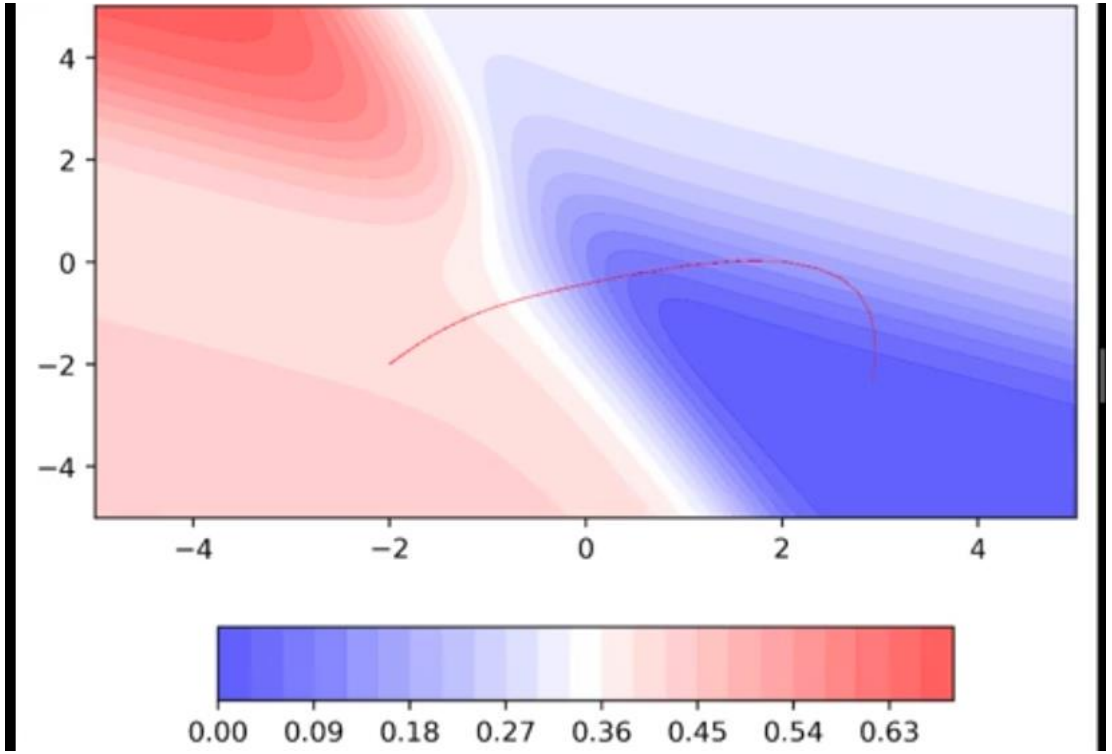
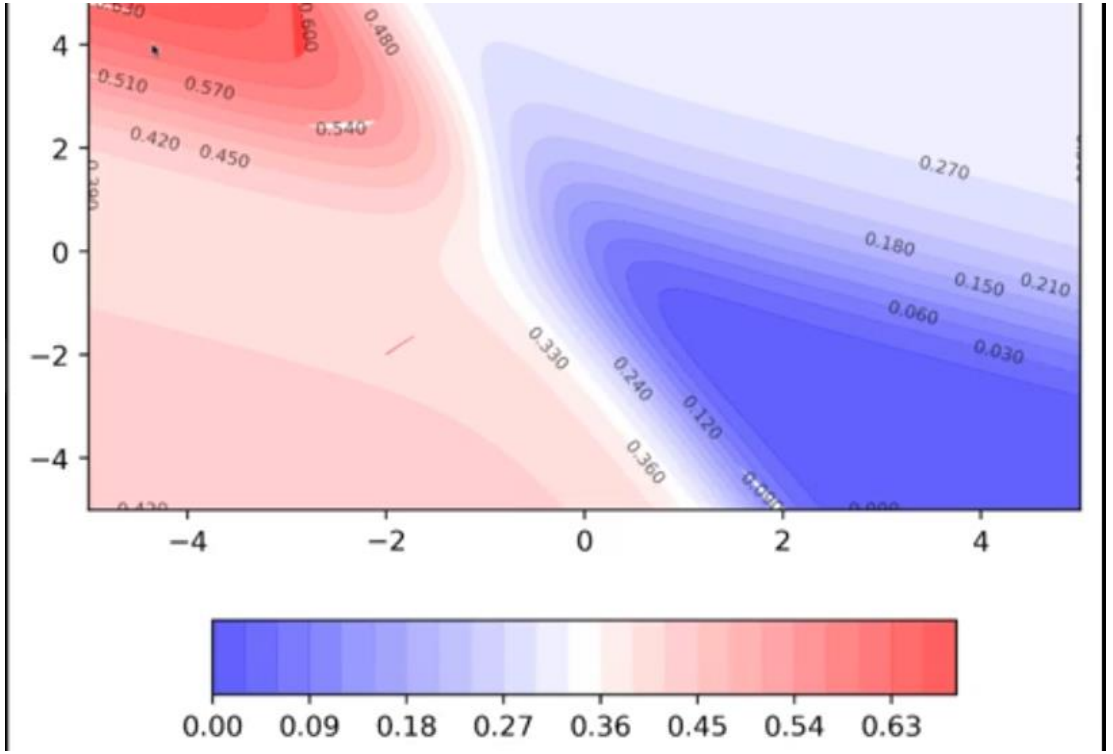
Deep Learning:
Feed Forward Neural Network

राष्ट्रीय प्रौद्योगिकी संस्थान सिक्किम
NATIONAL INSTITUTE OF TECHNOLOGY SIKKIM

Course Instructor:
Dr. Bam Bahadur Sinha
Assistant Professor
Computer Science & Engineering
National Institute of Technology
Sikkim



Momentum Gradient Descent



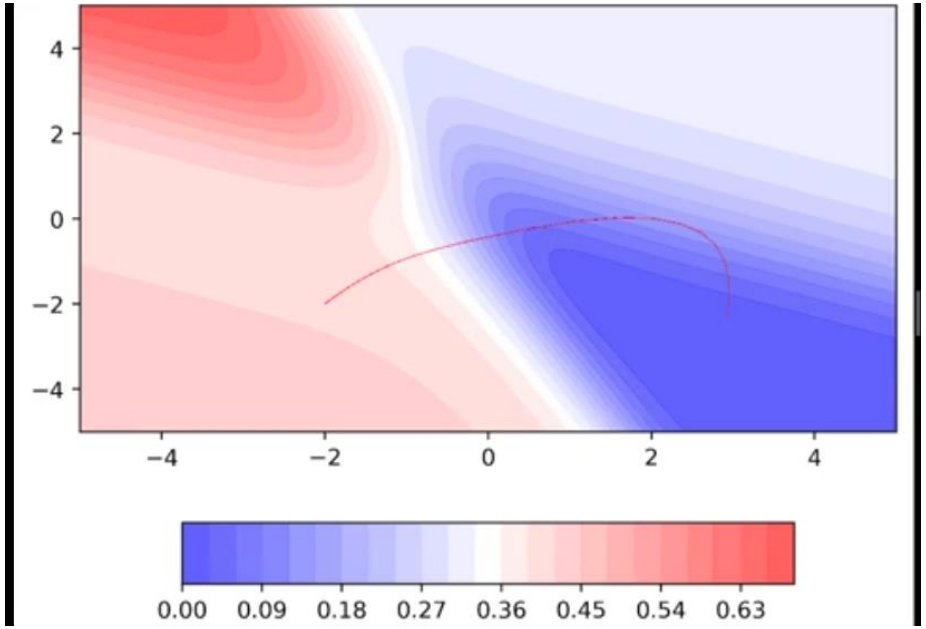
Gradient Descent

Convergence Graph - GD Vs MGD

Gradient Descent Update Rule

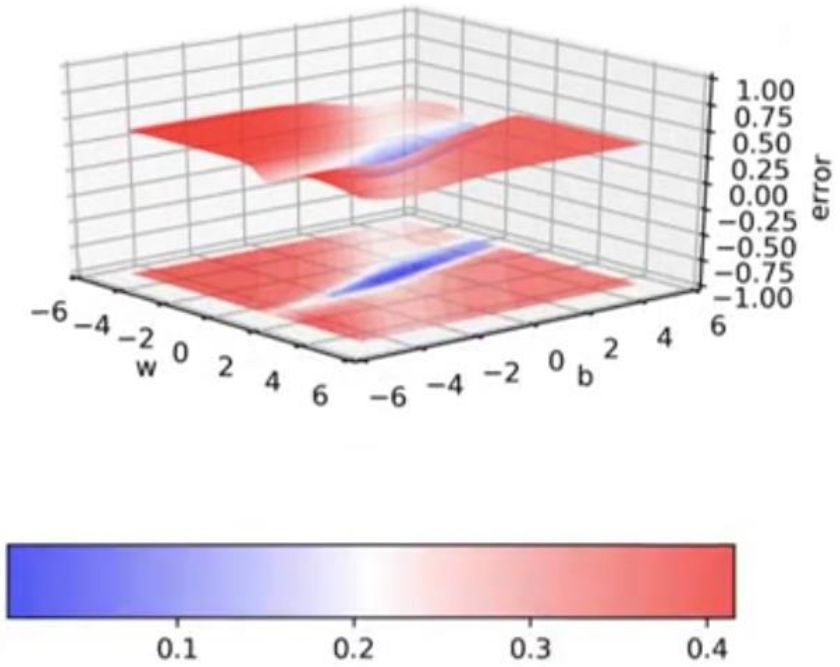
Observations

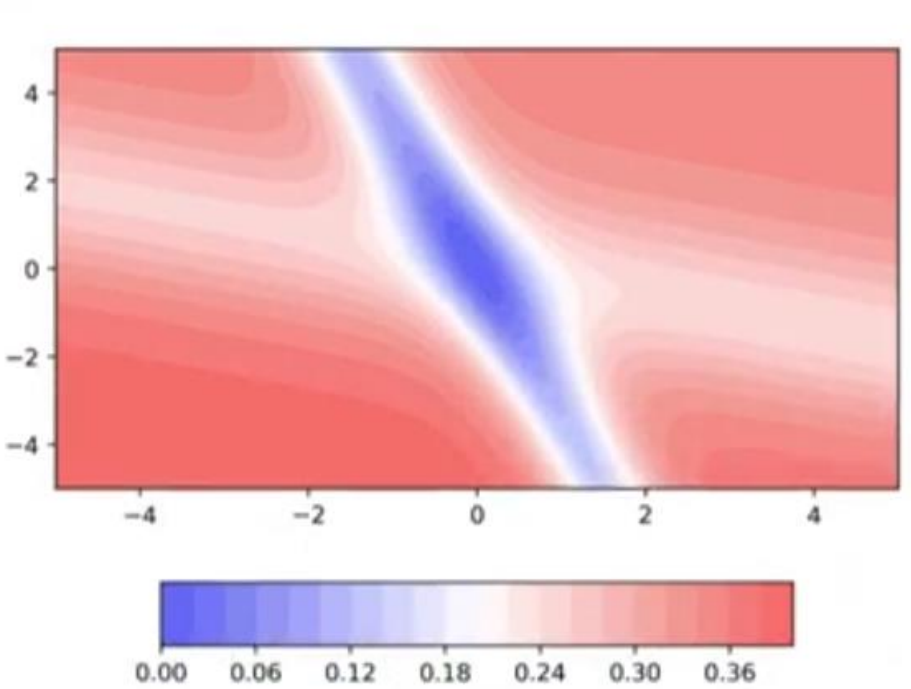
Even in the regions having gentle slopes, momentum based gradient descent is able to take large steps because the momentum carries it along



Questions

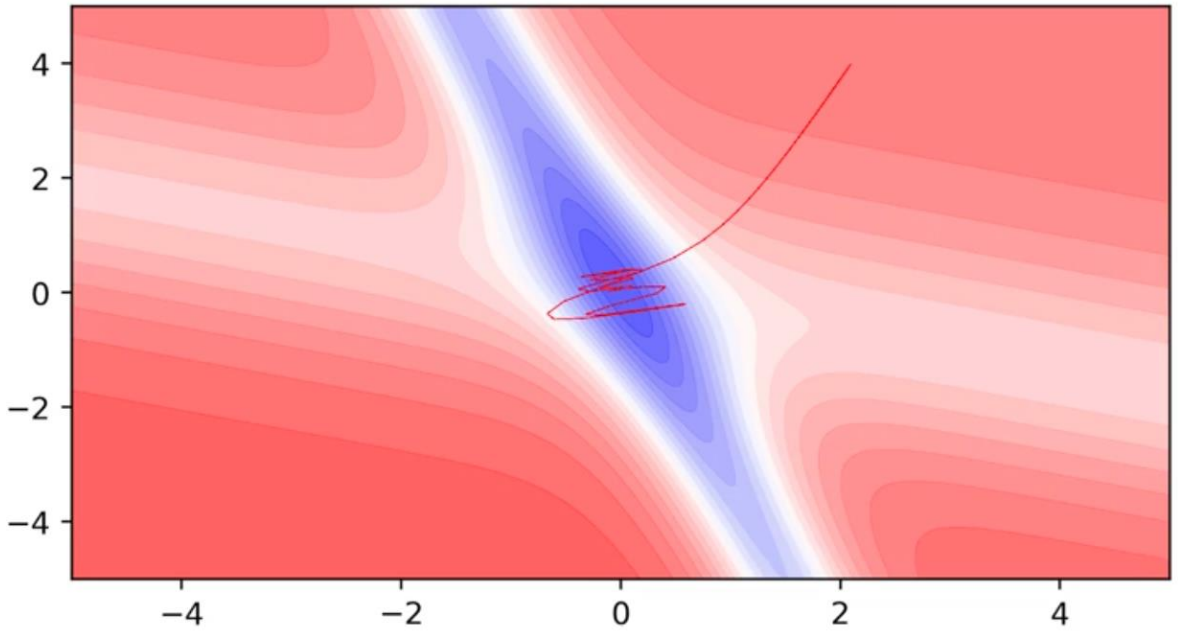
- Is moving fast always good?
- Would there be a situation where momentum would cause us to run pass our goal?





Would there be cases where momentum could be detrimental?

Would there be cases where momentum could be detrimental?



- Momentum based gradient descent oscillates in and out of the minima valley (u-turns)
- Despite these u-turns it still converges faster than vanilla gradient descent.

Nesterov Accelerated Gradient Descent

Momentum based Gradient Descent Update Rule

$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - v_t$$

$$w_{t+1} = w_t - \gamma * v_{t-1} - \eta \nabla \dot{w}_t$$

NAG Update Rule

$$w_{temp} = w_t - \gamma * v_{t-1}$$

$$w_{t+1} = w_{temp} - \eta \nabla w_{temp}$$

$$v_t = \gamma * v_{t-1} + \eta \nabla w_{temp}$$

- Looking ahead helps NAG in correcting its course quicker than momentum based gradient descent.
- Hence the oscillations are smaller and the chances of escaping the minima valley also smaller.

Implementation Code Snippet

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x):
   #sigmoid with parameters w, b
   return 1.0 / (1.0 + np.exp(-(w*x + b))

def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
     return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

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
```

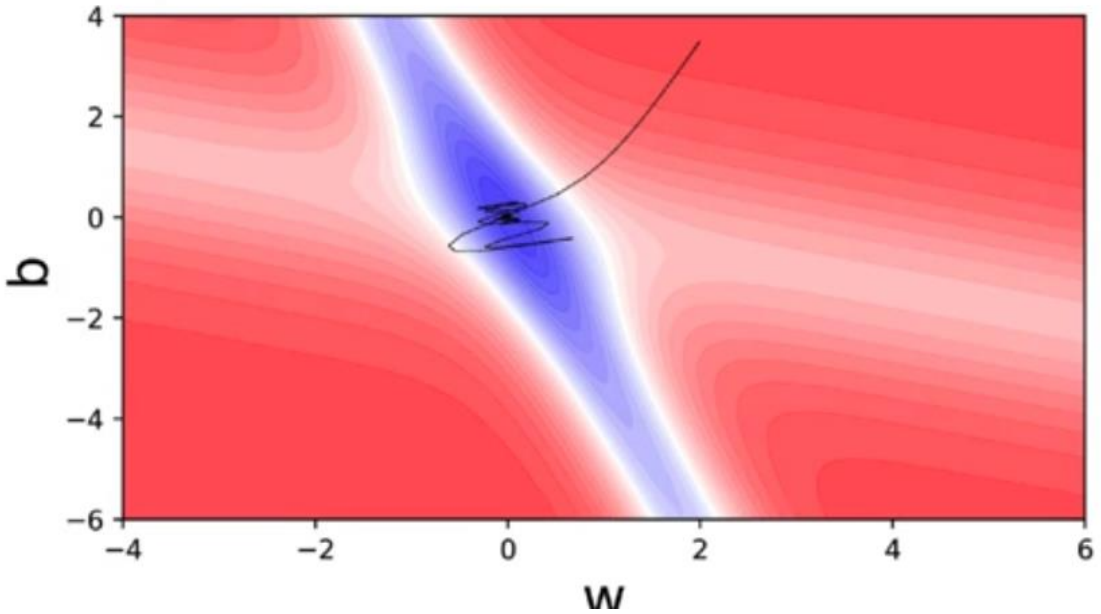
NAG Update Rule

$$w_{temp} = w_t - \gamma * v_{t-1}$$

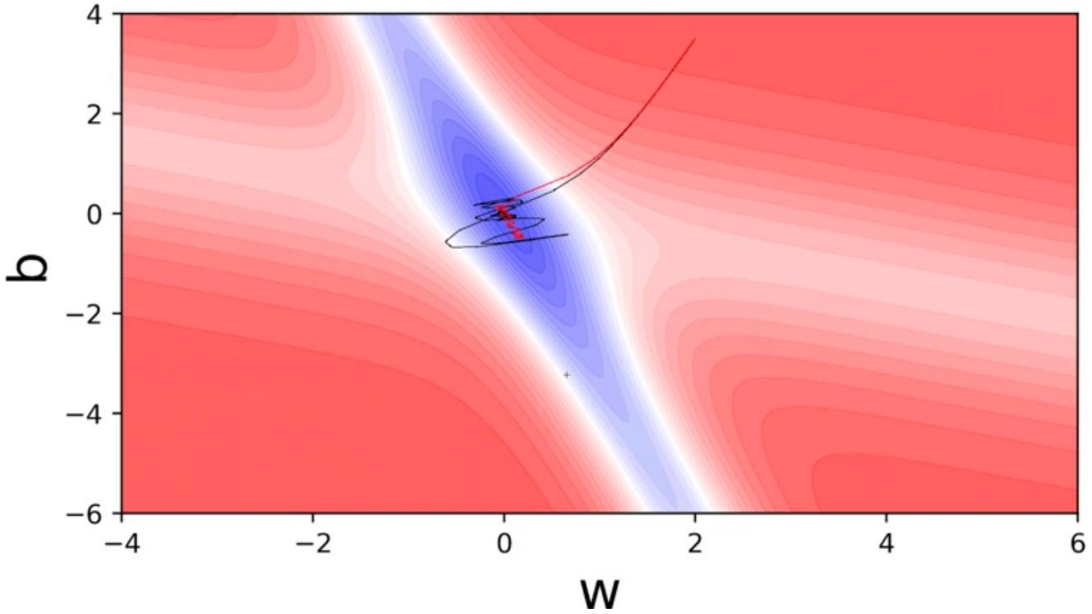
$$w_{t+1} = w_{temp} - \eta \nabla w_{temp}$$

$$v_t = \gamma * v_{t-1} + \eta \nabla w_{temp}$$

```
def do_nag_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    v_w, v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs):
        dw, db = 0, 0

        #Compute the lookahead value
        w = w - gamma*v_w
        b = b - gamma*v_b

        for x, y in zip(X, Y) :
            #Compute derivatives using the lookahead v
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        #Now move further in the direction of that gra
        w = w - eta * dw
        b = b - eta * db

        #Now update the history
        v_w = gamma * v_w + eta * dw
        v_b = gamma * v_b + eta * db
```



Momentum Gradient Descent



NAG Descent

In NAGD, the oscillations are smaller and the chances of escaping the minima valley also smaller.

Better Learning Algorithms

- How do we compute the gradients?
- How do you use the gradients?
- Can we come up with a better update rule?
- What data should we use to compute the gradients?

Gradient Descent Update Rule

$$w = w - \eta \frac{\partial \mathcal{L}(w)}{\partial w}$$

Batch GD, Stochastic GD, Mini-Batch GD

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x):
   #sigmoid with parameters w, b
   return 1.0 / (1.0 + np.exp(-(w*x + b))

def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
     return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

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
```

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

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

Batch Gradient Descent

- In batch gradient descent, the entire training dataset is used to compute the gradient of the cost function with respect to the model parameters in each iteration.
- This means that the algorithm processes all the training examples simultaneously to make a single update to the model parameters.
- Batch gradient descent provides a precise estimate of the gradient but can be computationally expensive for large datasets, as it requires storing and processing the entire dataset in memory for each iteration.
- Despite being computationally expensive, batch gradient descent guarantees convergence to the global minimum.

Mini-Batch Gradient Descent

- In mini-batch gradient descent, the training dataset is divided into small batches of fixed size (mini-batches), and the gradient of the cost function is computed using one of these mini-batches in each iteration.
- By processing a subset of the training data in each iteration, mini-batch gradient descent can be more computationally efficient than batch gradient descent, especially for large datasets.
- The size of the mini-batch is a hyperparameter that needs to be tuned. Common mini-batch sizes are powers of 2, such as 32, 64, 128, etc.

Stochastic Gradient Descent

- It is a variant of gradient descent that processes one training example at a time rather than the entire dataset in each iteration.
- In SGD, rather than computing the gradient of the cost function with respect to the model parameters using the entire dataset (as in batch gradient descent), the gradient is computed using only one training example at a time. This results in faster iterations, especially for large datasets.
- Because SGD computes the gradient using only one training example at a time, the updates to the model parameters can be noisy, leading to fluctuations in the optimization trajectory.
- To strike a balance between the computational efficiency of SGD and the stability of batch gradient descent, mini-batch SGD is often used.

Batch GD, Stochastic GD, Mini-Batch GD

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x):
   #sigmoid with parameters w, b
   return 1.0 / (1.0 + np.exp(-(w*x + b))

def error(w, b):
     err = 0.0
     for x, y in zip(X, Y):
        fx = f(w, b, x)
        err += 0.5* (fx - y) ** 2
     return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

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
```

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

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

```python
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
```

Batch Gradient Descent
Vs
Stochastic Gradient Descent

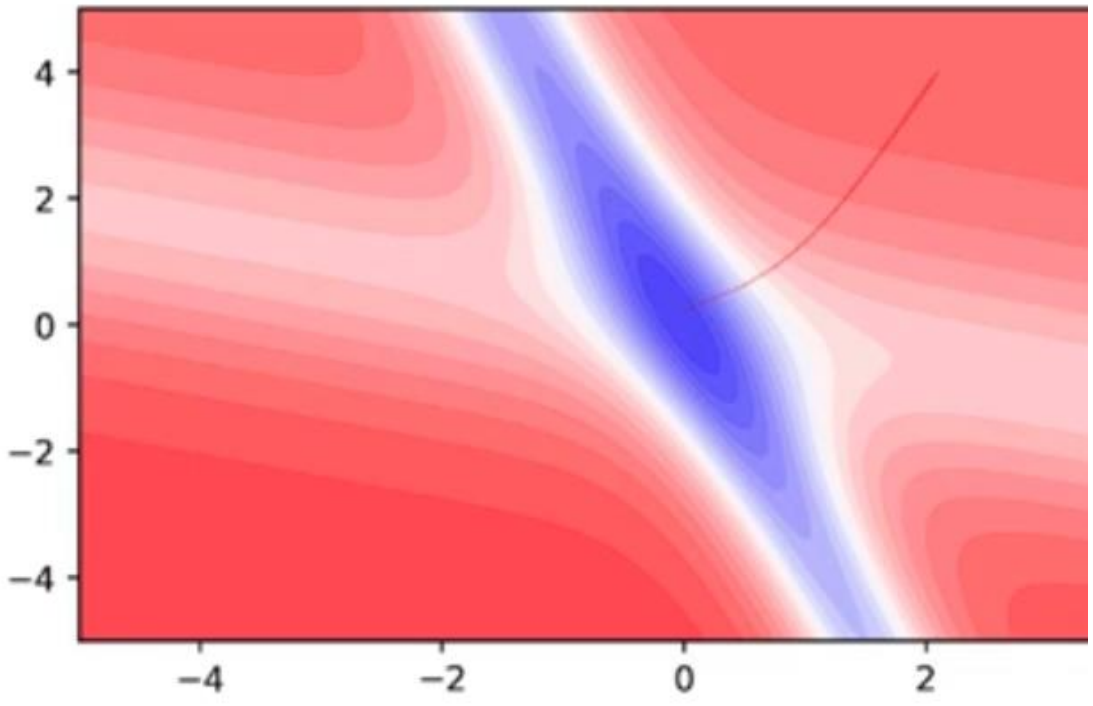
```python
def do_stochastic_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            w = w - eta * dw
            b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

Advantage

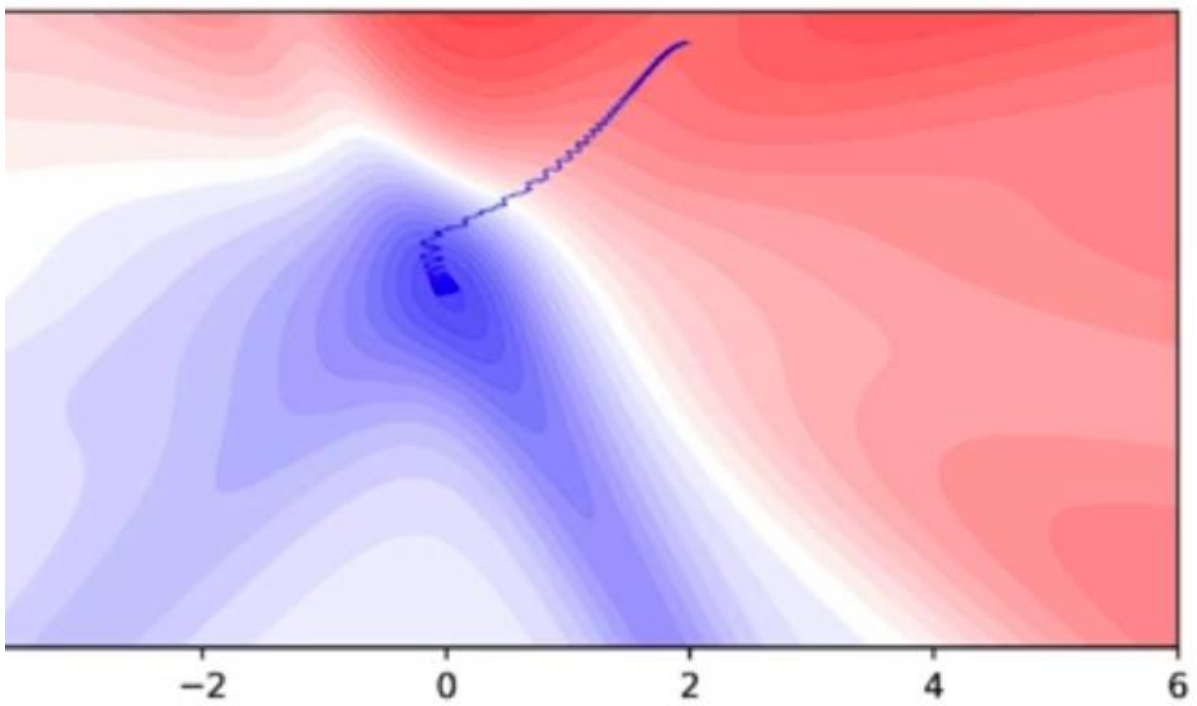
- Quicker updates
- Many updates in one pass of the data

Disadvantage

- Approximate (stochastic) gradient
- Almost like tossing a coin only once and computing the value of P(heads)



Batch Gradient Descent



Stochastic Gradient Descent

Convergence of Batch GD & Stochastic GD

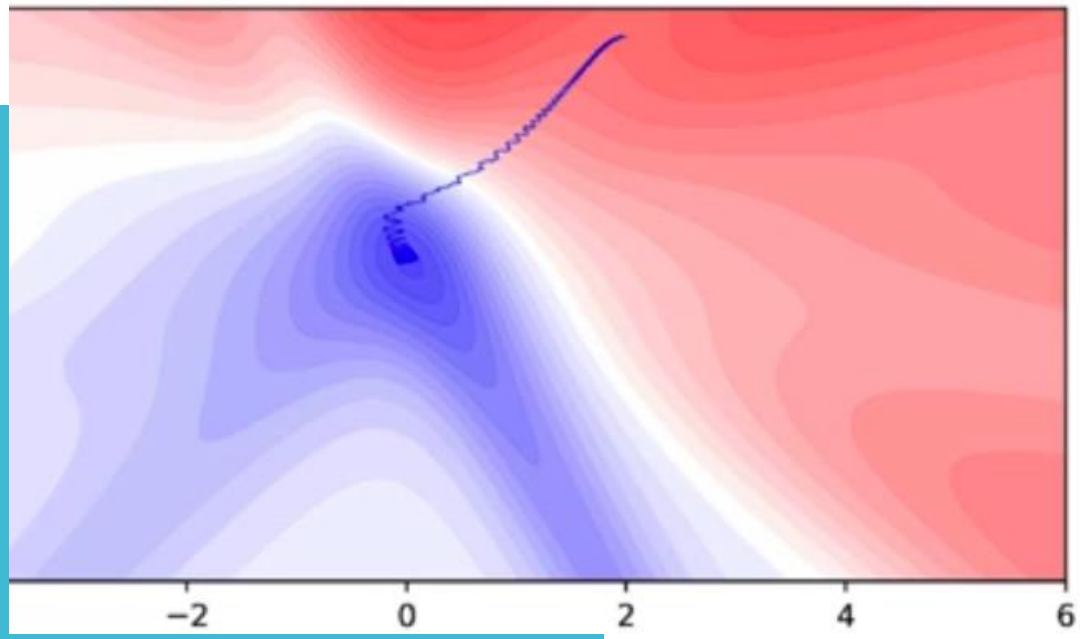
```
def do_gradient_descent():
    w, b, eta = -2, -2, 1.0
    max_epochs = 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

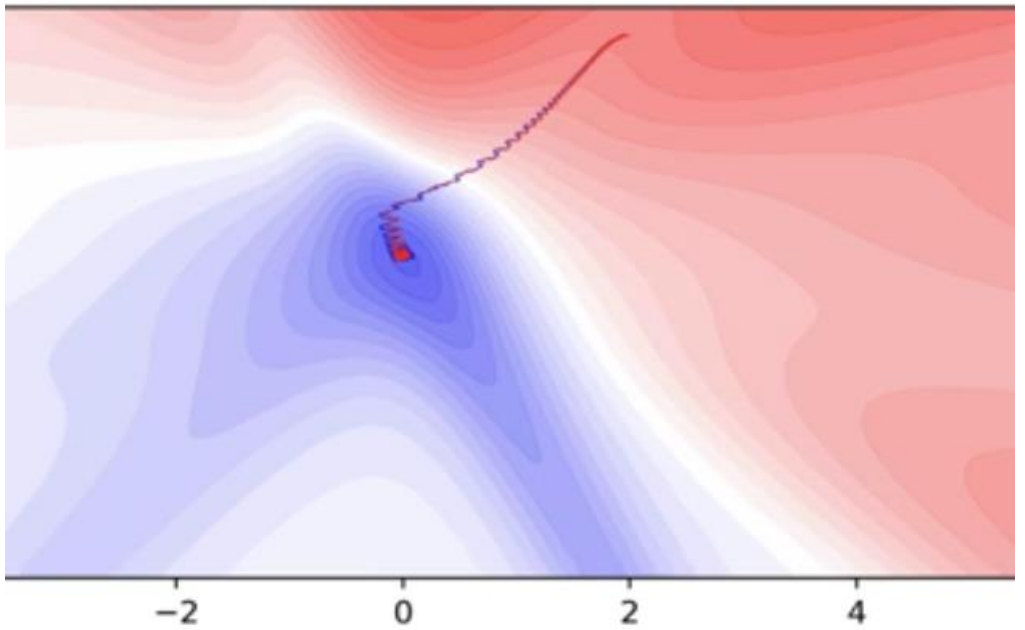
Mini Batch GD

```
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
```

```
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
```

- Approximation of mini-batch GD is slightly better than stochastic GD
- Higher the value of batch size, the more accurate are the estimates.





- Even with a batch size of k=2, the oscillations have reduced slightly.
- We now have slightly better estimates of the gradient.
- The higher the value of ‘k’ the more accurate are the estimates.

Convergence of Stochastic GD vs Mini-Batch GD

What is an EPOCH and What is a STEP?

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

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

```python
def do_momentum_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    v_w, v_b = 0, 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)
        v_w = gamma*v_w + eta * dw
        v_b = gamma*v_b + eta * dw

        w = w - v_w
        b = b - v_w
```

```python
def do_stochastic_momentum_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    v_w, v_b = 0, 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)
            v_w = gamma*v_w + eta * dw
            v_b = gamma*v_b + eta * db

            w = w - v_w
            b = b - v_b
```

Stochastic Version of NAG and Momentum Based GD

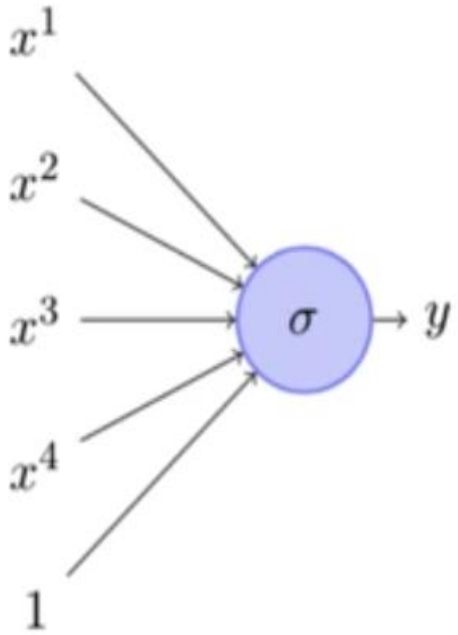
```python
def do_nag_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    v_w, v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs):
        dw, db = 0, 0

        #Compute the lookahead value
        w = w - gamma*v_w
        b = b - gamma*v_b

        for x, y in zip(X, Y) :
            #Compute derivatives using the lookahead v
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        #Now move further in the direction of that gra
        w = w - eta * dw
        b = b - eta * db

        #Now update the history
        v_w = gamma * v_w + eta * dw
        v_b = gamma * v_b + eta * db
```

```python
    for i in range(max_epochs):
        dw, db = 0, 0

        #Compute the lookahead value
        w = w - gamma*v_w
        b = b - gamma*v_b

        for x, y in zip(X, Y) :
            #Compute derivatives using the lookahead v
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        #Now move further in the direction of that gra
            w = w - eta * dw
            b = b - eta * db

        #Now update the history
            v_w = gamma * v_w + eta * dw
            v_b = gamma * v_b + eta * db
```

NOTE

- Typically, we use mini-batch gradient descent in most cases with batch size = 32, 64, 128.
- The batch size is treated as hyperparameters.
- What data should we use for computing the gradients? - Batch GD, Stochastic GD, Mini-Batch GD.
- Can we design a better update rule? – Momentum based GD, Nesterov Accelerated GD

Why do we need a different learning rate for every feature?



$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

$$\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$$
$$\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$$

Can we have a different learning rate for each parameter which takes care of the frequency of features ?