Computational Statistics
Optimisation

Luca Bortolussi

Department of Mathematics and Geosciences
University of Trieste

Office 238, third floor, H2bis
luca@dmi.units.it

Trieste, Winter Semester 2016/2017

Outline

BASICS

- Consider a function $f(\mathbf{x})$, from $\mathbb{R}^n$ to $\mathbb{R}$, twice differentiable. Their minima are points such that $\nabla f(\mathbf{x}) = 0$.
- At a minimum $\mathbf{x}^*$ of f, the Hessian matrix $H_f(\mathbf{x}^*)$ is positive semidefinite, i.e. $\mathbf{v}^T H_f \mathbf{v} \geq 0$.
- If a point $\mathbf{x}^*$ is such that (a) $\nabla f(\mathbf{x}) = 0$ and (b) $H_f(\mathbf{x}^*)$ is positive definite, then $\mathbf{x}^*$ is a minimum of f.
- For a quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} + c$ the condition $\nabla f(\mathbf{x}) = 0$ reads $A\mathbf{x} - \mathbf{b} = 0$.
- If A is invertible and positive definite, then the point $\mathbf{x}^* = A^{-1}\mathbf{b}$ is the unique minimum of f, as f is a convex function.

GRADIENT DESCENT

- Notation. $\mathbf{x}_k$ denotes the sequence of points of the descent. $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$. The update is in the direction $\mathbf{p}_k$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta_k \mathbf{p}_k$$

- In gradient descent, at a point $\mathbf{x}$, take a step towards $-\nabla f(\mathbf{x})$, hence in the update rule becomes we set $\mathbf{p}_k = -\mathbf{g}_k$.
- In the simplest case, $\eta_k = \eta$. If η is not small enough, we can step over the minimum. If η is very small this usually not happens, but convergence is very slow.
- For a quadratic function, we have that $\mathbf{p}_k = -A\mathbf{x}_k + \mathbf{b}$.

STOCHASTIC GRADIENT DESCENT

- If the function to minimise is of the form $f(\mathbf{x}) = \sum_{i=1}^{N} f_i(\mathbf{x})$, as is the case for ML problems, then we can use stochastic gradient descent, which instead of taking a step along $\mathbf{g}_k$, it steps along the direction $-\nabla f_i(\mathbf{x_k})$.
- The algorithm iterates over the dataset one or more times, typically shuffling it each time.
- Alternatively to one single observations, small batches (mini-batches) of observations can be used to improve the method.

SGD, CROSS ENTROPY, AND MINI-BATCHES

- The cross-entropy between distributions p and q is:

$$H(p, q) = H(p) + D_{KL}(p \parallel q) = -\sum_x p(x) \log q(x)$$

- The empirical distribution p_{emp} of the dataset $(\mathbf{x}_i, y_i)$ gives to each observed point probability $1/N$, for N total points.
- Maximizing the log likelihood is the same as minimizing the cross entropy between the empirical probability and the probability predicted by the model. Calling the loss function $L(f(\mathbf{x}_i, \theta), y_i) = \log p(y_i \mid x_i, \theta)$, this is

$$H(p_{emp}, p) = \frac{1}{N} \sum_i L(f(\mathbf{x}_i, \theta), y_i)$$

SGD, CROSS ENTROPY, AND MINI-BATCHES

- The cross entropy between the model and the true data distribution is

$$H(p_{data}, p) = \mathbb{E}_{((\mathbf{x},y)\sim p_{data})}[L(f(\mathbf{x}, \theta), y)]$$

- If we sample N points from p_{data}, $H(p_{data}, p)$ is approximated by $H(p_{emp}, p)$ in a statistical sense.
- Similarly, the gradient $\nabla_\theta H(p_{data}, p)$ can be approximated by $\nabla_\theta H(p_{emp}, p)$.
- We can see the use of a mini-batch of size m (with a single pass on the data) in the SGD as a statistical approximation of the gradient $\nabla_\theta H(p_{data}, p)$: hence we minimize the generalization error.
- For very big data, we may not even use all data points in training.

SGD AND LEARNING RATE

- The learning rate η of the SGD algorithm cannot be kept fixed at each iteration. In fact, the algorithm would not converge in this case, due to the noisy evaluations fo the gradient. Hence, η_k must depend on the iteration
- A sufficient condition for convergence of SGD is :

$$\sum_{k=1}^{\infty} \eta_k = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \eta_k^2 < \infty$$

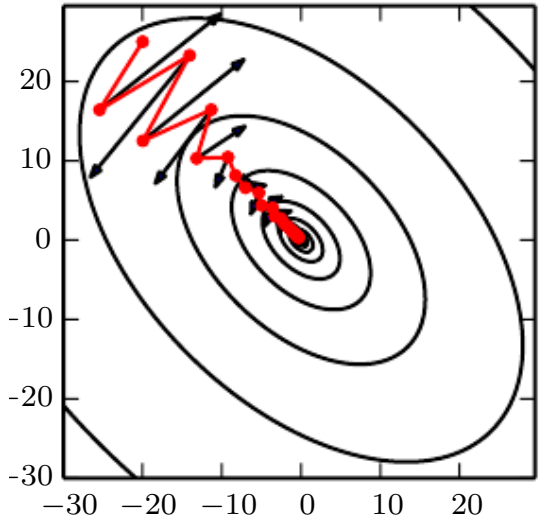
- Typically, one sets $\eta_k = (1 - \frac{k}{\tau})\eta_0 + \frac{k}{\tau}\eta_\tau$, where τ is equal to the number of iterations for few epochs of the algorithm (epoch = one iteration over the dataset). For deep models (i.e. very complex), $\tau \approx 100$. Furthermore, $\eta_\tau \approx 0.01\eta_0$.
- The choice of η_0 is delicate. Too large and the algorithm may diverge, too small and it may take forever. Strategy: monitor the first 50-100 iterations (plot the estimated cost function, using the same minibatch used for gradient), and find an "optimal" η_0. Then choose a larger one, avoiding instabilities.

MOMENTUM AND NESTEROV-MOMENTUM

- Introduces memory in the gradient, by averaging the current value with previous ones:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$



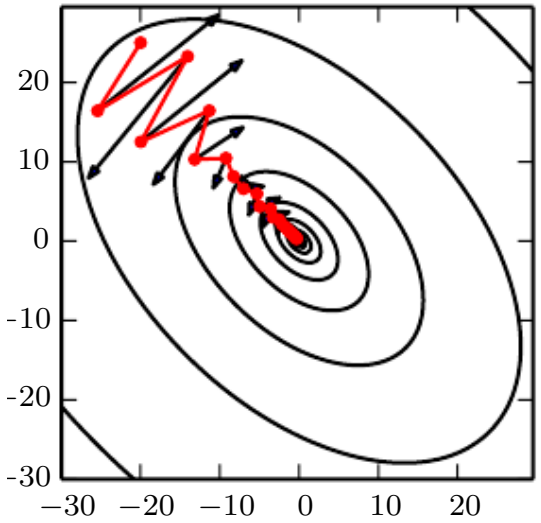
  - If all gradients are aligned, momentum accelerates by multiplying by $1/(1-\alpha)$. Generally, $\alpha = 0.5$ or 0.9 or 0.99.
  - We can see the algorithm as a physical system subject to a Newtown forces and evolving in continuous time. The cost function is taken as a potential and modulated by η, and the momentum term corresponds to viscous friction (proportional to velocity). Initial velocity is equal to the initial gradient.

MOMENTUM AND NESTEROV-MOMENTUM

- Introduces memory in the gradient, by averaging the current value with previous ones:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$



Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

MOMENTUM AND NESTEROV-MOMENTUM

- Nesterov momentum evaluates the gradient in an intermediate point. It can be shown that it modifies standard GD convergence rate to $O(1/k^2)$

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right]$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
    Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

INITIALIZATION OF THE OPTIMISATION ALGORITHM

- The initial point of the optimization algorithm is crucial for convergence, especially in high dimensions. If we are in the basin of attraction of a good minimum/ area with good cost function, then the SGD will work fine. Otherwise not.
- We can randomise initial conditions and try the optimisation several times.
- If we have some extra information about the solution, better incorporate it: As a general rule, always use asymmetric initial conditions (especially if the model has symmetries: see neural networks).
- Sample from a (zero mean) Gaussian or an uniform. Range is important: if too large may result in instabilities. If too small, may introduce too little variation.
- Heuristics depend on the model to learn.

ADAPTIVE LEARNING RATE: ADAGRAD

- Introduce a different rate for each parameter. Modify them to take the curvature of the search space into account.
- AdaGrad: scales the learning rate inversely proportional to the square root of the sum of all their historical squared values. Good for convex problems.

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$
    Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

ADAPTIVE LEARNING RATE: RMSPROP

- RMSProp: performs better in non-convex setting than AdaGrad, by changing gradient accumulation into an exponentially weighted moving average.

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

ADAPTIVE LEARNING RATE: RMSPROP

- RMSProp can be also combined with Nesterov Momentum. There is an extra hyperparameter controlling the length scale of moving average.

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

ADAPTIVE LEARNING RATE: ADAM

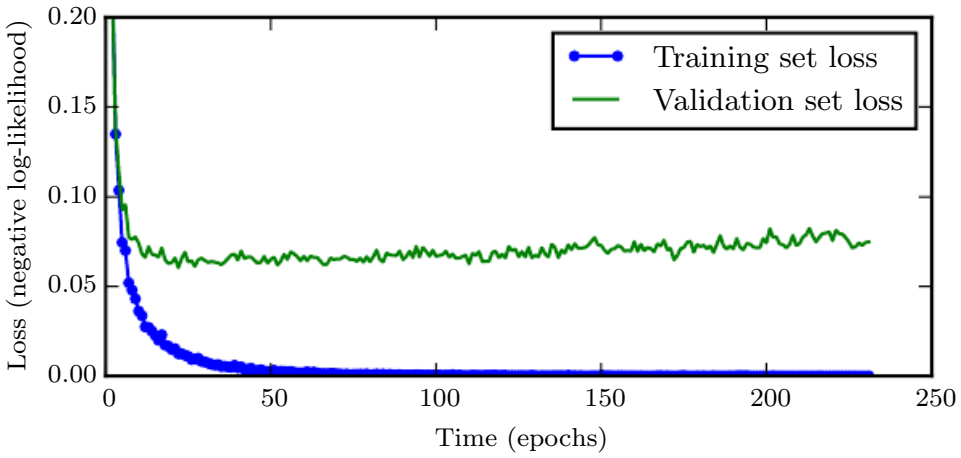
- Adam integrates RMSProp with momentum. Introduces a second order correction. Quite stable w.r.t. hyperparameters.

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

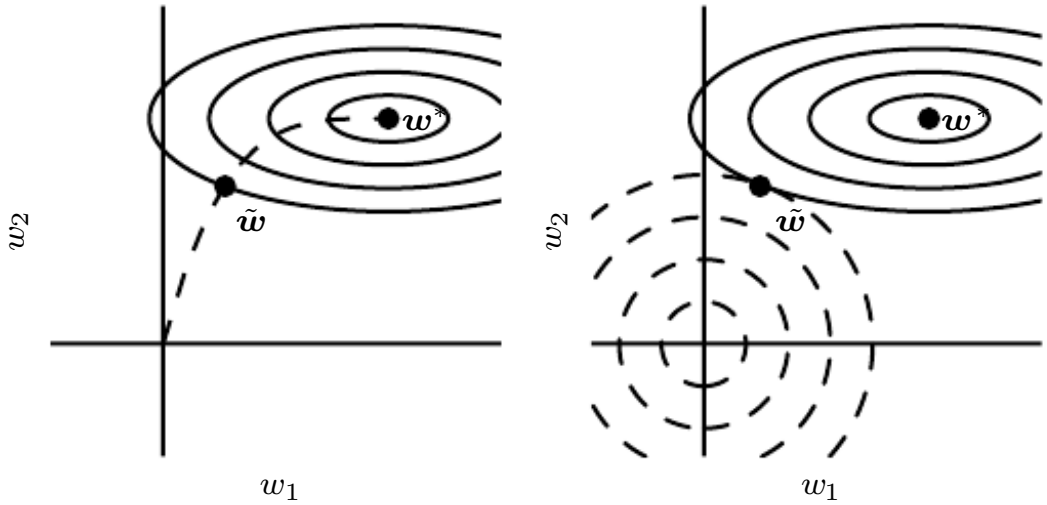
DIGRESSION: REGULARIZATION BY EARLY STOPPING

- One of the most used regularization strategies, particularly for deep models, is early stopping. Idea is that, for a complex model (prone to overfitting), the best generalization is not found at an optimum. A better solution can be found along the trajectory going to it.
- One uses a validation dataset to check during optimization how validation error decreases, and stops at a minimum of the validation curve. Time is thus treated as a hyperparameter.



DIGRESSION: REGULARIZATION BY EARLY STOPPING

- One can show that early stopping, for linear models, has a similar effect as L_2 regularization.
- Early stopping is a very cheap form of regularization.



DIGRESSION: REGULARIZATION BY EARLY STOPPING

Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Outline

GRADIENT DESCENT

- Notation. $\mathbf{x}_k$ denotes the sequence of points of the descent. $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$. The update is in the direction $\mathbf{p}_k$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \eta_k \mathbf{p}_k$$

- In gradient descent, at a point $\mathbf{x}$, take a step towards $-\nabla f(\mathbf{x})$, hence in the update rule becomes we set $\mathbf{p}_k = -\mathbf{g}_k$.
- In the simplest case, $\eta_k = \eta$. If η is not small enough, we can step over the minimum. If η is very small this usually not happens, but convergence is very slow.
- For a quadratic function, we have that $\mathbf{p}_k = -A\mathbf{x}_k + \mathbf{b}$.

GRADIENT DESCENT WITH LINE SEARCH

- One possibility to improve gradient descent is to take the best step possible, i.e. set η_k to a value minimising the function $f(\mathbf{x}_k + \lambda \mathbf{p_k})$ along the line with direction $\mathbf{p}_k$.
- The minimum is obtained by solving for λ the equation

$$\nabla f(\mathbf{x}_k + \lambda \mathbf{p_k})^T \mathbf{p}_k = \mathbf{g}_{k+1}^T \mathbf{p}_k = 0$$

  and setting η_k to this solution.
- for a quadratic function, we have that the best learning rate is given by

$$\eta_k = \frac{(\mathbf{b} - A\mathbf{x}_k)^T \mathbf{p}_k}{\mathbf{p}^T A \mathbf{p}}$$

CONJUGATE GRADIENTS

- Consider a quadratic minimisation problem. If the matrix A would be diagonal, we could solve separately n different 1-dimensional optimisation problems.
- We can change coordinates by an orthogonal matrix P that diagonalises the matrix A. By letting $\mathbf{x} = P\mathbf{y}$, we can rewrite the function $f(\mathbf{x})$ as

$$f(\mathbf{y}) = \frac{1}{2}\mathbf{y}^T P^T A P\mathbf{y} - \mathbf{B}^T P\mathbf{y} + c$$

- The columns of P are called conjugate vectors and satisfy $\mathbf{p}_i^T A\mathbf{p}_j = 0$ and $\mathbf{p}_i^T A\mathbf{p}_i > 0$. They are linearly independent and are very good directions to follows in a descent method.

CONJUGATE GRADIENTS

- To construct conjugate vectors, we can use the Gram-Schmidt orthogonalisation procedure: if $\mathbf{v}$ is linearly independent of $\mathbf{p}_1, \ldots, \mathbf{p}_k$, then

$$\mathbf{p}_{k+1} = \mathbf{v} - \sum_{j=1}^{k} \frac{\mathbf{p}_j^T A \mathbf{v}}{\mathbf{p}_j^T A \mathbf{p}_j} \mathbf{p}_j$$

- We can start from a basis and construct the conjugate vectors $\mathbf{p}_1, \ldots, \mathbf{p}_n$.
- In the conjugate vectors algorithm, we take step $k+1$ along $\mathbf{p}_{k+1}$. The best η_k, according to line search, is

$$\eta_k = \frac{-\mathbf{p}_k^T \mathbf{g}_k}{\mathbf{p}_k^T A \mathbf{p}_k}$$

- It holds that $\nabla f(\mathbf{x}_{k+1})^T \mathbf{p}_i = 0$ for all $i = 1, \ldots, k$ (Lunenberg expanding subspace theorem).

CONJUGATE GRADIENTS

- The conjugate gradients method constructs $\mathbf{p}_k$'s on the fly. Works well also for non-quadratic problems. For quadratic problems converges in at most n steps.
- A good choice for a linearly independent vector $\mathbf{v}$ at step $k+1$ to construct $\mathbf{p}_{k+1}$ is thus $\nabla f(\mathbf{x}_{k+1})$.
- In this case, after some algebra, we can compute:

$$\eta_{k+1} = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{p}_{k+1}^T A \mathbf{p}_{k+1}}$$

$$\mathbf{p}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{p}_k$$

with

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k} \quad \text{or} \quad \beta_k = \frac{\mathbf{g}_{k+1}^T (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \mathbf{g}_k}$$

known as the Fletcher-Reeves or Polak-Ribière (preferrable for non-quadratic problems) formulae .

Outline

NEWTON-RAPSON METHOD

- As an alternative optimisation for small n, we can use the Newton-Rapson method, which has better convergence properties than gradient descent.
- By Taylor expansion

$$f(\mathbf{x}+\Delta) \approx f(\mathbf{x}) + \Delta^T \nabla f(\mathbf{x}) + \frac{1}{2}\Delta^T H_f(\mathbf{x})\Delta$$

  where $\mathbf{H}_f$ is the Hessian of $f(\mathbf{x})$.
- Differentiating w.r.t. Δ, the minimum of the r.h.s. is when $\nabla f(\mathbf{x}) = -H_f(\mathbf{x})\Delta$, hence for $\Delta = -\mathbf{H}_f^{-1}(\mathbf{x})\nabla f(\mathbf{x})$
- Thus we obtain the update rule:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \mathbf{H}_f^{-1}(\mathbf{x}_k)\nabla f(\mathbf{x}_k)$$

  with $0 < \eta < 1$ to improve convergence.