TEXAS
The University of Texas at Austin

TOPIC 4 NONLINEAR PROGRAMMING

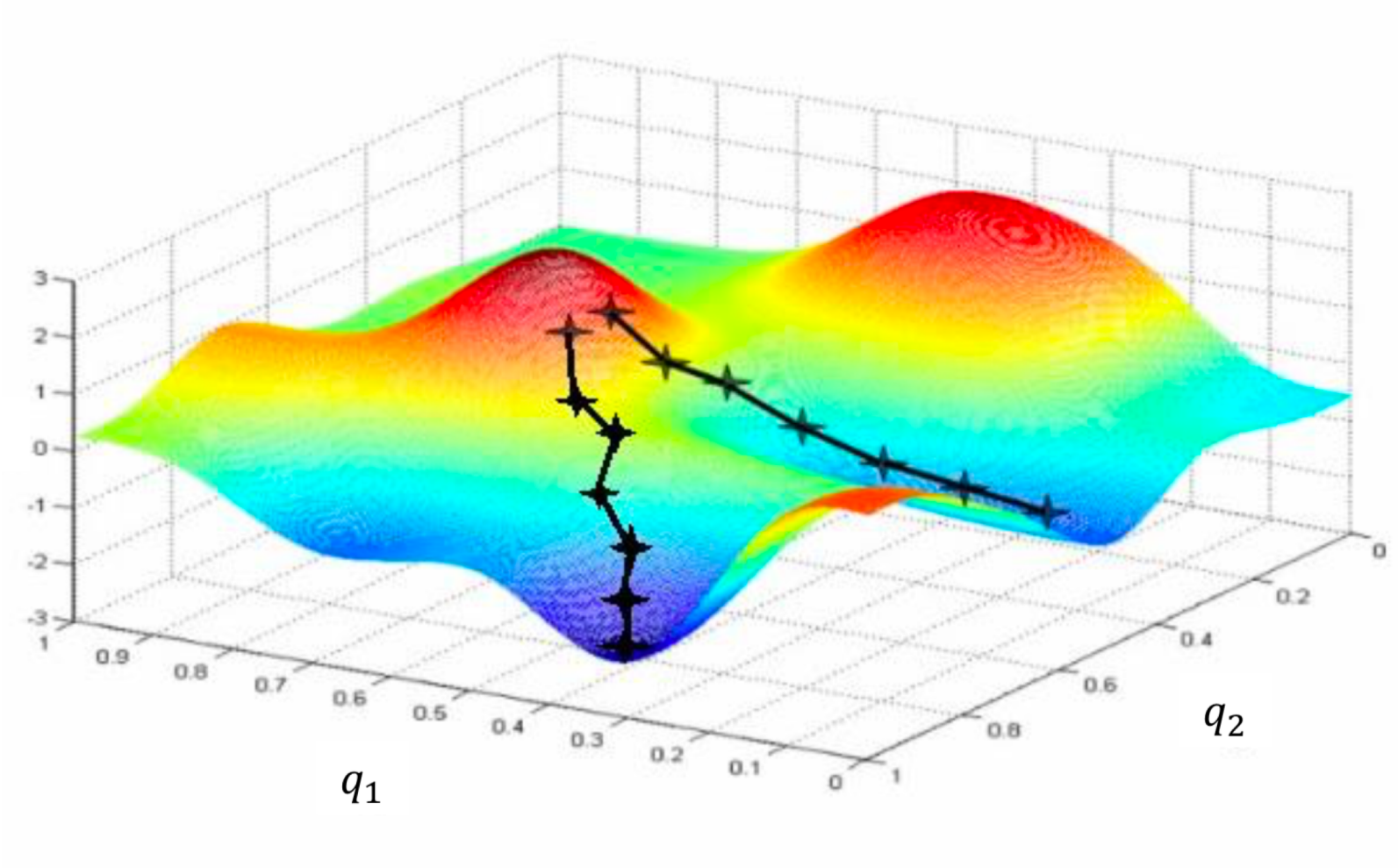
Solution Algorithms

- There are LOTS of algorithms to solve NLPs
- Most of them are guaranteed to find local optimal solutions (sometimes local is global)
- Very few algorithms exist that are guaranteed to find the global solution
  - Most are VERY slow
  - Hopefully, finding a local optimum is good enough?

Gradient Descent

- The most well-known algorithm is called gradient descent
- Imagine you're on top of a mountain trying to get to the ground but it's very foggy
  - You can only see a few feet in any direction
- You take a small step in a direction
- Once you make a step you can see a little more and make another step
- Which way should you take your step?
  - The direction of steepest descent

Gradient Descent



Gradient Descent

- Gradient descent is guaranteed to find a local optimal solution
- How do we find the steepest descent direction?
- It turns out calculus tells us that this is the negative of the gradient
  - Vector of partial derivatives
- $x^{n+1} = x^n - \lambda \nabla f(x^n)$

Gradient Descent

- Imagine our objective is to minimize a mean squared error on n data points (regression/ML/NN)
  - $\min \frac{1}{n} \sum_{i=1}^{n} Loss_i$
- The gradient is of the form $\frac{1}{n} \sum_{i=1}^{n} \nabla Loss_i$
  - This is an average of several gradients
- $x_{k+1} = x_k - \lambda \frac{1}{n} \sum_{i=1}^{n} \nabla Loss_i$
- λ is called the learning rate

Gradient Solvers

- Let's look at some gradient descent algorithms on a regression problem in python
- The gradient of regression is easier to calculate than a neural network..
- We know the true solution, so we can compare answers

Problems with Gradient Descent

- It can be slow
  - Lots of iterations to converge
  - Each iteration may take a long time if there are
    - Many data points
    - Many decision variables (parameters)
- It can get stuck at local minima

Stochastic Gradient Descent

- If n is big or there are lots of parameters, calculating the gradient can be VERY slow
- One quick way to speed this up is to remember stats 101
  - We can *approximate* a population average by using a sample
  - Don't use every data point when calculating the gradient!

Stochastic Gradient Descent

- Shuffle your data randomly (stochastic)
- Split the shuffled data into batches
- Take a gradient descent step for each batch (adjust n)
- All these steps together is called an epoch
- This process is called stochastic gradient descent

Stochastic Gradient Descent

1. Randomly split data into B batches of (roughly) equal size
2. for b = 1, 2, …, B
   a. Calculate $\nabla Loss_i$ at x_k for each i in batch b
   b. $x_{k+1} = x_k - \lambda \frac{1}{n_b} \sum_{i=1}^{n_b} \nabla Loss_i$
3. Go back to 1 until error tolerance is met

- Step 2 is called an epoch

Stochastic Gradient Descent

- By going through all the data once, we have made many steps with SGD instead of 1 step with GD
- However, each step was just an approximation of the steepest descent direction
- When we're walking down the mountain, we're a little tipsy…
- This can be a good thing though
  - You might get out of local minima valleys!

Stochastic Gradient Descent

- There have been many modifications of SGD recently to help the accuracy
- One of the best solvers today is ADAM, which makes a couple tweaks to SGD
  - Momentum
  - Adaptive learning rate with momentum
- https://arxiv.org/abs/1412.6980

Momentum

- We approximate the gradient at each step
- Let's tie those approximations together
  - An error in 1 batch can be offset by errors in other batches
- Instead of using just this batch's gradient to step, we will use a weighted average of all past gradients to step
- Errors in each batch will be averaged with other errors to make the approximate gradient more accurate

Momentum

1. Start with $m_1 = 0$
2. In each SGD step, k, call the approximate gradient g_k
   - $g_k = \frac{1}{n_b}\sum_{i=1}^{n_b} \nabla Loss_i$
3. Let $m_k = \theta_1 m_{k-1} + (1-\theta_1) g_k$
4. Assign $\hat{m}_k = \frac{m_k}{(1-\theta_1^k)}$
5. Take an SGD step: $x_{k+1} = x_k - \lambda \hat{m}_k$

- $\theta_1 = 0.9$ tends to work pretty well

Adaptive Learning Rate

- In each SGD step, each parameter uses the same learning rate
- If a parameter has a small gradient, relative to others, its parameter will not be updated very much

- It will take lots of SGD steps to find the optimum
- This problem is often referred to as sparse gradients

Adaptive Learning Rate

- One way to address this would be to just use the sign of the gradient (+/- 1)
  - This is a little too aggressive, non-smooth, and won't converge
- We fix this by giving each parameter its own learning rate
  - Adaptive learning rate
  - The gradient with its adaptive learning rate behaves kind of like the sign of the gradient, but better…
  - Let's use the same momentum trick on the learning rates!
- Note that $sign(f) = \frac{f}{\sqrt{f^2}}$

Adaptive Learning Rate

1. Start with $v_1 = 0$
2. In each SGD step, k, call the approximate gradient g_k
   - $g_k = \frac{1}{n_b}\sum_{i=1}^{n_b} \nabla Loss_i$
3. Let $v_k = \theta_2 v_{k-1} + (1-\theta_2) g_k^2$
4. Call $\hat{v}_k = \frac{v_k}{(1-\theta_2^k)}$
5. Take an SGD step: $x_{k+1} = x_k - \lambda \frac{\hat{m}_k}{\sqrt{\hat{v}_k}+\epsilon}$ (elementwise division)

- This is ADAM!
- $\theta_2 = 0.999$ is commonly used

AdaGrad

- AdaGrad was developed before ADAM
- It just uses an adaptive learning rate, without momentum

1. Start with $G_1 = 0$
2. In each SGD step, k, call the approximate gradient g_k
3. Let $G_k = G_{k-1} + g_k^2$
4. Take an SGD step: $x_{k+1} = x_k - \lambda \frac{g_k}{\sqrt{G_k + \epsilon}}$

- Unfortunately, G can get very big, and learning might stop before convergence ☹

RMSProp

- RMSProp is another popular SGD variant
- It uses the same adaptive learning rate as Adam, but no momentum

1. Start with $v_1 = 0$
2. In each SGD step, k, call the approximate gradient g_k
   - $g_k = \frac{1}{n_b}\sum_{i=1}^{n_b} \nabla Loss_i$
3. Let $v_k = \theta_2 v_{k-1} + (1-\theta_2) g_k^2$
4. Call $\hat{v}_k = \frac{v_k}{(1-\theta_2^k)}$
5. Take an SGD step: $x_{k+1} = x_k - \lambda \frac{g_k}{\sqrt{\hat{v}_k}+\epsilon}$

Neural Networks

- These solvers are often used to find the parameters of neural networks
- Neural networks often have
  - Large data sets
  - Many parameters
  - Sparse gradients
- ADAM tends to work very well for neural networks