서강대학교 SOGANG UNIVERSITY

Machine Learning with Keras

Systems Programming
Department of Computer Science and Engineering
Sogang University

Keras

https://keras.io/

1 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow.

- TensorFlow: An end-to-end open source machine learning platform, developed by **Google**.

It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

2 Keras guiding principles

- **User friendliness.** Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- **Modularity.** A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions and regularization schemes are all standalone modules that you can combine to create new models.
- **Easy extensibility.** New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Work with Python**. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

XOR with Keras

Use “03.Basic_Keras.ipynb”

1 Creating the model

Colaboratory has all the necessary modules installed with Keras. You just use it.

```
[ ]  import keras
```

```
Using TensorFlow backend.
```

Creating a model in Keras is very easy and simple.

```
[ ]  from keras.models import Sequential
```

```
[ ]  model = Sequential()
```

1 Creating the model

Add fully connected layer to model.

```
[ ]  model.add(Dense(2, activation='sigmoid', input_dim=2))
     model.add(Dense(1, activation='sigmoid'))
```

`model.add(layer)`

Add layers to the model.

`keras.layers.core.Dense(units, activation, ·····)`

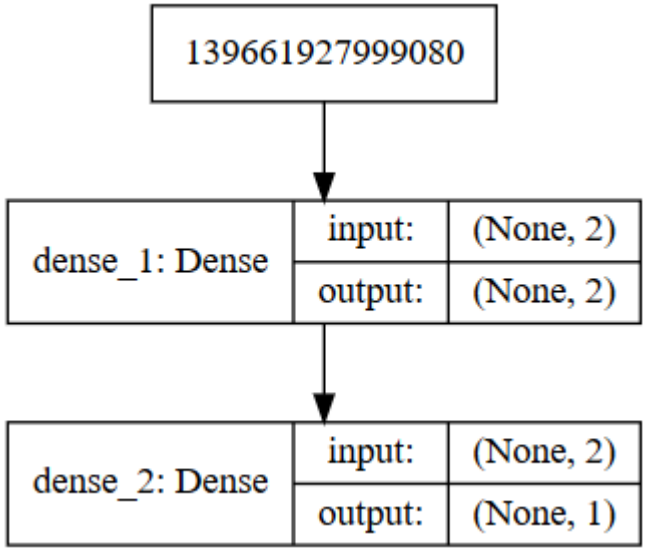
Fully-connected layer.
Arguments: units, activation and so on.

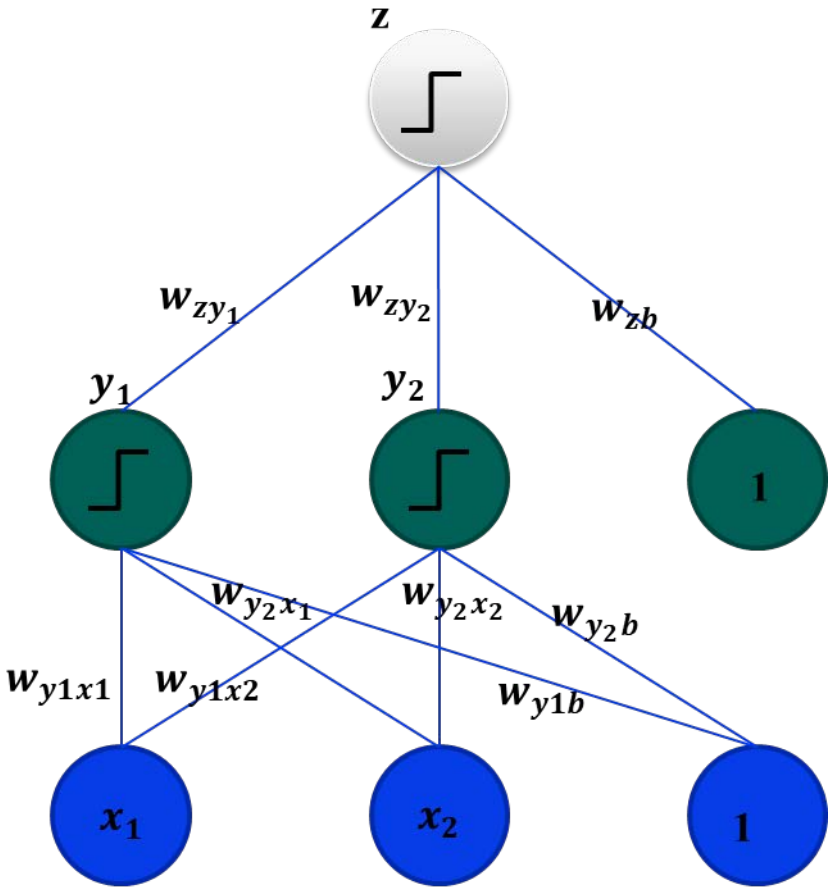
- units: dimensionality of the output space.
- activation: Activation function to use.

```
# as first layer in a sequential model:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
# now the model will take as input arrays of shape (*, 16)
# and output arrays of shape (*, 32)

# after the first layer, you don't need to specify
# the size of the input anymore:
model.add(Dense(32))
```

1 Creating the model





2 Compiling and training the model

서강대학교 SOGANG UNIVERSITY

You have to decide how to training. This is called an optimizer.

```
[ ]  optimizer = keras.optimizers.SGD(lr=learning_rate)
```

When everything is done, you compile the model.

```
[ ]  model.compile(loss=keras.losses.mean_squared_error,
                   optimizer=optimizer,
                   metrics=['accuracy'])
```

```
model.compile(optimizer, loss, metrics, ·····)
```

Configures the model for training.
Arguments: optimizer, loss, metrics and so on.

2 Compiling and training the model

서강대학교 SOGANG UNIVERSITY

Then you just start training by using fit.

```
[ ]  model.fit(x, y, epochs=epoch, verbose=0)
```

```
Epoch 0 : 0.26976197957992554
Epoch 500 : 0.2500118315219879
Epoch 1000 : 0.24801373481750488
Epoch 1500 : 0.2183685004711151
Epoch 2000 : 0.17950747907161713
Epoch 2500 : 0.08454935252666473
Epoch 3000 : 0.031153008341789246
Epoch 3500 : 0.01767469197511673
Epoch 4000 : 0.012043490074574947
Epoch 4500 : 0.009033693931996822
```

```
model.fit(x, y, batch_size, epochs, verbose, ·····)
```

Trains the model for a given number of epochs.
Arguments: x, y, batch_size, epochs, verbose, validation_data, shuffle and so on.

3 Evaluating and prediction the model

서강대학교 SOGANG UNIVERSITY

After training, you have to evaluate your model.

```
[ ]  score = model.evaluate(x, y, verbose=0)
     print('Loss:', score[0])
     print('Accuracy:', score[1])
```

```
Loss: 0.007184308022260666
Accuracy: 1.0
```

```
model.evaluate(x, y, batch_size, verbose, ·····)
```

Returns the loss value & metrics values for the model in test mode.
Arguments: x, y, batch_size, verbose and so on.

3 Evaluating and prediction the model

If you want to see the result of your model, use the predict.

```
[ ]  y_pred = model.predict(x)
```

```
Input          Label          Output
[[ 1  1]       [[0]           [[0.07717173]
 [ 1 -1]        [1]            [0.9072809 ]
 [-1 -1]        [0]            [0.07721792]
 [-1  1]]       [1]]           [0.909323  ]]
```

```
model.predict(x, batch_size, verbose, ·····)
```

Generates output predictions for the input samples.
Arguments: x, batch_size, verbose and so on.