CS7643 Deep Learning: Homework 1

Yousef Emam[1*]

September 26, 2019

*[1]Y. Emam is with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA emamy@gatech.edu

1 Gradient Descent

1.1 Optimizer of Unconstrained Opt. Problem

Function to minimize:

$$g(w) = f(w^t) + (w - w^t)^\top \nabla f(w^t) + \frac{\lambda}{2}||w - w^t||^2$$

Take the gradient w.r.t. w and set it to 0:

$$\nabla g(w^*) = \nabla f(w^t) + \lambda(w^* - w^t) = 0,$$

which implies:

$$w^* = w^t - \frac{1}{\lambda}\nabla f(w^t).$$

This means that gradient descent is the optimal update with respect to the unconstrained problem if $\eta = \frac{1}{\lambda}$.

1.2 Prove Lemma

Q1.2

INCOMPLETE ANSWER

Let $f(v^{(t)}) = \frac{1}{2}\|v^{(t)}\|^2 \Rightarrow \nabla f(v^{(t)}) = v^{(t)}$

$$\Rightarrow f(w^*) \geq f(v^{(t)}) + < w^* - v^{(t)}, \nabla f(v^{(t)}) >$$

$$\Rightarrow \frac{\|w^*\|^2}{2} \geq \frac{1}{2}\|v_t\|^2 + < w^* - v^{(t)}, v^{(t)} >$$

Note that $v^{(t)} = \frac{-w^{(t)} + w^{(t-1)}}{\eta}$

$$\Rightarrow f(w^*) \geq \frac{1}{2}\|v^{(t)}\|^2 + < w^* + \frac{w^{(t)} - w^{(t-1)}}{\eta}, v^{(t)} >$$

$$\Rightarrow T f(w^*) \geq \frac{1}{2}\sum_{t=1}^{T}\|v^{(t)}\|^2 + \sum_{t=1}^{T} < w^* + \frac{w^{(t)} - w^{(t-1)}}{\eta}, v^{(t)} >$$

Figure 1: Question 1b- Scanned Answer

1.3 Bound Convergence of Gradient Descent

Q1. 3

$$\bar{w} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$$

$$f(\bar{w}) - f(w^*) = f\left(\frac{1}{T}\sum_{t=1}^{T} w^{(t)}\right) - f(w^*) \leq \frac{1}{T}\sum_{t=1}^{T}\left(f(w^{(t)}) - f(w^*)\right) \quad (1)$$

$$f(w^*) \geq f(w^{(t)}) + \langle w^* - w^{(t)}, \nabla f(w^{(t)})\rangle$$

$$\Rightarrow -f(w^*) \leq -f(w^{(t)}) + \langle w^* - w^{(t)}, \nabla f(w^{(t)})\rangle \quad (2)$$

$\Rightarrow$ Plug (2) into RHS of (1):

$$\Rightarrow f(\bar{w}) - f(w^*) \leq \frac{1}{T}\sum_{t=1}^{T}\left(\langle w^{(t)} - w^*, \nabla f(w^{(t)})\rangle\right) \leftarrow \text{Desired Expression}$$

$$\leq \frac{1}{T}\left(\frac{\|w^*\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T}\|\nabla f(w^{(t)})\|^2\right)$$

Plug in upper bounds & $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$

$$\leq \frac{1}{T}\left(\frac{B\sqrt{\rho^2 T}}{2} + \frac{\rho\sqrt{T}}{2}\right)$$

$$= \boxed{\frac{1}{\sqrt{T}}\left(\frac{\rho}{2}(B+1)\right)}$$

Figure 2: Question 1c- Scanned Answer

1.4 SGD Improvement Guarantee

Given function:

$$f(w) = \frac{1}{2}(w-2)^2 + \frac{1}{2}(w+1)^2 = (w - \frac{1}{2})^2 + 2.25$$

Gradient is given by:

$$\nabla f(w) = (w-2) + (w+1)$$

Assume $w^t = 0$, and $N = 2$ is selected:

$$f(0) = (-\frac{1}{2})^2 + 2.25 = 2.5,$$

and

$$w^{t+1} = 0 - \eta \nabla f_2(0) = -\eta.$$

Then:

$$f(w^{t+1}) = (-1/2 - \eta)^2 + 2.25 \geq f(w^t = 0) = (-\frac{1}{2})^2 + 2.25 \,\forall \eta > 0.$$

The above provides a counter example proving that SGD is not guaranteed to decrease the overall loss function in every iteration.

2 Automatic Differentiation (Figure 7)

2.1 Compute the value of f at $\vec{w} = (1, 2)$

$$f(1, 2) = [7.80207426 * 10^{24}, 2.73105858]$$

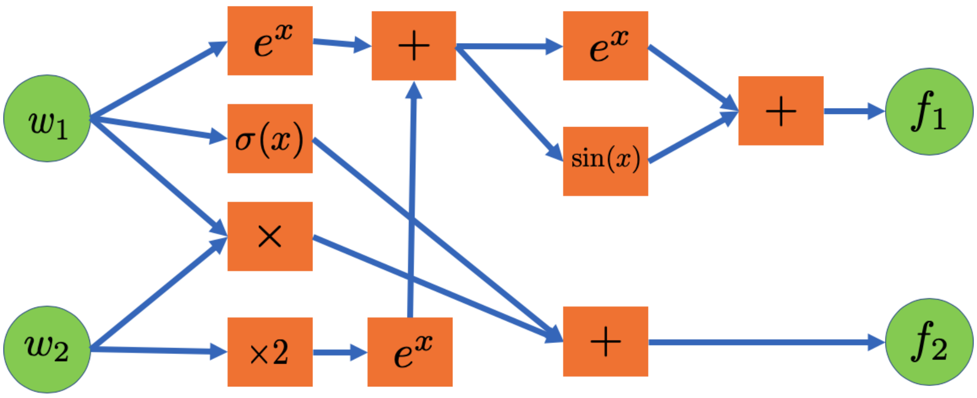

Figure 3: Question 2a- Computational Graph

```python
def forward(w):
    w1 = w[0]
    w2 = w[1]
    temp = np.exp(w1)+np.exp(2*w2)
    f1 = np.exp(temp) + np.sin(temp)
    f2 = w1*w2 + sigmoid(w1)
    return np.array([f1,f2])
```

Figure 4: Question 2a- Forward Pass

2.2 Compute the Jacobian using num. diff. with $\Delta w = 0.01$

$$\frac{\partial f(1,2)}{\partial \vec{w}} \approx \begin{bmatrix} 2.18016202 * 10^{25}, 2.19655301 \\ 6.93442399 * 10^{30}, 1.00000000 \end{bmatrix}.$$

Computed using the finite distance formula. Specifically:

$$f'(x) = (f(w+h) - f(w-h))/2h.$$

```
# Q2.b
print("Numerical Differentation: ")
delta_w1 = np.array([0.1, 0])
delta_w2 = np.array([0, 0.1])
print((forward(w+delta_w1)-forward(w-delta_w1))/(2*0.1))
print((forward(w+delta_w2)-forward(w-delta_w2))/(2*0.1))
```

Figure 5: Question 2b

2.3 Compute the Jacobian using forward mode auto-diff.

$$\frac{\partial f(1,2)}{\partial \vec{w}} = \begin{bmatrix} 2.12082367 * 10^{25}, 2.19661193 \\ 8.51957643 * 10^{26}, 1.00000000 \end{bmatrix}.$$

```
def forward_auto(w):
    w1 = w[0]
    w2 = w[1]
    dw1 = 1
    dw2 = 1
    temp = np.exp(w1)+np.exp(2*w2)
    dtemp_dw1 = np.exp(w1)*dw1
    dtemp_dw2 =  2*np.exp(2*w2)*dw2
    f = np.zeros((2,1))
    f[0] = np.exp(temp) + np.sin(temp)
    f[1] = w1*w2 + sigmoid(w1)
    df_dw = np.zeros((2,2))
    df_dw[0,0] = np.exp(temp)*dtemp_dw1 + np.cos(temp)*dtemp_dw1
    df_dw[1,0] = np.exp(temp)*dtemp_dw2 + np.cos(temp)*dtemp_dw2
    df_dw[0,1] = dw1*w2 + sigmoid(w1)*(1-sigmoid(w1))*dw1
    df_dw[1,1] = w1*dw2
    return (f, df_dw)
```

Figure 6: Question 2c- Forward Auto-Diff

2.4 Compute the Jacobian using backward mode auto-diff.

$$\frac{\partial f(1,2)}{\partial \vec{w}} = \begin{bmatrix} 2.12082367 * 10^{25}, 2.19661193 \\ 8.51957643 * 10^{26}, 1.00000000 \end{bmatrix}.$$

The fact that this is the same result obtained from the forward mode auto-differentiation is not surprising since both forward and backward mode auto-differentiation compute exact derivative.

```
def backward_auto(w):
    w1 = w[0]
    w2 = w[1]
    dw1 = 1
    dw2 = 1
    temp = np.exp(w1)+np.exp(2*w2)
    f = np.zeros((2,1))
    f[0] = np.exp(temp) + np.sin(temp)
    f[1] = w1*w2 + sigmoid(w1)

    df1_dtemp = np.exp(temp) + np.cos(temp)
    df1_w1 = df1_dtemp * np.exp(w1)
    df1_w2 = df1_dtemp * 2 * np.exp(2*w2)
    df_dw[0,0] = df1_w1
    df_dw[1,0] = df1_w2
    df_dw[0,1] = w2 + sigmoid(w1)*(1-sigmoid(w1))
    df_dw[1,1] = w1
    return (f, df_dw)
```

Figure 7: Question 2d- Backward Auto-Diff

2.5 Don't you love that software does this for us?

Yes.

3 Q3: Directed Acyclic Graphs

3.1 If G is a DAG, then G has a topological order

Using the Lemma that every DAG has at least 1 node with in-degree 0, we can construct a topological ordering using the following algorithm:

```
topOrder = {}
while G is not empty do
    temp ← {v ∈ V : deg_in(v) = 0}
    topOrder ← topOrder ⋃ enumerate(temp)
    G ← remove(G, {v ∈ V : deg_in(v) = 0})
end
return topOrder
```

Algorithm 1: Create Topological Order

The loop is guaranteed to terminate since when 0 in-degree nodes ($\{v \in V : \deg_{in}(v) = 0\}$) are removed from the graph, along with their edges, the resulting graph is also a DAG. Therefore, the resulting graph also has at least one node with in-degree 0. Moreover, in this topological ordering, if $n_{v_i} < n_{v_j}$ then there can exist a directed path from v_i to v_j but no path from v_j to v_i.

3.2 If G has a topological order, then G is a DAG

Since many topological orderings can be generated from a DAG, no algorithm can be use to reconstruct the exact DAG given a topological ordering. However, the statement can be proven using the fact that if $n_{v_i} < n_{v_j}$ then there can exist a directed path from v_i to v_j but no path from v_j to v_i. That is simply because if there existed a directed cycle in G, then there must exist two nodes, v_{k_1} and v_{k_2}, such that there is a directed graph from v_{k_1} to v_{k_2} and vice-versa. This in turn implies that $n_{v_{k_1}} > n_{v_{k_2}}$ and $n_{v_{k_2}} > n_{v_{k_1}}$ which is impossible.