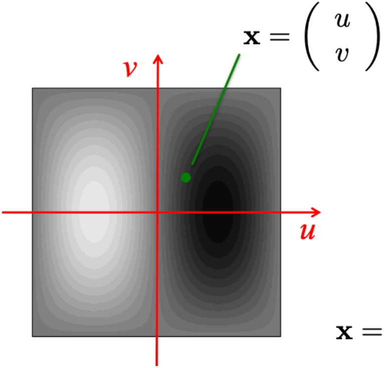
Introduction to Keras

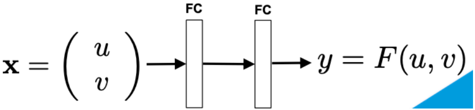
from V. Lepetit

Example 1: Two-layer Network

A Two-Layer Network

We will see how to train a two-layer network to approximate a 2D function $F(u, v)$:



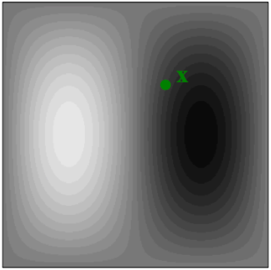


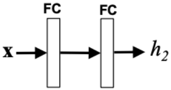
FC: Fully-Connected

Our Two-Layer Network

The input is a 2D point $\mathbf{x}$;

The output is a scalar value h_2





Hidden layer:

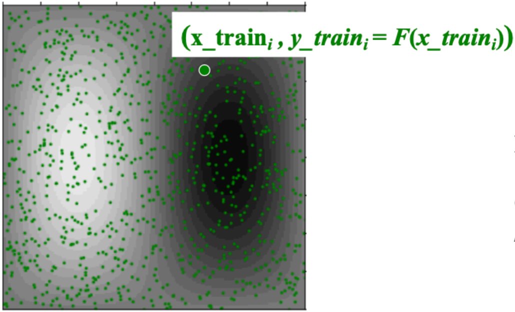
$\mathbf{h}_1 = \mathrm{ReLU}(\mathbf{W}_1\,\mathbf{x} + \mathbf{b}_1)$

Output layer:

$h_2 = \mathbf{W}_2\,\mathbf{h}_1 + b_2$

Loss Function

Training set:



Hidden layer:

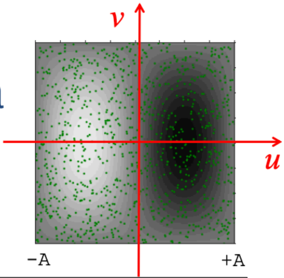
$$\mathbf{h}_1 = \mathrm{ReLU}(\mathbf{W}_1\, \mathbf{x} + \mathbf{b}_1)$$

Output layer:

$$h_2 = \mathbf{W}_2\, \mathbf{h}_1 + b_2$$

$$\text{Loss} = \frac{1}{N_s} \sum_{i=1}^{N_s} (h_2(\mathbf{x\_train}_i) - y\_train_i)^2$$

Generating Training Data




```
import numpy as np

def F(x1, x2):
    return np.sin(np.pi * x1 / 2.0) * np.cos(np.pi * x2 / 4.0)

A = 2
nb_samples = 1000
X_train = np.random.uniform(-A, +A, (nb_samples, 2))
Y_train = np.vectorize(F)(X_train[:,0], X_train[:,1])
```

Models

In Keras, a deep architecture is called a *model*.

A model can be an arbitrary graph of layers.

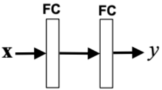
For this first example, we can use a `Sequential` model.

```
from keras.models import Sequential

model = Sequential()
```

Defining the Network

Hidden layer: $\mathbf{h}_1 = \mathrm{ReLU}(\mathbf{W}_1\, \mathbf{x} + \mathbf{b}_1)$

Output layer: $h_2 = \mathbf{W}_2\, \mathbf{h}_1 + b_2$




```
from keras.layers import Dense, Activation

nb_neurons = 20
model.add(Dense(nb_neurons, input_shape=(2,)))
model.add(Activation('relu'))
model.add(Dense(1))
```

Shortcut

```
from keras.models import Sequential
from keras.layers import Dense, Activation

nb_neurons = 20
model = Sequential([
  Dense(nb_neurons, input_shape=(2,)),
  Activation('relu'),
  Dense(1)])
```

Defining the Optimization Method

```
from keras.optimizers import SGD

sgd = SGD(lr=0.01,
          decay=1e-6, momentum=0.9,
          nesterov=True)

model.compile(loss='mean_squared_error',
              optimizer=sgd)
```

$$\text{Loss} = \frac{1}{N_s}\sum_{i=1}^{N_s}(h_2(\mathbf{x\_train}_i) - y\_train_i)^2$$

Task: regression; Loss: mean squared error

Running the Optimization

```
model.fit(X_train, Y_train, epochs=10, batch_size=32)
```

Output:

```
Epoch 1/10
1000/1000 [==============================] - 0s 490us/step - loss: 0.0487
Epoch 2/10
1000/1000 [==============================] - 0s 43us/step - loss: 0.0415
Epoch 3/10
1000/1000 [==============================] - 0s 49us/step - loss: 0.0345
Epoch 4/10
1000/1000 [==============================] - 0s 44us/step - loss: 0.0290
Epoch 5/10
1000/1000 [==============================] - 0s 52us/step - loss: 0.0235
Epoch 6/10
1000/1000 [==============================] - 0s 43us/step - loss: 0.0190
Epoch 7/10
1000/1000 [==============================] - 0s 45us/step - loss: 0.0154
Epoch 8/10
1000/1000 [==============================] - 0s 47us/step - loss: 0.0122
Epoch 9/10
1000/1000 [==============================] - 0s 50us/step - loss: 0.0098
Epoch 10/10
1000/1000 [==============================] - 0s 48us/step - loss: 0.0082
```

Prediction

```
x = [1.5, 0.5]
print(F(x[0], x[1]))

x = np.array(x).reshape(1, 2)
print(x)
print( model.predict(x) )
print( model.predict(x)[0][0] )
```

Output:

```
0.6532814824381883
[[1.5, 0.5]]
[[0.5451795]]
0.5451795
```

Visualization

```
Width = 200
Height = 200
U = np.linspace(-A, +A, Width)
V = np.linspace(-A, +A, Height)
# Computes cartesian product between U and V:
UV = np.transpose([np.tile(U, len(V)), np.repeat(V, len(U))])
print(UV)
ys = model.predict(UV)
print(ys)
I = ys.reshape(Width, Height)
```

Output:

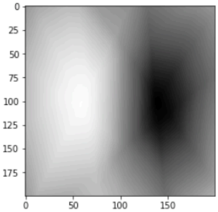
```
[[-2.          -2.          ]
 [-1.9798995   -2.          ]
 ...
 [ 1.95979899   2.          ]
 [ 2.           2.          ]]

[[ 0.02076489]
 [-0.00082633]
 ...
 [-0.11296707]
 [-0.12041384]]
```

Visualization (2)

```
I = ys.reshape(Width, Height)

import matplotlib.pyplot as plt
import matplotlib.cm as cm

#Makes imageplotlib show the images inline
#in Jupyter notebooks:
%matplotlib inline
plt.imshow(I, cmap = cm.Greys)
```

Output:

```
<matplotlib.image.AxesImage at 0x117590390>
```



Example 2: CNN for MNIST

MNIST

Loading the Dataset

```
from keras.datasets import mnist

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

print( X_train.shape )
```

Output:

```
(60000, 28, 28)
```

Visualizing one Sample

```python
print(y_train.shape)
print(y_train[0:3])

from keras.utils import np_utils
y_train = np_utils.to_categorical(y_train, 10)
print(y_train[0])
```

Output:

```
(60000,)
[5 0 4]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Reformating the Input

```python
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
          input_shape=(28, 28, 1)))
print(model.output_shape)
```

Output:

```
(None, 26, 26, 32)
```

Reformating the Desired Output

```
print(y_train.shape)
print(y_train[0:3])

from keras.utils import np_utils
y_train = np_utils.to_categorical(y_train, 10)
print(y_train[0])
```

Output:

```
(60000,)
[5 0 4]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

One-hot vector

Creating the Model

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu',
          input_shape=(28, 28, 1)))
print(model.output_shape)
```

Output:

```
(None, 26, 26, 32)
```

Question: How many parameters?

Creating the Model (3)

```
from keras.layers import Flatten
model.add(Flatten())
print(model.output_shape)

from keras.layers import Dense
model.add(Dense(128, activation='relu'))
print(model.output_shape)

model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
print(model.output_shape)
```

Output:

```
(None, 5408)
(None, 128)
(None, 10)
```

“Flatten” before “Dense” (Fully-Connected)

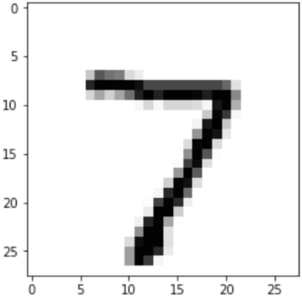
Optimization

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.fit(X_train, Y_train,
          batch_size=32, epochs=10, verbose=1)
```

Testing

```
plt.imshow(X_test[0].reshape(28,28), cmap = cm.Greys)

print(model.predict(X_test[0].reshape(1, 28, 28, 1)))
```

Output:




```
[[2.7737193e-10 2.5943342e-08 2.5428611e-07 3.6613658e-06 1.0967714e-10
  7.6563078e-10 1.0837641e-14 9.9999535e-01 2.9037880e-08 7.2273968e-07]]
```

Example 3: Using Pre-trained Network (VGG)

Using VGG to Recognize Object in Images

```
from keras.applications.vgg16 import VGG16
model = VGG16()
from keras.preprocessing.image import load_img
image = load_img('cat.jpg', target_size=(224, 224))
y_pred = model.predict(image)
from keras.applications.vgg16 import decode_predictions
labels_pred = decode_predictions(y_pred)
print(labels_pred)
```