B. M. D. S. Karunarathna

180308C

EN2550 : Image Processing and Machine Vision

ASSIGNMENT 04

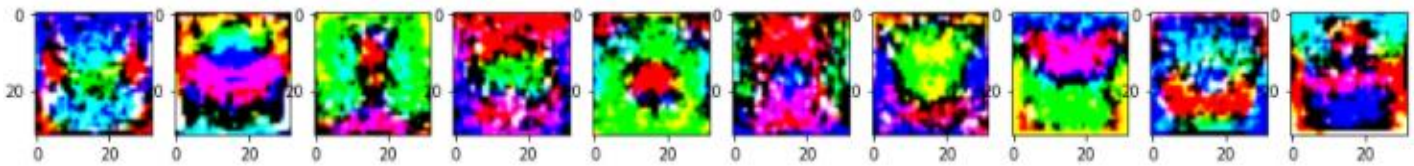
01)

(a)

```
std=1e-5
w1 = std*np.random.randn(Din, K)
b1 = np.zeros(K)
print("w1:", w1.shape)
print("b1:", b1.shape)

batch_size = Ntr
iterations =300
lr =1.8e-2
lr_decay=0.999
reg =5e-6
loss_history = []
train_acc_history = []
val_acc_history = []
seed = 0

for t in range(iterations):
    batch_indices=np.random.choice(Ntr,batch_size)
    x=x_train[batch_indices]
    y=y_train[batch_indices]
    y_pred=x.dot(w1)+b1
    loss = 1./batch_size*np.square(y_pred - y).sum() + reg *(np.sum(w1*w1))
    loss_history.append(loss)
    print('iteration %d / %d: loss %f'%(t,iterations,loss))

    dy_pred=1./batch_size*2.0*(y_pred-y)
    dw1=x.T.dot(dy_pred)+reg*w1
    db1=dy_pred.sum(axis=0)
    w1-=lr*dw1
    b1-=lr*db1
    lr*=lr_decay
```

(b)

```
for i in range(10):
    img=w1[:,i].reshape(32,32,3)
    axarr[i].imshow(img*1000)
plt.show()
```



(c)

```
x_t=x_train
y_pred=x_t.dot(w1)+b1
train_acc=1.0 - 1/(9*Ntr)*(np.abs(np.argmax(y_train,axis=1)-np.argmax(y_pred,axis=1))).sum()
train_loss=1./Ntr*np.square(y_pred - y_train).sum() + reg *(np.sum(w1*w1))

x_t=x_test
y_pred=x_t.dot(w1)+b1
test_acc=1.0 - 1/(9*Nte)*(np.abs(np.argmax(y_test,axis=1)-np.argmax(y_pred,axis=1))).sum()
test_loss=1./Nte*np.square(y_pred - y_test).sum() + reg *(np.sum(w1*w1))
```

- Initial learning rate - 0.018
- Training Loss - 0.7809448887589691
- Testing Loss - 0.7867270854419913
- Training accuracy - 0.7560644444444444
- Testing accuracy - 0.749

```
x_train-> (50000, 3072)
train_acc = 0.7560644444444444
train_loss = 0.7809448887589691
x_test-> (10000, 3072)
test_acc = 0.749
test_loss = 0.7867270854419913
```

02)

(a)

```
H=200
std=1e-5
w1 = std*np.random.randn(Din, H)
w2=std*np.random.randn(H,K)
b1 = np.zeros(H)
b2=np.zeros(K)

batch_size = Ntr
iterations =300
lr =1.5e-2
lr_decay=0.999
reg =5e-6
loss_history = []
train_acc_history = []
val_acc_history = []
seed = 0

for t in range(iterations):
    batch_indices=np.random.choice(Ntr,batch_size)
    x=x_train[batch_indices]
    y=y_train[batch_indices]
    h=1.0/(1.0+np.exp(-(x.dot(w1)+b1)))
    y_pred=h.dot(w2)+b2
    loss=1./batch_size*np.square(y_pred-y).sum()+reg*(np.sum(w2*w2)+np.sum(w1*w1))
    loss_history.append(loss)
    print('iteration %d / %d: loss %f'%(t,iterations,loss))

    dy_pred=1./batch_size*2.0*(y_pred-y)
    dw2=h.T.dot(dy_pred)+reg*w2
    db2=dy_pred.sum(axis=0)
    dh=dy_pred.dot(w2.T)
    dw1=x.T.dot(dh*h*(1-h))+reg*w1
    db1=(dh*h*(1-h)).sum(axis=0)
    w1-=lr*dw1
    w2-=lr*dw2
    b1-=lr*db1
    b2-=lr*db2
    lr*=lr_decay
```

(b)

```
x_t=x_train
h=1.0/(1.0+np.exp(-(x_t.dot(w1)+b1)))
y_pred=h.dot(w2)+b2
train_acc=1.0 - 1/(9*Ntr)*(np.abs(np.argmax(y_train,axis=1)-np.argmax(y_pred,axis=1))).sum()
train_loss=1./Ntr*np.square(y_pred-y_train).sum()+reg*(np.sum(w2*w2)+np.sum(w1*w1))

x_t=x_test
h=1.0/(1.0+np.exp(-(x_t.dot(w1)+b1)))
y_pred=h.dot(w2)+b2
test_acc=1.0 -1/(9*Nte)*(np.abs(np.argmax(y_test,axis=1)-np.argmax(y_pred,axis=1))).sum()
test_loss=1./Nte*np.square(y_pred-y_test).sum()+reg*(np.sum(w2*w2)+np.sum(w1*w1))
```

- Initial learning rate - 1.5e-2
- Training Loss - 0.7386732453059858
- Testing Loss - 0.760596695560745
- Training accuracy - 0 0.7755577777777778
- Testing accuracy - 0.7636666666666667

```
x_train-> (10000, 3072)
train_acc = 0.7755577777777778
train_loss = 0.7386732453059858
x_test-> (10000, 3072)
test_acc = 0.7636666666666667
test_loss = 0.760596695560745
```

03)

```
batch_size = 500
iterations =300
lr =1.5e-2
lr_decay=0.999
reg =5e-6
loss_history = []
train_acc_history = []
val_acc_history = []
seed = 0
rng = np.random.default_rng(seed=seed)
No_of_groups =int(Ntr/batch_size)

for t in range(iterations):
    indices=np.arange(Ntr)
    indices=np.split(indices, No_of_groups)
    for batch_indices in indices:
        rng.shuffle(batch_indices)
        x=x_train[batch_indices]
        y=y_train[batch_indices]
        h=1.0/(1.0+np.exp(-(x.dot(w1)+b1)))
        y_pred=h.dot(w2)+b2
        loss=1./batch_size*np.square(y_pred-y).sum()+reg*(np.sum(w2*w2)+np.sum(w1*w1))

        dy_pred=1./batch_size*2.0*(y_pred-y)
        dw2=h.T.dot(dy_pred)+reg*w2
        db2=dy_pred.sum(axis=0)
        dh=dy_pred.dot(w2.T)
        dw1=x.T.dot(dh*h*(1-h))+reg*w1
        db1=(dh*h*(1-h)).sum(axis=0)
        w1-=lr*dw1
        w2-=lr*dw2
        b1-=lr*db1
        b2-=lr*db2
        lr*=lr_decay
    loss_history.append(loss)
    print('iteration %d / %d: loss %f'%(t,iterations,loss))
```

(a)

- Training Loss – 0.5966890039883246
- Testing Loss – 0.7436201209993112
- Training accuracy – 0.8624288888888889
- Testing accuracy - 0.7778555555555555

```
x_train-> (10000, 3072)
train_acc = 0.8624288888888889
train_loss = 0.5966890039883246
x_test-> (10000, 3072)
test_acc = 0.7778555555555555
test_loss = 0.7436201209993112
```

(b)

	Gradient Descent	Mini batch stochastic gradient descent
Training Loss	0.7386732453059858	0.5966890039883246
Testing Loss	0.760596695560745	0.7436201209993112
Training accuracy	0.7755577777777778	0.8624288888888889
Testing accuracy	0.7636666666666667	0.7778555555555555

In mini batch stochastic gradient descent, the training dataset is split into small batches that are used to calculate the losses. We can say that the model is trained more when mini batch stochastic gradient is used rather than gradient descent.
The training and testing accuracies are increased and losses are decreased when stochastic gradient descent is carried out than the gradient descent.
Therefore we can consider that using mini batch stochastic gradient is optimum than the gradient descent.

04)

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.summary()

y_train_categorical = keras.utils.to_categorical(
    y_train, num_classes=10, dtype = "float32")

y_test_categorical = keras.utils.to_categorical(
    y_test, num_classes=10, dtype = "float32")

opt = tf.keras.optimizers.SGD(learning_rate=1.4e-2, momentum=0.9)
model.compile(optimizer=opt,
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
history = model.fit(x_train, y_train_categorical, epochs=10,batch_size=50,
                    validation_data=(x_test, y_test_categorical))
```

(a) Total Learnable Parameters = 73,418

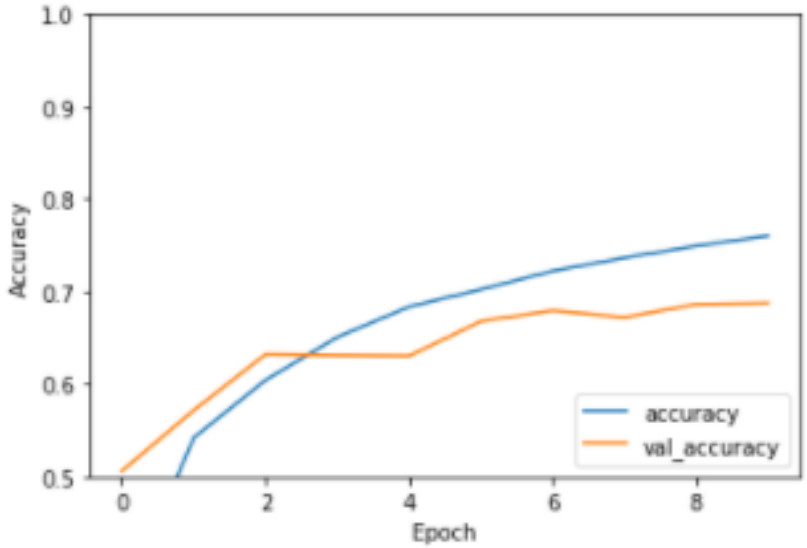
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 2, 2, 64)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 64)	16448
dense_1 (Dense)	(None, 10)	650

Total params: 73,418
Trainable params: 73,418
Non-trainable params: 0

(b) Learning rate = 0.014
Momentum = 0.9

(c) Training Loss = 0.6151214241981506
Testing Loss = 0.9769625067710876
Training accuracy = 0.7812600135803223
Testing accuracy = 0.6875

```
test_loss= 0.9769625067710876
test_acc= 0.6875
train_loss= 0.6151214241981506
train_acc= 0.7812600135803223
```



Github Link:

EN2550_Assignment04/Assignment04.ipynb at main · DineshikaKarunarathna/EN2550_Assignment04 (github.com)