# Deep Learning

## 3.5 More on Gradient Descent

Dr. Konda Reddy Mopuri
kmopuri@iittp.ac.in
Dept. of CSE, IIT Tirupati

# Gradient Descent

1. By far the most common way to train neural networks
2. DL libraries provide various ways of implementing Gradient Descent

# Recap the Loss in Logistic Regression

1. $$\mathcal{L}(\mathbf{w}, \mathbf{b}) = -\sum_{n} \log(\sigma(y_n(\mathbf{w}\mathbf{x_n} + \mathbf{b})))$$

# Recap the Loss in Logistic Regression

1. $$\mathcal{L}(\mathbf{w}, \mathbf{b}) = -\sum_{n} \log(\sigma(y_n(\mathbf{w}\mathbf{x_n} + \mathbf{b})))$$

2. We computed the loss over all the training data and then computed the gradient

# Recap the Loss in Logistic Regression

1. $$\mathcal{L}(\mathbf{w}, \mathbf{b}) = -\sum_{n} \log(\sigma(y_n(\mathbf{w}\mathbf{x_n} + \mathbf{b})))$$
2. We computed the loss over all the training data and then computed the gradient
3. This is vanilla or Batch Gradient Descent

# Recap the Loss in Logistic Regression

1. $$\mathcal{L}(\mathbf{w}, \mathbf{b}) = -\sum_{n} \log(\sigma(y_n(\mathbf{w}\mathbf{x_n} + \mathbf{b})))$$
2. We computed the loss over all the training data and then computed the gradient
3. This is vanilla or Batch Gradient Descent
4. Sometimes very slow and intractable (datasets that do not fit in the memory)

# Recap the Loss in Logistic Regression

1. $$\mathcal{L}(\mathbf{w}, \mathbf{b}) = -\sum_n \log(\sigma(y_n(\mathbf{w}\mathbf{x_n} + \mathbf{b})))$$
2. We computed the loss over all the training data and then computed the gradient
3. This is vanilla or Batch Gradient Descent
4. Sometimes very slow and intractable (datasets that do not fit in the memory)
5. It doesn't allow updating the model online (i.e., with the arrival of new data samples, on the fly)

# Batch Gradient Descent

```
for i in range(nb_epochs):
  params_grad = evaluategradient(lossfunction, data, params)
  params = params - learning_rate * params_grad
```

# Batch Gradient Descent

```
for i in range(nb_epochs):
  params_grad = evaluategradient(lossfunction, data, params)
  params = params - learning_rate * params_grad
```

1. Batch GS is guaranteed to converge to global minima in case of convex functions, and to a local minima in case of non-convex functions

# Stochastic Gradient Descent (SGD)

1. Performs updates parameters for each training example
   $w = w - \eta \nabla_w \mathcal{L}(w, x^i, y^i)$

# Stochastic Gradient Descent (SGD)

1. Performs updates parameters for each training example
   $w = w - \eta \nabla_w \mathcal{L}(w, x^i, y^i)$
2. In case of large datasets, Batch GD computes redundant gradients for similar examples for each parameter update

# Stochastic Gradient Descent (SGD)

1. Performs updates parameters for each training example
   $w = w - \eta \nabla_w \mathcal{L}(w, x^i, y^i)$
2. In case of large datasets, Batch GD computes redundant gradients for similar examples for each parameter update
3. SGD does away with redundancy and generally faster and can be used to learn online

# Stochastic Gradient Descent (SGD)

1. However, frequent updates with a high variance cause the objective function to fluctuate heavily

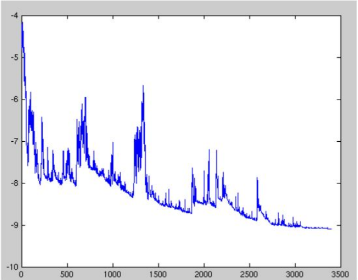



Figure credits: Wikipedia

# Stochastic Gradient Descent (SGD)

1. SGD's fluctuations enable it to jump to new and potentially better local minima

# Stochastic Gradient Descent (SGD)

1. SGD's fluctuations enable it to jump to new and potentially better local minima
2. This complicates the convergence, as it overshoots

# Stochastic Gradient Descent (SGD)

1. SGD's fluctuations enable it to jump to new and potentially better local minima
2. This complicates the convergence, as it overshoots
3. However, if the learning rate is slowly decreased, we can show similar convergence to Batch GD

# Stochastic Gradient Descent (SGD)

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for example in data:
    params_grad = evaluate_gradient(loss_function,
example, params)
    params = params - learning_rate * params_grad
```

# Mini-batch Gradient Descent

1. Takes the best of both worlds, updates the parameters for every mini-batch of n samples
$w = w - \eta \nabla_w \mathcal{L}(w, x^{i:i+n}, y^{i:i+n})$

# Mini-batch Gradient Descent

1. Takes the best of both worlds, updates the parameters for every mini-batch of n samples
   $w = w - \eta \nabla_w \mathcal{L}(w, x^{i:i+n}, y^{i:i+n})$
2. - Reduces the variance of the parameter updates, which can lead to more stable convergence
   - Can make use of highly optimized matrix optimizations

# Mini-batch Gradient Descent

1. Takes the best of both worlds, updates the parameters for every mini-batch of n samples
   $w = w - \eta \nabla_w \mathcal{L}(w, x^{i:i+n}, y^{i:i+n})$
2. 
   - Reduces the variance of the parameter updates, which can lead to more stable convergence
   - Can make use of highly optimized matrix optimizations
3. Common mini-batch sizes vary from 50 to 1024, depending on the application

# Mini-batch Gradient Descent

1. Takes the best of both worlds, updates the parameters for every mini-batch of n samples
   $w = w - \eta \nabla_w \mathcal{L}(w, x^{i:i+n}, y^{i:i+n})$
2. 
   - Reduces the variance of the parameter updates, which can lead to more stable convergence
   - Can make use of highly optimized matrix optimizations
3. Common mini-batch sizes vary from 50 to 1024, depending on the application
4. This is the algorithm of choice while training DNNs (also, incorrectly referred to as SGD in general)

# Mini-batch Gradient Descent

```
for i in range(nb_epochs):
  np.random.shuffle(data)
  for batch in get_batches(data, batch_size = 50):
    params_grad = evaluate_gradient(loss_function, batch,
params)
    params = params - learning_rate * params_grad
```

# Some challenges

1. Choosing a proper learning rate

# Some challenges

1. Choosing a proper learning rate
   - Learning rate schedules try to adjust it during the training

# Some challenges

1. Choosing a proper learning rate
   - Learning rate schedules try to adjust it during the training
   - However, these schedules are defined in advance and hence unable to adapt to the task at hand

# Some challenges

1. Choosing a proper learning rate
   - Learning rate schedules try to adjust it during the training
   - However, these schedules are defined in advance and hence unable to adapt to the task at hand
2. Same learning rate applies to all the parameters

# Some challenges

1. Choosing a proper learning rate
   - Learning rate schedules try to adjust it during the training
   - However, these schedules are defined in advance and hence unable to adapt to the task at hand
2. Same learning rate applies to all the parameters
3. Avoiding numerous sub-optimal local minima

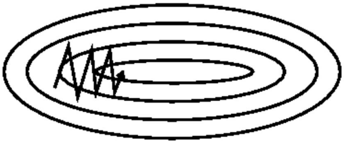
# Different update versions in GD

To deal with the discussed challenges, researchers proposed variety of update equations for GD

- SGD with momentum
- Nesterov Accelerated Gradient
- AdaGrad
- Adadelta
- Adam
- RMSProp
- etc.

# SGD with momentum

1. SGD has trouble when navigating through ravines (areas where the loss surface curves sharply in one direction than other; common near local optima)



# SGD with momentum

1. SGD has trouble when navigating through ravines (areas where the loss surface curves sharply in one direction than other; common near local optima)
2. SGD progresses slowly; oscillating in the ravine

# SGD with momentum

1. Momentum is a method that helps to accelerate in the relevant direction and dampens the oscillations

# SGD with momentum

1. Momentum is a method that helps to accelerate in the relevant direction and dampens the oscillations
2. Adds a fraction $\gamma$ of the previous update vector to the current one

$$
\begin{aligned}
v_t &= \gamma v_{t-1} + \eta \nabla_w \mathcal{L}(w) \\
w &= w - v_t
\end{aligned}
$$

# SGD with momentum

1. Momentum is a method that helps to accelerate in the relevant direction and dampens the oscillations
2. Adds a fraction $\gamma$ of the previous update vector to the current one

$$
\begin{aligned}
v_t &= \gamma v_{t-1} + \eta \nabla_w \mathcal{L}(w) \\
w &= w - v_t
\end{aligned}
$$

3. $\gamma$ is usually set to 0.9

# SGD with momentum

$$
\begin{aligned}
v_t =& \gamma v_{t-1} + \eta \nabla_w \mathcal{L}(w) \\
w =& w - v_t
\end{aligned}
$$

1. Momentum term
    - Increases the update for the components whose gradient points in the same direction
    - Decreases for the dimensions whose gradient change direction across iterations