Large Scale Data Analysis Using Deep Learning

Optimization for Training Deep Models - 2

U Kang
Seoul National University

In This Lecture

- Optimization strategies and meta-algorithms
  - Batch Normalization
  - Coordinate Descent
  - Polyak Averaging
  - Supervised Pretraining
  - Designing Models to Aid Optimization
  - Continuation Methods and Curriculum Learning

Batch Normalization

- A method of adaptive reparameterization, motivated by the difficulty of training very deep models
- Gradient tells how to update each parameter in a layer, under the assumption that the other layers do not change
- In practice, we update all of the layers simultaneously; thus, unexpected results can happen because many functions composed together are changed simultaneously, using updates that were computed under the assumption that the other functions remain constant

Batch Normalization

- E.g., a deep neural network with only one unit per layer and no activation function: $\hat{y} = xw_1w_2w_3 \ldots w_l$. Hidden layer $h_i = h_{i-1}w_i$
- Suppose we compute gradient $g = \nabla_w \hat{y}$. What would happen for a gradient descent update $w \leftarrow w - \epsilon g$
- First-order Tayler approximation of $\hat{y}$ predicts that $\hat{y}$ will decrease by $\epsilon g^T g$; thus, if we want to decrease $\hat{y}$ by .1, then we can set ϵ to $\frac{.1}{g^T g}$
- However, the actual update will include second-order and third-order effects, on up to effects of order l, since all parameters are changed simultaneously
  - $\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2) \ldots (w_l - \epsilon g_l)$
- An example of a second-order term is $\epsilon^2 g_1 g_2 \prod_{i=3}^{l} w_i$. This term might be very small or very large based on w_i
- This makes it very hard to choose an appropriate learning rate, because the effects of an update to a parameter depends on other parameters

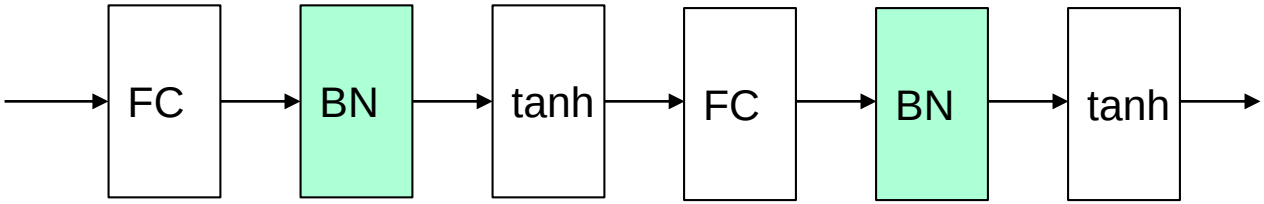
Batch Normalization

- Batch normalization helps in
  - Reducing the problem of coordinating updates across many layers.
  - Mitigate 'exploding gradient' or 'vanishing gradient' problem
  - Improves gradient flow through the network
  - Allows higher learning rate
  - Reduces the strong dependence on initialization
  - Acts as a regularization method: processing of x_i depends on other x_js; reduces the need for dropout

- Batch normalization can be applied to any input or hidden layer

Batch Normalization

- Consider a batch of activations at a layer. Batch normalization applies the following transformation for each dimension
  - $\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$
  - This makes $x^{(k)}$ unit Gaussian if $x^{(k)}$ is Gaussian
  - Note that this is a differentiable function of $x^{(k)}$
- Batch Normalization (BN) is inserted after Fully Connected layer, and before nonlinearity



But, we might give too small values to tanh if we stop at this point

Batch Normalization

- Scaling and shifting the unit Gaussian
  - Normalize: $\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$
  - Then, allow the network to change the range if needed:
    $y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$
  - Important point: the network can learn
    $\gamma^{(k)} = \sqrt{Var[x^{(k)}]}, \beta^{(k)} = E[x^{(k)}]$
    to recover the identity mapping
  - $\gamma^{(k)}$ and $\beta^{(k)}$ are parameters learned through back propagation

Batch Normalization: Full Version

- Input: a mini-batch B=$\{x_1, \dots, x_m\}$
- Output: batch-normalized input: $\{y_i = BN_{\gamma,\beta}(x_i)\}$
  - $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$
  - ${\sigma_B}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$
  - $\widehat{x_i} = \frac{x_i - \mu_B}{\sqrt{{\sigma_B}^2 + \epsilon}}$
  - $y_i = \gamma\widehat{x_i} + \beta$

- At test time, μ and σ are replaced by running averages that were collected during training time

Coordinate Descent

- Coordinate descent
  - Arrive at a (local) minimum by minimizing f(x) w.r.t. a single variable x_i, then another variable x_j and so on, repeatedly cycling through all variables
- Block coordinate descent: minimizing with respect to a subset of the variables simultaneously
- Coordinate descent is useful when the different variables can be clearly separated into groups that play relatively isolated roles, or optimization wrt one group of variables is significantly more efficient than optimization wrt all of the variables

Coordinate Descent

- Example: sparse coding
  - $J(H, W) = \sum_{i,j} |H_{i,j}| + \sum_{i,j} (X - W^T H)^2_{i,j}$
  - Goal: find W and H so that a weight matrix W (dictionary) can decode H (code representation) to reconstruct the training set X
  - The function J is not convex. However, minimizing J with respect to either one of W and H is convex
  - Thus, block coordinate descent gives us an optimization strategy (alternate between optimizing W with H fixed, then H with W fixed)

Polyak Averaging

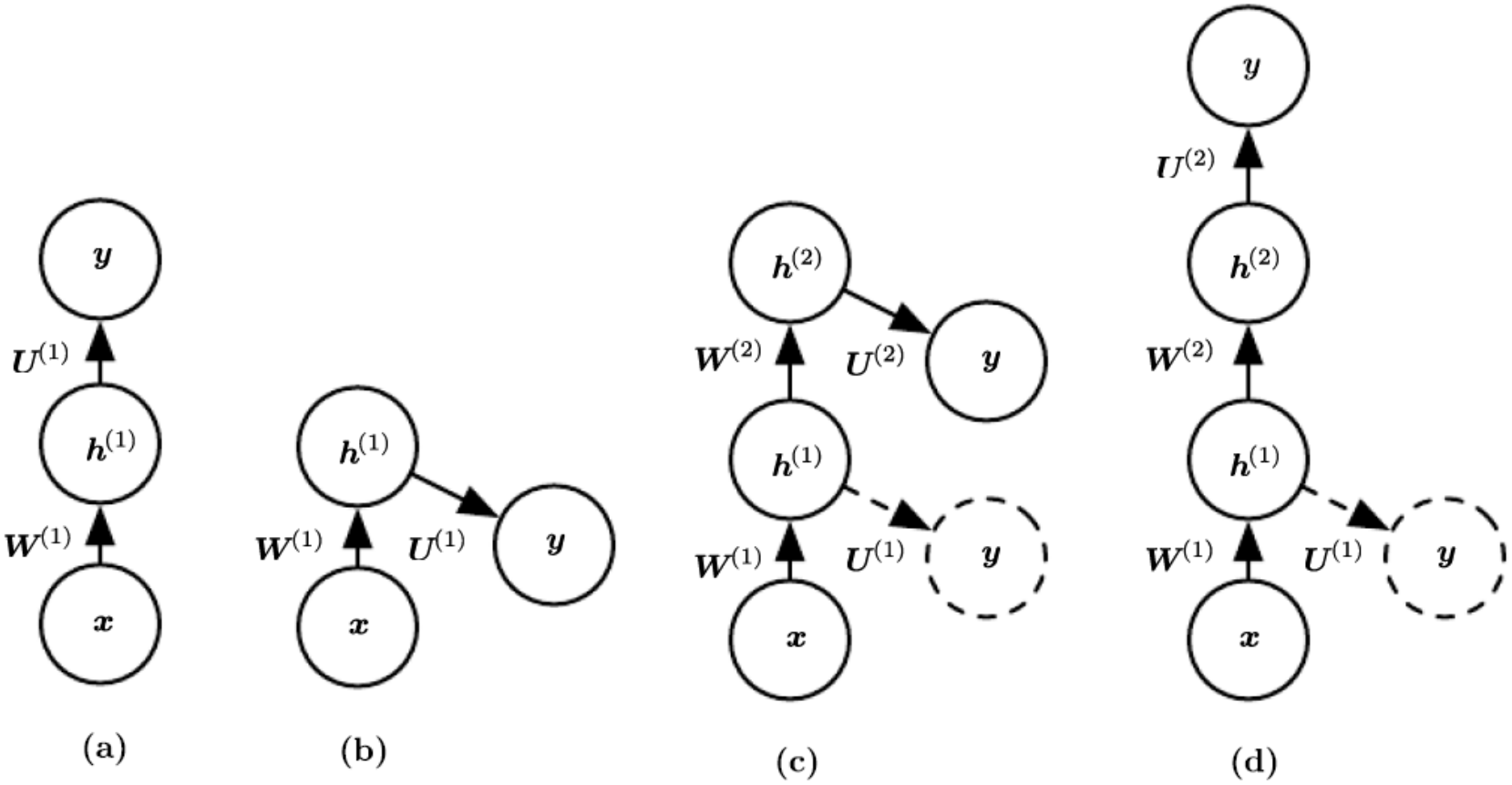
- Average together several points in the parameter space visited by an optimization algorithm
- If t iterations of gradient descent visit $\theta^{(1)}, \ldots, \theta^{(t)}$, then the output of Polyak averaging is $\hat{\theta}^{(t)} = \frac{1}{t}\sum_i \theta^{(i)}$
- On some problem classes (e.g., gradient descent on convex problem), this approach has strong convergence guarantees
- On neural networks, this approach is more heuristic, but performs well in practice
- Main idea: optimization may leap back and forth across a valley several times without visiting the bottom of the valley; average of the previous parameters should be close to the bottom of the valley
- In non-convex problems, the previous points may belong to many different regions; thus, in this case exponentially decaying running average is useful: $\hat{\theta}^{(t)} = \alpha\hat{\theta}^{(t-1)} + (1-\alpha)\theta^{(t)}$

Supervised Pretraining

- Training a complex model is difficult
- Pretraining: training simple models on simple tasks before confronting the challenge of training the desired model to perform the desired task
- Greedy algorithms are used for pretraining
  - Greedy algorithms break a problem into many components, then solve for the optimal version of each component in isolation
  - Combining the individually optimal components is not guaranteed to yield an optimal complete solution
  - However, greedy algorithms are efficient, and the quality of a greedy solution is often acceptable if not optimal
  - Greedy algorithms can be followed a find-tuning stage where a joint optimization algorithm searches for an optimal solution to the full problem
    - Initializing the joint optimization algorithm with a greedy solution can greatly speed up learning and improve the quality of the solution

Supervised Pretraining

- Greedy supervised pretraining



Why does it work?
It gives better guidance to the intermediate levels of a deep hierarchy

Design Models to Aid Optimization

- To improve optimization, the best strategy is not always to improve the optimization algorithm; a better way is to design models easier to optimize
- Most of the advances in neural network learning over the past 30 years have been obtained by changing the model family rather than changing the optimization procedure
  - SGD with momentum, which was used in the 1980s, remains one of the state of the art methods for modern neural network
- Examples
  - Mitigate vanishing gradient problem by
    - Using ReLU activation function
    - Skip connections between layers

Continuation Methods and Curriculum Learning

- Continuation methods
  - A family of strategies to make optimization easier by choosing initial points to ensure that local optimization spends most of its time in well-behaved regions of space
  - Idea: construct a series of objective function over the same parameters: to minimize $J(\theta)$, we construct new cost functions $\{J^{(0)}, \dots, J^{(n)}\}$
  - These functions are designed to be increasingly difficult, where $J^{(0)}$ is the easiest
    - $J^{(i)}$ is easier than $J^{(i+1)}$ when $J^{(i)}$ is well behaved over more of θ space

Continuation Methods and Curriculum Learning

- Continuation methods
  - Continuation methods were designed with the goal of overcoming the challenge of local minima: reach a global minimum despite the presence of many local minima
  - To do so, these continuation methods construct easier cost functions by "blurring" the original cost function, which can be done by approximating

    $J^{(i)}(\theta) = E_{\theta' \sim N(\theta, \sigma^{(i)^2})} J(\theta')$ via sampling

    - Intuition: some non-convex functions become approximately convex when blurred
  - In many cases, the blurring preserves enough information about the location of a global minimum which can be found by solving progressively less blurred versions of the problem
    - $\sigma(i)$ decreases as i increases

Continuation Methods and Curriculum Learning

- Curriculum learning (or shaping)
  - A continuation method
  - Idea: Plan a learning process to begin by learning simple concepts, and progress to learning more complex concepts that depend on these simpler concepts
    - This is useful for animal training as well as machine learning
  - Earlier $J^{(i)}$ are made easier by increasing the influence of simpler examples (either by assigning their contributions to the cost function larger coefficients, or by sampling them more frequently)
    - How to distinguish simple and difficult examples? Apply a well-known algorithm; correct high-confidence examples would become simple examples, and wrong or low-confidence examples would be difficult examples
  - Successful in natural language processing and computer vision

What you need to know

- Optimization strategies and meta-algorithms
  - Batch Normalization
  - Coordinate Descent
  - Polyak Averaging
  - Supervised Pretraining
  - Designing Models to Aid Optimization
  - Continuation Methods and Curriculum Learning

Questions?