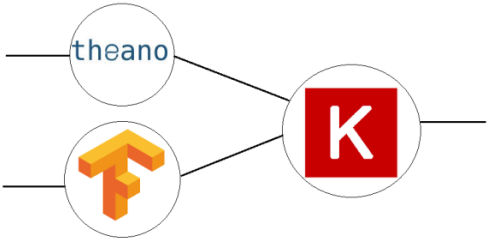
Neural Networks and Deep Learning: Deep Learning Resources: Keras

Nicolas Thome

Conservatoire National des Arts et Métiers (Cnam)
Département Informatique

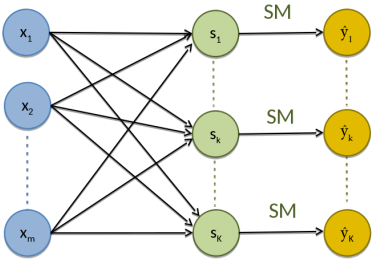
Keras



- Keras: Python wrapper on top of Tensorflow / Theano (F. Chollet)
- Install (with pip): `pip install keras`
- `https://keras.io/`

Keras: Logistic Regression for Classification

- Simple example: Logistic Regression (LR)
- $\mathbf{x}_i$ vector, $\mathbf{s}_i = \mathbf{x}_i\mathbf{W} + \mathbf{b}$
- Soft-Max (SM): $\hat{\mathbf{y}_k} \sim P(k/\mathbf{x_i}, \mathbf{W}, \mathbf{b}) = \frac{e^{\mathbf{s}_k}}{\sum_{k'=1}^{K} e^{\mathbf{s}_{k'}}}$
- Training with cross-entropy: $\frac{1}{N}\sum_{i=1}^{N} \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$
- Example in MNSIT database: $K = 10$ classes
- Input: flattened images: $d = 28^2 = 784$
- # parameters: $784 * 10 + 10 = 7850$



Keras: Logistic Regression

- Load MNIST data:

```
from keras.datasets import mnist
# MNIST data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# Some pre-processing
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
```

Keras: Logistic Regression

- Keras: use class `Sequential` for (chain) feedfoward network
- Define an (empty) feedfoward network

```
from keras.models import Sequential
model = Sequential()
```

1

- Add fully connected layer (size 10) + softmax activation

```
from keras.layers import Dense, Activation
model.add(Dense(10, input_dim=784, name='fc1'))
model.add(Activation('softmax'))
```

2

Keras: Logistic Regression

- Visualize (text) architecture:

```
model.summary()
```

1

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 10)                7850
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 7,850.0
Trainable params: 7,850.0
Non-trainable params: 0.0
```

Keras: Logistic Regression

- Compile model with cross-entropy loss and `sgd` optimizer

```
from keras.optimizers import SGD
learning_rate = 0.5
sgd = SGD(learning_rate)
model.compile(loss='categorical_crossentropy',optimizer=sgd,metrics=['accuracy'])
```

- Optimize model parameters to fit training data (*e.g.* MNIST)

```
# Fit model to data
model.fit(X_train, y_train,batch_size=128, epochs=20,verbose=1)
```

Keras: Logistic Regression

- Evaluate performances on test set:

```
scores = model.evaluate(X_test, Y_test, verbose=0)
print("%s TEST: %.2f%%" % (model.metrics_names[0], scores[0]*100))
print("%s TEST: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

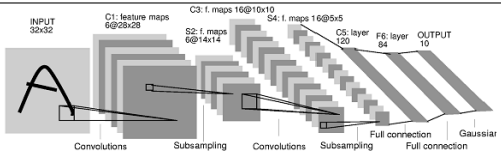
2

```
_________________________________________________________________
Epoch 1/10
60000/60000 [==============================] - 1s - loss: 0.4778 - acc: 0.8692
Epoch 2/10
60000/60000 [==============================] - 1s - loss: 0.3357 - acc: 0.9062
Epoch 3/10
60000/60000 [==============================] - 1s - loss: 0.3132 - acc: 0.9117
Epoch 4/10
60000/60000 [==============================] - 1s - loss: 0.3016 - acc: 0.9150
Epoch 5/10
60000/60000 [==============================] - 1s - loss: 0.2941 - acc: 0.9180
Epoch 6/10
60000/60000 [==============================] - 1s - loss: 0.2883 - acc: 0.9196
Epoch 7/10
60000/60000 [==============================] - 1s - loss: 0.2840 - acc: 0.9205
Epoch 8/10
60000/60000 [==============================] - 1s - loss: 0.2804 - acc: 0.9215
Epoch 9/10
60000/60000 [==============================] - 1s - loss: 0.2775 - acc: 0.9230
Epoch 10/10
```

Keras: more complex models

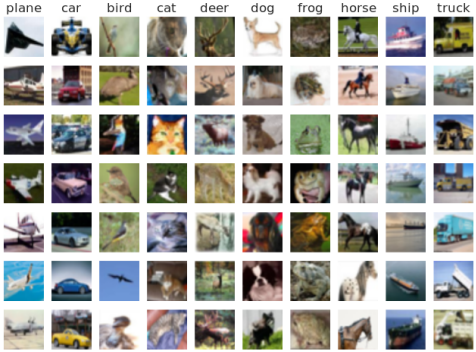
- Design more complex model by adding layers:
  - Fully connected, convolution, non-linearity, pooling, *etc*
  - Class for convolution on images: `Conv2D`
- Code for training remains unchanged (back-prop does the job)

```
s=(5,5)
ish=(28,28,1)
model = Sequential()
model.add(Conv2D(32,kernel_size=s,activation='sigmoid',input_shape=ish,padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (5, 5), activation='sigmoid', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='sigmoid'))
model.add(Dense(nb_classes, activation='softmax'))
```



Keras: more complex models

- Many other layers implemented, *e.g.* locally-connected, dropout, normalization
- Various optimizer implemented, *e.g.* Nesterov, Adam
- Various loss fonctions, *e.g.* hinge loss, regression, kullback leibler divergence
- Various datasets, *e.g.* CIFAR



Keras: GPU

- Automatic detection of existing resources and selection
- Several GPU cards ⇒ use them all
- Ex for selecting a specific card:
  `CUDA_VISIBLE_DEVICES=0 python my_deep_script.py`

Keras: Conclusion

- Intuitive librairy: work with layers
  - Does not expose computation graph
- Very quickly train a model and evaluate performances
- Use all tensorflow / theano backend features (*e.g.* tensorboard)
- Try it: `https://keras.io/`
- **Deep Features and transfert**
  ⇒ following!