Deep Learning

Thierry Artières

Ecole Centrale Marseille - Option DIGITALE

October 16, 2019

Outline

MLP = Universal approximators

One layer is enough !

- Theorem [Cybenko 1989]: Let $\phi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $\mathcal{C}(I_m)$. Then, given any $\epsilon > 0$, there exists an integer N, such that for any function $f \in \mathcal{C}(I_m)$, there exist real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \phi\left(w_i^T x + b_i\right)$$

  as an approximate realization of the function f where f is independent of ϕ ; that is : $|F(x) - f(x)| < \epsilon$ for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $\mathcal{C}(I_m)$.

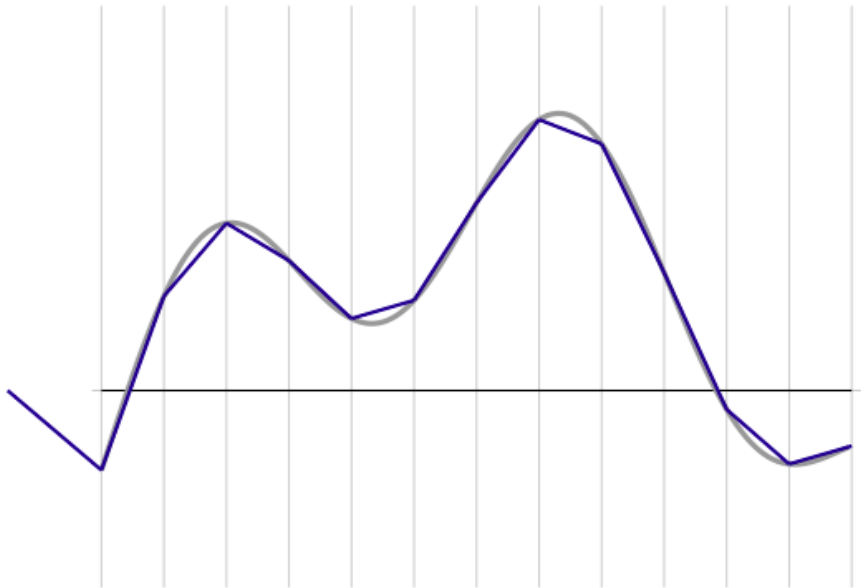
- Existence theorem only
- Many reasons for not getting good results in practice

MLP = Universal approximators: Proof (From Francois Fleuret slides)

simple real function

- Any function in $\mathcal{C}([a, b], \mathbb{R})$ may be approximated with any desired prevision with a linear combination of translated / scaled Relu functions.

$$f(x) = w_1 \times R(x - a_1) + w_2 \times R(x - a_2) + ...$$



MLP = Universal approximators: Proof (From Francois Fleuret slides)

More general functions in $\Psi \in \mathcal{C}([0,1]^D, \mathbb{R})$

- Previous result hods for sin function

$$\forall A > 0, \epsilon > 0, \exists N, (\alpha_n, a_n) \in \mathbb{R} \times \mathbb{R}, n = 1, ..., N,$$

$$\text{s.t.} \max_{x \in [-A,A]} |\sin(x) - \sum_{n=1}^{N} \alpha_n R(x - a_n)|| \leq \epsilon$$

- Density of Fourier series

$$\forall \Psi, \delta > 0, \exists M, (v_m, \gamma_m, c_m) \in \mathbb{R}^D \times \mathbb{R} \times \mathbb{R}, m = 1, ..., M,$$

$$\text{s.t.} \max_{x \in [0,1]^D} |\Psi(x) - \sum_{m=1}^{N} \gamma_m \sin(v_m.x - c_m)| \leq \delta$$

- Result easily follows

Optimal learning rate and convergence speed

Second order point of view

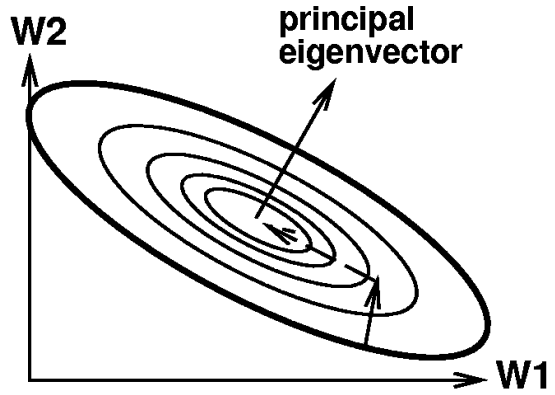
- Taylor expansion, noting $\nabla^2 C(w)$ the Hessian (a $N \times N$ matrix with N a model with parameters)

$$\nabla C(w)|_{w'} = \nabla C(w)|_w + \nabla^2 C(w)(w' - w)$$

- $\rightarrow$ optimum rule (setting $\nabla C(w)|_{w'}$ to 0):

$$w' = w - (\nabla^2 C(w))^{-1} \nabla C(w)$$

- Optimal move not in the direction of the gradient
- In Order 1 Gradient descent the optimal the optimal value of ϵ depends on eigen values of the Hessian $\nabla^2 C(w)$



[Lecun et al, 93]

Optimization routines

Many SGD variants popular in DL

- SGD with Momentum
- Nesterov accelerated gradient
- Adragrad
- Adadelta
- RmsProp
- ...

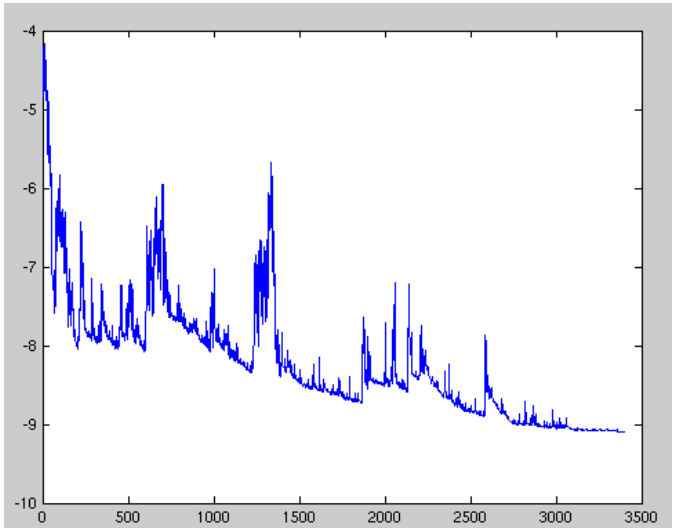
Using Momentum

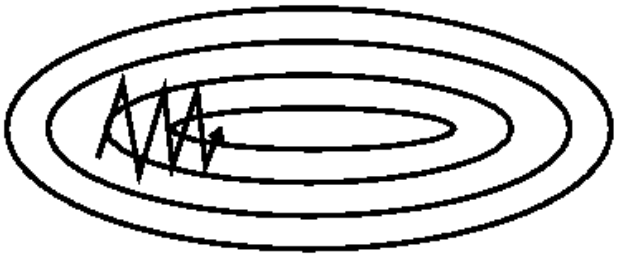
SGD with Momentum

- Standard Stochastic Gradient descent : $w = w - \epsilon \frac{\partial C(w)}{\partial w}$
- SGD with Momentum:

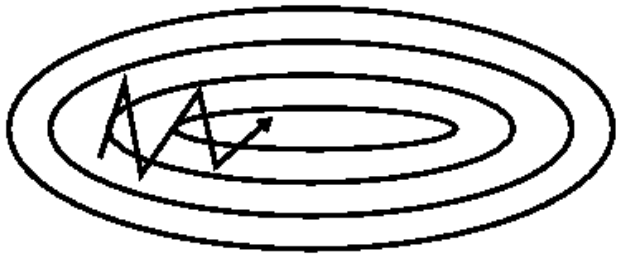
$$v = \gamma v + \epsilon \frac{\partial C(w)}{\partial w}$$

$$w = w - v$$





SGD standard



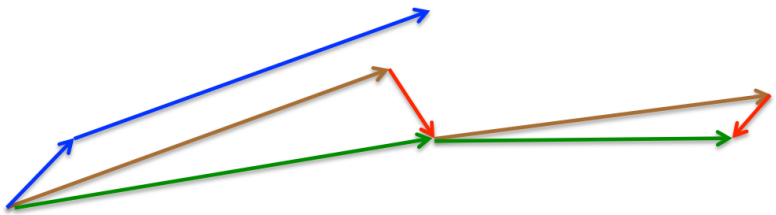
SGD avec momentum

Nesterov Accelerated Gradient

Principle

- Idea: Better anticipate when to slow down by looking forward

$$v_{t+1} = \gamma v_t + \epsilon \nabla C(w)|_{w_t - \gamma v_t}$$
$$w_{t+1} = w_t - v_{t+1}$$



- Blue vectors: standard momentum
- Brown vectors: jump
- Red vectors: correction
- Green vectors: accumulated gradient

Adagrad

Reminder: Optimally one needs to adapt the learning rate to every weight

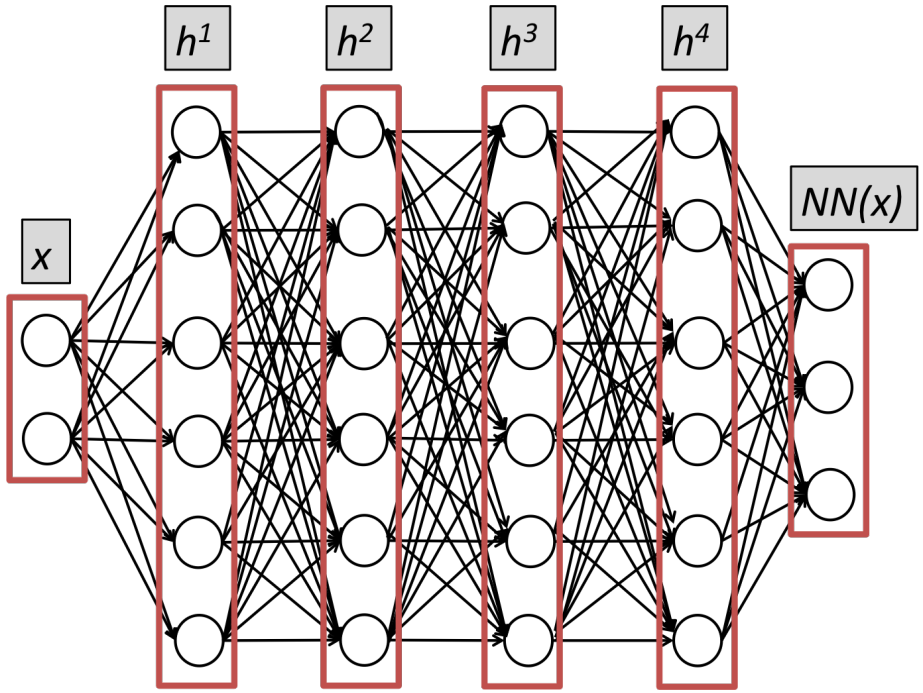
- Define $g_{t,i} = \frac{\partial C(w)}{\partial w_i}$ the derivative wrt a single weight value w_i
- $w_{t+1,i} = w_{t,i} - \frac{\epsilon}{\sqrt{G_{t,ii}+\gamma}} g_{t,i}$
  - where $G_{t,ii}$ is a diagonal matrix with i^{th} element equal to $\sum_t g_{t,i}^2$
  - γ is a very small value to avoid numerical exceptions
  - Standard value $\epsilon = 0.01$
- Variants that aim at minimizing the aggressive feature of Adagrad: Adadelta , Adam, and RmsProp

Outline

Deep Learning = Representation Learning

Hierarchy of representation spaces by successive hidden layers

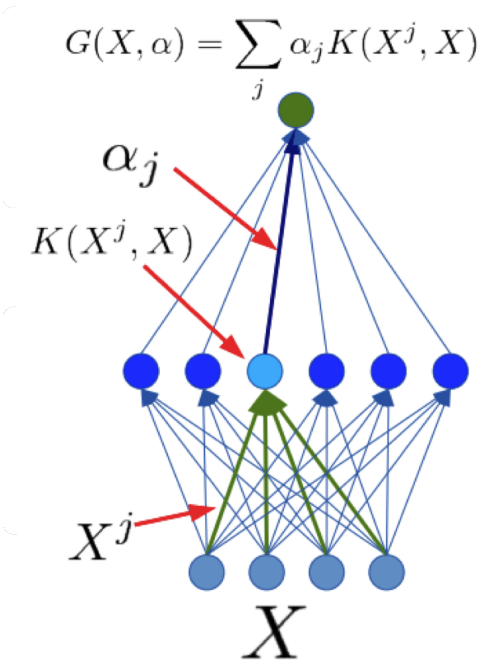
$$h^i(x) = g(W^i \times h^{i-1}(x))$$



Shallow vs deep models

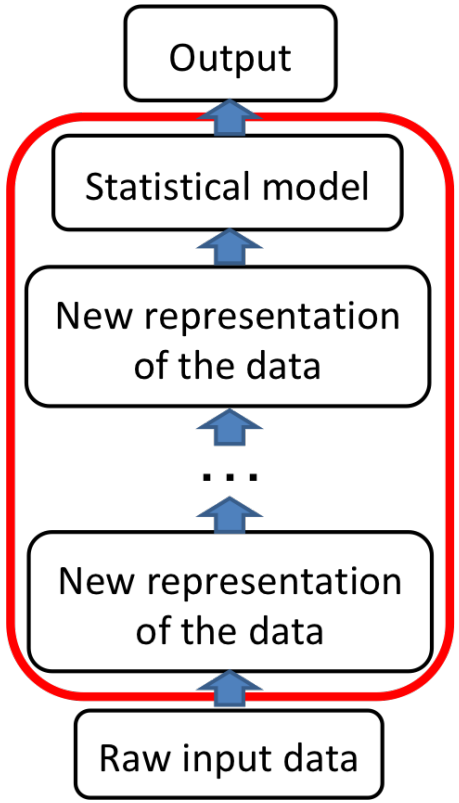
From [LeCun tutorial Statlearn]

- Neural Networks with one hidden layer are shallow models
- SVMs are shallow models



DL

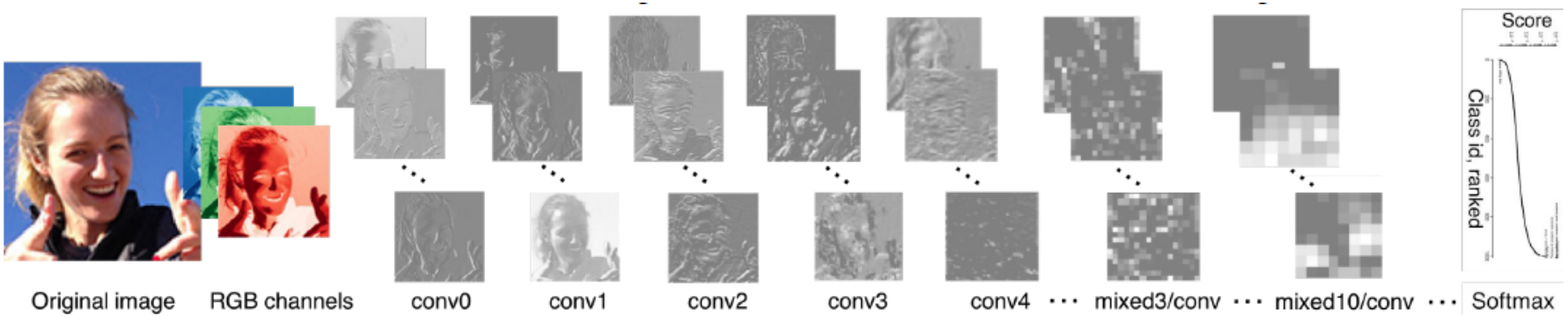
- Joint learning of a hierarchy of representations and of a prediction model



Feature hierarchy : from low to high level

What feature hierarchy means ?

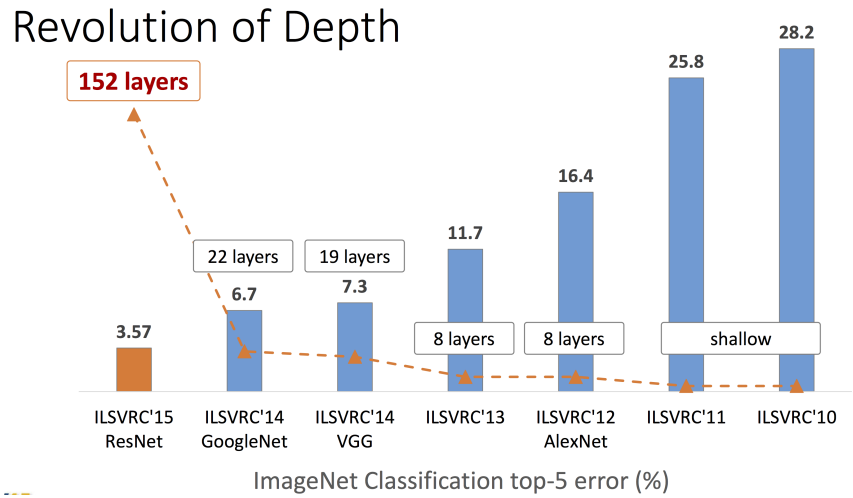
- Low-level features are shared among categories
- High-level features are more global and more invariant



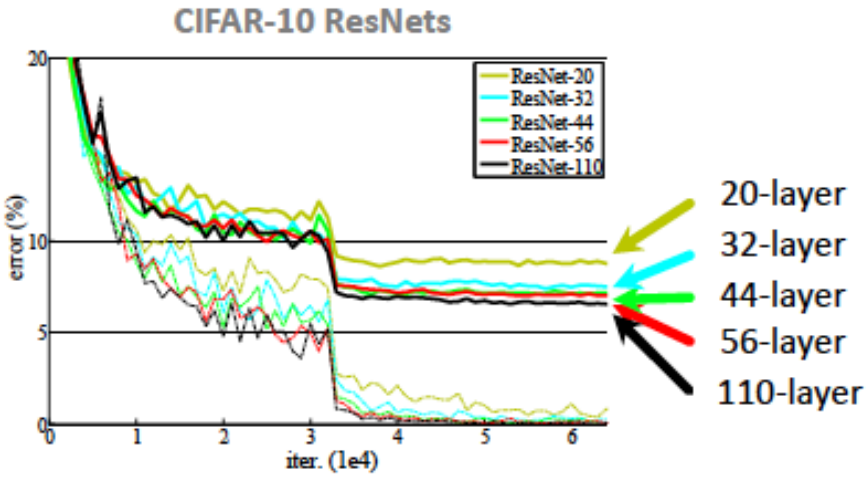
[From Taigman et al., 2014]

Outline

Deep networks are powerful



[Kaiming He]



(Dense) Deep vs Shalow: Increased capacity

The power of depth [Eldan and Shamir, 2016]

- There is a simple function expressible by a 3-layer network that may not be approximated by a 2-layer network to more than a certain accuracy unless its width is exponential in the input dimension

Characterizing the complexity of functions a DNN may implement [Pascanu and al., 2014]

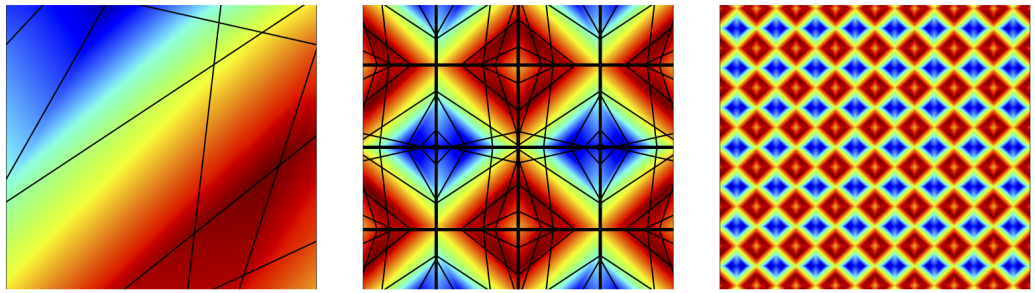
- DNNs with RELU activation function $\Rightarrow$ piecewise linear function
- Capacity as a function of the number of linear regions one may divide the input space
- Exponentially more regions per parameter in terms of number of HL
  - Case of p_0 inputs and $p = 2p_0$ hidden cells per HL (with k HL) :
    - Maximum number of regions at least : $2^{(k-1)p_0} \sum_{j=0}^{p_0} \binom{2p_0}{j}$

Deep vs Shalow ?

Characterizing the complexity of functions a DNN may implement [Pascanu and al., 2014]

- DNNs with RELU activation function $\Rightarrow$ piecewise linear function
- Complexity of DNN function as the Number of linear regions on the input data
- Case of n_0 inputs and $n = 2n_0$ hidden cells per HL (k HL) :
  - Maximum number of regions : $2^{(k-1)n_0} \sum_{j=0}^{n_0} \binom{2n_0}{j}$
- Example: $n_0 = 2$
  - Shallow model: $4n_0$ units $\rightarrow$ 37 regions
  - Deep model with 2 hidden layers with $2n_0$ units each $\rightarrow$ 44 regions
  - Shallow model: $6n_0$ units $\rightarrow$ 79 regions
  - Deep model with 3 hidden layers with $2n_0$ units each $\rightarrow$ 176 regions
- Exponentially more regions per parameter in terms of number of HL
- At least order (k-2) polynomially more regions per parameter in terms of width of HL n

Deep vs Shalow ?



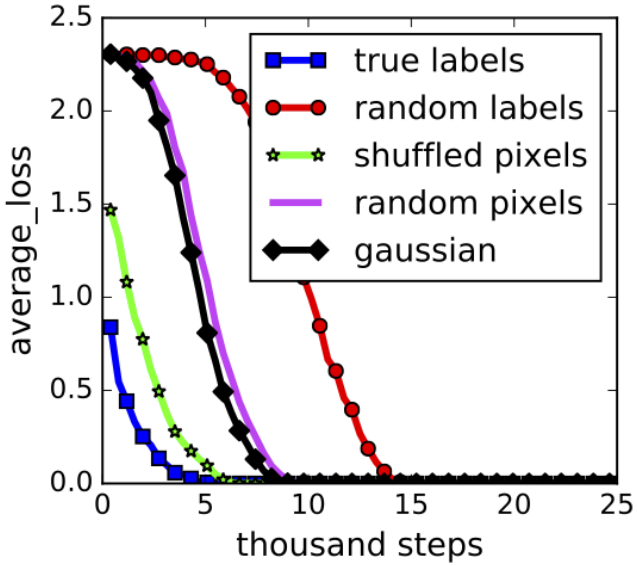
From [Pascanu and al., 2014]

- Left: Regions computed by a layer with 8 RELU hidden neurons on the input space of two dimensions (i.e. the output of previous layer)
- Middle: Heat map of a function computed by a rectifier network with 2 inputs, 2 hidden layers of width 4, and one linear output unit. Black lines delimit regions of linearity of the function
- Right: Heat map of a function computed by a 4 layer model with a total of 24 hidden units. It takes at least 137 hidden units on a shallow model to represent the same function.

DNNs are overparameterized

Large DNNs may even learn noise

- For instance : Learn after random permutation of the labels of the training samples
- It learns, but it takes more time...
- Note that the same (large) architectures that may learn random labels generalize well when trained on non perturbated data

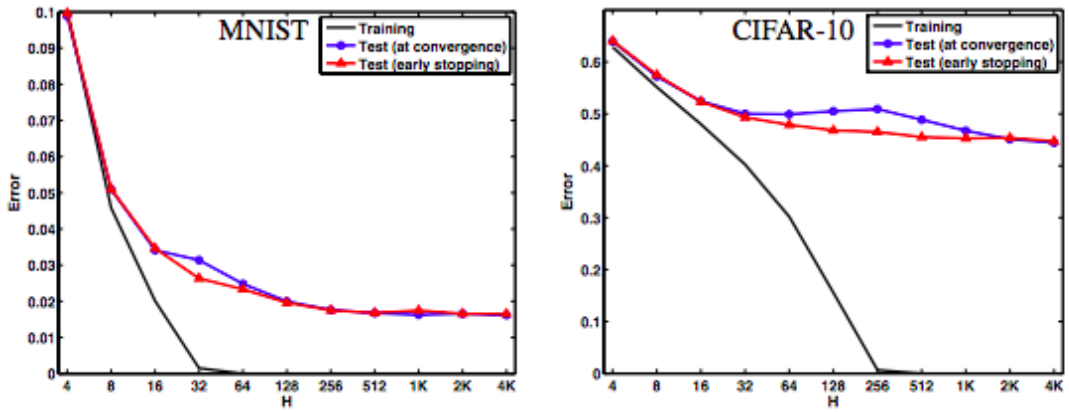


[Zhang and al., 2017]

DNNs and overfitting

Actually NNs do not easily overfit

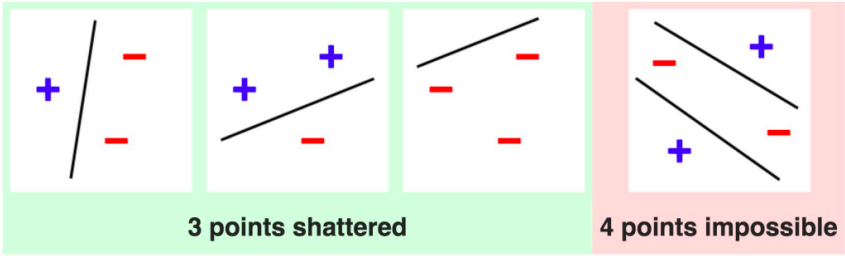
- The more you learn the better it generalizes
- Experiments on Mnist and CIFAR data (downsampled): 1 hidden layer (size H) NNs without any regularization $\rightarrow$ no overfitting observed



[Neyshabur 2017]

Capacity

Vapnik dimension



Rademacher capacity

$$R_n(H) = E_\sigma[sup_{h \in H} \frac{1}{n} \sum_{i=1}^{n} \sigma_i h(x_i)]$$

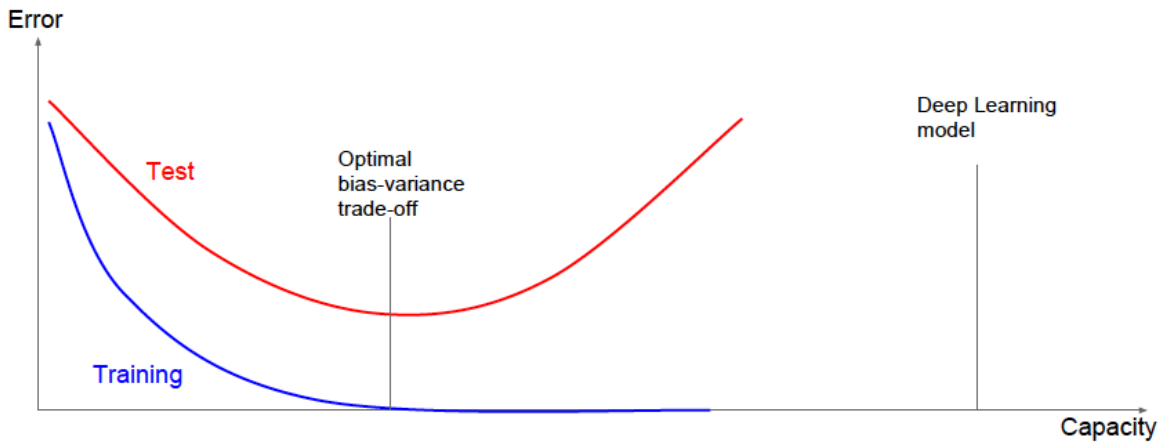
with $\sigma_i \in \{-1, 1\}$

- Clearly looks like the randomization test
- Trivial upperbound (=1): useless

DNNs' capacity

Vapnik dimension of deep NNs with ReLU

- With L hidden layer of p neurons the Vapnik dimension of deep ReLU NNs is $h = \Theta(L^2 p^2)$
- Considering classical generalization bound : $R(w) \leq R_{emp}(w) + \tilde{O}(\sqrt{\frac{L^2 p^2}{n}})$
- This does not explain generalization behavior

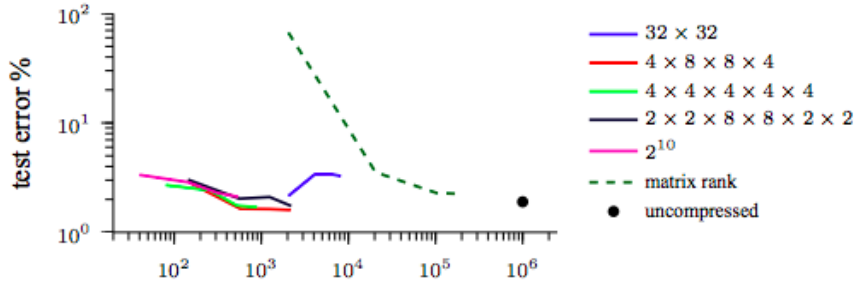


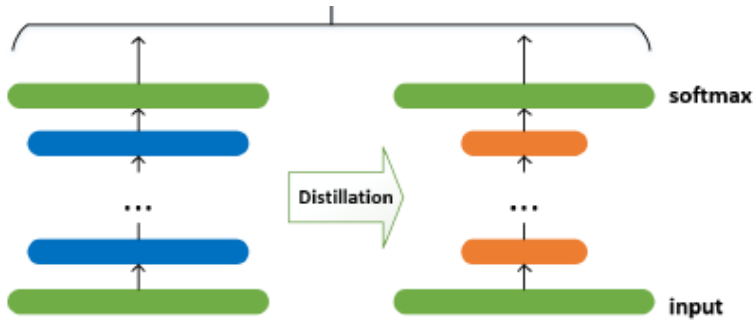
[O. Bousquet, tutorial 2017]

Deep nets do not actually need to be huge

Size helps learning but one may simplify once learned !

- Low rank tensor approximation (CP, Tucker, TensorTrain) of layer weight matrices (FC, Conv, RNN) [Novikov et al. 2015]
- Distillation strategy [Hinton et al., 2015]
  - Learn a deep and complex model f_{NN} (or en ensemble of deep models) on a dataset D
  - Create a new learning task by computing the output vectors o of f_{NN} for samples in D (better use logits than outputs of the softmax)
  - Learn a narrower model to predict o vectors for samples in D

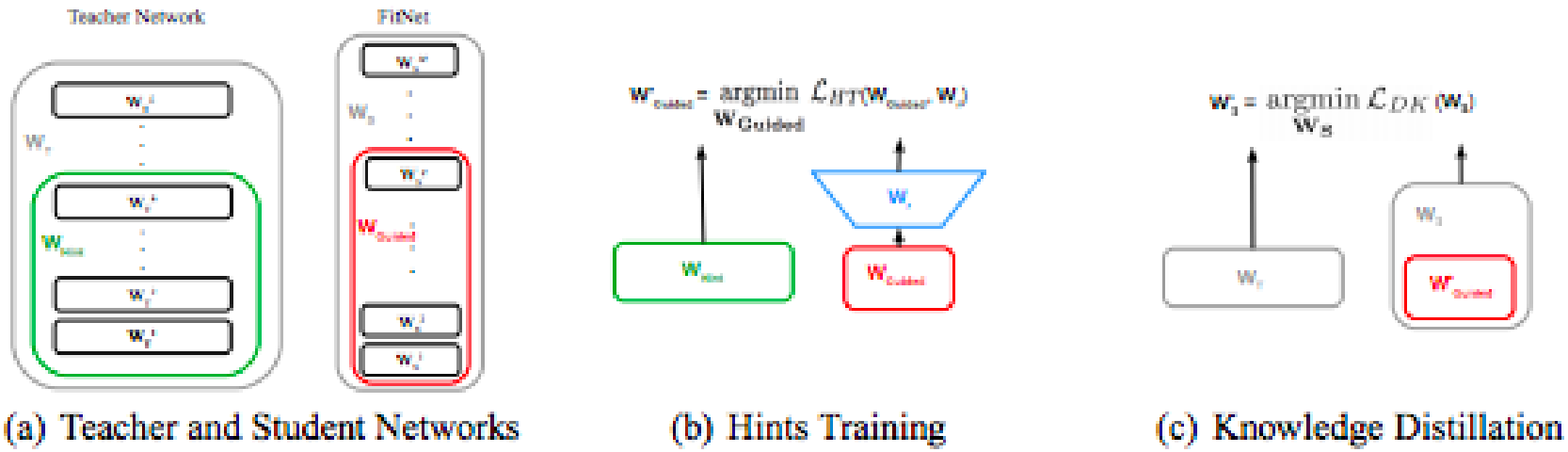




FitNets [Romero et al., 2015]

Going further in distillation with intermediate transfer

- Knowledge distillation + intermediate distillation losses



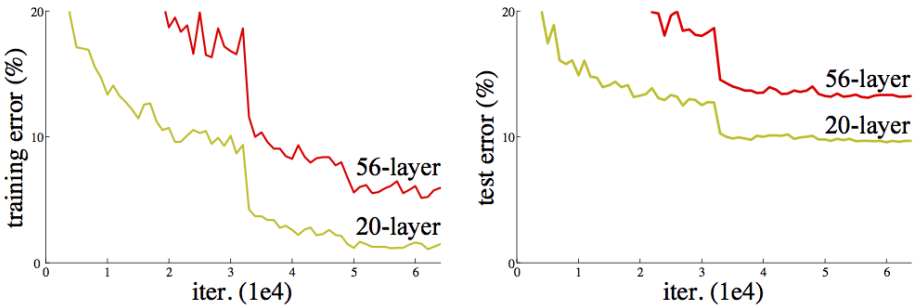
(a) Teacher and Student Networks (b) Hints Training (c) Knowledge Distillation

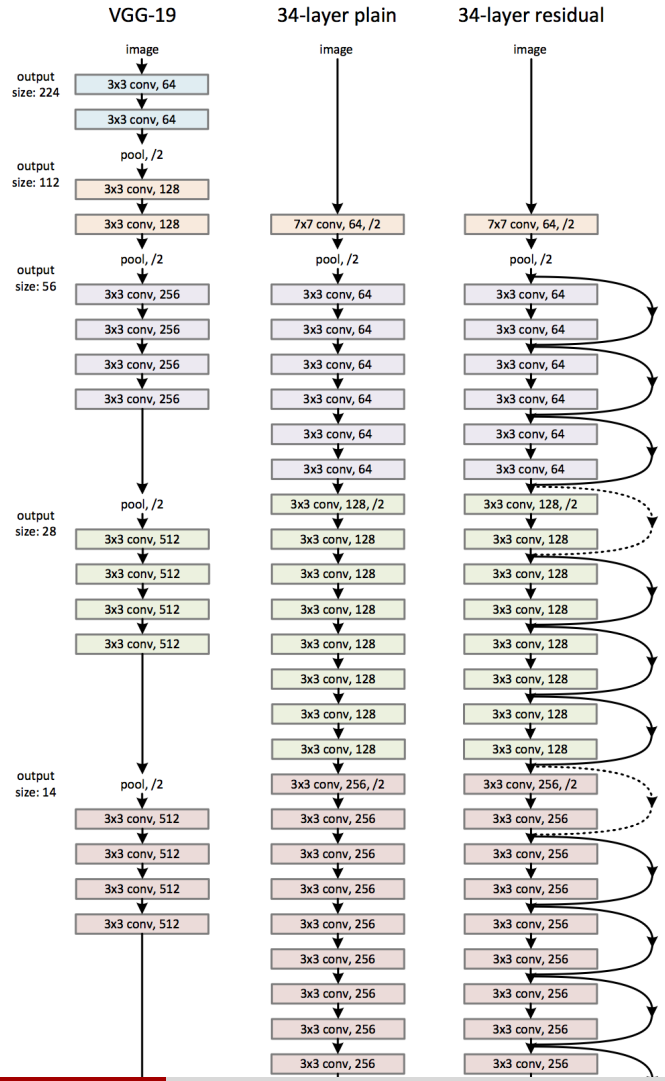
Outline

From shalow to deep

Simply stacking layers does not work (CIFAR results) ! (figures form [He and al., 2015])

...

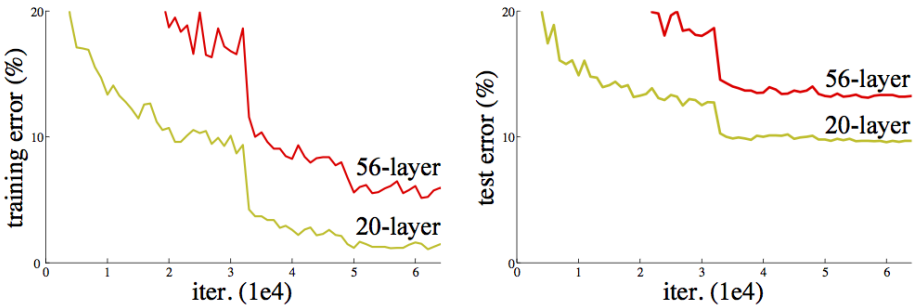


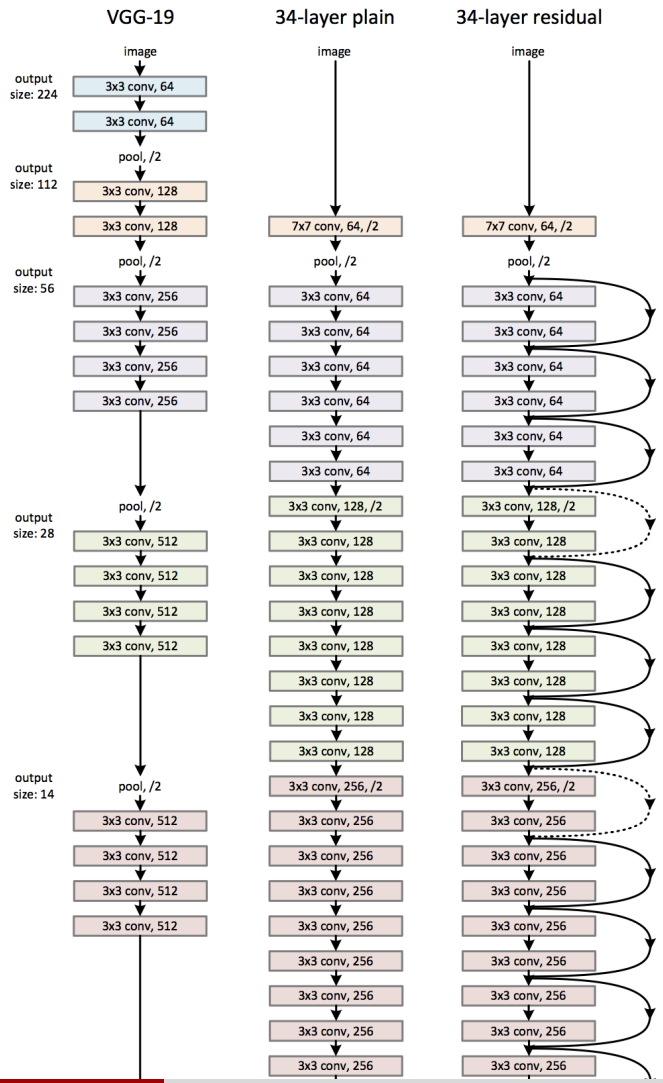


From shalow to deep

Simply stacking layers does not work (CIFAR results) ! (figures form [He and al., 2015])

...





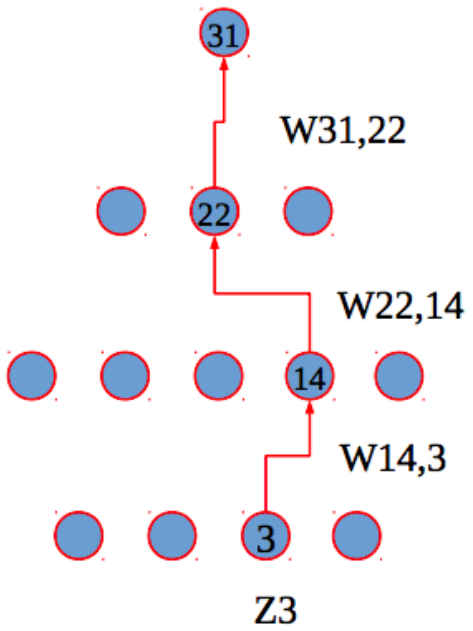
While NN optimization is reasonably easy

(Easier) analysis for ReLU DNNs [LeCun Statlearn tutorial]

- ... but expectation of generalized results to other activation functions
- With ReLu and MaxPooling operators one may formalize what happens on a path from an input to the output
- The output may be computed as :

$$\hat{y} = \sum_{P} \delta_P(W, X)(\prod_{(ij)\in P} w_{ij}) x_{j_{start}}$$

- $\delta_P(W, X)$: 1 if active path, 0 otherwise
- Implemented function is piece-wise linear



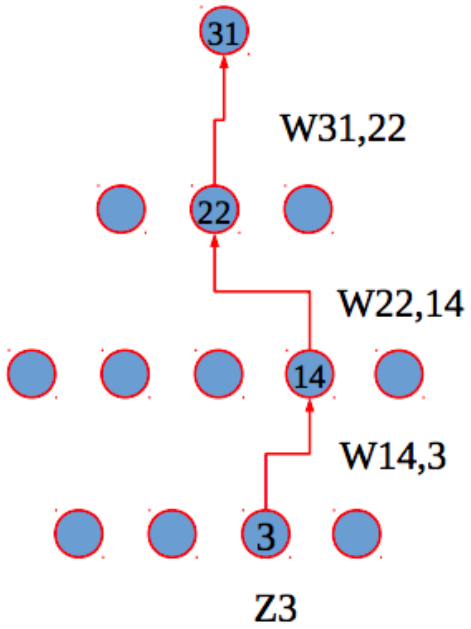
While NN optimization is reasonably easy

(Easier) analysis for ReLU DNNs [LeCun Statlearn tutorial]

- Objective function : piece wise polynomial (degree = number of hidden layers) with partially random coefficients

$$C(W) = \sum_{P} C_P(X, Y, W)\left(\prod_{(ij)\in P} w_{ij}\right)$$

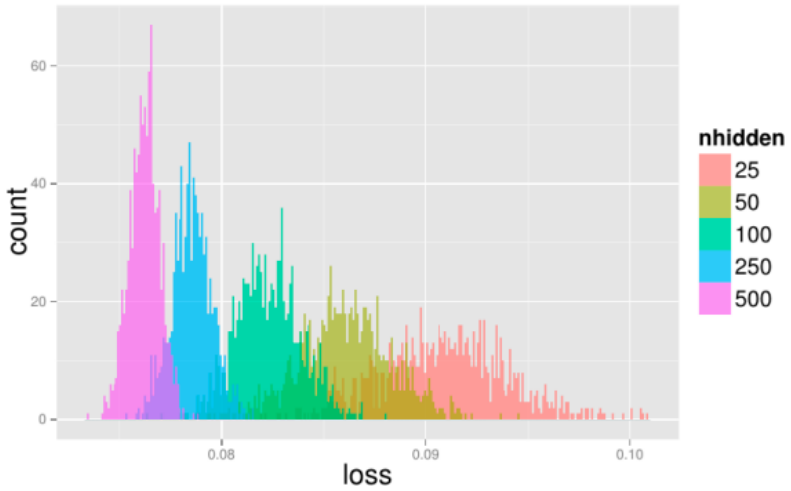
- Hint from results on distribution of critical points for polynomials with random coefficients



Deep ReLu networks

Easier analysis... [LeCun Statlearn tutorial]

- Experiments by [Choromanska and al., 2015]: Train 2-layers nets on Mnist from multiple initializations and measure loss on the test set
- Many close local minimas for large nets
- Objective function do not exhibit lots of saddle points and most local minima are good and close to globale minimas



[Choromanska, Henaff, Mathieu, Ben Arous, LeCun 2015]

Yet the depth alone is not enough

While SGD works well for shallow NNs, the optimization of DNNs requires careful design and tricks

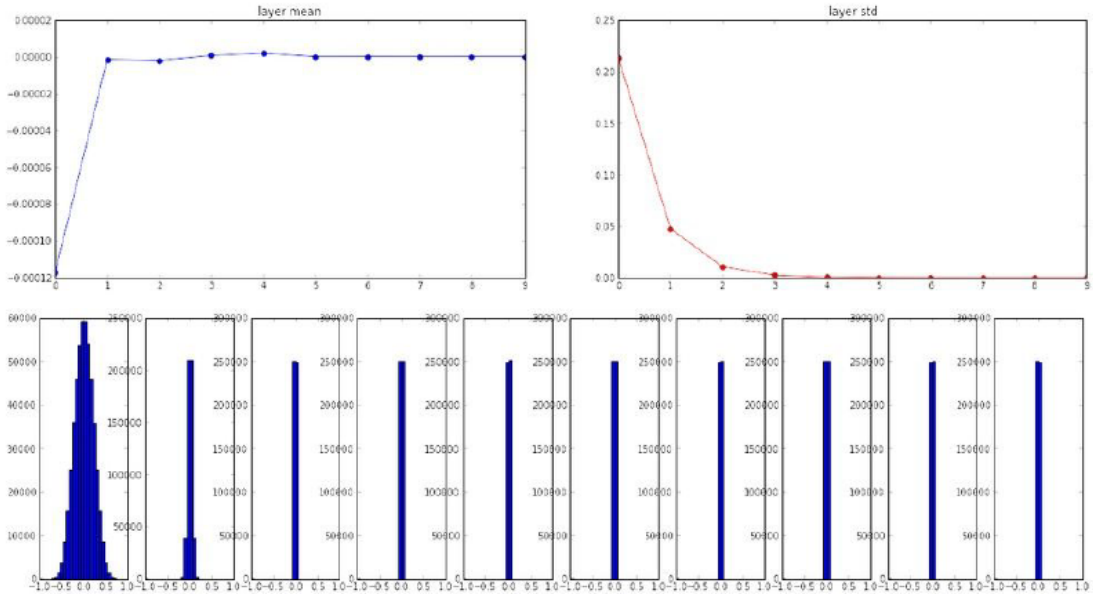
- Make the gradient flow
  - Use normalization strategies (e.g. Batch Normalization)
  - Use auxiliary losses
  - Dropout regularization
  - Include structural constraints like Including the identity mapping as a possible path from the input to the output of a layers (e.g. ResNet building block [He and al., 2015]])

Activity propagation in deep NNs [He et al., 2016]

Few slides from *Fei Fei Li*

Standard initialization schema for MLPs

- 10 layers networks (500 neurones each, with tanh)
- Initialization : gaussian random with small (std=0.01) values (what if all null initialization?)
- All activations at 0
- What about the gradient ?

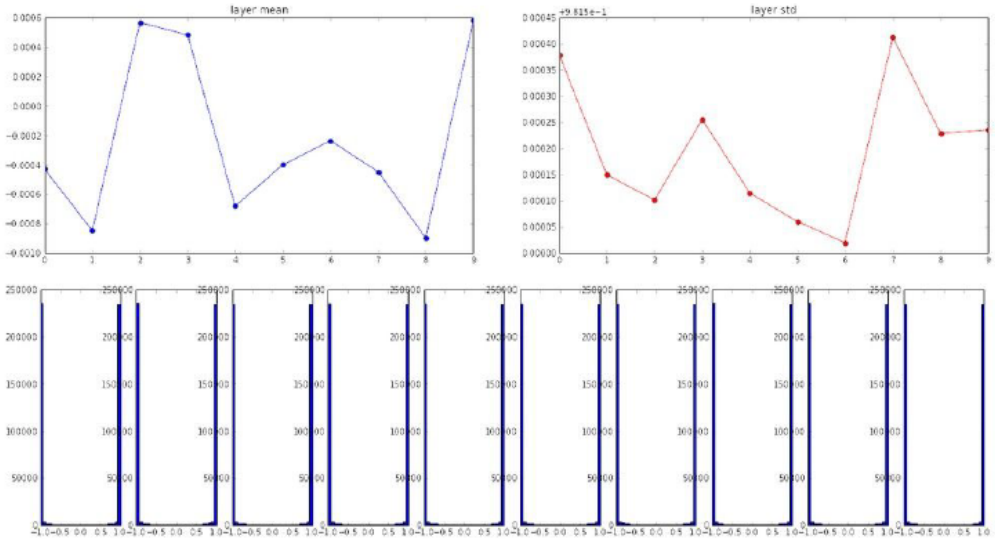


From Fei Fei Li's slides

Tuning weights initialization

Increasing weights initial values comes with neuron saturation problem

- 10 layers networks (500 neurones each, with tanh)
- Initialization : gaussian random with normal (std=1.0) values
- All neurons saturate
- No gradient backpropagated

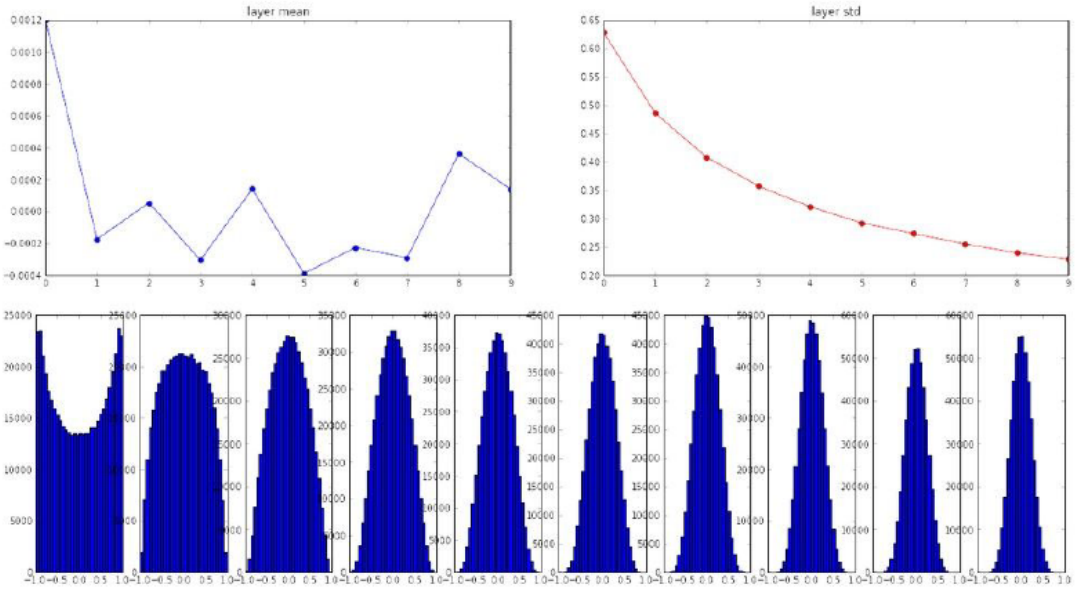


From Fei Fei Li's slides

Smarter intialization

Good (but not enough)

- 10 layers networks (500 neurones each, with tanh)
- Xavier initialization : random gaussian with std dev $= \frac{1}{N_{previouslayer}}$
- Much better behavior but fails with RELU activation (assuming normalized inout data)

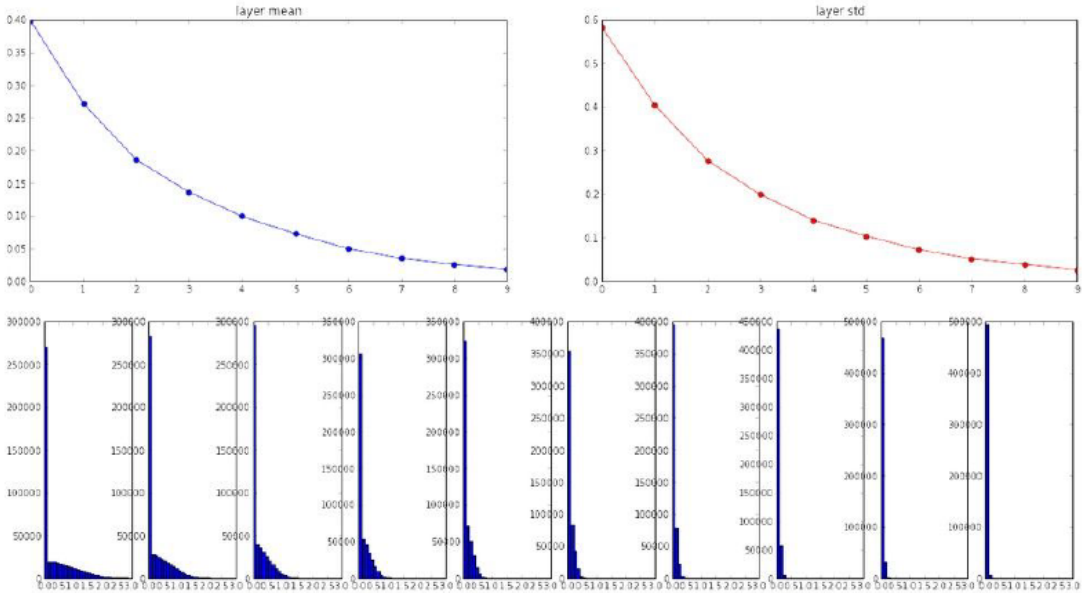


From Fei Fei Li's slides

Smarter intialization

Good (but not enough)

- 10 layers networks (500 neurones each, with tanh)
- Xavier initialization : random gaussian with std dev $= \frac{1}{N_{previouslayer}}$
- Much better behavior but fails with RELU activation (assuming normalized inout data)

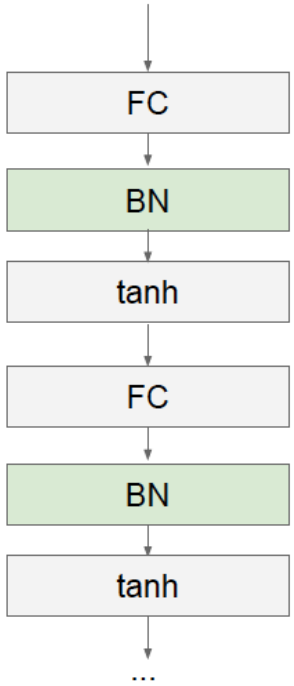


From Fei Fei Li's slides

Batch Normalization

Main idea

- Usually inputs to neural networks are normalized to either the range of [0, 1] or [-1, 1] or to mean=0 and variance=1
- BN essentially performs Whitening to the intermediate layers of the networks.
- Usually placed before nonlinearities



From Fei Fei Li's slides

Batch Normalization

BN layer

- Normalizes the output of a layer by scaling neuron's outputs within a minibatch (of size M)
- For one neuron of the input layer, its output is modified according to:

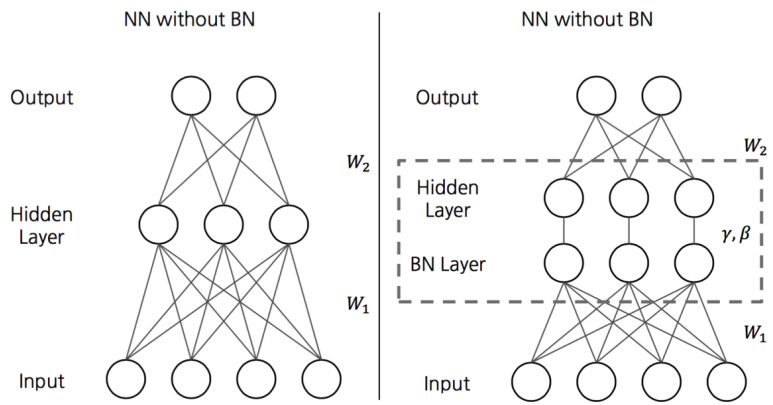
$$\mu_B = \frac{1}{M}\sum_{i=1}^{M} x_i \tag{1}$$

$$\sigma_B^2 = \frac{1}{M}\sum_{i=1}^{M}(x_i - \mu_B)^2 \tag{2}$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \tau}} \tag{3}$$

$$y_i = \gamma x_i + \beta \tag{4}$$

- Use a different computation at inference time (empirical mean and variance computed on the full training set)

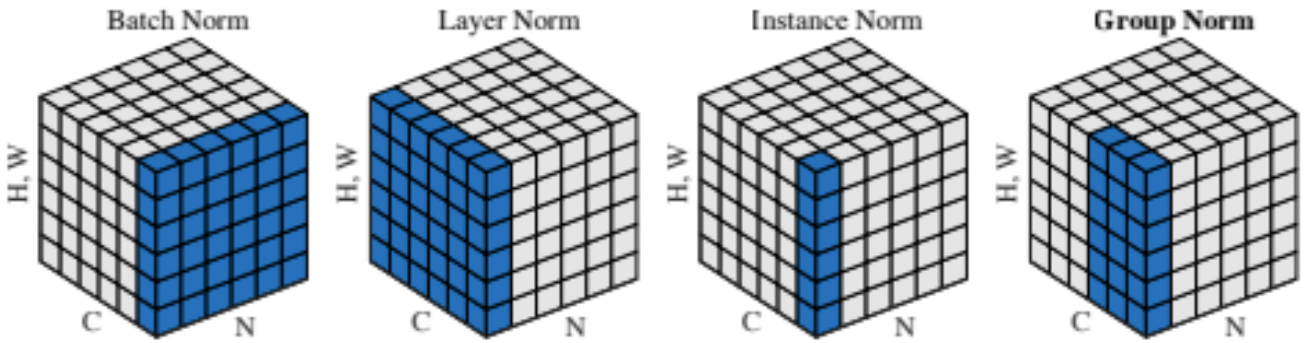


From Fei Fei Li's slides

Other normalization methods

many variants

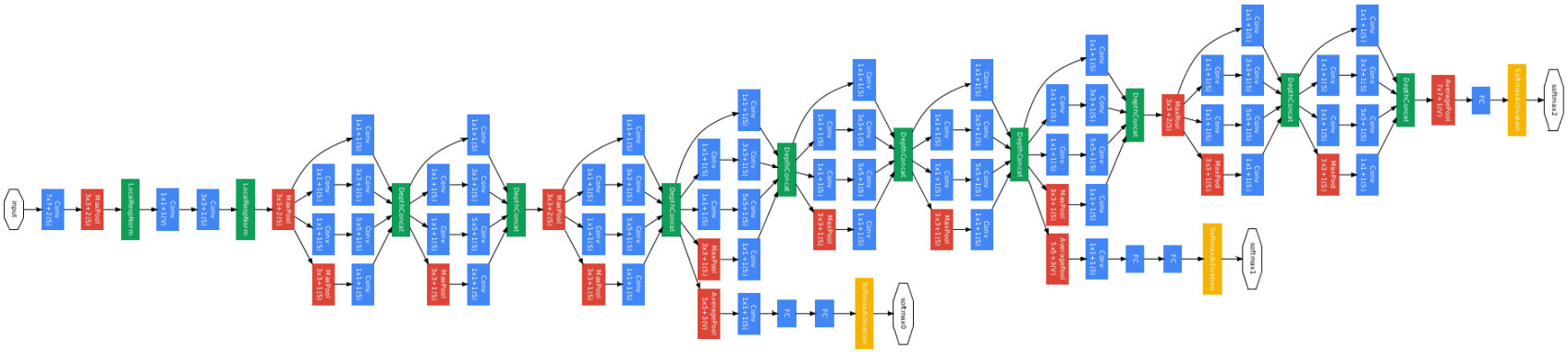
- ...



Auxiliary loss on intermediate layers

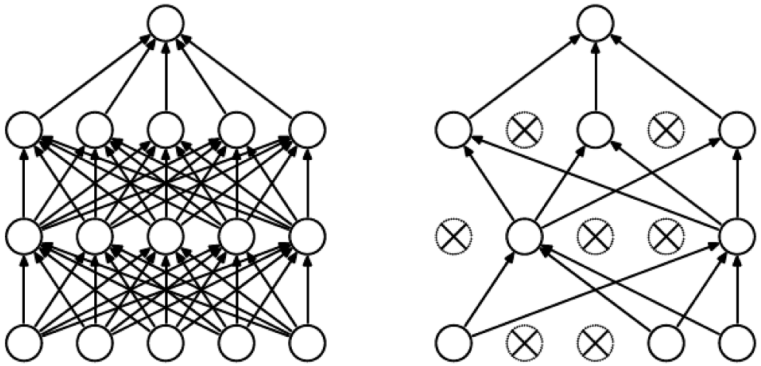
Google net architecture

- Auxiliary loss brings some gradient to first layers



Dropout [Hinton 2012]

Principle

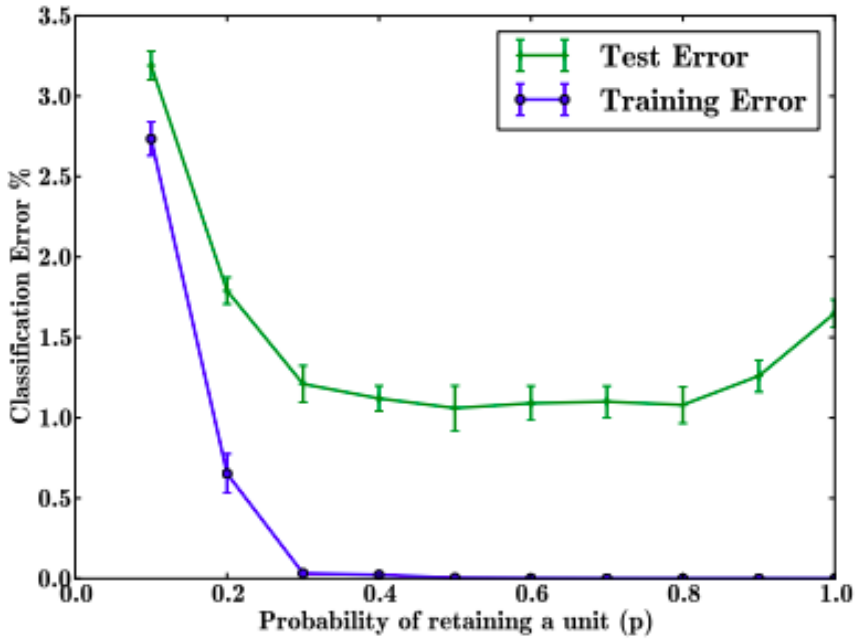


(a) Standard Neural Net (b) After applying dropout.

- First method that allowed learning rellay deep networks without pretraining and smart initialization
- Related to ensemble of models
- Weights are normalized at inference time

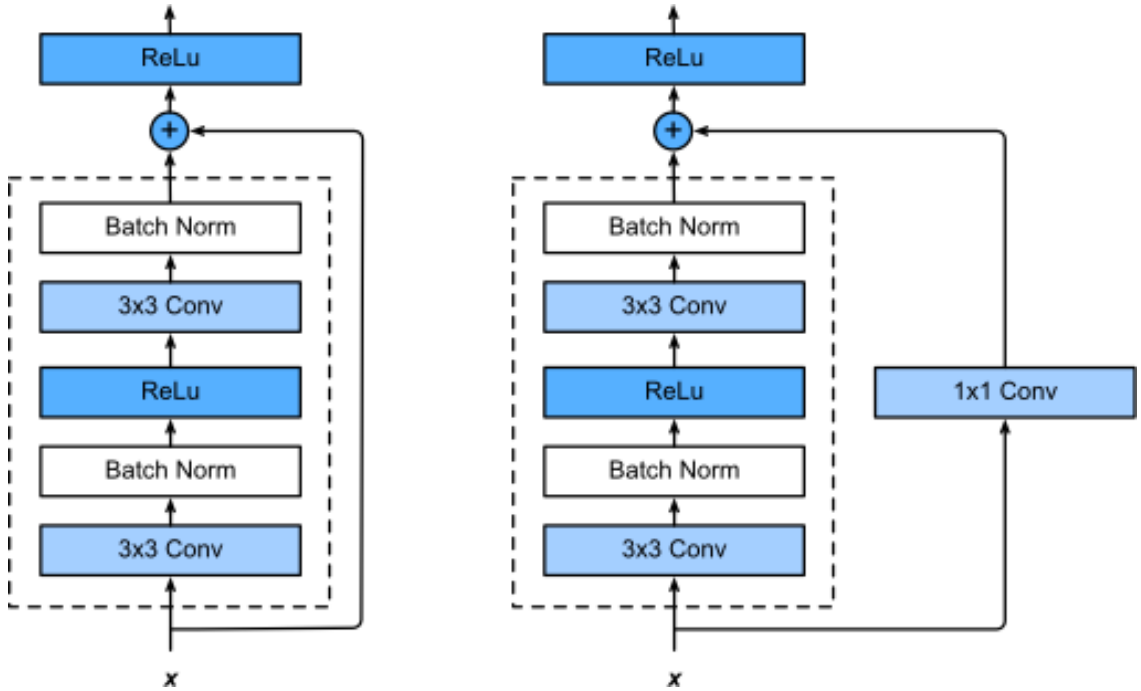
Dropout [Hinton 2012]

Do not ever fear overfitting !



RESNET implementation

Include identity connexions in the architecture



RESNET implementation

Include identity connexions in the architecture

- Standard 2 block layers

```
def Unit(x,filters):

    out = BatchNormalization()(x)
    out = Activation("relu")(out)
    out = Conv2D(filters=filters, kernel_size=[3, 3], strides=[1,
1], padding="same")(out)

    out = BatchNormalization()(out)
    out = Activation("relu")(out)
    out = Conv2D(filters=filters, kernel_size=[3, 3], strides=[1,
1], padding="same")(out)

    return out
```

- ResNet building block [He and al., 2015]

```
def Unit(x,filters):
    res = x
    out = BatchNormalization()(x)
    out = Activation("relu")(out)
    out = Conv2D(filters=filters, kernel_size=[3, 3], strides=[1,
1], padding="same")(out)

    out = BatchNormalization()(out)
    out = Activation("relu")(out)
    out = Conv2D(filters=filters, kernel_size=[3, 3], strides=[1,
1], padding="same")(out)

    out = keras.layers.add([res,out])

    return out
```

Outline

DL vs standard ML

Traditional Machine Learning

- Overfitting is the enemy
- One may control generalization with appropriate regularization
- Suboptimal optimization due to multiple local minima

DL: mysterious phenomenon

- Huge capacity without overfitting
- The size helps learning
- Overfitting idea should be revised for DNNs [Zhand and al., 2017] ?
- Regularization may slightly improve performance but is not THE answer for improving generalization
- Not clear what in the DNN may allow to predict its generalization ability

Réferences

- Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, Neural Machine Translation by Jointly Learning to Align and Translate. CoRR abs/1409.0473 (2014)
- Mickaël Chen, Ludovic Denoyer, Thierry Artières, Multi-View Data Generation Without View Supervision. ICLR, 2018.
- Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, Yann LeCun, The Loss Surfaces of Multilayer Networks. AISTATS 2015
- Ronen Eldan, Ohad Shamir, The Power of Depth for Feedforward Neural Networks. COLT 2016: 907-940
- Clément Farabet, Camille Couprie, Laurent Najman, Yann LeCun: Learning Hierarchical Features for Scene Labeling. IEEE Trans. Pattern Anal. Mach. Intell. 35(8): 1915-1929 (2013)
- Fei Fei Li slides : http://cs231n.stanford.edu/
- Yaroslav Ganin, Victor S. Lempitsky, Unsupervised Domain Adaptation by Backpropagation. ICML 2015: 1180-1189
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition. CVPR 2016: 770-778
- Geoffrey E. Hinton, Oriol Vinyals, Jeffrey Dean, Distilling the Knowledge in a Neural Network. CoRR abs/1503.02531 (2015)
- Reducing the Dimensionality of Data with Neural Networks G. E. Hinton and R. R. Salakhutdinov, VOL 313 SCIENCE, 2006
- Sepp Hochreiter, Jürgen Schmidhuber, Long Short-Term Memory. Neural Computation 9(8): 1735-1780 (1997)
- Krizhevsky, Sutskever, Hinton, ImageNet Classification with deep convolutional neural networks NIPS 2012

Réferences

- Cho, Kyunghyun; van Merrienboer, Bart; Gulcehre, Caglar; Bahdanau, Dzmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, Arxiv, 2014
- LeCun, Bottou, Bengio and Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998
- Lecun tutorial ICML 2013: https://cs.nyu.edu/ yann/talks/lecun-ranzato-icml2013.pdf
- Gilles Louppe, Michael Kagan, Kyle Cranmer, Learning to Pivot with Adversarial Networks. NIPS 2017: 982-991
- Mehdi Mirza, Simon Osindero, Conditional Generative Adversarial Nets. CoRR abs/1411.1784 (2014)
- Guido F. Montúfar, Razvan Pascanu, KyungHyun Cho, Yoshua Bengio, On the Number of Linear Regions of Deep Neural Networks. NIPS 2014: 2924-2932
- Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, Nati Srebro, Exploring Generalization in Deep Learning. NIPS 2017: 5949-5958
- Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, Dmitry P. Vetrov, Tensorizing Neural Networks. NIPS 2015: 442-450
- Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, Jose M. Álvarez, Invertible Conditional GANs for image editing. CoRR abs/1611.06355 (2016)
- Alec Radford, Luke Metz, Soumith Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. CoRR abs/1511.06434 (2015)
- Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B. Tenenbaum, Hugo Larochelle, Richard S. Zemel: Meta-Learning for Semi-Supervised Few-Shot Classification. CoRR abs/1803.00676 (2018)

Réferences

- Marco Túlio Ribeiro, Sameer Singh, Carlos Guestrin, Why Should I Trust You? Explaining the Predictions of Any Classifier. KDD 2016: 1135-1144
- Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, Yoshua Bengio, FitNets: Hints for Thin Deep Nets. CoRR abs/1412.6550 (2014)
- Shreyas Saxena, Jakob Verbeek, Convolutional Neural Fabrics. NIPS 2016: 4053-4061
- Karen Simonyan, Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR abs/1409.1556 (2014)
- Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus, End-To-End Memory Networks. NIPS 2015: 2440-2448
- Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Going deeper with convolutions. CVPR 2015: 1-9
- Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf, DeepFace: Closing the Gap to Human-Level Performance in Face Verification. CVPR 2014: 1701-1708
- Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson, How transferable are features in deep neural networks? NIPS 2014: 3320-3328
- Matthew D. Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks. ECCV (1) 2014: 818-833
- Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, Oriol Vinyals, Understanding deep learning requires rethinking generalization. CoRR abs/1611.03530 (2016)