OPTIMIZATION METHODS IN DEEP LEARNING

Based on Deep Learning, chapter 8 by Ian Goodfellow, Yoshua Bengio and Aaron Courville

Presented By Nadav Bhonker

Tutorial outline

- Optimization vs Learning
  - Surrogate loss function and early stopping
  - Batch and minibatch algorithms
- Challenges in Neural Network Optimization
- Basic Algorithms
  - SGD
  - Momentum
  - AdaGrad
- Regularization
  - L2, L1
  - Data augmentation

Optimization vs Learning

- Optimization
  - “The selection of a best element (with regard to some criterion) from some set of available alternatives.”
  - Or “Maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.”
- Machine Learning
  - Optimization of a function of our data…?
  - No!
Goal of Machine Learning

- Minimize Risk
  \[ J^*(\theta) = \mathbb{E}_{(x,y) \sim p_{data}} L(f(x; \theta), y) \]
  - \( x \) - data
  - \( y \) - label
  - \( p_{data} \) - true distribution of the \((x, y)\)
  - \( L \) - a loss function

  “Reduce the chance of error on a prediction task regarding a distribution \( P \)”

- But \( p_{data} \) is unknown!

Empirical Risk Minimization

\[ J(\theta) = \mathbb{E}_{(x,y) \sim \hat{p}_{data}} L(f(x; \theta), y) = \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \]

- Minimize the empirical risk function
- Now we’re back to optimization

- Models with high capacity tend to overfit
- Some loss functions don’t have useful gradients (i.e. 0-1 loss)

  - One common solution…

Surrogate Loss Function

- Replace the loss function we care about with a “nicer” function
  - E.g. replace 0-1 with log-likelihood

- Optimization is performed until we meet some convergence criterion
  - E.g. minimum of a validation loss

Batch algorithms

- The objective function of ML is usually a sum over all the training examples.
  \[ \theta_{ML} = \arg \max_{\theta} \sum_{i=1}^{N} \log p_{model}(x^{(i)}, y^{(i)}; \theta) \]

- Computing the loss over the entire dataset:
  1. Computationally expensive!
  2. Small variance reduction. According to CLT: \( \sigma_{\text{eff}} \approx \frac{\sigma}{\sqrt{m}} \)
  3. May use redundant data

- The solution – minibatch algorithms
**Minibatch algorithms**

- Estimate the loss from a subset of samples at each step

\[ \theta_{ML} = \arg \max_\theta \sum_1^m \log P_{model}(x^{(i)}, y^{(i)}; \theta) \]

- Larger batch → more accurate estimate of the gradient (though less than linear returns)

- Small batches act as regularizers due to the noise they add

**Minibatch - Random Sampling**

- Random batches!
  - Many datasets are most naturally arranged in a way where successive examples are highly correlated.
  - Sampling from such a dataset would introduce a heavy bias that would impair the generalization (catastrophic forgetting)

**CHALLENGES IN NEURAL NETWORK OPTIMIZATION**

**Reminder**

- Hessian matrix:
  \[ H(f)(x)_L = \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \]

- Taylor series expansion:
  \[ f(x) = f(x^{(0)}) + (x - x^{(0)})^T g + \frac{1}{2} (x - x^{(0)})^T H(x - x^{(0)}) \]

  \( g \) - gradient
Ill-Conditioning

- The Taylor expansion for an update with learning rate $\epsilon$:
  \[ f(x^{(0)} - \epsilon g) \approx f(x^{(0)}) + \epsilon g^T g + \frac{1}{2} \epsilon^2 g^T H g \]
- We recognize here three terms:
  1. $f(x^{(0)})$ - the original value of the function
  2. $\epsilon g^T g$ - the slope of the function
  3. $\frac{1}{2} \epsilon^2 g^T H g$ - the correction we must apply to account for the curvature of the function

Ill-Conditioning – cont.

- In many cases, the gradient norm does not shrink significantly throughout learning, but the $\frac{1}{2} \epsilon^2 g^T H g$ term grows by more than an order of magnitude.
- Learning becomes very slow despite the strong gradient.
- The learning rate must be shrunk to compensate for even stronger curvature.

Local minima

- Neural networks have many local minima due to 2 reasons:
  1. Model identifiability – Given $\exists \theta_1, \theta_2$ s.t. $f_{\theta_1}(x) = f_{\theta_2}(x) \forall x$
     Different combinations of weights may result in the same classification function
  2. Local minima that are worse than global minima – Not clear if this is a real problem or not
Plateaus, Saddle Points and Other Flat Regions

- In high dimensional space random functions tend to have few local minima but many saddle points\(^1\)

\[ \text{Result for shallow linear autoencoder}\]

- Reminder - Hessian matrix eigenvalues (at zero gradient):
  1. All positive – local minimum.
  2. All negative – local maximum.


Plateaus, Saddle Points and Other Flat Regions – cont

- Due to this property, (unmodified) Newton method is unsuitable for deep networks

- Empirically, gradient descent methods manage to escape saddle points\(^1\)


Stochastic Gradient Descent

**Algorithm 8.1** Stochastic gradient descent (SGD) update at training iteration \(k\)

**Require:** Learning rate \(\epsilon_k\).

**Require:** Initial parameter \(\theta\)

**while** stopping criterion not met \(\text{do}\)

  Sample a minibatch of \(m\) examples from the training set \(\{x^{(1)}, \ldots, x^{(m)}\}\) with corresponding targets \(y^{(i)}\).

  Compute gradient estimate: \(\hat{g} \leftarrow \frac{1}{m} \sum_i \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})\)

  Apply update: \(\theta \leftarrow \theta - \epsilon \hat{g}\)

**end while**

Note that \(\epsilon_k\) can change each iteration.

**How to choose?**

“This is more of an art than a science, and most guidance on this subject should be regarded with some skepticism.”
**Learning Rate Consideration**

- Considering a linear decrease:
  \[ \epsilon_k = (1 - \alpha) \epsilon_0 + \alpha \epsilon_{k-1}, \quad \alpha = \frac{k}{r} \]
- Rules of thumb:
  - Set \( r \) to make a few hundred passed through the training data
  - \( \epsilon_r = \epsilon_0 \times 0.01 \)
  - \( \epsilon_r \) trial & error
- Decrease learning rate when some loss criteria is met

**Momentum**

- Modify the parameter update rule:
  \[ v \leftarrow \alpha v - \epsilon \nabla \theta \left( \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \right) \]
  \[ \theta \leftarrow \theta + v. \]
- Introducing a variable \( v \) (velocity) — it is the direction and speed at which the parameters move through parameter space.
- The velocity is set to an exponentially decaying average of the negative gradient.

**Momentum: Physical Analogy**

- Continuous gradient descent:
  \[ \frac{d\theta}{dt} = -\epsilon \nabla \theta (\theta) \]
- Consider the Newtonian equation for a point mass \( m \) moving in a viscous medium with friction coefficient \( \mu \) under the influence of a conservative force field with potential energy \( E(\theta) \):
  \[ m \frac{d^2 \theta}{dt^2} + \mu \frac{d\theta}{dt} = -\nabla \theta (\theta) \]

**Momentum: Physical Analogy – cont.**

- Discretizing the equation we get
  \[ \frac{\theta_{t+\Delta t} + \theta_{t-\Delta t}}{2} - \theta_{t} = -\frac{(\Delta t)^2}{m + \mu \Delta t} \nabla \theta (\theta) + \frac{m}{m + \mu \Delta t} (\theta_{t} - \theta_{t-\Delta t}) \]
- Which is equivalent to the previous formula:
  \[ v \leftarrow \alpha v - \epsilon \nabla \theta \left( \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \right) \]
  \[ \theta \leftarrow \theta + v. \]
Algorithms with Adaptive Learning Rates

- The learning rate is the hyper-parameter with the largest impact

- The cost may be more sensitive in some directions than in others—a learning rate per parameter may be even better

AdaGrad

- Per parameter learning rate scaled inversely proportional to the square root of historical squared values

```
Algorithm 8.4 The AdaGrad algorithm
Require: Global learning rate ε
Require: Initial parameter θ
Require: Small constant β, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable r ← 0
while stopping criterion not met do
    Sample a minibatch of m examples from the training set \{(x^{(i)}, \ldots, x^{(m)}\} with corresponding targets y^{(i)}.
    Compute gradient: g ← ∇θ \sum_{i} \left( f(x^{(i)}, \theta) - y^{(i)} \right)
    Accumulate squared gradient: r ← r + g \circ g
    Compute update: Δθ ← \frac{-1}{\sqrt{\epsilon + \text{RMSProp}}} \circ g
    Apply update: θ ← θ + Δθ
end while
```
Which Optimizer to use?

Definition and Goal

- “Any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

- Goals:
  - Encode prior knowledge (i.e. spatial structure – conv nets)
  - Promote a simpler model
  - Trade bias for variance

Parameter Norm Penalties

- Limit a norm of the parameters:
  \[ \tilde{f}(\theta; X, y) = f(\theta; X, y) + a\Omega(\theta) \]

- Two common approaches:
  1. \( L^2 \) regularisation: \( \Omega(\theta) = \frac{1}{2}\|w\|_2^2 \)
     a.k.a – weight decay, ridge regression, Tikhonov regularization
     w - parameters to be regularized
  2. \( L^1 \) regularisation: \( \Omega(\theta) = \|w\| = \sum_i |w_i| \)
**$L^2$ Regularization**

- Why weight decay?
  \[
  \tilde{J}(\theta; X, y) = J(\theta; X, y) + \frac{\alpha}{2} w^T w
  \]
  \[
  \nabla_w \tilde{J}(\theta; X, y) = \alpha w + \nabla_w J(\theta; X, y)
  \]

  When updating $w$:
  \[
  w \leftarrow w - \epsilon (\alpha w + \nabla_w J(\theta; X, y))
  \]
  Or:
  \[
  w \leftarrow (1 - \epsilon \alpha) w - \epsilon \nabla_w J(\theta; X, y)
  \]
  Weight decay

**$L^2$ Regularization - Intuition**

- Consider the linear regression task:
  \[
  \tilde{J}(\theta; X, y) = (Xw - y)^T (Xw - y) + \frac{\alpha}{2} w^T w
  \]

  This changes the exact solution from:
  \[
  w = (X^T X)^{-1} X^T y
  \]
  To:
  \[
  w = (X^T X + \alpha I)^{-1} X^T y
  \]

  The additional $\alpha$ on the diagonal corresponds to increase the variance of each feature of $X$.

**$L^1$ Regularization**

- The modified gradient
  \[
  \tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha |w|
  \]
  \[
  \nabla_w \tilde{J}(\theta; X, y) = \text{sign}(w) + \nabla_w J(\theta; X, y)
  \]

  - $L^1$ regularization tends to make solutions **sparse**

**Dataset Augmentation**

- Best way to generalize – lots of data!
- But data is limited – so we can fake data
- Common augmentations for object recognition: translation, scale, rotation, random crops...
- Noise injection is also common and effective