Optimization for Training Deep Models

Jinhwan Suk

Department of Mathematical Science, KAIST

May 14, 2020

Contents

Approximate Second-Order Methods

- Objective function : *empirical risk*

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

- Second-order approximation of $J(\theta)$ near θ_0 is

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)$$

$$\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)$$

Contents

Newton's Method

Algorithm 1: Newton's Method

Require: Initial parameter θ_0

Require: Training set of m examples

while *stopping criterion not met* **do**

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

Compute Hessian : $H \leftarrow \frac{1}{m} \nabla_\theta^2 \sum_i \mathcal{L}(f(x^{(i)}; \theta), y^{(i)})$;

Compute Hessian inverse : H^{-1};

Compute update : $\Delta\theta = -H^{-1}g$;

Apply update : $\theta = \theta + \Delta\theta$

end

- Saddle point is problematic for Newton's method $\rightarrow$ Regularization strategy
- Computational complexity of requiring the inverse of $n \times n$ matrix is $O(n^3)$.

Contents

Conjugate Gradients

Motivation :

$$J(x) \approx J(x_0) + (x - x_0)^T \nabla J(x_0) + \frac{1}{2}(x - x_0)^T H(x - x_0)$$

$$\hat{x} = x_0 + \arg\min_{v \in \mathbb{R}^n} \left(\frac{1}{2} v^T H v + v^T \nabla J(\theta) \right)$$

- Newton method : solve

$$\frac{\partial}{\partial v} \left(\frac{1}{2} v^T H v + v^T \nabla J(\theta) \right) = 0$$

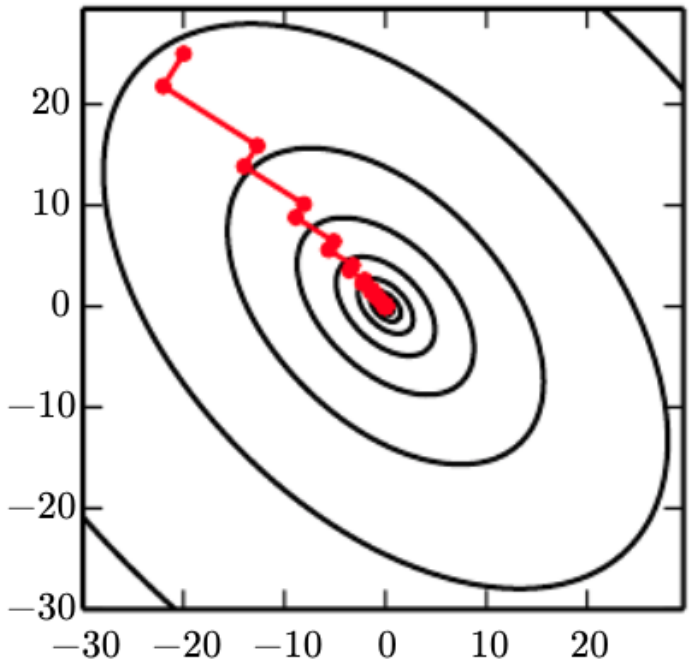
- Newton method is fast but we need to calculate $H^{-1} \nabla_\theta J(\theta)$.
- So we use descent method to solve the minimization problem.

Steepest Descent Method

- The first-order approximation of $L(x+v)$ around x is

$$L(x+v) \approx L(x) + \nabla L(x)^T v$$

- $\nabla L(x)^T v$ is minimized when v is opposite direction as gradient if v is constrained by $\|v\|_2 = 1$.
- Two ways of choosing step size
  1. choose constant $\epsilon > 0$ and update $x_{new} = x - \epsilon \nabla L(x)$
  2. **line search** : choose ϵ that results in the smallest $L(x - \epsilon \nabla L(x))$



- d_{t-1} : previous search direction

$$\nabla_\theta L(\theta) \cdot d_{t-1} = 0, \quad d_t = \nabla_\theta L(\theta)$$

- We must re-minimize the objective in the previous gradient direction.
- In the method of **conjugate gradients**, it will not undo progress made in that direction.

Conjugate Gradients

Definition (Conjugate direction)

Let H_0 be a symmetric matrix. $\{d_1, ..., d_m\} \subset \mathbb{R}^n$ is H_0-conjugate if

$$d_i^T H_0 d_j = 0, \ \forall \, i \neq j$$

Lemma

Let H be positive definite and $\{v_1, ..., v_n\}$ be a orthogonal eigenvectors of H. Then $\{v_1, ..., v_n\}$ is H-conjugate.

Lemma

Let H be positive definite. If $\{d_1, ..., d_m\} \subset \mathbb{R}^n$ is H-conjugate, then they are linearly independent.

Conjugate Gradients

Theorem (Conjugate Direction Theorem)

Optimization problem : $\hat{x} = x_0 + \arg\min_{v \in \mathbb{R}^n} \left(\frac{1}{2} v^T H v + v^T \nabla J(\theta) \right)$

Let $\{d_1, ..., d_n\}$ *be H-conjugate and* x_0 *be an initial point.*

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

$g_k = H x_k + \nabla J(\theta)$

$\alpha_k = -\frac{g_k^T d_k}{d_k^T H d_k}$

Then, after n steps, $x_n = \hat{x}$

- Hence, conjugate descent method is improved method than steepest descent method.
- How can we find the H-conjugate set?? Eigenvectors??

There is inductive ways to find H-conjugate vectors. initialize $d_0 = 0$
Choose $d_t = \nabla J(\theta) + \beta_t d_{t-1}$ where β_t is given by

1. Fletcher-Reeves :

$$\beta_t = \frac{\nabla_\theta J(\theta_t)^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})}$$

2. Polak-Ribiere :

$$\beta_t = \frac{(\nabla_\theta J(\theta_t) - \nabla_\theta J(\theta_{t-1}))^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})}$$

Then, $\{d_1, ..., d_t\}$ is H-conjugate. Computational complexity is $O(n)$.

Conjugate Gradient Method

Algorithm 2: Conjugate Gradient Method

Require: Initial parameter θ_0

Require: Training set of m examples

Initialize $\rho_0 = 0$; $g_0 = 0$; $t = 1$;

while *stopping criterion not met* **do**

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

Compute $\beta_t = \frac{(g_t - g_{t-1})^T g_t}{g_{t-1}^T g_{t-1}}$ (Polak-Ribiere);

Compute search direction: $\rho_t = g_t + \beta_t \rho_{t-1}$;

Perform line search to find :

$\epsilon^* = \arg\min_\epsilon \frac{1}{m} \sum_1^m \mathcal{L}(f(x^{(i)}; \theta_t + \epsilon \rho_t), y^{(i)})$;

Apply update : $\theta = \theta + \epsilon^* \rho_t$

end

Nonlinear Conjugate Gradients

- Loss function in Deep learning is far from quadratic.
- Conjugate directions are no longer assured to remain at minimum of the objective for previous directions.
- The nonlinear conjugate gradient algorithm occasionally set $\beta_t = 0$ so that it is same as steepest descent method.

Contents

BFGS

- Recall that Newton's update is given by

$$\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)$$

- **BFGS** algorithm is to approximate the inverse with a matrix M_t and then determine the direction of descent method by $\rho_t = M_t g_t$.
- It requires $O(n^2)$ memory.
- **Limited Memory BFGS**(or L-BFGS) decreases the memory cost of BFGS.

Contents

Batch Normalizaton

Ioffe and Szegedy, (2015)

- Let $P_s(x)$ be the density of x in the observed data and $P_t(x)$ be the density of x for evaluation of the predictive performance.
- The situation $P_s(x) \neq P_t(x)$ is called **covariate shift** in distribution.
  e.g) *domain adaptation problem* : source domain $\neq$ target domain
- **Internal covariate shift** is the change in the distributions of internal nodes of a deep network in the course of training.
  - Author says that eliminating it offers a promise of faster training.
- *By whitening the inputs to each layer*, we would achieve the fixed distributions of inputs that may remove the ill effects of internal covariate shift.

Batch Normalizaton

Ioffe and Szegedy, (2015)

- The full whitening of each layer's inputs is costly and not differentiable. $\rightarrow$ normalization

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{Var(x^{(k)})}}$$

- Normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity.

$$y^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)}$$

Batch Normalizaton

Ioffe and Szegedy, (2015)

Algorithm 3: Batch Normalizing Transform

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1, ..., x_m\}$

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

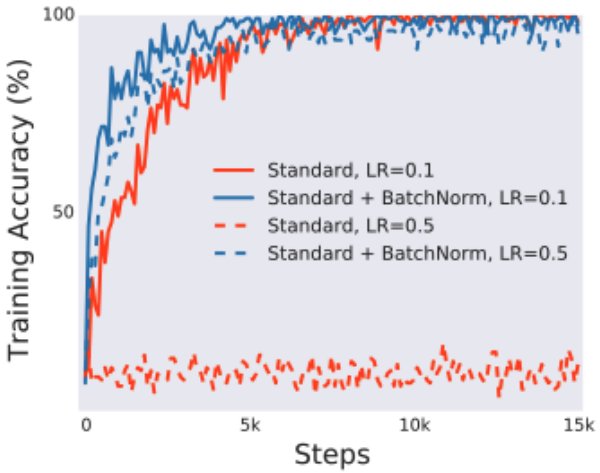
$$\sigma_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

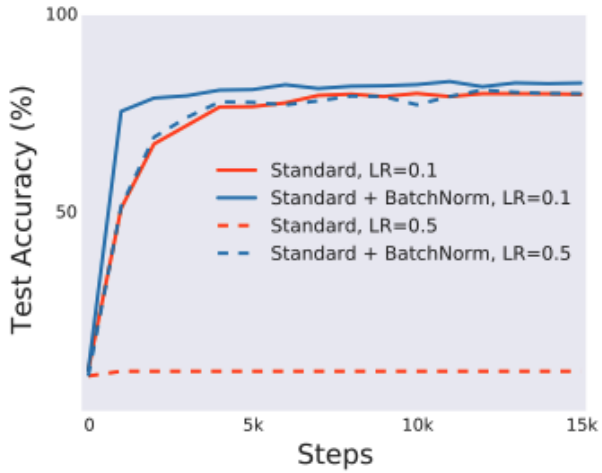
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$$

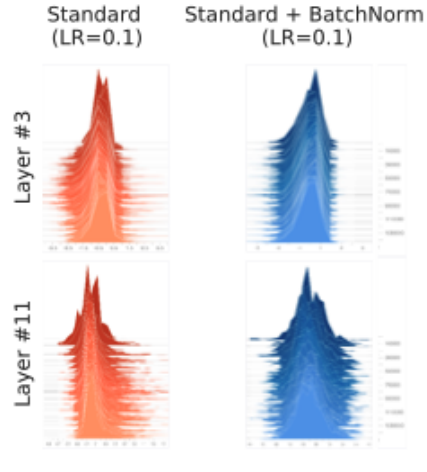
$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$$

Batch Normalizaton

Ioffe and Szegedy, (2015)





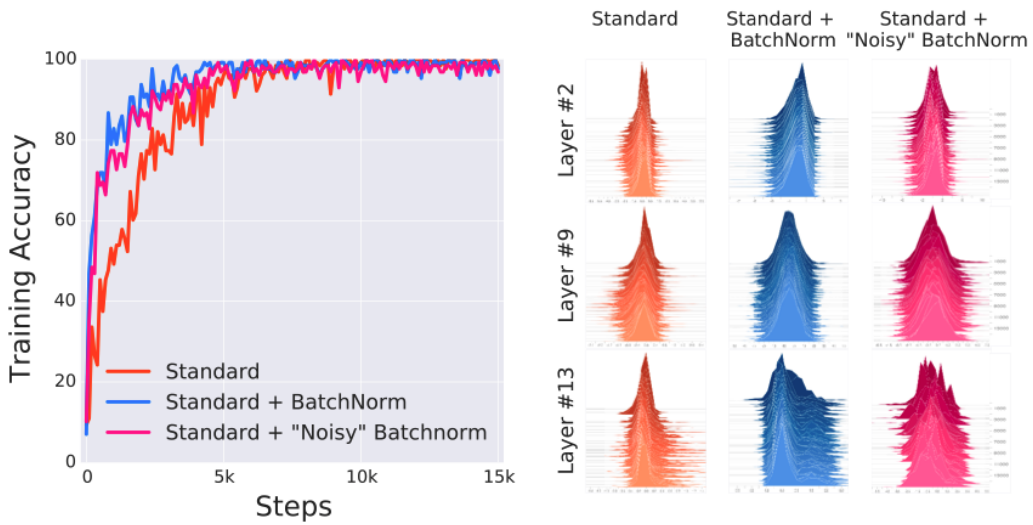


- The gap between the performance of BatchNorm and Non-BatchNorm is clear.
- The difference in the evolution of layer input distributions seems to be much less pronounced.

Batch Normalizaton

How Does Batch Normalization Help Optimization?, *NeurIPS'18*

1. Is the effectiveness of BatchNorm indeed related to internal covariate shift?
2. Is BatchNorm's stabilization of layer input distributions even effective in reducing internal covariate shift(ICS)?



ICS is not directly connected to the training performance!!

Batch Normalizaton

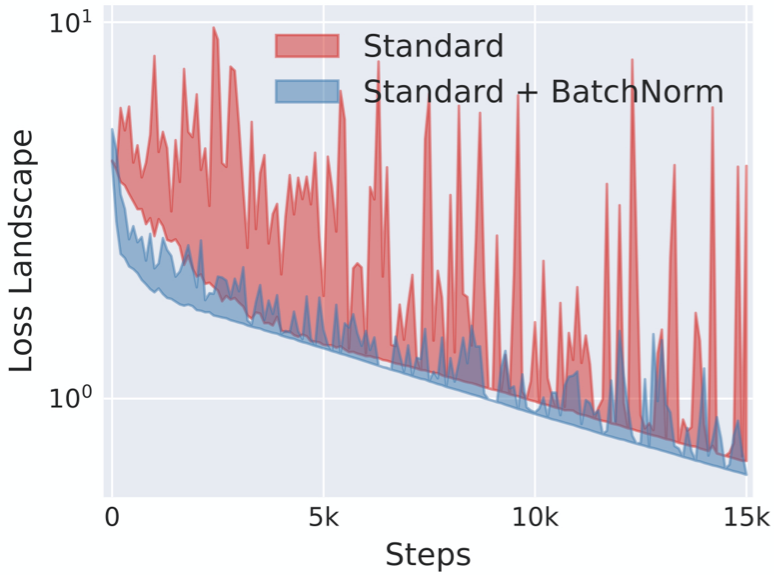
How Does Batch Normalization Help Optimization?, *NeurIPS'18*

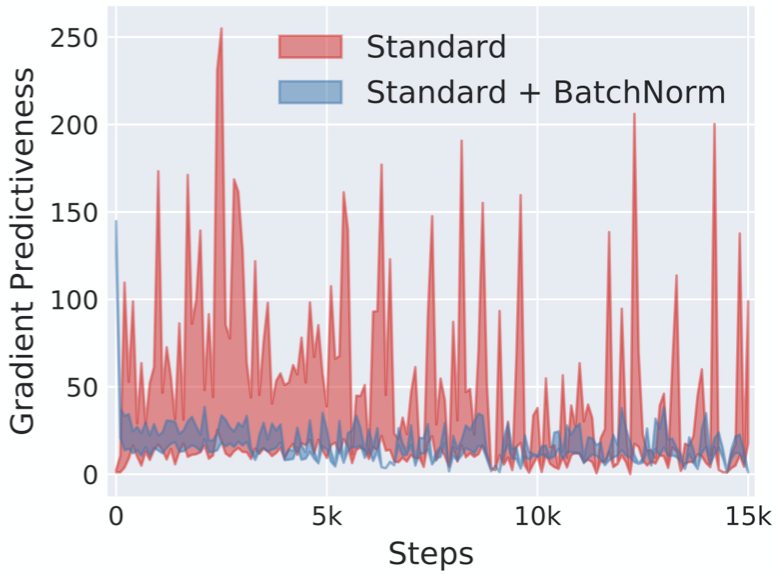
- Instead of ICS, we concentrate on stability of optimization.
- We measure variation of the values of the loss :

$$\mathcal{L}(\theta + \eta\mathcal{L}(\theta)), \quad \eta \in [0.05, 0.4]$$

- We also measure gradient predictiveness, i.e., the change of the gradient of loss :

$$\left\| \nabla\mathcal{L}(\theta) - \nabla\mathcal{L}(\theta + \eta\nabla\mathcal{L}(x)) \right\|, \quad \eta \in [0.05, 0.4]$$





Contents

Coordinate Descent

- minimize $\mathcal{L}(\theta)$ with respect to a single variable w_i, then minimize it with respect to another variable w_j, and so on. (**Coordinate descent**)
- **Block coordinate descent** refers to minimizing with respect to a subset of the variables.

Consider the cost function

$$J(H, W) = \sum_{i,j} \left(X - W^T H\right)^2_{i,j} + \sum_{i,j} |H_{i,j}|$$

$J(H, W)$ is not convex, but $J(H, W_0)$ and $J(H_0, W)$ are convex.

$$\text{Bad Case : } J(w_1, w_2) = (w_1 - w_2)^2 + \alpha(w_1^2 + w_2^2), \quad \alpha > 0$$

Contents

Polyak Averaging

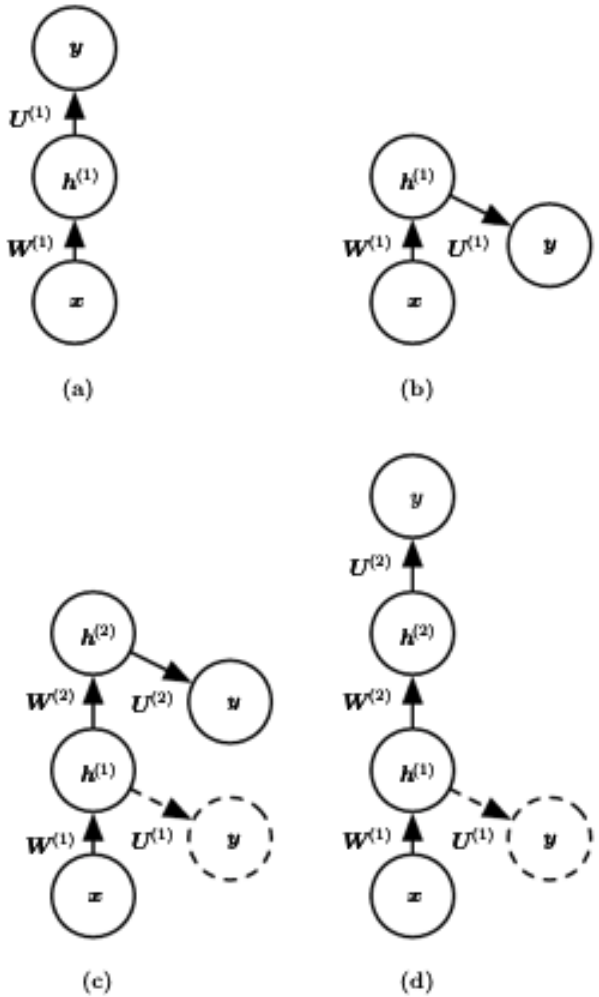
- gradient descent visits $\theta^{(1)} \rightarrow \theta^{(2)} \rightarrow \cdots \rightarrow \theta^{(t)} \rightarrow \cdots$
- **Polyak averaging** algorithm : $\hat{\theta}^{(t)} = \frac{1}{t} \sum_{i=1}^{t} \theta^{(i)}$
- Has strong convergence guarantees in convex settings
- When applying to nonconvex problems, it is typical to use an exponentially decaying running average :

$$\hat{\theta}^{(t)} = \alpha \hat{\theta}^{(t-1)} + (1 - \alpha)\hat{\theta}^{(t)}$$

Contents

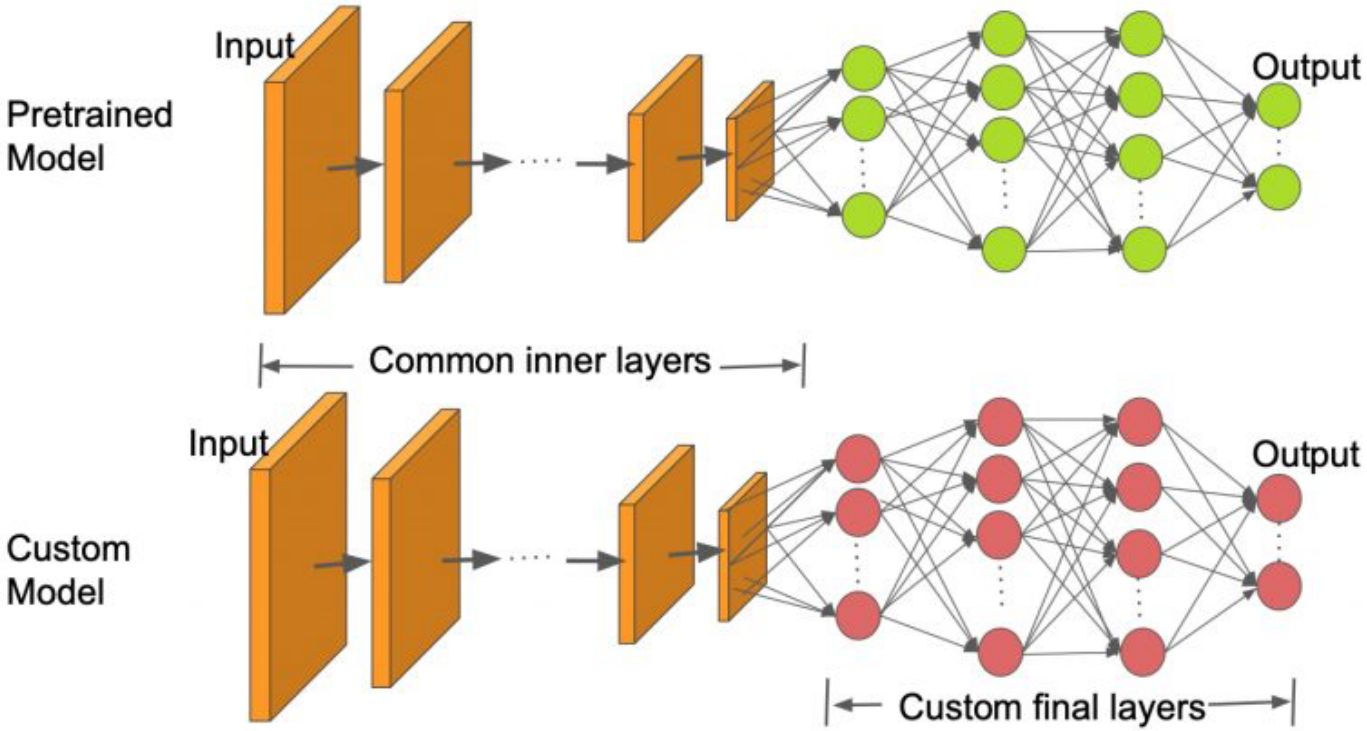
Supervised Pretraining

Greedy Supervised Pretraining



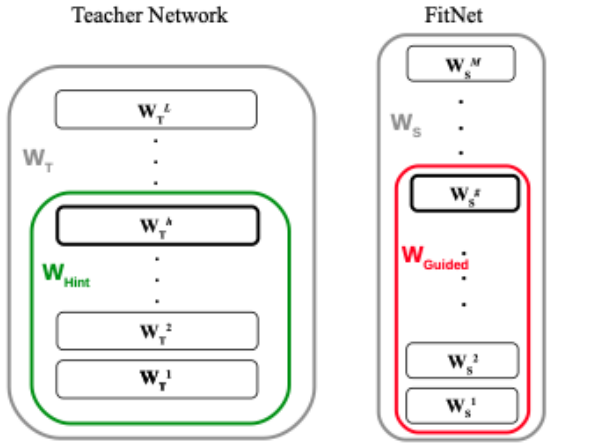
Supervised Pretraining

Transfer Learning

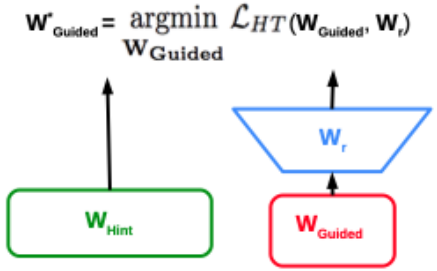


Supervised Pretraining

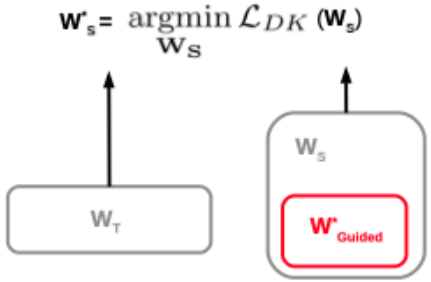
FitNets approach



(a) Teacher and Student Networks



(b) Hints Training



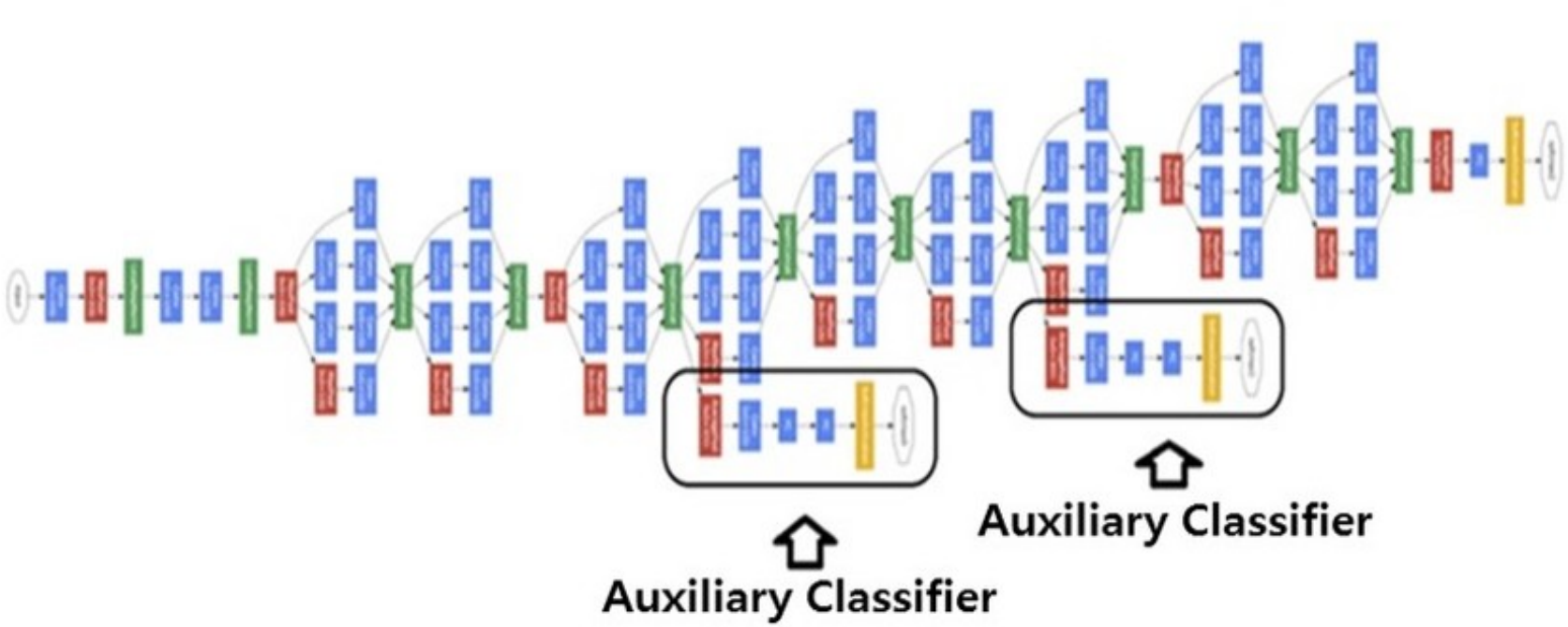
(c) Knowledge Distillation

$$\mathcal{L}_{HT}(W_{Guided}, W_r) = \frac{1}{2}\left\| u_h(x; W_{Hint}) - r(v_g(x; W_{Guided}; W_r)) \right\|^2$$

$$\mathcal{L}_{KD}(W_s) = \mathcal{H}(y_{true}, P_S) + \lambda \mathcal{H}(P_T^\tau, P_S^\tau)$$

Contents

Designing Models to Aid Optimization



- Ensure that the lower layers receive a large gradient.
- When training complete, the auxiliary heads may be discarded.

Contents

Continuation Methods

- Heuristic method that provide good suboptimal solutions.
- It starts by solving an easy problem, and progressively changes it to the actual complex task.

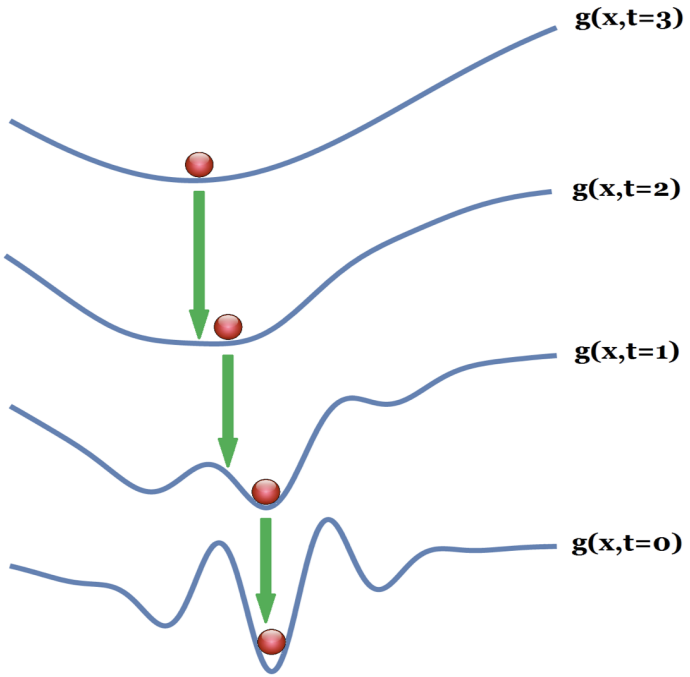


Figure 1: Plots show g versus x for each fixed time t.

Continuation Methods

Mobahi and Fisher(2015)

Definition (Gaussian Homotopy)

The **Gaussian homotopy** $g : \mathcal{X} \times \mathcal{T} \to \mathbb{R}$ for a function $f : \mathcal{X} \to \mathbb{R}$ is defined as the convolution of f with $k_t = \frac{1}{(\sqrt{2\pi}t)^d} e^{-\frac{\|x\|^2}{2t^2}}$,

$$g(x, t) := [f * k_t](x) = \int_{\mathcal{X}} f(y) k_t(x - y)\, dy$$

Algorithm 4: Continuation Method

Input: $f : \mathcal{X} \to \mathbb{R}$, Sequence $t_0 > t_1 > \cdots > t_N = 0$

$x_0 =$ global minimizer of $g(x; t_0)$;

for $k = 1$ *to* N **do**

$\quad x_k =$ Local minimizer of $g(x; t_k)$, initialized at x_{k-1}

end

Output: x_N

Curriculum Learning

Bengio et al. (2009)

- **Basic idea :** start out with only easy examples and then gradually increase the task.
- Curriculum can also be seen as a sequence of training criteria.
- Each training criterion in the sequence is associated with a different set of weights on the training examples

Formulation of Curriculum Learning

Let $P(Y|X)$ be the target training distribution and $0 \leq W_\lambda(x) \leq 1$ be the weight applied to example x at step λ in the curriculum sequence, with $0 \leq \lambda \leq 1$, and $W_1(x) = 1$, $\forall x$. For each λ, we need to learn $Q_\lambda(y|x) \propto W_\lambda(x)P(y|x)$.

Curriculum Learning

Bengio et al. (2009)

Definition (Curriculum)

We call the corresponding sequence of distribution Q_λ a **curriculum** if the entropy of these distributions increases

$$H(Q_\lambda) < H(Q_{\lambda+\epsilon}) \quad \forall \epsilon > 0$$

and $W_\lambda(x)$ is monotonically increasing in λ.

- How can we define "easy examples"?
- *Stochastic curriculum* : Random mix of easy and difficult examples is presented to learner, but the proportion of the difficult examples is gradually increased.