Optimization for Deep Learning: Stochastic Gradient Descent

Industrial AI Lab.

Prof. Seungchul Lee

POSTECH

Today

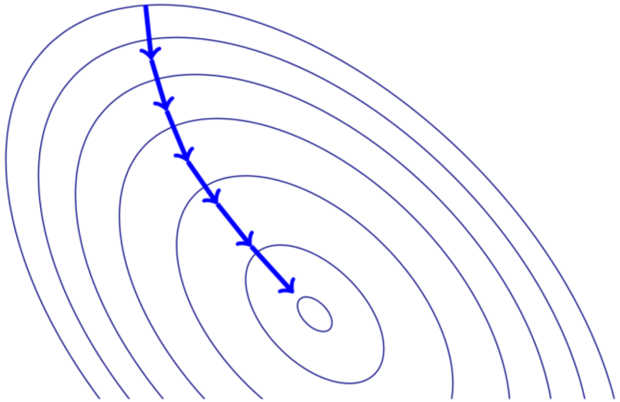
- We will cover gradient descent algorithm and its variants:
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent

- We will explore the concept of these three gradient descent algorithms with a logistic regression model in TensorFlow

- Limitation of the Gradient Descent
  - Learning Rate
  - Gradient Descent with Momentum

Batch Gradient Descent

Repeat: $\omega \leftarrow \omega - \alpha \nabla f(\omega)$ for some step size (or learning rate) $\alpha > 0$



Batch Gradient Descent

- Loss function ℓ has been the average loss over the training examples:

$$\mathcal{E}(\omega) = \frac{1}{m}\sum_{i=1}^{m} \ell(\hat{y}_i, y_i) = \frac{1}{m}\sum_{i=1}^{m} \ell(h_\omega(x_i), y_i)$$

- By linearity,

$$\nabla_\omega \mathcal{E} = \nabla_\omega \frac{1}{m}\sum_{i=1}^{m} \ell(h_\omega(x_i), y_i) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial \omega}\ell(h_\omega(x_i), y_i)$$

$$\omega \leftarrow \omega - \alpha \nabla_\omega \mathcal{E}$$

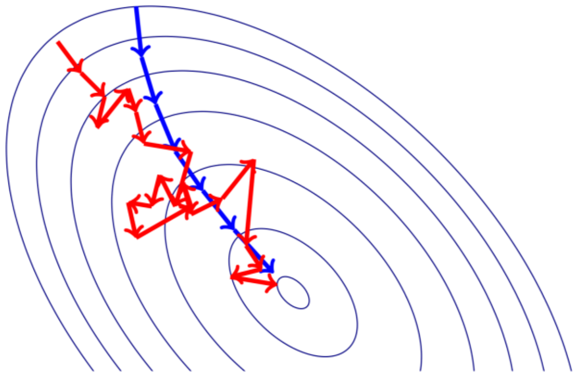
- Computing the gradient requires summing over all of the training examples.
- This is known as batch training.
- Batch training is impractical if you have a large dataset (e.g. millions of training examples) !

Stochastic Gradient Descent (SGD)

- Stochastic gradient descent (SGD): update the parameters based on the gradient for a randomly selected single training example:

$$\ell(\hat{y}_i, y_i) = \ell(h_\omega(x_i), y_i) = \ell^{(i)}$$

$$\omega \leftarrow \omega - \alpha \frac{\partial \ell^{(i)}}{\partial \omega}$$

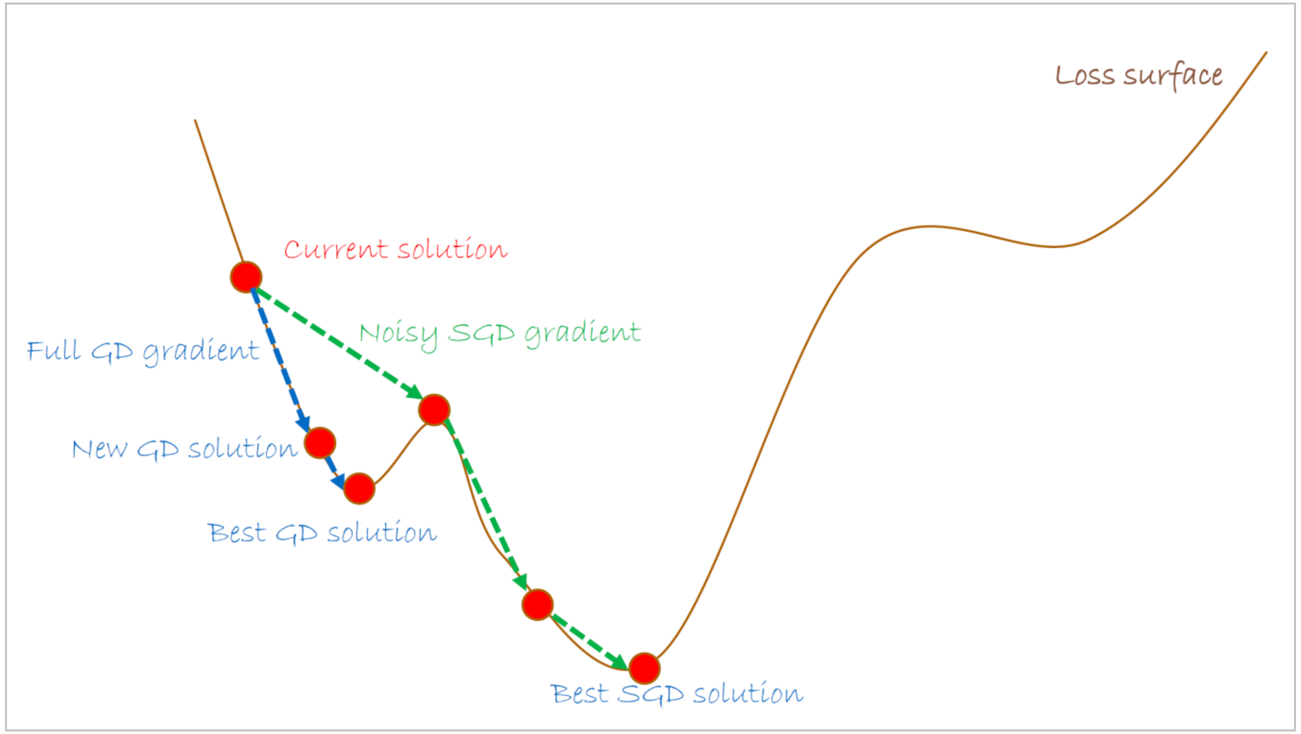


  - SGD takes steps in a noisy direction, but moves downhill on average.

- Mathematical justification: if you sample a training example at random, the stochastic gradient is an unbiased estimate of the batch gradient:

$$\mathbb{E}\left[\frac{\partial \ell^{(i)}}{\partial \omega}\right] = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial \ell^{(i)}}{\partial \omega} = \frac{\partial}{\partial \omega}\left[\frac{1}{m}\sum_{i=1}^{m}\ell^{(i)}\right] = \frac{\partial \mathcal{E}}{\partial \omega}$$

SGD is Sometimes Better



- No guarantee that this is what is going to always happen.
- But the noisy SGD gradients can help some times escaping local optima

Mini-batch Gradient Descent

- Potential problem of SGD: gradient estimates can be very noisy

- Compromise approach: compute the gradients on a medium-sized set of training examples $s\ (< m)$, called a mini-batch.

$$\mathcal{E}(\omega) = \frac{1}{s}\sum_{i=1}^{s} \ell(\hat{y}_i, y_i) = \frac{1}{s}\sum_{i=1}^{s} \ell(h_\omega(x_i), y_i) = \frac{1}{s}\sum_{i=1}^{s} \ell^{(i)}$$

$$\omega \leftarrow \omega - \alpha \nabla_\omega \mathcal{E}$$

- Stochastic gradients computed on larger mini-batches have smaller variance:

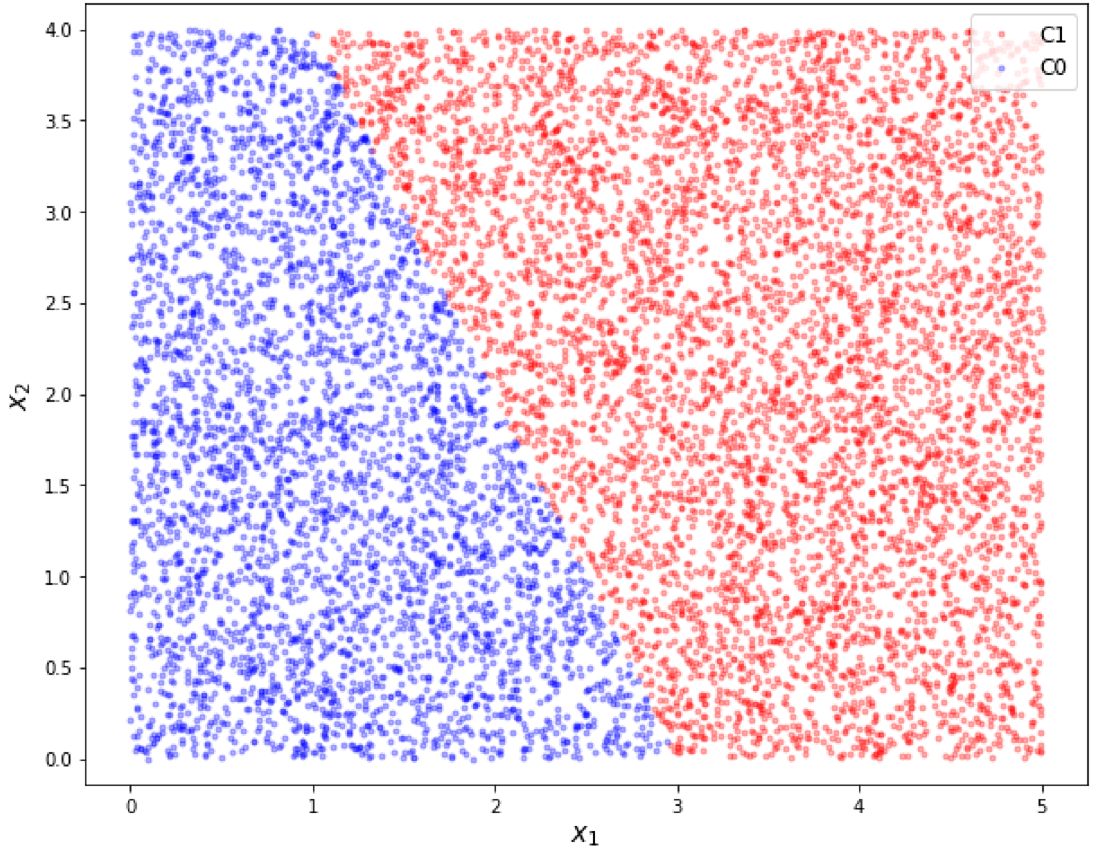
$$\mathrm{var}\left[\frac{1}{s}\sum_{i=1}^{s} \frac{\partial \ell^{(i)}}{\partial \omega}\right] = \frac{1}{s^2}\mathrm{var}\left[\sum_{i=1}^{s} \frac{\partial \ell^{(i)}}{\partial \omega}\right] = \frac{1}{s}\mathrm{var}\left[\frac{\partial \ell^{(i)}}{\partial \omega}\right]$$

- The mini-batch size s is a hyper-parameter that needs to be set.

Implementation with TensorFlow

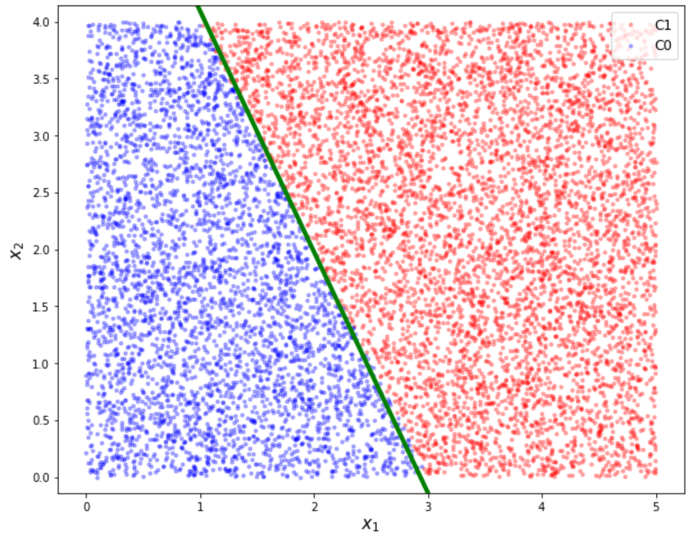
Batch Gradient Descent with TensorFlow

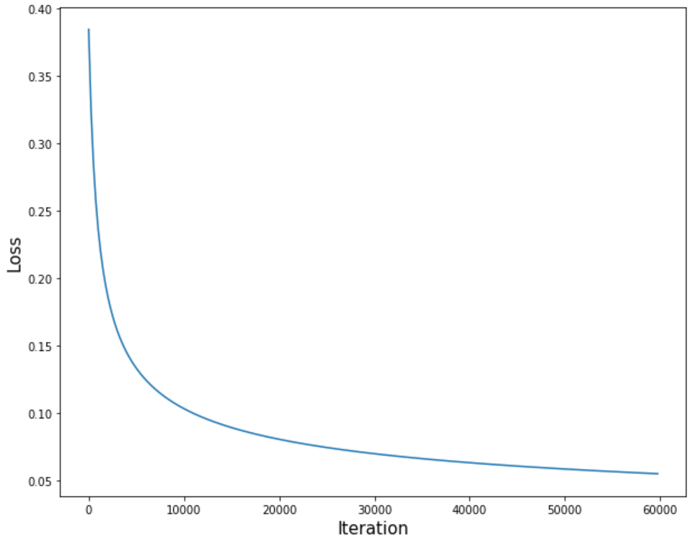
- We will explore the python codes of these three gradient descent algorithms with a logistic regression model.



Batch Gradient Descent with TensorFlow

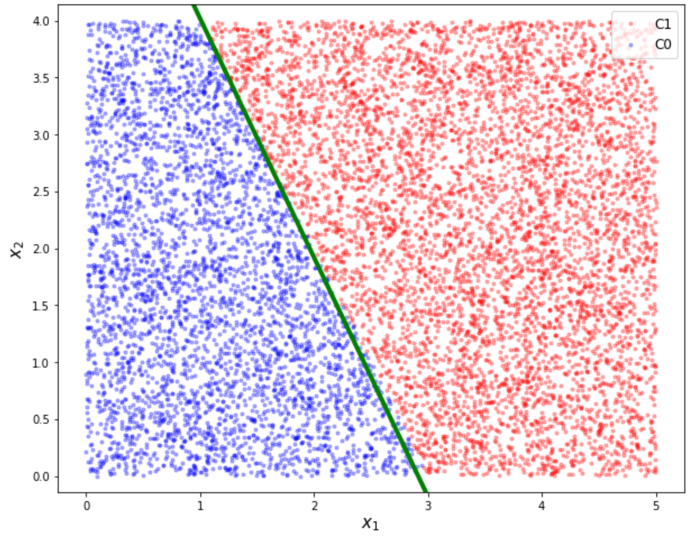
```
LR = 0.04
n_iter = 60000
n_prt = 250

x = tf.placeholder(tf.float32, [m, 3])
y = tf.placeholder(tf.float32, [m, 1])

w = tf.Variable([[0],[0],[0]], dtype = tf.float32)

y_pred = tf.matmul(x,w)
loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_pred, labels=y)
loss = tf.reduce_mean(loss)

optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)
init = tf.global_variables_initializer()

start_time = time.time()

loss_record = []
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(n_iter):
        sess.run(optm, feed_dict = {x: train_X, y: train_y})

        if (epoch + 1) % n_prt == 0:
            loss_record.append(sess.run(loss, feed_dict = {x: train_X, y: train_y}))

    w_hat = sess.run(w)
```

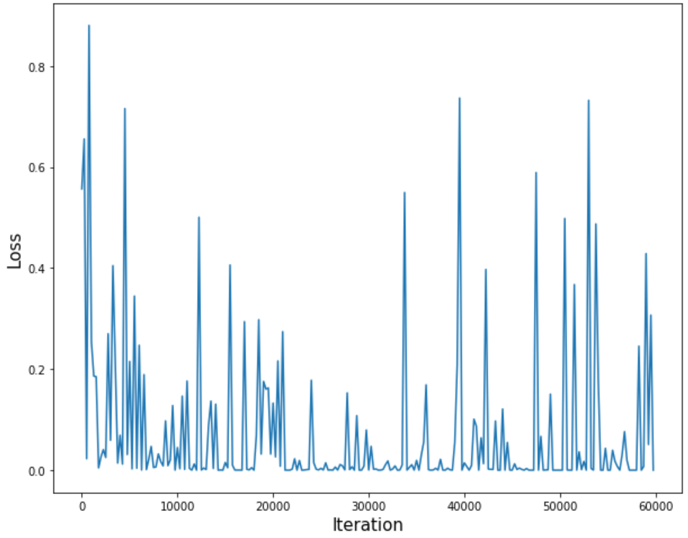




Stochastic Gradient Descent (SGD) with TensorFlow

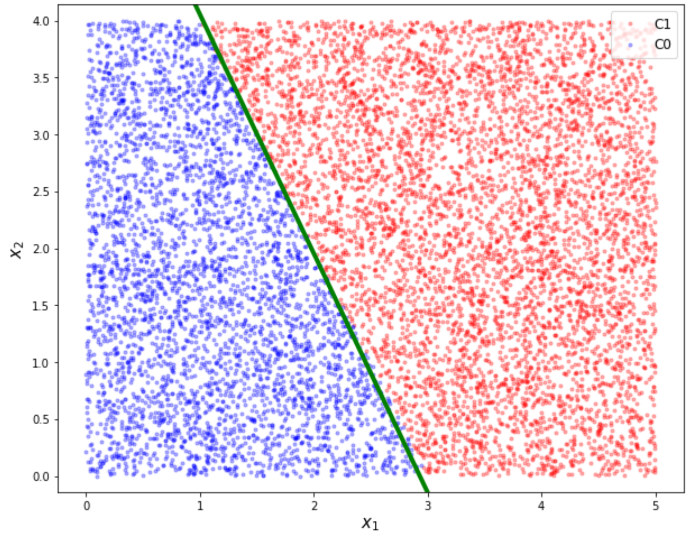
```
LR = 0.04
n_iter = 60000
n_prt = 250

x = tf.placeholder(tf.float32, [1, 3])
y = tf.placeholder(tf.float32, [1, 1])

w = tf.Variable(tf.random_normal([3,1]), dtype = tf.float32)

y_pred = tf.matmul(x,w)
loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_pred, labels=y)
loss = tf.reduce_mean(loss)

optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)
init = tf.global_variables_initializer()

start_time = time.time()

loss_record = []
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(n_iter):
        idx = np.random.choice(m, 1)
        batch_X = train_X[idx,:]
        batch_y = train_y[idx]
        sess.run(optm, feed_dict = {x: batch_X, y: batch_y})

        if (epoch + 1) % n_prt == 0:
            loss_record.append(sess.run(loss, feed_dict = {x: batch_X, y: batch_y}))

    w_hat = sess.run(w)
```

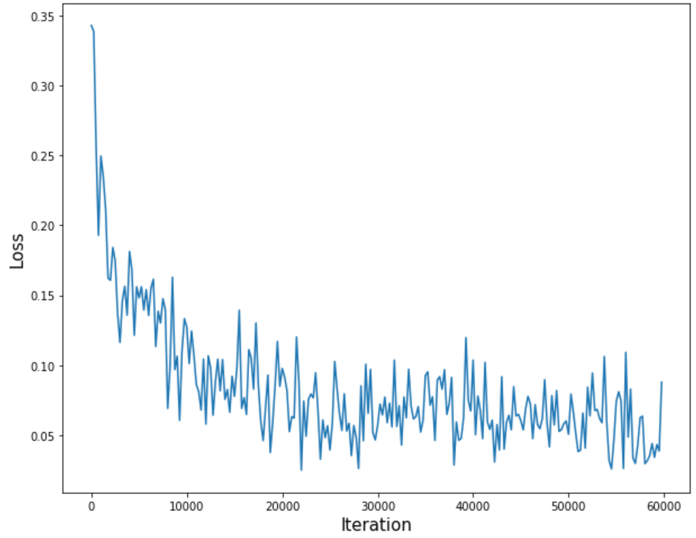




Mini-batch Gradient Descent with TensorFlow

```
LR = 0.04
n_iter = 60000
n_batch = 50
n_prt = 250

x = tf.placeholder(tf.float32, [n_batch, 3])
y = tf.placeholder(tf.float32, [n_batch, 1])

w = tf.Variable(tf.random_normal([3,1]), dtype = tf.float32)

y_pred = tf.matmul(x,w)
loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=y_pred, labels=y)
loss = tf.reduce_mean(loss)

optm = tf.train.GradientDescentOptimizer(LR).minimize(loss)
init = tf.global_variables_initializer()

start_time = time.time()

loss_record = []
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(n_iter):
        idx = np.random.choice(m, size = n_batch)
        batch_X = train_X[idx,:]
        batch_y = train_y[idx]
        sess.run(optm, feed_dict = {x: batch_X, y: batch_y})

        if (epoch + 1) % n_prt == 0:
            loss_record.append(sess.run(loss, feed_dict = {x: batch_X, y: batch_y}))

    w_hat = sess.run(w)
```





Limitation of the Gradient Descent

Setting the Learning Rate

- How can we set the learning rate?

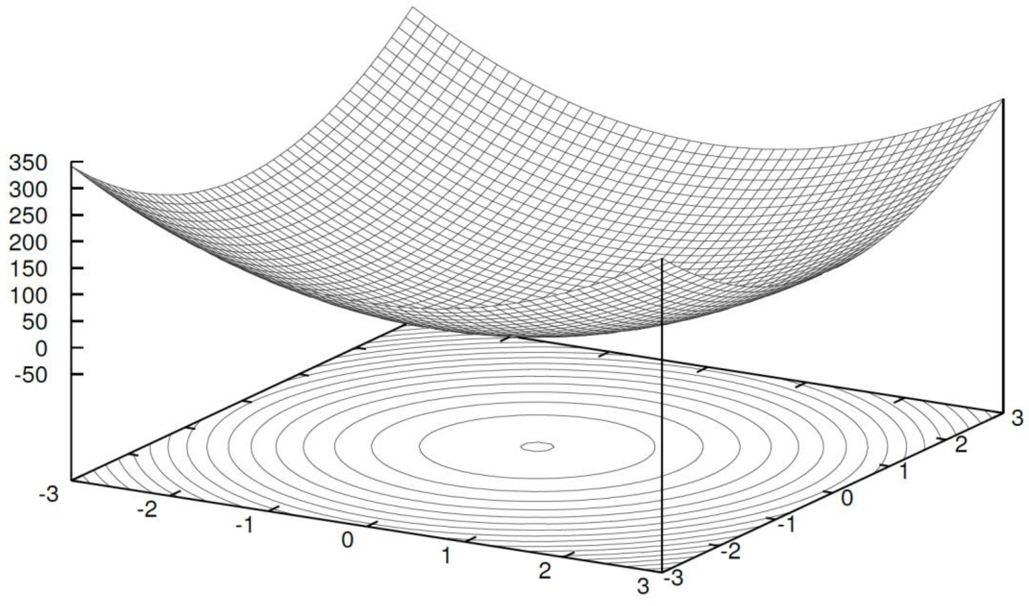
$$\omega_{k+1} = \omega_k - \alpha \nabla f(\omega_k)$$

- Small learning rate converges slowly and gets stuck in false local minima
- Large learning rates overshoot, become unstable and diverge

- Idea 1
  - Try lots of different learning rates and see what works “just right”
- Idea 2
  - Do something smarter! Design an adaptive learning rate that “adapts” to the landscape
  - Temporal and spatial

SGD Learning Rate (= Step Size)

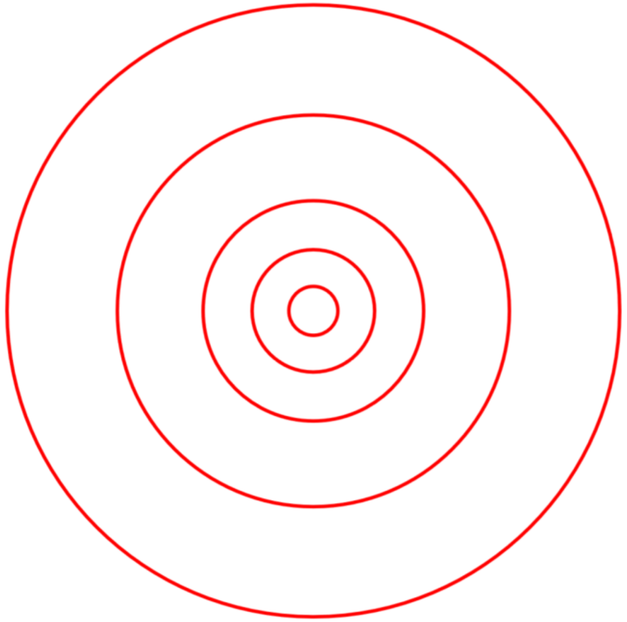
- The gradient descent method makes a strong assumption about the magnitude of the “local curvature” to fix the step size, and about its isotropy so that the same step size makes sense in all directions.



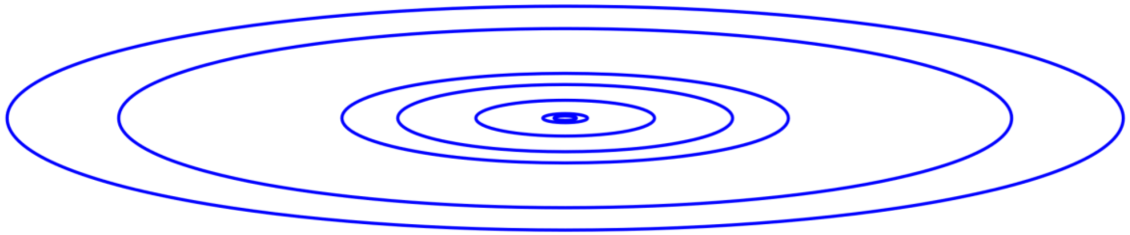
Source: Dr. Francois Fleuret at EPFL

SGD Learning Rate: Spatial

- We assign the same learning rate to all features



Nice (all features are equally important)



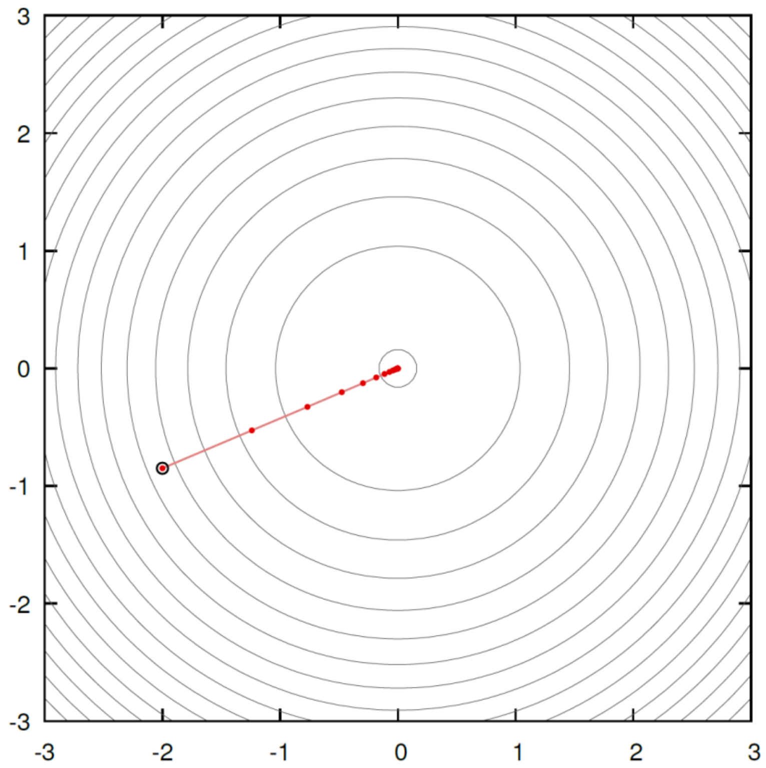
Harder !

Source: Dr. Francois Fleuret at EPFL

SGD Learning Rate: Spatial

- The gradient descent method makes a strong assumption about the magnitude of the “local curvature” to fix the step size, and about its isotropy so that the same step size makes sense in all directions.
- Nice (all features are equally important)

$\eta = 1.0e-2$

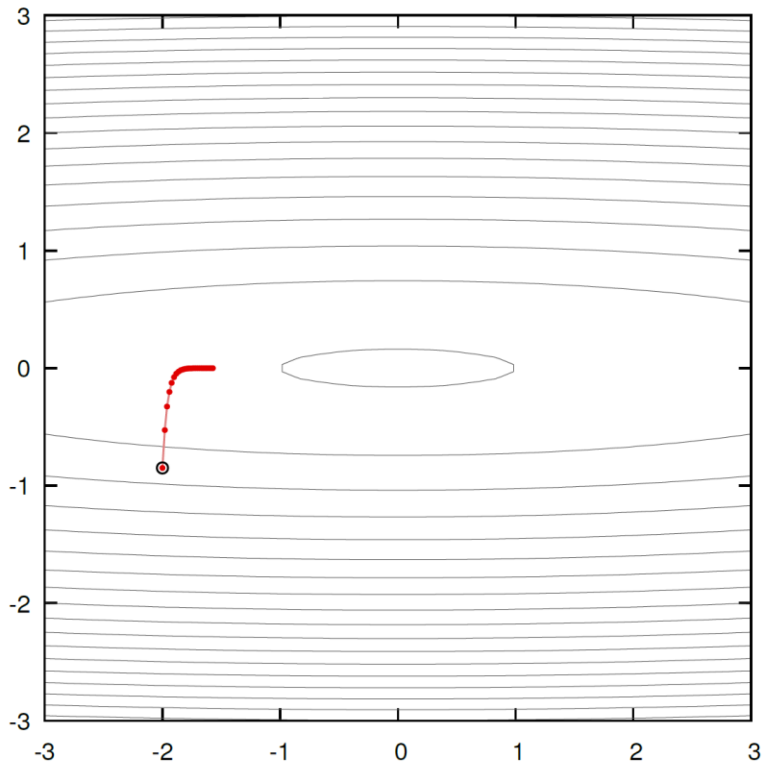


Source: Dr. Francois Fleuret at EPFL

SGD Learning Rate: Spatial

- The gradient descent method makes a strong assumption about the magnitude of the “local curvature” to fix the step size, and about its isotropy so that the same step size makes sense in all directions.
- Harder !

$\eta = 1.0e - 2$

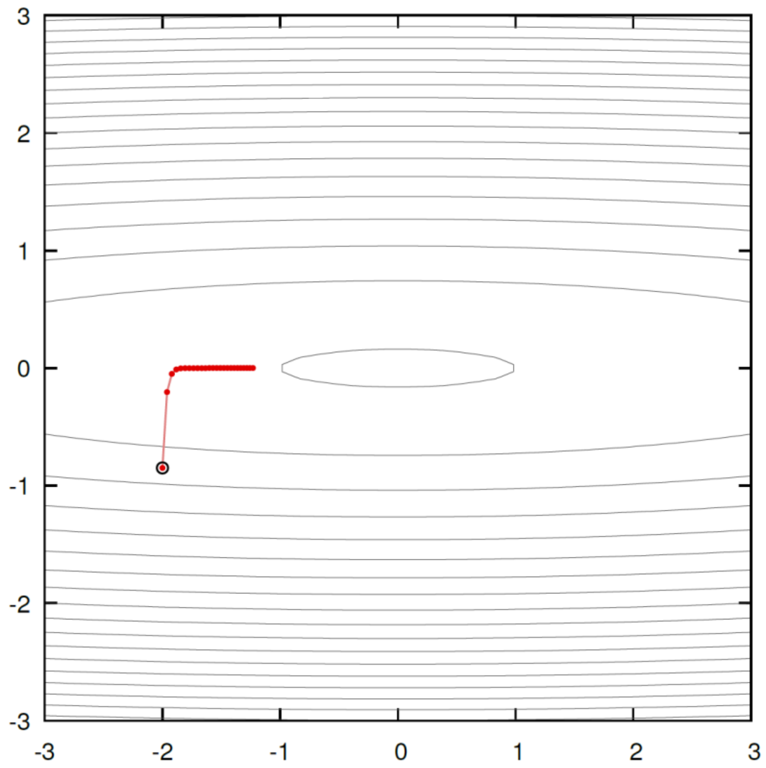


Source: Dr. Francois Fleuret at EPFL

SGD Learning Rate: Spatial

- The gradient descent method makes a strong assumption about the magnitude of the "local curvature" to fix the step size, and about its isotropy so that the same step size makes sense in all directions.
- Harder !

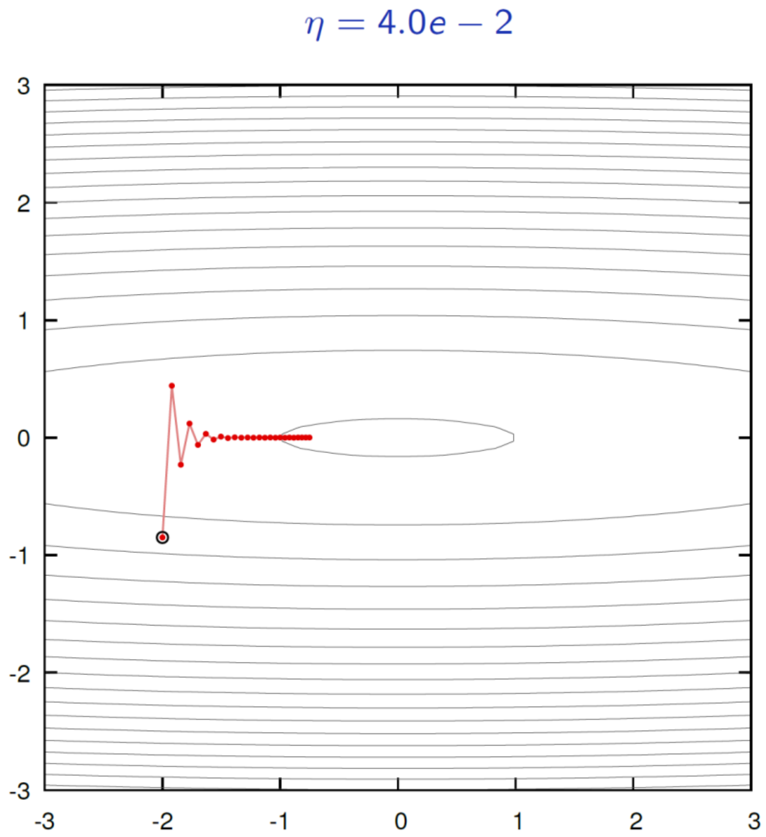
$\eta = 2.0e - 2$



Source: Dr. Francois Fleuret at EPFL

SGD Learning Rate: Spatial

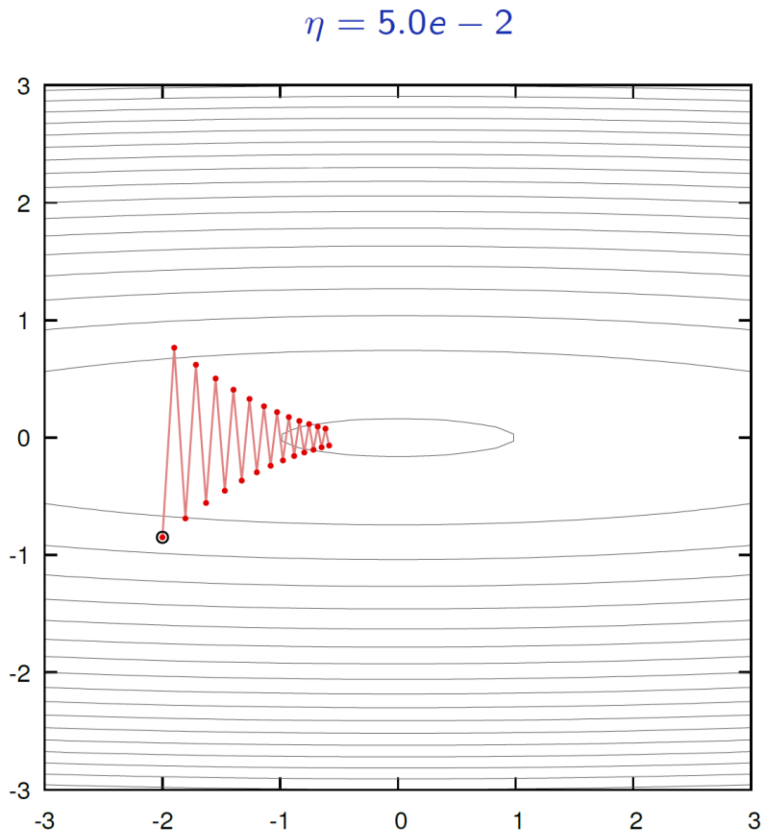
- The gradient descent method makes a strong assumption about the magnitude of the "local curvature" to fix the step size, and about its isotropy so that the same step size makes sense in all directions.
- Harder !



Source: Dr. Francois Fleuret at EPFL

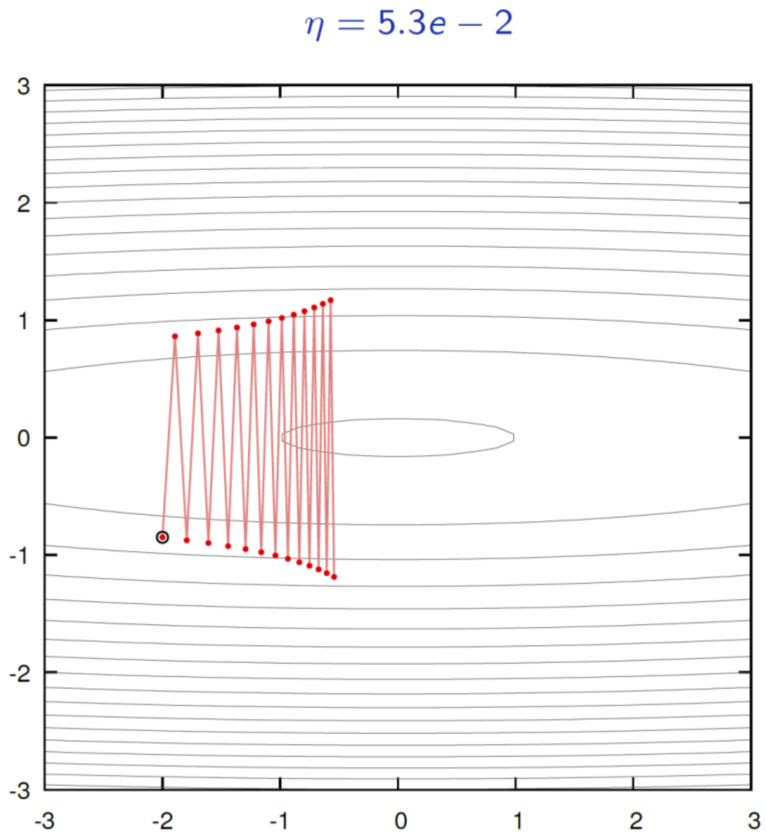
SGD Learning Rate: Spatial

- The gradient descent method makes a strong assumption about the magnitude of the "local curvature" to fix the step size, and about its isotropy so that the same step size makes sense in all directions.
- Harder !



SGD Learning Rate: Spatial

- The gradient descent method makes a strong assumption about the magnitude of the "local curvature" to fix the step size, and about its isotropy so that the same step size makes sense in all directions.
- Harder !



Source: Dr. Francois Fleuret at EPFL

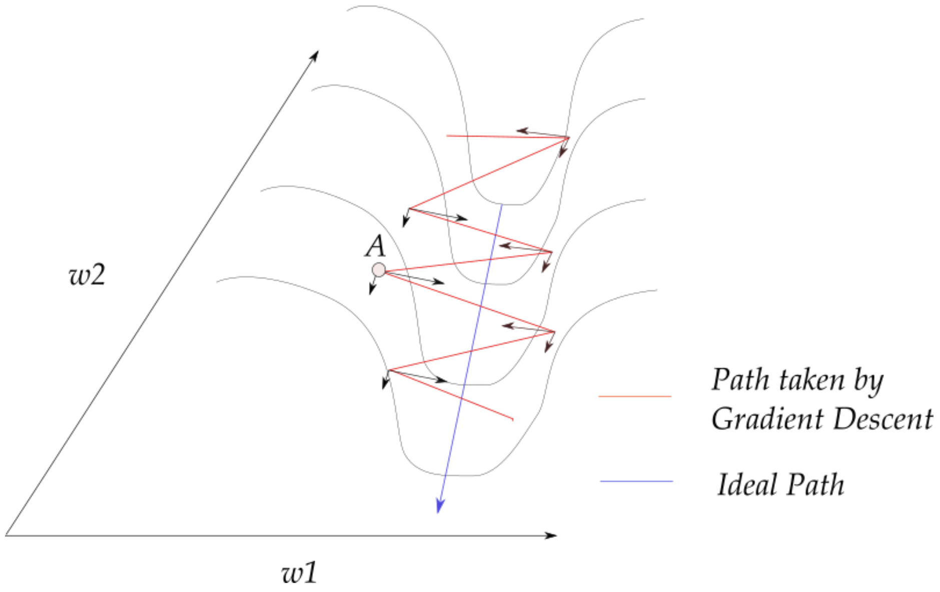
SGD Learning Rate: Temporal

- Deep-learning generally relies on a smarter use of the gradient, using statistics over its past values to make a “smarter step” with the current one.

- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum
  - Gradually decay the learning rate to reduce the fluctuations

Momentum

- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
  - Error surface has high curvature
  - The gradients are very noisy (zig-zag walk)



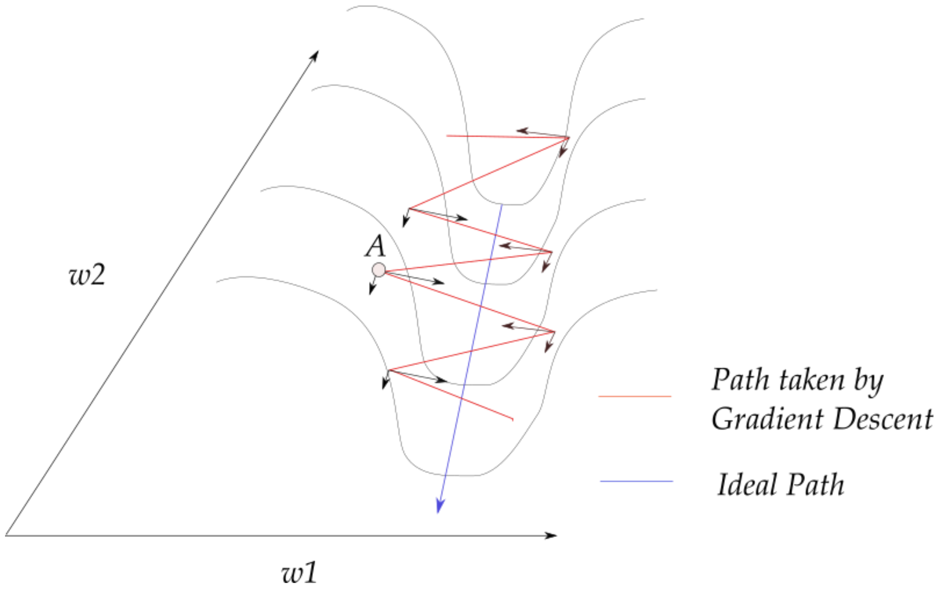
- Gradient Descent would move quickly down the walls, but very slowly through the valley floor
- How do we try and solve this problem?

Momentum

- Introduce a new variable v, the velocity

- We think of v as the direction and speed by which the parameters move as the learning dynamics progresses

- The velocity is an exponentially decaying moving average of the negative gradients



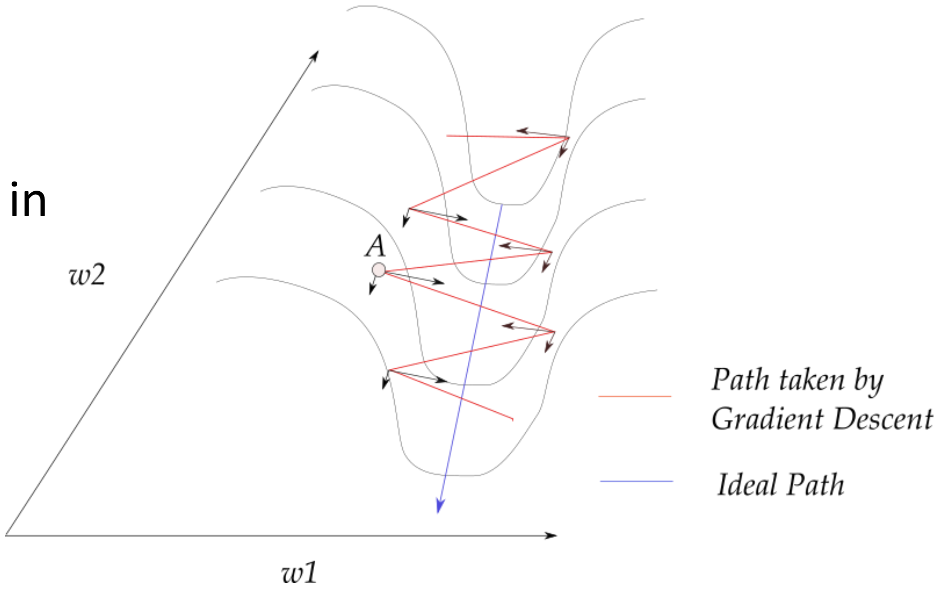
Mini-batch SGD with Momentum

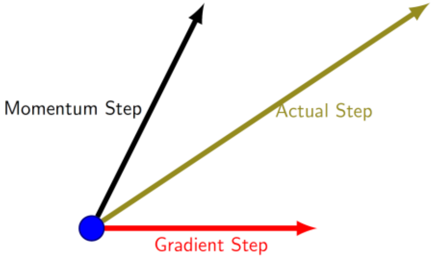
- The “vanilla” mini-batch stochastic gradient descent (SGD) consists of

$$\omega_{k+1} = \omega_k - \alpha \nabla f(\omega_k)$$

- The improvement is the use of a “momentum” to add inertia in the choice of the step direction

$$\omega_{k+1} = \omega_k + v_k$$
$$v_k = \gamma v_{k-1} - \alpha \nabla f(\omega_k)$$

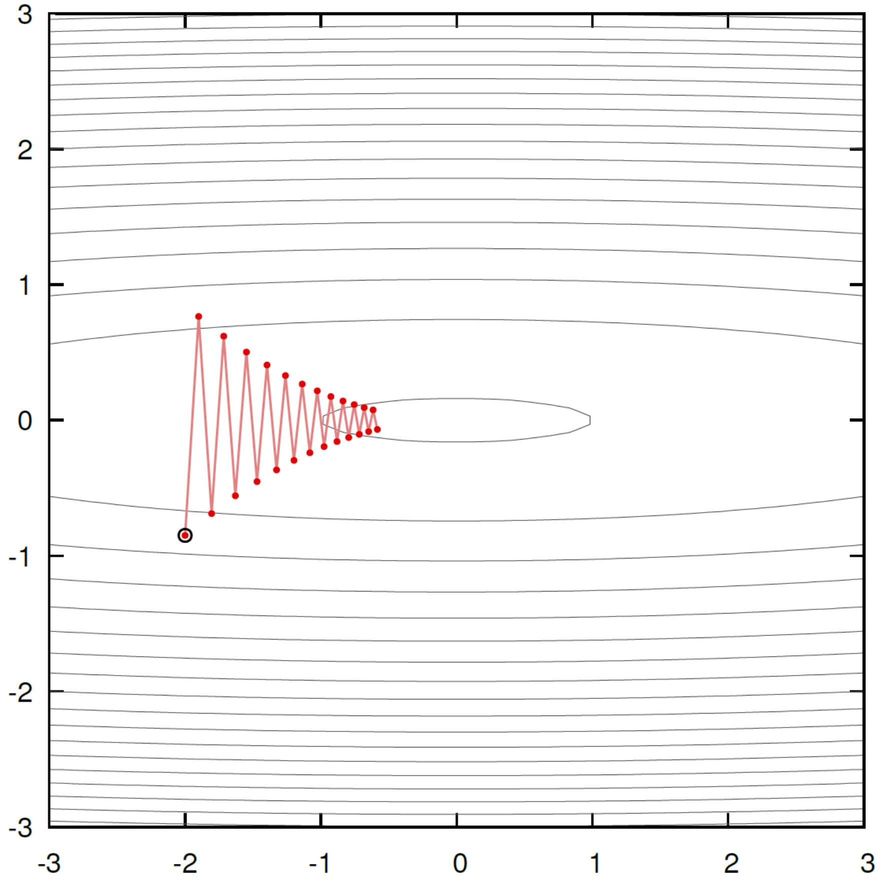




- With $\gamma = 0$, this is the same as vanilla SGD.

Mini-batch SGD with Momentum

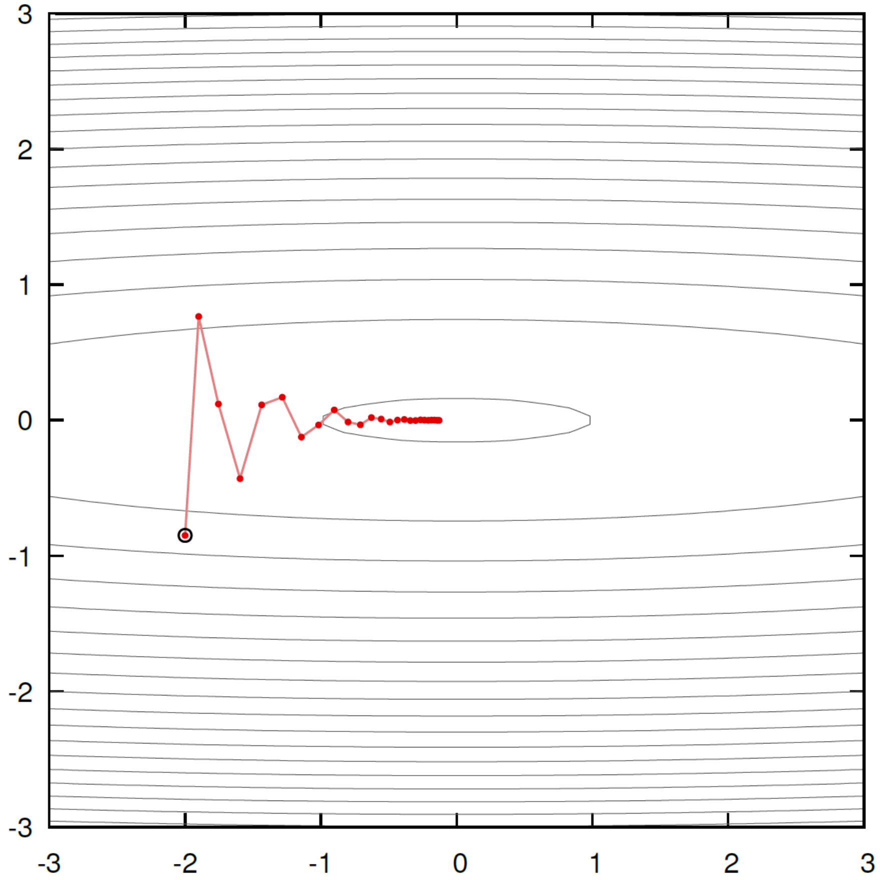
$\eta = 5.0e-2, \gamma = 0$



Source: Dr. Francois Fleuret at EPFL

Mini-batch SGD with Momentum

$\eta = 5.0e - 2, \gamma = 0.5$



Source: Dr. Francois Fleuret at EPFL

Adaptive Learning Rate Methods

- Momentum
- Adagrad
- Adadelta
- Adam
- RMSProp

tf.train.MomentumOptimizer

tf.train.AdagradOptimizer

tf.train.AdadeltaOptimizer

tf.train.AdamOptimizer

tf.train.RMSPropOptimizer

Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.

Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.

Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.

Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.

- Additional detail: http://ruder.io/optimizing-gradient-descent/