Deep Learning basics

Optimization and Gradient Descent

Learning goals

- Recall the concept of optimization
- Understand basics of gradient descent

OPTIMIZATION

- Optimization: Minimize some function $J(\vec{\theta})$ by altering $\vec{\theta}$.
- Maximize $f(\vec{\theta})$ by minimizing $J(\vec{\theta}) = -f(\vec{\theta})$
- $J(\vec{\theta})$:
  - *“criterion”, “objective function”, “cost function”, “loss function”, “error function”*
  - In a probabilistic machine learning setting often (conditional) negative log-likelihood:

    $$-\log p(\vec{X}; \vec{\theta})$$

    or

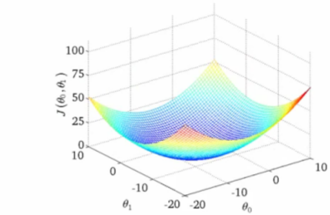
    $$-\log p(\vec{y}|\vec{X}; \vec{\theta})$$

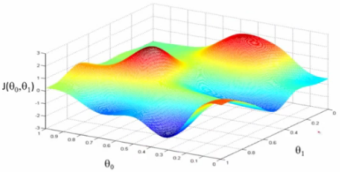
    as a function of $\vec{\theta}$
  - $\vec{\theta}^* = \arg\min_{\vec{\theta}} J(\vec{\theta})$

OPTIMIZATION

- If $J(\vec{\theta})$ is convex, it is minimized where $\nabla_{\vec{\theta}} J(\vec{\theta}) = \vec{0}$
- If $J(\vec{\theta})$ is not convex, the gradient can help us to improve our objective nevertheless (and find a local optimum).
- Many optimization techniques were originally developed for convex objective functions, but are found to be working well for non-convex functions too.
- Use the fact that gradient indicates the slope of the function in the direction of steepest increase.

OPTIMIZATION

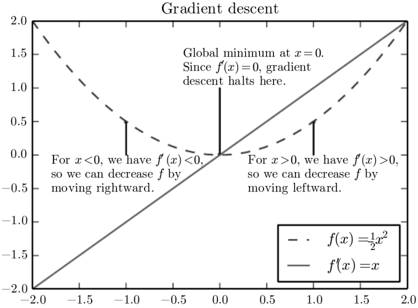




GRADIENT-BASED OPTIMIZATION

- Derivative: Given a small change in input, what is the corresponding change in output?

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x)$$



- $f(x - \epsilon \, \mathrm{sign}\, f'(x)) < f(x)$ for small enough ϵ

GRADIENT DESCENT

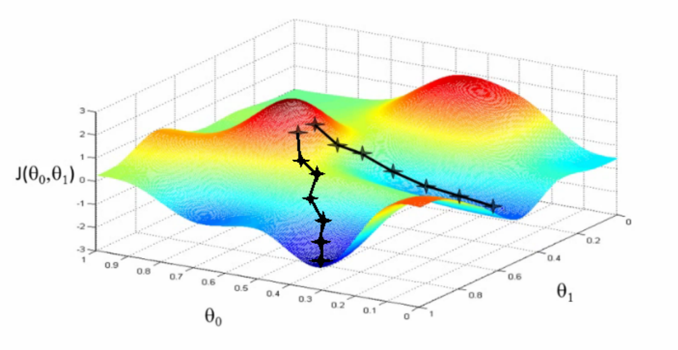
- For $J(\vec{\theta}) : \mathbb{R}^n \to \mathbb{R}$
- If partial derivative $\frac{\partial J(\vec{\theta})}{\partial \theta_j} > 0$, $J(\vec{\theta})$ will increase for small increases of θ_j
  $\Rightarrow$ go in opposite direction of gradient (since we want to minimize)
- Steepest descent: iterate

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \eta \nabla_{\vec{\theta}} J(\vec{\theta}_t)$$

  where $\vec{\theta}_t$ is the actual parameter, $J(\vec{\theta}_t)$ is the objective function evaluated at $\vec{\theta}_t$ and $\vec{\theta}_{t+1}$ is the updated parameter.
- η is the learning rate (set to small positive constant).
- Converges if $\nabla_{\vec{\theta}} J(\vec{\theta})$ is (close to) $\vec{0}$

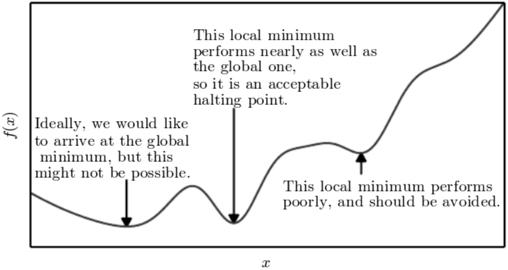
LOCAL MINIMA

- If the function is non-convex, different results can be obtained at convergence, depending on initialization of $\vec{\theta}$.

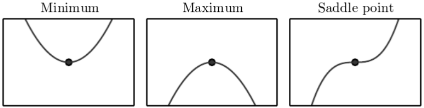


LOCAL MINIMA

- Minima can be global or local:



- Critical (stationary) points: $f'(x) = 0$



- For neural networks, only good (not perfect) parameter values can be found.

GRADIENT DESCENT FOR LOGISTIC REGRESSION

$$\nabla_{\vec{\theta}} NLL(\vec{\theta}) = -\nabla_{\vec{\theta}} \sum_{i=1}^{m} y^{(i)} \log \sigma(\vec{\theta}^T \vec{x^{(i)}}) + (1 - y^{(i)}) \log(1 - \sigma(\vec{\theta}^T \vec{x^{(i)}}))$$

$$= -\sum_{i=1}^{m} (y^{(i)} - \sigma(\vec{\theta}^T \vec{x^{(i)}})) \vec{x^{(i)}}$$

- The gradient descent update becomes:

$$\vec{\theta}_{t+1} := \vec{\theta}_t + \eta \sum_{i=1}^{m} (y^{(i)} - \sigma(\vec{\theta}_t^T \vec{x}^{(i)})) \vec{x}^{(i)}$$

- Note: Which feature weights are increased, which are decreased?

DERIVATION OF GRADIENT FOR LOGISTIC REGRESSION

This is a great exercise! Use the following facts:

Gradient	$(\nabla_{\vec{\theta}} f(\vec{\theta}))_j = \frac{\partial f(\vec{\theta})}{\partial \theta_j}$
Derivative of a sum	$\frac{d}{dz} \sum_i f_i(z) = \sum_i \frac{df_i(z)}{dz}$
Chain rule	$F(z) = f(g(z)) \ \Rightarrow \ F'(z) = f'(g(z))g'(z)$
Derivative of logarithm	$\frac{d \log z}{dz} = 1/z$
D. of logistic sigmoid	$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$
Partial d. of dot-product	$\frac{\partial \vec{\theta}^T \vec{x}}{\partial \theta_j} = \vec{x}_j$

GRADIENT DESCENT: SUMMARY

- Iterative method for function minimization.
- Gradient indicates rate of change in objective function, given a local change to feature weights.
- Substract the gradient:
  - **decrease** parameters that (locally) have **positive** correlation with objective
  - **increase** parameters that (locally) have **negative** correlation with objective
- Gradient updates only have the desired properties in a small region around previous parameters $\vec{\theta}_t$. Control locality by step-size η.
- Gradient descent is slow: For relatively small step in the right direction, all of training data has to be processed.
- This version of gradient descent is often also called *batch gradient descent*.

STOCHASTIC GRADIENT DESCENT (SGD)

- *Batch gradient descent* is slow: For relatively small step in the right direction, all of training data has to be processed.

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \eta \nabla_{\vec{\theta}} \sum_{i=1}^{m} \log p(y_i | \vec{x}_i; \vec{\theta})$$

- *Stochastic gradient descent* in a nutshell:
  - For each update, only use random sample $\mathbb{B}_t$ of training data (mini-batch).

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \eta \nabla_{\vec{\theta}} \sum_{i \in \mathbb{B}_t} \log p(y_i | \vec{x}_i; \vec{\theta})$$

  - Mini-batch size can also just be 1.

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \eta \nabla_{\vec{\theta}} \log p(y_t | \vec{x}_t; \vec{\theta})$$

- $\Rightarrow$ More frequent updates.

STOCHASTIC GRADIENT DESCENT (SGD)

- The actual gradient is *approximated* using only a sub-sample of the data.
- For objective functions that are highly non-convex, the random deviations of these approximations may even help to escape local minima.
- Treat batch size and learning rate as hyper-parameters.