Introduction to Deep Learning

Arijit Mondal
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
arijit@iitp.ac.in

Optimization for Training Deep Models

Minimization of cost function

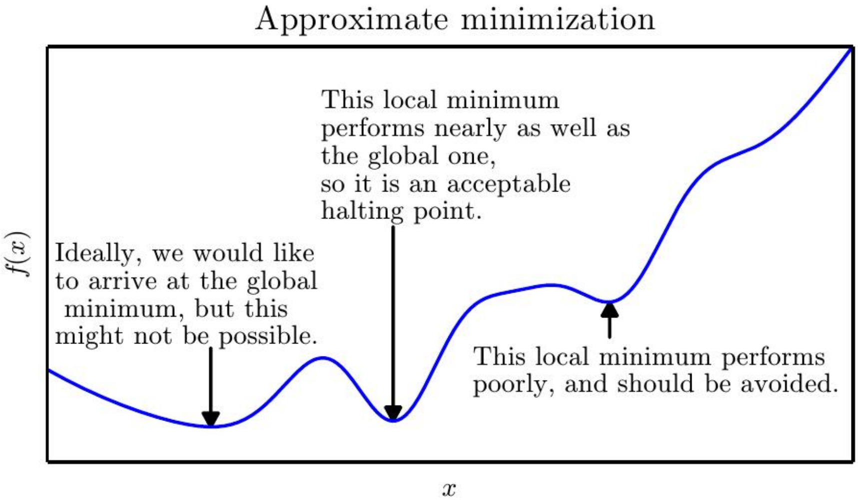

Image source: Deep Learning Book

Curvature

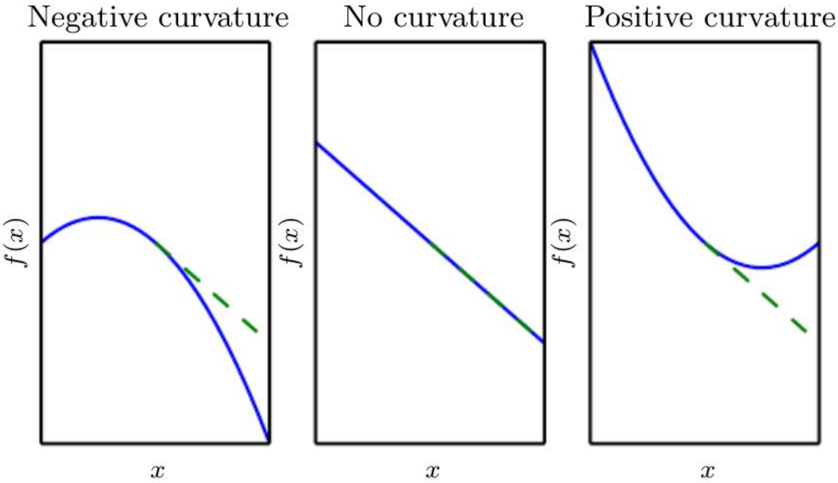

Image source: Deep Learning Book

Problem of optimization

- Differs from traditional pure optimization problem
- Performance of a task is optimized indirectly
- We optimize $J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}} L(f(\boldsymbol{x},\boldsymbol{\theta}),y)$ where $\hat{p}$ is the empirical distribution
- We would like to optimize $J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{\text{data}}} L(f(\boldsymbol{x},\boldsymbol{\theta}),y)$ where p is the data generating distribution
  - Also known as risk
- We hope minimizing J will minimize j^*

Empirical risk minimization

- Target is to reduce risk
- If the true distribution is known, risk minimization is an optimization problem
- When $p_{\text{data}}(\boldsymbol{x}, y)$ is unknown, it becomes machine learning problem
- Simplest way to convert machine learning problem to optimization problem is to minimize expected cost of training set
- We minimize empirical risk

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

- We can hope empirical risk minimizes the risk as well
  - In many cases empirical risk minimization is not feasible
  - Empirical risk minimization is prone to overfitting
  - Gradient based solution approach may lead to problem with 0-1 loss cost function

Surrogate loss function

- Loss function may not be optimized efficiently
  - Exact minimization of 0-1 loss is typically intractable
- Surrogate loss function is used
  - Proxy function for the actual loss function
  - Negative log likelihood of correct class used as surrogate function
- There are cases when surrogate loss function results in better learning
  - 0-1 loss of test set often continues to decrease for a long time after training set 0-1 loss has reached to 0
- A training algorithm does not halt at local minima usually
  - Tries to minimize surrogate loss function but halts when validation loss starts to increase
- Training function can halt when surrogate function has huge derivative

Batch

- Objective function usually decomposes as a sum over training example
- Typically in machine learning update of parameters is done based on an expected value of the cost function estimated using only a subset of of the terms of full cost function
- Maximum likelihood problem $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$
- Maximizing this sum is equivalent to maximizing the expectation over empirical distribution $J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x}, y, \boldsymbol{\theta})$
- Common gradient is given by $\nabla_{\boldsymbol{\theta}} = \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\text{model}}(\boldsymbol{x}, y, \boldsymbol{\theta})$
  - It becomes expensive as we need to compute for all examples
  - Random sample is chosen, then average of the same is taken
  - Standard error in estimating the variance is $\frac{\sigma}{\sqrt{n}}$ where σ is the true standard deviation
  - Redundancy in training examples is an issue

Batch

- Optimization algorithm that uses entire training set is called batch of deterministic gradient descent
- Optimization algorithm that uses single example at a time are known as stochastic gradient descent or online method

Minibatch

- Larger batch provides more accurate estimate of the gradient but with lesser than linear returns
- Multicore architecture are usually underutilized by small batches
- If all examples are to be processed parallely then the amount of memory scales with batch size
- Sometime, better run time is observed with specific size of the array
- Small batch can add regularization effect due to noise they add in learning process
- Methods that update the parameters based on $\boldsymbol{g}$ only are usually robust and can handle small batch size ~ 100
- With Hessian matrix batch size becomes $\sim 10{,}000$ (Require to minimize $\boldsymbol{H}^{-1}\boldsymbol{g}$)
- SGD minimizes generalization error on minibatches drawn from a stream of data

Issues in optimization

- Ill conditioning
- Local minima
- Plateaus
- Saddle points
- Flat region
- Cliffs
- Exploding gradients
- Vanishing gradients
- Long term dependencies
- Inexact gradients

Ill conditioning

- Ill conditioning of Hessian matrix
  - Common problem in most of the numerical optimization
  - The ratio of smallest to largest eigen value determines the condition number
  - We have the following

$$
\begin{aligned}
f(\boldsymbol{x}) &= f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^T \boldsymbol{H} (\boldsymbol{x} - \boldsymbol{x}^{(0)}) \\
f(\boldsymbol{x} - \epsilon \boldsymbol{g}) &= f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2} \epsilon \boldsymbol{g}^T \boldsymbol{H} \epsilon \boldsymbol{g}
\end{aligned}
$$

  - It becomes a problem when $\frac{1}{2}\epsilon^2 \boldsymbol{g}^T \boldsymbol{H}\boldsymbol{g} - \epsilon \boldsymbol{g}^T \boldsymbol{g} > 0$
  - In many cases gradient norm does not shrink much during learning and $\boldsymbol{g}^T \boldsymbol{H}\boldsymbol{g}$ grows more rapidly
  - Makes the learning process slow

Local minima

- For convex optimization problem local minima is often acceptable
- For nonconvex function like neural network many local minima are possible
  - This is not a major problem
- Neural network and any models with multiple equivalently parameterized latent variables results in local minima
  - This is due to model identifiability
  - Model is identifiable if sufficiently large training set can rule out all but one setting of model parameters
  - Model with latent variables are often not identifiable as exchanging of two variables does not change the model
    - m layers with n unit each can result in $(n!)^m$ arrangements
    - This non-identifiability is known as weight space symmetry
  - Neural network has other non-identifiability scenario
    - ReLU or MaxOut — weight is scaled by α and output is scaled by $\frac{1}{\alpha}$

Local minima

- Model identifiability issues mean that there can be uncountably infinite number of local minima
- Non-identifiability result is local minima and are equivalent to each other in cost function
- Local minima can be problematic if they have high cost compared to global minima

Other issues

- Saddle points
  - Gradient is 0 but some have higher and some have lower value around the point
  - Hessian matrix has both positive and negative eigen value
- In high dimension local minima are rare, saddle points are very common
  - For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, the expected ratio of number of saddle points to local minima grows exponentially with n
    - Eigenvalue of Hessian matrix
- Cliffs - uses gradient clipping
- Long term dependency — mostly applicable for recurrent neural network
  - $\boldsymbol{w}^t = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})^t \boldsymbol{V}^{-1}$
  - vanishing and exploding gradient
- Inexact gradients — bias in estimation of gradient

Stochastic gradient descent

- Inputs — Learning rate (ϵ_k), weight parameters ($\boldsymbol{\theta}$)
- Algorithm for SGD:

while stopping criteria not met

Sample a minibatch from training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$

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

Update parameters $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon_k \hat{g}$

end while

Stochastic gradient descent

- Learning rate is a crucial parameter
- Learning rate ϵ_k is used in the kth iteration
- Gradient does not vanishes even when we reach minima as minibatch can introduce noise
- True gradient becomes small and then 0 when batch gradient descent is used
- Sufficient condition on learning rate for convergence of SGD
  - $\sum_{k=1}^{\infty} \epsilon_k = \infty$
  - $\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$
- Common way is to decay the learning rate $\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$ with $\alpha = \frac{k}{\tau}$

Stochastic gradient descent

- Choosing learning rate is an art than science!
  - Typically ϵ_τ is 1% of ϵ_0
- SGD usually performs well for most of the cases
- For large task set SGD may converge within the fixed tolerance of final error before it has processed all training examples

Momentum

- SGD is the most popular. However, learning may be slow sometime
- Idea is to accelerate learning especially in high curvature, small but consistent gradients
- Accumulates an exponential decaying moving average of past gradients and continue to move in that direction
- Introduces a parameter $\boldsymbol{v}$ that play the role of velocity
  - The velocity is set to an exponentially decaying average of negative gradients
- Update is given by

$$\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)}) \right)$$

  - α — hyperparameter, denotes the decay rate

Momentum

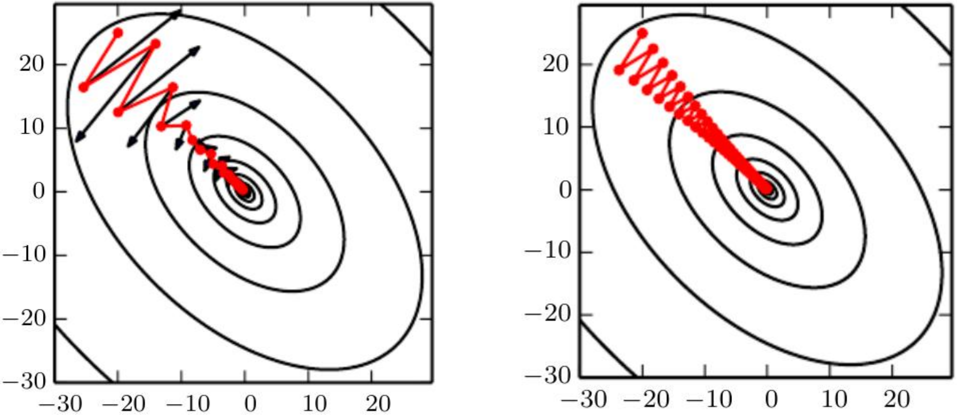

Image source: Deep Learning Book

Stochastic gradient descent with momentum

- Inputs — Learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), momentum parameter (α), initial velocity ($\boldsymbol{v}$)
- Algorithm:

while stopping criteria not met

Sample a minibatch from training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$

Estimate of gradient: $\boldsymbol{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)})$

Update of velocity: $\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

Update parameters: $\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{v}$

end while

Momentum

- The step size depends on how large and how aligned a sequence gradients are
- Largest when many successive gradients are in same direction
- If it observes $\boldsymbol{g}$ always, then it will accelerate in $-\boldsymbol{g}$ with terminal velocity $\dfrac{\epsilon|\boldsymbol{g}|}{1-\alpha}$
- Typical values for α is 0.5, 0.9, 0.99. However this parameter can be adapted.

Nesternov momentum

- Inputs — Learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), momentum parameter (α), initial velocity ($\boldsymbol{v}$)
- Algorithm:

while stopping criteria not met

Sample a minibatch from training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$

Interim update: $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \alpha \boldsymbol{v}$

Gradient at interim point: $\boldsymbol{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}, \tilde{\boldsymbol{\theta}}), y^{(i)})$

Update of velocity: $\boldsymbol{v} = \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

Update parameters: $\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{v}$

end while

Parameter initialization

- Training algorithms are iterative in nature
- Require to specify initial point
- Training deep model is difficult task and affected by initial choice
  - Convergence
  - Computation time
  - Numerical instability
- Need to break symmetry while initializing the parameters

Adaptive learning rate

- Learning rate can affect the performance of the model
- Cost may be sensitive in one direction and insensitive in the other directions
- If partial derivative of loss with respect to model remains the same sign then the learning rate should decrease
  - Applicable for full batch optimization

AdaGrad

- Adapts the learning rate of all parameters by scaling them inversely proportional to the square root of the sum of all historical squared values of the gradient
  - Parameters with largest partial derivative of the loss will have rapid decrease in learning rate and vice-versa
  - Net effect is greater progress
- It performs well on some models

Steps for AdaGrad

- Inputs — Global learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), small constant (δ), gradient accumulation ($\boldsymbol{r}$)
- Algorithm:

while stopping criteria not met

Sample a minibatch from training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$

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

Accumulated squared gradient: $\boldsymbol{r} = \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

RMSProp

- Gradient is accumulated using an exponentially weighted moving average
  - Usually, AdaGrad converges rapidly in case of convex function
  - AdaGrad reduces the learning rate based on entire history
- RMSProp tries to discard history from extreme past
- This can be combined with momentum

Steps for RMSProp

- Inputs — Global learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), small constant (δ), gradient accumulation ($\boldsymbol{r}$), decay rate (ρ)
- Algorithm:

while stopping criteria not met

Sample a minibatch from training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$

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

Accumulated squared gradient: $\boldsymbol{r} = \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

end while

Steps for RMSProp with Nesternov

- Inputs — Global learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), small constant (δ), gradient accumulation ($\boldsymbol{r}$), decay rate (ρ), initial velocity ($\boldsymbol{v}$), momentum coefficient (α)
- Algorithm:

while stopping criteria not met

Sample a minibatch from training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$

Interim update: $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \alpha \boldsymbol{v}$

Gradient: $\boldsymbol{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}, \tilde{\boldsymbol{\theta}}), y^{(i)})$

Accumulated squared gradient: $\boldsymbol{r} = \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Update of velocity: $\boldsymbol{v} = \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{v}$

end while

Approximate 2nd order method

- Taking 2nd order term to train deep neural network
- The cost function at $\boldsymbol{\theta}$ near the point $\boldsymbol{\theta}_0$ is given by

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Solution for critical point provides $\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$
  - If the function is quadratic then it jumps to minimum
  - If the surface is not quadratic but $\boldsymbol{H}$ is positive definite then this approach is also applicable
- This approach is known as Newton's method

Steps for Newton's method

- Inputs — Initial parameters ($\boldsymbol{\theta}_0$)
- Algorithm:

while stopping criteria not met

Sample a minibatch from training set $\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(m)}\}$ with labels $\{\boldsymbol{y}^{(i)}\}$

Compute gradient: $\boldsymbol{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)})$

Compute Hessian: $\boldsymbol{H} = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}}^2 L(f(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}), y^{(i)})$

Compute inverse Hessian: $\boldsymbol{H}^{-1}$

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

Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Batch normalization

- Reduces internal covariate shift
- Issues with deep neural network
  - Vanishing gradients
    - Use smaller learning rate
    - Use proper initialization
    - Use ReLU or MaxOut which does not saturate
- This approach provides inputs that has zero mean and unit variance to every layer of input in neural network

Batch normalization transformation

- Applying to activation x over a mini-batch
- Input — values of x over a minibatch $\mathcal{B} = \{x_{1\ldots m}\}$, parameters to be learned — γ, β
- Output — $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$
  - Minibatch mean: $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i$
  - Minibatch variance: $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$
  - Normalize: $\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
  - Scale and shift: $y_i = \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$

Reference: Batch normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, S Ioffe, C Szegedy, 2015

Computational graph for BN

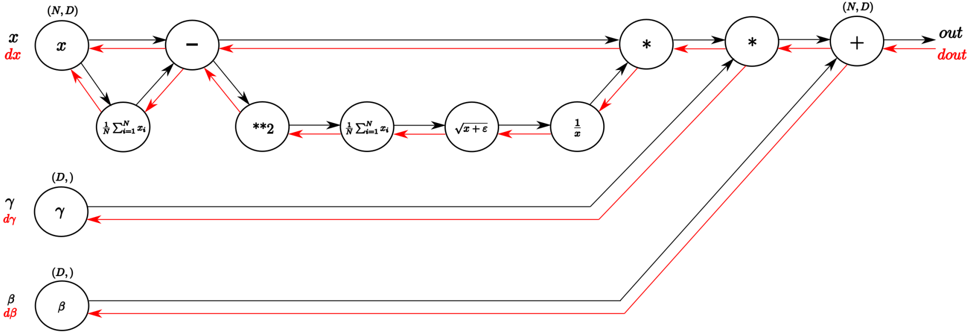

Image source:https://kratzert.github.io

Back-propagation for BN (9)

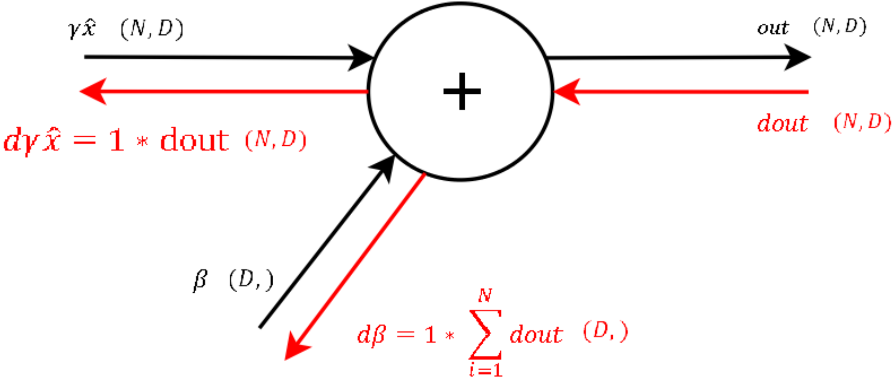

Image source:https://kratzert.github.io

Back-propagation for BN (8)

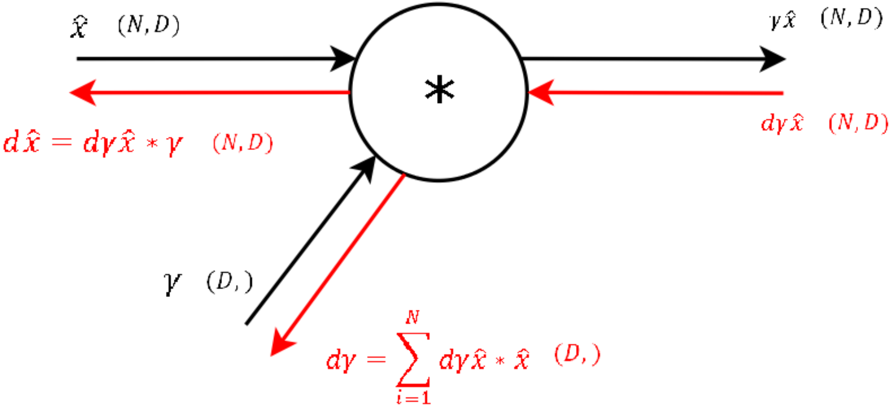

Image source:https://kratzert.github.io

Back-propagation for BN (7)

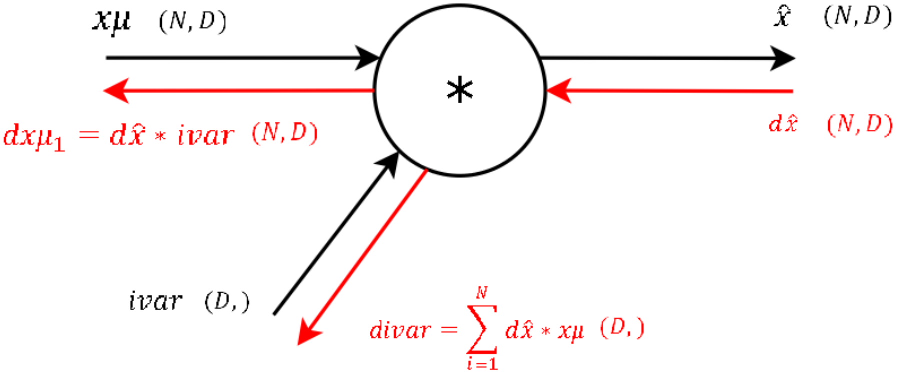

Image source:https://kratzert.github.io

Back-propagation for BN (6)

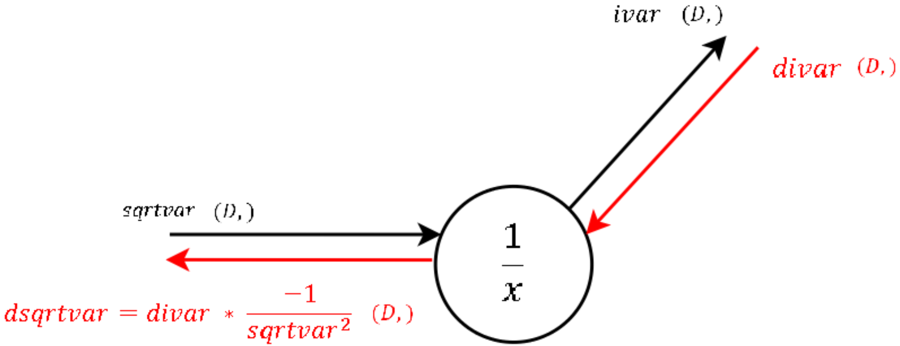

Image source:https://kratzert.github.io

Back-propagation for BN (5)

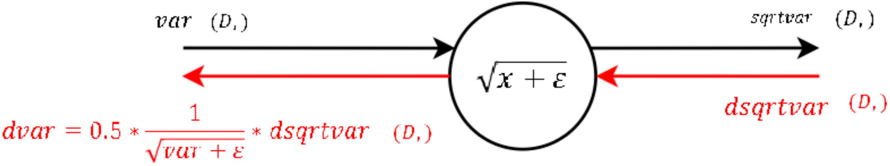

Image source:https://kratzert.github.io

Back-propagation for BN (4)

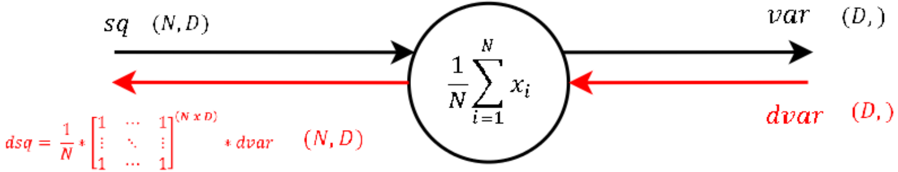

Image source:https://kratzert.github.io

Back-propagation for BN (3)

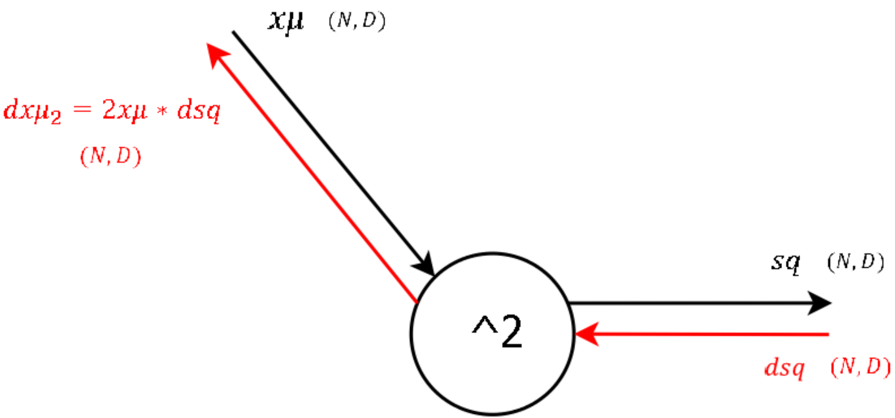

Image source:https://kratzert.github.io

Back-propagation for BN (2)

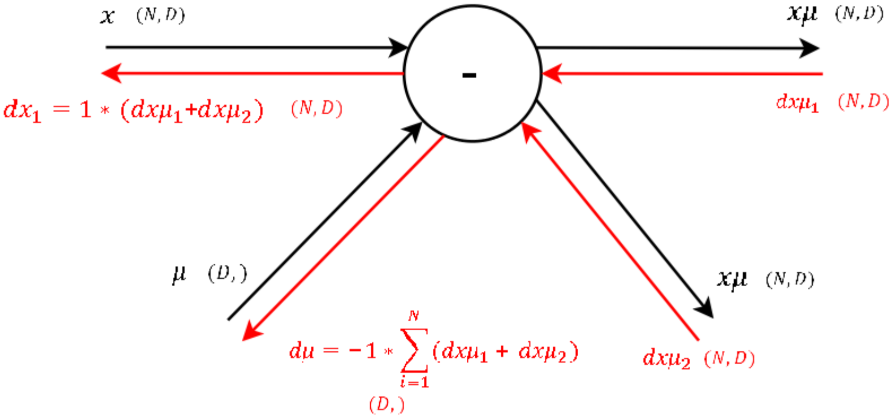

Image source:https://kratzert.github.io

Back-propagation for BN (1)

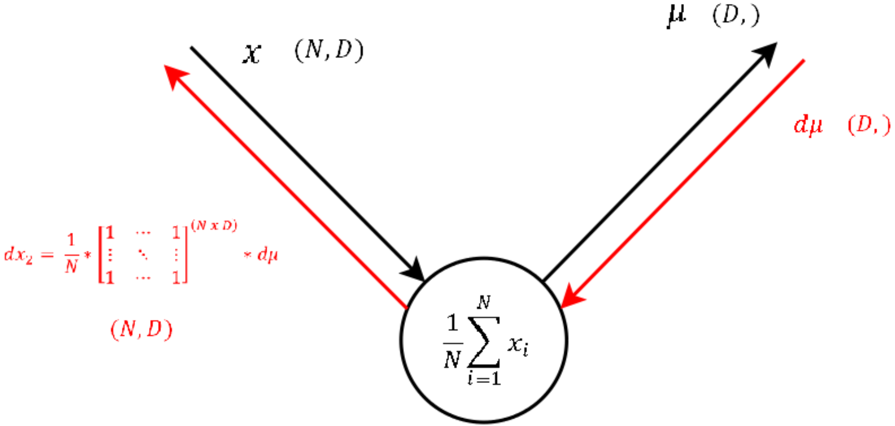

Image source:https://kratzert.github.io

Back-propagation for BN (0)

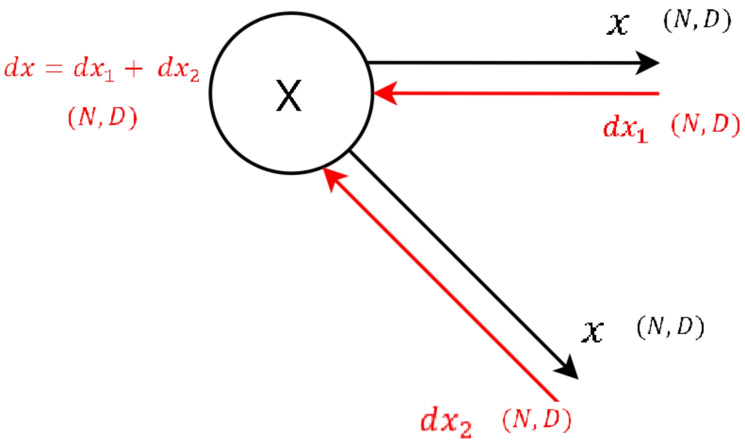

Image source:https://kratzert.github.io

Training & inference using batch-normalization

- Input — Network N with trainable parameters $\boldsymbol{\theta}$, subset of activations $\{x^{(k)}\}_{k=1}^{K}$
- Output — Batch-normalized network for inference $N_{\text{BN}}^{\text{inf}}$
- Steps:
  - Training BN network: $N_{\text{BN}}^{\text{tr}} = N$
  - **for** $k = 1, \ldots, K$
    - Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}} = N$
    - Modify each layer in $N_{\text{BN}}^{\text{tr}} = N$ with input $x^{(k)}$ to take $y^{(k)}$ instead
  - Train $N_{\text{BN}}^{\text{tr}}$ and optimize $\boldsymbol{\theta} \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
  - $N_{\text{BN}}^{\text{inf}} = N_{\text{BN}}^{\text{tr}}$
  - **for** $k = 1, \ldots, K$
    - Process multiple training minibatches and determine $\mathbb{E}[x] = \mathbb{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $V[x] = \frac{m}{m-1}\mathbb{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$
    - In $N_{\text{BN}}^{\text{inf}}$ replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{V[x]+\epsilon}}x + (\beta - \frac{\gamma\mathbb{E}[x]}{\sqrt{V[x]+\epsilon}})$