Deep Learning and Its Application to Signal and Image Processing and Analysis

CLASS III - SPRING 2022

TAMMY RIKLIN RAVIV, ELECTRICAL AND COMPUTER ENGINEERING

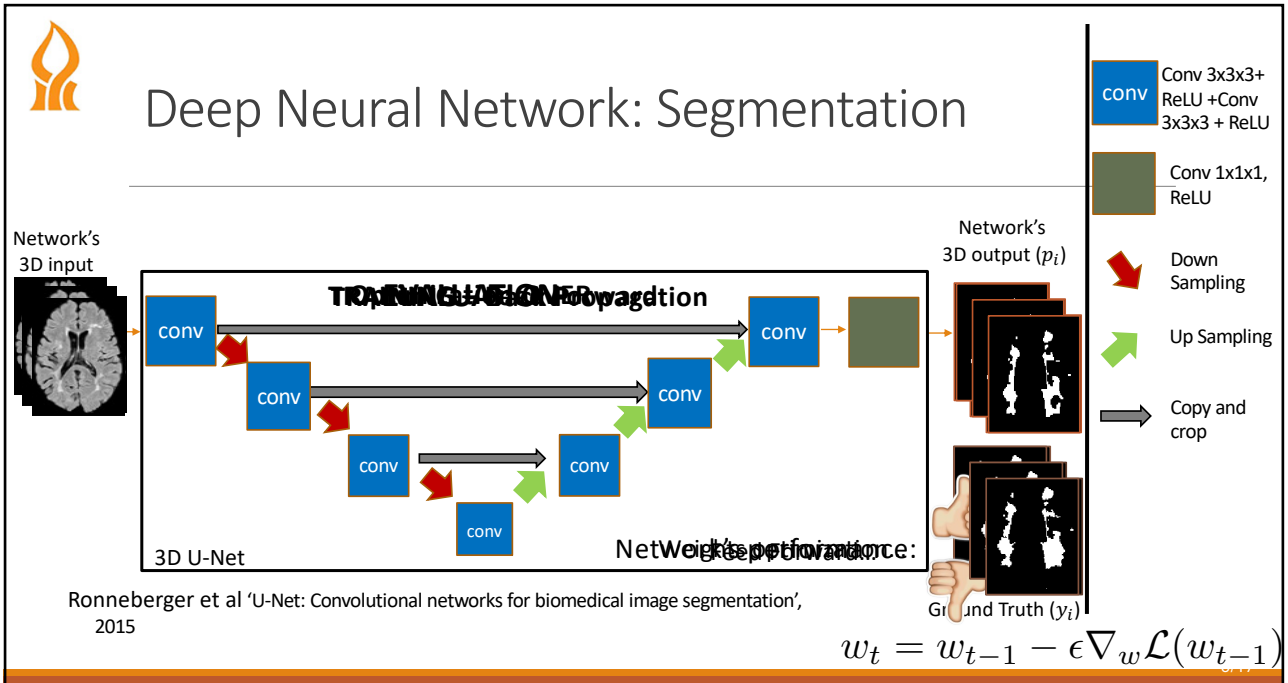
1

Today's topics

- Stochastic Gradient Descent
- Backpropagation
- Vanishing/Exploiding gradients
- Optimization & Optimizers
- Training, Evaluation & Test
- Batch Normalization
- Dropout

2

Deep Neural Network: Segmentation



Ronneberger et al 'U-Net: Convolutional networks for biomedical image segmentation', 2015

$$w_t = w_{t-1} - \epsilon \nabla_w \mathcal{L}(w_{t-1})$$

3

Optimization for training Deep Models

Optimization problem: find the parameters θ that reduce a cost function $J(\theta)$.

Deep Learning algorithms: $J(\theta) = \mathbb{E}_{(x,y)\sim\hat{p}_{\text{data}}} L(f(x;\theta), y)$

- Empirical Distribution
- Per example Loss function
- Predicted output when the input is x

Optimization is performed with respect to the training data

4

Optimization for training Deep Models

Optimization problem: find the parameters θ that reduce a cost function $J(\theta)$.

Deep Learning algorithms: $J^*(\theta) = \mathbb{E}_{(x,y)\sim p_{\text{data}}} L(f(x;\theta), y)$

Data generating distribution

Per example Loss function

Predicted output when the input is x

$\mathbb{E}_{(x,y)\sim \hat{p}_{\text{data}}} L(f(x;\theta), y)$ Empirical risk

5

Empirical Risk Minimization

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

No. of training examples

Main problem: Subject to overfitting

6

Optimization

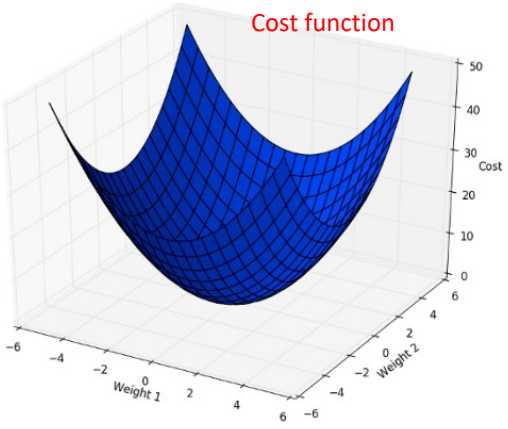
Cost function values: the discrepancies between the outputs (NN estimations) and the training set data points.

Goal : find the set of weights for which a global minimum Is obtained

the cost function is parameterized by the network's weights — we control our loss function by changing the weights.

In reality the cost function is not convex.

Cost function



7

Optimization

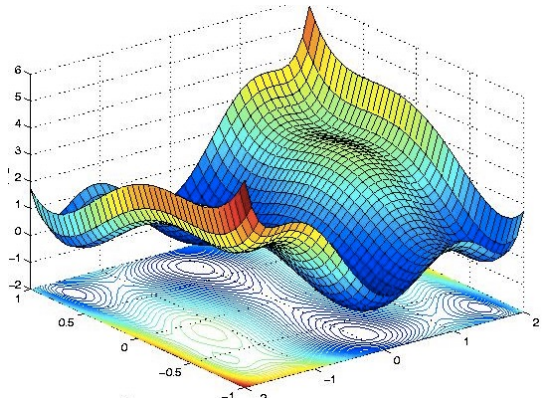
Cost function values: the discrepancies between the outputs (NN estimations) and the training set data points.

Goal : find the set of weights for which a global minimum Is obtained

the cost function is parameterized by the network's weights — we control our loss function by changing the weights.

In reality the cost function is not convex.

Cost function



8

Optimization

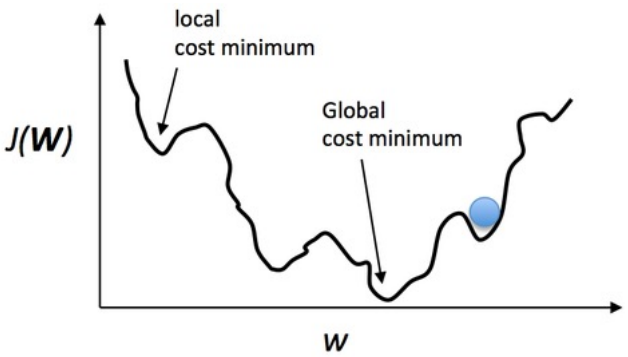
Cost function values: the discrepancies between the outputs (NN estimations) and the training set data points.

Goal : find the set of weights for which a global minimum Is obtained

the cost function is parameterized by the network's weights — we control our loss function by changing the weights.

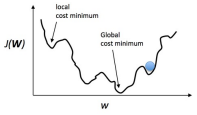
In reality the cost function is not convex.

Cost function



9

Optimization –



Why is it difficult?

1. There is no simple equation that can be solved analytically
2. High-dimensional function
3. Function might have many local minima & maxima

Common approach:

Iterative optimization algorithms, e.g. gradient descent

10

Gradients

1D

Numerical evaluation

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- approximate
- very slow to evaluate

nD

$$\nabla f = \frac{\partial f}{\partial x_1}\mathbf{e}_1 + \cdots + \frac{\partial f}{\partial x_n}\mathbf{e}_n$$

11

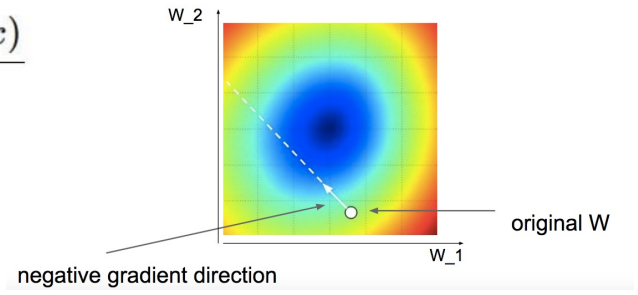
Gradients

1D

Numerical evaluation

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- approximate
- very slow to evaluate



nD

$$\nabla f = \frac{\partial f}{\partial x_1}\mathbf{e}_1 + \cdots + \frac{\partial f}{\partial x_n}\mathbf{e}_n$$

12

Numerical vs. analytic gradient

- Numerical gradient: approximate, slow, easy to write
- Analytic gradient: exact, fast, error-prone

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check.**

13

Gradient descent

$$W := W - \alpha \frac{\partial L(W)}{\partial W}$$ Update rule

$$W := W - \alpha \nabla_W L(W)$$

Learning rate

Learning rate: An important hyperparameter
too small – very slow convergence or gets stuck in local minima
Too Big – may "skip" the target minimum; may go in the wrong direction

14

Backpropagation to train multilayer architectures

The backpropagation procedure to computes the gradient of an objective function with respect to the weights of a multilayer stack of modules.

Practical application of the chain rule.

The key insight is that the gradient of the objective with respect to the input of a module can be computed by working backwards from the gradient with respect to the output of that module (or the input of the subsequent module) .

The backpropagation equation can be applied repeatedly to propagate gradients through all modules, starting from the output at the top (where the network produces its prediction) all the way to the bottom (where the external input is fed).

Once these gradients have been computed, it is straightforward to compute the gradients with respect to the weights of each module.

16

Chain rule of calculus

x - a real number

$f()$ and $g()$ - functions mapping from a real number to a real number.

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

chain rule: $$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

17

Chain rule – vector notation

$\boldsymbol{x} \in \mathbb{R}^m$, $\boldsymbol{y} \in \mathbb{R}^n$ and $z \in \mathbb{R}$

$g : \mathbb{R}^m \to \mathbb{R}^n$ $\quad$ $f : \mathbb{R}^n \to \mathbb{R}$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \right)^\top \nabla_{\boldsymbol{y}} z$$

$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ is the $n \times m$ Jacobian matrix of g

18

Jacobian Matrix

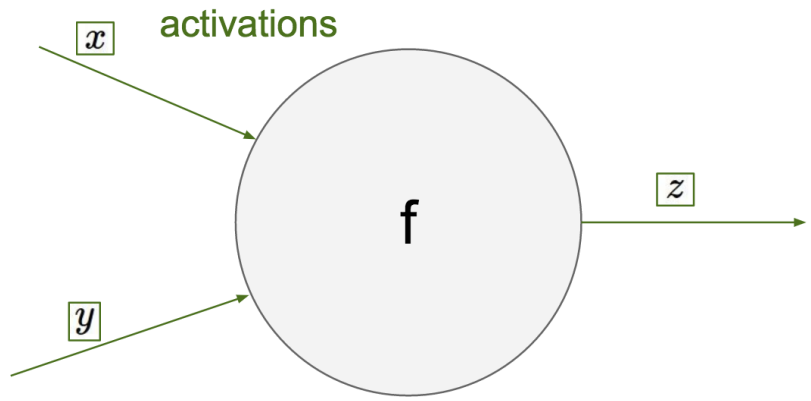
$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

or, component-wise:

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j}.$$

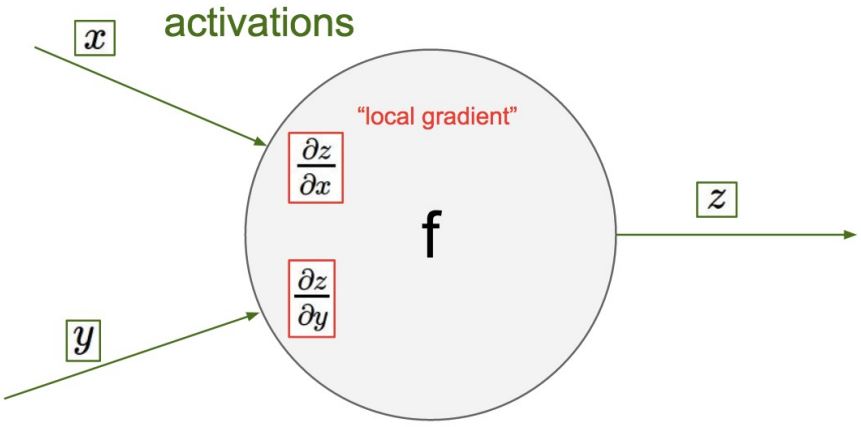
19

Backprop



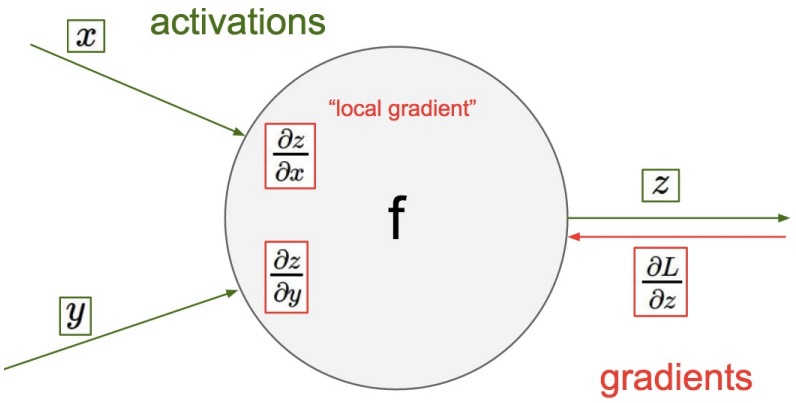
20

Backprop



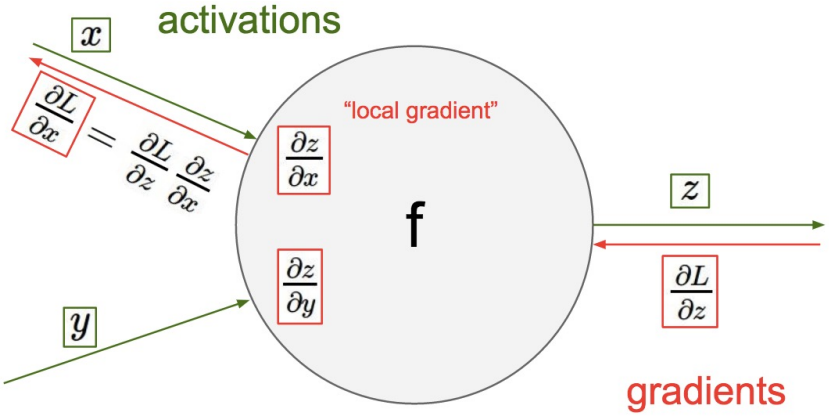
21

Backprop



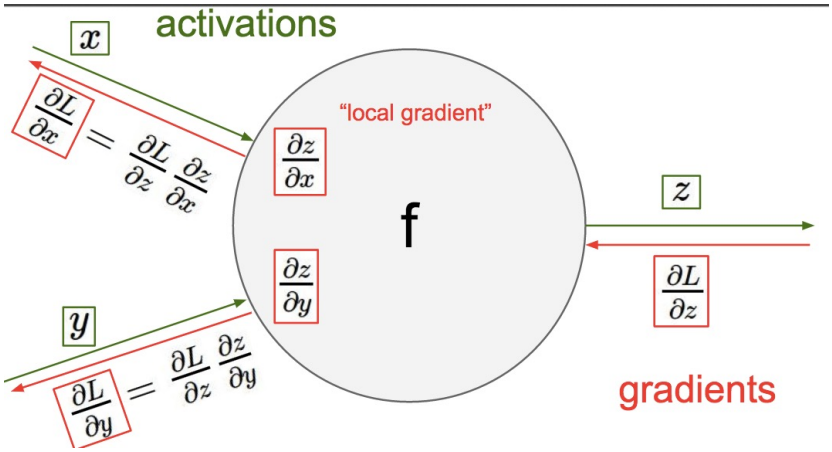
22

Backprop



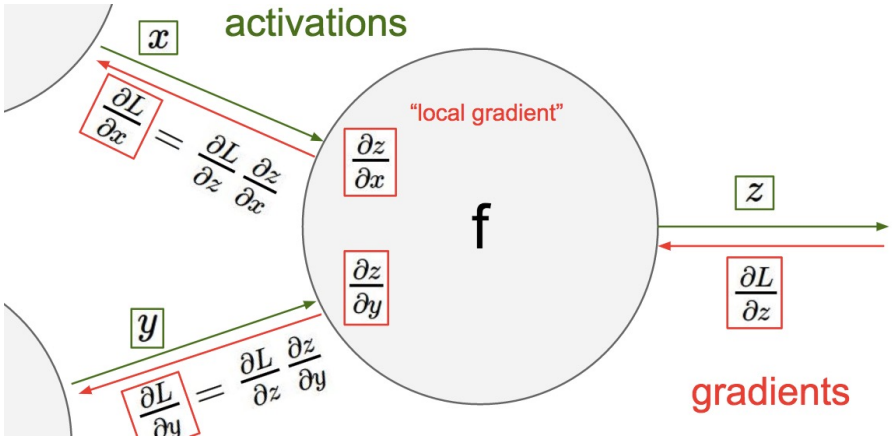
23

Backprop



24

Backprop



25

Backpropagation - Limitations

- Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum
- It has trouble crossing plateaux in the error function landscape.
- This issue, caused by the non-convexity of error functions in neural networks, was long thought to be a major drawback, but in a 2015 review article, Yann LeCun *et al.* (Deep Learning, Nature) argue that in many practical problems, it is not.
- Backpropagation learning does not require normalization of input vectors; however, normalization could improve performance

29

Modes of learning

There are two modes of learning to choose from: stochastic and batch.

Stochastic learning: each propagation is followed immediately by a weight update.

Batch learning: many propagations occur before updating the weights, accumulating errors over the samples within a batch.

Stochastic learning introduces "noise" into the gradient descent process, using the local gradient calculated from one data point; this reduces the chance of the network getting stuck in a local minima. Yet batch learning typically yields a faster, more stable descent to a local minima, since each update is performed in the direction of the average error of the batch samples. In modern applications a common compromise choice is to use "mini-batches", meaning batch learning but with a batch of small size and with stochastically selected samples.

30

Training data collections

Online learning is used for dynamic environments that provide a continuous stream of new training data patterns.

Offline learning makes use of a training set of static patterns.

31

Internal Covariate Shift

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

Batch normalization is a method intended to mitigate internal covariate shift for neural networks.

https://medium.com/analytics-vidhya/internal-covariate-shift-an-overview-of-how-to-speed-up-neural-network-training-3e2a3dcdd5cc

32

Internal Covariate Shift

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

Deeper networks are more affected by the internal covariate shift.

Key idea: stabilize the input values for each layer (defined as **z = Wx + b,** where z is the linear transformation of the W weights/parameters and the biases).

33

Internal Covariate Shift

- **Covariate shift:** when the mini-batches have images that are not uniformly sampled from the entire distribution
- **Solution** for the input layer is to randomize the data before creating mini-batches.

What about the hidden layers?

- In a neural network, each hidden unit's input distribution

changes every time there is a parameter update in the previous layer.

- Called **internal covariate shift**
- Makes training slow and requires a very small learning rate and a good parameter initialization
- Solution: Batch Normalization

https://learnopencv.com/batch-normalization-in-deep-networks/

38

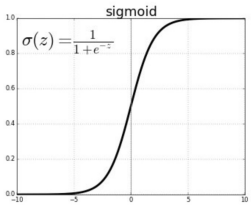
Batch Normalization

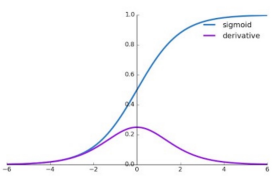
- Introduced by two researchers at Google, Sergey Ioffe and Christian Szegedy in their paper '*Batch Normalization:* ***Accelerating Deep Network Training by Reducing Internal Covariate Shift***' in 2015.
- They showed that batch normalization improved the top result of ImageNet (2014) by a significant margin using only 7% of the training steps
- Today, Batch Normalization is used in almost all CNN architectures.

39

(Other?) benefits to BatchNorm

- Allows higher learning rate without vanishing or exploding gradients.
- Have a regularizing effect such that the network improves its generalization properties
- The network becomes more robust to different initialization schemes and learning rates.





As z increases the derivative goes to zero

40

(Other?) benefits to BatchNorm

- Allows higher learning rate without vanishing or exploding gradients.
- Have a regularizing effect such that the network improves its generalization properties
- The network becomes more robust to different initialization schemes and learning rates.
- While it is widely accepted idea it is unclear how batch normalization relates to internal covariate shift.

How Does Batch Normalization Help Optimization?

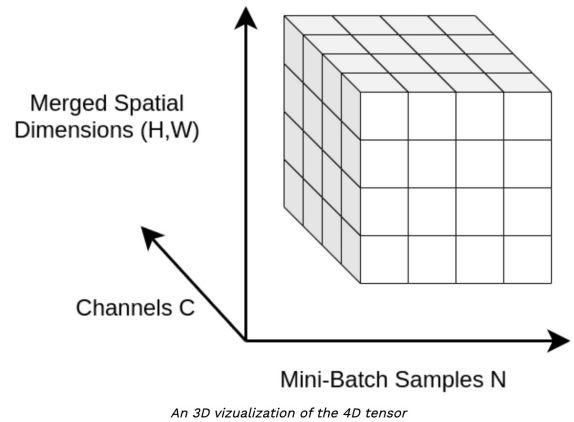
Santurkar et al 2019

https://arxiv.org/pdf/1805.11604.pdf

- Improves optimization landscape

41

In-layer normalization techniques for deep learning



An 3D vizualization of the 4D tensor

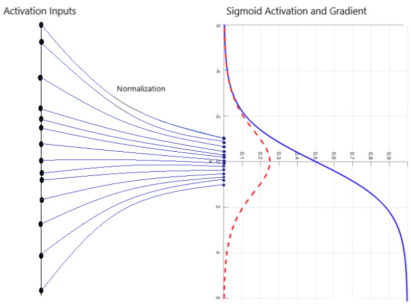
$$x, y, m \in R^{N \times C \times H \times W}$$

Source: https://theaisummer.com/normalization/

42

Batch Normalization

Batch Normalization (BN) normalizes the mean and standard deviation **for each individual feature channel/map.**



bringing the features of the image in the same range.

43

Batch Normalization

we demand from our features to follow a Gaussian distribution with zero mean and unit variance.

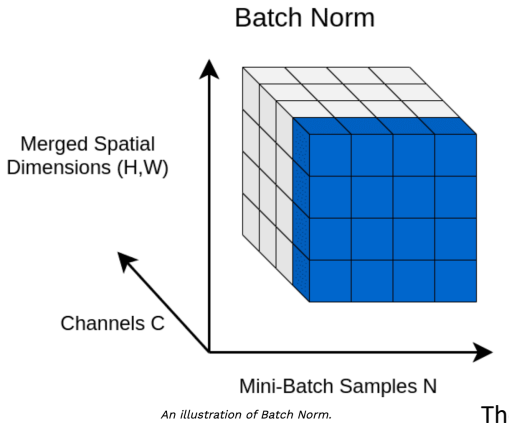
$$BN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

$$\mu_c(x) = \frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_c(x))^2}$$

44

Batch Normalization



An illustration of Batch Norm.

The spatial dimensions, as well as the image batch, are averaged.

45

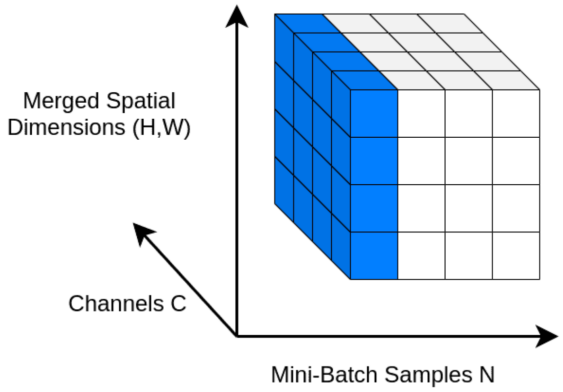
Layer Normalization

In BN, the statistics are computed across the batch and the spatial dims.

In contrast, in **Layer Normalization** (**LN**), the statistics (mean and variance) are **computed across all channels and spatial dims**. Thus, the statistics are independent of the batch. This layer was initially introduced to handle vectors (mostly the RNN outputs).

46

Layer Normalization



$$LN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

$$\mu_n(x) = \frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}$$

$$\sigma_n(x) = \sqrt{\frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_n(x))^2}$$

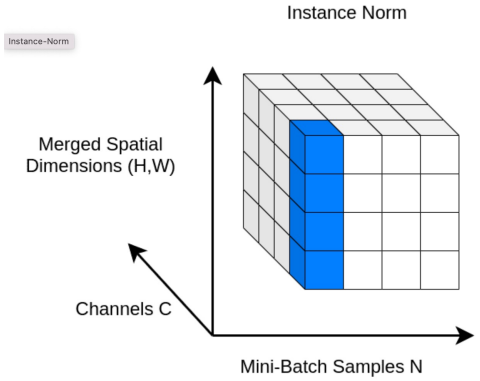
47

Instance Normalization

Instance Normalization (IN) is computed **only across the features' spatial dimensions**. It is independent for each channel and sample.

48

Instance Normalization



$$IN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

$$\mu_{nc}(x) = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{chw}$$

$$\sigma_{nc}(x) = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_c(x))^2}$$

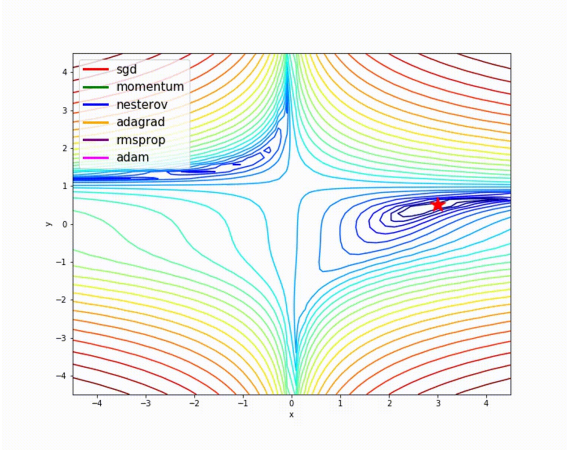
49

Instance Normalization

The affine parameters in **IN can completely change the style** of the output image. As opposed to BN, **IN can normalize the style of each individual sample to a target style** (modeled by γ and β). For this reason, training a model to transfer to a specific style is easier.

50

Which Optimizer to choose?



56

Stochastic Gradient Descent (SGD)

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
$k \leftarrow 1$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
$k \leftarrow k + 1$
end while

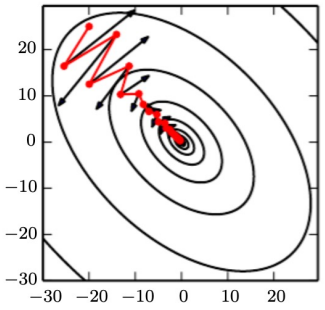
$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \qquad \alpha = \frac{k}{\tau}$$

Itaration

57

Momentum

- Accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.
- Accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
- Addresses **ill-conditioning, i.e.,** when SGD gets "stuck" in the sense that even very small steps increase the cost function



58

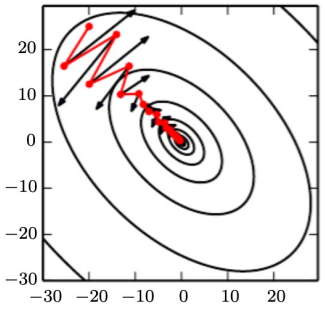
Momentum

Velocity

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

$\alpha \in [0, 1)$ Hyperparameter - determines how quickly the contributions of previous gradients exponentially decay



59

SGD with Momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
end while

The size of the step depends on how large and how aligned a sequence of gradients are

60

Nesterov Momentum

If the value of the momentum is too high, our model can even cross the minima and again start going up.

Solution: take into account the gradient in the next time step which gives a better idea whether to slow down or not.

61

Nesterov Momentum

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v},$$

- The gradient is evaluated after the current velocity is applied
- Add a correction factor to the standard method of momentum.
- Improves the rate of convergence in Convex batch gradient case

62

SGD with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
end while

63

Algorithms with Adaptive Learning Rates

- Assumption: directions of sensitivity are somewhat axis aligned
- Use a separate learning rate for each parameter
- Automatically adapt these learning rates throughout the course of learning

64

AdaGrad (Duchi et al.,2011)

- **Individually** adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient
- Parameters with the **largest** partial derivative of the loss have correspondingly **rapid decrease** in their learning rate
- Parameters with small partial derivative of the loss have correspondingly **small decrease** in their learning rate
- The net effect is greater progress in the more gently sloped directions of parameter space
- Can be inefficient

65

The AdaGrad Algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

66

RMSProp (Hinton, 2012)

- AdaGrad is designed to converge rapidly when applied to a convex function.
- When applied to a nonconvex function to train a neural network, the learning trajectory may pass through many different structures and eventually arrive at a region that is a locally convex bowl.
- AdaGrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure

The

- RMSProp algorithm modifies AdaGrad to perform better in the nonconvex setting

67

RMSProp (Hinton, 2012)

- Changes the gradient accumulation into an exponentially weighted moving average.
- Discard history from the extreme past so that it can converge rapidly after finding a convex bowl, as if it were an instance of the AdaGrad algorithm initialized within that bowl.

The

- RMSProp algorithm modifies AdaGrad to perform better in the **nonconvex setting**

68

RMSProp Algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
end while

69

RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
end while

70

Adam Optimizer (Adaptive Moment)

- A variant on the combination of RMSProp and momentum with a few important distinctions.
- Momentum is incorporated directly as an estimate of the first-order moment (with exponential weighting) of the gradient.
- Momentum is applied to the rescaled gradients
- Includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin

71

ADAM Algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

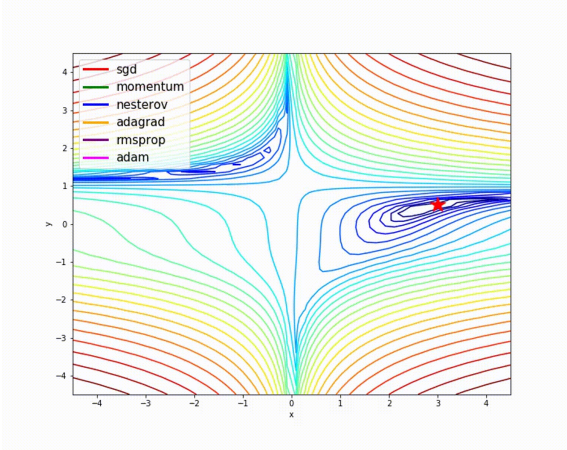
72

NADAM

Nadam is an extension of the Adam version of gradient descent that incorporates Nesterov momentum

73

Which Optimizer to choose?



74