Advanced optimization

Machine Learning

Natalia Khanzhina

07.10.2019

Lecture plan

- Vanishing/exploding gradients
- Activation functions for DNNs
- Data preprocessing for DNNs
- Improving descent for DNNs
- Batch normalization

- The presentation is prepared with materials of
  - K.V. Vorontsov's course "Machine Leaning",
  - D. Polykovsky and K. Khrabrov "Neural networks in machine learning".

Lecture plan

- Vanishing/exploding gradients
- Activation functions for DNNs
- Data preprocessing for DNNs
- Improving descent for DNNs
- Batch normalization

Example

- Imagine we have a deep feedforward network with d layers
- Each derivative on each level will result into

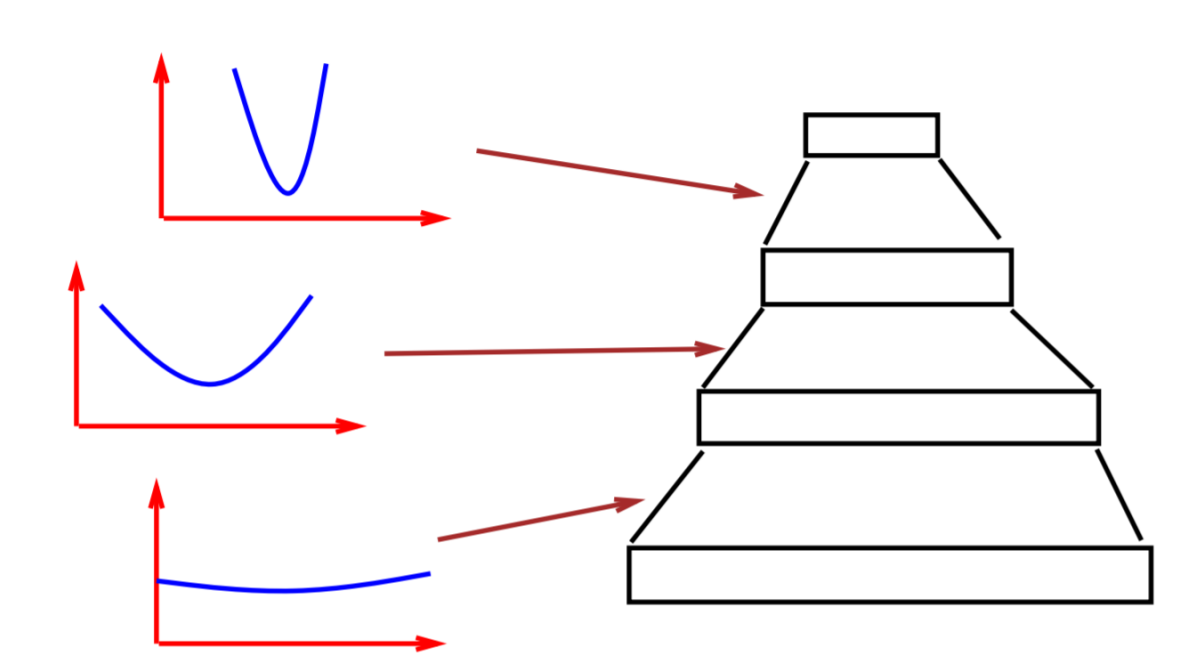
$$\frac{\partial L(w)}{\partial u_d} = \frac{\partial L(w)}{\partial a} \cdot \frac{\partial a}{\partial u_d} = (y - a)\sigma'(w_d u_d) w_d \leq 2 \cdot \frac{1}{4} w_d$$

$$\frac{\partial L(w)}{\partial u_{d-1}} = \frac{\partial L(w)}{\partial u_d} \cdot \frac{\partial u_d}{\partial u_{d-1}} \leq 2 \cdot \left(\frac{1}{4}\right)^2 w_d w_{d-1}$$

It either vanishes or explodes.

Second order methods?

- Second derivative is smaller on lower layers



- There are some improvements

Lecture plan

- Vanishing/exploding gradients
- Activation functions for DNNs
- Data preprocessing for DNNs
- Improving descent for DNNs
- Batch normalization

Tanh

- Activation function $a = \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- Gradient with respect to the input

$$\frac{\partial a}{\partial x} = 1 - \tanh^{2}(x)$$

- Similar to sigmoid, but with different output range $[-1, +1]$
- Stronger gradients, because data is centered around 0 (not 0.5)
- Less bias to hidden layer neurons as now outputs can be both positive and negative (more likely to have zero mean in the end)

ReLU

- Activation function a $= h(x) = \max(0, x)$
- Gradient with respect to the input

$$\frac{\partial a}{\partial x} = \begin{cases} 1, \text{if } x > 0, \\ 0, \text{otherwise.} \end{cases}$$

- Very popular in computer vision and speech recognition

ReLU analysis

- Much faster computations, gradients
- No vanishing or exploding problems, only comparison, addition, multiplication
- People claim biological plausibility
- Sparse activations
- No saturation

- Non-symmetric
- Non-differentiable at 0
- A large gradient during training can cause a neuron to "die". Higher learning rates mitigate the problem

Softplus

Soft approximation (softplus):

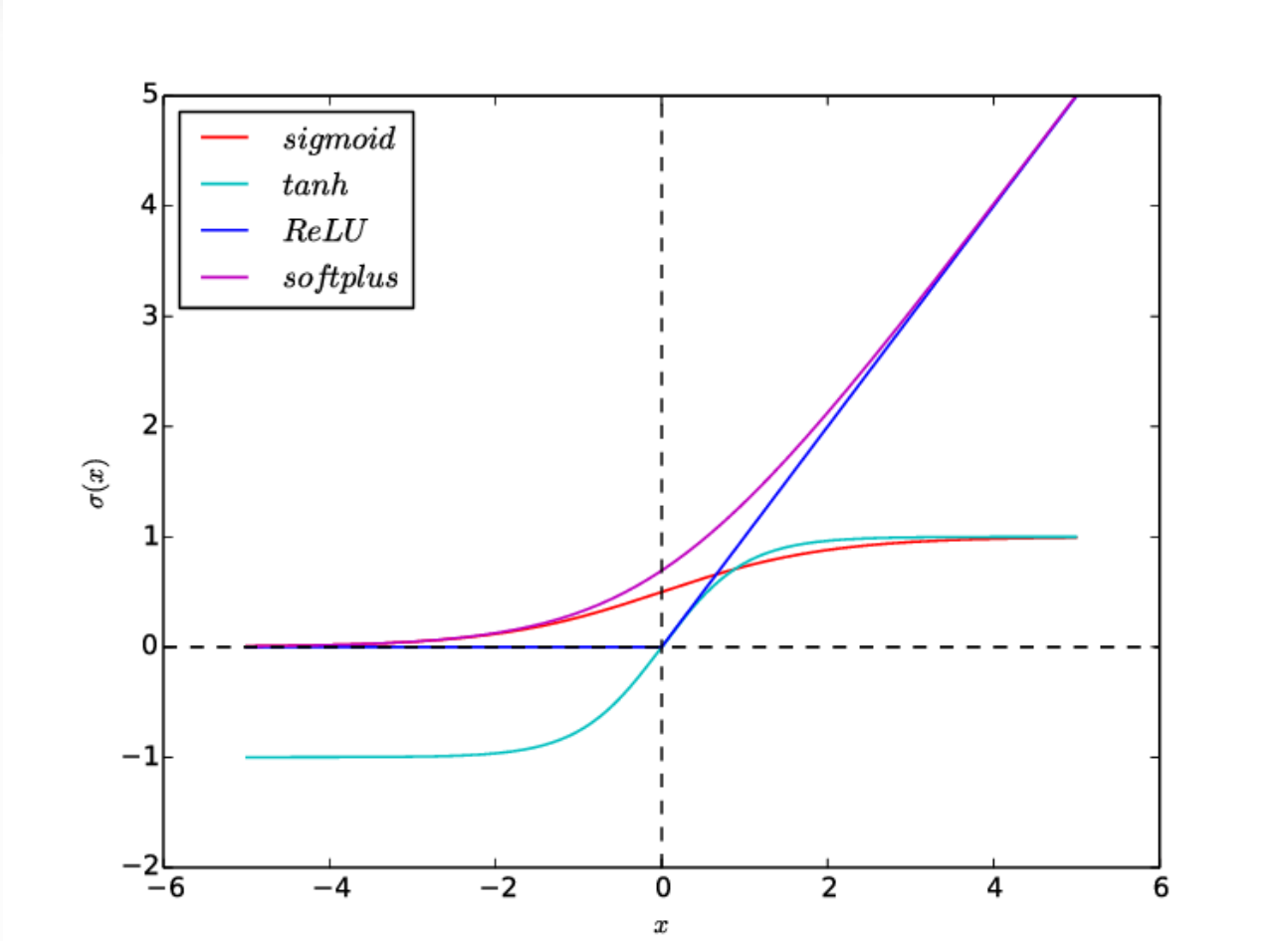
$$a = h(x) = \ln(1 + e^x)$$

- Differentiable at 0

- Slower
- Empirically, do not outperforms ReLU

10

Activation functions in one plot



11

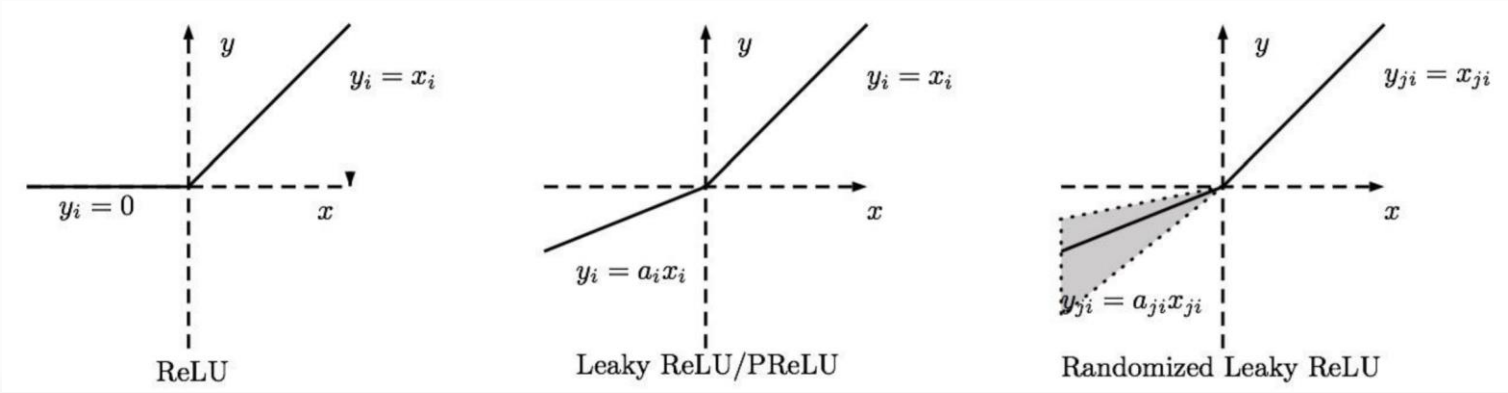
Other ReLUs

Noisy ReLU: $h(x) = \max(0, x + \varepsilon), \varepsilon \sim N(0, \sigma(x))$

Leaky ReLU: $h(x) \begin{cases} x, \text{if } x > 0, \\ 0.01x, \text{otherwise.} \end{cases}$

Parametric ReLu: $h(x) = \begin{cases} x, \text{if } x > 0 \\ \beta x, \text{otherwise} \end{cases}$

(parameter β is trainable)



12

Lecture plan

- Vanishing/exploding gradients
- Activation functions for DNNs
- Data preprocessing for DNNs
- Improving descent for DNNs
- Batch normalization

Data preprocessing

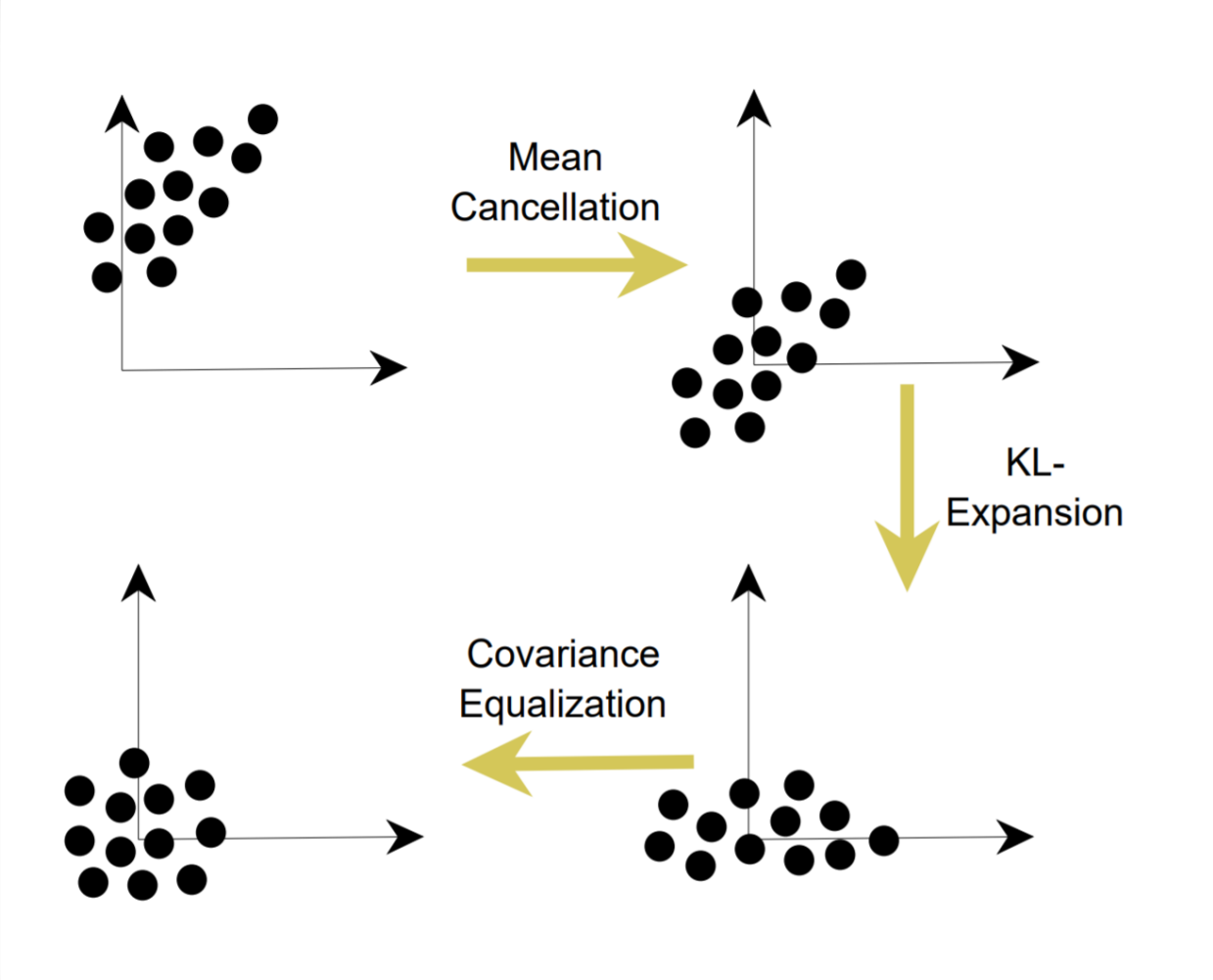
Data preprocessing is useful in general and is very important for deep learning optimization. Three main steps:

Types of data augmentation:

- Mean cancellation (centers data)
- Decorrelation [Karhunen-Loeve expansion]
- Scaling

14

Data preprocessing



15

Decorrelation

Covariance matrix: $\mathrm{Cov}(X) = \frac{1}{N} X X^{\mathrm{T}}$

Decorrelation: $\hat{X} = \mathrm{Cov}^{-1/2}(X) \cdot X$

$\mathrm{Cov}(\hat{X}) = I$

16

Initial weights selection

- Selection of weights is important to the quality of solution and even convergence of descent.
- Typical scenario is to initialize weights with something small random

Xavier motivation

- Assume we have activation function f, which is linear nearby 0:

$$f(x) = x$$

- tanh is an example of such function

Main idea is to put weights in such linear region and maintain variance to be constant

Evaluating variance (1/2)

$$u_{d+1} = f(u_d w_d) \approx u_d w_d$$

$$\mathrm{D}(u_{d+1,k}) = \mathrm{D}\left(\sum_{i=1}^{n_d} u_{d,i} w_{d,i,k}\right) = \sum_{i=1}^{n_d} \mathrm{D}(u_{d,i} w_{d,i,k})$$

n_d is a number of neurons at dth layer

We can assume that they are independent

$$\mathrm{D}(u_{d+1,k}) = n_i \mathrm{D}(u_{d,i} w_{d,i,k}) =$$

$$= n_i \left(\mathrm{E}(u_{d,i}^2)\mathrm{E}(w_{d,i,k}^2) - \mathrm{E}^2(u_{d,i})\mathrm{E}^2(w_{d,i,k})\right) =$$

$$n_i \mathrm{D}(u_{d,i})\mathrm{D}(w_{d,i,k})$$

Evaluating variance (2/2)

$$\mathrm{D}(u_{d+1}) = \mathrm{D}(x) \prod_{j=1}^{d} n_j \, \mathrm{D}(w_j)$$

$$\mathrm{D}\left(\frac{\partial L}{\partial u_d}\right) = \mathrm{D}\left(\frac{\partial L}{\partial u_N}\right) \prod_{j=d}^{N} n_{j+1} \, \mathrm{D}(w_j)$$

Our requirements $\forall d, h \leq N$:

$$\mathrm{D}(u_d) = \mathrm{D}(u_h)$$

$$\mathrm{D}\left(\frac{\partial L}{\partial u_d}\right) = \mathrm{D}\left(\frac{\partial L}{\partial u_h}\right)$$

Xavier

$\mathrm{D}(u_d) = \mathrm{D}(u_h), \mathrm{D}\left(\frac{\partial L}{\partial u_d}\right) = \mathrm{D}\left(\frac{\partial L}{\partial u_h}\right)$ is equivalent to

$$\forall d \begin{cases} n_d D(w_d) = 1 \\ n_{d+1} D(w_d) = 1 \end{cases}$$

Trade off: $D(w_d) = \frac{2}{n_d + n_{d+1}}$

$$w_d \sim U\left[\frac{-\sqrt{6}}{n_d + n_{d+1}}, \frac{\sqrt{6}}{n_d + n_{d+1}}\right].$$

What to do with ReLU? He

$$\mathrm{D}(u_{d+1}) = \mathrm{D}(x)\prod_{j=1}^{d}\frac{1}{2}n_j\,\mathrm{D}(w_j)$$

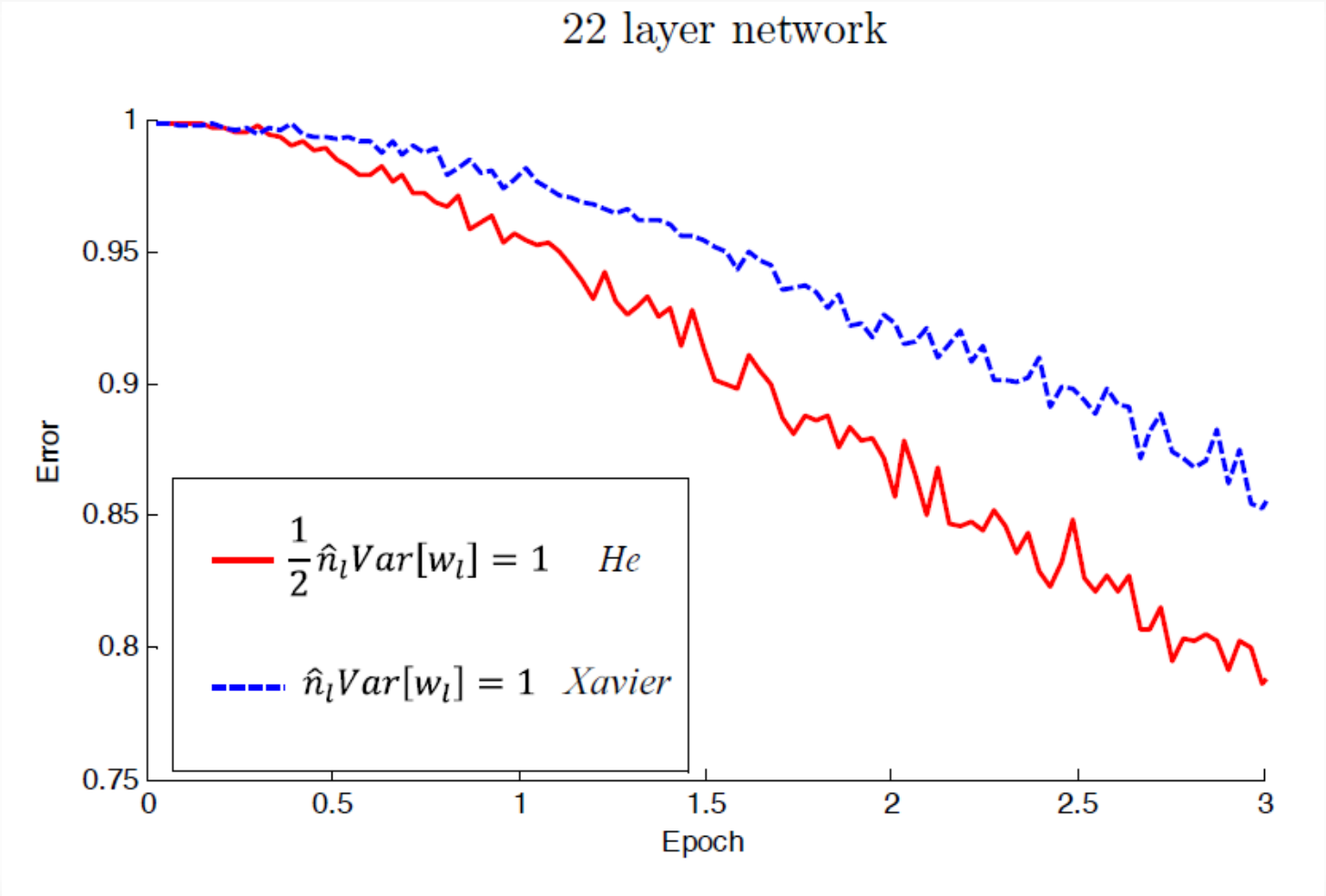
$$\mathrm{D}\left(\frac{\partial L}{\partial u_d}\right) = \mathrm{D}\left(\frac{\partial L}{\partial u_N}\right)\prod_{j=d}^{N}\frac{1}{2}n_{j+1}\,\mathrm{D}(w_j)$$

$$\forall d \begin{cases} n_d D(w_d) = 1/2 \\ n_{d+1} D(w_d) = 1/2 \end{cases}$$

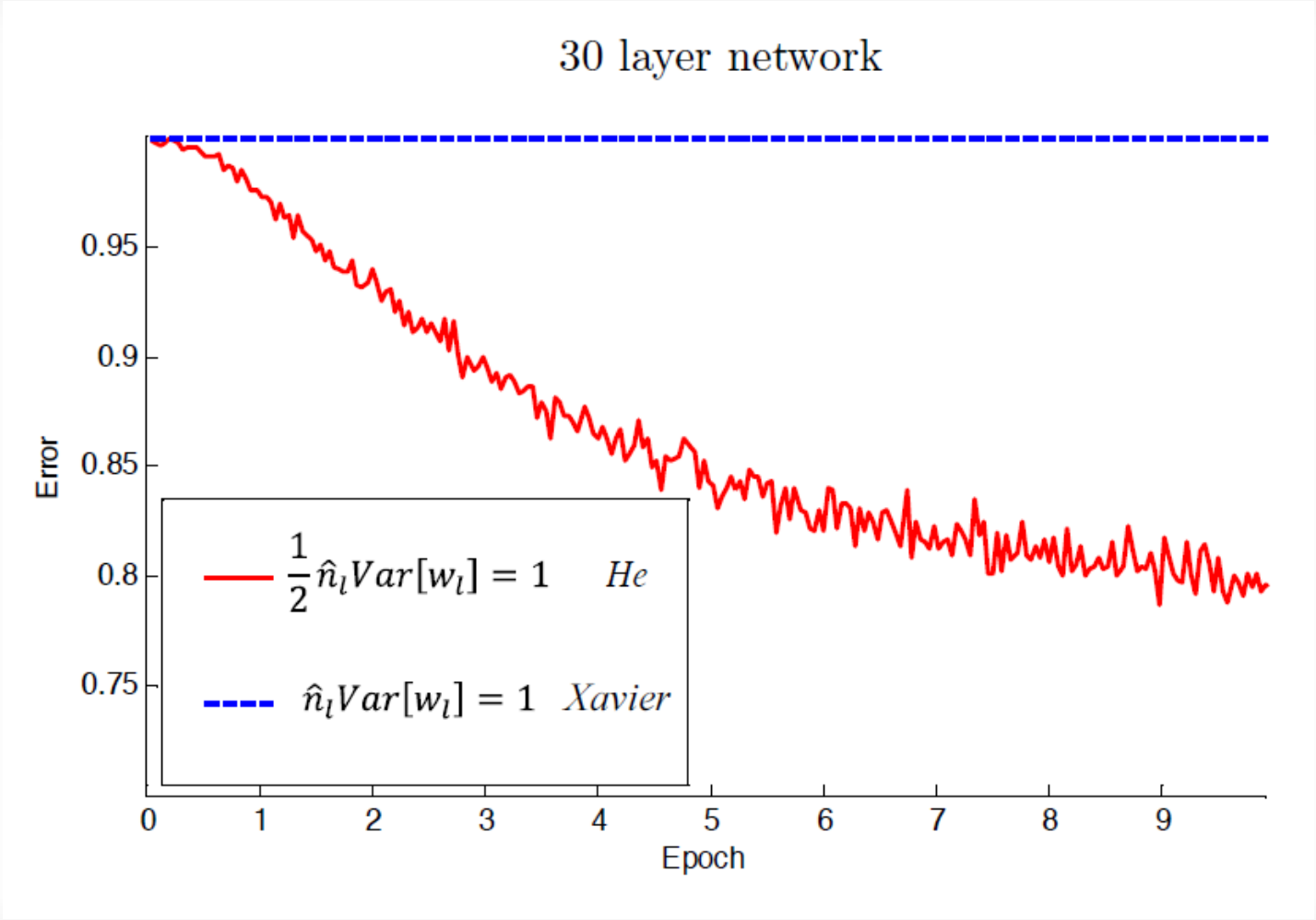
Gaussian is often used:

$$w_d \sim N\left[0, \frac{2}{n_d}\right] \text{ or } w_d \sim N\left[0, \frac{2}{n_{d+1}}\right]$$

Xavier vs He (1/2)



Xavier vs He (2/2)



Lecture plan

- Vanishing/exploding gradients
- Activation functions for DNNs
- Data preprocessing for DNNs
- Improving descent for DNNs
- Batch normalization

Stochastic gradient descent (reminder)

Stochastic gradient descent:

$w^{(0)}$ is an initial guess values

$$w^{(k+1)} = w^{(k)} - \mu \frac{\partial L(w^{(k)})}{\partial w}$$

Momentum

Momentum:

$w^{(0)}$ is an initial guess values;

ν are updates:

$$w^{(k+1)} = w^{(k+1)} - \nu^{(k+1)}$$

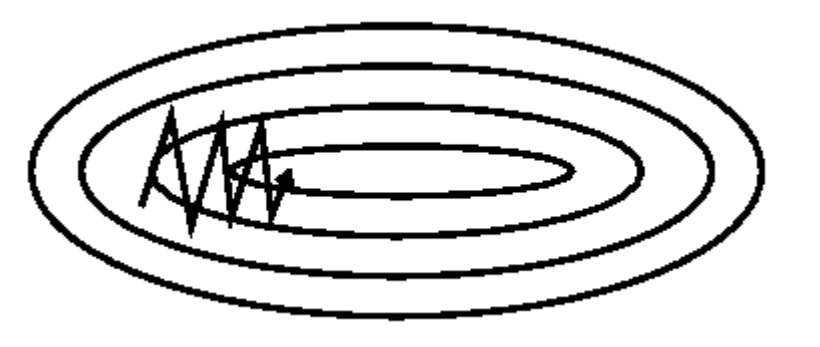
$$\nu^{(k+1)} = \gamma\nu^{(k)} + \mu\frac{\partial_L(w^{(k)})}{\partial w},$$

γ is momentum, usually set to 0.9

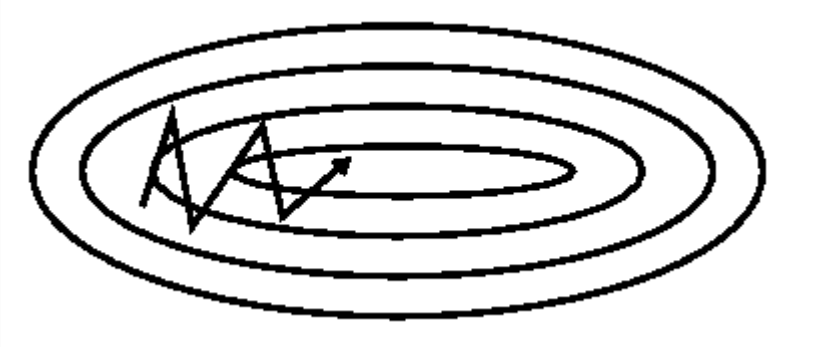
Momentum discussion

Advantages:

- in general is faster in complex terrain when moving in right direction



without Momentum



with Momentum

Disadvantages:

- may fly over minima

Nesterov accelerated gradient

NAG:

$w^{(0)}$ is an initial guess values;

ν are updates:

$$w^{(k+1)} = w^{(k+1)} - \nu^{(k+1)}$$

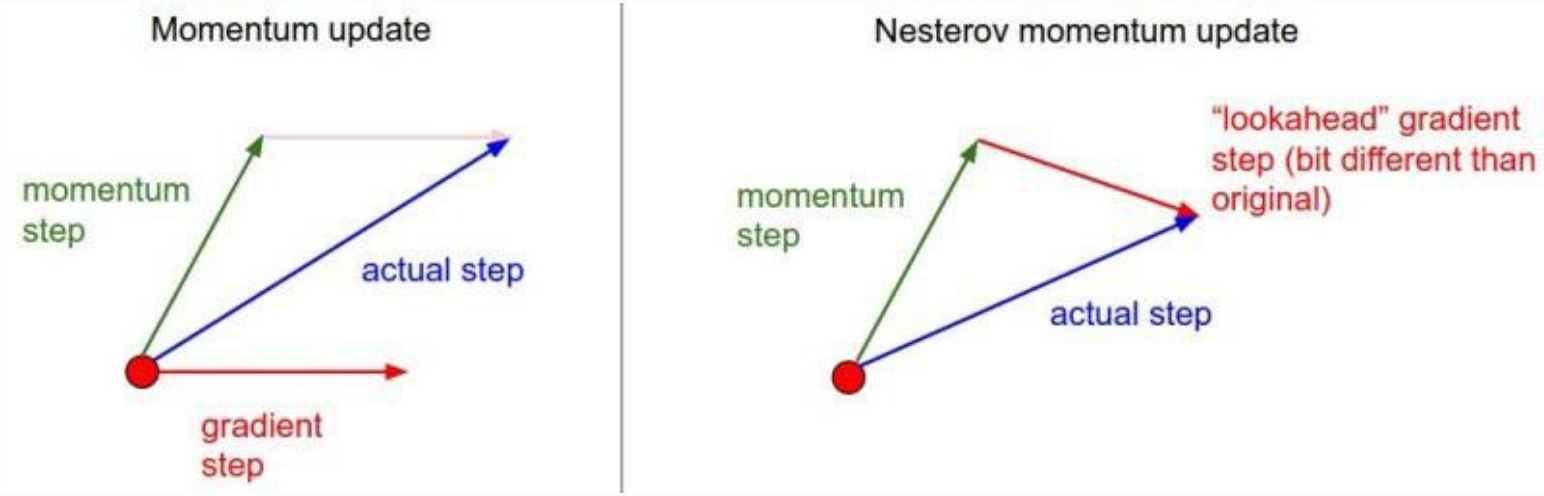
$$\nu^{(k+1)} = \gamma\nu^{(k)} + \mu\frac{\partial L\left(w^{(k)} - \nu^{(k)}\right)}{\partial w},$$

γ is momentum, usually set to 0.9

NAG discussion

Advantages:

- In general, works better
- Convergence proven in certain conditions



- How to choose learning rate?

Adagrad

$$g_{i,(k)} = \frac{\partial L\left(w_i^{(k)}\right)}{\partial w_i}.$$

Adagrad:

$w^{(0)}$ is an initial guess values;

for each i

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\mu}{\sqrt{G_{i,i}^{(k)} + \varepsilon}} g_{i,(k)},$$

where G is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients $g_{i,(k)}$ up to time step k and

ε is a smoothing term that avoids division by zero.

Adagrad discussion

Advantages:

- Eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that.

Disadvantages:

- Accumulation of the squared gradients in the denominator leads to the sum keeping growing during training. Eventually, algorithm stops to learn anything.

RMSProp

$$E^{(k)}[g_i^2] = \gamma E^{(k-1)}[g_i^2] + (1-\gamma) g_{i,(k)}^2$$

RMSProp:

$w^{(0)}$ is an initial guess values;

for each i

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\mu}{\sqrt{E^{(k)}[g_i^2]+\varepsilon}} g_{i,(k)},$$

where ε is a smoothing term that avoids division by zero.

Set γ to be 0.9

Adadelta (1/3)

$$w^{(k+1)} = w^{(k)} - \mu \left(Q''\left(w^{(k)}\right) \right)^{-1} Q'\left(w^{(k)}\right)$$

$$w^{(k+1)} = w^{(k)} + \Delta w^{(k)}$$

$\left(Q''\left(w^{(k)}\right) \right)^{-1}$ is hard to evaluate, so let think it is diagonal

$$\left(Q''\left(w^{(k)}\right) \right) \approx \text{diag}\left(\frac{\partial Q^2\left(w_i^{(k)}\right)}{\partial w_i^2} \right)$$

$$\Delta w_i^{(k)} \approx \left(\frac{\partial Q^2\left(w_i^{(k)}\right)}{\partial w_i^2} \right)^{-1} \left(\frac{\partial Q\left(w_i^{(k)}\right)}{\partial w_i} \right)$$

$$\frac{\partial Q^2\left(w_i^{(k)}\right)}{\partial w_i^2} \approx \frac{\left(\frac{\partial Q\left(w_i^{(k)}\right)}{\partial w_i} \right)}{\Delta w_i^{(k)}}$$

Adadelta (2/3)

$$E^{(k)}[g_i^2] = \gamma E^{(k-1)}[g_i^2] + (1-\gamma) g_{i,(k)}^2$$

$$RMS^{(k)}[g_i] = \sqrt{E^{(k)}[g_i^2] + \varepsilon}$$

$$RMS^{(k)}[\Delta w_i] = \sqrt{E^{(k)}\left[\Delta w_i^2\right] + \varepsilon}$$

$$\frac{\partial Q^2\left(w_i^{(k)}\right)}{\partial w_i^2} \approx \frac{\left(\frac{\partial Q\left(w_i^{(k)}\right)}{\partial w_i}\right)}{\Delta w_i^{(k)}} \approx \frac{g_i^{(k)}}{\Delta w_i^{(k-1)}} = \frac{RMS^{(k)}[g_i]}{RMS^{(k-1)}[\Delta w_i]}.$$

Adadelta (3/3)

Adadelta:

$w^{(0)}$ is an initial guess values;

for each i

$$w_i^{(k+1)} = w_i^{(k)} - \frac{RMS^{(k-1)}[\Delta w_i]}{RMS^{(k)}[g_i]} g_i^{(k)}$$

No learning rate is required!

In practice, learning rate is still added to improve performance.

Adam (Adaptive Moment Estimation)

$$m_{(k)} = E^{(k)}[g_i] = \gamma_1 E^{(k-1)}[g_i] + (1-\gamma_1) g_{i,(k)}$$
$$b_{(k)} = E^{(k)}[g_i^2] = \gamma_2 E^{(k-1)}[g_i^2] + (1-\gamma_2) g_{i,(k)}^2$$

We want them to be unbiased:

$$\mathrm{E}(m_{(k)}) = \mathrm{E}(g_{(k)}), \mathrm{E}(b_{(k)}) = \mathrm{E}(g_{(k)}^2)$$

To satisfy it, we need amendment:

$$\begin{cases} \widehat{m}_{(k)} = \dfrac{m_{(k)}}{1-\gamma_1^k} \\ \widehat{b}_{(k)} = \dfrac{b_{(k)}}{1-\gamma_2^k} \end{cases}$$

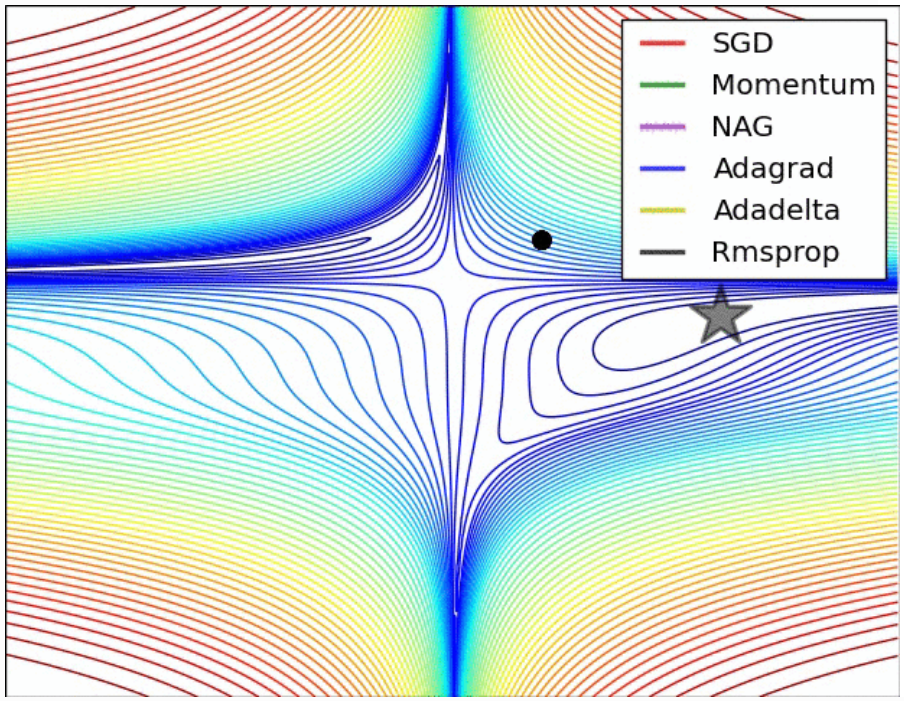
Adam (Adaptive Moment Estimation)

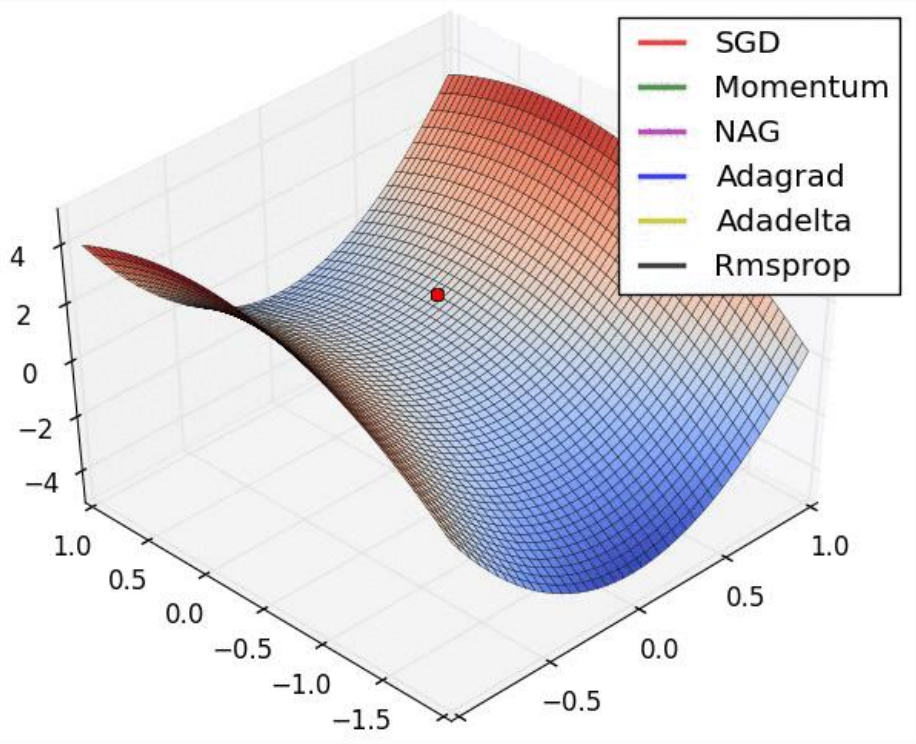
Adam:

$w^{(0)}$ is an initial guess values

$$w^{(k)} = w^{(k)} - \frac{\mu}{\sqrt{\hat{b}_{(k)}^2 + \varepsilon}} \hat{m}_{(k)}$$

Comparison


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Additional steps

- Shuffling and Curriculum Learning
- Early stopping
- Gradient noise
- Batch normalization

Lecture plan

- Vanishing/exploding gradients
- Activation functions for DNNs
- Data preprocessing for DNNs
- Improving descent for DNNs
- Batch normalization

Layer-wise SGD problem

After updating weights, domains are updates

Main idea: maintain covariance constant for each layer input:

$$\hat{x}_d = \frac{x_d - \mathrm{E}(x_i)}{\sqrt{\mathrm{D}(x_d) + \varepsilon}}$$

E and D should be evaluated on each mini-batch

Parametric layer for batch normalization

Add a parametric layer with rescaling:

$$\hat{y}_d = \gamma_d \hat{x}_d + \beta_d$$

γ and β can be learned

Batch normalization algorithm

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

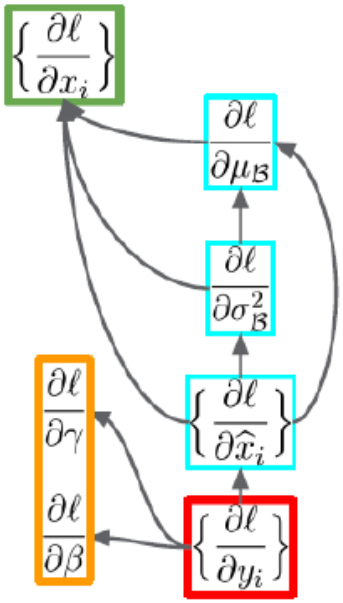
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Gradient descent for BN

We will learn BN layer as a layer



$$\frac{\partial \ell}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma^2_{\mathcal{B}} + \epsilon)^{-3/2}$$

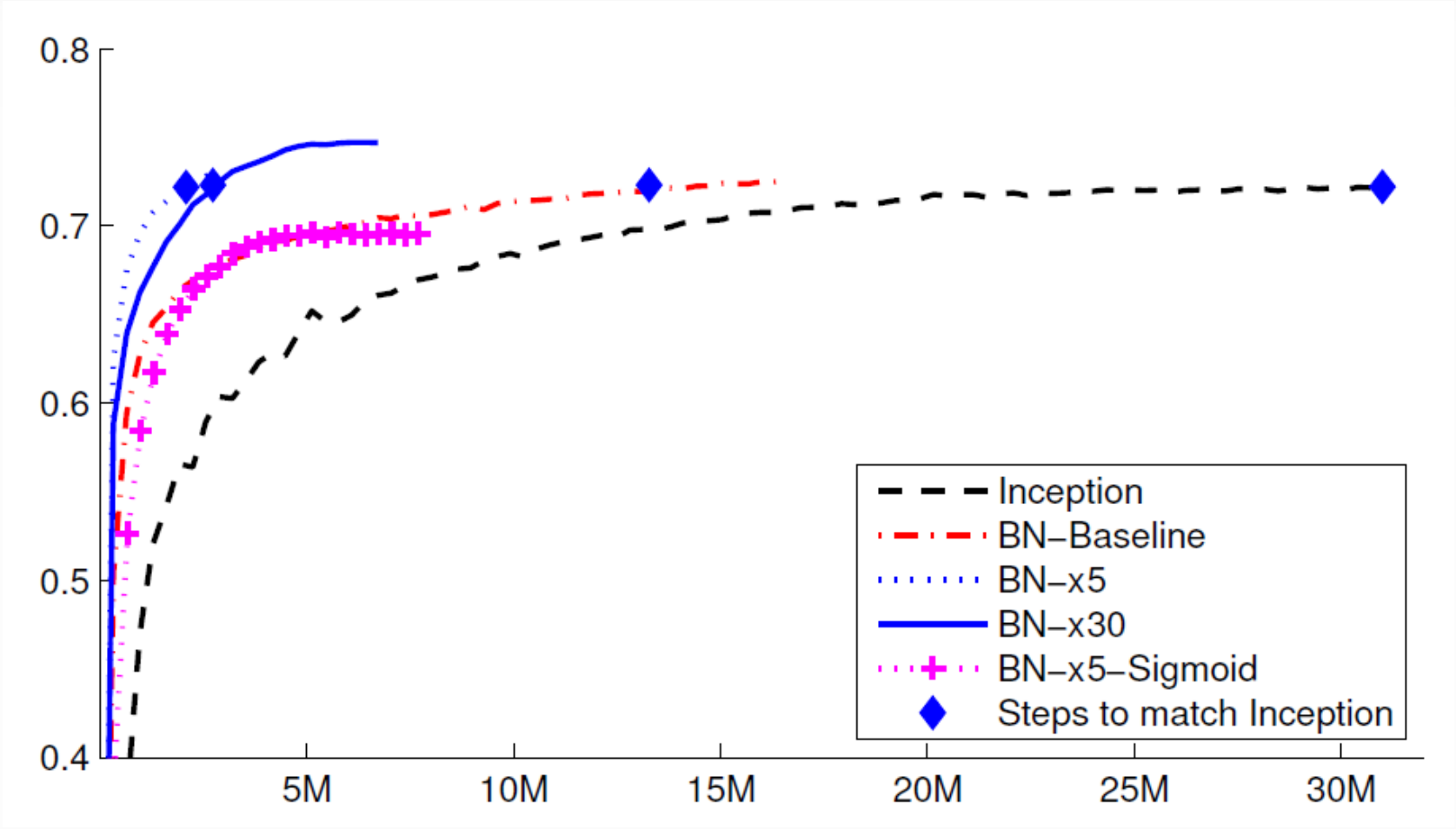
$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m-1}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + \frac{\partial \ell}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m-1} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Batch-normalization comparison



Batch normalization analysis

- Works fast
- Converge fast
- Make other regularization not so useful

Next time on screen

- Convolutional neural networks and computer vision problems
- Prerequisite
  1. Tensors
     https://medium.com/@quantumsteinke/whats-the-difference-between-a-matrix-and-a-tensor-4505fbdc576c (in short)
     https://sundoc.bibliothek.uni-halle.de/habil-online/06/06H055/t7.pdf (in details)

48