L101: Optimization fundamentals
Previous lecture

Logistic regression parameter learning:

\[ w^* = \arg \min_w \sum_{(x,y) \in D} -y \log \sigma(w \cdot \phi(x)) - (1 - y) \log(1 - \sigma(w \cdot \phi(x))) \]

Supervised machine learning algorithms typically involve optimizing a loss over the training data:

\[ w^* = \arg \min_w L(w; \mathcal{D}), w \in \mathbb{R}^k \]

This is an instance of **numerical optimization**, i.e. optimize the value of a function with respect to some parameters.

A scientific field of its own; this lecture just gives some useful pointers
Types of optimization problems

Continuous: \[ x^* = \arg \min_x f(x), x \in \mathbb{R}^k \]

Discrete: \[ x^* = \arg \min_x L(x), x \in \mathbb{Z}^k \]

Sounds rare in NLP?
Inference in classification/structured prediction: a label is either applied or not

Constraints: \[ x^* = \arg \min_x L(x), c(x) \geq 0 \]

Examples: SVM parameter training, enforcing constraints on the output graph
Convexity

For sets:
\[ \forall x, y \in S : ax + (1 - a)y \in S, \ a \in [0, 1] \]

For functions:
If \( f \) concave, \(-f\) is convex
For sets the relation is more complicated

\[ f(tx_1 + (1 - t)x_2) \leq tf(x_1) + (1 - t)f(x_2), \ t \in [0, 1] \]
Taylor’s theorem

For a function $f$ that is continuously differentiable, there is $t$ such that:

$$f(x + p) = f(x) + \nabla f(x + tp)p, \quad t \in (0, 1)$$

If twice differentiable:

$$\nabla f(x + p) = \nabla f(x) + \int_0^1 \nabla^2 f(x + tp)p \, dt$$

$$f(x + p) = f(x) + \nabla f(x)p + \frac{1}{2} p \nabla^2 f(x + tp)p, \quad t \in (0, 1)$$

- Given value and gradients, can approximate function elsewhere
- Higher degree gradient, better approximation
Types of optimization algorithms

- Line search
- Trust region
- Gradient free
- Constrained optimization
Line search

At the current solution $x_k$, pick a descent direction first $p_k$, then find a stepsize $\alpha$:

$$\min_{\alpha > 0} f(x_k + \alpha p_k)$$

and calculate the next solution:

$$x_{k+1} = x_k + \alpha_k p_k$$

General definition of direction:

$$p_k = -B_k^{-1} \nabla f(x_k)$$

Gradient descent:

$$B_k = I$$

Newton method (assuming $f$ twice differentiable and $B_k$ invertible):

$$B_k = \nabla^2 f(x_k)$$
Gradient descent (for supervised MLE training)

**Input:** training examples $\mathcal{D} = \{(x^1, y^1), \ldots (x^M, y^M)\}$, learning rate $\alpha$

Initialize weights $w$

```
while $\nabla_w \text{NLL}(w; \mathcal{D}) \neq 0$ do
    Update $w = w - \alpha \nabla_w \text{NLL}(w; \mathcal{D})$
end while
```

To make it stochastic, just look at one training example in each iteration and go over each of them. Why is this a good idea?

What can go wrong?
Gradient descent

Wrong step size:

Line search converges to the minimizer when the iterates follow the Wolfe conditions on sufficient decrease and curvature (Zoutendijk’s theorem)

Back tracking: start with a large stepsize and reduce it to get sufficient decrease

Stochastic: noisy gradients (a single datapoint might be misleading)

https://srdas.github.io/DLBook/GradientDescentTechniques.html
Second order methods

Using the Hessian (line search Newton’s method):

\[ x_{k+1} = x_k - \alpha_k \nabla^2 f(x_k)^{-1} \nabla f(x_k) \]

Expensive to compute. Can we approximate?

Yes, based on the first order gradients:

\[ B_{k+1} = \frac{\nabla f(x_{k+1}) - \nabla f(x_k)}{x_{k+1} - x_k} \]

BFGS calculates \( B_{k+1}^{-1} \) directly without moving too far from \( B_k^{-1} \)
What is a good optimization algorithm?

Fast convergence:
- Few iterations
  - Stochastic gradient descent will have more than standard gradient descent
- Cheap iterations; what makes them expensive?
  - Function evaluations for backtracking with line search (this is the reason for researching adaptive learning rates)
  - (approximate) second order gradients

Memory requirements? Storing second order gradients requires $|w|^2$. One of the key variants of BFGS is L(imited memory)-BFGS.

One can learn the updates: Learning to learn gradient descent by gradient descent
Trust region

Taylor’s theorem:

\[ f(x + p) = f(x) + \nabla f(x)p + \frac{1}{2} p \nabla^2 f(x + tp)p, \quad t \in (0, 1) \]

Assuming an approximation \( m \) to the function \( f \) we are minimizing:

\[ m_k(p) = f(x_k) + \nabla f(x_k)p + \frac{1}{2} p \nabla^2 f(x_k + p)p \]

Given a radius \( \Delta \) (max stepsize, trust region), choose a direction \( p \) such that:

\[ \min_p m_k(p), \quad p \leq \Delta_k \]

Measuring trust:

\[ \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} \]
Trust region

Worth considering with relatively few dimensions.

Recent success in reinforcement learning
Gradient free

What if we don’t have/want gradients?
- Function is a black box to us, can only test values
- Gradients too expensive/complicated to calculate, e.g.: hyperparameter optimization

Two large families:
- Model-based (similar to trust region but without gradients for the approximation model)
- Sampling solutions according to some heuristic
  - Nelder-Mead
  - Evolutionary/genetic algorithms, particle swarm optimization
Bayesian Optimization

- Model approximation based on Gaussian Process regression
- Acquisition function tells us where to sample next

Frazier (2018)
Constraints

Reminder: \( x^* = \arg\min_x f(x), c(x) \geq 0 \)

Minimizing the Lagrangian function converts it to unconstrained optimization (for equality constraints, for inequalities it is slightly more involved):

\[
L(x, \lambda) = f(x) + \lambda c(x)
\]

Example:

\[
f(x_1, x_2) = x_1 + x_2
\]

\[
c(x_1, x_2) = x_1^2 + x_2^2 - 2 = 0
\]

\[
\nabla_x L(x, \lambda) = 0 \Rightarrow \nabla f(x^*) = \lambda^* \nabla c(x^*)
\]
Overfitting

A function (separating hyperplane)

The training data

Regularization

We want to optimize the function/fit the data but not too much:

$$w^* = \arg \min_w L(w; D) + \lambda R(w)$$

Some options for the regularizer:

- L2: $\Sigma w^2$
- L1 (Lasso): $\Sigma |w|$
- Ridge: L1 + L2
- L-infinity: max($w$)
Words of caution

Sometimes we are saved from overfitting by not optimizing well enough.

There is often a discrepancy between loss and evaluation objective; often the latter are not differentiable (e.g. BLEU scores).

Check your objectives if it tells you the right thing: optimizing less aggressively and getting better generalization is OK, having to optimize badly to get results is not.

Construct toy problems: if you have a good initial set of weights, does your optimizing the objective leave them unchanged?
Harder cases

- Non-convex
- Non-smooth

Saddle points: zero gradient is a first order necessary condition, not sufficient

https://en.wikipedia.org/wiki/Saddle_point
Bibliography

- On integer (linear) programming in NLP: https://ilpinference.github.io/eacl2017/
- Francisco Orabona’s blog: https://parameterfree.com
- Dan Klein’s Lagrange Multipliers without Permanent Scarring