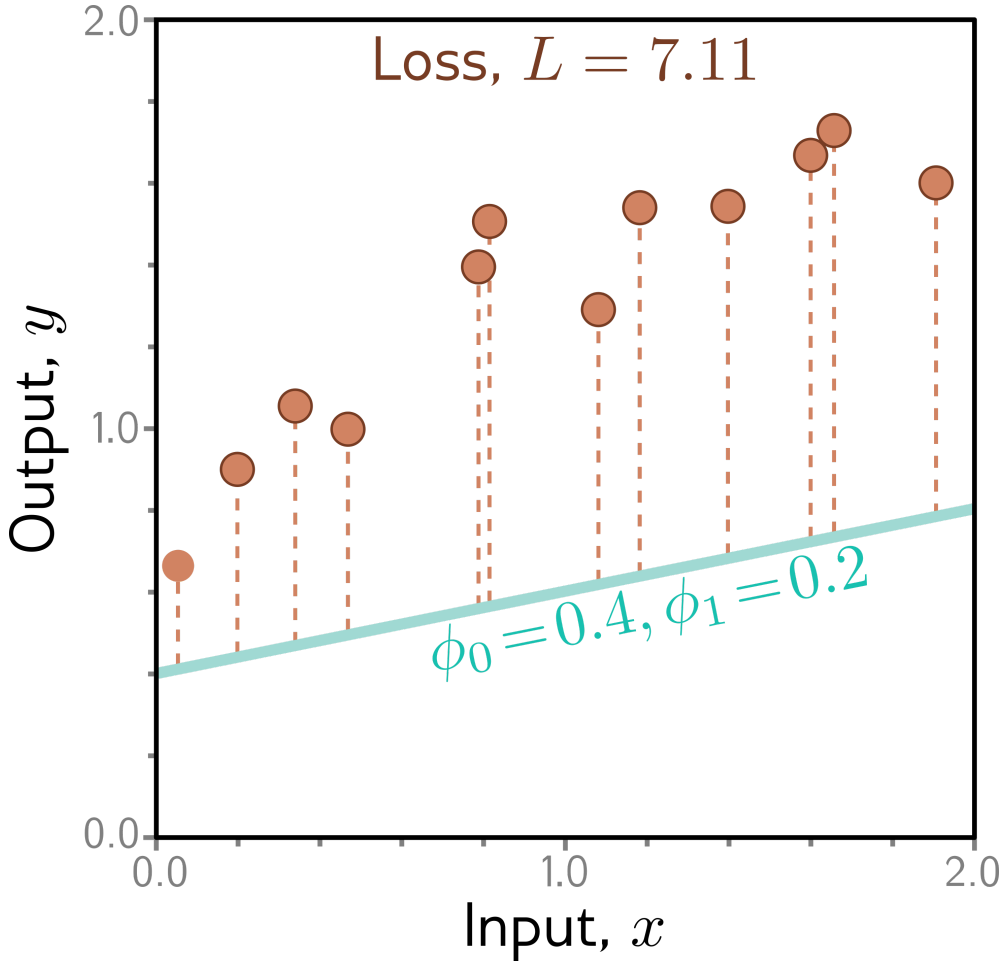
$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$

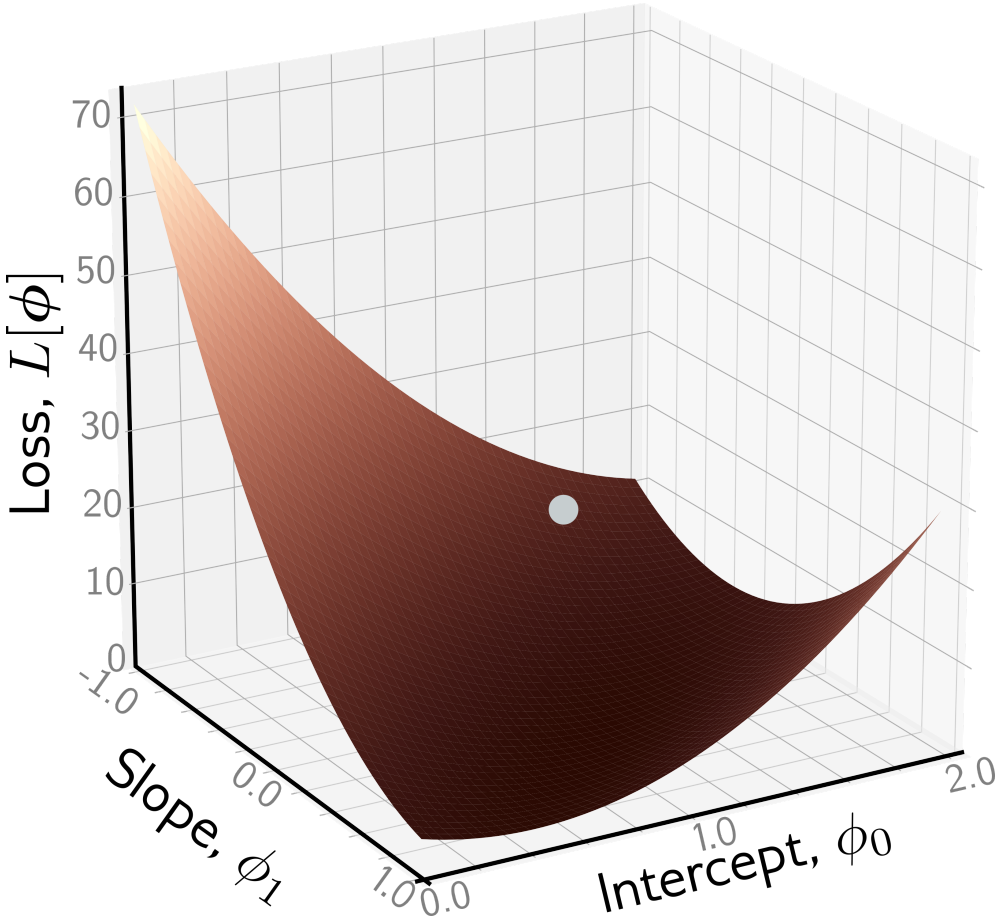
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

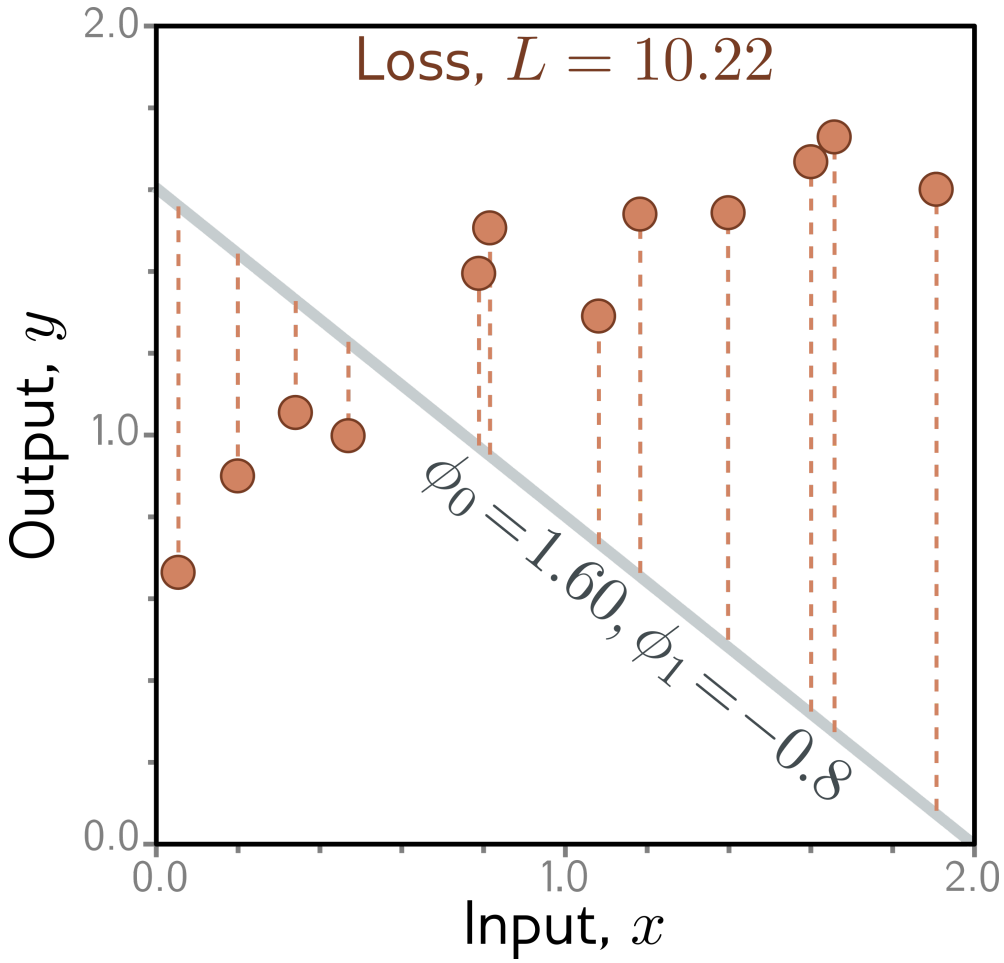
Least squares loss function

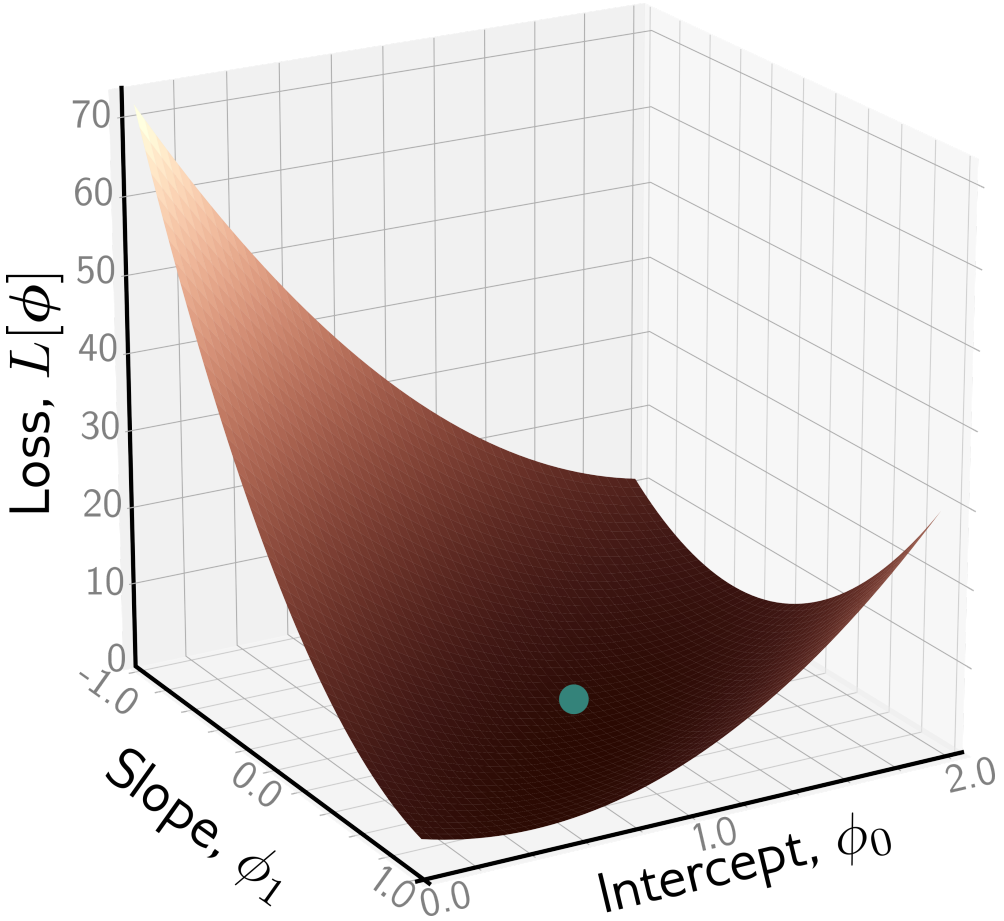

Loss, $L[\phi]$
70
60
50
40
30
20
10
0
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0

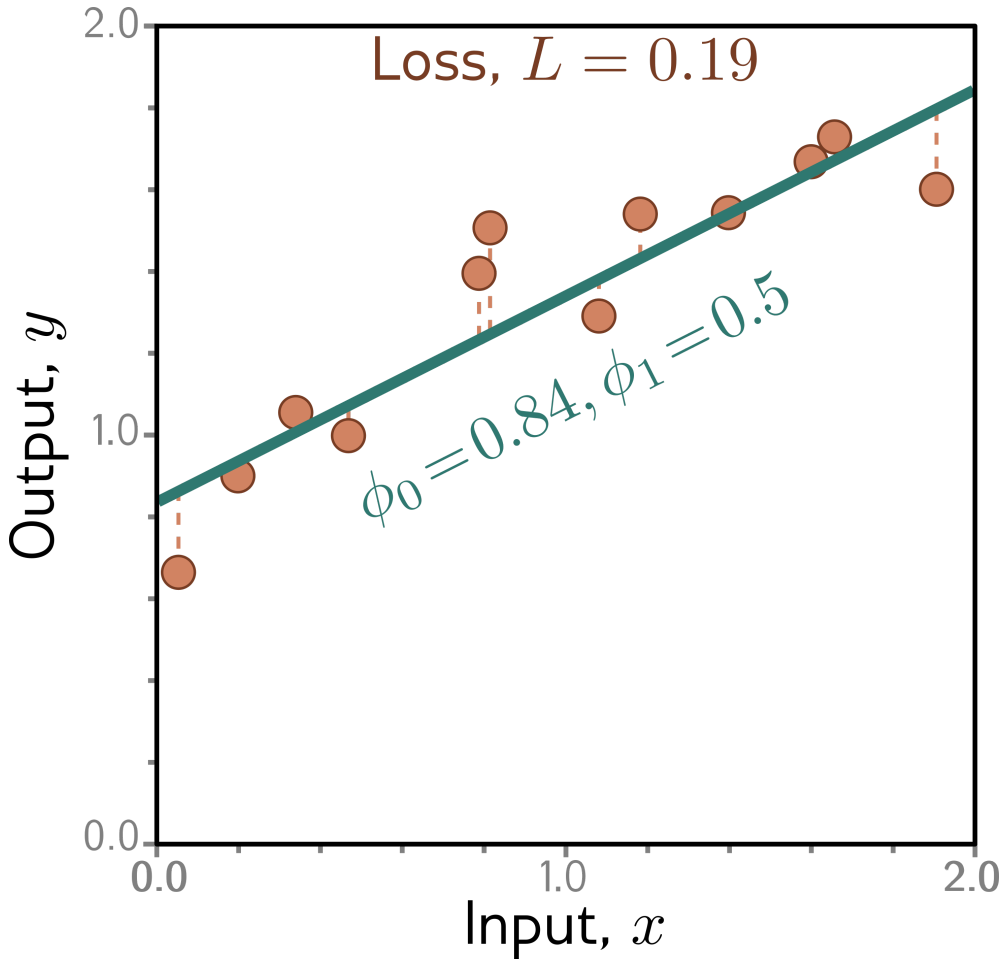

Loss, $L[\phi]$
70
60
50
40
30
20
10
0
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0

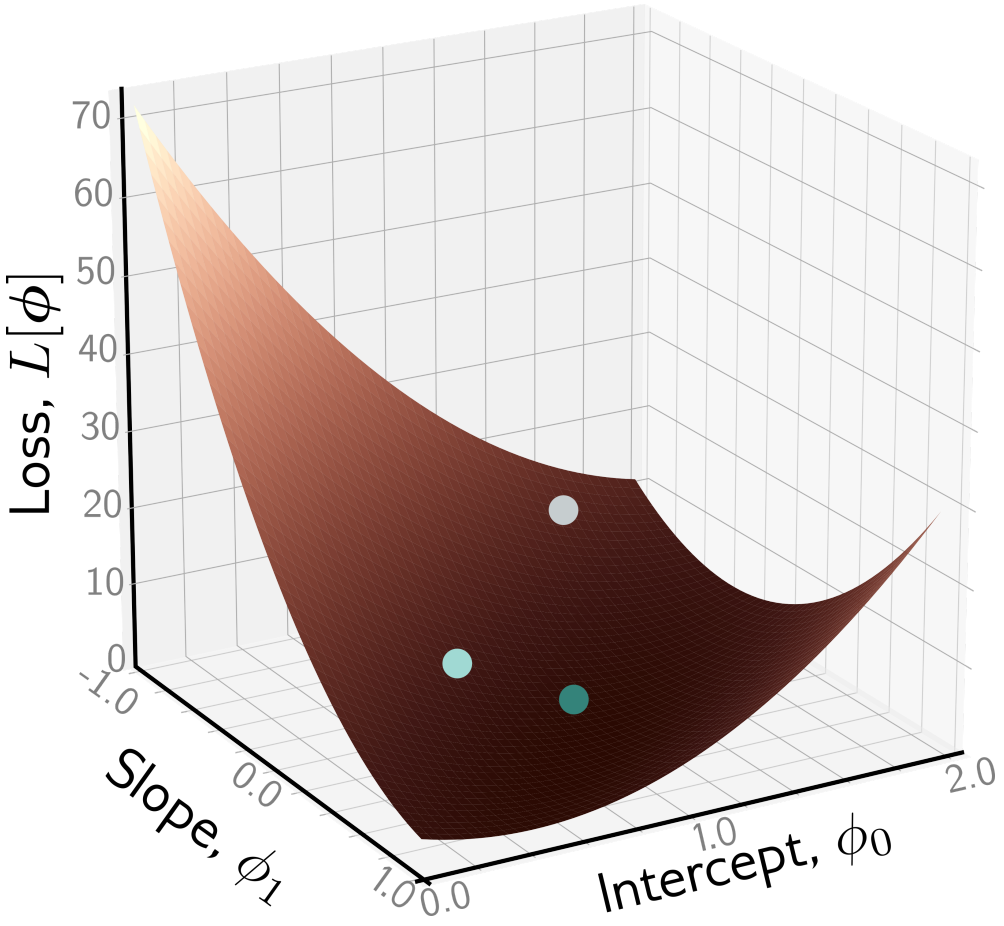

Loss, $L = 7.11$
2.0
1.0
0.0
Output, y
0.0
1.0
2.0
Input, x
$\phi_0 = 0.4, \phi_1 = 0.2$

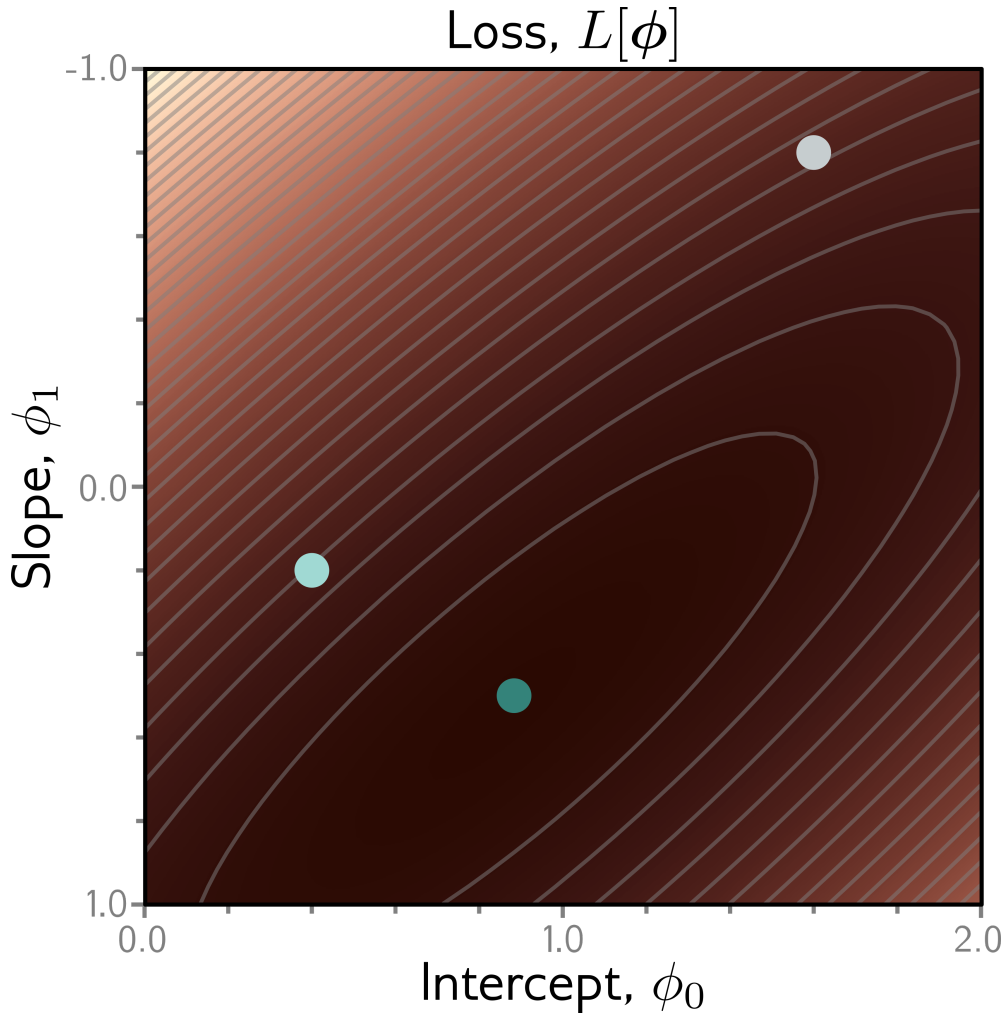

Loss, $L[\phi]$
70
60
50
40
30
20
10
0
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0

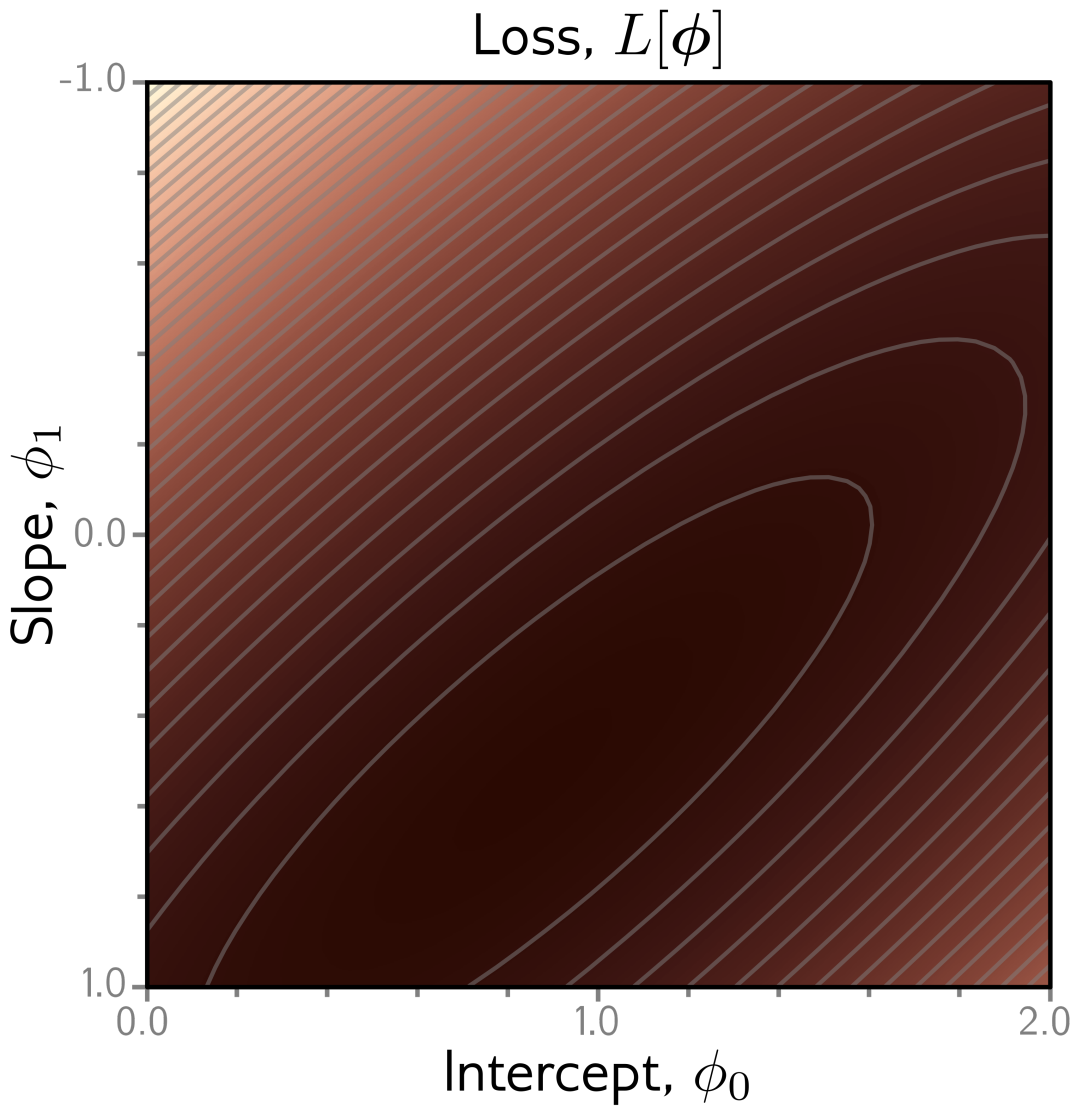

Loss, $L = 10.22$
2.0
1.0
0.0
Output, y
0.0
1.0
2.0
Input, x
$\phi_0 = 1.60, \phi_1 = -0.8$

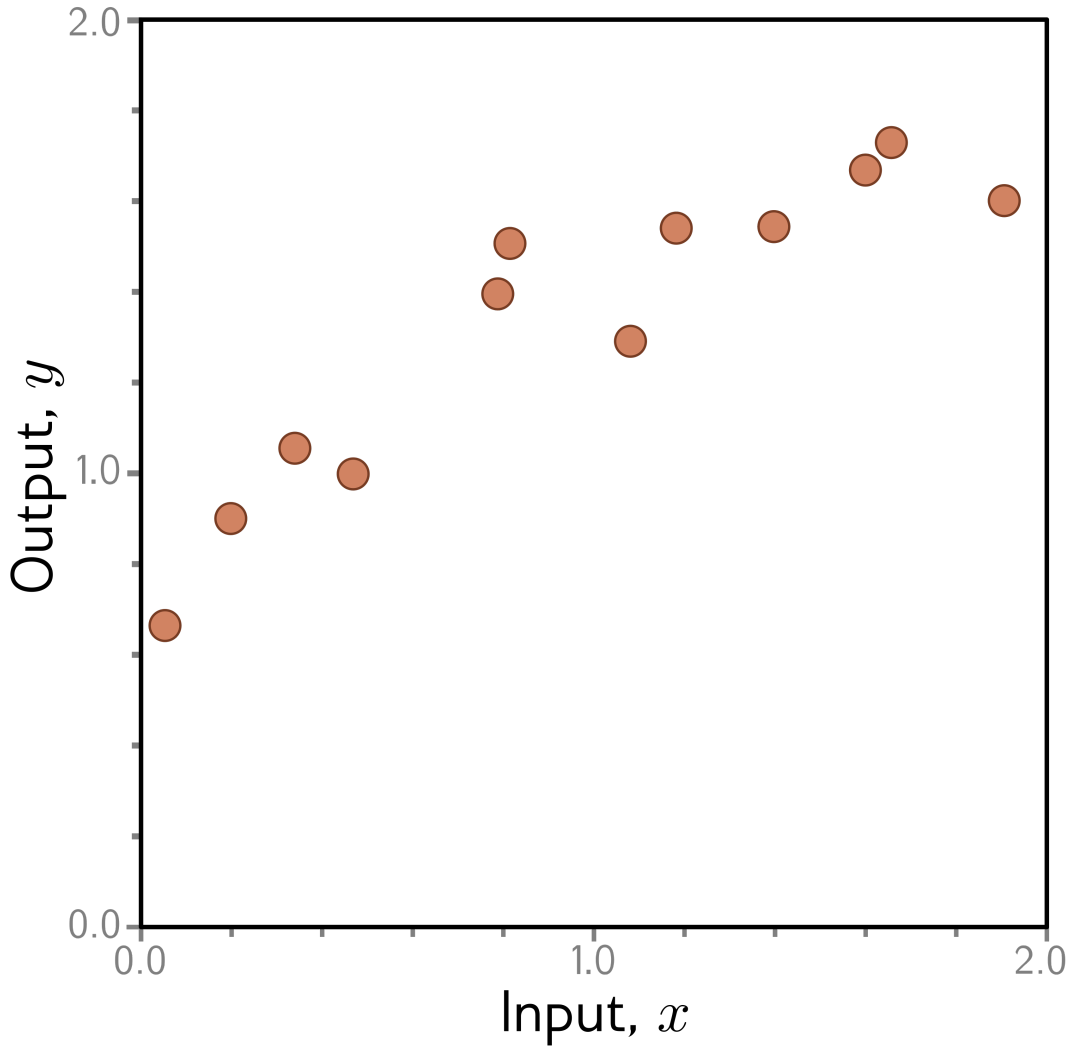

Loss, $L[\phi]$
70
60
50
40
30
20
10
0
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0

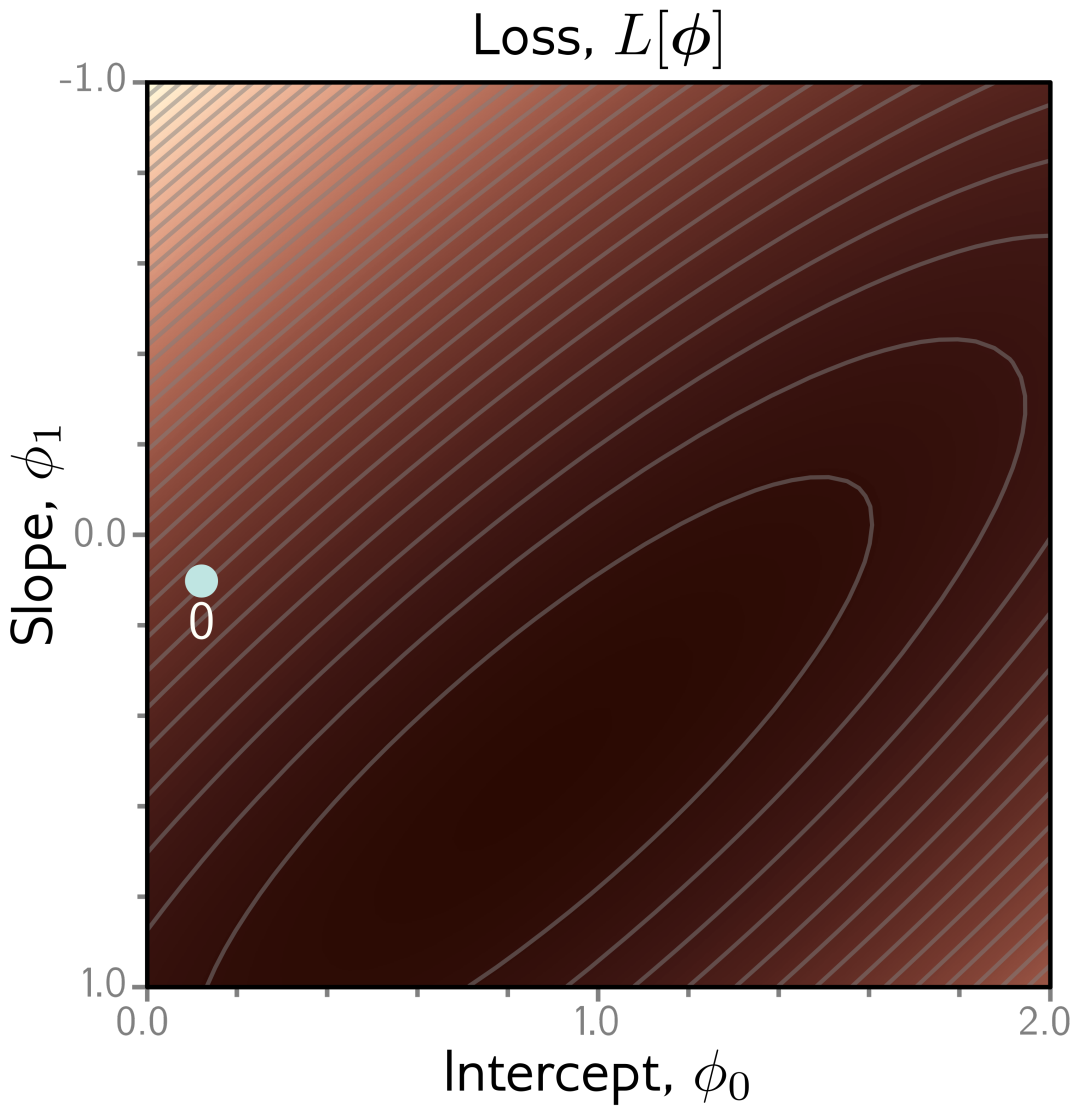

Loss, $L = 0.19$
$\phi_0 = 0.84, \phi_1 = 0.5$
Output, y
2.0
1.0
0.0
0.0
1.0
2.0
Input, x

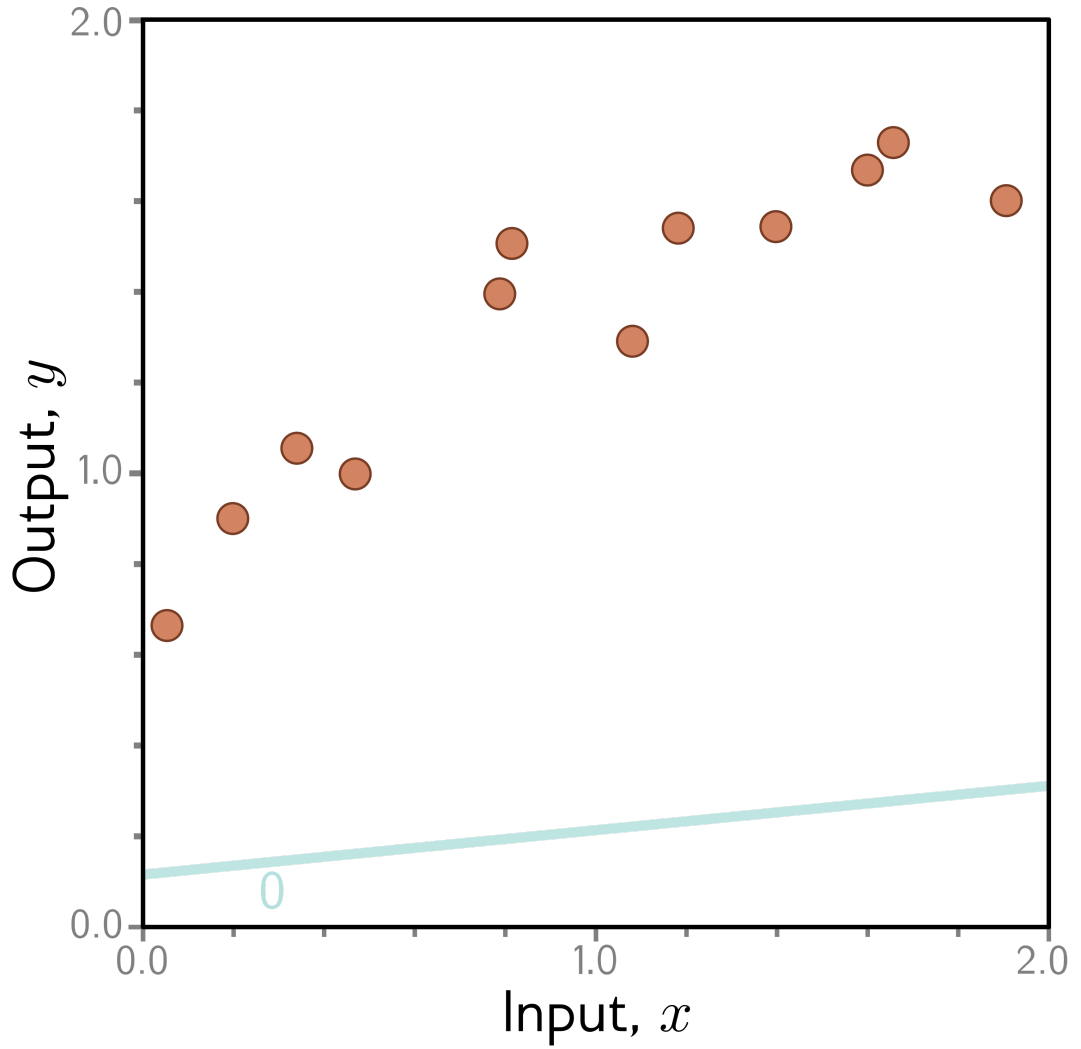

Loss, $L[\phi]$
70
60
50
40
30
20
10
0
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0

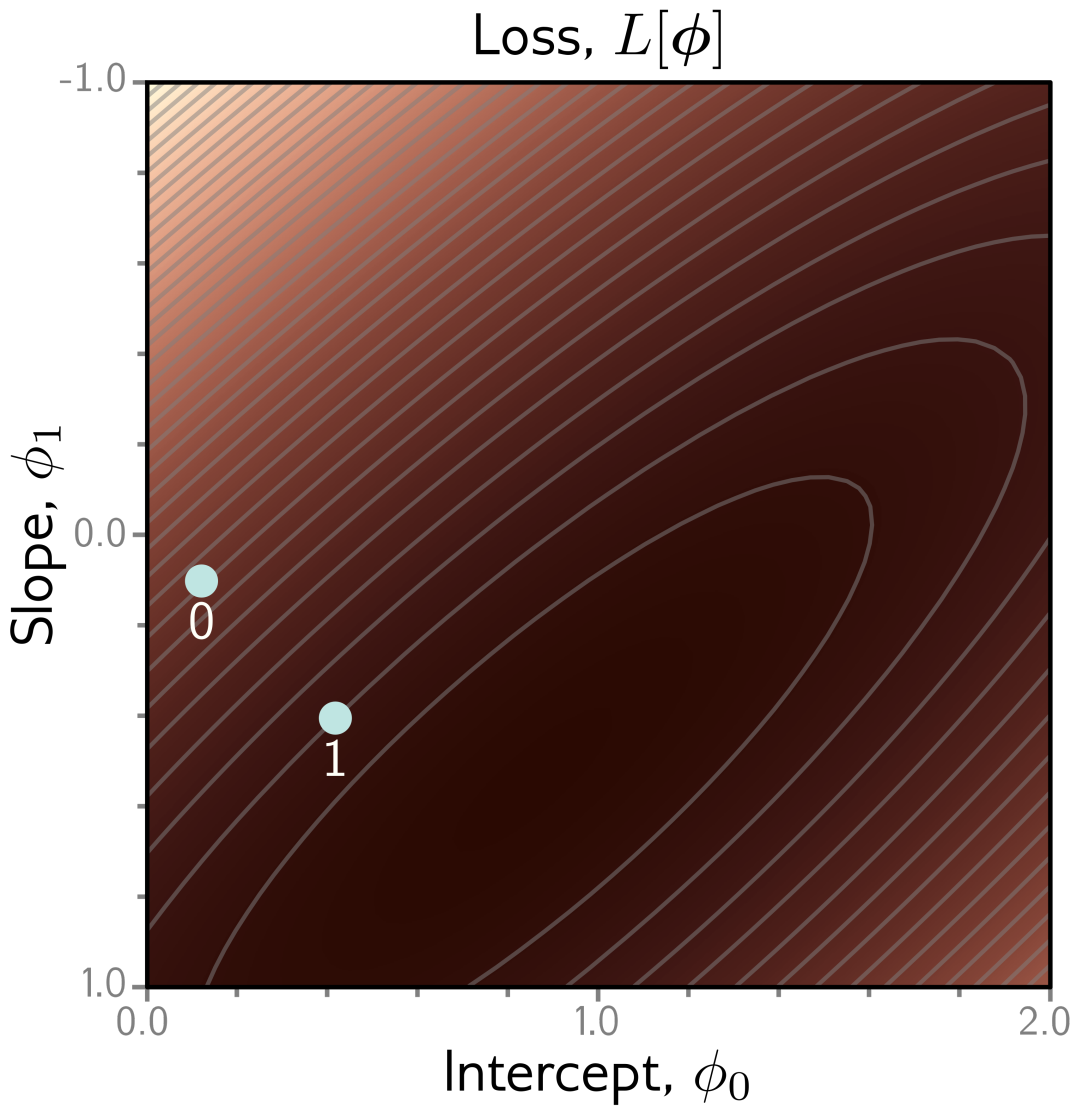

Loss, $L[\phi]$
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0

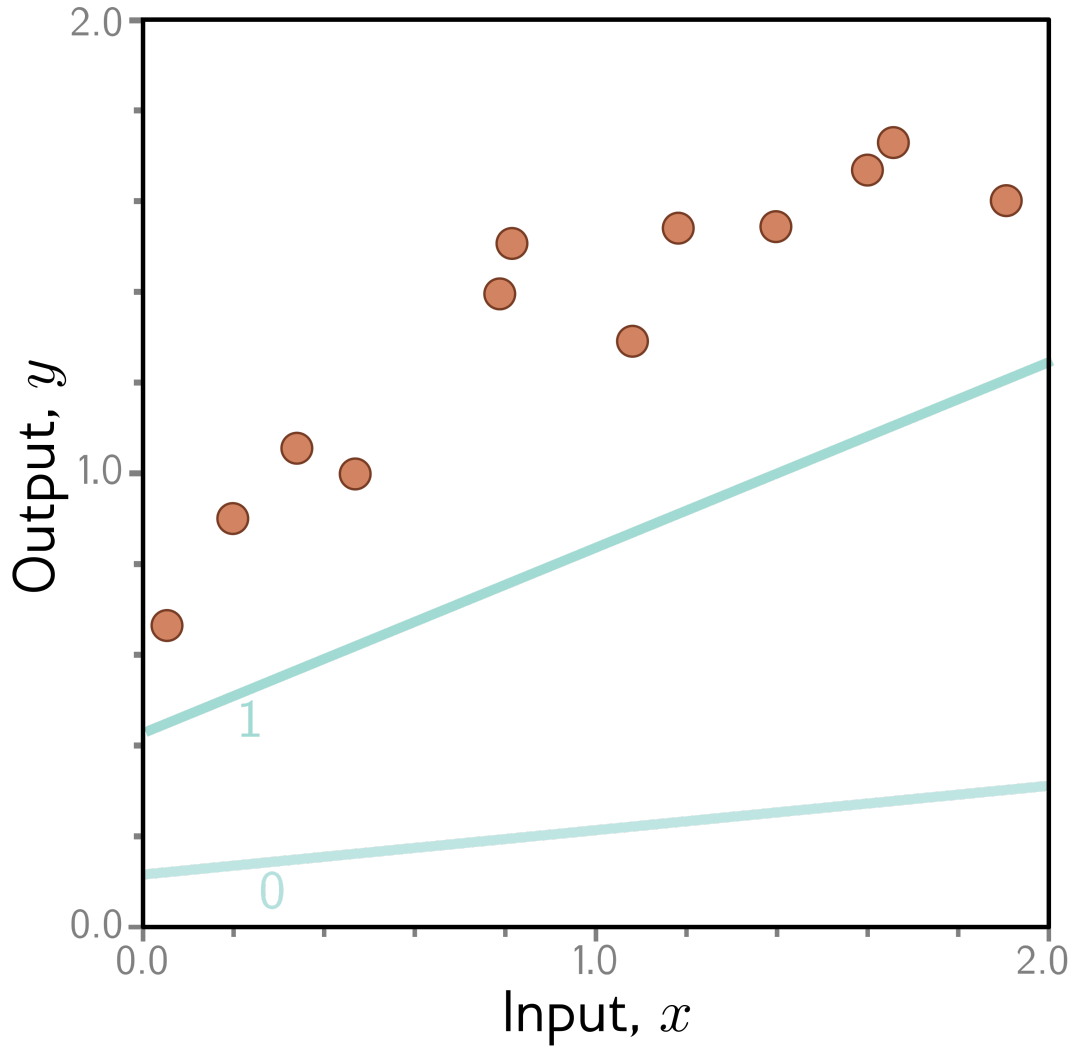

Loss, $L[\phi]$
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0

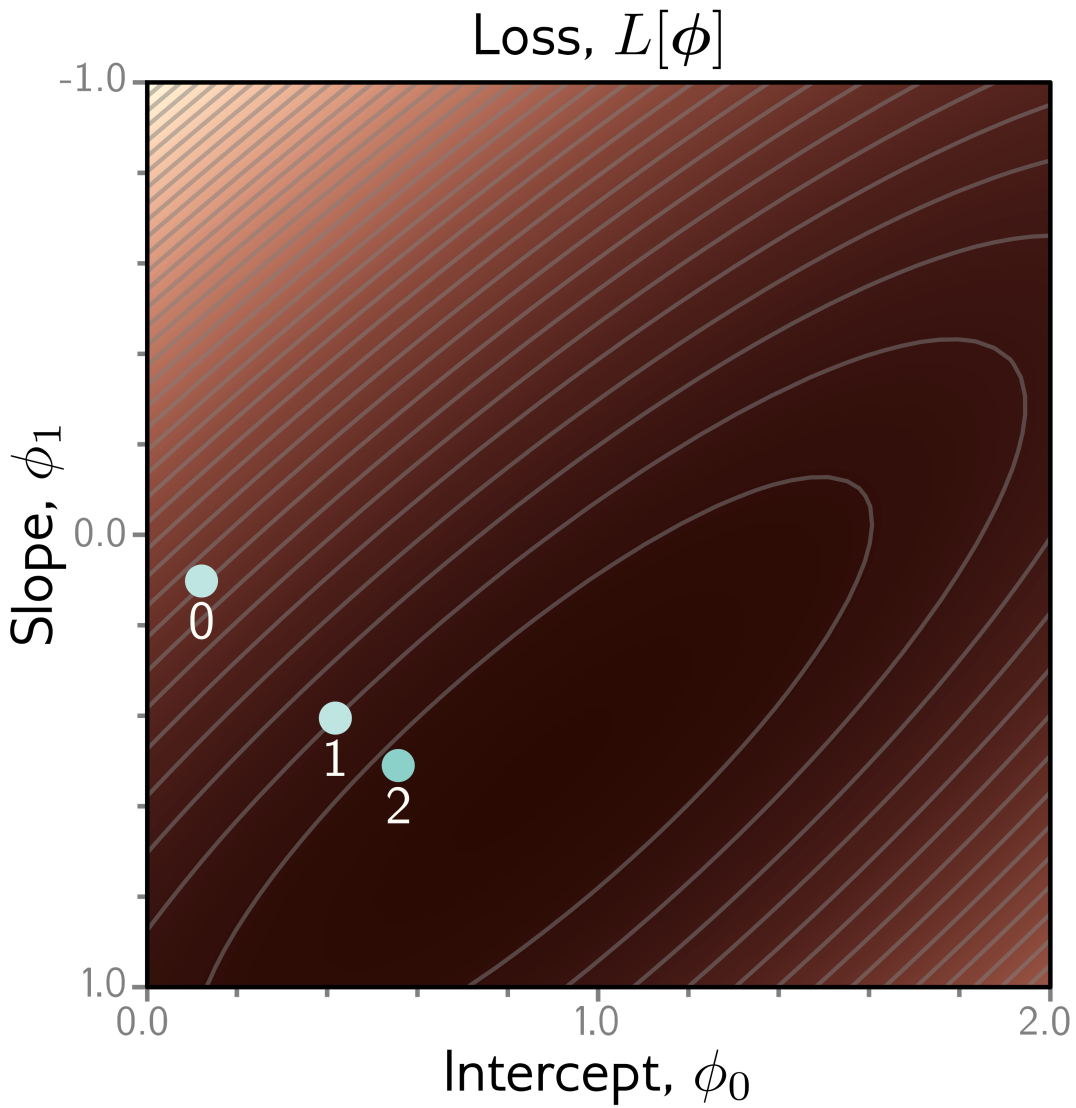

2.0
1.0
0.0
Output, y
0.0
1.0
2.0
Input, x

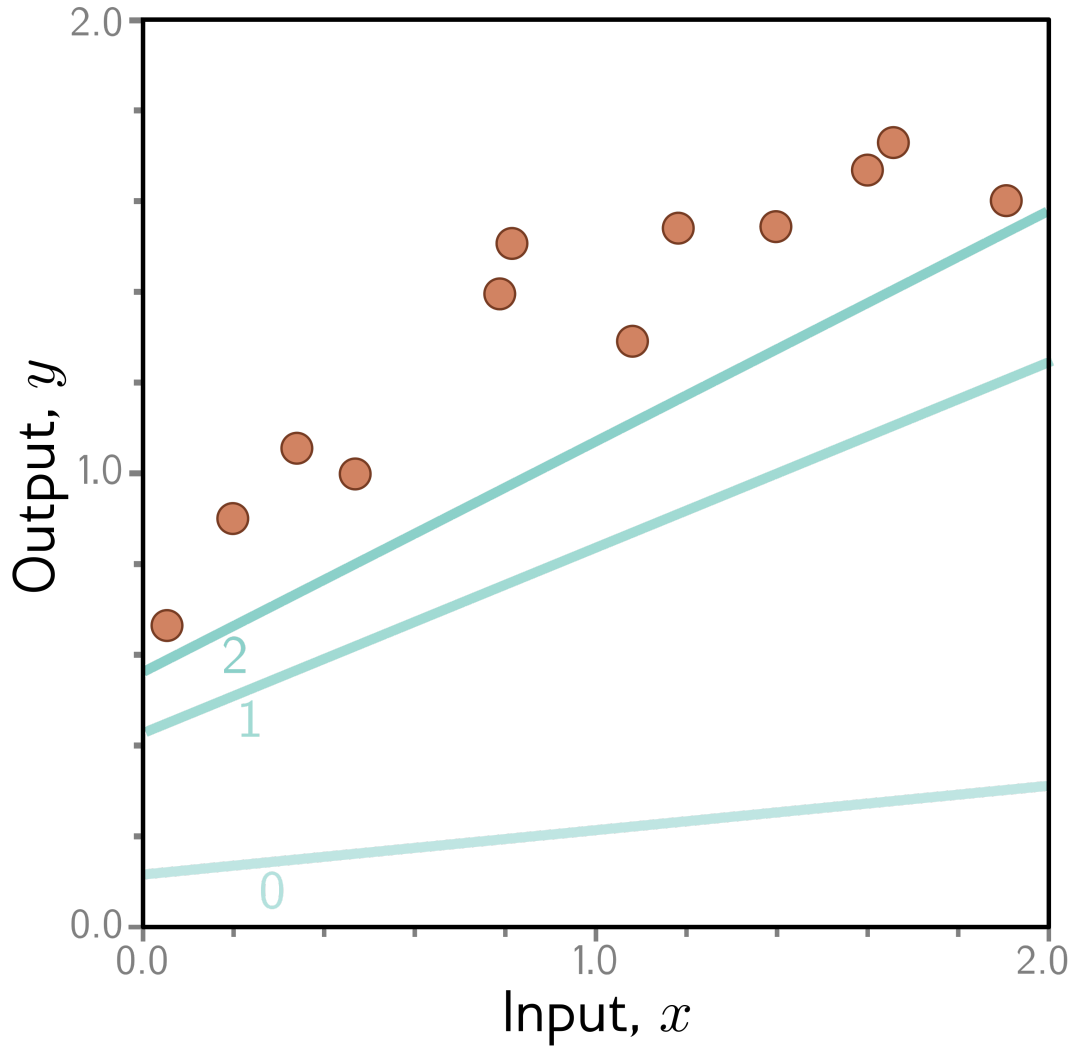

Loss, $L[\phi]$
-1.0
0.0
1.0
Slope, ϕ_1
0
0.0
1.0
2.0
Intercept, ϕ_0

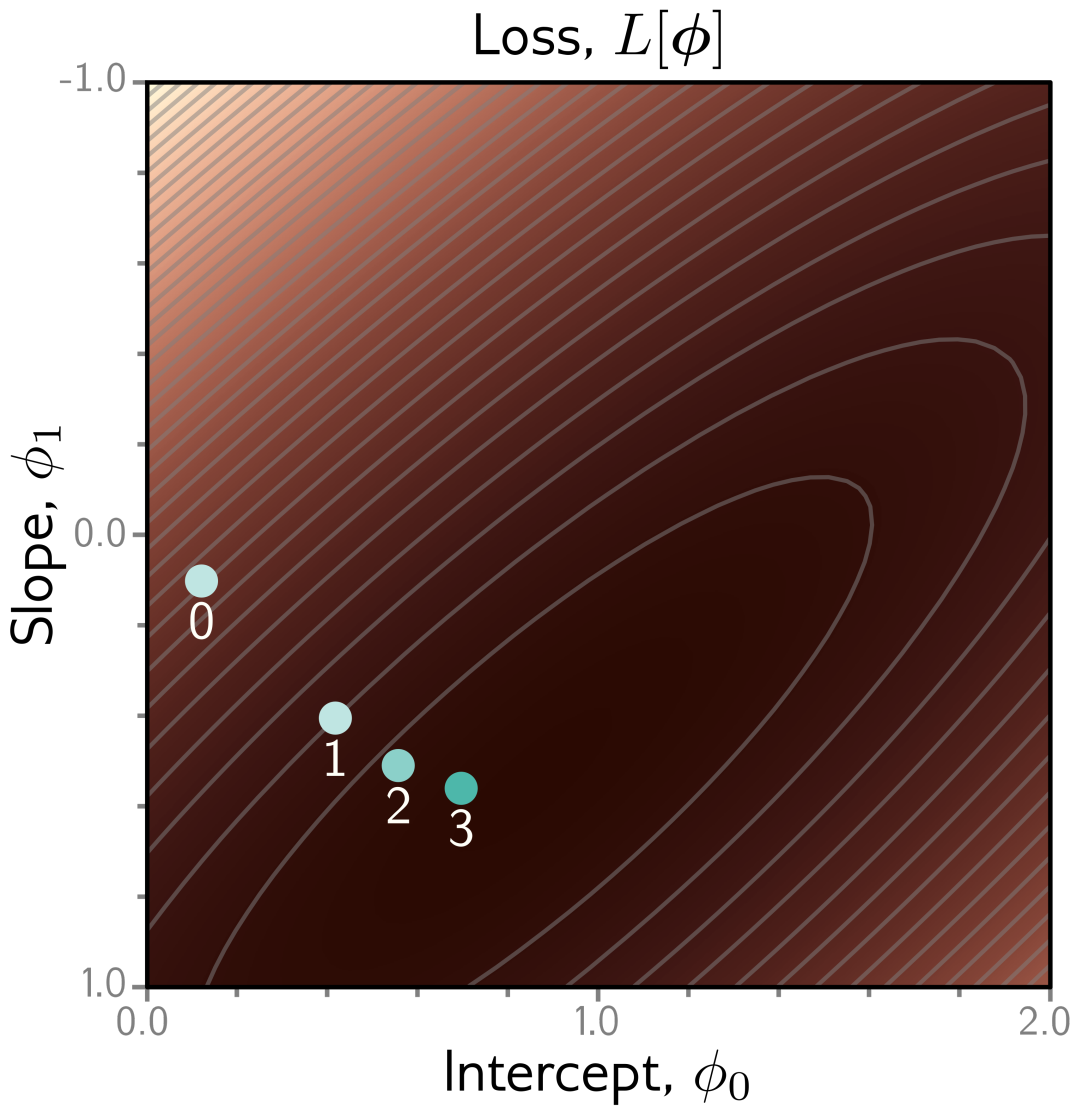

2.0
1.0
0.0
Output, y
0
0.0
1.0
2.0
Input, x

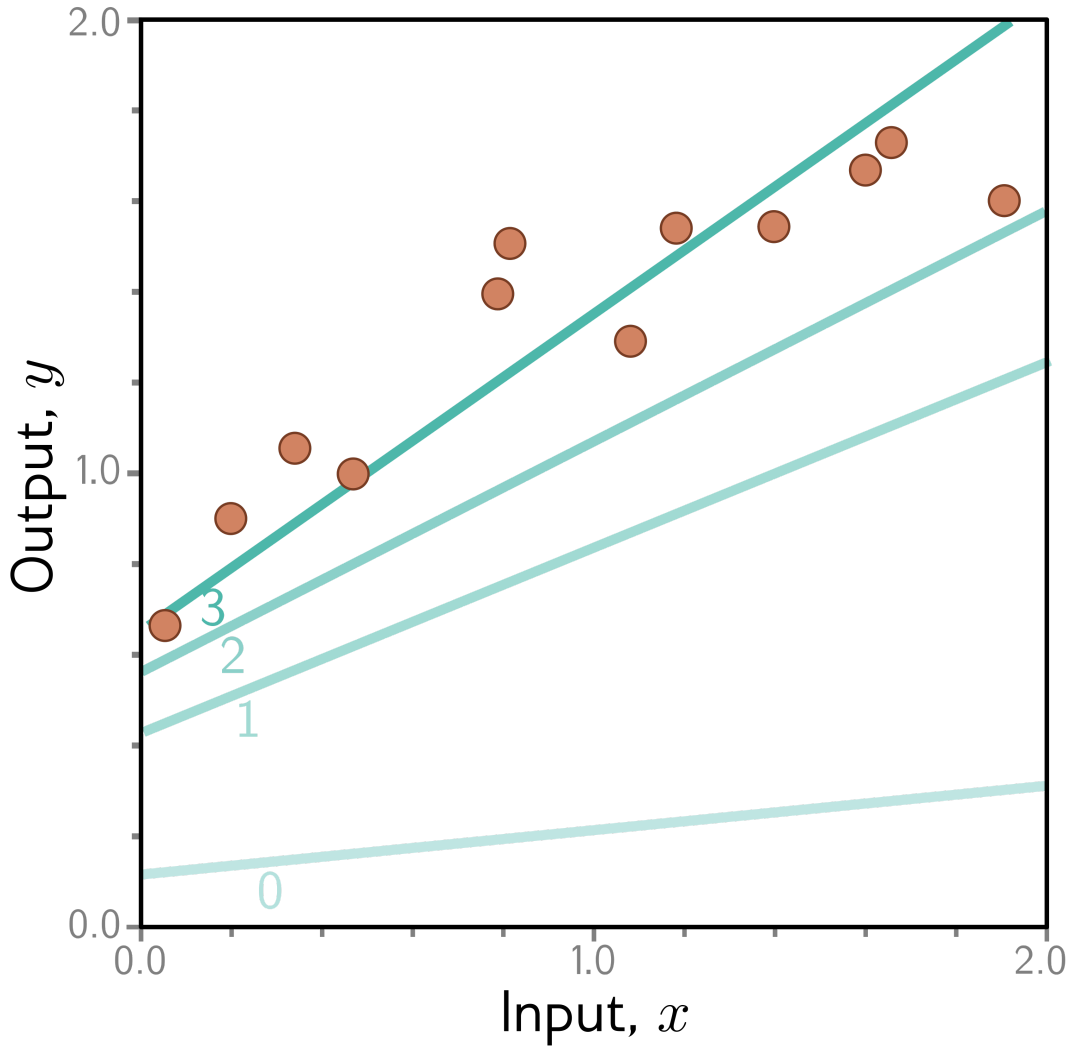

Loss, $L[\phi]$
-1.0
0.0
1.0
Slope, ϕ_1
0
1
0.0
1.0
2.0
Intercept, ϕ_0

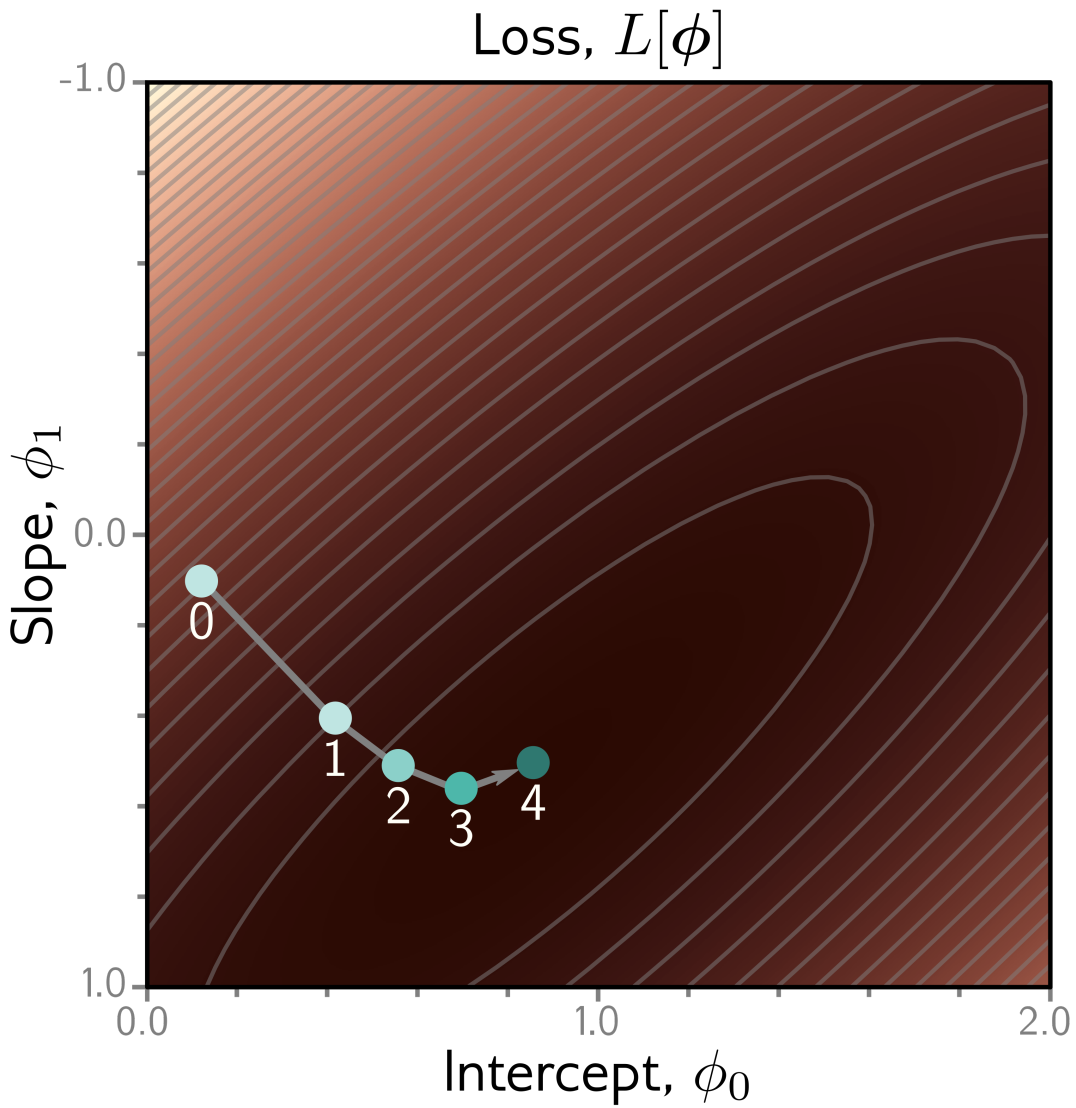

2.0
1.0
0.0
Output, y
1
0
0.0
1.0
2.0
Input, x

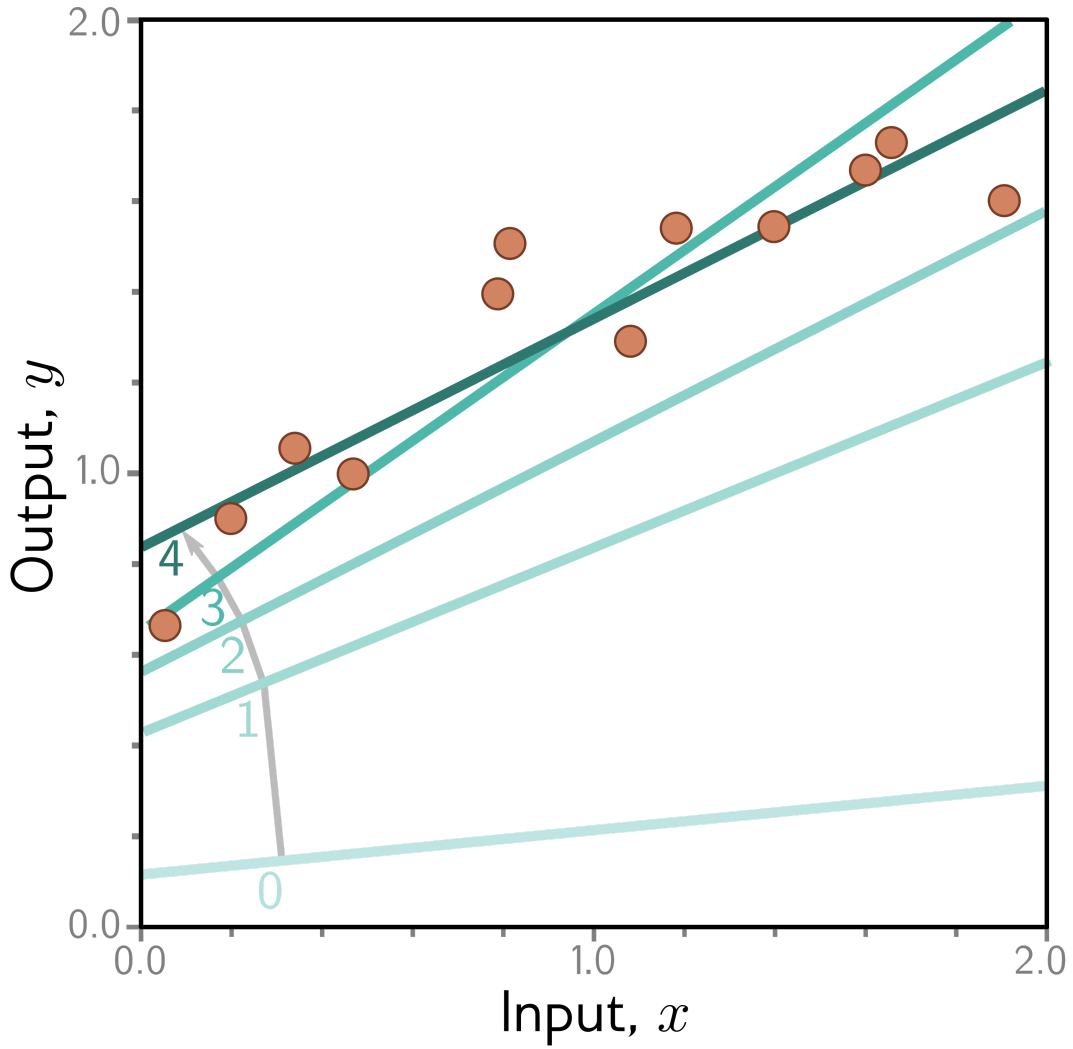

Loss, $L[\phi]$
-1.0
0.0
1.0
Slope, ϕ_1
0
1
2
0.0
1.0
2.0
Intercept, ϕ_0


2.0
1.0
0.0
Output, y
2
1
0
0.0
1.0
2.0
Input, x


Loss, $L[\phi]$
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0
0
1
2
3


2.0
1.0
0.0
Output, y
0.0
1.0
2.0
Input, x
3
2
1
0


Loss, $L[\phi]$
-1.0
0.0
1.0
Slope, ϕ_1
0.0
1.0
2.0
Intercept, ϕ_0
0
1
2
3
4


2.0
1.0
0.0
Output, y
0.0
1.0
2.0
Input, x
4
3
2
1
0

Possible objections?

But you can fit the line model in closed form!

True – but only because we we look at very simple cases so far, we won't be able to do this for more complex models

But we could exhaustively try every slope and intercept combo!

True – but we won't be able to do this when there are a million parameters

What do we aim for in the end?

We test with different set of paired input/output data to measure performance

Degree to which we get the same performance as in training = **generalization**

Might not generalize well because the model is too simple

Or the Model is too complex

It fits to statistical peculiarities of the specific training data we used, not some “general characteristics”

This is known as **overfitting**

Where is all of this going?

- Shallow neural networks (a more flexible model)
- Deep neural networks (an even more flexible model)
- Loss functions (where did least squares come from?)
- How to train neural networks (gradient descent and variants)
- How to measure performance of neural networks (generalization)

Still for today: a practical outlook on Word2Vec

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space (arXiv:1301.3781). arXiv. http://arxiv.org/abs/1301.3781

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

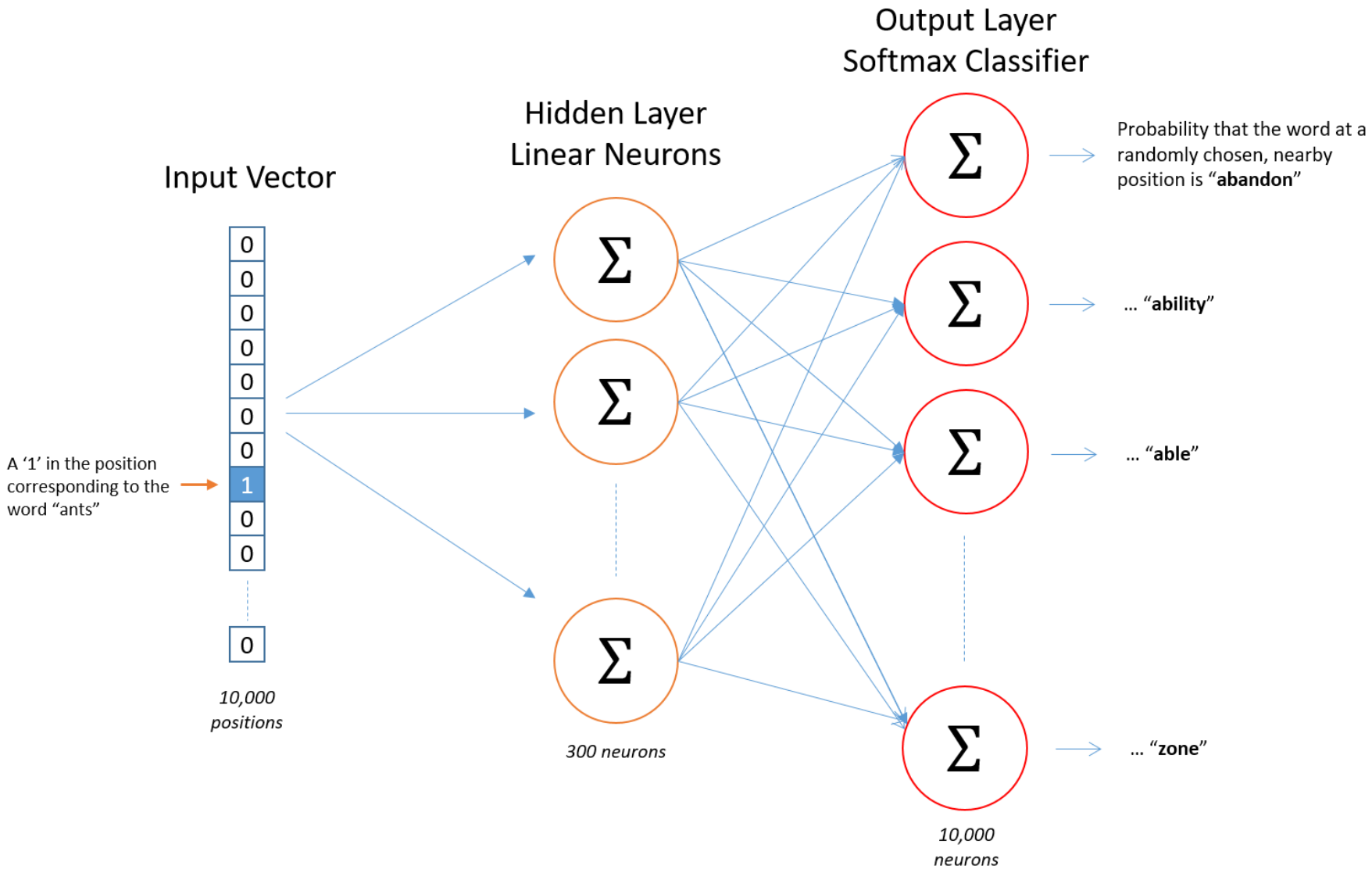
Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems (Vol. 26). Curran Associates, Inc.



https://israelg99.github.io/2017-03-23-Word2Vec-Explained/

Word2Vec

A shallow neural net, which we will cover next time

Surprising relationships could be found in vector space by computing similarities between word vectors

“[...] simple algebraic operations are performed on the word vectors, [and] it was shown for example that vector(”King”) - vector(“Man”) + vector(“Woman”) results in a vector that is closest to the vector representation of the word “Queen”.”

Although the apporach is dated by now: it was used in a large number of different (published) studies in the social sciences over the last years (sometimes also even today)