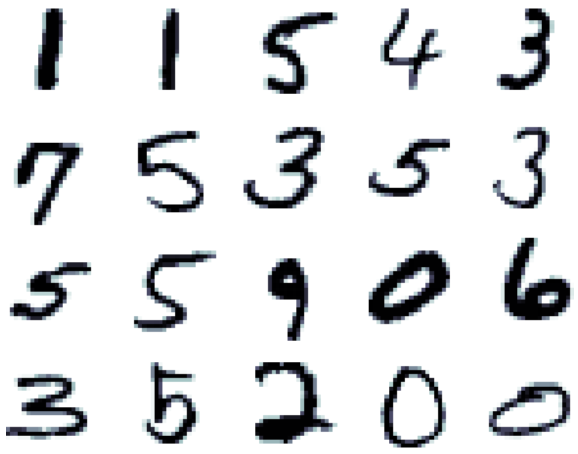
In [1]:

```
# set tf 1.x for colab
%tensorflow_version 1.x
```

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

MNIST digits classification with TensorFlow



In [2]:

```
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 sys
sys.path.append("../..")
import grading

import matplotlib_utils
from importlib import reload
reload(matplotlib_utils)

import grading_utils
reload(grading_utils)

import keras_utils
from keras_utils import reset_tf_session
```

```
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:516: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:517: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:518: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:519: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:520: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:525: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])

We're using TF 1.14.0

/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:541: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,))
/ '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,))
/ '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:543: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,))
/ '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:544: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,))
/ '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:545: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,))
/ '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWarning: P
assing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will be understood as (type, (1,))
/ '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
Using TensorFlow backend.
```

Fill in your Coursera token and email

To successfully submit your answers to our grader, please fill in your Coursera submission token and email

In [3]:

```
grader = grading.Grader(assignment_key="XtD7ho3TEeiHQBLWejjYAA",
                        all_parts=["9XaAS", "vmogZ", "RMv95", "i8bgs", "rE763"])
```

In [4]:

```
# token expires every 30 min
COURSERA_TOKEN = "xsKaGouklo4Fsk8K"
COURSERA_EMAIL = "lxwvictor@gmail.com"
```

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.

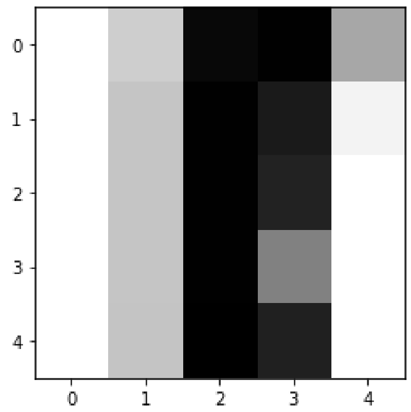
In [5]:

```
import preprocessed_mnist
X_train, y_train, X_val, y_val, X_test, y_test = preprocessed_mnist.load_dataset()
```

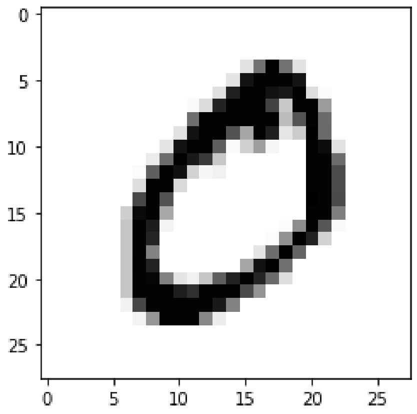
In [6]:

```
# 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]
```

Linear model

Your task is to train a linear classifier $\vec{x} \rightarrow y$ with SGD using TensorFlow.

You will need to calculate a logit (a linear transformation) z_k for each class:

$$z_k = \vec{x} \cdot \vec{w_k} + b_k \quad k = 0..9$$

And transform logits z_k to valid probabilities p_k with softmax:

$$p_k = \frac{e^{z_k}}{\sum_{i=0}^{9} e^{z_i}} \quad k = 0..9$$

We will use a cross-entropy loss to train our multi-class classifier:

$$\text{cross-entropy}(y, p) = -\sum_{k=0}^{9} \log(p_k)[y = k]$$

where

$$[x] = \begin{cases} 1, & \text{if } x \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

Cross-entropy minimization pushes p_k close to 1 when $y = k$, which is what we want.

Here's the plan:

- Flatten the images (28x28 -> 784) with `X_train.reshape((X_train.shape[0], -1))` to simplify our linear model implementation
- Use a matrix placeholder for flattened `X_train`
- Convert `y_train` to one-hot encoded vectors that are needed for cross-entropy

- Use a shared variable `W` for all weights (a column $\vec{w}_k$ per class) and `b` for all biases.
- Aim for ~0.93 validation accuracy

In [7]:

```
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 [8]:

```
import keras

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 [9]:

```
# run this again if you remake your graph
s = reset_tf_session()
```

```
WARNING:tensorflow:From ../../keras_utils.py:68: The name tf.get_default_session is deprecated. Please use tf.compat.v1.get_default
_session instead.

WARNING:tensorflow:From /Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:95: Th
e name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.

WARNING:tensorflow:From /Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:98: Th
e name tf.placeholder_with_default is deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:102: T
he name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From ../../keras_utils.py:75: The name tf.ConfigProto is deprecated. Please use tf.compat.v1.ConfigProto instea
d.
```

In [10]:

```
# Model parameters: W and b
### YOUR CODE HERE ### tf.get_variable(...) with shape[0] = 784
W = tf.get_variable('W', shape=(784, 10), dtype=tf.float32)
### YOUR CODE HERE ### tf.get_variable(...)
b = tf.get_variable('b', shape=(10), dtype=tf.float32)
```

```
WARNING:tensorflow:From /Users/Victor/anaconda3/envs/tfspark/lib/python3.7/site-packages/tensorflow/python/ops/init_ops.py:1251: ca
lling VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future versi
on.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
```

In [11]:

```
# Placeholders for the input data
### YOUR CODE HERE ### tf.placeholder(...) for flat X with shape[0] = None for any batch size
input_X = tf.placeholder(tf.float32, shape=(None, 784))
### YOUR CODE HERE ### tf.placeholder(...) for one-hot encoded true labels
input_y = tf.placeholder(tf.int32, shape=(None, 10))
```

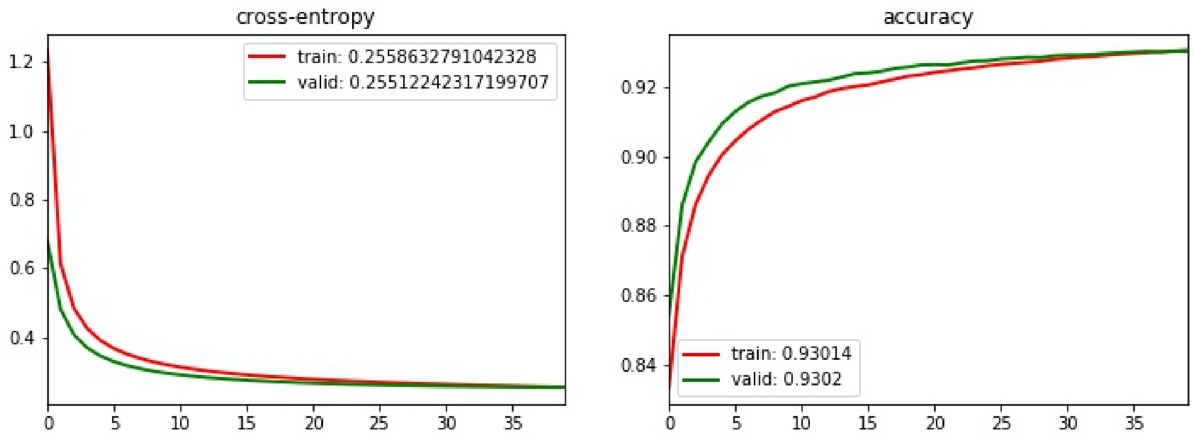
In [12]:

```
# Compute predictions
### YOUR CODE HERE ### logits for input_X, resulting shape should be [input_X.shape[0], 10]
logits = input_X @ W + b
# logits = tf.add(tf.matmul(input_X, W), b)
### YOUR CODE HERE ### apply tf.nn.softmax to logits
probas = tf.nn.softmax(logits)
### YOUR CODE HERE ### apply tf.argmax to find a class index with highest probability
classes = tf.argmax(probas, axis=1)

# Loss should be a scalar number: average loss over all the objects with tf.reduce_mean().
# Use tf.nn.softmax_cross_entropy_with_logits on top of one-hot encoded input_y and logits.
# It is identical to calculating cross-entropy on top of probas, but is more numerically friendly (read the docs).
### YOUR CODE HERE ### cross-entropy loss
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=input_y, logits=logits))

# Use a default tf.train.AdamOptimizer to get an SGD step
### YOUR CODE HERE ### optimizer step that minimizes the loss
step = tf.train.AdamOptimizer().minimize(loss)
```

```
WARNING:tensorflow:From <ipython-input-12-6e9b97a498be>:14: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) i
s deprecated and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.
```

In [13]:

```python
s.run(tf.global_variables_initializer())

BATCH_SIZE = 512
EPOCHS = 40

# for logging the progress right here in Jupyter (for those who don't have TensorBoard)
simpleTrainingCurves = matplotlib_utils.SimpleTrainingCurves("cross-entropy", "accuracy")

for epoch in range(EPOCHS):  # we finish an epoch when we've looked at all training samples

    batch_losses = []
    for batch_start in range(0, X_train_flat.shape[0], BATCH_SIZE):  # data is already shuffled
        _, batch_loss = s.run([step, loss], {input_X: X_train_flat[batch_start:batch_start+BATCH_SIZE],
                                             input_y: y_train_oh[batch_start:batch_start+BATCH_SIZE]})
        # collect batch losses, this is almost free as we need a forward pass for backprop anyway
        batch_losses.append(batch_loss)

    train_loss = np.mean(batch_losses)
#     print('train_loss', train_loss)
    val_loss = s.run(loss, {input_X: X_val_flat, input_y: y_val_oh})  # this part is usually small
#     print('val_loss', val_loss)
    train_accuracy = accuracy_score(y_train, s.run(classes, {input_X: X_train_flat}))  # this is slow and usually skipped
    valid_accuracy = accuracy_score(y_val, s.run(classes, {input_X: X_val_flat}))
    simpleTrainingCurves.add(train_loss, val_loss, train_accuracy, valid_accuracy)
```



Submit a linear model

In [14]:

```python
## GRADED PART, DO NOT CHANGE!
# Testing shapes
grader.set_answer("9XaAS", grading_utils.get_tensors_shapes_string([W, b, input_X, input_y, logits, probas, classes]))
# Validation loss
grader.set_answer("vmogZ", s.run(loss, {input_X: X_val_flat, input_y: y_val_oh}))
# Validation accuracy
grader.set_answer("RMv95", accuracy_score(y_val, s.run(classes, {input_X: X_val_flat})))
```

In [15]:

```python
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

MLP with hidden layers

Previously we've coded a dense layer with matrix multiplication by hand. But this is not convenient, you have to create a lot of variables and your code becomes a mess. In TensorFlow there's an easier way to make a dense layer:

```python
hidden1 = tf.layers.dense(inputs, 256, activation=tf.nn.sigmoid)
```

That will create all the necessary variables automatically. Here you can also choose an activation function (remember that we need it for a hidden layer!).

Now define the MLP with 2 hidden layers and restart training with the cell above.

You're aiming for ~0.97 validation accuracy here.

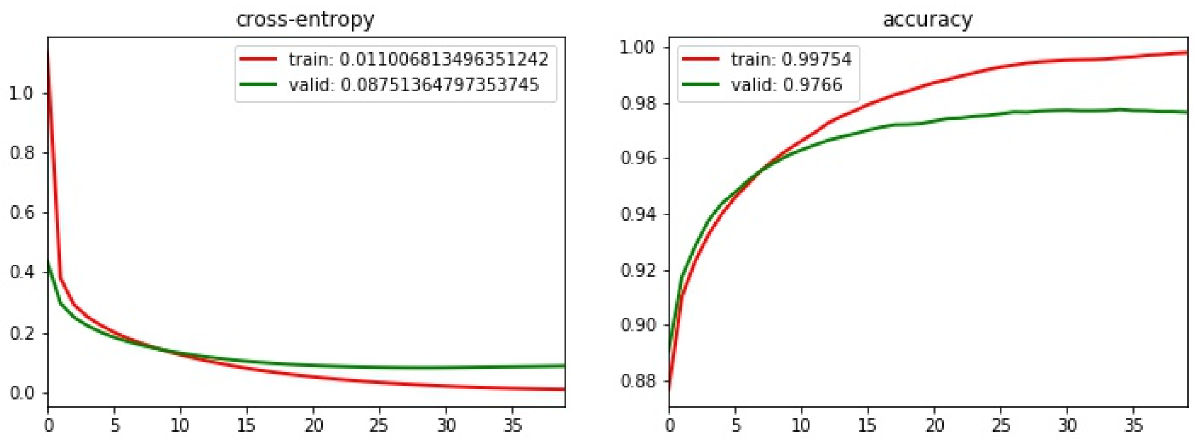
In [16]:

```
# write the code here to get a new `step` operation and then run the cell with training loop above.
# name your variables in the same way (e.g. logits, probas, classes, etc) for safety.
### YOUR CODE HERE ###
hidden1 = tf.layers.dense(input_X, 256, activation=tf.nn.sigmoid)
hidden2 = tf.layers.dense(hidden1, 256, activation=tf.nn.sigmoid)
logits = tf.layers.dense(hidden2, 10)

probas = tf.nn.softmax(logits, 1)
classes = tf.argmax(probas, axis=1)
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=input_y, logits=logits))

step = tf.train.AdamOptimizer().minimize(loss)
```

```
WARNING:tensorflow:From <ipython-input-16-9f11ab2822ef>:4: dense (from tensorflow.python.layers.core) is deprecated and will be rem
oved in a future version.
Instructions for updating:
Use keras.layers.dense instead.
WARNING:tensorflow:Entity <bound method Dense.call of <tensorflow.python.layers.core.Dense object at 0x1a773abe10>> could not be tr
ansformed and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the verbosity to 10 (on L
inux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound method Dense.call of <tensorflow.pytho
n.layers.core.Dense object at 0x1a773abe10>>: AssertionError: Bad argument number for Name: 3, expecting 4
WARNING: Entity <bound method Dense.call of <tensorflow.python.layers.core.Dense object at 0x1a773abe10>> could not be transformed
and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `exp
ort AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound method Dense.call of <tensorflow.python.layers.co
re.Dense object at 0x1a773abe10>>: AssertionError: Bad argument number for Name: 3, expecting 4
WARNING:tensorflow:Entity <bound method Dense.call of <tensorflow.python.layers.core.Dense object at 0x1a773aba90>> could not be tr
ansformed and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the verbosity to 10 (on L
inux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound method Dense.call of <tensorflow.pytho
n.layers.core.Dense object at 0x1a773aba90>>: AssertionError: Bad argument number for Name: 3, expecting 4
WARNING: Entity <bound method Dense.call of <tensorflow.python.layers.core.Dense object at 0x1a773aba90>> could not be transformed
and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `exp
ort AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound method Dense.call of <tensorflow.python.layers.co
re.Dense object at 0x1a773aba90>>: AssertionError: Bad argument number for Name: 3, expecting 4
WARNING:tensorflow:Entity <bound method Dense.call of <tensorflow.python.layers.core.Dense object at 0x1a77273410>> could not be tr
ansformed and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the verbosity to 10 (on L
inux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound method Dense.call of <tensorflow.pytho
n.layers.core.Dense object at 0x1a77273410>>: AssertionError: Bad argument number for Name: 3, expecting 4
WARNING: Entity <bound method Dense.call of <tensorflow.python.layers.core.Dense object at 0x1a77273410>> could not be transformed
and will be executed as-is. Please report this to the AutgoGraph team. When filing the bug, set the verbosity to 10 (on Linux, `exp
ort AUTOGRAPH_VERBOSITY=10`) and attach the full output. Cause: converting <bound method Dense.call of <tensorflow.python.layers.co
re.Dense object at 0x1a77273410>>: AssertionError: Bad argument number for Name: 3, expecting 4
```

In [17]:

```
s.run(tf.global_variables_initializer())

BATCH_SIZE = 512
EPOCHS = 40

# for logging the progress right here in Jupyter (for those who don't have TensorBoard)
simpleTrainingCurves = matplotlib_utils.SimpleTrainingCurves("cross-entropy", "accuracy")

for epoch in range(EPOCHS):  # we finish an epoch when we've looked at all training samples

    batch_losses = []
    for batch_start in range(0, X_train_flat.shape[0], BATCH_SIZE):  # data is already shuffled
        _, batch_loss = s.run([step, loss], {input_X: X_train_flat[batch_start:batch_start+BATCH_SIZE],
                                             input_y: y_train_oh[batch_start:batch_start+BATCH_SIZE]})
        # collect batch losses, this is almost free as we need a forward pass for backprop anyway
        batch_losses.append(batch_loss)

    train_loss = np.mean(batch_losses)
#     print('train_loss', train_loss)
    val_loss = s.run(loss, {input_X: X_val_flat, input_y: y_val_oh})  # this part is usually small
#     print('val_loss', val_loss)
    train_accuracy = accuracy_score(y_train, s.run(classes, {input_X: X_train_flat}))  # this is slow and usually skipped
    valid_accuracy = accuracy_score(y_val, s.run(classes, {input_X: X_val_flat}))
    simpleTrainingCurves.add(train_loss, val_loss, train_accuracy, valid_accuracy)
```



Submit the MLP with 2 hidden layers

Run these cells after training the MLP with 2 hidden layers

In [18]:

```
## GRADED PART, DO NOT CHANGE!
# Validation loss for MLP
grader.set_answer("i8bgs", s.run(loss, {input_X: X_val_flat, input_y: y_val_oh}))
# Validation accuracy for MLP
grader.set_answer("rE763", accuracy_score(y_val, s.run(classes, {input_X: X_val_flat})))
```

In [19]:

```
# you can make submission with answers so far to check yourself at this stage
grader.submit(COURSERA_EMAIL, COURSERA_TOKEN)
```

```
Submitted to Coursera platform. See results on assignment page!
```

In [ ]:

```

```