Deep Learning

AlejandroMllo

This document serves as a very brief summary of the topics covered in each chapter of the book Deep Learning [1].

Disclaimer: This document is completely extracted from [1], the author does not attribute any ownership over the material.

8 Optimization for Training Deep Models

- The focus is finding the parameters $\boldsymbol{\theta}$ of a NN that significantly reduce a cost function $J(\boldsymbol{\theta})$.
- Most ML scenarios care about some performance measure P, that is defined w.r.t. the test set and may also be intractable. Then it reduces a different cos function $J(\boldsymbol{\theta})$ in the hope that doing so will improve P.
- Typically, the cost function can be written as an average over the training set, such as

  $$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y),$$

  where L is the per-example loss function, $f(\boldsymbol{x};\boldsymbol{\theta})$ is the predicted output when the input is $\boldsymbol{x}$, and $\hat{p}_{data}$ is the empirical distribution.
  It is preferred to minimize the corresponding objective function where the expectation is taken across the $data-generating\ distribution\ p_{data}$ rather than just over the finite training set:

  $$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y),$$

- When the true distribution $p_{data}(\boldsymbol{x}, y)$ is not known, and only is available a training set of samples, then you have a ML problem. To convert a ML problem back into an optimization problem is to minimize the expected loss on the training set, Empirical risk:

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

- Optimization algorithms for ML typically compute each update to the parameters based on an expected value of the cost function estimated using only a subset of the terms of the full cost function.
- Optimization algorithms that use the entire training set are called batch or deterministic gradient methods; those that use a single example at a time are sometimes called stochastic or online methods. Minibatch or minibatch stochastic methods use more than one but fewer than all the training examples; this are commonly used in deep learning.
- Generalization error is often best for a batch size of 1 (this might require a small learning rate).
- Minibatch stochastic gradient descent follows the gradient of the true generalization error as long as no examples are repeated.
- To obtain an unbiased estimator of the exact gradient of the generalization error, sample a minibatch of examples $\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(i)}$ with corresponding targets $y^{(i)}$ from the data-generating distribution p_{data}, then computing the gradient of the loss w.r.t. the parameters for that minibatch:

  $$\hat{\boldsymbol{g}} = \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)}).$$

  Updating $\boldsymbol{\theta}$ in the direction of $\hat{\boldsymbol{g}}$ performs SGD on the generalization error.
- When optimizing a convex function, a good solution is reached if a critical point of any kind is found.
- Local minima can be problematic if they have high cost in comparison to the global minimum. A test can rule out local minima as the problem is plotting the norm of the gradient over time. If the norm of the gradient does not shrink to insignificant size, the problem is neither local minima nor any other kind of critical point.

- Gradient clipping heuristic: when the traditional gradient descent algorithm proposes making a very large step (it is on a cliff), this heuristic intervenes to reduce the step size, making it less likely to go outside the region where the gradient indicates the direction of approximately steepest descent.
- Some optimization problems might be avoided if there exists a region of space connected reasonably directly to a solution by a path that local descent can follow, and if it is possible to initialize learning within that well-behaved region.
- A sufficient condition to guarantee convergence of SGD is that $\sum_{k=1}^{\infty} \epsilon_k = \infty$ *and* $\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$, where ϵ_k is the learning rate at iteration k. In practice, it is common to decay the learning rate linearly until iteration τ: $\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$.
- Usually τ may be set to the number of iterations required to make a few hundred passes through the training set, and ϵ_τ should be set to roughly 1 percent the value of ϵ_0.
- For a large enough dataset, SGD may converge to within some fixed tolerance of its final test set error before it has processed the entire training set.
- To study the convergence rate of an optimization algorithm it is common to measure the excess error $J(\boldsymbol{\theta}) - \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, which is the amount by which the current cost function exceeds the minimum possible cost.
- Momentum: this optimization method accelerates learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients. It accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
  - It introduces a variable $\boldsymbol{v}$ that gives the direction and speed at which the parameters move through parameter space. It is set to an exponentially decaying average of the negative gradient.
  - A hyperparameter $\alpha \in [0, 1)$ determines how quickly the contributions of previous gradients exponentially decay.
$$\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$$
  - See algorithm 8.2 for an example of SGD with momentum.
- Nesterov Momentum: similar to momentum, but here the gradient is evaluated after the current velocity is applied. See algorithm 8.3.
- Training algorithms for deep learning models are usually iteratively and thus require the user to specify some initial point from which to begin the iterations. Some initial points may be beneficial from the viewpoint of optimization but detrimental from the viewpoint of generalization.
- The initial parameters need to "break symmetry" between different units. If two hidden units with the same activation function are connected to the same inputs, then these units must have different initial parameters.
- Typically, the biases for each unit are set to heuristically chosen constants, and the weights are initialized randomly.
- AdaGrad: this algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient. The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. See algorithm 8.4.
- RMSProp: this algorithm modifies AdaGrad to perform better in the nonconvex setting by changing the gradient accumulation function into an exponentially weighted moving average. It uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl, as if it were an instance of AdaGrad initialized within that bowl. See algorithm 8.5.
- Adam: see algorithm 8.7. Here, first, momentum is incorporated directly as an estimate of the first-order moment (with exponential weighting) of the gradient. Second, it includes bias corrections to the estimates of both the first order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin.
- Newton's method is an optimization scheme based on using a second-order Taylor series expansion to approximate $J(\boldsymbol{\theta})$ near some point $\boldsymbol{\theta}_0$, ignoring derivatives of higher order. See algorithm 8.8.
- Conjugate gradients is a method to efficiently avoid the calculation of the inverse Hessian by iteratively descending conjugate directions (that is, directions that will not undo progress made in the previous direction). See algorithm 8.9.

- The nonlinear conjugate gradients algorithm includes occasional resets where the method of conjugate gradients is restarted with line search along the unaltered gradient. This is used when it is not known if the objective is quadratic.
- The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm attempts to bring some of the advantages of Newton's method without the computational burden. There is also a limited memory version.
- Batch normalization provides an elegant way of reparametrizing almost any deep network. Let $\boldsymbol{H}$ be a minibatch of activations of the layer to normalize, arranged as a design matrix, with the activations for each example appearing in a row of the matrix. To normalize $\boldsymbol{H}$, replace it with $\boldsymbol{H}' = \frac{\boldsymbol{H}-\boldsymbol{\mu}}{\boldsymbol{\sigma}}$, where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are vectors containing the mean and standard deviation of each unit, respectively.
- If $f(\boldsymbol{x})$ is minimized w.r.t. a single variable x_i, then minimized w.r.t. another variable x_j, and so on, repeatedly cycling through all variables, at some moment it will arrive at a (local) minimum. This is known as coordinate descent. Block coordinate descent refers to minimizing w.r.t. a subset of the variables simultaneously.
- Polyak averaging consists of averaging several points in the trajectory through parameter space visited by an optimization algorithm. If t operations of gradient descent visit points $\boldsymbol{\theta}^{(1)}, \cdots, \boldsymbol{\theta}^{(t)}$, then the output of the algorithm is $\hat{\boldsymbol{\theta}}^{(t)} = \frac{1}{t}\sum_i \boldsymbol{\theta}^{(i)}$.
- Pretraining: training a simple model on simple tasks before confronting the challenge of training the desired model to perform the desired task.
- Greedy supervised pretraining: pretraining algorithms that break supervised learning problems into other simpler supervised learning problems.
- Many improvements in the optimization of deep models have come from designing the models to be easier to optimize.
- Modern NN have been designed so that their local gradient information corresponds reasonably well to moving toward a distant solution.
- Continuation methods are a family of strategies that can make optimization easier by choosing initial points to ensure that local optimization spends most of its time in well-behaved regions of space. The idea behind them is to construct a series of objective functions over the same parameters. To minimize a cost function $J(\boldsymbol{\theta})$, new cost function $J^{(0)}, \cdots, J^{(n)}$ are constructed (they are designed to be increasingly difficult to minimize).
- Curriculum learning: planning a learning process to begin by learning simple concepts and progress to learning more complex concepts that depend on these simpler concepts.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.