Keras: Handwritten Digit Recognition using MNIST Dataset

IIT PATNA

February 9, 2017

OUTLINE

Keras

Keras is

- high-level neural networks library
- written in Python
- capable of running on top of either TensorFlow (open source software library for numerical computation) or Theano (numerical computation library for Python)
- developed with a focus on enabling fast experimentation

Guiding principles

- Modularity
  - neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models
- Minimalism
  - Each module should be kept short and simple.
- Easy extensibility
  - New modules are simple to add (as new classes and functions)
  - easily create new modules allows, for total expressiveness, making Keras suitable for advanced research.
- Work with Python
  - Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

Dependencies

- Python 2.7+
- numpy: fundamental package for scientific computing with Python
- scipy: library used for scientific computing and technical computing
- Matplotlib (Optional, recommended for exploratory analysis)
- HDF5 and h5py (Optional, required if you use model saving/loading functions)
- Theano

Installation

How to install Keras?

- Follow instruction provided in "keras installation" file

Keras support

- Model
  - core data structure of Keras
  - a way to organize layers
  - sequence or graph of standalone modules (neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes)
- Two types:
  - Sequential
  - Model class used with functional API
- Sequential Model: a linear stack of layers.

Creating Sequential Model

Sequential Layers in Keras

- Dense: fully connected NN layer
- Activation: Applies an activation function
- Dropout: Applies Dropout to the input. Dropout consists in randomly setting a fraction p of input units to 0 at each update during training time, which helps prevent overfitting
- Convolutional Layers
- Pooling Layers
- ... and many more.

Creating Sequential Model

Activation Functions in Keras

- Activations can either be used through an Activation layer, or through the activation argument supported by all forward layers
- model.add(Dense(64))
  model.add(Activation('tanh'))
  is equivalent to:
  model.add(Dense(64, activation='tanh'))
- Available activations
  - softmax: usually used on the output layer to turn the outputs into probability-like values
  - relu: rectified linear unit (ReLU), most popular activation function, $f(x) = max(x, 0)$
  - tanh: hyperbolic tangent
  - linear
  - and many more.

Creating Sequential Model

Creating Sequential Model

- use constructor:
  model = Sequential([Dense(32, input_dim=784), Activation('relu'),
  Dense(10), Activation('softmax'),]),
  or
- add layers via the .add() method:
  model = Sequential()
  model.add(Dense(32, input_dim=784)) model.add(Activation('relu'))

Creating Sequential Model

- The model needs to know what input shape it should expect
- first layer in a Sequential model (and only the first, because following layers can do automatic shape inference) needs to receive information about its input shape
- Dense is regular fully connected NN layer
- Dense(32, input_dim=784) specifies that it is
  - first (input) layer
  - output dimension is 32 (1^{st} argument
  - input dimension is 784
  - If no activation function specified, no activation is applied (ie. "linear" activation: a(x) = x).
- Dense(10), Activation('softmax') specifies that
  - fully connected
  - not first layer (no need to specify inpute shape)
  - 10 is output shape
  - softmax is activation function

Step 1: Import libraries and Initialize seed value

- import libraries
- initilize seed

Step 2: Loading MNIST data

- database of handwritten digits
- training set of 60,000 examples, and a test set of 10,000 examples
- Keras library provide function to load data set
- images are 28 pixels x 28 pixels each
- plot samples in matplotlib

Step 3: Preprocess input data for Keras

- dataset is a 3-dimensional array of instance, image width and image height
- For a multi-layer perceptron model we must reduce the images down into a vector of pixels
- In this case the 28x28 sized images will be 784 pixel input values
- using the reshape() function
- We can also reduce our memory requirements by forcing the precision of the pixel values to be 32 bit

Step 4: Preprocess class labels for Keras

- shape of our class label data: 10 different classes, one for each digit, but only have a 1-dimensional array.
- y_train and y_test data are not split into 10 distinct class labels, but rather are represented as a single array with the class values

Step 5: Define model architecture

- model is a simple neural network
- one hidden layer with the same number of neurons as there are inputs (784)
- init: name of initialization function for the weights of the layer. normal for values randomly drawn from normal distribution.
- there are many other initializations available in Keras
- rectifier activation function is used for the neurons in the hidden layer
- softmax activation function is used on the output layer to turn the outputs into probability-like values and allow one class of the 10 to be selected as the models output prediction

Step 6: Compile model

- Before training, configure the learning process, using compile() method. Three argements:
  - optimizer: ANN training process is an optimization task with the aim of finding a set of weights to minimize some objective function
  - Many optimizers are available in Keras
  - loss function: the objective function that model try to minimize
  - Many loss functions are available in Keras
  - list of metrics: used to judge performance of model, similar to objective function however not used for training purpose
- Logarithmic loss is used as the loss function
- ADAM gradient descent algorithm is used to learn the weights

Step 7: Train model

using fit() function

Step 8: Evaluate model on test data

using evaluate() function

Problem with MLP

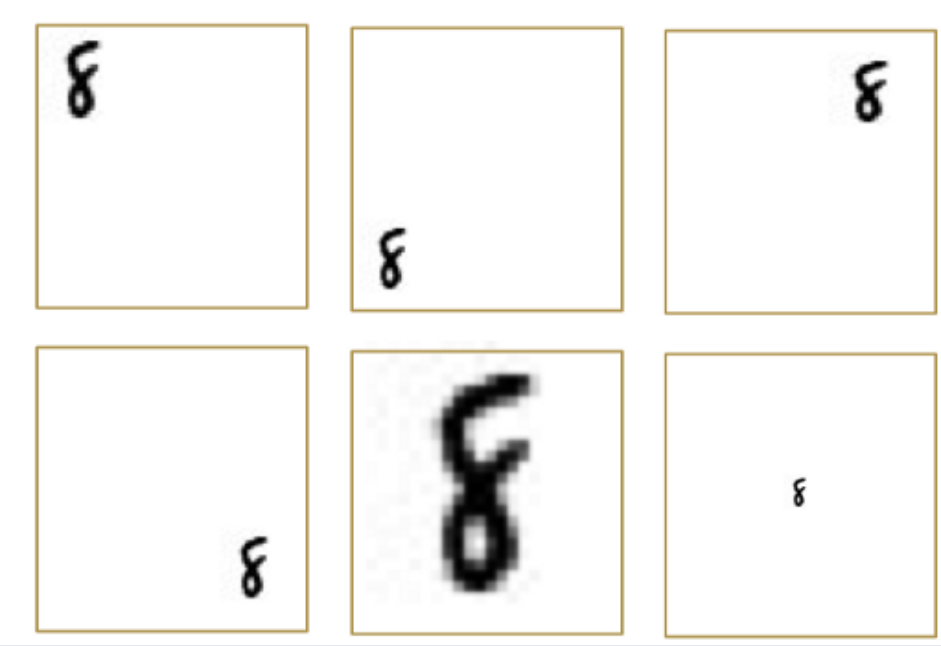

Figure : All are different and must be trained

- overfitting

Constructing the Right Network

which steps to follow to make an efficient image classifier?

Loading Your own Data

How to load your own training and test data set in Keras?

Important Links

Links

1. Keras Official Documentation Page
2. Github Page
3. Another Github Page

The End