AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

श्रद्धावान् लभते ज्ञानम्

19CSE437
DEEP LEARNING FOR COMPUTER VISION
L-T-P-C: 2-0-3-3

Feed Forward Neural Networks

- **Optimization – Hyper Parameter Tunings**
  - **Gradient Descent Variants**

Citation Note: content, of this presentation were inspired by the awesome lectures and the material offered by Prof. ***Mitesh M. Khapra*** *on NPTEL's Deep Learning course*

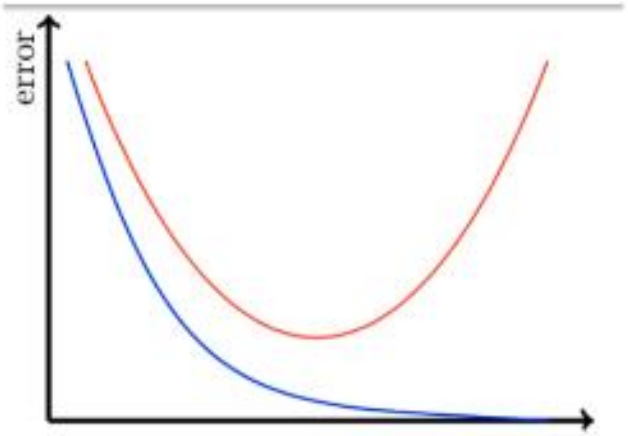
Feed Forward NN - Hyper Parameter Tunings

Network Architectures

- Number of layers
- Number of neurons

Algorithms

- Vanilla/Momentum /Nesterov GD
- AdaGrad
- RMSProp
- Adam



Activation Functions

- tanh (RNNs)
- relu (CNNs, DNNs)
- leaky relu (CNNs)

Strategies

- Batch
- Mini-Batch (32, 64, 128)
- Stochastic
- Learning rate schedule

Regularization

- L2
- Early stopping
- Dataset augmentation
- Drop-out
- Batch Normalizat

Initialization Methods

- *Xavier*
- *He*

Recap-Gradient Descent

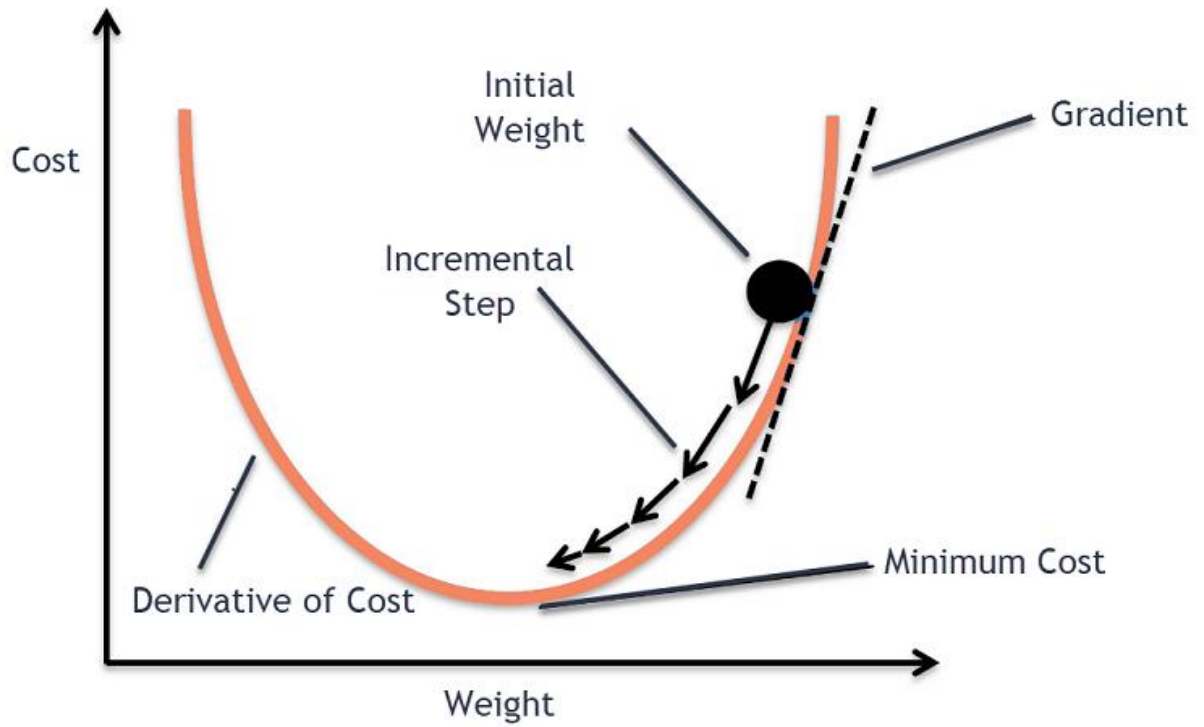
Gradient descent is a way to minimize an objective function parameterized by a model's parameters by updating the parameters in the opposite direction of the gradient of the objective function

The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley

Steps

1.Compute the slope (gradient) that is the first-order derivative of the function at the current point

2.Move-in the opposite direction of the slope increase from the current point by the computed amount



$$w_{t+1} = w_t - \eta \Delta w_t$$
$$b_{t+1} = b_t - \eta \Delta b_t$$
$$where\ \Delta w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}\ at\ w=w_t,b=b_t, \Delta b_t = \frac{\partial \mathscr{L}(w,b)}{\partial b}\ atw=w_t,b=b_t$$

Gradient Descent Algorithm- Tuning

- **Gradient Descent Variants**
  - Batch gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent

How do you compute the gradients ?

or

What data should you use for computing the gradients?

How do you use the gradients?

or

Can you come up with a better update rule ?

- **Gradient descent optimization algorithms**
  - **Momentum**
  - **Nesterov accelerated gradient**
  - **Adagrad**
  - Adadelta
  - **RMSprop**
  - **Adam**
  - AdaMax
  - Nadam
  - AMSGrad

Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update.

Variants of Gradient Descent

- *What is one epoch? One step?*

$$\mathscr{L} = \sum_{i=1}^{i=N} (f(x_i) - y_i)^2$$

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = mini-batch size

$$\Delta w = \frac{\partial \mathscr{L}}{\partial w} = \sum_{i=1}^{i=N} \frac{\partial}{\partial w} (f(x_i) - y_i)^2$$

Algorithm	# of steps in one epoch
Batch gradient descent	1
Stochastic gradient descent	N
Mini-Batch gradient descent	N/B

$$w_{t+1} = w_t - \eta \Delta w_t$$
$$b_{t+1} = b_t - \eta \Delta b_t$$

$$where\ \Delta w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}\ at\ w=w_t, b=b_t, \Delta b_t = \frac{\partial \mathscr{L}(w,b)}{\partial b}\ atw=w_t, b=b_t$$

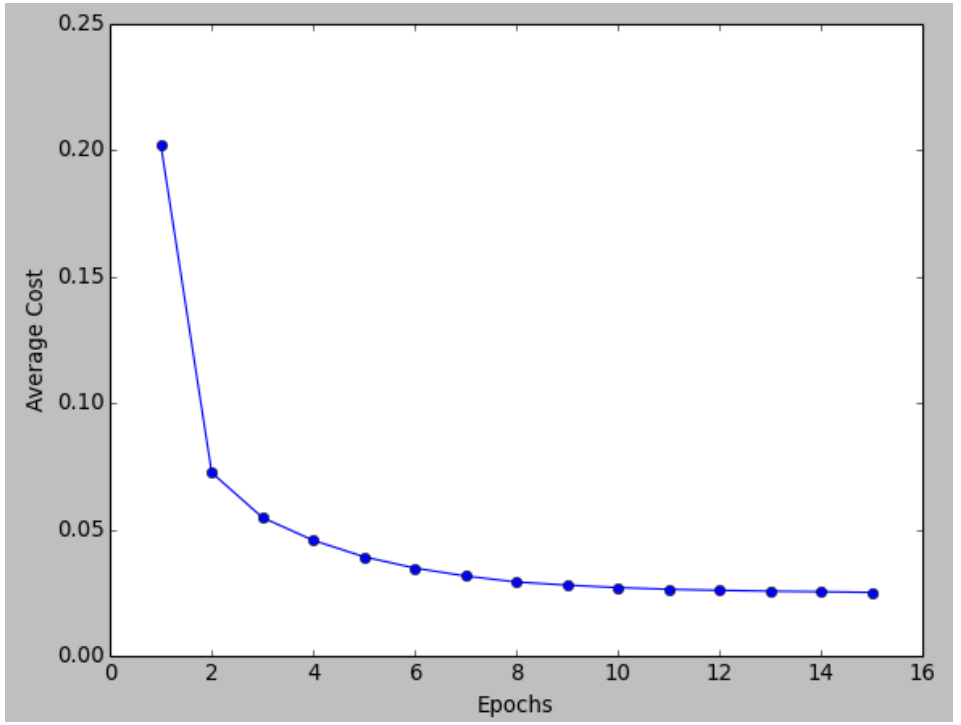
Batch Gradient Descent

1 step in 1 epoch

In Batch Gradient Descent,

- All the training data is taken into consideration to take a single step.
- We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters.
- So that's just one step of gradient descent in one epoch.

Batch gradient descent



Batch Gradient Descent is great for convex or relatively smooth error manifolds. In this case, we move somewhat directly towards an optimum solution.

But what if our dataset is very huge. Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples.??

Stochastic Gradient Descent

N steps in 1 epoch

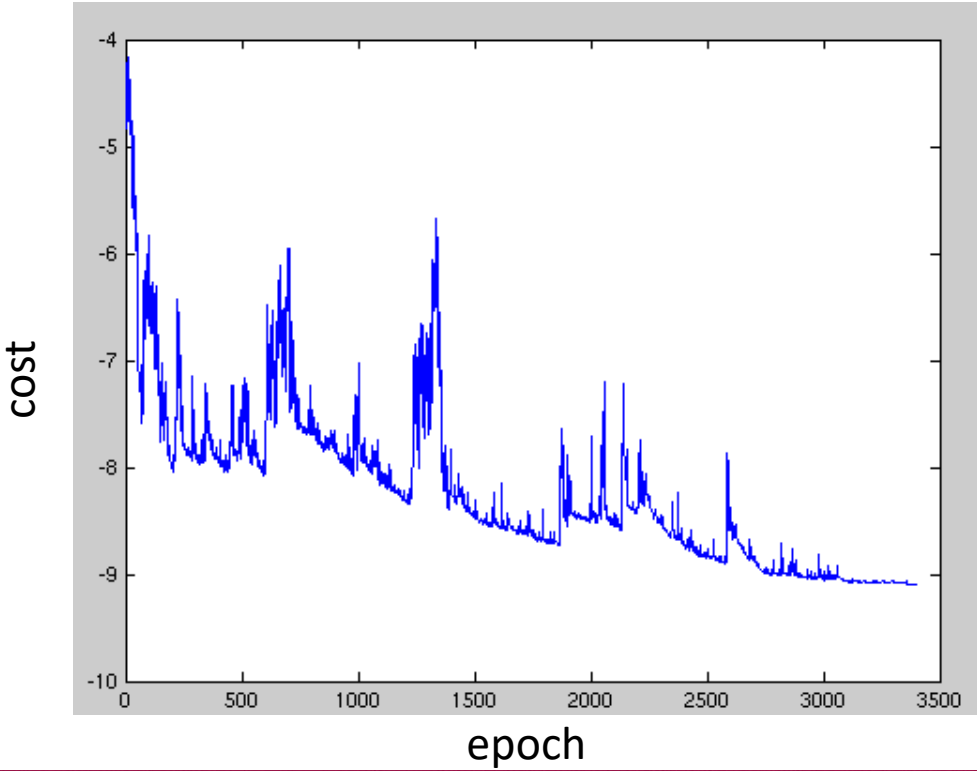
In Batch Gradient Descent we were considering all the examples for every step of Gradient Descent. But what if our dataset is very huge. Deep learning models crave for data. The more the data the more chances of a model to be good

Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples

In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step.

Observations

- Since we are considering just one example at a time the cost will fluctuate over the training examples and it will **not** necessarily decrease.
- But in the long run, you will see the cost decreasing with fluctuations.
- Because the cost is so fluctuating, it will never reach the minima but it will keep dancing around it.
- SGD can be used for larger datasets. It converges faster when the dataset is large as it causes updates to the parameters more frequently



Mini Batch Gradient Descent

N/B steps in 1 epoch

Batch Gradient Descent can be used for smoother curves. SGD can be used when the dataset is large.

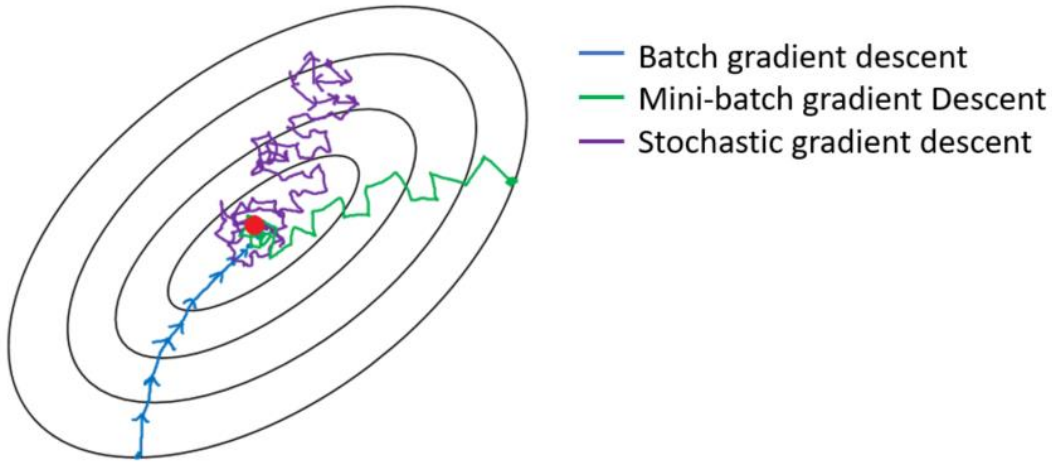
Batch Gradient Descent converges directly to minima. SGD converges faster for larger datasets.

But, since in SGD we use only one example at a time, we cannot implement the vectorized implementation on it. This can slow down the computations. To tackle this problem, a mixture of Batch Gradient Descent and SGD is used- **Mini Batch GD.**

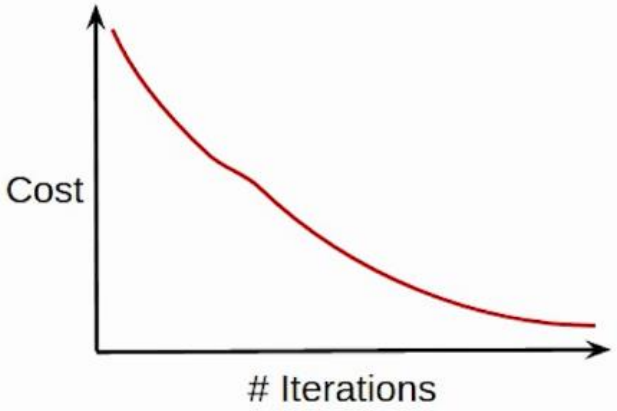
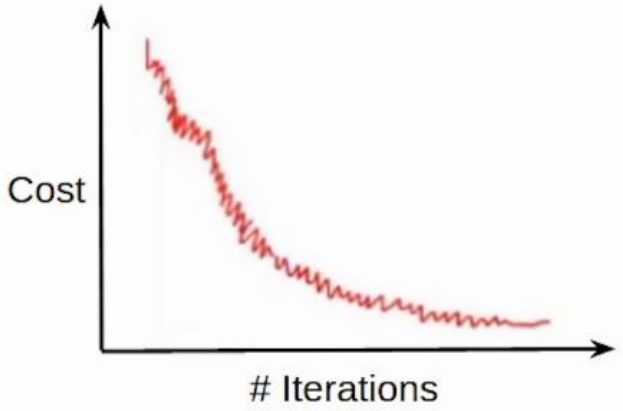
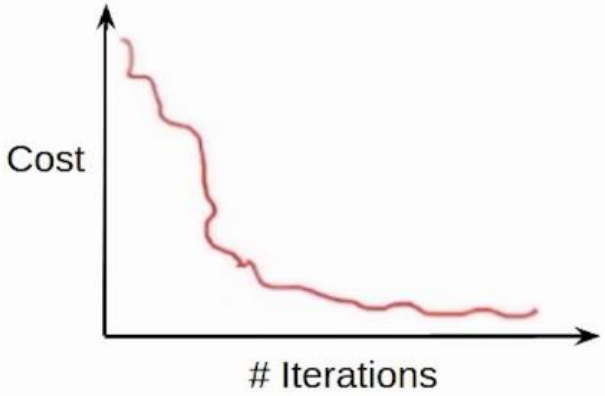
Mini Batch gradient Descent

We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of both Batch and stochastic GD. So, after creating the mini-batches of fixed size, iterate over mini-batches of size B:

N/B step in one epoch (N : total no of data points, B batch size)



Normal Batch size- 32,64,128

Batch Gradient Descent	Stochastic Gradient Descent (SGD)	Mini-Batch Gradient Descent
• Entire dataset for updation	• Single observation for updation	• Subset of data for updation
• Cost function reduces smoothly 	• Lot of variations in cost function 	• Smoother cost function as compared to SGD 
• Computation cost is very high	• Computation time is more	• Computation time is lesser than SGD • Computation cost is lesser than Batch Gradient Descent

Batch gradient descent

```
def do_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

Stochastic gradient descent

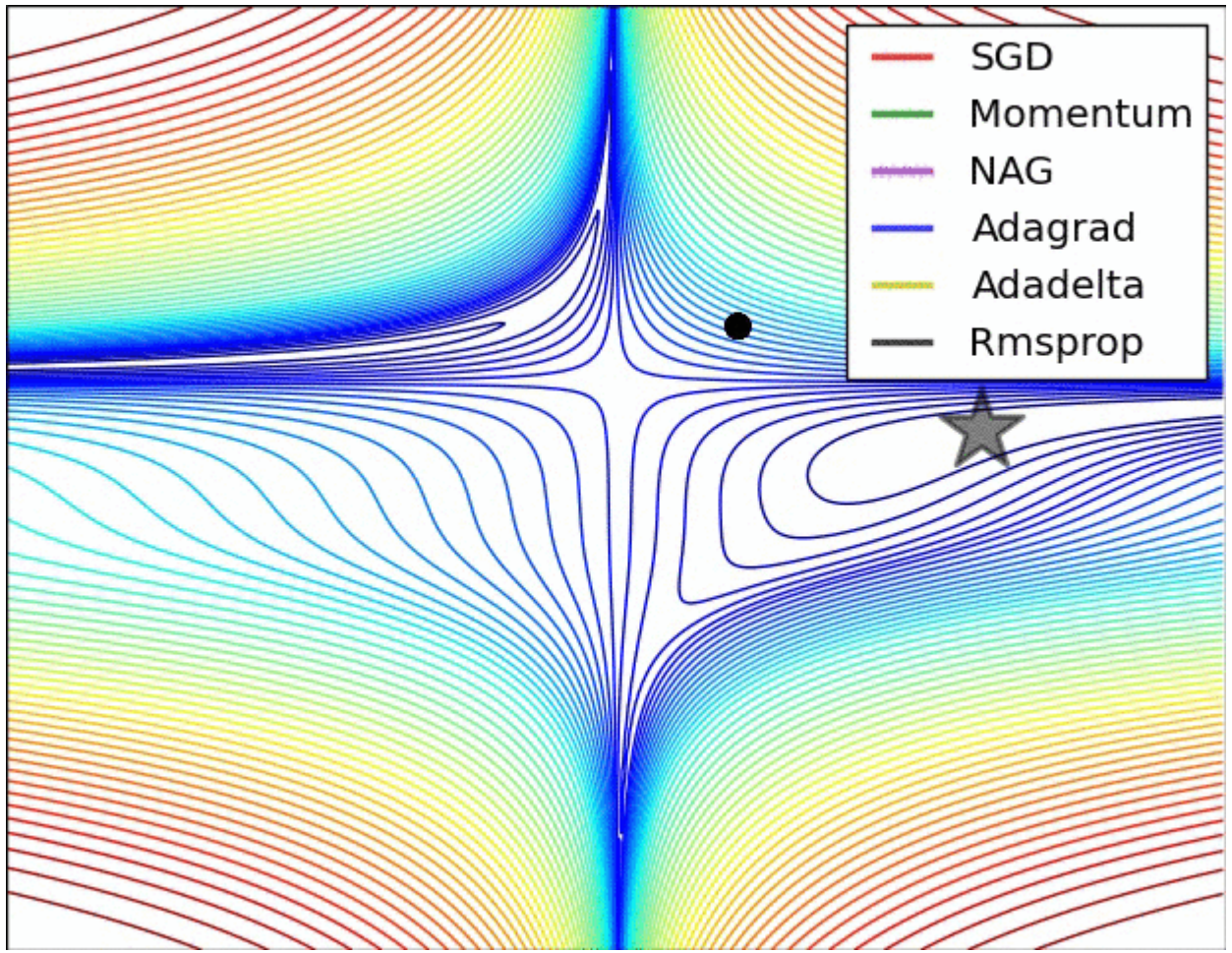
```
def do_stochastic_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

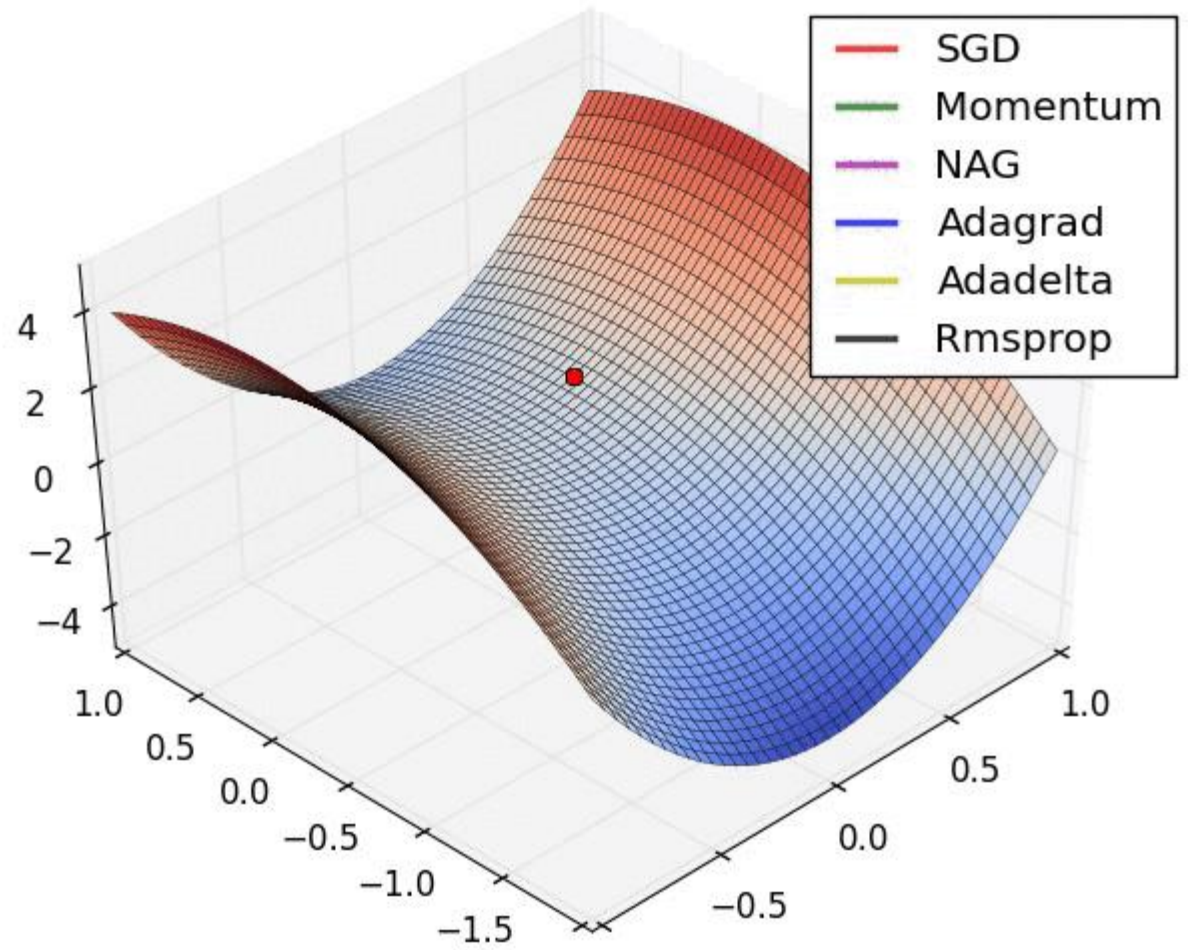
Mini-batch gradient descent

```
def do_mini_batch_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    mini_batch_size = 0
    num_points_seen = 0
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points_seen += 1

            if num_points_seen % mini_batch_size == 0:
                w = w - eta * dw
                b = b - eta * db
```

Comparison- Visualisation





Namah Shivaya