Exercise 5: Solution

Affine Layers

Affine Layer– Forward

```
def affine_forward(x, w, b):

    out = None
    ###########################################################################
    # TODO: Implement the affine forward pass. Store the result in out.     #
    # You will need to reshape the input into rows.                          #
    ###########################################################################
    x_reshaped = np.reshape(x, (x.shape[0], -1))
    out = x_reshaped.dot(w) + b
    ###########################################################################
    #                             END OF YOUR CODE                           #
    ###########################################################################
    cache = (x, w, b)
    return out, cache
```

Remark: the input $\boldsymbol{x}$, weights $\boldsymbol{w}$ and bias $\boldsymbol{b}$ are saved in cache, such that the backward pass can access them.

Affine Layer – Backward

```python
def affine_backward(dout, cache):

    x, w, b = cache
    dx, dw, db = None, None, None
    ########################################################################
    # TODO: Implement the affine backward pass.                            #
    # Hint: Don't forget to average the gradients dw and db                #
    ########################################################################
    n = x.shape[0]

    dw = (np.reshape(x, (x.shape[0], -1)).T).dot(dout) / n
    dw = np.reshape(dw, w.shape)

    db = np.mean(dout, axis=0, keepdims=False)

    dx = dout.dot(w.T)
    dx = np.reshape(dx, x.shape)
    ########################################################################
    #                           END OF YOUR CODE                           #
    ########################################################################
    return dx, dw, db
```

Remark:
Make sure the dw and dx have the same shape as w and x.

Here, we take the average of the gradient, because otherwise we have the sum over all gradients $\sum_{i=1}^{N} \nabla L_i(\theta)$ of the entire minibatch.

Hint: This averaging operation can also be done in the backward pass of the loss function. If so. we don't average n in the layer.

Sigmoid – Forward

```python
def sigmoid_forward(x):
    """
    Computes the forward pass for a layer of sigmoids.

    :param x: Inputs, of any shape

    :return out: Output, of the same shape as x
    :return cache: out
    """
    out = None
    ########################################################################
    # TODO: Implement the Sigmoid forward pass.                            #
    ########################################################################
    out = 1 / (1 + np.exp(-x))
    ########################################################################
    #                           END OF YOUR CODE                           #
    ########################################################################
    cache = out
    return out, cache
```

Remark:
The output of sigmoid function is stored in the cache for the computation in backward pass.

Sigmoid – Backward

```python
def sigmoid_backward(dout, cache):
    """
    Computes the backward pass for a layer of sigmoids.

    :param dout: Upstream derivatives, of any shape
    :param cache: y, output of the forward pass, of same shape as dout

    :return dx: Gradient with respect to x
    """
    dx = None
    y = cache
    ########################################################################
    # TODO: Implement the Sigmoid backward pass.                           #
    ########################################################################
    dx = dout * y * (1 - y)
    ########################################################################
    #                             END OF YOUR CODE                         #
    ########################################################################
    return dx
```

Remark:
The derivative of sigmoid function is
is $sigmoid * (1 - sigmoid)$

Two Layer Network

Forward Pass

```
########################################################################
# TODO                                                                 #
# Implement the forward pass using the layers you implemented.         #
# It consists of 3 steps:                                              #
#   1. Forward the first affine layer                                  #
#   2. Forward the sigmoid layer                                       #
#   3. Forward the second affine layer                                 #
# (Dont't forget the caches)                                           #
########################################################################

# Forward first layer
h, cache_affine1 = affine_forward(X, W1, b1)

# Activation function
h_, cache_sigmoid = sigmoid_forward(h)

# Forward second layer
y, cache_affine2 = affine_forward(h_, W2, b2)

########################################################################
#                           END OF YOUR CODE                           #
########################################################################
```

Remark:
The weights and biases are initialized in __init__ of the class.

The first affine layer takes the input $\boldsymbol{X}$, weights $\boldsymbol{W1}$, bias $\boldsymbol{b1}$, and returns the output $\boldsymbol{h}$ for the input of next layer and cache $\boldsymbol{cache\_affine1}$ for later gradient computation.

Same procedure applies to other layers.

Backward Pass

```
##########################################################################
# TODO                                                                   #
# Implement the backward pass using the layers you implemented.         #
# Like the forward pass, it consists of 3 steps:                         #
#   1. Backward the second affine layer                                  #
#   2. Backward the sigmoid layer                                        #
#   3. Backward the first affine layer                                   #
# You should now have the gradients wrt all model parameters             #
##########################################################################

# Backward second layer
dh_, dW2, db2 = affine_backward(dy, cache_affine2)

# Backward activation function
dh = sigmoid_backward(dh_, cache_sigmoid)

# Backward first layer
dx, dW1, db1 = affine_backward(dh, cache_affine1)

##########################################################################
#                             END OF YOUR CODE                           #
##########################################################################
```

Review of Exercise 5

https://docs.google.com/forms/d/e/1FAIpQLSdfENBvUMsnyXkO7Bf_VlNaU9TrEv5EBWYMGhq8Q77tCrrjqg/viewform?usp=sf_link

Questions? Moodle

☺