Optimization and Learning

Torkel A. Haufmann
SINTEF Digital

Outline

1. Recap
2. SVM
3. Neural networks
4. First-order methods in learning
5. Learning in optimization
6. Final recap

Recap

- Three weeks ago: Convexity, duality, KKT conditions.
- Two weeks ago:
  - Gradient descent: Backtracking and $d_k = -\nabla f(x_k)$.
  - Newton's method: Backtracking/unitary step size, $d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$.
  - Subgradient method: Replace $\nabla f(x_k)$ by *a* subgradient. Not descent!
  - Stochastic gradient method: Step in direction of gradient with respect to one (or more!) data points.
  - Extend to constrained optimization by adding a projection step.
  - Newton's method for non-linear equations.
  - Interior-point methods.

Recap

Interior-point method:

1. Choose x_0, w_0 and find an initial ν. Select μ.
2. Repeat until "close enough":
   1. Get the search direction from Newton's method on the KKT conditions.
   2. Compute the step lengths.
   3. Update the variables.
3. Update μ - go to step 2 if the problem isn't solved

Recall that from the barrier problem we essentially get perturbed KKT conditions: The complementary slackness condition replaces the right hand side with μ, and then we let $\mu \to 0$.
A general implementation: `ipopt`.

SVM, again

Recall that we want to solve the SVM problem:

$$\text{minimize} \quad \frac{1}{2}\|\beta\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i(\beta^T \mathbf{x}_i - \alpha))$$

We rephrased it as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\beta\|^2 + \frac{1}{n}\sum_{i=1}^{n} w_i \\ \text{subject to} \quad & y_i(\beta^T x_i - \alpha) \geq 1 - w_i, \\ & w_i \geq 0. \end{aligned}$$

This is a convex quadratic optimization problem.

Let's try it in binder

SVM with constrained support

We are interested in *sparse classification*, which in the SVM case is a question of solving

$$\begin{aligned} \text{minimize} \quad & \tfrac{1}{2}\|\beta\|^2 + \tfrac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i \beta^T \mathbf{x}_i) \\ \text{subject to} \quad & \|\beta\|_0 \leq k, \end{aligned}$$

Question

Why did we just get rid of the α?

Question

How do we enforce sparsity?

We'll follow Bertsimas, Dunn and look at an *outer-approximation algorithm* (chapter 6). Define $S_k^p = \{s \in \{0,1\}^p, e^T s \leq k\}$. We'll often use $S = \text{diag}(s)$.

SVM with constrained support

Following the book, we rephrase the problem in this way:

$$\min_{s \in S_k^p} \left(\min_{\beta \in \mathbb{R}^p} \frac{1}{2\gamma} \|S\beta\|^2 + \sum_{i=1}^{n} \max(0, 1 - y_i (S\beta)^T \mathbf{x}_i) \right).$$

Question

Why does this work?

Now, let

$$c(s) = \min_{\beta \in \mathbb{R}^p} \frac{1}{2\gamma} \|S\beta\|^2 + \sum_{i=1}^{n} \max(0, 1 - y_i (S\beta)^T \mathbf{x}_i).$$

SVM with constrained support

Theorem

$$c(s) = \max_{\alpha \in \mathbb{R}^n} \left(-\sum_{i=1}^{n} y_i \alpha_i - \frac{\gamma}{2} \sum_{j=1}^{p} s_j \alpha^T X_j X_j^T \alpha \right) \quad \textit{s.t. } y_i \alpha_i \in [-1, 0],$$

where the constraint holds for all i.

Observe that c is convex for $s \in \{0, 1\}^p$.

Exercise

Prove this directly by introducing $z_i = (SB)^T x_i$ and finding the dual of the (non-differentiable) Lagrangian. Hint:

$$\max(0, 1 - y_i z_i) - \alpha_i z_i = \begin{cases} -\alpha_i y_i & \text{if } \alpha_i y_i \in [-1, 0], \\ -\infty & \text{otherwise.} \end{cases}$$

SVM with constrained support

It can also be shown that the gradient of c with respect to s is given by

$$\frac{\partial c(s)}{\partial s_j} = -\frac{\gamma}{2}\alpha^*(s)^T X_j X_j^T \alpha^*(s),$$

where $\alpha^*(s)$ is the maximizer of $c(s)$.

We know how to compute $c(s)$ and $\nabla c(s)$ for any s, using `cvxopt`.

Exercise

Try to implement the computation of $c(s)$! A tip: $\beta^* = -\gamma X^T \alpha^*$.

SVM with constrained support

Consider the outer-approximation algorithm (Bertsimas, Dunn, Chapter 6):

- We assume s_0 is given.
- Repeat for $k = 0, 1, 2, \ldots$ until done:
  - Take s_{k+1}, η_{k+1} to be the solutions of

$$\begin{aligned} \text{minimize} \quad & \eta \\ \text{subject to} \quad & \eta \geq c(s_i) + \nabla c(s_i)^T (s - s_i), \quad (i \in [k]) \\ & s \in S_k^p. \end{aligned}$$

We won't get into solving the mixed-integer program today, but the book gives the following result.

Theorem

The above algorithm terminates in a finite number of steps $\hat{k}$, and $s_{\hat{k}}$ is then an optimal solution to the original problem.

Neural networks

Training is hard!

- Generalizability,
- batch sizes,
- dropout,
- learning rate,
- cross-validation,
- regularization
- and more we won't talk about today.

The process of training is to adjust the weights to minimize some measure of *wrongness* over the input data:

$$\text{minimize}_\theta l(X, \theta) + \text{regularization}.$$

Convex loss functions are commonly used!

Neural networks

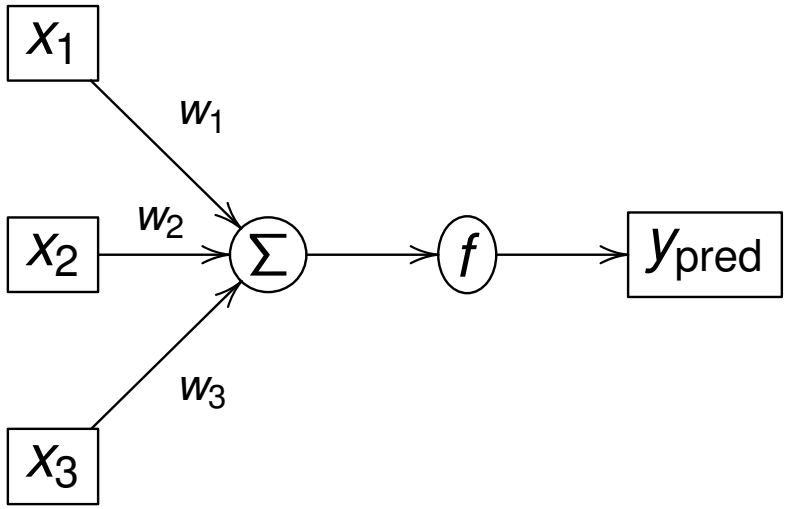
- We know several algorithms for convex minimization.
- Only missing piece of the puzzle: Gradient of the loss with respect to θ.

This is where *backpropagation* comes in. All about the *chain rule*:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Neural networks

Let's consider a single-layer neural network. We wish to train this network to give better outputs. For illustration:



f is an activation function, for example the *sigmoid*:

$$f(x) = \frac{1}{1 + e^{-x}}.$$

Neural networks

We wish to minimize some loss function L, which is a function of x, y and the weights w. Then by the chain rule:

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w_i},$$

where $z = \sum_{i=1}^{3} w_i x_i$. Now,

$$\begin{aligned} \frac{\partial L}{\partial z} &= f(z)(1 - f(z)), \\ \frac{\partial z}{\partial w_i} &= x_i, \end{aligned} \tag{1}$$

and so we know how to compute the gradient of L with respect to w_i.

This can be repeated for a much more complicated network, following the computation graph. The approach is known as *automatic differentiation*.

Automatic differentiation

- Every (sort of!) implemented function *does* have an explicit description!
- Since the values of the input traverse the computation graph we can also trace the operations that happen ("dual numbers").
- Using operator overloading we can "smuggle" this evaluation into existing programs.

Many tough questions:

- Reverse mode vs forward mode.
- Branching.
- Looping.
- And so on.

Standard for packages like PyTorch – see `pytorch.autograd`. Uses sub-gradient with minimum norm if no gradient is available (and the function is convex).

Let's try it in binder

Neural networks

We used `torch.optim.SGD`. Let's check out the module:

```
>>> import torch
>>> dir(torch.optim)
['ASGD', 'Adadelta', 'Adagrad', 'Adam', 'AdamW', 'Adamax', 'LBFGS', 'NAdam',
'Optimizer', 'RAdam', 'RMSprop', 'Rprop', 'SGD', 'SparseAdam', '__builtins__',
'__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__path__', '__spec__', '_functional', '_multi_tensor', 'lr_scheduler', 'swa_utils']
```

- A lot of `*ada*` stuff.
- LBFGS is an acquaintance of ours.
- SGD: Stochastic gradient descent.

Let's look at the theory for a little while longer.

Definition

f is μ-strongly convex if for any $x, y \in \mathbb{R}^p$:

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \|x - y\|^2.$$

Nesterov acceleration

If you look up the documentation of `torch.optim.SGD`, there are many mentions of Nesterov. The *Nesterov accelerated gradient* is a descent method that is optimal in a certain sense:

$$f(x_k) - f^* = O((1 - \sqrt{\mu/L})^k)$$

for μ-strongly convex problems.

Definition

For a given function f, an *estimate sequence* is a pair of sequences $\Phi_k : \mathbb{R}^p \to \mathbb{R}$, λ_k that satisfies:

$$\begin{aligned} \lambda_k &\to 0 \text{ as } k \to 0, \\ \Phi_k(x) &\leq (1 - \lambda_k) f(x) + \lambda_k \Phi_0(x) \text{ for all } x \in \mathbb{R}^p, k \geq 0. \end{aligned}$$

This definition may seem odd, but...

Nesterov acceleration

Let x^* be a minimal point of f, i.e. $f(x^*) = f^*$.

Lemma (see Nesterov, 1998)

If we have

$$f(x_k) \leq \min_{x \in \mathbb{R}^p} \Phi_k(x),$$

then

$$f(x_k) - f^* \leq \lambda_k(\Phi_0(x^*) - f^*) \to 0.$$

Proof.

$$\begin{aligned} f(x_k) &\leq \min_{x \in \mathbb{R}^p} \Phi_k(x) \\ &\leq \min_{x \in \mathbb{R}^p} (1 - \lambda_k) f(x) + \lambda_k \Phi_0(x) \\ &\leq (1 - \lambda_k) f(x^*) + \lambda_k \Phi_0(x^*), \end{aligned}$$

and the result follows from rearranging terms. □

Nesterov acceleration

Question

What does last slide's lemma really tell us?

Implied strategy:

- Form an estimate sequence that works for the lemma.
- Conclude on convergence rate directly from the sequence λ_k.

Lemma

Let f be μ-strongly convex, $\Phi_0(x)$ arbitrary, y_k an arbitrary sequence in $\mathbb{R}^p$, α_k a sequence of scalars in $(0,1)$ that is not summable and $\lambda_0 = 1$. Then:

$$\begin{aligned} \lambda_{k+1} &= (1-\alpha_k)\lambda_k, \\ \Phi_{k+1} &= (1-\alpha_k)\Phi_k(x) + \alpha_k \left(f(y_k) + \nabla f(y_k)^T (x - y_k) + \tfrac{\mu}{2}\|x - y_k\|^2 \right), \end{aligned}$$

is an estimate sequence.

Nesterov acceleration

Exercise

Prove the lemma on the previous slide. Hint:

$$\alpha_k f(x) = (1 - (1 - \alpha_k)\lambda_k)f(x) - (1 - \alpha_k)(1 - \lambda_k)f(x).$$

From this Nesterov derives a scheme using a quadratic function as $\Phi_0(x)$, and at each iteration choosing x_{k+1} so the estimate sequence condition is satisfied (proof too long). Skipping ahead we get:

Theorem

If L is the Lipschitz constant of the derivative of f the sequence generated by the scheme satisfies

$$f(x_k) - f^* \leq L \min\left((1 - \sqrt{\mu/L})^k, 4/(k+2)^2\right) \|x_0 - x^*\|^2.$$

It can be shown that this is the optimal bound for a reasonable class of function.

Nesterov acceleration

After some manipulation, Nesterov arrives at the following simplified scheme: Choose $y_0 = x_0 \in \mathbb{R}^p$. Then for each k, set

$$\begin{aligned} x_{k+1} &= y_k - \tfrac{1}{L}\nabla f(y_k), \\ y_{k+1} &= x_{k+1} + \tfrac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}(x_{k+1} - x_k) \end{aligned} \tag{2}$$

Exercise

(Optional!) Try to implement this for a quadratic function. Note that for $f(x) = x^T Ax$ over a ball of radius R, the Lipschitz constant is $2R\rho(A)$. What about μ?

Nesterov acceleration, implemented

Pytorch cites Sutskever et al.. Fix step size to s and recall the gradient step:

$$x_{k+1} = x_k - s\nabla f(x_k).$$

Momentum method: Instead use *two* sequences, v_k and x_k:

$$\begin{aligned} v_{k+1} &= \mu v_k - s\nabla f(x_k), \\ x_{k+1} &= x_k + v_{k+1}. \end{aligned} \tag{3}$$

Here $\mu \in [0, 1]$ is the *momentum coefficient*.
Nesterov acceleration (See Sutskever et al., derived from a slightly different scheme than we discussed above):

$$\begin{aligned} v_{k+1} &= \mu v_k - s\nabla f(x_k + \mu v_k), \\ x_{k+1} &= x_k + v_{k+1}. \end{aligned} \tag{4}$$

Nesterov acceleration, implemented

$$\begin{aligned} v_{k+1} &= \mu v_k - s\nabla f(x_k + \mu v_k), \\ x_{k+1} &= x_k + v_{k+1}. \end{aligned} \tag{5}$$

- Gradient descent is historyless.
- The v sequence carries information about the past for us.
- We can interpret $x_k + \mu v_k$ as a "partial update", getting a better estimate for the gradient.

Sutskever et al. show that for convex quadratic objectives, Nesterov acceleration is more stable than momentum methods (both of which are better than gradient methods).

Exercise

Extend your gradient descent implementation with Nesterov acceleration.

Note: In practice both μ, s and f may vary.

Adam

Another major method: *Adam*, based on adaptive estimates of moments. As described in the paper:

- Set e.g. $\alpha = 0.001$, $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$.
- Assume f, x_0 are given.
- Set $m_0 = 0$, $v_0 = 0$.
- For $k = 0, 1, 2, \ldots$:
  1. $g_{k+1} = \nabla f_{k+1}(x_k)$.
  2. $m_{k+1} = \beta_1 m_k + (1 - \beta_1) g_{k+1}$.
  3. $v_{k+1} = \beta_2 v_k + (1 - \beta_2)(g_{k+1} \odot g_{k+1})$.
  4. $\hat{m}_{k+1} = m_{k+1}/(1 - \beta_1^k)$.
  5. $\hat{v}_{k+1} = v_{k+1}/(1 - \beta_2^k)$.
  6. $x_{k+1} = x_k - \alpha \hat{m}_{k+1}/(\sqrt{\hat{v}_{k+1}} + \epsilon)$.

The subscript on f_k is meant to indicate that we can e.g. evaluate the loss only on random samples from the training data.

Convergence?

Theorem (Shi et al., 2021)

For "randomly shuffled Adam" with $\beta_1 = 0$, convergence is assured if β_2 is large enough.

" Large enough" depends on characteristics of f.
A few reflections:

- We proved (almost!) convergence of Nesterov accelerated gradients for certain problems.
- But the method often works for other problems as well!
- Adam (and friends) are known to possibly diverge with some parameter settings.
- But they are huge successes in practice!
- Everything changes fast in this space: Models, strategies, learning rate schedules...
- Practice often seems ahead of the theory.

Learning in optimization

As taught, optimization often revolves around:

- Exact methods
- Convergence proofs
- Complexity results

In a number of cases, “optimal” methods are presented, for a given class of problem (we did this today!).

As used, practice is often ahead of theory:

- Poorly justified convergence.
- Application outside of proven goodness.
- Heuristic decisions.
- Many practitioners want *improvement*, not *optimality*.

Learning in optimization

The simple approach is to train a neural net to guess the solution to an optimization problem. This is (in my opinion!) the *incorrect* approach.

Optimization methods abound with choices:

- Initial values.
- Search directions.
- Formulation choices.
- Decompositions.
- And more...

In the literature we can find:

- Lower bound approximations.
- Branching policies in branch and bound.
- Algorithm choice/configuration.
- Learned objectives.
  - Through automatic differentiation a neural network objective is differentiable! We can use e.g. interior-point methods.
- And more...

One key point: Organizing available choices.

Learning in optimization

Today we'll look briefly at an example from the literature (Donti et al., 2021). A known tough problem: AC optimal power flow.

Power is represented as complex numbers, and we have a graph with (using the paper's notation) b buses and a matrix $W \in \mathbb{C}^{b \times b}$ describing admittances. The input variables $p_d \in \mathbb{R}^b$, $q_d \in \mathbb{R}^b$ represent real/reactive power demands in the graph. The problem to solve is:

$$\begin{aligned} \text{minimize} \quad & p_g^T A p_g + b^T p_g \\ \text{subject to} \quad & p_g^{\min} \leq p_g \leq p_g^{\max}, \\ & q_g^{\min} \leq q_g \leq q_g^{\max}, \\ & v^{\min} \leq |v| \leq v^{\max}, \\ & (p_g - p_d) + (q_g - q_d)i = \operatorname{diag}(v)\overline{W}\overline{v}, \end{aligned} \tag{6}$$

where the decision variables are $p_g \in \mathbb{R}^b, q_g \in \mathbb{R}^b, v \in \mathbb{C}^b$. This is *not a nice* problem.

Learning in optimization

The approach: Train a neural network N_θ to be used in the following algorithm:

1. Compute a partial solution $z = N_\theta(x)$.
2. Complete the solution to a set $y = [z^T \phi_x(z)^T]^T$ satisfying equality constraints.
3. Using gradient descent, move to a point $\hat{y}$ also satisfying inequality constraints ("correction").

Key point: How to backpropagate the correction and completion when computing $\nabla_\theta l(\hat{y})$? Answer: Read the paper...
Let's take a look at their results on
`https://github.com/locuslab/DC3`.

What we've learned

- Convexity is very useful!
- Duality and optimality conditions are core tools.
- Various forms of descent methods, including for training.
- Interior-point methods.
- Machine learning needs optimization.
- The future of ML-enhanced optimization is bright (?)!

Exercises

Exercise

Train an SVM by starting with the dual problem (Derive it if you forgot it! Or check a previous lecture).

Exercise

(Easy!) If $X \in \mathbb{R}^{n \times p}$, $s \in \{0,1\}^p$ and $S = \text{diag}(s)$, verify that

$$XS(XS)^T = XSX^T = \sum_{j \in [p]} s_j X_j X_j^T,$$

where X_j is the j-th column of X.

Exercise

Implement and test Adam.

References

- Bertsimas, D. and Dunn, J., 2019. Machine Learning Under a Modern Optimization Lens.
- Boyd, S. and Vandenberghe, L., 2004. Convex Optimization.
- Donti et al., 2021. DC3: A learning method for optimization with hard constraints.
- Nesterov, Y. 1998. Introductory Lectures on Convex Programming.
- Nocedal, J. and Wright, S. J., 2006. Numerical Optimization - Second Edition.
- Sutskever et al. 2013. On the importance of initialization and momentum in deep learning.