Q1

1.1.1

As the model is over parameterized, then by considering the gradient of L w.r.t W, $\nabla_w L = \frac{\partial}{\partial w}(\|Xw-t\|^2) = 2X^T(Xw-t)$

as $Xw-t$ is a vector, when X^T multiplied with it, the resulting gradient can be treated as a linear combination of rows of X (columns of X^T)

Now consider a minibatch from X;

as each element in the minibatch is still a row of X, according to minibatch weight update method, the weight update can be treated as the sum of each minibatch's vector x_i with scalar value $(x_i w - t_i)$.

Which, can be treated as a linear combination of elements in minibatch.

Consider the linear combination of elements from minibatch:

as the span of all elements in minibatch is a subset of X's row space, minibatch linear combination can also be considered a linear combination of rows of X, where those rows not in minibatch has a scalar coefficient zero.

As the update of minibatch gradient descent can be treated as a linear combination of rows of X, and the weight update converges, can write $w = X^T a$. By HW1, the result of a is $(XX^T)^{-1}t$,

thus $w = X^T(XX^T)^{-1}t$, as desired.

Now need to show the weight acquired has minimum norm.

Consider a zero loss solution $\hat{w}$; (will apply the intuition from HW1 3.3.1, that x is perpendicular to w-space)

Note the model is overparameterized, thus the space of 'w' has infinitely

many possible values; by previous deduction, as gradient-descent-updated weight w is a linear combination of X's rows, will show the span of X's rows (thus w) is perpendicular to $\hat{w}$'s space.

Consider normal form of weight vector $w, \hat{w}$:

$$(w-\hat{w})^T w = \left(X^T(XX^T)^{-1}t - \hat{w}\right)^T \left(X^T(XX^T)^{-1}t\right)$$

$$= t^T\left[(XX^T)^{-1}\right]^T XX^T(XX^T)^{-1}t - \hat{w}^T X^T(XX^T)^{-1}t$$

$$= t^T(XX^T)^{-1}t - \hat{w}^T X^T(XX^T)^{-1}t$$

as $\hat{w}$ is a zero loss weight, $\hat{w}^T X^T = t^T$

thus $(w-\hat{w})^T w = 0$, indicating w and $\hat{w}$ are indeed perpendicular.

Thus by geometry, the distance $\|w\|$ is the shortest distance to the space of weight from zero initialization, thus has minimum norm.

Therefore minibatch gradient descent with zero initialization converges to a minimized norm solution.

1.2.1:

Will show python code and parts of output result.

realizing that $w_{final} = X^T(XX^T)^{-1}t = \begin{bmatrix}2\\1\end{bmatrix}\left(\begin{bmatrix}2 & 1\end{bmatrix}\begin{bmatrix}2\\1\end{bmatrix}\right)^{-1} \cdot 2$

$$= \left[\frac{4}{5}, \frac{2}{5}\right]^T$$

The following code trains the gradient using adaptive method:

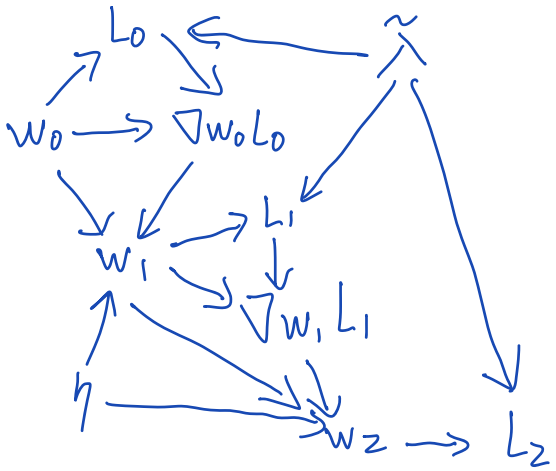
```python
import math

X = [2, 1]

t = 2

w0 = [0, 0]

l = 0.01

v = [0.0000000001, 0.0000000001]

def compute_loss(weight):
    return X[0] * weight[0] + X[1] * weight[1] - t

def compute_derivative(loss):
    return [2 * X[0] * loss, 2 * X[1] * loss]

def compute_v(previous_v, derivative):
    return [0.9 * previous_v[0] + 0.1 * (derivative[0] ** 2), 0.9 * previous_v[1] + 0.1 * (derivative[1] ** 2)]

def gradient_update(derivative, weight, v):
    return [weight[0] - (l * derivative[0] / math.sqrt(v[0])), weight[1] - (l * derivative[1] / math.sqrt(v[1]))]

for i in range(200):
    loss = compute_loss(w0)
    derivative = compute_derivative(loss)
    v = compute_v(v, derivative)
    w0 = gradient_update(derivative, w0, v)
    print(str(i) + ": " + str(w0))
```

```
0: [0.03162277660146144, 0.03162277660794406]
1: [0.054029072356855196, 0.054029072355952806]
2: [0.07251527044394393, 0.07251527044291373]
3: [0.08873738067741722, 0.08873738067630438]
4: [0.1034580687289188, 0.10345806872774721]
5: [0.11710026805811825, 0.11710026805690235]
6: [0.12992612967722628, 0.12992612967597558]
91: [0.6653212782695349, 0.6653212782680443]
92: [0.6656669649764252, 0.6656669649749346]
93: [0.6659377148450333, 0.6659377148435426]
94: [0.6661458119294372, 0.6661458119279465]
95: [0.6663025437576482, 0.6663025437561575]
96: [0.6664180390901754, 0.6664180390886847]
97: [0.6665011662555964, 0.6665011662541057]
98: [0.6665594934059184, 0.6665594934044277]
99: [0.6665993074715474, 0.6665993074700567]
100: [0.6666256844818009, 0.6666256844803102]
101: [0.666642600668602, 0.6666426006671113]
189: [0.6666666666671636, 0.6666666666656729]
190: [0.6666666666671636, 0.6666666666656729]
191: [0.6666666666671636, 0.6666666666656729]
192: [0.6666666666671636, 0.6666666666656729]
193: [0.6666666666671636, 0.6666666666656729]
194: [0.6666666666671636, 0.6666666666656729]
195: [0.6666666666671636, 0.6666666666656729]
196: [0.6666666666671636, 0.6666666666656729]
197: [0.6666666666671636, 0.6666666666656729]
198: [0.6666666666671636, 0.6666666666656729]
199: [0.6666666666671636, 0.6666666666656729]
```

The out put indicates the weight converges to approximately $[\frac{2}{3}, \frac{2}{3}]$ which is not $[\frac{4}{5}, \frac{2}{5}]$ $\Rightarrow$ a counter example.

Thus RMSProp doesn't always converge to minimum norm solution.

2.1.1:



2.1.2:

forward pass: $O(1)$ by treating d, N as constant, and consider memory reusing.

backward pass: $O(k)$ as each iteration's result needs to be stored.

2.2.1

w_1 is the result of gradient descent from w_0:

$$w_1 = w_0 - \eta \nabla_{w_0} L = w_0 - \eta \nabla_{w_0} (Xw_0 - t)$$

$$= w_0 - \frac{2\eta}{n} X^T (Xw_0 - t)$$

Thus to find loss L_1:

$$L_1 = \frac{1}{n} \|Xw_1 - t\|^2 = \frac{1}{n} \|X(w_0 - \frac{2\eta}{n} X^T a) - t\|^2 \quad \text{where } a = Xw_0 - t$$

2.2.3

$\nabla_\eta L_1 = \frac{\partial}{\partial \eta} \frac{1}{n} \| X(w_0 - \frac{2\eta}{n} x^T a) - t \|^2$ (as it is the sum of element wise square,)

$= \frac{2}{n} (X(w_0 - \frac{2\eta}{n} x^T a) - t) \frac{\partial}{\partial \eta} (X(w_0 - \frac{2\eta}{n} x^T a) - t)$

$= \frac{2}{n} (a - \frac{2\eta}{n} x x^T a)(- \frac{2\eta}{n} x x^T a)$

now set gradient to zero:

$$\frac{2}{n} a^T (I - \frac{2\eta}{n} x x^T)(-\frac{2\eta}{n} x x^T) a = 0$$

$$-\frac{4}{n^2} a^T x x^T a + \frac{8\eta}{n^3} a^T x x^T x x^T a = 0$$

$$\eta = \frac{\frac{4}{n^2}(a^T x)^2}{\frac{8}{n^3} a^T x x^T x x^T a} = \frac{n}{2} \frac{(a^T x)^2}{a^T x x^T x x^T a}$$

2.3.1:

$L = \frac{1}{n} \| X w_0 - t \|^2 + \tilde{\lambda} \| w_0 \|^2$

$\Rightarrow \frac{\partial L}{\partial w_0} = \frac{2}{n} (X w_0 - t)^T X + (\tilde{\lambda} w_0 I) 2$

$\hexagram\ w_1 = w_0 - \eta \cdot \frac{\partial L}{\partial w_0} = w_0 - \frac{2\eta}{n} (X^T X w_0 - X^T t) - 2\tilde{\lambda} I w_0 \eta$

$\hexagram\ w_1 = (1-\lambda) w_0 - \eta \cdot \frac{2}{n} (X^T X w_0 - X^T t)$

2.3.2:

$w_0 - \frac{2\eta}{n} (X^T X w_0 - X^T t) - 2\tilde{\lambda} I w_0 \eta = (1-\lambda) w_0 - \eta \cdot \frac{2}{n} (X^T X w_0 - X^T t)$

$$(1-\lambda)w_0 = w_0 - 2\tilde{\lambda}\eta w_0$$

$$1-\lambda = 1-2\tilde{\lambda}\eta \qquad \tilde{\lambda} = \frac{\lambda}{2\eta}$$

3.1: flip & filter:

$flip = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$; use zero padding with length 1 to ensure output dimension matches;

final result:

$$\begin{bmatrix} 0 & -1 & -2 & -3 & -2 \\ -2 & -3 & -3 & -2 & -1 \\ -1 & -1 & -1 & 1 & 1 \\ 2 & 2 & 2 & 1 & 1 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

it's a horizontal line detector;

3.2:

CNN:

Neurons: $32\times32 + 16\times16 + 16\times16 + 8\times8 + 8\times8$

parameters: $3\times3 + \quad +3\times3 \quad +3\times3$

FCNN:

neurons: $32\times32+16\times16+16\times16+8\times8+8\times8$

parameters: $(32\times32)^2 + \quad (16\times16)^2 + \quad (8\times8)^2$

Having more trainable parameters can significantly prolong

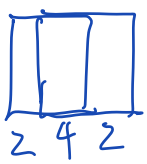
training time

3.3:

after layer 1 pooling : each neuron has receptive field 6×6

consider layer 2 convolution : with stride 2:

each adjacent neuron has 4 block coincidence on receptive field



⇒ as there are 5 neurons,
a total of $6+2+2+2+2=14$ width
receptive field is represented by one neuron
after layer 2 convolution layer ⇒ 14×14 receptive field

number of layers and stride of filter can affect
receptive field size.