In [1]:
```
# set tf 1.x for colab
%tensorflow_version 1.x
```

```
UsageError: Line magic function `%tensorflow_version` not found.
```

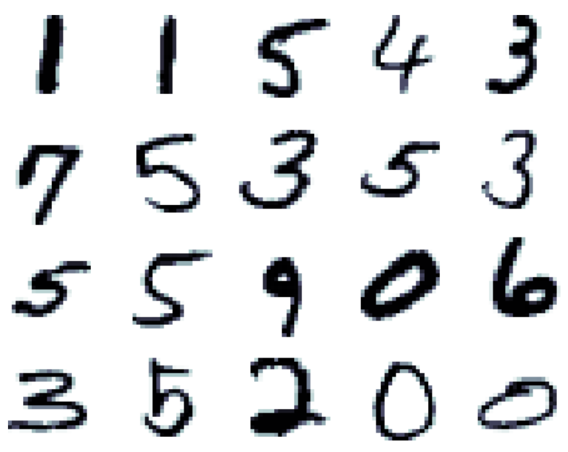
In [2]:
```
import warnings
warnings.filterwarnings('ignore', category=DeprecationWarning)
warnings.filterwarnings('ignore', category=FutureWarning)
```

MNIST digits classification with Keras

We don't expect you to code anything here because you've already solved it with TensorFlow.

But you can appreciate how simpler it is with Keras.

We'll be happy if you play around with the architecture though, there're some tips at the end.



In [3]:
```
import numpy as np
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
%matplotlib inline
import tensorflow as tf
print("We're using TF", tf.__version__)
import keras
print("We are using Keras", keras.__version__)

import sys
sys.path.append("../..")
import keras_utils
from keras_utils import reset_tf_session
```

```
We're using TF 1.14.0
We are using Keras 2.3.1

Using TensorFlow backend.
```

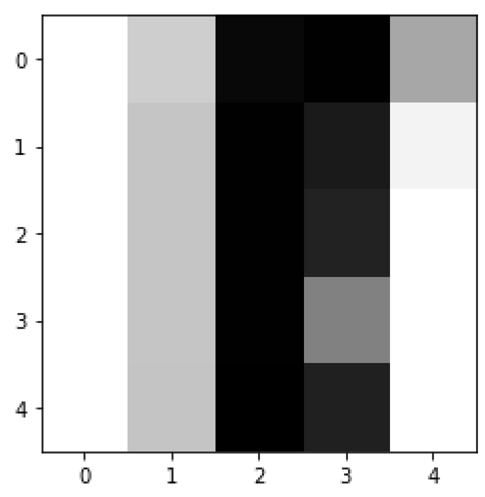
Look at the data

In this task we have 50000 28x28 images of digits from 0 to 9. We will train a classifier on this data.

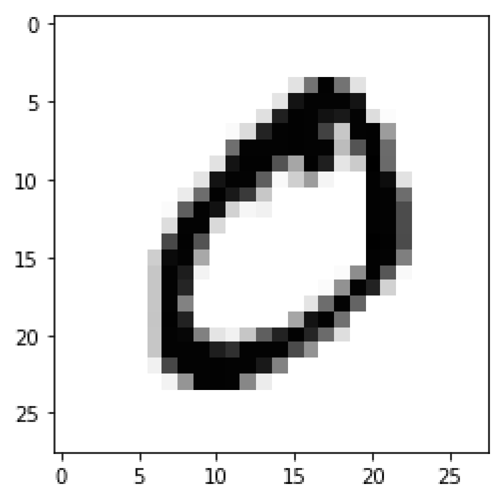
In [4]:
```
import preprocessed_mnist
X_train, y_train, X_val, y_val, X_test, y_test = preprocessed_mnist.load_dataset()
```

In [5]:
```python
# X contains rgb values divided by 255
print("X_train [shape %s] sample patch:\n" % (str(X_train.shape)), X_train[1, 15:20, 5:10])
print("A closeup of a sample patch:")
plt.imshow(X_train[1, 15:20, 5:10], cmap="Greys")
plt.show()
print("And the whole sample:")
plt.imshow(X_train[1], cmap="Greys")
plt.show()
print("y_train [shape %s] 10 samples:\n" % (str(y_train.shape)), y_train[:10])
```

```
X_train [shape (50000, 28, 28)] sample patch:
 [[0.         0.29803922 0.96470588 0.98823529 0.43921569]
 [0.         0.33333333 0.98823529 0.90196078 0.09803922]
 [0.         0.33333333 0.98823529 0.8745098  0.        ]
 [0.         0.33333333 0.98823529 0.56862745 0.        ]
 [0.         0.3372549  0.99215686 0.88235294 0.        ]]
A closeup of a sample patch:
```



```
And the whole sample:
```



```
y_train [shape (50000,)] 10 samples:
 [5 0 4 1 9 2 1 3 1 4]
```

In [6]:
```python
# flatten images
X_train_flat = X_train.reshape((X_train.shape[0], -1))
print(X_train_flat.shape)

X_val_flat = X_val.reshape((X_val.shape[0], -1))
print(X_val_flat.shape)
```

```
(50000, 784)
(10000, 784)
```

In [7]:
```python
# one-hot encode the target
y_train_oh = keras.utils.to_categorical(y_train, 10)
y_val_oh = keras.utils.to_categorical(y_val, 10)

print(y_train_oh.shape)
print(y_train_oh[:3], y_train[:3])
```

```
(50000, 10)
[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]] [5 0 4]
```

In [8]:
```python
# building a model with keras
from keras.layers import Dense, Activation
from keras.models import Sequential

# we still need to clear a graph though
s = reset_tf_session()

model = Sequential()  # it is a feed-forward network without loops like in RNN
model.add(Dense(256, input_shape=(784,)))  # the first layer must specify the input shape (replacing placeholders)
model.add(Activation('sigmoid'))
model.add(Dense(256))
model.add(Activation('sigmoid'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

```
WARNING:tensorflow:From ../..\keras_utils.py:68: The name tf.get_default_session is deprecated. Please use tf.compat.v
1.get_default_session instead.

WARNING:tensorflow:From ../..\keras_utils.py:75: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigP
roto instead.

WARNING:tensorflow:From ../..\keras_utils.py:77: The name tf.InteractiveSession is deprecated. Please use tf.compat.v1.
InteractiveSession instead.
```

In [9]:
```python
# you can look at all layers and parameter count
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 256)               200960    
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792     
_________________________________________________________________
activation_2 (Activation)    (None, 256)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 10)                2570      
_________________________________________________________________
activation_3 (Activation)    (None, 10)                0         
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```python
# now we "compile" the model specifying the loss and optimizer
model.compile(
    loss='categorical_crossentropy', # this is our cross-entropy
    optimizer='adam',
    metrics=['accuracy']  # report accuracy during training
)
```

In [11]:
```python
# and now we can fit the model with model.fit()
# and we don't have to write loops and batching manually as in TensorFlow
model.fit(
    X_train_flat,
    y_train_oh,
    batch_size=512,
    epochs=40,
    validation_data=(X_val_flat, y_val_oh),
    callbacks=[keras_utils.TqdmProgressCallback()],
    verbose=0
)
```

```
WARNING:tensorflow:From C:\Users\Xiaowei\Anaconda3\envs\tfspark\lib\site-packages\keras\backend\tensorflow_backend.py:4
22: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.


Epoch 1/40

 loss: 1.2463; accuracy: 0.4566; val_loss: 0.4872; … 99/? [00:18<00:00, 5.42it/s]


Epoch 2/40

 loss: 0.3977; accuracy: 0.8839; val_loss: 0.2996; … 99/? [00:01<00:00, 63.09it/s]


Epoch 3/40

 loss: 0.2938; accuracy: 0.9146; val_loss: 0.2477; … 99/? [00:16<00:00, 6.01it/s]


Epoch 4/40

 loss: 0.2496; accuracy: 0.9255; val_loss: 0.2177; … 99/? [00:15<00:00, 6.30it/s]


Epoch 5/40

 loss: 0.2199; accuracy: 0.9347; val_loss: 0.1978; … 99/? [00:02<00:00, 42.21it/s]


Epoch 6/40

 loss: 0.1944; accuracy: 0.9422; val_loss: 0.1795; … 99/? [00:01<00:00, 63.38it/s]


Epoch 7/40

 loss: 0.1740; accuracy: 0.9478; val_loss: 0.1628; … 99/? [00:13<00:00, 7.41it/s]


Epoch 8/40

 loss: 0.1559; accuracy: 0.9533; val_loss: 0.1509; … 99/? [00:12<00:00, 7.87it/s]


Epoch 9/40

 loss: 0.1423; accuracy: 0.9586; val_loss: 0.1385; … 99/? [00:02<00:00, 41.75it/s]


Epoch 10/40

 loss: 0.1284; accuracy: 0.9604; val_loss: 0.1319; … 99/? [00:01<00:00, 62.24it/s]


Epoch 11/40

 loss: 0.1156; accuracy: 0.9662; val_loss: 0.1222; … 99/? [00:10<00:00, 9.68it/s]


Epoch 12/40
```

```
 loss: 0.1052; accuracy: 0.9702; val_loss: 0.1164; … 99/? [00:09<00:00, 10.49it/s]

Epoch 13/40
 loss: 0.0960; accuracy: 0.9718; val_loss: 0.1148; … 99/? [00:02<00:00, 41.83it/s]

Epoch 14/40
 loss: 0.0875; accuracy: 0.9757; val_loss: 0.1060; … 99/? [00:01<00:00, 62.78it/s]

Epoch 15/40
 loss: 0.0795; accuracy: 0.9771; val_loss: 0.0998; … 99/? [00:07<00:00, 14.00it/s]

Epoch 16/40
 loss: 0.0722; accuracy: 0.9792; val_loss: 0.0970; … 99/? [00:06<00:00, 15.75it/s]

Epoch 17/40
 loss: 0.0660; accuracy: 0.9809; val_loss: 0.0937; … 99/? [00:02<00:00, 41.79it/s]

Epoch 18/40
 loss: 0.0595; accuracy: 0.9842; val_loss: 0.0898; … 99/? [00:01<00:00, 62.24it/s]

Epoch 19/40
 loss: 0.0544; accuracy: 0.9859; val_loss: 0.0897; … 99/? [00:03<00:00, 25.19it/s]

Epoch 20/40
 loss: 0.0503; accuracy: 0.9880; val_loss: 0.0839; … 99/? [00:03<00:00, 31.55it/s]

Epoch 21/40
 loss: 0.0450; accuracy: 0.9884; val_loss: 0.0847; … 99/? [00:02<00:00, 42.12it/s]

Epoch 22/40
 loss: 0.0407; accuracy: 0.9895; val_loss: 0.0821; … 99/? [00:01<00:00, 62.70it/s]

Epoch 23/40
 loss: 0.0364; accuracy: 0.9907; val_loss: 0.0819; … 99/? [00:00<00:00, 126.36it/s]

Epoch 24/40
 loss: 0.0340; accuracy: 0.9913; val_loss: 0.0811; … 99/? [00:02<00:00, 40.71it/s]

Epoch 25/40
 loss: 0.0302; accuracy: 0.9927; val_loss: 0.0810; … 99/? [00:01<00:00, 60.83it/s]

Epoch 26/40
```

```
 loss: 0.0268; accuracy: 0.9943; val_loss: 0.0759; … 99/? [00:34<00:00, 2.88it/s]

Epoch 27/40

 loss: 0.0244; accuracy: 0.9945; val_loss: 0.0754; … 99/? [00:33<00:00, 2.95it/s]

Epoch 28/40

 loss: 0.0218; accuracy: 0.9959; val_loss: 0.0780; … 99/? [00:02<00:00, 41.19it/s]

Epoch 29/40

 loss: 0.0202; accuracy: 0.9959; val_loss: 0.0770; … 99/? [00:01<00:00, 62.24it/s]

Epoch 30/40

 loss: 0.0181; accuracy: 0.9968; val_loss: 0.0762; … 99/? [00:31<00:00, 3.18it/s]

Epoch 31/40

 loss: 0.0160; accuracy: 0.9975; val_loss: 0.0756; … 99/? [00:30<00:00, 3.27it/s]

Epoch 32/40

 loss: 0.0140; accuracy: 0.9979; val_loss: 0.0760; … 99/? [00:02<00:00, 42.13it/s]

Epoch 33/40

 loss: 0.0127; accuracy: 0.9981; val_loss: 0.0751; … 99/? [00:01<00:00, 62.80it/s]

Epoch 34/40

 loss: 0.0118; accuracy: 0.9985; val_loss: 0.0776; … 99/? [00:27<00:00, 3.54it/s]

Epoch 35/40

 loss: 0.0099; accuracy: 0.9991; val_loss: 0.0766; … 99/? [00:27<00:00, 3.64it/s]

Epoch 36/40

 loss: 0.0096; accuracy: 0.9983; val_loss: 0.0781; … 99/? [00:02<00:00, 42.68it/s]

Epoch 37/40

 loss: 0.0081; accuracy: 0.9991; val_loss: 0.0778; … 99/? [00:01<00:00, 63.54it/s]

Epoch 38/40

 loss: 0.0074; accuracy: 0.9994; val_loss: 0.0774; … 99/? [00:24<00:00, 3.97it/s]

Epoch 39/40

 loss: 0.0064; accuracy: 0.9995; val_loss: 0.0768; … 99/? [00:24<00:00, 4.10it/s]

Epoch 40/40
```

```
loss: 0.0058; accuracy: 0.9993; val_loss: 0.0787; … 99/? [00:00<00:00, 130.69it/s]
```

```
Out[11]: <keras.callbacks.callbacks.History at 0x241c290cd08>
```

Here're the notes for those who want to play around here

Here are some tips on what you could do:

- **Network size**
  - More neurons,
  - More layers, (docs (https://keras.io/))
  - Other nonlinearities in the hidden layers
    - tanh, relu, leaky relu, etc
  - Larger networks may take more epochs to train, so don't discard your net just because it could didn't beat the baseline in 5 epochs.
- **Early Stopping**
  - Training for 100 epochs regardless of anything is probably a bad idea.
  - Some networks converge over 5 epochs, others - over 500.
  - Way to go: stop when validation score is 10 iterations past maximum
- **Faster optimization**
  - rmsprop, nesterov_momentum, adam, adagrad and so on.
    - Converge faster and sometimes reach better optima
    - It might make sense to tweak learning rate/momentum, other learning parameters, batch size and number of epochs
- **Regularize** to prevent overfitting
  - Add some L2 weight norm to the loss function, theano will do the rest
    - Can be done manually or via - https://keras.io/regularizers/ (https://keras.io/regularizers/)
- **Data augmemntation** - getting 5x as large dataset for free is a great deal
  - https://keras.io/preprocessing/image/ (https://keras.io/preprocessing/image/)
  - Zoom-in+slice = move
  - Rotate+zoom(to remove black stripes)
  - any other perturbations
  - Simple way to do that (if you have PIL/Image):
    - `from scipy.misc import imrotate,imresize`
    - and a few slicing
  - Stay realistic. There's usually no point in flipping dogs upside down as that is not the way you usually see them.

```
In [ ]:
```