In [1]:

```
import numpy as np
import tensorflow as tf
from sklearn.base import BaseEstimator

class DNNBNClassifier(BaseEstimator):

    def __init__(self,
                 hidden_units = [],
                 n_classes = None,
                 learning_rate = 0.03,
                 momentum = 0.9,
                 n_epochs = 30,
                 accuracy = 0.995,
                 batch_size = 50,
                 model_file = "./dnn.ckpt"):
        self.hidden_units = hidden_units
        self.n_classes = n_classes
        self.learning_rate = learning_rate
        self.momentum = momentum
        self.n_epochs = n_epochs
        self.accuracy = accuracy
        self.batch_size = batch_size
        self.model_file = model_file
        self.tf_graph = None
        self.tf_inst = None
        self.tf_logits = None
        self.tf_saver = None

    def fit(self, insts, labels):
        self.tf_graph = None
        self.tf_inst = None
        self.tf_logits = None
        self.tf_saver = None
        tf_graph = tf.Graph()
        with tf_graph.as_default():
            tf_inst = tf.placeholder(tf.float64, shape = (None, insts.shape[1]),
 name = "inst")
            tf_label = tf.placeholder(tf.int64, shape = (None), name = "label")
            tf_training = tf.placeholder_with_default(False, shape = (), name =
"training")
            tf_he_init = tf.contrib.layers.variance_scaling_initializer()
            tf_last = tf_inst
            index = 1
            with tf.name_scope("dnn"):
                for n_hidden in self.hidden_units:
                    tf_layer = tf.layers.dense(tf_last, n_hidden, name = "hidden
%s" % index, kernel_initializer = tf_he_init)
                    tf_norm = tf.layers.batch_normalization(tf_layer, training =
 tf_training, momentum = 0.9)
                    tf_act = tf.nn.elu(tf_norm)
                    tf_last = tf_act
                    index = index + 1
```

```
                tf_out = tf.layers.dense(tf_last, self.n_classes, name = "ouput
s")
                tf_logits = tf.layers.batch_normalization(tf_out, name = "logit
s", training = tf_training, momentum = 0.9)
            with tf.name_scope("loss"):
                tf_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labe
ls = tf_label, logits = tf_logits)
                tf_loss = tf.reduce_mean(tf_entropy, name = "loss")
            with tf.name_scope("train"):
                tf_optimizer = tf.train.MomentumOptimizer(learning_rate = self.l
earning_rate, momentum = self.momentum, use_nesterov = True)
                tf_training_op = tf_optimizer.minimize(tf_loss)
            with tf.name_scope("eval"):
                tf_correct = tf.nn.in_top_k(tf.cast(tf_logits, tf.float32), tf_l
abel, 1)
                tf_accuracy = tf.reduce_mean(tf.cast(tf_correct, tf.float32))
            tf_init_op = tf.global_variables_initializer()
            tf_update_op = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
            tf_saver = tf.train.Saver()
            with tf.Session() as sess:
                tf_init_op.run()
                for epoch in range(self.n_epochs):
                    start = 0
                    for iteration in range(insts.shape[0] // self.batch_size):
                        end = start + self.batch_size
                        if end >= insts.shape[0]:
                            break
                        insts_batch = insts[start:end]
                        labels_batch = labels[start:end]
                        sess.run([tf_training_op, tf_update_op], feed_dict = {tf
_training: True, tf_inst: insts_batch, tf_label: labels_batch})
                        start = start + self.batch_size
                    acc_train = tf_accuracy.eval(feed_dict = {tf_inst: insts, tf
_label: labels})
                    print(epoch, acc_train)
                    if acc_train > self.accuracy:
                        break
                tf_saver.save(sess, self.model_file)
        self.tf_graph = tf_graph
        self.tf_inst = tf_inst
        self.tf_logits = tf_logits
        self.tf_saver = tf_saver

    def predict(self, insts):
        if self.tf_graph is None or self.tf_saver is None or self.tf_logits is N
one or self.tf_inst is None:
            tf_graph = tf.Graph()
            with tf_graph.as_default():
                with tf.Session() as sess:
                    tf_saver = tf.train.import_meta_graph(self.model_file + ".me
ta")
            self.tf_graph = tf_graph
            self.tf_saver = tf_saver
            self.tf_logits = tf_graph.get_tensor_by_name("dnn/logits/batchnorm/a
dd_1:0")
            self.tf_inst = tf_graph.get_tensor_by_name("inst:0")
        with self.tf_graph.as_default():
            with tf.Session() as sess:
                self.tf_saver.restore(sess, self.model_file)
                z = self.tf_logits.eval(feed_dict = {self.tf_inst: insts})
                return np.argmax(z, axis = 1)
```

Die Funktion *fetch_mnist* lädt die MNIST-Daten. Dabei handelt es sich um 60000 handgeschriebene Ziffern in einer Auflösung von 28 x 28 Pixeln. Die ersten 50000 Ziffern werden später für das Training des neuronalen Netzes verwendet. Anhand der restlichen 10000 Ziffern kann der Generalisierungsfehler beurteilt werden.

In [2]:

```
from sklearn.datasets import fetch_mldata
import numpy as np

def fetch_mnist():
    mnist = fetch_mldata("MNIST original")
    insts, labels = mnist["data"], mnist["target"]
    shuffle = np.random.permutation(60000)
    insts, labels = insts[shuffle].astype(np.float64), labels[shuffle].astype(np
.integer)
    train_insts, train_labels = insts[0:50000], labels[0:50000]
    test_insts, test_labels = insts[50000:60000], labels[50000:60000]
    return train_insts, train_labels, test_insts, test_labels

train_insts, train_labels, test_insts, test_labels = fetch_mnist()
```

Hier wird eine Instanz des *DNNBNClassifier* erzeugt. Die Zahl der Neuronen in der Eingangsschicht ergibt sich aus der Zahl der Attribute der Daten zu 784, das ist 28 x 28. Die erste verborgene Schicht hat 160 Neuronen, die zweite hat 25 Neuronen.

Der Klassifizierer wird mit den Trainingsdaten trainiert.

In [3]:

```
dnn = DNNBNClassifier(hidden_units = [160, 25], n_classes = 10)
dnn.fit(train_insts, train_labels)
dnn.get_params()
```

```
0 0.9564
1 0.96958
2 0.97532
3 0.98024
4 0.98372
5 0.98626
6 0.9888
7 0.9913
8 0.9925
9 0.99358
10 0.99422
11 0.99502
```

Out[3]:

```
{'accuracy': 0.995,
 'batch_size': 50,
 'hidden_units': [160, 25],
 'learning_rate': 0.03,
 'model_file': './dnn.ckpt',
 'momentum': 0.9,
 'n_classes': 10,
 'n_epochs': 30}
```

Anhand der Testdaten wird die *Confusion Matrix* berechnet. Die Diagonale der Matrix gibt an, wie viele Ziffern korrekt klassifiziert wurden. Die anderen Elemente der Matrix zählen die falsch erkannten Ziffern.

In [4]:

```
from sklearn.metrics import confusion_matrix

pred = dnn.predict(test_insts)
conf = confusion_matrix(test_labels, pred)
conf
```

```
INFO:tensorflow:Restoring parameters from ./dnn.ckpt
```

Out[4]:

```
array([[1010,    0,    0,    0,    0,    0,    2,    1,    3,    3],
       [   0, 1114,    2,    0,    1,    0,    0,    3,    2,    0],
       [   3,    3,  956,    4,    5,    1,    3,    5,    4,    0],
       [   1,    2,    4,  936,    0,   15,    0,    6,    8,    7],
       [   1,    1,    1,    1,  998,    2,    1,    3,    0,    4],
       [   3,    0,    0,    7,    1,  872,    4,    3,    7,    3],
       [   3,    0,    0,    0,    1,    8,  982,    0,    2,    0],
       [   1,    0,    4,    0,    4,    1,    0,  997,    1,    4],
       [   0,    2,    3,    2,    5,    5,    0,    1,  928,    8],
       [   7,    1,    0,    6,    8,    6,    1,    7,    0,  98
6]])
```

Der folgende Code stellt die ersten vier Ziffern aus den Testdaten an.

In [5]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib

n_digits = 4
d = test_insts[0:n_digits]
dimg = d.reshape(n_digits * 28, 28)
plt.imshow(dimg, cmap=matplotlib.cm.binary, interpolation = "nearest")
plt.axis("off")
plt.show()
```

Und zum Schluss sieht man, was der Klassifizierer erkennt.

In [6]:

```
dnn = DNNBNClassifier()
dnn.predict(d) # Modell und Variablen werden aus den Dateien gelesen
```

```
INFO:tensorflow:Restoring parameters from ./dnn.ckpt
```

Out[6]:

```
array([9, 8, 0, 8])
```