Deep Learning for Computer Vision

Dr. Konda Reddy Mopuri
Mehta Family School of Data Science and Artificial Intelligence
IIT Guwahati
Aug-Dec 2022

Previously on DA621....

- Scoring function (e.g., linear classifier: $f(x) = W^T x + b$)

Previously on DA621....

- Scoring function (e.g., linear classifier: $f(x) = W^T x + b$)
- Loss function (e.g., hinge, softmax losses)

Optimization

- How to find the model parameters that minimize the loss function?

$$w^* = \underset{w}{\operatorname{argmin}}\, L(w)$$

Optimization

- How to find the model parameters that minimize the loss function?

$$w^* = \underset{w}{\operatorname{argmin}} \, L(w)$$

- General and vast, but we will discuss within our context

Optimization

- Finding the parameters that minimize the training loss

$$W^*, \mathbf{b}^* = \underset{W, \mathbf{b}}{\operatorname{argmin}} \, \mathcal{L}(f(\cdot; W, \mathbf{b}); \mathcal{D})$$

Optimization

- Finding the parameters that minimize the training loss

$$W^*, \mathbf{b}^* = \underset{W,\mathbf{b}}{\operatorname{argmin}}\, \mathcal{L}(f(\cdot; W, \mathbf{b}); \mathcal{D})$$

- How do we find these optimal parameters?

Optimization

- Finding the parameters that minimize the training loss

$$W^*, \mathbf{b}^* = \underset{W,\mathbf{b}}{\operatorname{argmin}}\, \mathcal{L}(f(\cdot; W, \mathbf{b}); \mathcal{D})$$

- How do we find these optimal parameters?
  - Closed form solution (e.g. linear regression)

Optimization

- Finding the parameters that minimize the training loss

$$W^*, \mathbf{b}^* = \underset{W,\mathbf{b}}{\operatorname{argmin}} \mathcal{L}(f(\cdot; W, \mathbf{b}); \mathcal{D})$$

- How do we find these optimal parameters?
  - Closed form solution (e.g. linear regression)
  - Ad-hoc recipes (e.g. Perceptron, K-NN classifier)

Optimization

- Finding the parameters that minimize the training loss

$$W^*, \mathbf{b}^* = \underset{W,\mathbf{b}}{\operatorname{argmin}}\, \mathcal{L}(f(\cdot; W, \mathbf{b}); \mathcal{D})$$

- How do we find these optimal parameters?
  - Closed form solution (e.g. linear regression)
  - Ad-hoc recipes (e.g. Perceptron, K-NN classifier)
  - What if the loss function can't be minimized analytically?

Optimization

Source: travelholicq.com

Optimization

Source: travelholicq.com

Optimization

Source: travelholicq.com

Not-so-intelligent idea!

- `Probe random directions`

Not-so-intelligent idea!

- Probe random directions
- Progress if you find a useful direction

Not-so-intelligent idea!

- `Probe random directions`
- `Progress if you find a useful direction`
- `Repeat`

Not-so-intelligent idea!

- Probe random directions
- Progress if you find a useful direction
- Repeat
- Very ineffective!

A better looking one: Follow the slope!

- `Sense the slope around the feet`

A better looking one: Follow the slope!

- Sense the slope around the feet
- Identify the steepest direction, make a brief progress

A better looking one: Follow the slope!

- Sense the slope around the feet
- Identify the steepest direction, make a brief progress
- Repeat until convergence!

Derivative and Gradient

- In 1D, derivative of a function gives the slope

$$\frac{\partial f}{\partial x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{h}$$

Derivative and Gradient

- In 1D, derivative of a function gives the slope

$$\frac{\partial f}{\partial x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{h}$$

- In higher dimensions, given a function

$$f : \mathcal{R}^D \to \mathcal{R}$$

gradient is the mapping

$$\begin{aligned} \nabla f : & \mathcal{R}^D \to \mathcal{R}^D \\ & x \to \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_D} \right) \end{aligned}$$

Derivative and Gradient

- In 1D, derivative of a function gives the slope

$$\frac{\partial f}{\partial x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{h}$$

- In higher dimensions, given a function

$$f : \mathcal{R}^D \to \mathcal{R}$$

gradient is the mapping

$$\begin{aligned} \nabla f : & \mathcal{R}^D \to \mathcal{R}^D \\ & x \to \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_D} \right) \end{aligned}$$

- ∇f vector gives the direction and rate of fastest increase for f.

Gradient Descent

- Goal is to minimize the error (or loss): determine the parameters w that minimize the loss $\mathcal{L}(w)$

Gradient Descent

- Goal is to minimize the error (or loss): determine the parameters w that minimize the loss $\mathcal{L}(w)$
- Gradient points uphill $\rightarrow$ negative of gradient points downhill

Gradient Descent

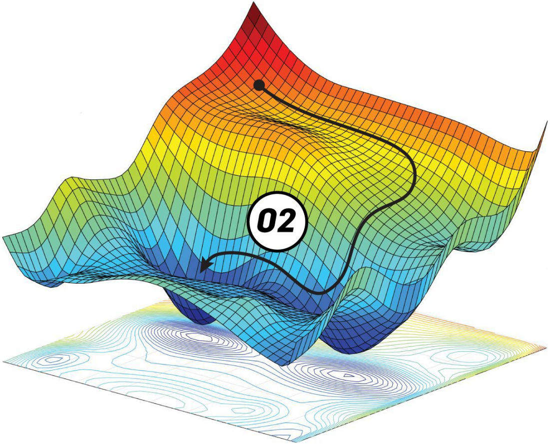


Figure credits:Ahmed Fawzy Gad

Gradient Descent

1. Start with an arbitrary initial parameter vector w_0

Gradient Descent

1. `Start with an arbitrary initial parameter vector` w_0
2. `Repeatedly modify it via updating in small steps`

Gradient Descent

1. Start with an arbitrary initial parameter vector w_0
2. Repeatedly modify it via updating in small steps
3. At each step, modify in the direction that produces steepest descent along the error surface

How to compute the gradient?

- Numerically, for each component of w using the derivative formula

$$\frac{\partial f}{\partial x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

How to compute the gradient?

- Numerically, for each component of w using the derivative formula

$$\frac{\partial f}{\partial x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

- Slow and approximate!

How to compute the gradient?

- Analytically, using calculus for computing the derivatives

$$L_i = \sum_{j \neq y_i} max\{0, s_j - s_{y_i} + 1\}$$

$$L = \frac{1}{N}\sum_i L_i + \sum_k w_k^2$$

$$s = f(x, W)$$

$$\nabla L_{iw}?$$

How to compute the gradient?

- Analytically, using calculus for computing the derivatives

$$L_i = \sum_{j \neq y_i} max\{0, s_j - s_{y_i} + 1\}$$

$$L = \frac{1}{N} \sum_i L_i + \sum_k w_k^2$$

$$s = f(x, W)$$

$$\nabla L_{iw}?$$

- Analytic way is fast, exact, but error-prone!

Batch Gradient Descent

```
for i in range(nb_epochs):
  ∇L_w = evaluate_gradient(L, 𝒟, w)
  w = w - η * ∇L_w
```

$$
\begin{aligned}
&\texttt{for i in range(nb\_epochs):}\\
&\quad \nabla L_w \;\texttt{=}\; \texttt{evaluate\_gradient(L,}\ \mathcal{D}\texttt{, w)}\\
&\quad \texttt{w = w -}\ \eta\ \texttt{*}\ \nabla L_w
\end{aligned}
$$

Batch Gradient Descent

for i in range(nb_epochs):
  ∇L_w = evaluate_gradient(L, $\mathcal{D}$, w)
  w = w - η * ∇L_w

1. 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

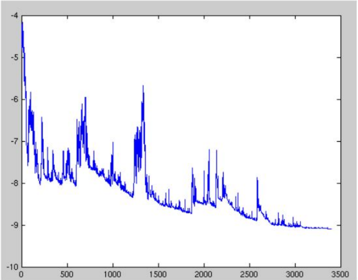


Figure credits: Wikipedia

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(𝒟)
  for x_i ∈ 𝒟:
    ∇L_w = evaluate_gradient(L, x_i, w)
    w = w − η * ∇L_w
```

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 32 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 32 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(𝒟)
  for batch in get_batches(𝒟, batch_size = 128):
   ∇L_w = evaluate_gradient(L, batch, w)
    w = w − η ∗ ∇L_w
```

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 γ 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 γ of the previous update vector to the current one

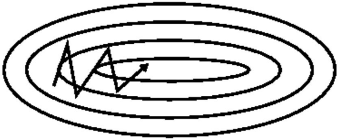
$$
\begin{aligned}
v_t =& \gamma v_{t-1} + \eta \nabla_w \mathcal{L}(w) \\
w =& w - v_t
\end{aligned}
$$

3. γ 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



References

https://ruder.io/optimizing-gradient-descent/