Chapter_13_Parallelizing_Neural_Network_Training_with_TensorFlow

March 20, 2024

0.1 TensorFlow and training performance

[4]:
```
from platform import python_version

print(python_version())
```

```
3.6.13
```

[9]:
```
# ! pip install tensorflow==1.3.0
```

[10]:
```
# import tensorflow as tf;
# print(tf.__version__)
```

[8]:
```
# ! pip install tensorflow-gpu
```

[11]:
```
import tensorflow as tf
## create a graph
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(dtype=tf.float32,shape=(None), name='x')
    w = tf.Variable(2.0, name='weight')
    b = tf.Variable(0.7, name='bias')
    z = w*x + b
    init = tf.global_variables_initializer()
## create a session and pass in graph g
with tf.Session(graph=g) as sess:
    ## initialize w and b:
    sess.run(init)
    ## evaluate z:
    for t in [1.0, 0.6, -1.8]:
        print('x=%4.1f --> z=%4.1f'%(t, sess.run(z, feed_dict={x:t})))
```

```
x= 1.0 --> z= 2.7
x= 0.6 --> z= 1.9
x=-1.8 --> z=-2.9
```

[12]:

```
with tf.Session(graph=g) as sess:
    sess.run(init)
    print(sess.run(z, feed_dict={x:[1., 2., 3.]}))
```

```
[2.7 4.7 6.7]
```

[13]:

```
import tensorflow as tf
import numpy as np
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(dtype=tf.float32,shape=(None, 2, 3),name='input_x')
    x2 = tf.reshape(x, shape=(-1, 6),name='x2')
    ## calculate the sum of each column
    xsum = tf.reduce_sum(x2, axis=0, name='col_sum')
    ## calculate the mean of each column
    xmean = tf.reduce_mean(x2, axis=0, name='col_mean')
with tf.Session(graph=g) as sess:
    x_array = np.arange(18).reshape(3, 2, 3)
    print('input shape: ', x_array.shape)
    print('Reshaped:\n',sess.run(x2, feed_dict={x:x_array}))
    print('Column Sums:\n',sess.run(xsum, feed_dict={x:x_array}))
    print('Column Means:\n',sess.run(xmean, feed_dict={x:x_array}))
```

```
input shape:  (3, 2, 3)
Reshaped:
 [[ 0.  1.  2.  3.  4.  5.]
 [ 6.  7.  8.  9. 10. 11.]
 [12. 13. 14. 15. 16. 17.]]
Column Sums:
 [18. 21. 24. 27. 30. 33.]
Column Means:
 [ 6.  7.  8.  9. 10. 11.]
```

[14]:

```
import tensorflow as tf
import numpy as np
X_train = np.arange(10).reshape((10, 1))
y_train = np.array([1.0, 1.3, 3.1,2.0, 5.0, 6.3,6.6, 7.4, 8.0,9.0])
```

[15]:

```
class TfLinreg(object):
    def __init__(self, x_dim, learning_rate=0.01,random_seed=None):
        self.x_dim = x_dim
        self.learning_rate = learning_rate
        self.g = tf.Graph()
        ## build the model
        with self.g.as_default():
            ## set graph-level random-seed
            tf.set_random_seed(random_seed)
```

```
            self.build()
            ## create initializer
            self.init_op = tf.global_variables_initializer()

    def build(self):
        ## define placeholders for inputs
        self.X = tf.placeholder(dtype=tf.float32,shape=(None, self.
↪x_dim),name='x_input')
        self.y = tf.placeholder(dtype=tf.float32,shape=(None),name='y_input')
        print(self.X)
        print(self.y)
        ## define weight matrix and bias vector
        w = tf.Variable(tf.zeros(shape=(1)),name='weight')
        b = tf.Variable(tf.zeros(shape=(1)),name="bias")
        print(w)
        print(b)
        self.z_net = tf.squeeze(w*self.X + b,name='z_net')
        print(self.z_net)
        sqr_errors = tf.square(self.y - self.z_net,name='sqr_errors')
        print(sqr_errors)
        self.mean_cost = tf.reduce_mean(sqr_errors,name='mean_cost')
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=self.
↪learning_rate,name='GradientDescent')
        self.optimizer = optimizer.minimize(self.mean_cost)
```

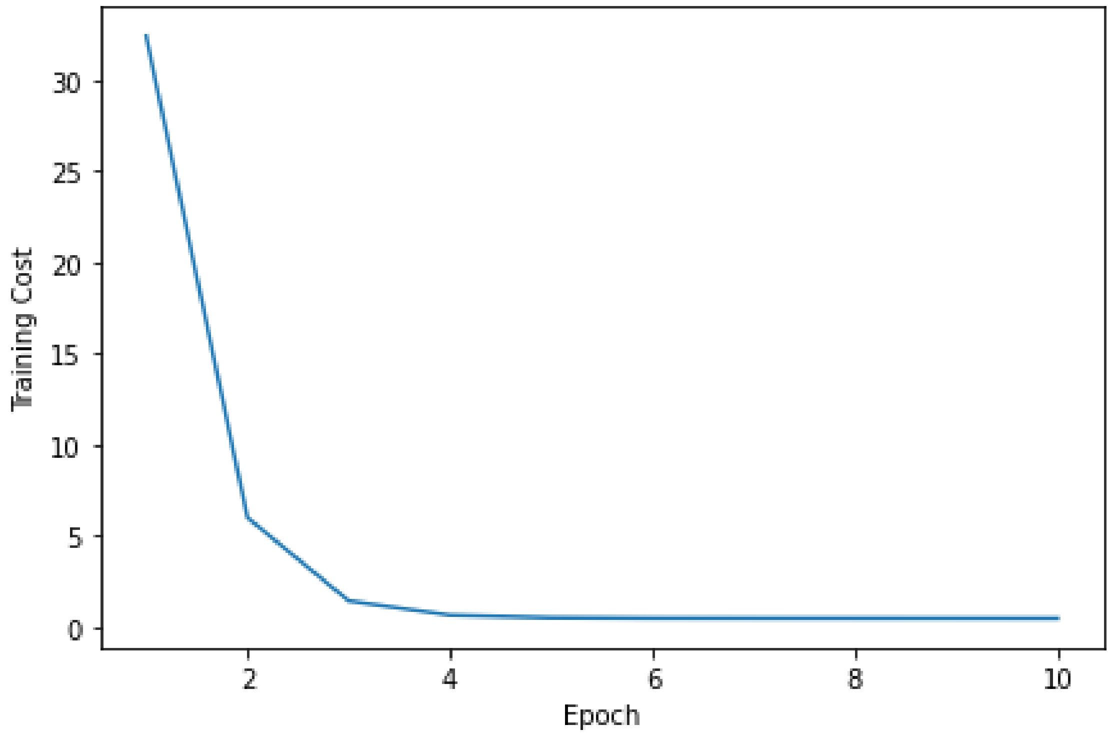
[16]:
```
lrmodel = TfLinreg(x_dim=X_train.shape[1], learning_rate=0.01)
```

```
Tensor("x_input:0", shape=(?, 1), dtype=float32)
Tensor("y_input:0", dtype=float32)
<tf.Variable 'weight:0' shape=(1,) dtype=float32_ref>
<tf.Variable 'bias:0' shape=(1,) dtype=float32_ref>
Tensor("z_net:0", dtype=float32)
Tensor("sqr_errors:0", dtype=float32)
```

[17]:
```
def train_linreg(sess, model, X_train, y_train, num_epochs=10):
    ## initialiaze all variables: W and b
    sess.run(model.init_op)
    training_costs = []
    for i in range(num_epochs):
        _, cost = sess.run([model.optimizer, model.mean_cost],feed_dict={model.
↪X:X_train,model.y:y_train})
        training_costs.append(cost)
    return training_costs
```

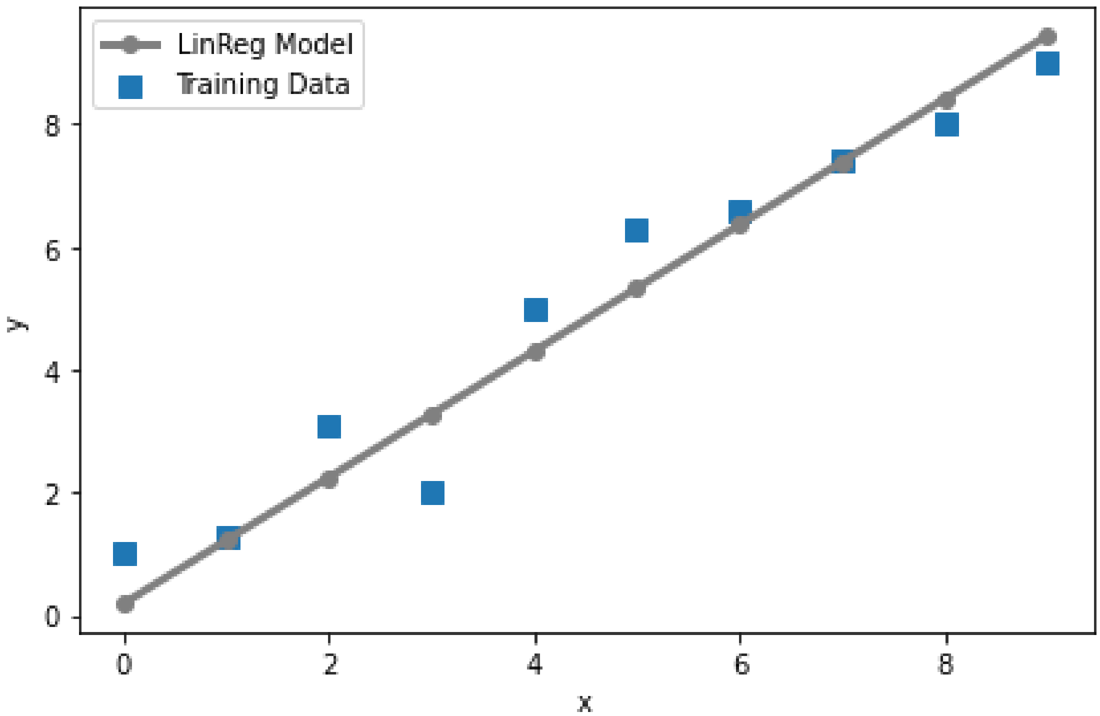
[18]:
```
sess = tf.Session(graph=lrmodel.g)
training_costs = train_linreg(sess, lrmodel, X_train, y_train)
```

[19]:
```python
import matplotlib.pyplot as plt
plt.plot(range(1,len(training_costs) + 1), training_costs)
plt.tight_layout()
plt.xlabel('Epoch')
plt.ylabel('Training Cost')
plt.show()
```



[20]:
```python
def predict_linreg(sess, model, X_test):
    y_pred = sess.run(model.z_net,feed_dict={model.X:X_test})
    return y_pred
```

[22]:
```python
plt.scatter(X_train, y_train,marker='s', s=50,label='Training Data')
plt.plot(range(X_train.shape[0]),predict_linreg(sess, lrmodel,␣
 ↪X_train),color='gray', marker='o',markersize=6, linewidth=3,label='LinReg␣
 ↪Model')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.tight_layout()
plt.show()
```



[26]:
```
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)
        images = ((images / 255.) - .5) * 2
    return images, labels
```

[27]:
```
 ## loading the data
X_train, y_train = load_mnist('./mnist/', kind='train')
print('Rows: %d, Columns: %d' %(X_train.shape[0],X_train.shape[1]))
X_test, y_test = load_mnist('./mnist/', kind='t10k')
print('Rows: %d, Columns: %d' %(X_test.shape[0],X_test.shape[1]))
## mean centering and normalization:
```

```
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
```

```
Rows: 60000, Columns: 784
Rows: 10000, Columns: 784
```

[28]:
```
X_train_centered = (X_train - mean_vals)/std_val
X_test_centered = (X_test - mean_vals)/std_val
```

[29]:
```
del X_train, X_test
```

[30]:
```
print(X_train_centered.shape, y_train.shape)
print(X_test_centered.shape, y_test.shape)
```

```
(60000, 784) (60000,)
(10000, 784) (10000,)
```

[31]:
```
import tensorflow as tf
n_features = X_train_centered.shape[1]
n_classes = 10
random_seed = 123
np.random.seed(random_seed)
g = tf.Graph()
with g.as_default():
    tf.set_random_seed(random_seed)
    tf_x = tf.placeholder(dtype=tf.float32,shape=(None, n_features),name='tf_x')
    tf_y = tf.placeholder(dtype=tf.int32,shape=None, name='tf_y')
    y_onehot = tf.one_hot(indices=tf_y, depth=n_classes)
    h1 = tf.layers.dense(inputs=tf_x, units=50,activation=tf.tanh,name='layer1')
    h2 = tf.layers.dense(inputs=h1, units=50,activation=tf.tanh,name='layer2')
    logits = tf.layers.dense(inputs=h2,units=10,activation=None,name='layer3')
    predictions = {
    'classes' : tf.argmax(logits, axis=1,
    name='predicted_classes'),
    'probabilities' : tf.nn.softmax(logits,
    name='softmax_tensor')
    }
```

[32]:
```
## define cost function and optimizer:
with g.as_default():
    cost = tf.losses.softmax_cross_entropy(onehot_labels=y_onehot,␣
 ↪logits=logits)
    optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
    train_op = optimizer.minimize(loss=cost)
    init_op = tf.global_variables_initializer()
```

[33]:
```
def create_batch_generator(X, y, batch_size=128, shuffle=False):
    X_copy = np.array(X)
```

```
    y_copy = np.array(y)

    if shuffle:
        data = np.column_stack((X_copy, y_copy))
        np.random.shuffle(data)
        X_copy = data[:, :-1]
        y_copy = data[:, -1].astype(int)

    for i in range(0, X.shape[0], batch_size):
        yield (X_copy[i:i+batch_size, :], y_copy[i:i+batch_size])
```

[34]:

```
## create a session to launch the graph
sess = tf.Session(graph=g)
## run the variable initialization operator
sess.run(init_op)
## 50 epochs of training:
for epoch in range(50):
    training_costs = []
    batch_generator = create_batch_generator(X_train_centered,␣
 ↪y_train,batch_size=64)
    for batch_X, batch_y in batch_generator:
    ## prepare a dict to feed data to our network:
        feed = {tf_x:batch_X, tf_y:batch_y}
        _, batch_cost = sess.run([train_op, cost], feed_dict=feed)
        training_costs.append(batch_cost)
    print(' -- Epoch %2d Avg. Training Loss: %.4f' % (epoch+1, np.
 ↪mean(training_costs)))
```

```
-- Epoch  1 Avg. Training Loss: 1.5573
-- Epoch  2 Avg. Training Loss: 0.9492
-- Epoch  3 Avg. Training Loss: 0.7499
-- Epoch  4 Avg. Training Loss: 0.6387
-- Epoch  5 Avg. Training Loss: 0.5668
-- Epoch  6 Avg. Training Loss: 0.5160
-- Epoch  7 Avg. Training Loss: 0.4781
-- Epoch  8 Avg. Training Loss: 0.4486
-- Epoch  9 Avg. Training Loss: 0.4247
-- Epoch 10 Avg. Training Loss: 0.4051
-- Epoch 11 Avg. Training Loss: 0.3884
-- Epoch 12 Avg. Training Loss: 0.3741
-- Epoch 13 Avg. Training Loss: 0.3617
-- Epoch 14 Avg. Training Loss: 0.3507
-- Epoch 15 Avg. Training Loss: 0.3408
-- Epoch 16 Avg. Training Loss: 0.3320
-- Epoch 17 Avg. Training Loss: 0.3239
-- Epoch 18 Avg. Training Loss: 0.3165
-- Epoch 19 Avg. Training Loss: 0.3097
-- Epoch 20 Avg. Training Loss: 0.3035
```

```
-- Epoch 21 Avg. Training Loss: 0.2976
-- Epoch 22 Avg. Training Loss: 0.2921
-- Epoch 23 Avg. Training Loss: 0.2870
-- Epoch 24 Avg. Training Loss: 0.2822
-- Epoch 25 Avg. Training Loss: 0.2776
-- Epoch 26 Avg. Training Loss: 0.2733
-- Epoch 27 Avg. Training Loss: 0.2693
-- Epoch 28 Avg. Training Loss: 0.2654
-- Epoch 29 Avg. Training Loss: 0.2617
-- Epoch 30 Avg. Training Loss: 0.2581
-- Epoch 31 Avg. Training Loss: 0.2547
-- Epoch 32 Avg. Training Loss: 0.2515
-- Epoch 33 Avg. Training Loss: 0.2483
-- Epoch 34 Avg. Training Loss: 0.2453
-- Epoch 35 Avg. Training Loss: 0.2425
-- Epoch 36 Avg. Training Loss: 0.2397
-- Epoch 37 Avg. Training Loss: 0.2370
-- Epoch 38 Avg. Training Loss: 0.2344
-- Epoch 39 Avg. Training Loss: 0.2319
-- Epoch 40 Avg. Training Loss: 0.2294
-- Epoch 41 Avg. Training Loss: 0.2271
-- Epoch 42 Avg. Training Loss: 0.2248
-- Epoch 43 Avg. Training Loss: 0.2226
-- Epoch 44 Avg. Training Loss: 0.2204
-- Epoch 45 Avg. Training Loss: 0.2183
-- Epoch 46 Avg. Training Loss: 0.2163
-- Epoch 47 Avg. Training Loss: 0.2143
-- Epoch 48 Avg. Training Loss: 0.2124
-- Epoch 49 Avg. Training Loss: 0.2105
-- Epoch 50 Avg. Training Loss: 0.2086
```

[35]:

```
## do prediction on the test set:
feed = {tf_x : X_test_centered}
y_pred = sess.run(predictions['classes'],feed_dict=feed)
print('Test Accuracy: %.2f%%' % (100*np.sum(y_pred == y_test)/y_test.shape[0]))
```

```
Test Accuracy: 93.89%
```

[36]:

```
X_train, y_train = load_mnist('mnist/', kind='train')
print('Rows: %d, Columns: %d' %(X_train.shape[0],X_train.shape[1]))
X_test, y_test = load_mnist('mnist/', kind='t10k')
print('Rows: %d, Columns: %d' %(X_test.shape[0],X_test.shape[1]))
## mean centering and normalization:
mean_vals = np.mean(X_train, axis=0)
std_val = np.std(X_train)
X_train_centered = (X_train - mean_vals)/std_val
X_test_centered = (X_test - mean_vals)/std_val
del X_train, X_test
```

```
print(X_train_centered.shape, y_train.shape)
print(X_test_centered.shape, y_test.shape)
```

```
Rows: 60000, Columns: 784
Rows: 10000, Columns: 784
(60000, 784) (60000,)
(10000, 784) (10000,)
```

[37]:
```
import tensorflow as tf
import tensorflow.contrib.keras as keras
np.random.seed(123)
tf.set_random_seed(123)
```

[38]:
```
# import tensorflow.keras as keras
```

[39]:
```
y_train_onehot = keras.utils.to_categorical(y_train)
print('First 3 labels: ', y_train[:3])
print('\nFirst 3 labels (one-hot):\n', y_train_onehot[:3])
```

```
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.]]
```

[40]:
```
model = keras.models.Sequential()
model.add(keras.layers.Dense(units=50,input_dim=X_train_centered.
  ↪shape[1],kernel_initializer='glorot_uniform',bias_initializer='zeros',activation='tanh'))
model.add(keras.layers.
  ↪Dense(units=50,input_dim=50,kernel_initializer='glorot_uniform',bias_initializer='zeros',ac
model.add(keras.layers.Dense(units=y_train_onehot.
  ↪shape[1],input_dim=50,kernel_initializer='glorot_uniform',bias_initializer='zeros',activati
sgd_optimizer = keras.optimizers.SGD(lr=0.001, decay=1e-7, momentum=.9)
model.compile(optimizer=sgd_optimizer,loss='categorical_crossentropy')
```

[41]:
```
history = model.fit(X_train_centered, y_train_onehot,batch_size=64,␣
  ↪epochs=50,verbose=1,validation_split=0.1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/50
54000/54000 [==============================] - 6s - loss: 0.7247 - val_loss:
0.3616
Epoch 2/50
54000/54000 [==============================] - 2s - loss: 0.3718 - val_loss:
0.2815
Epoch 3/50
54000/54000 [==============================] - 2s - loss: 0.3087 - val_loss:
0.2447
```

```
Epoch 4/50
54000/54000 [==============================] - 2s - loss: 0.2728 - val_loss:
0.2216
Epoch 5/50
54000/54000 [==============================] - 2s - loss: 0.2475 - val_loss:
0.2042ss: 0 - ETA: 2s - loss: - ETA: 1s - loss: 0 - ETA: 1s - - ETA: 0s -
Epoch 6/50
54000/54000 [==============================] - 2s - loss: 0.2277 - val_loss:
0.1918
Epoch 7/50
54000/54000 [==============================] - 2s - loss: 0.2115 - val_loss:
0.1810
Epoch 8/50
54000/54000 [==============================] - 2s - loss: 0.1979 - val_loss:
0.1719os
Epoch 9/50
54000/54000 [==============================] - 2s - loss: 0.1860 - val_loss:
0.1646
Epoch 10/50
54000/54000 [==============================] - 2s - loss: 0.1758 - val_loss:
0.1591
Epoch 11/50
54000/54000 [==============================] - 2s - loss: 0.1667 - val_loss:
0.1543
Epoch 12/50
54000/54000 [==============================] - 2s - loss: 0.1589 - val_loss:
0.1491
Epoch 13/50
54000/54000 [==============================] - 2s - loss: 0.1516 - val_loss:
0.1451
Epoch 14/50
54000/54000 [==============================] - 2s - loss: 0.1450 - val_loss:
0.1420ss:
Epoch 15/50
54000/54000 [==============================] - 2s - loss: 0.1389 - val_loss:
0.1386
Epoch 16/50
54000/54000 [==============================] - 2s - loss: 0.1333 - val_loss:
0.1363
Epoch 17/50
54000/54000 [==============================] - 2s - loss: 0.1283 - val_loss:
0.1331
Epoch 18/50
54000/54000 [==============================] - 2s - loss: 0.1234 - val_loss:
0.1327
Epoch 19/50
54000/54000 [==============================] - 3s - loss: 0.1191 - val_loss:
0.1293ss
```

```
Epoch 20/50
54000/54000 [==============================] - 3s - loss: 0.1148 - val_loss:
0.1282
Epoch 21/50
54000/54000 [==============================] - 2s - loss: 0.1109 - val_loss:
0.1270
Epoch 22/50
54000/54000 [==============================] - 2s - loss: 0.1071 - val_loss:
0.1265
Epoch 23/50
54000/54000 [==============================] - 2s - loss: 0.1037 - val_loss:
0.1243
Epoch 24/50
54000/54000 [==============================] - 2s - loss: 0.1003 - val_loss:
0.1229 ETA: 0s - loss: 0.1 - ETA: 0s - loss: 0.10
Epoch 25/50
54000/54000 [==============================] - 3s - loss: 0.0971 - val_loss:
0.1216
Epoch 26/50
54000/54000 [==============================] - 2s - loss: 0.0941 - val_loss:
0.1212
Epoch 27/50
54000/54000 [==============================] - 2s - loss: 0.0912 - val_loss:
0.1200
Epoch 28/50
54000/54000 [==============================] - 2s - loss: 0.0884 - val_loss:
0.1202
Epoch 29/50
54000/54000 [==============================] - 2s - loss: 0.0858 - val_loss:
0.1189
Epoch 30/50
54000/54000 [==============================] - 2s - loss: 0.0834 - val_loss:
0.1184
Epoch 31/50
54000/54000 [==============================] - 2s - loss: 0.0810 - val_loss:
0.1184
Epoch 32/50
54000/54000 [==============================] - 2s - loss: 0.0787 - val_loss:
0.1189
Epoch 33/50
54000/54000 [==============================] - 2s - loss: 0.0765 - val_loss:
0.1183
Epoch 34/50
54000/54000 [==============================] - 2s - loss: 0.0743 - val_loss:
0.1196TA: 0s - loss:
Epoch 35/50
54000/54000 [==============================] - 3s - loss: 0.0723 - val_loss:
0.1179
```

```
Epoch 36/50
54000/54000 [==============================] - 2s - loss: 0.0703 - val_loss:
0.1174
Epoch 37/50
54000/54000 [==============================] - 2s - loss: 0.0684 - val_loss:
0.1184
Epoch 38/50
54000/54000 [==============================] - 2s - loss: 0.0665 - val_loss:
0.1187
Epoch 39/50
54000/54000 [==============================] - 2s - loss: 0.0647 - val_loss:
0.1171ss: 0.06 - ETA: 0s - loss: 0 - ETA: 0s - loss
Epoch 40/50
54000/54000 [==============================] - 2s - loss: 0.0629 - val_loss:
0.1172s
Epoch 41/50
54000/54000 [==============================] - 2s - loss: 0.0613 - val_loss:
0.1175
Epoch 42/50
54000/54000 [==============================] - 2s - loss: 0.0597 - val_loss:
0.1170
Epoch 43/50
54000/54000 [==============================] - 2s - loss: 0.0581 - val_loss:
0.1168
Epoch 44/50
54000/54000 [==============================] - 3s - loss: 0.0566 - val_loss:
0.1166ss: 0.0 - ETA: 1s - loss:  - ETA: 1s - loss: 0.05 - ETA: 1s - loss - ETA:
0s - loss: 0.05 - ETA: 0s - loss
Epoch 45/50
54000/54000 [==============================] - 2s - loss: 0.0552 - val_loss:
0.1166
Epoch 46/50
54000/54000 [==============================] - 2s - loss: 0.0537 - val_loss:
0.1162
Epoch 47/50
54000/54000 [==============================] - 2s - loss: 0.0523 - val_loss:
0.1170
Epoch 48/50
54000/54000 [==============================] - ETA: 0s - loss: 0.051 - 2s -
loss: 0.0510 - val_loss: 0.1172
Epoch 49/50
54000/54000 [==============================] - 2s - loss: 0.0498 - val_loss:
0.1171ss: 0.049
Epoch 50/50
54000/54000 [==============================] - 2s - loss: 0.0485 - val_loss:
0.1174
```

[42]:
```
y_train_pred = model.predict_classes(X_train_centered, verbose=0)
print('First 3 predictions: ', y_train_pred[:3])
```

```
First 3 predictions:  [5 0 4]
```

[43]:
```
y_train_pred = model.predict_classes(X_train_centered,verbose=0)
correct_preds = np.sum(y_train == y_train_pred, axis=0)
train_acc = correct_preds / y_train.shape[0]
print('First 3 predictions: ', y_train_pred[:3])
print('Training accuracy: %.2f%%' % (train_acc * 100))
y_test_pred = model.predict_classes(X_test_centered,verbose=0)
correct_preds = np.sum(y_test == y_test_pred, axis=0)
test_acc = correct_preds / y_test.shape[0]
print('Test accuracy: %.2f%%' % (test_acc * 100))
```

```
First 3 predictions:  [5 0 4]
Training accuracy: 98.88%
Test accuracy: 96.04%
```

[44]:
```
import numpy as np
X = np.array([1, 1.4, 2.5]) ## first value must be 1
w = np.array([0.4, 0.3, 0.5])
def net_input(X, w):
    return np.dot(X, w)
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))
```

```
P(y=1|x) = 0.888
```

[45]:
```
W = np.array([[1.1, 1.2, 0.8, 0.4],[0.2, 0.4, 1.0, 0.2],[0.6, 1.5, 1.2, 0.7]])
A = np.array([[1, 0.1, 0.4, 0.6]])
```

[46]:
```
Z = np.dot(W, A[0])
y_probas = logistic(Z)
print('Net Input: \n', Z)
print('Output Units:\n', y_probas)
```

```
Net Input:
 [1.78 0.76 1.65]
Output Units:
 [0.85569687 0.68135373 0.83889105]
```

[47]:
```
y_class = np.argmax(Z, axis=0)
print('Predicted class label: %d' % y_class)
```

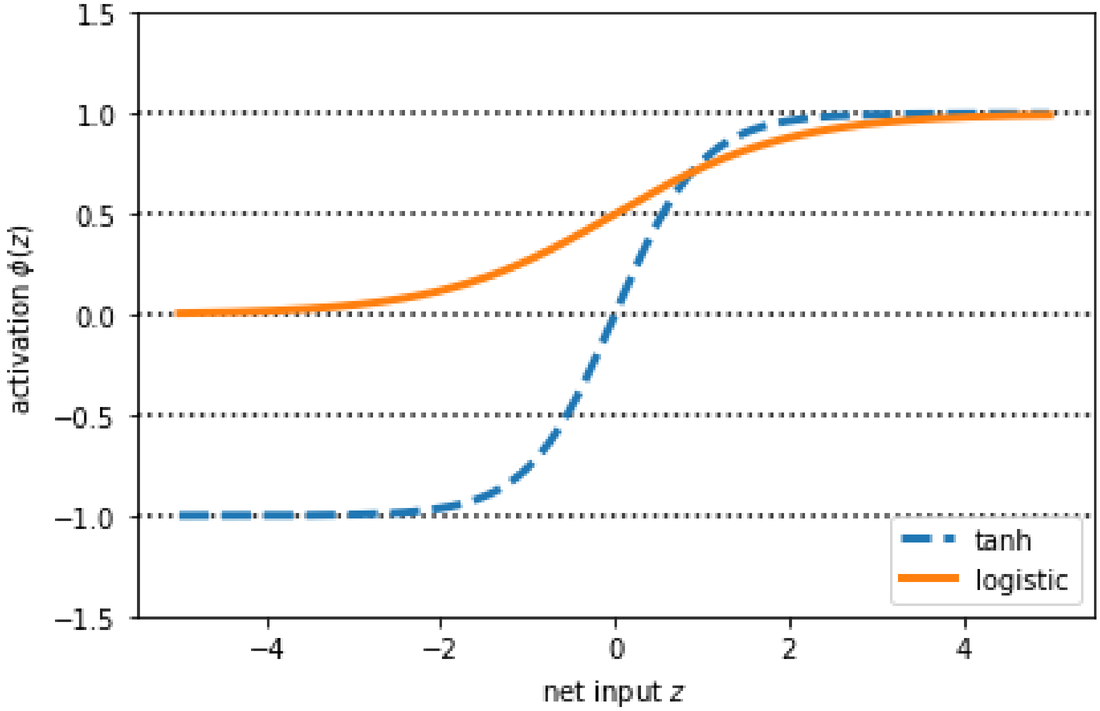
```
Predicted class label: 0
```

[48]:
```python
def softmax(z):
    return np.exp(z) / np.sum(np.exp(z))
y_probas = softmax(Z)
print('Probabilities:\n', y_probas)
np.sum(y_probas)
```

```
Probabilities:
 [0.44668973 0.16107406 0.39223621]
```

[48]: 1.0

[49]:
```python
import matplotlib.pyplot as plt
def tanh(z):
    e_p = np.exp(z)
    e_m = np.exp(-z)
    return (e_p - e_m) / (e_p + e_m)
z = np.arange(-5, 5, 0.005)
log_act = logistic(z)
tanh_act = 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(-0.5, color='black', linestyle=':')
plt.axhline(-1, color='black', linestyle=':')
plt.plot(z, tanh_act,linewidth=3, linestyle='--',label='tanh')
plt.plot(z, log_act,linewidth=3,label='logistic')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()
```



[50]:
```
tanh_act = np.tanh(z)
```

[51]:
```
tanh_act
```

[51]:
```
array([-0.9999092 , -0.99990829, -0.99990737, …,  0.99990644,
        0.99990737,  0.99990829])
```

[52]:
```
from scipy.special import expit
log_act = expit(z)
```

[53]:
```
log_act
```

[53]:
```
array([0.00669285, 0.00672617, 0.00675966, …, 0.99320669, 0.99324034,
       0.99327383])
```

[ ]: