Lecture 7. Convolutional Neural Networks, advanced technics

Alex Avdyushenko

Kazakh-British Technical University

October 24, 2022

2022-10-21

Lecture 7. Convolutional Neural Networks, advanced technics

Lecture 7. Convolutional Neural Networks, advanced technics

Alex Avdyushenko

Kazakh-British Technical University

October 24, 2022

Hello, today we continue to talk about Convolutional Neural Networks, advanced optimization methods ans architecture.

And again we will start with three questions for five minutes based on the materials of the previous lecture.

Five-minutes block

Five-minutes block

Please, write answers or send photos with them directly to me in private messages here in Teams, so that others cannot read your message. Last time, a lot of people did it, so I believe that this time you all will succeed.

Five-minutes block

- Define a convolution operation for neural networks
- Write down the main features of the AlexNet network
- Write the formulas for updating the weights in the Momentum method

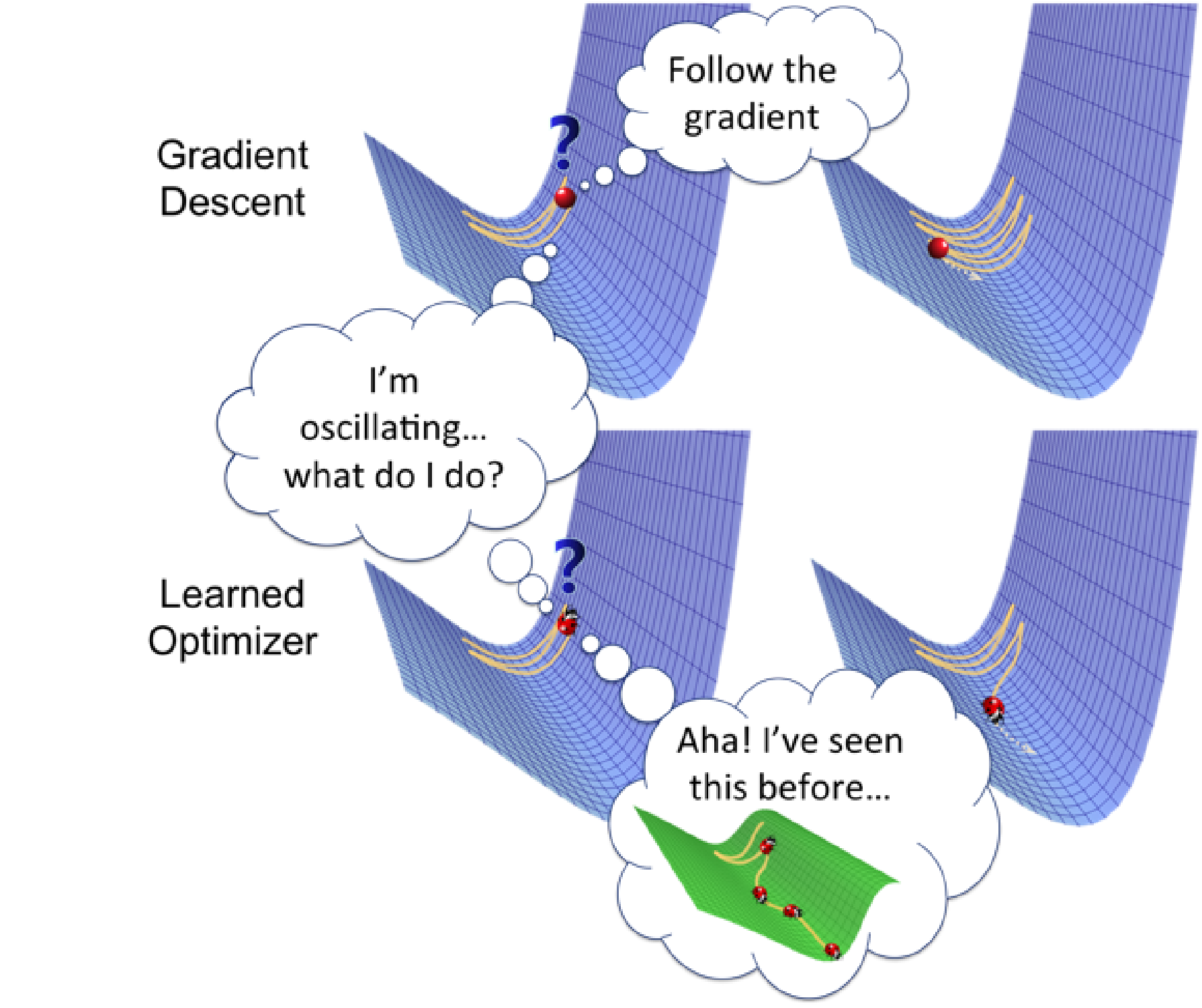
Lecture 7. Convolutional Neural Networks, advanced technics

Five-minutes block

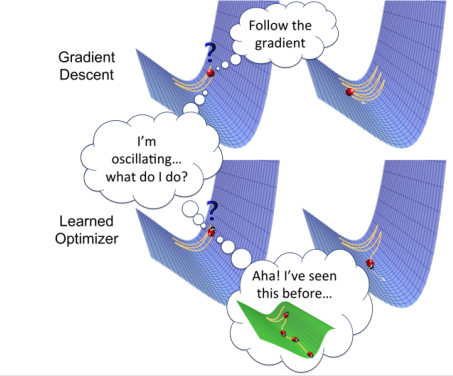
Five-minutes block

- Define a convolution operation for neural networks
- Write down the main features of the AlexNet network
- Write the formulas for updating the weights in the Momentum method

Please, write answers or send photos with them directly to me in private messages here in Teams, so that others cannot read your message. Last time, a lot of people did it, so I believe that this time you all will succeed.


Gradient Descent
Follow the gradient
?
I'm oscillating… what do I do?
Learned Optimizer
?
Aha! I've seen this before…

Lecture 7. Convolutional Neural Networks, advanced technics



Let's look at this picture. In many ways, it reflects the entire content of today's lecture.

Neural networks optimization methods

Recall the Momentum method

A very popular method for improving the convergence of SGD is the momentum accumulation method. In it, from a physical point of view, the derivative of the loss function becomes the acceleration of the change in model parameters, and not the speed as in classical SGD.

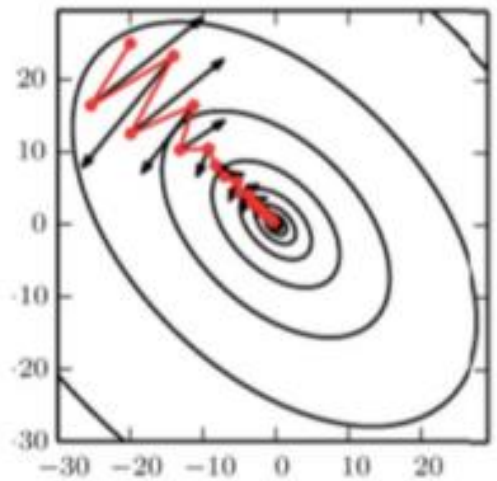
Neural networks optimization methods

Recall the Momentum method

Momentum accumulation method [B.T.Polyak, 1964] — exponential moving average of the gradient over $\frac{1}{1-\gamma}$ last iterations:

$$\nu = \gamma\nu + (1-\gamma)\mathcal{L}'_i(w)$$

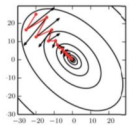
$$w = w - \eta\nu$$



Momentum accumulation method [B.T.Polyak, 1964] — exponential moving average of the gradient over $\frac{1}{1-\gamma}$ last iterations:

$$\nu = \gamma\nu + (1-\gamma)\mathcal{L}_i'(w)$$

$$w = w - \eta\nu$$



A very popular method for improving the convergence of SGD is the momentum accumulation method. In it, from a physical point of view, the derivative of the loss function becomes the acceleration of the change in model parameters, and not the speed as in classical SGD.

AdaGrad

Another interesting approach is the adaptive gradient method. You accumulate squares of gradient vector and then change the model parameters by the component-wise way.

AdaGrad

$$G = G + \mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$,
$\odot, \oslash$ — component-wise multiplication and division of vectors

AdaGrad

$$G = G + \mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$,
$\odot, \oslash$ — component-wise multiplication and division of vectors

Another interesting approach is the adaptive gradient method. You accumulate squares of gradient vector and then change the model parameters by the component-wise way.

AdaGrad

$$G = G + \mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$,
$\odot, \oslash$ — component-wise multiplication and division of vectors
Advantages:

- $\mathcal{L}'_i(w)$ — gradient on i-th object
- separate choice of adaptation for each direction
- is suitable for sparse data

AdaGrad

$$G = G + \mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$,
$\odot, \oslash$ — component-wise multiplication and division of vectors
Advantages:

- $\mathcal{L}'_i(w)$ — gradient on i-th object
- separate choice of adaptation for each direction
- is suitable for sparse data

Another interesting approach is the adaptive gradient method. You accumulate squares of gradient vector and then change the model parameters by the component-wise way.

AdaGrad

$$G = G + \mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$,
$\odot, \oslash$ — component-wise multiplication and division of vectors
Advantages:

- $\mathcal{L}'_i(w)$ — gradient on i-th object
- separate choice of adaptation for each direction
- is suitable for sparse data

Disadvantage:

- G is constantly increasing, which can cause learning to stop

Lecture 7. Convolutional Neural Networks, advanced technics

AdaGrad

AdaGrad

$$G = G + \mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$,
$\odot, \oslash$ — component-wise multiplication and division of vectors
Advantages:

- $\mathcal{L}'_i(w)$ — gradient on i-th object
- separate choice of adaptation for each direction
- is suitable for sparse data

Disadvantage:

- G is constantly increasing, which can cause learning to stop

Another interesting approach is the adaptive gradient method. You accumulate squares of gradient vector and then change the model parameters by the component-wise way.

RMSProp (running mean square)

The development of AdaGrad is RMSProp — optimization method, which you really have in modern libraries.

RMSProp (running mean square)

Equalization of rates of change of weights by moving average

$$G = \alpha G + (1 - \alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$

RMSProp (running mean square)

Equalization of rates of change of weights by moving average

$$G = \alpha G + (1 - \alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$

The development of AdaGrad is RMSProp — optimization method, which you really have in modern libraries.

RMSProp (running mean square)

Equalization of rates of change of weights by moving average

$$G = \alpha G + (1 - \alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$
Advantages:

- same as AdaGrad
- and G does not grow uncontrollably

Lecture 7. Convolutional Neural Networks, advanced technics

RMSProp (running mean square)

RMSProp (running mean square)

Equalization of rates of change of weights by moving average

$$G = \alpha G + (1 - \alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$
Advantages:

- same as AdaGrad
- and G does not grow uncontrollably

The development of AdaGrad is RMSProp — optimization method, which you really have in modern libraries.

RMSProp (running mean square)

Equalization of rates of change of weights by moving average

$$G = \alpha G + (1 - \alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$
Advantages:

- same as AdaGrad
- and G does not grow uncontrollably

Disadvantages:

- at the very beginning of training G is averaged over zero previous values
- no moment count

Lecture 7. Convolutional Neural Networks, advanced technics

RMSProp (running mean square)

The development of AdaGrad is RMSProp — optimization method, which you really have in modern libraries.

RMSProp (running mean square)

Equalization of rates of change of weights by moving average

$$G = \alpha G + (1 - \alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w)$$
$$w = w - \eta_t \mathcal{L}'_i(w) \oslash \sqrt{G + \varepsilon}$$

η_t can be fixed, for example $\eta_t = 0.01$
Advantages:

- same as AdaGrad
- and G does not grow uncontrollably

Disadvantages:

- at the very beginning of training G is averaged over zero previous values
- no moment count

Adam (adaptive momentum)

Adam (adaptive momentum)

Adam is also one of modern optimization method, which you really may meet in modern applications.

Adam (adaptive momentum)

$$\nu = \gamma\nu + (1-\gamma)\mathcal{L}'_i(w) \qquad \hat{\nu} = \nu(1-\gamma^k)^{-1}$$

$$G = \alpha G + (1-\alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w) \qquad \hat{G} = G(1-\alpha^k)^{-1}$$

$$w = w - \eta_t \hat{\nu} \oslash (\sqrt{\hat{G} + \varepsilon})$$

k — iteration number, $\gamma = 0.9, \alpha = 0.999, \varepsilon = 10^{-8}$

Lecture 7. Convolutional Neural Networks, advanced technics

Adam (adaptive momentum)

Adam (adaptive momentum)

$$\nu = \gamma\nu + (1-\gamma)\mathcal{L}_i'(w) \qquad \hat{\nu} = \nu(1-\gamma^k)^{-1}$$
$$G = \alpha G + (1-\alpha)\mathcal{L}_i'(w) \odot \mathcal{L}_i'(w) \qquad \hat{G} = G(1-\alpha^k)^{-1}$$
$$w = w - \eta_t \hat{\nu} \oslash (\sqrt{\hat{G} + \varepsilon})$$

k — iteration number, $\gamma = 0.9, \alpha = 0.999, \varepsilon = 10^{-8}$

Adam is also one of modern optimization method, which you really may meet in modern applications.

Adam (adaptive momentum)

$$\nu = \gamma\nu + (1-\gamma)\mathcal{L}'_i(w) \qquad \hat{\nu} = \nu(1-\gamma^k)^{-1}$$

$$G = \alpha G + (1-\alpha)\mathcal{L}'_i(w) \odot \mathcal{L}'_i(w) \qquad \hat{G} = G(1-\alpha^k)^{-1}$$

$$w = w - \eta_t \hat{\nu} \oslash (\sqrt{\hat{G} + \varepsilon})$$

k — iteration number, $\gamma = 0.9, \alpha = 0.999, \varepsilon = 10^{-8}$

Advantages:

- are almost the same as RMSProp
- calibration $\hat{\nu}, \hat{G}$ increases ν, G on the first iterations
- is count of moment

Lecture 7. Convolutional Neural Networks, advanced technics

Adam (adaptive momentum)

Adam (adaptive momentum)

$$\nu = \gamma\nu + (1-\gamma)\mathcal{L}_i'(w) \qquad \hat{\nu} = \nu(1-\gamma^k)^{-1}$$
$$G = \alpha G + (1-\alpha)\mathcal{L}_i'(w) \odot \mathcal{L}_i'(w) \qquad \hat{G} = G(1-\alpha^k)^{-1}$$
$$w = w - \eta_t \hat{\nu} \oslash (\sqrt{\hat{G} + \varepsilon})$$

k — iteration number, $\gamma = 0.9, \alpha = 0.999, \varepsilon = 10^{-8}$
Advantages:

- are almost the same as RMSProp
- calibration $\hat{\nu}, \hat{G}$ increases ν, G on the first iterations
- is count of moment

Adam is also one of modern optimization method, which you really may meet in modern applications.

Nadam (Nesterov-accelerated adaptive momentum)

Same formulas for $\nu, \hat{\nu}, G, \hat{G}$

$w = w - \eta_t \left(\gamma \hat{\nu} + \frac{1-\gamma}{1-\gamma^k} \mathcal{L}'_i(w) \right) \oslash (\sqrt{\hat{G} + \varepsilon})$

k — iteration number, $\gamma = 0.9, \alpha = 0.999, \varepsilon = 10^{-8}$

T. Dozat. Incorporating Nesterov Momentum into Adam. ICLR-2016.

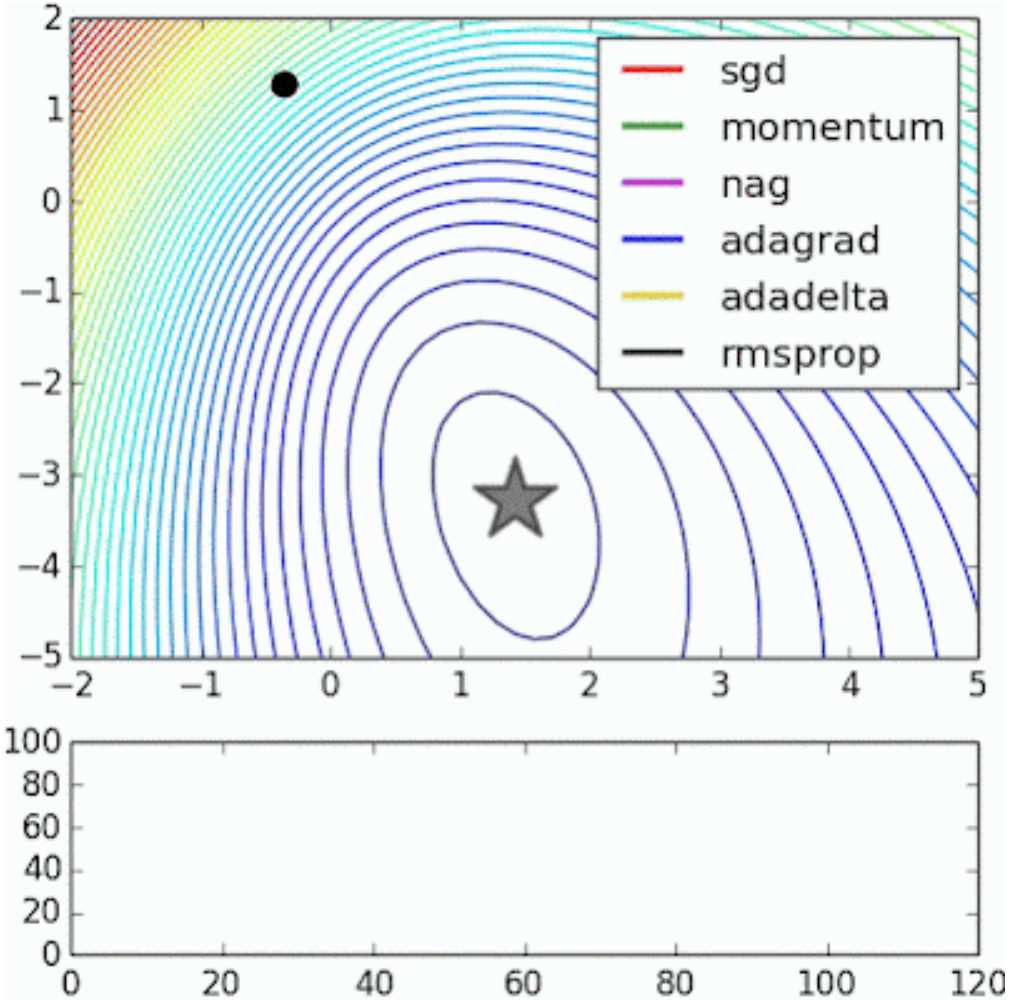
Nadam (Nesterov-accelerated adaptive momentum)

Same formulas for $\nu, \hat{\nu}, G, \hat{G}$
$w = w - \eta_t \left(\gamma\hat{\nu} + \frac{1-\gamma}{1-\gamma^k}\mathcal{L}'_i(w)\right) \oslash (\sqrt{\hat{G} + \varepsilon})$
k — iteration number, $\gamma = 0.9, \alpha = 0.999, \varepsilon = 10^{-8}$

T. Dozat. Incorporating Nesterov Momentum into Adam. ICLR-2016.

And Nadam (Nesterov-accelerated adaptive momentum) — is modern modification of Adam method.

Convergence example



Source: http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

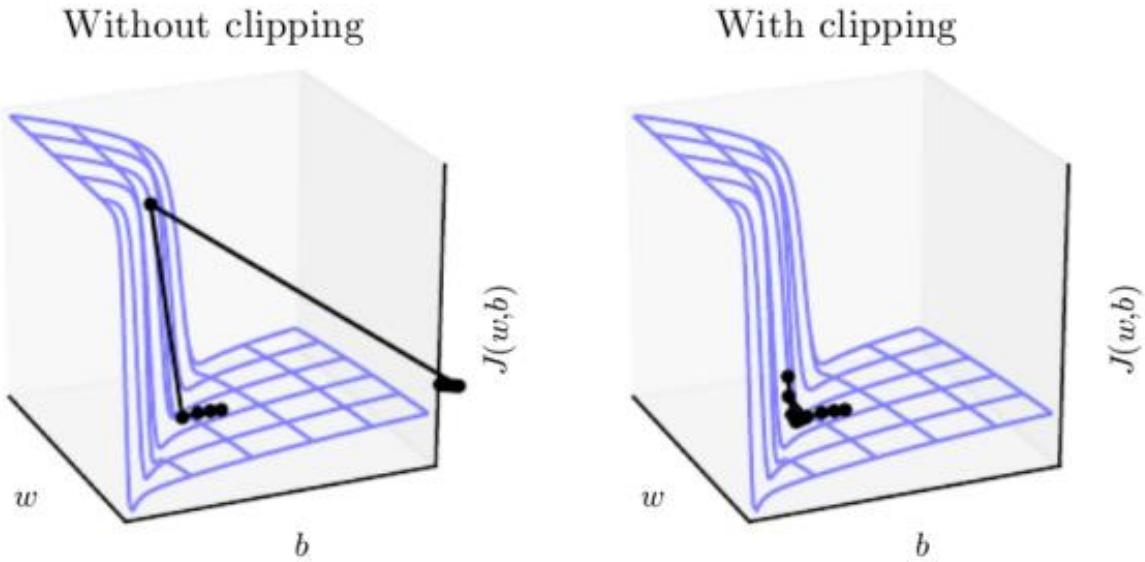
Source: http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Let's look on example of convergence of considered optimization methods.

Question

Which method and how to choose?

Gradient exploding



You need to numpy.clip by value or by absolute value.

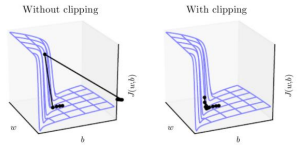
Source: https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem

Lecture 7. Convolutional Neural Networks, advanced technics

Gradient exploding

There is typical problem in neural network optimization, it is gradient exploding.

Gradient exploding



You need to numpy.clip by value or by absolute value.

Source: https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem

Regularization Methods

Dropout is a method of random shutdowns of neurons. It may have two stages: learning stage and application stage.

Regularization Methods

Dropout is a method of random shutdowns of neurons

Dropout is a method of random shutdowns of neurons. It may have two stages: learning stage and application stage.

Regularization Methods

Dropout is a method of random shutdowns of neurons

Learning stage: make the gradient step $\mathcal{L}_i(w) \to \min\limits_w$, disable the h-th neuron of the ℓ-th layer with probability p_ℓ:

$$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h\Big(\sum_j w_{jh} x_{ij}^\ell\Big), \quad P(\xi_h^\ell = 0) = p_\ell$$

Lecture 7. Convolutional Neural Networks, advanced technics

Regularization Methods

Regularization Methods

Dropout is a method of random shutdowns of neurons
Learning stage: make the gradient step $\mathcal{L}_i(w) \to \min_w$, disable the h-th neuron of the ℓ-th layer with probability p_ℓ:
$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h(\sum_j w_{jh} x_{ij}^\ell), \quad P(\xi_h^\ell = 0) = p_\ell$

Dropout is a method of random shutdowns of neurons. It may have two stages: learning stage and application stage.

Regularization Methods

Dropout is a method of random shutdowns of neurons

Learning stage: make the gradient step $\mathcal{L}_i(w) \to \min_w$, disable the h-th neuron of the ℓ-th layer with probability p_ℓ:

$$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h\Big(\sum_j w_{jh} x_{ij}^\ell\Big), \quad P(\xi_h^\ell = 0) = p_\ell$$

Application stage: turn on all neurons, but with a correction:

$$x_{ih}^{\ell+1} = (1 - p_\ell)\sigma_h\Big(\sum_j w_{jh} x_{ij}^\ell\Big)$$

Lecture 7. Convolutional Neural Networks, advanced technics

Regularization Methods

Regularization Methods

Dropout is a method of random shutdowns of neurons
Learning stage: make the gradient step $\mathcal{L}_i(w) \to \min_w$, disable the h-th neuron of the ℓ-th layer with probability p_ℓ:
$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h(\sum_j w_{jh} x_{ij}^\ell), \quad P(\xi_h^\ell = 0) = p_\ell$
Application stage: turn on all neurons, but with a correction:
$x_{ih}^{\ell+1} = (1 - p_\ell)\sigma_h(\sum_j w_{jh} x_{ij}^\ell)$

Dropout is a method of random shutdowns of neurons. It may have two stages: learning stage and application stage.

Regularization Methods

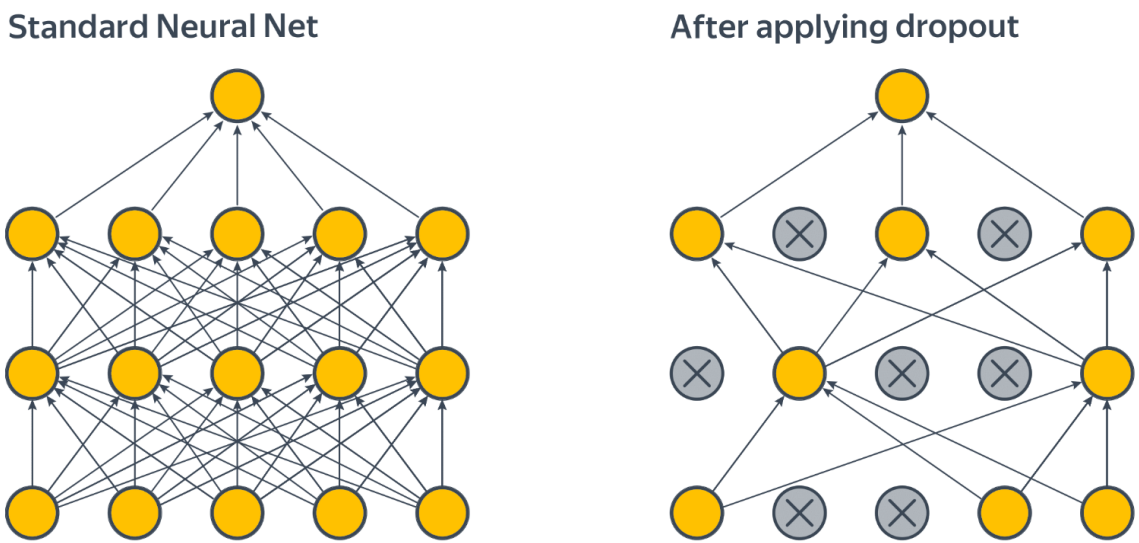
Dropout is a method of random shutdowns of neurons

Learning stage: make the gradient step $\mathcal{L}_i(w) \to \min_w$, disable the h-th neuron of the ℓ-th layer with probability p_ℓ:

$$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h\Big(\sum_j w_{jh} x_{ij}^\ell\Big), \quad P(\xi_h^\ell = 0) = p_\ell$$

Application stage: turn on all neurons, but with a correction:

$$x_{ih}^{\ell+1} = (1 - p_\ell)\sigma_h\Big(\sum_j w_{jh} x_{ij}^\ell\Big)$$



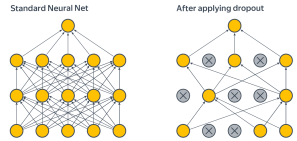
Lecture 7. Convolutional Neural Networks, advanced technics

Regularization Methods

Regularization Methods

Dropout is a method of random shutdowns of neurons
Learning stage: make the gradient step $\mathcal{L}_i(w) \to \min_w$, disable the h-th neuron of the ℓ-th layer with probability p_ℓ:
$x_{ih}^{\ell+1} = \xi_h^\ell \sigma_h(\sum_j w_{jh} x_{ij}^\ell), \quad P(\xi_h^\ell = 0) = p_\ell$

Application stage: turn on all neurons, but with a correction:
$x_{ih}^{\ell+1} = (1 - p_\ell)\sigma_h(\sum_j w_{jh} x_{ij}^\ell)$



Dropout is a method of random shutdowns of neurons. It may have two stages: learning stage and application stage.

Question

What is the peculiarity of this case of dropout implementation?

Inverted Dropout

In practice **Inverted Dropout** is used

Inverted Dropout

In practice **Inverted Dropout** is used

Learning stage:

$$x_{ih}^{\ell+1} = \frac{1}{1-p_\ell}\xi_h^\ell \sigma_h\Big(\sum_j w_{jh}x_{ij}^\ell\Big), \quad P(\xi_h^\ell = 0) = p_\ell$$

Inverted Dropout

In practice **Inverted Dropout** is used
Learning stage:

$$x_{ih}^{\ell+1} = \frac{1}{1-p_\ell}\xi_h^\ell \sigma_h\Big(\sum_j w_{jh}x_{ij}^\ell\Big), \quad P(\xi_h^\ell = 0) = p_\ell$$

Application stage doesn't require neither modification nor knowledge of p_ℓ:

$$x_{ih}^{\ell+1} = \sigma_h\Big(\sum_j w_{jh}x_{ij}^\ell\Big)$$

L_2-regularization prevents growth of training parameters:

$$\mathcal{L}_i(w) + \frac{\lambda}{2}\|w\|^2 \to \min_w$$

Gradient step with Dropout and L_2 regularization:

$$w = w(1-\eta\lambda) - \eta\frac{1}{1-p_\ell}\xi_h^\ell \mathcal{L}_i'(w)$$

Dropout interpretations

Dropout interpretations

- we approximate simple voting over 2^N networks with a common set of N neurons

Dropout interpretations

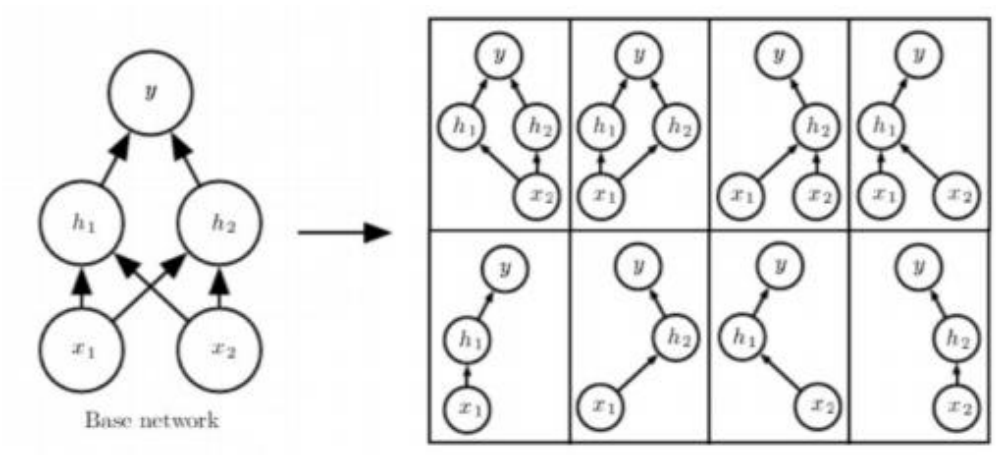
- we approximate simple voting over 2^N networks with a common set of N neurons
- regularization: we choose from all networks the most resistant to the loss of pN neurons, simulating the reliability of the brain

Dropout interpretations

- we approximate simple voting over 2^N networks with a common set of N neurons
- regularization: we choose from all networks the most resistant to the loss of pN neurons, simulating the reliability of the brain
- reducing retraining by forcing different parts of the network to solve the same original task

Dropout interpretations

- we approximate simple voting over 2^N networks with a common set of N neurons
- regularization: we choose from all networks the most resistant to the loss of pN neurons, simulating the reliability of the brain
- reducing retraining by forcing different parts of the network to solve the same original task



Batch normalization

$B = \{x_i\}$ — mini-batch of whole dataset
Averaging the $\mathcal{L}_i(w)$ gradients over the batch speeds up the convergence.
$B^\ell = \{u_i^\ell\}$ — vectors of x_i objects at the output of the ℓ-th layer

Batch normalization

$B = \{x_i\}$ — mini-batch of whole dataset
Averaging the $\mathcal{L}_i(w)$ gradients over the batch speeds up the convergence.
$B^\ell = \{u_i^\ell\}$ — vectors of x_i objects at the output of the ℓ-th layer
1. Normalize each j-th component of the vector u_i^ℓ by the batch

$$\hat{u}_{ij}^\ell = \frac{u_{ij}^\ell - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}; \quad \mu_j = \frac{1}{|B|}\sum_{x_i \in B} u_{ij}^\ell; \quad \sigma_j^2 = \frac{1}{|B|}\sum_{x_i \in B}(u_{ij}^\ell - \mu_j)^2$$

Batch normalization

$B = \{x_i\}$ — mini-batch of whole dataset
Averaging the $\mathcal{L}_i(w)$ gradients over the batch speeds up the convergence.
$B^\ell = \{u_i^\ell\}$ — vectors of x_i objects at the output of the ℓ-th layer
1. Normalize each j-th component of the vector u_i^ℓ by the batch

$$\hat{u}_{ij}^\ell = \frac{u_{ij}^\ell - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}; \quad \mu_j = \frac{1}{|B|}\sum_{x_i \in B} u_{ij}^\ell; \quad \sigma_j^2 = \frac{1}{|B|}\sum_{x_i \in B}(u_{ij}^\ell - \mu_j)^2$$

2. Add a linear layer with custom weights:

$$\tilde{u}_{ij}^\ell = \gamma_j^\ell \hat{u}_{ij}^\ell + \beta_j^\ell$$

Batch normalization

$B = \{x_i\}$ — mini-batch of whole dataset
Averaging the $\mathcal{L}_i(w)$ gradients over the batch speeds up the convergence.
$B^\ell = \{u_i^\ell\}$ — vectors of x_i objects at the output of the ℓ-th layer
1. Normalize each j-th component of the vector u_i^ℓ by the batch

$$\hat{u}_{ij}^\ell = \frac{u_{ij}^\ell - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}; \quad \mu_j = \frac{1}{|B|}\sum_{x_i \in B} u_{ij}^\ell; \quad \sigma_j^2 = \frac{1}{|B|}\sum_{x_i \in B}(u_{ij}^\ell - \mu_j)^2$$

2. Add a linear layer with custom weights:

$$\tilde{u}_{ij}^\ell = \gamma_j^\ell \hat{u}_{ij}^\ell + \beta_j^\ell$$

3. Parameters $\gamma_j^\ell, \beta_j^\ell$ are configured by BackProp

Initial Approximation of the Weights

Monitoring changes in variances

Both of neuron values and backpropagation gradients

Equalization of variances of both neurons and gradients in different layers — Xavier initialization for tanh:

$$w_j \sim U\left[-\frac{6}{\sqrt{n_{in}+n_{out}}}, \frac{6}{\sqrt{n_{in}+n_{out}}}\right]$$

n_{in}, n_{out} — the number of neurons in the previous and current layers, respectively

The proof and details are in the SHAD textbook (in Russian yet).

Monitoring changes in variances

Both of neuron values and backpropagation gradients

Equalization of variances of both neurons and gradients in different layers — Xavier initialization for tanh:

$$w_j \sim U\left[-\frac{6}{\sqrt{n_{in}+n_{out}}}, \frac{6}{\sqrt{n_{in}+n_{out}}}\right]$$

n_{in}, n_{out} — the number of neurons in the previous and current layers, respectively

The proof and details are in the SHAD textbook (in Russian yet).

Question

What to do in case of ReLU activation?

Other ways of initialization

1. Layer-by-layer training of neurons as linear models
   - or by random subsample
   - or by a random subset of inputs
   - or from various random initial distributions

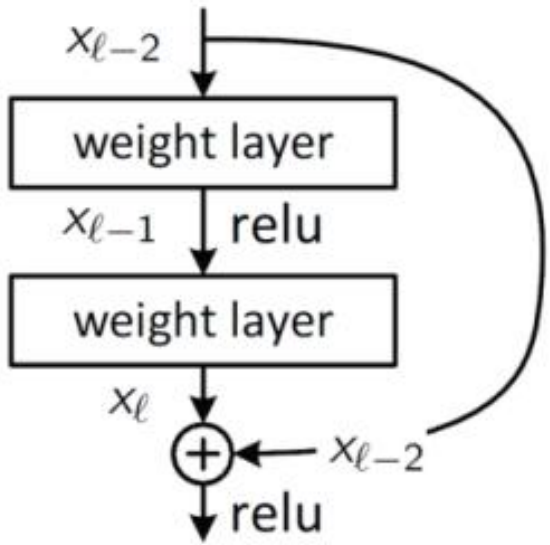
   This ensures the diversity of neurons.
2. Weight initialization of a pre-trained model (for example, an autoencoder)

ResNet — Residual Net

Skip-connection of the ℓ layer with the previous $\ell - d$ layer:

$$x_\ell = \sigma(Wx_{\ell-1}) + x_{\ell-d}$$

The ℓ layer learns not the new vector representation x_ℓ, but its increment $x_\ell - x_{\ell-d}$

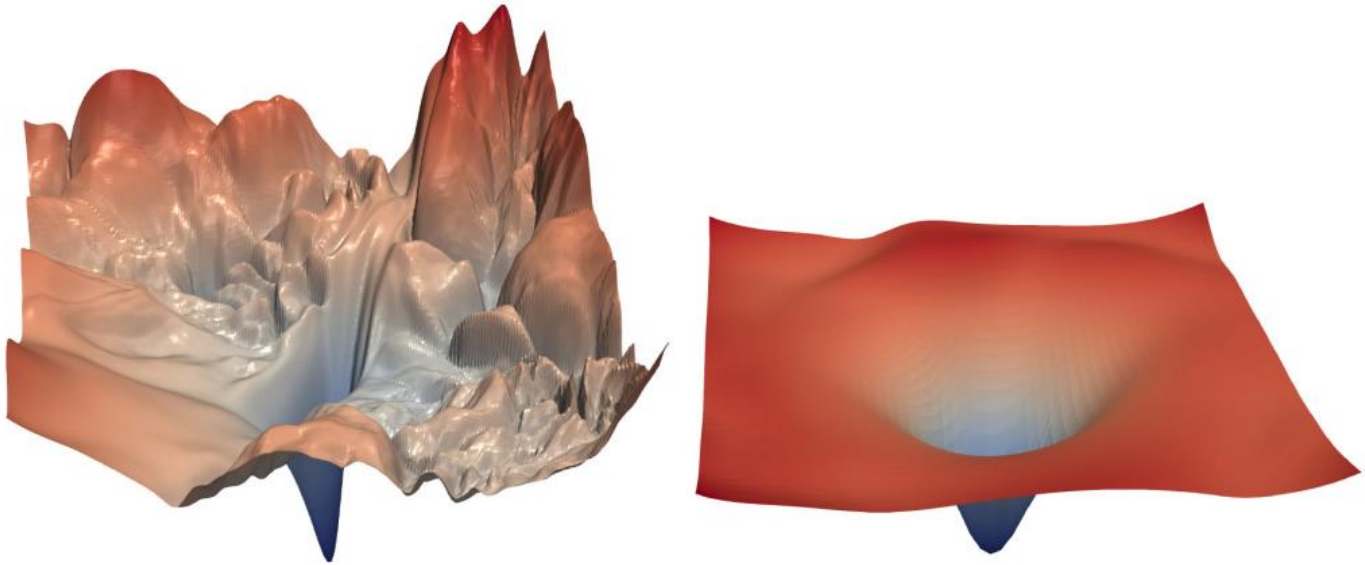


- (main) it is possible to increase the number of layers
- increments are more stable $\rightarrow$ convergence is better

---

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. 2015.
R.K. Srivastava, K. Greff, J. Schmidhuber. Highway Networks. 2015

ResNet: visualization of loss function



without skip connections

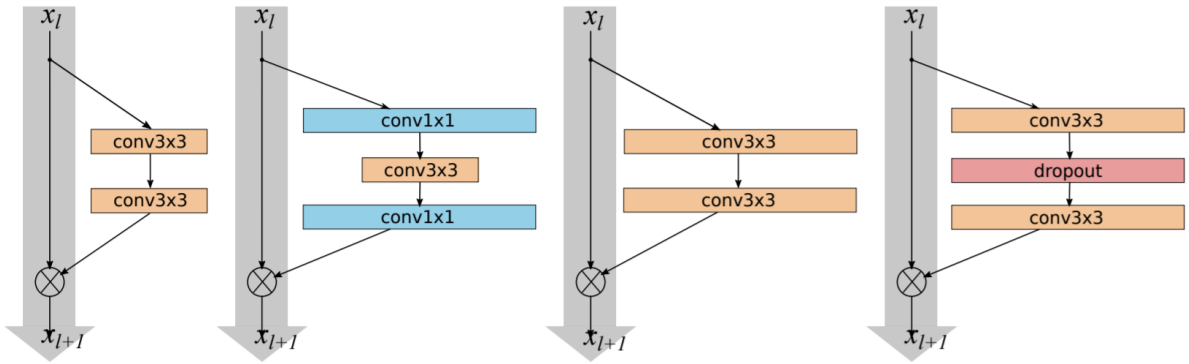
with skip connections

Hao Li et al. Visualizing the Loss Landscape of Neural Nets. 2018.

WideResNet — WRN

Disadvantages of ResNet

- is too deep — up to 1000 layers :)
- take a long time to train to SotA



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

- a simple WRN network of 16 layers defeated all ResNets
- WRN-40-4 (i.e. 40 layers and 4x width) trains 8 times faster than ResNet-1001

S. Zagoruyko, N. Komodakis Wide Residual Networks, 2017

Summary

- Convolutional networks are very well suited for image processing
- Various optimization algorithms: adam, RMSProp
- Regularization methods: dropout, L_2 and batch normalization
- The choice of the initial approximation is also important (weight initialization)
- ResNet and skip-connections, WideResNet

Summary

- Convolutional networks are very well suited for image processing
- Various optimization algorithms: adam, RMSProp
- Regularization methods: dropout, L_2 and batch normalization
- The choice of the initial approximation is also important (weight initialization)
- ResNet and skip-connections, WideResNet

What else can you see?

- Course lecture at Stanford about convolutional networks

Levenberg-Marquardt Diagonal Method

Reminder

Newton-Raphson method (second order):

$$w = w - \eta(\mathcal{L}_i''(w))^{-1}\mathcal{L}_i'(w),$$

where $\mathcal{L}_i''(w) = \frac{\partial^2 \mathcal{L}_i(w)}{\partial w_{jh}\partial w_{j'h'}}$ — Hessian

Heuristics. We assume that the Hessian is diagonal:

$$w_{jh} = w_{jh} - \eta\left(\frac{\partial^2 \mathcal{L}_i(w)}{\partial w_{jh}^2} + \mu\right)^{-1}\frac{\partial \mathcal{L}_i(w)}{\partial w_{jh}},$$

η is the learning rate, we can assume $\eta = 1$

μ — a parameter that prevents the denominator from being set to zero

The ratio η/μ is the learning rate on flat sections of the functional $\mathcal{L}_i(w)$, where the second derivative vanishes.