

# Lecture 7. Convolutional Neural Networks, advanced techniques

Alex Avdyushenko

Kazakh-British Technical University

October 24, 2022



# 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

Gradient  
Descent

Follow the  
gradient

I'm  
oscillating...  
what do I do?

Learned  
Optimizer

Aha! I've seen  
this before...

# Neural networks optimization methods

Recall the Momentum method

# Neural networks optimization methods

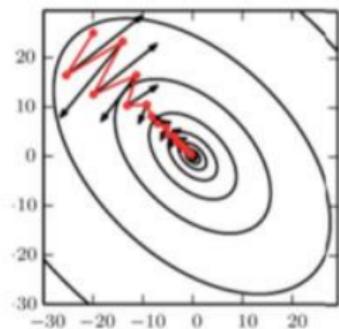
Recall the Momentum method

Momentum accumulation method

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

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

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



# AdaGrad

# AdaGrad

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

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

$\odot, \oslash$  — component-wise multiplication and division of vectors

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

$\eta_t$  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

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

$\eta_t$  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

# RMSProp (running mean square)

## RMSProp (running mean square)

Equalization of rates of change of weights by moving average

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

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

# RMSProp (running mean square)

Equalization of rates of change of weights by moving average

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

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

Advantages:

- same as AdaGrad
- and  $G$  does not grow uncontrollably

# RMSProp (running mean square)

Equalization of rates of change of weights by moving average

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

$\eta_t$  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)

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

$$\hat{\nu} = \nu(1 - \gamma^k)^{-1}$$

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

$$\hat{G} = G(1 - \alpha^k)^{-1}$$

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

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

## Adam (adaptive momentum)

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

$$\hat{\nu} = \nu(1 - \gamma^k)^{-1}$$

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

$$\hat{G} = G(1 - \alpha^k)^{-1}$$

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

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

Advantages:

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

## 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.

# Convergence example

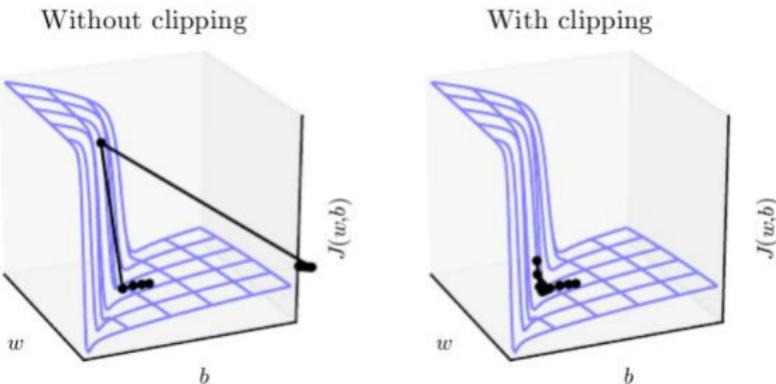
---

Source: <http://www.denizyuret.com/2015/03/alec-radford-s-animations-for.html>

## 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>

# Regularization Methods

# Regularization Methods

**Dropout** is a method of random shutdowns of neurons

# Regularization Methods

**Dropout** is a method of random shutdowns of neurons

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

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

# Regularization Methods

**Dropout** is a method of random shutdowns of neurons

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

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

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

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

# Regularization Methods

**Dropout** is a method of random shutdowns of neurons

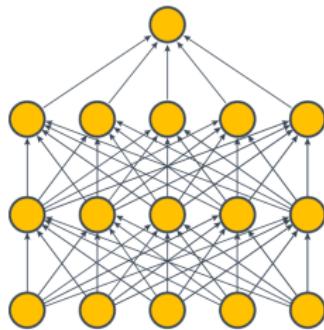
**Learning stage:** make the gradient step  $\mathcal{L}_i(w) \rightarrow \min_w$ , disable the  $h$ -th neuron of the  $\ell$ -th layer with probability  $p_\ell$ :

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

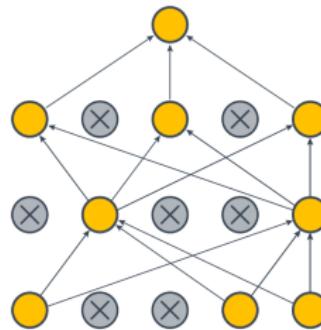
**Application stage:** turn on all neurons, but with a correction:

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

Standard Neural Net



After applying dropout



## 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:**

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

# Inverted Dropout

In practice **Inverted Dropout** is used

**Learning stage:**

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

**Application stage** doesn't require neither modification nor knowledge of  $p_\ell$ :

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

$L_2$ -regularization prevents growth of training parameters:

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

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

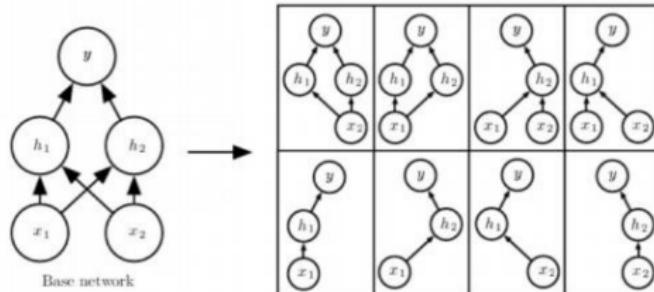
- we approximate simple voting over  $2^N$  networks with a common set of  $N$  neurons
- regularization: we choose from all networks the most resistant to the loss of  $pN$  neurons, simulating the reliability of the brain

# 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  $\ell$ -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  $\ell$ -th layer

1. Normalize each  $j$ -th component of the vector  $u_i^\ell$  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  $\ell$ -th layer

1. Normalize each  $j$ -th component of the vector  $u_i^\ell$  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  $\ell$ -th layer

1. Normalize each  $j$ -th component of the vector  $u_i^\ell$  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

**When you initialise your ML  
noob friend's NN weights with zeros**



# 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.

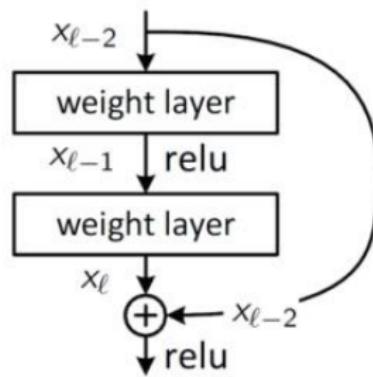
## ② Weight initialization of a pre-trained model (for example, an autoencoder)

# ResNet — Residual Net

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

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

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

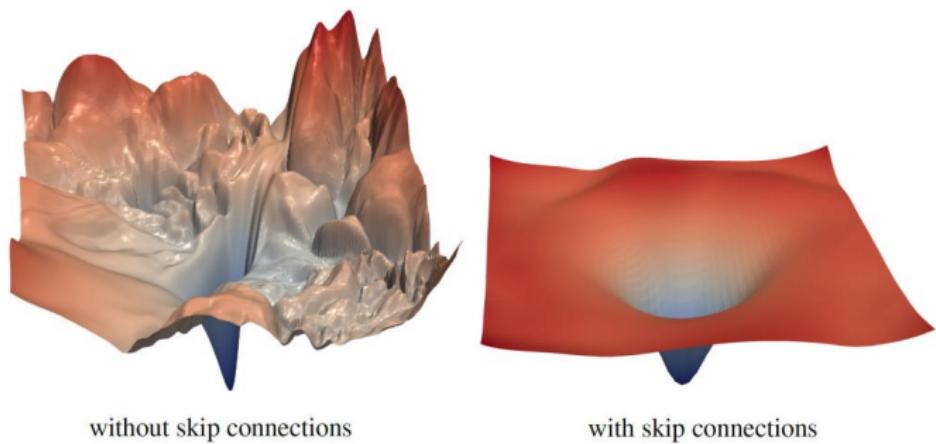


- (main) it is possible to increase the number of layers
- increments are more stable → convergence is better

---

*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



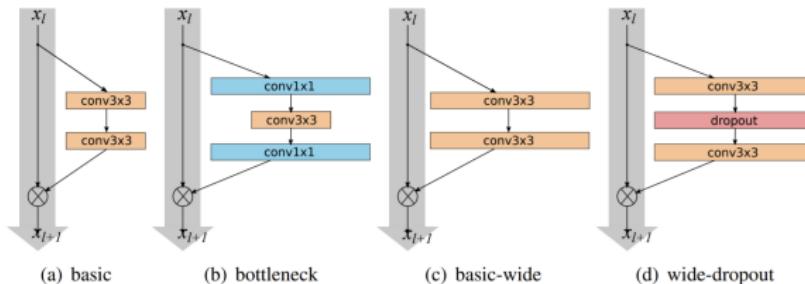
---

*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 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

# 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:

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

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

$\mu$  — a parameter that prevents the denominator from being set to zero

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