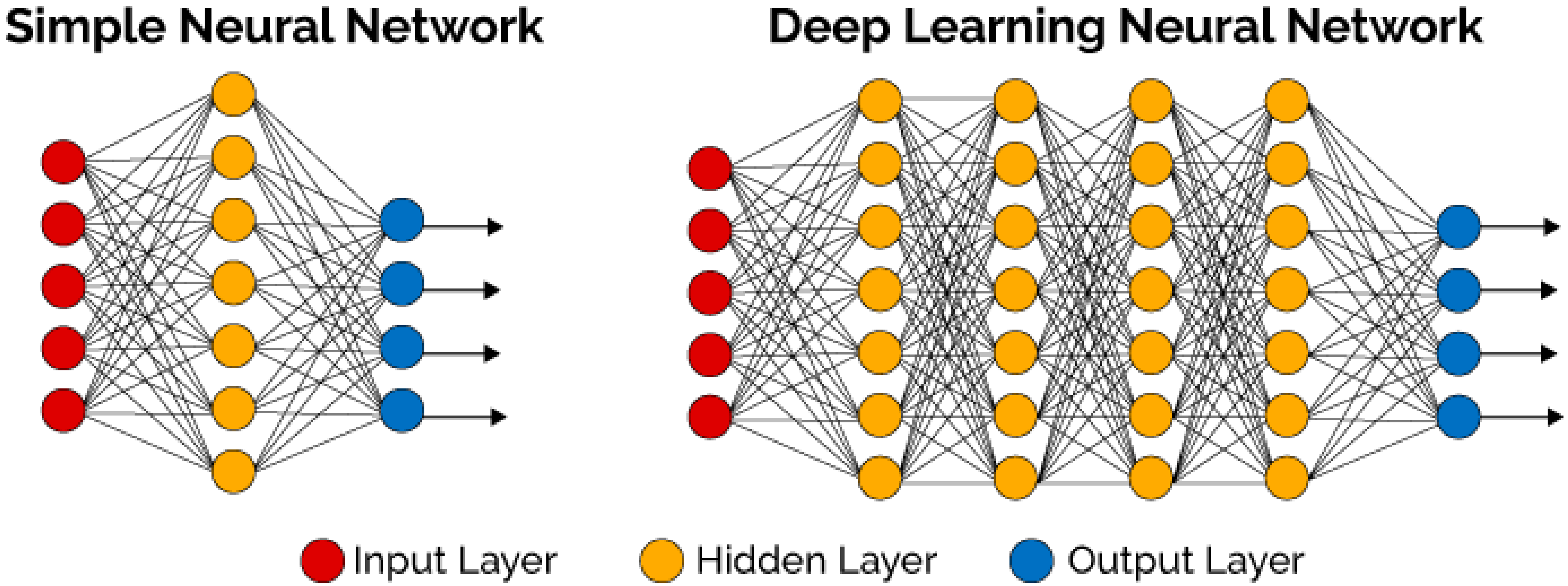
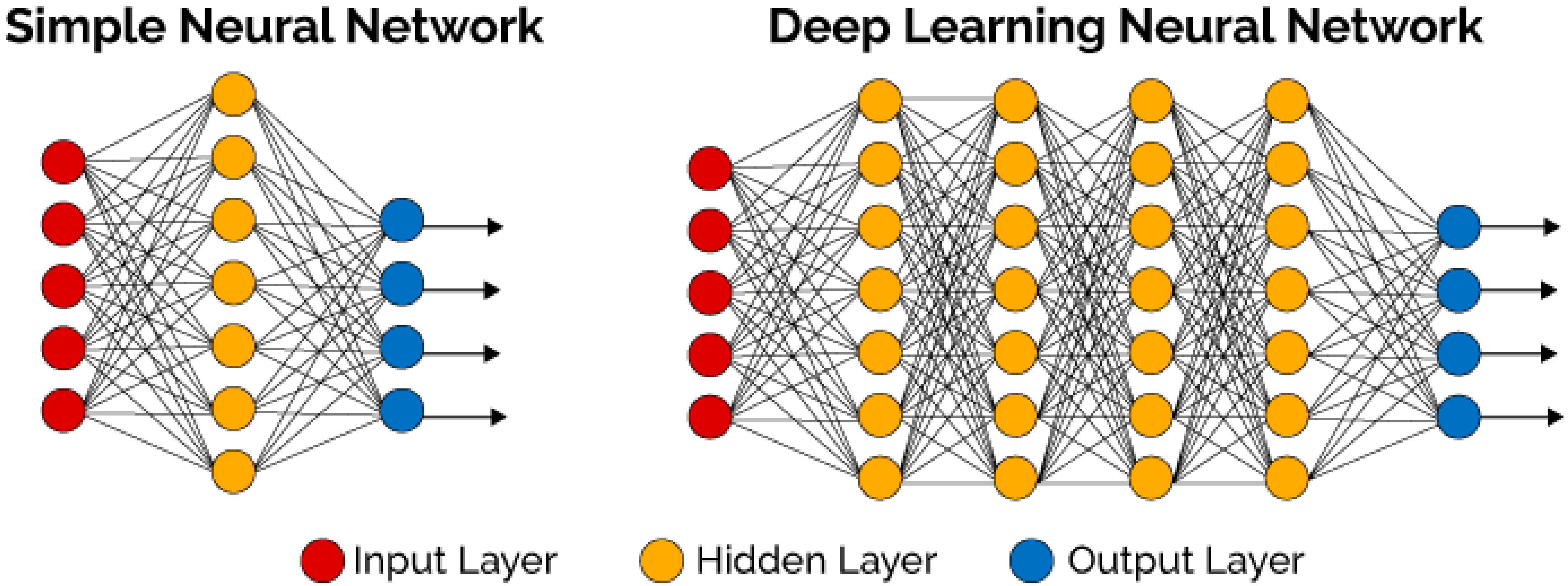
Introduction to KERAS Framework

Why we need a framework?



Forward Computing

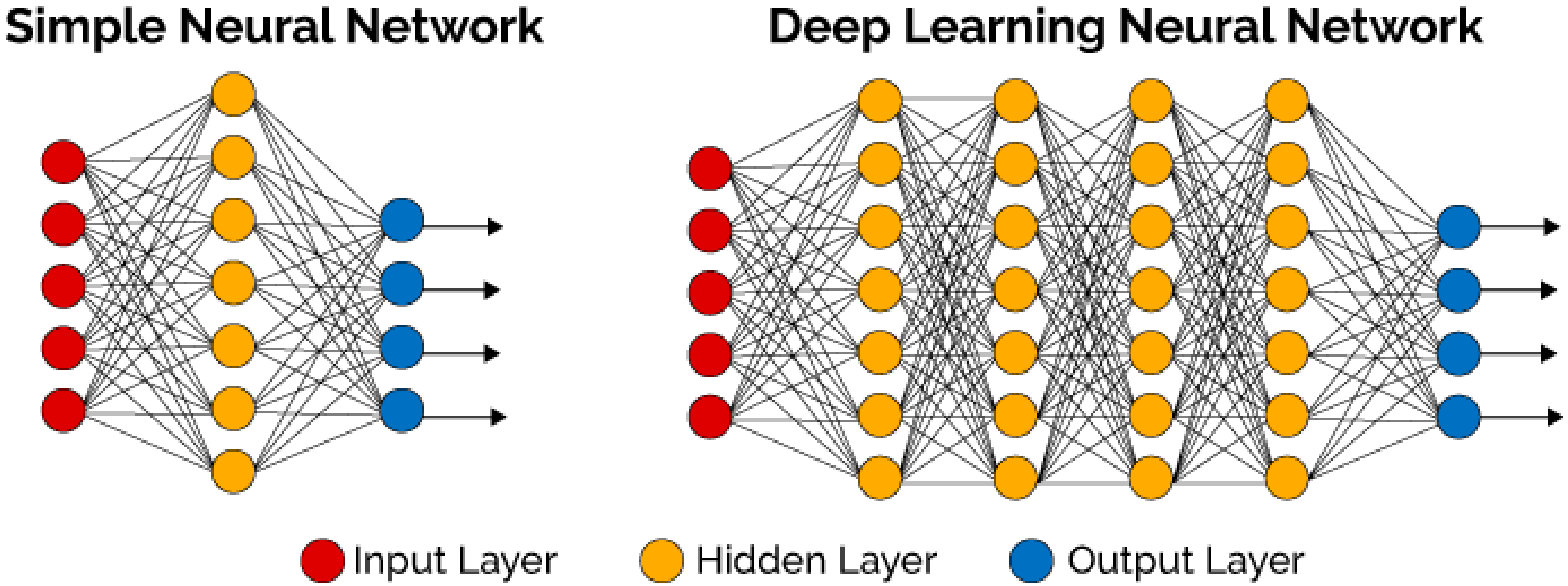
Why we need a framework?



Backward updating

Why We Need a Framework?

It involve a lot of Computation

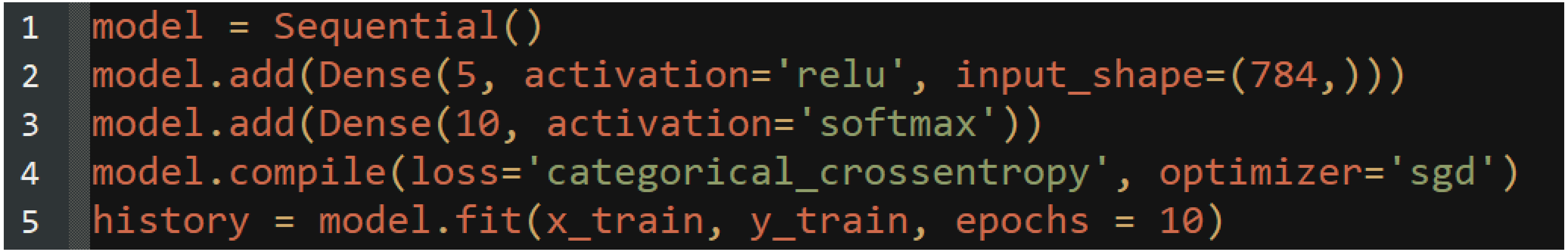


Backward updating

Why We Need a Framework?

Even for a simple neural network:

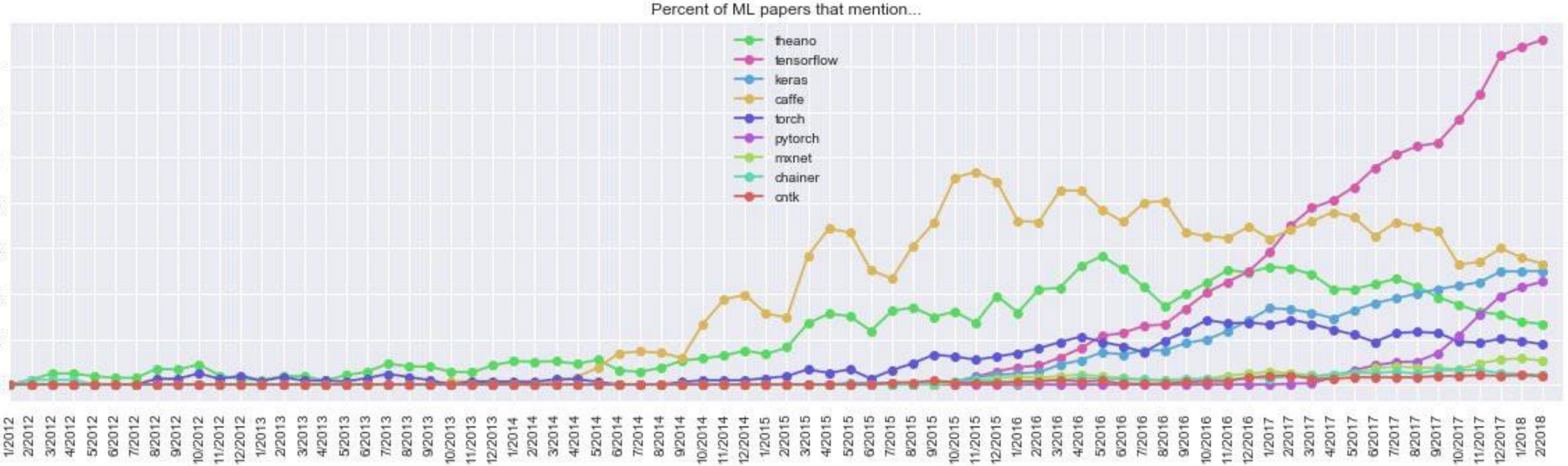
- hundreds lines of code is needed, in python.
- In a DL framework (TensorFlow, PyTorch, etc) you may write it in 20-50 lines.
- In Keras, it can be done in 5 lines of code!



```
model = Sequential()
model.add(Dense(5, activation='relu', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd')
history = model.fit(x_train, y_train, epochs = 10)
```

What a DL framework usually do for us?

- Computes Automatic Backpropagation.
- Contains deep learning networks.
- makes computing on GPU easier.



Why we have chosen Keras?

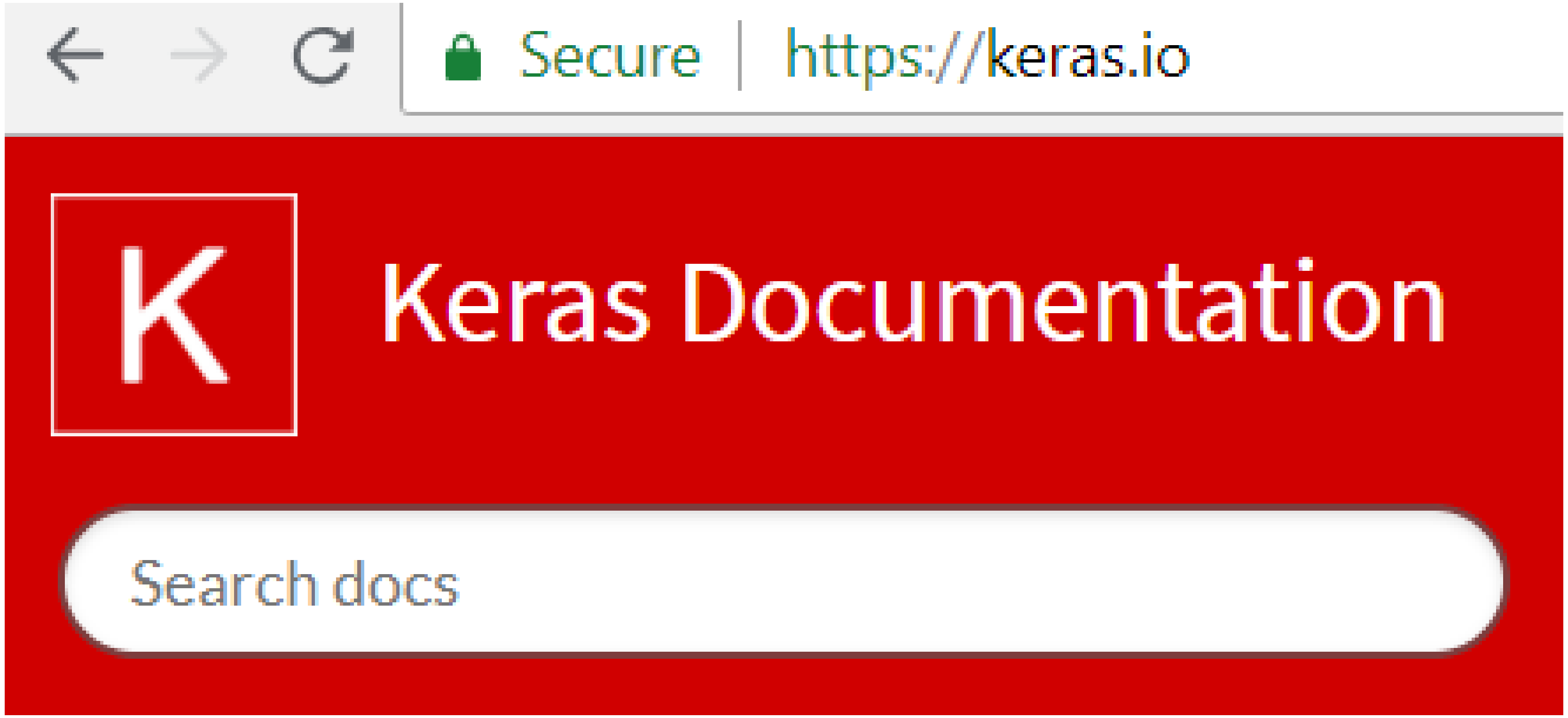
Keras: Deep Learning for humans

You have just found Keras.

- Allows for easy and fast prototyping
- Supports both convolutional and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

What more about Keras?

- Good Documentation at: Keras.io



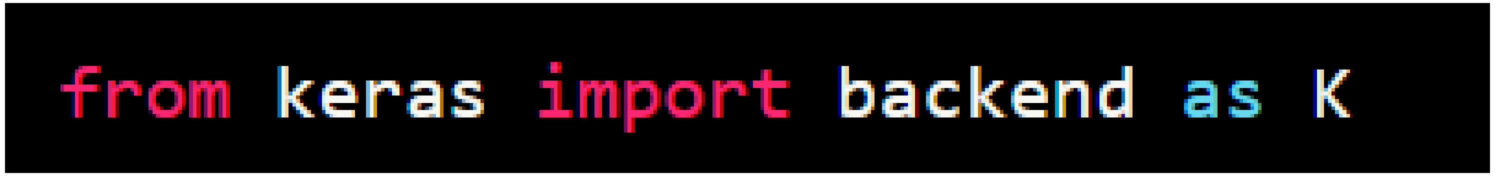
Secure | https://keras.io

Keras Documentation

Search docs

What more about Keras?

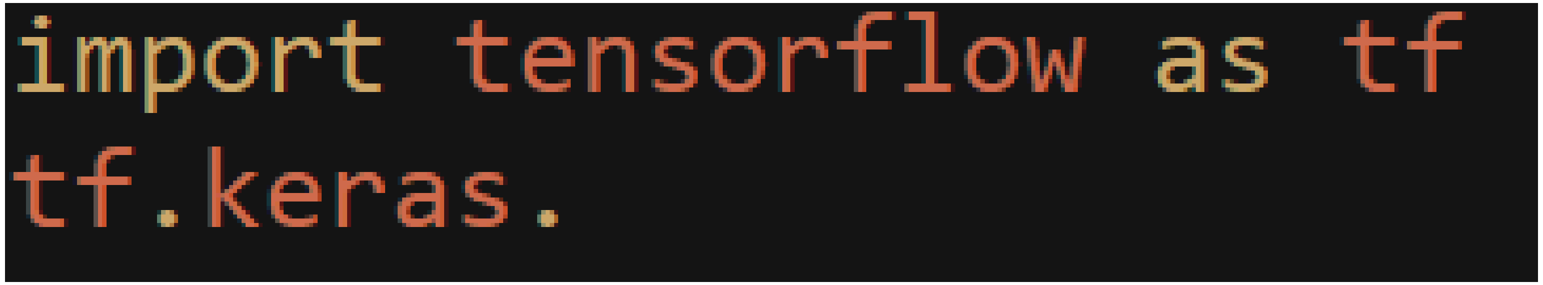
- You can use TensorFlow (generally any backend) code in Keras:



```
from keras import backend as K
```

What more about Keras?

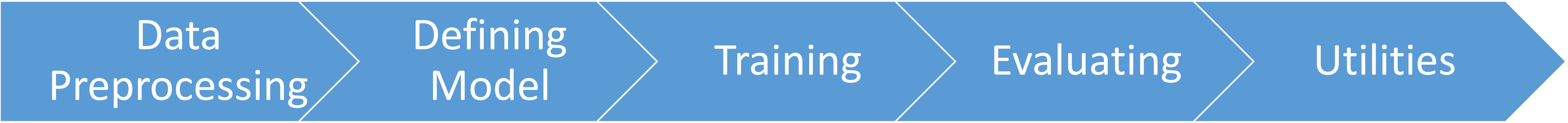
- You can use Keras in TensorFlow



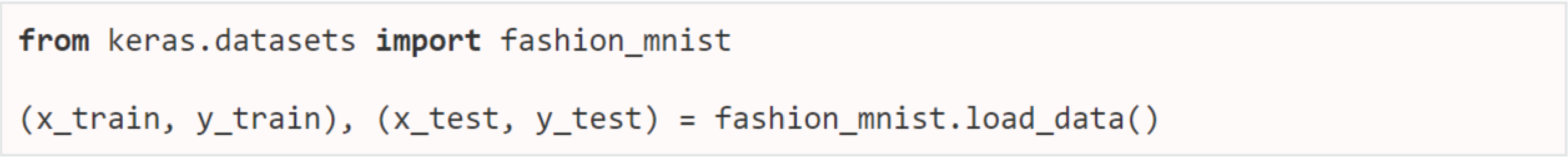
```
import tensorflow as tf
tf.keras.
```

A deep learning problem in Keras?

- You can use Keras in TensorFlow



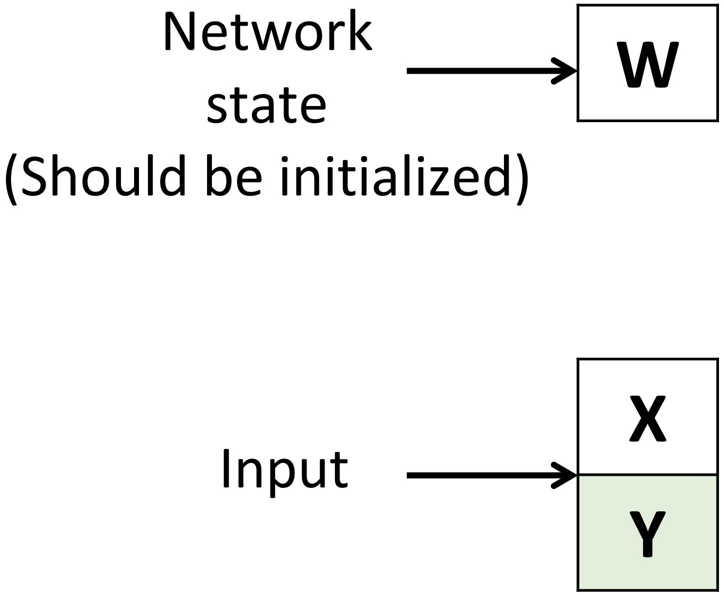
- keras.datasets
  - MNIST database of handwritten digits
  - CIFAR-10 small image classification
  - CIFAR-100 small image classification
  - IMDB Movie Reviews Sentiment Classification
  - …

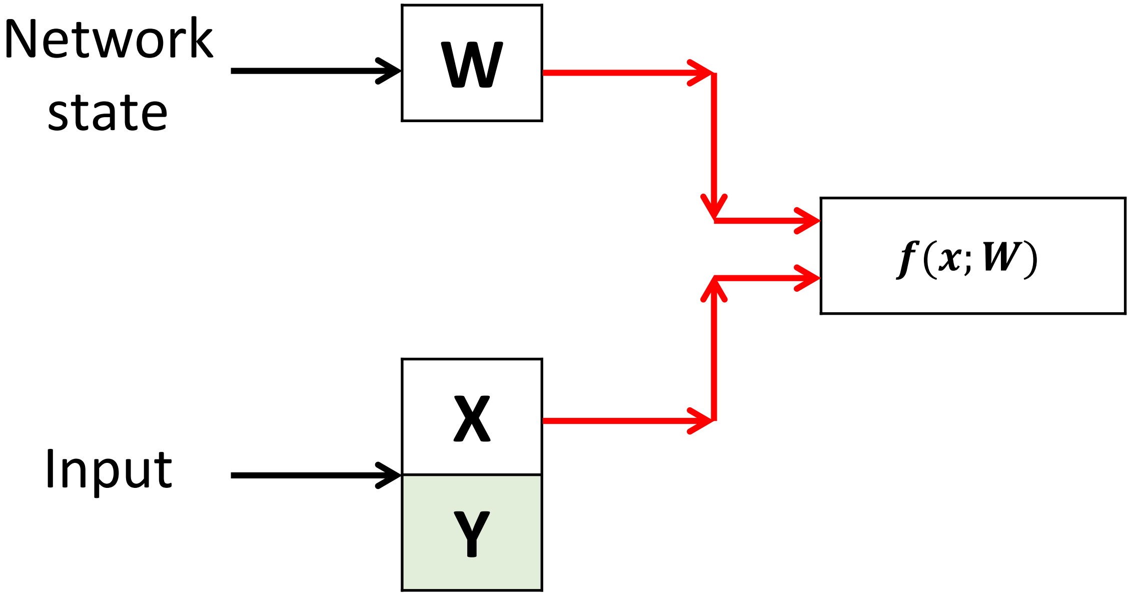


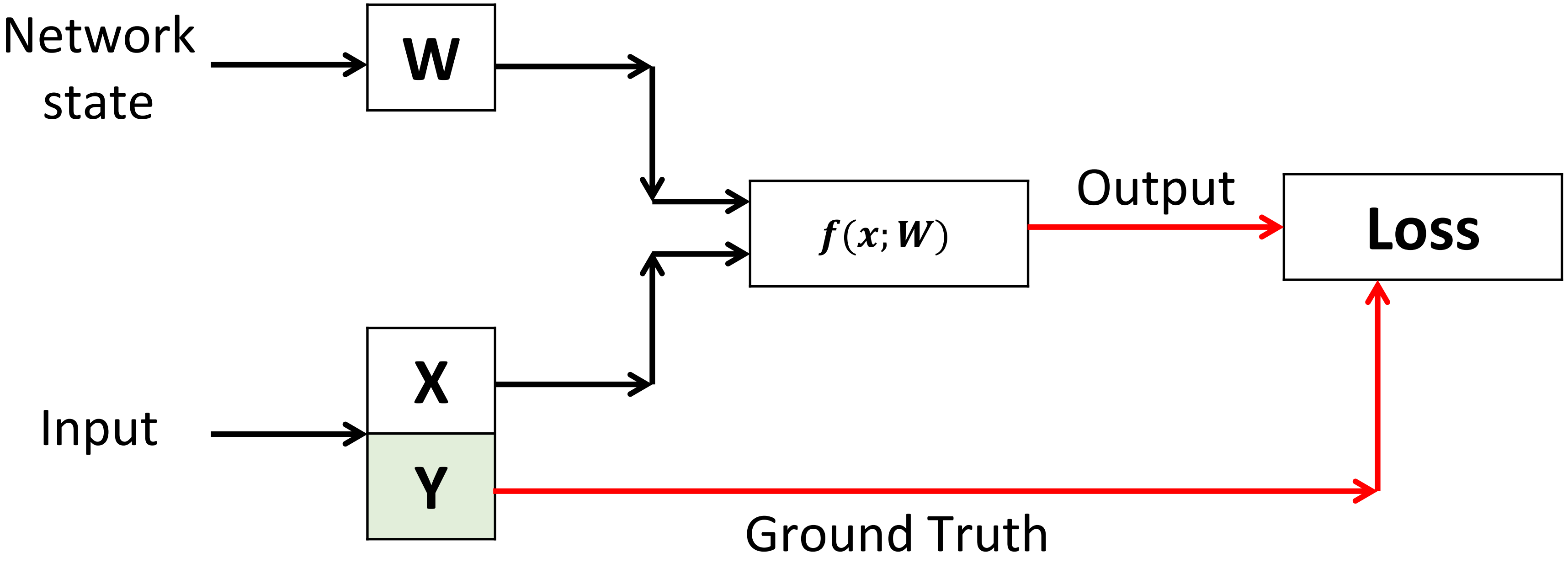
```
from keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
```

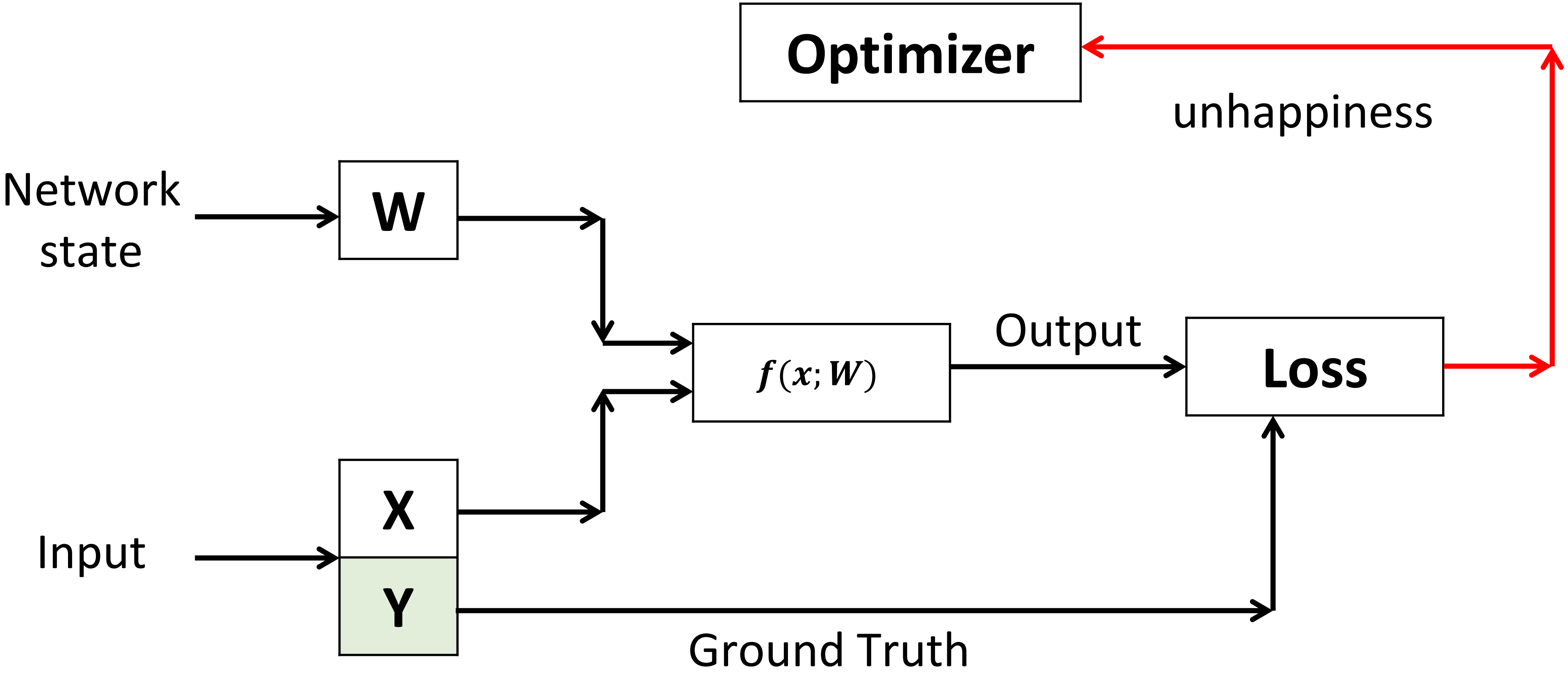
- keras.preprocessing.image:
  - ImageDataGenerator:
    - Generate Batches of tensor image data
    - Feature-wise and sample-wise normalization
    - Whitening
    - Any other preprocessing function.
    - Train Validation Split.
    - Real-Time data Augmentation (will be covered in CNN section)

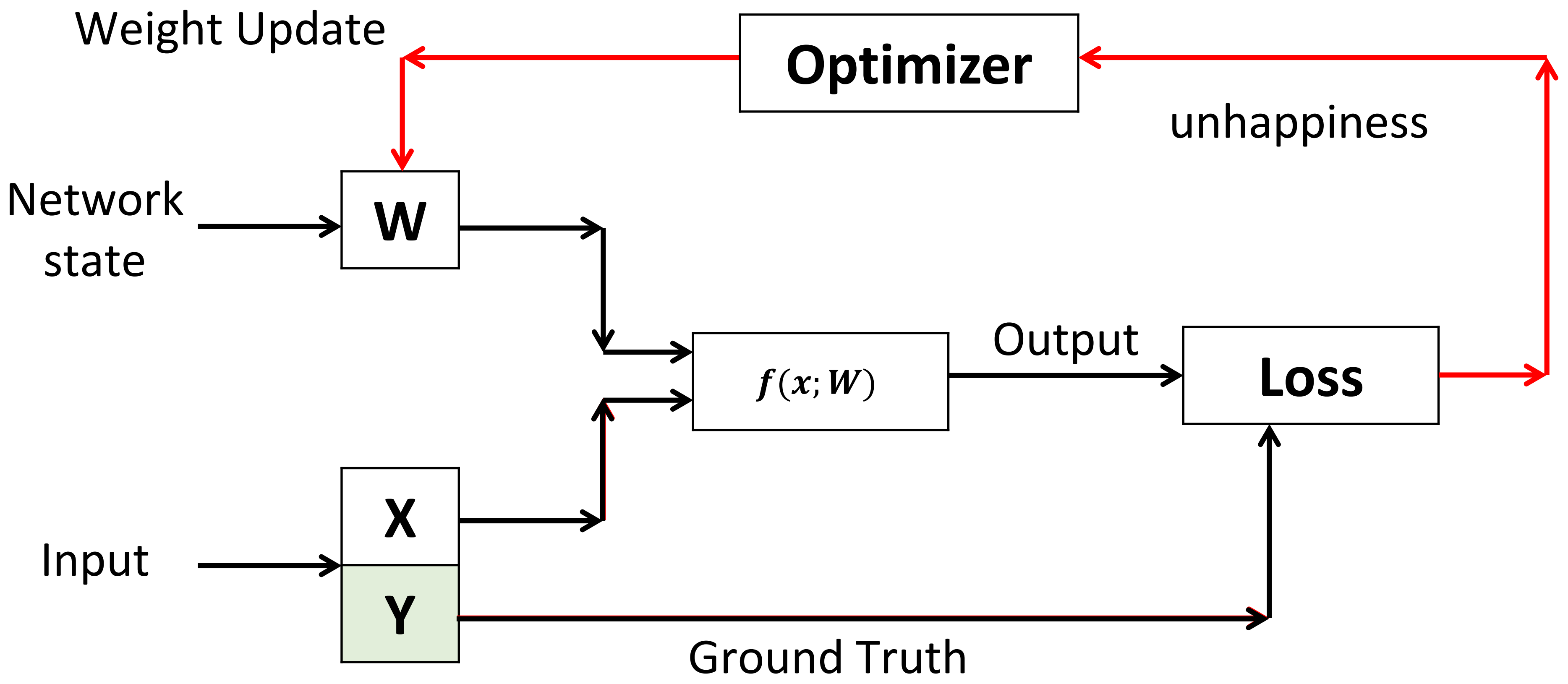
- keras.preprocessing.text:
  - text_to_word_sequence
  - one_hot
  - …
- keras.preprocessing.sequence:
  - TimeseriesGenerator Class
  - pad_sequences


Network state
(Should be initialized)
W
Input
X
Y


Network state
W
Input
X
Y
$f(x;W)$


Network state
W
Input
X
Y
$f(x;W)$
Output
Loss
Ground Truth


Optimizer
unhappiness
Network state
W
$f(x; W)$
Output
Loss
X
Input
Y
Ground Truth


Weight Update
Optimizer
unhappiness
Network state
W
$f(x; W)$
Output
Loss
Input
X
Y
Ground Truth

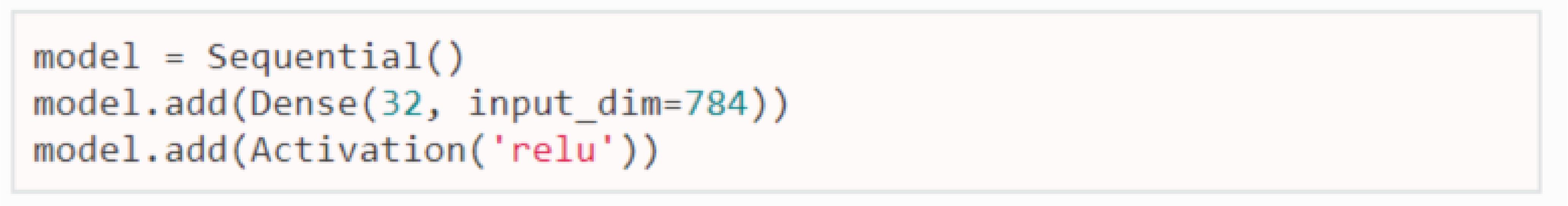
Weight Update:

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

$$\Delta \mathrm{w}_{\mathrm{ji}} = -\eta \frac{\partial E}{\partial \mathrm{w}_{\mathrm{ji}}}$$

There are two main types of models available in Keras:

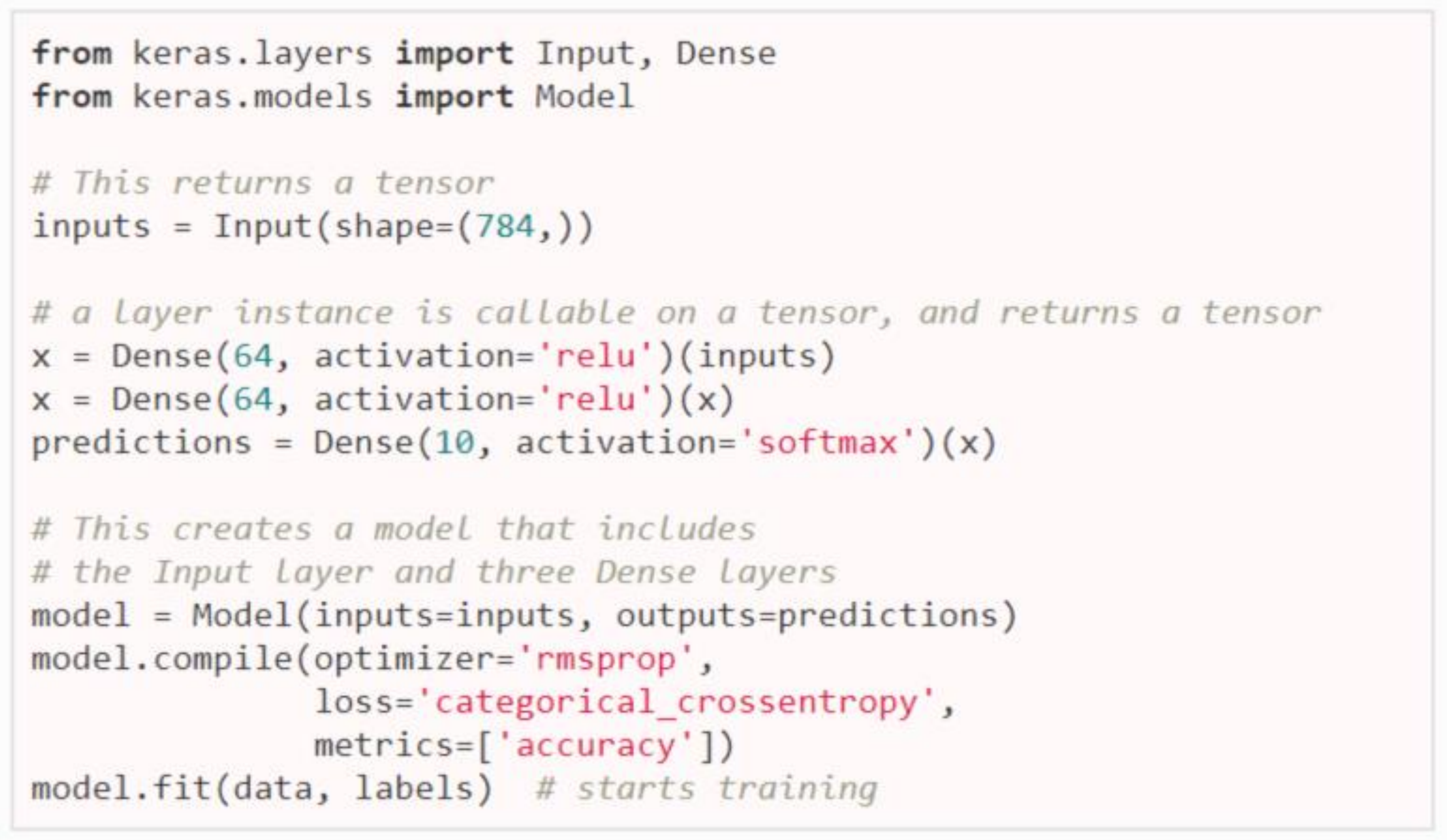
- The Sequential model



```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

There are two main types of models available in Keras:

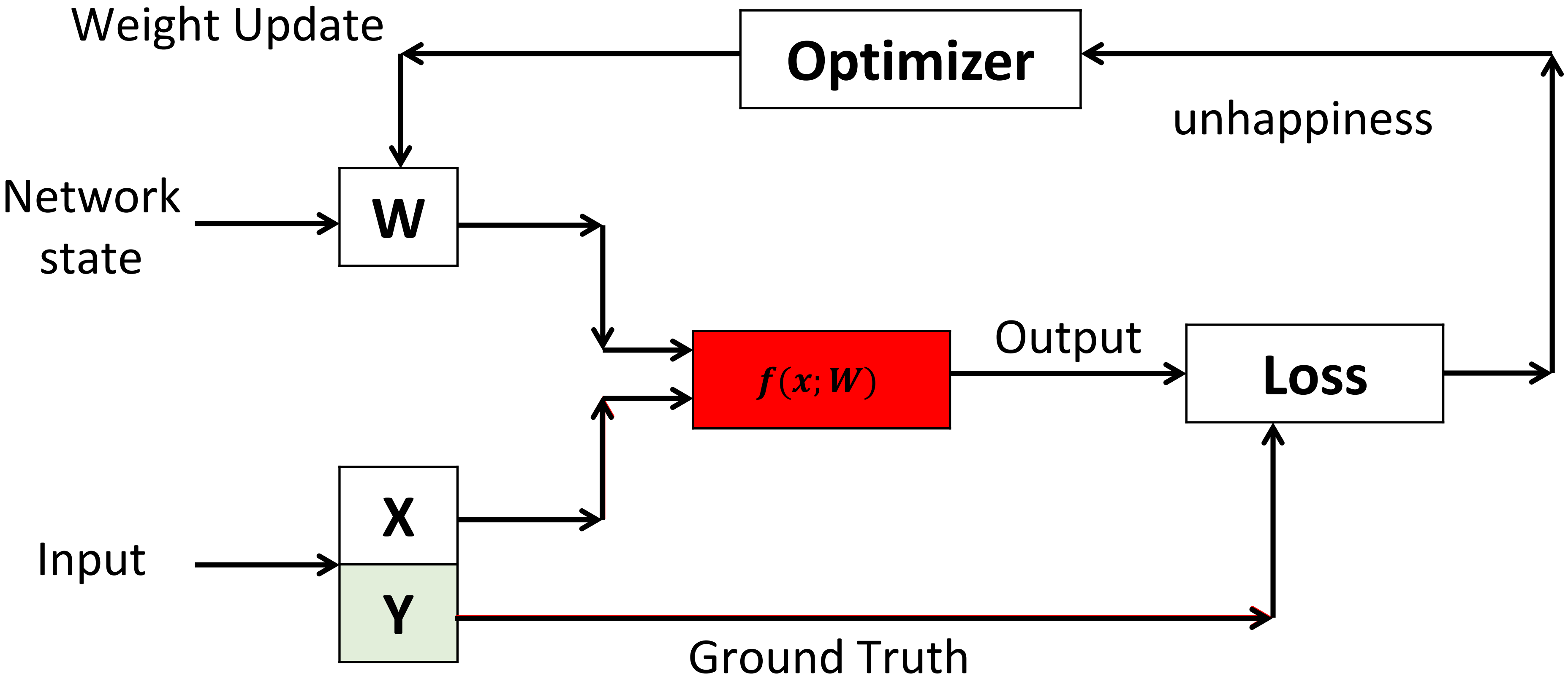
- The Sequential model
- The Model class



```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

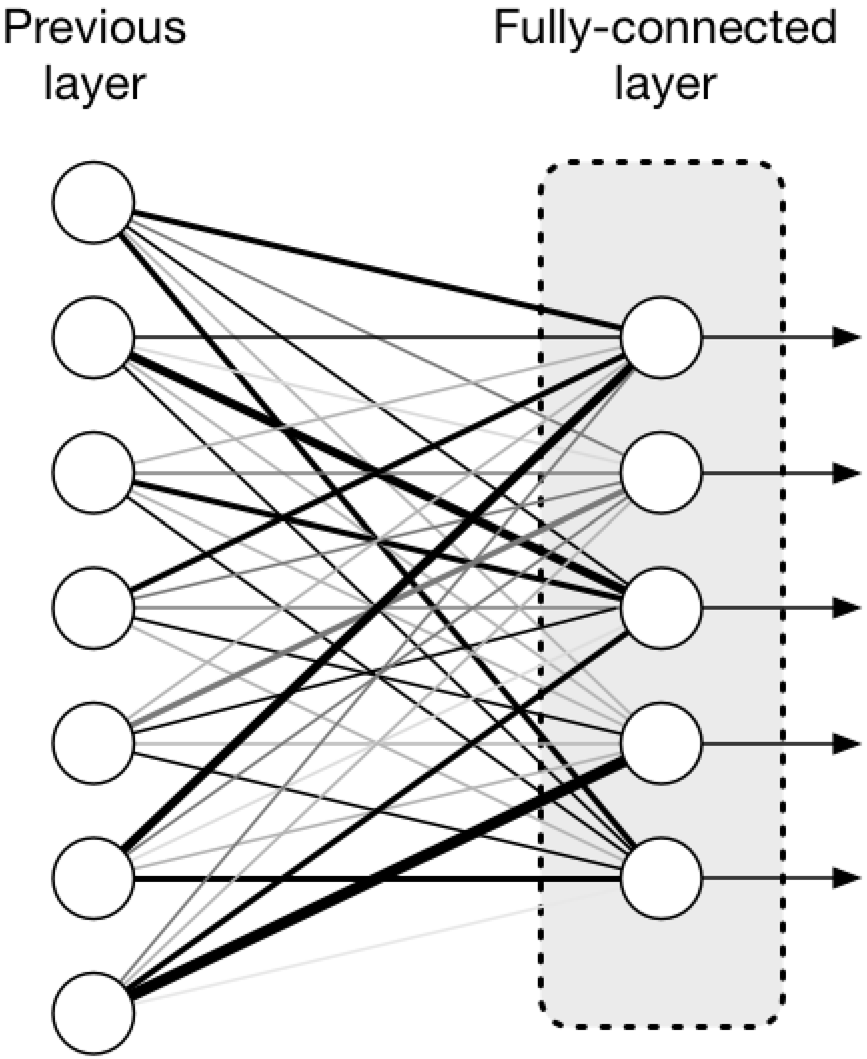
There are two main types of models available in Keras:

- The Sequential model
- The Model class
- Model subclassing


Weight Update
Optimizer
unhappiness
Network state
W
$f(x;W)$
Output
Loss
X
Input
Y
Ground Truth

Keras.layers

- Core layers:
  - **Dense**
  - Activation
  - Input
  - Reshape
  - Flatten
  - …
- Convolutional Layers (will be covered later)
- Recurrent Layers (will be covered later)

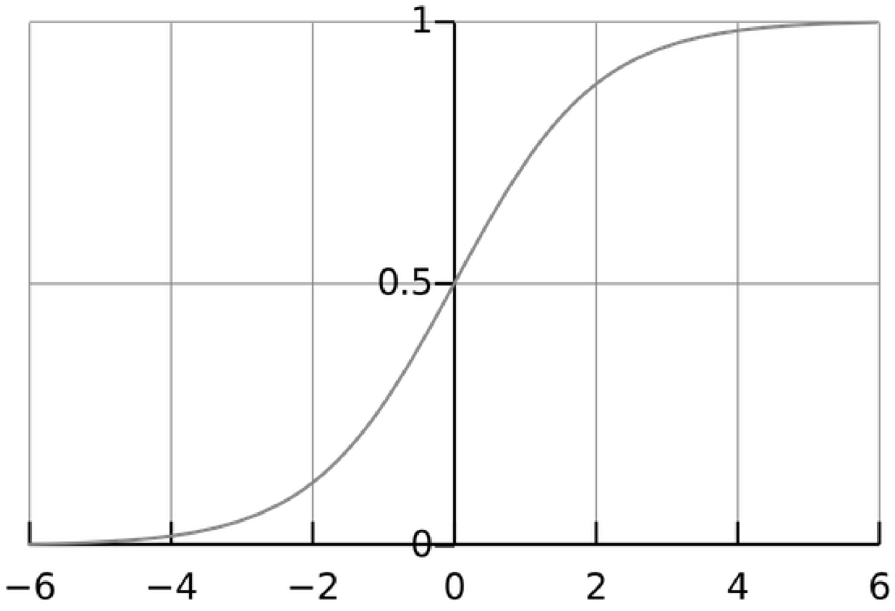


Keras.layers

- Core layers:
  - Dense
  - Activation
  - Input
  - Reshape
  - Flatten
  - …
- Convolutional Layers (will be covered later)
- Recurrent Layers (will be covered later)

Activation:

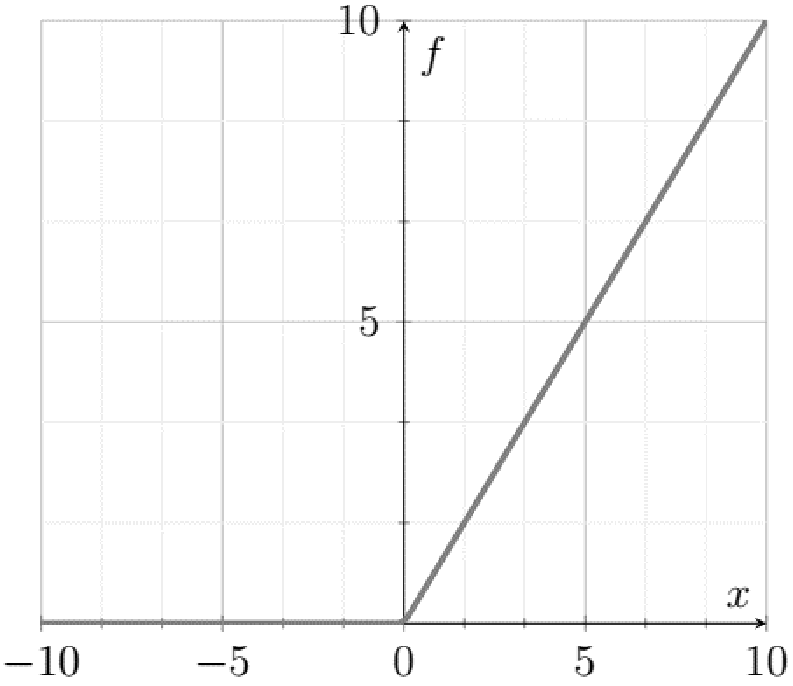
Sigmoid
Relu
Tanh
Softmax
…



Keras.layers

- Core layers:
  - Dense
  - **Activation**
  - Input
  - Reshape
  - Flatten
  - …
- Convolutional Layers (will be covered later)
- Recurrent Layers (will be covered later)

Sigmoid
Relu
Tanh
Softmax
…

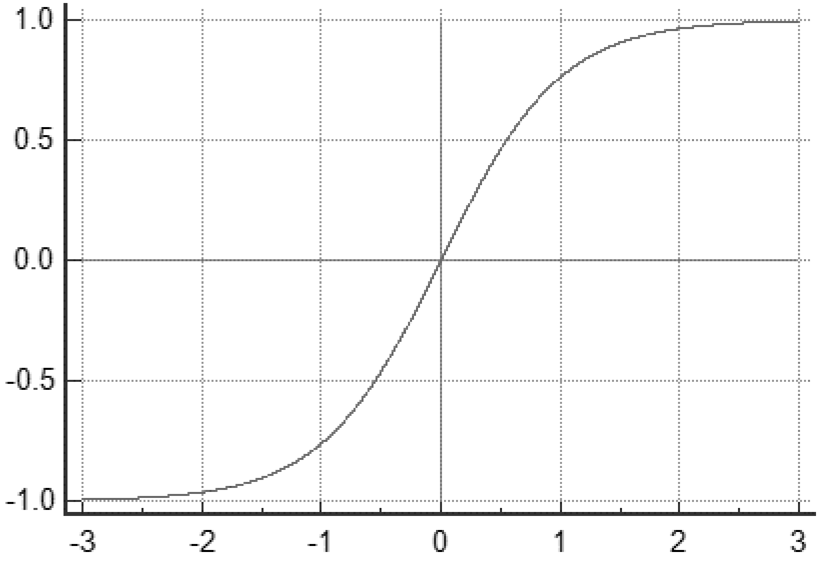


Keras.layers

- Core layers:
  - Dense
  - **Activation**
  - Input
  - Reshape
  - Flatten
  - …
- Convolutional Layers (will be covered later)
- Recurrent Layers (will be covered later)

Activation:

Sigmoid
Relu
Tanh
Softmax
…



Keras.layers

- Core layers:
  - Dense
  - **Activation**
  - Input
  - Reshape
  - Flatten
  - …
- Convolutional Layers (will be covered later)
- Recurrent Layers (will be covered later)

Sigmoid
Relu
Tanh
Softmax
…

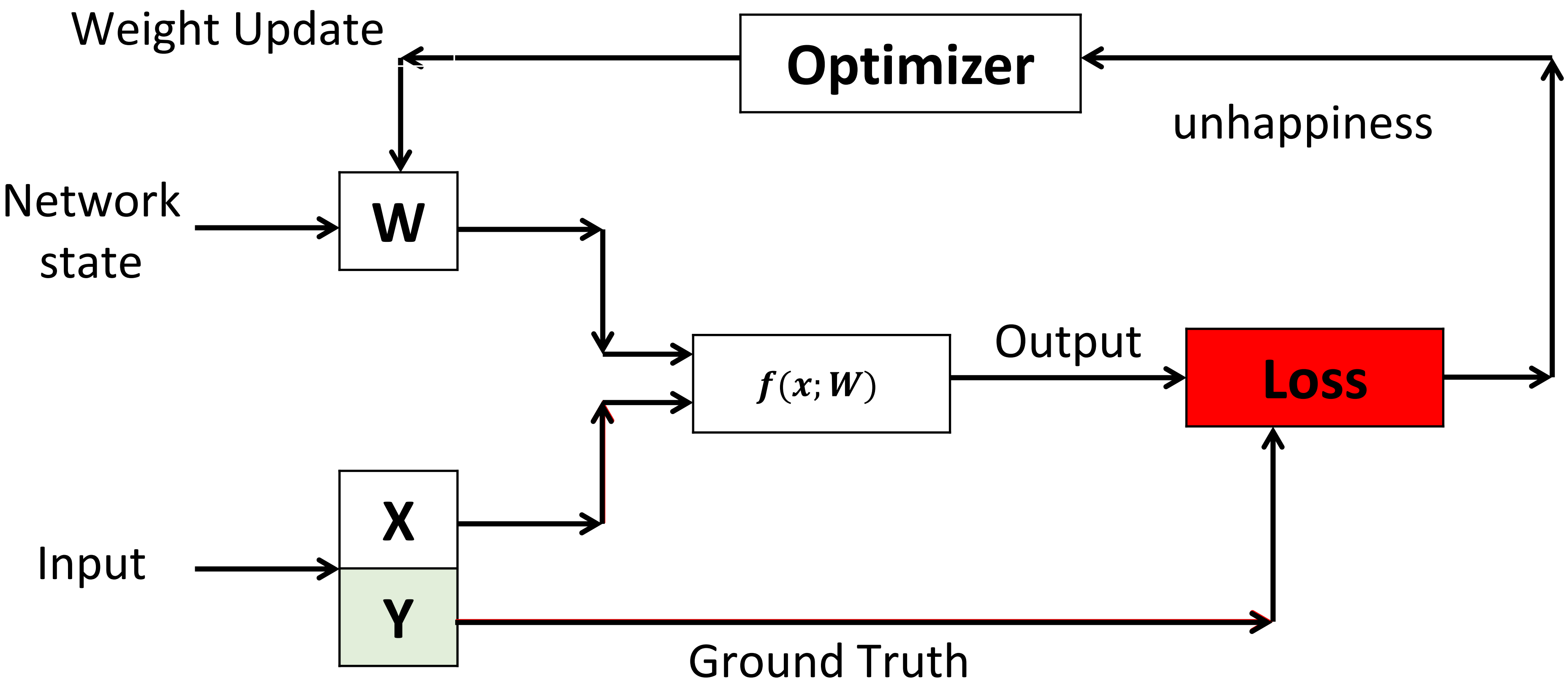
$$\mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Keras.layers

- Core layers:
  - Dense
  - Activation
  - **Input**
  - **Reshape**
  - **Flatten**
  - …
- Convolutional Layers (will be covered later)
- Recurrent Layers (will be covered later)

Keras.layers

- Core layers:
  - Dense
  - Activation
  - Input
  - Reshape
  - Flatten
  - …
- **Convolutional Layers (will be covered later)**
- **Recurrent Layers (will be covered later)**


Weight Update
Optimizer
unhappiness
Network state
W
$f(x; W)$
Output
Loss
X
Input
Y
Ground Truth

Keras.losses

- **Mean_squared_error**
- Mean_absolute_error
- Categorical_crossentropy
- …

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i]$$

$$H = -\sum_{x} y_i \log(f_i)$$

Keras.losses

- Mean_squared_error
- **Mean_absolute_error**
- Categorical_crossentropy
- …

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i]$$

$$H = -\sum_{x} y_i \log(f_i)$$

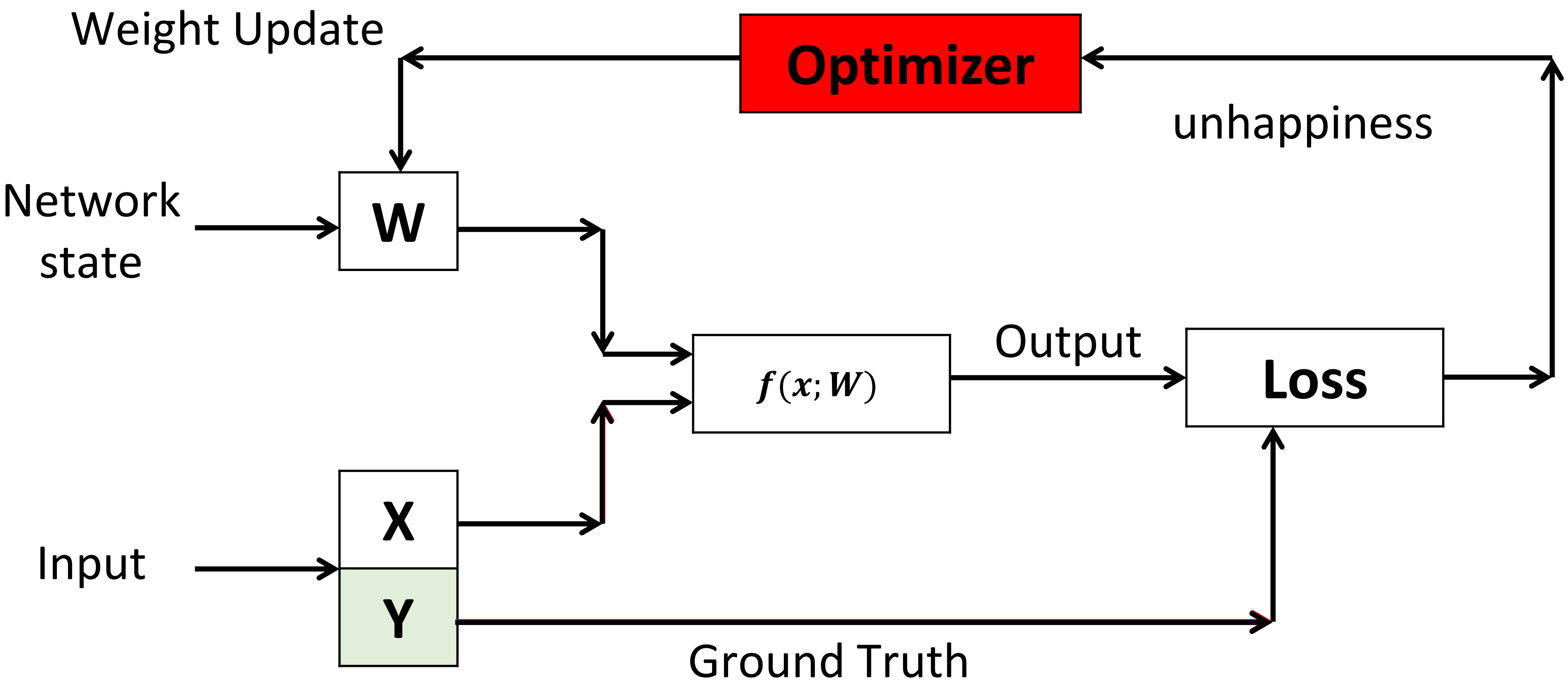
Keras.losses

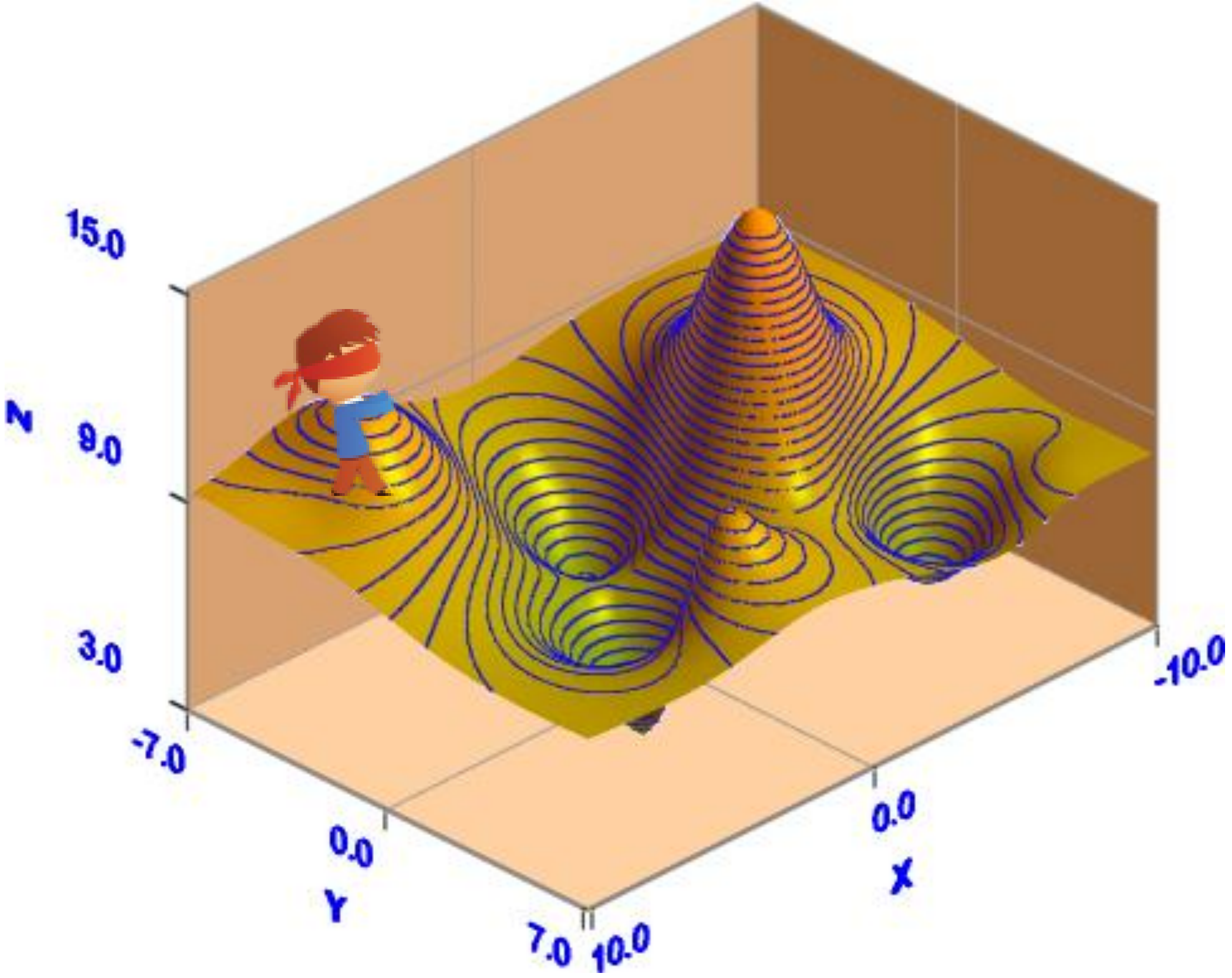
- Mean_squared_error
- Mean_absolute_error
- **Categorical_crossentropy**
- …

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|f_i - y_i]$$

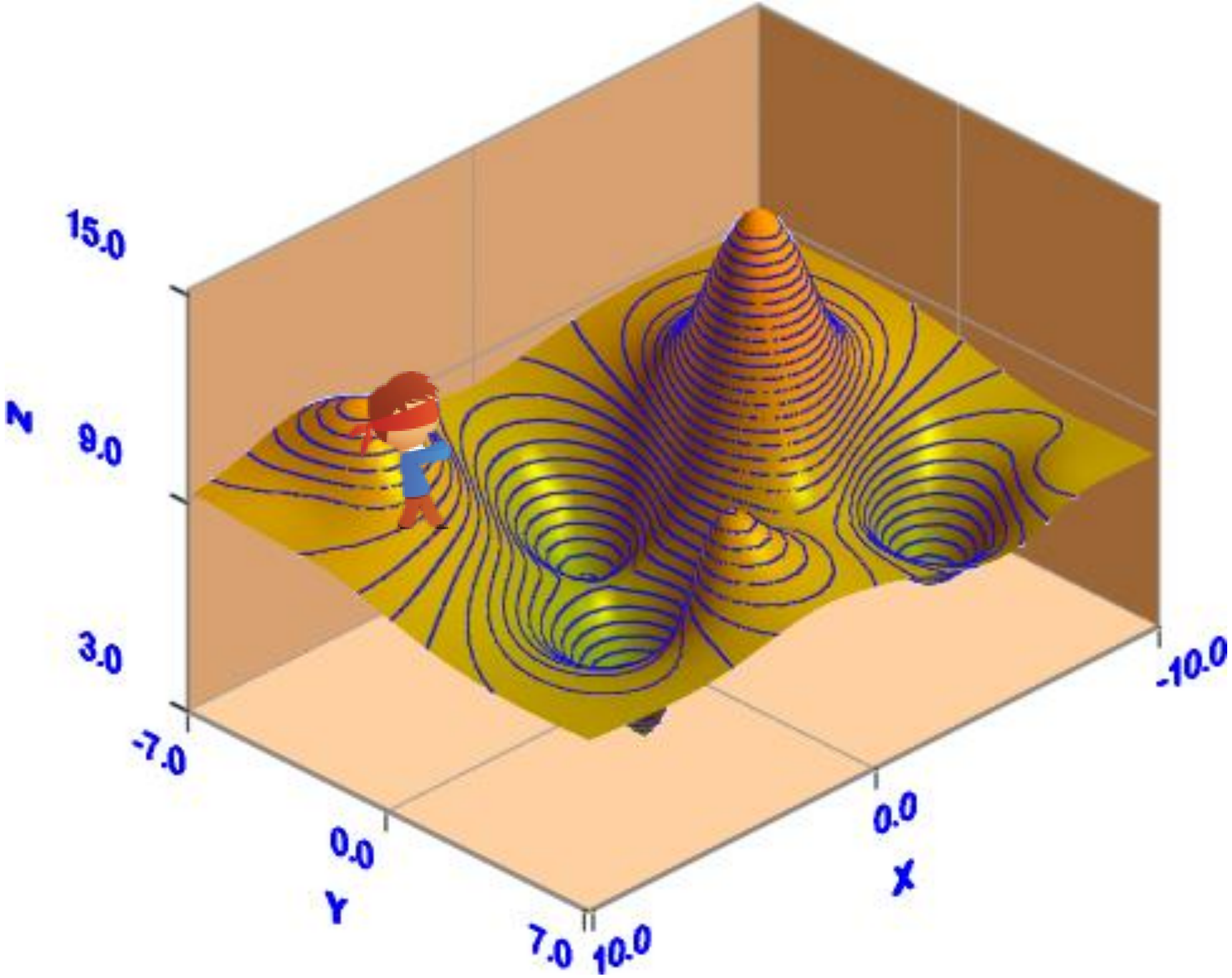
$$H = -\sum_{x} y_i \log(f_i)$$


Weight Update
Optimizer
unhappiness
Network state
W
$f(x;W)$
Output
Loss
X
Input
Y
Ground Truth


15.0
Z
9.0
3.0
-7.0
0.0
Y
7.0
10.0
0.0
X
-10.0

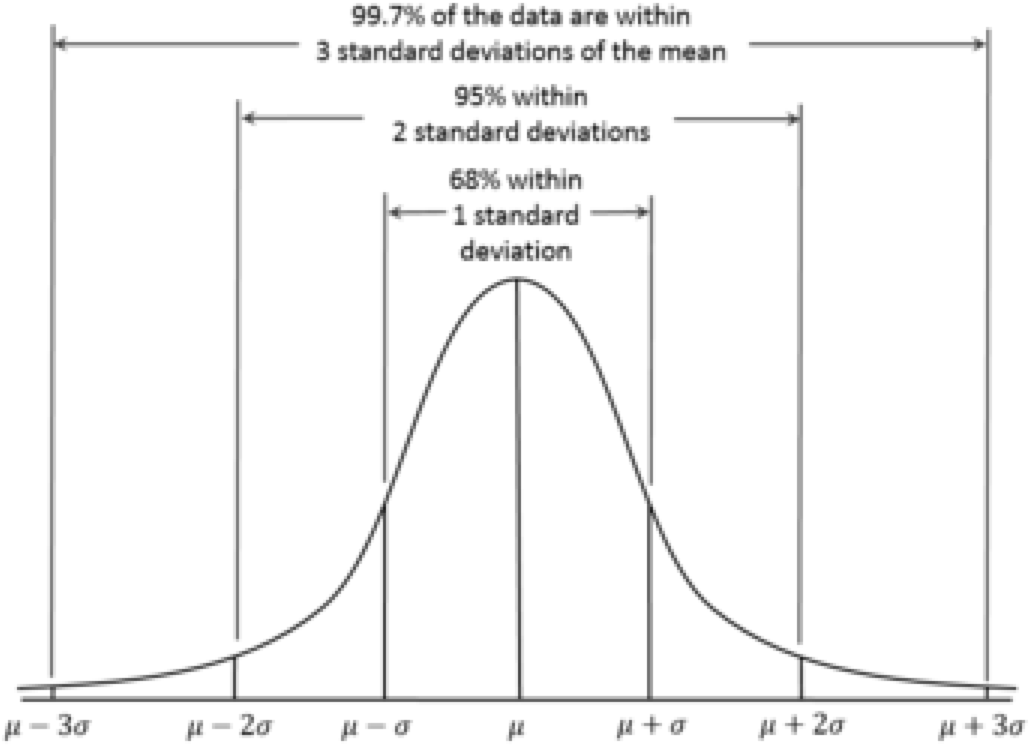
Keras.Optimizers

- Stochastic Gradient Descent
- RMSprop
- Adam



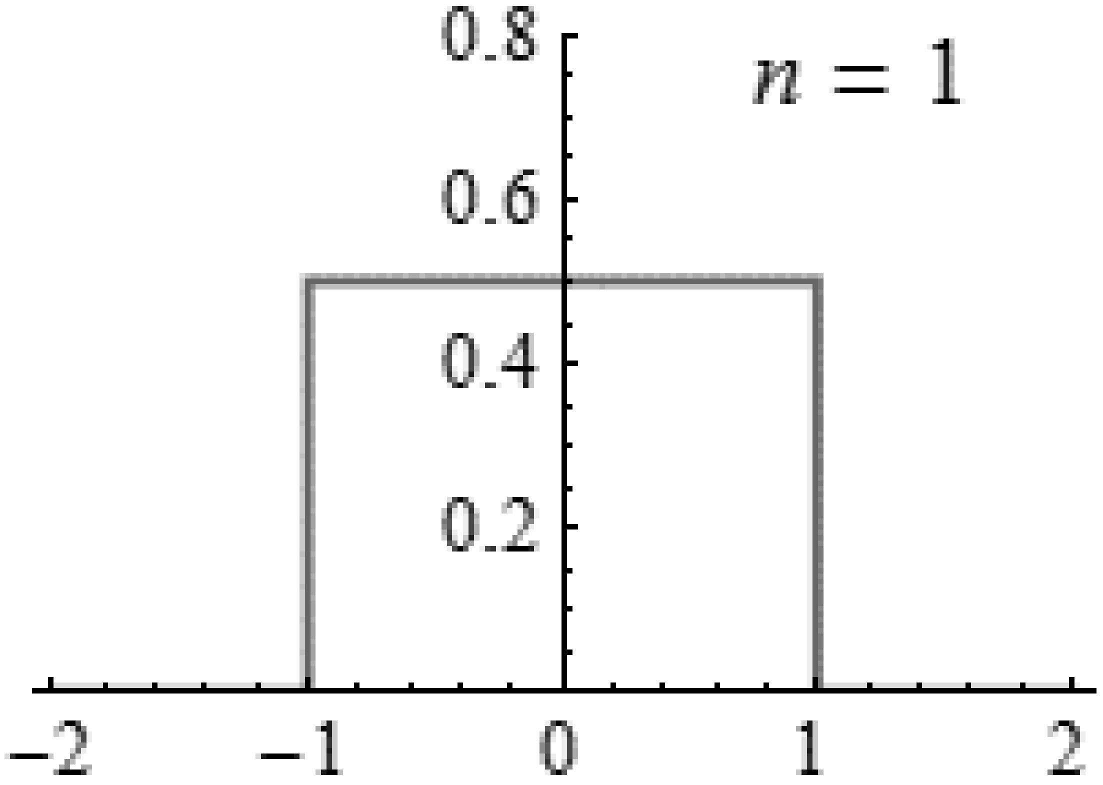
Keras.initializer

- **RandomNormal**
- RandomUniform
- Glorot_uniform
- Glorot_normal


99.7% of the data are within 3 standard deviations of the mean
95% within 2 standard deviations
68% within 1 standard deviation
$\mu - 3\sigma$
$\mu - 2\sigma$
$\mu - \sigma$
μ
$\mu + \sigma$
$\mu + 2\sigma$
$\mu + 3\sigma$

Keras.initializer

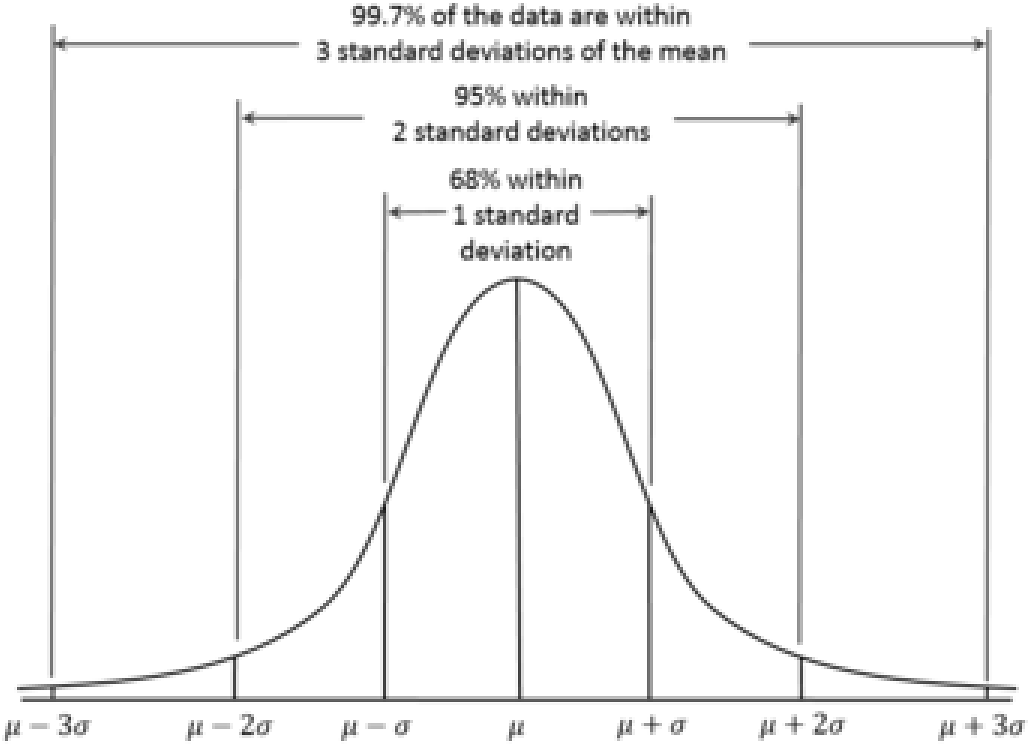
- RandomNormal
- **RandomUniform**
- Glorot_uniform
- Glorot_normal



Keras.initializer

- RandomNormal
- RandomUniform
- **Glorot_normal**
- Glorot_uniform

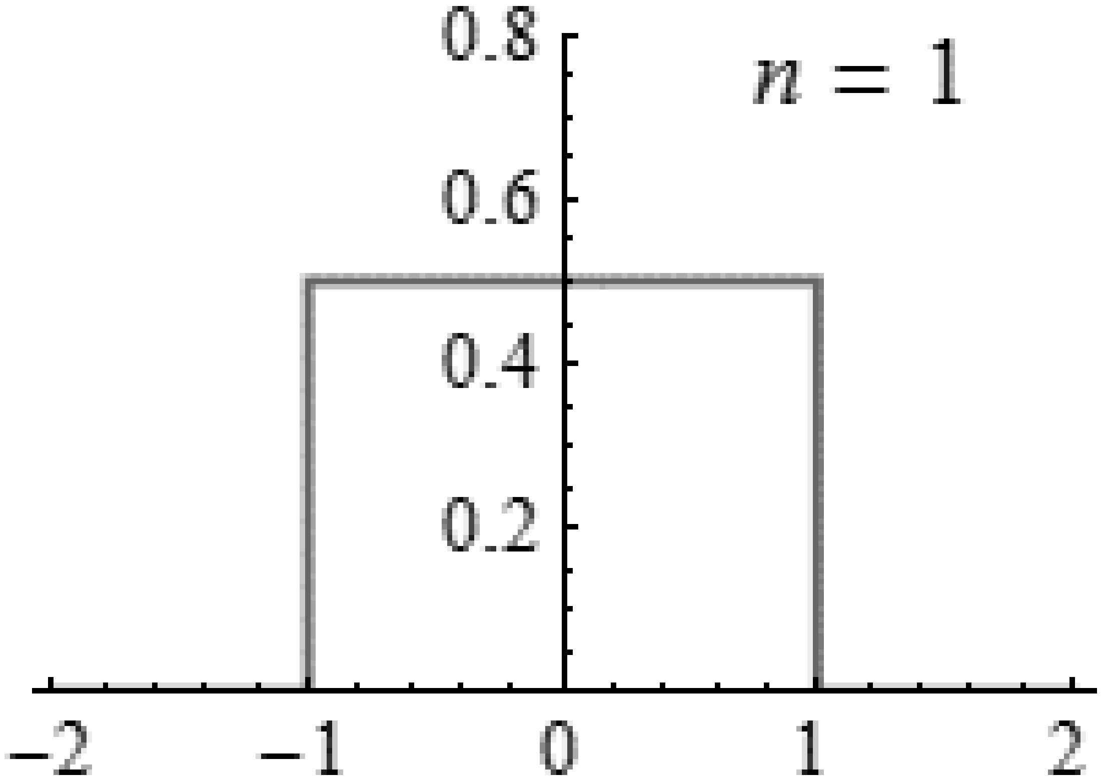
$$\sigma = \sqrt{\frac{2}{fan_{in}+fan_{out}}}$$

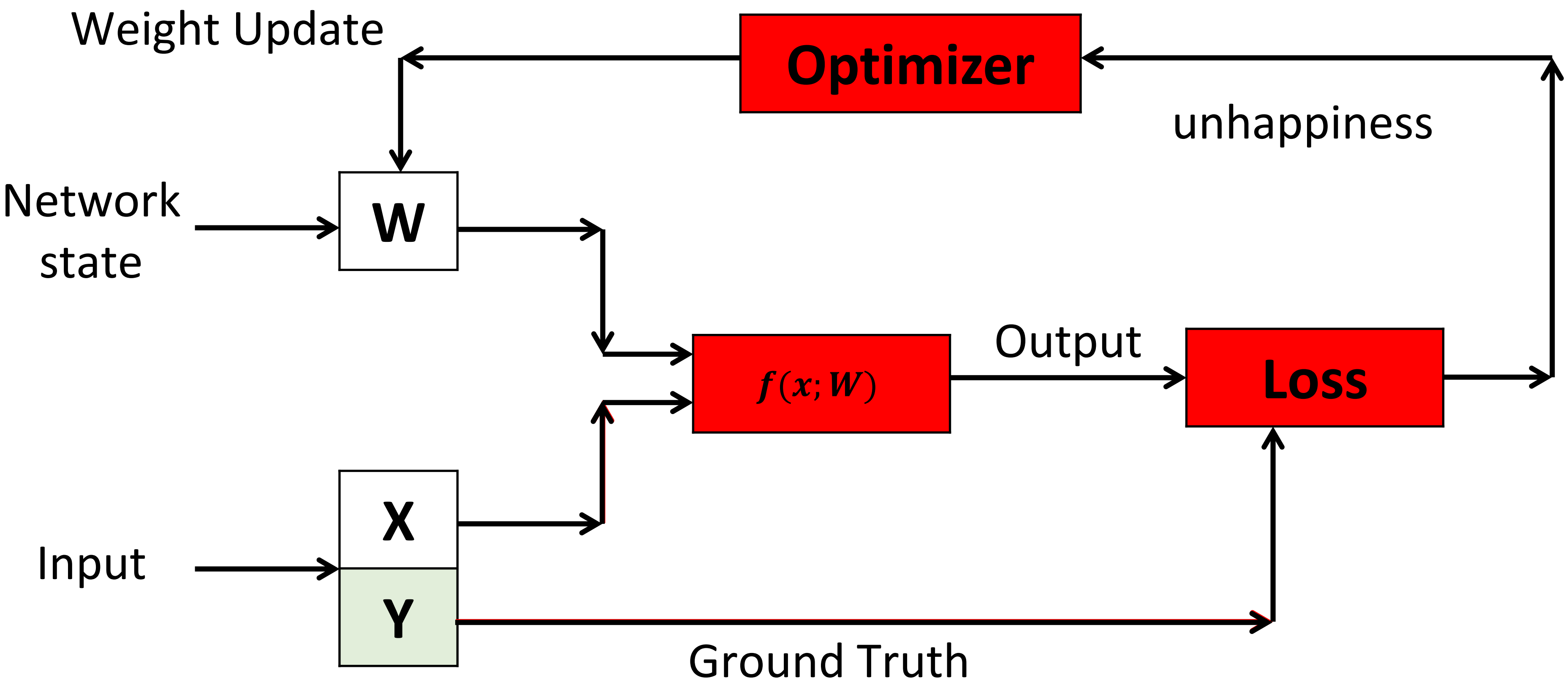


Keras.initializer

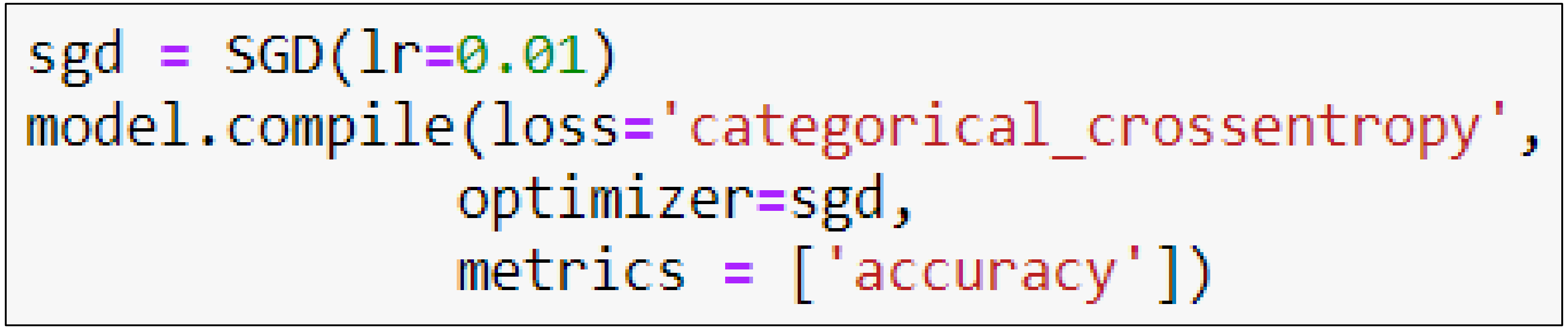
- RandomNormal
- RandomUniform
- Glorot_normal
- **Glorot_uniform**

$$n = \sqrt{\frac{6}{fan_{in} + fan_{out}}}$$

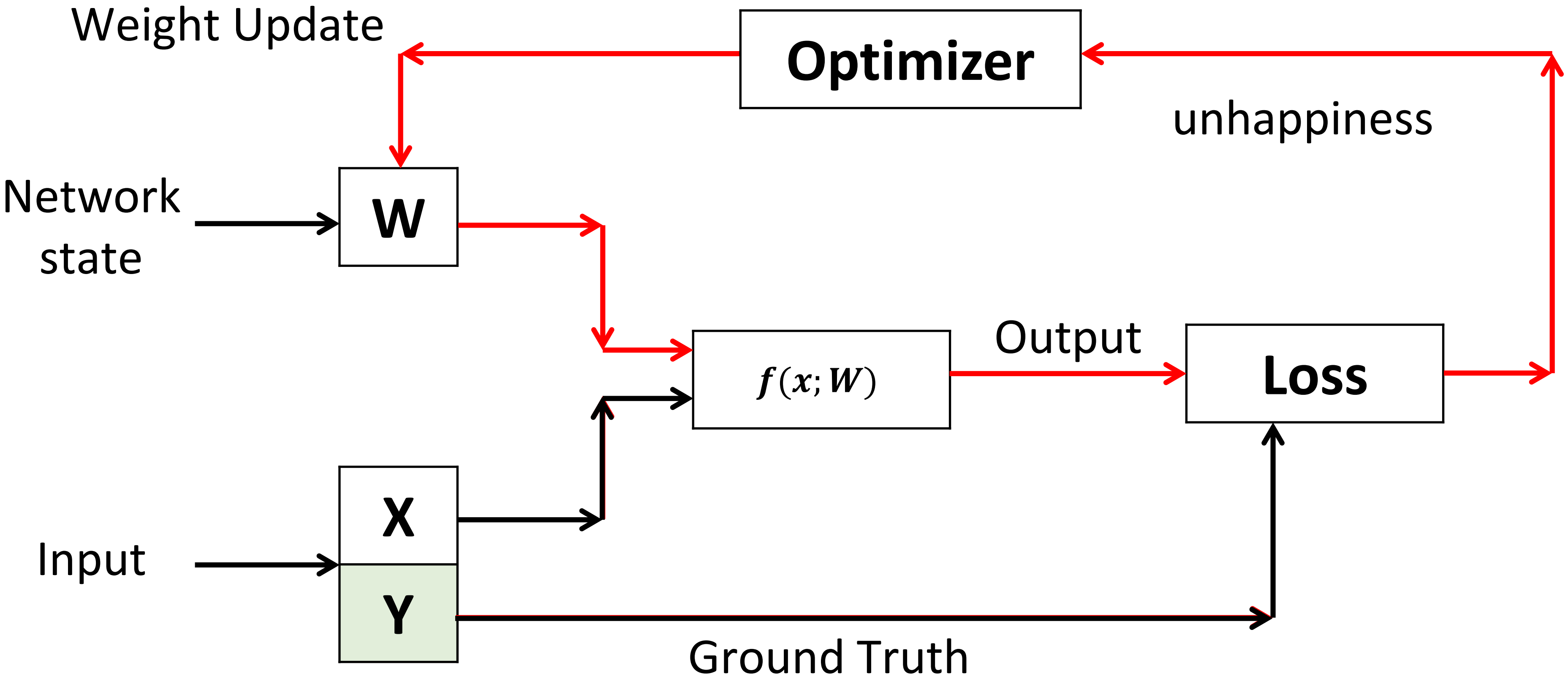



Weight Update
Optimizer
unhappiness
Network state
W
$f(x;W)$
Output
Loss
X
Input
Y
Ground Truth

model.compile



```
sgd = SGD(lr=0.01)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics = ['accuracy'])
```



Weight Update

Optimizer

unhappiness

Network state

W

$f(x; W)$

Output

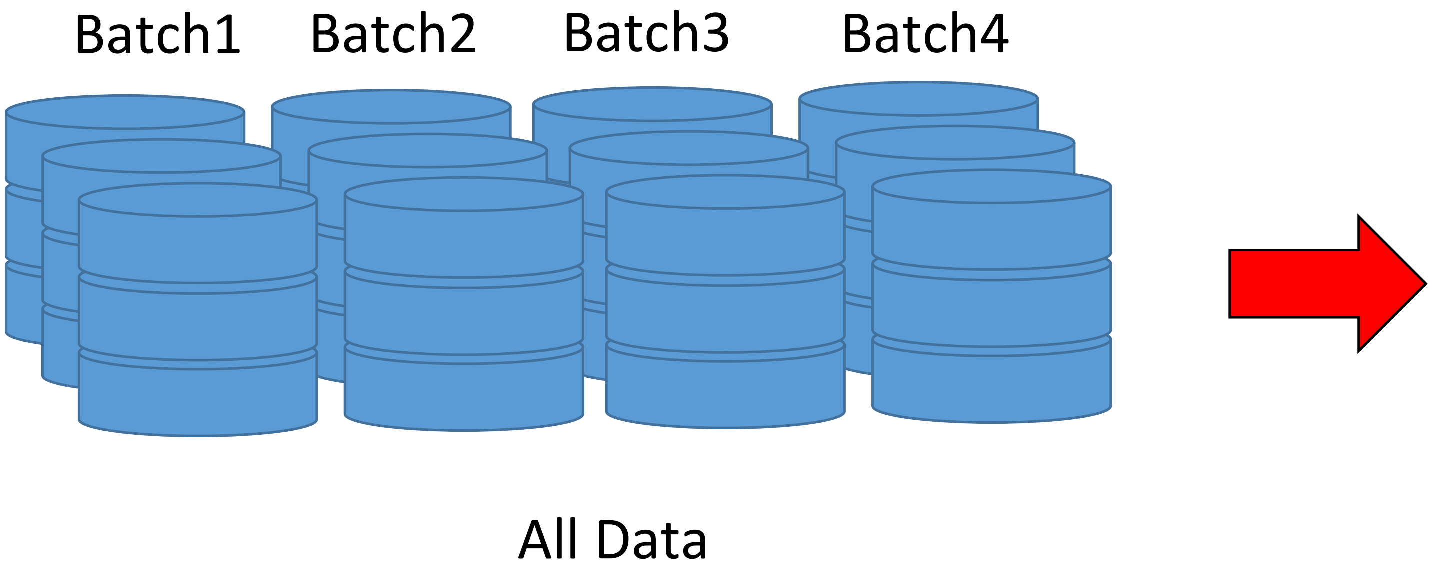
Loss

Input

X

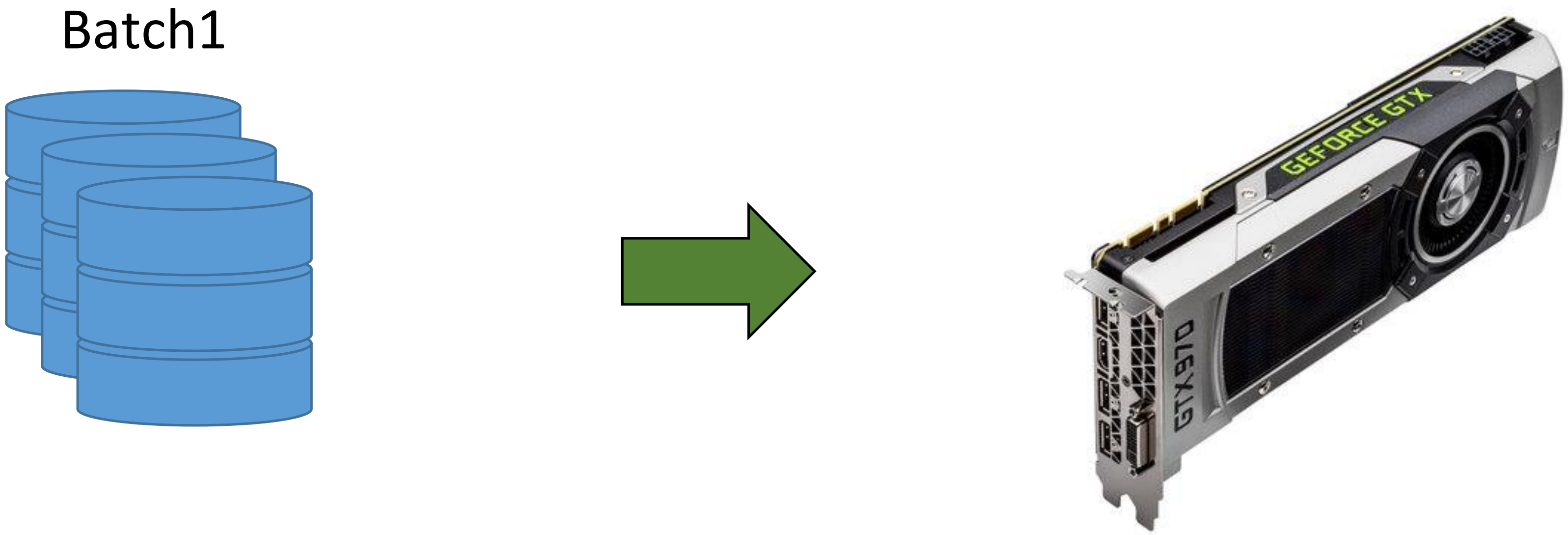
Y

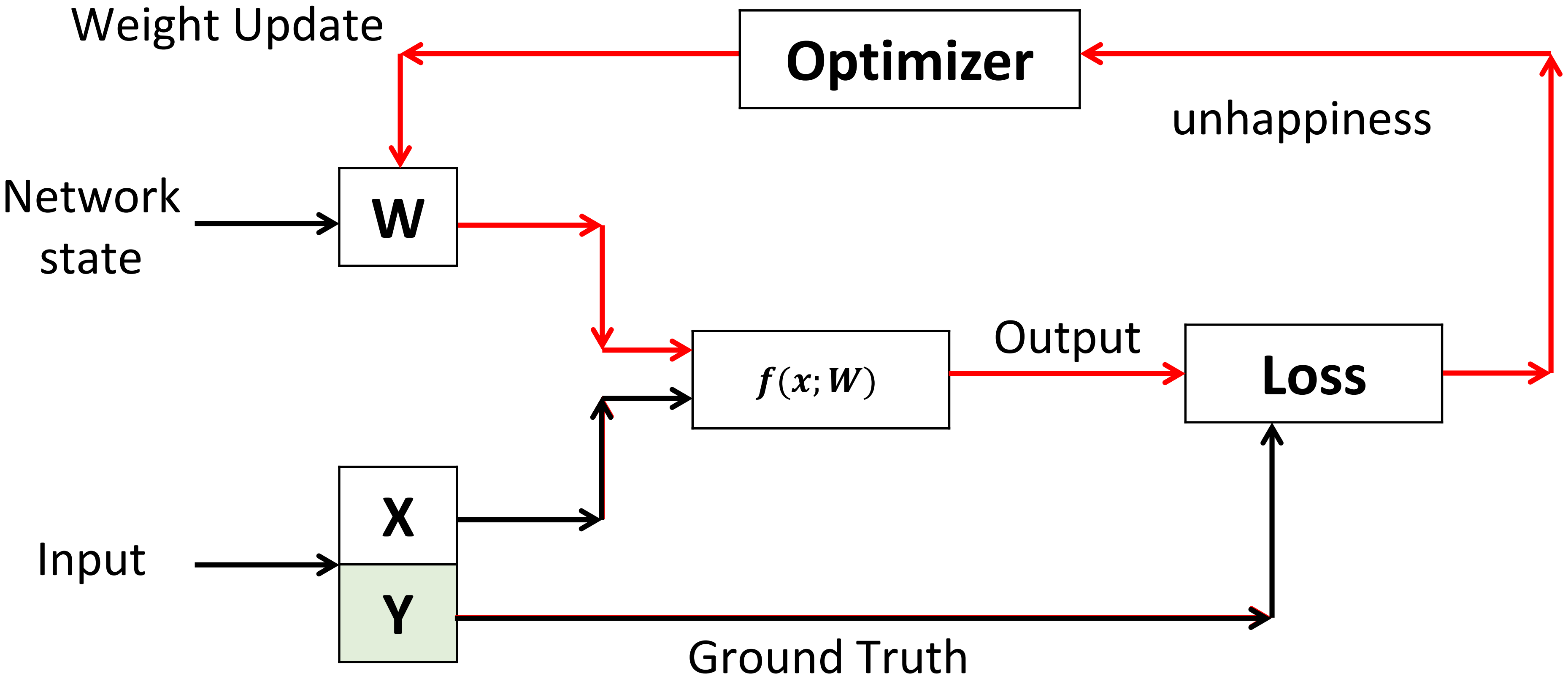
Ground Truth



Batch1 Batch2 Batch3 Batch4

All Data


Batch1
GEFORCE GTX
GTX970



Weight Update

Optimizer

unhappiness

Network state

W

$f(x;W)$

Output

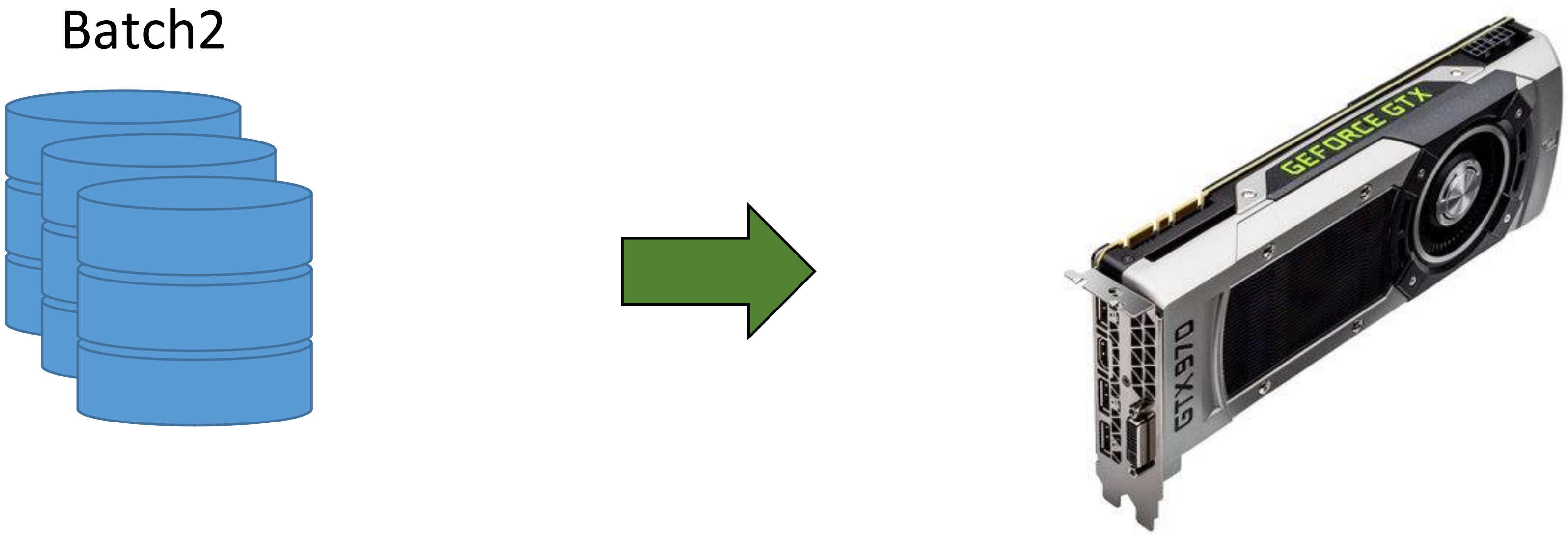
Loss

Input

X

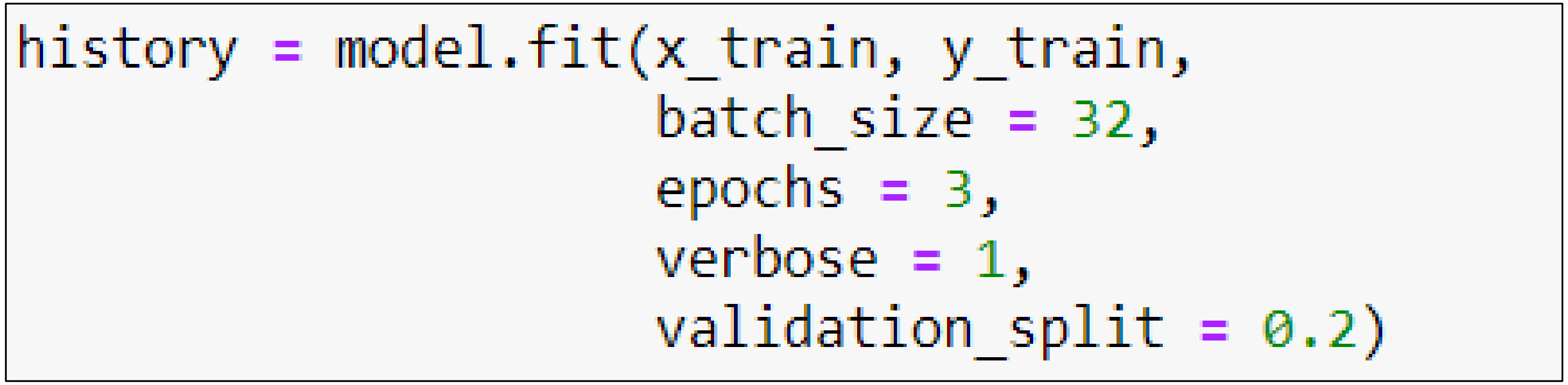
Y

Ground Truth



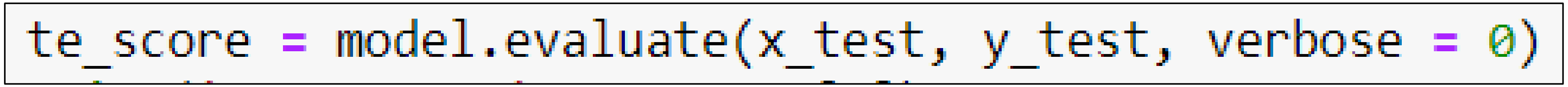
Batch2

model.fit



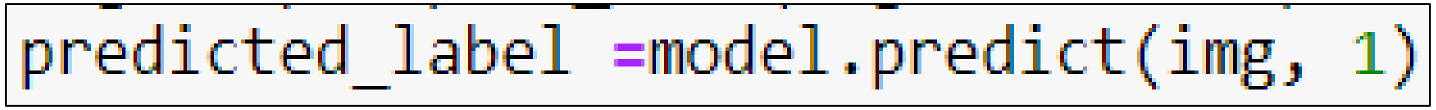
```
history = model.fit(x_train, y_train,
                    batch_size = 32,
                    epochs = 3,
                    verbose = 1,
                    validation_split = 0.2)
```

model.evaluate



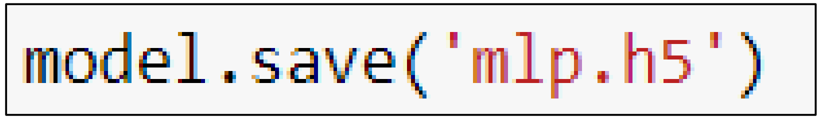
```
te_score = model.evaluate(x_test, y_test, verbose = 0)
```

model.predict

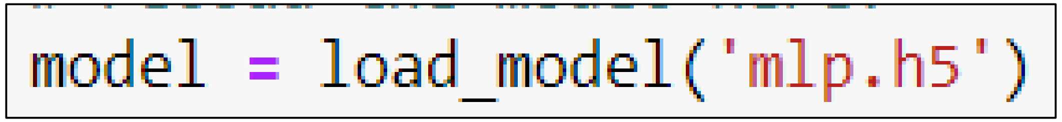


```
predicted_label =model.predict(img, 1)
```

Save/Load model



```
model.save('mlp.h5')
```



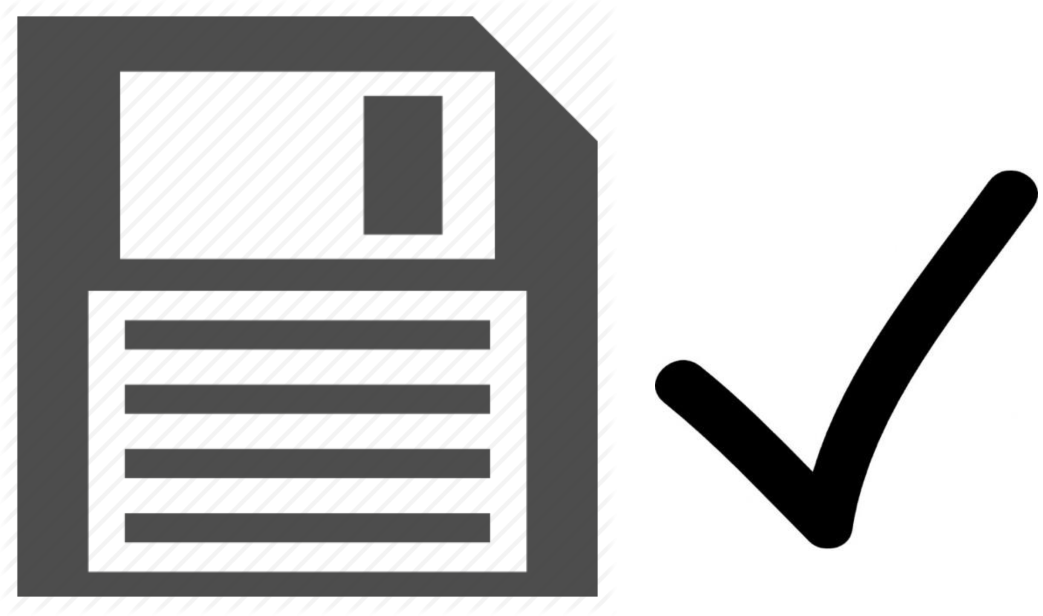
```
model = load_model('mlp.h5')
```

Keras.callbacks:

- **Teminate on NaN.**
- Model Checkpoint
- EarlyStopping
- LearningRateScheduler
- TensorBoard
- CSVLogger

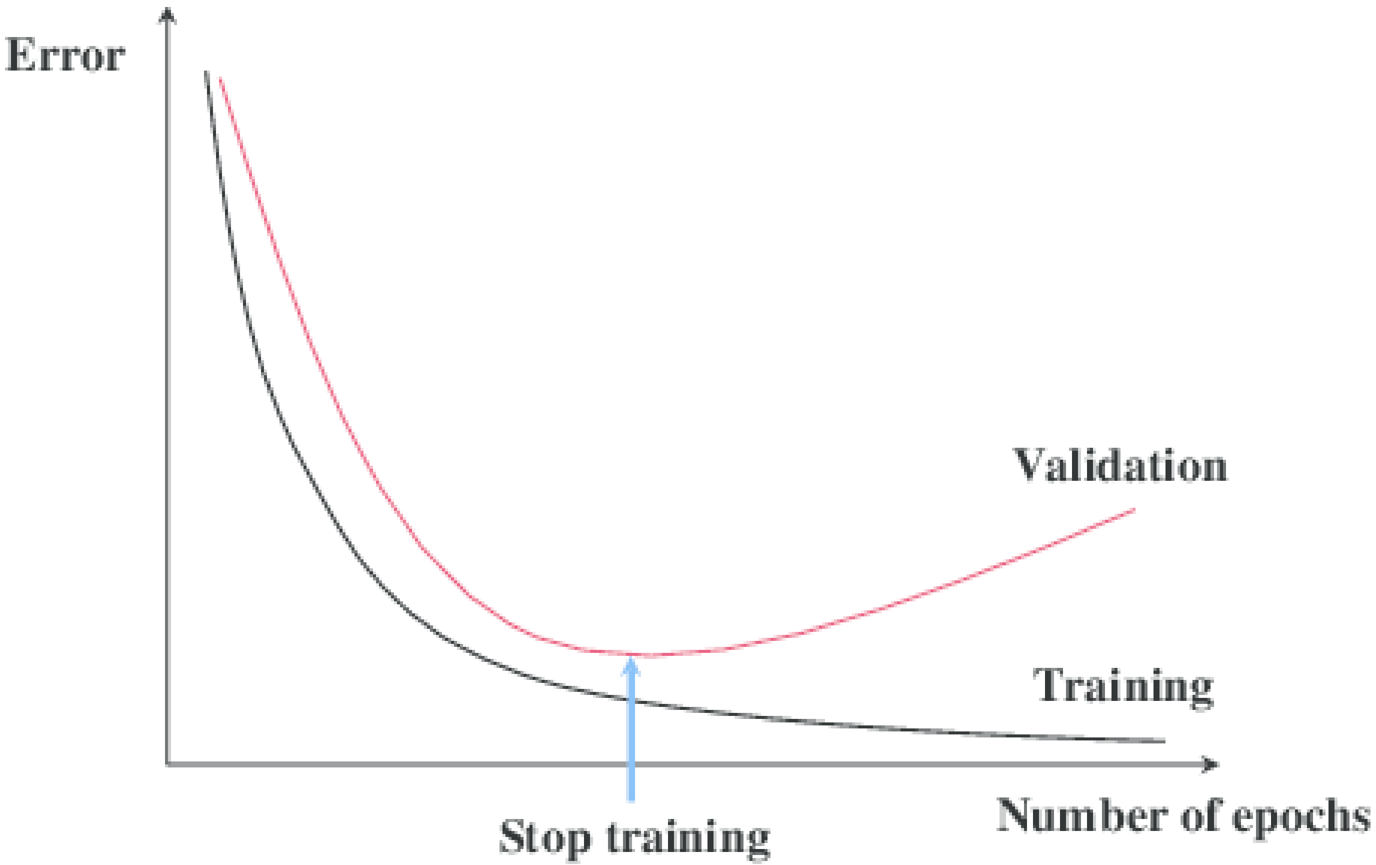
Keras.callbacks:

- Teminate on NaN.
- **Model Checkpoint**
- EarlyStopping
- LearningRateScheduler
- TensorBoard
- CSVLogger



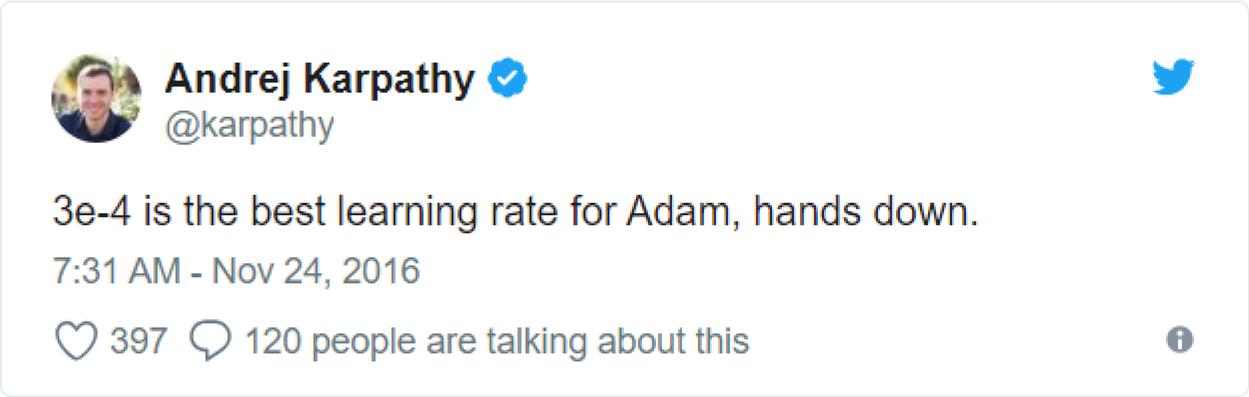
Keras.callbacks:

- Teminate on NaN.
- Model Checkpoint
- **EarlyStopping**
- LearningRateScheduler
- TensorBoard
- CSVLogger



Keras.callbacks:

- Teminate on NaN.
- Model Checkpoint
- EarlyStopping
- **LearningRateScheduler**
- TensorBoard
- CSVLogger

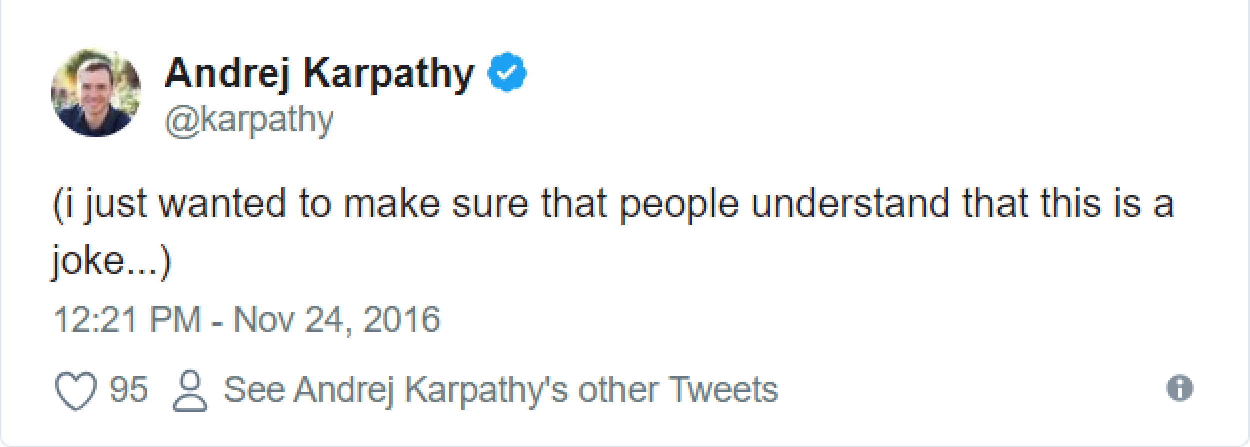


Andrej Karpathy
@karpathy

3e-4 is the best learning rate for Adam, hands down.

7:31 AM - Nov 24, 2016

♡ 397 120 people are talking about this



Andrej Karpathy
@karpathy

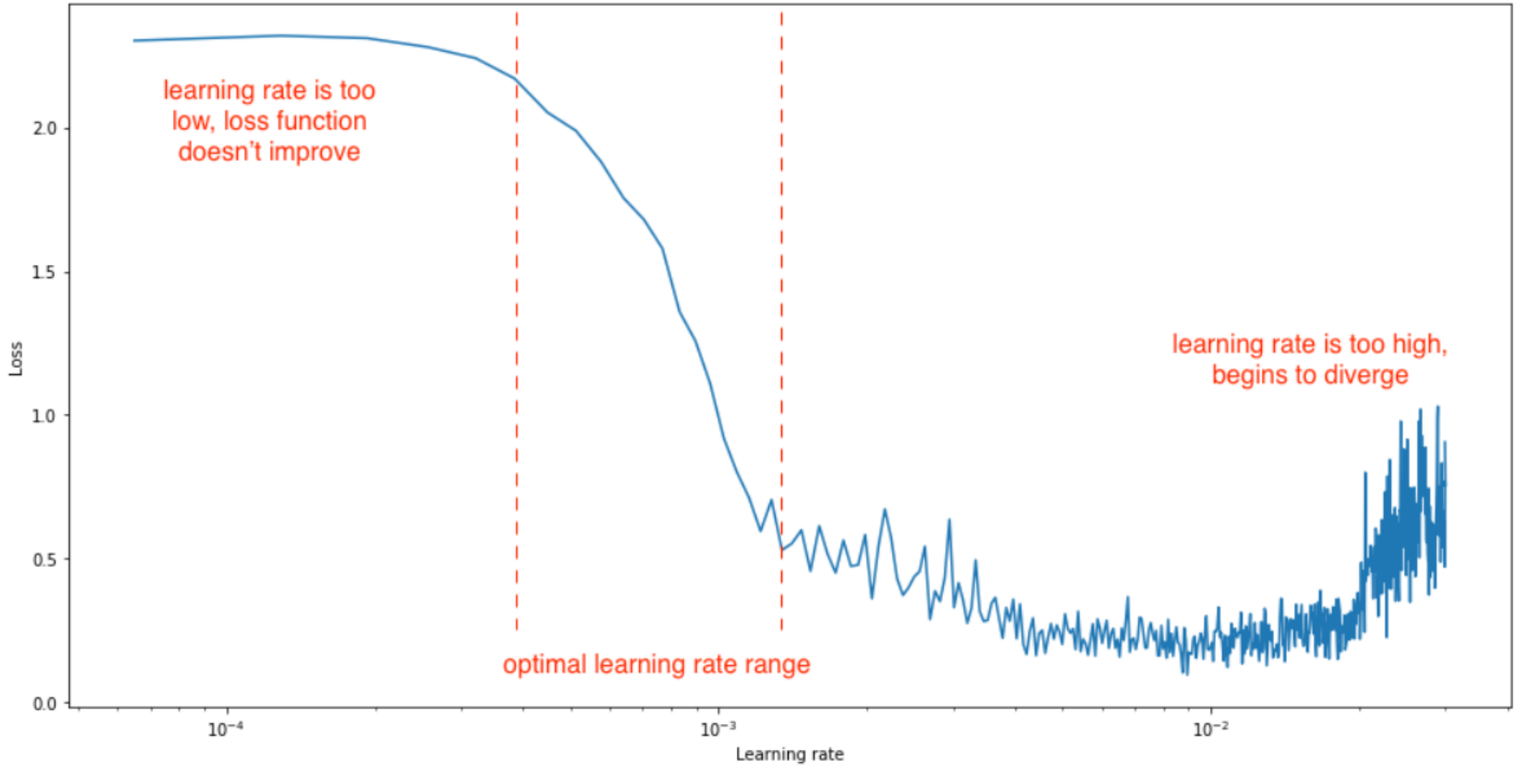
(i just wanted to make sure that people understand that this is a joke...)

12:21 PM - Nov 24, 2016

♡ 95 See Andrej Karpathy's other Tweets

Keras.callbacks:

- Teminate on NaN.
- Model Checkpoint
- EarlyStopping
- **LearningRateScheduler**
- TensorBoard
- CSVLogger



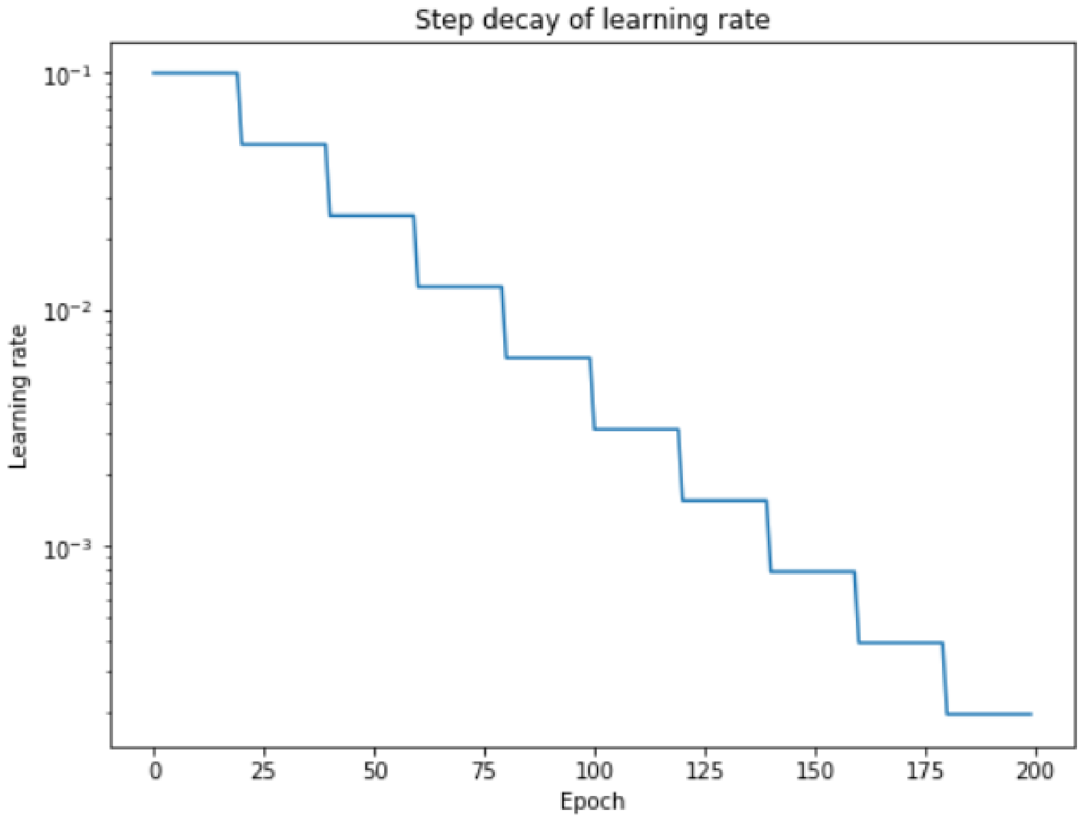
learning rate is too low, loss function doesn't improve

optimal learning rate range

learning rate is too high, begins to diverge

Keras.callbacks:

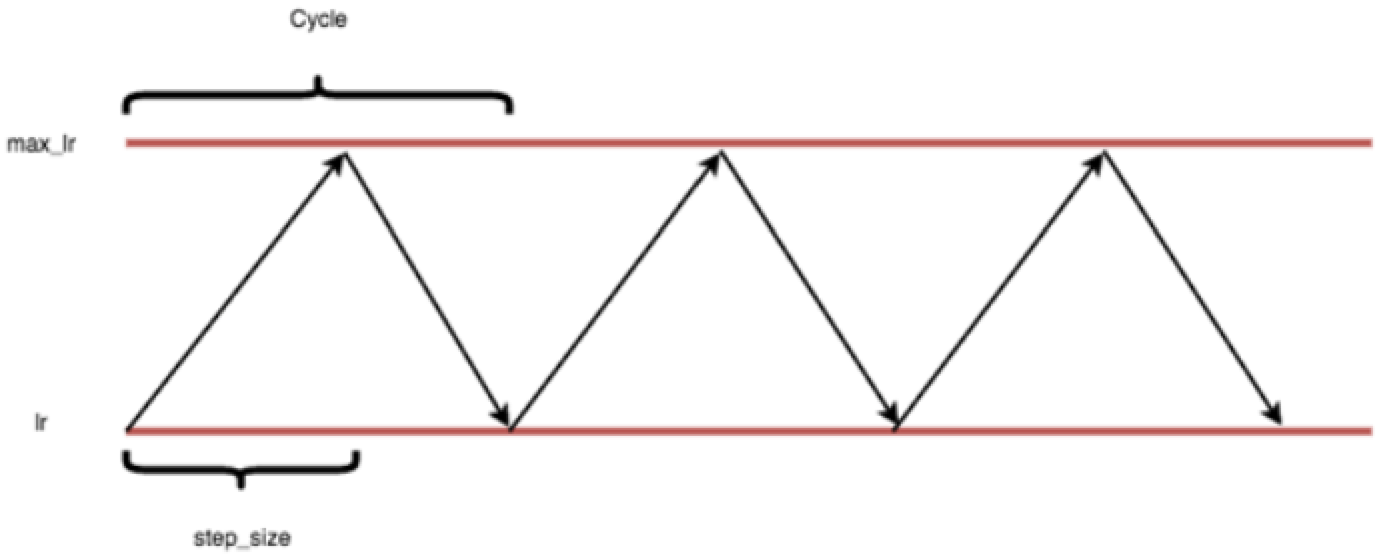
- Teminate on NaN.
- Model Checkpoint
- EarlyStopping
- **LearningRateScheduler**
- TensorBoard
- CSVLogger



Keras.callbacks:

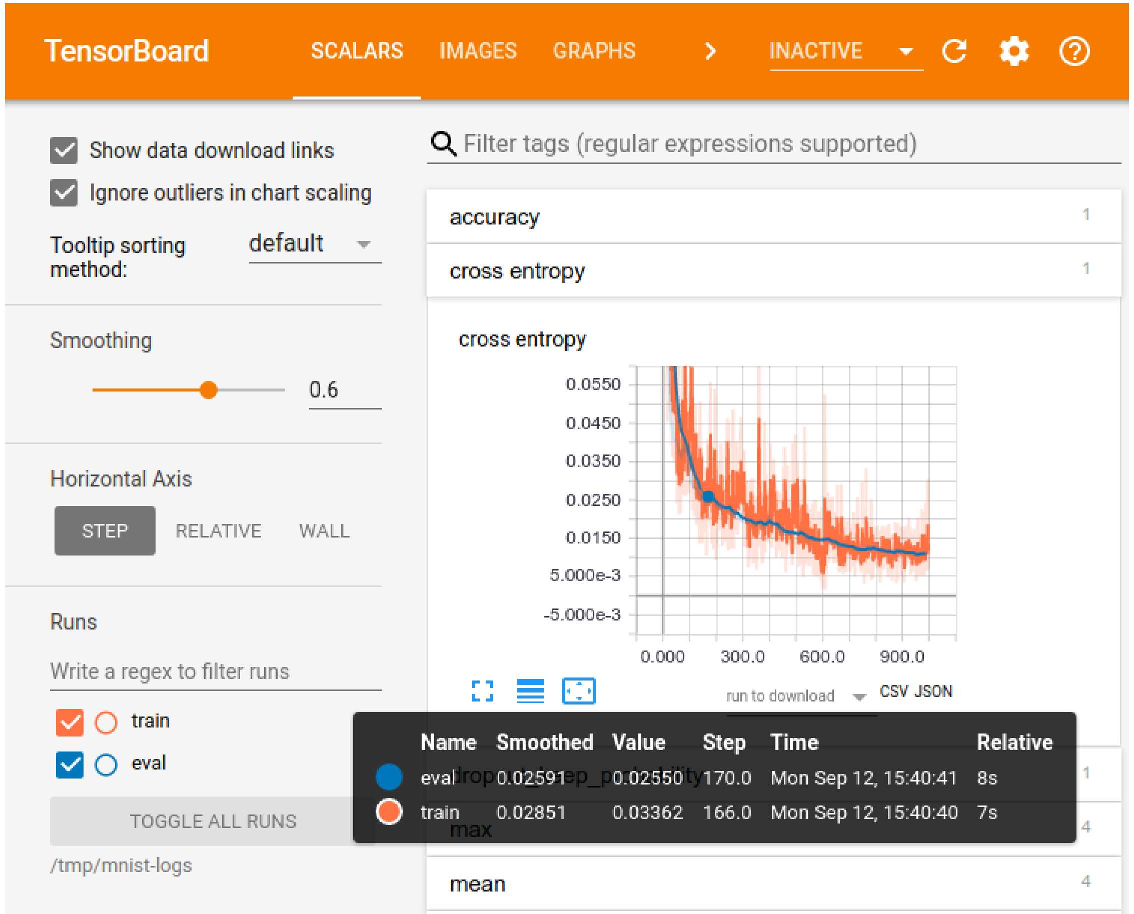
- Teminate on NaN.
- Model Checkpoint
- EarlyStopping
- **LearningRateScheduler**
- TensorBoard
- CSVLogger

Triangular schedule



Keras.callbacks:

- Teminate on NaN.
- Model Checkpoint
- EarlyStopping
- LearningRateScheduler
- **TensorBoard**
- CSVLogger



☑ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP

Runs

☑ ○ train

☑ ○ eval

/tmp/mnist-logs

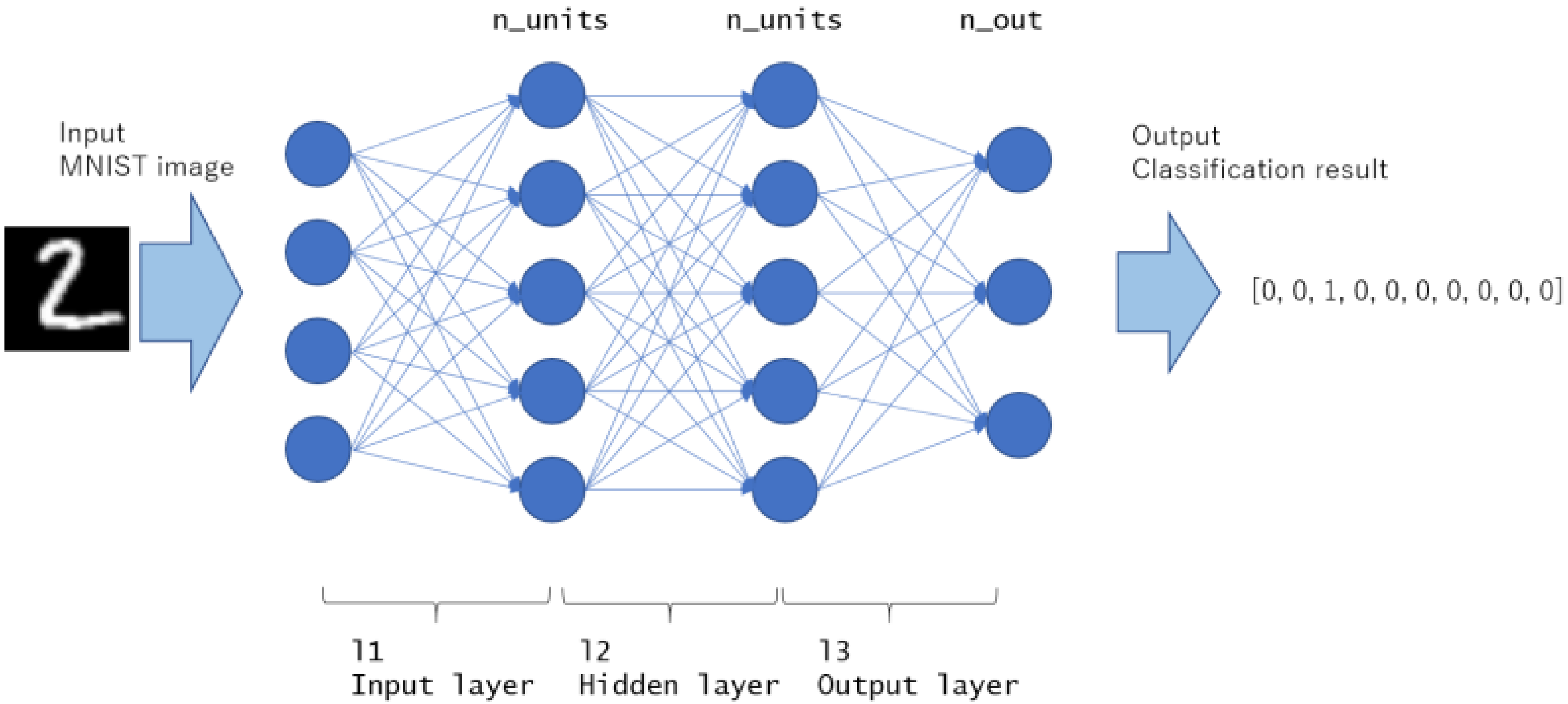
	Name	Smoothed	Value	Step	Time	Relative
●	eval	0.02591	0.02550	170.0	Mon Sep 12, 15:40:41	8s
●	train	0.02851	0.03362	166.0	Mon Sep 12, 15:40:40	7s

Keras.callbacks:

- Teminate on NaN.
- Model Checkpoint
- EarlyStopping
- LearningRateScheduler
- TensorBoard
- **CSVLogger**

Hands-On Session:

Training a fully-connected neural network on MNIST.



Thank You For Your Attention