CNN Training Parameters

Save & Load Model

```
from keras.models import Sequential, load_model
model.save('my_first_model.h5')
del model #delete the existing model
model = load_model('my_first_model.h5')
```

Model.fit

```
model.fit(X, y, nb_epoch=10, batch_size=64, class_weight=None, verbose=1)
      callbacks = [EarlyStopping(monitor='val_loss', patience=3, verbose=0),]
model.fit(X, y, batch_size=32, nb_epoch=10,
                shuffle=True, verbose=2, validation_data=(X_valid, Y_valid),
                callbacks=callbacks)
```

●返回：记录字典，包括每一次迭代的训练误差率和验证误差率
● X：训练图像
● y：标签

● nb_epoch：在一个 epoch 中，所有训练集数据使用一次
one epoch = one forward pass and one backward pass of all the training examples

● batch_size：每次训练和梯度更新块的大小。
一般情况下，一个训练集中会有大量的 samples，为了提高训练速度，会将整个 training set 分为 n_batch 组，每组包含 batch_size 个 samples
整个数据集 samples 个数 = batch_size * n_batch

● iterations：每次 iteration 进行的工作为：利用某个 batch 的 samples 对 model 进行训练 number of iterations = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes)

```
iterations = 0
for i in range(n_epochs ):
    for j in range (n_batch):
       #利用第j组batch进行training
       train (j)
       # iterations个数加1
       iterations = iterations  +1
```

可见：iterations = epoch * n_batch
即，每个epoch进行n_batch次training，每次training，利用batch_size个samples

● alpha = 1　学习率

● verbose：进度表示方式。0 表示不显示数据，1 表示显示进度条，2 表示用只显示一个数据。

● callbacks：回调函数列表。就是函数执行完后自动调用的函数列表。

● validation_split：验证数据的使用比例。

● validation_data：被用来作为验证数据的(X, y)元组。会代替 validation_split 所划分的验证数据。

● shuffle：类型为 boolean 或 str('batch')。是否对每一次迭代的样本进行 shuffle 操作（可以参见博文 Theano 学习笔记 01--Dimshuffle()函数）。'batch'是一个用于处理 HDF5（keras 用于存储权值的数据格式）数据的特殊选项。

● show_accuracy:每次迭代是否显示分类准确度。

● class_weigh：分类权值键值对。原文：dictionary mapping classes to a weight value, used for scaling the lossfunction (during training only)。键为类别，值为该类别对应的权重。只在训练过程中衡量损失函数用。

● sample_weight : list or numpy array with1:1 mapping to the training samples, used for scaling the loss function (duringtraining only). For time-distributed data, there is one weight per sample pertimestep, i.e. if your output data is shaped(nb_samples, timesteps, output_dim), your mask should be of shape (nb_samples, timesteps, 1). This allows you to maskout or reweight individual output timesteps, which is useful in sequence tosequence learning.

Model.compile

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

● optimizer：指定模型训练的优化器；
● loss：目标函数
● class_mode: "categorical"和"binary"中的一个，只是用来计算分类的精确度或 using the predict_classes method
● theano_mode: Atheano.compile.mode.Mode instance controllingspecifying compilation options

激活函数

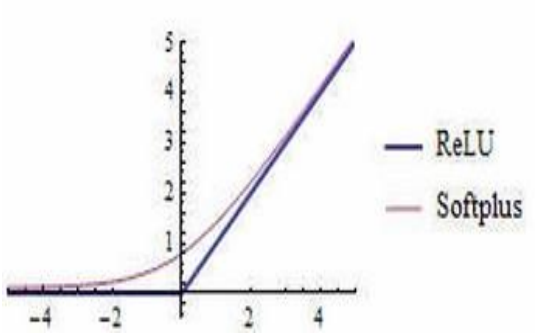
深度学习中的激活函数有：sigmoid，tanh，ReLUs，Softplus。目前最好的是 rectified linear units （ReLUs）修正线性单元。

多层的神经网络如果用 sigmoid 或 tanh 激活函数也不做 pre-training 的话会因为 gradient vanishing problem 而会无法收敛。使用 ReLU 则这没有这个问题。

预训练的用处：规则化，防止过拟合；压缩数据，去除冗余；强化特征，减小误差；加快收敛速度。

标准的 sigmoid 输出不具备稀疏性，需要用一些惩罚因子来训练出一大堆接近 0 的冗余数据来，从而产生稀疏数据，例如 L1、L1/L2 或 Student-t 作惩罚因子。因此需要进行无监督的预训练。

而 ReLU 是线性修正，公式为：g(x) = max(0, x)，函数图如下。它的作用是如果计算出的值小于 0，就让它等于 0，否则保持原来的值不变。这是一种简单粗暴地强制某些数据为 0 的方法，然而经实践证明，训练后的网络完全具备适度的稀疏性。而且训练后的可视化效果和传统方式预训练出的效果很相似，这也说明了 ReLU 具备引导适度稀疏的能力。



Dropout

Dropout： 防止过拟合
做法：在训练时，FP 中随机将 Hidden layer 中的节点输出值按照比例随机设置为 0，同时 BP 过程中相对应的被置为 0 的 Hidden layer 节点的误差也为 0，稀疏化。这样会使得神经元不得不去学习一些更加具备鲁棒性以及更加抽象的 features.

Softmax

对于多分类问题，我们可以使用多项 Logistic 回归，该方法也被称之为 softmax 函数。

softmax 分类器是在 logistic 回归模型在多分类问题上的推广。在多分类问题中，分类标签 y 可以取两个以上的值。softmax 分类器对于诸如 MNIST 手写数字分类等问题上有很好的效果。

Model Frame

```
def create_model():
    model = Sequential()
    #--------------------------------------------------------------------------
    model.add(ZeroPadding2D((1, 1), input_shape=(3, 32, 32), dim_ordering='th'))
    model.add(Convolution2D(4, 3, 3, activation='relu', dim_ordering='th'))
    # kernel 4, kernel size 3
    # 用 4 个滤波器扫描同一张图片，每个滤波器会总结出一个 feature。每个滤波器会生成一整张图片，有 4 个滤波器就会生成 4 张代表不同特征的图片
    # border_mode='same'(这里没有添加)
    # border_mode 代表这个滤波器在过滤时候用什么方式，这里用 same。
    # dim_ordering='th',    if use tensorflow, to set the input dimension order to theano ("th") style, but you can change it.
    # input_shape=(1,         # channels
                   28, 28,)    # height & width
    # (这里没有添加)
    # 因为是第一层，所以需要定义输入数据的维度，1, 28, 28 就是图片图片的维度。
    # 滤波器完成之后，会生成 4 层的数据，但是图片的长和宽是不变的，仍然是 28×28。
    # 激活函数 activation='relu'

    model.add(ZeroPadding2D((1, 1), dim_ordering='th'))
    model.add(Convolution2D(4, 3, 3, activation='relu', dim_ordering='th'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering='th'))
    # 向下取样
    # pool_size 是向下取样的时候，考虑多长多宽的图片。
    # strides 步长，是取完一个样之后要跳几步再取样，再跳几步再取样。
    # border_mode='same',    # Padding method   (这里没有添加)
    #--------------------------------------------------------------------------

    model.add(ZeroPadding2D((1, 1), dim_ordering='th'))
    model.add(Convolution2D(8, 3, 3, activation='relu', dim_ordering='th'))
    model.add(ZeroPadding2D((1, 1), dim_ordering='th'))
```

```
    model.add(Convolution2D(8, 3, 3, activation='relu', dim_ordering='th'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering='th'))
    #--------------------------------------------------------------------------
    model.add(ZeroPadding2D((1, 1), dim_ordering='th'))
    model.add(Convolution2D(16, 3, 3, activation='relu', dim_ordering='th'))
    model.add(ZeroPadding2D((1, 1), dim_ordering='th'))
    model.add(Convolution2D(16, 3, 3, activation='relu', dim_ordering='th'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering='th'))
    model.add(Dropout(0.2))
    #--------------------------------------------------------------------------
    model.add(Flatten())  # 用 Flatten 把卷出来的三维的层，抹平成二维的。
    model.add(Dense(32, activation='relu'))
    # 接下来就加一个 Dense 全联接层，抹平就是为了可以把这一个一个点全连接成一个层.
    model.add(Dropout(0.5))

    model.add(Dense(32, activation='relu'))
    model.add(Dropout(2.5)) #changed this from 0.5

    model.add(Dense(8, activation='softmax'))
    # 第三个全连接层，输出 8 个 unit，用 softmax 作为分类
    #--------------------------------------------------------------------------
    sgd = SGD(lr=1e-2, decay=1e-6, momentum=0.9, nesterov=True)
    model.compile(optimizer=sgd, loss='categorical_crossentropy',
                  metrics=['accuracy'])
    # optimizer：指定模型训练的优化器；
    # loss：目标函数
    # class_mode: ”categorical”和”binary”中的一个，只是用来计算分类的精确度或 using the
predict_classes method
    #--------------------------------------------------------------------------
    return model
```

Reference:

[1] http://blog.csdn.net/tina_ttl/article/details/51034869
[2] http://blog.csdn.net/ligang_csdn/article/details/53967031 (***)
[3] http://www.jianshu.com/p/9efae7a20493
[4] http://blog.csdn.net/llp1992/article/details/48057419 (***)
[5] http://blog.csdn.net/niuwei22007/article/details/49207187 (*****)
[6] http://mourafiq.com/2016/08/10/playing-with-convolutions-in-tensorflow.html