Exercise 7.1

a)
b) See **multilayer_perceptron.py**:
Accuracy:

- (i) Gradient Descent: 0.8865
- (ii) Gradient Descent with Momentum with momentum parameter = 0.5: 0.9015
- (iii) AdaGrad with initial accumulator value = 0.1: 0.6659
- (iv) RMSProp with decay = 0.9, momentum = 0: 0.9321

c)
Update rule of Gradient Descent:

$$\theta = \theta - \epsilon_k.G$$

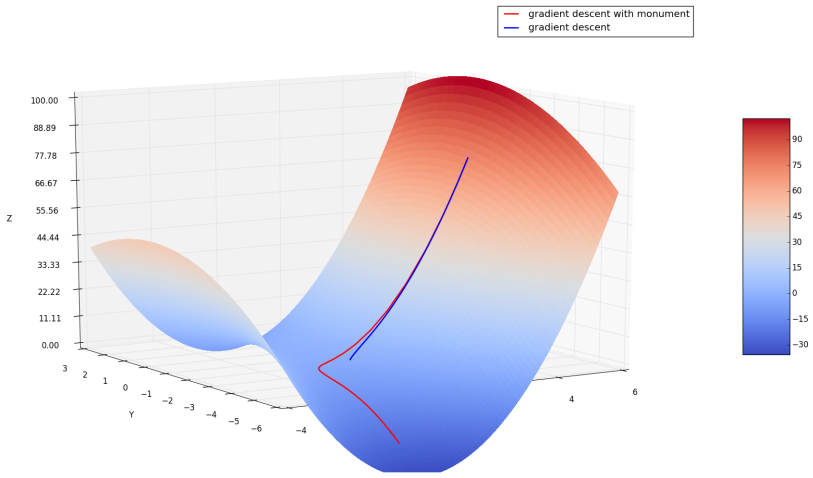
Update rule of AdaGrad:

$$r = r + G^T G$$
$$\theta = \theta - \frac{\epsilon}{\sqrt{r}}.G$$

In AdaGrad optimizer, with gradient G having large magnitude, we have accumulation variable r also large. Observe that the pure learning rate ϵ is already small like in Gradient Descent, this will lead to the situation that we update the parameter θ with gradient G scaled by an amount of $\frac{\epsilon}{\sqrt{r}}$, which is very small. Hence, we would need more training epochs with AdaGrad in comparison with Gradient Descent to reach the local minimum. However, both 2 optimizers are tested with the same number of epochs (5 epochs), it is clear that Gradient Descent performs better then AdaGrad.

Exercise 7.2

a) See **7.2ab.py**:
b) See **7.2ab.py**:
c)

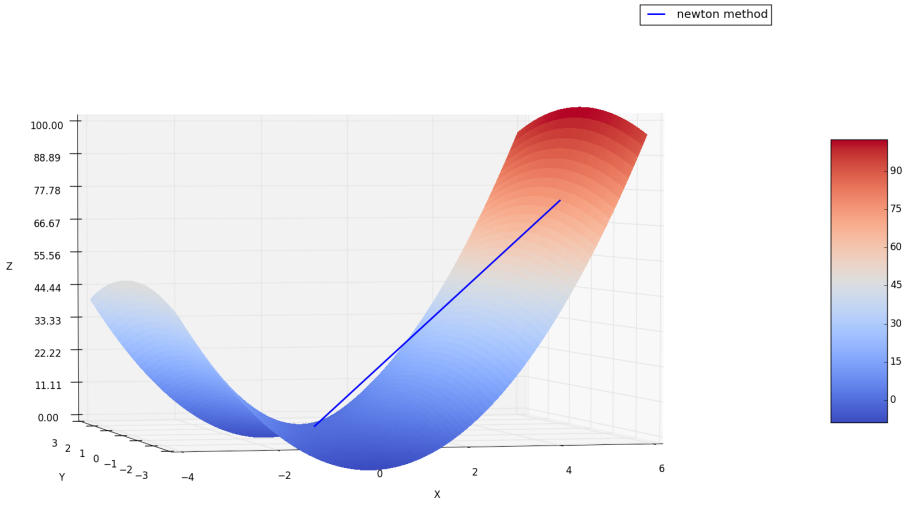


d)
After 30 iterations:

- GD stops at point: (0.781278030833, -1.811361584103) with loss = -1.449844703977
- GD with Monument stops at point: (0.017928494126, -5.035721987536) with loss = -25.357531643053

Hence, we can see that Gradient Descent with Monument works better than Normal Gradient Descent in minimizing this function $3x^2 - y^2$.

e) See **7.2e.py**



Analyzing the steps of Newton method we have:

- Step 1: old_(x,y) = (5, -1); Gradient = (5, -1); new_(x,y) = (0, 0); loss = 0
- Step 2: old_(x,y) = (0, 0); Gradient = (0, 0); new_(x,y) = (0, 0); loss = 0
- Step 3: old_(x,y) = (0, 0); Gradient = (0, 0); new_(x,y) = (0, 0); loss = 0
- Step 4: old_(x,y) = (0, 0); Gradient = (0, 0); new_(x,y) = (0, 0); loss = 0
- Step 5: old_(x,y) = (0, 0); Gradient = (0, 0); new_(x,y) = (0, 0); loss = 0);

After the first step, the method will jump to the saddle point (0, 0) which has gradient zero. The next steps from 2 to 5 will stay at this saddle point. This is an example showing why plain vanilla Newton method is not suited for training deep neural networks. The reason is in deep neural network, the loss function is typically non-convex (has a lot of extreme points), hence, when we need to minimize the loss function, we actually need some methods that could help us get away from these local maximum or saddle points. Meanwhile, Newton method always looks at points with zero gradient (could be local minimum, local maximum or saddle point), so it is not suitable for our purpose.

Vinh Thinh Ho (2562630)
Noshaba Cheema (2562653)

Exercise 7.3

a) See **7.3.py**
b) See **7.3.py**
c)