Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 22

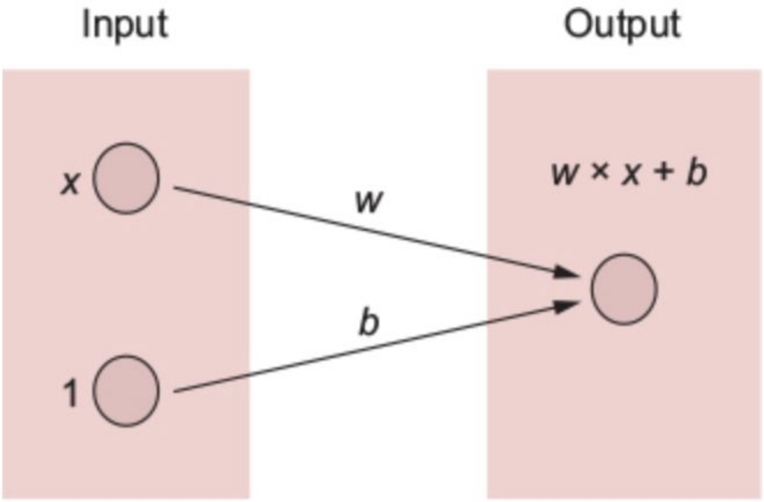
A peek into autoencoders

Overview

- Getting to know neural networks
- Designing autoencoders
- Representing images by using an autoencoder

Neural networks

A graphical representation of the linear equation $f(x) = w \times x + b$. The nodes are represented as circles, and edges are represented as arrows. The values on the edges are often called weights, and they act as a multiplication on the input. When two arrows lead to the same node, they act as a summation of the inputs.



Neural networks

Exercise 1

Is $f(x) = |x|$ a linear function?

Neural networks

Exercise 1

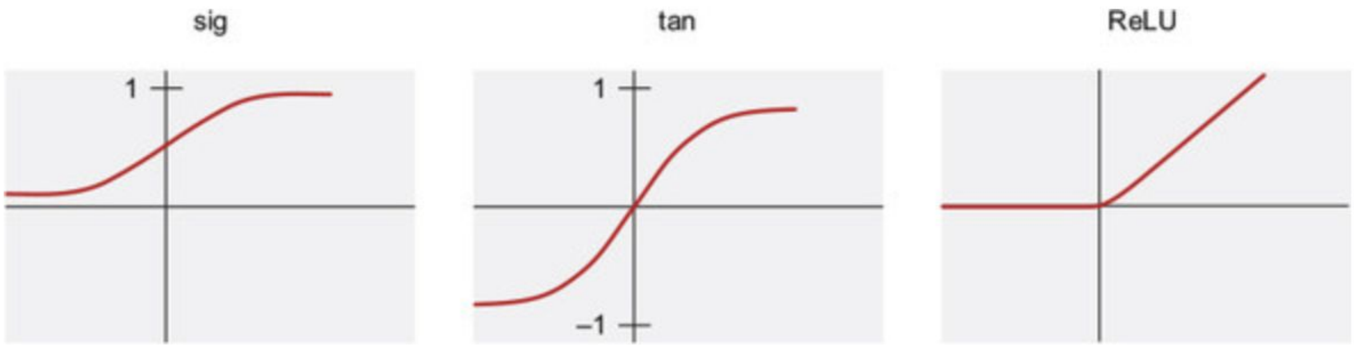
Is $f(x) = |x|$ a linear function?

ANSWER

No. It's two linear functions stitched together at zero, and that's not a single straight line.

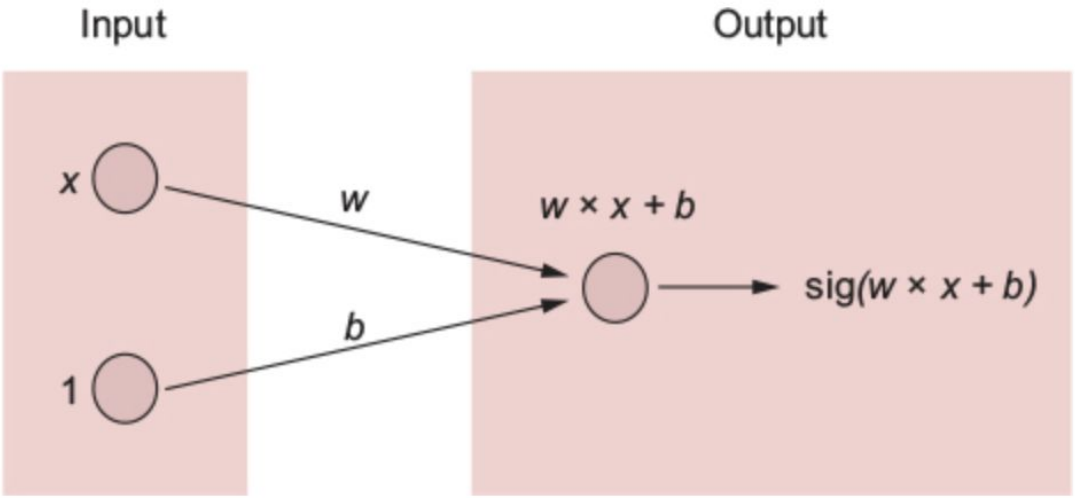
Neural networks

Use nonlinear functions such as sig, tan, and ReLU to introduce nonlinearity to your models.



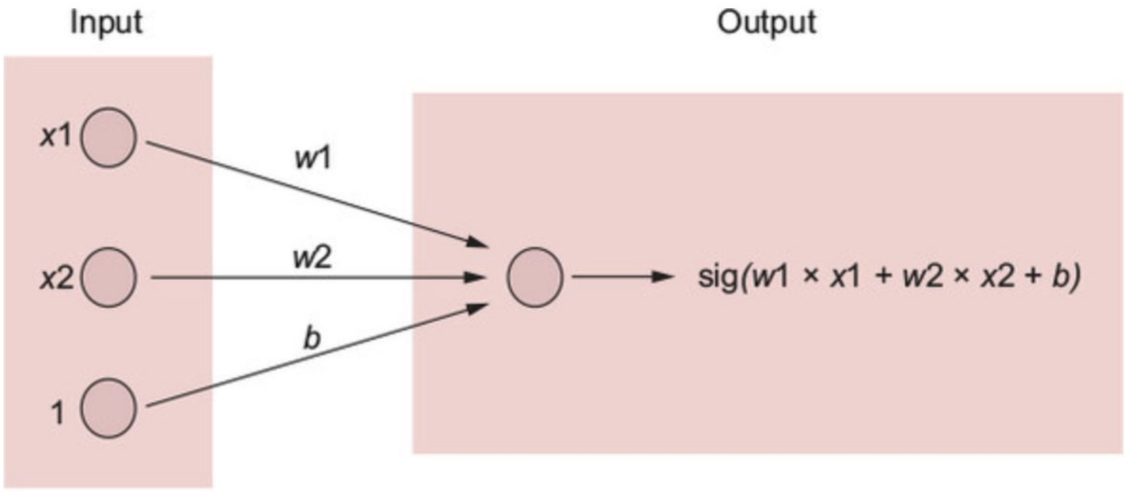
Neural networks

A nonlinear function, such as sigmoid, is applied to the output of a node.



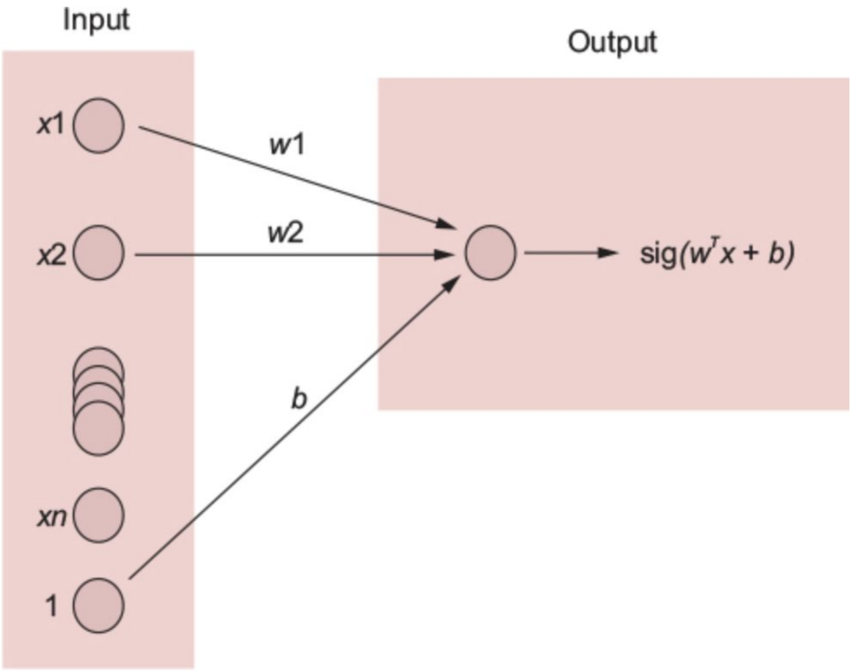
Neural networks

A two-input network will have three parameters (w1, w2, and b). Remember, multiple lines leading to the same node indicate summation.



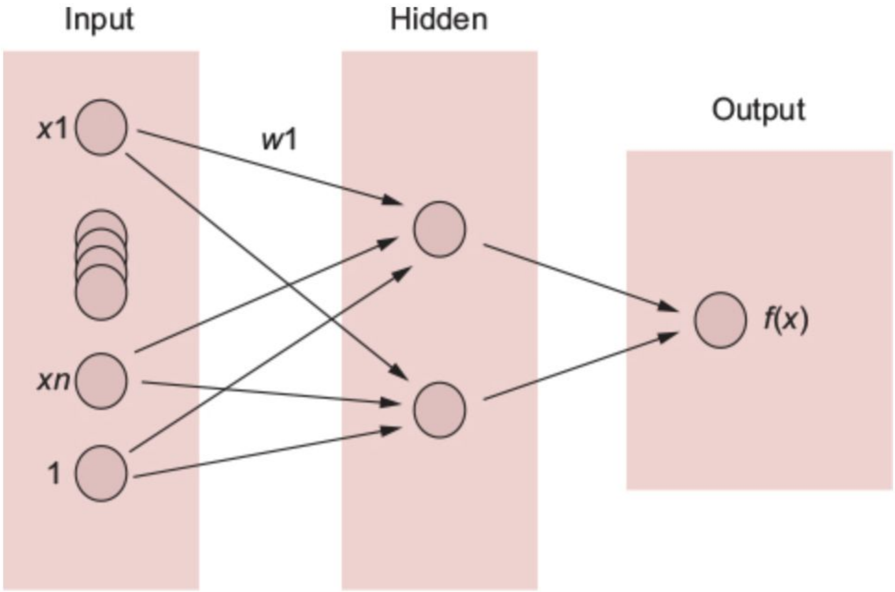
Neural networks

The input dimension can be arbitrarily long. For example, each pixel in a grayscale image can have a corresponding input xi. This neural network uses all inputs to generate a single output number, which you might use for regression or classification. The notation wT means you're transposing w, which is an n × 1 vector, into a 1 × n vector. That way, you can properly multiply it with x (which has the dimensions n × 1). Such a matrix multiplication is also called a dot product, and it yields a scalar (one-dimensional) value.



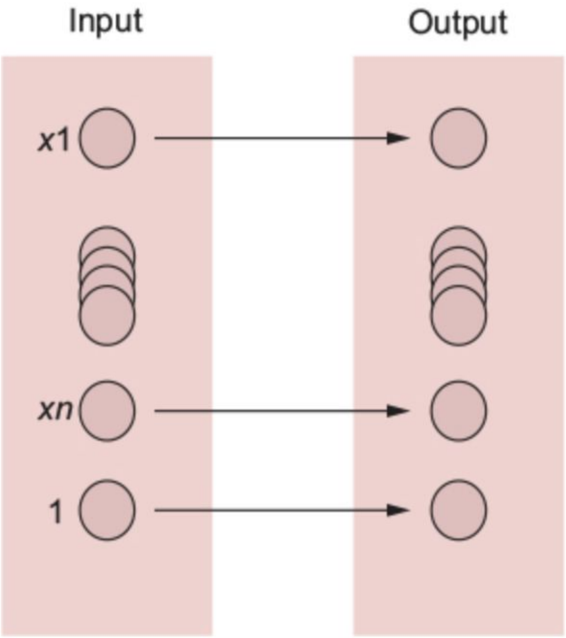
Neural networks

Nodes that don't interface to both the input and the output are called hidden neurons. A hidden layer is a collection of hidden units that aren't connected to each other.



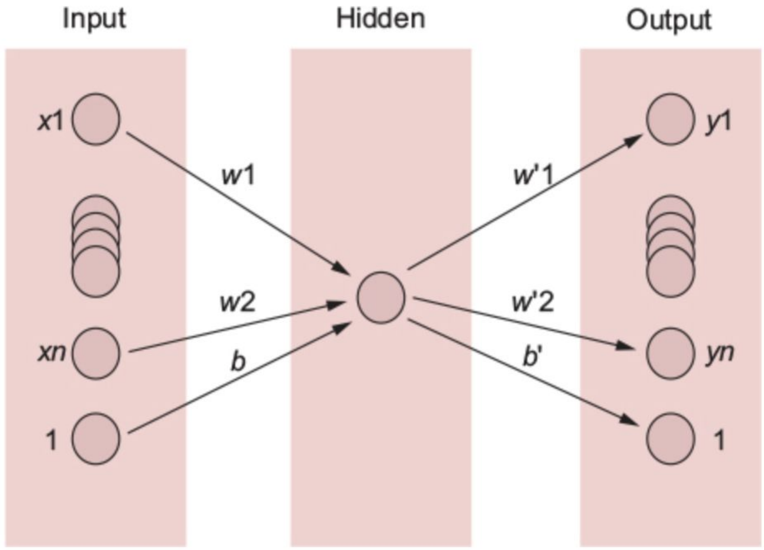
Autoencoders

If you want to create a network where the input equals the output, you can connect the corresponding nodes and set each parameter's weight to 1.



Autoencoders

Here, you introduce a restriction to a network that tries to reconstruct its input. Data will pass through a narrow channel, as illustrated by the hidden layer. In this example, there’s only one node in the hidden layer. This network is trying to encode (and decode) an n-dimensional input signal into just one dimension, which will likely be difficult in practice.



Autoencoders

Exercise 2

Let x denote the input vector (x1, x2, ..., xn), and let y denote the output vector (y1, y2, ..., yn). Lastly, let w and w' denote the encoder and decoder weights, respectively. What’s a possible cost function to train this neural network?

Autoencoders

Exercise 2

Let x denote the input vector (x1, x2, ..., xn), and let y denote the output vector (y1, y2, ..., yn). Lastly, let w and w' denote the encoder and decoder weights, respectively. What's a possible cost function to train this neural network?

ANSWER

See the loss function in listing 7.3.

Autoencoders

Python class schema

```
class Autoencoder:
    def __init__(self, input_dim, hidden_dim):    1

    def train(self, data):                         2

    def test(self, data):                          3
```

1 Initializes variables
2 Trains on a dataset
3 Tests on some new data

Autoencoders

Using name scopes

```
with tf.name_scope('encode'):
    weights = tf.Variable(tf.random_normal([input_dim,
hidden_dim],
     dtype=tf.float32), name='weights')
    biases = tf.Variable(tf.zeros([hidden_dim]), name='biases')
```

Autoencoders

Autoencoder class

```
import tensorflow as tf
import numpy as np

class Autoencoder:
    def __init__(self, input_dim, hidden_dim, epoch=250,
     learning_rate=0.001):
        self.epoch = epoch                                   1
        self.learning_rate = learning_rate                          2

        x = tf.placeholder(dtype=tf.float32, shape=[None,
input_dim])     3
```

1 Number of learning cycles
2 Hyperparameter of the optimizer
3 Defines the input layer dataset

Autoencoders

Autoencoder class

```
        with tf.name_scope('encode'):                                    4
            weights = tf.Variable(tf.random_normal([input_dim,
hidden_dim],
     dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([hidden_dim]),
name='biases')
            encoded = tf.nn.tanh(tf.matmul(x, weights) + biases)
        with tf.name_scope('decode'):                                    5
            weights = tf.Variable(tf.random_normal([hidden_dim,
input_dim],
     dtype=tf.float32), name='weights')
            biases = tf.Variable(tf.zeros([input_dim]),
name='biases')
            decoded = tf.matmul(encoded, weights) + biases
```

4 Defines the weights and biases under a name scope so you can tell them apart from the decoder’s weights and biases
5 The decoder’s weights and biases are defined under this name scope.

Autoencoders

Autoencoder class

```
        self.x = x                                              6
        self.encoded = encoded                                          6
        self.decoded = decoded                                          6

        self.loss =
tf.sqrt(tf.reduce_mean(tf.square(tf.subtract(self.x,
    self.decoded))))                                               7
        self.train_op =

tf.train.RMSPropOptimizer(self.learning_rate).minimize(self.l
oss)    8
        self.saver = tf.train.Saver()                                   9
```

6 These will be method variables.
7 Defines the reconstruction cost
8 Chooses the optimizer
9 Sets up a saver to save model parameters as they're being learned

Autoencoders

Training the autoencoder

```
def train(self, data):
    num_samples = len(data)
    with tf.Session() as sess:                              1
        sess.run(tf.global_variables_initializer())         1
        for i in range(self.epoch):                         2
            for j in range(num_samples):                    3
                l, _ = sess.run([self.loss, self.train_op], 3
                feed_dict={self.x: [data[j]]})              3
            if i % 10 == 0:                                 4
                print('epoch {0}: loss = {1}'.format(i, l)) 4
                self.saver.save(sess, './model.ckpt')       5
        self.saver.save(sess, './model.ckpt')               5
```

1 Starts a TensorFlow session, and initializes all variables
2 Iterates through the number of cycles defined in the constructor
3 One sample at a time, trains the neural network on a data item
4 Prints the reconstruction error once every 10 cycles
5 Saves the learned parameters to file

Autoencoders

Testing the model on data

```
    def test(self, data):
        with tf.Session() as sess:
            self.saver.restore(sess, './model.ckpt')
1
            hidden, reconstructed = sess.run([self.encoded,
self.decoded],
     feed_dict={self.x: data})                                    2
        print('input', data)
        print('compressed', hidden)
        print('reconstructed', reconstructed)
        return reconstructed
```

1 Loads the learned parameters
2 Reconstructs the input

Autoencoders

Using your Autoencoder class

```
from autoencoder import Autoencoder
from sklearn import datasets

hidden_dim = 1
data = datasets.load_iris().data
input_dim = len(data[0])
ae = Autoencoder(input_dim, hidden_dim)
ae.train(data)
ae.test([[8, 4, 6, 2]])
```

Autoencoders

The test function shows info about the encoding and decoding process:

```
('input', [[8, 4, 6, 2]])
('compressed', array([[ 0.78238308]], dtype=float32))
('reconstructed', array([[ 6.87756062,  2.79838109,
6.25144577,
     2.23120356]], dtype=float32))
```

Batch training

Batch helper function

```
def get_batch(X, size):
    a = np.random.choice(len(X), size, replace=False)
    return X[a]
```

Batch training

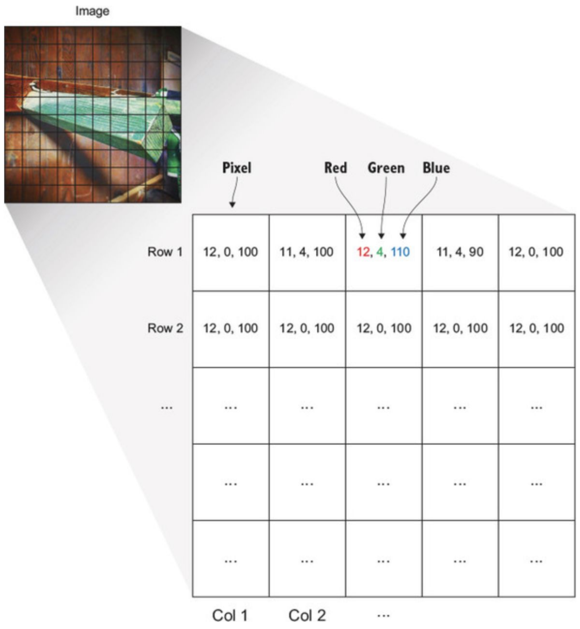
Batch learning

```
def train(self, data, batch_size=10):
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            for i in range(self.epoch):
                for j in range(500):                                   1
                    batch_data = get_batch(data, self.batch_size)
2
                    l, _ = sess.run([self.loss, self.train_op],
     feed_dict={self.x: batch_data})
                if i % 10 == 0:
                    print('epoch {0}: loss = {1}'.format(i, l))
                    self.saver.save(sess, './model.ckpt')
            self.saver.save(sess, './model.ckpt')
```

1 Loops through various batch selections
2 Runs the optimizer on a randomly selected batch

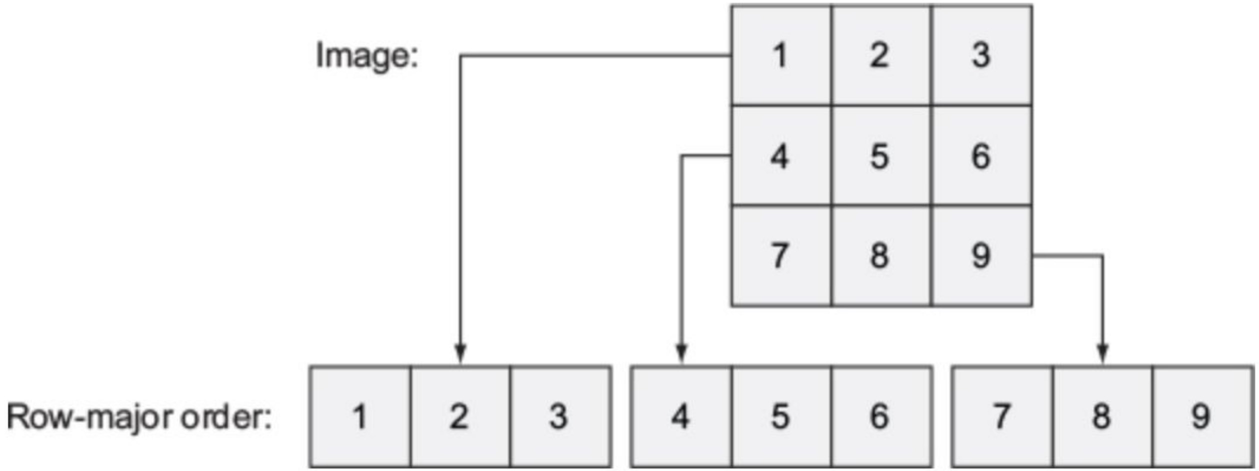
Working with images

A colored image is composed of pixels, and each pixel contains values for red, green, and blue.



Working with images

An image can be represented in row-major order. That way, you can represent a two-dimensional structure as a one dimensional structure.



Working with images

Loading images

```
from scipy.misc import imread, imresize

gray_image = imread(filepath, True)                  1
small_gray_image = imresize(gray_image, 1. / 8.)    2
x = small_gray_image.flatten()                       3
```

1 Loads an image as grayscale
2 Resizes it to something smaller
3 Converts it to a one-dimensional structure

Working with images

Exercise 3

Can you name other online image datasets? Search online and look around for more!

Working with images

Exercise 3

Can you name other online image datasets? Search online and look around for more!

ANSWER

Perhaps the most used in the deep-learning community is ImageNet (www.image-net.org). A great list can also be found online at http://deeplearning.net/datasets.

Working with images

Reading from the extracted CIFAR-10 dataset

```
import pickle

def unpickle(file):                                    1
    fo = open(file, 'rb')
    dict = pickle.load(fo, encoding='latin1')
    fo.close()
    return dict
```

1 Reads the CIFAR-10 file, returning the loaded dictionary

Working with images

Reading all CIFAR-10 files to memory

```
import numpy as np

names =
unpickle('./cifar-10-batches-py/batches.meta')['label_names']
data, labels = [], []
for i in range(1, 6):                                          1
    filename = './cifar-10-batches-py/data_batch_' + str(i)
    batch_data = unpickle(filename)                            2
    if len(data) > 0:
        data = np.vstack((data, batch_data['data']))           3
        labels = np.hstack((labels, batch_data['labels']))     4
    else:
        data = batch_data['data']
        labels = batch_data['labels']
```

1 Loops through the six files
2 Loads the file to obtain a Python dictionary
3 The rows of a data sample represent each sample, so you stack it vertically.
4 Labels are one-dimensional, so you stack them horizontally.

Working with images

Converting CIFAR-10 image to grayscale

```
def grayscale(a):
    return a.reshape(a.shape[0], 3, 32, 32).mean(1).reshape(a.shape[0], -1)

data = grayscale(data)
```

Working with images

Setting up the autoencoder

```
from autoencoder import Autoencoder

x = np.matrix(data)
y = np.array(labels)

horse_indices = np.where(y == 7)[0]

horse_x = x[horse_indices]

print(np.shape(horse_x))  # (5000, 3072)

input_dim = np.shape(horse_x)[1]
hidden_dim = 100
ae = Autoencoder(input_dim, hidden_dim)
ae.train(horse_x)
```

Working with images

The output will trace loss values of every 10 epochs:

```
epoch 0: loss = 99.8635025024
epoch 10: loss = 35.3869667053
epoch 20: loss = 15.9411172867
epoch 30: loss = 7.66391372681
epoch 40: loss = 1.39575612545
epoch 50: loss = 0.00389165547676
epoch 60: loss = 0.00203850422986
epoch 70: loss = 0.00186171964742
epoch 80: loss = 0.00231492402963
epoch 90: loss = 0.00166488380637
epoch 100: loss = 0.00172081717756
epoch 110: loss = 0.0018497039564
epoch 120: loss = 0.00220602494664
epoch 130: loss = 0.00179589167237
epoch 140: loss = 0.00122790911701
epoch 150: loss = 0.0027100709267
epoch 160: loss = 0.00213225837797
epoch 170: loss = 0.00215123943053
epoch 180: loss = 0.00148373935372
epoch 190: loss = 0.00171591725666
```

Application of autoencoders

- A stacked autoencoder
- A denoising autoencoder
- A variational autoencoder