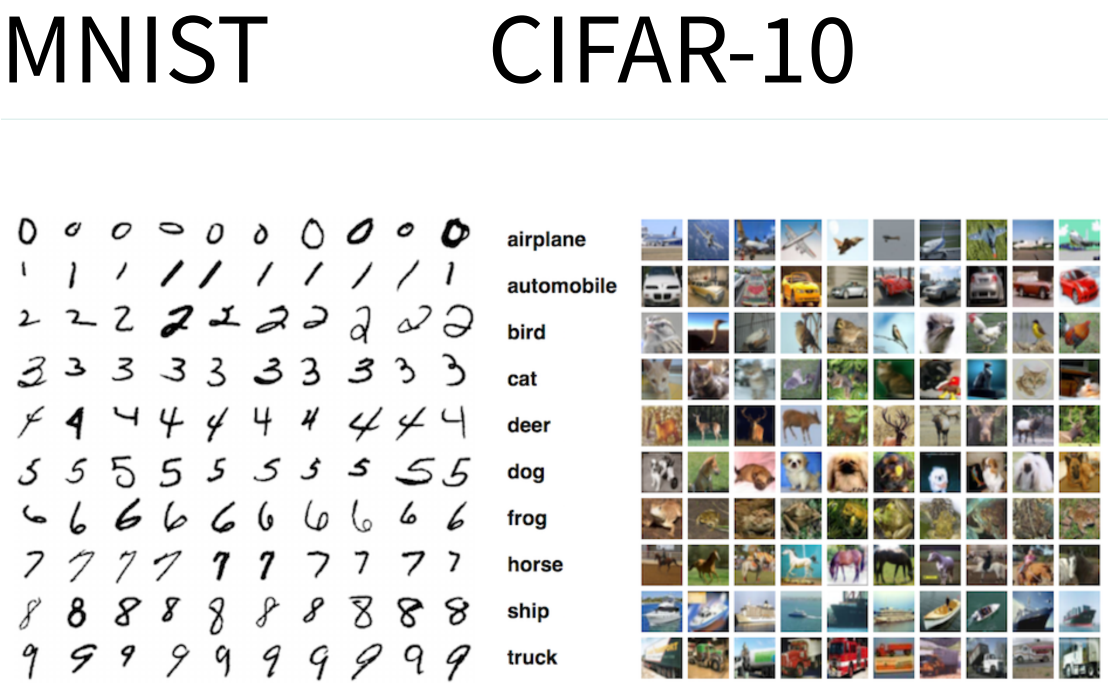
Deep learning for complete beginners: neural network fine-tuning techniques

[Edited on 20 March 2017, to account for API changes introduced by the release of Keras 2]

Introduction

Welcome to the third (and final) in a series of blog posts that is designed to get you quickly up to speed with *deep learning*; from first principles, all the way to discussions of some of the intricate details, with the purposes of achieving respectable performance on two established machine learning benchmarks: MNIST (http://yann.lecun.com/exdb/mnist (classification of handwritten digits) and CIFAR-10 (https://www.cs.toronto.edu/~kriz/c (classification of small images across 10 distinct classes: plane, car, bird, cat, deer, dog, frog, horse, ship & truck).



Last time around (https://cambridgespark.com/conte neural-networks-with-keras/index.html), I have introduced the *convolutional neural network* model, and illustrated how, combined with a simple but effective regularisation method of *dropout*, it may quickly achieve an accuracy level of 78.6% on CIFAR-10, leveraging the Keras (https://keras.io) deep learning framework.

By now, you have acquired the fundamental skills necessary to apply deep learning to most problems of interest (a notable exception, outside of the scope of these tutorials, is the problem of processing *time-series of arbitrary length*, for which a *recurrent neural network* (RNN) model is often preferable). In this tutorial, I will wrap up with an important but often overlooked aspect of tutorials such as this one – the tips and tricks for properly *fine-tuning* a model, to make it generalise better than the initial baseline you started out with.

This tutorial will, for the most part, assume familiarity with the previous two in the series.

Hyperparameter tuning and the baseline model

Typically, the design process for neural networks starts off by designing a simple network, either directly applying architectures that have shown successes (http://rodrigob.github.io/are_we_t for similar problems, or trying out hyperparameter values that generally seem effective. Eventually, we will hopefully attain performance values that seem like a nice baseline starting point, after which we may look into modifying every fixed detail in order to extract the maximal performance capacity out of the network. This is commonly known as *hyperparameter tuning*, because it involves modifying the components of the network which need to be specified before training.

While the methods described here can yield far more tangible improvements on CIFAR-10, due to the relative difficulty of rapid prototyping on it without a GPU, we will focus specifically on improving performance on the MNIST benchmark. Of course, I do invite you to have a go at applying methods like these to CIFAR-10 and see the kinds of gains you may achieve compared to the basic CNN approach, should your resources allow for it.

We will start off with the baseline CNN given below. If you find any aspects of this code unclear, I invite you to familiarise yourself with the previous two tutorials in the series – all the relevant concepts have already been introduced there.

```
from keras.datasets import mnist # subroutines for fetching the MNIST dataset
from keras.models import Model # basic class for specifying and training a neural network
from keras.layers import Input, Dense, Flatten, Convolution2D, MaxPooling2D, Dropout
from keras.utils import np_utils # utilities for one-hot encoding of ground
```

```
truth values

batch_size = 128 # in each iteration, we consider 128 training examples at once
num_epochs = 12 # we iterate twelve times over the entire training set
kernel_size = 3 # we will use 3x3 kernels throughout
pool_size = 2 # we will use 2x2 pooling throughout
conv_depth = 32 # use 32 kernels in both convolutional layers
drop_prob_1 = 0.25 #
```

```
dropout after pooling with probability 0.25
drop_prob_2 = 0.5 # dropout in the FC layer with probability 0.5
hidden_size = 128 # there will be 128 neurons in both hidden layers

num_train = 60000 # there are 60000 training examples in MNIST
num_test = 10000 # there are 10000 test examples in MNIST
```

```
height, width, depth = 28,
28, 1 # MNIST images are
28x28 and greyscale
num_classes = 10 # there
are 10 classes (1 per
digit)

(X_train,         y_train),
(X_test,     y_test)      =
mnist.load_data() # fetch
MNIST data

X_train                   =
X_train.reshape(X_train.sha
 height, width, depth)
X_test                    =
X_test.reshape(X_test.shape
 height, width, depth)
X_train                    =
X_train.astype('float32')
X_test                     =
X_test.astype('float32')
X_train /= 255 # Normalise
data to [0, 1] range
X_test /= 255 # Normalise
data to [0, 1] range

Y_train                    =
np_utils.to_categorical(y_
  num_classes)  #  One-hot
encode the labels
Y_test                     =
np_utils.to_categorical(y_
  num_classes) # One-hot
encode the labels

inp = Input(shape=(height,
width, depth)) # N.B.
TensorFlow back-end
expects channel dimension
last
# Conv [32] -> Conv [32] -
> Pool (with dropout on
the pooling layer)
conv_1 =
Convolution2D(conv_depth,
(kernel_size,
kernel_size),
```