



香港中文大學
The Chinese University of Hong Kong

CMSC5743

L03: CNN Training

Bei Yu

(Latest update: September 27, 2020)

Fall 2020

Overview



CNN Initialization

Gradient Descent Methods

Learning rate annealing

Normalization



CNN Initialization

Gradient Descent Methods

Learning rate annealing

Normalization

CNN Initialization



- ▶ Constant initialization or Random initialization
- ▶ Orthogonal initialization
- ▶ Xavier initialization
- ▶ Kaiming initialization

Initialization methods contain/adapt materials developed by

- ▶ Kaiming He et al. (Dec. 2015). “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*
- ▶ Andrew M. Saxe, James L. McClelland, and Surya Ganguli (2013). *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*. eprint: [arXiv:1312.6120](https://arxiv.org/abs/1312.6120)
- ▶ Xavier Glorot and Yoshua Bengio (13–15 May 2010). “Understanding the difficulty of training deep feedforward neural networks”. In: ed. by Yee Whye Teh and Mike Titterton. Vol. 9. *Proceedings of Machine Learning Research*. Chia Laguna Resort, Sardinia, Italy: JMLR Workshop and Conference Proceedings, pp. 249–256. URL: <http://proceedings.mlr.press/v9/glorot10a.html>

Constant initialization or Random initialization



- ▶ Constant Initialization

Constant Initialization

All weights are initialized by a constant

This will reduce the learning efficiency and generalization ability of a neural network

- ▶ Random initialization

Uniform distribution initialization and Gaussian distribution initialization

$$W \sim U[a, b]$$

Gaussian initialization

$$W \sim \text{Normal}(\mu, \sigma^2)$$

Xaiver initialization



Definition

z^i is the input vector of layer i , and s^i is the feature vector of the activation function at layer i , we have $s^i = z^i \cdot W^i + b^i$ and $z^{i+1} = f(s^i)$ where f is the symmetric activation function with unit derivative at 0 (i.e., $f'(0) = 1$)

Insight

For the stable mapping from the input domain to the target domain, we want to make the standard derivation of the input domain and the target domain are as equal as possible (i.e., keep the information flowing better in the neural networks)

Forward propagation

$$\text{Var}(z^i) = \text{Var}(x) \cdot \prod_{i'=0}^{i-1} n_{i'} \text{Var}(W^{i'})$$

Backward propagation

$$\text{Var}\left(\frac{\partial \text{Cost}}{\partial s^i}\right) = \text{Var}\left(\frac{\partial \text{Cost}}{\partial s^d}\right) \cdot \prod_{i'=i}^d n_{i'+1} \text{Var}(W^{i'})$$



Xavier initialization

Forward propagation

$$\forall(i), n_i \text{Var}(W^i) = 1$$

Backward propagation

$$\forall(i), n_{i+1} \text{Var}(W^i) = 1$$

As a compromise between two constraints

$$\forall(i), \text{Var}(W^i) = \frac{2}{n_i + n_{i+1}}$$

Assume we use a Uniform distribution

$$W \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

Xavier initialization

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}\right]$$



Insight

W has a symmetric distribution around zero and $E[b] = 0$

Follow the same rule with Xavier initialization

We have

$$\text{Var}(z^i) = \frac{1}{2} \text{Var}(x^i)$$

Kaiming initialization

$$W \sim U\left[-\sqrt{\frac{6}{n_i}}, \sqrt{\frac{6}{n_{i+1}}}\right]$$

Orthogonal Initialization



Insight

Solve the problem of gradient vanishing and exploding in Recurrent Neural Networks

Assume that we are taking a RNN as a form below

A form of a RNN

$$y_t = W^t \cdot y_0$$

We apply orthogonal decomposition to W

We have

$$y_t = Q\Lambda^tQ \cdot y_0$$

Since an orthogonal matrix has the eigenvalue of one, so the gradients would not vanish or explode during the training phase

Overview



CNN Initialization

Gradient Descent Methods

Learning rate annealing

Normalization

Basic gradient descent methods



- ▶ Batch gradient descent
- ▶ Stochastic gradient descent
- ▶ Mini-batch gradient descent

Reference

- ▶ [Sebastian Ruder \(2016\)](#). *An overview of gradient descent optimization algorithms*.
eprint: [arXiv:1609.04747](#)

Batch gradient descent



The gradient of the cost function w.r.t. the parameters θ

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$$

Features

- ▶ The algorithm performs slowly
- ▶ The algorithm cannot update the model online (i.e., with new examples on-the-fly)
- ▶ The datasets cannot fit in memory possibly

Tips

Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces



The gradient of the cost function w.r.t. the parameters θ

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^i, y^i)$$

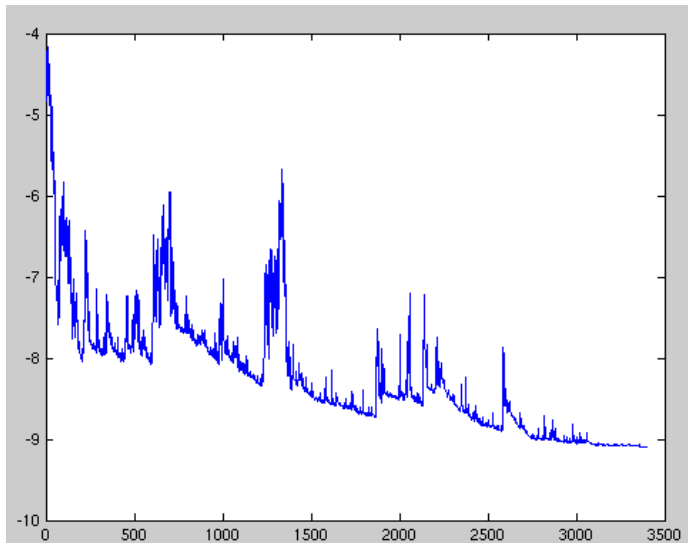
Features

- ▶ The algorithm usually much faster
- ▶ The algorithm can be used to learn online
- ▶ SGD cause the objective function to fluctuate heavily since it performs frequent updates with a high variance

Tips

The fluctuation can help to jump to new and potentially better local minima

Stochastic gradient descent



1

¹By Joe pharos at the English language Wikipedia, CC BY-SA 3.0

Mini-batch gradient descent



The gradient of the cost function w.r.t. the parameters θ

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}, y^{(i:i+n)})$$

Features

- ▶ Reduce the variance of the parameter updates, which can lead to more stable convergence
- ▶ Make use of highly optimized matrix optimizations libraries to accelerate the gradient (i.e., very efficient)

Tips

Most of training methods utilize mini-batch gradient descent method



Challenges

- ▶ It is difficult to decide a proper learning rate
- ▶ It is difficult to decide a proper learning rate schedule
- ▶ It is not reasonable to update all parameters using the same learning rate
- ▶ It is difficult to jump out of local minima and also saddle points

Gradient descent optimization methods



- ▶ Momentum
- ▶ Nesterov
- ▶ Adagrad
- ▶ Adadelata
- ▶ RMSProp
- ▶ Adam

Gradient descent optimization methods contain/adapt materials developed by

- ▶ Ning Qian (1999). “On the momentum term in gradient descent learning algorithms”. In: *Neural networks* 12.1, pp. 145–151
- ▶ Yurii E Nesterov (1983). “A method for solving the convex programming problem with convergence rate $O(1/k^2)$ ”. In: *Dokl. akad. nauk Sssr.* Vol. 269, pp. 543–547
- ▶ Matthew D Zeiler (2012). “Adadelata: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701*
- ▶ John Duchi, Elad Hazan, and Yoram Singer (2011). “Adaptive subgradient methods for online learning and stochastic optimization.”. In: *Journal of machine learning research* 12.7
- ▶ Diederik P Kingma and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*



2

Momentum

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$
$$\theta = \theta - v_t$$

Features

- ▶ The momentum increases for dimensions whose gradients point in the same directions
- ▶ The momentum reduces updates for dimensions whose gradients change directions

Tips

The momentum term γ is usually set to 0.9 or a similar value

²Genevieve B. Orr, Willamette University cs449.



Nesterov

$$\begin{aligned}v_t &= \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t\end{aligned}$$

Features

- ▶ NAG uses predicted update to prevent us from going too fast compared with Momentum
- ▶ It increases the performance of RNNs on a number of tasks



Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$
$$g_{t,i} = \nabla_{\theta_t} \mathbf{J}(\theta_{t,i})$$

Where

$$g_{t,i} = \nabla_{\theta_t} \mathbf{J}(\theta_{t,i})$$

$G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element $G_{t,ii}$ is the sum of squares of the gradients w.r.t. θ_i up to time step t , ϵ is a smoothing term which avoids division by zero (e.g., $1e - 8$)

Features

- ▶ The learning rate η for different parameters can be varied based on the past gradients
- ▶ The accumulated sum of the denominator keeps growing during training, which may cause gradient vanishing

Tips

The initial learning rate η of most applications uses 0.01 as a default value



Adadelta

$$\begin{aligned}\theta_{t+1,i} &= \theta_{t,i} + \Delta\theta_{t,i} \\ \Delta\theta_{t,i} &= -\frac{RMS[\Delta\theta]_{t-1,i}}{RMS[g]_{t,i}}g_{t,i}\end{aligned}$$

where

$$RMS[\Delta\theta]_{t,i} = \sqrt{E[\Delta\theta^2]_{t,i} + \epsilon}$$

and

$$E[\Delta\theta^2]_{t,i} = \gamma E[\Delta\theta^2]_{t-1,i} + (1 - \gamma)\Delta\theta_{t,i}^2$$

Tips

With Adadelta, we do not need to set a default learning rate, since it has been eliminated from the update rule



RMSprop

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma)g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}}g_t$$

Tips

Hinton suggests γ to be set to 0.9 and a good default value for the learning rate η is 0.001³

³https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf



Adam

Where

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Tips

The authors propose the default value of β_1 to be 0.9, 0.999 for β_2 and 10^{-8} for ϵ

Which optimizer to choose?



Choose an optimizer

- ▶ Choose one of the adaptive learning-rate optimization methods usually obtain better performance and you will not need to tune the learning rate with the default value
- ▶ Adam might be the best overall choice
- ▶ Vanilla SGD is robust to different initialization methods and learning rate annealing schedulers

Overview



CNN Initialization

Gradient Descent Methods

Learning rate annealing

Normalization

Different annealing methods



- ▶ Step
- ▶ Cosine
- ▶ Cyclical

Learning rate annealing methods contain/adapt materials developed by

- ▶ Leslie N Smith (2017). “Cyclical learning rates for training neural networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, pp. 464–472
- ▶ Ilya Loshchilov and Frank Hutter (2016). “Sgdr: Stochastic gradient descent with warm restarts”. In: *arXiv preprint arXiv:1608.03983*

Step learning rate scheduler



Step learning rate scheduler

Decay the learning rate by a pre-defined step size exponentially characterized by γ

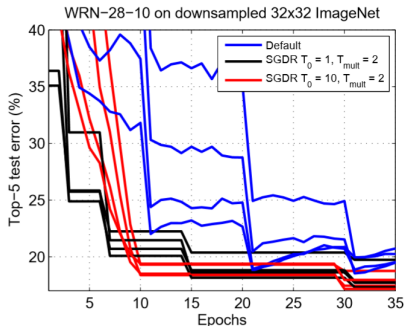
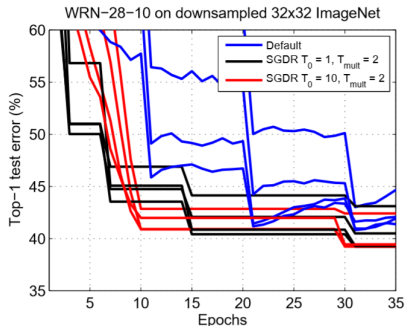
Cosine learning rate scheduler



Cosine annealing method / SGDR (i.e., Stochastic Gradient Descent with warm Restarts)

$$\eta_t = \eta_{min}^t + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i} \pi))$$

Where T_i represents the total epoch numbers



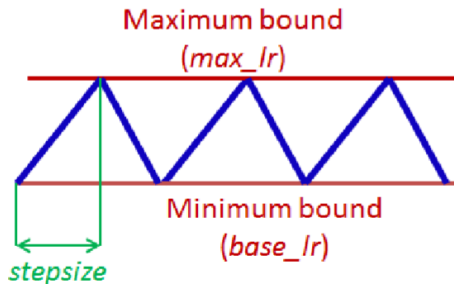
Top-1 and Top-5 test errors obtained by SGD with momentum (i.e., Default) and SGDR on WRN-28-10 trained on a version of ImageNet

Cyclical learning rate scheduler



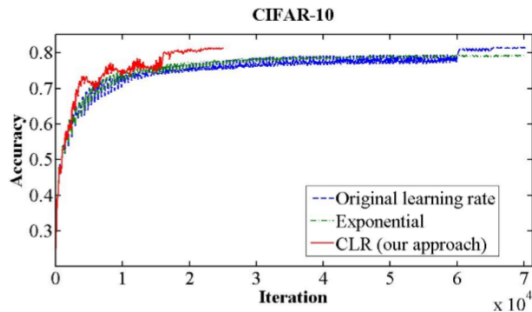
Insight

The essence of this learning rate policy comes from the observation that increasing the learning rate might have a short term negative effect and yet achieve a longer term beneficial effect



Triangular learning rate policy

Cyclical learning rate scheduler

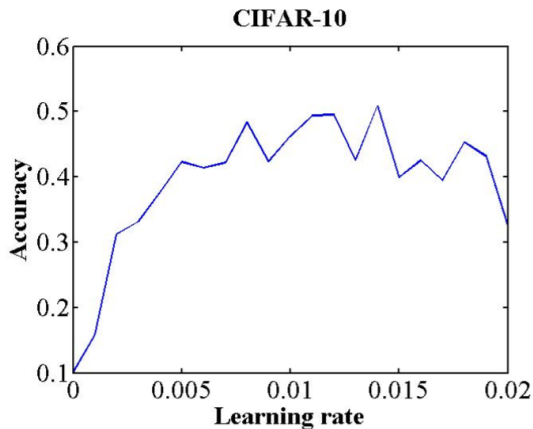


Classification accuracy while training CIFAR-10

Tips

- ▶ Experiments show that it often is good to set **stepsize** equal to 2-10 times the number of iterations in an epoch (e.g., $stepsize = 8 \cdot epoch$)
- ▶ Use **LR range test** to estimate reasonable minimum and maximum boundary values

Cyclical learning rate scheduler



Classification accuracy as a function of increasing learning rate for 8 epochs

The figure shows that the model starts to converge from the beginning, so it is reasonable to set the baseline learning rate to 0.001. Furthermore, when the learning rate is above 0.006, the accuracy rise gets rough and eventually begins to drop so it is reasonable to set the maximum learning rate to 0.006

Overview



CNN Initialization

Gradient Descent Methods

Learning rate annealing

Normalization

Normalization



Reduce **Internal Covariate Shift**, accelerate the training and improve the performance of a neural network

- ▶ Batch normalization
- ▶ Layer normalization
- ▶ Instance normalization
- ▶ Weight normalization

Normalization methods contain/adapt materials developed by

- ▶ [Sergey Ioffe and Christian Szegedy \(2015\)](#). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167*
- ▶ [Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton \(2016\)](#). “Layer normalization”. In: *arXiv preprint arXiv:1607.06450*
- ▶ [Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky \(2016\)](#). “Instance normalization: The missing ingredient for fast stylization”. In: *arXiv preprint arXiv:1607.08022*
- ▶ [Tim Salimans and Durk P Kingma \(2016\)](#). “Weight normalization: A simple reparameterization to accelerate training of deep neural networks”. In: *Advances in neural information processing systems*, pp. 901–909



Internal Covariate Shift

Internal Covariate Shift is as the change in the distribution of network activations due to the change in network parameters during training

Algorithm 1 Batch Normalization Transform

Input: Values of x over a mini-batch: $\mathcal{B} = x_{1,2,\dots,m}$;

Parameters to be learned: γ, β

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

- 1: // mini-batch mean
- 2: $\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i$
- 3: // mini-batch variance
- 4: $\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$
- 5: // normalize
- 6: $\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$
- 7: // scale and shift
- 8: $y_i = \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$



Back propagation with Batch Normalization

Use chain rules to optimize γ and β

$$\begin{aligned}\frac{\partial \ell}{\partial \hat{x}_i} &= \frac{\partial \ell}{\partial y_i} \cdot \gamma \\ \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} &= \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{\frac{-3}{2}} \\ \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} &= \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \\ \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m} \\ \frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i \\ \frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^m \frac{\partial \ell}{\partial y_i}\end{aligned}$$



Algorithm 2 Training a Batch Normalization Network

Input: Network N with trainable parameters Θ ;

subset of activations $x_{k=1}^{kK}$

Output: Batch-normalized network for inference, N_{BN}^{inf}

1: $N_{BN}^{tr} \leftarrow N$ // Training BN network

2: **for** $k = 1$ to K **do**

3: Add transformation $y^{(k)} = BN_{\gamma^k, \beta^{(k)}}(x^{(k)})$ to N_{BN}^{tr} (i.e., Algorithm of Batch Normalization Transform)

4: Modify each layer in N_{BN}^{tr} with input $x^{(k)}$ to take $y^{(k)}$ instead

5: **end for**

6: Train N_{BN}^{tr} to optimize the parameters $\Theta \cup \gamma^k, \beta_{k=1}^{kK}$

7: $N_{BN}^{inf} \leftarrow N_{BN}^{tr}$ // Inference BN network with frozen parameters

8: **for** $k = 1$ to K **do**

9: // For clarity, $x \equiv x^{(k)}$, $\gamma \equiv \gamma^{(k)}$, $\mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$

10: Process multiple training mini-batches \mathcal{B} , each of size m , and average over them:

$$E[x] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$VAR[x] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In N_{BN}^{inf} , replace the transform $y = BN_{\gamma, \beta}(x)$ with $y = \frac{\gamma}{\sqrt{VAR[x] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{VAR[x] + \epsilon}} \right)$

12: **end for**



Weight normalization

$$w = \frac{g}{\|v\|} v$$

The gradients of g and v

$$\begin{aligned}\nabla_g L &= \frac{\nabla_w L \cdot v}{\|v\|} \\ \nabla_v L &= \frac{g}{\|v\|} \nabla_w L - \frac{g \nabla_w L}{\|v\|^2} v\end{aligned}$$

The relationship between Batch normalization

The pre-activation t for each mini-batch in the Batch normalization

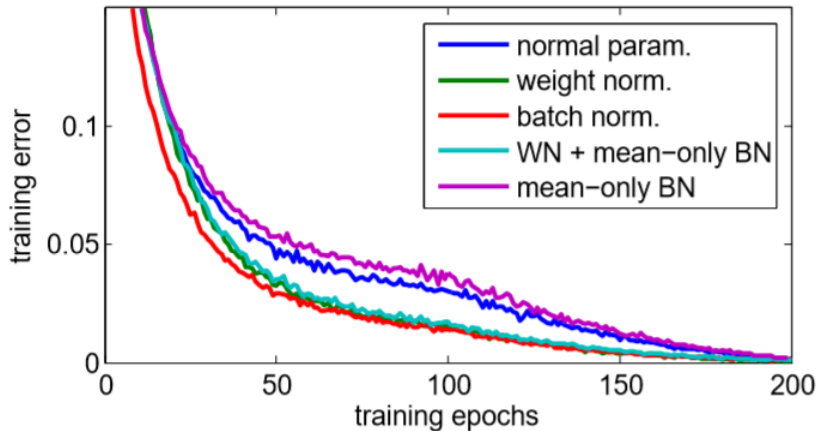
$$t' = \frac{t - \mu[t]}{\sigma[t]}$$

and

$$t = v \cdot x$$

For the special case of a single layer with the input features x that has been whitened (i.e., $\mu[x] = 0$ and $\sigma[x] = 1$) Weight normalization is given by $\mu[t] = 0$ and $\sigma[t] = \|v\|$

Weight normalization



Training error for CIFAR-10 using different network parameterizations



Layer normalization

In a standard RNN, the summed inputs in the recurrent layer can be computed as

$$a^t = W_{hh}h^{t-1} + W_{xh}x^t$$

where x^t is the current input, h^{t-1} is the previous vector of hidden states

$$h^t = f\left[\frac{g}{\sigma^t} \odot (a^t - \mu^t) + b\right]$$

$$\mu^t = \frac{1}{H} \sum_{i=1}^H a_i^t$$

$$\sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^t - \mu^t)^2}$$

where W_{hh} is the recurrent hidden to hidden weights, W_{xh} is the bottom up input to hidden weights, \odot is the element-wise multiplication between two vectors, b and g are defined as the bias and gain parameters of the same dimension as h^t

Features

- ▶ Layer normalization does not calculate the mean and standard deviation based on the mini-batch, which can be utilized in RNN rather than Batch normalization
- ▶ All the hidden units share the same normalization terms μ and σ
- ▶ Layer normalization does not impose any constraint on the size of a mini-batch and it can be used in the pure online regime with batch size 1



Instance normalization

Let $x \in \mathbb{R}^{T \times C \times W \times H}$ and x_{tijk} denote its $tijk$ -th element where j and k span spatial dimensions, i is the feature channel (color channel if the input is an RGB image), and t is the index of the image in the batch. The instance normalization is formulated as

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ii}^2 + \epsilon}}$$
$$\mu_{ti} = \frac{1}{HW} \sum_{l=1}^W \sum_{m=1}^H x_{tilm} = 1^H x_{tilm}$$
$$\sigma_{ii}^2 = \frac{1}{HW} \sum_{l=1}^W \sum_{m=1}^H (x_{tilm} - \mu_{ti})^2$$

Features

- ▶ Instance normalization is used in the style transfer task (i.e., task-specific technique)
- ▶ It is based on Batch normalization

Instance normalization



Stylization examples using instance normalization. First row: style images, second row: original images and its stylized versions