On the Importance of Consistency in Training Deep Neural Networks

Parimala K, Sri Krishna Priya D

November 14, 2017

Overview

Difficulties of training DNNs

First order vs second order

- Performing gradient descent iterations is the most costly and time-consuming among the various operations in training deep neural networks.
- First Order :
  - Most current training algorithms adopt first-order methods, i.e., modifications of the steepest descent method to conduct gradient decent process.
  - The convergence rate of first-order methods is inherently sub-optimal for large scale non-linear regression problems, such as the training of deep neural networks.

First order vs second order contd.

- Second Order :
  - Second-order optimization algorithms take into account the local geometry of the underlying problem.
  - During each iteration, second-order methods are computationally more expensive than first-order ones.
  - Direct computation of the Hessian matrix and its inverse, is computationally intractable, and using approximations of the Hessian is still more costly compared to the simple steepest descent method.
  - The Gauss-Newton algorithm can find proper step sizes for each direction and converge very fast; especially, if the error function has a quadratic surface, it can converge directly in the first iteration. But this improvement only happens when the quadratic approximation of error function is reasonable. Otherwise, the Gauss-Newton algorithm would be mostly divergent.

On Using Second order

- Due to these difficulties, in most works deep neural networks are trained with first-order methods.
- So why second order now?
  - They have superlinear convergence rate, while first-order methods have linear convergence rate.
  - They have better invariance properties: the optimal step sizes are determined with the use of curvature information and as a result are usually close to 1.
- Training these networks has been found technically difficult for decades because of a combination of three challenges. All three can be explained as inconsistency in different layers of the networks.
  1. Speed Inconsistency
  2. Scale Inconsistency
  3. Vanishing gradient Problem

Superlinear convergence

- $\lim_{k \to \infty} \frac{|x_{k+1} - L|}{|x_k - L|} = \mu$
- The above sequence converges to L with rate of convergence μ.
- If the sequence converges, and
  - μ_k varies from step to step with $\mu_k \to 0$ for $k \to \infty$, then the sequence is said to converge superlinearly.
  - μ_k varies from step to step with $\mu_k \to 1$ for $k \to \infty$, then the sequence is said to converge sublinearly.

Speed Consistency

Motivation - Law of the minimum

- In a layered structure such as the neural network, if certain layers are more difficult to train than others, these layers will impede the training of the whole network.
- This is usually the case with first-order gradient descent methods, as there is no guarantee that a universal learning rate fits all layers.
- Therefore, the speed of training the slowest layer(s) becomes the bottleneck of the whole training process.
- This phenomenon is reflected in the well known law of the minimum, which motivates us to design an approach that trains each individual layer at the consistent, full speed.

Optimization problem

- Minimizing the final regression loss is equivalent to minimizing a set of gradients known as regression residuals: r $= \frac{\partial z}{\partial y}$ at the network output y
- Setup:
  - Let z be the final output of the network.
  - The parameters in the i-th layer : W_i, input : x_i, and output : $y_i = W_i x_i$
  - The residual in layer i is defined to be the gradient $r_i(W_i) = \frac{\partial z}{\partial y_i}$
  - Given a displacement P, the new residual : $r_i(W_i + P) \approx Px_i + r_i(W_i)$.
- Optimization problem for each layer : $\min_P \frac{1}{2}||PX_i + R_i||^2$
- The solution of above equation satisfies the normal equations: $PX_iX_i^T = -R_iX_i^T$
- Proof:
  - $\frac{\partial}{\partial P}(\frac{1}{2}||PX_i + R_i||^2) = 0 \Rightarrow (PX_i + R_i)\frac{\partial(PX_i)}{\partial P} = 0 \Rightarrow (PX_i + R_i)X_i^T = 0$

Trust Region method

- Trust-region methods define a region around the current iterate within which they trust the model to be an adequate representation of the objective function, and then choose the step to be the approximate minimizer of the model in this region.
- In effect, they choose the direction and length of the step simultaneously.
- If a step is not acceptable, they reduce the size of the region and find a new minimizer. In general, the direction of the step changes whenever the size of the trust region is altered.
- Setup:
  - Let x_0 be the starting point.
  - Let $x_1, x_2, x_3, \ldots, x_k, x_{k+1}, \ldots$ be the sequence of iterated generated by optimization algorithm
  - Model $m^{(k)}$ that approximates the objective function in a finite region Δ near $x^{(k)}$. Δ where the model is a good approximation of f, is called the trust-region.

Trust Region method contd

- The step length Δ is chosen to move towards the approximate minimum of the model in this region.
- Update it at each iteration using heuristics.
- $m^{(k)}$ is obtained by a Taylor series expansion of f around $x^{(k)}$

$$m^{(k)} = f(x^{(k)}) + \Delta f(x^{(k)}).p + \frac{1}{2}p^T Hp$$

where H is the Hessian matrix.

- Finding the step length to take during the iteration :

$$\min_{||p|| \leq \Delta} \quad f(x^{(k)}) + \Delta f(x^{(k)}).p + \frac{1}{2}p^T Hp$$

- Iteration Step : $x^{(k+1)} = x^{(k)} + p_k$

Trust Region method contd

- The heuristic to update the size of the trust-region depends on $\rho_k = \frac{f(x^{(k)}) - f(x^{(k)} + p_k)}{m^{(k)}(0) - m^{(k)}(p_k)}$, the ration of the expected change in f to the predicted change.
- If there is a good agreement between predicted and actual values ($\rho_k \approx 1$), then ∇ is increased.
- If the agreement is poor (ρ_k is small), then ∇ is decreased.
- If ρ_k is smaller than a threshold value ($\approx 10^{-4}$), the step is rejected and the value of $x^{(k)}$ is retained but ∇ is decreased accordingly.

Levenberg Marquardt Algorithm

- The function to be minimized : $f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x)$
- Where r_j's are smooth functions from R^n to R and referred as residuals.
- r_j's can be represented as a Residual Vector r : $R^n \rightarrow R^m$ defined by,

$$r(x) = (r_1(x), r_2(x), \ldots r_m(x))$$
$$\Rightarrow f(x) = \frac{1}{2}||r(x)||_2^2$$

- The model function :

$$m^{(k)} = f(x^{(k)}) + \nabla f(x^{(k)}).p + \frac{1}{2}p^T Hp$$
$$= \frac{1}{2}\ ||r(x_k)||_2^2\ +\ p^T\ J(x_k)^T\ r(x_k)\ + \frac{1}{2}\ p^T\ J(x_k)^T\ J(x)\ p$$

Trust Region method contd

- To find the step size to take during the iteration:

$$min_{||p|| \leq \triangle} \frac{1}{2}||r(x)||_2^2 + p^T J(x_k)^T r(x_k) + \frac{1}{2} p^T J(x_k)^T J(x) p$$

- Theorem : The vector $p*$ is a global solution of the trust region problem

$$\min_{p \in R^n} \frac{1}{2} \ ||r(x)||_2^2 \ + \ p^T \ J(x_k)^T \ r(x_k) \ + \ \frac{1}{2} \ p^T \ J(x_k)^T \ J(x) p$$

$$s.t ||p|| \leq \Delta$$

iff p^* is feasible and there is a scalar $\lambda \geq 0$such that following equations satisfied.

1. $(H + \lambda I)p^* = -g$
2. $\lambda(\triangle - |p|) = 0$
3. $(H + \lambda I)$ is positive semi definite

Trust Region method contd

- If the solution lies strictly inside the trust region. We must have

$$\lambda = 0$$
$$p^* = -H^{-1}g$$

H is positive semi definite matrix

- If the solution lies on the boundary of the trust region in such cases we must have

$$||p^*|| = \triangle \quad \& \quad \lambda > 0$$

H is positive definite matrix for such cases we define

$$p(\lambda) = -(B + \lambda I)^{-1}g$$

Solution to the optimization problem

- When $X_i X_i^T$ is well-conditioned and positive definite, P is the descent direction that minimizes the regression loss.
- When $X_i X_i^T$ is ill-conditioned, regularizations are used to stabilize the problem.
- The trust region method is a simple modification that seeks the solution of the following equation: $P(X_i X_i^T + \lambda I) = -R_i X_i^T$
- RHS of the above equation is the negative gradient of the parameters in layer i: $\frac{\partial z}{\partial W_i} = \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial W_i} = r_i x_i^T \Rightarrow P = -\frac{\partial z}{\partial W_i}(X_i X_i^T + \lambda I)^{-1}$
- The above optimization solution is a layer-wise application of the Levenberg-Marquardt Algorithm.
- In this solution, each layer has its own optimization problem that is simple and intuitive: given input x_i and residual r_i in layer i, how can we adjust the parameters in this layer to minimize r_i? This can be done with gradient descent using iterations.

Well conditioned Matrix

- A matrix A is said to be well conditioned if it has a low condition number.
- Condition number $= \frac{\sigma_1}{\sigma_n}$, where σ_1 is the highest singular value of the matrix and σ_n is the lowest singular value of the matrix.
- The condition number measures how much the output value of a function can change for small change in the input argument.
- Consider AX = B, A small change in B brings about a small change in solution X.
- Sometimes matrix A has zero or close to zero eigenvalues, due to which the matrix becomes singular; adding a small value λ to the diagonal of the matrix A makes it stable. Bigger value of λ implies more stability, but you are further away from true solution. Smaller value of λ implies less stability, closer is your inverted matrix to true inverted matrix but it might become unstable.

Comparison to previous work

- For CNNs, the method focuses on a simplified version by ignoring pixel wise correlation among feature maps.
- We treat each feature map as a hidden unit and calculate the correlation across different feature maps. Inverse correction is then applied to the convolution kernel gradients as shown in this equation : $P = -\frac{\partial z}{\partial W_i}(X_i X_i^T + \lambda I)^{-1}$
- Previous attempts of designing second-order algorithms did not make explicit use of the layer residuals. Therefore, the Hessian matrix in each middle layer depends on Hessian matrices in deeper layers.

Computational Bottlenecks in this method

1. The matrix multiplication of $X_i X_i^T$: in convolutional networks, X_i is usually a highly redundant wide matrix.
   - By performing a uniform subsampling we can significantly accelerate the matrix multiplications with no noticeable loss of accuracy.
   - The computational cost for this is negligible compared to the whole training process.

Computational Bottlenecks in this method Contd.

2. The matrix inversions: this can be performed reasonably fast only at the small scale of hundreds of entries. For larger layers that contain thousands of hidden units, matrix multiplications and inversions are extremely slow.
    - Use a simple stochastic preconditioning strategy to speed up the decorrelation process
    - In each training iteration, the hidden units are stochastically divided into chunks (of hundreds) so that matrix multiplications and inversions can be calculated quickly.
    - Observation : This stochastic division and preconditioning technique is an effective way of conducting acceleration.

Scale Consistency

Motivation

- A neural network is an ordered set of transforms into different spaces.
- During the transform, there is no constraint guaranteeing that the input signal x_i in layer i will be on a meaningful scale with residual r_i.
- The first layer of the network, inputs may be in the range of [-1, 1] and the residuals in the range of [-0.1, 0.1], which is well-posed.
- In a deeper layer, the inputs may be in the range of [-0.001, 0.001] and the residuals in the range of [-10, 10].
- The optimization for such deeper layers becomes much harder, affecting any method, including gradient descent and l^2 optimization.

Solution

- Use normalization techniques.
- The widely-adopted batch normalization algorithm was designed specifically for first-order methods. A globally-optimal learning rate is almost impossible to find with first-order methods in a deep layered structure.
- The batch normalization algorithm introduces an extra pair of parameters, which scales and shifts the normalized results to alleviate this problem

Using Second order methods

- The following normal equations figure out the scaling and shifting automatically.

$$P(X_i X_i^T + \lambda I) = -R_i X_i^T$$
$$P = -\frac{\partial z}{\partial W_i}(X_i X_i^T + \lambda I)^{-1}$$

- Speed is consistently optimal in each layer as long as the inputs are approximately on the same scale.
- The rescaling in batch normalization potentially leads to model instability if the scaling factor is larger than 1.
- Hence, we adopt a minimal normalization strategy and divide the inputs by their smoothed root mean square:

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

Using Second order methods

- RMS can be estimated from minibatches and smoothed using moving average. This RMS normalization is also necessary for second-order methods so that the regularization in following equation takes consistent effects in each layer.

$$P(X_i X_i^T + \lambda I) = -R_i X_i^T$$

- Eg. $\lambda = 0.1$ is appropriate if the inputs are in the range of [-1, 1], but very likely ineffective if the inputs are in [-0.001, 0.001]. In the latter case, the displacement P in the equations at the top of this page is likely to explode.

Vanishing gradient with Operator Analysis

Motivation

- Even after normalizing the inputs into well-defined ranges we encounter failures and noticeable worse performance when the network goes beyond 10 - 20 layers.
- Therefore, there is a need for further mathematical analysis.

Vanishing gradient with Operator analysis

- Let τ be the operator that is 'the best linear approximate' of one or several layers of the forward transform.
- In terms of differential geometry, τ is the differential of the forward transform.
- In its well-known natural state, τ (expansive or non-contractive) has singular value(s) larger than 1.
- Using the notation of operator theory, τ is an operator that maps Hilbert space H_1 to H_2.
- Hilbert adjoint $\tau*$ is used in the back propagation process and maps H_2 back to H_1.
- According to operator theory, $\tau*$ has the same characteristics as τ, and therefore it should also have singular value(s) larger than 1.

Vanishing gradient problem

- Even though our operator τ, therefore $\tau*$, is non contractive, gradient propagation shows a significant energy decay when we move to the shallower layers.
- Empirically, this contractive effect is known as the vanishing gradient problem in the training of deep neural networks.
- Let the forward transform be: $z = f(y) = f(\tau(x))$

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

- Therefore, if $\frac{\partial z}{\partial y}$ falls into the kernel of $\tau*$ (null space of $\tau*$), then the gradient propagation stops.
- Hence, the following constraint has to be enforced.

$$\frac{\partial z}{\partial y} \notin ker(\tau*) = im(\tau)^{\perp} \Rightarrow \tau* = \tau^{T} \text{(In the real space)}$$

- Constraints : $\frac{\partial z}{\partial y} \neq 0, \quad \frac{\partial z}{\partial y} \in im(\tau)$

Experiments

Injecting noise

- The optimization process stochastically perturbs the gradient in order to avoid that the gradients fall into the kernel space, where P now becomes:

$$P = -(\frac{\partial z}{\partial W_i} + \epsilon)((X_i X_i^T + \lambda I)^{-1})$$

- Here, small random noise is a valuable ingredient that adds a factor of random exploration in the training.
- Observations :
  - the ReLU operator is not the best option for gradient propagation.
  - Hence used modulus unit ModU as activation function instead of ReLU.
  - In the real number space the modulus unit computes the absolute value of its input:

$$ModU(x) = \begin{cases} x, & \text{if } x >= 0 \\ -x, & \text{if } x < 0 \end{cases}$$

  - In order to create a larger image space or equivalently a smaller kernel space, activation function is changed from ReLU to ModU

The End