Advantages of using Batch Normalization in Deep Learning

Kamil Krzyk

Data Scientist at:

AZIMO

Contact:

krzyk.kamil@gmail.com

@F1sherKK

AzimoLabs

FisherKK/F1sherKK-MyRoadToAI

We are talking about 3 years old technique
yet still actively used today.

We are talking about 3 years old technique yet still actively used today.

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so

Terminology Introduction

Terminology Introduction

Batch

Terminology Introduction

Batch Normalization

Terminology Introduction

Batch Normalization

What is Batch?

Training Neural Network

- It's a Supervised Learning Algorithm.
- It requires Training Data Set – group of examples based on which it can learn.

Training Neural Network

- It's a Supervised Learning Algorithm.
- It requires Training Data Set – group of examples based on which it can learn.

Data type examples:

Training Neural Network

- It's a Supervised Learning Algorithm.
- It requires Training Data Set – group of examples based on which it can learn.

Data type examples:

Images

Training Neural Network

- It's a Supervised Learning Algorithm.
- It requires Training Data Set – group of examples based on which it can learn.

Data type examples:

Images

Raw Numerical Data
(e.g. describing users)

Training Neural Network

- It's a Supervised Learning Algorithm.
- It requires Training Data Set – group of examples based on which it can learn.

Data type examples:

Images

Raw Numerical Data
(e.g. describing users)

Text

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

0	0	0	0	0	0
0	255	0	0	255	0
0	255	0	0	255	0
0	255	255	255	255	0
0	0	0	0	255	0
0	0	0	0	255	0

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

start of vector

0	0	0	0	0	0	0	255	0	0	255	0	0	255	0	0	255	0

0	255	255	255	255	0	0	0	0	0	255	0	0	0	0	0	255	0

end of vector

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

start of vector

end of vector

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

start of vector

end of vector

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

start of vector

[0, 0, 0, 0, 0, 0, 0, 255, 0, 0, 255, . . . 0, 0, 0, 255, 0, 0, 0, 0, 0, 255, 0]

end of vector

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

start of vector

end of vector

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

start of vector

end of vector

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.

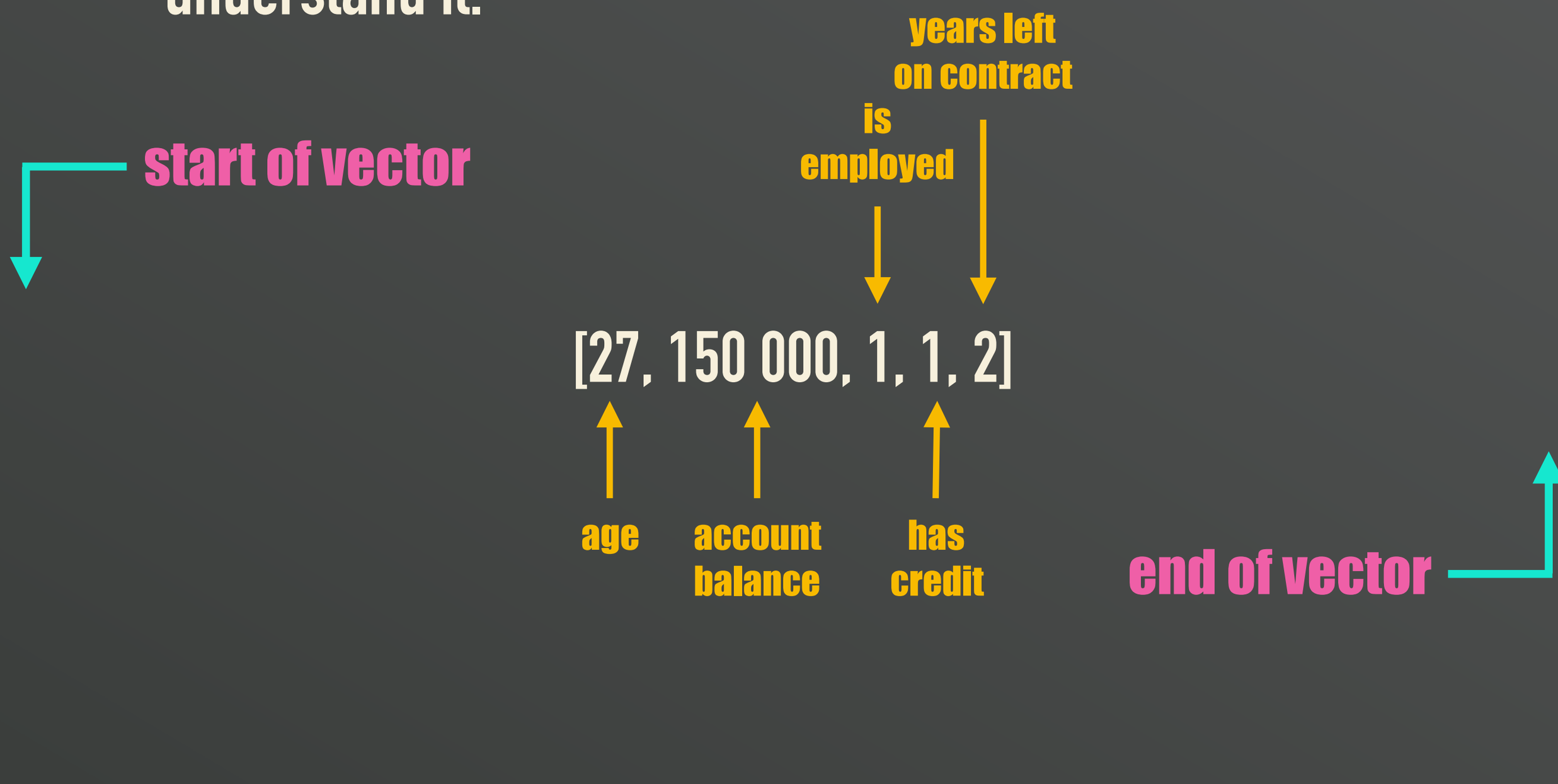
start of vector

[27, 150 000, 1, 1, 2]

end of vector

Training Neural Network

- Data needs to be in numerical form so algorithm can understand it.



Training Neural Network

- We usually work with thousands or even millions of data samples. Each of them is turned into a vector (some exceptions).

Training Neural Network

- We usually work with thousands or even millions of data samples. Each of them is turned into a vector (some exceptions).

[number, number, . . . number]
[number, number, . . . number]
[number, number, . . . number]
[number, number, . . . number]
[number, number, . . . number]
. . .
[number, number, . . . number]

Training Neural Network

- We usually work with thousands or even millions of data samples. Each of them is turned into a vector (some exceptions).

matrix

X:

[number, number, . . . number]
[number, number, . . . number]
[number, number, . . . number]
[number, number, . . . number]
[number, number, . . . number]
. . .
[number, number, . . . number]

Training Neural Network

- We usually work with thousands or even millions of data samples. Each of them is turned into a vector (some exceptions).

matrix

X:

sample 0: [number, number, . . . number]
sample 1: [number, number, . . . number]
sample 2: [number, number, . . . number]
sample 3: [number, number, . . . number]
sample 4: [number, number, . . . number]
. . .
sample m: [number, number, . . . number]

rows

Training Neural Network

- We usually work with thousands or even millions of data samples. Each of them is turned into a vector (some exceptions).

matrix X:

	feature 0:	feature 1:		feature n:
sample 0:	[number,	number,	. . .	number]
sample 1:	[number,	number,	. . .	number]
sample 2:	[number,	number,	. . .	number]
sample 3:	[number,	number,	. . .	number]
sample 4:	[number,	number,	. . .	number]
		. . .		
sample m:	[number,	number,	. . .	number]

cols

rows

Training Neural Network

- We usually work with thousands or even millions of data samples. Each of them is turned into a vector (some exceptions).

matrix X:

	feature 0:	feature 1:	…	feature n:	cols
sample 0:	[number,	number,	…	number]	[expected result]
sample 1:	[number,	number,	…	number]	[expected result]
sample 2:	[number,	number,	…	number]	[expected result]
sample 3:	[number,	number,	…	number]	[expected result]
sample 4:	[number,	number,	…	number]	[expected result]
		…			…
sample m:	[number,	number,	…	number]	[expected result]

rows

Training Neural Network

- We usually work with thousands or even millions of data samples. Each of them is turned into a vector (some exceptions).

matrix **X:**

	feature 0:	feature 1:	…	feature n:
sample 0:	[number,	number,	…	number]
sample 1:	[number,	number,	…	number]
sample 2:	[number,	number,	…	number]
sample 3:	[number,	number,	…	number]
sample 4:	[number,	number,	…	number]
…				
sample m:	[number,	number,	…	number]

rows

vector **y:**

cols
[expected result]
[expected result]
[expected result]
[expected result]
[expected result]
…
[expected result]

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Result 0

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Expected Result 0

Result 0

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

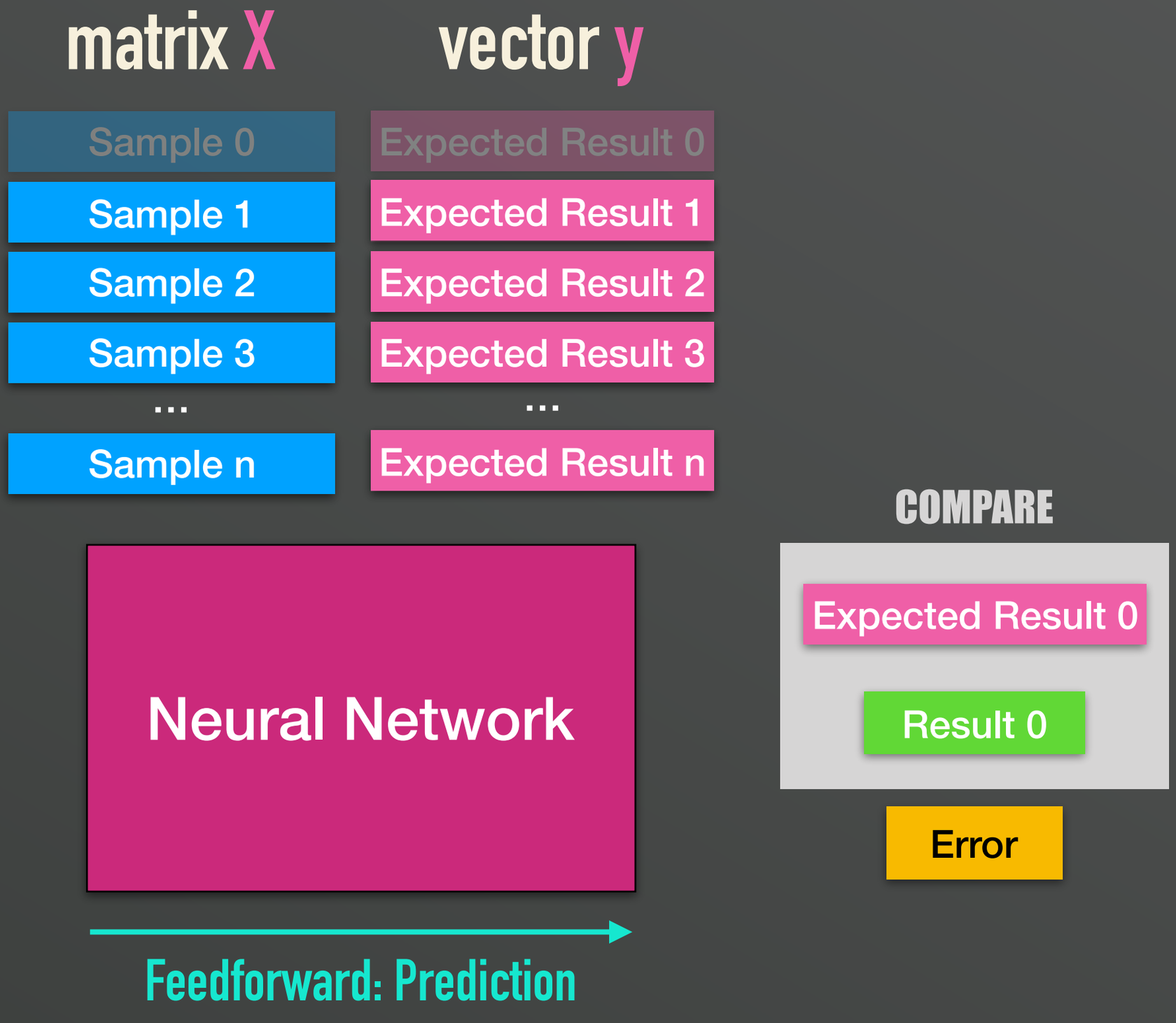
Feedforward: Prediction

COMPARE

Expected Result 0

Result 0

Training Neural Network



Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 0

Result 0

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

COMPARE

Expected Result 0

Result 0

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

COMPARE

Expected Result 0

Result 0

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

COMPARE

Expected Result 0

Result 0

Backpropagation: Updating Model Parameters

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Result 1

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Expected Result 1

Result 1

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 1

Result 1

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 1

Result 1

Error

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 1

Result 1

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

COMPARE

Expected Result 1

Result 1

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

COMPARE

Expected Result 1

Result 1

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

COMPARE

Expected Result 1

Result 1

Backpropagation: Updating Model Parameters

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

- When you update model parameters with only one sample per iteration then this technique is called Stochastic Gradient Descent.

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

BATCH of size 2

Sample 0

Sample 1

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Result 0

Result 1

Feedforward: Prediction

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Feedforward: Prediction

Expected Result 0

Expected Result 1

Result 0

Result 1

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Error 0

Error 1

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Error 0

Error 1

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Error Mean

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

Feedforward: Prediction

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Training Neural Network

matrix X

Sample 0

Sample 1

Sample 2

Sample 3

...

Sample n

vector y

Expected Result 0

Expected Result 1

Expected Result 2

Expected Result 3

...

Expected Result n

Neural Network

COMPARE

Expected Result 0

Expected Result 1

Result 0

Result 1

Backpropagation: Updating Model Parameters

Training Neural Network

matrix X	vector y
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
...	...
Sample n	Expected Result n

Neural Network

Training Neural Network

- When you update model parameters with more than one sample per iteration but less than all available samples then this technique is called Mini-Batch Gradient Descent.

Training Neural Network

- When you update model parameters with more than one sample per iteration but less than all available samples then this technique is called Mini-Batch Gradient Descent.

- When you update model parameters with all available samples then this technique is called Batch Gradient Descent.

Summary

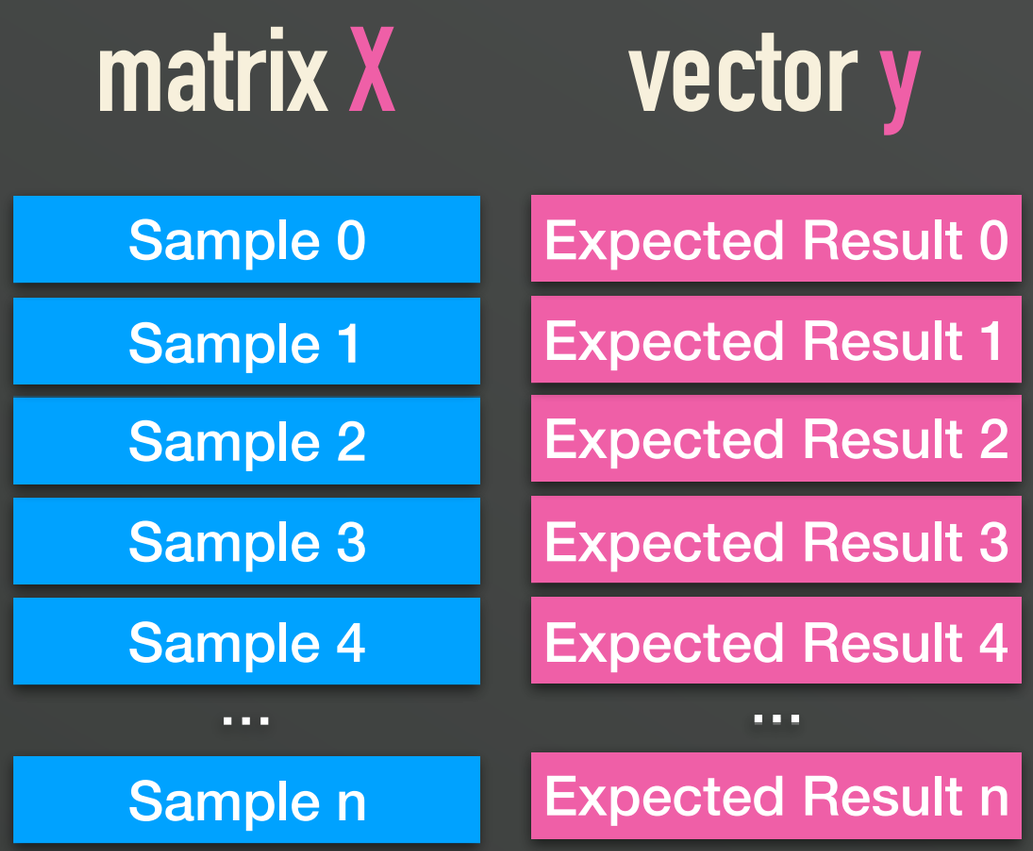
- Batch is a number of samples fed to Neural Network during single iteration.

Summary

- Batch is a number of samples fed to Neural Network during single iteration.

Summary

- Batch is a number of samples fed to Neural Network during single iteration.



Summary

- Batch is a number of samples fed to Neural Network during single iteration.

matrix X

vector y

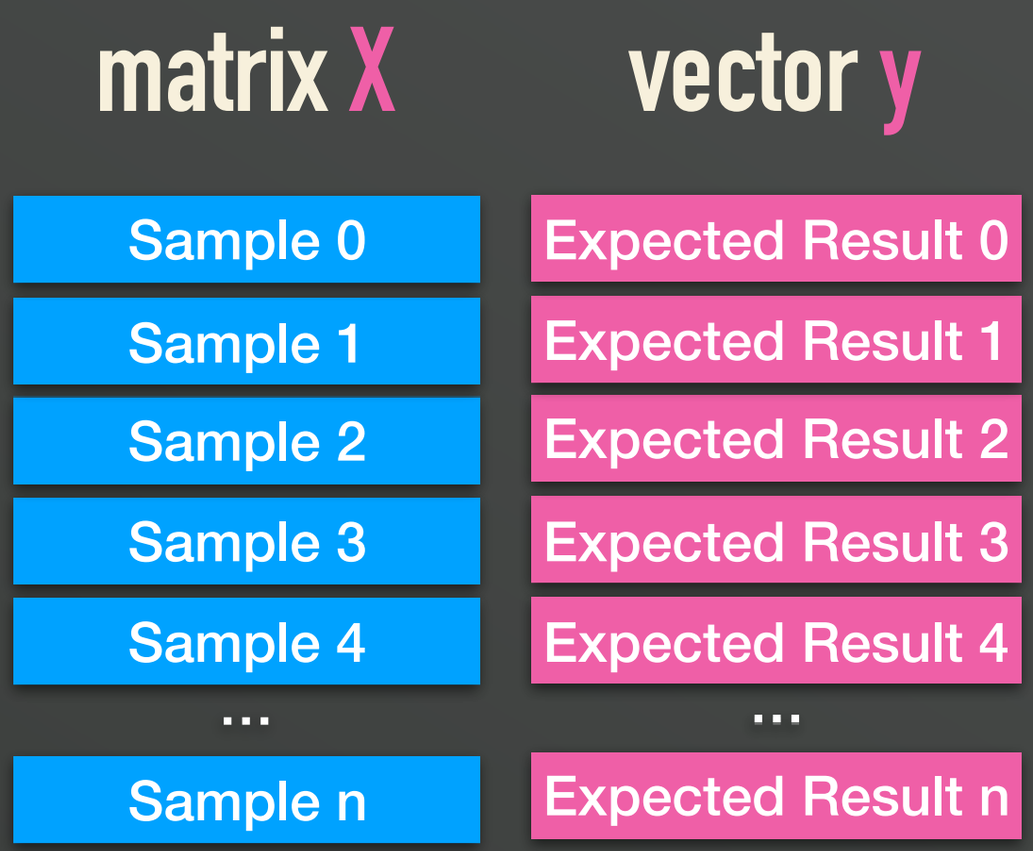
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
Sample 4	Expected Result 4
...	...
Sample n	Expected Result n

BATCH SIZE == NUMBER OF SAMPLES

Batch Gradient Descent

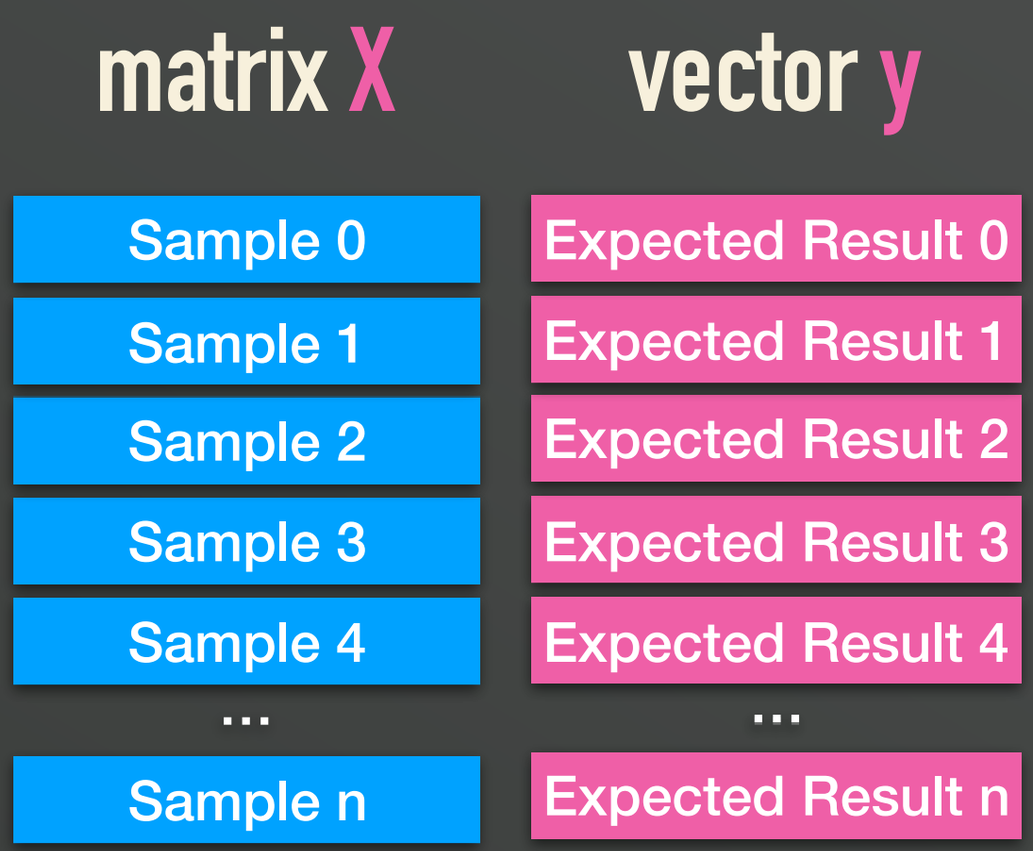
Summary

- Batch is a number of samples fed to Neural Network during single iteration.



Summary

- Batch is a number of samples fed to Neural Network during single iteration.



Summary

- Batch is a number of samples fed to Neural Network during single iteration.

matrix X vector y

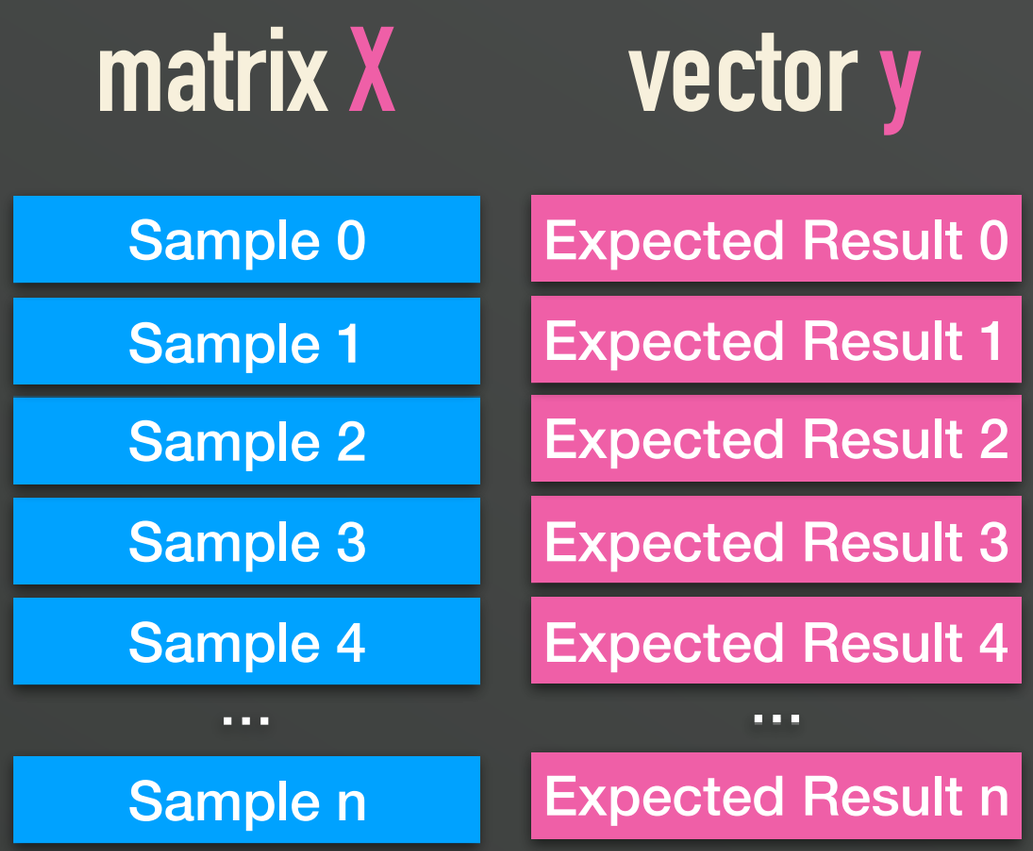
Sample 0	Expected Result 0
Sample 1	Expected Result 1
Sample 2	Expected Result 2
Sample 3	Expected Result 3
Sample 4	Expected Result 4
...	...
Sample n	Expected Result n

1 < BATCH SIZE < NUMBER OF SAMPLES

Mini-Batch Gradient Descent

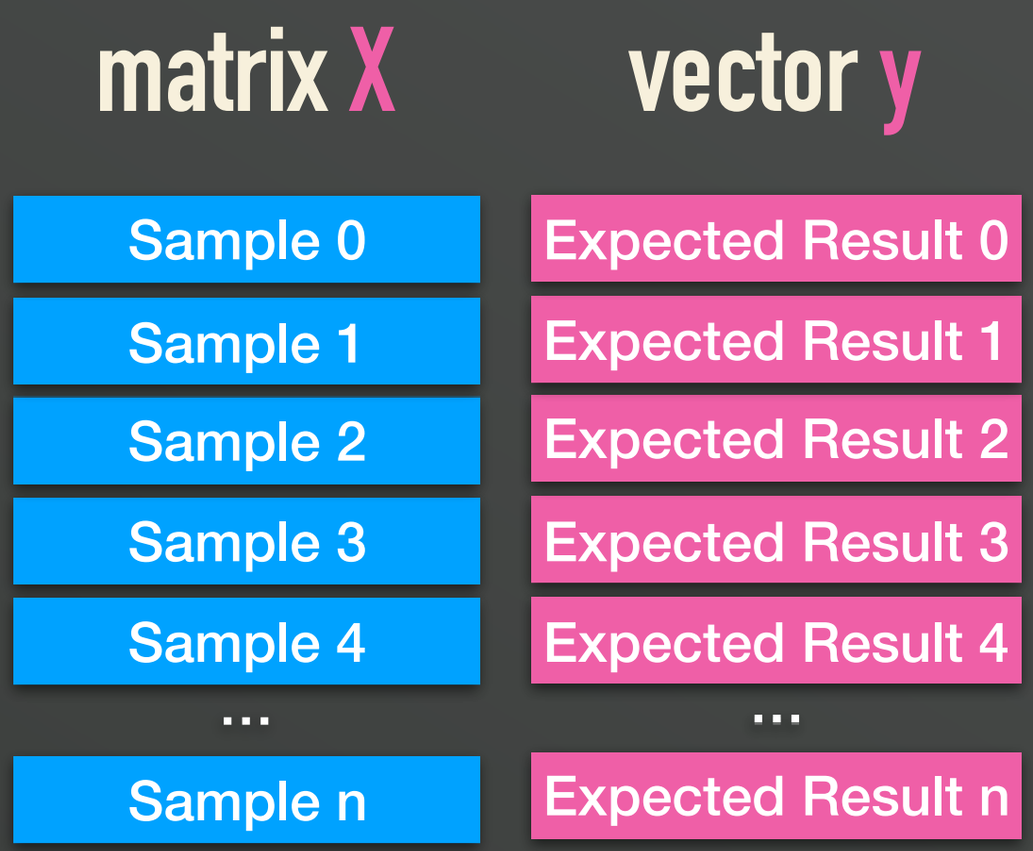
Summary

- Batch is a number of samples fed to Neural Network during single iteration.



Summary

- Batch is a number of samples fed to Neural Network during single iteration.



Summary

- Batch is a number of samples fed to Neural Network during single iteration.

matrix X

Sample 0
Sample 1
Sample 2
Sample 3
Sample 4
...
Sample n

vector y

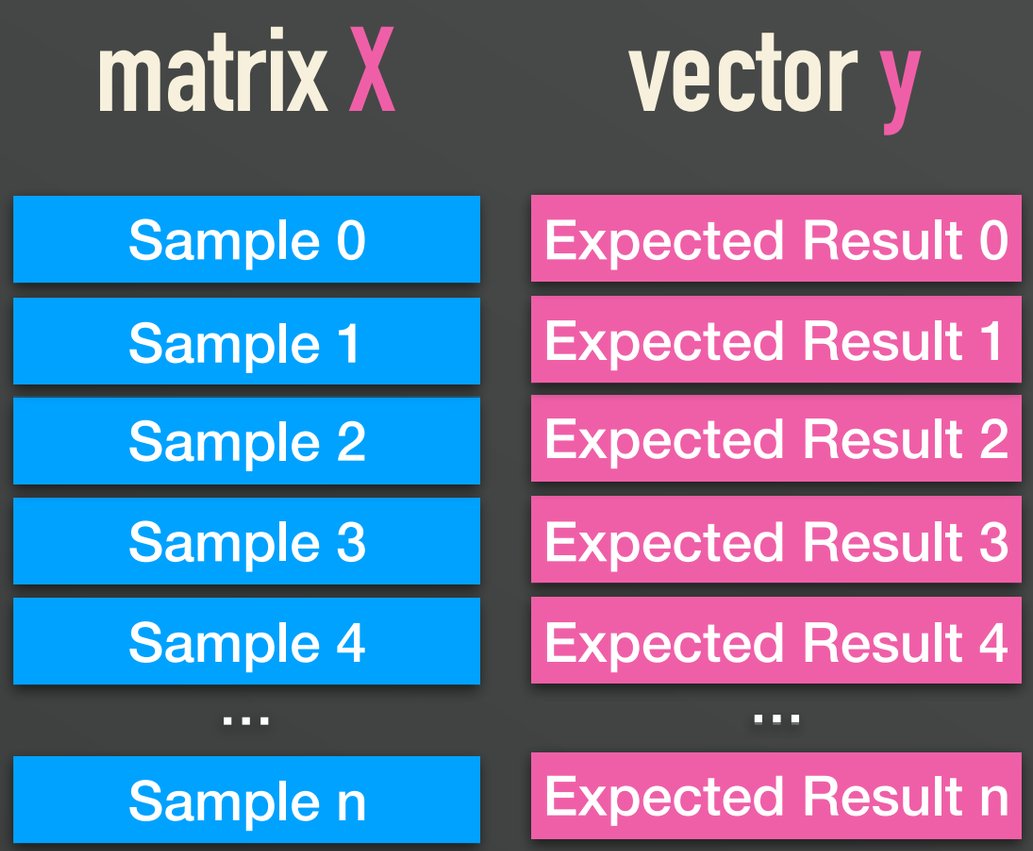
Expected Result 0
Expected Result 1
Expected Result 2
Expected Result 3
Expected Result 4
...
Expected Result n

BATCH SIZE == 1

Stochastic Gradient Descent

Summary

- Batch is a number of samples fed to Neural Network during single iteration.



What is Normalization?

Intuition Behind Normalization

- Normalization is a technique of adjusting values that were measured on different scales to notionally common scale.

Intuition Behind Normalization

- **Normalization** is a technique of adjusting values that were measured on different scales to notionally common scale.

Normalization types in Statistics:

Name	Formula	Use
Standard score	$\frac{X - \mu}{\sigma}$	Normalizing errors when population parameters are known. Works well for populations that are normally distributed
Student's t-statistic	$\frac{X - \overline{X}}{s}$	Normalizing residuals when population parameters are unknown (estimated).
Studentized residual	$\frac{\hat{\epsilon}_i}{\hat{\sigma}_i} = \frac{X_i - \hat{\mu}_i}{\hat{\sigma}_i}$	Normalizing residuals when parameters are estimated, particularly across different data points in regression analysis.
Standardized moment	$\frac{\mu_k}{\sigma^k}$	Normalizing moments, using the standard deviation σ as a measure of scale.
Coefficient of variation	$\frac{\sigma}{\mu}$	Normalizing dispersion, using the mean μ as a measure of scale, particularly for positive distribution such as the exponential distribution and Poisson distribution.
Feature scaling	$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$	Feature scaling is used to bring all values into the range [0,1]. This is also called unity-based normalization. This can be generalized to restrict the range of values in the dataset between any arbitrary points a and b using $X' = a + \frac{(X - X_{\min})(b - a)}{X_{\max} - X_{\min}}$.

Intuition Behind Normalization

- **Normalization** is a technique of adjusting values that were measured on different scales to notionally common scale.

Normalization types in Statistics:

Name	Formula	Use
Standard score	$\frac{X - \mu}{\sigma}$	Normalizing errors when population parameters are known. Works well for populations that are normally distributed
Student's t-statistic	$\frac{X - \overline{X}}{s}$	Normalizing residuals when population parameters are unknown (estimated).
Studentized residual	$\frac{\hat{\epsilon}_i}{\hat{\sigma}_i} = \frac{X_i - \hat{\mu}_i}{\hat{\sigma}_i}$	Normalizing residuals when parameters are estimated, particularly across different data points in regression analysis.
Standardized moment	$\frac{\mu_k}{\sigma^k}$	Normalizing moments, using the standard deviation σ as a measure of scale.
Coefficient of variation	$\frac{\sigma}{\mu}$	Normalizing dispersion, using the mean μ as a measure of scale, particularly for positive distribution such as the exponential distribution and Poisson distribution.
Feature scaling	$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$	Feature scaling is used to bring all values into the range [0,1]. This is also called unity-based normalization. This can be generalized to restrict the range of values in the dataset between any arbitrary points a and b using $X' = a + \frac{(X - X_{\min})(b - a)}{X_{\max} - X_{\min}}$.

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[19,	1 500]
person 1:	[33,	22 000]
person 2:	[57,	435 000]
person 3:	[26,	80 000]

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[19,	1 500]
person 1:	[33,	22 000]
person 2:	[57,	435 000]
person 3:	[26,	80 000]

How computer can know if age 30 is a lot or if 400 000 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[19,	1 500]
person 1:	[33,	22 000]
person 2:	[57,	435 000]
person 3:	[26,	80 000]

By comparing how far is it from min or max value:

$$\text{age\_diff} = |57 - 30| = 27$$
$$\text{balance\_diff} = |435000 - 400000| = 35000$$

How computer can know if age 30 is a lot or if 400 000 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[19,	1 500]
person 1:	[33,	22 000]
person 2:	[57,	435 000]
person 3:	[26,	80 000]

By comparing how far is it from min or max value:

$$\text{age\_diff} = |57 - 30| = 27$$
$$\text{balance\_diff} = |435000 - 400000| = 35000$$

age - a lot

balance - small

How computer can know if age 30 is a lot or if 400 000 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[19,	1 500]
person 1:	[33,	22 000]
person 2:	[57,	435 000]
person 3:	[26,	80 000]

By comparing how far is it from min or max value:

$$\text{age\_diff} = |57 - 30| = 27$$
$$\text{balance\_diff} = |435000 - 400000| = 35000$$

age - a lot
balance - small

How computer can know if age 30 is a lot or if 400 000 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[19,	1 500]
person 1:	[33,	22 000]
person 2:	[57,	435 000]
person 3:	[26,	80 000]

How computer can know if age 30 is a lot or if 400 000 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[19,	1 500]
person 1:	[33,	22 000]
person 2:	[57,	435 000]
person 3:	[26,	80 000]

How computer can know if age 30 is a lot or if 400 000 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:		
person 1:		
person 2:		
person 3:		

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:		
person 1:		
person 2:		
person 3:		

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[0,	0]
person 1:	[0.37,	0.47]
person 2:	[1,	1]
person 3:	[0.18,	0.18]

How computer can know if age 0.29 is a lot or if 0.91 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[0,	0]
person 1:	[0.37,	0.47]
person 2:	[1,	1]
person 3:	[0.18,	0.18]

By comparing how far is it from min or max value:

$$\text{age\_diff} = |1 - 0.29| = 0.71$$
$$\text{balance\_diff} = |1 - 0.91| = 0.09$$

How computer can know if age 0.29 is a lot or if 0.91 is much?

Intuition Behind Normalization

- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[0,	0]
person 1:	[0.37,	0.47]
person 2:	[1,	1]
person 3:	[0.18,	0.18]

By comparing how far is it from min or max value:

$$\text{age\_diff} = |1 - 0.29| = 0.71$$
$$\text{balance\_diff} = |1 - 0.91| = 0.09$$

age - small

balance - large

How computer can know if age 0.29 is a lot or if 0.91 is much?

Intuition Behind Normalization

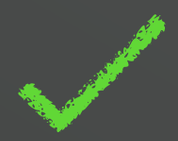
- Some Machine Learning algorithms use geometrical distance (e.g. Manhattan, Euclidean) in order to compare features. Because of that features need to be on the same scale.

	Age	Balance
person 0:	[0,	0]
person 1:	[0.37,	0.47]
person 2:	[1,	1]
person 3:	[0.18,	0.18]

By comparing how far is it from min or max value:

$$\text{age\_diff} = |1 - 0.29| = 0.71$$
$$\text{balance\_diff} = |1 - 0.91| = 0.09$$

age - small
balance - large



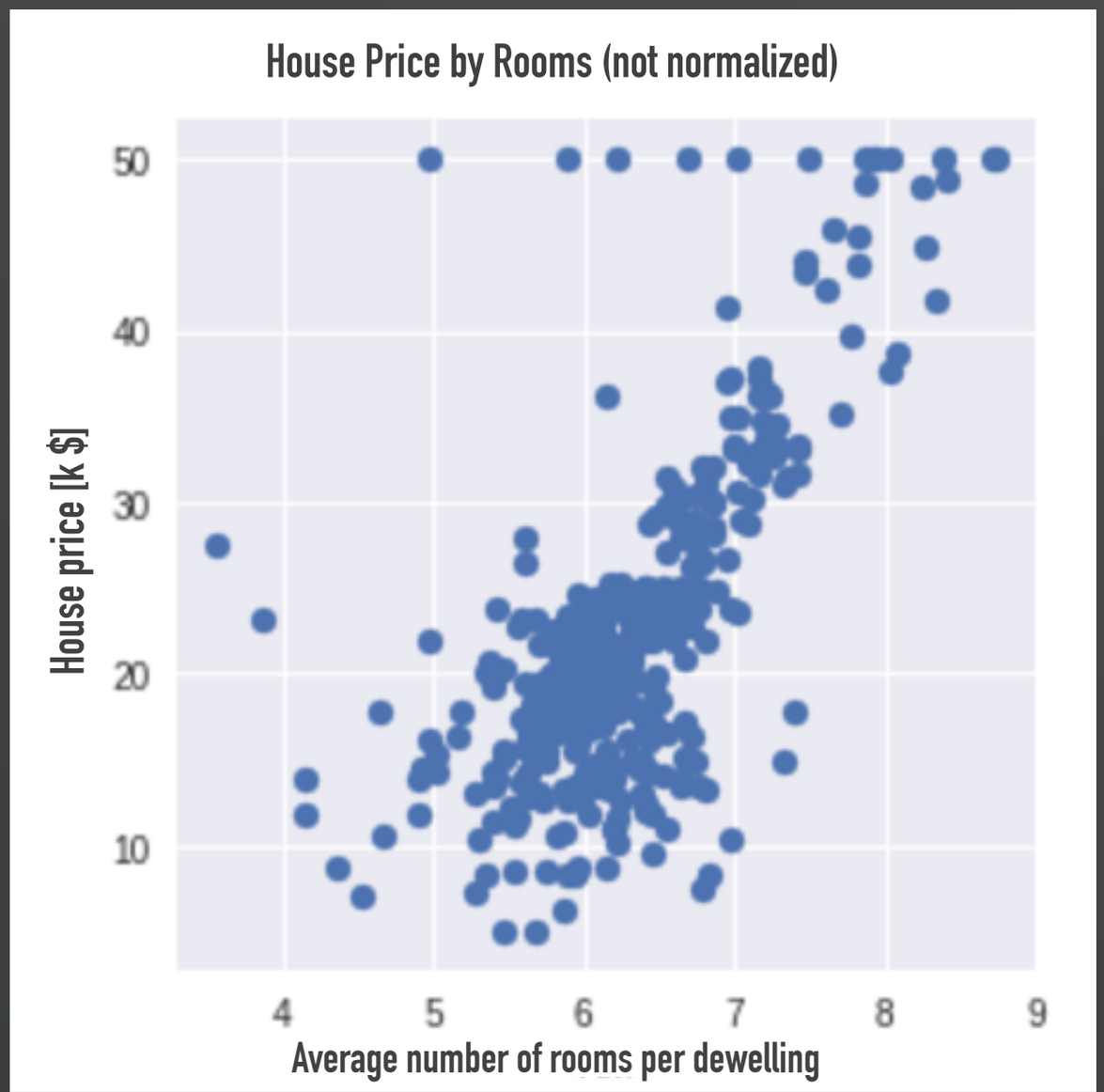
How computer can know if age 0.29 is a lot or if 0.91 is much?

Intuition Behind Normalization

- Feature values are only rescaled – so distribution of data does not change at all.

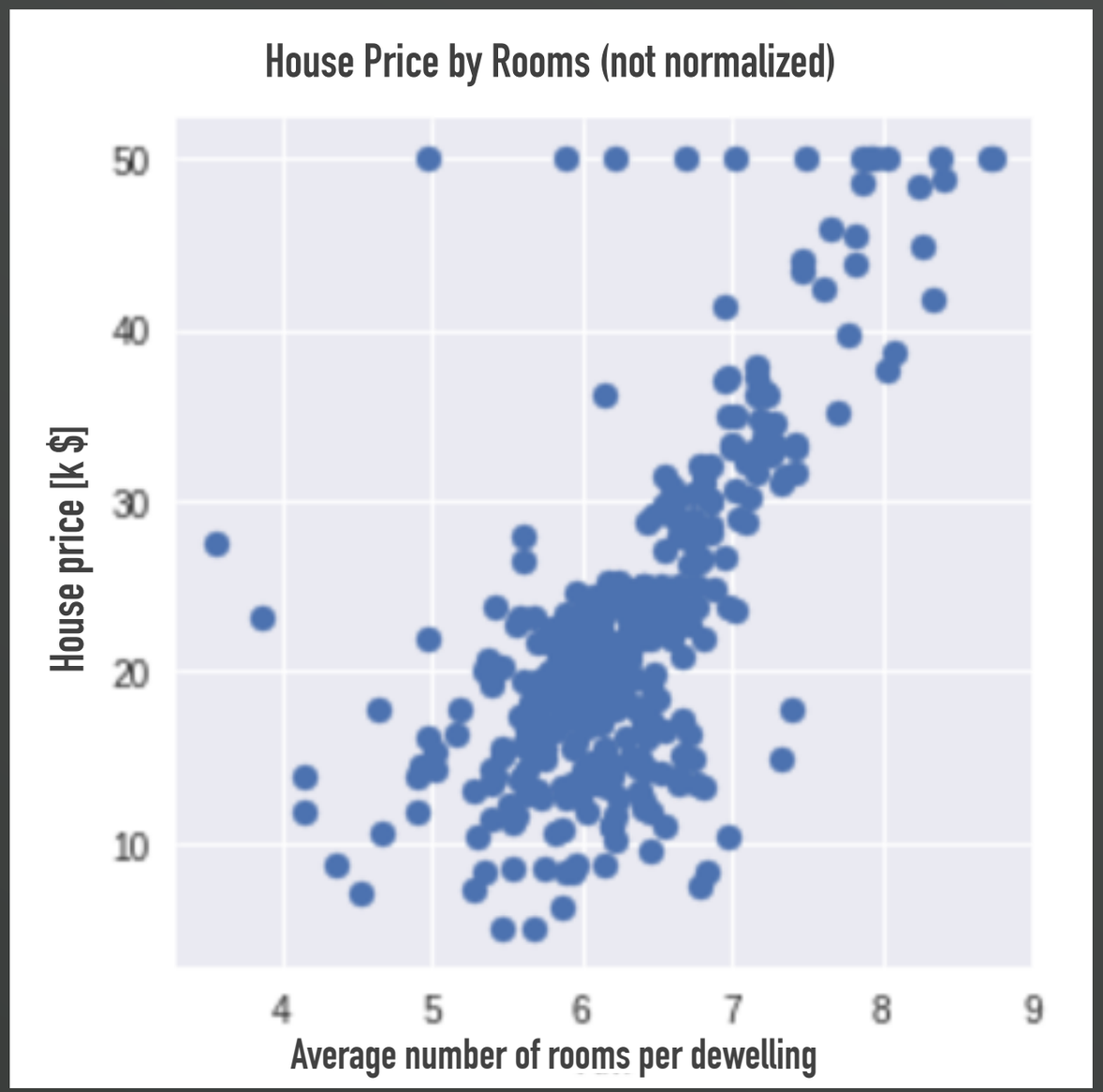
Intuition Behind Normalization

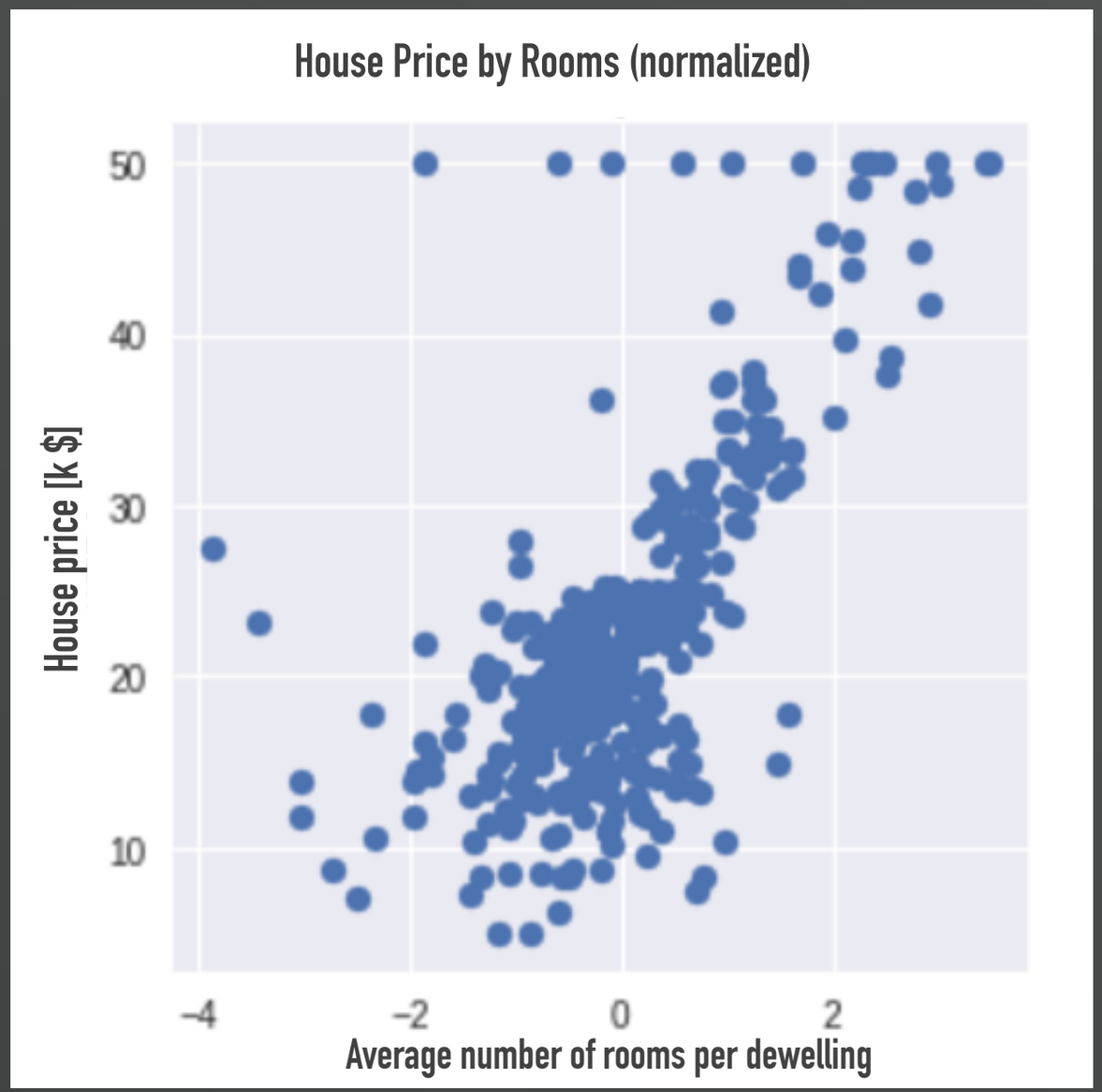
- Feature values are only rescaled – so distribution of data does not change at all.



Intuition Behind Normalization

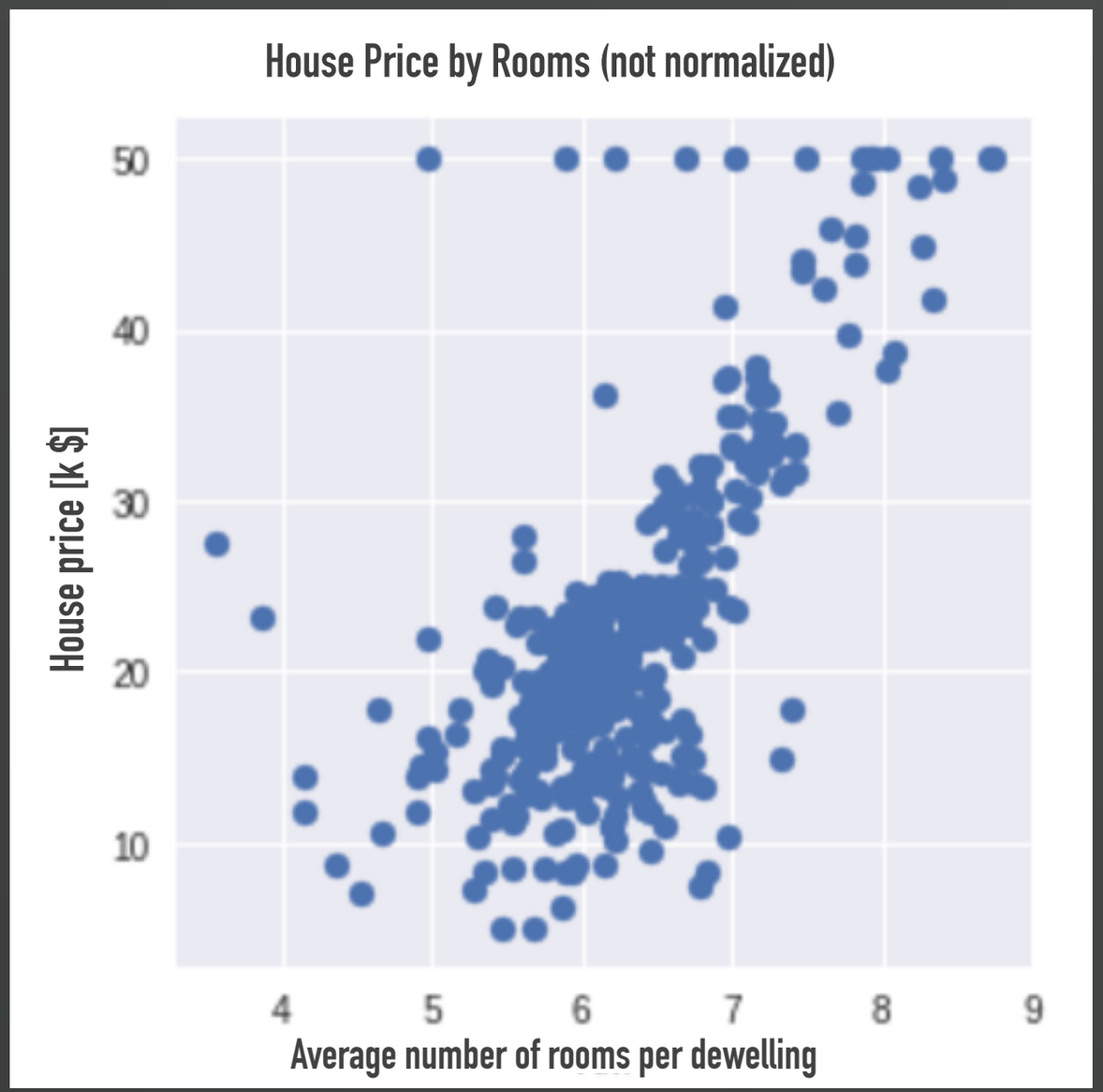
- Feature values are only rescaled – so distribution of data does not change at all.

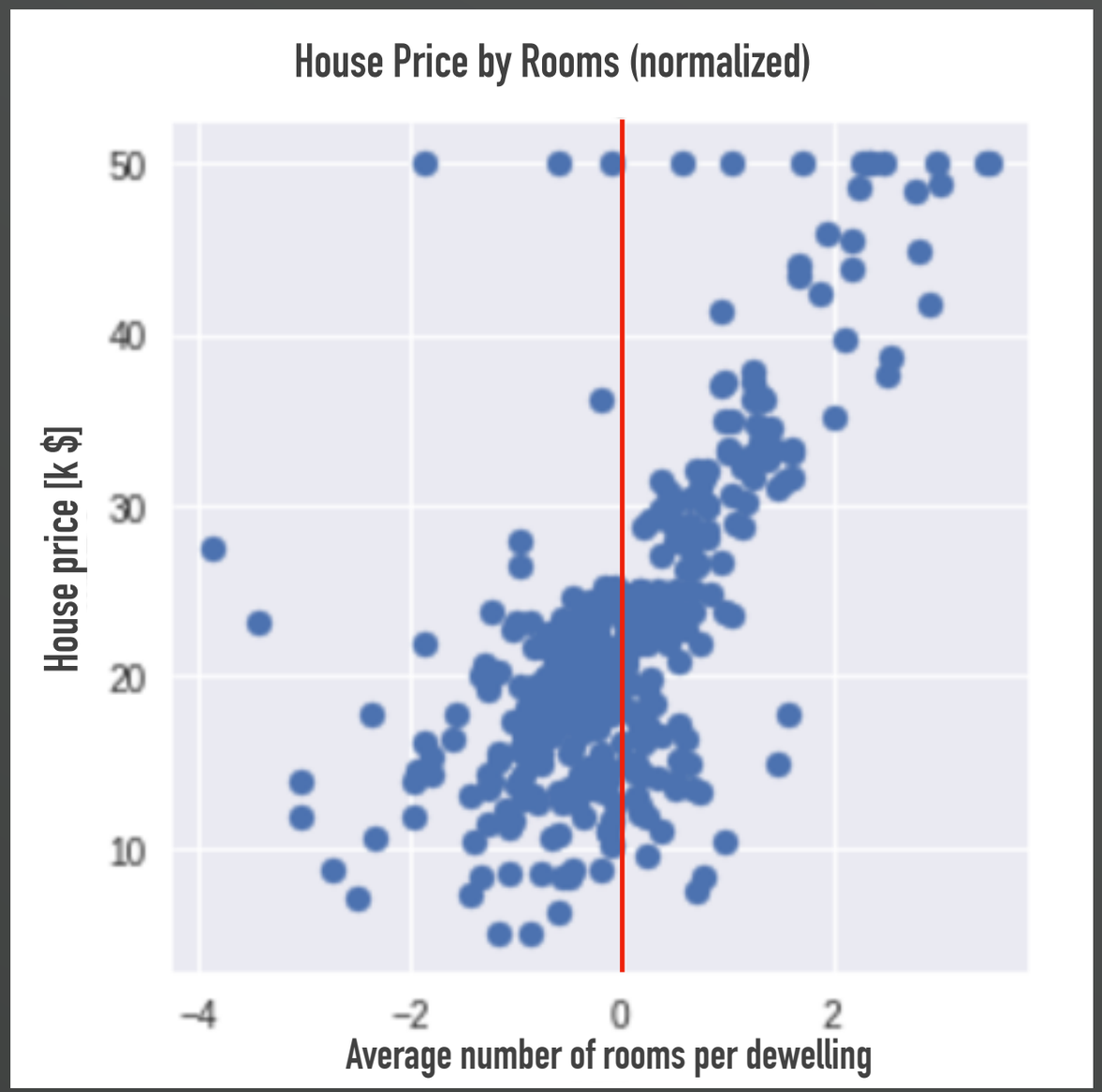




Intuition Behind Normalization

- Feature values are only rescaled – so distribution of data does not change at all.





Intuition Behind Normalization

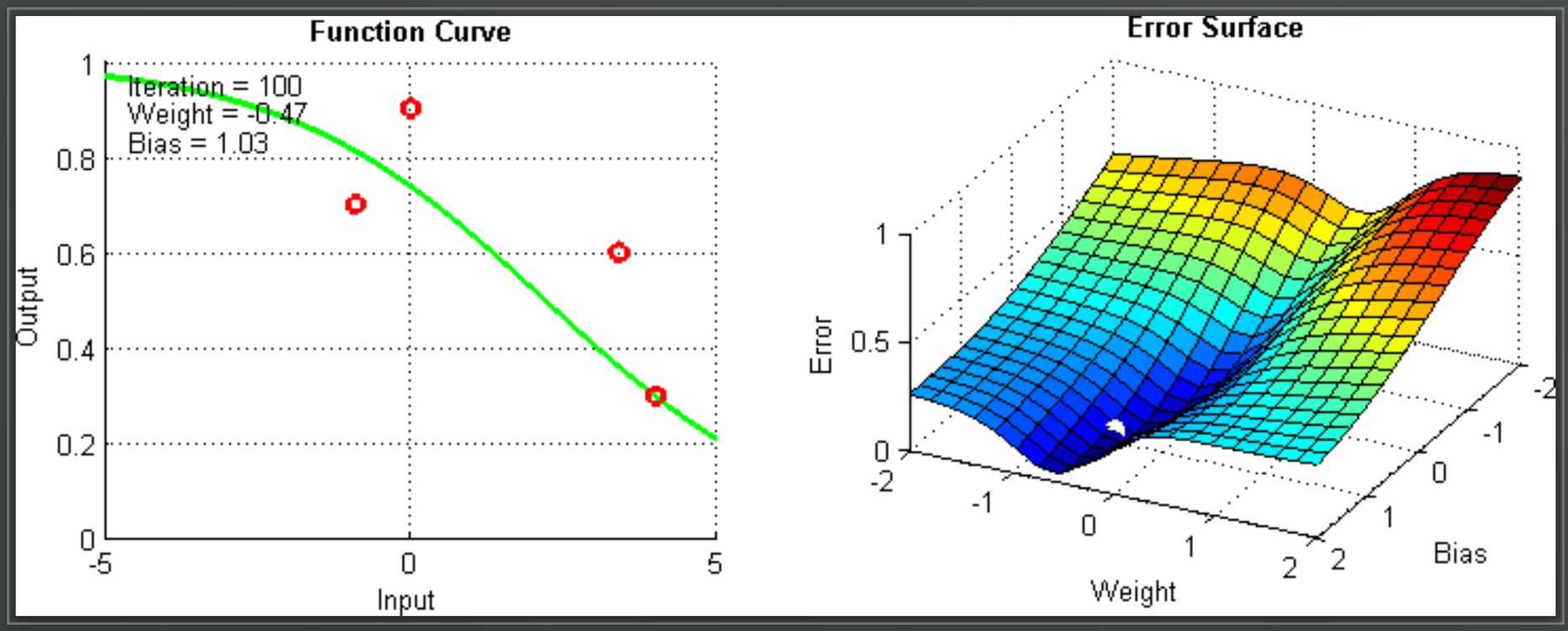
- It makes algorithms more computationally stable. Because range of values after normalization are much smaller, it's less likely to exceed value range of variables when performing math on large numbers.

Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.

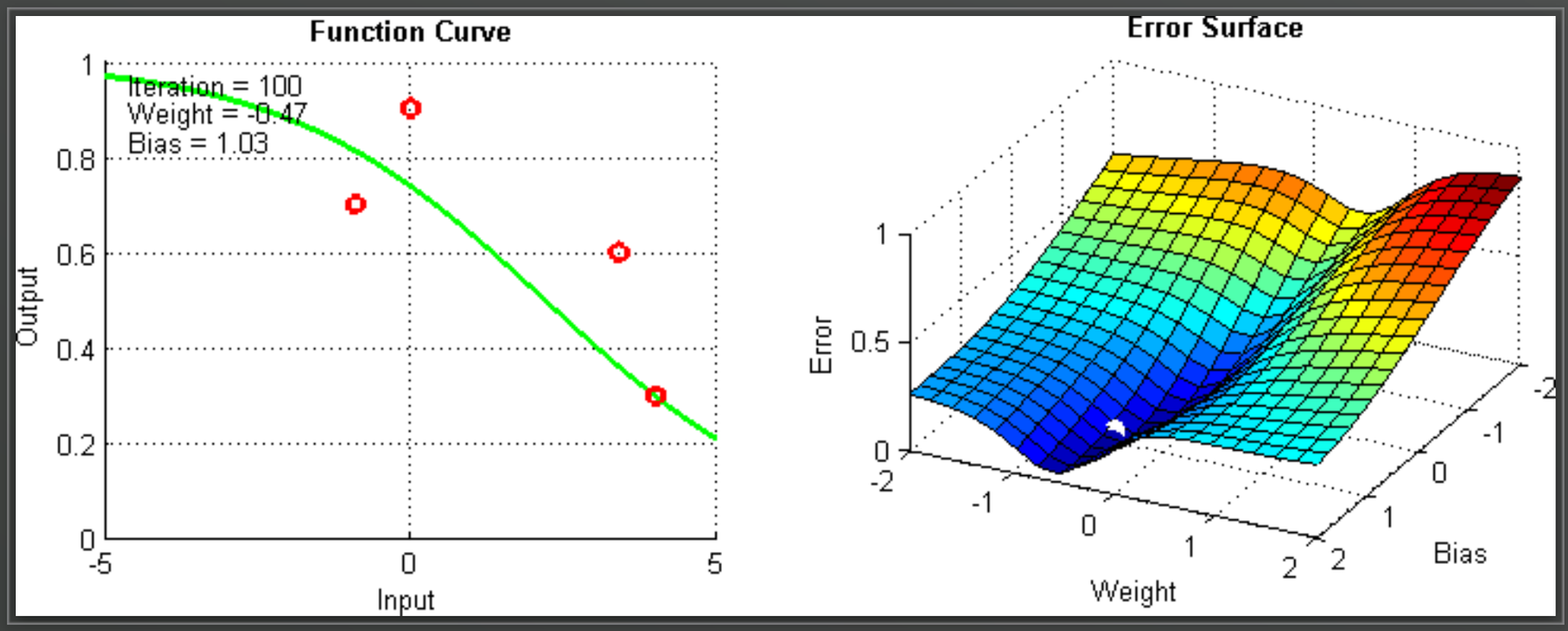
Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



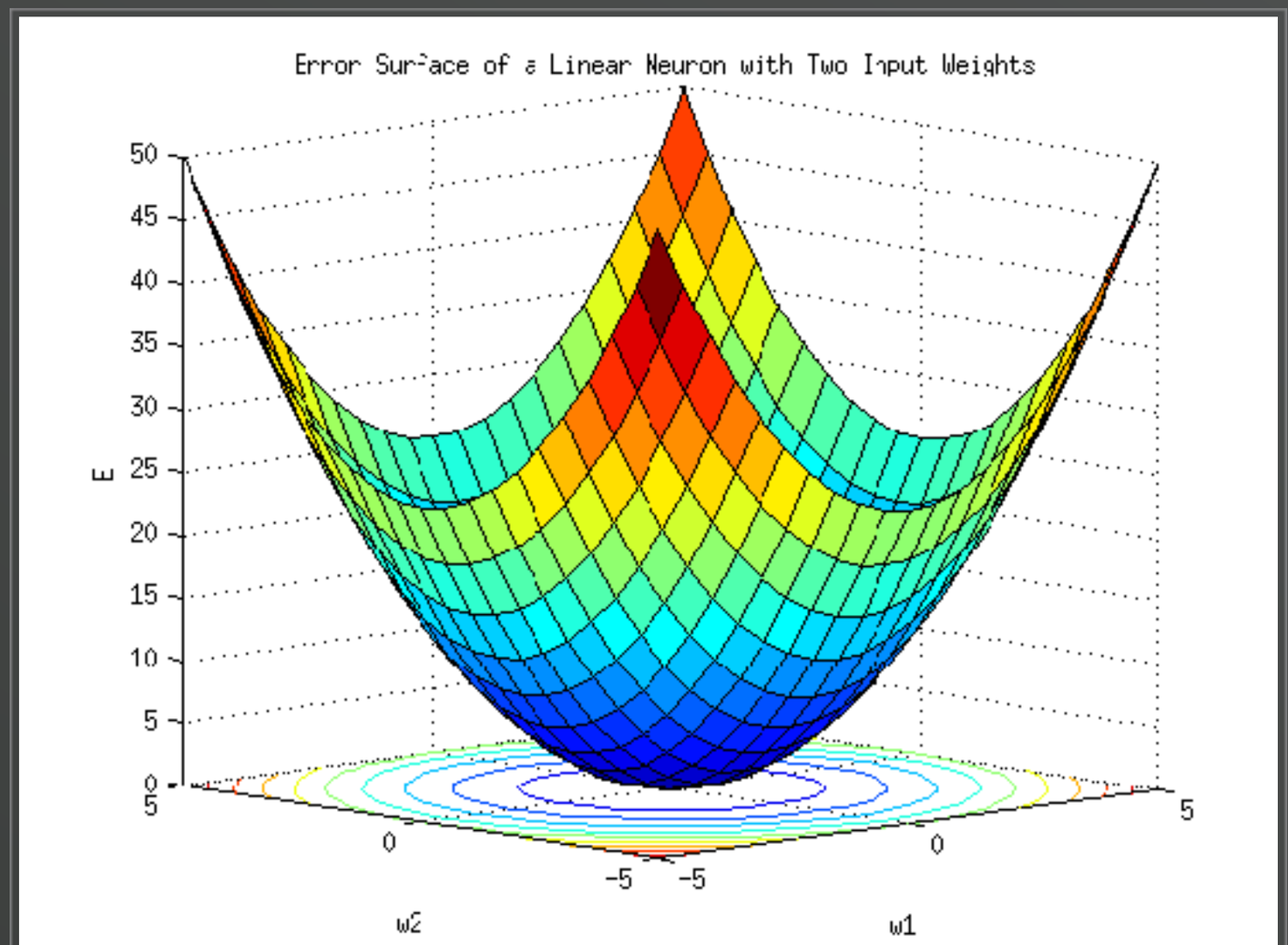
Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



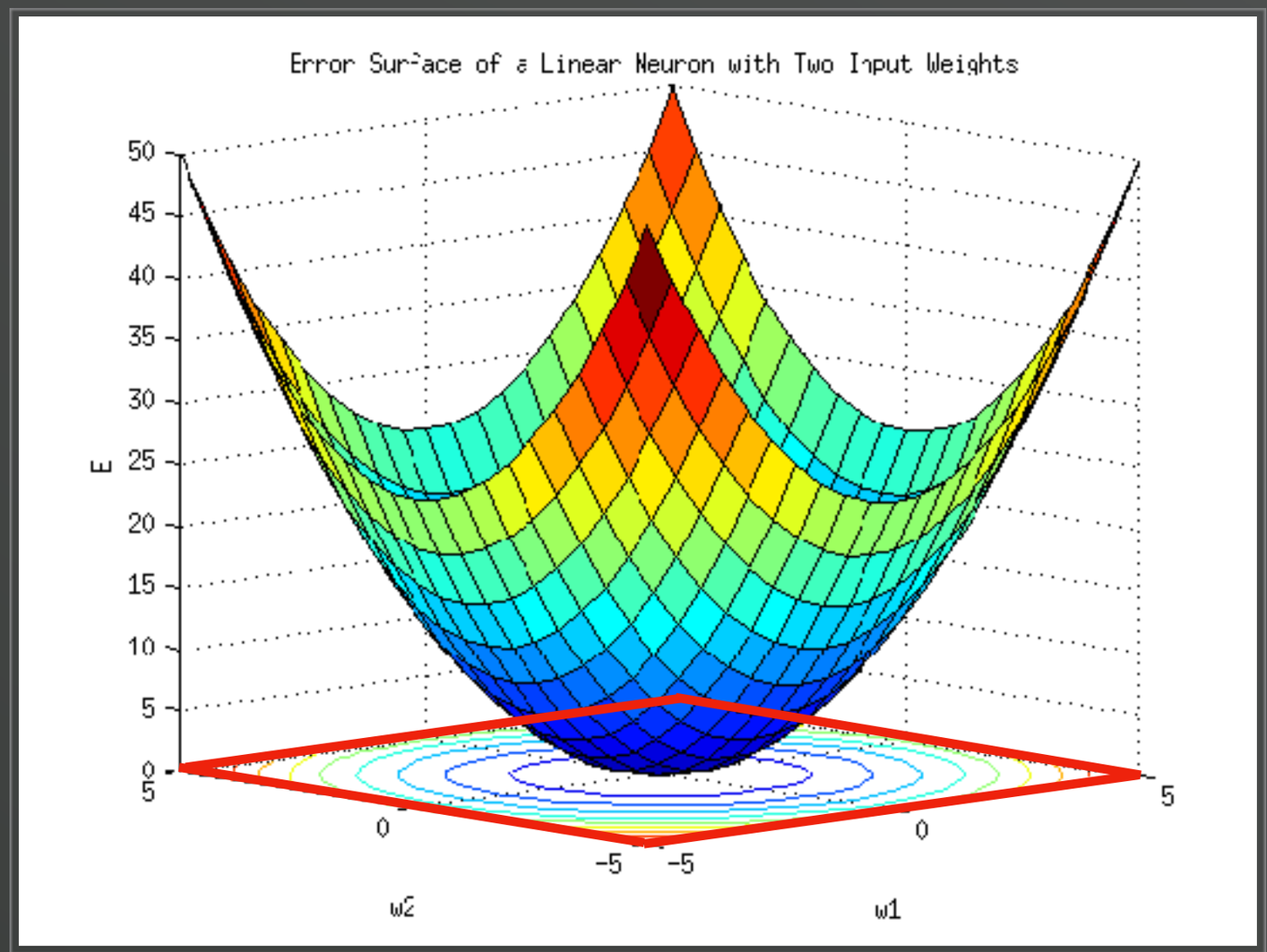
Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



Intuition Behind Normalization

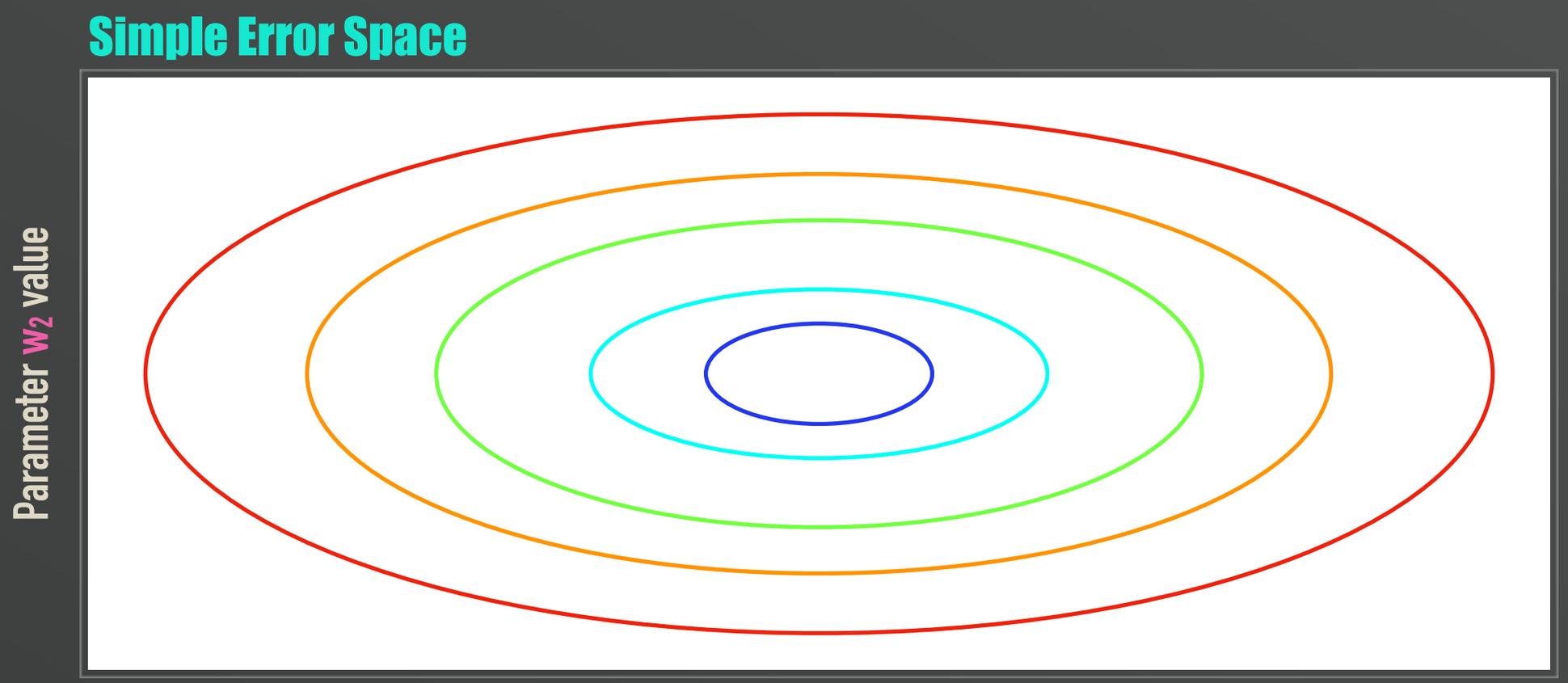
- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.

Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.

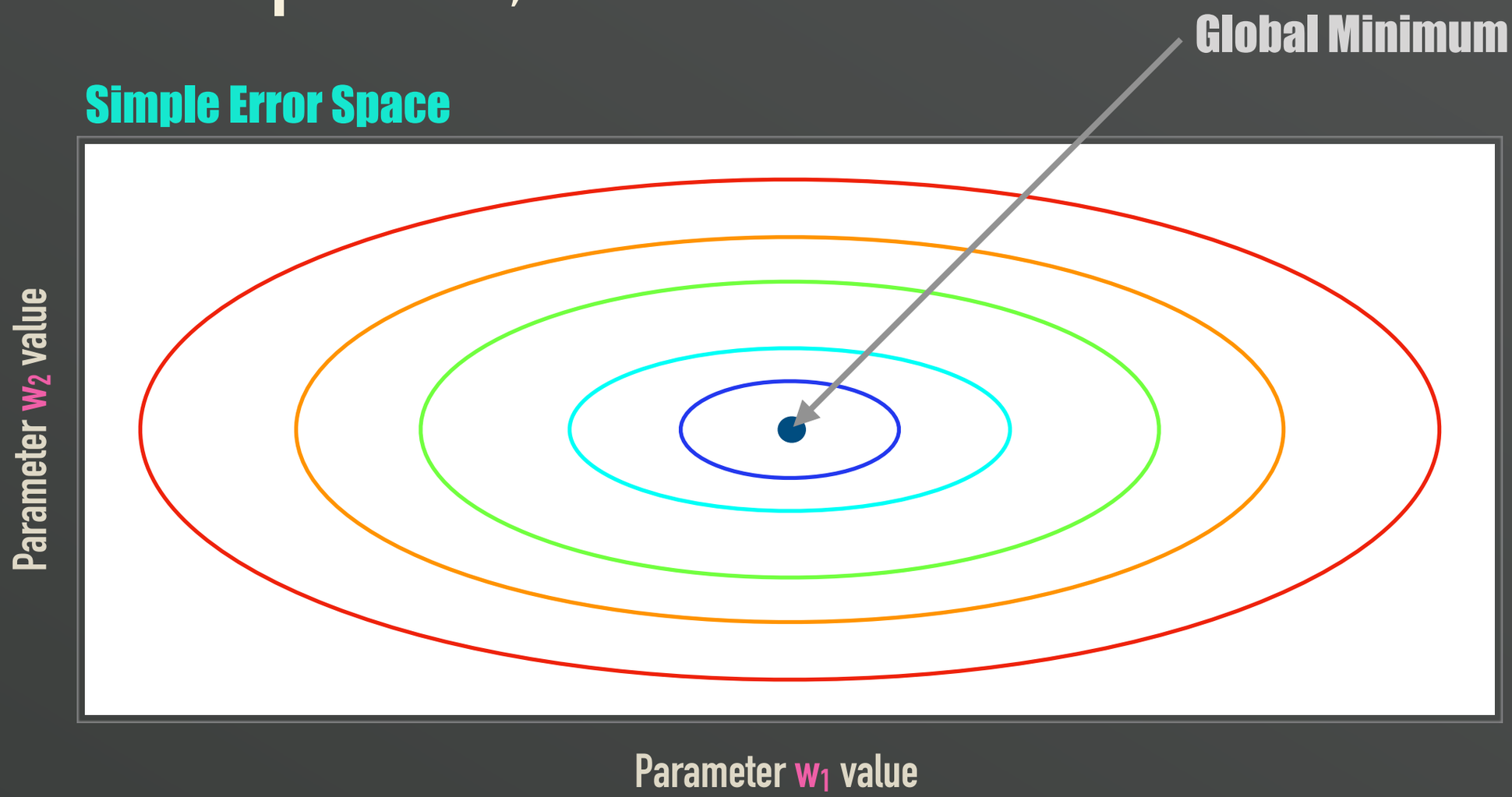
Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



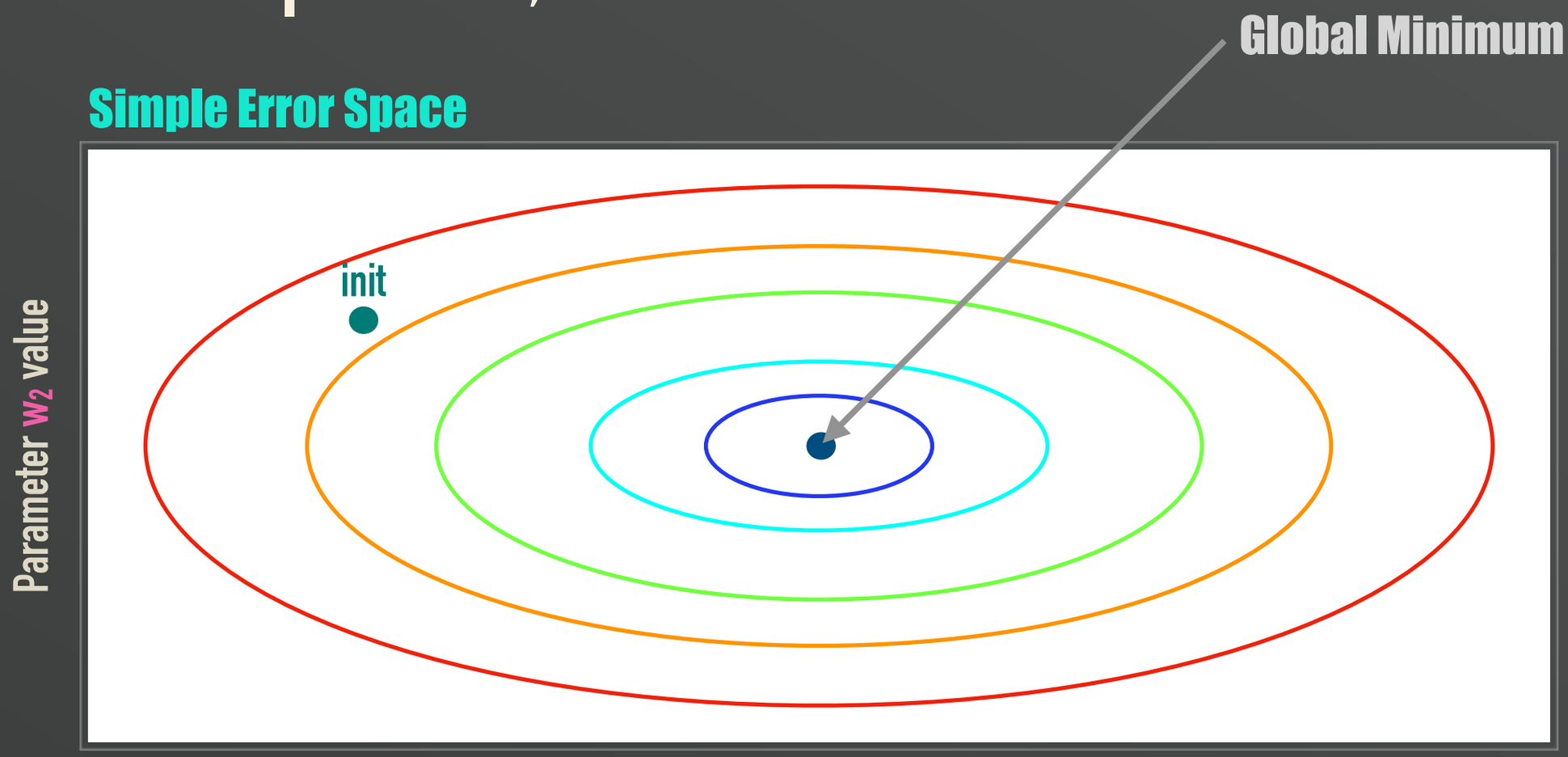
Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



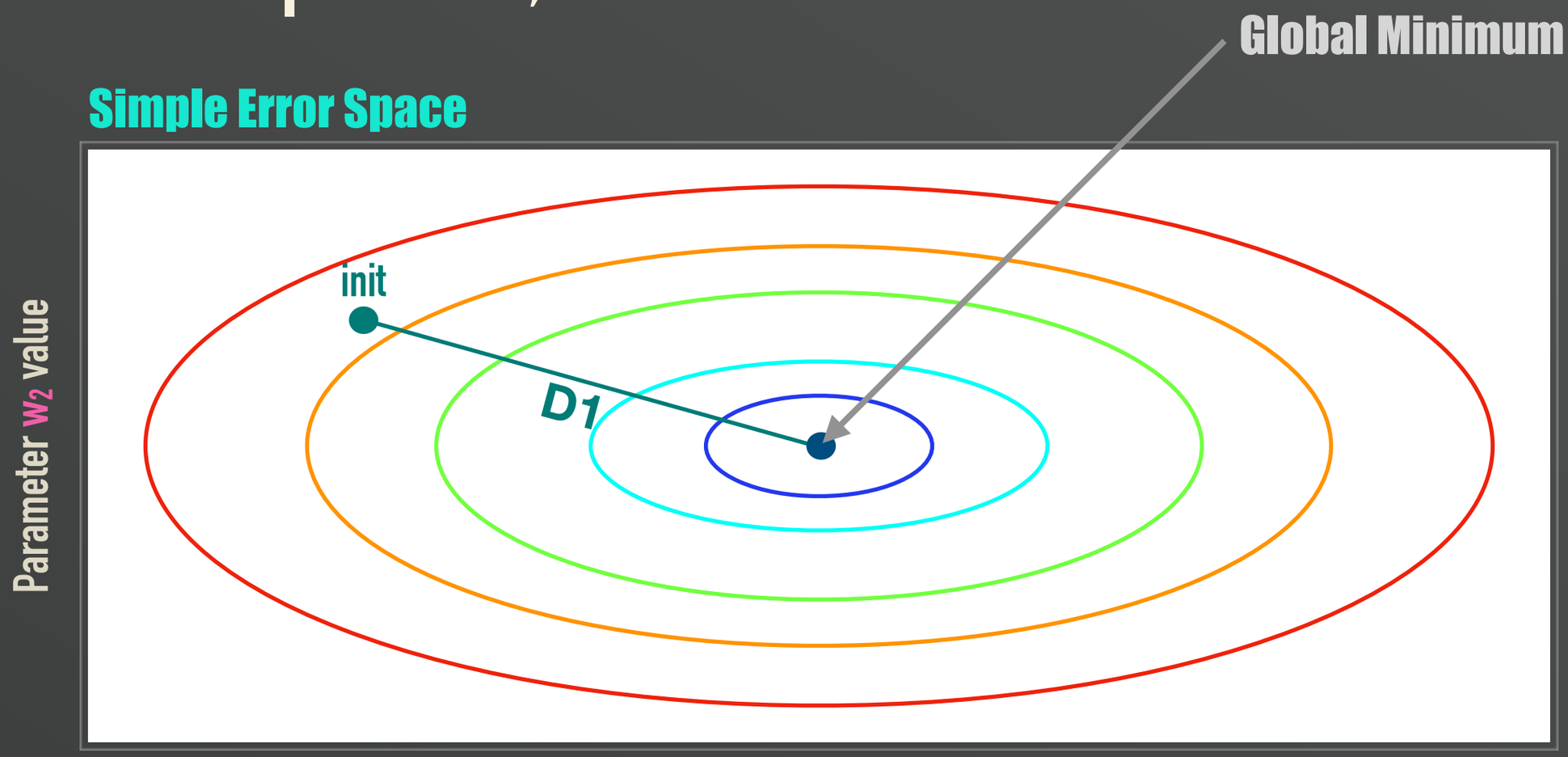
Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



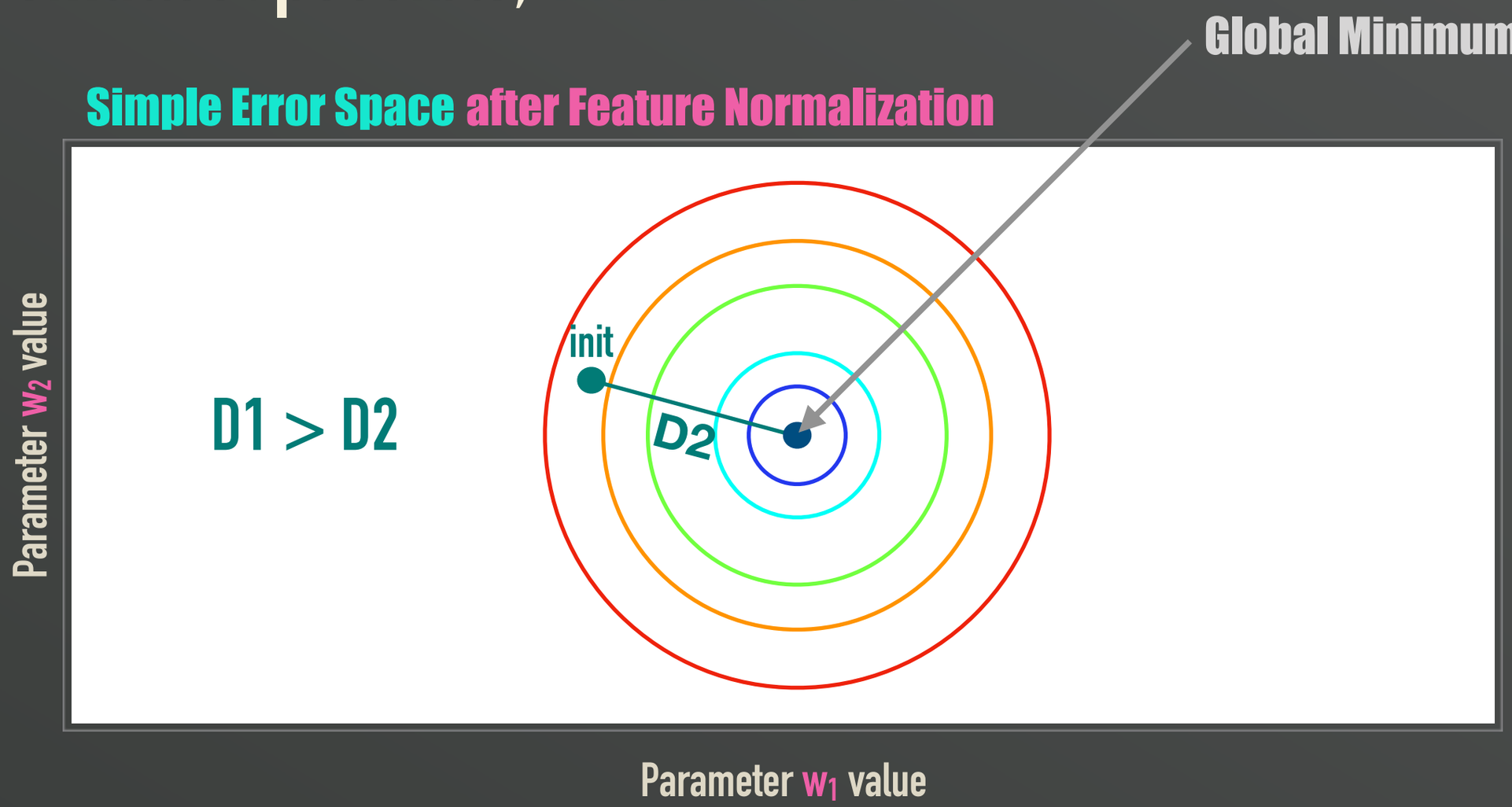
Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



Intuition Behind Normalization

- It makes Neural Networks converge (reach global minimum – state of parameters for which error is the smallest possible) much faster.



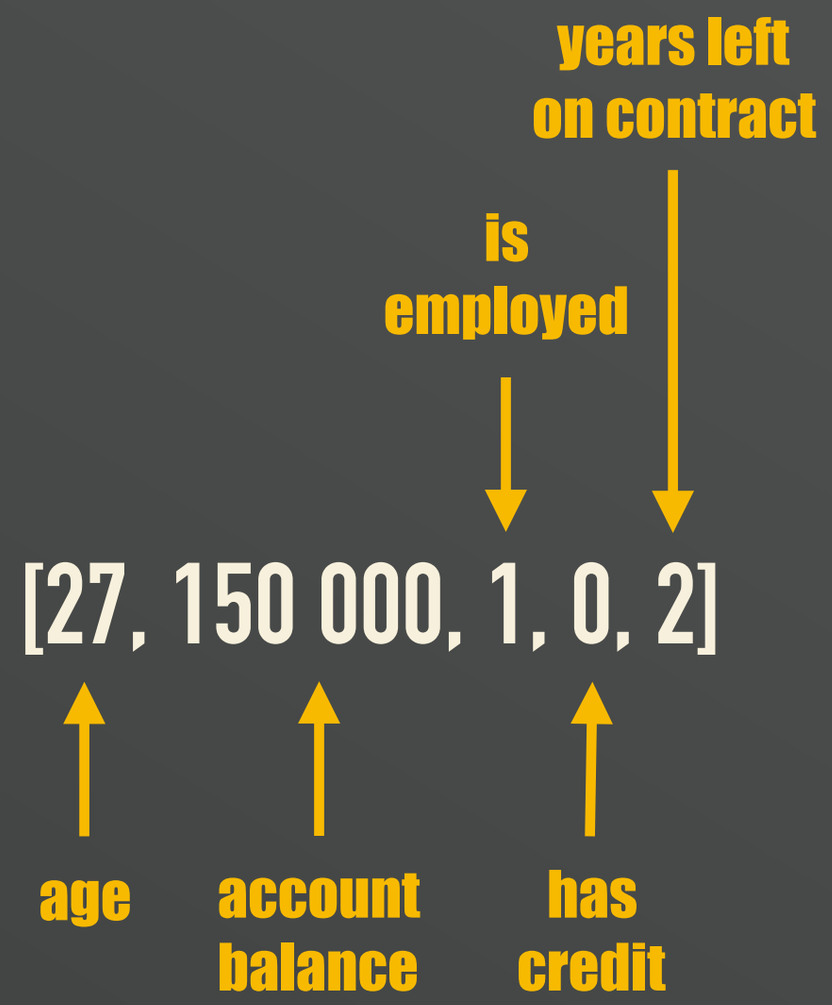
What is Batch Normalization?

Classic Neuron Activation

Classic Neuron Activation

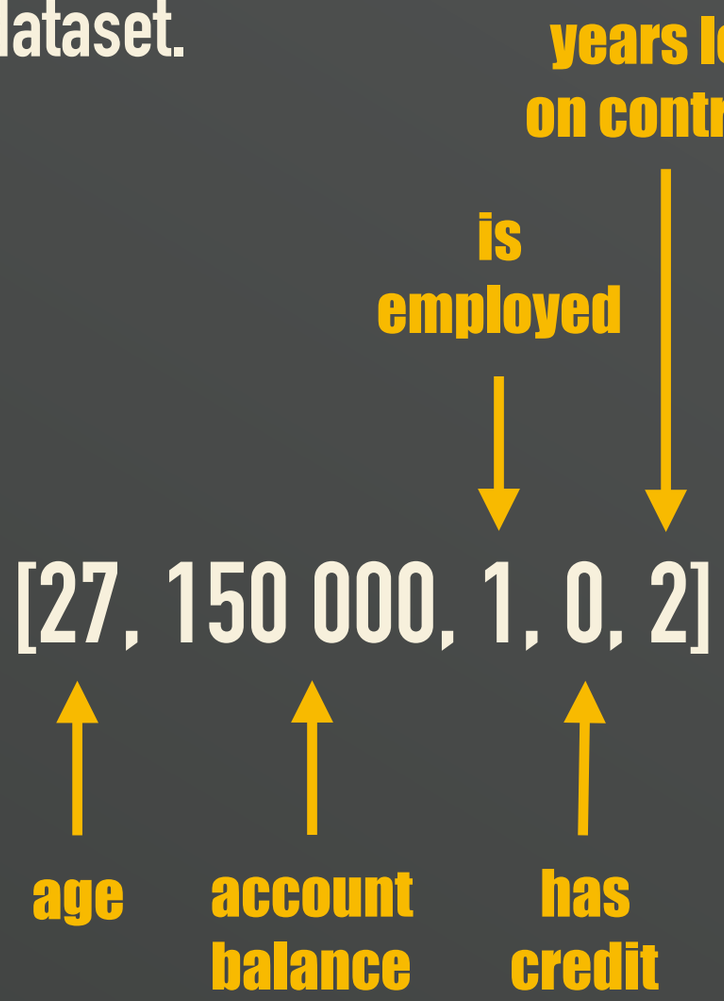
[27, 150 000, 1, 0, 2]

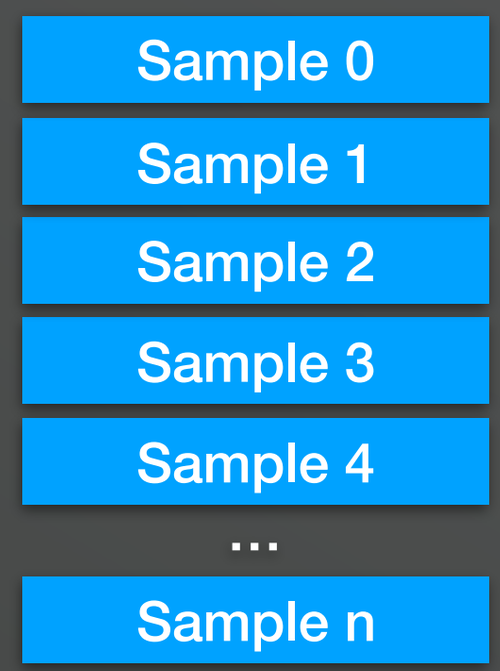
Classic Neuron Activation



Classic Neuron Activation

- Sample is part of larger dataset.





Classic Neuron Activation

- Sample is part of larger dataset.
- Normalize according to whole dataset.

[0.29, 0.43, 1, 0, 0.91]

matrix X

Sample 0
Sample 1
Sample 2
Sample 3
Sample 4
...
Sample n

Classic Neuron Activation

- Sample is part of larger dataset.
- Normalize according to whole dataset.
- Do the same for each sample.

matrix X

Sample 0
Sample 1
Sample 2
Sample 3
Sample 4
...
Sample n

$$[\, x_0 \,,\, x_1 \,,\, x_2 \,,\, x_3 \,,\, x_4 \,]$$

Classic Neuron Activation

x_0 x_1 x_2 x_3 x_4

Classic Neuron Activation

x_0

x_1

x_2

x_3

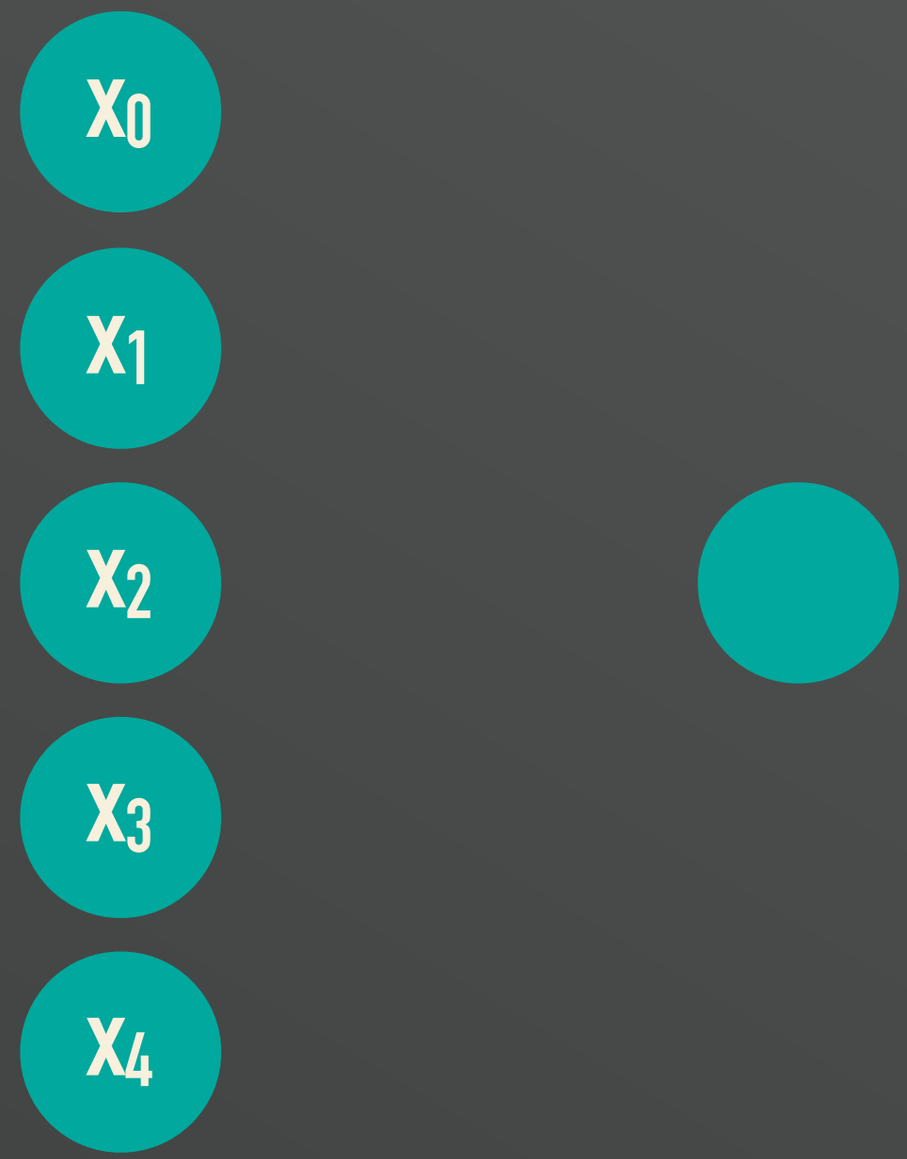
x_4

Classic Neuron Activation

- Feed data to Neural Network.

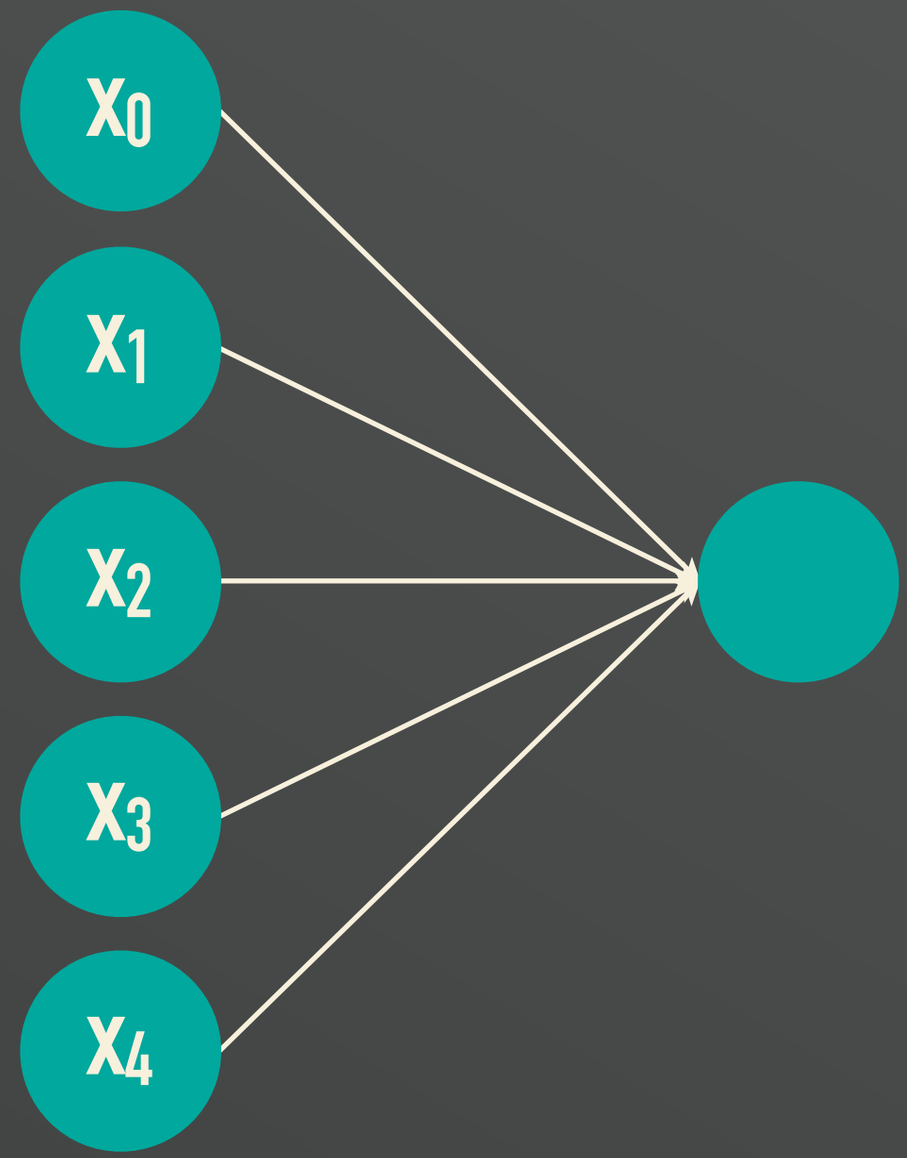
Classic Neuron Activation

- Feed data to Neural Network.



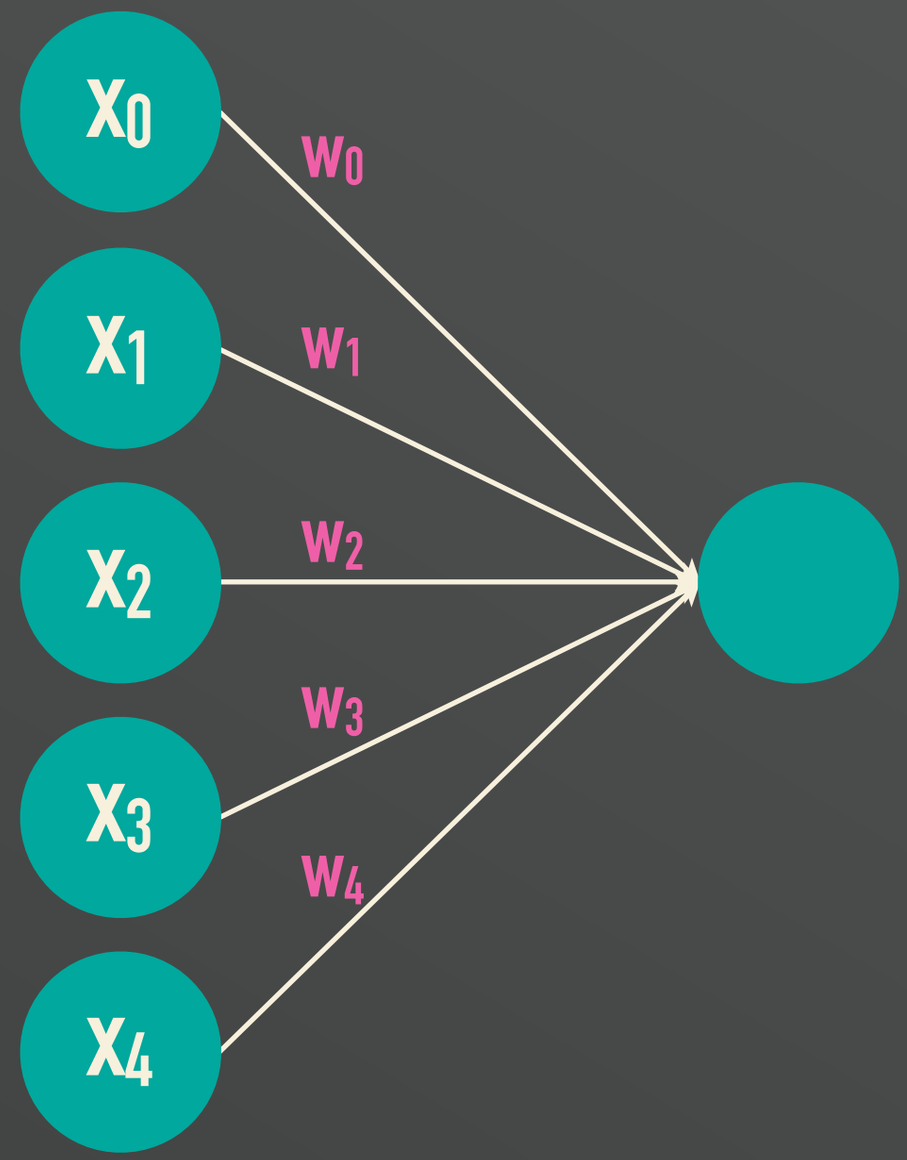
Classic Neuron Activation

- Feed data to Neural Network.
- Data is connected to one of neurons.



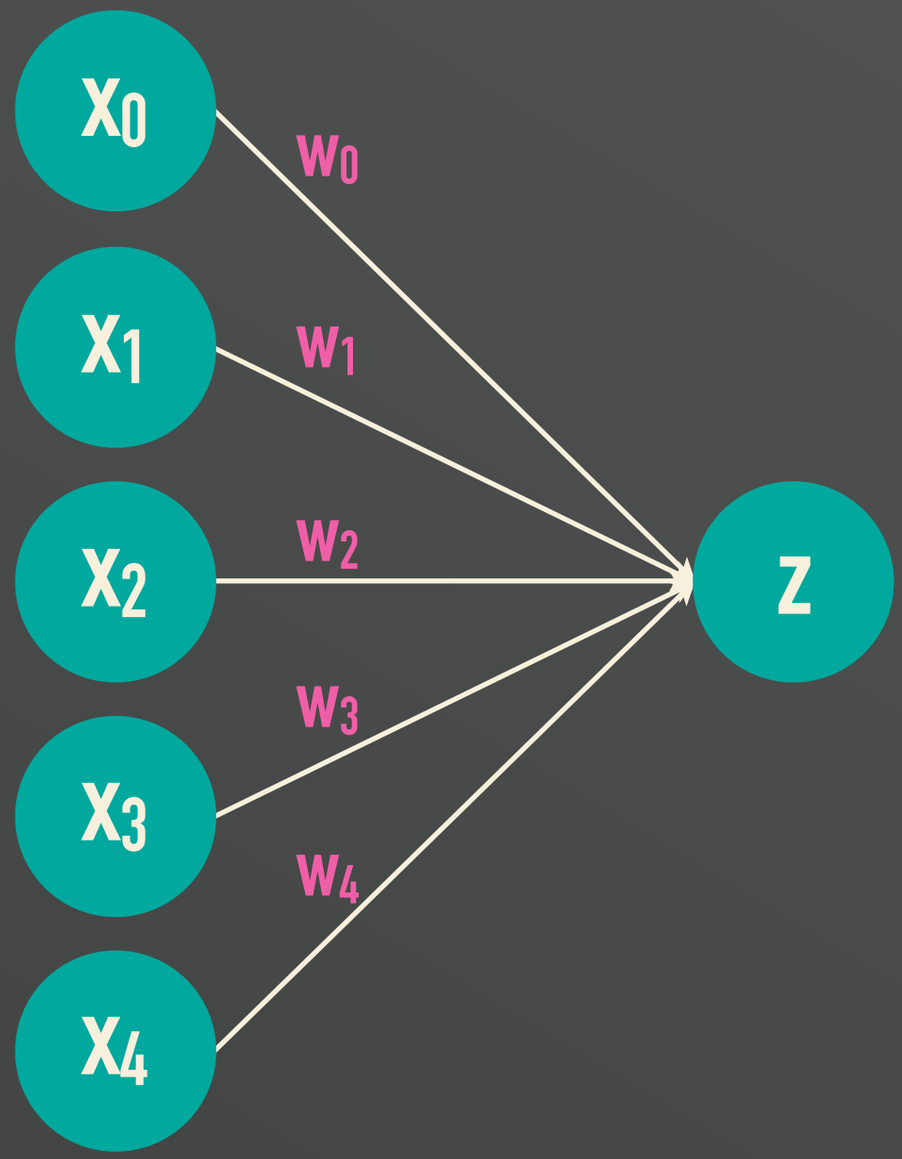
Classic Neuron Activation

- Feed data to Neural Network.
- Data is connected to one of neurons.
- Model attach trainable Weight to each feature.



Classic Neuron Activation

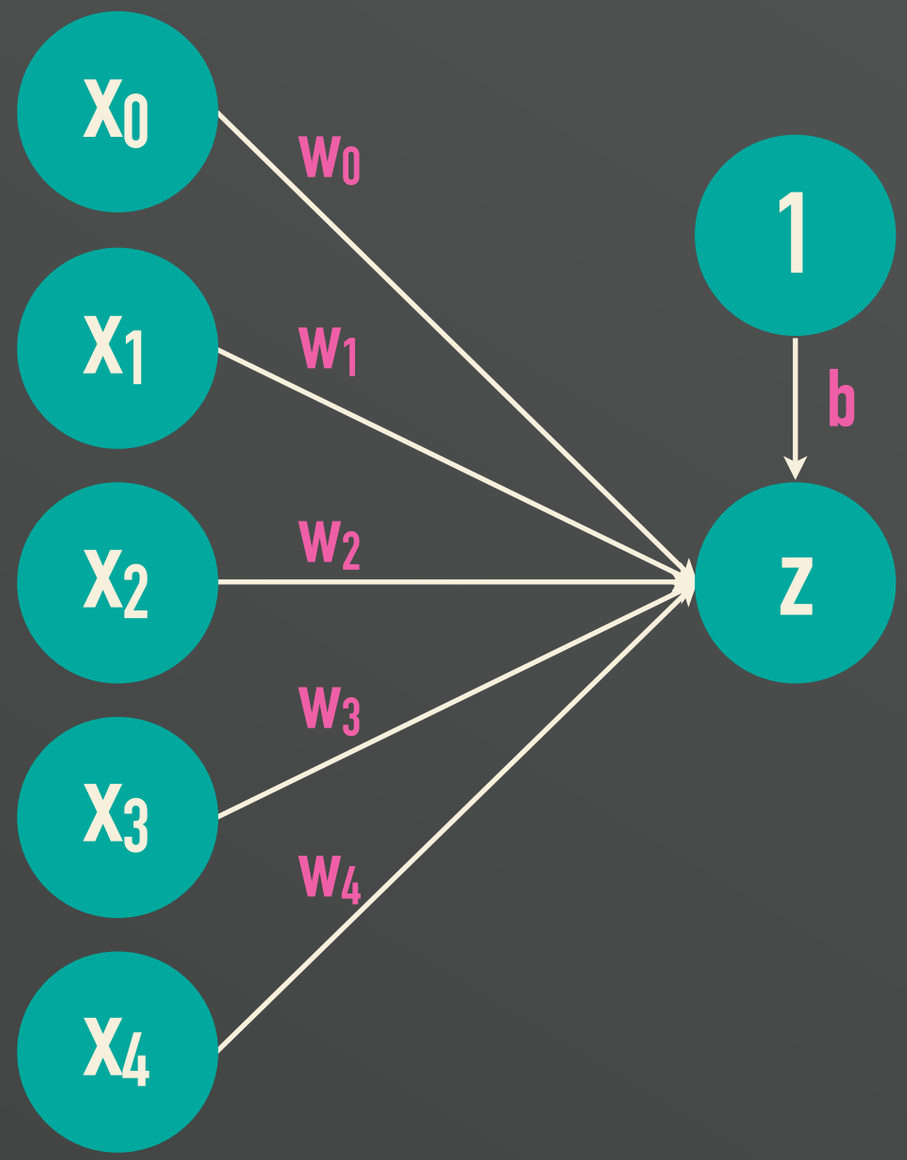
- Feed data to Neural Network.
- Data is connected to one of neurons.
- Model attach trainable Weight to each feature.
- Calculate z which stands for Linear Combination.



$$Z = W_0X_0 + W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4$$

Classic Neuron Activation

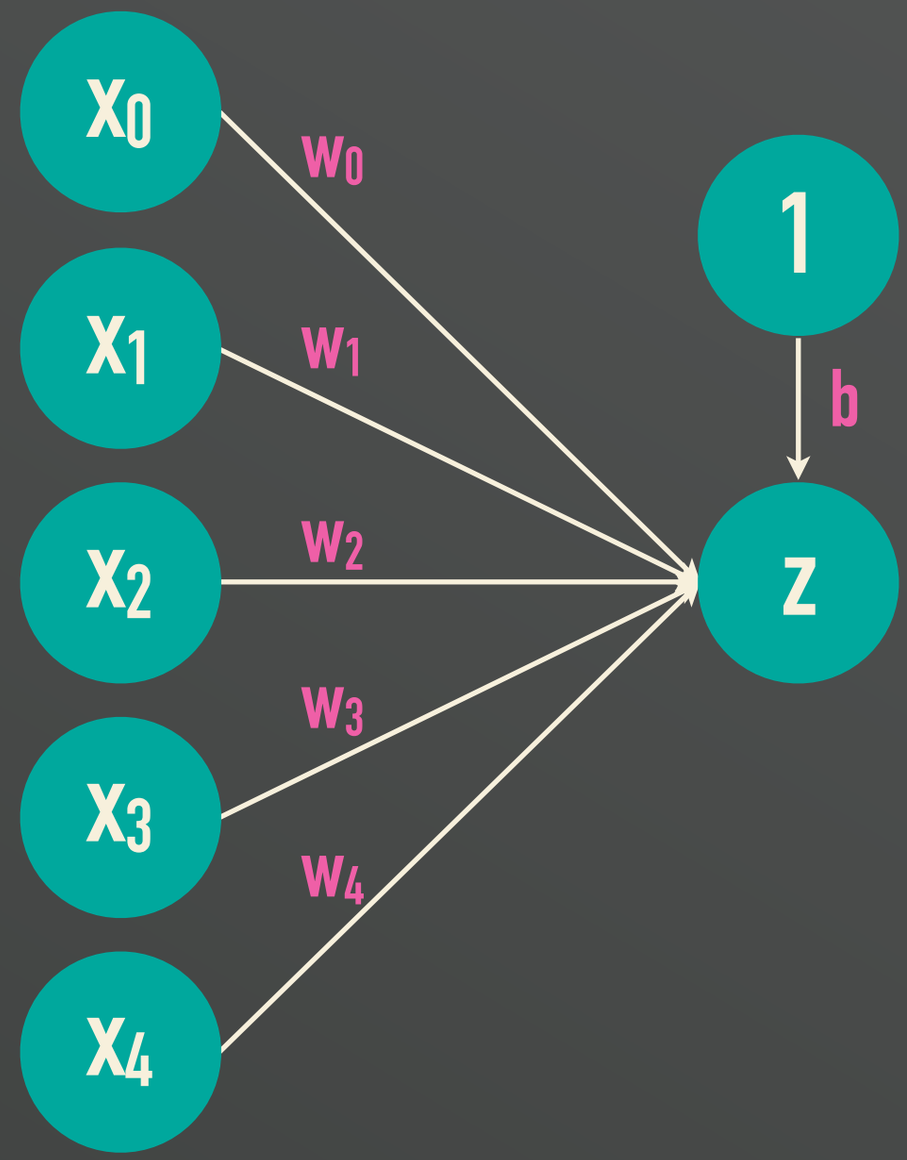
- Feed data to Neural Network.
- Data is connected to one of neurons.
- Model attach trainable Weight to each feature.
- Calculate z which stands for Linear Combination.
- Add trainable Bias to z which is usually omitted on graph.



$$z = W_0X_0 + W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + b$$

Classic Neuron Activation

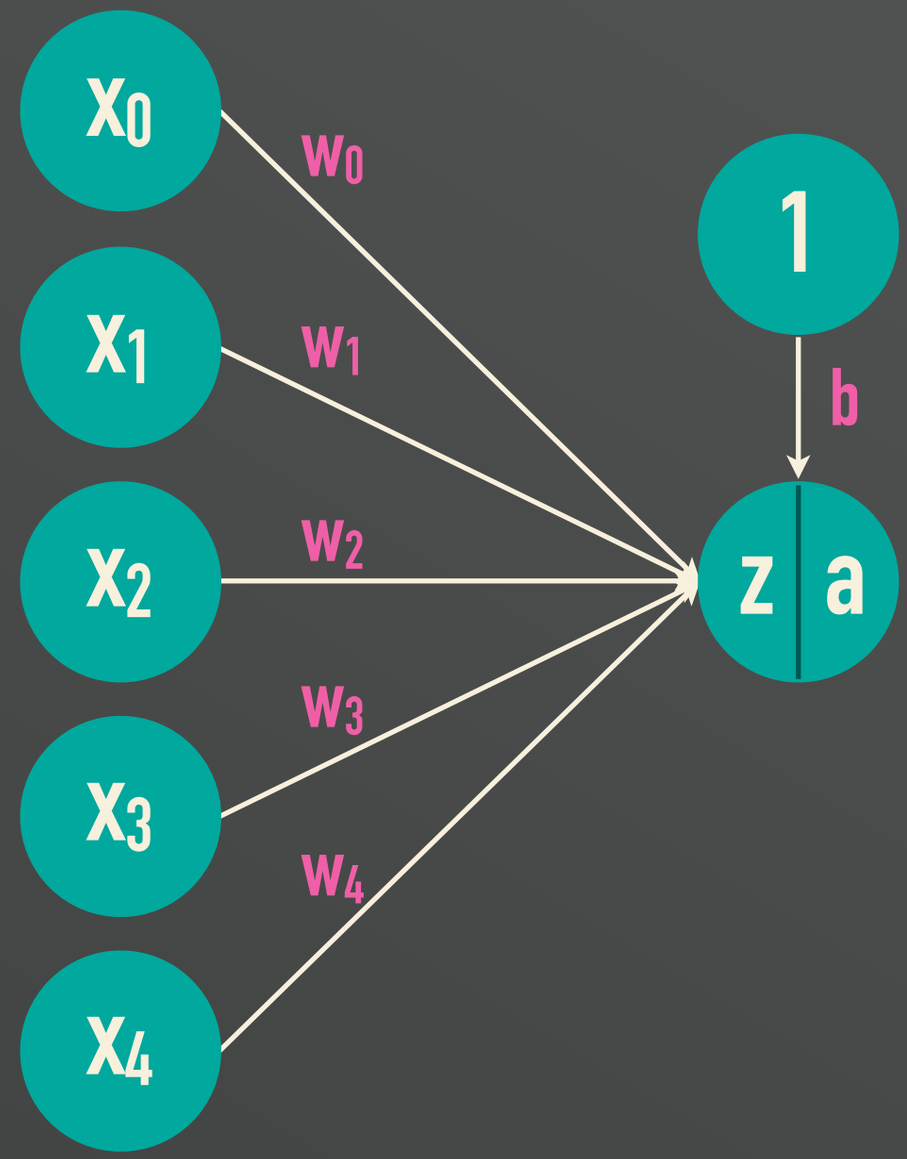
- Feed data to Neural Network.
- Data is connected to one of neurons.
- Model attach trainable Weight to each feature.
- Calculate z which stands for Linear Combination.
- Add trainable Bias to z which is usually omitted on graph.
- Activate z with Activation Function f_a.



$$z = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

Classic Neuron Activation

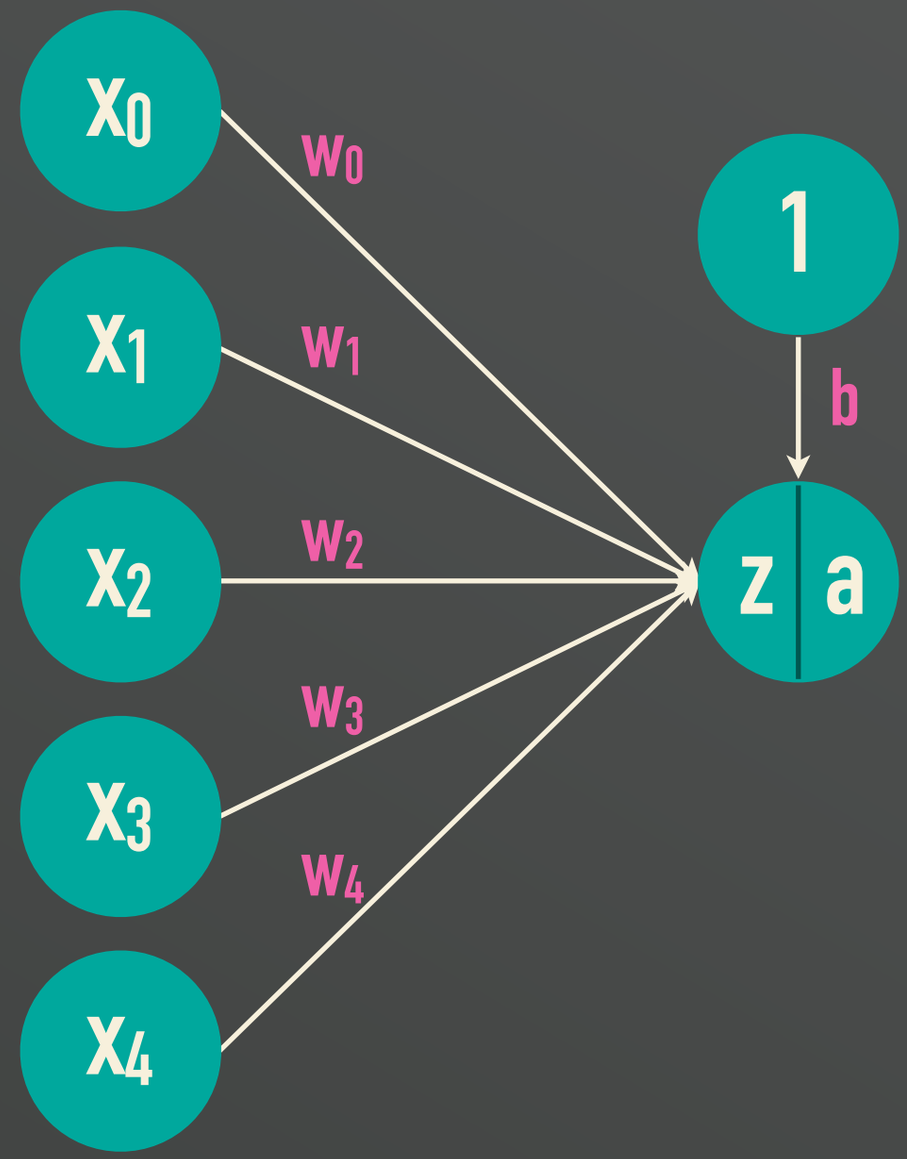
- Feed data to Neural Network.
- Data is connected to one of neurons.
- Model attach trainable Weight to each feature.
- Calculate z which stands for Linear Combination.
- Add trainable Bias to z which is usually omitted on graph.
- Activate z with Activation Function f_a.



$$z = w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

Classic Neuron Activation

- Feed data to Neural Network.
- Data is connected to one of neurons.
- Model attach trainable Weight to each feature.
- Calculate z which stands for Linear Combination.
- Add trainable Bias to z which is usually omitted on graph.
- Activate z with Activation Function f_a.



$$a = f_a(w_0x_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b)$$

Input Layer
Our Features

x_0

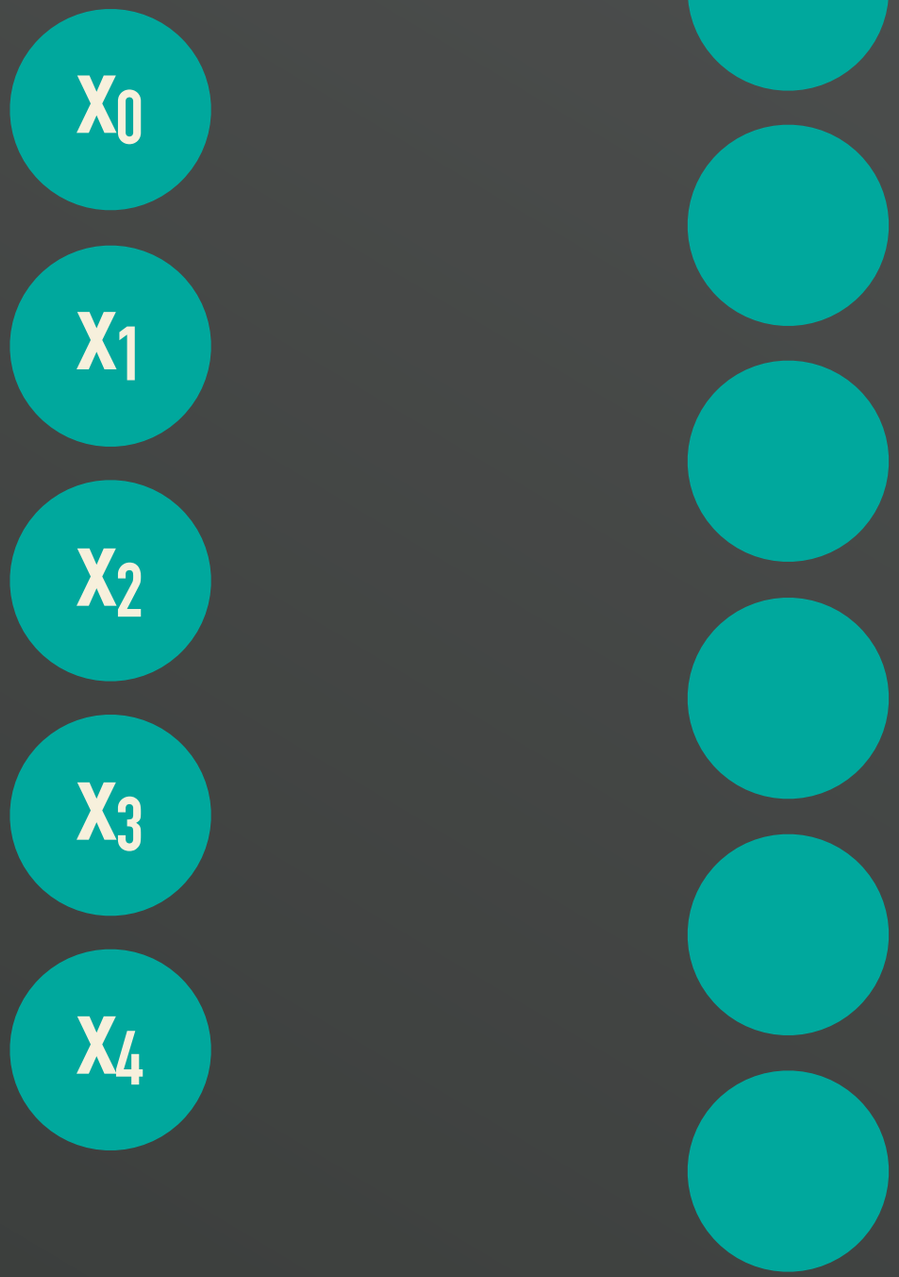
x_1

x_2

x_3

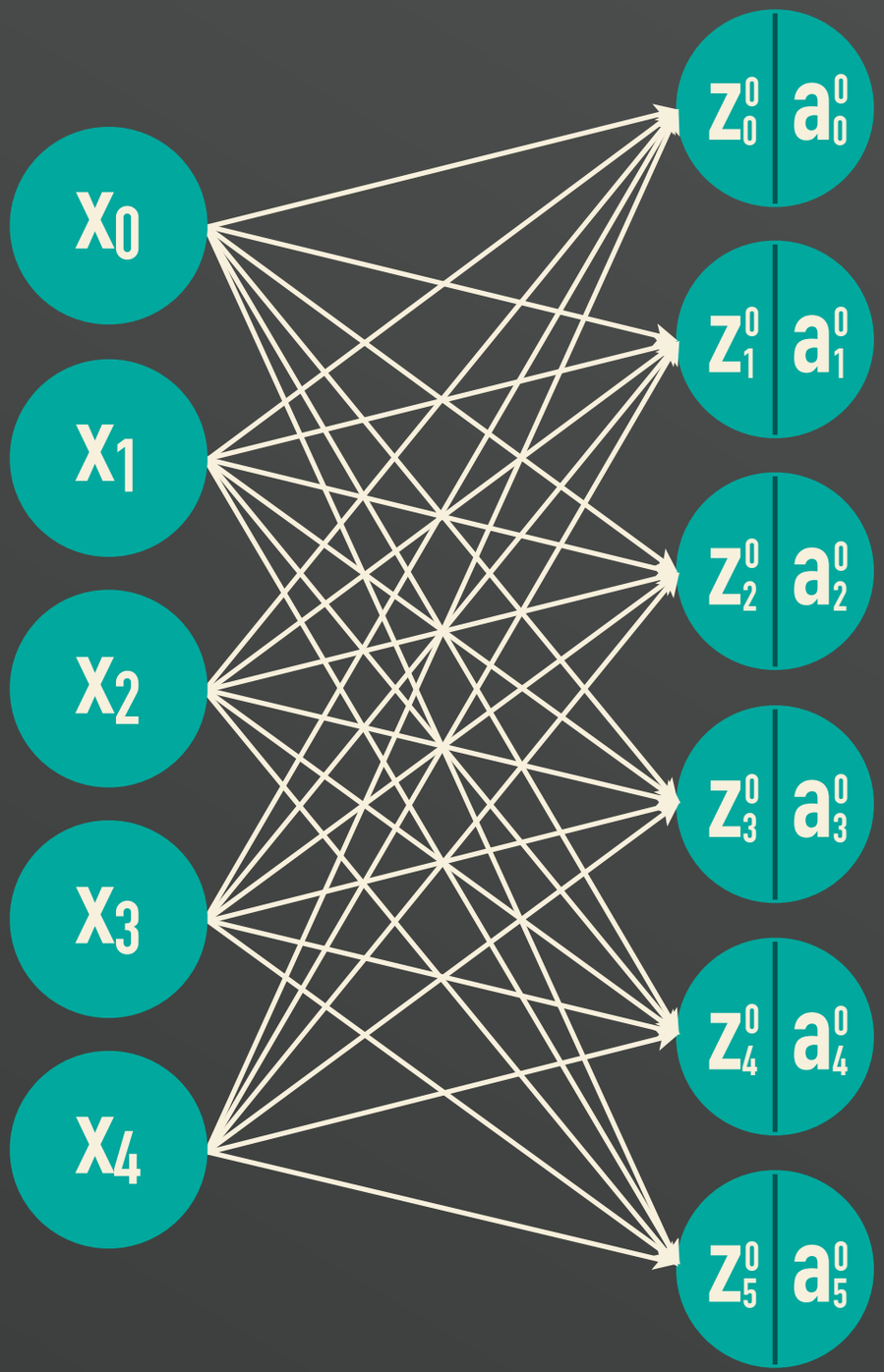
x_4

Input Layer
Our Features



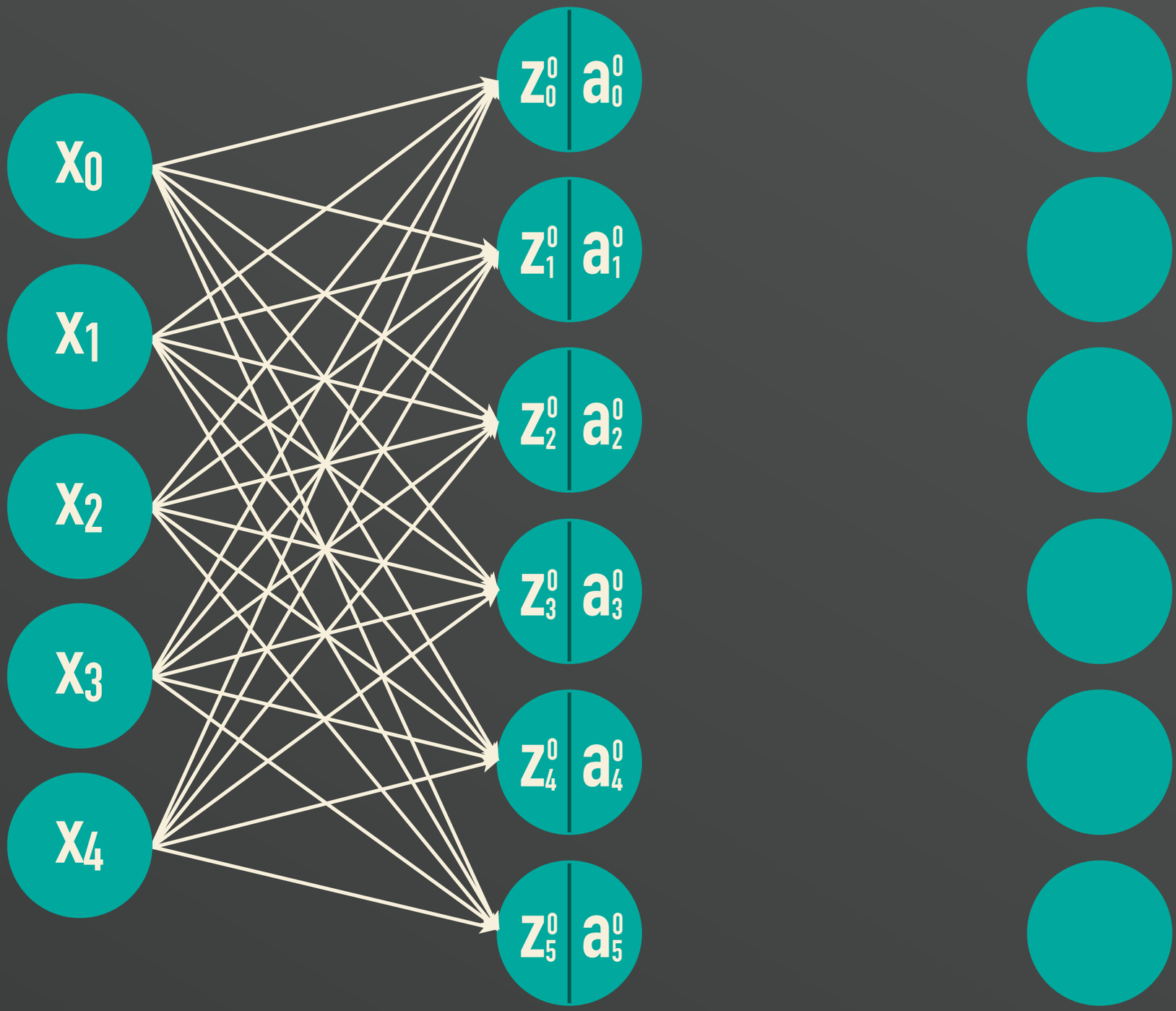
Input Layer
Our Features

Hidden Layer 0
New Features



Input Layer
Our Features

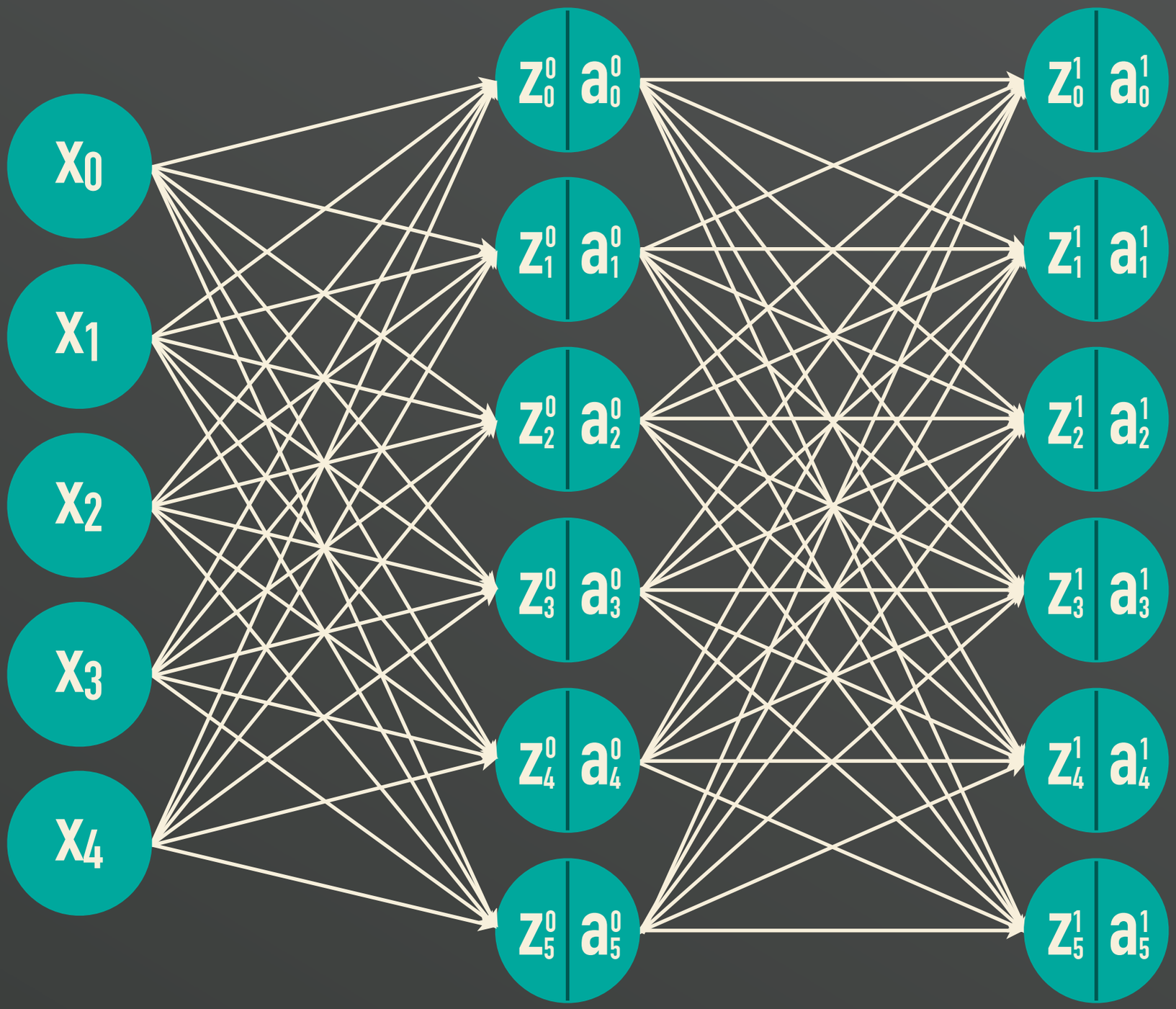
Hidden Layer 0
New Features



Input Layer
Our Features

Hidden Layer 0
New Features

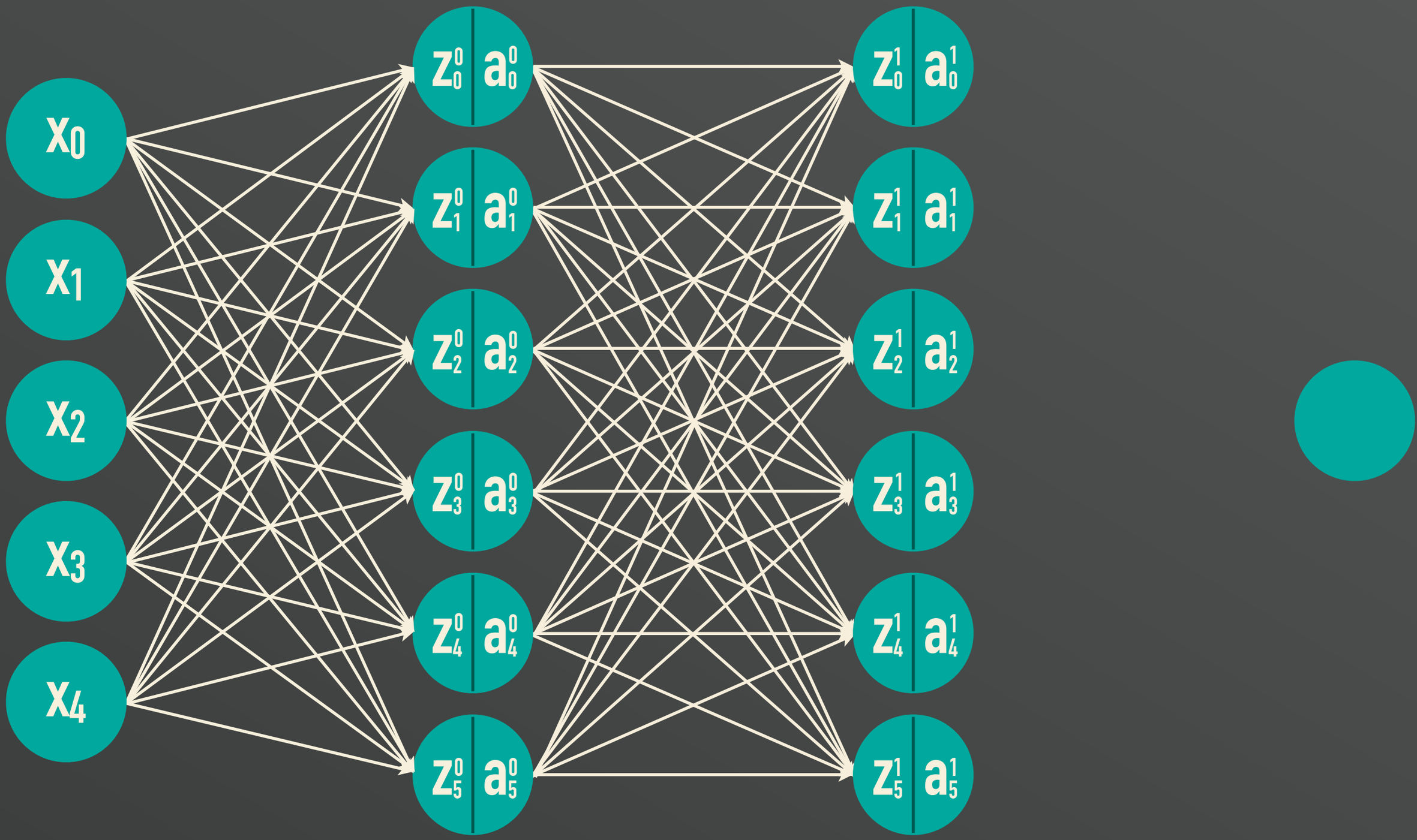
Hidden Layer 1
New Features



Input Layer
Our Features

Hidden Layer 0
New Features

Hidden Layer 1
New Features

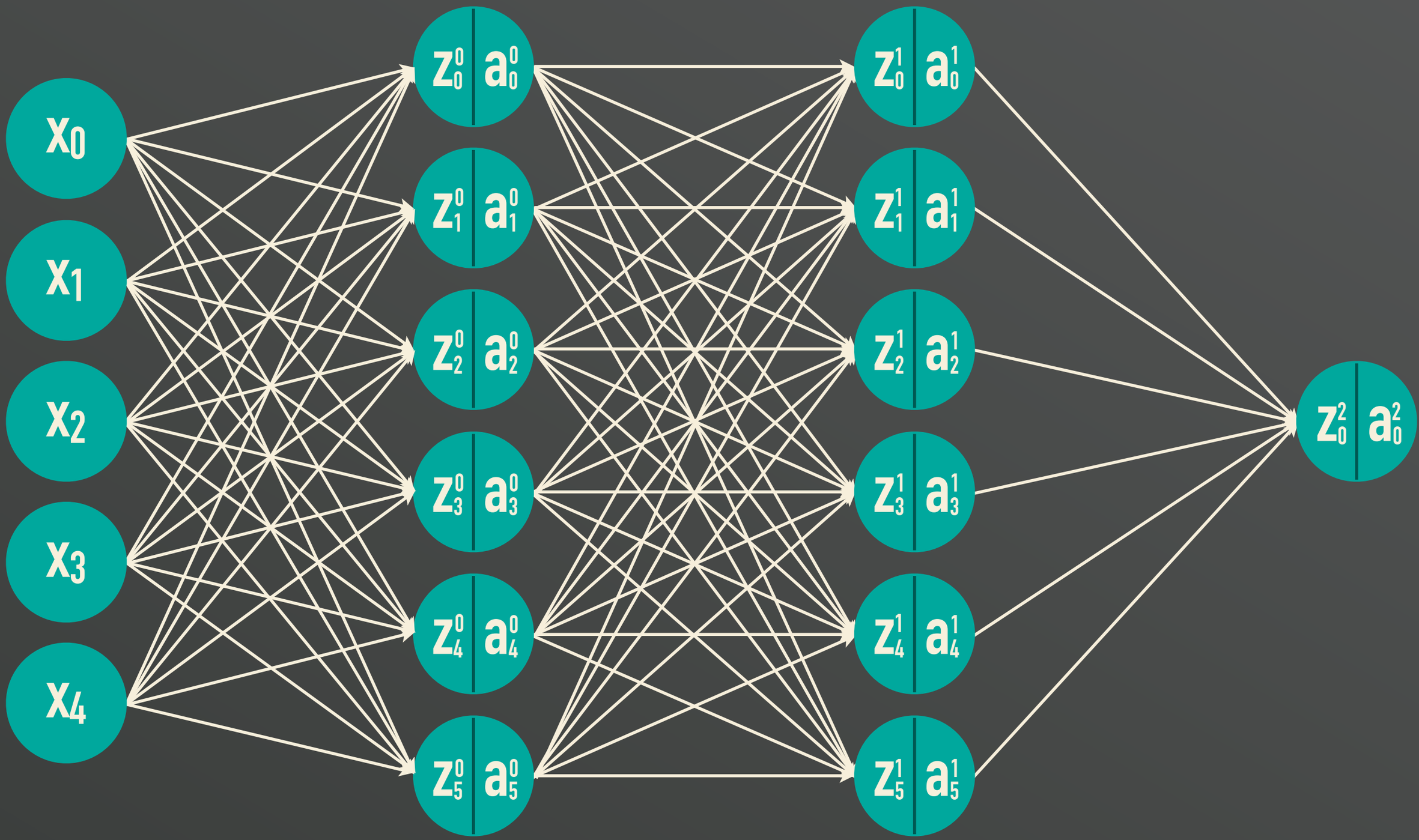


Input Layer
Our Features

Hidden Layer 0
New Features

Hidden Layer 1
New Features

Output Layer
Result

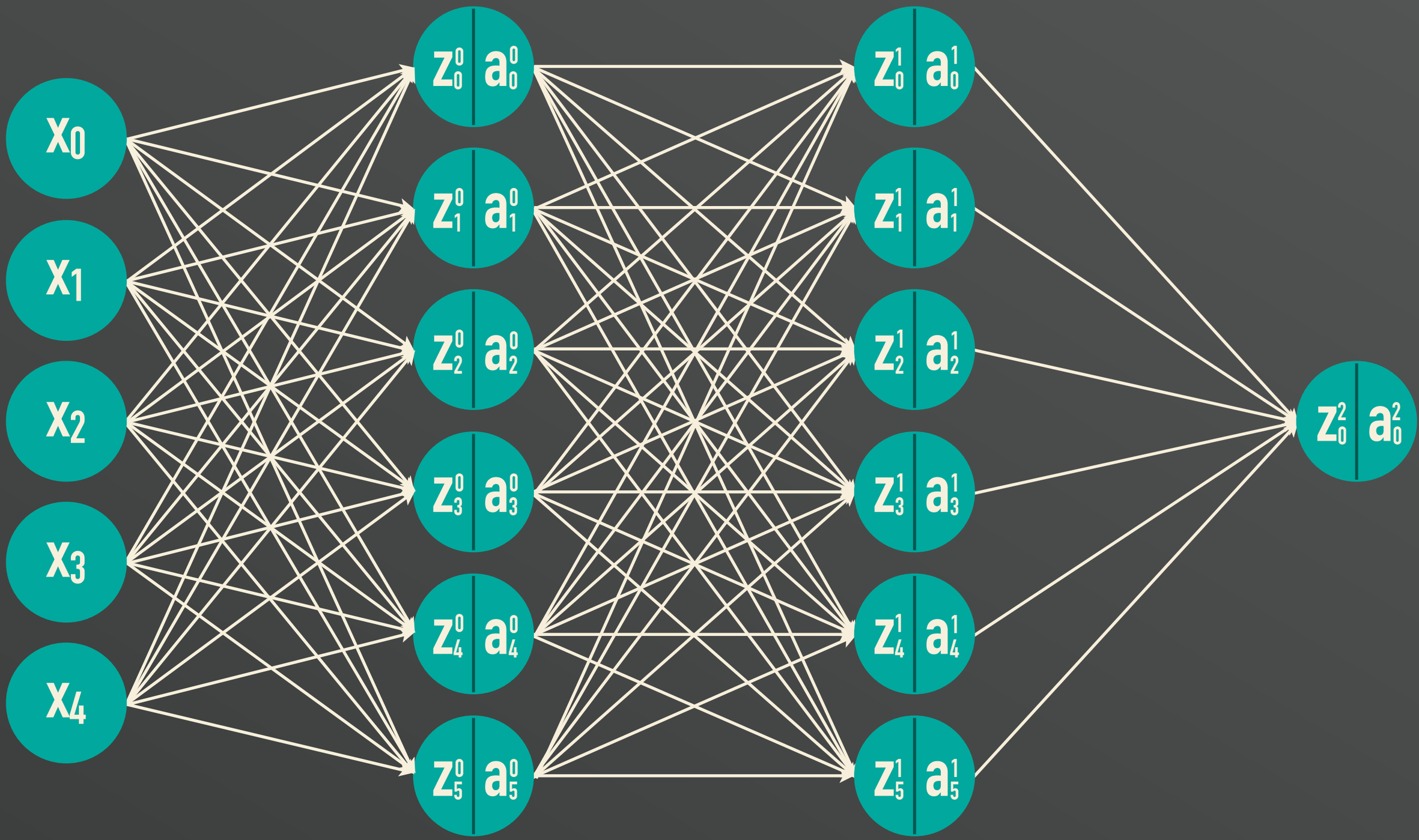


Input Layer
Our Features

Hidden Layer 0
New Features

Hidden Layer 1
New Features

Output Layer
Result



Classic Scenario

Input Layer
Our Features

x_0

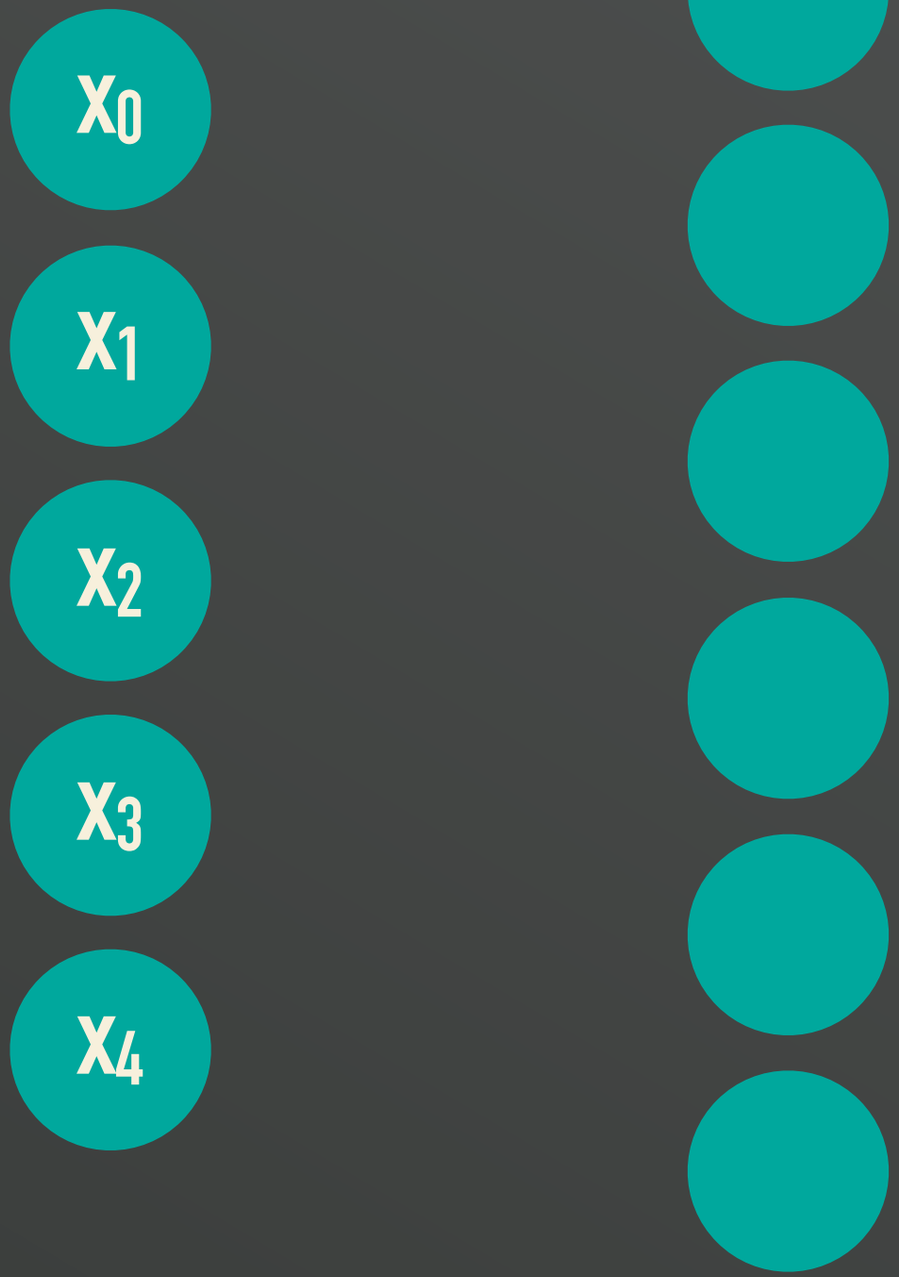
x_1

x_2

x_3

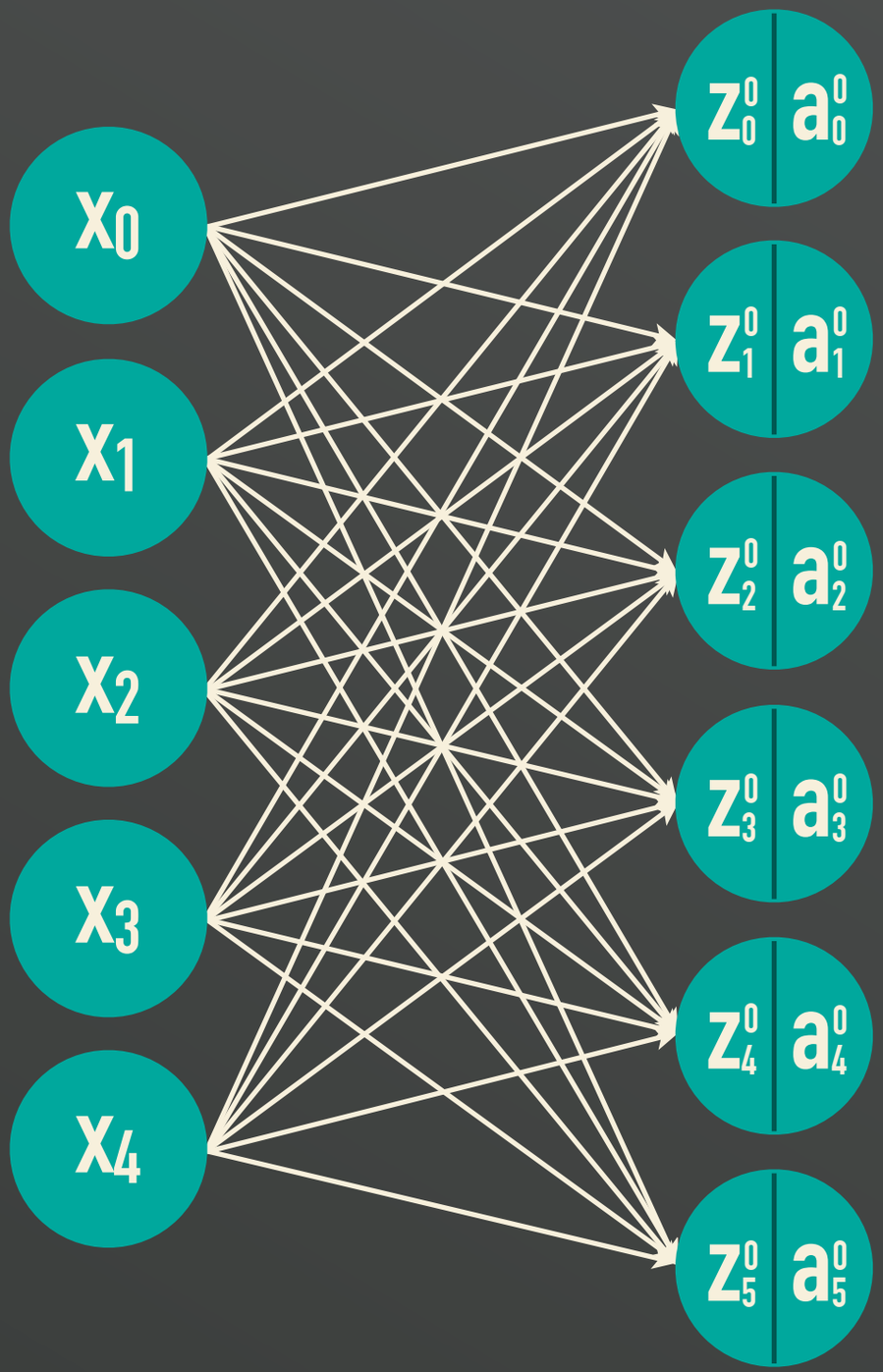
x_4

Input Layer
Our Features

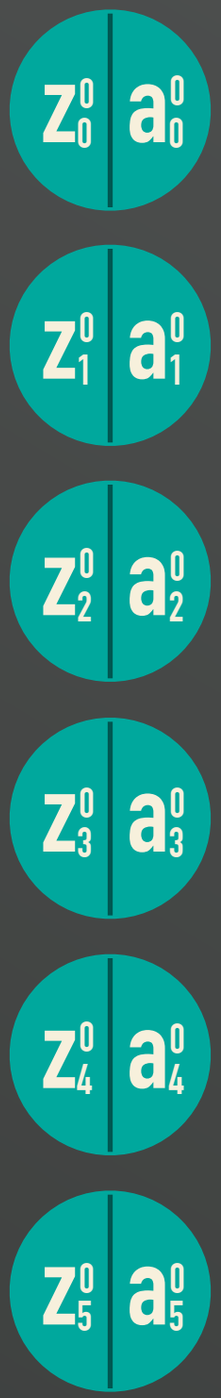


Input Layer
Our Features

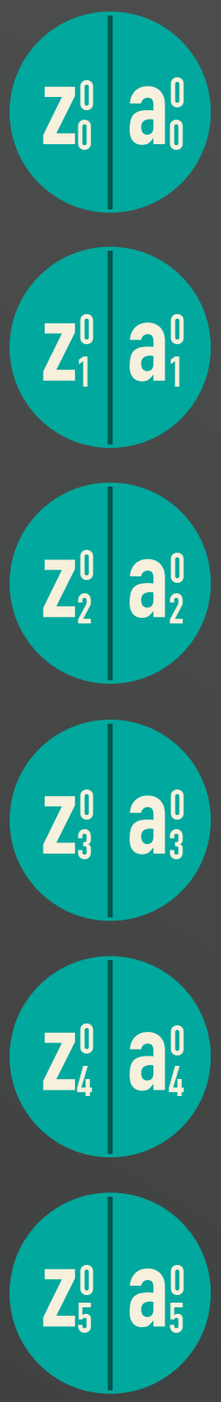
Hidden Layer 0
New Features



Hidden Layer 0
New Features

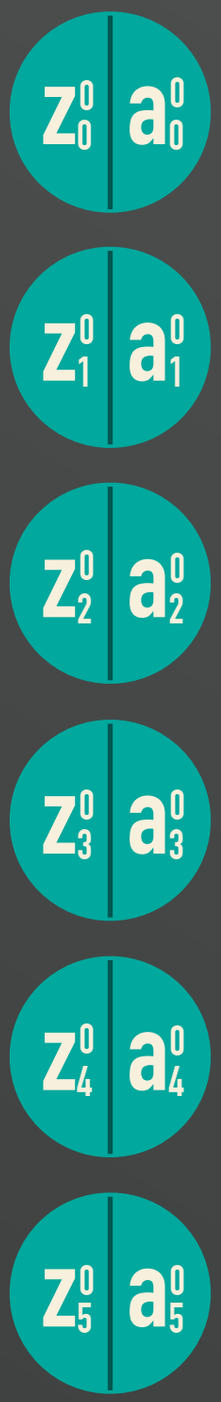


Input Layer
Features



What if we think about
this layer as a new input layer?

Input Layer
Features



What if we think about
this layer as a new input layer?

We can Normalize it!

x_0

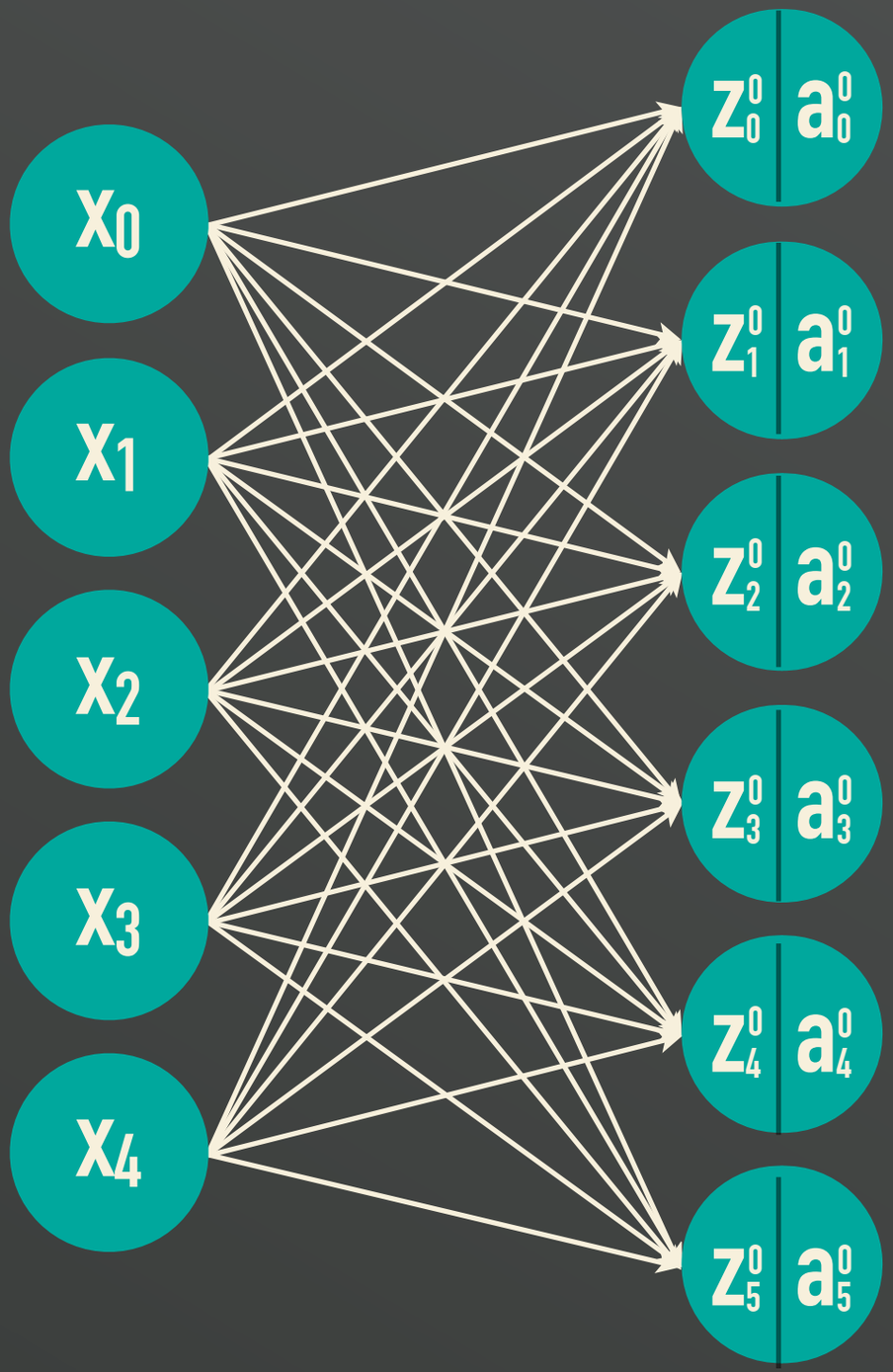
x_1

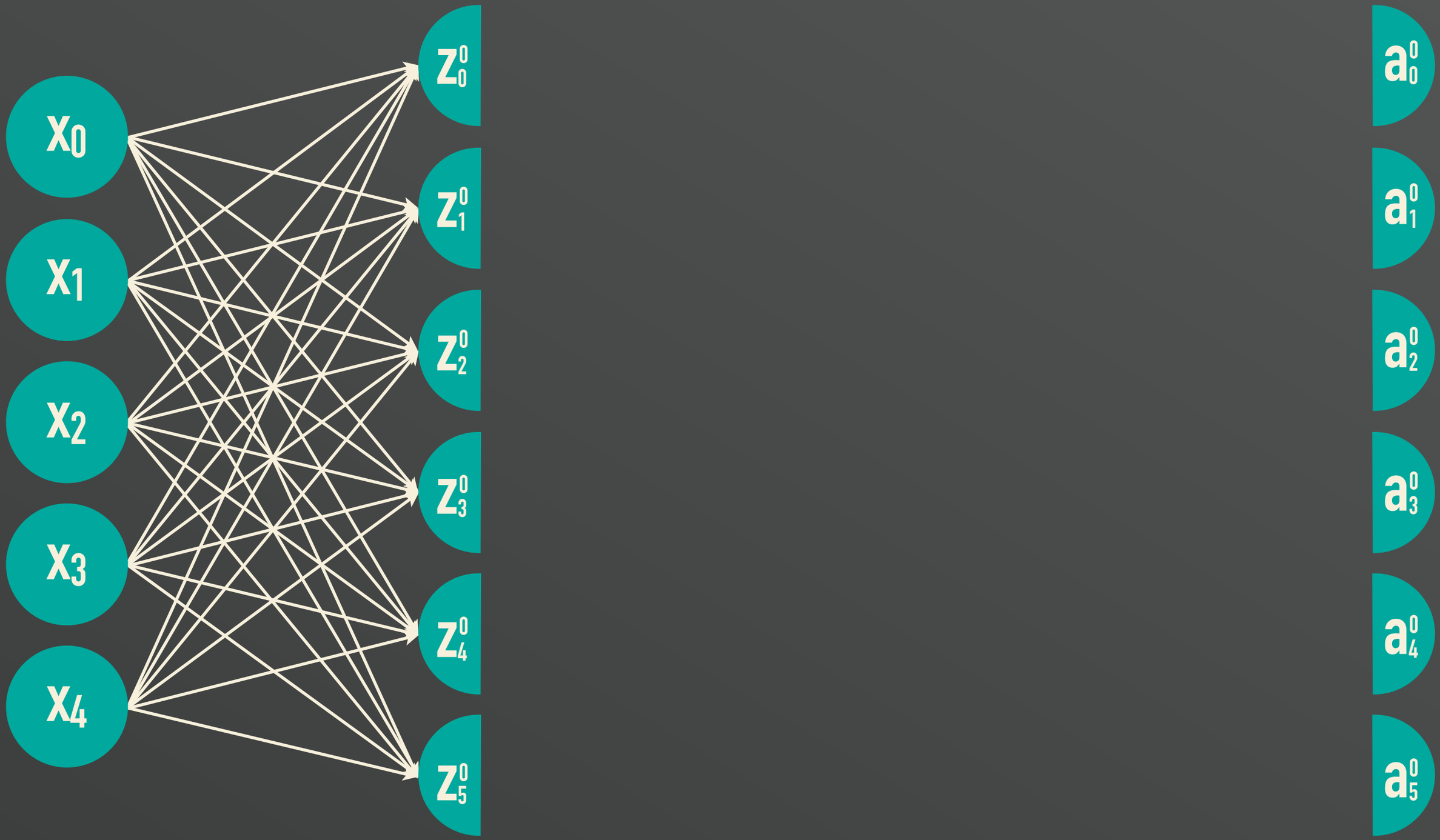
x_2

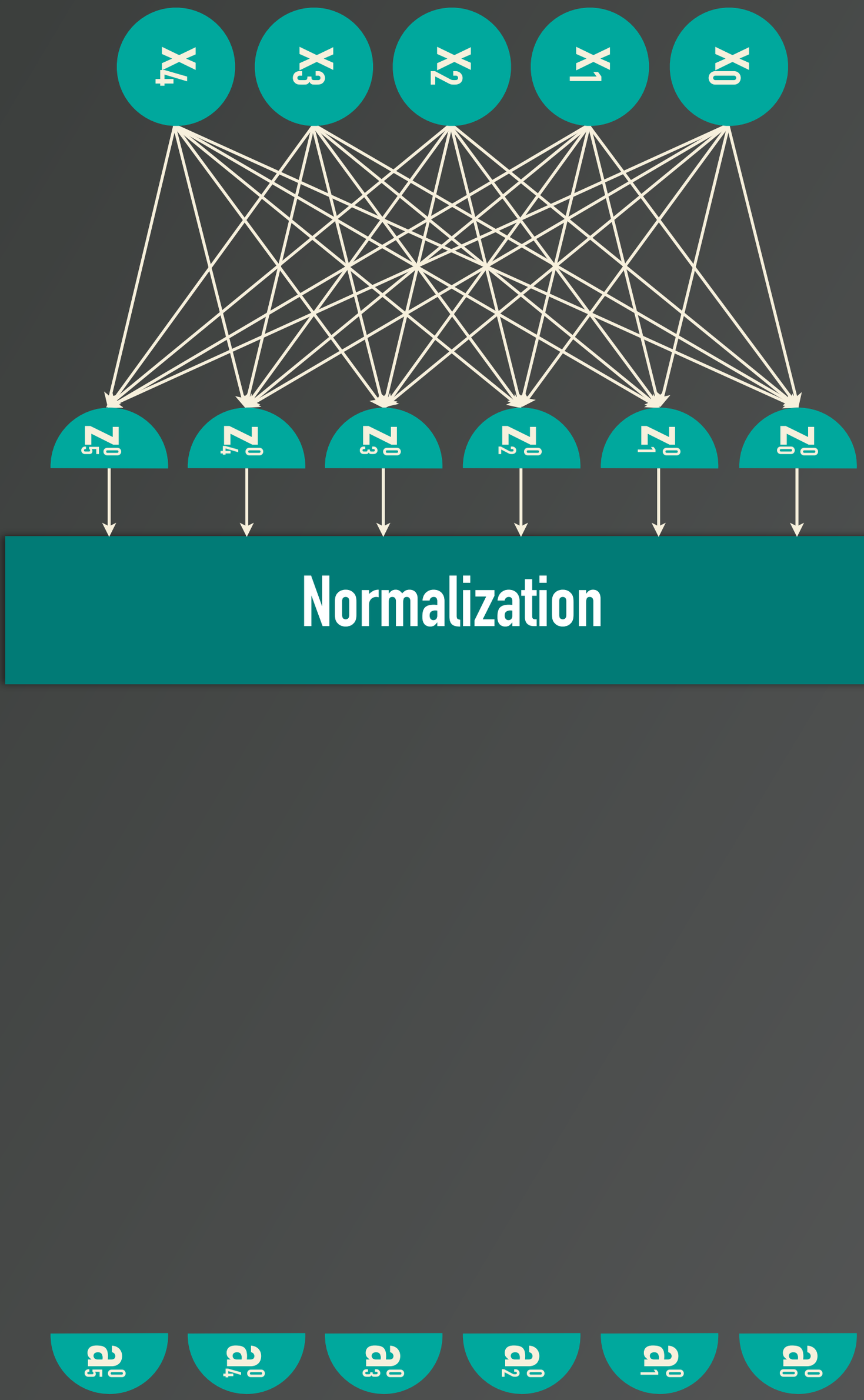
x_3

x_4

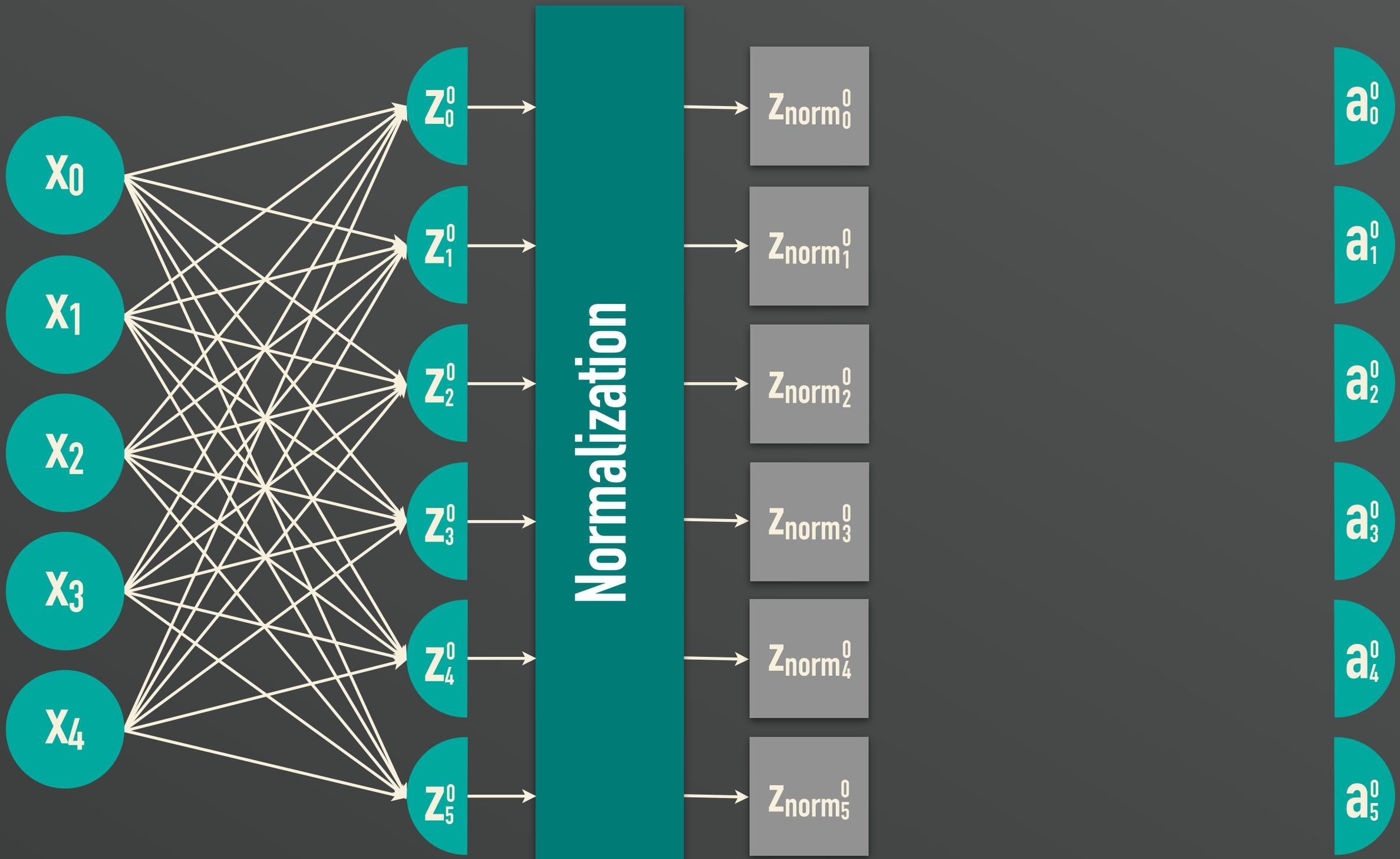

x_0
x_1
x_2
x_3
x_4


x_0
x_1
x_2
x_3
x_4
z_0^0 a_0^0
z_1^0 a_1^0
z_2^0 a_2^0
z_3^0 a_3^0
z_4^0 a_4^0
z_5^0 a_5^0

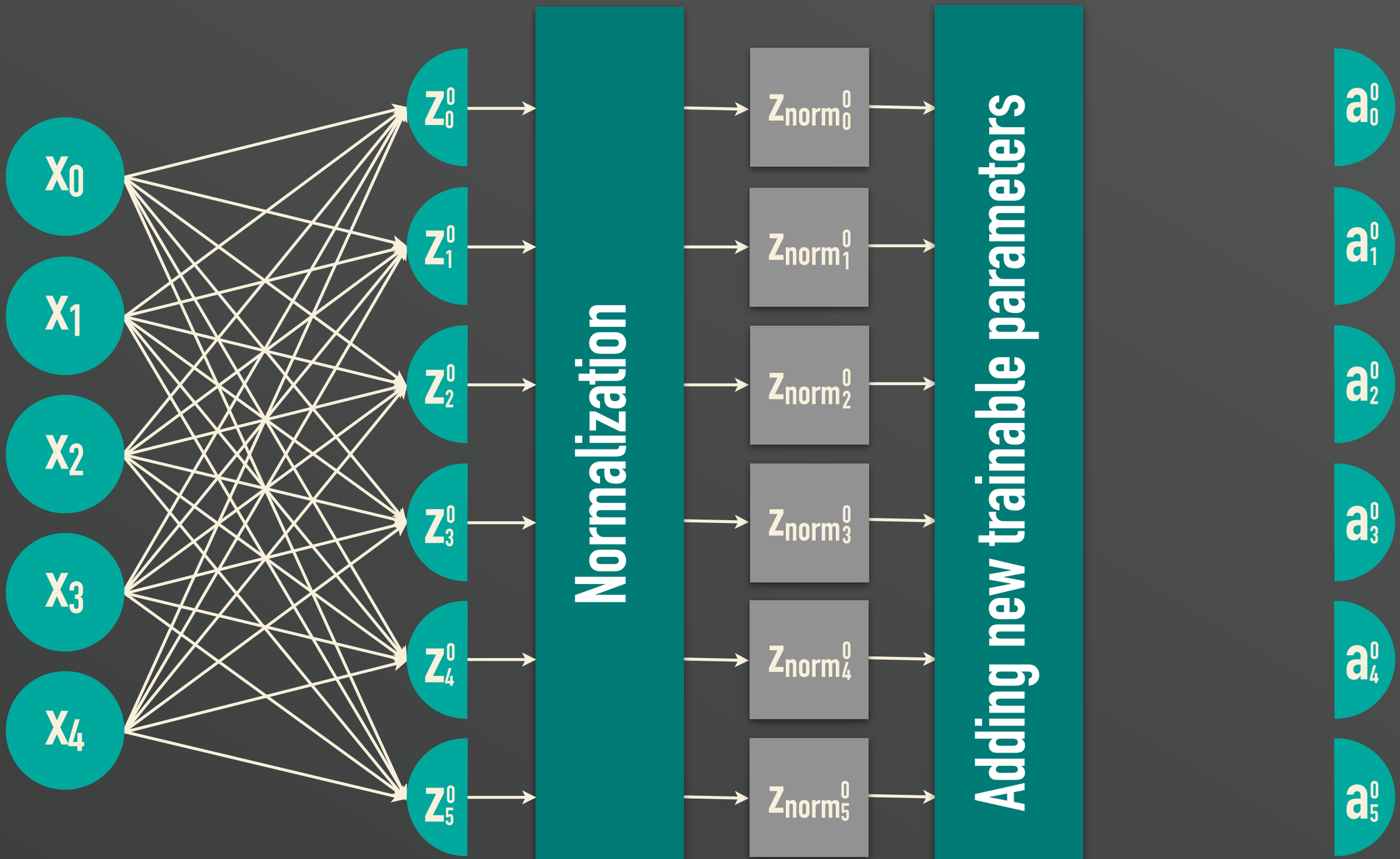

x_0
x_1
x_2
x_3
x_4
z_0^0
z_1^0
z_2^0
z_3^0
z_4^0
z_5^0
a_0^0
a_1^0
a_2^0
a_3^0
a_4^0
a_5^0


X4
X3
X2
X1
X0
Z50
Z40
Z30
Z20
Z10
Z00
Normalization
a50
a40
a30
a20
a10
a00

$$z^n_{norm} = \frac{z^n - mean(batch)}{\sqrt{variance(batch) + \text{very\_small\_number}}}$$

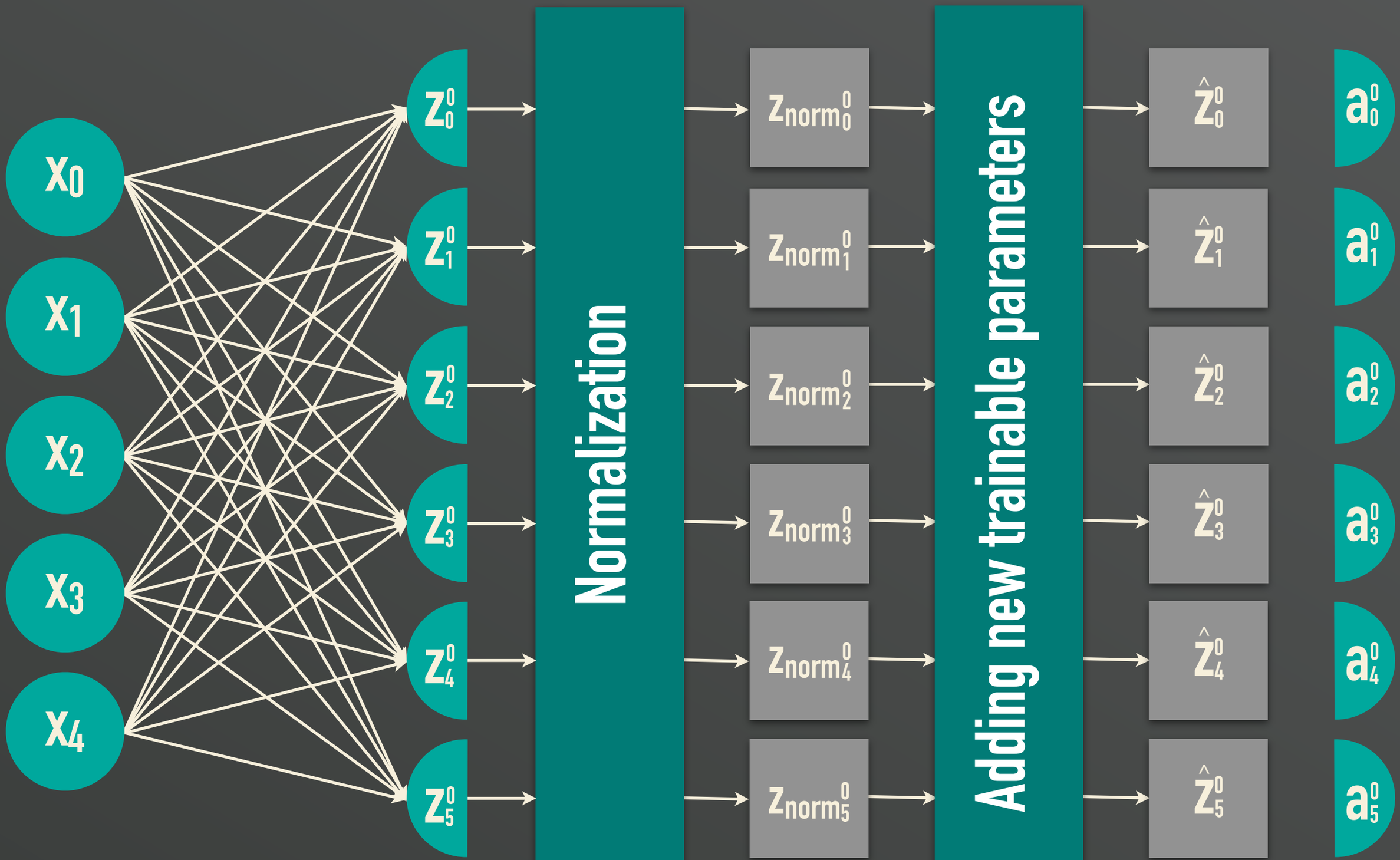


$$z^n_{norm} = \frac{z^n - mean(batch)}{\sqrt{variance(batch) + \text{very\_small\_number}}}$$



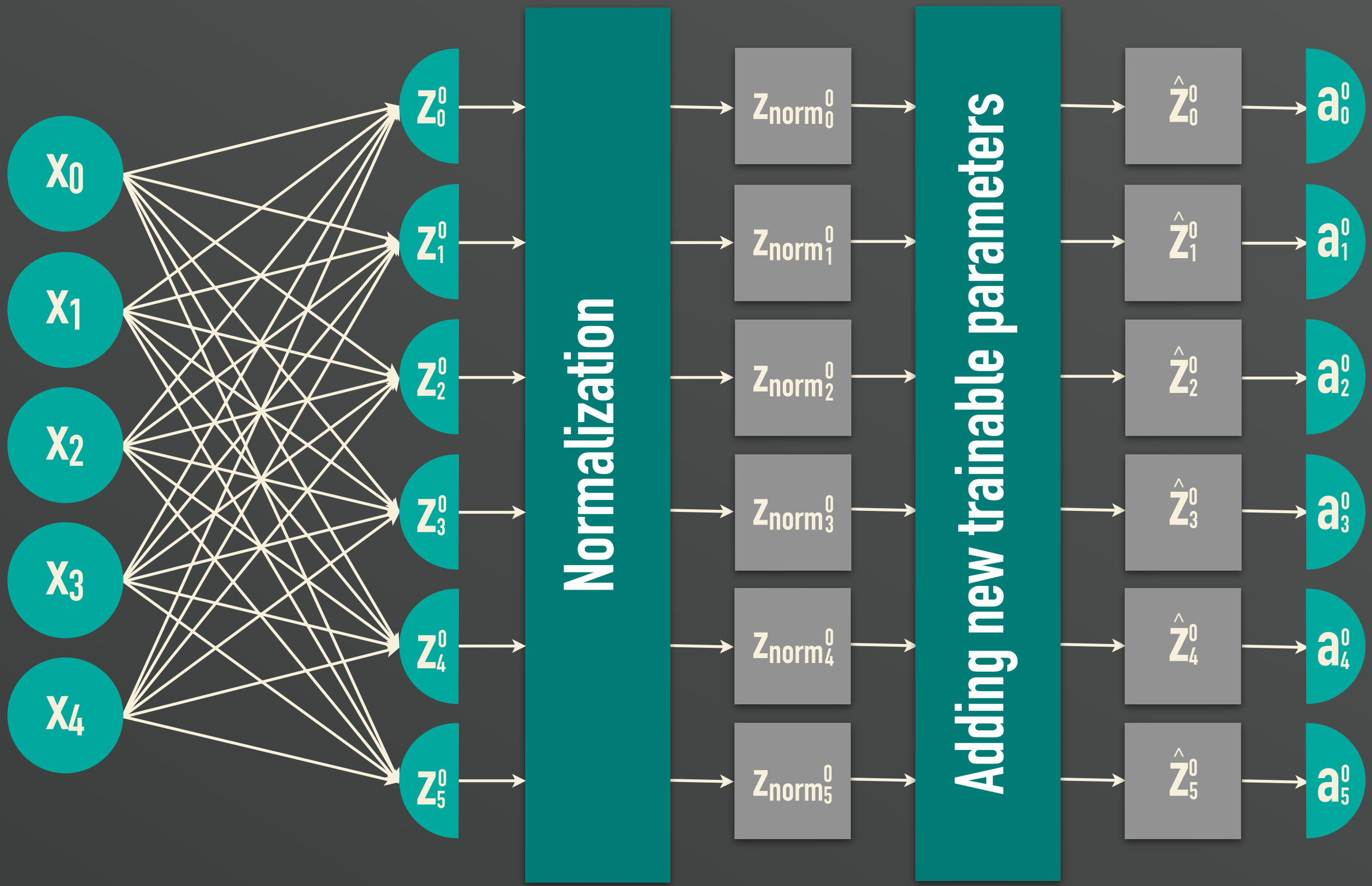
$$z^n_{norm} = \frac{z^n - mean(batch)}{\sqrt{variance(batch) + \text{very\_small\_number}}}$$

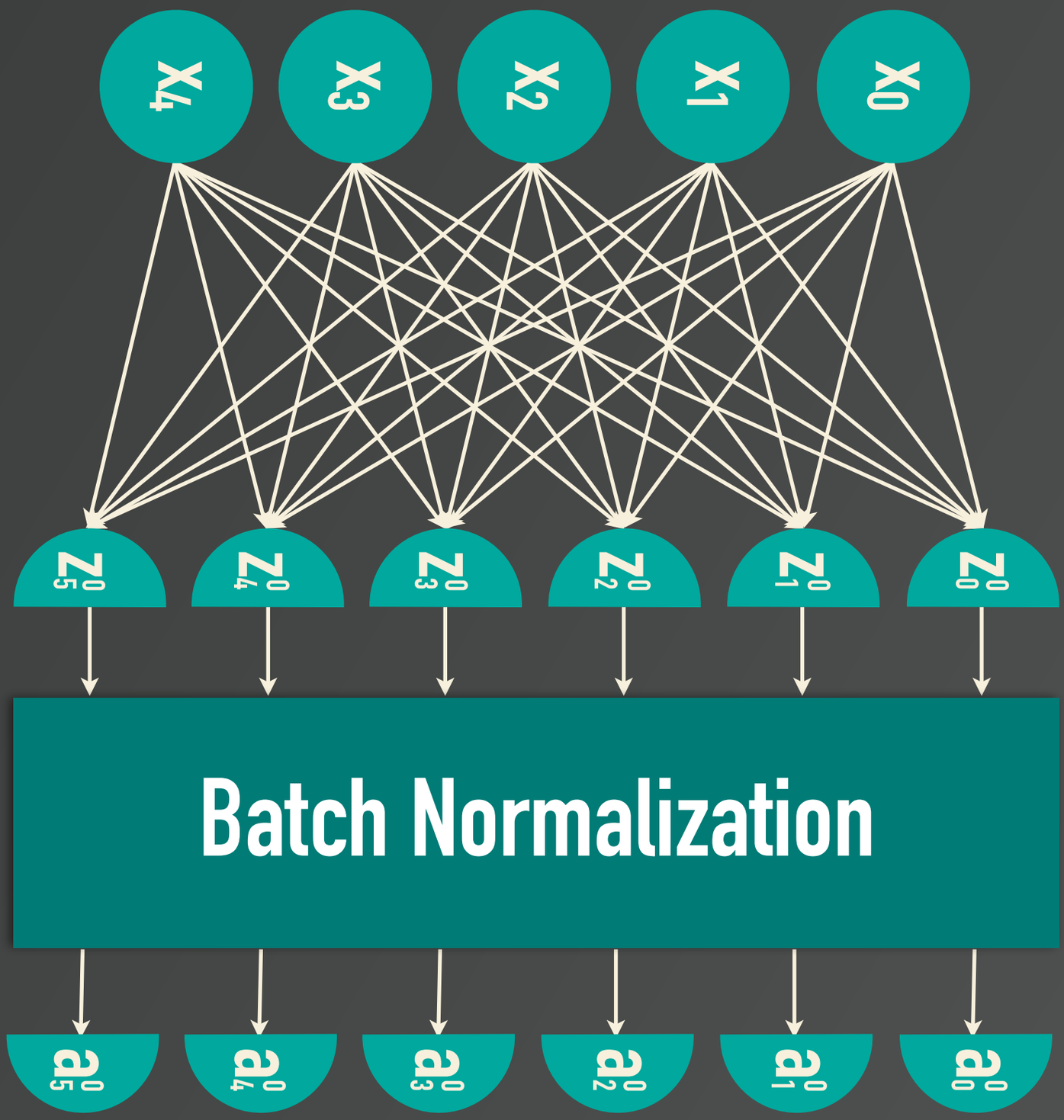
$$\hat{z}^n = \gamma^n z^n_{norm} + \beta^n$$

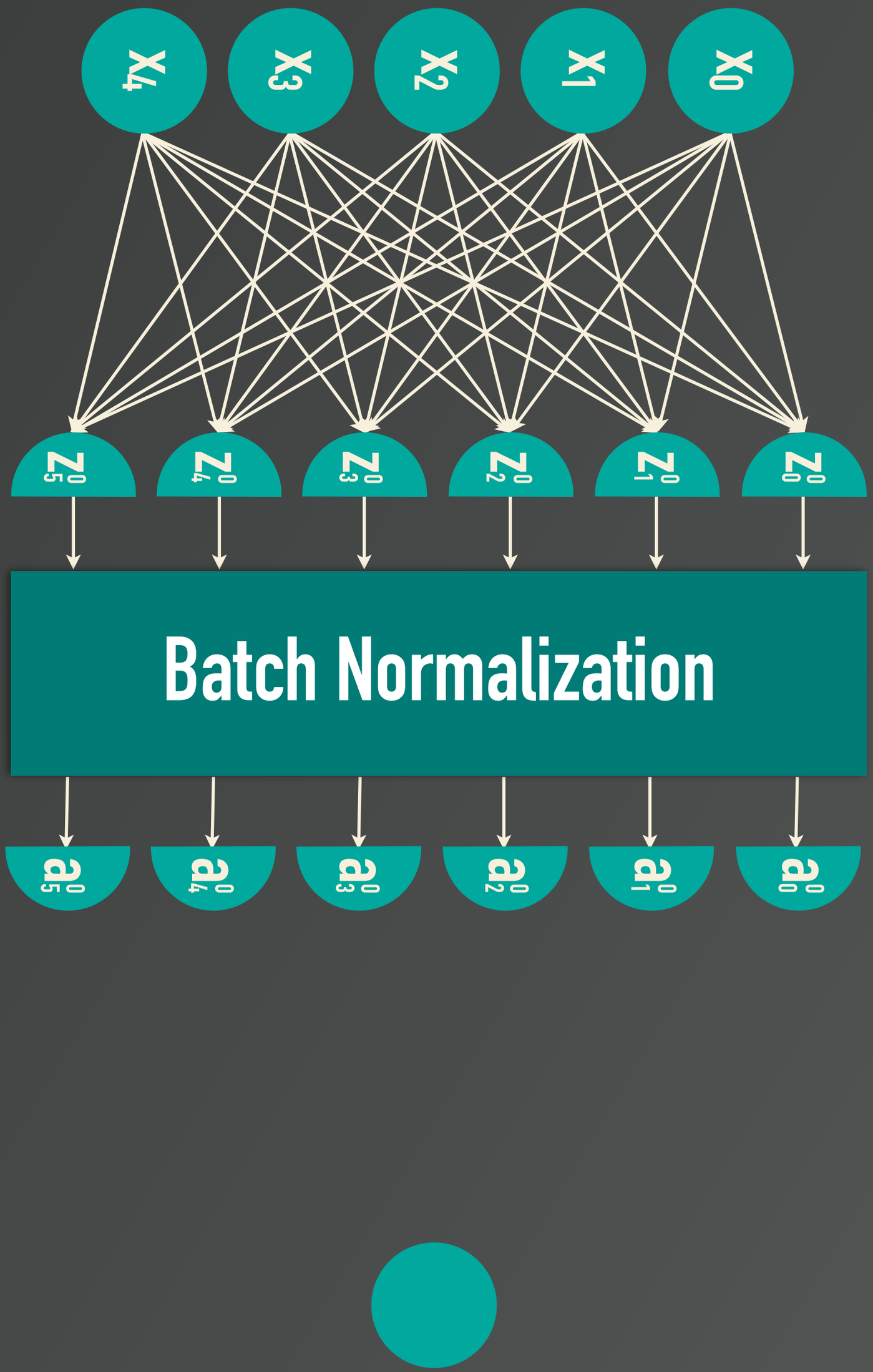


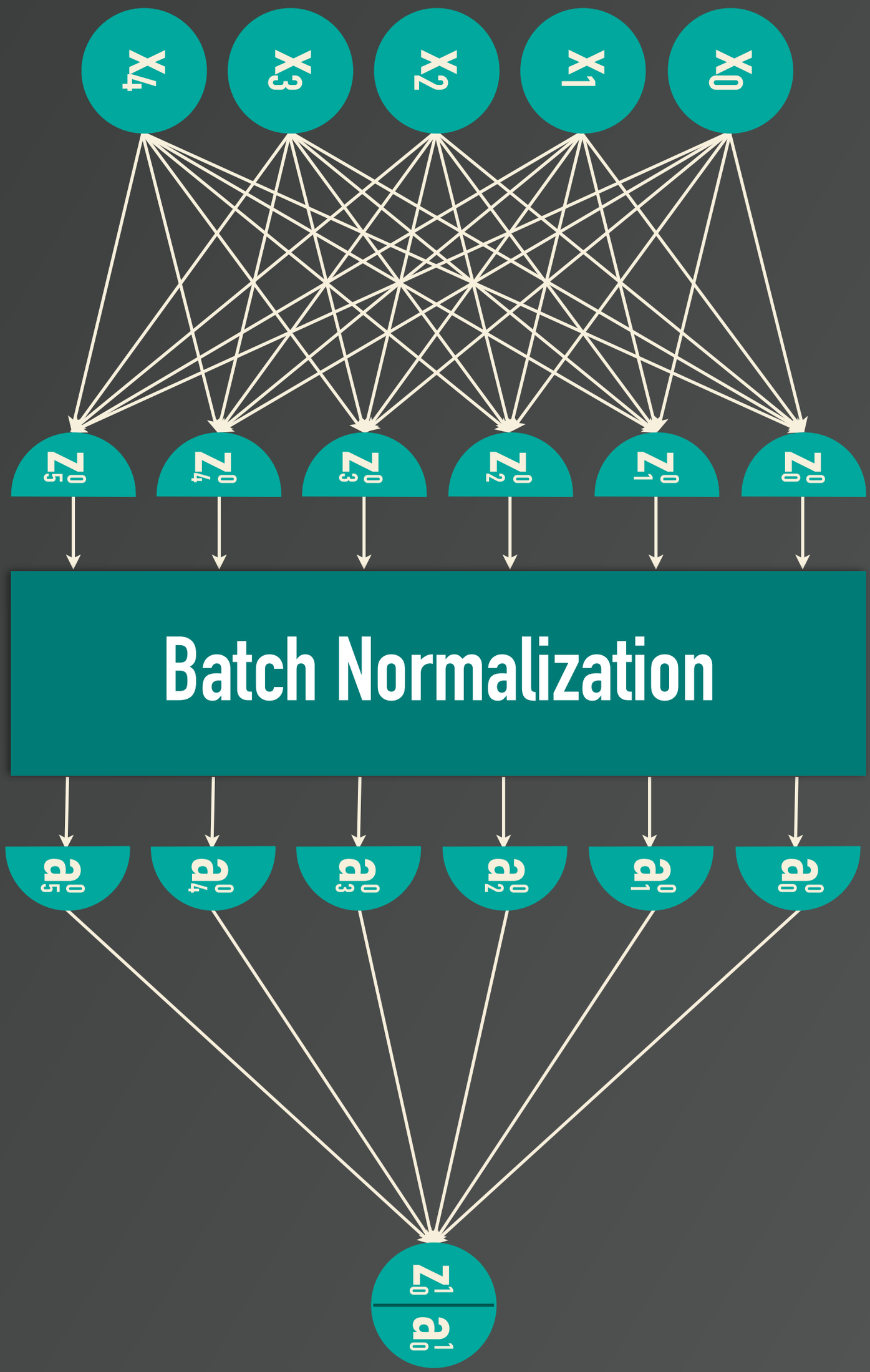
$$z^n_{norm} = \frac{z^n - mean(batch)}{\sqrt{variance(batch) + \text{very\_small\_number}}}$$

$$\hat{z}^n = \gamma^n z^n_{norm} + \beta^n$$




x_4
x_3
x_2
x_1
x_0
z_{05}
z_{04}
z_{03}
z_{02}
z_{01}
z_{00}
Batch Normalization
a_{05}
a_{04}
a_{03}
a_{02}
a_{01}
a_{00}


x_4
x_3
x_2
x_1
x_0
z_{05}
z_{04}
z_{03}
z_{02}
z_{01}
z_{00}
Batch Normalization
a_{05}
a_{04}
a_{03}
a_{02}
a_{01}
a_{00}


x_4
x_3
x_2
x_1
x_0
z_{50}
z_{40}
z_{30}
z_{20}
z_{10}
z_{00}
Batch Normalization
a_{05}
a_{04}
a_{03}
a_{02}
a_{01}
a_{00}
z_{01}
a_{01}

Trainable Parameters

Trainable Parameters

Input Layer

Linear

Batch Normalization

Activation

...

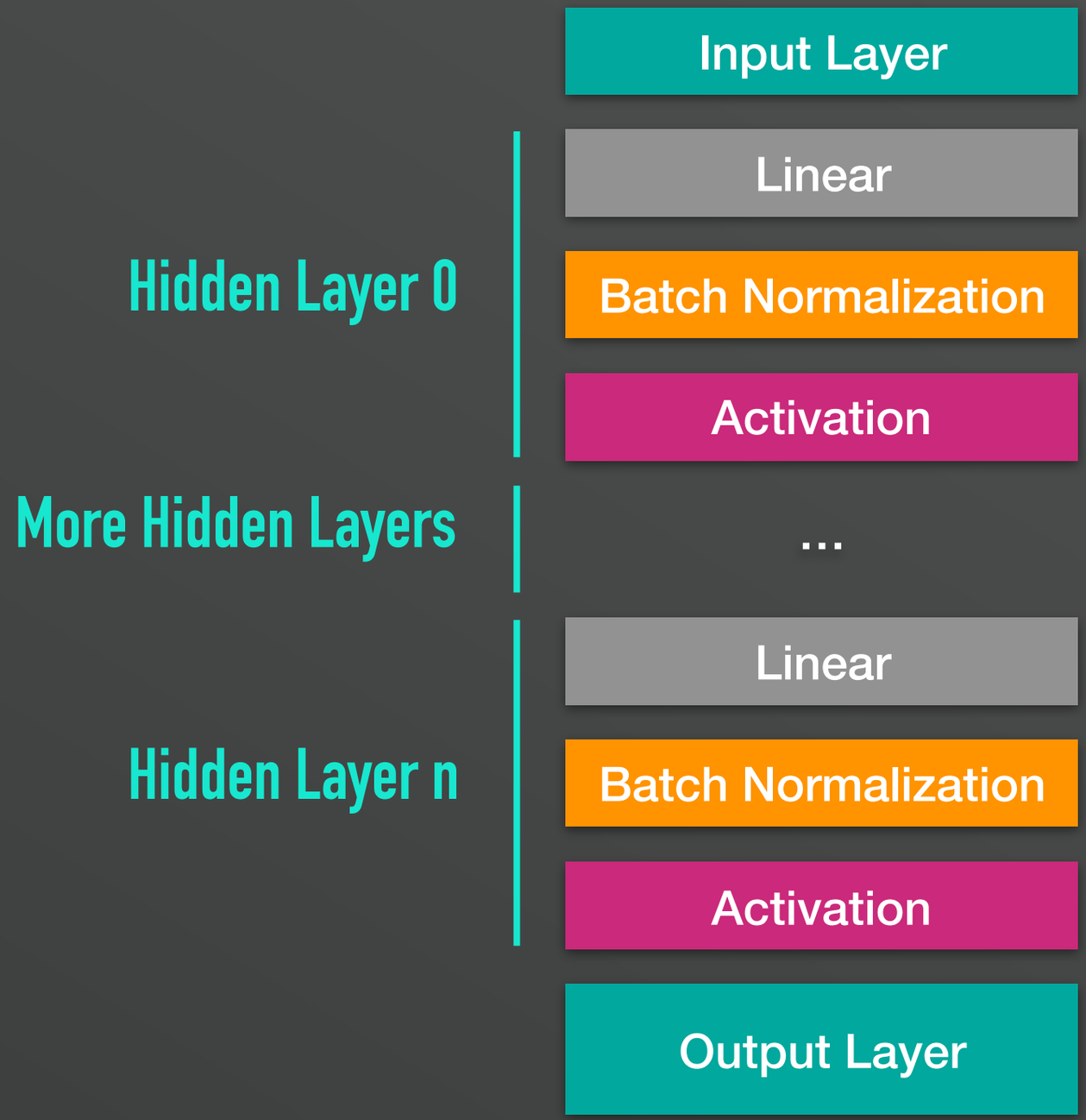
Linear

Batch Normalization

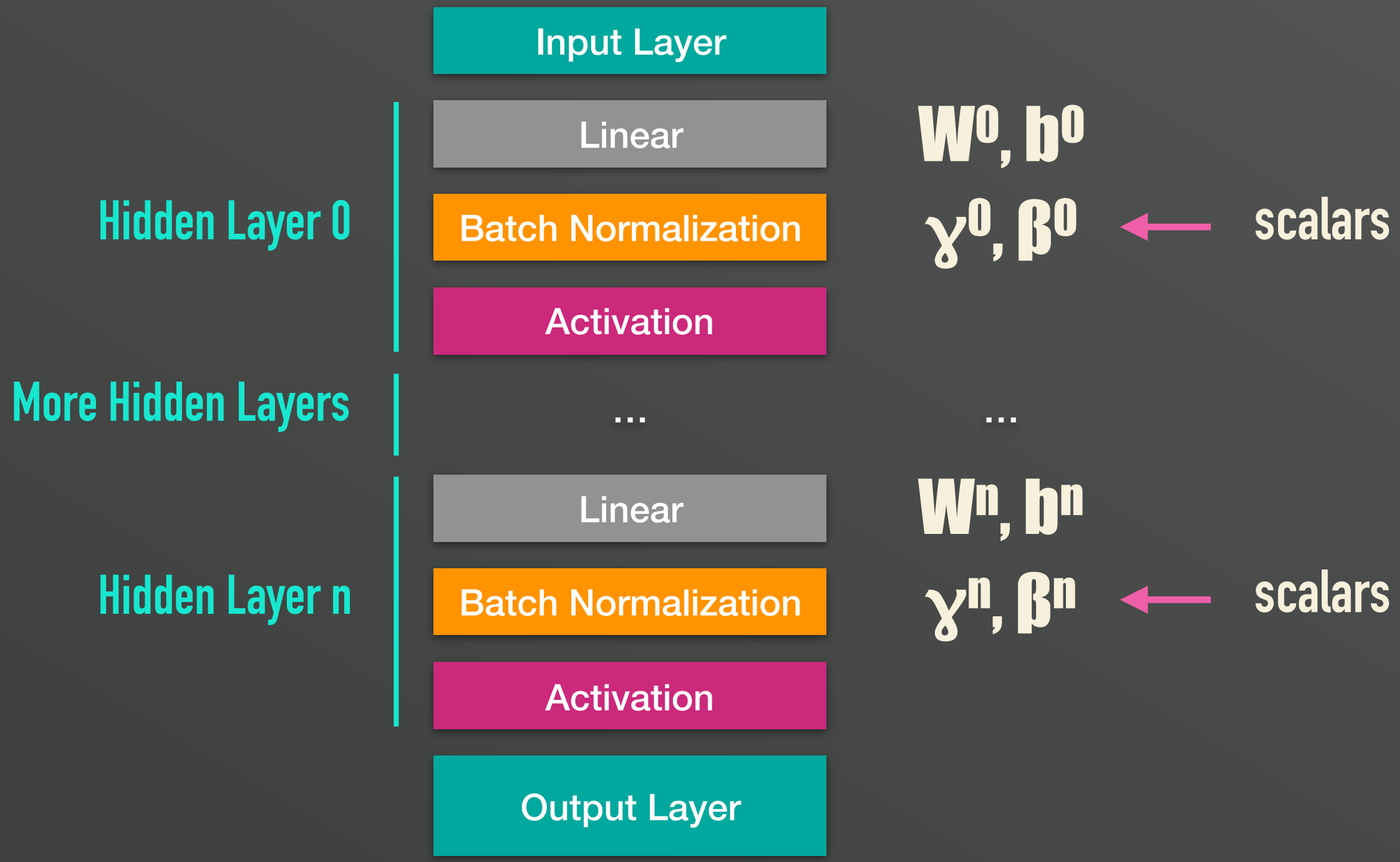
Activation

Output Layer

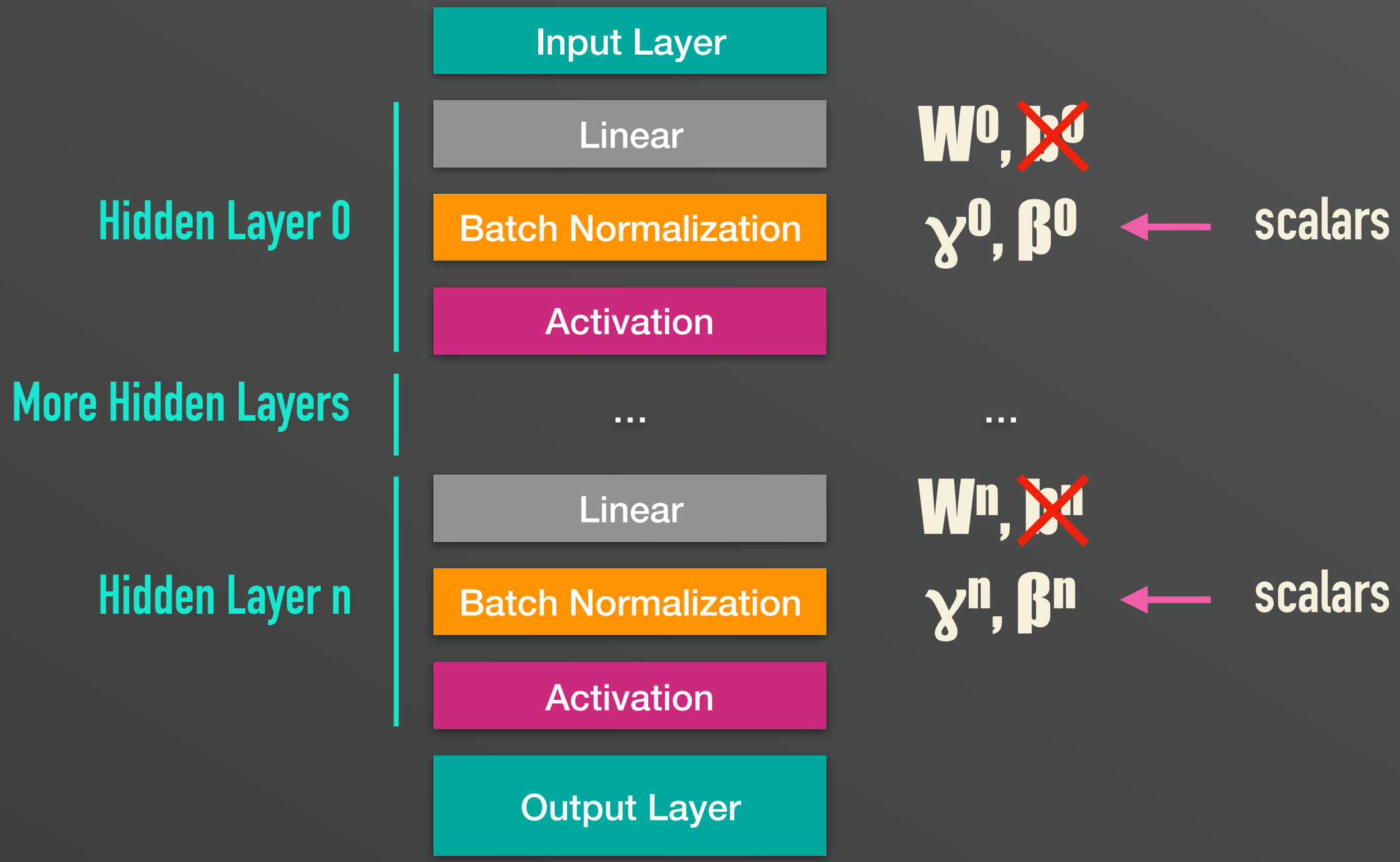
Trainable Parameters



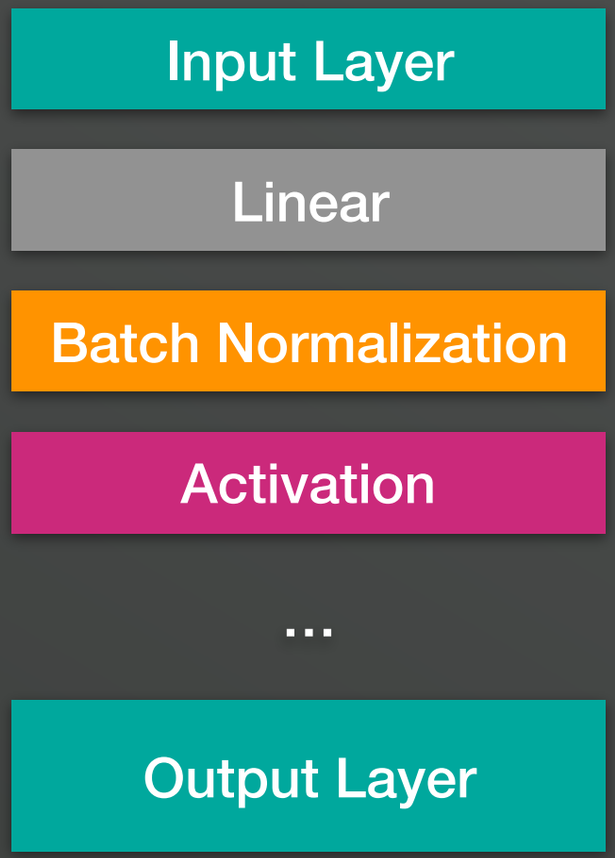
Trainable Parameters



Trainable Parameters

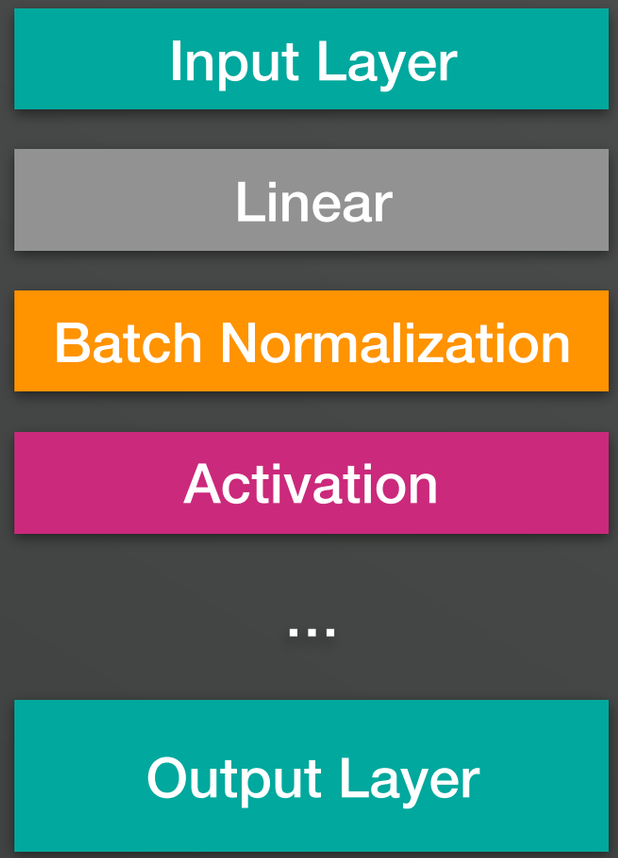


Regularization Effect



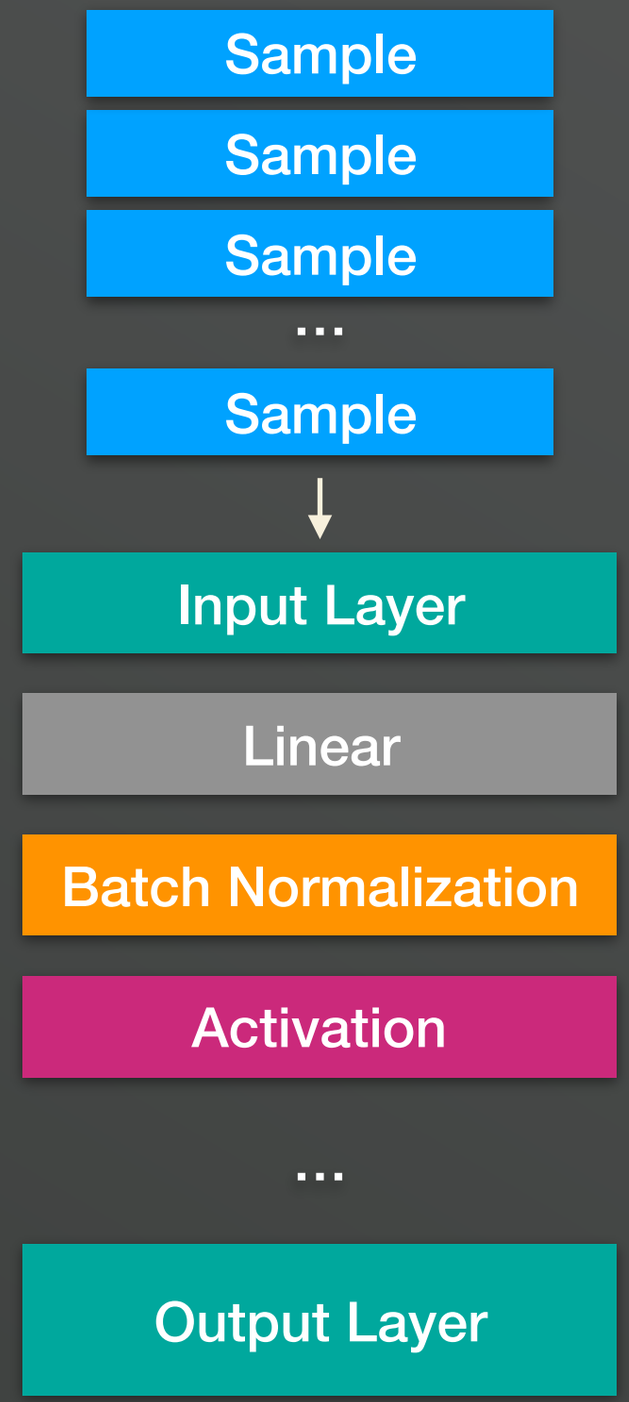
Regularization Effect

Iter 0



Regularization Effect

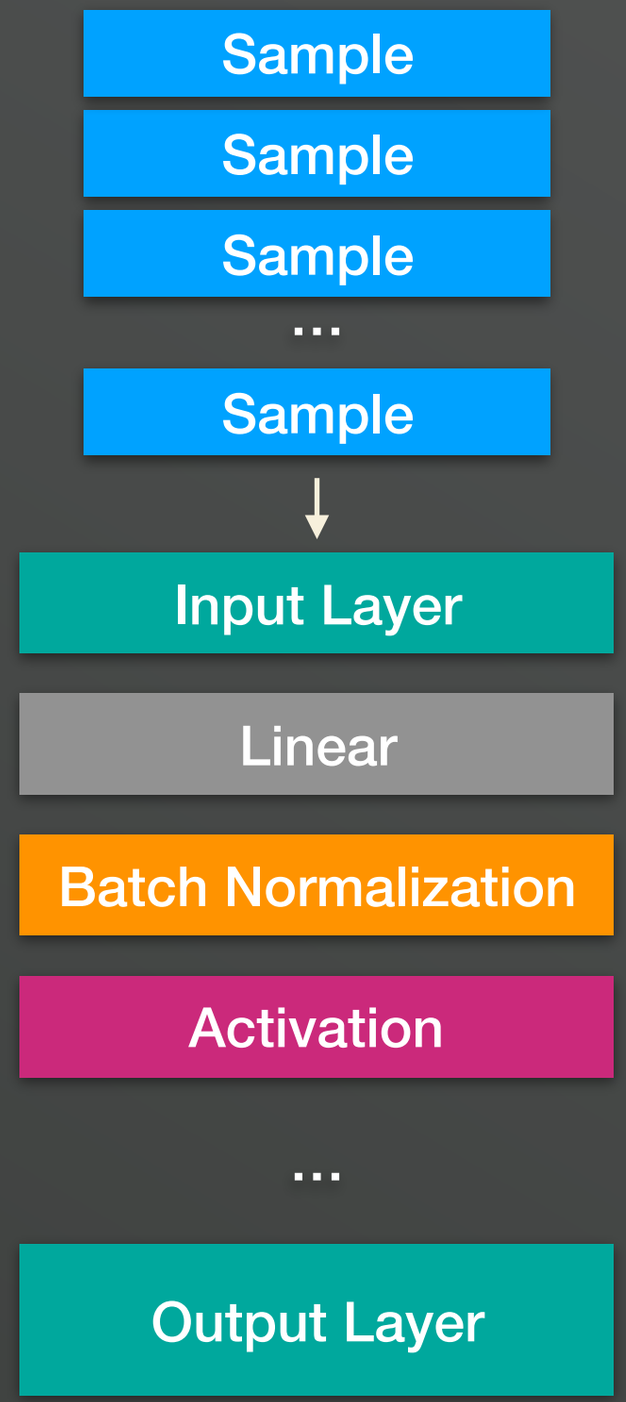
Randomly Sampled Batch



Iter 0

Regularization Effect

Randomly Sampled Batch



$Mean_0$ ← large values

$Variance_0$ ← small values

Iter 0

Regularization Effect

Randomly Sampled Batch

Sample

Sample

Sample

...

Sample

$Mean_0$ ← large values

$Variance_0$ ← small values

Iter 0

Input Layer

Linear

Batch Normalization

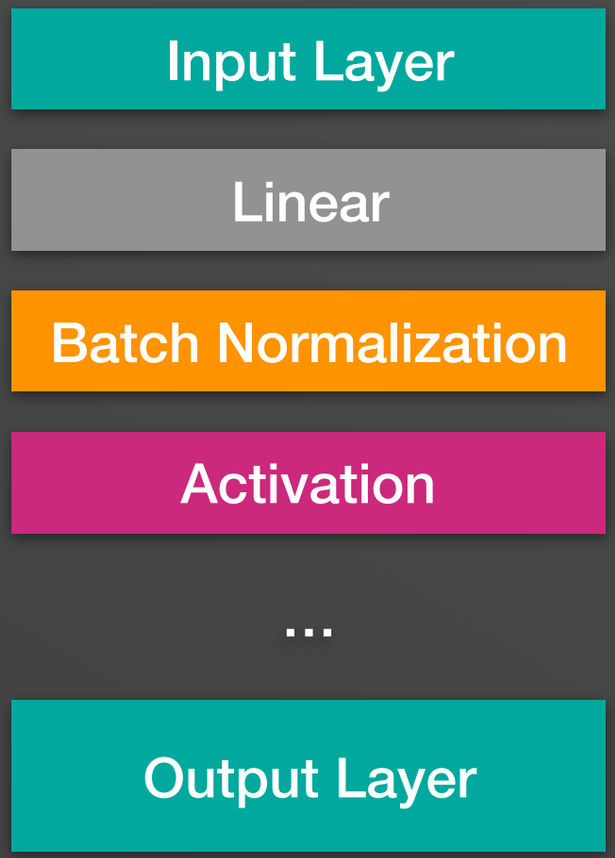
used by

Activation

...

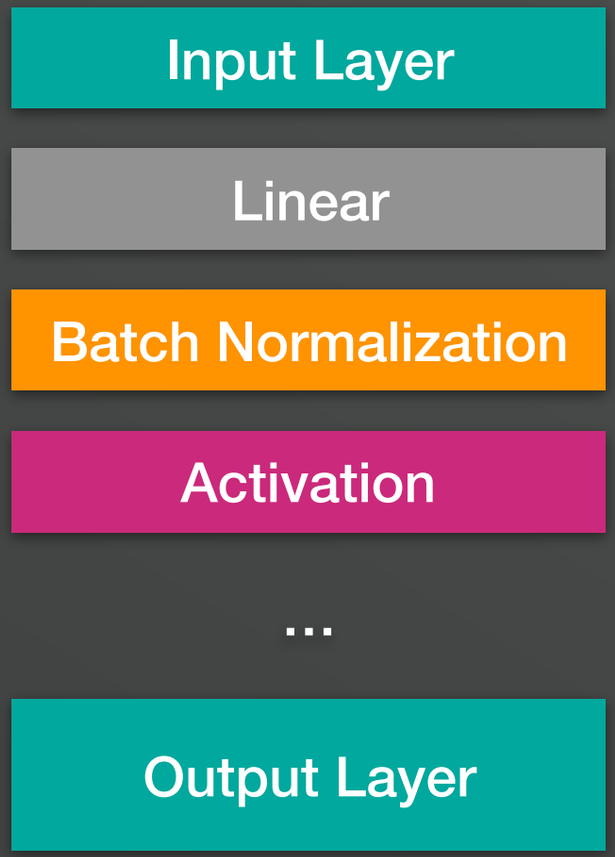
Output Layer

Regularization Effect



Regularization Effect

Iter 1



Regularization Effect

Randomly Sampled Batch

Sample

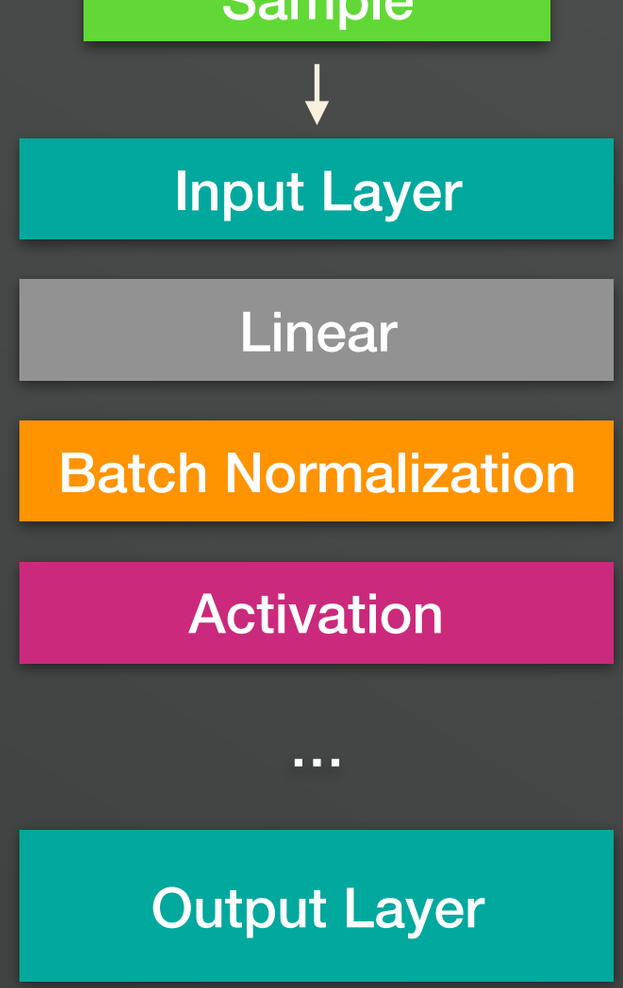
Sample

Sample

...

Sample

Iter 1



Regularization Effect

Iter 1

Randomly Sampled Batch

Sample

Sample

Sample

...

Sample

$Mean_1$ ← small values

$Variance_1$ ← large values

Input Layer

Linear

Batch Normalization

Activation

...

Output Layer

Regularization Effect

Iter 1

Randomly Sampled Batch

Sample

Sample

Sample

...

Sample

Input Layer

Linear

Batch Normalization

Activation

...

Output Layer

$Mean_1$ ← small values

$Variance_1$ ← large values

used by

Regularization Effect

Randomly Sampled Batch

Sample
Sample
Sample
...
Sample

$Mean_1$ ← small values
$Variance_1$ ← large values

Randomly Sampled Batch

Sample
Sample
Sample
...
Sample

$Mean_0$ ← large values
$Variance_0$ ← small values

Regularization Effect

Randomly Sampled Batch

Sample
Sample
Sample
...
Sample

$Mean_1$ ← small values
$Variance_1$ ← large values

Randomly Sampled Batch

Sample
Sample
Sample
...
Sample

$Mean_0$ ← large values
$Variance_0$ ← small values

During every epoch we normalize by totally different mean and variance!

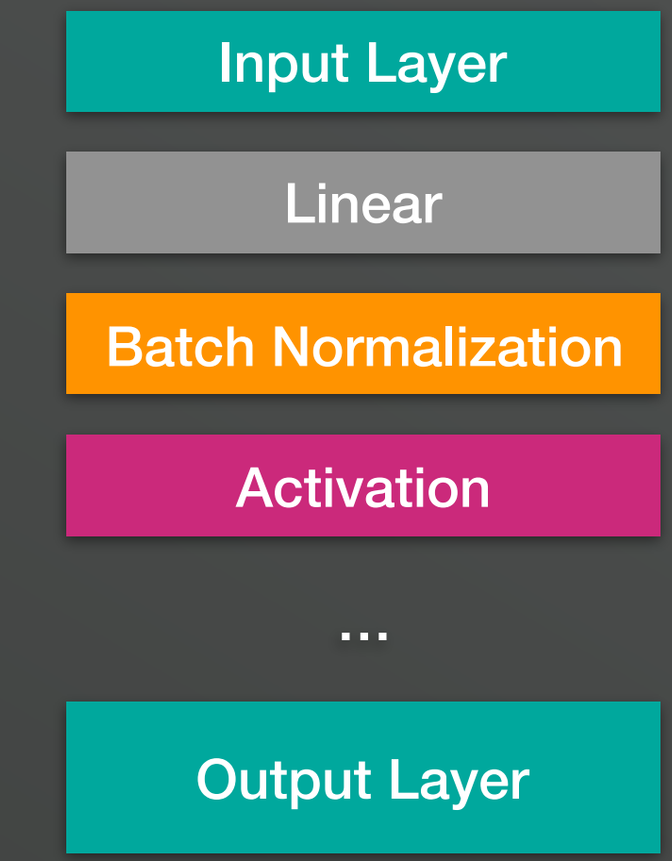
Regularization Effect

Iter 0

Whole Data

$Mean_0$

$Variance_0$



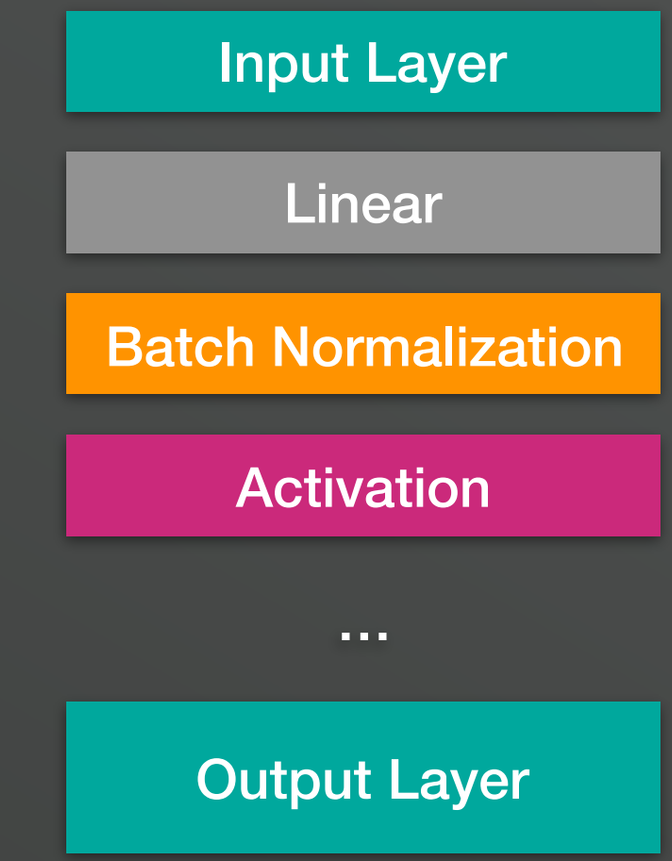
Regularization Effect

Iter 1

Whole Data

$Mean_1$

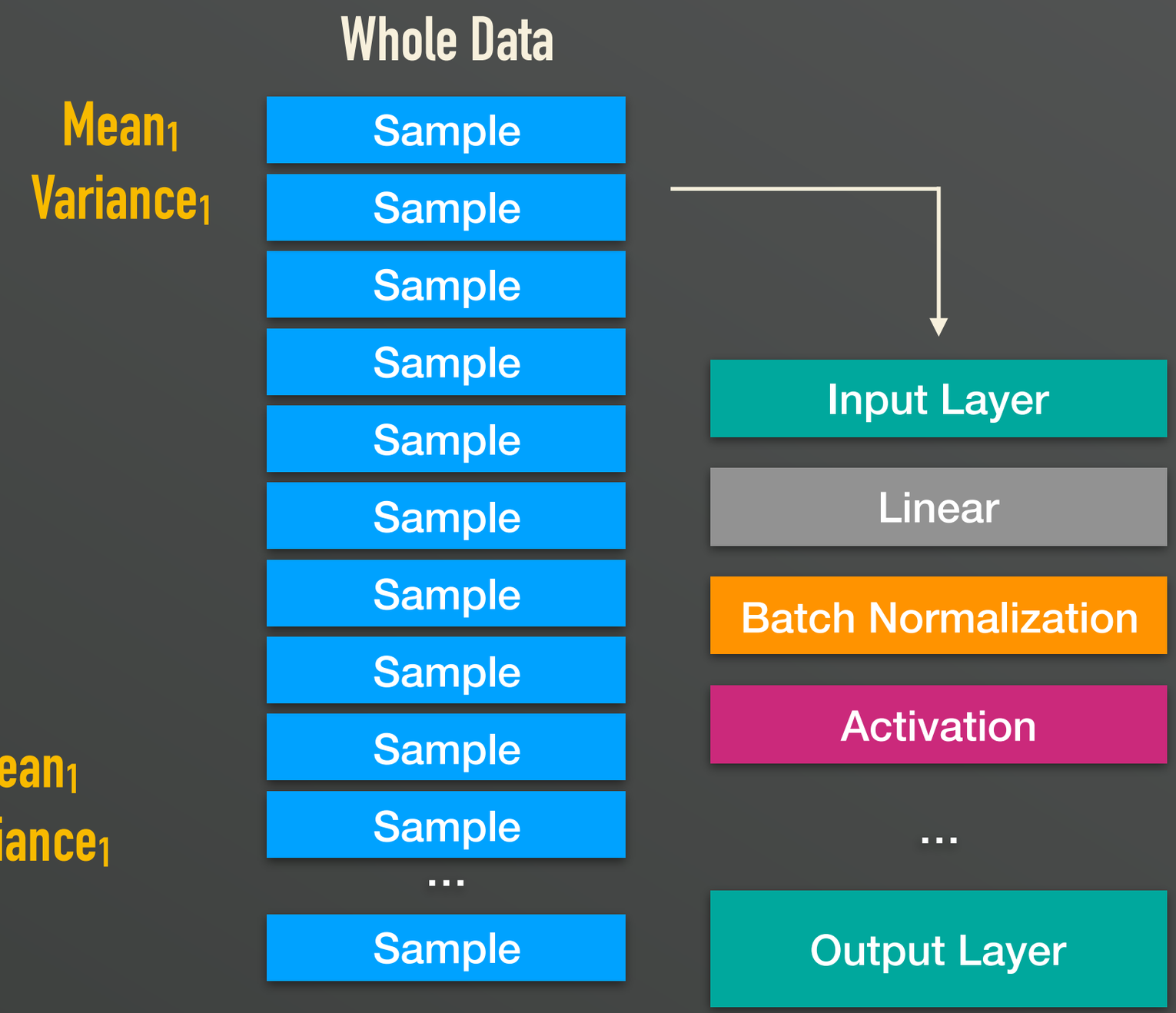
$Variance_1$



Regularization Effect

Iter 1

$Mean_0$ $Variance_0$ = $Mean_1$ $Variance_1$



Regularization Effect Summary

Regularization Effect Summary

Small batch size adds randomness to training process.

Because mean and variance vary a lot on every iteration.

Regularization Effect Summary

Small batch size adds randomness to training process.
Because mean and variance vary a lot on every iteration.

Large batch size reduces randomness in training process.
Because mean and variance used for normalization vary less.

Regularization Effect Summary

Small batch size adds randomness to training process.
Because mean and variance vary a lot on every iteration.

Large batch size reduces randomness in training process.
Because mean and variance used for normalization vary less.

Randomness causes slight regularization effect. That's why it's good to use Batch Normalization with low value of batch_size parameter.

Regularization Effect Summary

Small batch size adds randomness to training process.
Because mean and variance vary a lot on every iteration.

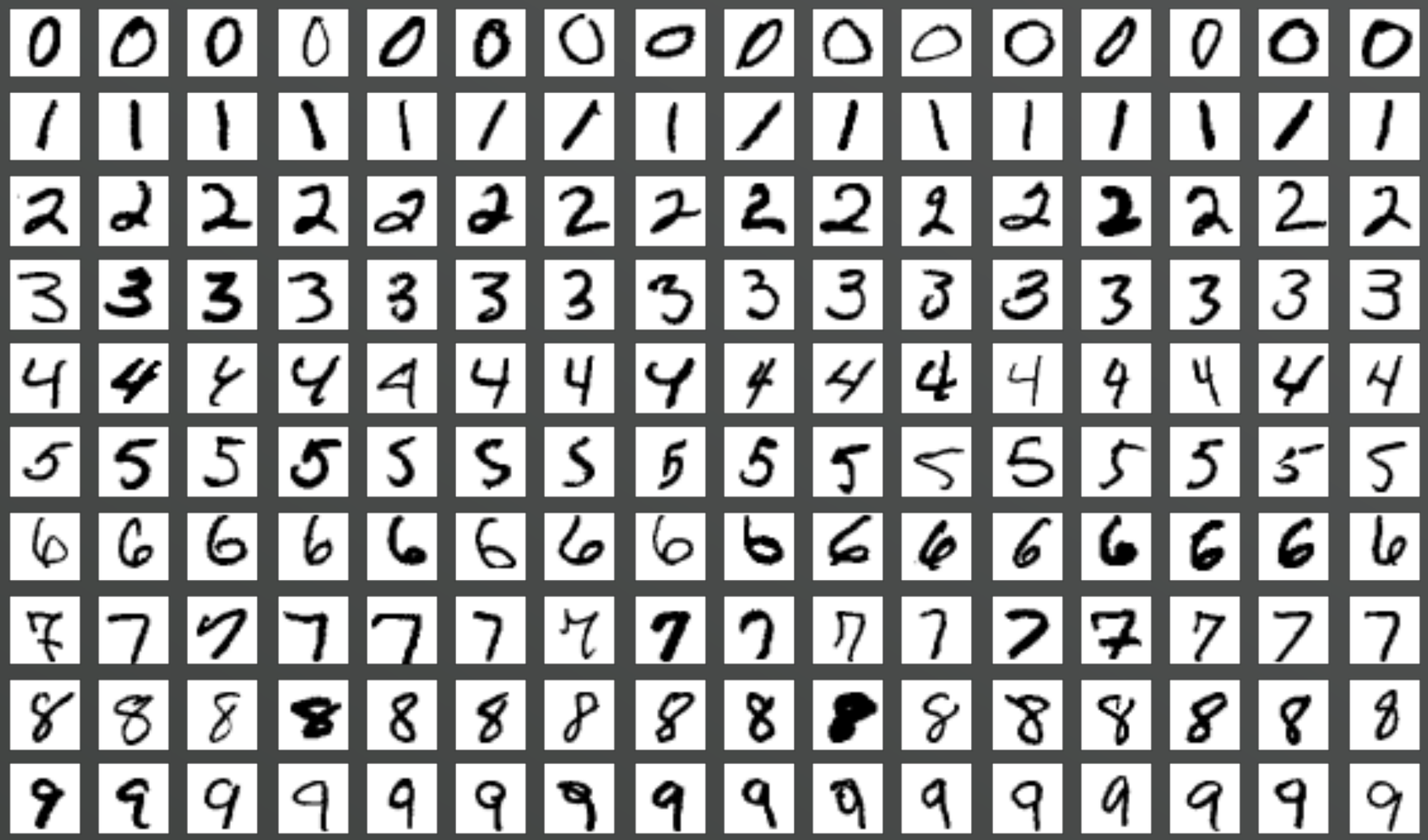
Large batch size reduces randomness in training process.
Because mean and variance used for normalization vary less.

Randomness causes slight regularization effect. That's why it's good to use Batch Normalization with low value of batch_size parameter.

It is ok to use Batch Normalization with
L2 Regularization or Dropout.

Advantagaes of using Batch Normalization


Keras



MNIST Dataset (28x28x1 Handwritten Digit Images)

60 000 Train Samples, 10 000 Test Samples

Train Set = 80% of Train Dataset = 48 000 images

Validation Set = 20% of Train Dataset = 12 000 images

Test Set = 100% of Test Dataset = 10 000 images

Loading Data

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Loading Data

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Unwrapping train images into vectors | One-Hot-Encoding labels

```
X_train_input = X_train.reshape(60000, 784)
X_train_input = X_train_input.astype('float32')

y_train_input = keras.utils.to_categorical(y_train, 10)
```

Loading Data

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Unwrapping train images into vectors | One-Hot-Encoding labels

```
X_train_input = X_train.reshape(60000, 784)
X_train_input = X_train_input.astype('float32')

y_train_input = keras.utils.to_categorical(y_train, 10)
```

Unwrapping test images into vectors | One-Hot-Encoding labels

```
X_test_input = X_test.reshape(10000, 784)
X_test_input = X_test_input.astype('float32')

y_test_input = keras.utils.to_categorical(y_test, 10)
```

Loading Data

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Unwrapping train images into vectors | One-Hot-Encoding labels

```
X_train_input = X_train.reshape(60000, 784)
X_train_input = X_train_input.astype('float32')

y_train_input = keras.utils.to_categorical(y_train, 10)
```

Unwrapping test images into vectors | One-Hot-Encoding labels

```
X_test_input = X_test.reshape(10000, 784)
X_test_input = X_test_input.astype('float32')

y_test_input = keras.utils.to_categorical(y_test, 10)
```

Normalize data

```
X_train_input /= 255
X_test_input /= 255
```

Splitting data into Train/Val sets

```
splt_ratio = 0.8
splt_i = int(X_train_input.shape[0] * splt_ratio)

X_train_cpy = X_train_input.copy()
y_train_cpy = y_train_input.copy()

X_train_input, y_train_input = X_train_cpy[:splt_i], y_train_cpy[:splt_i]
X_val_input, y_val_input = X_train_cpy[splt_i:], y_train_cpy[splt_i:]
```

Input Layer (768)

ReLU (256)

ReLU (128)

ReLU (64)

Output Layer
Softmax (10)

```
# Creating model
model_without_bn = Sequential()

# Adjusting model structure
model_without_bn.add(Dense(256, activation="relu", input_shape=(784,)))
model_without_bn.add(Dense(128, activation="relu"))
model_without_bn.add(Dense(64, activation="relu"))
model_without_bn.add(Dense(10, activation="softmax"))
```

Input Layer (768)

Linear (256) (no bias)

Batch Normalization

ReLU

Linear (256) (no bias)

Batch Normalization

ReLU

Linear (256) (no bias)

Batch Normalization

ReLU

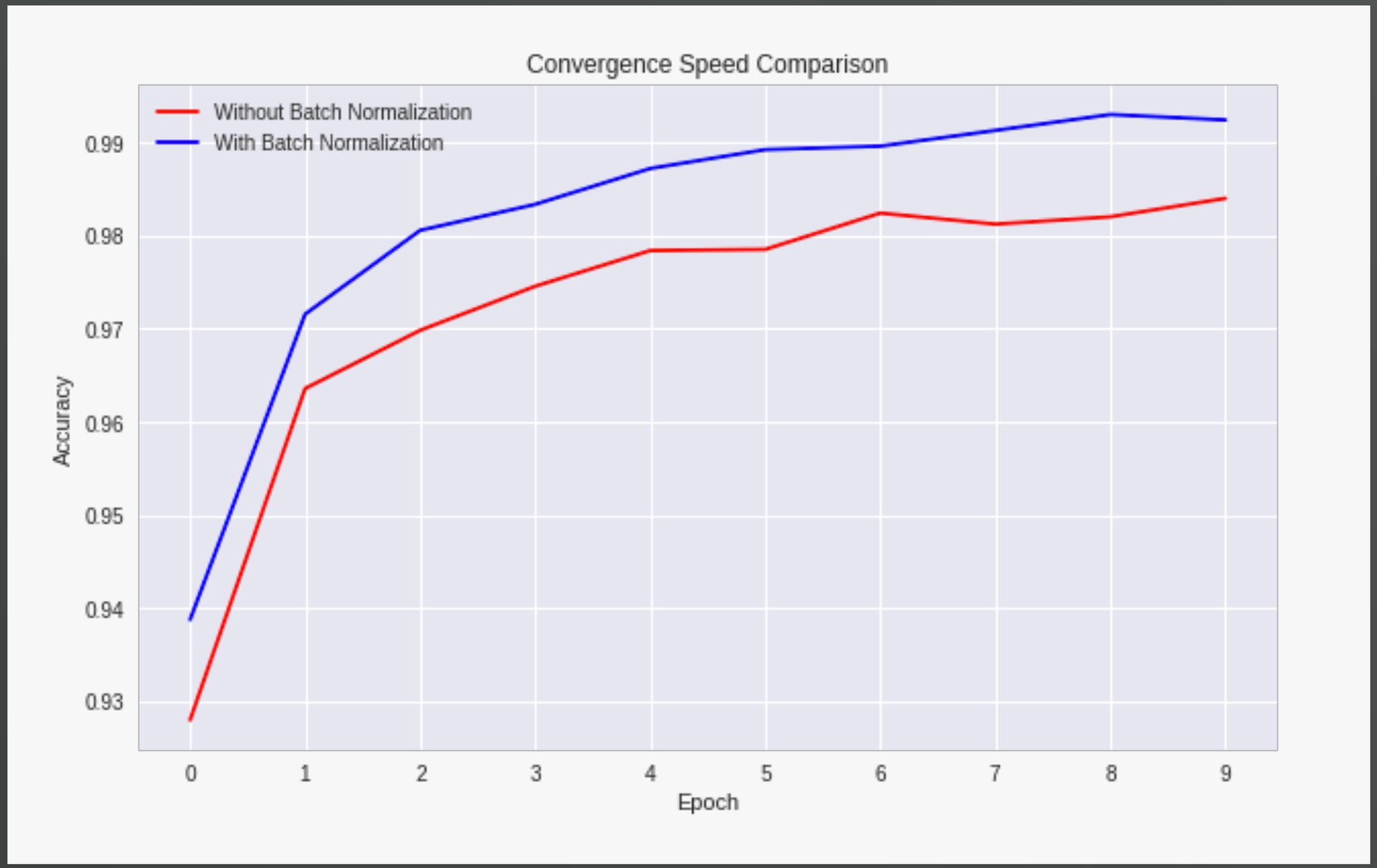
Output Layer
Softmax (10)

```
# Creating model
model_with_bn = Sequential()

# Adjusting model structure
model_with_bn.add(Dense(256, use_bias=False, input_shape=(784,)))
model_with_bn.add(BatchNormalization())
model_with_bn.add(Activation("relu"))

model_with_bn.add(Dense(128, use_bias=False))
model_with_bn.add(BatchNormalization())
model_with_bn.add(Activation("relu"))

model_with_bn.add(Dense(64, use_bias=False))
model_with_bn.add(BatchNormalization())
model_with_bn.add(Activation("relu"))

model_with_bn.add(Dense(10, activation="softmax"))
```

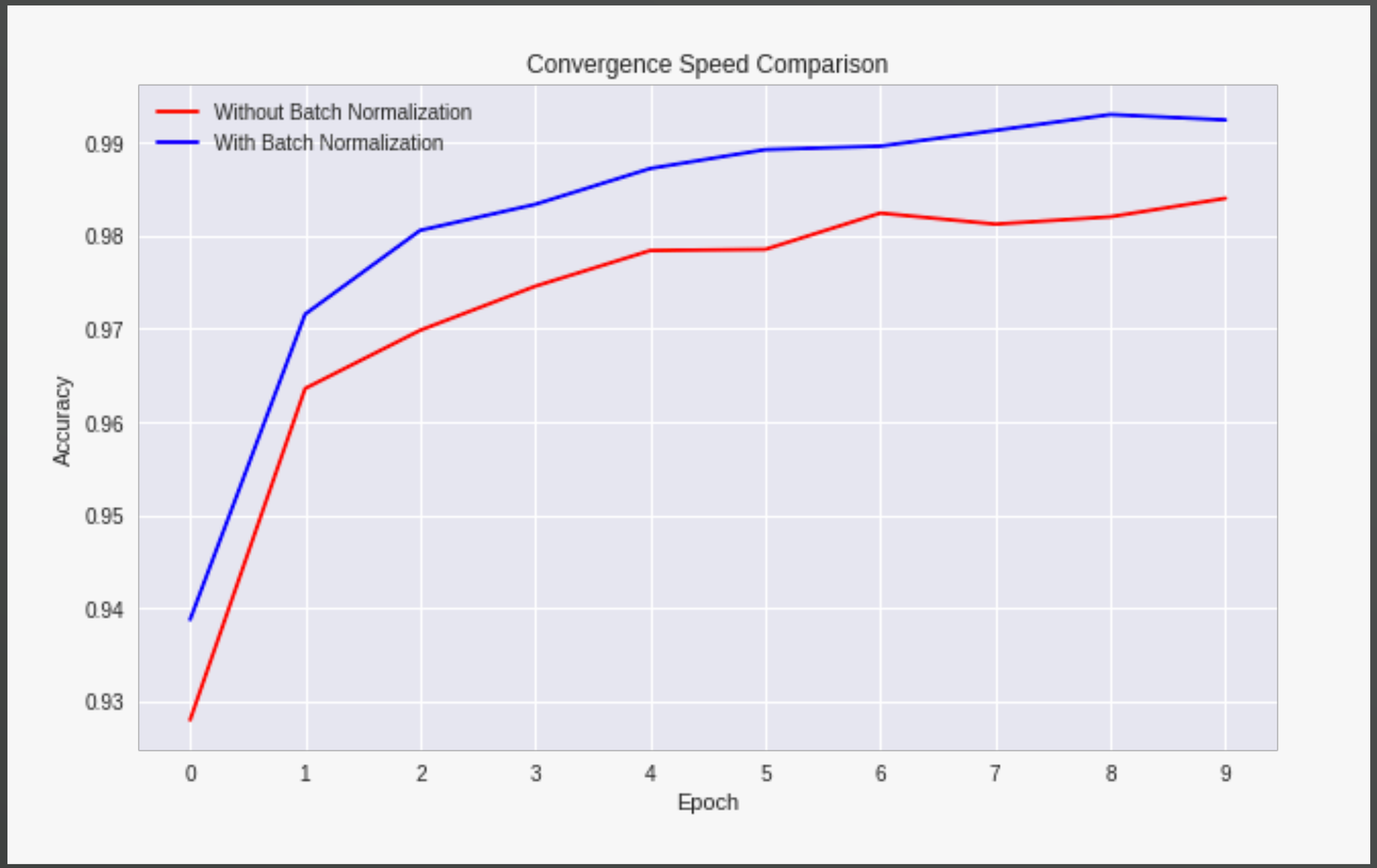
Comparison 1

```
compare_models(X_train_input, y_train_input, X_val_input, y_val_input,
               X_test_input, y_test_input, layers=[256, 128, 64],
               learning_rate=0.01, activation="relu", batch_size=128,
               epochs=10, weight_init=glorot_uniform(seed=3939),
               experiment_title="Convergence Speed Comparison")
```

Comparison 1



Comparison 1



Without batch norm test-acc: 0.9657

With batch norm test-acc: 0.9739

Comparison 2

```
compare_models(X_train_input, y_train_input, X_val_input, y_val_input,
               X_test_input, y_test_input, layers=[256, 128, 64],
               learning_rate=1, activation="relu", batch_size=1024,
               epochs=10, weight_init=glorot_uniform(seed=3939),
               experiment_title="Big Learning Rate")
```

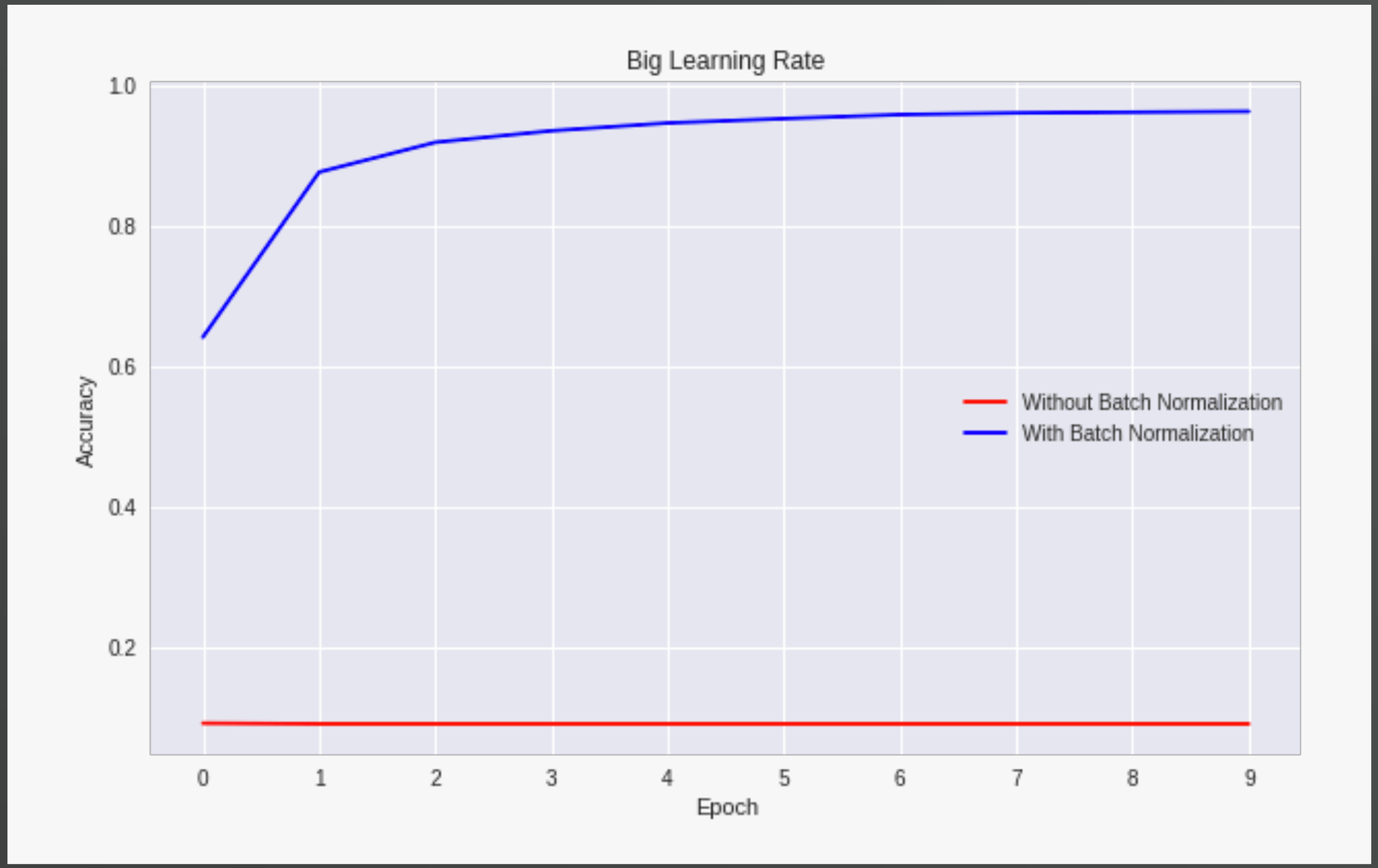
Comparison 2

```
compare_models(X_train_input, y_train_input, X_val_input, y_val_input,
               X_test_input, y_test_input, layers=[256, 128, 64],
learning_rate=1, activation="relu", batch_size=1024,
epochs=10, weight_init=glorot_uniform(seed=3939),
experiment_title="Big Learning Rate")
```

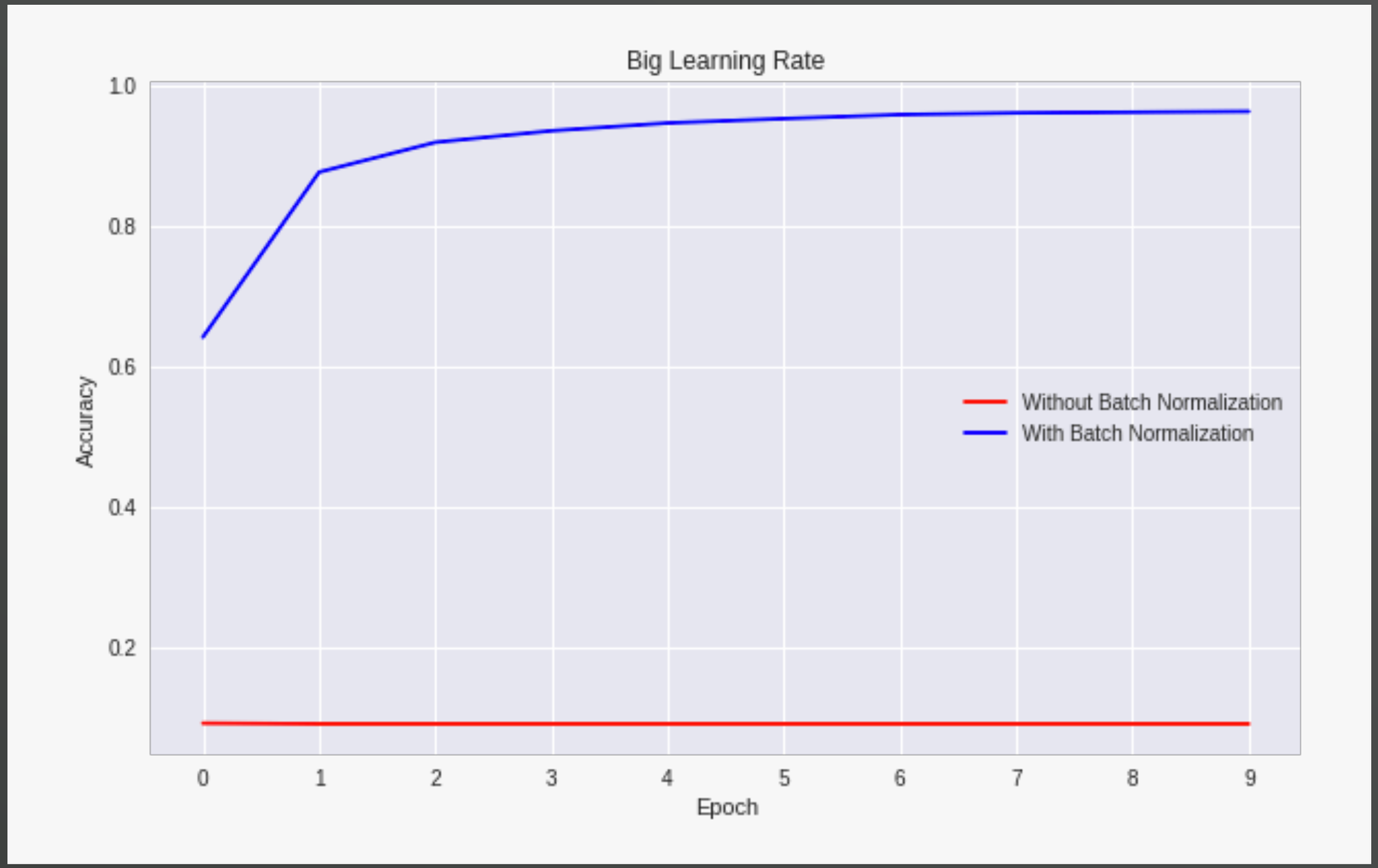
learning_rate: 0.01 → 1

batch_size: 128 → 1024

Comparison 2



Comparison 2



Without batch norm test-acc: 0.0892

With batch norm test-acc: 0.9505

Comparison 3

```
compare_models(X_train_input, y_train_input, X_val_input, y_val_input,
               X_test_input, y_test_input, layers=[256, 128, 64],
               learning_rate=0.01, activation="relu", batch_size=128,
               weight_init=RandomUniform(minval=-5, maxval=5, seed=3939),
               epochs=10,experiment_title="Too Big Weights")
```

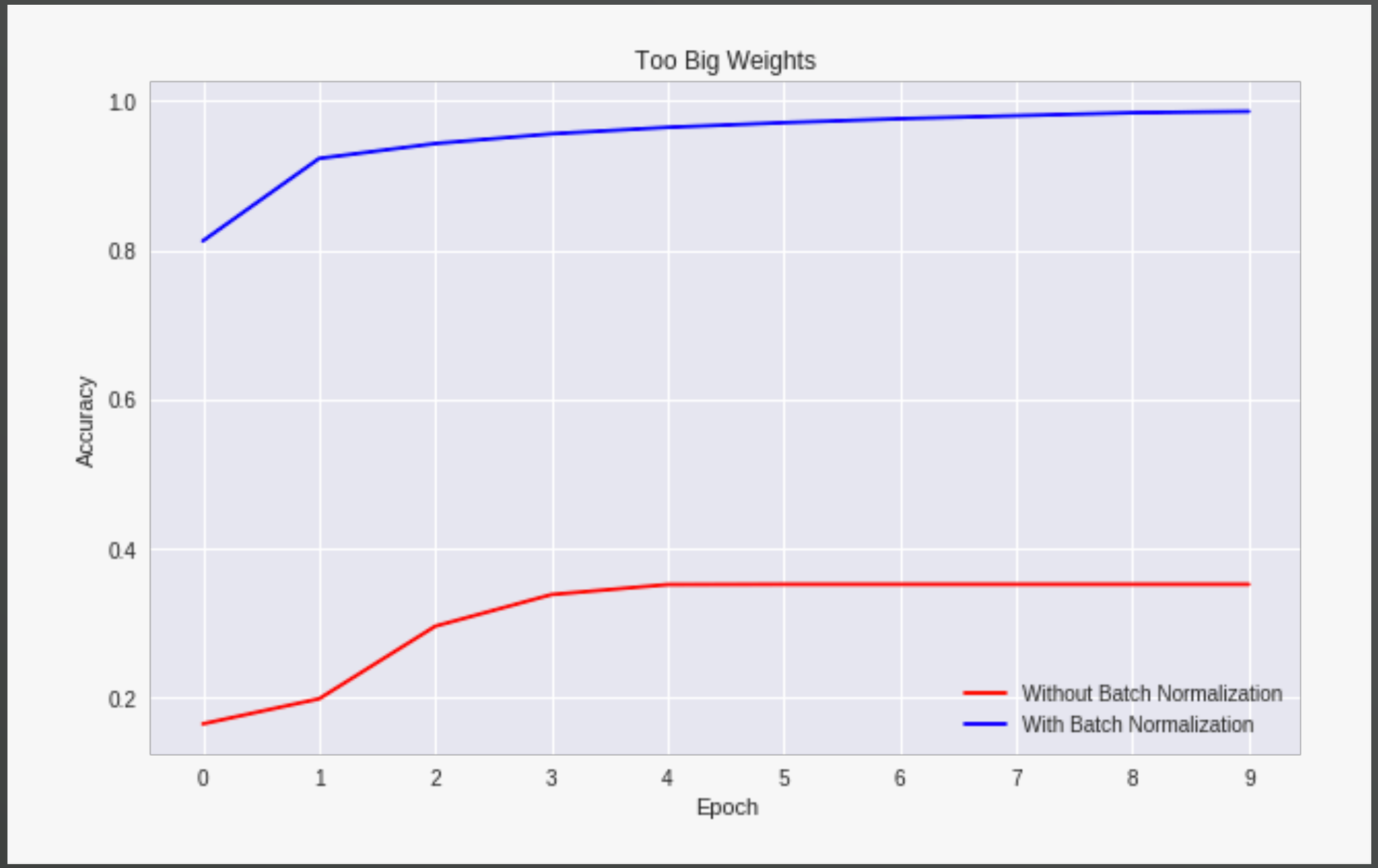
Comparison 3

```
compare_models(X_train_input, y_train_input, X_val_input, y_val_input,
               X_test_input, y_test_input, layers=[256, 128, 64],
               learning_rate=0.01, activation="relu", batch_size=128,
               weight_init=RandomUniform(minval=-5, maxval=5, seed=3939),
               epochs=10,experiment_title="Too Big Weights")
```

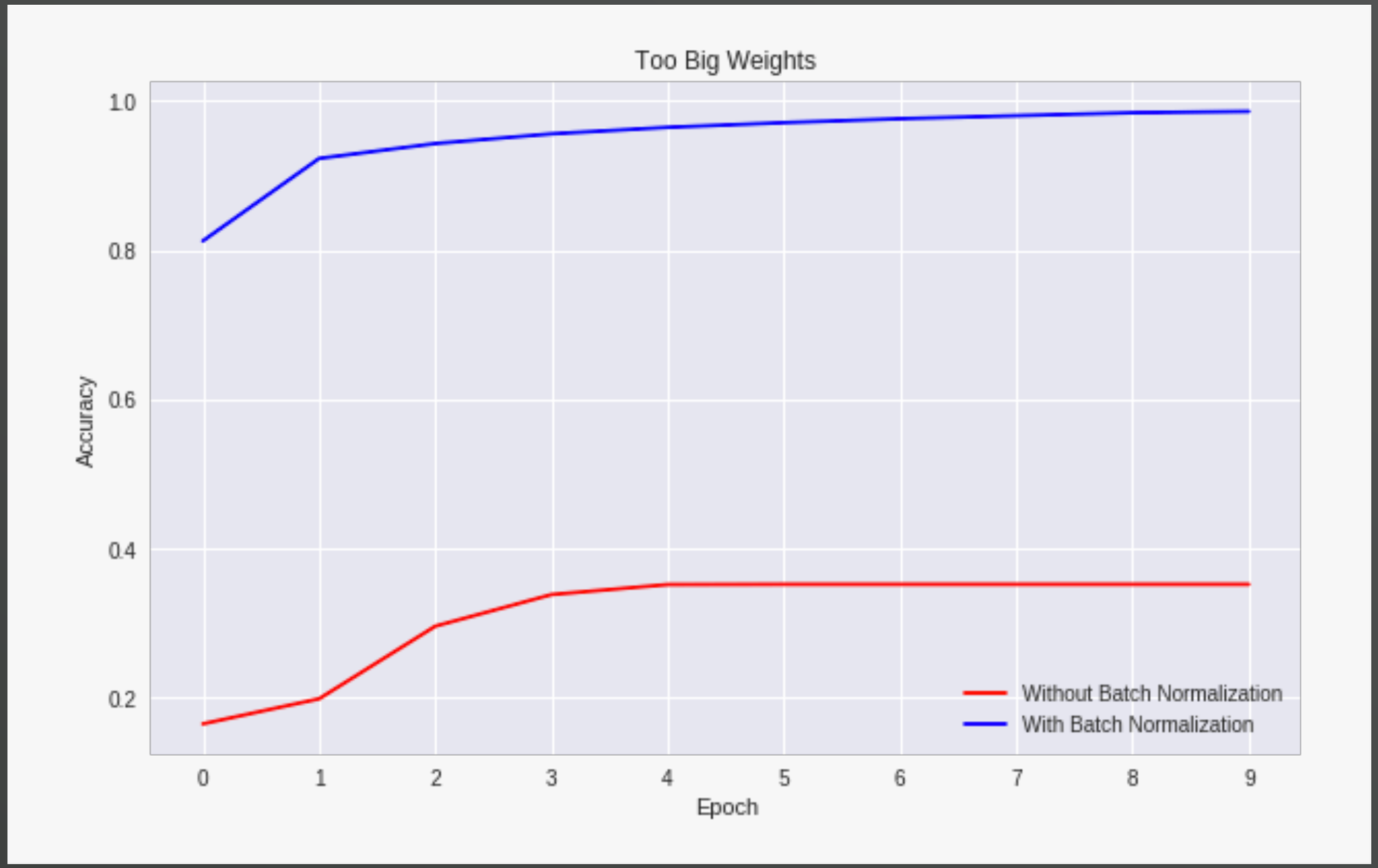
weight_init:

glorot_uniform → RandomUniform(-5, 5)

Comparison 3



Comparison 3



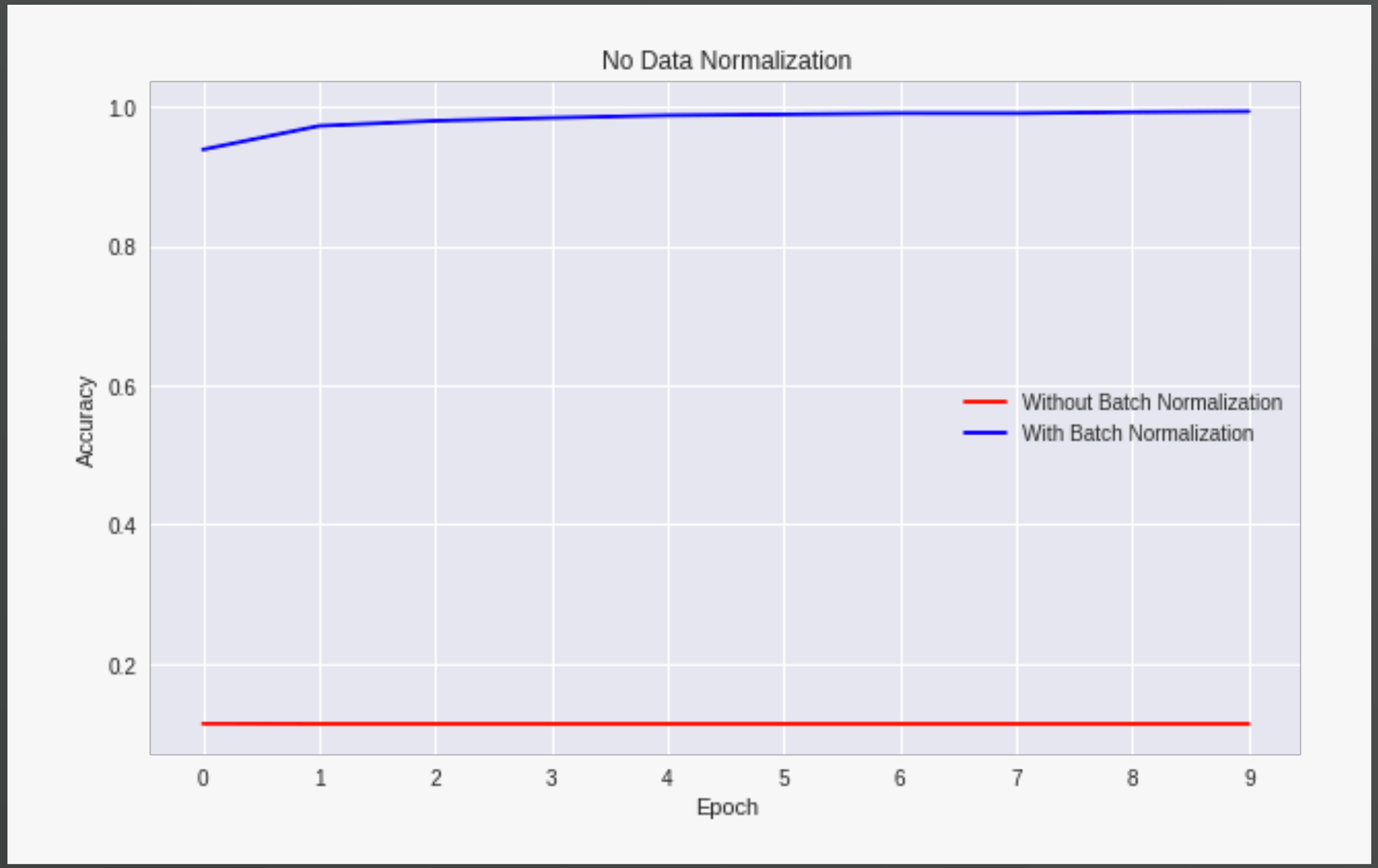
Without batch norm test-acc: 0.3525

With batch norm test-acc: 0.9660

Comparison 4

```
# Undoing normalization
X_train_input *= 255
X_val_input *= 255
X_test_input *= 255

compare_models(X_train_input, y_train_input, X_val_input, y_val_input,
               X_test_input, y_test_input, layers=[256, 128, 64],
               learning_rate=0.01, activation="relu", batch_size=128,
               epochs=10, weight_init=glorot_uniform(seed=3939),
               experiment_title="No Data Normalization")
```

Comparison 4



Comparison 4

Without batch norm test-acc: 0.1135

With batch norm test-acc: 0.9766

Summary

Advantages of using Batch Normalization:

- allows larger learning rates
- model converges in less number of epochs
- allows us to be more careless about weight initialization
- no need for data normalization
- allows to use more activation functions freely
- adds slight regularization effect if used with small batch

Thank you for your attention!