LAB 6 SVM

```
# TODO: Select four random digits and plot them usi
def plt_digit(x):
    nrow = 28
    ncol = 28
    xsq = x.reshape((nrow,ncol))
    plt.imshow(xsq, cmap='Greys_r') #print the pict
    plt.xticks([])
    plt.yticks([])

# Select random digits
nplt = 4
nsamp = Xdig.shape[0]
Iperm = np.random.permutation(nsamp)

# Plot the images using the subplot command
for i in range(nplt):
    ind = Iperm[i]
    plt.subplot(1,nplt, i + 1)
    plt_digit(Xdig[ind, :])
    title = 'label={0:d}'.format(ind.astype(int))
    plt.title(title)
```

Create Extended Training Data

Now, create an extended data set by combining `ndig=5000` randomly selected digit samples and `nlet=1000` letters.

- Select `ndig=5000` random samples from `Xdigs` and their labels in `ydig`.
- Rescale the letters `Xlet` to a new matrix `Xlets = 2*Xlet-1` to make the pixel values go from -1 to 1.
- Use the `np.vstack` command to create a 6000 element alpha-numeric data set `X`
- Create a corresponding label vector `y` where all the non-digit characters are labeled with a non-digit label, `letter_lbl=10`.

```
# TODO
# X = ...  Array with 6000 characters (5000 digits + 1000 letters)
# y = ...  Array with 6000 labels (0-9 for the digits, 10 = non-digit)
ndig = 5000
letter_lbl=10
Xdigs_5000 = np.zeros((ndig, npix))
y = np.zeros(6000)
nsamp = Xdigs.shape[0]
Iperm = np.random.permutation(nsamp)
for i in range(ndig):
    ind = Iperm[i]
    Xdigs_5000[i,:] = Xdigs[ind,:]
    y[i] = ydig[ind]
Xlets = 2*Xlet[0]/255.0 - 1

y[5000:] = letter_lbl
X = np.vstack((Xdigs_5000, Xlets))
```

Get 5000 training samples `Xtr,ytr` and 1000 test samples `Xts,yts`. Remember to randomly select them.

```
# TODO
ntr = 5000
nts = 1000
Iperm2 = np.random.permutation(X.shape[0])
Xtr = X[Iperm2[:ntr],:]
ytr = y[Iperm2[:ntr]]
Xts = X[Iperm2[ntr:ntr+nts],:]
yts = y[Iperm2[ntr:ntr+nts]]
```

Measure the accuracy on the test samples. You should to run.

```
# TODO
yhat_ts = svc.predict(Xts)
acc = np.mean(yhat_ts == yts)
print('Accuaracy = {0:f}'.format(acc))
```

Plotting some error samples

We now plot some errors. Plot up to four images where yhat == 10 but yts != 10. That is, the true image was a digit, but the classifier classified it as a non-digit. Note there may be less than four such errors (when I ran it I got only three such errors). In that case, just plot only the errors you got. If there are no errors, print "No such error found"

```
# TODO
Ierr3 = []
for i in range (yts.shape[0]):
    if (yhat_ts[i] != yts[i] and yhat_ts[i] < 10 and yts[i] < 10):
        Ierr3.append(i)
Ierr3 = np.array(Ierr3)
print(Ierr3)
nplt = Ierr3.shape[0]
try:
    if (nplt <= 0):
        raise ImgException("No such error found")
    index = np.random.permutation(nplt)
    plt.figure(figsize=(100, 40))
    for i in range(4):
        plt.subplot(1,nplt,i+1)
        ind = Ierr3[index[i]]
        plt_digit(Xts[ind,:])
        title = 'true={0:d} est={1:d}'.format(yts[ind].astype(int), yhat_ts[ind].astype(int))
        plt.title(title)
except ImgException as e:
    print(e.msg)
```

sLAB 7 Neural Network for music classification

数据标准化，正则化

```
# TODO Scale the training and test matrices
Xtr_std = np.std(Xtr, axis = 0)
Xtr_mean = np.mean(Xtr, axis = 0)
Xts_std = np.std(Xts, axis = 0)
Xts_mean = np.mean(Xts, axis = 0)

# if we scale by:
# Xtr_scale = (Xtr - Xtr_mean) / Xtr_std
# Xts_scale = (Xts - Xts_mean) / Xts_std
# we eventually get val_acc around 95%
# otherwise if we scale by:
Xtr_scale = (Xtr - Xts_mean) / Xts_std
Xts_scale = (Xts - Xts_mean) / Xts_std
# we eventually get val_acc over 99%
```

建神经网络模型

```
from keras.models import Model, Sequential
from keras.layers import Dense, Activation
```

```
# TODO clear session
import keras.backend as K
K.clear_session()
```

```
# TODO: construct the model
nin = Xtr.shape[1]  # dimension of input data
nh = 256     # number of hidden units
# number of outputs = 10 since there are 10 classes
nout = int(np.max(yts) - np.min(yts)+1)
model = Sequential()
model.add(Dense(nh, input_shape=(nin,), activation='sigmoid', name='hidden'))
model.add(Dense(nout, activation='softmax', name='output'))
```

```
# TODO:  Print the model summary
model.summary()
```

```
class LossHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        # TODO:  Create two empty lists, self.loss and self.val_acc
        self.loss = []
        self.val_acc= []
    def on_batch_end(self, batch, logs={}):
        # TODO:  This is called at the end of each batch.
        # Add the loss in logs.get('loss') to the loss list
        self.loss.append(logs.get('loss'))
    def on_epoch_end(self, epoch, logs):
        # TODO:  This is called at the end of each epoch.
        # Add the test accuracy in logs.get('loss') to the val_acc list
        self.val_acc.append(logs.get('val_acc'))

        return logs;
# Create an instance of the history callback
history_cb = LossHistory()
```

Create an optimizer and compile the model. Select the appropriate loss function and metrics. For the optimizer, use 0.001

```
# TODO
from keras import optimizers
opt = optimizers.Adam(lr=0.001) # beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Fit the model for 10 epochs using the scaled data for both the training and validation. Use the `validation_data` c callback class create above. Use a batch size of 100. Your final accuracy should be >99%.

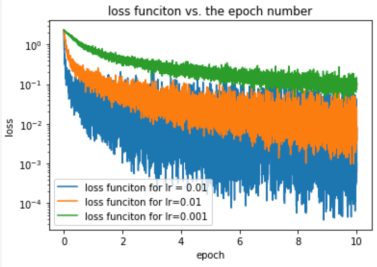
```
# TODO
batch_size = 100
model.fit(Xtr_scale, ytr, epochs=10, batch_size=batch_size,
                    validation_data=(Xts_scale,yts), callbacks=[history_cb])
```

```
# TODO
print(history_cb.val_acc)
# summarize history for accuracy
plt.plot(history.history['val_acc'])
plt.title('validation accuracy in the history_cb')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.show()
```

```
[0.94820182829279442, 0.97691895154875197, 0.9842324230661
4161953, 0.99013687781409687, 0.98879495666222872, 0.99141
```

不同学习率下的情况

```
rates = [0.01,0.001,0.0001]

ntr = Xtr.shape[0]
loss_len=len(history_cb.loss)
step = np.linspace(0,loss_len, loss_len)
batch_size = 100
epoch = step * batch_size / ntr
loss_hist = []
val_acc_hist = []
history = []

# TODO
for i in range(3):
    K.clear_session()
    nin = Xtr.shape[1]
    nh = 256
    nout = int(max(yts) - min(yts) + 1)

    model = Sequential()
    model.add(Dense(nh, input_shape=(nin,), activation='sigmoid', name='hidden'))
    model.add(Dense(nout, activation='sigmoid', name='output'))

    opt = optimizers.Adam(lr=rates[i])
    model.compile(optimizer=opt,
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])


    model.fit(Xtr_scale, ytr, epochs=10, batch_size=batch_size,
                            validation_data=(Xts_scale,yts),callbacks=[history_cb])

    plt.semilogy(epoch, history_cb.loss)
```



LAB 8 Pre-trained Deep Neural Network

1. 下载图片，下载很多图片，用 vgg16 辨别

2. 展示图片，8 张

```
# Display the image
def disp_image(im):
    if (len(im.shape) == 2):
        # Gray scale image
        plt.imshow(im, cmap='gray')
    else:
        # Color image.
        im1 = (im-np.min(im))/(np.max(im)-np.min(im))*255
        im1 = im1.astype(np.uint8)
        plt.imshow(im1)

    # Remove axis ticks
    plt.xticks([])
    plt.yticks([])
```

```
# TODO
X, y = train_generator.next()

plt.figure(figsize=(20,20))
nplt = 8
for i in range(nplt):
    plt.subplot(1, nplt, i+1)
    disp_image(X[i])
    title = 'lable={0:d}'.format(y[i].astype(int))
    plt.title(title)
```

3. 训练模型

Train the Model

Compile the model. Select the correct `loss` function, `optimizer` and `metrics`. Remember that we are performing binary classification.

```
# TODO.
# model.compile(...)
opt = optimizers.Adam(lr=0.001,beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
model.compile(optimizer=opt,
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
# TODO
steps_per_epoch = train_generator.n // batch_size
validation_steps = test_generator.n // batch_size
```

```
nepochs = 5  # Number of epochs

# Call the fit function
model.fit_generator(
    train_generator,
    steps_per_epoch=steps_per_epoch,
    epochs=nepochs,
    validation_data=test_generator,
    validation_steps=validation_steps)
```

HW6

1. Consider the data set for four points with features $\mathbf{x}_i = (x_{i1}, x_{i2})$ and binary class labels $y_i = \pm 1$.

x_{i1}	0	1	1	2
x_{i2}	0	0.3	0.7	1
y_i	-1	-1	1	1

(a) Find a linear classifier that separates the two classes. Your classifier should be of the form

$$\hat{y} = \begin{cases} 1 & \text{if } b + w_1 x_1 + w_2 x_2 > 0 \\ -1 & \text{if } b + w_1 x_1 + w_2 x_2 < 0 \end{cases}$$

State the intercept b and weights w_1 and w_2 for your classifier. Note there is no unique answer as there are multiple linear classifiers that could separate the classes.

(b) Find the maximum γ such that

$$y_i(b + w_1 x_{i1} + w_{i2} x_{i2}) \geq \gamma, \text{ for all } i,$$

for the classifier in part (a)?

(c) Compute the margin of the classifier

$$m = \frac{\gamma}{\|\mathbf{w}\|}, \quad \|\mathbf{w}\| = \sqrt{w_1^2 + w_2^2}.$$

(d) Which samples i are on the margin for your classifier?

(a)

$$\hat{y} = \begin{cases} 1 & \text{if } x_2 > 0.5 \\ -1 & \text{if } x_2 < 0.5 \end{cases} = \begin{cases} 1 & \text{if } b + w_1 x_1 + w_2 x_2 > 0 \\ -1 & \text{if } b + w_1 x_1 + w_2 x_2 < 0, \end{cases}$$

where $b = -0.5$, $w_1 = 0$ and $w_2 = 1$.

(b)

x_{i1}	0	1	1	2	
x_{i2}	0	0.3	0.7	1	
y_i	-1	-1	1	1	
z_i	-0.5	-0.2	0.2	0.5	
$y_i z_i$	0.5	0.2	0.2	0.5	

Since we require $z_i y_i \geq \gamma$ for all i, the largest value of γ we can take is $\gamma = 0.2$.

(c) (c) The margin is

$$m = \frac{\gamma}{\|\mathbf{w}\|} = \frac{0.2}{\sqrt{0+1}} = 0.2.$$

2. Consider the data set with scalar features x_i and binary class labels $y_i = \pm 1$.

x_i	0	1.3	2.1	2.8	4.2	5.7
y_i	-1	-1	-1	1	-1	1

Consider a linear classifier for this data of the form,

$$\hat{y} = \begin{cases} 1 & z > 0 \\ -1 & z < 0, \end{cases} \qquad z = x - t,$$

where t is a threshold. For each threshold t, let $J(t)$ denote the sum hinge loss,

$$J(t) = \sum_i \epsilon_i, \quad \epsilon_i = \max(0, 1 - y_i z_i).$$

(a) Write a short python program to plot $J(t)$ vs. t for 100 values of t in the interval $t \in [0, 5]$.

(b) Based on the plot, what is one value of t that minimizes $J(t)$.

(c) For the value of t in part (b), find the corresponding slack variables ϵ_i.

(d) Which samples i violate the margin ($\epsilon_i > 0$) and which samples i are misclassified ($\epsilon_i > 1$).

(a) You can compute and plot $J(t)$ with the following code. The figure is shown in Fig. 1.

```
x = np.array([0,1.3,2.1,2.8,4.2,5.7])
y = np.array([-1,-1,-1,1,-1,1])
tvals = np.linspace(0,6,100)
J = []
for t in tvals:
    z = x-t
    Jt = np.sum(np.maximum(0,1-y*z))
    J.append(Jt)
J = np.array(J)

plt.plot(tvals, J)
plt.xlabel('t')
plt.ylabel('J(t)')
plt.grid()

plt.savefig('Jt.png')
```

(b) From Fig. 1, we see that $t = 4$ is a minimizer.

(c) For the value of t in part (b), find the corresponding slack variables ϵ_i.
We can compute the slack variables by the python code:

```
t = 4
z = x-t
eps = np.maximum(0, 1-y*z)
eps

>> array([ 0. ,  0. ,  0. ,  2.2,  1.2,  0. ])
```

(d) We see that $\epsilon_i > 1$ for samples $i = 3, 4$ so both of these samples will be misclassified (and violate the margin).

Consider an image recognition problem, where an image $\mathbf{X}$ and filter $\mathbf{W}$ are 4×4 matrices:

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

(a) Recall that in linear classification, the 4×4 image matrices $\mathbf{X}$ and $\mathbf{W}$ can be represented as 16-dimensional vectors, $\mathbf{x} = \text{vec}(\mathbf{X})$ and $\mathbf{w} = \text{vec}(\mathbf{W})$ by stacking the columns of the matrices vertically. What are $\mathbf{x}$ and $\mathbf{w}$ for the matrices above.

(b) What is the inner product $z = \mathbf{w}^T\mathbf{x}$.

(c) What is the inner product $z = \mathbf{w}^T\mathbf{x}_{\text{right}}$ where $\mathbf{x}_{\text{right}}$ is the vector corresponding to the matrix $\mathbf{X}$ right shifted by one pixel with the left column filled with zeros.

(d) What is the inner product $z = \mathbf{w}^T\mathbf{x}_{\text{left}}$ where $\mathbf{x}_{\text{left}}$ is the vector corresponding to the matrix $\mathbf{X}$ left shifted by one pixel with the right column filled with zeros.

(e) Write the python command that can covert a 4×4 image matrix, `Xmat` to the 16-dimensional vector, `x`. What is the python command to go from `x` to `Xmat`.

a) Going down the columns of $\mathbf{X}$ and $\mathbf{W}$, the vectors are:

$$\mathbf{x} = [0,0,0,0,\ 0,0,0,0,\ 0,1,1,1,\ 0,0,0,0]$$
$$\mathbf{w} = [0,0,0,0,\ 0,1,1,0,\ 0,1,1,0,\ 0,0,0,0]$$

b) The inner product will be

$$z = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{16} x_i w_i.$$

Since x_i and $w_i = 0$ or 1, $x_i w_i = 1$ only on the pixels where $x_i = w_i = 1$. Hence $z = \mathbf{w}^T\mathbf{x}$ is the number of pixels where two images overlap. Thus, we have $z = 2$.

c) If $\mathbf{X}$ is shifted to the right we have

$$\mathbf{X}_{\text{right}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This has no overlap with $\mathbf{W}$, so $z = \mathbf{w}^T\mathbf{x}_{\text{right}} = 0$.

(d) If $\mathbf{X}$ is shifted to the left we have

$$\mathbf{X}_{\text{left}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

This overlaps with two pixels in $\mathbf{W}$. Hence, $z = \mathbf{w}^T\mathbf{x}_{\text{left}} = 2$.

4. Consider the data set with scalar features x_i and binary class labels $y_i = \pm 1$.

x_i	0	1	2	3
y_i	1	-1	1	-1

A support vector classifier is of the form

$$\hat{y} = \begin{cases} 1 & z > 0 \\ -1 & z < 0, \end{cases} \quad z = \sum_i \alpha_i y_i K(x_i, x),$$

where $K(x, x')$ is the radial basis function, $K(x, x') = e^{-\gamma(x-x')^2}$, and $\gamma > 0$ and $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_4]$ are parameters of the classifier.

(a) Use python to plot z vs. x and $\hat{y}$ vs. x when $\gamma = 3$ and $\boldsymbol{\alpha} = [0, 0, 1, 1]$.

(b) Repeat (a) with $\gamma = 0.3$ and $\boldsymbol{\alpha} = [1, 1, 1, 1]$.

(c) Which classifier makes more errors on the training data.

```
y = np.array([1,-1,1,-1])

def plot_zrbf(x,y,a,gam):
    xp = np.linspace(-2,5,100)
    xpmat,xmat = np.meshgrid(xp,x)
    dist = (xpmat-xmat)**2
    z = (y*a).dot(np.exp(-gam*dist))
    plt.plot(xp,z)
    plt.grid()
    plt.xlabel('x',fontsize=16)

plt.subplot(1,2,1)
a = np.array([0,0,1,1])
plot_zrbf(x,y,a,gam=3)
plt.title('gamma=3')

plt.subplot(1,2,2)
a = np.array([1,1,1,1])
plot_zrbf(x,y,a,gam=0.3)
plt.title('gamma=0.3')

plt.savefig('rbf.png')
```

The result function z is plotted in the left of Fig. 2.

(b) The function z for $\gamma = 3$ is plotted in the right of Fig. 2.

(c) The classifier takes $\hat{y}_i = \text{sign}(z_i)$. The resulting values are shown in the table below. We see that the classifier in part (b) makes no errors. Since it uses a higher γ it is able to fit the data better.

x_i	0	1	2	3
y_i	1	-1	1	-1
$\hat{y}_i = \text{sign}(z_i)$ for (a)	1	1	1	-1
$\hat{y}_i = \text{sign}(z_i)$ for (b)	1	-1	1	-1

HW7

– Hard threshold:

$$g_{\text{act}}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0. \end{cases} \quad (2)$$

激励函数

– Sigmoid: $g_{\text{act}}(z) = 1/(1 + e^{-z})$.

– Rectified linear unit (ReLU): $g(z) = \max\{0, z\}$.

$$\hat{y} = \begin{cases} 1 & z^{\text{O}} \geq 0 \\ 0 & z^{\text{O}} < 0. \end{cases} \quad (4)$$

决策函数

1. Consider the neural network (1) with a scalar input x and parameters

$$W^{\text{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b^{\text{H}} = \begin{bmatrix} -1 \\ -3 \end{bmatrix} \quad W^{\text{O}} = [-1, 2], \quad b^{\text{O}} = 0.5,$$

using a hard threshold activation function (2) and threshold output function (4).

(a) What is N_h, the number of hidden units? What is N_o, the number of output units?

(b) Write $\mathbf{z}^{\text{H}}$ in terms of x. Draw the functions for each component z_j^{H}.

(c) Write $\mathbf{u}^{\text{H}}$ in terms of x. Draw the functions for each component u_j^{H}.

(d) Write z^{O} in terms of x.

(e) What is $\hat{y}$ in terms of x?

1. (a) Since $\mathbf{W}^{\text{H}}$ has 2 ouptuts, $N_h = 2$. Since W^{O} has 1 output, $N_o = 1$.

(b) The components of the linear variables $\mathbf{z}^{\text{H}}$ in the hidden layer are:

$$\mathbf{z}^{\text{H}} = \mathbf{W}^{\text{H}}\mathbf{x} + \mathbf{b}^{\text{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x + \begin{bmatrix} -1 \\ -3 \end{bmatrix} = \begin{bmatrix} x - 1 \\ x - 3 \end{bmatrix}.$$

(c) The activation outputs in the hidden layer are

$$\mathbf{u}^{\text{H}} = g_{\text{act}}(\mathbf{z}^{\text{H}}) = \begin{bmatrix} g_{\text{act}}(x - 1) \\ g_{\text{act}}(x - 3) \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{\{x \geq 1\}} \\ \mathbb{I}_{\{x \geq 3\}} \end{bmatrix}.$$

(d) The linear variable in the output layer is

$$z^{\text{O}} = W^{\text{O}}\mathbf{u}^{\text{H}} + b^{\text{O}} = [-1, 2] \begin{bmatrix} \mathbb{I}_{\{x \geq 1\}} \\ \mathbb{I}_{\{x \geq 3\}} \end{bmatrix} + 0.5$$
$$= \mathbb{I}_{\{x \geq 1\}} - 2\mathbb{I}_{\{x \geq 3\}} + 0.5 = \begin{cases} 0.5 & \text{when } x < 1 \\ -0.5 & \text{when } x \in [1, 3) \\ 1.5 & \text{when } x \geq 3 \end{cases}$$

(e) The predicted output $\hat{y}$ is

$$\hat{y} = \mathbb{I}_{\{z^{\text{O}} \geq 0\}} = \begin{cases} 0 & \text{if } x \in [1, 3) \\ 1 & \text{else} \end{cases}$$

2. Consider the data set for four points with scalar features x_i and binary class labels $y_i = 0, 1$.

x_i	0	1	3	5
y_i	0	0	1	0

(a) Find a neural network with $N_h = 2$ units, $N_o = 1$ output units such that $\hat{y}_i = y_i$ on all four data points. Use a hard threshold activation function (2) and threshold output function (4). State the weights and biases used in each layer.

(b) Compute the values of $\hat{y}_i$ and all the intermediate variables $\mathbf{z}_i^{\text{H}}$, $\mathbf{u}_i^{\text{H}}$ and z_i^{O} for each sample $x = x_i$.

(c) Now suppose we are given a new sample, $x = 3.5$. What does the network predict as $\hat{y}$.

$$\mathbf{W}^{\text{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b^{\text{H}} = \begin{bmatrix} -2 \\ -4 \end{bmatrix}, \qquad W^{\text{O}} = [1, -2], \quad b^{\text{O}} = -0.5.$$

Then,

so that

$$\mathbf{z}^{\text{H}} = \mathbf{W}^{\text{H}}\mathbf{x} + \mathbf{b}^{\text{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x + \begin{bmatrix} -2 \\ -4 \end{bmatrix} = \begin{bmatrix} x - 2 \\ x - 4 \end{bmatrix}.$$

$$z^{\text{O}} = W^{\text{O}}\mathbf{u}^{\text{H}} + b^{\text{O}} = [1, -2] \begin{bmatrix} \mathbb{I}_{\{x \geq 2\}} \\ \mathbb{I}_{\{x \geq 4\}} \end{bmatrix} - 0.5$$

Then, the activation outputs in the hidden layer are

$$= \mathbb{I}_{\{x \geq 2\}} - 2\mathbb{I}_{\{x \geq 4\}} - 0.5 = \begin{cases} -0.5 & \text{when } x < 2 \\ 0.5 & \text{when } x \in [2, 4) \\ -1.5 & \text{when } x \geq 4. \end{cases}$$

$$\mathbf{u}^{\text{H}} = g_{\text{act}}(\mathbf{z}^{\text{H}}) = \begin{bmatrix} g_{\text{act}}(x - 1) \\ g_{\text{act}}(x - 3) \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{\{x \geq 2\}} \\ \mathbb{I}_{\{x \geq 4\}} \end{bmatrix}.$$

The predicted output $\hat{y}$ is

$$\hat{y} = \mathbb{I}_{\{z^{\text{O}} \geq 0\}} = \begin{cases} 1 & \text{if } x \in [2, 4) \\ 0 & \text{else} \end{cases}$$

(b) Table 1 computes the values for all intermediate variables for the four training data points. We can see that $\hat{y}_i = y_i$ for all four points.

(c) The value for $x = 3.5$ is shown in the final column of Table 1. For this value of $x = 3.5$, we get $\hat{y} = 1$.

3. *Two-dimensional example:* Consider a neural network on a 2-dimensional input $\mathbf{x} = (x_1, x_2)$ with weights and biases:

$$W^{\text{H}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad b^{\text{H}} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \quad W^{\text{O}} = [1, 1, -1], \quad b^{\text{O}} = -1.5.$$

Assume the network uses hard a threshold activation function (2) and threshold output function (4).

(a) Write the components of $\mathbf{z}^{\text{H}}$ and $\mathbf{u}^{\text{H}}$ as a function of (x_1, x_2). For each component j, indicate where in the (x_1, x_2) plane $u_j^{\text{H}} = 1$.

(b) Write z^{O} as a function of (x_1, x_2). In what region is $\hat{y} = 1$?

(a) The linear functions in the hidden layer are:

$$\mathbf{z}^{\text{H}} = \mathbf{W}^{\text{H}}\mathbf{x} + \mathbf{b}^{\text{H}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 + x_2 - 1 \end{bmatrix}$$

Hence, the activation functions are

$$\mathbf{u}^{\text{H}} = g_{\text{act}}(\mathbf{z}^{\text{H}}) = \begin{bmatrix} g_{\text{act}}(x_1) \\ g_{\text{act}}(x_2) \\ g_{\text{act}}(x_1 + x_2 - 1) \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{\{x_1 \geq 0\}} \\ \mathbb{I}_{\{x_2 \geq 0\}} \\ \mathbb{I}_{\{x_1 + x_2 \geq 1\}} \end{bmatrix}.$$

b) The output z^{O} is

$$z^{\mathrm{O}} = W^{\mathrm{O}}\mathbf{u}^{\mathrm{H}} + b^{\mathrm{O}} = [1,1,-1]\begin{bmatrix} \mathbb{1}_{\{x_1\geq 0\}} \\ \mathbb{1}_{\{x_2\geq 0\}} \\ \mathbb{1}_{\{x_1+x_2\geq 1\}} \end{bmatrix} - 1.5$$
$$= \mathbb{1}_{\{x_1\geq 0\}} + \mathbb{1}_{\{x_2\geq 0\}} - \mathbb{1}_{\{x_1+x_2\geq 1\}} - 1.5.$$

It is best to draw this. If you do that (I will add the figures later), you will see that in the triangle defined by the lines

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_1 + x_2 < 1, \tag{12}$$

we have

$$z^o = (1) + (1) - (0) - 1.5 = 0.5.$$

Outside this region, $z^{\mathrm{O}} < 0$. Hence, $z^{\mathrm{O}} \geq 0$ only in the region (12). Therefore,

$$\hat{y} = \mathbb{1}_{\{z^{\mathrm{O}}\geq 0\}} = \begin{cases} 1 & \text{if } x_1 \geq 0, x_2 \geq 0, x_1 + x_2 < 1 \\ 0 & \text{else} \end{cases}$$

5. *Implementation in python:* Write python code for implementing the following steps for a batch of samples:

$$\mathbf{Z}^{\mathrm{H}} = \mathbf{X}\left[\mathbf{W}^{\mathrm{H}}\right]^{\mathsf{T}} + \mathbf{B}^{\mathrm{H}}, \quad \mathbf{U}^{\mathrm{H}} = g_{\mathrm{act}}(\mathbf{Z}^{\mathrm{H}}) \tag{10a}$$
$$\mathbf{Z}^{\mathrm{O}} = \mathbf{U}^{\mathrm{H}}\left[\mathbf{W}^{\mathrm{O}}\right]^{\mathsf{T}} + \mathbf{B}^{\mathrm{O}} \quad \mathbf{U}^{\mathrm{O}} = g_{\mathrm{out}}\left(\mathbf{Z}^{\mathrm{O}}\right), \tag{10b}$$

(a) The operations in the hidden layer in (10) can be implemented as:

```
Zh = X.dot(Wh.T) + bh[None,:]
Uh = 1/(1+np.exp(-Zh))
```

where `Wh` and `bh` are the weight matrix and bias vector. Note that we have used python broadcasting on the bias vector `bh`

(b) For a binary classification problem, $N_o = 1$ and we can represent $\mathbf{Z}^{\mathrm{O}}$ as an N-dimensional vector, `Zo`. Also, we can represent the weight $\mathbf{W}^{\mathrm{H}}$ as an N_h-dimensional vector `Wh` and the bias b^{H} as a scalar `bh`. In this case,

```
Zo = Uh.dot(Wo) + bo
Uo = 1/(1+np.exp(-Zo))
```

(c) For K-class classification, we perform

```
Zo = Uh.dot(Wo.T) + bo[None,:]
expZo = np.exp(Zo)
Uo = expZo / np.sum(expZo,axis=1)[None,:]
```

Note the method for normalizing the outputs in a softmax classifier.

Section2:

1. Compute the gradient $\partial f(\mathbf{x})/\partial \mathbf{x}$ of the function,

$$f(\mathbf{x}) = (x_1 + x_2x_3, x_1^2 + 3x_2).$$
$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} 1 & x_3 & x_2 \\ 2x_1 & 3 & 0 \end{bmatrix}$$

2. Consider the logistic loss function,

$$J = \sum_{i=1}^{N} \left[\ln(1+e^{z_i}) - y_i z_i\right], \quad z_i = \sum_{j=1}^{p} x_{ij}w_j + b.$$

(a) What is the gradient $\partial J/\partial \mathbf{z}$? That is, find the components $\partial J/\partial z_i$.

(b) What are the gradient $\partial \mathbf{z}/\partial \mathbf{w}$ and $\partial \mathbf{z}/\partial b$?

(c) Use the chain rule to compute the gradients $\partial J/\partial \mathbf{w}$ and $\partial J/\partial b$.

(a) The components of the gradient $dJ/\partial \mathbf{z}$ are given by the partial derivatives,

$$\frac{\partial J}{\partial z_i} = -y_i + \frac{e^{z_i}}{1+e^{z_i}}.$$

(b) The components of the gradient $\partial \mathbf{z}/\partial \mathbf{w}$ and $\partial \mathbf{z}/\partial b$ are

$$\frac{\partial z_i}{\partial w_i} = x_{ij}, \quad \frac{\partial z_i}{\partial b} = 1.$$

(c) Using chain rule,

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial z_i}\frac{\partial z_i}{\partial w_j} = \frac{\partial J}{\partial z_i}x_{ij},$$
$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial z_i}\frac{\partial z_i}{\partial b} = \frac{\partial J}{\partial z_i},$$

where $\partial J/\partial z_i$ is given in part (a).

3. Suppose that

$$\mathbf{u} = f(\mathbf{z}) = (f(z_1), \ldots, f(z_N)), \quad f(z_i) = \frac{1}{1+e^{-z_i}},$$

so that both $\mathbf{u}$ and $\mathbf{z}$ are N-dimensional vectors.

(a) What is the gradient $\partial J/\partial \mathbf{z}$?

(b) If $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, what are the gradients $\partial J/\partial \mathbf{W}$ and $\partial J/\partial \mathbf{b}$?

(c) In part (b), what is $\partial J/\partial \mathbf{x}$?

(a) Since u_i only depends on z_i, we have that

$$\frac{\partial u_i}{\partial z_j} = 0 \text{ for } i \neq j.$$

For $i = j$,

$$\frac{\partial u_i}{\partial z_i} = f'(z_i) = \frac{e^{-z_i}}{(1+e^{-z_i})^2}.$$

Therefore, the gradient is the diagonal matrix,

$$\frac{\partial \mathbf{u}}{\partial \mathbf{z}} = \begin{bmatrix} f'(z_1) & 0 & \cdots & 0 \\ 0 & f'(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'(z_N) \end{bmatrix} = \mathrm{diag}(f'(z_1), f'(z_2), \cdots, f'(z_N)).$$

(b) Using chain rule with (13),

$$\frac{\partial u_i}{\partial W_{ij}} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} = f'(z_i)x_j,$$

and

$$\frac{\partial u_i}{\partial b_i} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial b_i} = f'(z_i).$$

(c) We again use chain rule with (13),

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial x_j} = f'(z_i)W_{ij}.$$

4. Let

$$L = \|\mathbf{y} - \mathbf{z}\|^2, \quad \mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b},$$

(a) What is the gradient $\partial L/\partial \mathbf{z}$?

(b) What are the gradients $\partial L/\partial \mathbf{W}$ and $\partial L/\partial \mathbf{b}$?

(c) What is the gradient $\partial L/\partial \mathbf{x}$?

4. (a) We have

$$L = \|\mathbf{y} - \mathbf{z}\|^2 = \sum_{j=1}^{N}(y_i - z_i)^2.$$

Hence, the components of the gradient $\partial J/\partial \mathbf{z}$ are,

$$\frac{\partial L}{\partial z_j} = 2(z_i - y_i).$$

(b) Using chain rule with (13),

$$\frac{\partial u_i}{\partial W_{ij}} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} = 2(z_i - y_i)x_j$$

and

$$\frac{\partial u_i}{\partial b_i} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial b_i} = 2(z_i - y_i).$$

(c) We again use chain rule with (13),

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial x_j} = 2(z_i - y_i)W_{ij}.$$

Section3:

1. Consider a neural network (1) acting on a mini-batch of training samples $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$. Suppose that $y_i \in \{1, \ldots, K\}$ so that the problem is K-class classification, we use the categorial cross-entropy loss (18) and a ReLU activation. Derive the back-propagation equations for all steps.

$$g_{\mathrm{loss}}(\mathbf{z}_i^{\mathrm{O}}, y_i) := \ln\left[\sum_{\ell=1}^{K} e^{z_{i\ell}^{\mathrm{O}}}\right] - \sum_{k=1}^{K} r_{ik}z_{ik}^{\mathrm{O}}, \tag{18}$$

1. We work our way backward starting at L along paths to get to the parameters:

- $L \to \mathbf{z}^{\mathrm{O}}$: The loss function is the categorial cross-entropy (18). Therefore, the gradient components are

$$\frac{\partial L}{\partial z_{ij}^{\mathrm{O}}} = \frac{\partial g_{\mathrm{loss}}(\mathbf{z}_i^{\mathrm{O}}, y_i)}{\partial z_{ij}^{\mathrm{O}}} = \frac{e^{z_{ij}^{\mathrm{O}}}}{\sum_{\ell=1}^{K} e^{z_{i\ell}^{\mathrm{O}}}} - r_{ij}.$$

- $\mathbf{z}^{\mathrm{O}} \to \mathbf{W}^{\mathrm{O}}, \mathbf{b}^{\mathrm{O}}, \mathbf{u}^{\mathrm{H}}$: We have $\mathbf{z}_i^{\mathrm{O}} = \mathbf{W}^{\mathrm{O}}\mathbf{u}_i^{\mathrm{H}} + \mathbf{b}^{\mathrm{O}}$. Therefore,

$$z_{ij}^{\mathrm{O}} = \sum_k W_{jk}^{\mathrm{O}}u_{ik}^{\mathrm{H}} + b_j^{\mathrm{O}}.$$

Hence, we have the derivatives,

$$\frac{\partial z_{ij}^{\mathrm{O}}}{\partial W_{jk}^{\mathrm{O}}} = u_{ik}^{\mathrm{H}}, \quad \frac{\partial z_{ij}^{\mathrm{O}}}{\partial b_j^{\mathrm{O}}} = 1, \quad \frac{\partial z_{ij}^{\mathrm{O}}}{\partial u_{ik}^{\mathrm{H}}} = W_{jk}.$$

Using chain rule,

$$\frac{\partial L}{\partial W_{jk}^{\mathrm{O}}} = \sum_{i=1}^{N}\frac{\partial L}{\partial z_{ik}^{\mathrm{O}}}\frac{\partial z_{ij}^{\mathrm{O}}}{\partial W_{jk}^{\mathrm{O}}} = \sum_{i=1}^{N}\frac{\partial L}{\partial z_{ik}^{\mathrm{O}}}u_{ik}^{\mathrm{H}},$$
$$\frac{\partial L}{\partial b_j^{\mathrm{O}}} = \sum_{i=1}^{N}\frac{\partial L}{\partial z_{ik}^{\mathrm{O}}}\frac{\partial z_{ij}^{\mathrm{O}}}{\partial b_j^{\mathrm{O}}} = \sum_{i=1}^{N}\frac{\partial L}{\partial z_{ik}^{\mathrm{O}}},$$
$$\frac{\partial L}{\partial u_{ik}^{\mathrm{H}}} = \sum_{j=1}^{N_o}\frac{\partial L}{\partial z_{ij}^{\mathrm{O}}}\frac{\partial z_{ij}^{\mathrm{O}}}{\partial u_{ik}^{\mathrm{H}}} \sum_{j=1}^{N_o}\frac{\partial L}{\partial z_{ij}^{\mathrm{O}}}W_{jk}.$$

- $\mathbf{u}^{\mathrm{H}} \to \mathbf{z}^{\mathrm{H}}$: We have

$$u_{ij}^{\mathrm{H}} = g_{\mathrm{act}}(z_{ij}^{\mathrm{H}}) = \max(0, z_{ij}^{\mathrm{H}}).$$

Hence,

$$\frac{\partial u_{ij}^{\mathrm{H}}}{\partial z_{ij}^{\mathrm{H}}} = \begin{cases} 1, & \text{if } z_{ij}^{\mathrm{H}} > 0 \\ 0, & \text{if } z_{ij}^{\mathrm{H}} < 0. \end{cases}.$$

Then, we apply chain rule,

$$\frac{\partial L}{\partial z_{ij}^{\mathrm{H}}} = \frac{\partial L}{\partial u_{ij}^{\mathrm{H}}}\frac{\partial u_{ij}^{\mathrm{H}}}{\partial z_{ij}^{\mathrm{H}}}.$$

- $\mathbf{z}^{\mathrm{H}} \to \mathbf{W}^{\mathrm{H}}, \mathbf{b}^{\mathrm{H}}$: We have $\mathbf{z}_i^{\mathrm{H}} = \mathbf{W}^{\mathrm{H}}\mathbf{x}_i + \mathbf{b}^{\mathrm{H}}$. Therefore,

$$z_{ij}^{\mathrm{H}} = \sum_k W_{jk}^{\mathrm{H}} x_{ik} + b_j^{\mathrm{H}}.$$

Hence, we have the derivatives,

$$\frac{\partial z_{ij}^{\mathrm{H}}}{\partial W_{jk}^{\mathrm{H}}} = x_{ik}, \quad \frac{\partial z_{ij}^{\mathrm{H}}}{\partial b_j^{\mathrm{H}}} = 1.$$

Using chain rule,

$$\frac{\partial L}{\partial W_{jk}^{\mathrm{H}}} = \sum_{i=1}^N \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}} \frac{\partial z_{ij}^{\mathrm{H}}}{\partial W_{jk}^{\mathrm{H}}} = \sum_{i=1}^N \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}} x_{ik},$$
$$\frac{\partial L}{\partial b_{jk}^{\mathrm{H}}} = \sum_{i=1}^N \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}} \frac{\partial z_{ij}^{\mathrm{H}}}{\partial b_j^{\mathrm{H}}} = \sum_{i=1}^N \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}}.$$

2. Suppose that we are given a mini-batch of training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$, and we try to fit a model to the data of the form,

$$\mathbf{z}_i = \mathbf{W}\mathbf{x}_i + \mathbf{b}, \quad \hat{y}_i = \sum_{j=1}^M \alpha_j \exp(z_{ij}),$$

for unknown parameters $\theta = (\mathbf{W}, \mathbf{b}, \alpha)$. Here M is the dimension of the hidden variables $\mathbf{z}_i$. We use the squared error loss function,

$$L = \sum_{i=1}^N L_i, \quad L_i = (y_i - \hat{y}_i)^2.$$

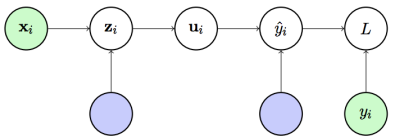
(a) Add an intermediate variable $\mathbf{u}_i = \exp(\mathbf{z}_i)$, by which we mean $u_{ij} = \exp(z_{ij})$. Draw the computation graph showing the mapping from the inputs $\mathbf{x}_i$ and the parameters θ to the loss function, L. Also, write the equations for each step in the computation graph.

(b) Write the back-propagation equations to obtain the gradients with respect to the parameters θ.

2. (a) If we substitute $\mathbf{u}_i = \exp(\mathbf{z}_i)$ we can write the mapping from the set of $\mathbf{x}_i$'s to L as

$$\mathbf{z}_i = \mathbf{W}\mathbf{x}_i + \mathbf{b}$$
$$\mathbf{u}_i = \exp(\mathbf{z}_i), \quad \hat{y}_i = \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{u}_i$$
$$L = \sum_{i=1}^N (y_i - \hat{y}_i)^2.$$

The computation graph is shown in Fig. 2.



(b) We perform back-propagation in the following steps:

- $L \to \hat{\mathbf{y}}$: The components of the gradient are:

$$\frac{\partial L}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i).$$

- $\hat{\mathbf{y}} \to \boldsymbol{\alpha}, \mathbf{u}$: We have

$$\hat{y}_i = \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{u}_i = \sum_{j=1}^M \alpha_j u_{ij} \Rightarrow \frac{\partial \hat{y}_i}{\partial u_{ij}}.$$

Therefore,

$$\frac{\partial \hat{y}_i}{\partial u_{ij}} = \alpha_j, \quad \frac{\partial \hat{y}_i}{\partial \alpha_j} = u_{ij}.$$

Applying chain rule,

$$\frac{\partial L}{\partial \alpha_j} = \sum_{i=1}^N \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial \alpha_j} = \sum_{i=1}^N \frac{\partial L}{\partial \hat{y}_i} u_{ij}$$
$$\frac{\partial L}{\partial u_{ij}} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial u_{ij}} = \sum_{i=1}^N \frac{\partial L}{\partial \hat{y}_i} \alpha_j.$$

- $\mathbf{u} \to \mathbf{z}$: Since $u_{ij} = \exp(z_{ij})$, we have the derivative,

$$\frac{\partial u_{ij}}{\partial z_{ij}} = \exp(z_{ij}).$$

Using chain rule:

$$\frac{\partial L}{\partial z_{ij}} = \frac{\partial L}{\partial u_{ij}} \frac{\partial u_{ij}}{\partial z_{ij}} = \frac{\partial L}{\partial u_{ij}} \exp(z_{ij}).$$

- $\mathbf{z} \to \mathbf{W}, \mathbf{b}$: Since $\mathbf{z}_i = \mathbf{W}\mathbf{x}_i + \mathbf{b}$, Therefore,

$$z_{ij} = \sum_k W_{jk} x_{ik} + b_j.$$

Hence, we have the derivatives,

$$\frac{\partial z_{ij}}{\partial W_{jk}} = u_{ik}, \quad \frac{\partial z_{ij}}{\partial b_j} = 1.$$

Applying chain rule,

$$\frac{\partial L}{\partial W_{jk}} = \sum_i \frac{\partial L}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial W_{jk}} = \sum_i \frac{\partial L}{\partial z_{ij}} u_{ik},$$
$$\frac{\partial L}{\partial b_j} = \sum_i \frac{\partial L}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial b_j} = \sum_i \frac{\partial L}{\partial z_{ij}}.$$

Homework 8: Convolutional Neural Networks

1. Let X and W be arrays,

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 0 \\ 0 & 3 & 3 & 3 & 0 \\ 0 & 3 & 2 & 3 & 0 \\ 0 & 3 & 2 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad W = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}.$$

Let Z be the 2D convolution (without reversal):

$$Z[i,j] = \sum_{k_1,k_2} W[k_1,k_2] X[i+k_1, j+k_2]. \tag{1}$$

Assume that the arrays are indexed starting at $(0,0)$.

(a) What are the limits of the summations over k_1 and k_2 in (1)?

(b) What is the size of the output $Z[i,j]$ if the convolution is computed only on the *valid* pixels (i.e. the pixel locations (i,j) where the summation in (1) does not exceed the boundaries of W or X).

(c) What is the largest positive value of $Z[i,j]$ and state one pixel location (i,j) where that value occurs.

(d) What is the largest negative value of $Z[i,j]$ and state one pixel location (i,j) where that value occurs.

(e) Find one pixel location where $Z[i,j] = 0$.

1. (a) Both indices go over the range of $W[k_1,k_2]$: $0 \le k_1, k_2 < 2$.

(b) Since, X is 6×5 and W is 2×2 and we are selecting valid locations only, the size will of Z will be

$$(6-2+1) \times (5-2+1) = 5 \times 4.$$

(c) We have that

$$Z[i,j] = X[i,j] + X[i+1,j] - X[i,j+1] - X[i+1,j+1].$$

So, $Z[i,j]$ will be the largest positive value when there is a large negative change across one column. This occurs at $(i,j) = (1,3)$:

$$Z[1,3] = X[1,3] + X[2,3] - X[1,4] - X[2,4] = 3+3-0-0 = 6.$$

We get the same value at $(2,3)$ and $(3,3)$.

(d) For a negative value, we need there to be a large positive change across one column, which occurs at

$$Z[1,0] = X[1,0] + X[2,0] - X[1,0] - X[2,0] = 0+0-3-3 = -6.$$

We get the same value at $(2,0)$ and $(3,0)$.

(e) You can take $(i,j) = (1,1)$ or $(1,2)$. For example,

$$Z[1,1] = X[1,1] + X[2,1] - X[1,2] - X[2,2] = 3+3-3-3 = 0.$$

2. Suppose that a convolutional layer of a neural network has an input tensor $X[i,j,k]$ and computes an output via a convolution and ReLU activation,

$$Z[i,j,m] = \sum_{k_1} \sum_{k_2} \sum_n W[k_1,k_2,n,m] X[i+k_1, j+k_2, n] + b[m],$$
$$U[i,j,m] = \max\{0, Z[i,j,m]\}.$$

for some weight kernel $W[k_1,k_2,n,m]$ and bias $b[m]$. Suppose that X has shape (48,64,10) and W has shape (3,3,10,20). Assume the convolution is computed on the *valid* pixels.

(a) What are the shapes of Z and U?

(b) What are the number of input channels and output channels?

(c) How many multiplications must be performed to compute the convolution in that layer?

(d) If W and b are to be learned, what are the total number of trainable parameters in the layer?

2. (a) Since each kernel in W is 3×3, each channel of the output is

$$(48-3+1) \times (64-3+1) = 46 \times 62.$$

There are 20 output channels, so Z is $46 \times 62 \times 20$.

(b) Since W is $3 \times 3 \times 10 \times 20$, there are 10 input channels and 20 output channels.

(c) Each output of $Z[i,j,m]$ requires summations over the indices

$$0 \le k_1, k_2 < 3, \quad 0 \le n < 10.$$

Therefore, there are (3)(3)(10) multiplications for each output of Z. Since there are (46)(62)(20) outputs, there are a total of

$$(46)(62)(20)(3)(3)(10) = 5.133(10)^6 \text{ multiplications.}$$

You can see why computing outputs in deep networks takes many operations.

(d) The number of parameters in W and b are:

$$W: \quad (3)(3)(10)(20) = 1800 \text{ parameters}$$
$$b: \quad 20 \text{ parameters.}$$

So, there are a total of 1820 parameters.

3. Suppose that a convolutional layer as a linear convolution followed by a sigmoid activation,

$$Z[i,j,m] = \sum_{k_1}\sum_{k_2}\sum_{n} W[k_1,k_2,n,m]X[i+k_1,j+k_2,n] + b[m],$$
$$U[i,j,m] = 1/(1+\exp(-Z[i,j,m])).$$

Suppose that during back-propagation, we have computed the gradient $\partial J/\partial U$ for some loss function J. That is, we have computed $\partial J/\partial U[i,j,m]$. Show how to compute the following:

(a) The gradient components $\partial J/\partial Z[i,j,m]$.

(b) The gradient components $\partial J/\partial W[k_1,k_2,n,m]$.

(c) The gradient components $\partial J/\partial X[i,j,n]$.

3. Suppose that a convolutional layer as a linear convolution followed by a sigmoid activation,

$$Z[i,j,m] = \sum_{k_1}\sum_{k_2}\sum_{n} W[k_1,k_2,n,m]X[i+k_1,j+k_2,n] + b[m],$$
$$U[i,j,m] = 1/(1+\exp(-Z[i,j,m])).$$

Suppose that during back-propagation, we have computed the gradient $\partial J/\partial U$ for some loss function J. That is, we have computed $\partial J/\partial U[i,j,m]$. Show how to compute the following:

(a) We have

$$\frac{\partial U[i,j,m]}{\partial Z[i,j,m]} = \frac{\exp(-Z[i,j,m])}{(1+\exp(-Z[i,j,m])^2} = U[i,j,m](1-U[i,j,m]).$$

By chain rule,

$$\frac{\partial J}{\partial Z[i,j,m]} = \frac{\partial J}{\partial U[i,j,m]}\frac{\partial U[i,j,m]}{\partial Z[i,j,m]} = \frac{\partial J}{\partial U[i,j,m]}U[i,j,m](1-U[i,j,m]).$$

(b) The gradient components $\partial J/\partial W[k_1,k_2,n,m]$. From the convolution equation,

$$\frac{\partial Z[i,j,m]}{\partial W[k_1,k_2,n,m]} = X[i+k_1,j+k_2,n].$$

By chain rule,

$$\begin{aligned}\frac{\partial J}{\partial W[k_1,k_2,n,m]} &= \sum_{i,j}\frac{\partial J}{\partial Z[i,j,m]}\frac{\partial Z[i,j,m]}{\partial W[k_1,k_2,n,m]} \\ &= \sum_{i,j}\frac{\partial J}{\partial Z[i,j,m]}X[i+k_1,j+k_2,n].\end{aligned}$$

(c) We want to first compute the partial derivatives,

$$\frac{\partial Z[i',j',m]}{\partial X[i,j,n]},$$

for all output components $Z[i',j',m]$ and inputs $X[i,j,n]$. Note that we had to add the indices i',j' at the output, to differentiate between the input indices i,j. To compute this derivative, we need to write $Z[i',j',m]$ in terms of the inputs $X[i,j,n]$. This is matter of re-indexing. First, rewrite the summation in the convolution as,

$$Z[i',j',m] = \sum_{k_1}\sum_{k_2}\sum_{n} W[k_1,k_2,n,m]X[i'+k_1,j'+k_2,n] + b[m].$$

All we have done here is replace i,j with i',j'. Next make the substitution,

$$i = i'+k_1, \quad j = j'+k_2 \Rightarrow k_1 = i-i', \quad k_2 = j-j'.$$

Then, we can sum over i,j instead of over k_1,k_2:

$$Z[i',j',m] = \sum_{i}\sum_{j}\sum_{n} W[i-i',j-j',n,m]X[i,j,n] + b[m].$$

Now, we have $Z[i',j',m]$ in terms of inputs $X[i,j,n]$.
From this, we see that

$$\frac{\partial Z[i',j',m]}{\partial X[i,j,n]} = W[i-i',j-j',n,m].$$

Hence, by chain rule,

$$\begin{aligned}\frac{\partial J}{\partial X[i,j,n]} &= \sum_{i'}\sum_{j'}\sum_{m}\frac{\partial J}{\partial Z[i',j',m]}\frac{\partial Z[i',j',m]}{\partial X[i,j,n]} \\ &= \sum_{i'}\sum_{j'}\sum_{m}\frac{\partial J}{\partial Z[i',j',m]}W[i-i',j-j',n,m].\end{aligned}$$

If you got this far, you will get full marks. But, if we let $k_1 = i-i'$ and $k_2 = j-j'$ and sum over k_1,k_2 instead of i',j', we get

$$\frac{\partial J}{\partial X[i,j,n]} = \sum_{k_1}\sum_{k_2}\sum_{n}\frac{\partial J}{\partial Z[i-k_1,j-k_2,m]}W[k_1,k_2,n,m].$$

We see that the gradient is also a convolution, but with the reversal.