Introduction to machine learning

Maksim Kretov

Lecture 7: Training deep neural networks

5vision, 2017

Course information I

Course

10 lectures + 2 seminars; February-May 2017.

Schedule and up-to-date syllabus

https://goo.gl/xExEuL

Contact information and discussion

Maksim Kretov (kretovmk@gmail.com)

Alexey Seleznev (a.o.seleznev@gmail.com)

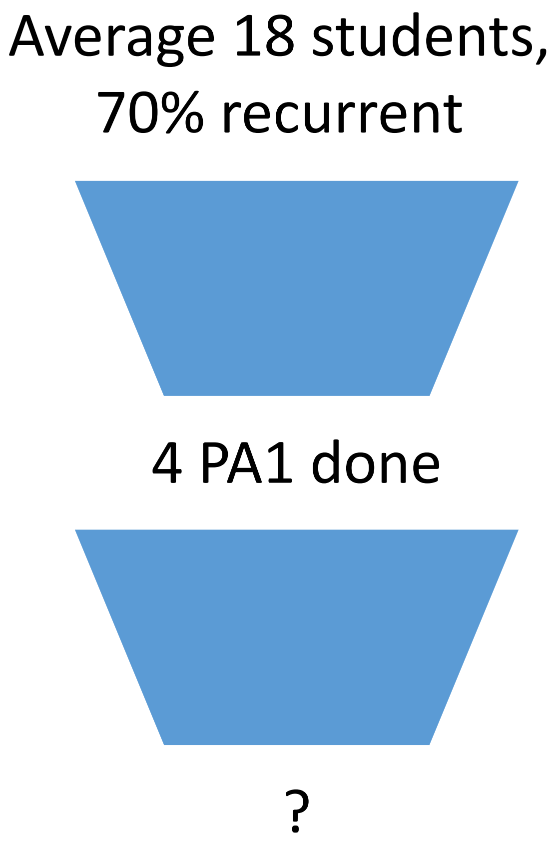
Slack group: https://miptmlcourse.slack.com

to get an invite, send e-mail to kretovmk@gmail.com.

Course information II

Grading policy
3 practical assignments, each gives 25, exam: 25
<u>=> Total: 100. Pass: >=50</u>

Maksim Kretov
Lectures 1-7, PA1 and exam
Alexey Seleznev
Lectures 8-12, PA2-3 and exam



Plan of the course

Math and basics of ML	(1-2)		*Theoretical tasks*
Some of ML methods	(3)		
Seminar on ML basics	(4)		
Basics of neural networks	(5)		*+Practical tasks*
Deep learning overview	(6)		
<u>Training deep networks</u>	(7)	← **Today**	
DL for Computer Vision	(8-9)	**Solving more complex ML tasks using NNs**	
DL for time series prediction	(10-11)		
Concluding seminar	(12)		

Plan for the lecture

A. Previous lecture
   1. ERM framework
   2. Deep learning
B. Improving convergence of BP
   1. Cross-entropy
   2. Weights initialization
C. Gradient descent
   1. Stochastic approximation
   2. Advanced GD: momentum, adagrad etc.
D. Regularization: Dropout, Batch normalization.
E. Baseline models
F. Practical assignments

A.1 Previous lectures: ERM framework

Empirical risk minimization approach (ERM)

Formula for fitting the model within ERM framework:

$$\theta^{opt} = \mathbf{argmin}_\theta \frac{1}{N} \sum_{n=1}^{N} L(y_n, \hat{y}_n) + \lambda \Omega(\theta)$$

$L\big(y_n, f(\mathbf{x}_{\boldsymbol{n}}, \theta)\big)$ is loss function; $\hat{y}_n = f(\mathbf{x}_{\boldsymbol{n}}, \theta)$ is prediction

$\Omega(\theta)$ is a regularizer **=> learning converted into optimization task!**

Training neural networks with ERM. Probabilistic interpretation:

Maximizing likelihood of correct class in predicted distribution.

A.2 Previous lectures: Deep learning

Just an informal definition of “deep” networks

Networks with up to 3 (2 hidden) layers → shallow

More than 3 layers → deep

Traditional methods: local smoothness assumption

Deep learning methods: complement with “compositionality” prior.

Deep Learning

Machine learning algorithms based on learning multiple levels of representation / abstraction.*

* Definition from NIPS 2015 Deep Learning tutorial

B. Improving convergence

Why do we need special training protocol for deep networks?

- Loss function is complex and non-convex (underfitting)
- Lots of parameters, more than examples in training set (overfitting)
- Huge datasets (slow, don't fit in memory)
- NN-specific problems (vanishing gradient)

B.1 Improving convergence: Cross-entropy

Cross-entropy cost

$L\big(Y, f(\mathbf{X}, \theta)\big) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk} \ln \hat{y}_{nk}$

y_{nk} are true labels and $\hat{y}_{nk}$ are predictions.

N – number of training examples; K is number of classes in classification task.

Each term is log probability of sampling true label from predicted distribution.

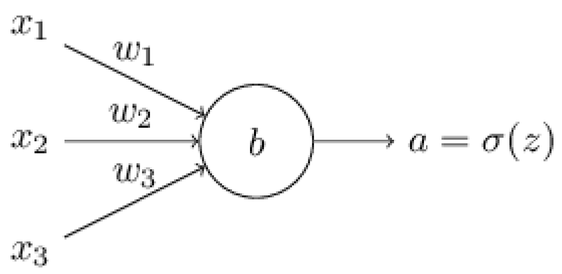
Such loss function meets our criteria:

- connected to a clear proxy such as accuracy
- smooth and easy to differentiate (accuracy is not smooth!)
- probabilistic interpretation: MLE estimation $f(\mathbf{x}, \theta) = p(y|\mathbf{x}, \theta)$ or statistics

B.1 Improving convergence: Cross-entropy

Cross-entropy + sigmoid output

Cross-entropy allows to avoid vanishing gradient for some activation functions in neural networks.



$$\frac{\partial L_{CE}(y_n, \hat{y}_n)}{\partial \omega_i} = x_{ni}(\hat{y}_n - y_n)$$

$$\frac{\partial L_{MSE}(y_n, \hat{y}_n)}{\partial \omega_i} = x_{ni}(\hat{y}_n - y_n)\sigma'(z_n)$$

If weighted input to neuron is close to 1, then $\sigma'(z_n)$ is close to 0 and learning starts to slowing down (“saturation”).

* Image from http://neuralnetworksanddeeplearning.com/chap3.html

B.1 Improving convergence: Cross-entropy

Cross-entropy + softmax output

Also provides non-saturated architecture:

$\log softmax(z)_i = z_i - \log \sum_j \exp z_j$ ***Exercise:*** *check it and MSE+softmax.*

Other considerations

1. In practice, logits are used as outputs: $z_i = \omega_i \ a_{i-1} + b_i$ (weighted input)

2. Multi-label classification:

Cross entropy can be recorded separately for each neuron with sigmoid:

$$L\big(Y, f(\mathbf{X}, \theta)\big) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} (y_{nk} \ln \hat{y}_{nk} + (1 - y_{nk}) \ln(1 - \hat{y}_{nk}))$$

B.2 Improving convergence: Weights init.

Weights initialization

Is the simplest solution possible?

If initialize with zeroes => equal gradient for all neurons.

Simple random initialization ("breaking symmetry"):

Use independent Gaussian RV with $\mu = 0, \sigma = 1$ (or uniform noise).

Problems with that approach:

Standard deviation increases as $\sim\sqrt{n}$, where n is number of inputs

=> may cause saturation of neurons or gradient exploding

Solution:

Initialize with variation $\sim 1/\sqrt{n_{inputs}}$.

B.2 Improving convergence: Weights init.

Weights initialization examples

1. Uniform distributions

$$\omega_{ij} \sim U\left(-\frac{1}{\sqrt{m}}, -\frac{1}{\sqrt{m}}\right)$$

$$\omega_{ij} \sim U\left(-\frac{\sqrt{6}}{\sqrt{m+n}}, -\frac{\sqrt{6}}{\sqrt{m+n}}\right)$$ “Glorot uniform”

2. Normal distribution

$$\omega_{ij} \sim N\left(0, \frac{2}{m}\right)$$ “He normal”

3. Initialization with random orthogonal matrices

5 minute break..

Questions?

C.1 Gradient descent: Stochastic appr.

Robbins-Monro algorithm

Task is to compute root of $f(\theta) = \mathrm{E}_{\tau\sim p(\tau)}[\varphi(\theta, \tau)]$

If $p(\tau)$ is not known => algorithm:

$\theta_n = \theta_{n-1} - \mu_n \varphi(\theta_{n-1}, \tau_n)$ converges to root of $f(\theta)$ under conditions:

$\sum_n \mu_n^2 < \infty$ and $\sum_n \mu_n \to \infty$

Adaptation to ML tasks

$\tilde{L}(\theta) = \mathrm{E}_{x,y\sim p_{data}}[L(\theta, x, y)]$

We interested in extremum (i.e. root of the corresponding gradient):

$\nabla\tilde{L}(\theta) = \mathrm{E}_{x,y\sim p_{data}}[\nabla L(\theta, x, y)]$ => $\theta_n = \theta_{n-1} - \mu_n \nabla L(\theta_{n-1}, x_n, y_n)$

C.2 Gradient descent: Advanced SGD

Gradient Descent flavors:

Batch GD: $\theta^{(i)} = \theta^{(i-1)} - \alpha_i \nabla_\theta L\big(Y, f(\mathbf{X}, \theta)\big)$ [all training set]

Redundant computations (many correlated examples in dataset)

Online GD: $\theta^{(i)} = \theta^{(i-1)} - \alpha_i \nabla_\theta L\big(y, f(\mathbf{x}, \theta)\big)$ [one example]

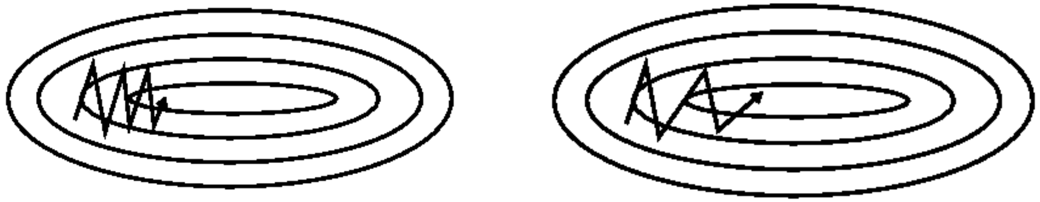
Variance is too big and not speed up from matrix computations

Mini-batch GD: $\theta^{(i)} = \theta^{(i-1)} - \alpha_i \nabla_\theta L\big(Y_m, f(\mathbf{X}_m, \theta)\big)$ [mini-batch]

Most widely-used now

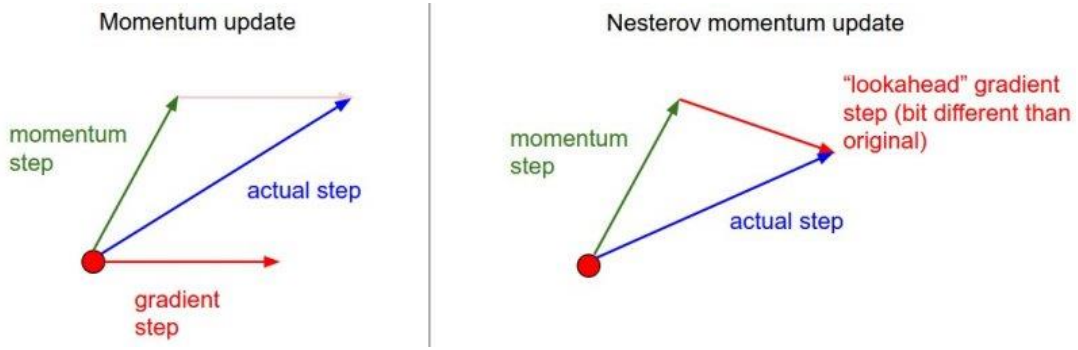
C.2 Gradient descent: Advanced SGD

Selecting better learning schedule



1. Momentum

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



2. Nesterov accelerated gradient

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

* Images from
http://sebastianruder.com/optimizing-gradient-descent/
http://cs231n.github.io/neural-networks-3/#update

C.2 Gradient descent: Advanced SGD

Annealing (scheduling) learning rate

1. Step decay:

Reduce the learning rate by some factor every few epochs.

2. Exponential decay

$\alpha_t = \alpha_0 \exp(-kt)$

3. 1/t decay

$\alpha_t = \alpha_0/(1 + kt)$

4. Linear decay

Reason: to prevent "bouncing" around minimums.

* Image from http://sebastianruder.com/optimizing-gradient-descent/

C.2 Gradient descent: Advanced SGD

Per-parameter adaptive learning rate methods

1. Adagrad

$c = c + [\nabla_\theta L(\theta)]^2$ *[this is a vector of sum of squared gradients]*

$\theta^{(i)} = \theta^{(i-1)} - \alpha_i \nabla_\theta L(\theta) / (\sqrt{c} + \varepsilon)$ *[ε is for numerical stability]*

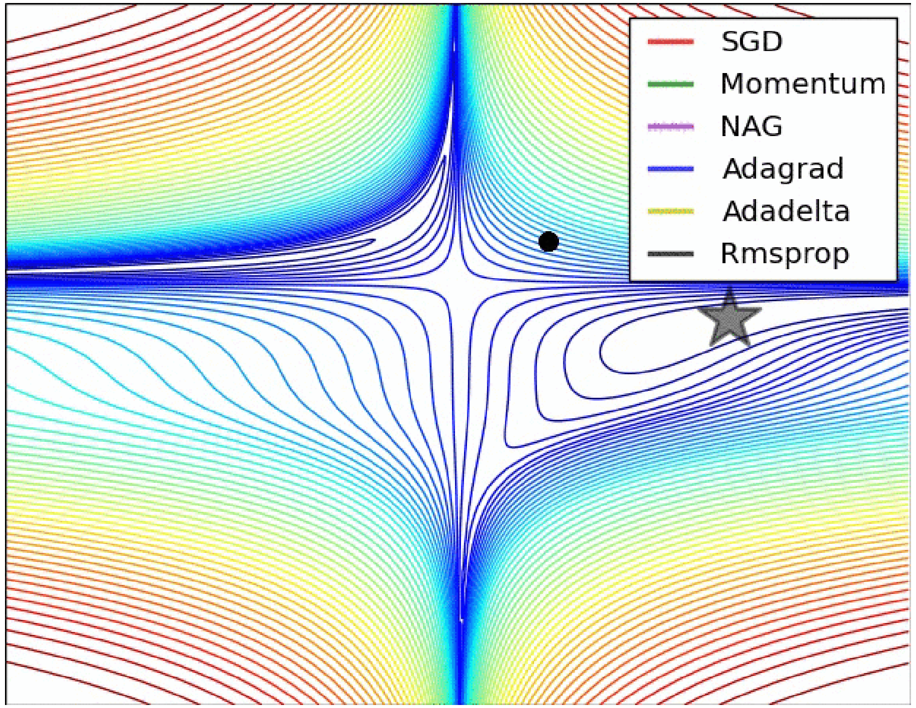
2. RMSprop

$c = \gamma c + (1 - \gamma)[\nabla_\theta L(\theta)]^2$ *[this is a vector of av. squared gradients]*

$\theta^{(i)} = \theta^{(i-1)} - \alpha_i \nabla_\theta L(\theta) / (\sqrt{c} + \varepsilon)$ *[ε is for numerical stability]*

C.2 Gradient descent: Advanced SGD

Different SGD techniques, demonstration



3. Adam

$$b = \alpha c + (1-\alpha)\nabla_\theta L(\theta)$$
$$c = \gamma c + (1-\gamma)[\nabla_\theta L(\theta)]^2$$
$$\theta^{(i)} = \theta^{(i-1)} - \alpha_i b/(\sqrt{c} + \varepsilon)$$

* Picture from http://sebastianruder.com/optimizing-gradient-descent/

D. Regularization

Overfitting problem

To prevent neural network from “memorizing” inputs.

[Number of parameters >> with numbers of training examples]

Techniques

1. Early stopping: stop training once validation error starts to increase.
2. L2 regularization

L2 regularization

Regularization L2: control over weights in neural network.

L2 regularization: add term $\frac{\lambda}{2n}||\omega||^2$ to cost function ($\lambda > 0$).

D. Regularization

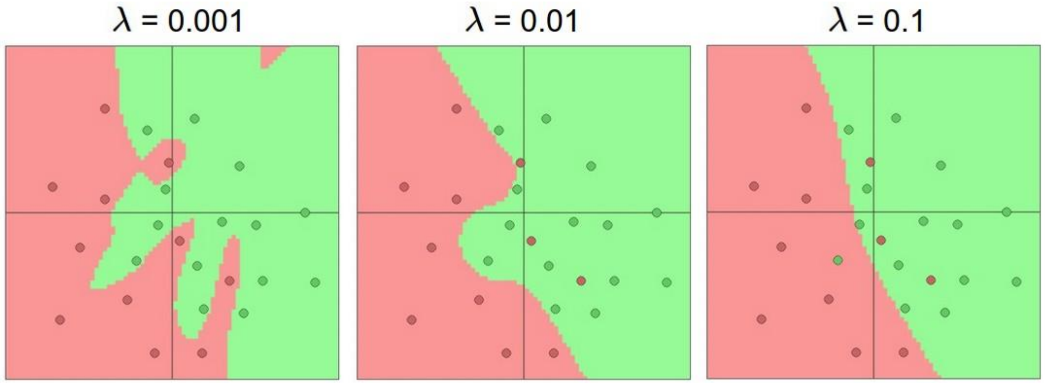
L2 regularization ("weight decay")

Ideas behind regularization:

to make model "simpler"

to make model more stable to random noise in input data

fewer weights ⇒ smaller VC dimension

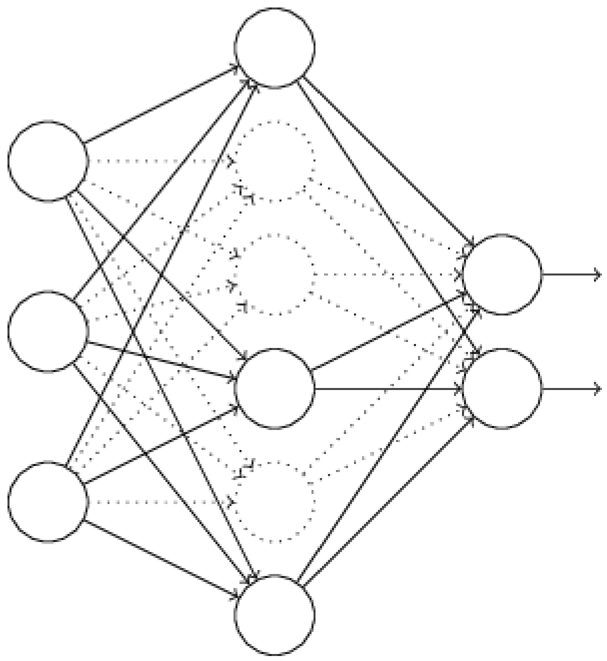


* Image from http://cs231n.github.io/neural-networks-1/#power

D. Regularization

Dropout:

1. Randomly and temporarily delete neurons
[forward and backward passes through modified NN]
2. Repeat for another mini-batch (select new subset of neurons for deletion).



Intuition behind procedure:

Training different neural networks

* Image from http://neuralnetworksanddeeplearning.com/chap3.html

D. Regularization

Batch normalization:

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Normalizing activations of layer.
γ, β are learnable parameters.

Robust to “bad” initializations of weights.

Address internal covariance shift problem for deeper layers of NN.

* Algorithm taken from original article https://arxiv.org/pdf/1502.03167.pdf

D. Regularization

Other regularization techniques

1. Batch normalization
2. L1 regularization (weights selection)
3. Dataset augmentation (object recognition in CV, NLP)
4. Adding noise to inputs / hidden / weights / targets and labels smoothing
5. Pre-training

E. Baseline models

After the simplest approaches tried…

Neural networks

1. Type of input determines general structure of NN:

    Images => CNN with common sets of layers

    Sequential data => LSTM / GRU with FC layers

    Other => few FC layers

2. Activation function: ReLU (leaky ReLU)

3. Optimizer: Adam (RMSprop, SGD with momentum and decaying lr)

4. Regularizer: Batch Normalization and early stopping (Dropout)

F. Practical assignments

Practical assignment #2: starting 12 Apr

Working with simplified version of Diabetic Retinopathy Detection competition on Kaggle.

Work with 512x512 images. Images already preprocessed. Baseline solution is provided.

Ref: https://github.com/5vision/miptmlcourse

Practical assignment #3: starting 19 Apr

Detection of types of physical activities.

Next week

Lecture 8 “Convolutional neural networks”

1. Motivation and premises
2. Key features of CNN architecture
3. Examples of CNNs

D. Homework

For all

1. Reading Ch.7-8 in [2].
2. Practical assignment #2.

Some cool reading:

http://sebastianruder.com/optimizing-gradient-descent/

http://distill.pub/2017/momentum/

http://blog.smola.org/post/4110255196/real-simple-covariate-shift-correction

Refs

1. Thorough review of relevant math topics:

http://info.usherbrooke.ca/hlarochelle/ift725/review.pdf

2*. Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning.

3. Kevin P. Murphy, Machine Learning: A probabilistic perspective.

4. David Barber, Bayesian Reasoning and Machine Learning.

5. Sergios Theodoridis, Machine Learning: A Bayesian and optimization perspective.

6*. See also refs in practical assignment and online courses, especially one from Hugo Larochelle (presentation from lecture 1).