



N P T E L O N L I N E C E R T I F I C A T I O N C O U R S E S

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 6 : Supervised Learning



PROF . PAWAN GOYAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Introduction to Supervised Learning
- Example: 1D Linear Regression

- Building systems that simulate intelligent behavior
- Wide range of approaches: logic, search, probabilistic reasoning, ...

- Building systems that simulate intelligent behavior
- Wide range of approaches: logic, search, probabilistic reasoning, ...

Machine learning

- A subset of AI that learns to make decisions by fitting mathematical models to observed data
- (incorrectly) almost synonymous with AI

Machine learning

Supervised learning

Define a mapping from input data to an output prediction

Unsupervised learning

Constructing a model from input data without corresponding output labels

Reinforcement learning

Introduces the idea of an agent that lives in the world and learns to choose actions leading to high reward

Machine learning

Supervised learning

Unsupervised learning

Reinforcement learning

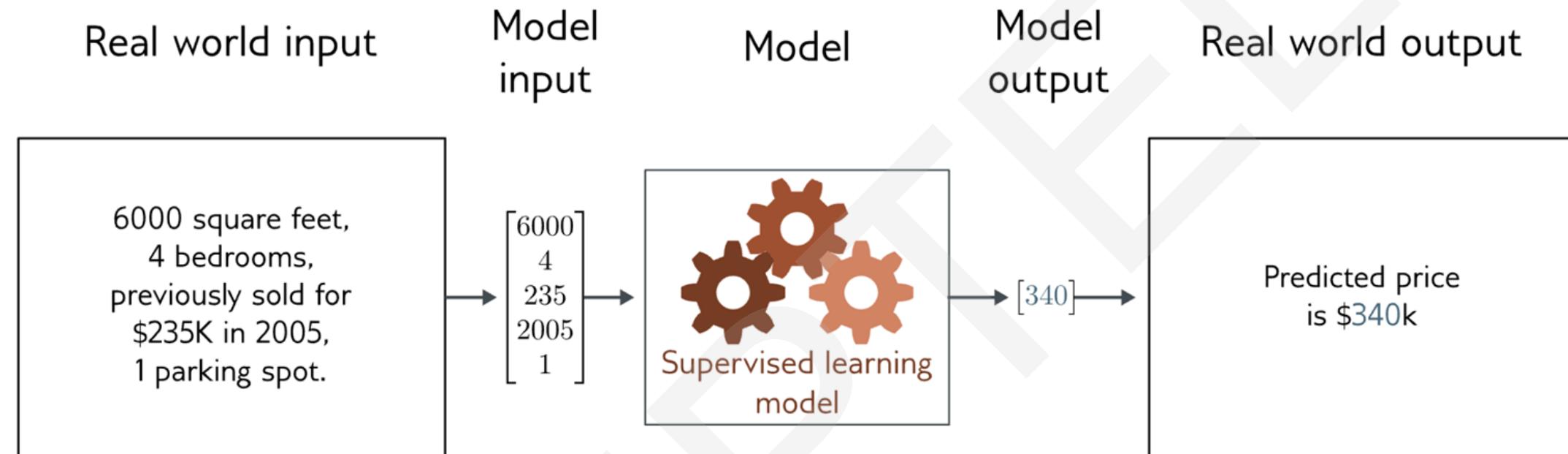
Deep learning

Deep Neural Networks contribute to each of the areas

Supervised learning

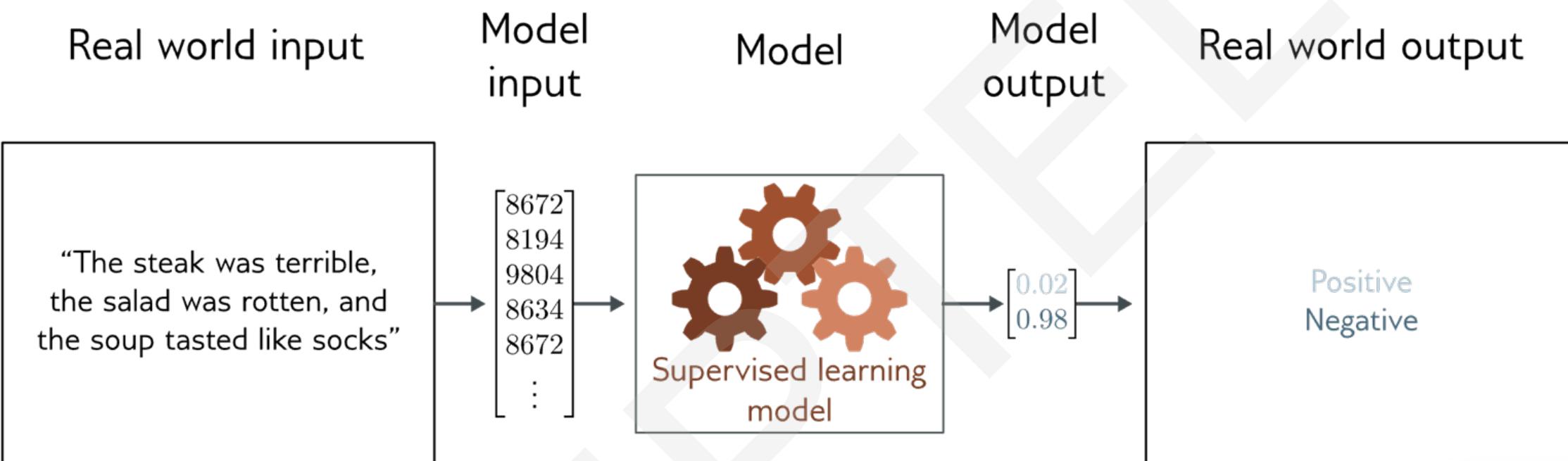
- Define a mapping from input to output
- Learn this mapping from paired input/output data examples

Regression



- Univariate regression problem (one output, real value)
- Fully connected network

Text classification



- Binary classification problem (two discrete classes)
- Recurrent Neural Networks / Transformer network

Image classification

Real world input



Model
input

$$\begin{bmatrix} 124 \\ 140 \\ 156 \\ 128 \\ 142 \\ 157 \\ \vdots \end{bmatrix}$$

Model



Model
output

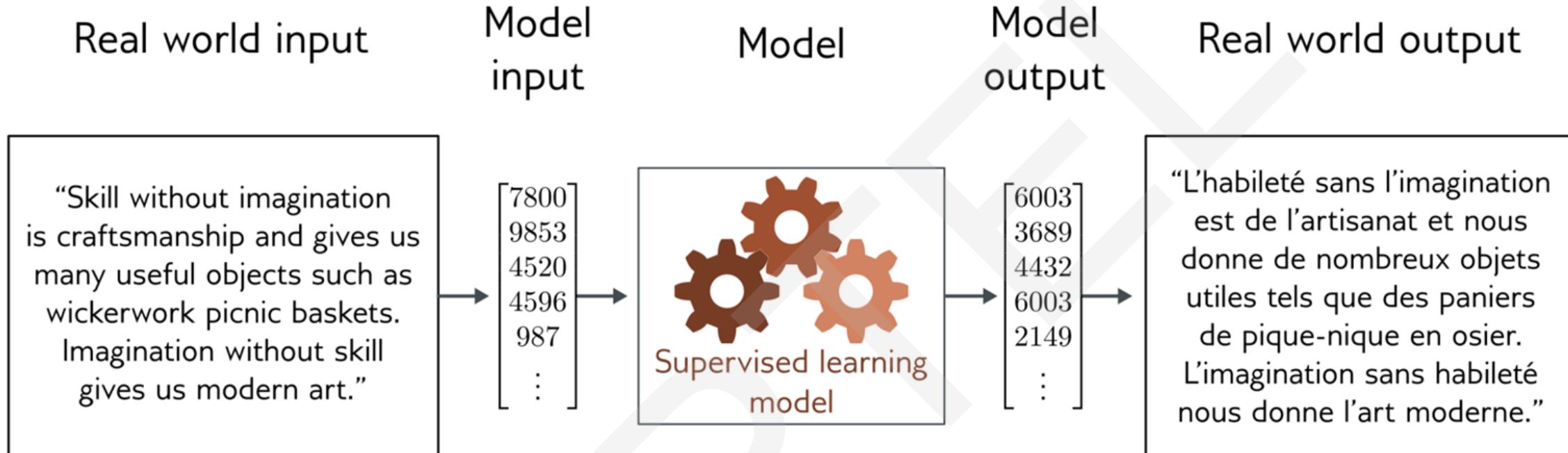
$$\begin{bmatrix} 0.00 \\ 0.00 \\ 0.01 \\ 0.89 \\ 0.05 \\ 0.00 \\ \vdots \\ 0.01 \end{bmatrix}$$

Real world output

Aardvark
Apple
Bee
Bicycle
Bridge
Clown
⋮

- Multiclass classification problem (discrete classes, >2 possible classes)
- Convolutional network

Translation



There can be structure in output as well!

Image captioning

Real world input



Model
input

$$\begin{bmatrix} 183 \\ 204 \\ 231 \\ 185 \\ 204 \\ 232 \\ \vdots \end{bmatrix}$$

Model



Model
output

$$\begin{bmatrix} 1 \\ 5593 \\ 7532 \\ 7924 \\ 1 \\ \vdots \end{bmatrix}$$

Real world output

"A Kazakh man on a
horse holding a
bird of prey"

There can be structure in output as well!

What do we see in these examples?

- There is a ***meaningful real-world input*** (Sentence, image, etc.)
- The input is encoded as a vector of numbers, forming the model input
- The model maps the input to an output vector
- The output vector is translated back to a ***meaningful real-word prediction.***

Supervised learning overview

- **Supervised learning model** = mapping from one or more inputs to one or more outputs
- Model is a mathematical equation
- Example:
 - Input is age and mileage of second hand Toyota Prius
 - Output is estimated price of car

Supervised learning overview

- Supervised learning model = mapping from one or more inputs to one or more outputs
- Model is a mathematical equation
- Computing the outputs from the inputs = inference
- Model also includes parameters
- Parameters affect outcome of equation

Supervised learning overview

- Supervised learning model = mapping from one or more inputs to one or more outputs
- ~~Model is a mathematical equation~~
- Model is a family of equations
- Computing the outputs from the inputs = inference
- Model also includes parameters
- Parameters affect outcome of equation
- Training a model = finding parameters that predict outputs “well” from inputs for a training dataset of input/output pairs

Supervised Learning: Notation

- Input:

$$\mathbf{x} = \begin{bmatrix} \text{age} \\ \text{mileage} \end{bmatrix}$$

Structured or
tabular data

- Output:

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

- Model:

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

Supervised Learning: Model

- Parameters:

$$\phi$$

- Model :

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

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:

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

model train data

or for short:

$$L[\phi] \leftarrow$$

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

Training

- Loss function:

$$L [\phi]$$



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

- Find the parameters that minimize the loss:

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

Testing

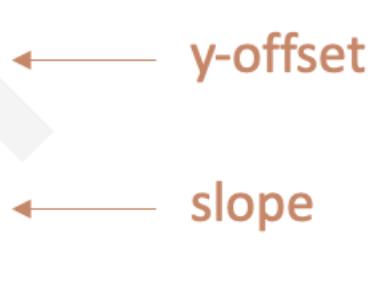
- To test the model, run on a separate **test dataset** of input / output pairs
- See how well it **generalizes** to new data

Example: 1D Linear regression model

- Model:

$$\begin{aligned}y &= f[x, \phi] \\&= \phi_0 + \phi_1 x\end{aligned}$$

- Parameters

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


y-offset
slope

Example: 1D Linear regression model

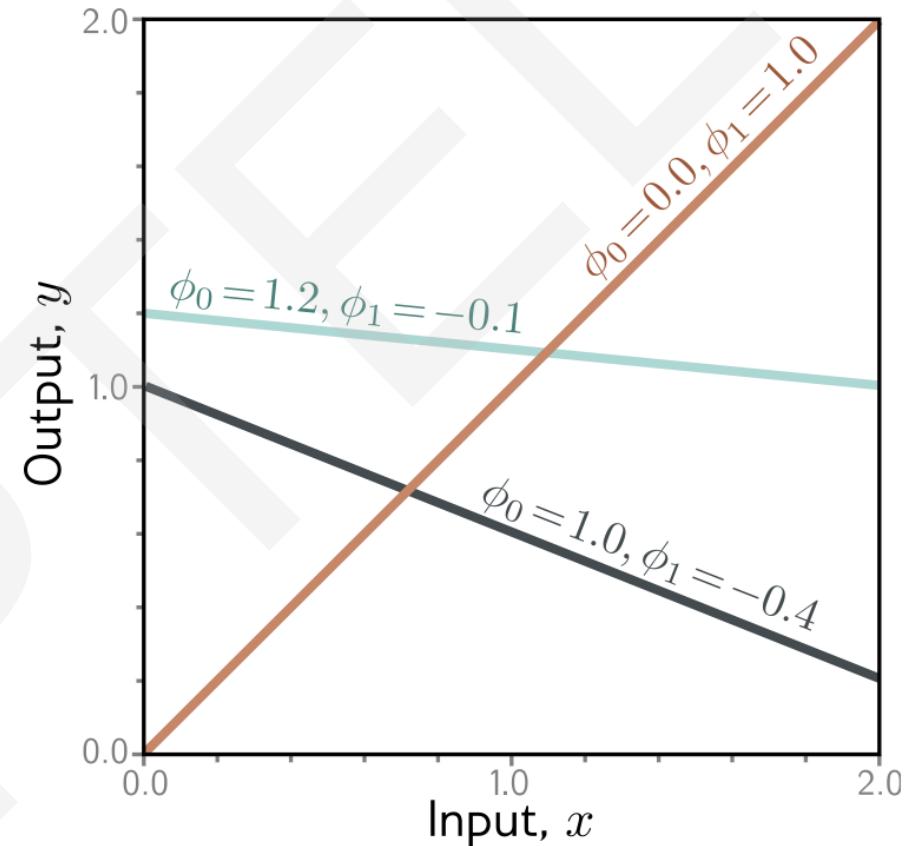
- Model:

$$y = f[x, \phi] \\ = \phi_0 + \phi_1 x$$

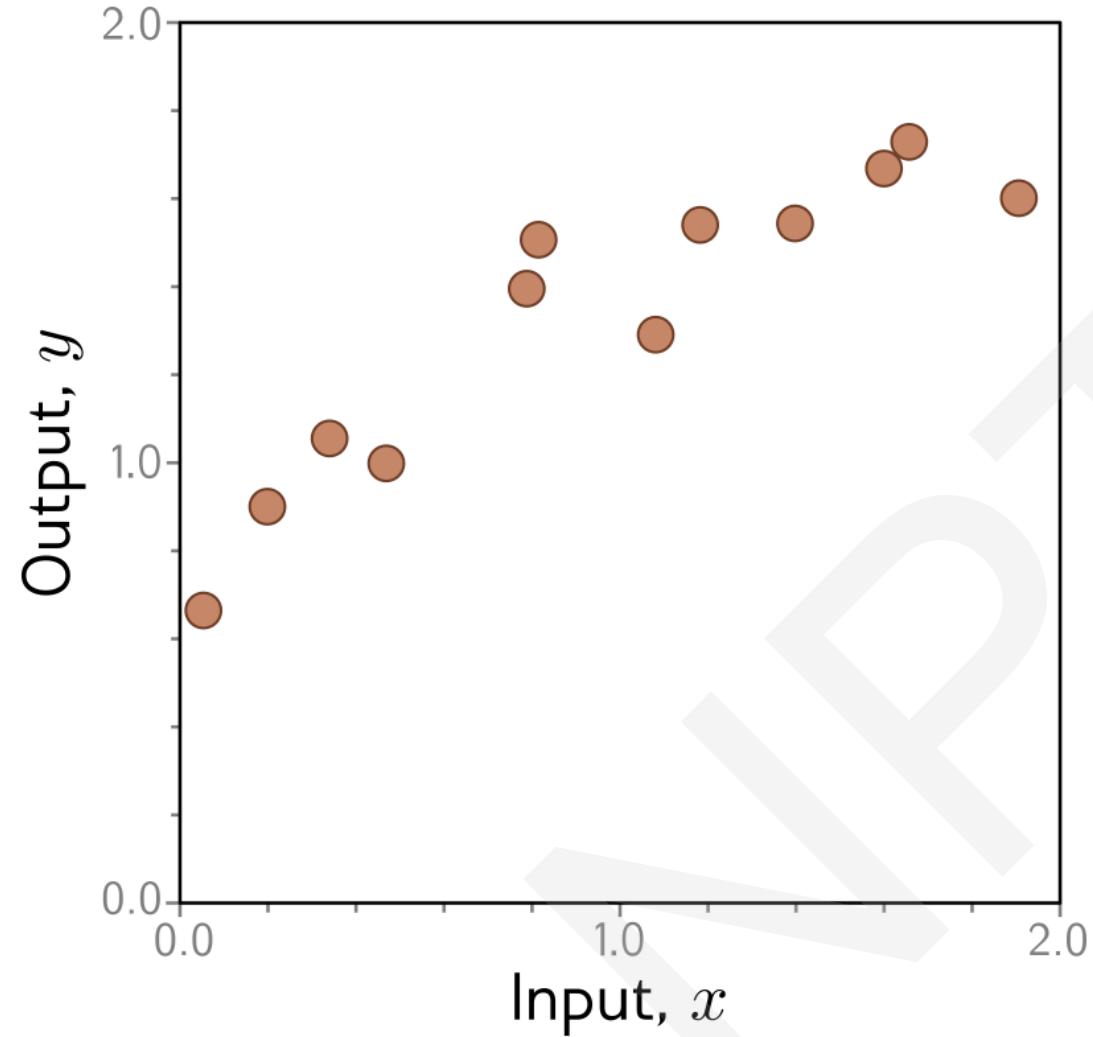
- Parameters

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

← y-offset
← slope



Example: 1D Linear regression training data

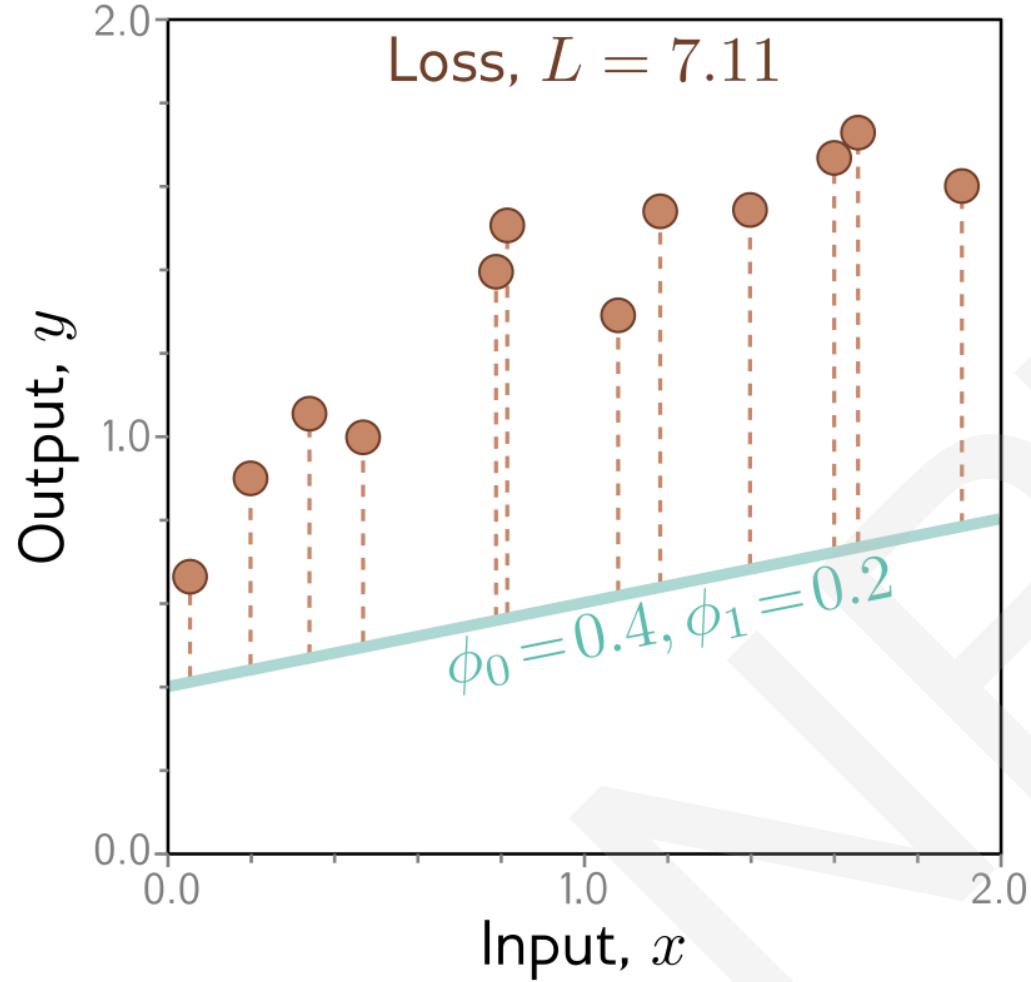


Loss function:

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

“Least squares loss function”

Example: 1D Linear regression training data

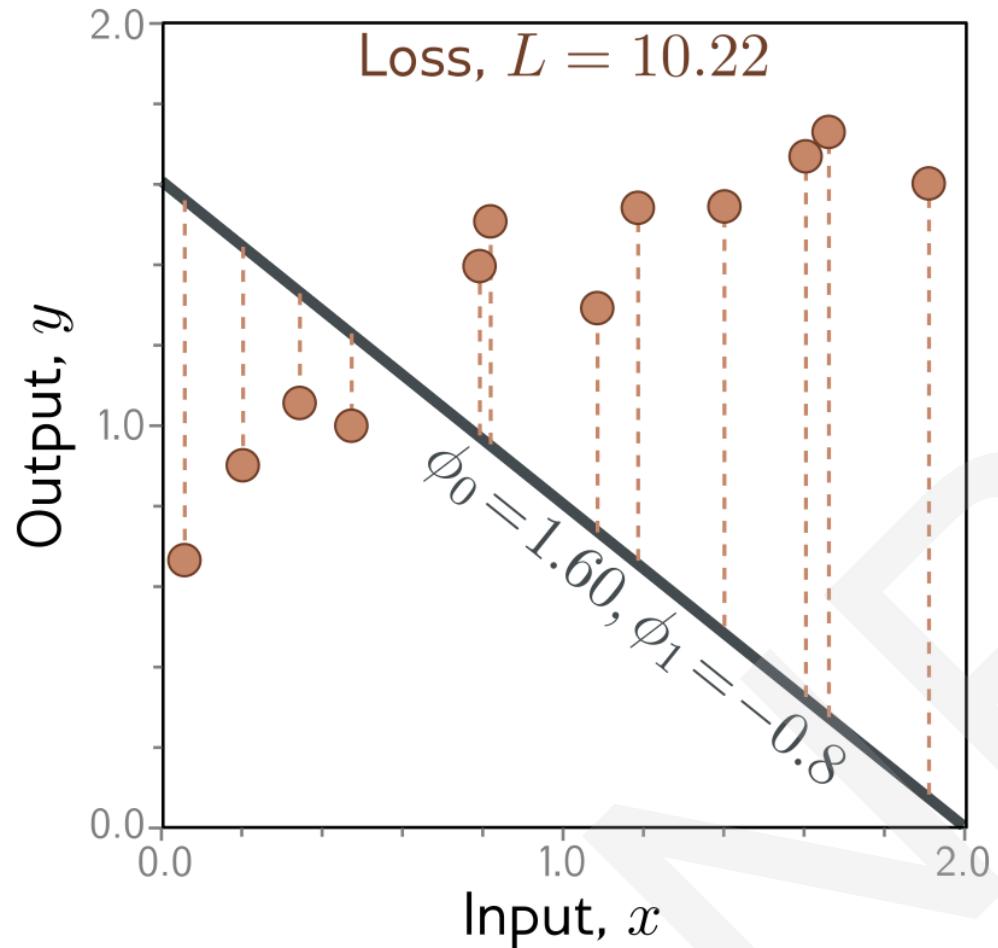


Loss function:

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

“Least squares loss function”

Example: 1D Linear regression training data



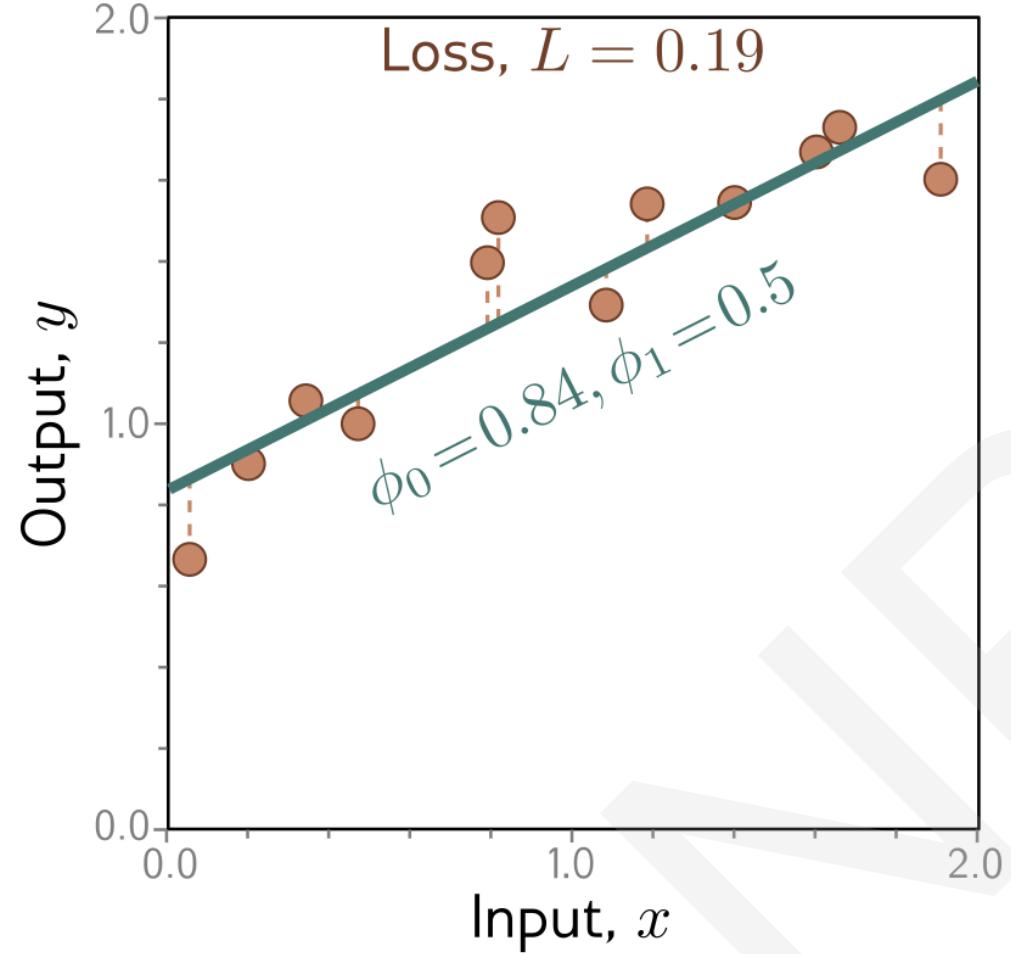
Loss function:

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

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

"Least squares loss function"

Example: 1D Linear regression training data



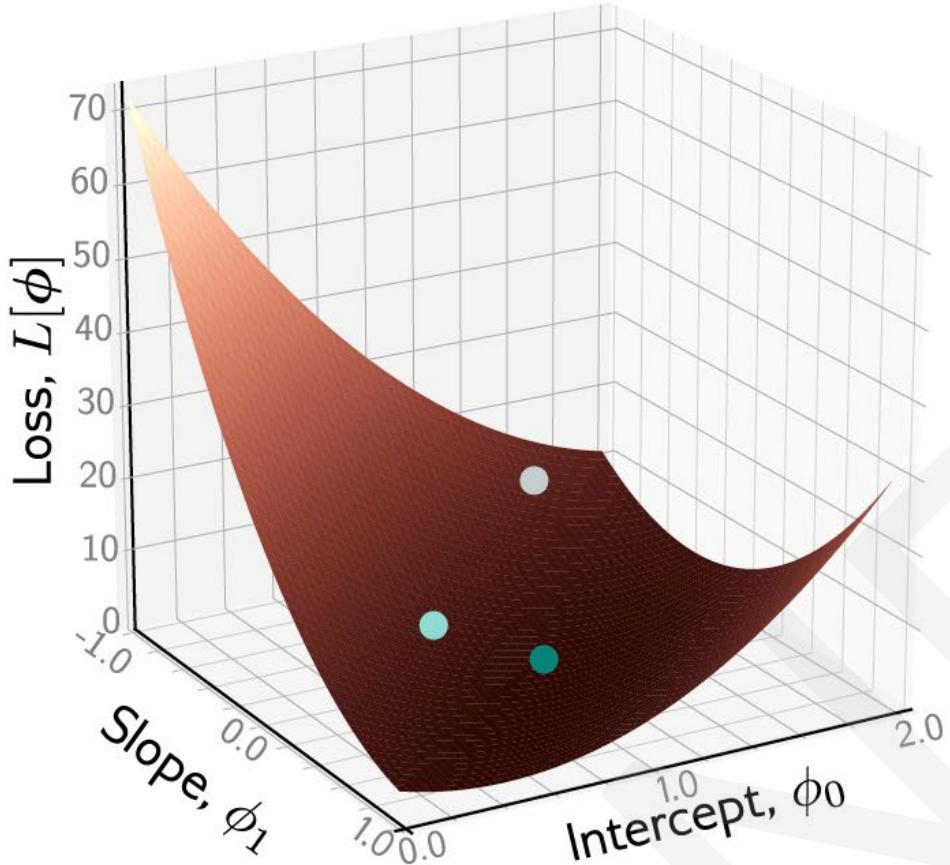
Loss function:

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

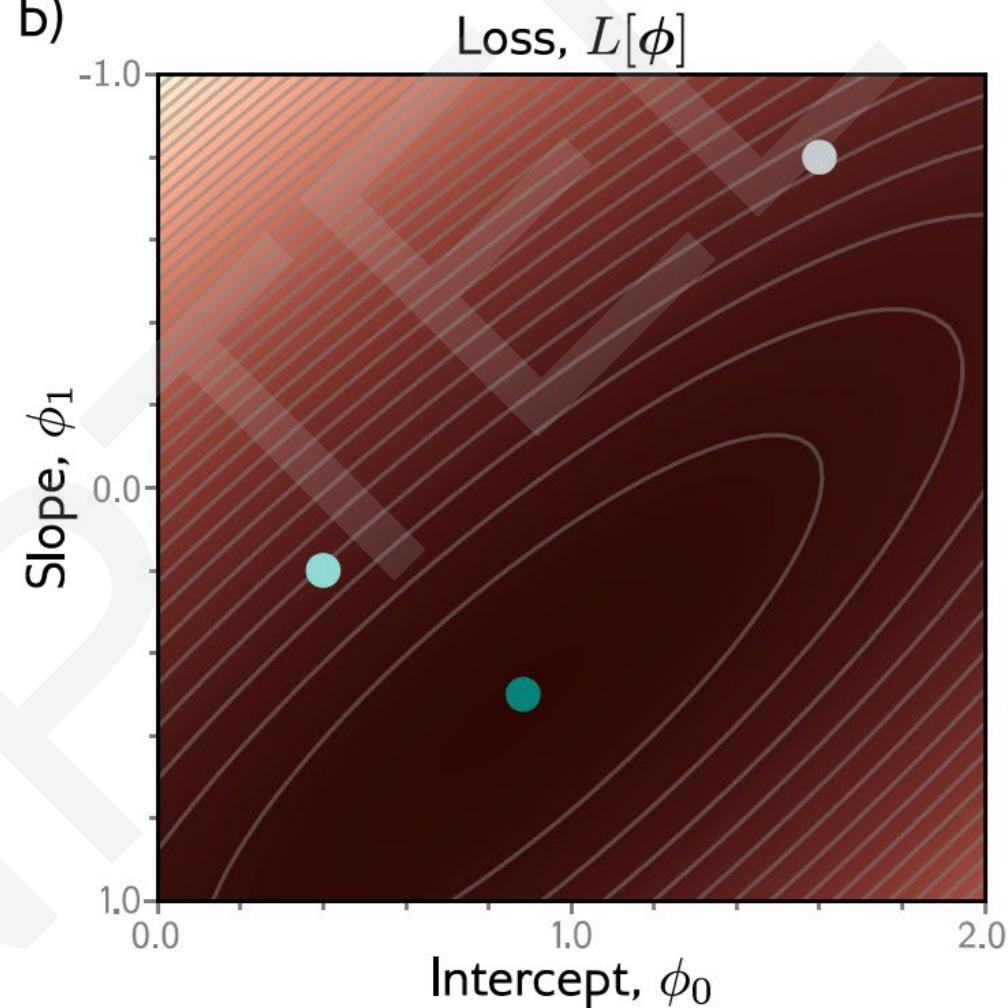
“Least squares loss function”

Example: 1D Linear regression loss function

a)

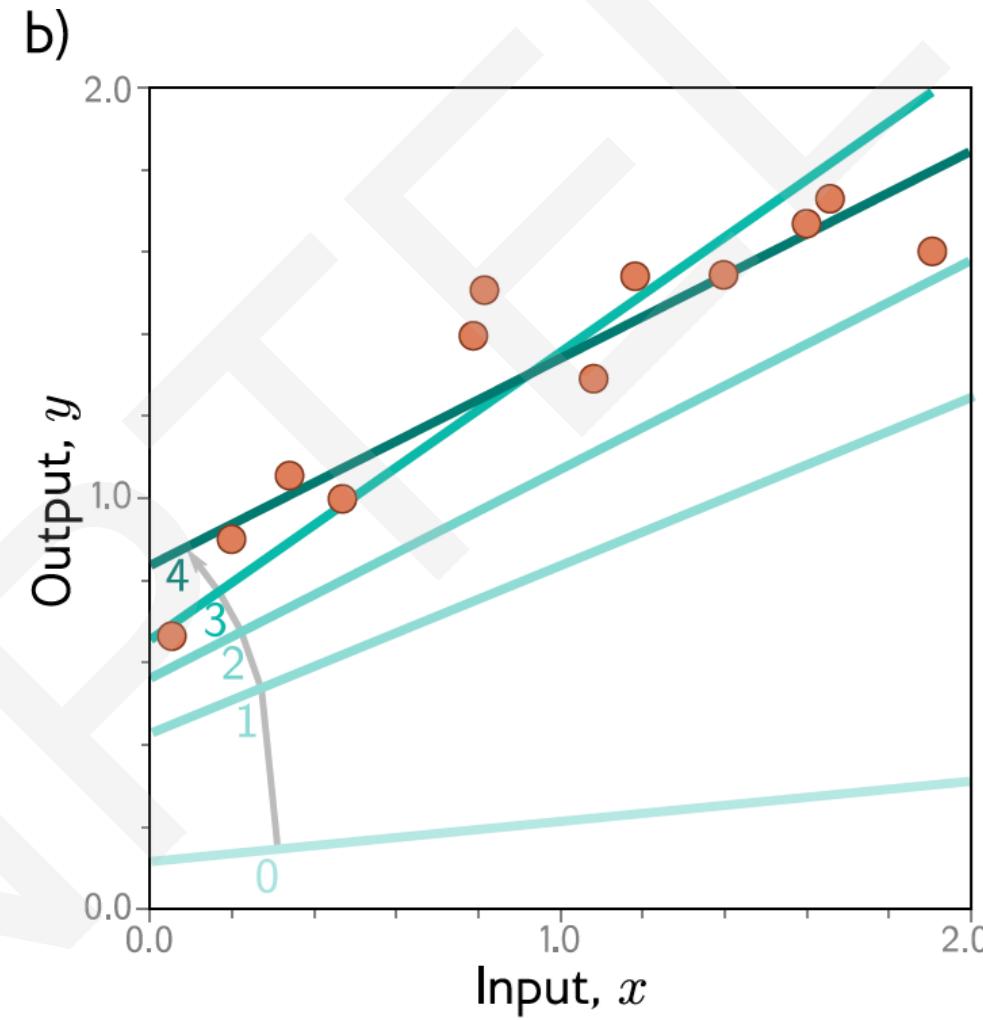
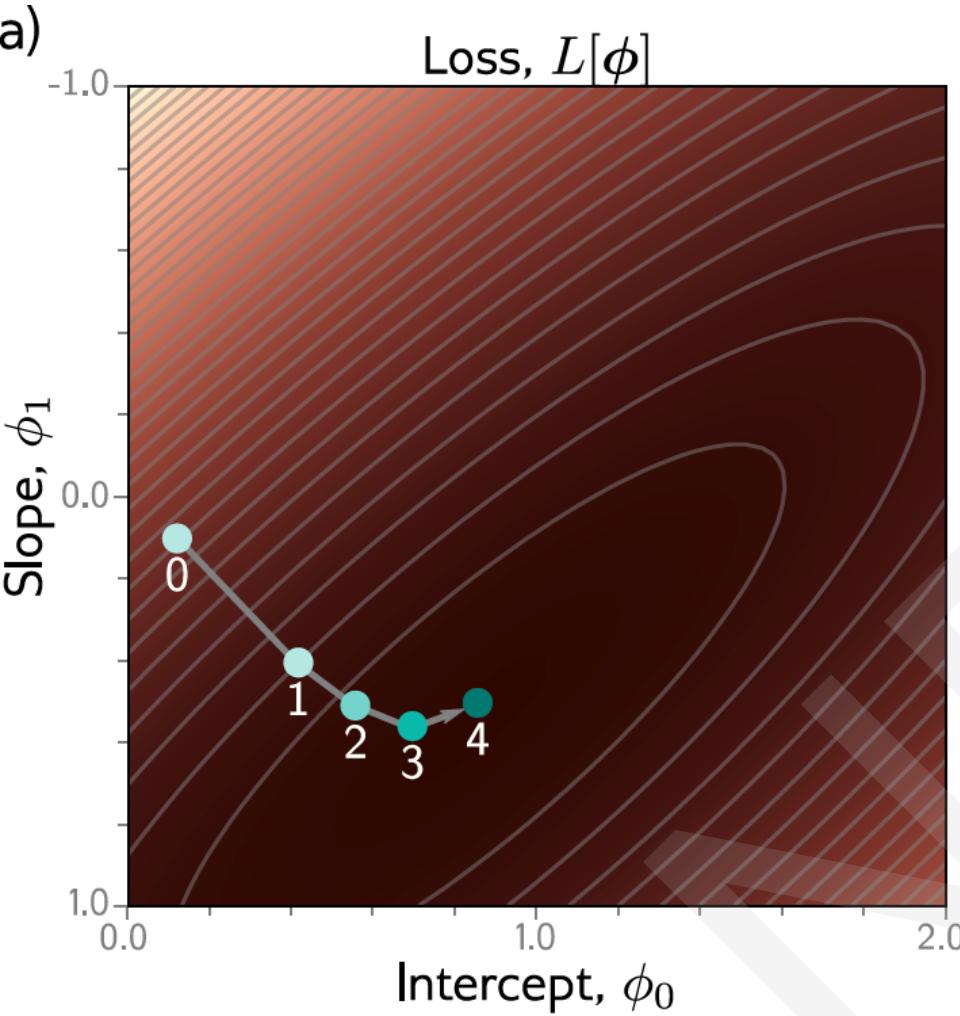


b)



*Slides adapted from <http://udlbook.com>

Example: 1D Linear regression training



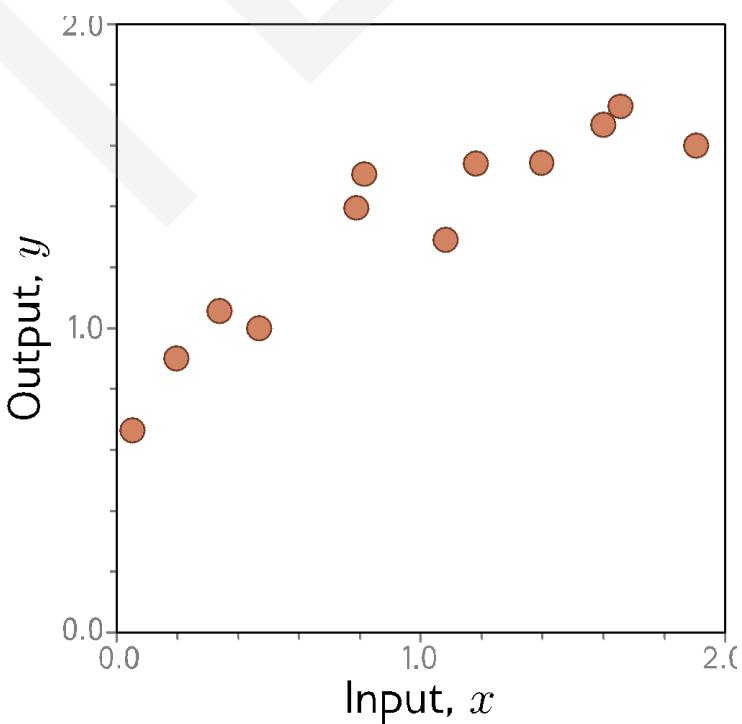
This technique is known as
gradient descent

Possible objections

- But you can fit the line model in closed form!
 - Yes – but we won't be able to do this for more complex models
- But we could exhaustively try every slope and intercept combo!
 - Yes – but we won't be able to do this when there are a million parameters

1D Linear regression testing

- Test with different set of paired input/output data
 - Measure performance
 - Degree to which this is same as training = **generalization**
- Might not generalize well because
 - Model too simple
 - Model too complex
 - fits to statistical peculiarities of data
 - this is known as **overfitting**



*Slides adapted from <http://udlbook.com>

REFERENCES

Understanding Deep Learning [<https://udlbook.github.io/udlbook/>]
[Chapter 1, 2]



THANK YOU



N P T E L O N L I N E C E R T I F I C A T I O N C O U R S E S

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 7 : Shallow Neural Networks



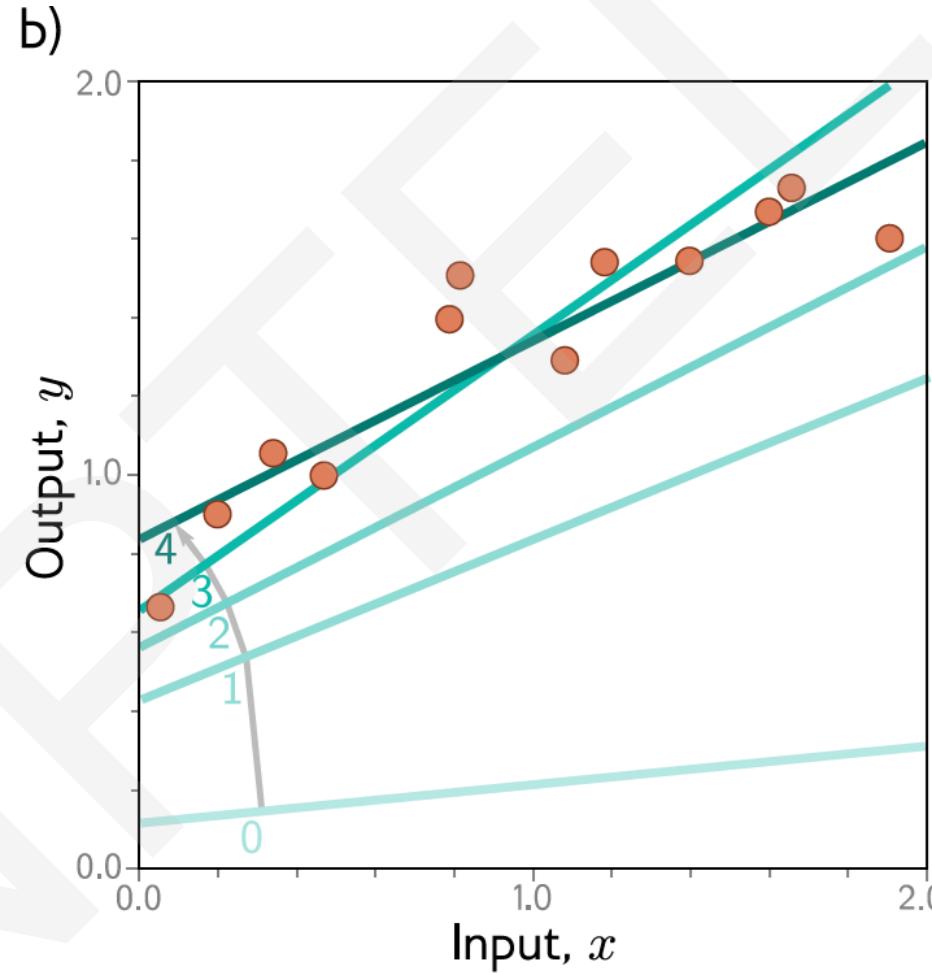
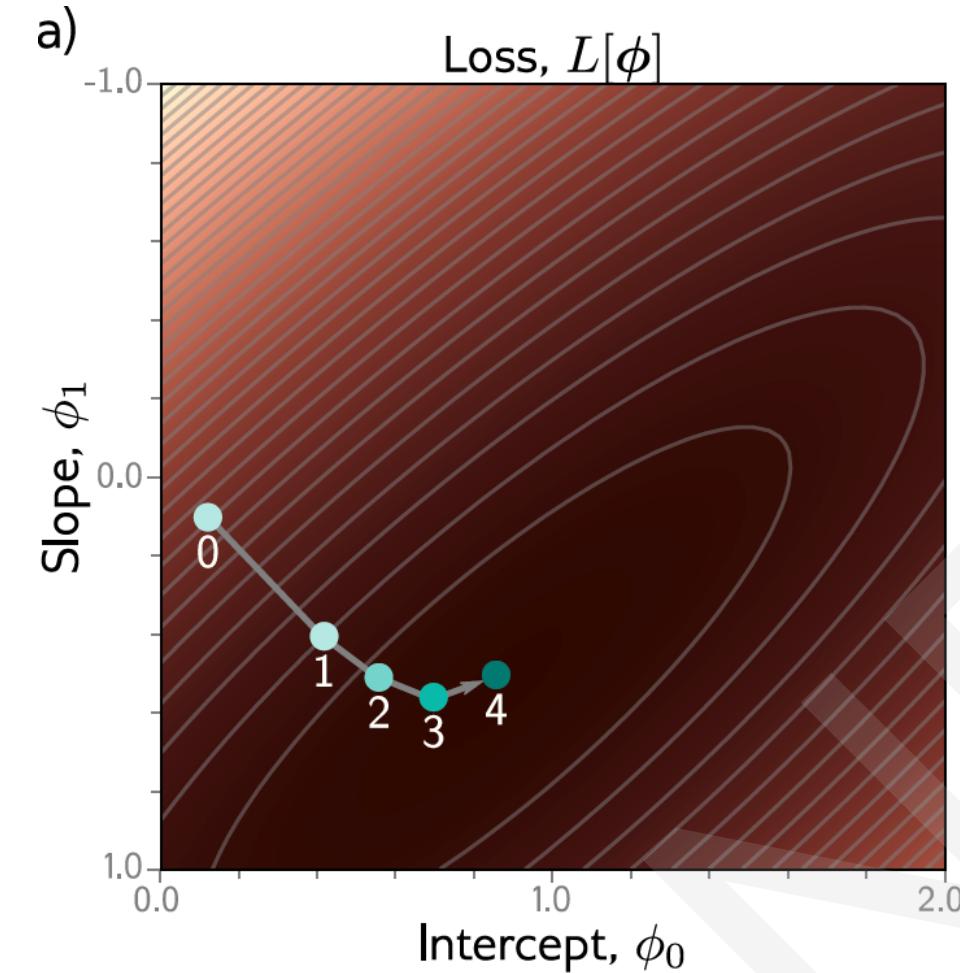
PROF . PAWAN GOYAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Shallow Neural Networks
- Universal Approximation Theorem
- Nomenclature

Recap: 1D Linear regression training



This technique is known as **gradient descent**

*Slides adapted from <http://udlbook.com>

Shallow neural networks

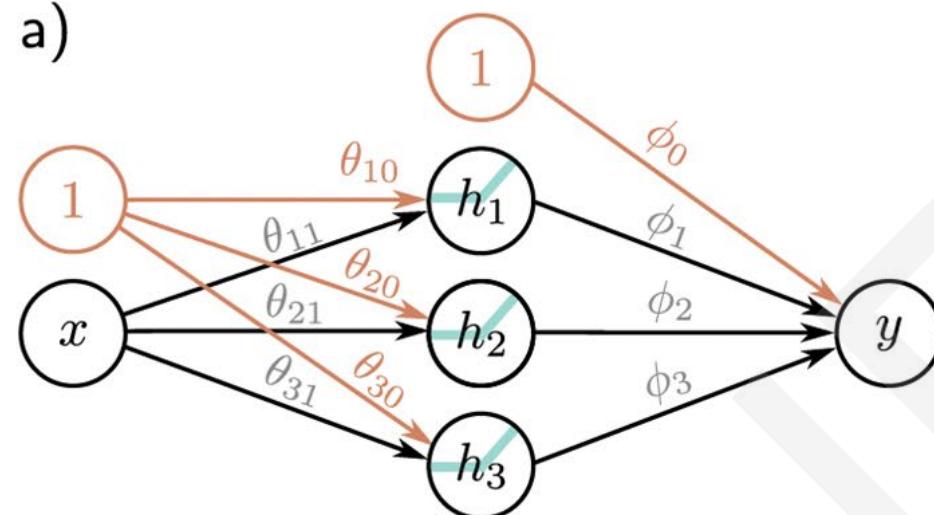
- 1D regression model is obviously limited
 - Want to be able to describe input/output that are not lines
 - Want multiple inputs
 - Want multiple outputs
- Shallow neural networks
 - Flexible enough to describe arbitrarily complex input/output mappings
 - Can have as many inputs as we want
 - Can have as many outputs as we want

Example shallow network

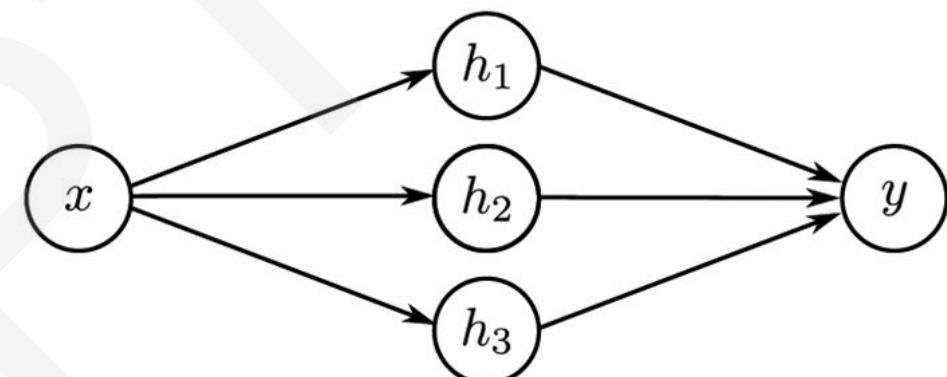
$$y = f[x, \phi]$$

$$= \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] + \phi_2 a[\theta_{20} + \theta_{21}x] + \phi_3 a[\theta_{30} + \theta_{31}x]$$

a)



b)



Example shallow network

$$y = f[x, \phi]$$

$$= \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] + \phi_2 a[\theta_{20} + \theta_{21}x] + \phi_3 a[\theta_{30} + \theta_{31}x]$$

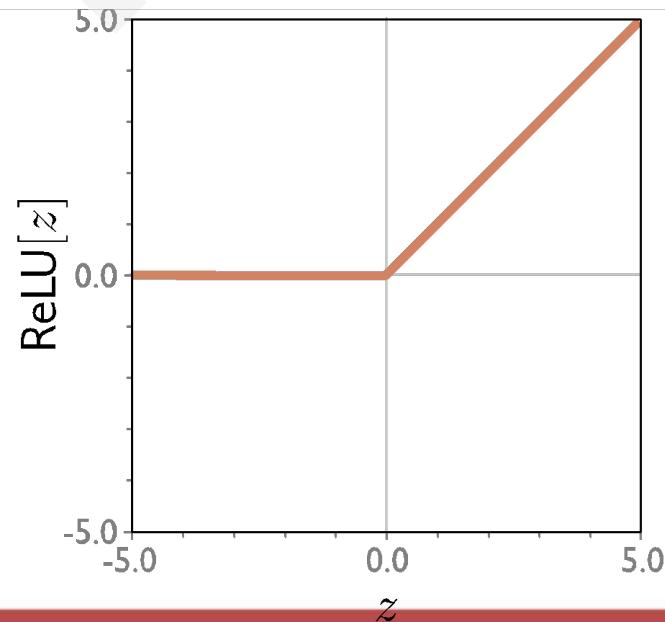
Activation function

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

Rectified Linear Unit

(particular kind of activation function)

*Slides adapted from <http://udlbook.com>



Example shallow network

$$y = f[x, \phi]$$

$$= \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] + \phi_2 a[\theta_{20} + \theta_{21}x] + \phi_3 a[\theta_{30} + \theta_{31}x]$$

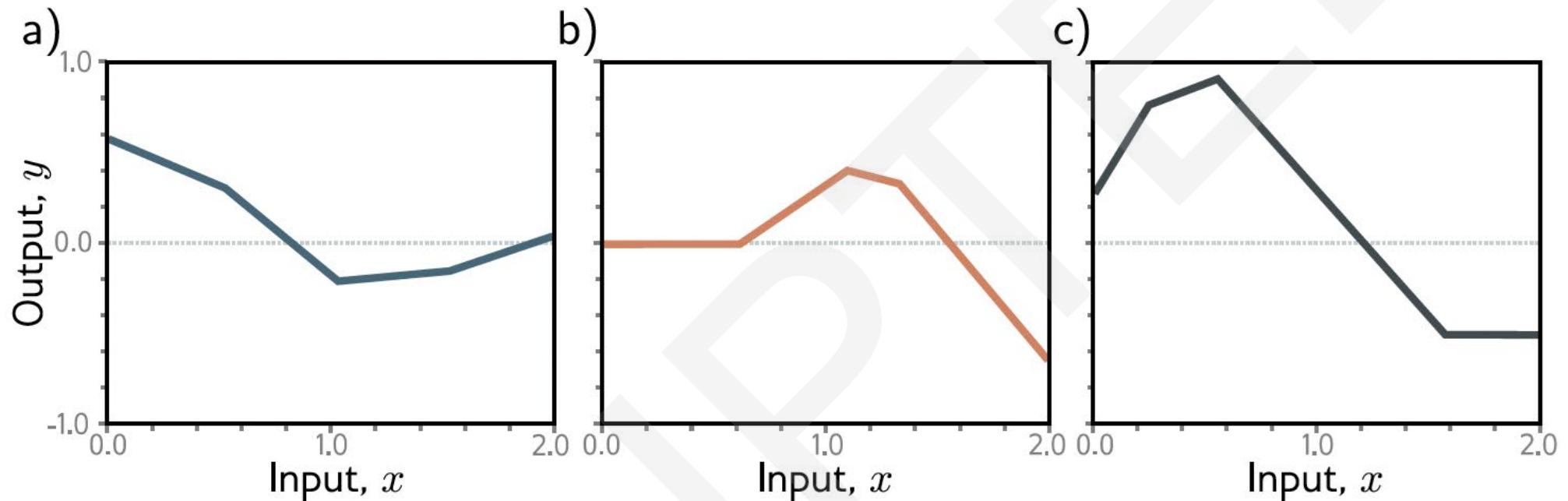
This model has 10 parameters:

$$\phi = \{\phi_0, \phi_1, \phi_2, \phi_3, \theta_{10}, \theta_{11}, \theta_{20}, \theta_{21}, \theta_{30}, \theta_{31}\}$$

- Represents a family of functions
- Parameters determine particular function
- Given parameters can perform inference (run equation)
- Given training dataset $\{x_i, y_i\}_{i=1}^I$
- Define loss function $L[\phi]$ (least squares)
- Change parameters to minimize loss function

Example shallow network

$$y = \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] + \phi_2 a[\theta_{20} + \theta_{21}x] + \phi_3 a[\theta_{30} + \theta_{31}x].$$



Family of functions defined by the equation for
three choices of parameters

Hidden units

$$y = \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] + \phi_2 a[\theta_{20} + \theta_{21}x] + \phi_3 a[\theta_{30} + \theta_{31}x].$$

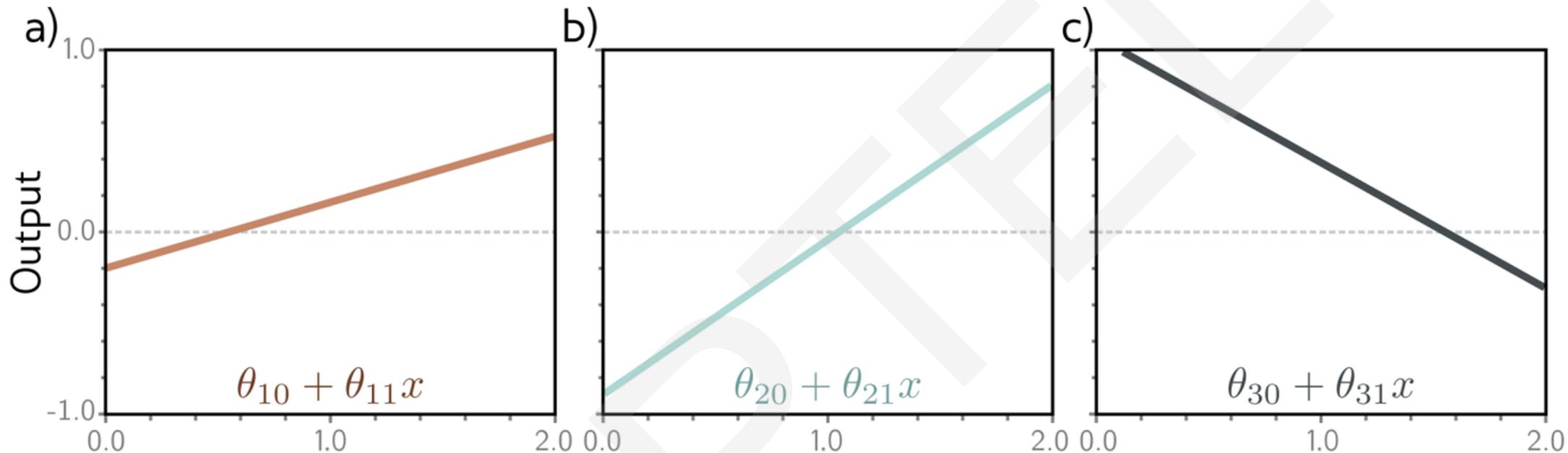
Break down into two parts:

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

where:

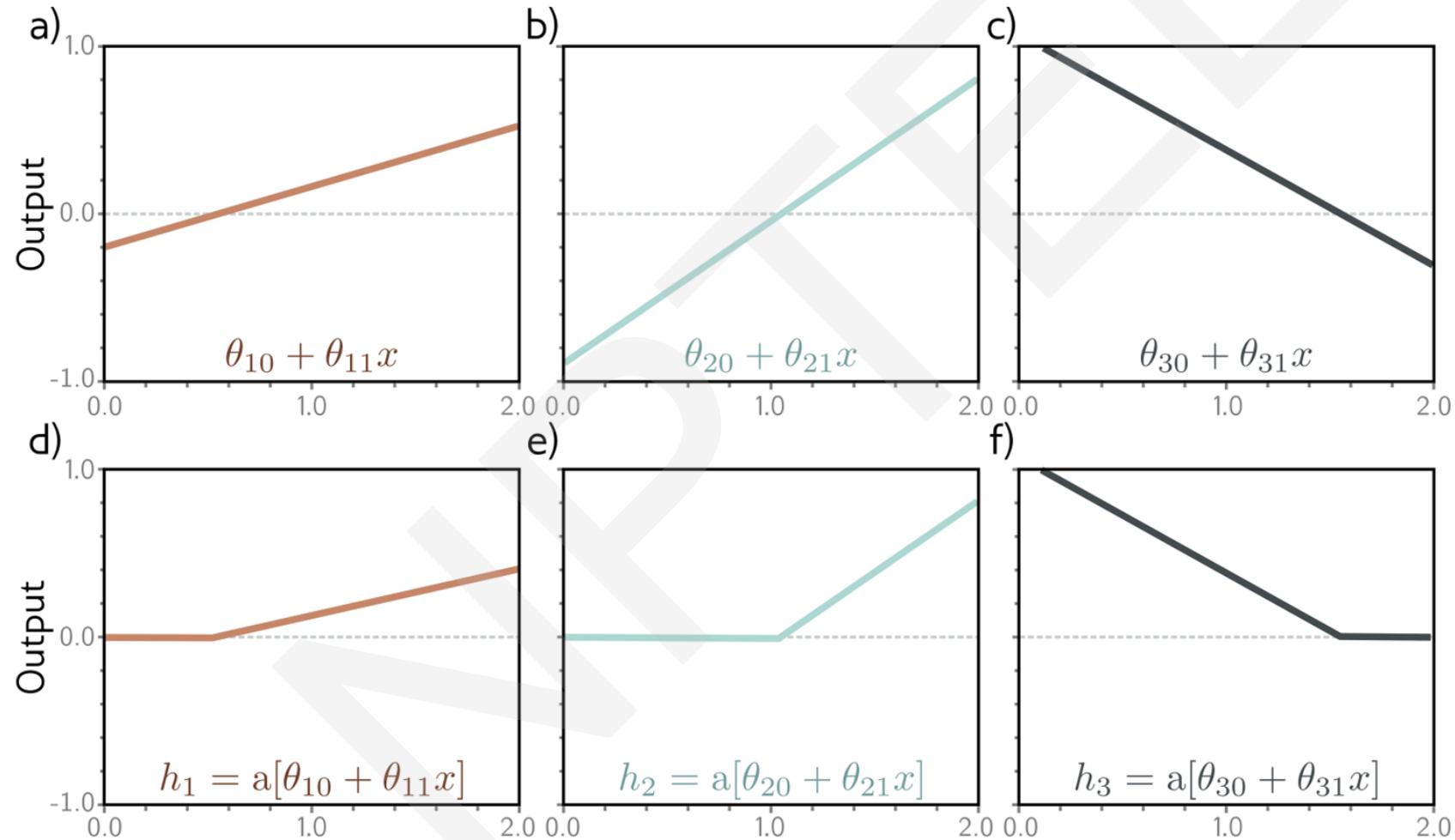
Hidden units $\left[\begin{array}{l} h_1 = a[\theta_{10} + \theta_{11}x] \\ h_2 = a[\theta_{20} + \theta_{21}x] \\ h_3 = a[\theta_{30} + \theta_{31}x] \end{array} \right]$

1. compute three linear functions



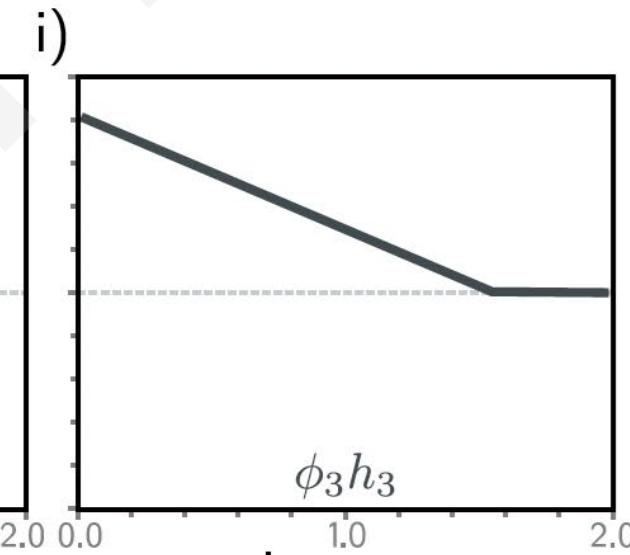
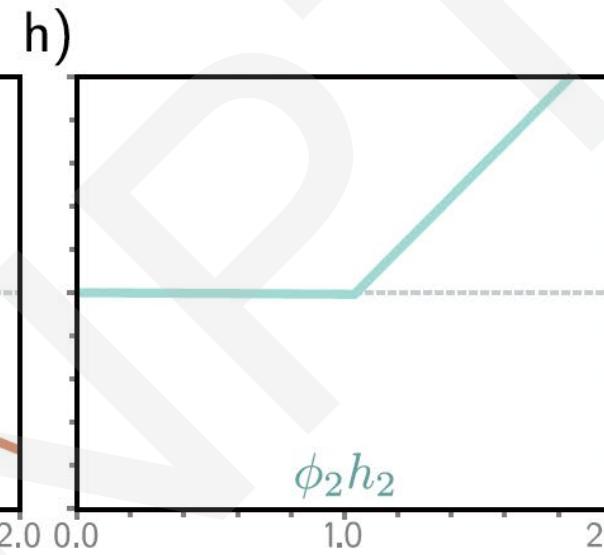
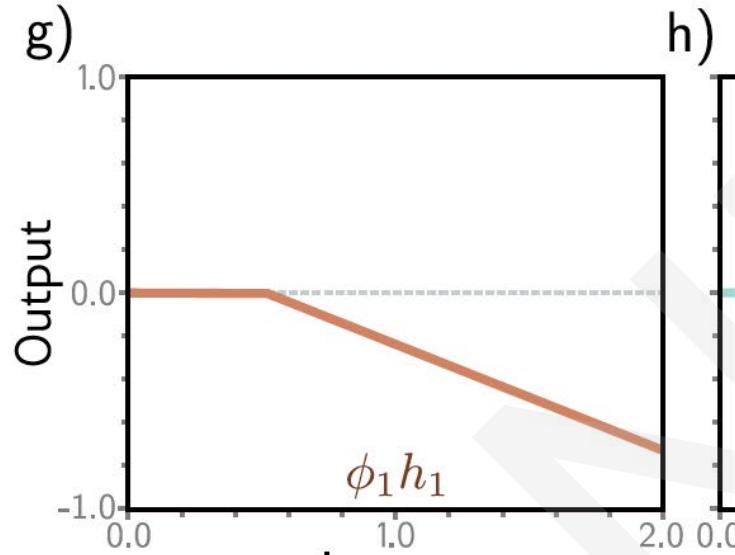
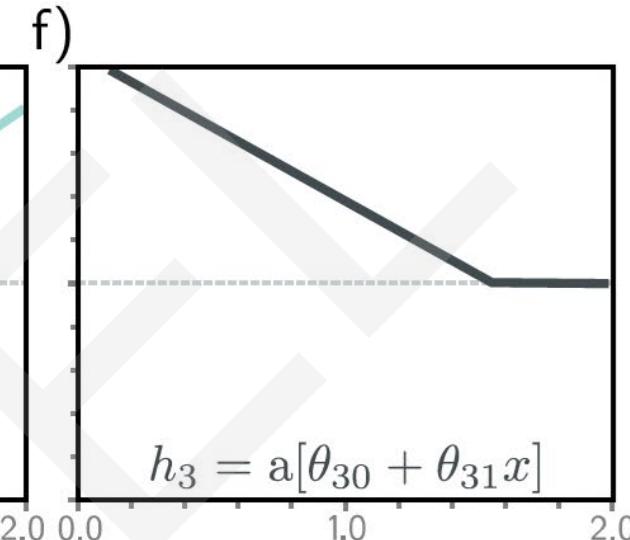
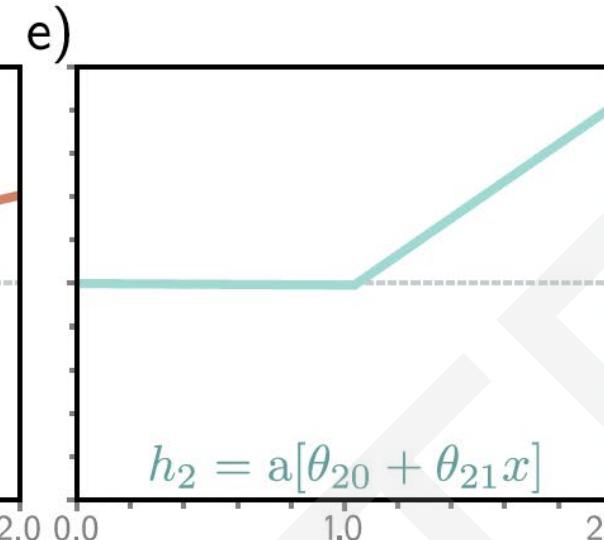
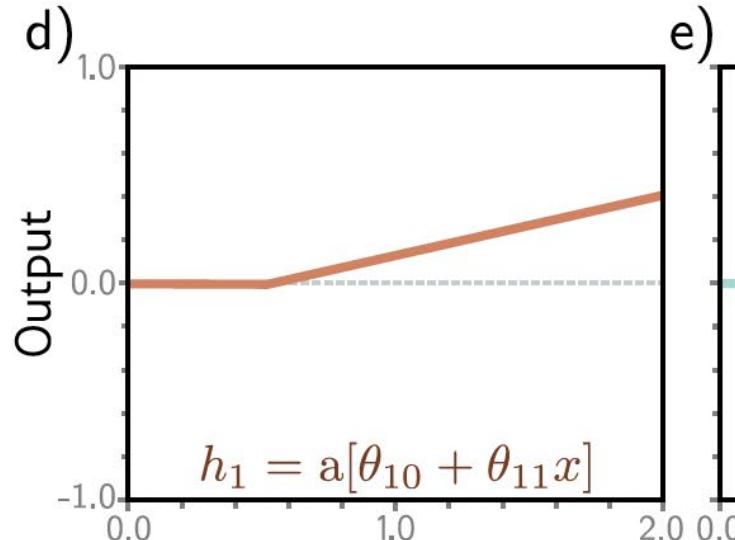
2. Pass through ReLU functions (creates hidden units)

$$h_1 = a[\theta_{10} + \theta_{11}x]$$
$$h_2 = a[\theta_{20} + \theta_{21}x]$$
$$h_3 = a[\theta_{30} + \theta_{31}x],$$



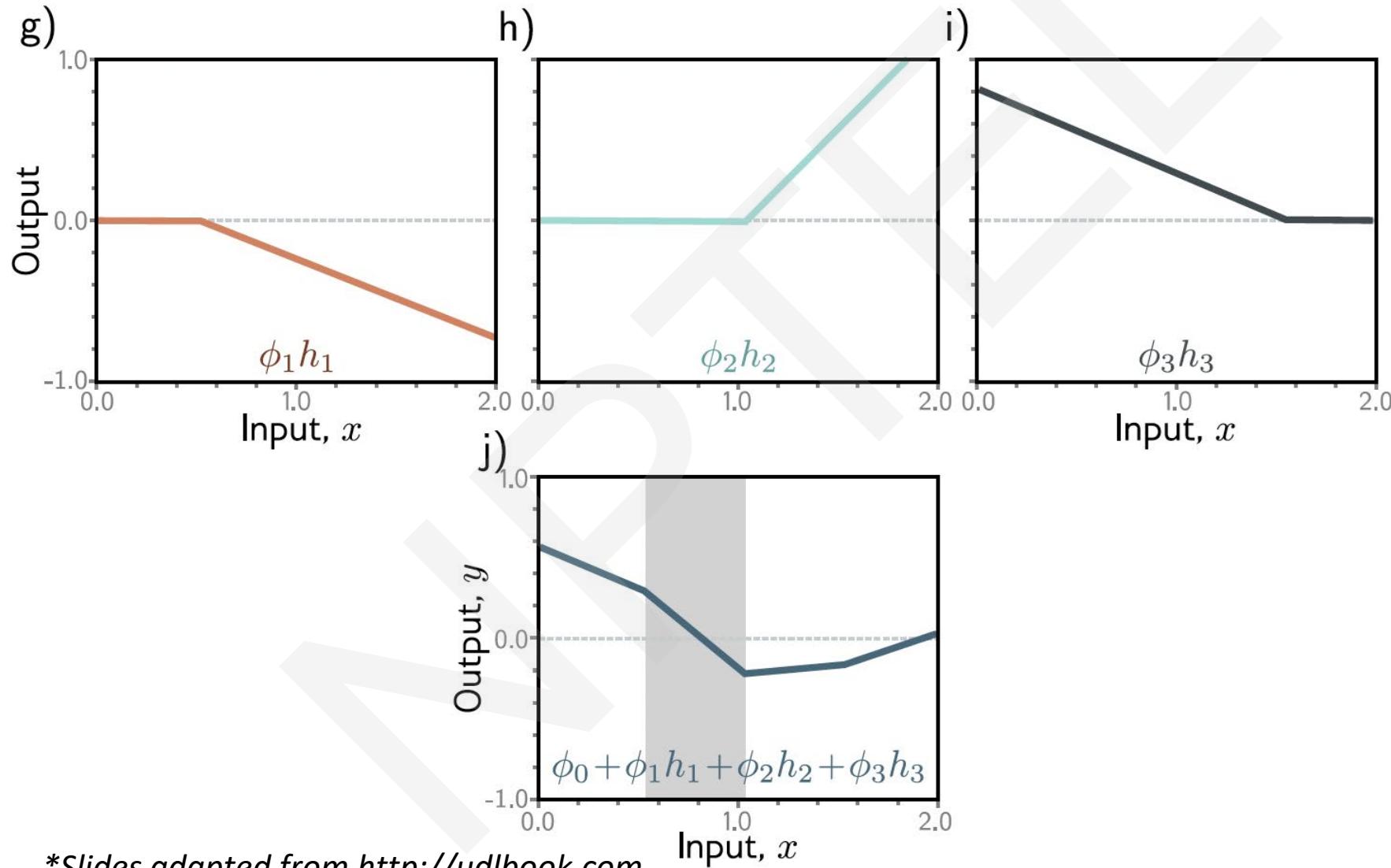
*Slides adapted from <http://udlbook.com>

2. Weight the hidden units



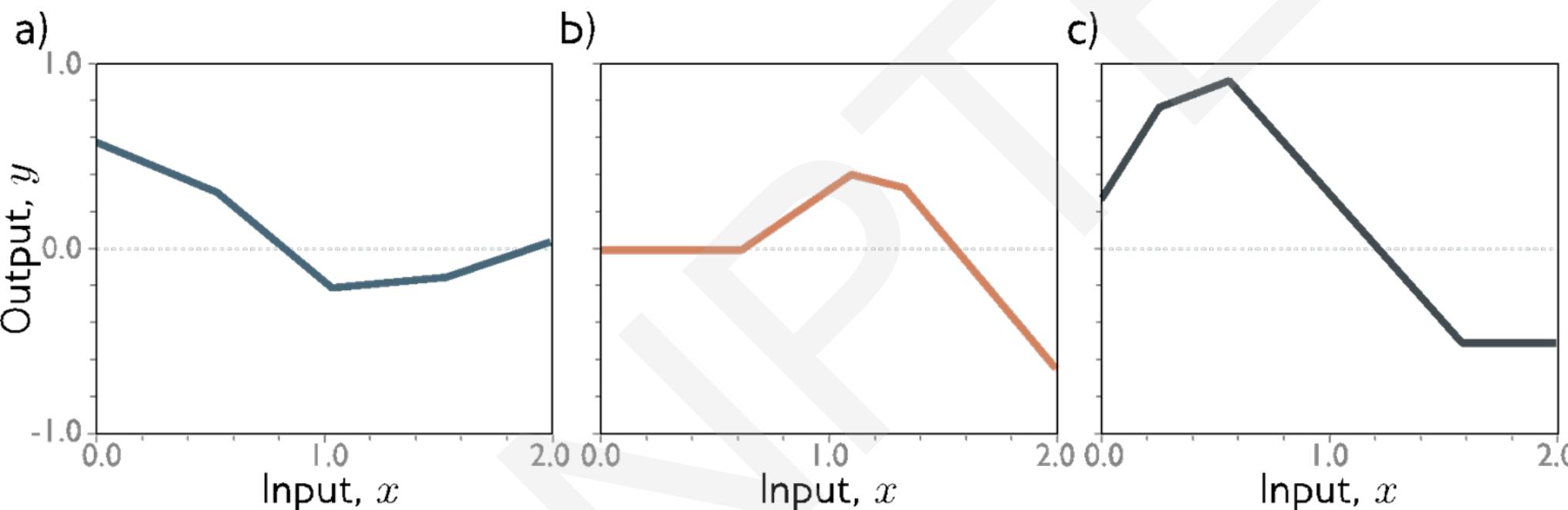
4. Sum the weighted hidden units to create output

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$



Example shallow network

$$y = \phi_0 + \phi_1 a[\theta_{10} + \theta_{11}x] + \phi_2 a[\theta_{20} + \theta_{21}x] + \phi_3 a[\theta_{30} + \theta_{31}x].$$



Example shallow network = piecewise linear functions
1 “joint” per ReLU function

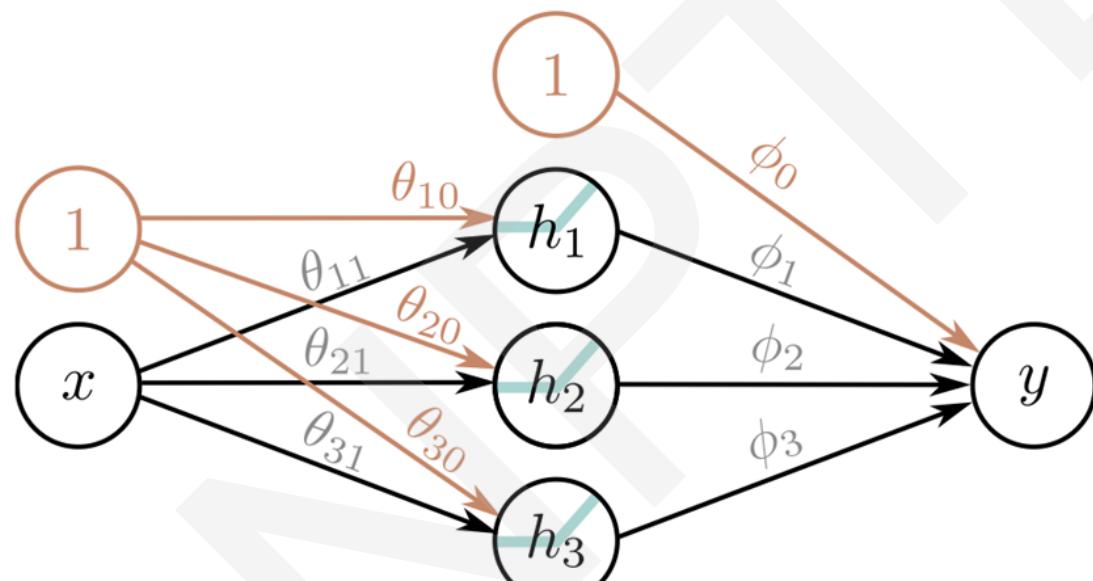
Depicting neural network

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

$$h_2 = a[\theta_{20} + \theta_{21}x]$$

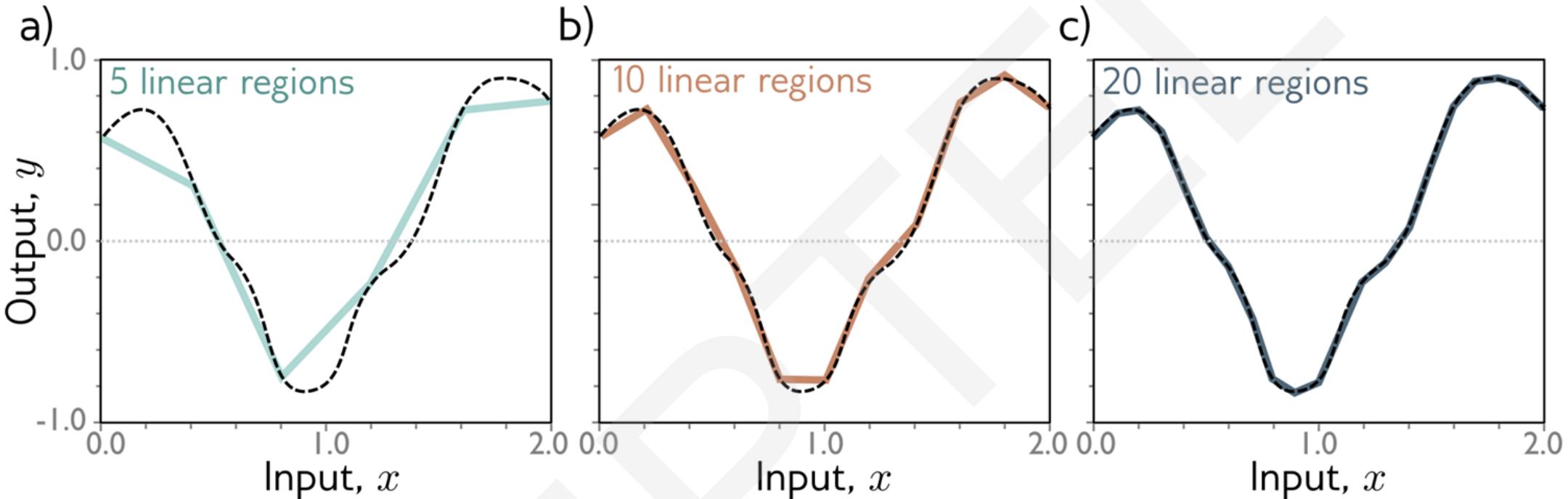
$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$



Each parameter multiplies its source and adds to its target

With enough hidden units...



... we can describe any 1D function to arbitrary accuracy

Universal approximation theorem

“a formal proof that, with enough hidden units, a shallow neural network can describe any continuous function on a compact subset of \mathbb{R}^D to arbitrary precision”

More than 1 outputs?

- 1 input, 4 hidden units, 2 outputs

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

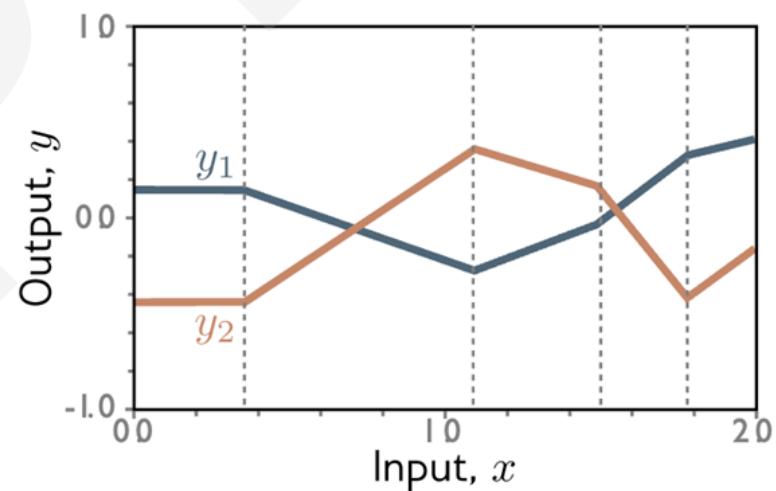
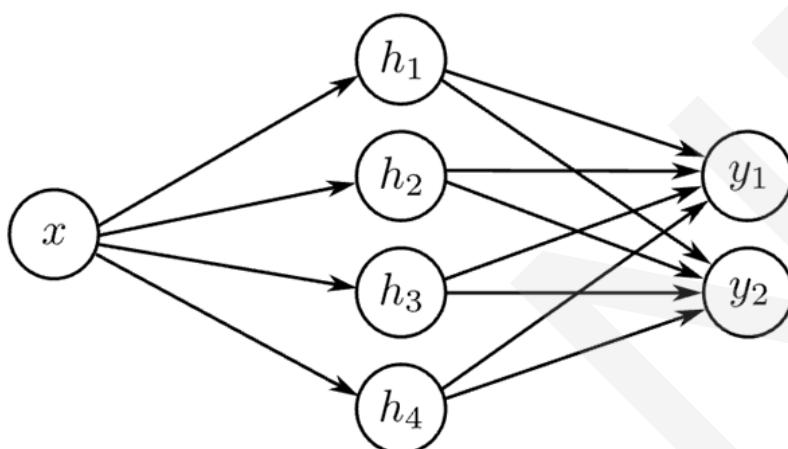
$$h_2 = a[\theta_{20} + \theta_{21}x]$$

$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$h_4 = a[\theta_{40} + \theta_{41}x]$$

$$y_1 = \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4$$

$$y_2 = \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4$$



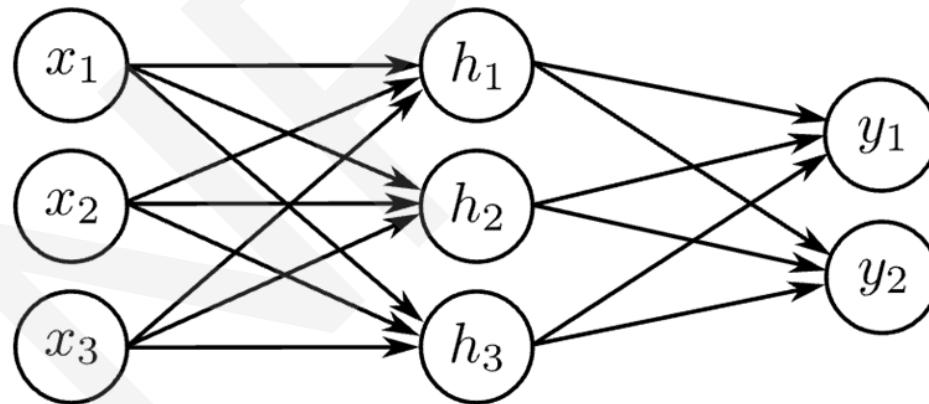
Arbitrary inputs, hidden units, outputs

- D_o Outputs, D hidden units, and D_i inputs

$$h_d = a \left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i \right]$$

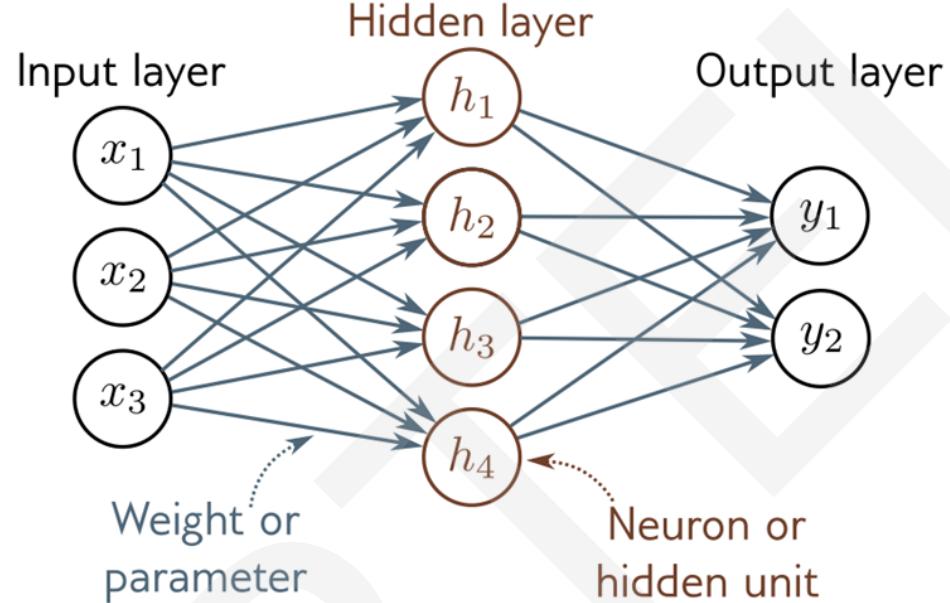
$$y_j = \phi_{j0} + \sum_{d=1}^D \phi_{jd} h_d$$

- e.g., Three inputs, three hidden units, two outputs



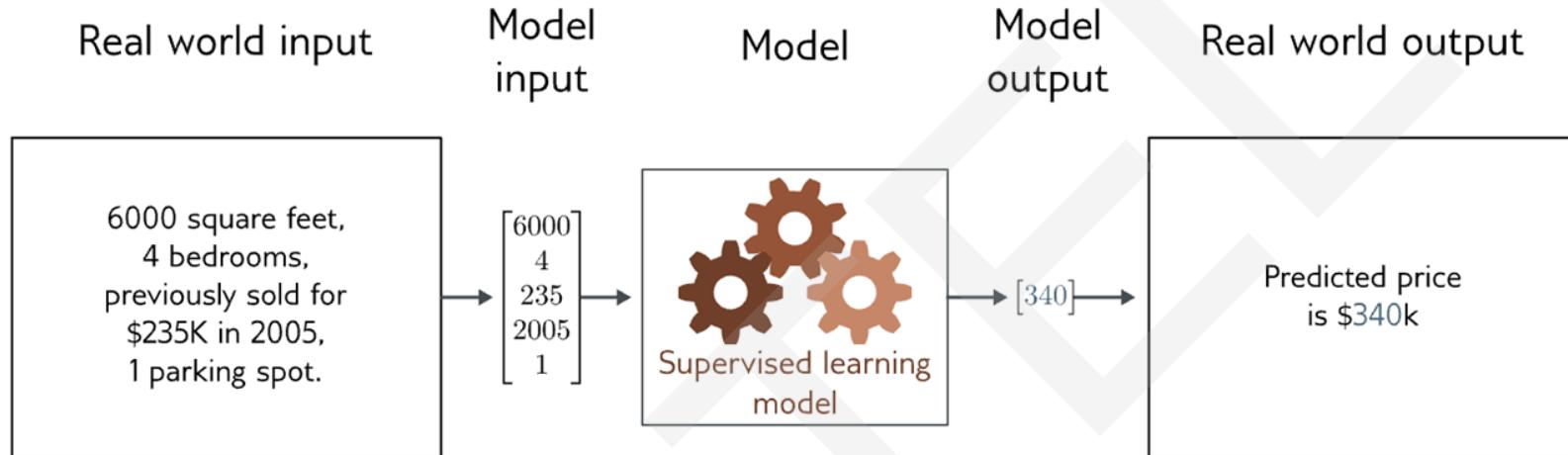
How many parameters does this model have?

Nomenclature



- Y-offsets = **biases**
- Slopes = **weights**
- Everything in one layer connected to everything in the next = **fully connected network**
- No loops = **feedforward network**
- Values after ReLU (activation functions) = **activations**
- Values before ReLU = **pre-activations**
- One hidden layer = **shallow neural network**
- More than one hidden layer = **deep neural network**
- Number of hidden units \approx **capacity**

Regression



We have built a model that can:

- take an arbitrary number of inputs
- output an arbitrary number of outputs
- model a function of arbitrary complexity between the two

$$h_d = \mathbf{a} \left[\theta_{d0} + \sum_{i=1}^{D_i} \theta_{di} x_i \right]$$

$$y_j = \phi_{j0} + \sum_{d=1}^D \phi_{jd} h_d$$

REFERENCES

Understanding Deep Learning [<https://udlbook.github.io/udlbook/>]
[Chapter 3]



THANK YOU



N P T E L O N L I N E C E R T I F I C A T I O N C O U R S E S

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 8 : Deep Neural Networks



PROF . PAWAN GOYAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Deep Neural Networks
- Networks as composing functions
- Shallow vs deep networks

Deep neural networks

- Networks with more than one hidden layer
- With ReLU activation, both shallow and deep neural networks describe piecewise linear mappings from input to output
- As the number of hidden units increases, the shallow NNs improve their descriptive power.
- With enough hidden units, shallow networks can describe arbitrarily complex functions in high dimensions.
- Then why go for Deep neural networks?

Why Deep Neural Networks?

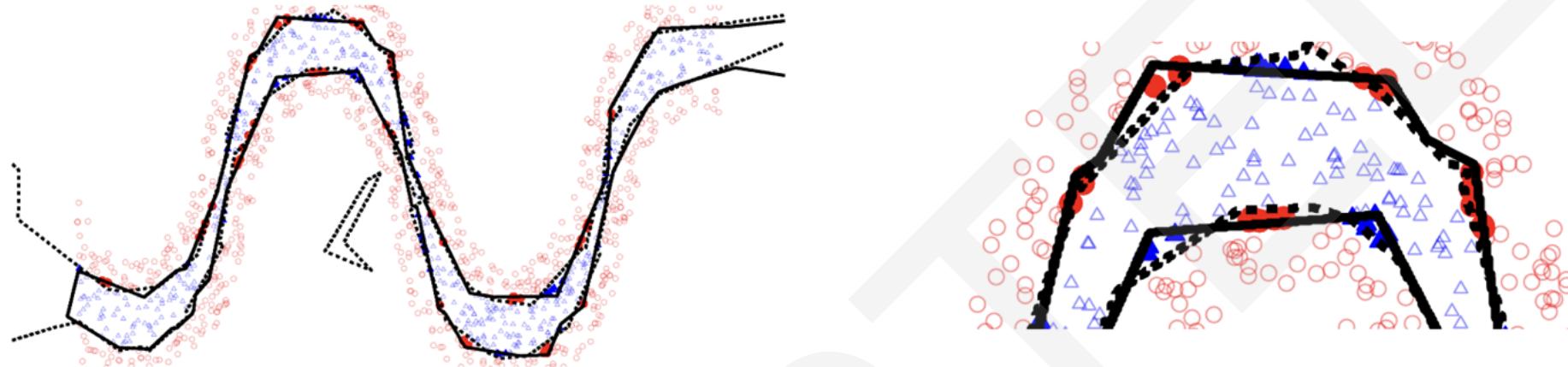


Figure 1: Binary classification using a shallow model with 20 hidden units (solid line) and a deep model with two layers of 10 units each (dashed line). The right panel shows a close-up of the left panel. Filled markers indicate errors made by the shallow model.

Deep networks can produce many more linear regions than shallow networks for the same number of parameters!

Composing two networks

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

Network 1:

$$h_2 = a[\theta_{20} + \theta_{21}x]$$

$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

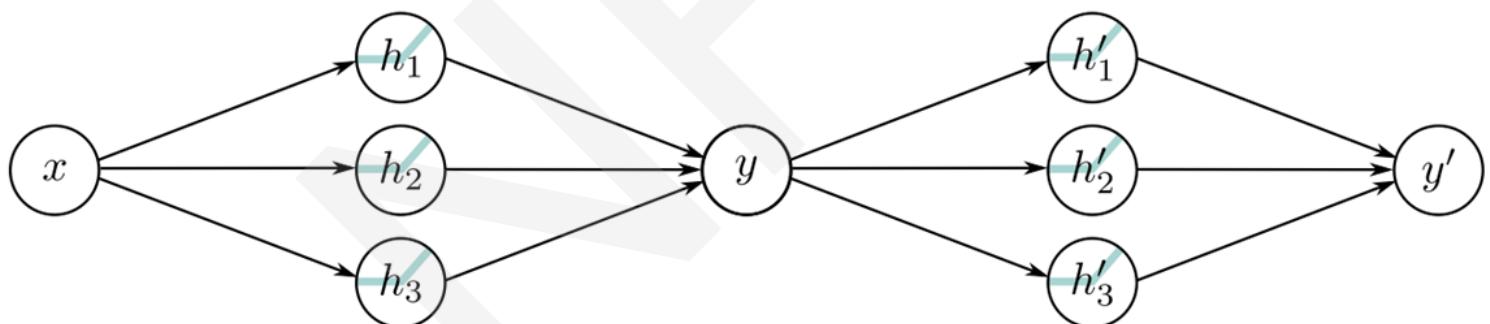
$$h'_1 = a[\theta'_{10} + \theta'_{11}y]$$

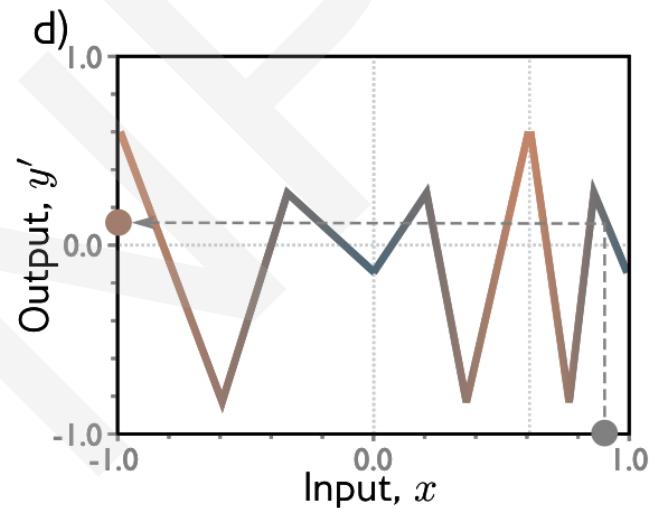
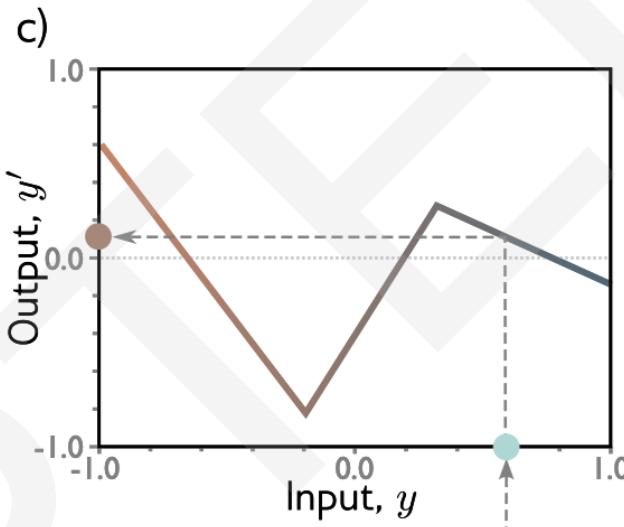
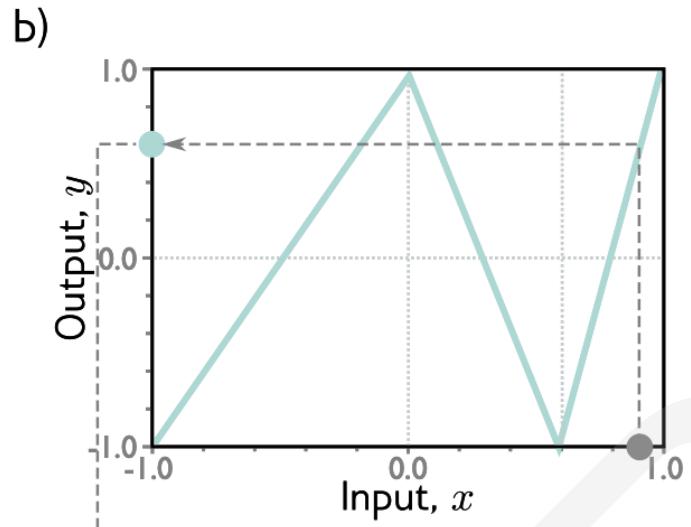
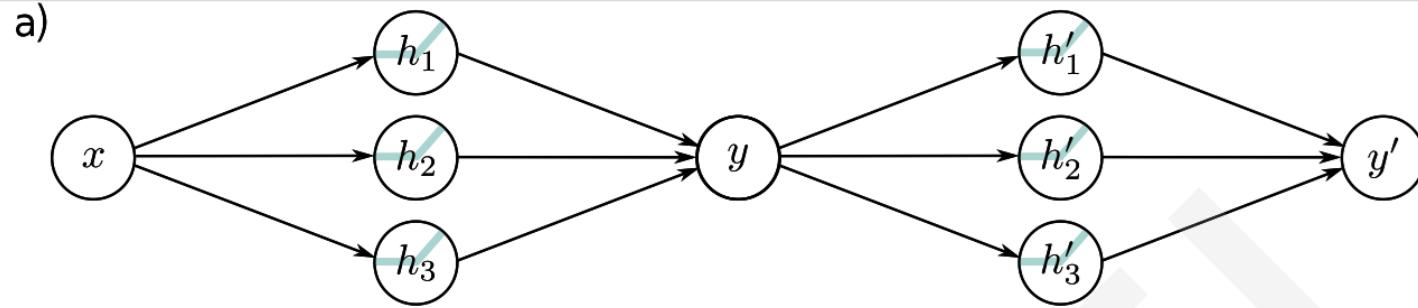
Network 2:

$$h'_2 = a[\theta'_{20} + \theta'_{21}y]$$

$$h'_3 = a[\theta'_{30} + \theta'_{31}y]$$

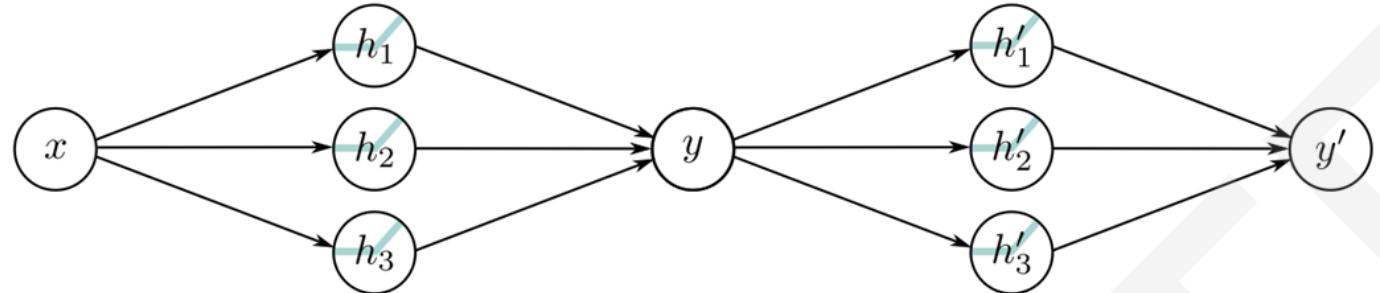
$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$



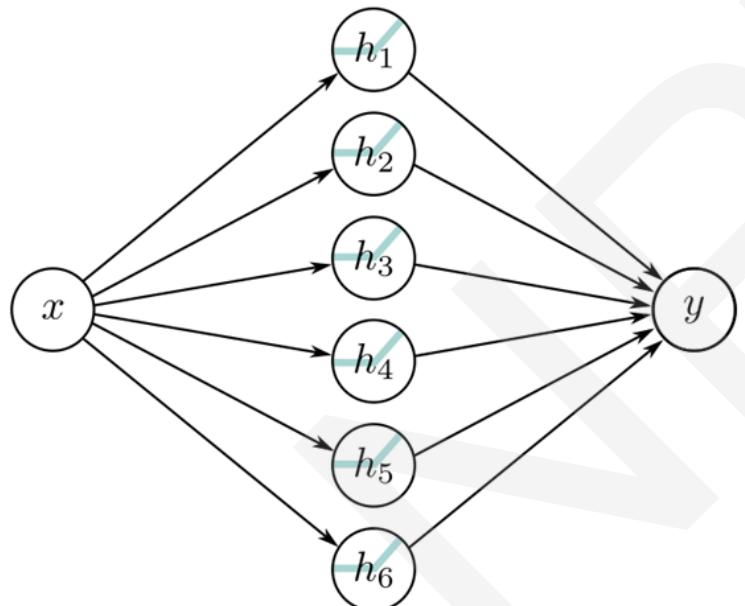


*Slides adapted from <http://udlbook.com>

Comparing to shallow with six hidden units



- 20 parameters
- (at least) 9 regions



- 19 parameters
- Max 7 regions

Combine two networks into one

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

Network 1:

$$h_2 = a[\theta_{20} + \theta_{21}x]$$

$$y = \phi_0 + \phi_1 h_1 + \phi_2 h_2 + \phi_3 h_3$$

$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$h'_1 = a[\theta'_{10} + \theta'_{11}y]$$

Network 2:

$$h'_2 = a[\theta'_{20} + \theta'_{21}y]$$

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$

$$h'_3 = a[\theta'_{30} + \theta'_{31}y]$$

Hidden units of second network in terms of first:

$$h'_1 = a[\theta'_{10} + \theta'_{11}y] = a[\theta'_{10} + \theta'_{11}\phi_0 + \theta'_{11}\phi_1 h_1 + \theta'_{11}\phi_2 h_2 + \theta'_{11}\phi_3 h_3]$$

$$h'_2 = a[\theta'_{20} + \theta'_{21}y] = a[\theta'_{20} + \theta'_{21}\phi_0 + \theta'_{21}\phi_1 h_1 + \theta'_{21}\phi_2 h_2 + \theta'_{21}\phi_3 h_3]$$

$$h'_3 = a[\theta'_{30} + \theta'_{31}y] = a[\theta'_{30} + \theta'_{31}\phi_0 + \theta'_{31}\phi_1 h_1 + \theta'_{31}\phi_2 h_2 + \theta'_{31}\phi_3 h_3]$$

Create new variables

$$h'_1 = a[\theta'_{10} + \theta'_{11}y] = a[\theta'_{10} + \theta'_{11}\phi_0 + \theta'_{11}\phi_1h_1 + \theta'_{11}\phi_2h_2 + \theta'_{11}\phi_3h_3]$$

$$h'_2 = a[\theta'_{20} + \theta'_{21}y] = a[\theta'_{20} + \theta'_{21}\phi_0 + \theta'_{21}\phi_1h_1 + \theta'_{21}\phi_2h_2 + \theta'_{21}\phi_3h_3]$$

$$h'_3 = a[\theta'_{30} + \theta'_{31}y] = a[\theta'_{30} + \theta'_{31}\phi_0 + \theta'_{31}\phi_1h_1 + \theta'_{31}\phi_2h_2 + \theta'_{31}\phi_3h_3]$$

$$h'_1 = a[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$

$$h'_2 = a[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3]$$

$$h'_3 = a[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]$$

Two-layer network

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

$$h_2 = a[\theta_{20} + \theta_{21}x]$$

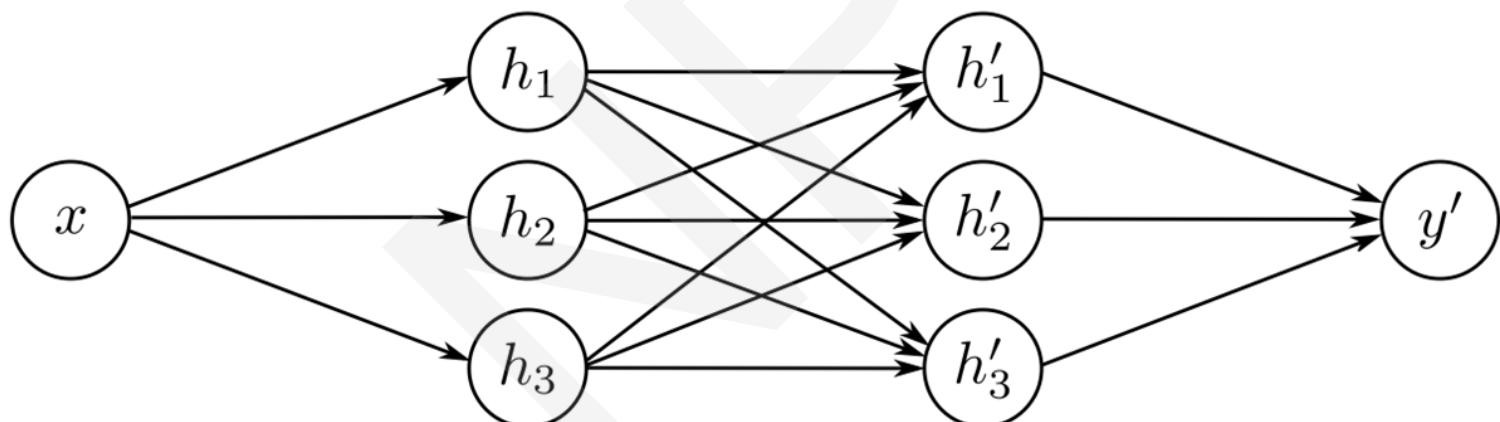
$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$h'_1 = a[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$

$$h'_2 = a[\psi_{20} + \psi_{21}h_2 + \psi_{22}h_2 + \psi_{23}h_3]$$

$$h'_3 = a[\psi_{30} + \psi_{31}h_2 + \psi_{32}h_2 + \psi_{33}h_3]$$

$$y' = \phi'_0 + \phi'_1 h'_1 + \phi'_2 h'_2 + \phi'_3 h'_3$$



Remember shallow network with two outputs?

1 input, 4 hidden units, 2 outputs

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

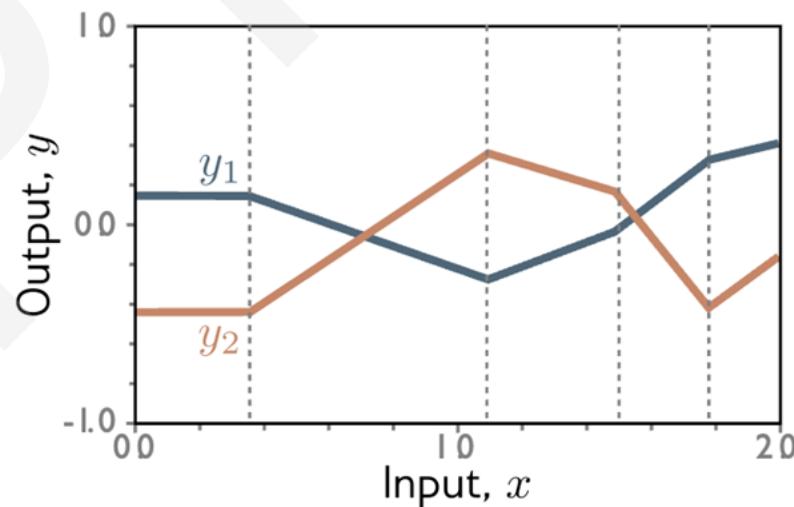
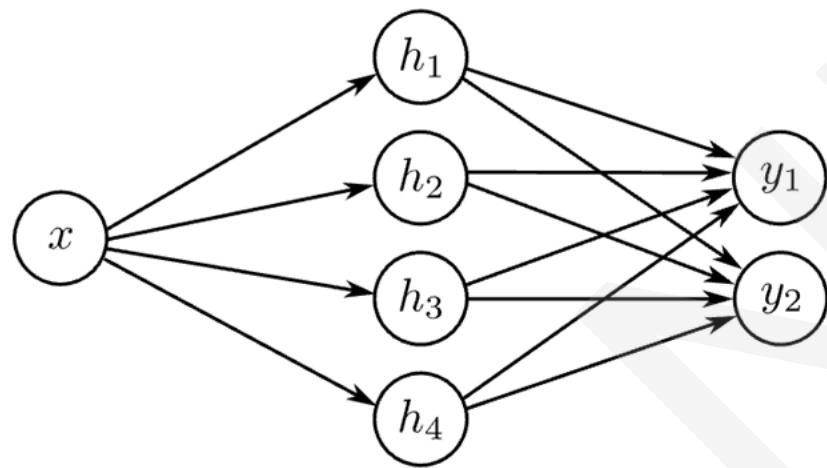
$$h_2 = a[\theta_{20} + \theta_{21}x]$$

$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$h_4 = a[\theta_{40} + \theta_{41}x]$$

$$y_1 = \phi_{10} + \phi_{11}h_1 + \phi_{12}h_2 + \phi_{13}h_3 + \phi_{14}h_4$$

$$y_2 = \phi_{20} + \phi_{21}h_1 + \phi_{22}h_2 + \phi_{23}h_3 + \phi_{24}h_4$$



Networks as composing functions

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

$$h_2 = a[\theta_{20} + \theta_{21}x]$$

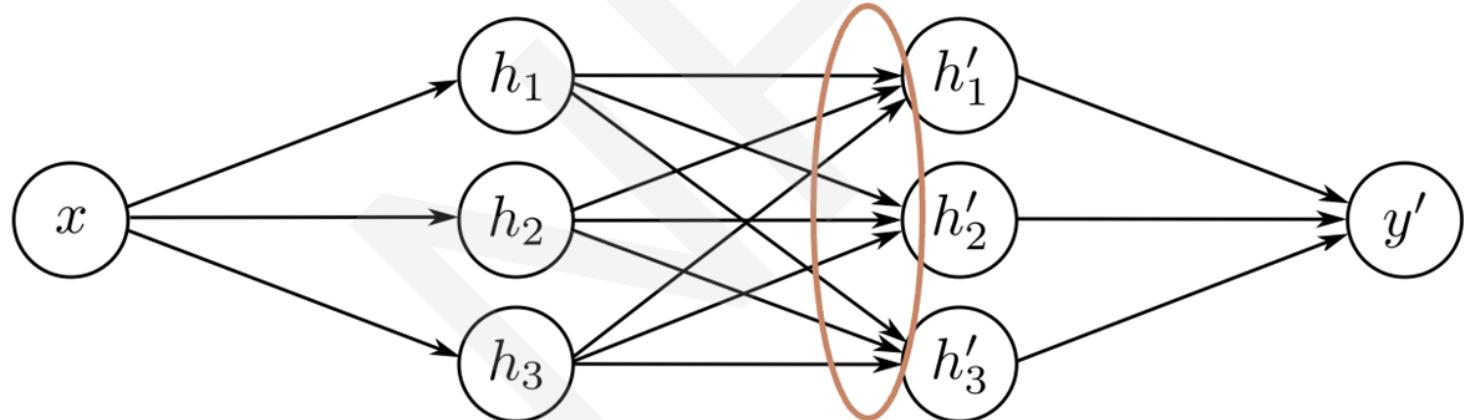
$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$h'_1 = a[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$

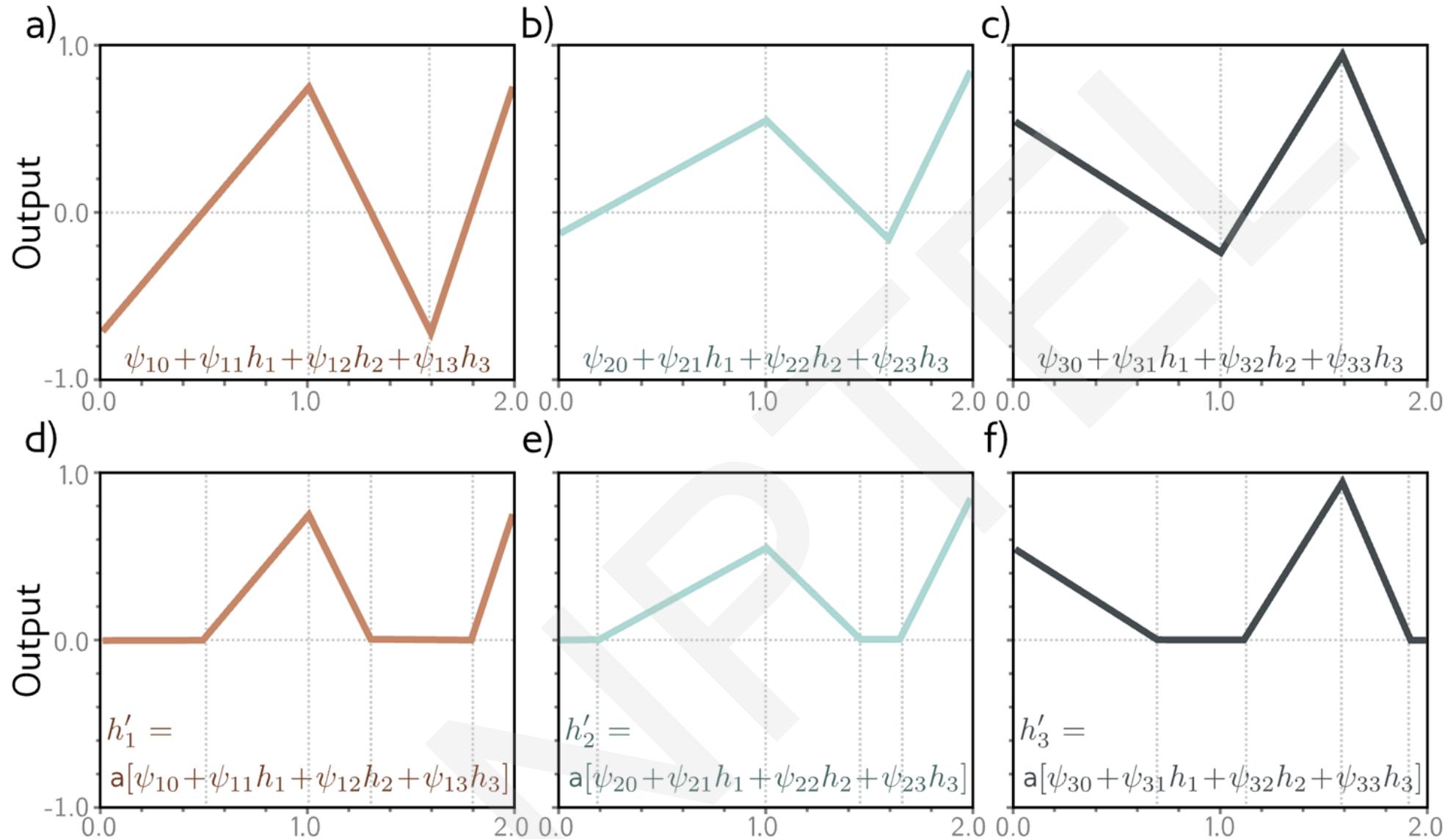
$$h'_2 = a[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3]$$

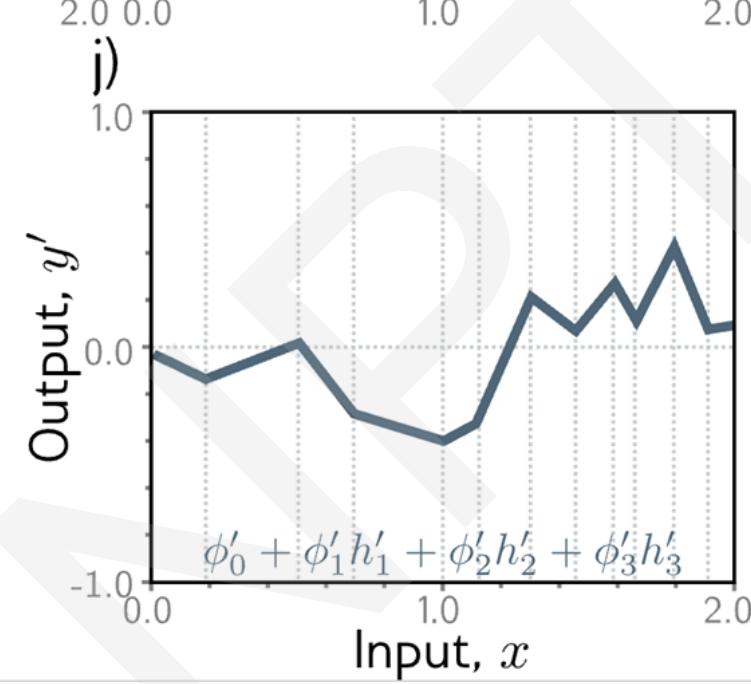
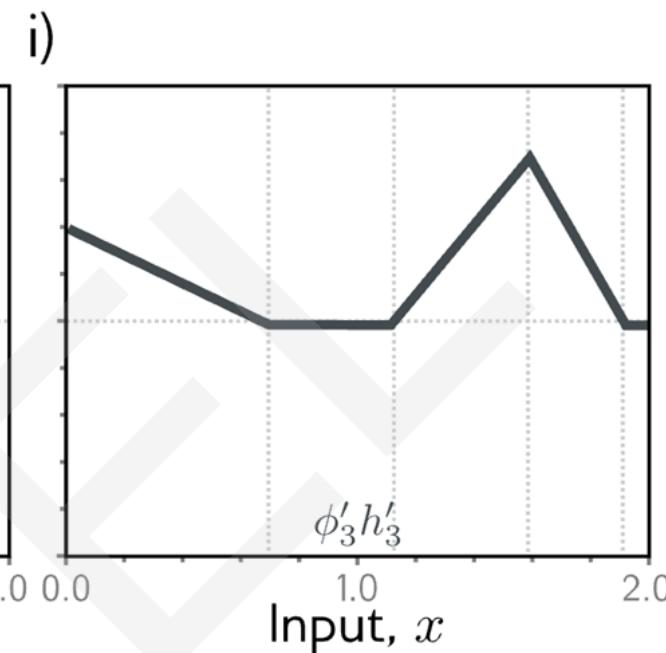
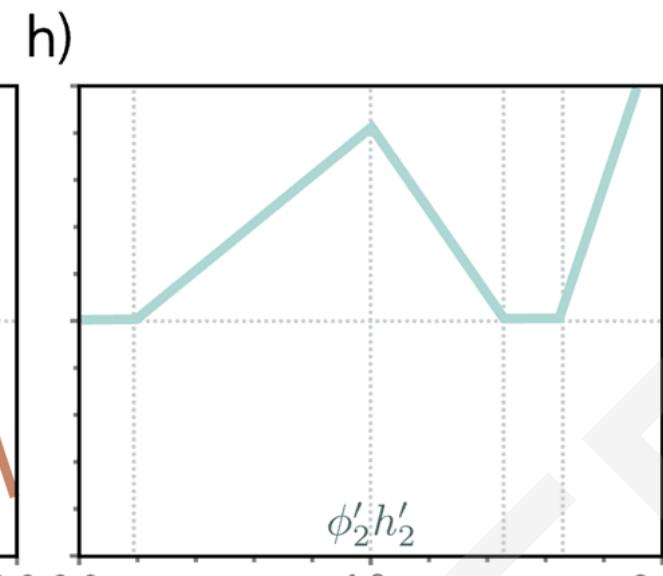
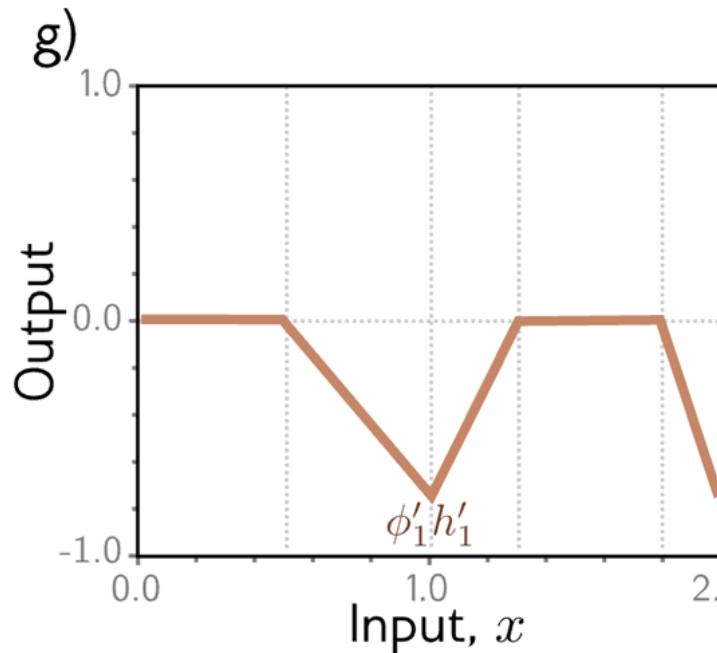
$$h'_3 = a[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]$$

Consider the pre-activations at the second hidden units
 At this point, it's a one-layer network with three outputs



*Slides adapted from <http://udlbook.com>





Networks as composing functions

$$h_1 = a[\theta_{10} + \theta_{11}x]$$

$$h_2 = a[\theta_{20} + \theta_{21}x]$$

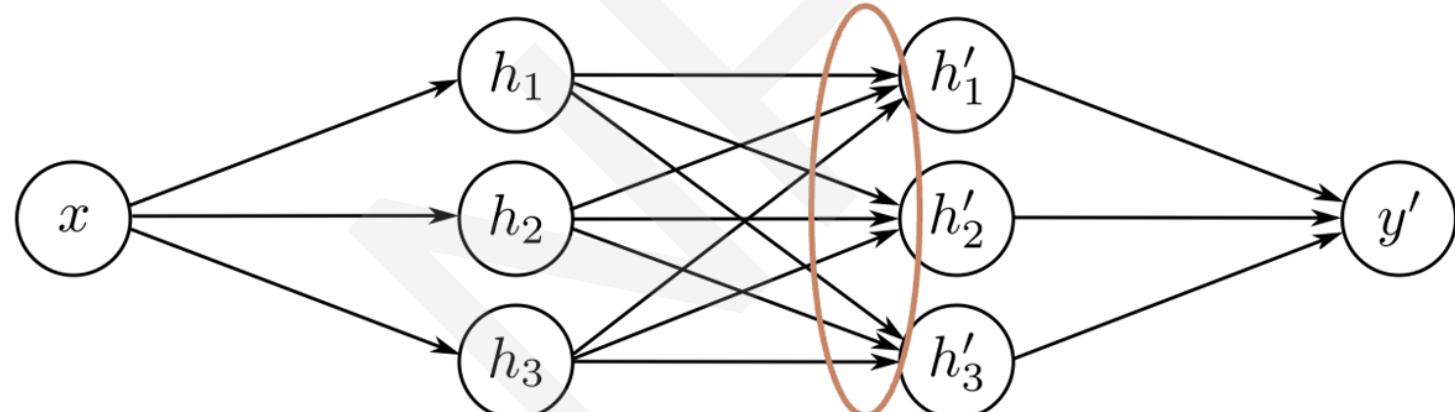
$$h_3 = a[\theta_{30} + \theta_{31}x]$$

$$h'_1 = a[\psi_{10} + \psi_{11}h_1 + \psi_{12}h_2 + \psi_{13}h_3]$$

$$h'_2 = a[\psi_{20} + \psi_{21}h_1 + \psi_{22}h_2 + \psi_{23}h_3]$$

$$h'_3 = a[\psi_{30} + \psi_{31}h_1 + \psi_{32}h_2 + \psi_{33}h_3]$$

Consider the pre-activations at the second hidden units
 At this point, it's a one--layer network with three outputs



Hyperparameters

- K layers = **depth of network**
- D_k hidden units per layer = **width of network**
- These are called **hyperparameters** – chosen before training the network
- Can try retraining with different hyperparameters – **hyperparameter optimization** or **hyperparameter search**

Notations

$$\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \mathbf{a} \left[\begin{bmatrix} \theta_{10} \\ \theta_{20} \\ \theta_{30} \end{bmatrix} + \begin{bmatrix} \theta_{11} \\ \theta_{21} \\ \theta_{31} \end{bmatrix} x \right] \longrightarrow$$

$$\mathbf{h} = \mathbf{a} [\boldsymbol{\theta}_0 + \boldsymbol{\theta} x]$$

$$\begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} = \mathbf{a} \left[\begin{bmatrix} \psi_{10} \\ \psi_{20} \\ \psi_{30} \end{bmatrix} + \begin{bmatrix} \psi_{11} & \psi_{12} & \psi_{13} \\ \psi_{21} & \psi_{22} & \psi_{23} \\ \psi_{32} & \psi_{32} & \psi_{33} \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \right] \longrightarrow$$

$$\mathbf{h}' = \mathbf{a} [\boldsymbol{\psi}_0 + \boldsymbol{\Psi} \mathbf{h}]$$

$$y' = \phi'_0 + [\phi'_1 \quad \phi'_2 \quad \phi'_3] \begin{bmatrix} h'_1 \\ h'_2 \\ h'_3 \end{bmatrix} \longrightarrow$$

$$y = \boldsymbol{\phi}'_0 + \boldsymbol{\phi}' \mathbf{h}'$$

Notations

$$\mathbf{h} = \mathbf{a} [\theta_0 + \theta x]$$



$$\mathbf{h}_1 = \mathbf{a} [\beta_0 + \Omega_0 \mathbf{x}]$$

$$\mathbf{h}' = \mathbf{a} [\psi_0 + \Psi \mathbf{h}]$$



$$\mathbf{h}_2 = \mathbf{a} [\beta_1 + \Omega_1 \mathbf{h}_1]$$

$$y = \phi'_0 + \phi' \mathbf{h}'$$



$$\mathbf{y} = \beta_2 + \Omega_2 \mathbf{h}_2$$

Notations

$$\mathbf{h} = \mathbf{a} [\theta_0 + \theta x]$$



Bias vector
Weight matrix

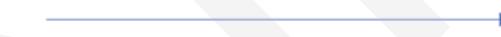
$$\mathbf{h}_1 = \mathbf{a} [\beta_0 + \Omega_0 \mathbf{x}]$$

$$\mathbf{h}' = \mathbf{a} [\psi_0 + \Psi \mathbf{h}]$$



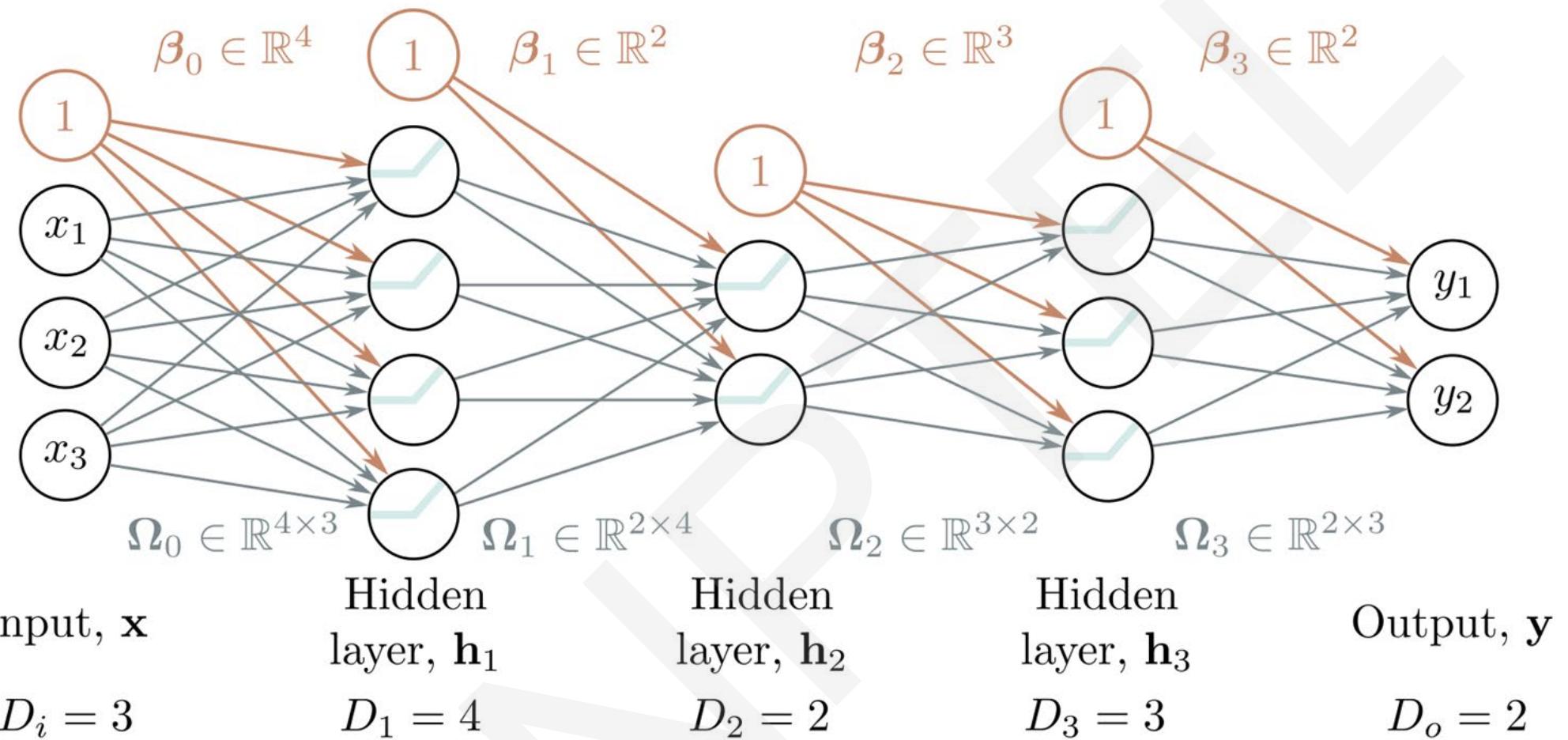
$$\mathbf{h}_2 = \mathbf{a} [\beta_1 + \Omega_1 \mathbf{h}_1]$$

$$y = \phi'_0 + \phi' \mathbf{h}'$$



$$\mathbf{y} = \beta_2 + \Omega_2 \mathbf{h}_2$$

Example



*Slides adapted from <http://udlbook.com>

Shallow vs. deep networks

Ability to approximate different functions?

Both obey the universal approximation theorem.

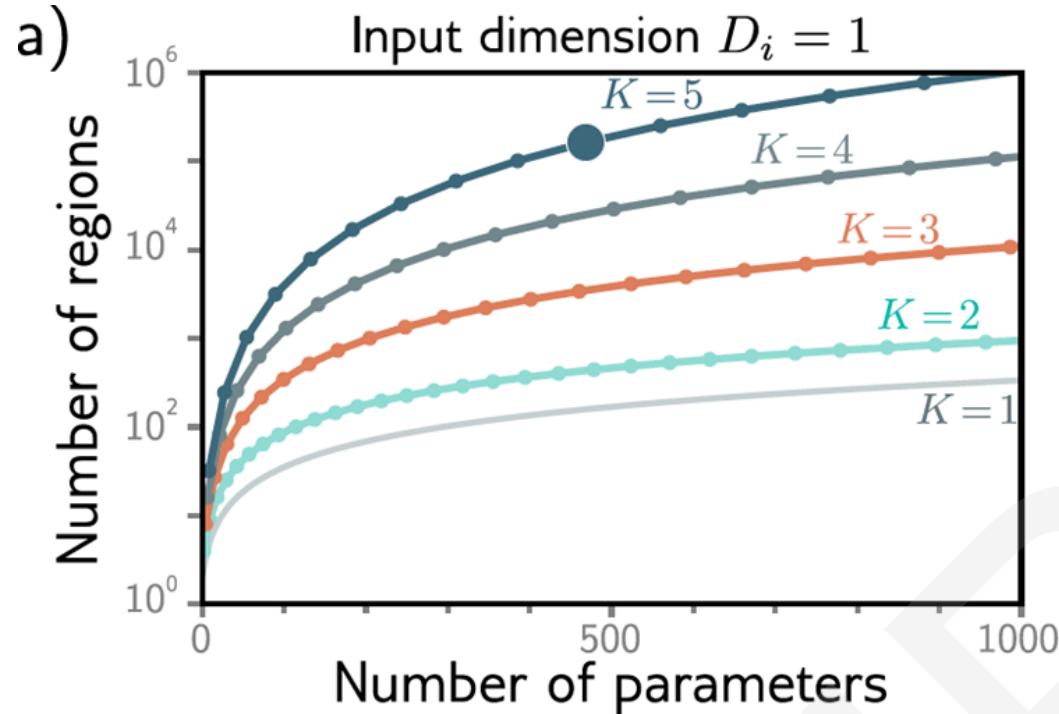
Argument: One layer is enough, and for deep networks could arrange for the other layers to compute the identity function.

Shallow vs. deep networks

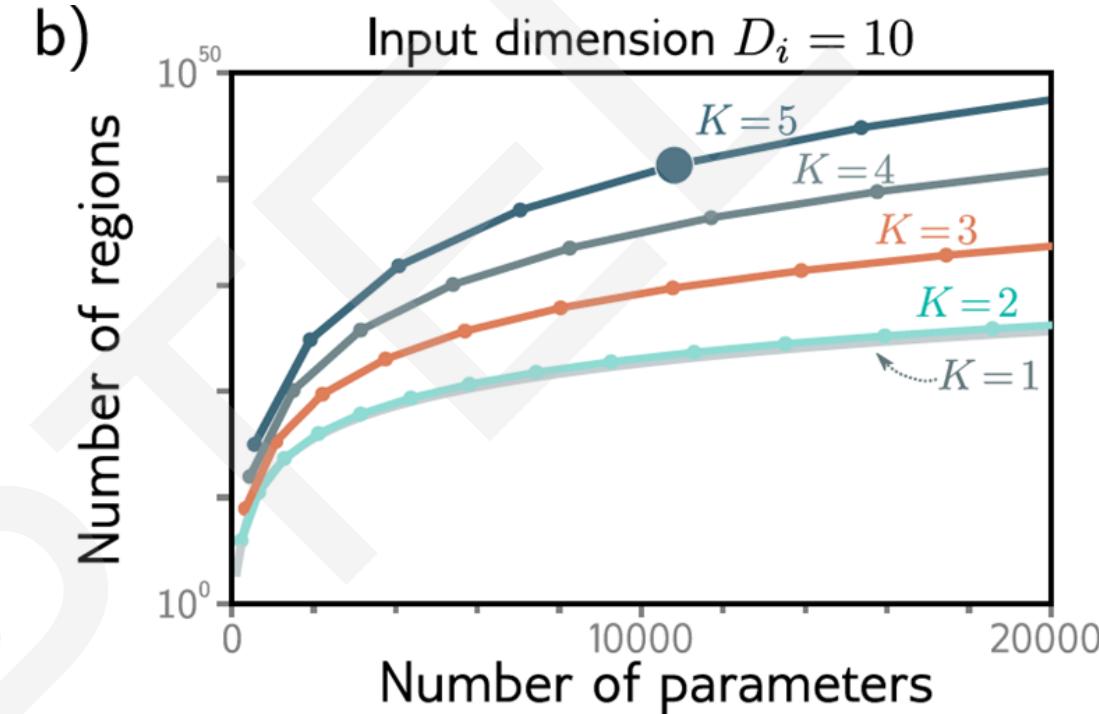
Number of linear regions per parameter

- Deep networks create many more regions per parameters

Number of linear regions per parameter



5 layers
10 hidden units per layer
471 parameters
161,501 linear regions

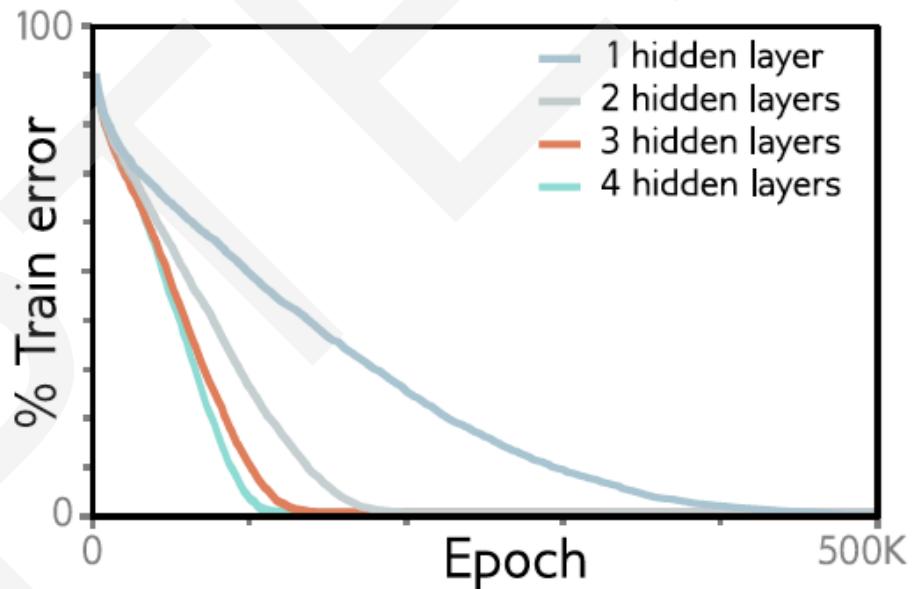


5 layers
50 hidden units per layer
10,801 parameters
 $>10^{40}$ linear regions

Shallow vs. Deep Networks

Fitting of deep models is also faster

Figure 20.2 MNIST-1D training. Four fully connected networks were fit to 4000 MNIST-1D examples with random labels using full batch gradient descent, He initialization, no momentum or regularization, and learning rate 0.0025. Models with 1,2,3,4 layers had 298, 100, 75, and 63 hidden units per layer and 15208, 15210, 15235, and 15139 parameters, respectively. All models train successfully, but deeper models require fewer epochs.



REFERENCES

Understanding Deep Learning [<https://udlbook.github.io/udlbook/>]
[Chapter 4]



THANK YOU



N P T E L O N L I N E C E R T I F I C A T I O N C O U R S E S

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 9 : Backpropagation



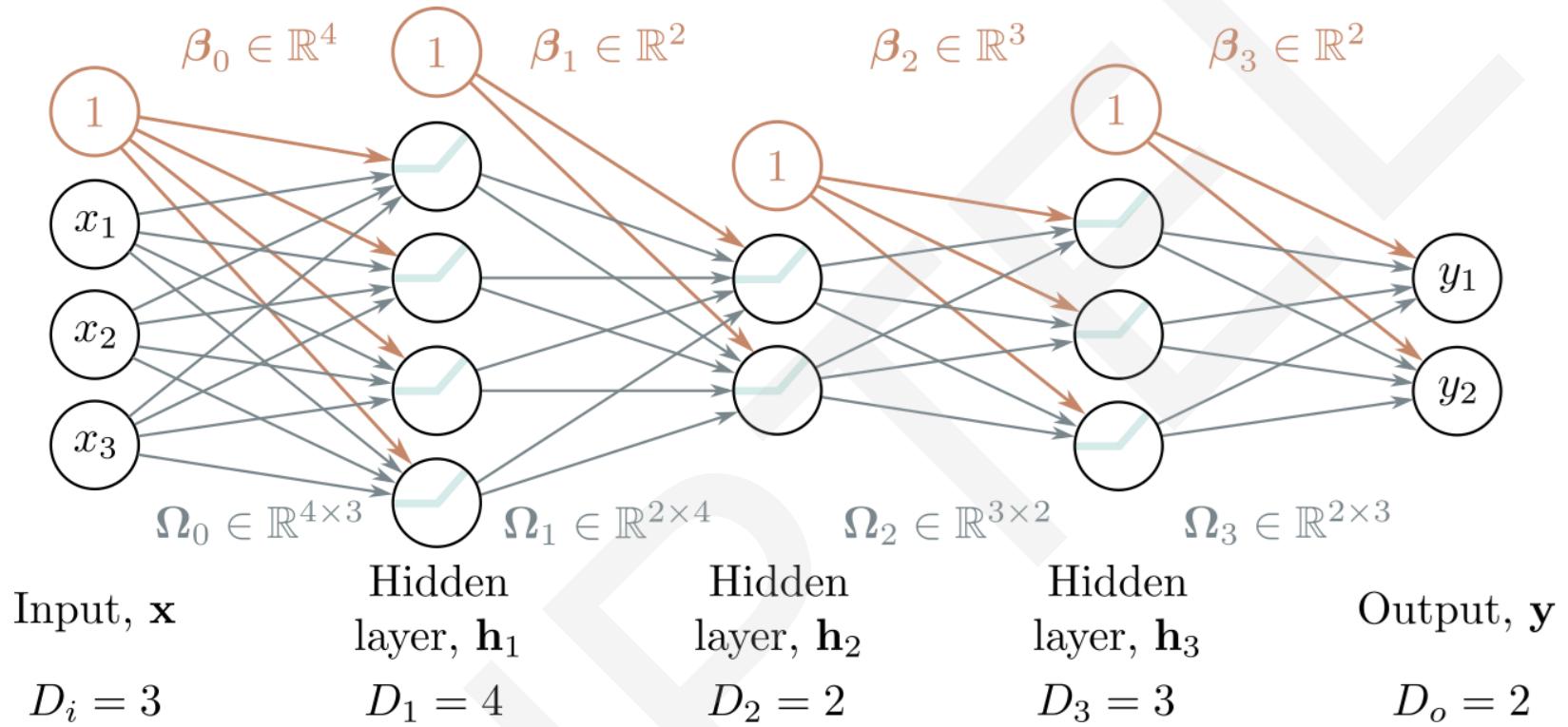
PROF . PAWAN GOYAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Computing gradients
- Backpropagation Algorithm on a toy example
- How to do this on a deep neural network?

Example of a Deep Neural Network



$$\mathbf{h}_1 = \mathbf{a}[\boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}]$$

$$\mathbf{h}_2 = \mathbf{a}[\boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1]$$

$$\mathbf{h}_3 = \mathbf{a}[\boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2]$$

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

*Slides adapted from <http://udlbook.com>

Computing gradients

Loss: sum of individual terms:

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

SGD Algorithm:

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

Parameters:

$$\phi = \{\beta_0, \Omega_0, \beta_1, \Omega_1, \beta_2, \Omega_2, \beta_3, \Omega_3\}$$

Need to compute gradients

$$\frac{\partial \ell_i}{\partial \beta_k} \quad \text{and} \quad \frac{\partial \ell_i}{\partial \Omega_k}$$

Why is this such a big deal?

- A neural network is just an equation:

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

- But it's a huge equation, and we need to compute derivative
 - for every parameter
 - for every point in the batch
 - for every iteration of SGD

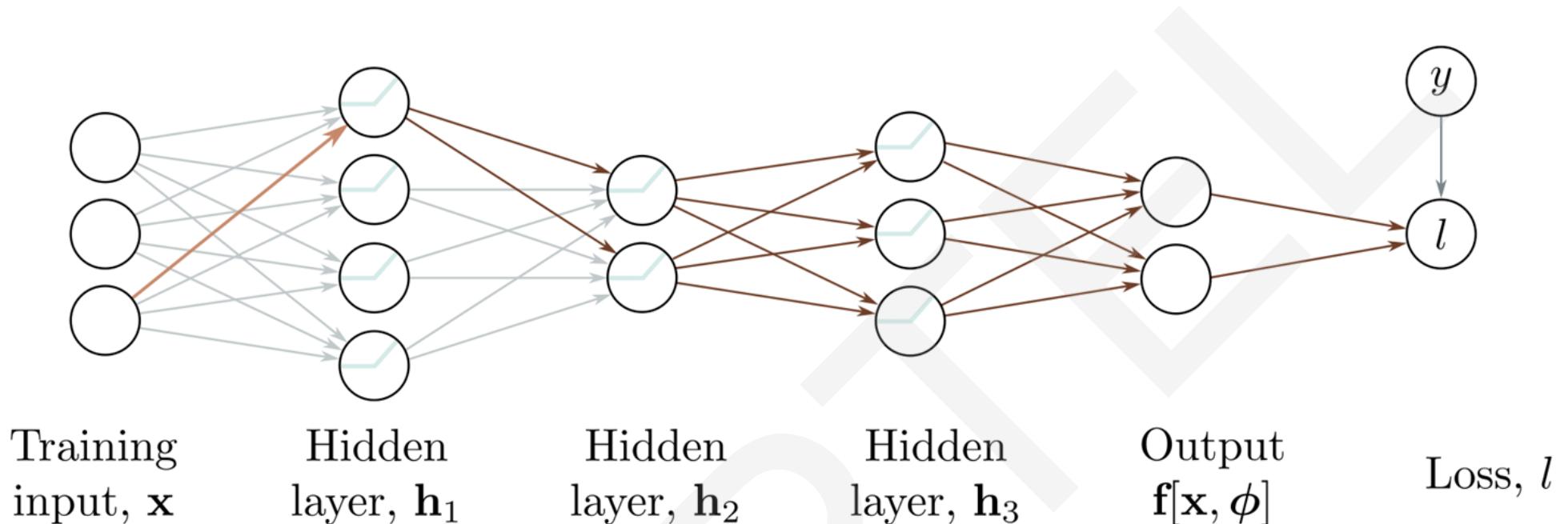
Algorithm to compute gradient efficiently



- “Backpropagation algorithm”
- Rumelhart, Hinton, and Williams (1986)

*Slides adapted from <http://udlbook.com>

BackProp intuition: the backward pass



To calculate how a small change in a weight or bias feeding into hidden layer \mathbf{h}_1 modifies the loss, we need to know:

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

Example: Toy Function

$$f[x, \phi] = \beta_3 + \omega_3 \cdot \cos[\beta_2 + \omega_2 \cdot \exp[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]]]$$

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

We want to calculate:

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

How does a small change in β_3 change the loss ℓ_i for the i 'th example?

Forward Pass

$$f[x, \phi] = \beta_3 + \omega_3 \cdot \cos[\beta_2 + \omega_2 \cdot \exp[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x]]]$$

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

1. Write this as a series of intermediate calculations

$$f_0 = \beta_0 + \omega_0 \cdot x_i$$

$$h_1 = \sin[f_0]$$

$$f_1 = \beta_1 + \omega_1 \cdot h_1$$

$$h_2 = \exp[f_1]$$

$$f_2 = \beta_2 + \omega_2 \cdot h_2$$

$$h_3 = \cos[f_2]$$

$$f_3 = \beta_3 + \omega_3 \cdot h_3$$

$$\ell_i = (f_3 - y_i)^2.$$

2. Compute these intermediate quantities



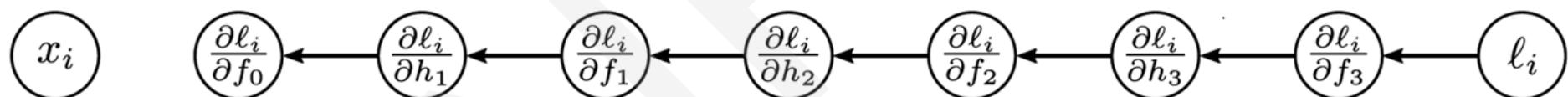
Backward pass

$$f[x, \phi] = \beta_3 + \omega_3 \cdot \cos \left[\beta_2 + \omega_2 \cdot \exp \left[\beta_1 + \omega_1 \cdot \sin[\beta_0 + \omega_0 \cdot x] \right] \right]$$

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

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

$$f_0 = \beta_0 + \omega_0 \cdot x_i$$

$$h_1 = \sin[f_0]$$

$$f_1 = \beta_1 + \omega_1 \cdot h_1$$

$$h_2 = \exp[f_1]$$

$$f_2 = \beta_2 + \omega_2 \cdot h_2$$

$$h_3 = \cos[f_2]$$

$$f_3 = \beta_3 + \omega_3 \cdot h_3$$

$$\ell_i = (f_3 - y_i)^2.$$

- The first of these derivatives is trivial

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

Backward pass

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

$$f_0 = \beta_0 + \omega_0 \cdot x_i$$

$$h_1 = \sin[f_0]$$

$$f_1 = \beta_1 + \omega_1 \cdot h_1$$

$$h_2 = \exp[f_1]$$

$$f_2 = \beta_2 + \omega_2 \cdot h_2$$

$$h_3 = \cos[f_2]$$

$$f_3 = \beta_3 + \omega_3 \cdot h_3$$

$$\ell_i = (f_3 - y_i)^2.$$

- The second derivative is computed via 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 h_3 change ℓ_i ?

How does a small change in h_3 change f_3 ?

How does a small change in f_3 change ℓ_i ?

Backward pass

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

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

- The remaining derivatives also calculated by further use of chain rule

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

Backward pass

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

$$f_0 = \beta_0 + \omega_0 \cdot x_i$$

$$h_1 = \sin[f_0]$$

$$f_1 = \beta_1 + \omega_1 \cdot h_1$$

$$h_2 = \exp[f_1]$$

$$f_2 = \beta_2 + \omega_2 \cdot h_2$$

$$h_3 = \cos[f_2]$$

$$f_3 = \beta_3 + \omega_3 \cdot h_3$$

$$\ell_i = (f_3 - y_i)^2.$$

- Another application of the chain rule
- Similarly for β parameters

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

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

Backward pass

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

$$f_0 = \beta_0 + \omega_0 \cdot x_i$$

$$h_1 = \sin[f_0]$$

$$f_1 = \beta_1 + \omega_1 \cdot h_1$$

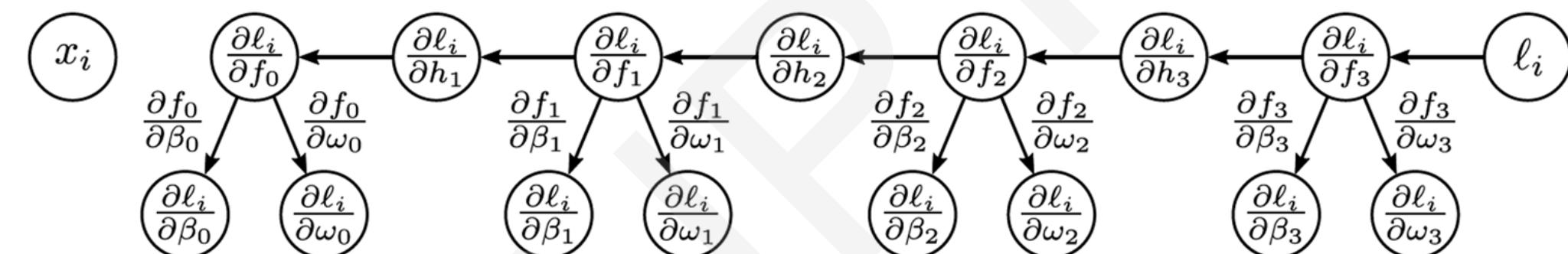
$$h_2 = \exp[f_1]$$

$$f_2 = \beta_2 + \omega_2 \cdot h_2$$

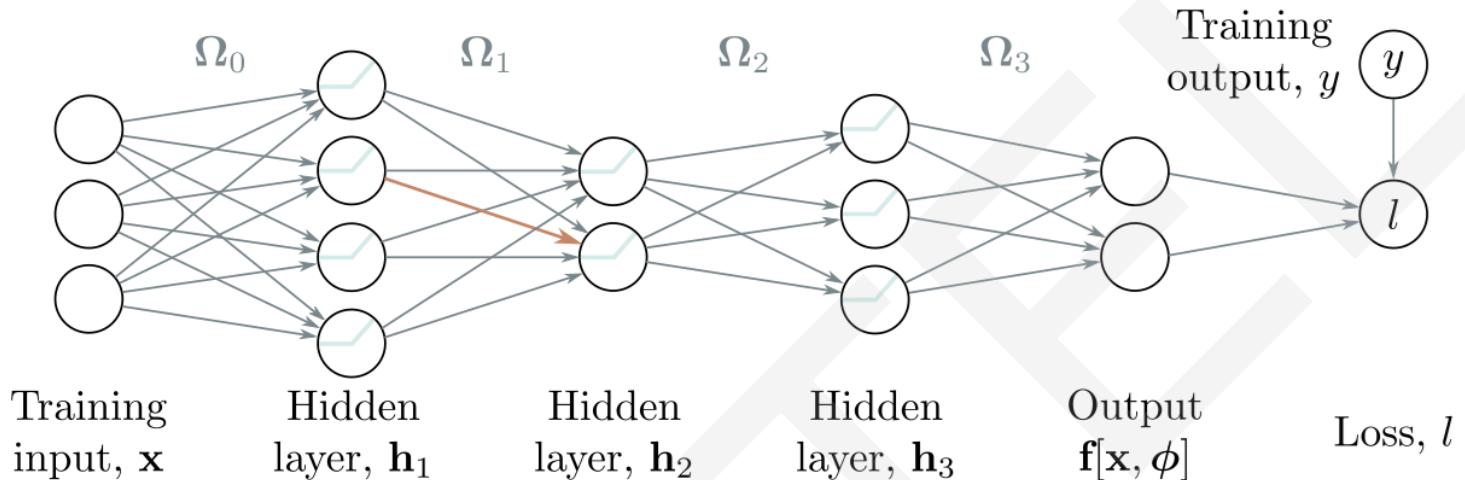
$$h_3 = \cos[f_2]$$

$$f_3 = \beta_3 + \omega_3 \cdot h_3$$

$$\ell_i = (f_3 - y_i)^2.$$



Taking a deep Neural Network



1. Write this as a series of intermediate calculations

$$\mathbf{f}_0 = \boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}_i$$

$$\mathbf{h}_1 = \mathbf{a}[\mathbf{f}_0]$$

$$\mathbf{f}_1 = \boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1$$

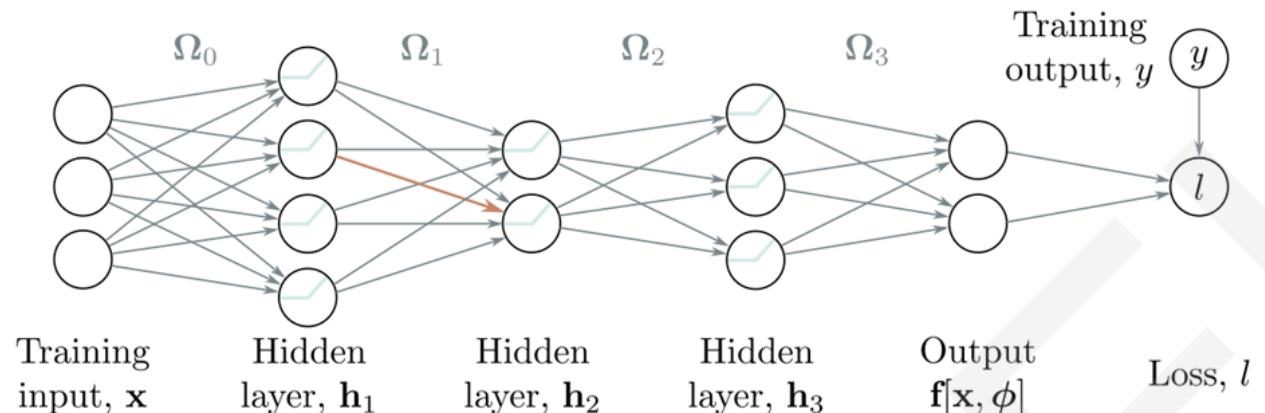
$$\mathbf{h}_2 = \mathbf{a}[\mathbf{f}_1]$$

$$\mathbf{f}_2 = \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2$$

$$\mathbf{h}_3 = \mathbf{a}[\mathbf{f}_2]$$

$$\mathbf{f}_3 = \boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3 \mathbf{h}_3$$

$$\ell_i = l[\mathbf{f}_3, y_i]$$



1. Write this as a series of intermediate calculations

$$\mathbf{f}_0 = \beta_0 + \Omega_0 \mathbf{x}_i$$

2. Compute these intermediate quantities

$$\mathbf{h}_1 = \mathbf{a}[\mathbf{f}_0]$$

$$\mathbf{f}_1 = \beta_1 + \Omega_1 \mathbf{h}_1$$

$$\mathbf{h}_2 = \mathbf{a}[\mathbf{f}_1]$$

$$\mathbf{f}_2 = \beta_2 + \Omega_2 \mathbf{h}_2$$

$$\mathbf{h}_3 = \mathbf{a}[\mathbf{f}_2]$$

$$\mathbf{f}_3 = \beta_3 + \Omega_3 \mathbf{h}_3$$

3. Take derivatives of output with respect to intermediate quantities

$$\ell_i = l[\mathbf{f}_3, y_i]$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_2} = \frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_1} = \frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \left(\frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_0} = \frac{\partial \mathbf{h}_1}{\partial \mathbf{f}_0} \frac{\partial \mathbf{f}_1}{\partial \mathbf{h}_1} \left(\frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

How to compute this for vectors?

Matrix Calculus

Scalar function $f[]$ of a vector \mathbf{a}

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

$$\frac{\partial f}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\partial f}{\partial a_1} \\ \frac{\partial f}{\partial a_2} \\ \frac{\partial f}{\partial a_3} \\ \frac{\partial f}{\partial a_4} \end{bmatrix}$$

Matrix Calculus

Scalar function $f[]$ of a matrix \mathbf{A}

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix}$$

$$\frac{\partial f}{\partial \mathbf{A}} = \begin{bmatrix} \frac{\partial f}{\partial a_{11}} & \frac{\partial f}{\partial a_{12}} & \frac{\partial f}{\partial a_{13}} \\ \frac{\partial f}{\partial a_{21}} & \frac{\partial f}{\partial a_{22}} & \frac{\partial f}{\partial a_{23}} \\ \frac{\partial f}{\partial a_{31}} & \frac{\partial f}{\partial a_{32}} & \frac{\partial f}{\partial a_{33}} \\ \frac{\partial f}{\partial a_{41}} & \frac{\partial f}{\partial a_{42}} & \frac{\partial f}{\partial a_{43}} \end{bmatrix}$$

Matrix Calculus

Vector function $\mathbf{f}[]$ of vector \mathbf{a}

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \quad \frac{\partial \mathbf{f}}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\partial f_1}{\partial a_1} & \frac{\partial f_2}{\partial a_1} & \frac{\partial f_3}{\partial a_1} \\ \frac{\partial f_1}{\partial a_2} & \frac{\partial f_2}{\partial a_2} & \frac{\partial f_3}{\partial a_2} \\ \frac{\partial f_1}{\partial a_3} & \frac{\partial f_2}{\partial a_3} & \frac{\partial f_3}{\partial a_3} \\ \frac{\partial f_1}{\partial a_4} & \frac{\partial f_2}{\partial a_4} & \frac{\partial f_3}{\partial a_4} \end{bmatrix}$$

Comparing vector and matrix

Scalar derivatives:

$$f_3 = \beta_3 + \omega_3 h_3$$

$$\frac{\partial f_3}{\partial h_3} = \frac{\partial}{\partial h_3} (\beta_3 + \omega_3 h_3) = \omega_3$$

Matrix derivatives:

$$\mathbf{f}_3 = \boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3 \mathbf{h}_3$$

$$\frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} = \frac{\partial}{\partial \mathbf{h}_3} (\boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3 \mathbf{h}_3) = \boldsymbol{\Omega}_3^T$$

Comparing vector and matrix

Scalar derivatives:

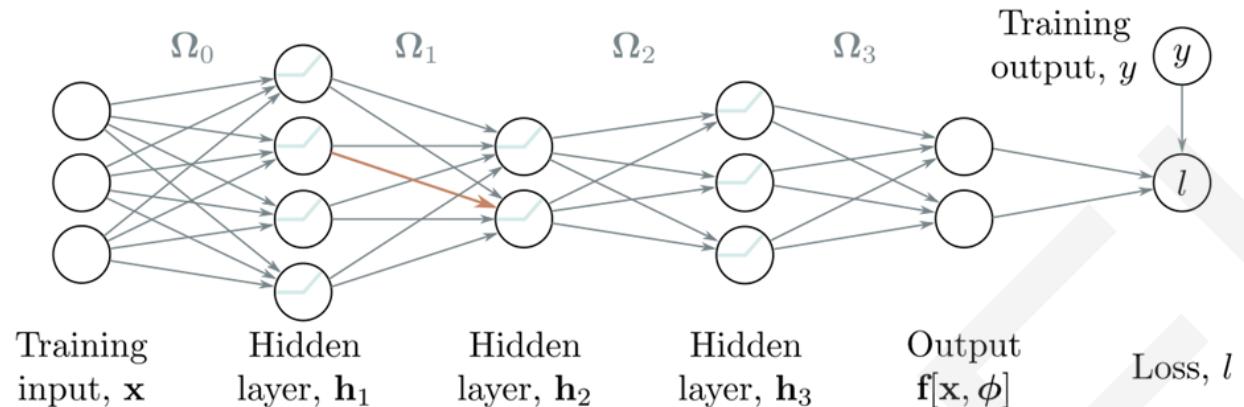
$$f_3 = \beta_3 + \omega_3 h_3$$

$$\frac{\partial f_3}{\partial \beta_3} = \frac{\partial}{\partial \omega_3} \beta_3 + \omega_3 h_3 = 1$$

Matrix derivatives:

$$\mathbf{f}_3 = \boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3 \mathbf{h}_3$$

$$\frac{\partial \mathbf{f}_3}{\partial \boldsymbol{\beta}_3} = \frac{\partial}{\partial \boldsymbol{\beta}_3} (\boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3 \mathbf{h}_3) = \mathbf{I}$$



1. Write this as a series of intermediate calculations

$$\mathbf{f}_0 = \beta_0 + \Omega_0 \mathbf{x}_i$$

2. Compute these intermediate quantities

$$\mathbf{h}_1 = \mathbf{a}[\mathbf{f}_0]$$

$$\mathbf{f}_1 = \beta_1 + \Omega_1 \mathbf{h}_1$$

$$\mathbf{h}_2 = \mathbf{a}[\mathbf{f}_1]$$

$$\mathbf{f}_2 = \beta_2 + \Omega_2 \mathbf{h}_2$$

$$\mathbf{h}_3 = \mathbf{a}[\mathbf{f}_2]$$

$$\mathbf{f}_3 = \beta_3 + \Omega_3 \mathbf{h}_3$$

3. Take derivatives of output with respect to intermediate quantities

$$\ell_i = l[\mathbf{f}_3, y_i]$$

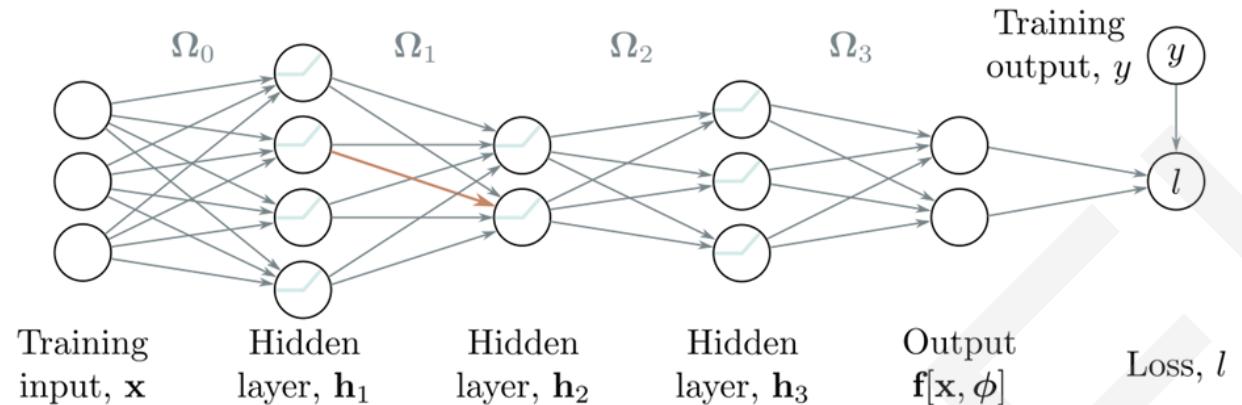
$$\frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_2} = \frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_1} = \frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \left(\frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_0} = \frac{\partial \mathbf{h}_1}{\partial \mathbf{f}_0} \frac{\partial \mathbf{f}_1}{\partial \mathbf{h}_1} \left(\frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

How to compute this for vectors?



1. Write this as a series of intermediate calculations

$$\mathbf{f}_0 = \boldsymbol{\beta}_0 + \Omega_0 \mathbf{x}_i$$

2. Compute these intermediate quantities

$$\mathbf{h}_1 = \mathbf{a}[\mathbf{f}_0]$$

$$\mathbf{f}_1 = \boldsymbol{\beta}_1 + \Omega_1 \mathbf{h}_1$$

$$\mathbf{h}_2 = \mathbf{a}[\mathbf{f}_1]$$

$$\mathbf{f}_2 = \boldsymbol{\beta}_2 + \Omega_2 \mathbf{h}_2$$

$$\mathbf{h}_3 = \mathbf{a}[\mathbf{f}_2]$$

$$\mathbf{f}_3 = \boldsymbol{\beta}_3 + \Omega_3 \mathbf{h}_3$$

3. Take derivatives of output with respect to intermediate quantities

$$\ell_i = l[\mathbf{f}_3, y_i]$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_2} = \frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_1} = \frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \left(\frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_0} = \frac{\partial \mathbf{h}_1}{\partial \mathbf{f}_0} \frac{\partial \mathbf{f}_1}{\partial \mathbf{h}_1} \left(\frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

$$\boxed{\frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} = \frac{\partial}{\partial \mathbf{h}_3} (\boldsymbol{\beta}_3 + \Omega_3 \mathbf{h}_3) = \Omega_3^T}$$

Derivative of RELU

1. Consider:

$$\mathbf{a} = \text{ReLU}[\mathbf{b}]$$

where:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

2. We could equivalently write:

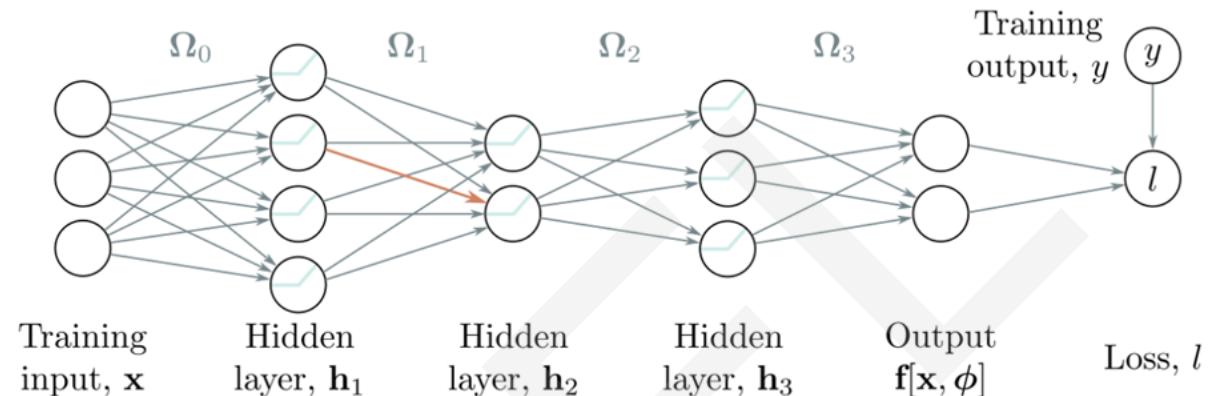
$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \text{ReLU}[b_1] \\ \text{ReLU}[b_2] \\ \text{ReLU}[b_3] \end{bmatrix}$$

3. Taking the derivative

$$\frac{\partial \mathbf{a}}{\partial \mathbf{b}} = \begin{bmatrix} \frac{\partial a_1}{\partial b_1} & \frac{\partial a_2}{\partial b_1} & \frac{\partial a_3}{\partial b_1} \\ \frac{\partial a_1}{\partial b_2} & \frac{\partial a_2}{\partial b_2} & \frac{\partial a_3}{\partial b_2} \\ \frac{\partial a_1}{\partial b_3} & \frac{\partial a_2}{\partial b_3} & \frac{\partial a_3}{\partial b_3} \end{bmatrix} = \begin{bmatrix} \mathbb{I}[b_1 > 0] & 0 & 0 \\ 0 & \mathbb{I}[b_2 > 0] & 0 \\ 0 & 0 & \mathbb{I}[b_3 > 0] \end{bmatrix}$$

4. We can equivalently pointwise multiply by diagonal

$$\mathbb{I}[\mathbf{b} > 0] \odot$$



1. Write this as a series of intermediate calculations

2. Compute these intermediate quantities

3. Take derivatives of output with respect to intermediate quantities

$$\mathbf{f}_0 = \beta_0 + \Omega_0 \mathbf{x}_i$$

$$\mathbf{h}_1 = \mathbf{a}[\mathbf{f}_0]$$

$$\mathbf{f}_1 = \beta_1 + \Omega_1 \mathbf{h}_1$$

$$\mathbf{h}_2 = \mathbf{a}[\mathbf{f}_1]$$

$$\mathbf{f}_2 = \beta_2 + \Omega_2 \mathbf{h}_2$$

$$\mathbf{h}_3 = \mathbf{a}[\mathbf{f}_2]$$

$$\mathbf{f}_3 = \beta_3 + \Omega_3 \mathbf{h}_3$$

$$\ell_i = \mathbb{I}[\mathbf{f}_3, y_i]$$

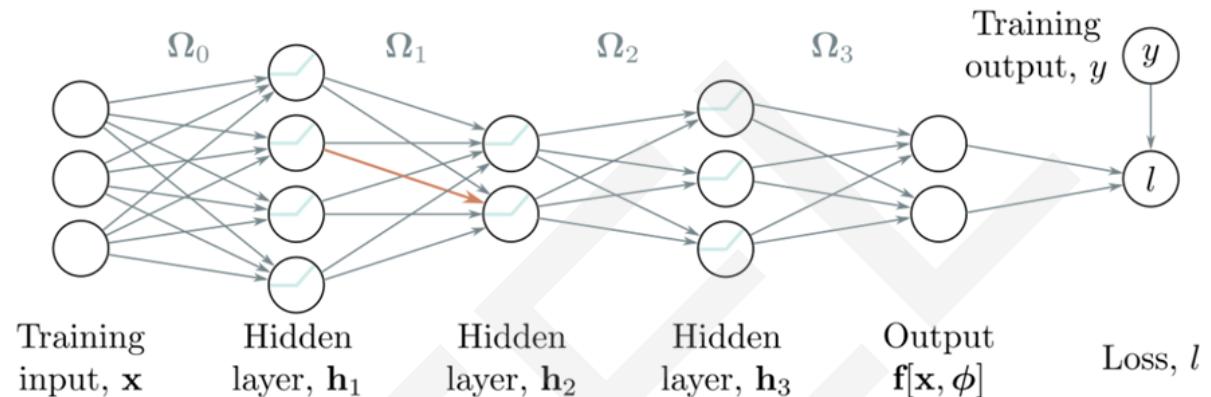
$$\frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_2} = \boxed{\frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2}} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3}$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_1} = \frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \left(\frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

$$\frac{\partial \ell_i}{\partial \mathbf{f}_0} = \frac{\partial \mathbf{h}_1}{\partial \mathbf{f}_0} \frac{\partial \mathbf{f}_1}{\partial \mathbf{h}_1} \left(\frac{\partial \mathbf{h}_2}{\partial \mathbf{f}_1} \frac{\partial \mathbf{f}_2}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_3}{\partial \mathbf{f}_2} \frac{\partial \mathbf{f}_3}{\partial \mathbf{h}_3} \frac{\partial \ell_i}{\partial \mathbf{f}_3} \right)$$

$\mathbb{I}[\mathbf{f}_2 > 0]$



1. Write this as a series of intermediate calculations

2. Compute these intermediate quantities

3. Take derivatives of output with respect to intermediate quantities

4. Take derivatives w.r.t. parameters

$$\mathbf{f}_0 = \boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}_i$$

$$\mathbf{h}_1 = \mathbf{a}[\mathbf{f}_0]$$

$$\mathbf{f}_1 = \boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1$$

$$\mathbf{h}_2 = \mathbf{a}[\mathbf{f}_1]$$

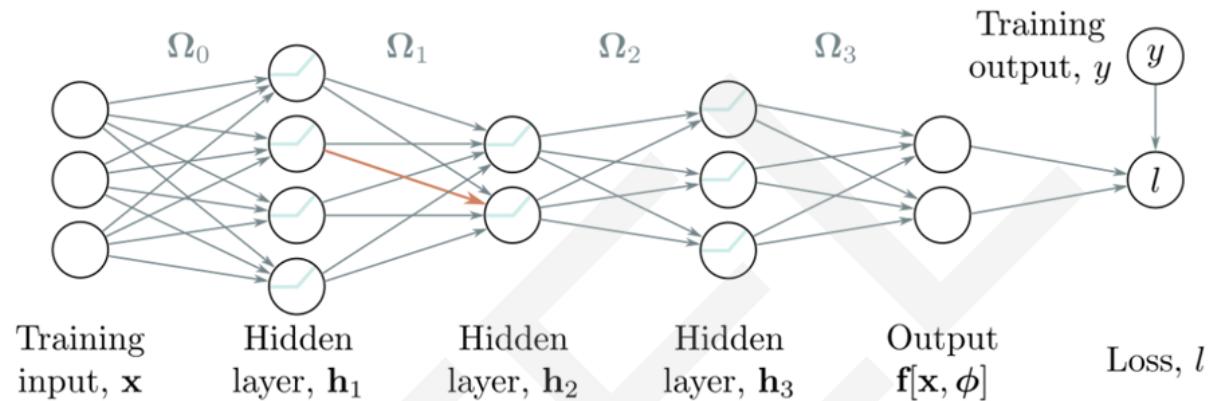
$$\mathbf{f}_2 = \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2$$

$$\mathbf{h}_3 = \mathbf{a}[\mathbf{f}_2]$$

$$\mathbf{f}_3 = \boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3 \mathbf{h}_3$$

$$\ell_i = l[\mathbf{f}_3, y_i]$$

$$\begin{aligned}\frac{\partial \ell_i}{\partial \boldsymbol{\beta}_k} &= \frac{\partial \mathbf{f}_k}{\partial \boldsymbol{\beta}_k} \frac{\partial \ell_i}{\partial \mathbf{f}_k} \\ &= \frac{\partial}{\partial \boldsymbol{\beta}_k} (\boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{h}_k) \frac{\partial \ell_i}{\partial \mathbf{f}_k} \\ &= \frac{\partial \ell_i}{\partial \mathbf{f}_k},\end{aligned}$$



1. Write this as a series of intermediate calculations
2. Compute these intermediate quantities
3. Take derivatives of output with respect to intermediate quantities
4. Take derivatives w.r.t. parameters

$$\mathbf{f}_0 = \boldsymbol{\beta}_0 + \boldsymbol{\Omega}_0 \mathbf{x}_i$$

$$\mathbf{h}_1 = \mathbf{a}[\mathbf{f}_0]$$

$$\mathbf{f}_1 = \boldsymbol{\beta}_1 + \boldsymbol{\Omega}_1 \mathbf{h}_1$$

$$\mathbf{h}_2 = \mathbf{a}[\mathbf{f}_1]$$

$$\mathbf{f}_2 = \boldsymbol{\beta}_2 + \boldsymbol{\Omega}_2 \mathbf{h}_2$$

$$\mathbf{h}_3 = \mathbf{a}[\mathbf{f}_2]$$

$$\mathbf{f}_3 = \boldsymbol{\beta}_3 + \boldsymbol{\Omega}_3 \mathbf{h}_3$$

$$\ell_i = \mathbf{l}[\mathbf{f}_3, y_i]$$

$$\begin{aligned}
 \frac{\partial \ell_i}{\partial \boldsymbol{\Omega}_k} &= \frac{\partial \mathbf{f}_k}{\partial \boldsymbol{\Omega}_k} \frac{\partial \ell_i}{\partial \mathbf{f}_k} \\
 &= \frac{\partial}{\partial \boldsymbol{\Omega}_k} (\boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{h}_k) \frac{\partial \ell_i}{\partial \mathbf{f}_k} \\
 &= \frac{\partial \ell_i}{\partial \mathbf{f}_k} \mathbf{h}_k^T
 \end{aligned}$$

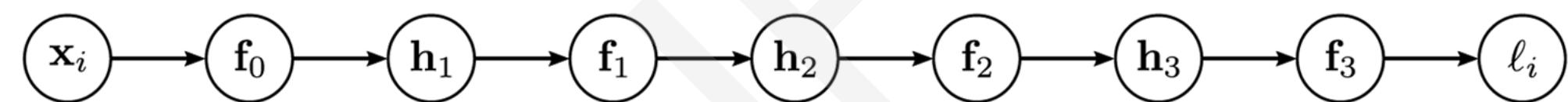
Backprop summary

Forward pass: We compute and store the following quantities:

$$\mathbf{f}_0 = \beta_0 + \Omega_0 \mathbf{x}_i$$

$$\mathbf{h}_k = \mathbf{a}[\mathbf{f}_{k-1}] \quad k \in \{1, 2, \dots, K\}$$

$$\mathbf{f}_k = \beta_k + \Omega_k \mathbf{h}_k. \quad k \in \{1, 2, \dots, K\}$$



Backprop summary

Backward pass: We start with the derivative $\partial\ell_i/\partial\mathbf{f}_K$ of the loss function ℓ_i with respect to the network output \mathbf{f}_K and work backward through the network:

$$\begin{aligned}
 \frac{\partial\ell_i}{\partial\beta_k} &= \frac{\partial\ell_i}{\partial\mathbf{f}_k} & k \in \{K, K-1, \dots, 1\} \\
 \frac{\partial\ell_i}{\partial\Omega_k} &= \frac{\partial\ell_i}{\partial\mathbf{f}_k} \mathbf{h}_k^T & k \in \{K, K-1, \dots, 1\} \\
 \frac{\partial\ell_i}{\partial\mathbf{f}_{k-1}} &= \mathbb{I}[\mathbf{f}_{k-1} > 0] \odot \left(\Omega_k^T \frac{\partial\ell_i}{\partial\mathbf{f}_k} \right), & k \in \{K, K-1, \dots, 1\} \tag{7.13}
 \end{aligned}$$

where \odot denotes pointwise multiplication and $\mathbb{I}[\mathbf{f}_{k-1} > 0]$ is a vector containing ones where \mathbf{f}_{k-1} is greater than zero and zeros elsewhere. Finally, we compute the derivatives with respect to the first set of biases and weights:

Algorithmic differentiation

- Modern deep learning frameworks compute derivatives automatically
- You just have to specify the model and the loss
- How? **Algorithmic differentiation**
 - 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 how the order of the components
 - It can compute the chain of derivatives
- Works with branches as long as it's still an acyclic graph

REFERENCES

Understanding Deep Learning [<https://udlbook.github.io/udlbook/>]
[Chapter 7]



THANK YOU



N P T E L O N L I N E C E R T I F I C A T I O N C O U R S E S

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 10 : Gradient Descent and Initialization



PROF . PAWAN GOYAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Initialization
- Gradient Descent, SGD, Momentum, Adam

Initialization

- Consider standard building block of NN in terms of *preactivations*:

$$\begin{aligned}\mathbf{f}_k &= \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{h}_k \\ &= \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{a}[\mathbf{f}_{k-1}]\end{aligned}$$

- Set all the biases to 0

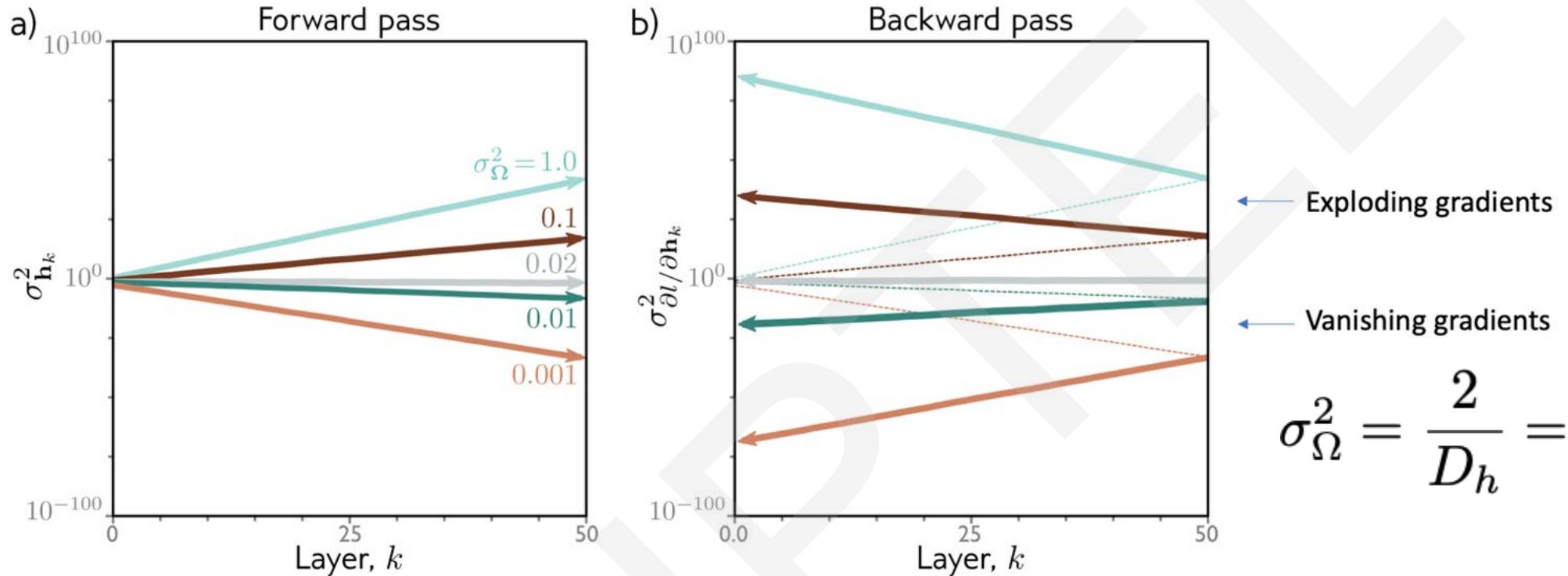
$$\boldsymbol{\beta}_k = \mathbf{0}$$

- Weights normally distributed

- mean 0
- variance σ_{Ω}^2

- What will happen as we move through the network if σ_{Ω}^2 is very small?
- What will happen as we move through the network if σ_{Ω}^2 is very large?

Initialization



$$\sigma_\Omega^2 = \frac{2}{D_h} = \frac{2}{100} = 0.02$$

Figure 7.4 Weight initialization. Consider a deep network with 50 hidden layers and $D_h = 100$ hidden units per layer. The network has a 100 dimensional input \mathbf{x} initialized with values from a standard normal distribution, a single output fixed at $y = 0$, and a least squares loss function. The bias vectors β_k are initialized to zero and the weight matrices Ω_k are initialized with a normal distribution with mean zero and five different variances $\sigma_\Omega^2 \in \{0.001, 0.01, 0.02, 0.1, 1.0\}$. a)

*Slides adapted from <http://udlbook.com>

Aim: keep variance same between two layers

$$\mathbf{f}' = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{h}$$

Consider the mean of the pre-activations:

$$\mathbb{E}[f'_i] = \mathbb{E} \left[\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j \right]$$

Aim: keep variance same between two layers

$$\mathbb{E}[f'_i] = \mathbb{E} \left[\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j \right]$$

$$= \mathbb{E} [\beta_i] + \sum_{j=1}^{D_h} \mathbb{E} [\Omega_{ij} h_j]$$

$$= \mathbb{E} [\beta_i] + \sum_{j=1}^{D_h} \mathbb{E} [\Omega_{ij}] \mathbb{E} [h_j]$$

$$= 0 + \sum_{j=1}^{D_h} 0 \cdot \mathbb{E} [h_j] = 0$$

Set all the biases to 0

Weights normally distributed

mean 0

variance σ_Ω^2

Aim: keep variance same between two layers

Set all the biases to 0

Weights normally distributed
mean 0
variance σ_{Ω}^2

$$\begin{aligned}
 \sigma_{f'}^2 &= \mathbb{E}[f_i'^2] - \mathbb{E}[f_i']^2 \\
 &= \mathbb{E} \left[\left(\beta_i + \sum_{j=1}^{D_h} \Omega_{ij} h_j \right)^2 \right] - 0 \\
 &= \mathbb{E} \left[\left(\sum_{j=1}^{D_h} \Omega_{ij} h_j \right)^2 \right] \\
 &= \sum_{j=1}^{D_h} \mathbb{E} [\Omega_{ij}^2] \mathbb{E} [h_j^2] \\
 &= \sum_{i=1}^{D_h} \sigma_{\Omega}^2 \mathbb{E} [h_j^2] = \sigma_{\Omega}^2 \sum_{i=1}^{D_h} \mathbb{E} [h_j^2]
 \end{aligned}$$

Aim: keep variance same between two layers

$$\sigma_{f'}^2 = \sigma_\Omega^2 \sum_{j=1}^{D_h} \mathbb{E} [h_j^2]$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \mathbb{E} [\text{ReLU}[f_j]^2]$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \int_{-\infty}^{\infty} \text{ReLU}[f_j]^2 Pr(f_j) df_j$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \int_{-\infty}^{\infty} (\mathbb{I}[f_j > 0] f_j)^2 Pr(f_j) df_j$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \int_0^{\infty} f_j^2 Pr(f_j) df_j$$

$$= \sigma_\Omega^2 \sum_{j=1}^{D_h} \frac{\sigma_f^2}{2} = \frac{D_h \sigma_\Omega^2 \sigma_f^2}{2}$$

Aim: keep variance same between two layers

$$\sigma_{f'}^2 = \frac{D_h \sigma_\Omega^2 \sigma_f^2}{2}$$

Should choose:

$$\sigma_\Omega^2 = \frac{2}{D_h}$$

This is called **He initialization**.

*Slides adapted from <http://udlbook.com>

Gradient Descent

Gradient descent algorithm

Step 1. Compute the derivatives of the loss with respect to the parameters:

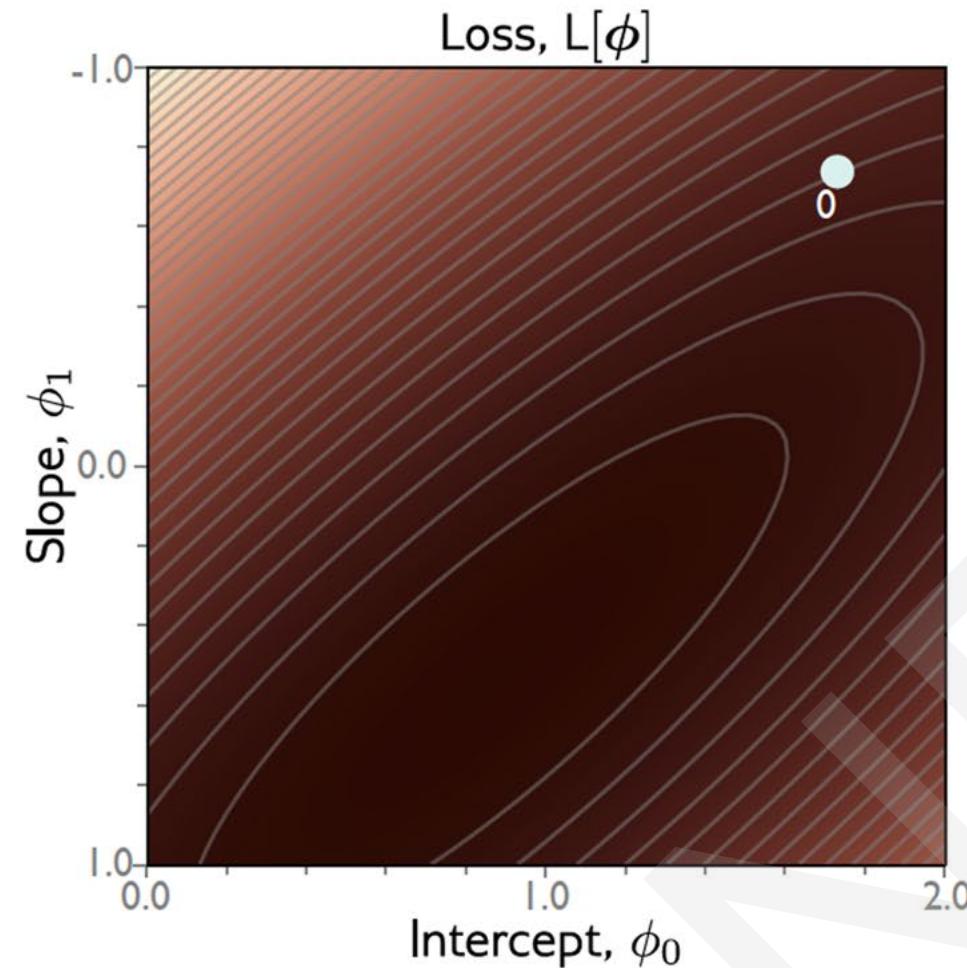
$$\frac{\partial L}{\partial \phi} = \begin{bmatrix} \frac{\partial L}{\partial \phi_0} \\ \frac{\partial L}{\partial \phi_1} \\ \vdots \\ \frac{\partial L}{\partial \phi_N} \end{bmatrix}.$$

Step 2. Update the parameters according to the rule:

$$\phi \leftarrow \phi - \alpha \frac{\partial L}{\partial \phi},$$

where the positive scalar α determines the magnitude of the change.

Gradient descent: Linear Regression



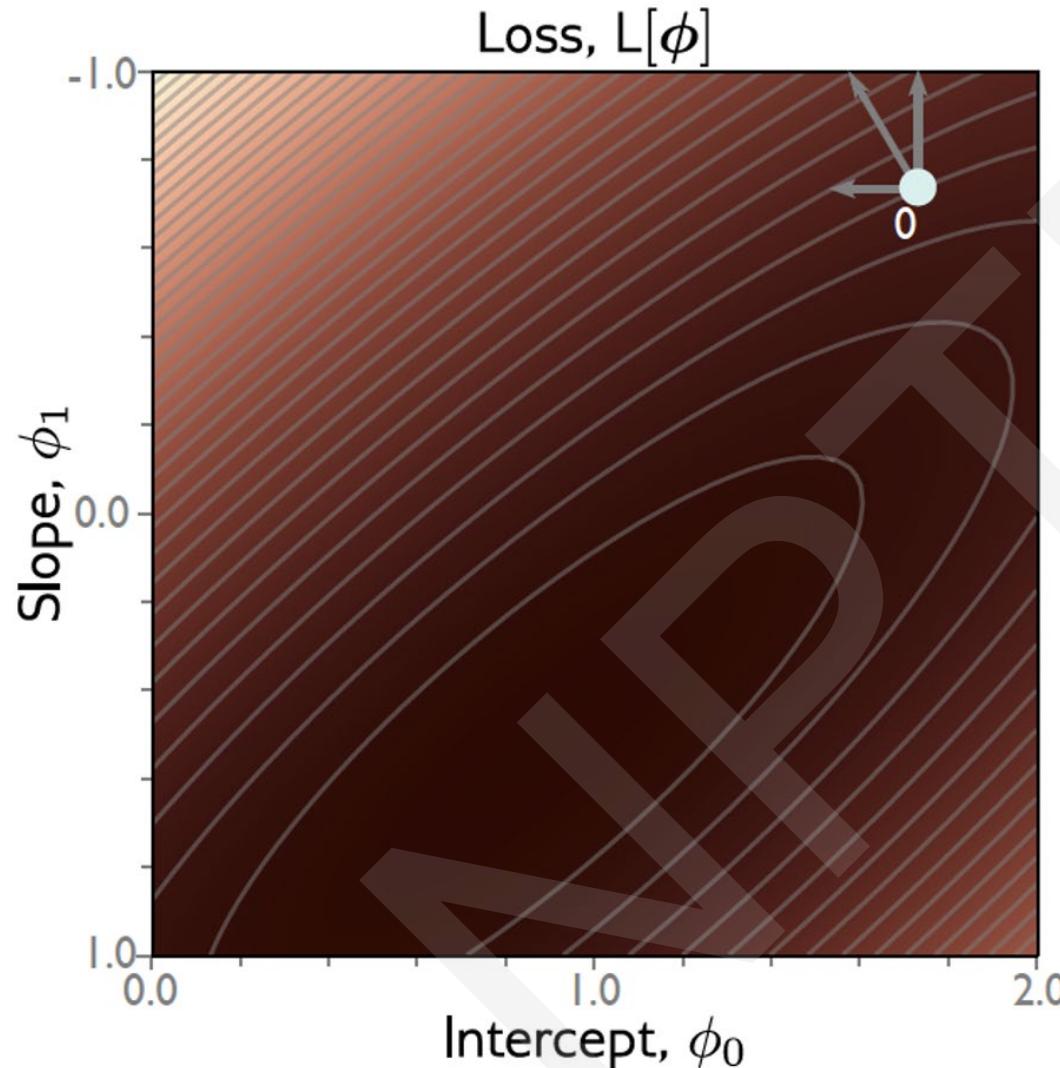
Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

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

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

Gradient descent: Linear Regression

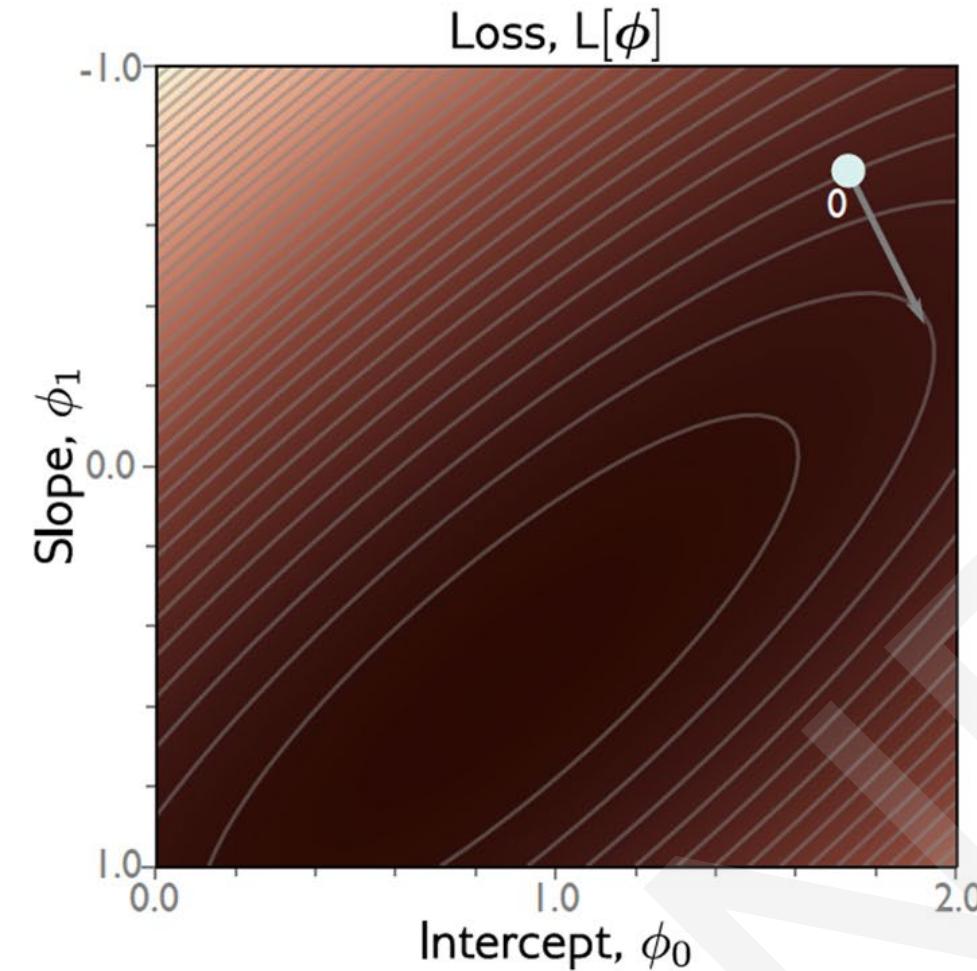


Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

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

Gradient descent: Linear Regression



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

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

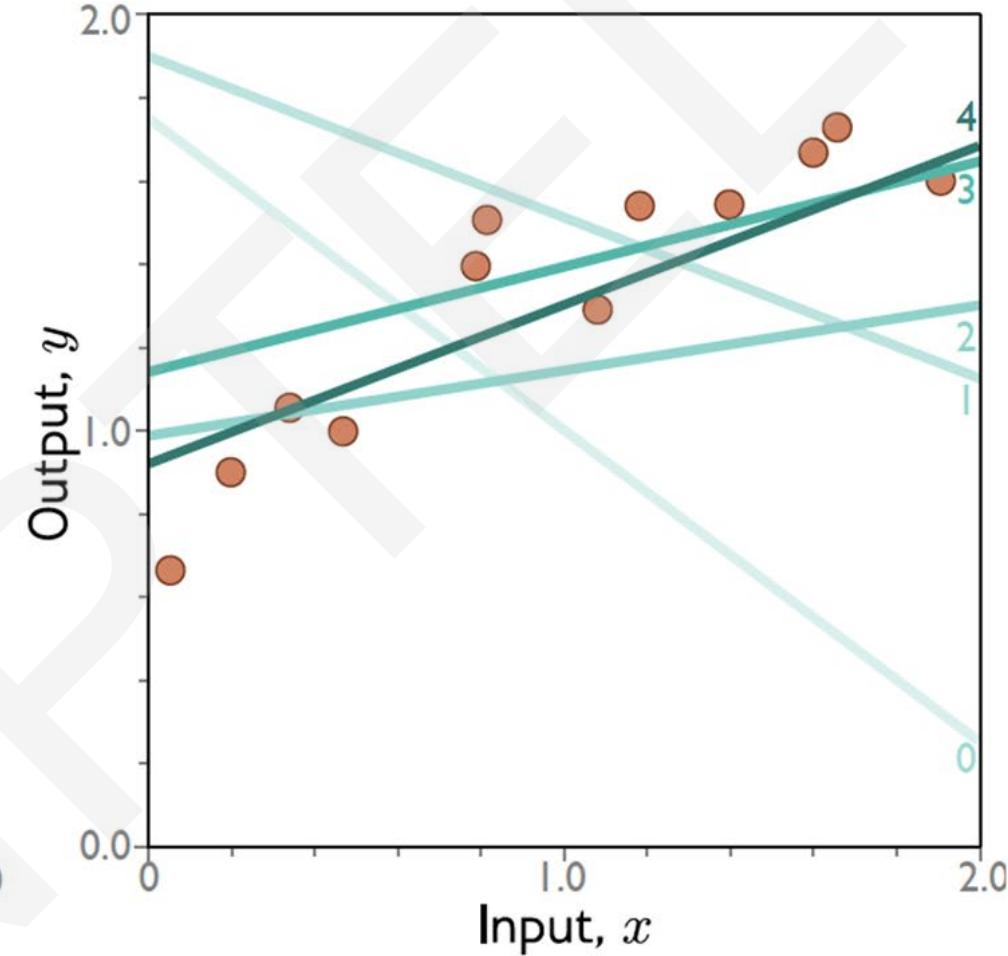
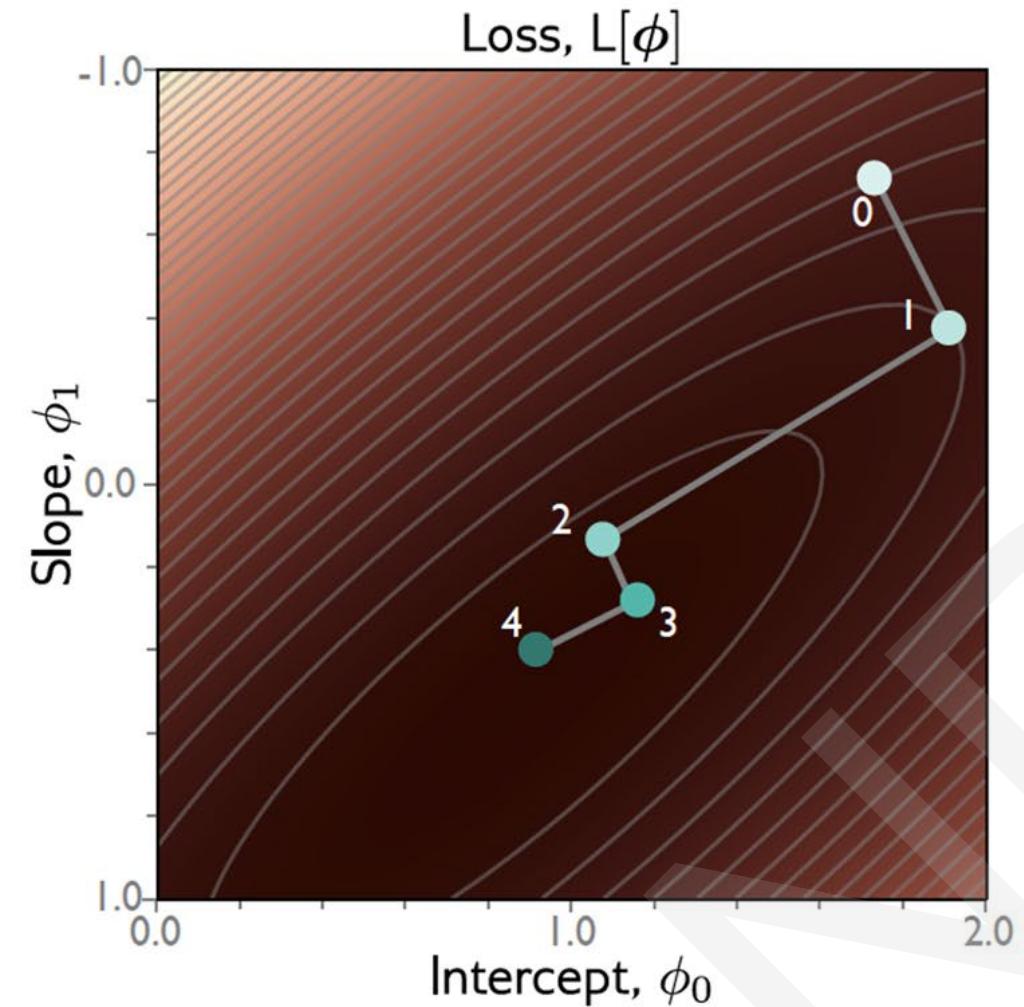
$$\frac{\partial \ell_i}{\partial \phi} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

$$\phi \leftarrow \phi - \alpha \frac{\partial L}{\partial \phi}$$

α = step size

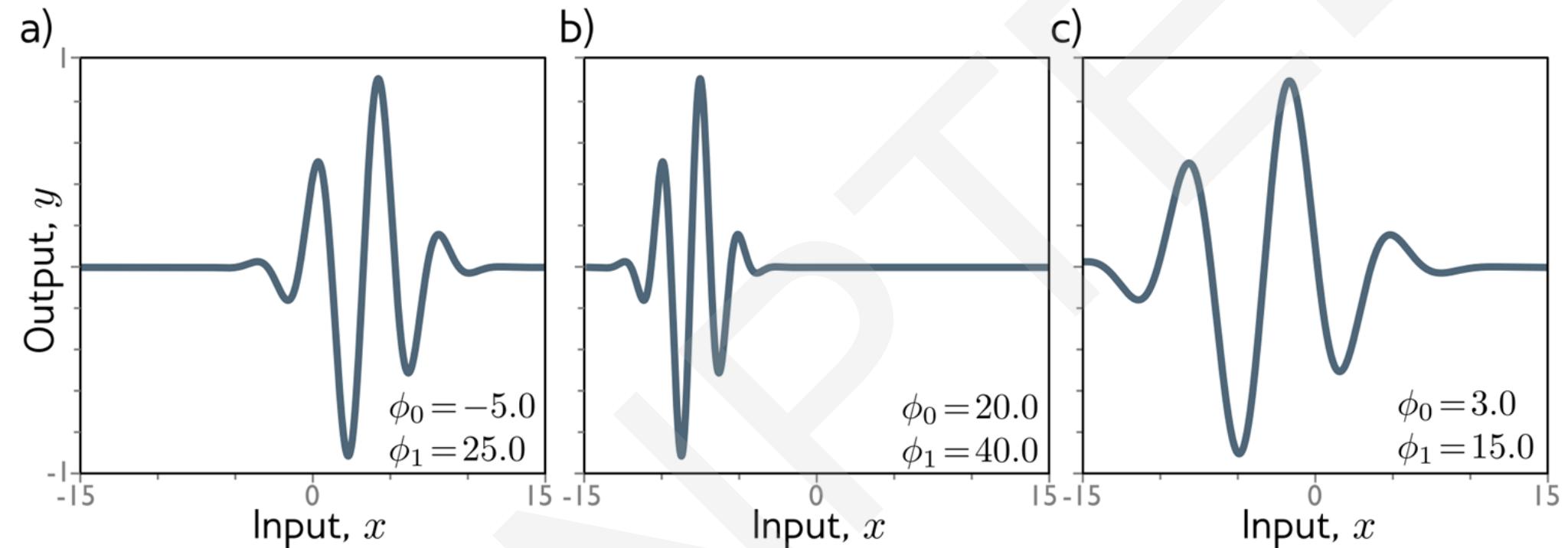
Gradient descent: Linear Regression



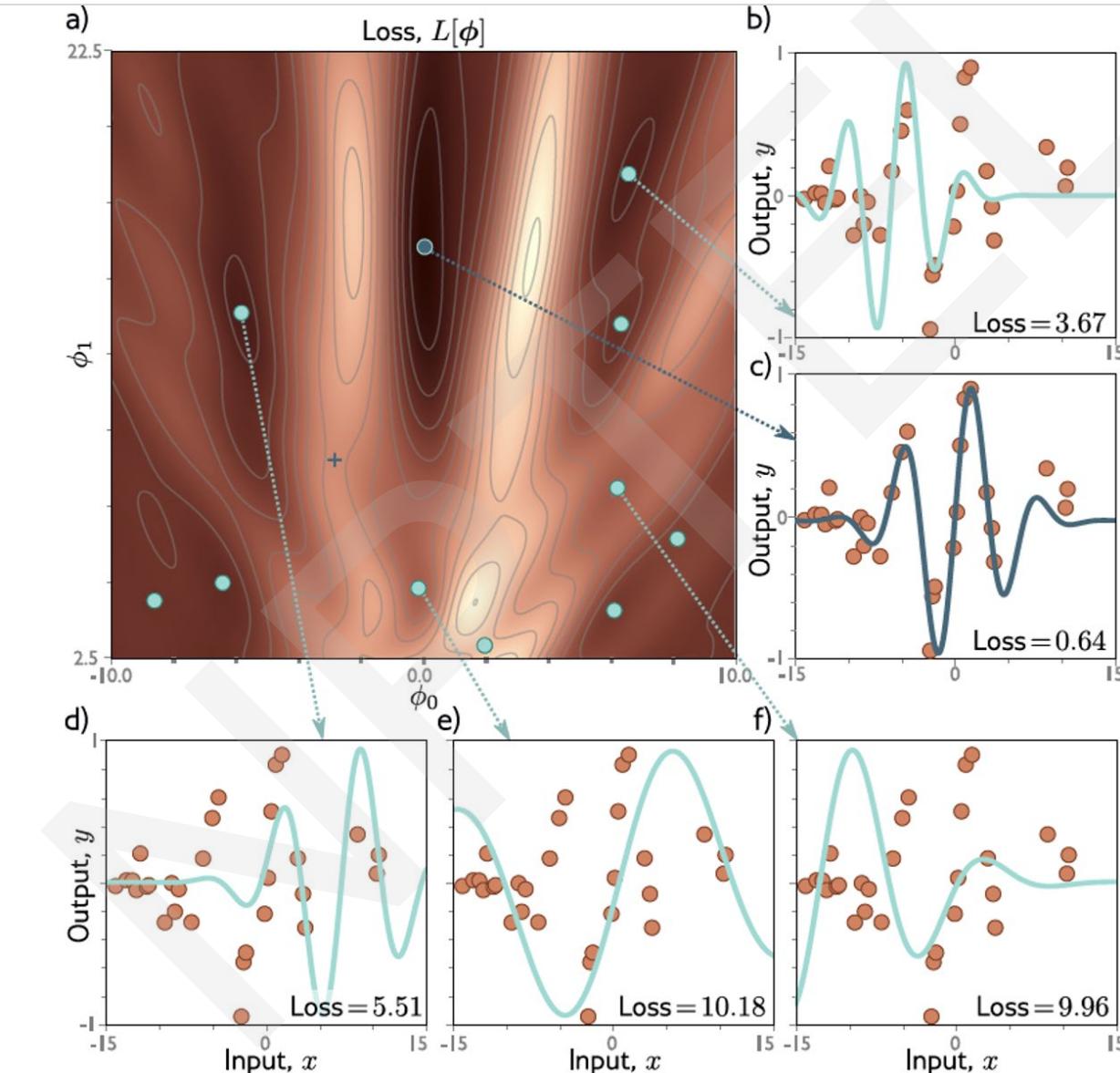
*Slides adapted from <http://udlbook.com>

Gradient descent: Gabor Model

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

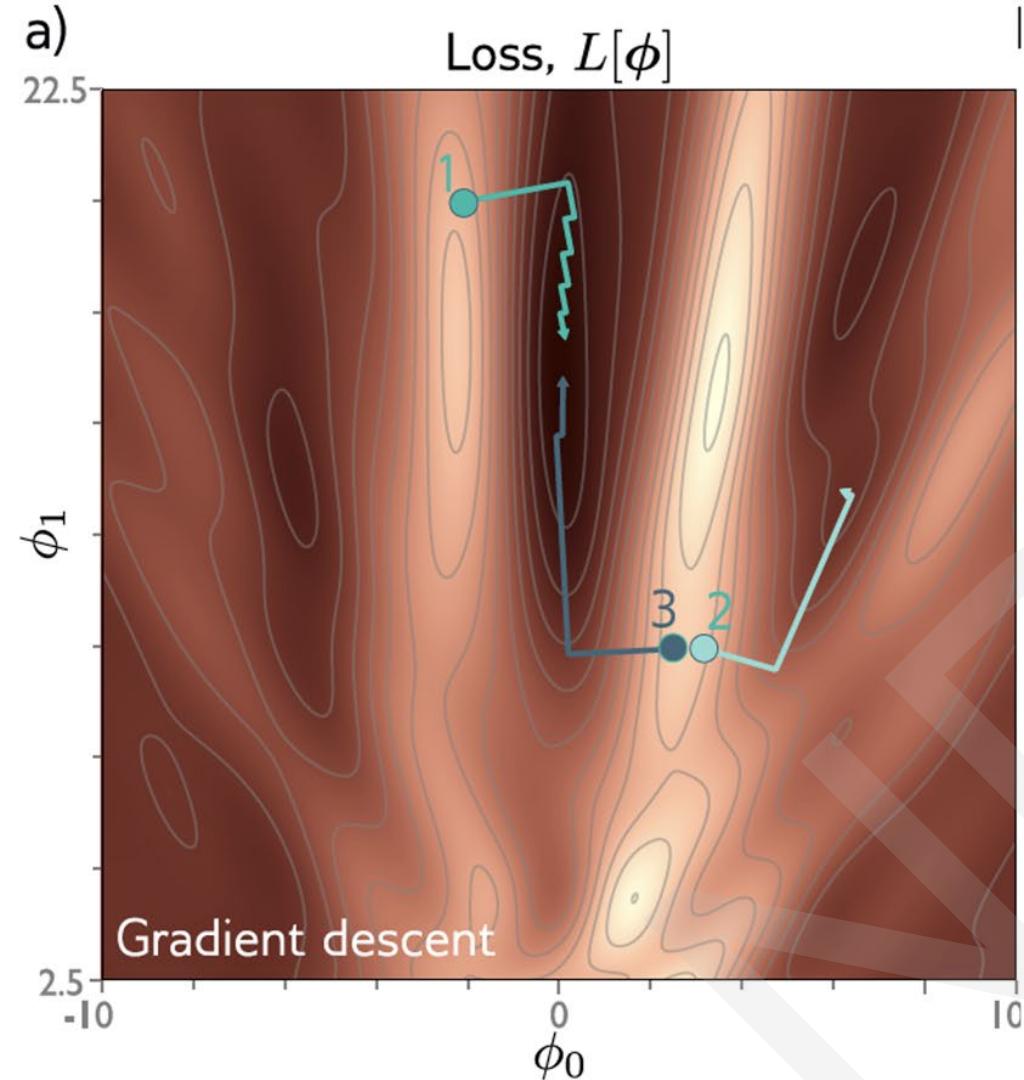


Gradient descent: Gabor Model



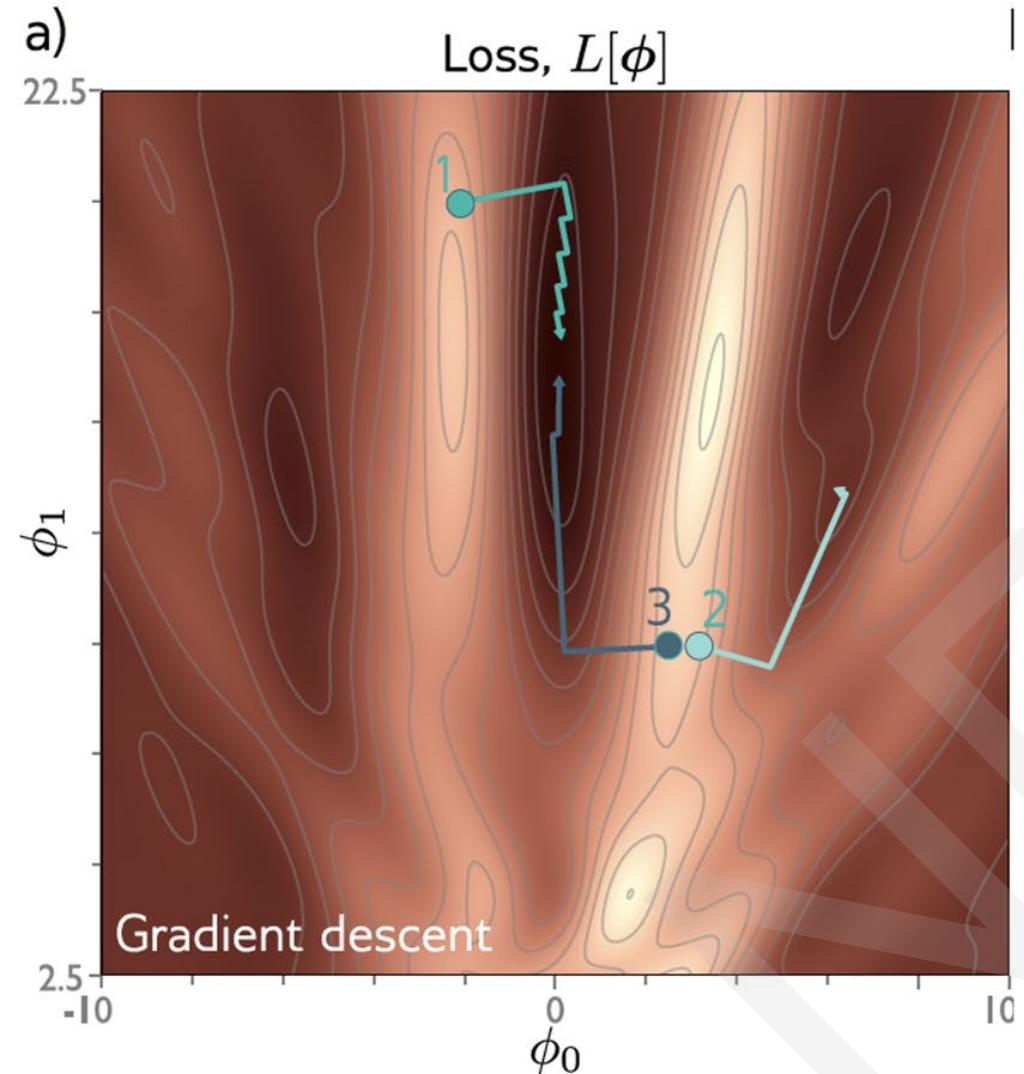
*Slides adapted from
<http://udlbook.com>

Gradient descent: Issues



- 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

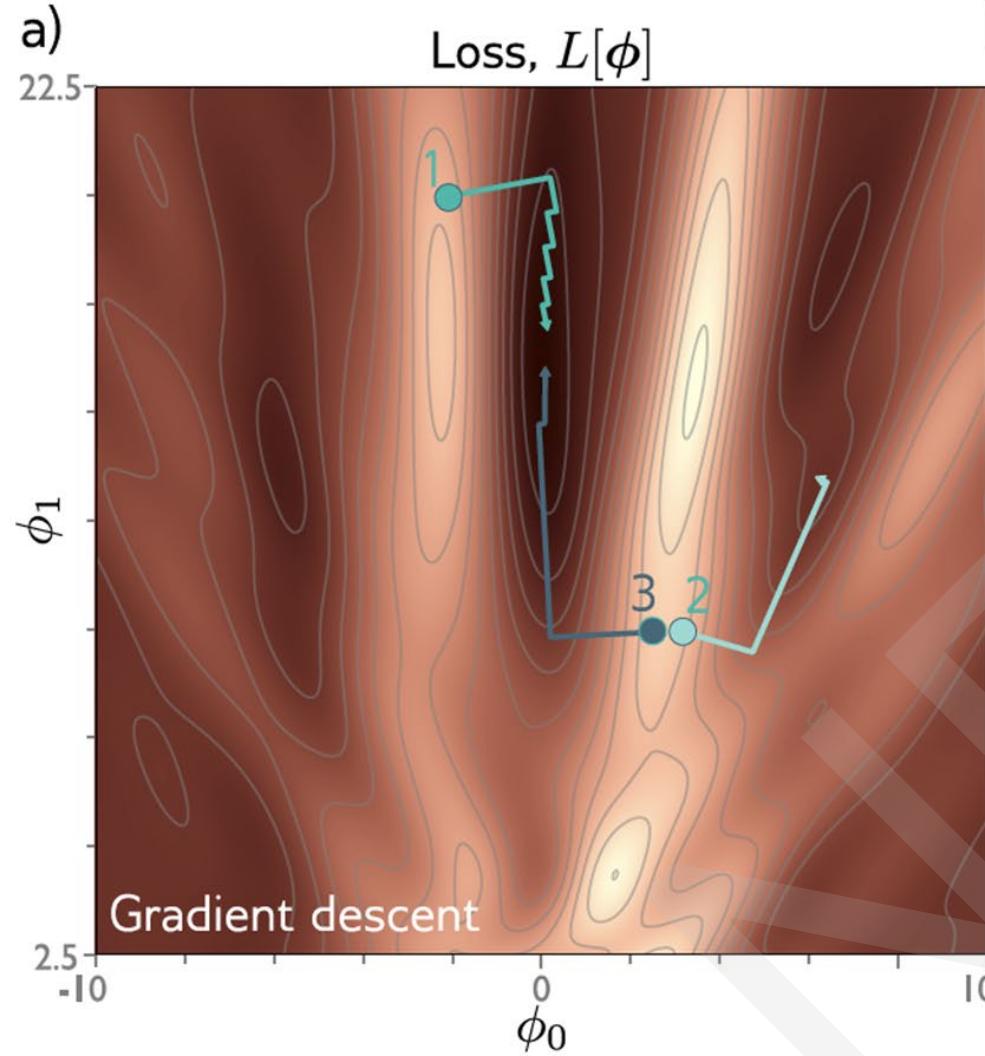
Stochastic Gradient Descent (SGD)



IDEA: 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 (SGD)



Stochastic gradient descent

Before (full batch descent)

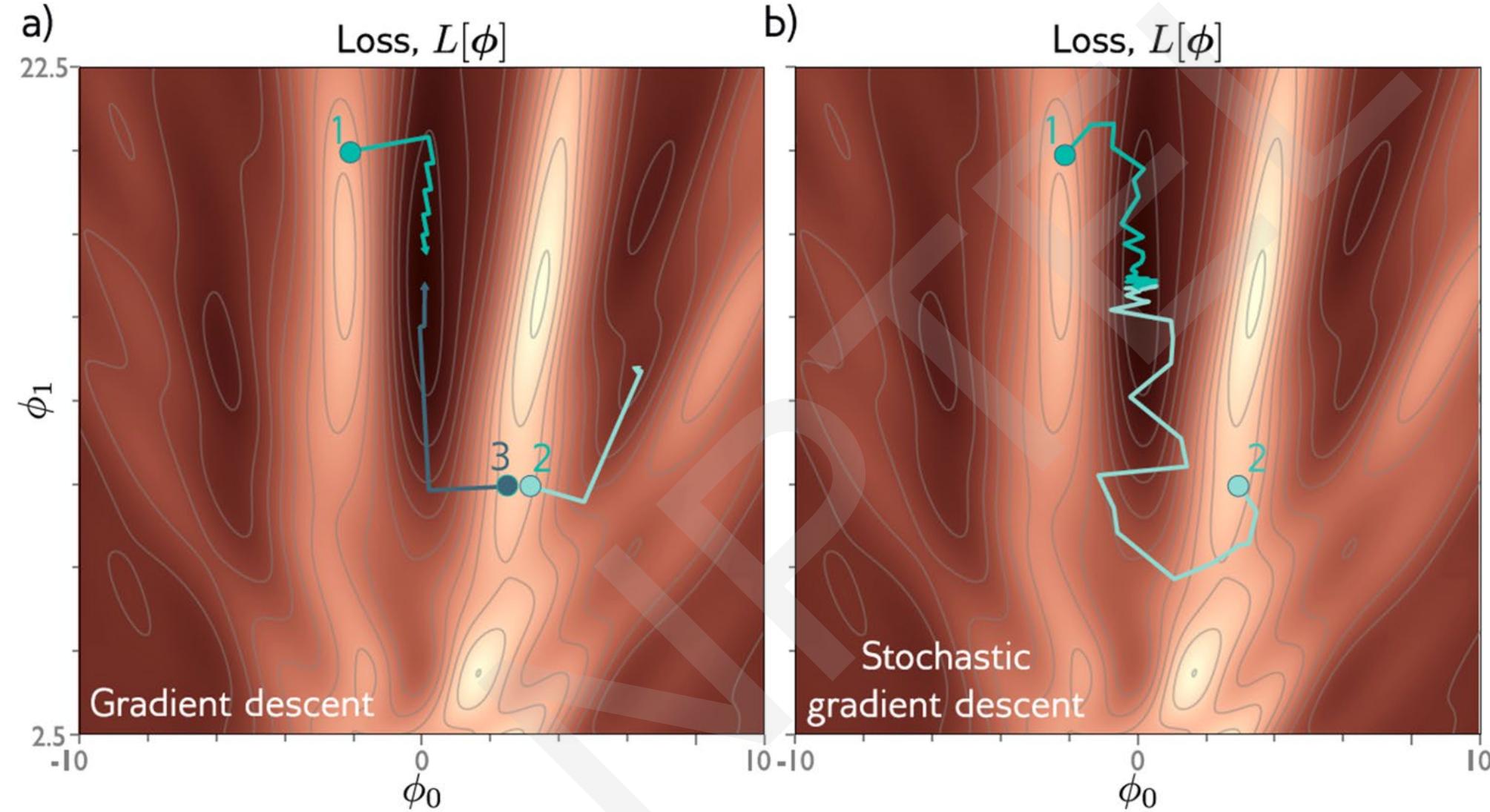
$$\phi_{t+1} \leftarrow \phi_t - \alpha \sum_{i=1}^I \frac{\partial \ell_i[\phi_t]}{\partial \phi},$$

After (SGD)

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

Fixed learning rate α

Stochastic Gradient Descent (SGD)



SGD can, in principle, escape local minima.

Adding Momentum

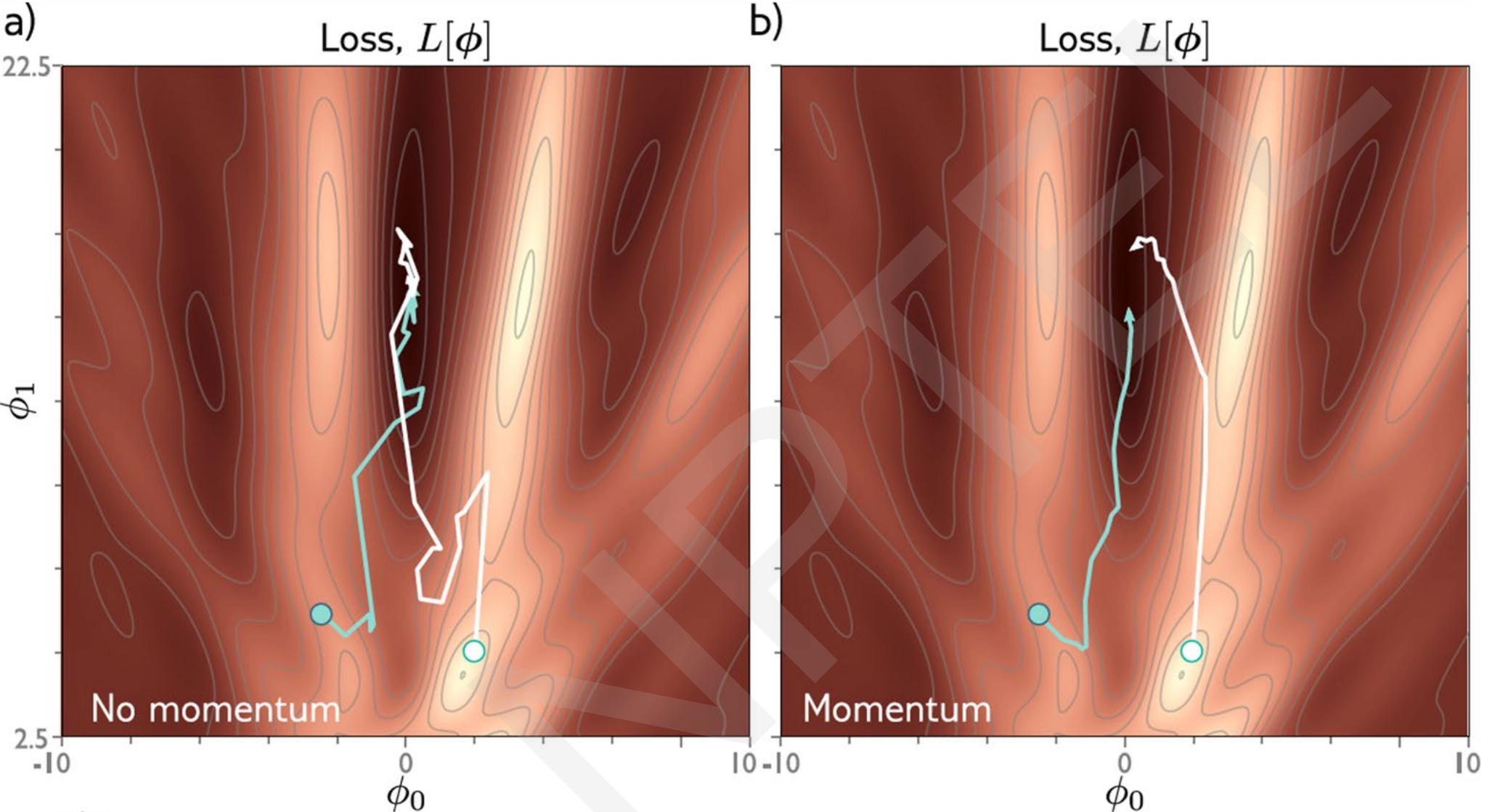
- Weighted sum of this gradient and previous gradient

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

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \mathbf{m}_{t+1}$$

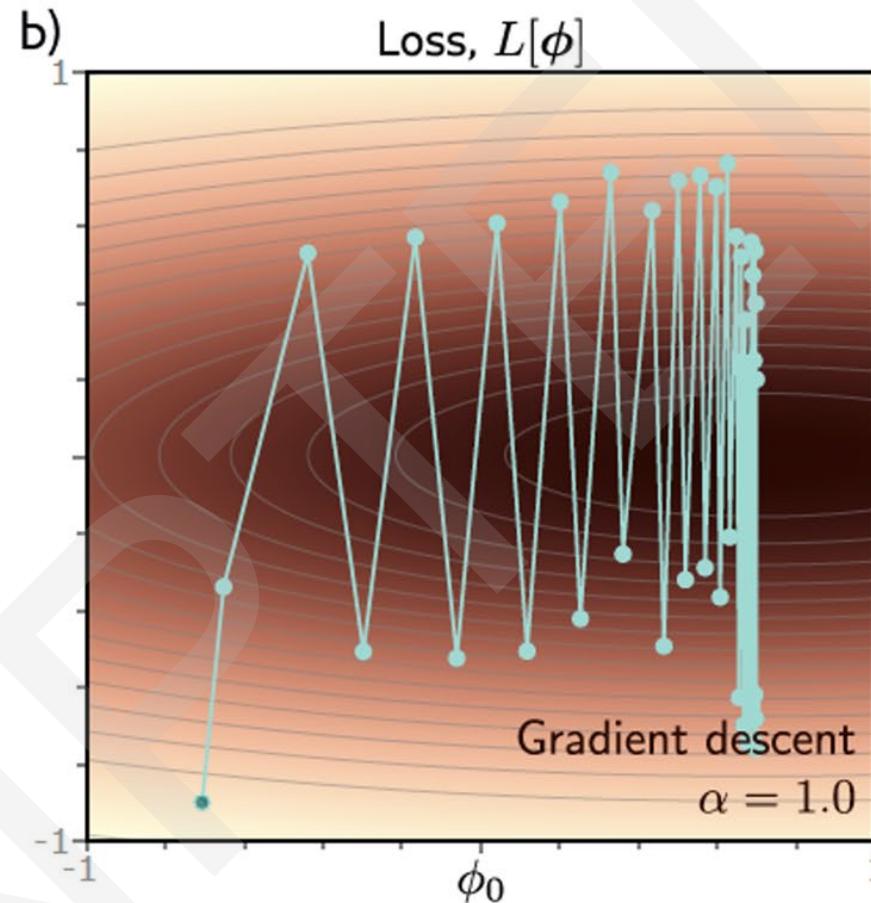
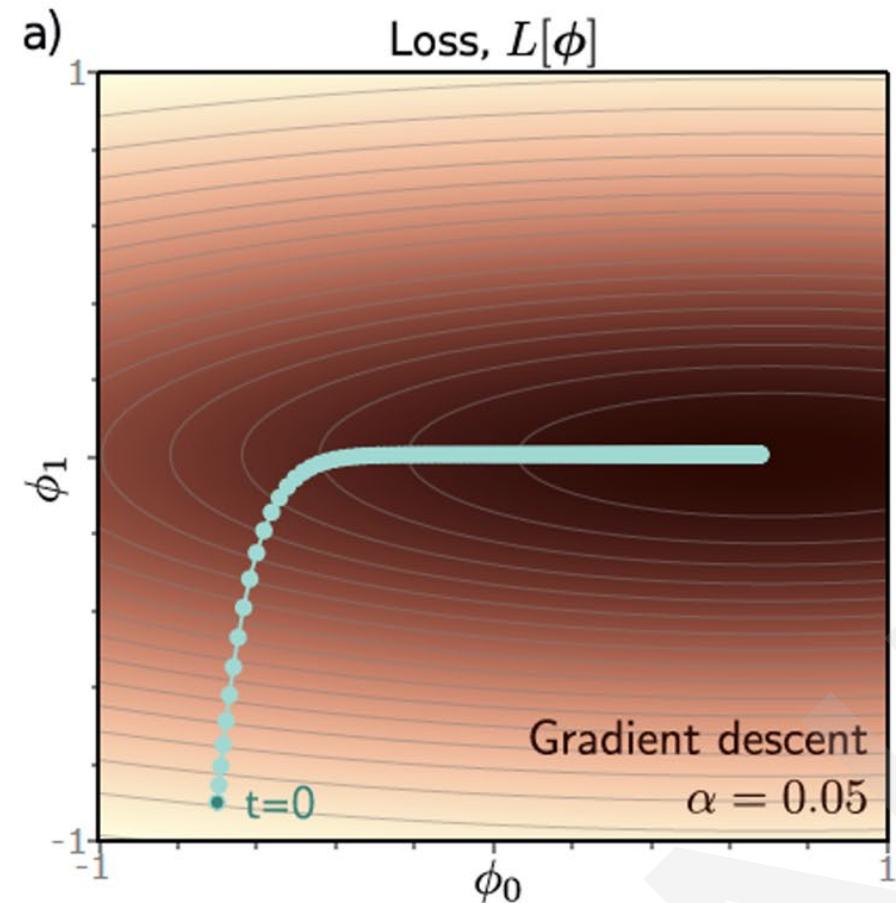
- A common modification to SGD is to add momentum term.
- The recursive formulation means that the gradient is an infinite weighted sum of all the previous gradients, with weights getting smaller as we move back in time.

Adding Momentum



*Slides adapted from <http://udlbook.com>

Adaptive Moment Estimation



With a fixed learning rate, there are large adjustments to parameters with large gradients (gradient much steeper in one direction than another, making it difficult to choose a good learning rate that makes good progress, and is stable)

What can be done? Can we normalize the gradients?

Normalized Gradients

- Measure mean and pointwise squared gradient

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

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

- Normalize:

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}$$

$$\mathbf{m}_{t+1} = \begin{bmatrix} 3.0 \\ -2.0 \\ 5.0 \end{bmatrix}$$

$$\mathbf{v}_{t+1} = \begin{bmatrix} 9.0 \\ 4.0 \\ 25.0 \end{bmatrix}$$

$$\frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon} = \begin{bmatrix} 1.0 \\ -1.0 \\ 1.0 \end{bmatrix}$$

Normalized Gradients

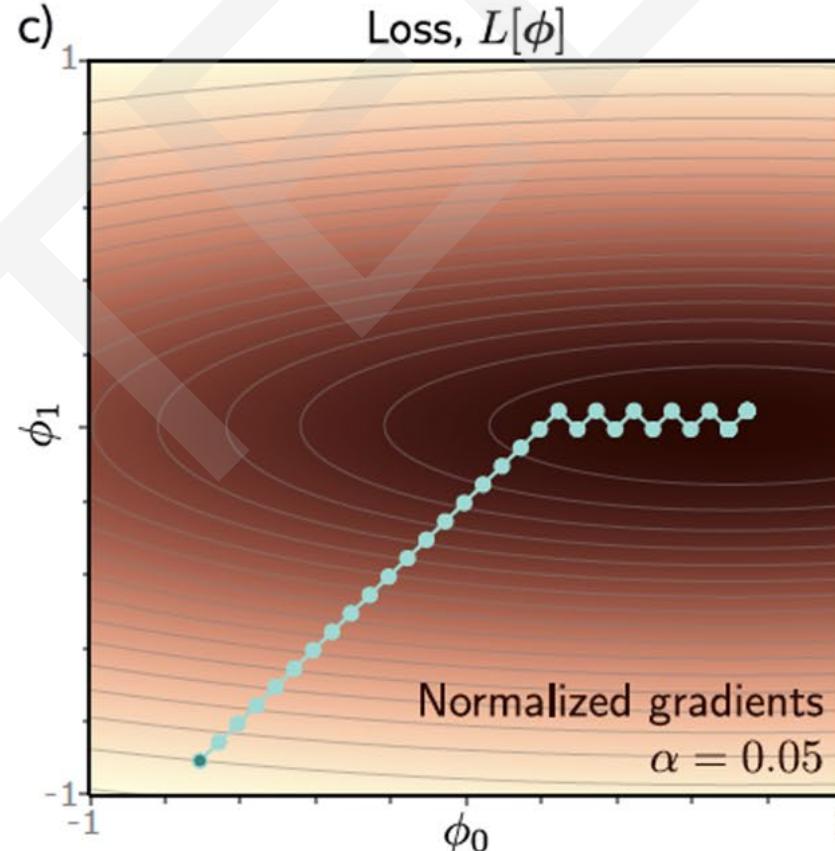
- Measure mean and pointwise squared gradient

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

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

- Normalize:

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\mathbf{m}_{t+1}}{\sqrt{\mathbf{v}_{t+1}} + \epsilon}$$



Problem with this approach: Makes good progress, but will not converge

Adaptive moment estimation (Adam)

- Compute mean and pointwise squared gradients with momentum

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

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

- Moderate near start of the sequence

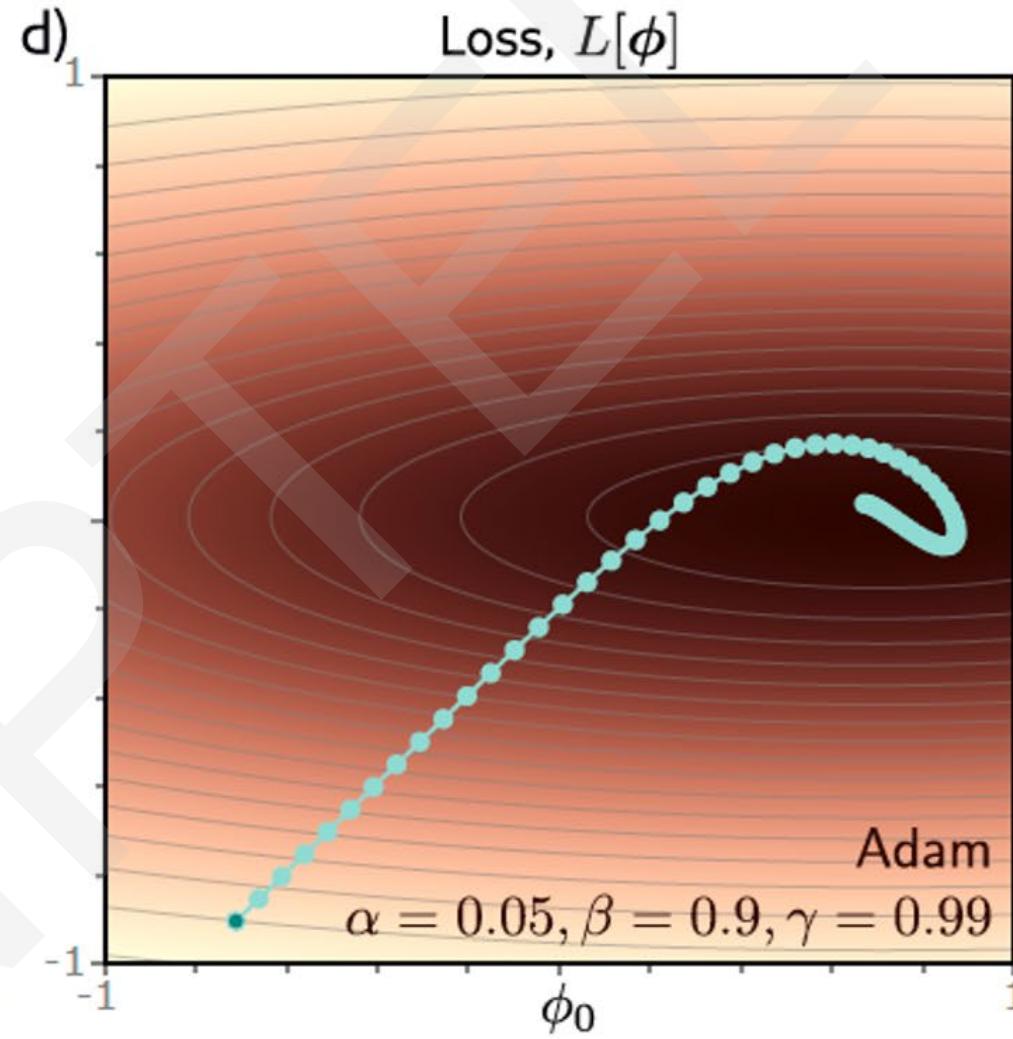
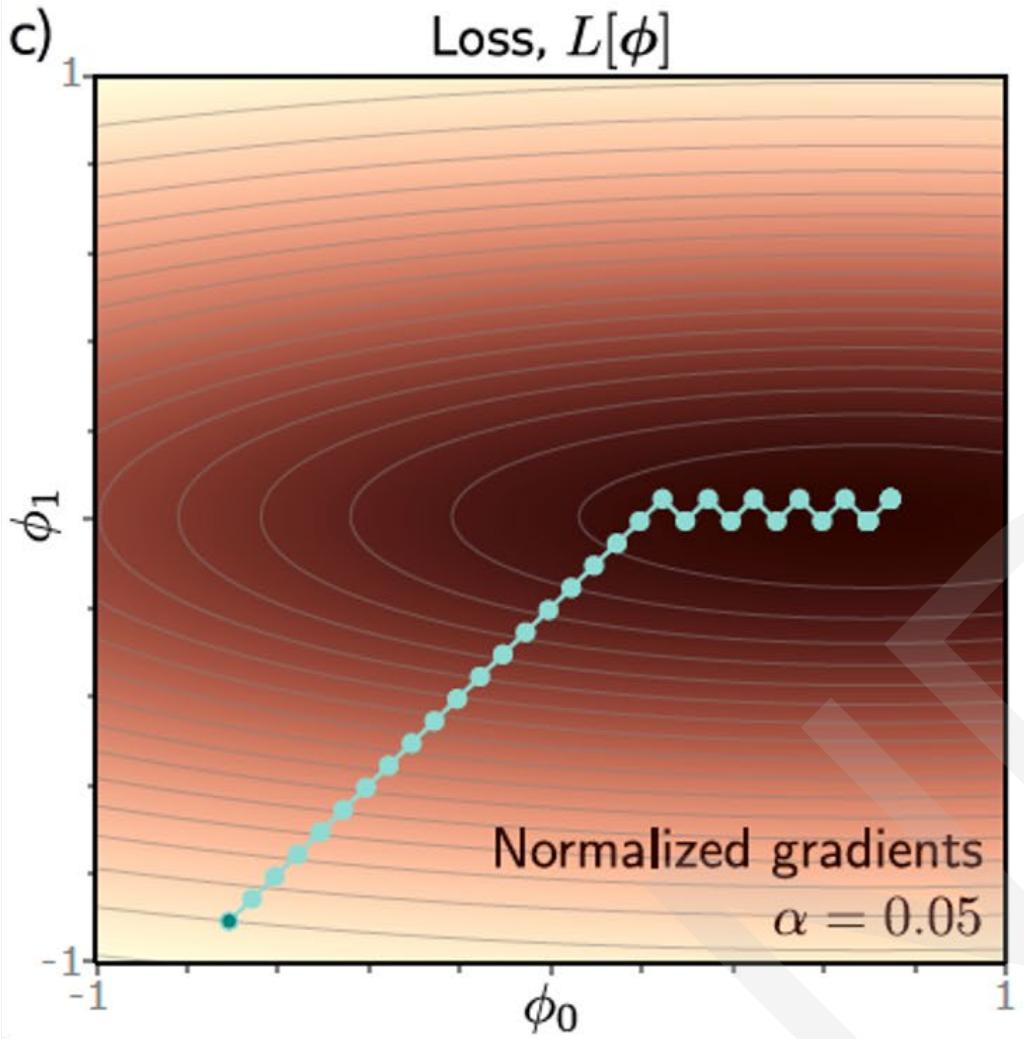
$$\tilde{\mathbf{m}}_{t+1} \leftarrow \frac{\mathbf{m}_{t+1}}{1 - \beta^{t+1}}$$

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

- Update the parameters

$$\phi_{t+1} \leftarrow \phi_t - \alpha \cdot \frac{\tilde{\mathbf{m}}_{t+1}}{\sqrt{\tilde{\mathbf{v}}_{t+1}} + \epsilon}$$

Adaptive moment estimation (Adam)



Training algorithm hyperparameters

- Choice of learning algorithm, batch size, learning rate schedule, momentum coefficients are all hyper-parameters of the training algorithm
- Common to train many models with different hyper-parameters and choose the best one
- Known as *hyper-parameter search*

REFERENCES

Understanding Deep Learning [<https://udlbook.github.io/udlbook/>]
[Chapter 6, 7]



THANK YOU