Machine Learning
Second Assignment

Deadline: 21 Jan 2020

The exercise should be implemented in the Python programming language using Keras.

- Code snippets to aid you with your exercise can be found in e-class.
- Your final report should contain enough details to make your results reproducible.
- For exercise 3 you are free to use any publicly available data set as long as you specify what you have used. You may also augment any of these data sets as you see fit.

Installation

- The python installation procedure depends on your operating system. You can find information to help you with your installation online. Make sure you also install the pip tool. If you are using Ubuntu 18.04, python is already installed and pip can be installed with the comand "sudo apt-get install python-pip".
- Once you have installed python and pip, type in your terminal "pip install keras tensorflow matplotlib numpy". You can install any additional library using "pip install ⟨ library_name ⟩".
- Additional resources to help you get familiar with Python and Keras will be uploaded in e-class.

Exercise 1. *40%*

(A) Rewrite the general form of the backpropagation equations given in class for MLPs for the following specific activation functions i) ReLU, ii) hyperbolic tangent and iii) sigmoid. For each activation function write down the range of the gradients.

(B) Download the MNIST data set and use it to train a fully-connected neural network to recognise handwritten digits. The MNIST data set can be loaded from *keras.datasets*. Make sure that your data is normalized. Each hidden layer should have 32 units and the output layer should have a softmax activation function. Compile your model to use the standard SGD optimizer with a learning rate of 0.01 and the categorical crossentropy loss function. Use (i) 5, (ii) 20 and (iii) 40 layers. For each choice (i), (ii) and (iii), use the (a) ReLU, (b) hyperbolic tangent and (c) sigmoid activation function on all hidden layers. Report the test scores for each model. What are your observations?

(C) For each of the models in (B), trained for 3 epochs on the MNIST data set, compute for each layer the maximum value of the gradient on a given mini-batch and create a plot of "layer depth vs. max gradient". Organize your plots as a grid so that your results for the different activation functions for each depth choice appear on the same subplot. Can you explain your observations? What insight do you gain for the observations in (B)?

(D) Train a model using the topology given in (B) and the activation function

$$LeCun(x) = 1.7159\ tanh(\frac{2}{3}x) + 0.01x.$$

Compare the learning curves of the models using LeCun and hyperbolic tangent activation functions. Write down the backpropagation equations and the gradient range for the LeCun activation function. Plot the gradients for the choices of depth given above for an untrained model using LeCun and hyperbolic tangent activations.

Exercise 2. *20%*

(A) Download the Street View House Numbers (SVHN) data set and train a convolutional neural network for classification. The data set can be found here *http://ufldl.stanford.edu/housenumbers/*. Your topology should be as follows: Use 3 convolutional layers and 3 max pooling layers interchangeably, then flatten your output and perform multi-label logistic regression on it. Use a 3×3 kernel for both your convolutional and max pooling layers. Use 9, 36 and 49 filters for your convolutional layers. Use an early stopping callback to monitor the validation loss of your model on a validation set of 7326 images (randomly chosen from the train set). Plot your model's confusion matrix.

(B) For a randomly chosen image from each class, plot your model's output for each convolutional filter and group your plots by layer on grids. What do you observe?

Exercise 3. 40%

Train a classifier to recognise handwritten digits. Your model must be saved in .h5 format and will be tested on digits written by us. Your marks for this exercise will be $40 * \frac{your\_test\_accuracy}{your\_class\_best\_test\_accuracy}$.

In [1]:
```
# import relevant libraries
import numpy as np
import matplotlib.pyplot as plt
import keras.layers as l
import keras.optimizers as o
import keras.models as m
import keras
from keras.utils import to_categorical
```

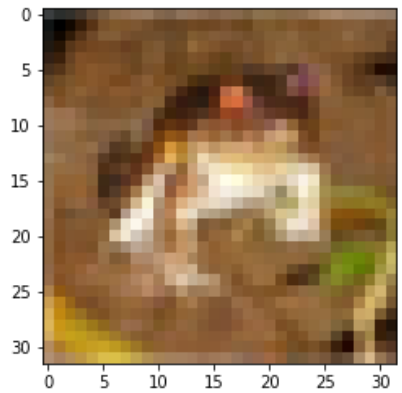
```
Using TensorFlow backend.
```

In [3]:
```
# load cifar10 dataset
from keras.datasets import cifar10

(x_train,y_train), (x_test,y_test) = cifar10.load_data()
# rescale pixel values
x_train = (1/255.0)*x_train
x_test = (1/255.0)*x_test
# labels -> one-hot encodings
y_train = to_categorical(y_train.reshape([-1, 1]))
y_test = to_categorical(y_test.reshape([-1, 1]))
```

In [4]:
```
# for jupyter notebook (ignore)
%matplotlib inline
```

In [5]:
```
# plot image
plt.imshow(x_train[0])
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7ff7359c2ef0>
```



Sequential API

In [6]:
```
# define model using sequential api
model = m.Sequential([
    l.Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(32,
    l.MaxPooling2D((3,3)),
    l.Conv2D(64, (3,3), padding='same', activation='relu'),
    l.MaxPooling2D((3,3)),
    l.Flatten(),
    l.Dense(32, activation='relu'),
    l.Dense(10, activation="softmax")
])

# print a model summary
model.summary()
```

```
WARNING: Logging before flag parsing goes to stderr.
W1230 13:55:03.580109 140701437798144 deprecation_wrapper.py:119] From /ho
me/gpik/miniconda3/lib/python3.7/site-packages/keras/backend/tensorflow_ba
ckend.py:74: The name tf.get_default_graph is deprecated. Please use tf.co
mpat.v1.get_default_graph instead.

W1230 13:55:03.607579 140701437798144 deprecation_wrapper.py:119] From /ho
me/gpik/miniconda3/lib/python3.7/site-packages/keras/backend/tensorflow_ba
ckend.py:517: The name tf.placeholder is deprecated. Please use tf.compat.
v1.placeholder instead.

W1230 13:55:03.611763 140701437798144 deprecation_wrapper.py:119] From /ho
me/gpik/miniconda3/lib/python3.7/site-packages/keras/backend/tensorflow_ba
ckend.py:4138: The name tf.random_uniform is deprecated. Please use tf.ran
dom.uniform instead.

W1230 13:55:03.630030 140701437798144 deprecation_wrapper.py:119] From /ho
me/gpik/miniconda3/lib/python3.7/site-packages/keras/backend/tensorflow_ba
ckend.py:3976: The name tf.nn.max_pool is deprecated. Please use tf.nn.max
_pool2d instead.


_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                18464
_________________________________________________________________
dense_2 (Dense)              (None, 10)                330
=================================================================
Total params: 38,186
Trainable params: 38,186
Non-trainable params: 0
_________________________________________________________________
```

In [7]:
```
# compile a model
model.compile(optimizer=o.Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
W1230 13:55:03.697125 140701437798144 deprecation_wrapper.py:119] From /ho
me/gpik/miniconda3/lib/python3.7/site-packages/keras/optimizers.py:790: Th
e name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Opt
imizer instead.

W1230 13:55:03.705946 140701437798144 deprecation_wrapper.py:119] From /ho
me/gpik/miniconda3/lib/python3.7/site-packages/keras/backend/tensorflow_ba
ckend.py:3295: The name tf.log is deprecated. Please use tf.math.log inste
ad.
```

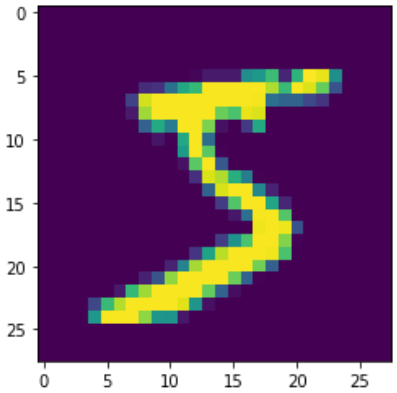
In [8]:
```
# fit model to data
model.fit(x_train,y_train, epochs=10, validation_data = (x_test,y_test))
```

```
W1230 13:55:03.814929 140701437798144 deprecation.py:323] From /home/gpik/
miniconda3/lib/python3.7/site-packages/tensorflow/python/ops/math_grad.py:
1250: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.ar
ray_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
W1230 13:55:03.862976 140701437798144 deprecation_wrapper.py:119] From /ho
me/gpik/miniconda3/lib/python3.7/site-packages/keras/backend/tensorflow_ba
ckend.py:986: The name tf.assign_add is deprecated. Please use tf.compat.v
1.assign_add instead.


Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 15s 306us/step - loss: 1.57
07 - acc: 0.4331 - val_loss: 1.2921 - val_acc: 0.5419
Epoch 2/10
50000/50000 [==============================] - 14s 290us/step - loss: 1.18
55 - acc: 0.5826 - val_loss: 1.1453 - val_acc: 0.5940
Epoch 3/10
```

Functional API

In [9]:
```
from keras.datasets import mnist

# load mnist data
(x_train,y_train), (x_test,y_test) = mnist.load_data()
# rescale pixel values
x_train = (1/255.0)*x_train
x_test = (1/255.0)*x_test
# labels -> one-hot encodings
y_train = to_categorical(y_train.reshape([-1, 1]))
y_test = to_categorical(y_test.reshape([-1, 1]))
plt.imshow(x_train[0])
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x7ff73471e898>
```



In [10]:
```
# define model & compile
input_layer = l.Input((28,28))
Intermediate_layer = l.Flatten()(input_layer)
Intermediate_layer=l.Dense(256, activation='relu')(Intermediate_layer)
output_layer = l.Dense(10, activation="softmax")(Intermediate_layer)

model = m.Model(input_layer, output_layer)
model.summary()
model.compile(optimizer=o.SGD(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28)            0
_________________________________________________________________
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               200960
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
model.fit(x_train,y_train, epochs=10, validation_data = (x_test,y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 3s 55us/step - loss: 0.6223
- acc: 0.8451 - val_loss: 0.3476 - val_acc: 0.9071
Epoch 2/10
60000/60000 [==============================] - 3s 53us/step - loss: 0.3284
- acc: 0.9082 - val_loss: 0.2839 - val_acc: 0.9204
Epoch 3/10
60000/60000 [==============================] - 3s 53us/step - loss: 0.2797
- acc: 0.9218 - val_loss: 0.2522 - val_acc: 0.9309
Epoch 4/10
60000/60000 [==============================] - 3s 53us/step - loss: 0.2490
- acc: 0.9309 - val_loss: 0.2309 - val_acc: 0.9333
Epoch 5/10
60000/60000 [==============================] - 3s 52us/step - loss: 0.2260
- acc: 0.9372 - val_loss: 0.2110 - val_acc: 0.9391
Epoch 6/10
60000/60000 [==============================] - 3s 52us/step - loss: 0.2072
- acc: 0.9423 - val_loss: 0.1973 - val_acc: 0.9444
Epoch 7/10
60000/60000 [==============================] - 3s 52us/step - loss: 0.1916
- acc: 0.9468 - val_loss: 0.1832 - val_acc: 0.9482
Epoch 8/10
60000/60000 [==============================] - 3s 53us/step - loss: 0.1781
- acc: 0.9502 - val_loss: 0.1716 - val_acc: 0.9509
Epoch 9/10
60000/60000 [==============================] - 3s 52us/step - loss: 0.1665
- acc: 0.9541 - val_loss: 0.1624 - val_acc: 0.9533
Epoch 10/10
60000/60000 [==============================] - 3s 55us/step - loss: 0.1563
- acc: 0.9564 - val_loss: 0.1558 - val_acc: 0.9551
```

Out[12]:
```
<keras.callbacks.History at 0x7ff7347d0c18>
```

Obtain model gradients

In [1]:
```
import keras.backend as K

# function to obtain grads for each parameter
def get_gradients(model, inputs, outputs):
  grads = model.optimizer.get_gradients(model.total_loss, model.trainable_w
  symb_inputs = (model._feed_inputs + model._feed_targets + model._feed_sam
  f = K.function(symb_inputs, grads)
  x, y, weight = model._standardize_user_data(inputs, outputs)
  output_grad = f(x + y + weight)
  return np.array(output_grad)
```

```
Using TensorFlow backend.
```

In [23]:
```
# test
grads = get_gradients(model, x_train[0:1], y_train[0:1])
```

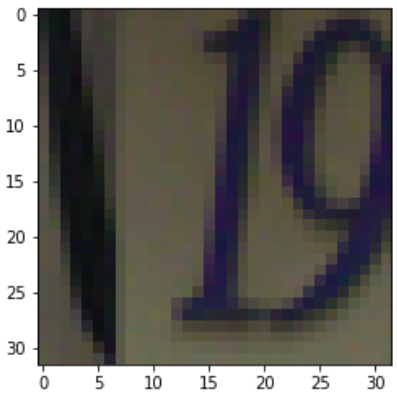
In [33]:
```
# Print max from each layer, weight and biases. Even indices hold biases
print(grads.shape)
for i,_ in enumerate(grads):
    print(grads[i].shape)
    max_gradient_layer_i = np.max(grads[i])
    print(max_gradient_layer_i)
```

```
(4,)
(784, 256)
0.0068208314
(256,)
0.0068208314
(256, 10)
0.034386244
(10,)
0.011868487
```

Load .mat files

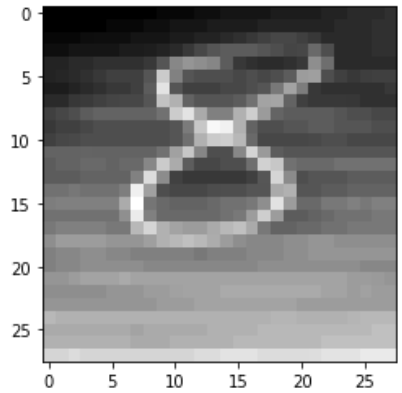
In [50]:
```
from scipy.io import loadmat

SVHN_directory = "train_32x32.mat"
# load .mat file
data_raw = loadmat(SVHN_directory)
data = np.array(data_raw['X'])
# make correct shape
data = np.moveaxis(data, -1, 0)
print(data.shape)
plt.imshow(data[0])
labels = data_raw['y']
print(labels.shape)
print(labels[0])
```

```
(73257, 32, 32, 3)
(73257, 1)
[1]
```



Make your own digits

In [96]:
```
import cv2 as cv
from skimage.transform import resize

image_dir = "test_digit2.jpg"
# load & smoothen image
kernel = np.ones((7,7),np.float32)/49
image = cv.imread(image_dir,cv.IMREAD_GRAYSCALE)
image = cv.filter2D(image,-1,kernel)

# make numpy array
image = np.array(image)
image = resize(image, (28,28))
# make negative
image = np.ones(image.shape) - image

plt.imshow(image, cmap="gray")
plt.show()
```



In [97]:
```
print(image.shape)
# predict label
np.argmax(model.predict(np.array([image])))
```

```
(28, 28)
```

Out[97]: 8

In [ ]: