Neural Nets II

Elliott Ash

Text Data Course, Bocconi 2018

Tuning NN Hyperparameters

- Number of hidden layers:
  - having a single hidden layer will generally give decent results.
    - more layers with fewer neurons can recover hierarchical relations and complex functions
    - for text classification, try one or two hidden layers as a baseline.
- Number of neurons:
  - a common practice is to set neuron counts like a funnel, with fewer and fewer neurons at each level
  - or just pick 150 neurons per layer
  - overall, better to have too many neurons, and use regularization
- Activation functions:
  - ReLU works well for hidden layers
  - softmax is good for the output layer in classification tasks

Xavier and He Initialization

Activation function	Uniform distribution [–r, r]	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

- Connection weights should be initialized randomly according to a uniform distribution or normal distribution, as indicated in the table (see Geron Chapter 11).

```
model.add(Dense(64, kernel_initializer='he_normal')
model.add(Dense(64, kernel_initializer='he_uniform'
```

Other Activation Functions

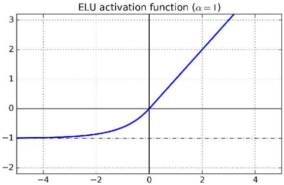
- Leaky ReLU

$$\max(\alpha z, z)$$

  where α is set to a small number, such as .01, or learned in training.

- Exponential linear unit

$$\text{ELU}(z) = \begin{cases} \alpha(\exp(z) - 1) & z < 0 \\ z & z \geq 0 \end{cases}$$



- In general, ELU has had the best performance so far, but it is slower than ReLU.

Batch normalization

- Another trick to speed up training:
  - in between layers, zero-center and normalize the inputs to variance one.
  - normally done before a non-linear activation function

```
from keras.layers.normalization import BatchNormalization
model.add(Dense(64, use_bias=False))
model.add(BatchNormalization())
model.add(Activation('elu'))
```

Regularization for Sparse Models

- As with linear models, neural network parameters can be regularized with an L1 and/or L2 penalty to push weak neurons to zero and produce a sparse model.

```
from keras.regularizers import l1, l2, l1_l2
model.add(Dense(64,
                kernel_regularizer=l2(0.01),
                activity_regularizer=l1(0.01)))
model.add(Dense(64,
                kernel_regularizer=l1_l2(l1=0.01,l2=.01),
                activity_regularizer=l1_l2(l1=0.01,l2=.01)))
```

Dropout

- Another major advance in neural nets is dropout.
  - at every training step, every neuron has some probability p (typically .5) of being temporarily dropped out, so that it will be ignored at this step.
  - after training, neurons dont get dropped any more.
- Neurons trained with dropout:
  - cannot co-adapt with neighboring neurons and must be independently useful.
  - cannot rely excessively on just a few input neurons, they have to pay attention to all input neurons.
    - this makes your model less sensitive to slight changes in the inputs.
- If a model is over-fitting, increase dropout. Dropout can be higher for large layers and lower for small layers.

```python
from keras.layers import Dropout
model.add(Dropout(0.5))
```

Optimizers and loss functions

- Choice of optimization algorithm is the topic of active research, which has shown that it can have a big impact on model performance.

```
model.compile(optimizer='adam',loss='binary_crossentropy')
model.compile(optimizer='sgd',loss='binary_crossentropy')
```

- A good starting choice is Adam (adaptive moment estimation), which is fast and usually works well. For robustness, can also try SGD.
- Loss functions:

Prediction Task	Loss Function to Use
binary classification	`binary_crossentropy`
multi-class classification	`categorical_crossentropy`
regression	`mean_squared_error`

Early stopping

- A popular/efficient regularization method is to continually evaluate your model at regular intervals, and then to stop training when the test-set accuracy starts to decrease.

```
from keras.callbacks import EarlyStopping
earlystop = EarlyStopping(monitor='val_acc',
                          min_delta=0.0001,
                          patience=5,
                          verbose=1,
                          mode='auto')
callbacks_list = [earlystop]
model.fit(X, Y, callbacks=callbacks_list,
          validation_split=0.2)
```

Practical Guidelines

Table 11-2. Default DNN configuration

Initialization	He initialization
Activation function	ELU
Normalization	Batch Normalization
Regularization	Dropout
Optimizer	Adam
Learning rate schedule	None

Batch Training with Large Data

- If data sets don't fit in memory, one can load the data in batches from disk.

```
from numpy import memmap
X_mm = memmap('X.pkl',shape=(32567, 472))

model.fit(X_mm, Y, batch_size=128,
          epochs=100,
          callbacks=callbacks_list,
          validation_split=0.2)
```

- can also continuously update a saved model.

Grid search for model choice

- The flexibility of DNNs is a blessing and a curse.
  - in general, one should make a complex model that allows regularization.
- But still, there are many choices to be made.
  - to choose the number of hidden layers, for example, one can use cross-validation grid search (as we did with standard scikit-learn models).

Grid search for model choice (code)

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

# instantiate KerasClassifier with build function
def create_model(hidden_layers=1):
    model = Sequential()
    model.add(Dense(16, input_dim=num_features))
    for i in range(hidden_layers):
        model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
    return model
clf = KerasClassifier(create_model)

# set oup grid search CV to select number of hidden layers
params = {'hidden_layers' : [0,1,2,3]}
grid = GridSearchCV(clf, param_grid=params)
grid.fit(X,Y)
grid.best_params_
```