In [1]:

```
%load_ext watermark
%watermark -a 'Sebastian Raschka' -u -d -v -p numpy,matplotlib,theano,keras
```

```
UsageError: unrecognized arguments: Raschka'
```

In [2]:

```
from IPython.display import Image
```

In [3]:

```
%matplotlib inline
```

In [4]:

```
Image(filename='13_01.png', width=500)
```

Out[4]:

Specifications	Intel® Core™ i7-5960X Processor Extreme Edition	NVIDIA GeForce® GTX™ 980 Ti
Base Clock Frequency	3.0 GHz	1.0 GHz
Cores	8	2816
Memory Bandwidth	68 GB/s	336.5 GB/s
Floating-Point Calculations	354 GFLOPS	5632 GFLOPS
Cost	$1000.00	$700.00

In [5]:

```
import theano
from theano import tensor as T
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install
m2w64-toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execu
te optimized C-implementations (for both CPU and GPU) and will default to Python i
mplementations. Performance will be severely degraded. To remove this warning, set
Theano flags cxx to an empty string.
```

In [6]:

```
# initialize
x1 = T.scalar()
w1 = T.scalar()
w0 = T.scalar()
z1 = w1 * x1 + w0

# compile
net_input = theano.function(inputs=[w1, x1, w0], outputs=z1)

# execute
net_input(2.0, 1.0, 0.5)
```

Out[6]:

```
array(2.5)
```

In [7]:

```
print(theano.config.floatX)
```

```
float64
```

In [8]:

```
theano.config.floatX = 'float32'
```

In [9]:

```
print(theano.config.device)
```

```
cpu
```

In [10]:

```
import numpy as np

# initialize
# if you are running Theano on 64 bit mode,
# you need to use dmatrix instead of fmatrix
x = T.fmatrix(name='x')
x_sum = T.sum(x, axis=0)

# compile
calc_sum = theano.function(inputs=[x], outputs=x_sum)

# execute (Python list)
ary = [[1, 2, 3], [1, 2, 3]]
print('Column sum:', calc_sum(ary))

# execute (NumPy array)
ary = np.array([[1, 2, 3], [1, 2, 3]], dtype=theano.config.floatX)
print('Column sum:', calc_sum(ary))
```

```
Column sum: [ 2.  4.  6.]
Column sum: [ 2.  4.  6.]
```

In [11]:

```
# initialize
x = T.fmatrix(name='x')
w = theano.shared(np.asarray([[0.0, 0.0, 0.0]],
                             dtype=theano.config.floatX))
z = x.dot(w.T)
update = [[w, w + 1.0]]

# compile
net_input = theano.function(inputs=[x],
                            updates=update,
                            outputs=z)

# execute
data = np.array([[1, 2, 3]], dtype=theano.config.floatX)
for i in range(5):
    print('z%d:' % i, net_input(data))
```

```
z0: [[ 0.]]
z1: [[ 6.]]
z2: [[ 12.]]
z3: [[ 18.]]
z4: [[ 24.]]
```

In [12]:

```
# initialize
data = np.array([[1, 2, 3]],
                dtype=theano.config.floatX)
x = T.fmatrix(name='x')
w = theano.shared(np.asarray([[0.0, 0.0, 0.0]],
                             dtype=theano.config.floatX))
z = x.dot(w.T)
update = [[w, w + 1.0]]

# compile
net_input = theano.function(inputs=[],
                            updates=update,
                            givens={x: data},
                            outputs=z)

# execute
for i in range(5):
    print('z:', net_input())
```

```
z: [[ 0.]]
z: [[ 6.]]
z: [[ 12.]]
z: [[ 18.]]
z: [[ 24.]]
```

In [13]:

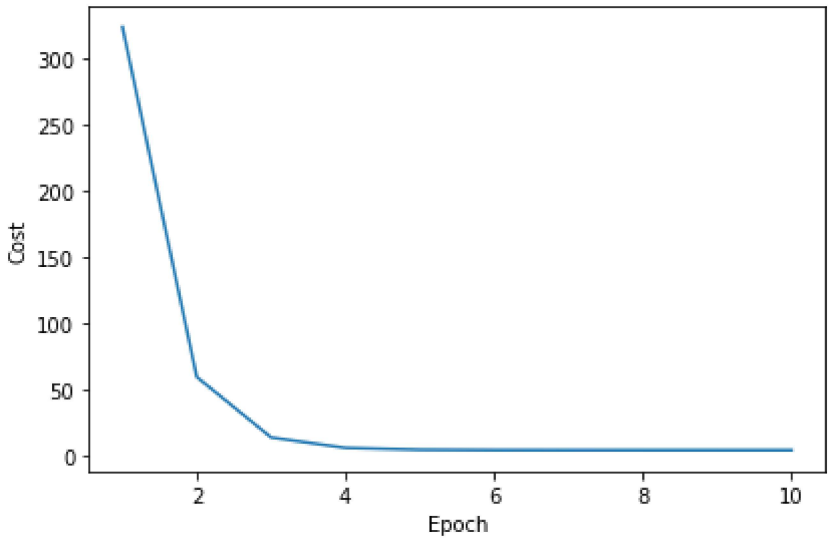
```
import numpy as np
X_train = np.asarray([[0.0], [1.0], [2.0], [3.0], [4.0],
                      [5.0], [6.0], [7.0], [8.0], [9.0]],
                     dtype=theano.config.floatX)

y_train = np.asarray([1.0, 1.3, 3.1, 2.0, 5.0,
                      6.3, 6.6, 7.4, 8.0, 9.0],
                     dtype=theano.config.floatX)
```

In [14]:

```
import theano
from theano import tensor as T
import numpy as np

def train_linreg(X_train, y_train, eta, epochs):

    costs = []
    # Initialize arrays
    eta0 = T.fscalar('eta0')
    y = T.fvector(name='y')
    X = T.fmatrix(name='X')
    w = theano.shared(np.zeros(
                      shape=(X_train.shape[1] + 1),
                      dtype=theano.config.floatX),
                      name='w')

    # calculate cost
    net_input = T.dot(X, w[1:]) + w[0]
    errors = y - net_input
    cost = T.sum(T.pow(errors, 2))

    # perform gradient update
    gradient = T.grad(cost, wrt=w)
    update = [(w, w - eta0 * gradient)]

    # compile model
    train = theano.function(inputs=[eta0],
                            outputs=cost,
                            updates=update,
                            givens={X: X_train,
                                    y: y_train})

    for _ in range(epochs):
        costs.append(train(eta))

    return costs, w
```

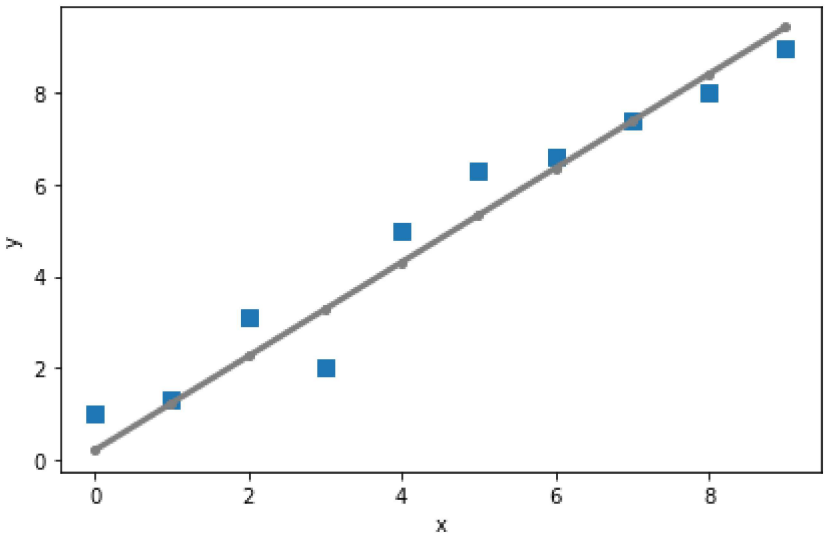
In [15]:

```
import matplotlib.pyplot as plt

costs, w = train_linreg(X_train, y_train, eta=0.001, epochs=10)

plt.plot(range(1, len(costs)+1), costs)

plt.tight_layout()
plt.xlabel('Epoch')
plt.ylabel('Cost')
plt.tight_layout()
# plt.savefig('./figures/cost_convergence.png', dpi=300)
plt.show()
```



In [16]:

```
def predict_linreg(X, w):
    Xt = T.matrix(name='X')
    net_input = T.dot(Xt, w[1:]) + w[0]
    predict = theano.function(inputs=[Xt], givens={w: w}, outputs=net_input)
    return predict(X)

plt.scatter(X_train, y_train, marker='s', s=50)
plt.plot(range(X_train.shape[0]),
         predict_linreg(X_train, w),
         color='gray',
         marker='o',
         markersize=4,
         linewidth=3)

plt.xlabel('x')
plt.ylabel('y')

plt.tight_layout()
# plt.savefig('./figures/linreg.png', dpi=300)
plt.show()
```



In [17]:

```
# note that first element (X[0] = 1) to denote bias unit

X = np.array([[1, 1.4, 1.5]])
w = np.array([0.0, 0.2, 0.4])

def net_input(X, w):
    z = X.dot(w)
    return z

def logistic(z):
    return 1.0 / (1.0 + np.exp(-z))

def logistic_activation(X, w):
    z = net_input(X, w)
    return logistic(z)

print('P(y=1|x) = %.3f' % logistic_activation(X, w)[0])
```

```
P(y=1|x) = 0.707
```

In [18]:

```
# W : array, shape = [n_output_units, n_hidden_units+1]
#          Weight matrix for hidden layer -> output layer.
# note that first column (W[:][0]) contains the bias units
W = np.array([[1.1, 1.2, 1.3, 0.5],
              [0.1, 0.2, 0.4, 0.1],
              [0.2, 0.5, 2.1, 1.9]])

# A : array, shape = [n_hidden+1, n_samples]
#          Activation of hidden layer.
# note that first element (A[0][0] = 1) is for the bias units

A = np.array([[1.0],
              [0.1],
              [0.3],
              [0.7]])

# Z : array, shape = [n_output_units, n_samples]
#          Net input of output layer.

Z = W.dot(A)
y_probas = logistic(Z)
print('Probabilities:¥n', y_probas)
```

```
Probabilities:
 [[ 0.87653295]
 [ 0.57688526]
 [ 0.90114393]]
```

In [19]:

```
y_class = np.argmax(Z, axis=0)
print('predicted class label: %d' % y_class[0])
```

```
predicted class label: 2
```

In [20]:

```
def softmax(z):
    return np.exp(z) / np.sum(np.exp(z))

def softmax_activation(X, w):
    z = net_input(X, w)
    return softmax(z)
```

In [21]:

```
y_probas = softmax(Z)
print('Probabilities:¥n', y_probas)
```

```
Probabilities:
 [[ 0.40386493]
 [ 0.07756222]
 [ 0.51857284]]
```

In [22]:

```
y_probas.sum()
```

Out[22]:

1.0

In [23]:

```
y_class = np.argmax(Z, axis=0)
y_class
```

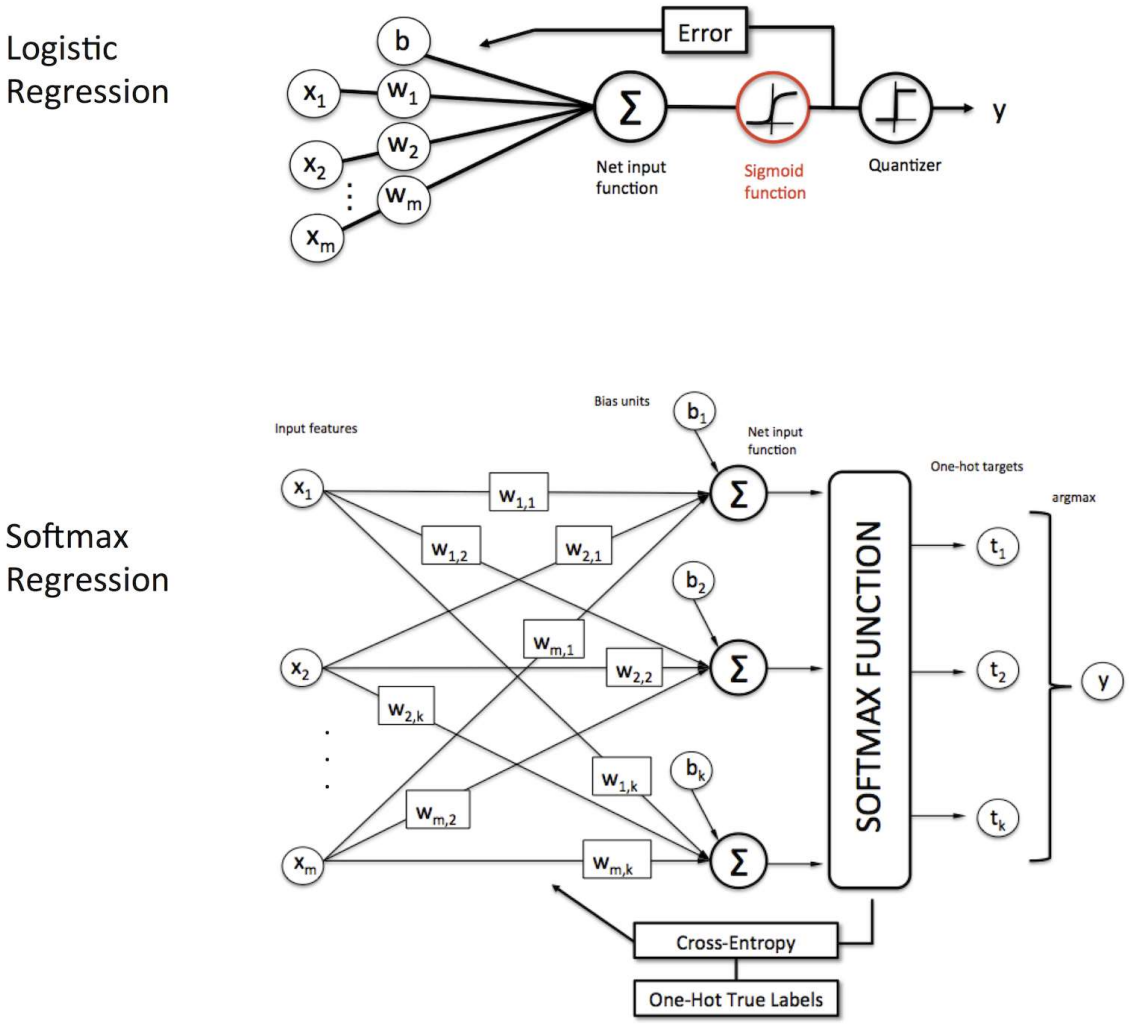
Out[23]:

array([2], dtype=int64)

In [24]:

```
Image(filename='bonus_softmax_1.png', width=800)
```

Out[24]:

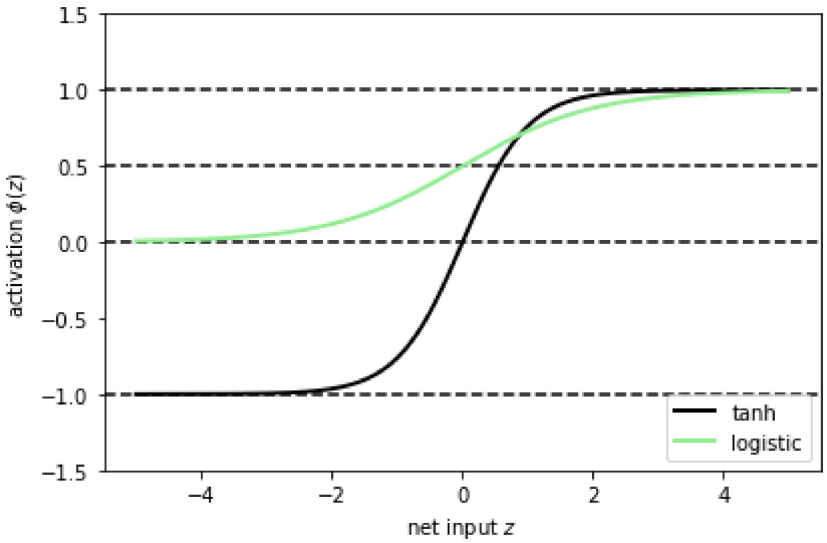


In [25]:

```
def tanh(z):
    e_p = np.exp(z)
    e_m = np.exp(-z)
    return (e_p - e_m) / (e_p + e_m)
```

In [26]:

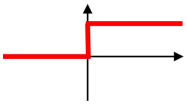
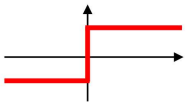
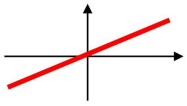
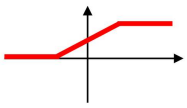
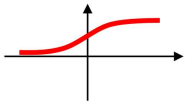
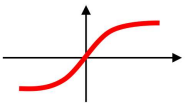
```
import matplotlib.pyplot as plt

z = np.arange(-5, 5, 0.005)
log_act = logistic(z)
tanh_act = tanh(z)

# alternatives:
# from scipy.special import expit
# log_act = expit(z)
# tanh_act = np.tanh(z)

plt.ylim([-1.5, 1.5])
plt.xlabel('net input $z$')
plt.ylabel('activation $¥phi(z)$')
plt.axhline(1, color='black', linestyle='--')
plt.axhline(0.5, color='black', linestyle='--')
plt.axhline(0, color='black', linestyle='--')
plt.axhline(-1, color='black', linestyle='--')

plt.plot(z, tanh_act,
         linewidth=2,
         color='black',
         label='tanh')
plt.plot(z, log_act,
         linewidth=2,
         color='lightgreen',
         label='logistic')

plt.legend(loc='lower right')
plt.tight_layout()
# plt.savefig('./figures/activation.png', dpi=300)
plt.show()
```



In [27]:

```
Image(filename='13_05.png', width=700)
```

Out[27]:

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

In [28]:

```
import os
import struct
import numpy as np

def load_mnist(path, kind='train'):
    """Load MNIST data from `path`"""
    labels_path = os.path.join(path,
                               '%s-labels-idx1-ubyte' % kind)
    images_path = os.path.join(path,
                               '%s-images-idx3-ubyte' % kind)

    with open(labels_path, 'rb') as lbpath:
        magic, n = struct.unpack('>II',
                                 lbpath.read(8))
        labels = np.fromfile(lbpath,
                             dtype=np.uint8)

    with open(images_path, 'rb') as imgpath:
        magic, num, rows, cols = struct.unpack(">IIII",
                                               imgpath.read(16))
        images = np.fromfile(imgpath,
                             dtype=np.uint8).reshape(len(labels), 784)

    return images, labels
```

In [29]:

```
X_train, y_train = load_mnist('./mnist', kind='train')
print('Rows: %d, columns: %d' % (X_train.shape[0], X_train.shape[1]))
```

```
Rows: 60000, columns: 784
```

In [30]:

```
X_test, y_test = load_mnist('mnist', kind='t10k')
print('Rows: %d, columns: %d' % (X_test.shape[0], X_test.shape[1]))
```

```
Rows: 10000, columns: 784
```

In [31]:

```
import theano

theano.config.floatX = 'float32'
X_train = X_train.astype(theano.config.floatX)
X_test = X_test.astype(theano.config.floatX)
```

In [32]:

```
from keras.utils import np_utils

print('First 3 labels: ', y_train[:3])

y_train_ohe = np_utils.to_categorical(y_train)
print('¥nFirst 3 labels (one-hot):¥n', y_train_ohe[:3])
```

```
Using TensorFlow backend.

First 3 labels:  [5 0 4]

First 3 labels (one-hot):
 [[ 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.]]
```

In [33]:

```
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers import SGD

np.random.seed(1)

model = Sequential()
model.add(Dense(input_dim=X_train.shape[1],
                units=50, # formerly output_dim=50 in Keras < 2
                kernel_initializer='uniform', # formerly init='uniform' in Keras < 2
                activation='tanh'))

model.add(Dense(input_dim=50,
                units=50,
                kernel_initializer='uniform',
                activation='tanh'))

model.add(Dense(input_dim=50,
                units=y_train_ohe.shape[1],
                kernel_initializer='uniform',
                activation='softmax'))

sgd = SGD(lr=0.001, decay=1e-7, momentum=.9)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(X_train, y_train_ohe,
          epochs=50, # Keras 2: former nb_epoch has been renamed to epochs
          batch_size=300,
          verbose=1,
          validation_split=0.1)
          # removed former show_accuracy=True
          # and added `metrics=['accuracy']` to the
          # model.compile call for Keras >= 2
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/50
54000/54000 [==============================] - 1s - loss: 2.2208 - acc: 0.4288 - v
al_loss: 2.0864 - val_acc: 0.6262
Epoch 2/50
54000/54000 [==============================] - 0s - loss: 1.8306 - acc: 0.6191 - v
al_loss: 1.5089 - val_acc: 0.6548
Epoch 3/50
54000/54000 [==============================] - 0s - loss: 1.2994 - acc: 0.6576 - v
al_loss: 1.0746 - val_acc: 0.7275
Epoch 4/50
54000/54000 [==============================] - 0s - loss: 0.9776 - acc: 0.7511 - v
al_loss: 0.8201 - val_acc: 0.8123
Epoch 5/50
54000/54000 [==============================] - 1s - loss: 0.7758 - acc: 0.8088 - v
al_loss: 0.6507 - val_acc: 0.8635
Epoch 6/50
54000/54000 [==============================] - 1s - loss: 0.6418 - acc: 0.8496 - v
al_loss: 0.5404 - val_acc: 0.8802
Epoch 7/50
54000/54000 [==============================] - 0s - loss: 0.5549 - acc: 0.8693 - v
al_loss: 0.4685 - val_acc: 0.8975
Epoch 8/50
54000/54000 [==============================] - 0s - loss: 0.4964 - acc: 0.8797 - v
al_loss: 0.4196 - val_acc: 0.9052
Epoch 9/50
54000/54000 [==============================] - 0s - loss: 0.4483 - acc: 0.8883 - v
al_loss: 0.3824 - val_acc: 0.9080
Epoch 10/50
54000/54000 [==============================] - 1s - loss: 0.4160 - acc: 0.8944 - v
al_loss: 0.3573 - val_acc: 0.9127
Epoch 11/50
54000/54000 [==============================] - 0s - loss: 0.3925 - acc: 0.8973 - v
al_loss: 0.3406 - val_acc: 0.9143
Epoch 12/50
54000/54000 [==============================] - 0s - loss: 0.3719 - acc: 0.9031 - v
al_loss: 0.3139 - val_acc: 0.9215
Epoch 13/50
54000/54000 [==============================] - 1s - loss: 0.3544 - acc: 0.9052 - v
al_loss: 0.3097 - val_acc: 0.9235
Epoch 14/50
54000/54000 [==============================] - 0s - loss: 0.3445 - acc: 0.9083 - v
al_loss: 0.2925 - val_acc: 0.9263
Epoch 15/50
54000/54000 [==============================] - 0s - loss: 0.3298 - acc: 0.9116 - v
al_loss: 0.2873 - val_acc: 0.9242
Epoch 16/50
54000/54000 [==============================] - 0s - loss: 0.3229 - acc: 0.9114 - v
al_loss: 0.2750 - val_acc: 0.9268
Epoch 17/50
54000/54000 [==============================] - 1s - loss: 0.3097 - acc: 0.9167 - v
al_loss: 0.2616 - val_acc: 0.9313
Epoch 18/50
54000/54000 [==============================] - 0s - loss: 0.3002 - acc: 0.9189 - v
al_loss: 0.2701 - val_acc: 0.9312
Epoch 19/50
54000/54000 [==============================] - 0s - loss: 0.2909 - acc: 0.9210 - v
al_loss: 0.2473 - val_acc: 0.9332
Epoch 20/50
54000/54000 [==============================] - 1s - loss: 0.2797 - acc: 0.9233 - v
al_loss: 0.2400 - val_acc: 0.9352
```

```
Epoch 21/50
54000/54000 [==============================] - 0s - loss: 0.2782 - acc: 0.9221 - v
al_loss: 0.2376 - val_acc: 0.9358
Epoch 22/50
54000/54000 [==============================] - 1s - loss: 0.2784 - acc: 0.9220 - v
al_loss: 0.2401 - val_acc: 0.9380
Epoch 23/50
54000/54000 [==============================] - 0s - loss: 0.2662 - acc: 0.9251 - v
al_loss: 0.2208 - val_acc: 0.9388
Epoch 24/50
54000/54000 [==============================] - 1s - loss: 0.2651 - acc: 0.9259 - v
al_loss: 0.2339 - val_acc: 0.9375
Epoch 25/50
54000/54000 [==============================] - 1s - loss: 0.2644 - acc: 0.9246 - v
al_loss: 0.2218 - val_acc: 0.9415
Epoch 26/50
54000/54000 [==============================] - 0s - loss: 0.2559 - acc: 0.9289 - v
al_loss: 0.2135 - val_acc: 0.9418
Epoch 27/50
54000/54000 [==============================] - 0s - loss: 0.2538 - acc: 0.9288 - v
al_loss: 0.2262 - val_acc: 0.9398
Epoch 28/50
54000/54000 [==============================] - 0s - loss: 0.2501 - acc: 0.9293 - v
al_loss: 0.2207 - val_acc: 0.9385
Epoch 29/50
54000/54000 [==============================] - 0s - loss: 0.2451 - acc: 0.9303 - v
al_loss: 0.2041 - val_acc: 0.9448
Epoch 30/50
54000/54000 [==============================] - 1s - loss: 0.2372 - acc: 0.9325 - v
al_loss: 0.2168 - val_acc: 0.9382
Epoch 31/50
54000/54000 [==============================] - 1s - loss: 0.2304 - acc: 0.9353 - v
al_loss: 0.2052 - val_acc: 0.9422
Epoch 32/50
54000/54000 [==============================] - 0s - loss: 0.2439 - acc: 0.9304 - v
al_loss: 0.2158 - val_acc: 0.9413
Epoch 33/50
54000/54000 [==============================] - 0s - loss: 0.2357 - acc: 0.9327 - v
al_loss: 0.2016 - val_acc: 0.9442
Epoch 34/50
54000/54000 [==============================] - 1s - loss: 0.2242 - acc: 0.9363 - v
al_loss: 0.1929 - val_acc: 0.9498
Epoch 35/50
54000/54000 [==============================] - 1s - loss: 0.2270 - acc: 0.9350 - v
al_loss: 0.2020 - val_acc: 0.9478
Epoch 36/50
54000/54000 [==============================] - 0s - loss: 0.2251 - acc: 0.9353 - v
al_loss: 0.1937 - val_acc: 0.9465
Epoch 37/50
54000/54000 [==============================] - 1s - loss: 0.2206 - acc: 0.9366 - v
al_loss: 0.1936 - val_acc: 0.9483
Epoch 38/50
54000/54000 [==============================] - 1s - loss: 0.2168 - acc: 0.9375 - v
al_loss: 0.1859 - val_acc: 0.9503
Epoch 39/50
54000/54000 [==============================] - 1s - loss: 0.2181 - acc: 0.9373 - v
al_loss: 0.1889 - val_acc: 0.9485
Epoch 40/50
54000/54000 [==============================] - 1s - loss: 0.2215 - acc: 0.9345 - v
al_loss: 0.1891 - val_acc: 0.9475
Epoch 41/50
```