Worksheet 1

Deep Learning: Problem Formulation

1. You have a single hidden-layer neural network for a binary classification task. The input is $X \in \mathbb{R}^{n \times m}$, output $\hat{y} \in \mathbb{R}^{1 \times m}$ and true label $y \in \mathbb{R}^{1 \times m}$. The squared-error loss function is $\mathcal{J}$. The forward propagation equations are:

$$\begin{aligned} z &= WX + B \\ \hat{y} &= \sigma(z) \\ \mathcal{J} &= (\hat{y} - y)^2 \end{aligned}$$

   (a) We want to compute how to change the weights W based on errors in the loss function $\mathcal{J}$. Define a partial derivative D to compute this.

   Solution: We want to compute $\partial\mathcal{J}/\partial W$.

   (b) Express D in terms of a sequence of partial derivatives in the network, i.e., including $\partial z/\partial X$.

   Solution: To compute $\partial\mathcal{J}/\partial W$, we can evaluate

$$\frac{\partial \mathcal{J}}{\partial W} = \frac{\partial \mathcal{J}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial W}$$

   (c) Compute a closed-form expression for D.

   Solution: We must compute each of the terms:

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial \hat{y}} &= 2(\hat{y} - y) \\ \frac{\partial \hat{y}}{\partial z} &= \sigma(z)[1 - \sigma(z)] \\ \frac{\partial z}{\partial W} &= X \end{aligned}$$

   This gives us $2(\hat{y} - y)\sigma(z)[1 - \sigma(z)]X$, which we can transform to

$$\begin{aligned} 2(\hat{y} - y)\sigma(z)[1 - \sigma(z)]X &= 2X[\sigma(z) - y]\sigma(z)[1 - \sigma(z)] \\ &= 2X\sigma(WX + B)[\sigma(WX + B) - y][1 - \sigma(WX + B)] \end{aligned}$$

   (d) Draw the computation graph corresponding to this network.

   (e) Write out the TensorFlow code for forward inference in the network.

Worksheet 1

```
# Python optimisation variables
learning_rate = 0.5
epochs = 10
batch_size = 100

# declare the training data placeholders
# input x - for 28 x 28 pixels = 784
x = tf.placeholder(tf.float32, [None, 784])
# declare the output data placeholder: 10 digits
y = tf.placeholder(tf.float32, [None, 10])

# now declare the weights connecting the input to the hidden layer
W1 = tf.Variable(tf.random_normal([784, 300], stddev=0.03), name='W1')
b1 = tf.Variable(tf.random_normal([300]), name='b1')

# calculate the output of the hidden layer
z = tf.add(tf.matmul(x, W1), b1)

# now calculate the hidden layer output -  a sigmoid activated
# output layer
y_ = tf.nn.sigmoid(z)

MSE = tf.reduce_sum((y_ -y) **2))


# add an optimiser
optimiser = tf.train.GradientDescentOptimizer(learning_rate=
    learning_rate).minimize(MSE)

# finally setup the initialisation operator
init_op = tf.global_variables_initializer()

# define an accuracy assessment operation
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{d}{dx}(1+e^{x})^{-1} \\
&= (1+e^{-x})^{-2}(e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^{2}} \\
&= \frac{1}{(1+e^{-x})}\frac{e^{-x}}{(1+e^{-x})} \\
&= \frac{1}{(1+e^{-x})}\frac{(1+e^{-x})-1}{(1+e^{-x})} \\
&= \frac{1}{(1+e^{x})}\left[\frac{(1+e^{x})}{(1+e^{x})} - \frac{1}{(1+e^{x})}\right] \\
&= \frac{1}{(1+e^{x})}\left[1 - \frac{1}{(1+e^{x})}\right] \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}$$

2. You are solving the binary classification task of classifying images as cat vs. non-cat. You design a CNN with a single output neuron. Let the output of this neuron be z. The final output of your network, $\hat{y}$ is given by: $\hat{y} = \sigma(ReLU(z))$.

   You classify all inputs with a final value $\hat{y} \geq 0.5$ as cat images. What problem are you going to encounter?

   Solution: $ReLU(z) = \max(0, z) \geq 0 \ \forall z$.

   $\sigma(w) \geq 0.5 \ \forall w \geq 0$.

   Hence $\hat{y} = \sigma(ReLU(z)) \geq 0.5 \ \forall z$ for $w = ReLU(z)$.

3. You train a simple network in which the final output of your network, $\hat{y}$ is given by a sigmoid activation function: $\hat{y} = \sigma(Wx + b)$.

   (a) If you initialise the weights in W, b to be large numbers, show analytically that the network will not learn for input $x \geq 0$.

   (b) If you initialise only the weights in b to be large numbers, will you have the same problem? Again, show why or why not.

4. You are given the following piece of code for forward propagation through a single hidden layer in a neural network. This layer uses the sigmoid activation. Identify and correct the error.

Worksheet 1

```
1 import numpy as np
2 def forward_prop(W, a_prev, b):
3     z = W*a_prev + b
4     a = 1/(1+np.exp(-z)) #sigmoid
5 return a
```

Solution: z = np.matmul(W, a prev) + b OR z = np.dot(W, a prev) + b