10-315: Introduction to Machine Learning
Fall2019

Devendra Singh Chaplot

Machine Learning Department
chaplot@cs.cmu.edu

Training Neural Networks

Used Resources

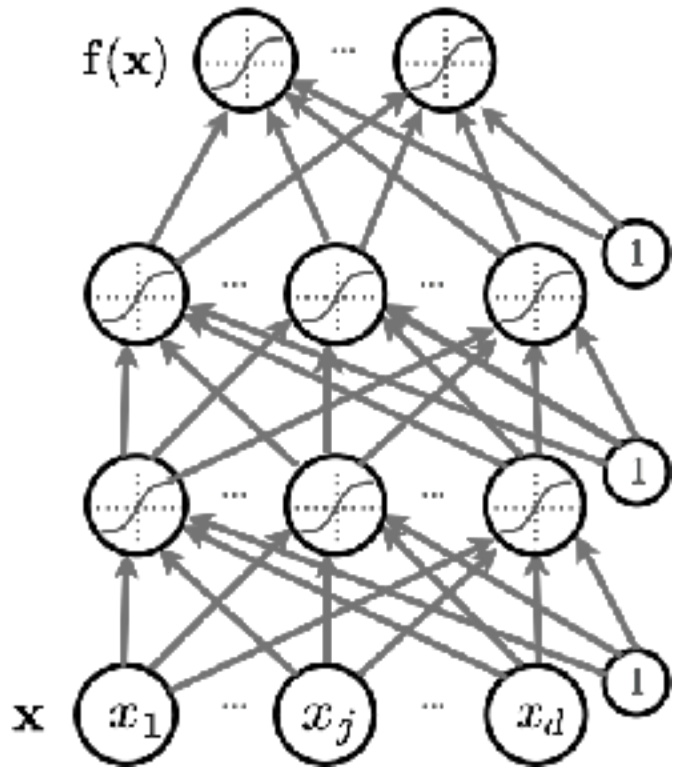
- **Disclaimer**: Much of the material in this lecture was borrowed from Russ Salakhutdinov’s class on Deep Learning (10-807)

- Hugo Larochelle’s class on Neural Networks: https://sites.google.com/site/deeplearningsummerschool2016/

- Some tutorial slides were borrowed from Rob Fergus’ CIFAR tutorial on ConvNets: https://sites.google.com/site/deeplearningsummerschool2016/speakers

- Some slides were borrowed from Marc'Aurelio Ranzato’s CVPR 2014 tutorial on Convolutional Nets https://sites.google.com/site/lsvrtutorialcvpr14/home/deeplearning

Feedforward Neural Networks



Stochastic Gradient Descent

- Perform updates after seeing each example:
  - Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
  - For t=1:T
    - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$$

Training epoch
=
Iteration of all examples

- To train a neural net, we need:
  - ➢ Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - ➢ Compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$

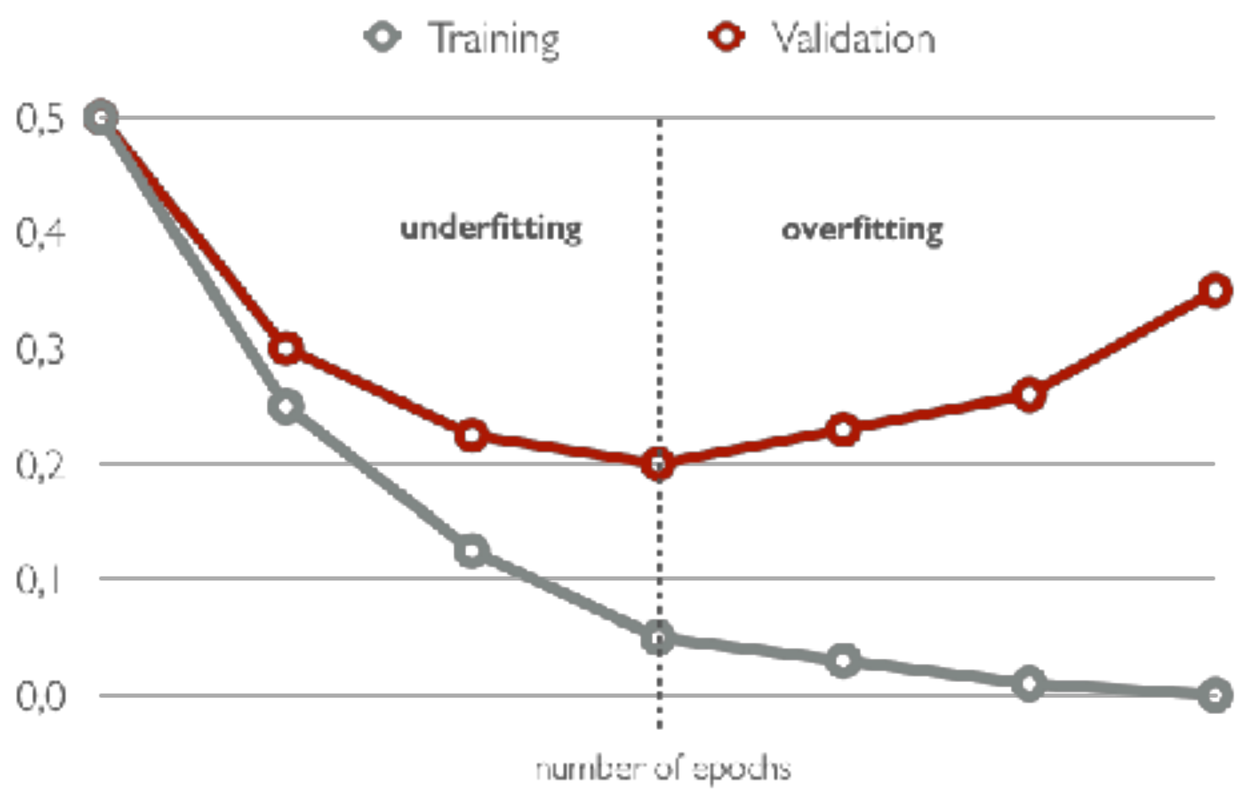
$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

Model Selection

- Training Protocol:
  - Train your model on the Training Set $\mathcal{D}^{\text{train}}$
  - For model selection, use Validation Set $\mathcal{D}^{\text{valid}}$
    - ➢ Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
  - Estimate generalization performance using the Test Set $\mathcal{D}^{\text{test}}$
- Remember: Generalization is the behavior of the model on **unseen examples**.

Early Stopping

- To select the number of epochs, stop training when validation set error increases (with some look ahead).



Mini-batch, Momentum

- Make updates based on a mini-batch of examples (instead of a single example):
  - ➢ the gradient is the average regularized loss for that mini-batch
  - ➢ can give a more accurate estimate of the gradient
  - ➢ can leverage matrix/matrix operations, which are more efficient

- Momentum: Can use an exponential average of previous gradients:

$$\overline{\nabla}_{\boldsymbol{\theta}}^{(t)} = \nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)}) + \beta \overline{\nabla}_{\boldsymbol{\theta}}^{(t-1)}$$

  - ➢ can get pass plateaus more quickly, by “gaining momentum”

Adapting Learning Rates

- Updates with adaptive learning rates ("one learning rate per parameter")

➢ Adagrad: learning rates are scaled by the square root of the cumulative sum of squared gradients

$$\gamma^{(t)} = \gamma^{(t-1)} + \left(\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})\right)^2 \quad \overline{\nabla}_\theta^{(t)} = \frac{\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})}{\sqrt{\gamma^{(t)}} + \epsilon}$$

➢ RMSProp: instead of cumulative sum, use exponential moving average

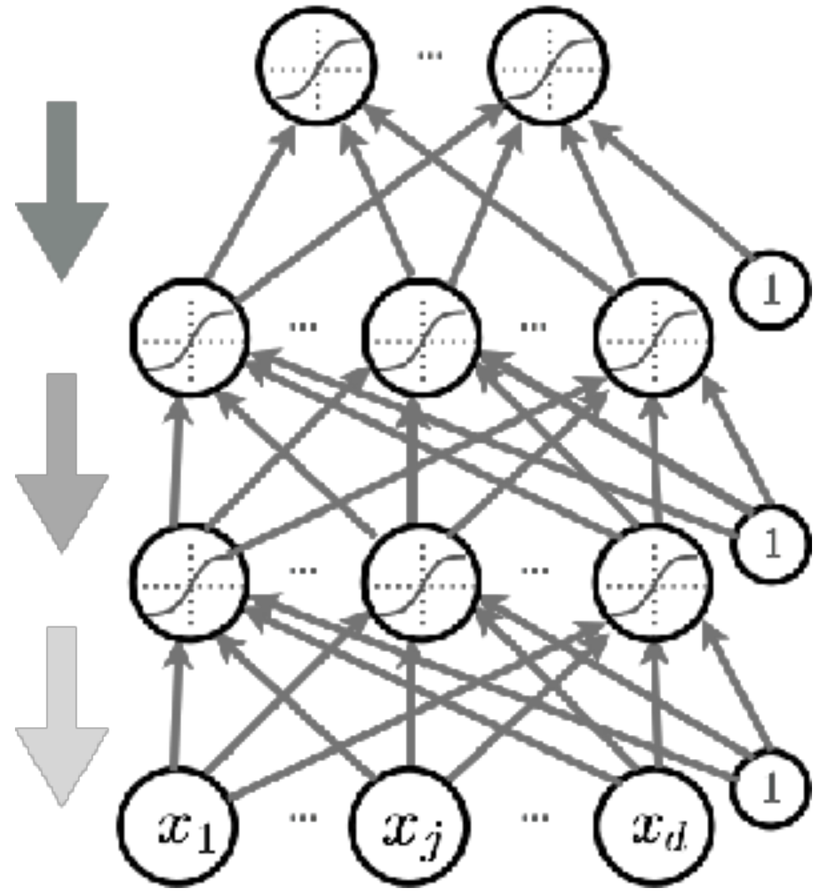
$$\gamma^{(t)} = \beta\gamma^{(t-1)} + (1-\beta)\left(\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})\right)^2$$

$$\overline{\nabla}_\theta^{(t)} = \frac{\nabla_\theta l(\mathbf{f}(\mathbf{x}^{(t)}), y^{(t)})}{\sqrt{\gamma^{(t)}} + \epsilon}$$

➢ Adam: essentially combines RMSProp with momentum

Why Training is Hard

- First hypothesis: Hard optimization problem (underfitting)
    - ➢ vanishing gradient problem
    - ➢ saturated units block gradient propagation
    - ➢ neural network does not have enough capacity

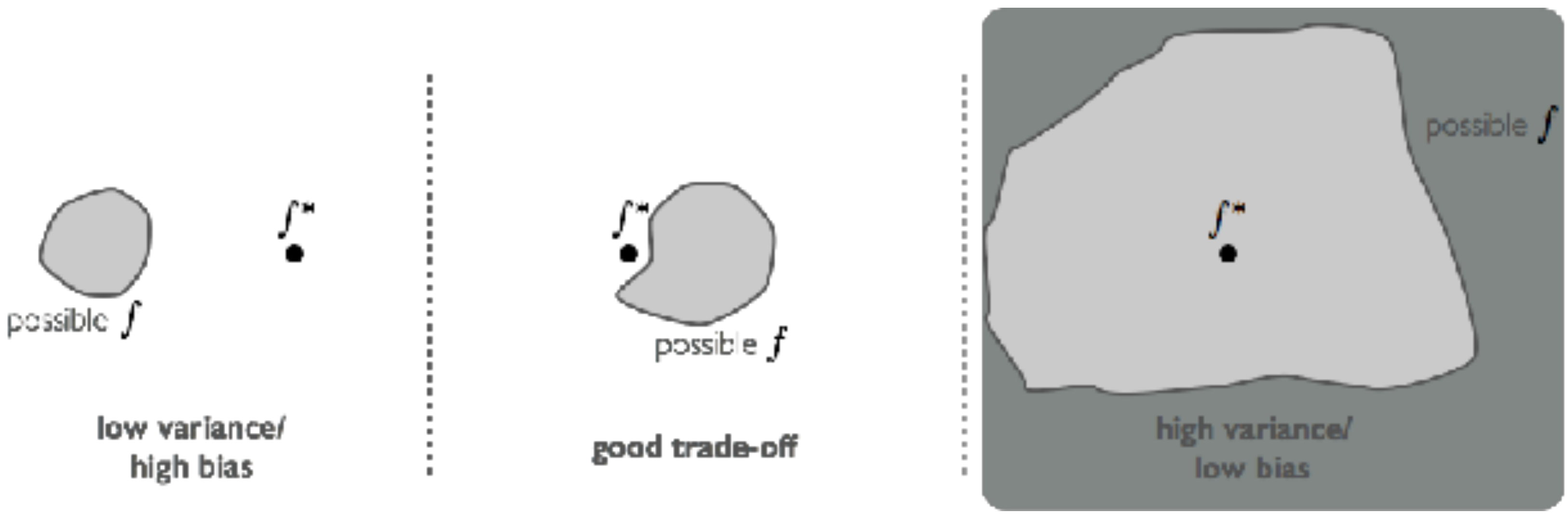


- Vanishing gradient: This is a well known problem in recurrent neural networks

Why Training is Hard

- Second hypothesis: Overfitting
  - ➢ we are exploring a space of complex functions
  - ➢ deep nets usually have lots of parameters

- Might be in a high variance / low bias situation



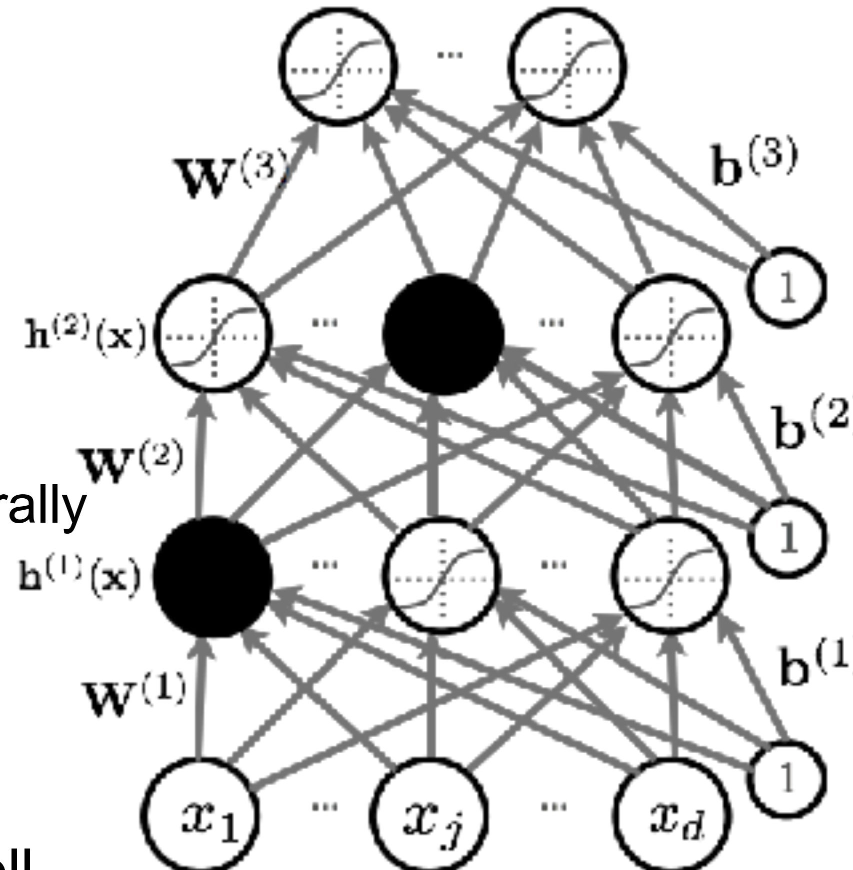
Why Training is Hard

- First hypothesis (underfitting): better optimize
  - ➢ Increase the capacity of the neural network
  - ➢ Tune learning rate
  - ➢ Check gradients
- Second hypothesis (overfitting): use better regularization
  - ➢ Dropout
  - ➢ Data augmentation
- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

Dropout

- Key idea: Cripple neural network by removing hidden units stochastically

  - ➢ each hidden unit is set to 0 with probability 0.5
  - ➢ hidden units cannot co-adapt to other units
  - ➢ hidden units must be more generally useful



- Could use a different dropout probability, but 0.5 usually works well

Dropout at Test Time

- At test time, we replace the masks by their expectation
  - ➢ This is simply the constant vector 0.5 if dropout probability is 0.5
  - ➢ For single hidden layer: equivalent to taking the geometric average of all neural networks, with all possible binary masks
- Beats regular backpropagation on many datasets
- Ensemble: Can be viewed as a geometric average of exponential number of networks.

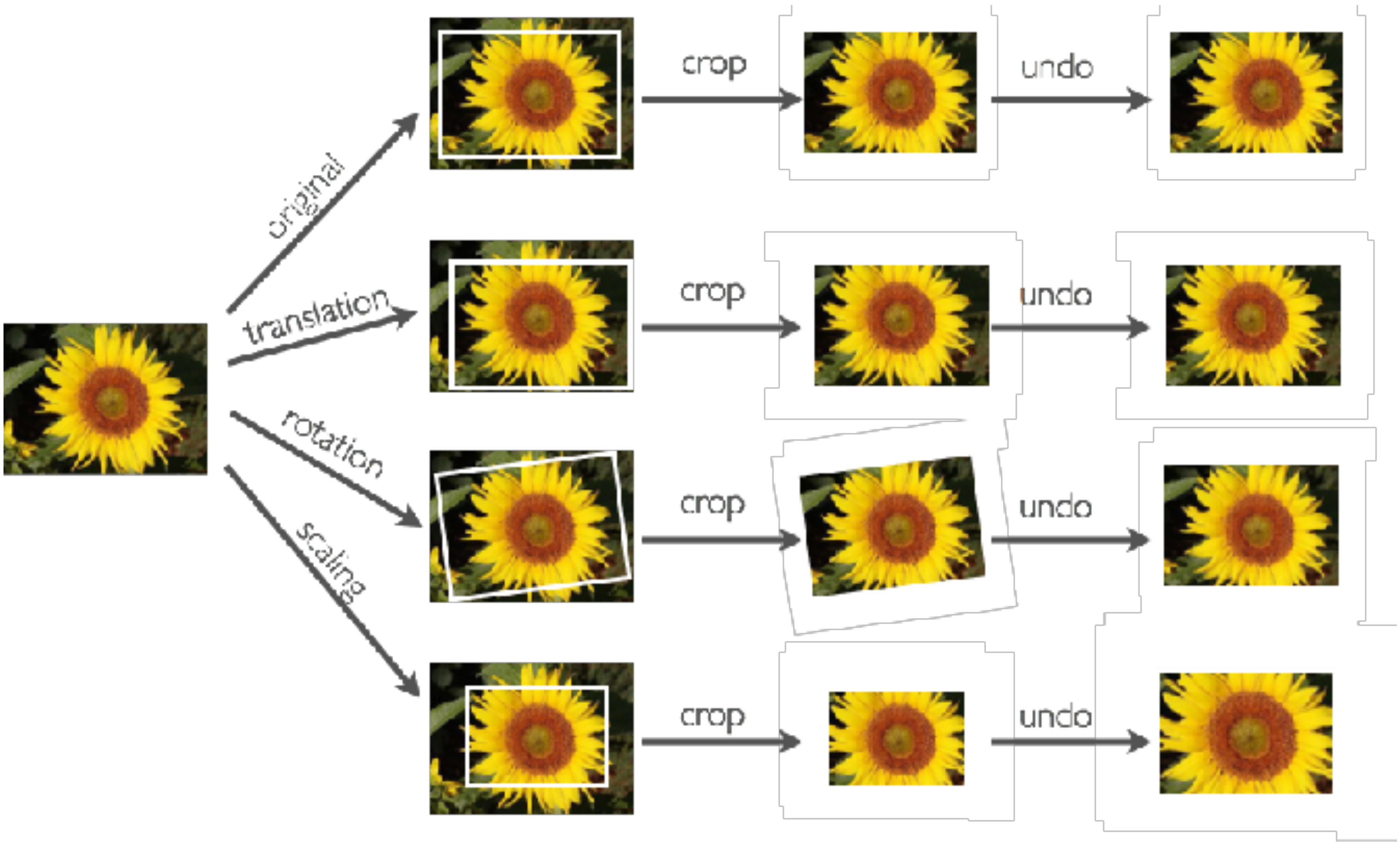
Why Training is Hard

- First hypothesis (underfitting): better optimize
  - ➢ Increase the capacity of the neural network
  - ➢ Tune learning rate
  - ➢ Check gradients
- Second hypothesis (overfitting): use better regularization
  - ➢ Dropout
  - ➢ Data augmentation
- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

Invariance by Data Augmentation

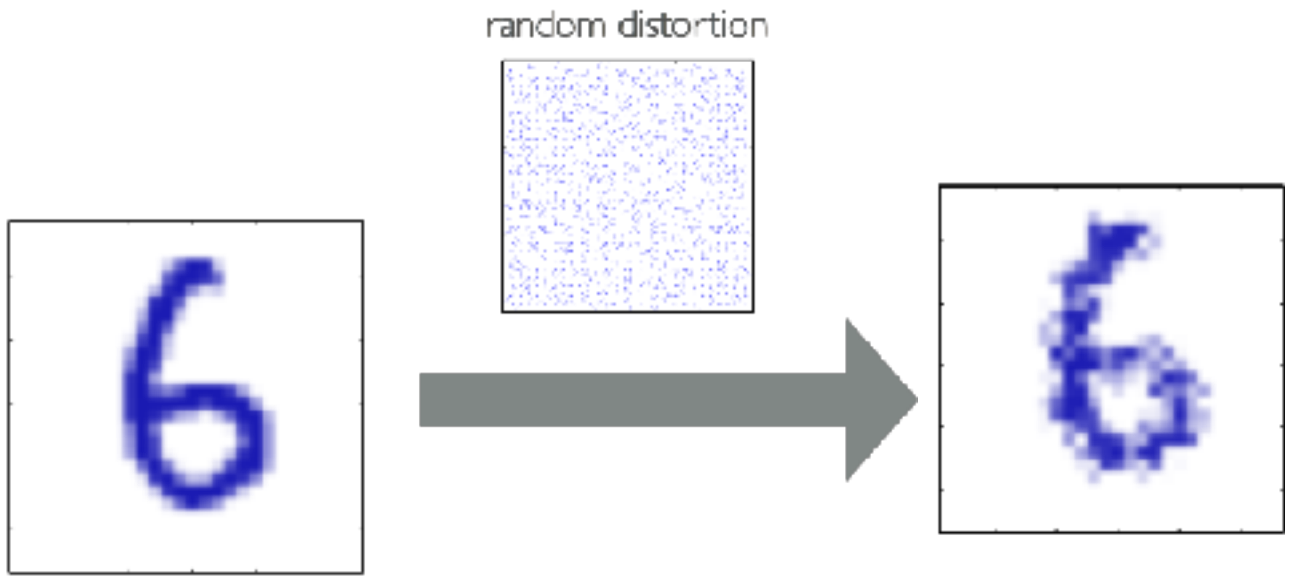
- Translation invariances built-in in convolutional network:
  - ➢ due to convolution and max pooling
- It is not invariant to other important variations such as rotations and scale changes
- However, it's easy to artificially generate data with such transformations
  - ➢ could use such data as additional training data
  - ➢ neural network can potentially learn to be invariant to such transformations

Generating Additional Examples



Elastic Distortions

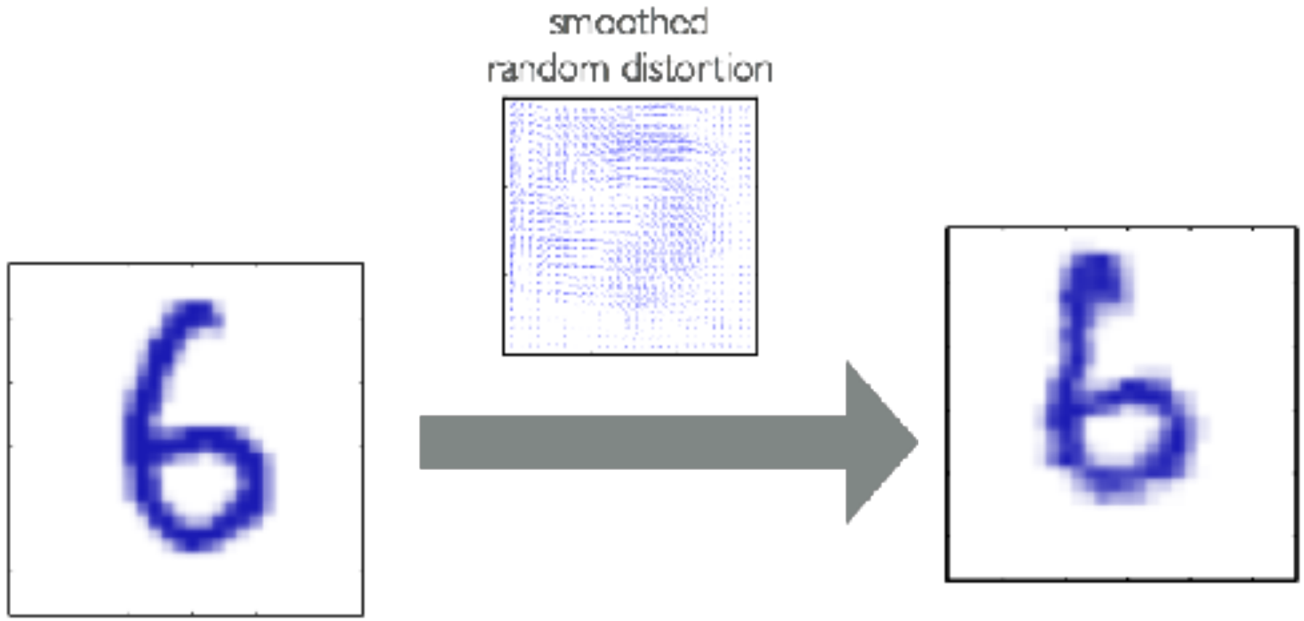
- Can add "elastic" deformations (useful in character recognition)
- We can do this by applying a "distortion field" to the image
  - ➢ a distortion field specifies where to displace each pixel value



Bishop's book

Elastic Distortions

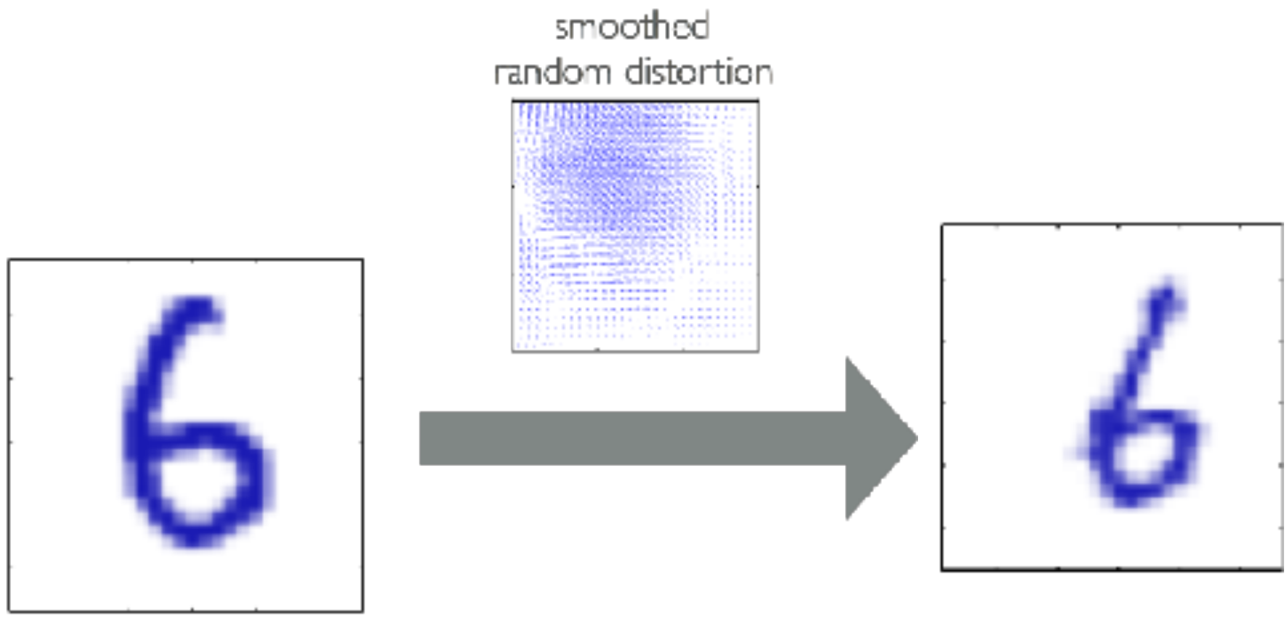
- Can add “elastic” deformations (useful in character recognition)
- We can do this by applying a “distortion field” to the image
  - ➢ a distortion field specifies where to displace each pixel value



Bishop’s book

Elastic Distortions

- Can add “elastic” deformations (useful in character recognition)
- We can do this by applying a “distortion field” to the image
  - ➢ a distortion field specifies where to displace each pixel value



Bishop’s book

Why Training is Hard

- First hypothesis (underfitting): better optimize
  - ➢ Increase the capacity of the neural network
  - ➢ Tune learning rate
  - ➢ Check gradients
- Second hypothesis (overfitting): use better regularization
  - ➢ Dropout
  - ➢ Data augmentation
- For many large-scale practical problems, you will need to use both: better optimization and better regularization!

Optimization Tricks

- Pick learning rate by running on a subset of the data
  - ➢ Start with large learning rate & divide by 2 until loss does not diverge
  - ➢ Decay learning rate by a factor of ~100 or more by the end of training

- Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.

- Tune the capacity of the neural network
  - Number of layers
  - Number of neurons per layer

Initialization

- Initialize biases to 0
- For weights
  - Can not initialize weights to 0 with tanh activation
    - ➢ All gradients would be zero (saddle point)
  - Can not initialize all weights to the same value
    - ➢ All hidden units in a layer will always behave the same
    - ➢ Need to break symmetry
  - Sample $\mathbf{W}_{i,j}^{(k)}$ from $U\left[-b, b\right]$, where

$$b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$$

Sample around 0 and break symmetry

Size of $\mathbf{h}^{(k)}(\mathbf{x})$

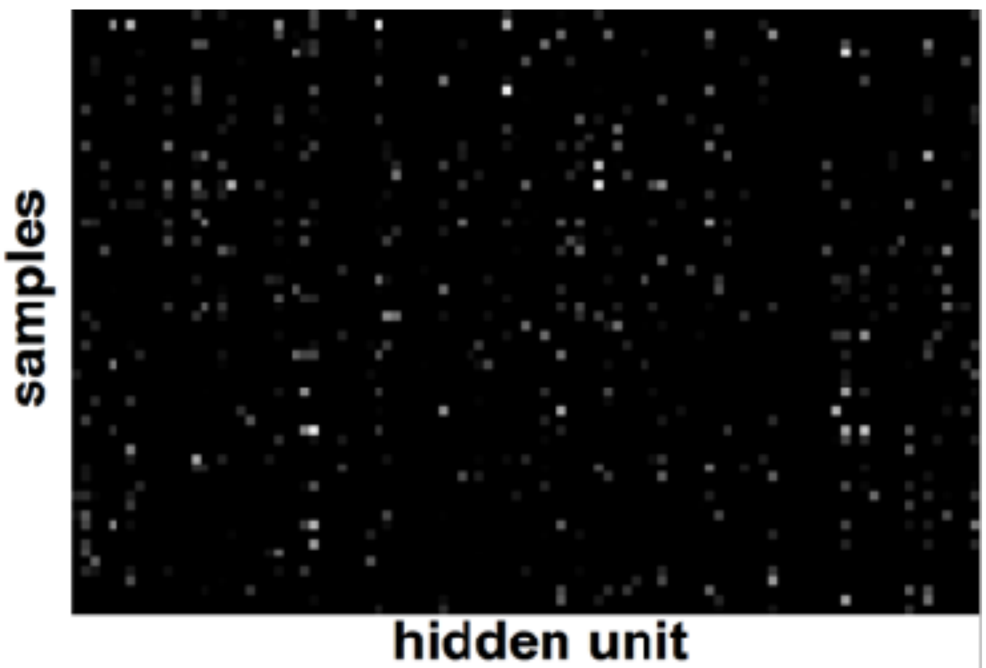
Choosing the Architecture

- Task dependent
- Cross-validation
- For image-based tasks:
  [Convolution → pooling]* + fully connected layer
- The more data: the more layers and the more neurons per layer
- Computational resources

Visualization

- Check weight and gradient norms

- Visualize features (feature maps need to be uncorrelated) and have high variance



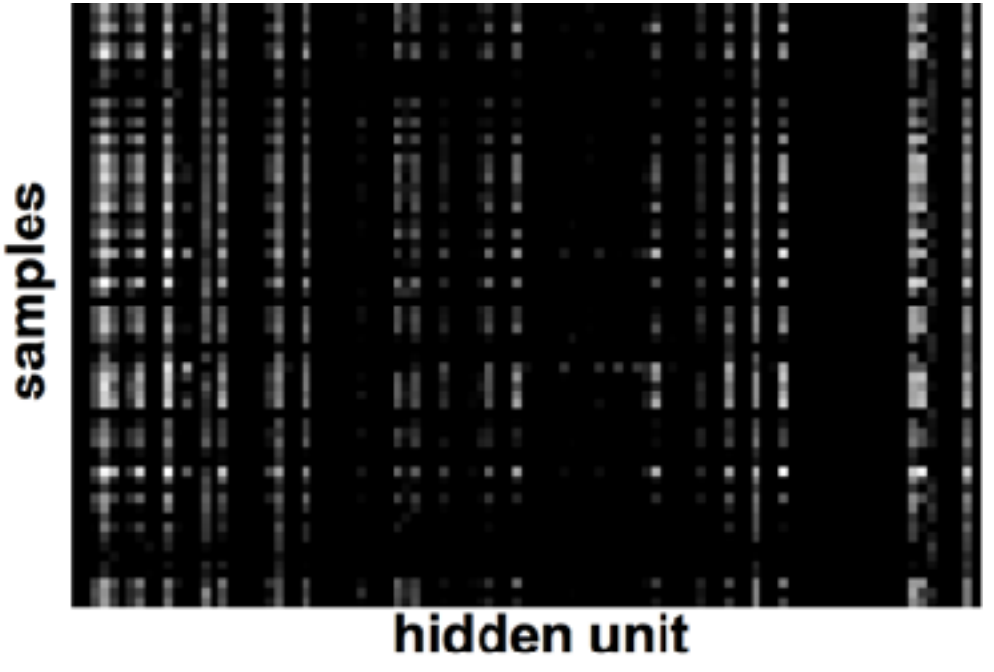
- Good training: hidden units are sparse across samples

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Visualization

- Check weight and gradient norms

- Visualize features (feature maps need to be uncorrelated) and have high variance



- Bad training: many hidden units ignore the input and/or exhibit strong correlations

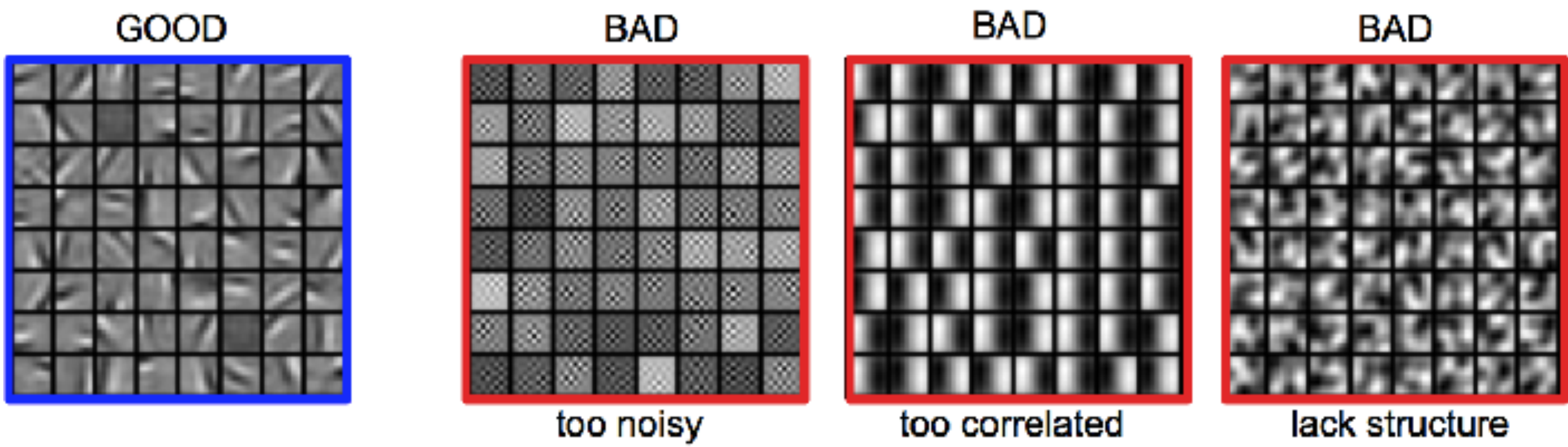
[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Visualization

• Check weight and gradient norms

• Visualize features (feature maps need to be uncorrelated) and have high variance

• Visualize parameters: learned filters should exhibit structure and should be uncorrelated



[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Visualization

- Check weight and gradient norms

- Visualize features (feature maps need to be uncorrelated) and have high variance

- Visualize parameters: learned filters should exhibit structure and should be uncorrelated

- Measure error on both training and validation set

- Test on a small subset of the data and check the error $\rightarrow 0$.

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

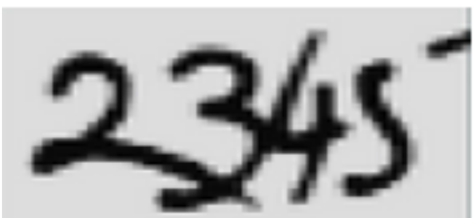
When it does not work

- Training diverges:
  - ➢ Learning rate may be too large → decrease learning rate
- Parameters collapse / loss is minimized but accuracy is low
  - ➢ Check loss function: Is it appropriate for the task you want to solve?
  - ➢ Does it have degenerate solutions?
- Network is underperforming
  - ➢ Compute flops and nr. params. → if too small, make net larger
  - ➢ Visualize hidden units/params → fix optimization
- Network is too slow
  - ➢ GPU, distrib. framework, make net smaller

[From Marc'Aurelio Ranzato, CVPR 2014 tutorial]

Conv Nets: Examples

- Optical Character Recognition, House Number and Traffic Sign classification



Ciresan et al. "MCDNN for image classification" CVPR 2012
Wan et al. "Regularization of neural networks using dropconnect" ICML 2013
Goodfellow et al. "Multi-digit nuber recognition from StreetView..." ICLR 2014
Jaderberg et al. "Synthetic data and ANN for natural scene text recognition" arXiv 2014

Conv Nets: Examples

- Pedestrian detection

Sermanet et al. “Pedestrian detection with unsupervised multi-stage..” CVPR 2013

Conv Nets: Examples

- Object Detection

Sermanet et al. “OverFeat: Integrated recognition, localization” arxiv 2013
Girshick et al. “Rich feature hierarchies for accurate object detection” arxiv 2013
Szegedy et al. “DNN for object detection” NIPS 2013

ImageNet Dataset

- 1.2 million images, 1000 classes

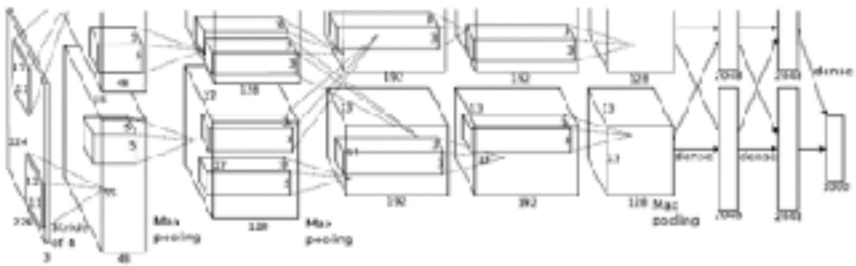
Examples of Hammer

Deng et al. “Imagenet: a large scale hierarchical image database” CVPR 2009

Important Breakthroughs

- Deep Convolutional Nets for Vision (Supervised)

  Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.



IMAGENET

1.2 million training images
1000 classes

Architecture

- How can we select the right architecture:
  - ➢ Manual tuning of features is now replaced with the manual tuning of architechtures

- Depth
- Width
- Parameter count

How to Choose Architecture

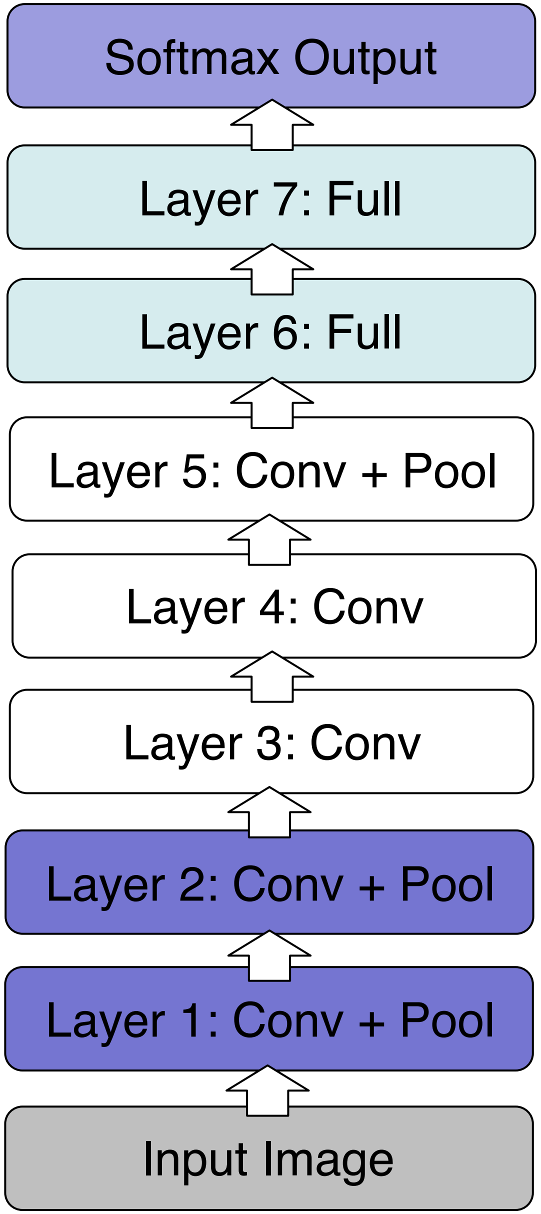
- Many hyper-parameters:
  - ➢ Number of layers, number of feature maps
- Cross Validation
- Grid Search (need lots of GPUs)
- Smarter Strategies
  - ➢ Random search
  - ➢ Bayesian Optimization

AlexNet

- 8 layers total

- Trained on Imagenet dataset [Deng et al. CVPR'09]

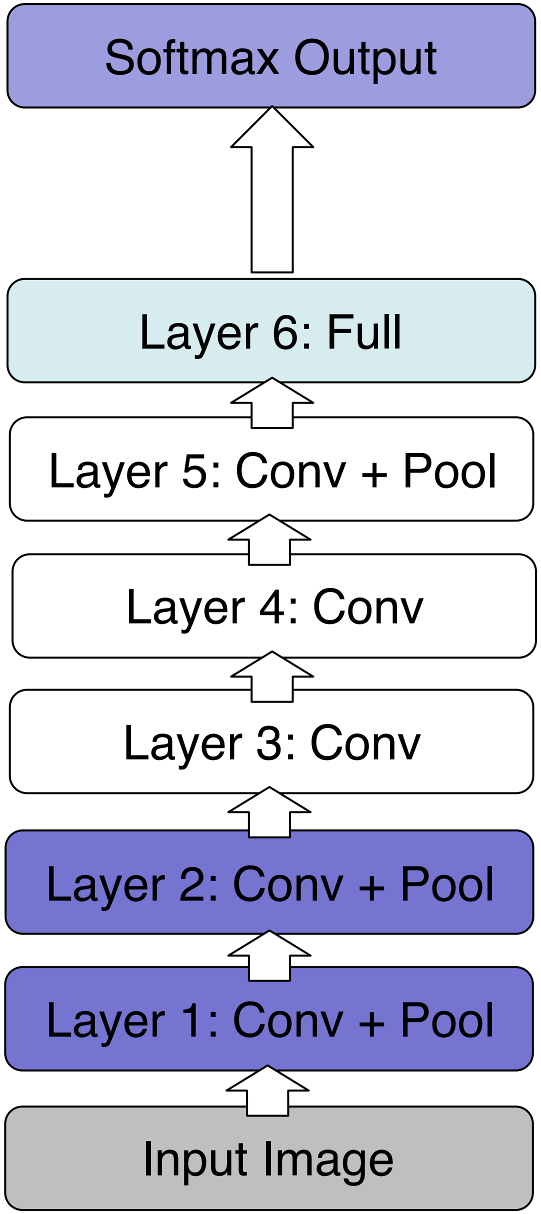
- 18.2% top-5 error



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

- Remove top fully connected layer 7
- Drop ~16 million parameters
- Only 1.1% drop in performance!



[From Rob Fergus' CIFAR 2016 tutorial]

AlexNet

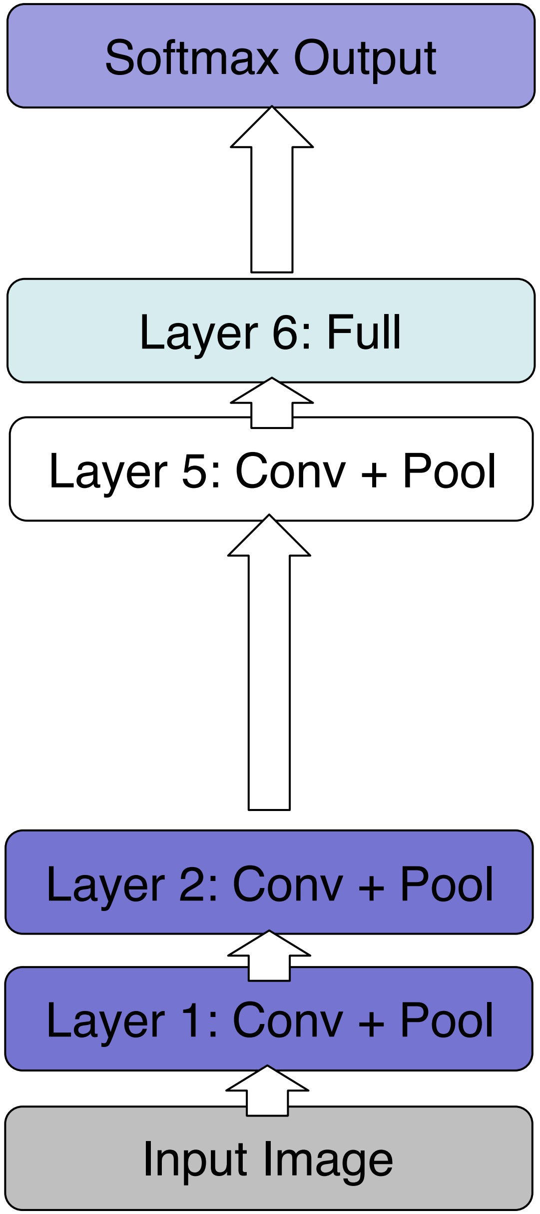
- Let us remove upper feature extractor layers and fully connected:

  ➢ Layers 3,4, 6 and 7

- Drop ~50 million parameters

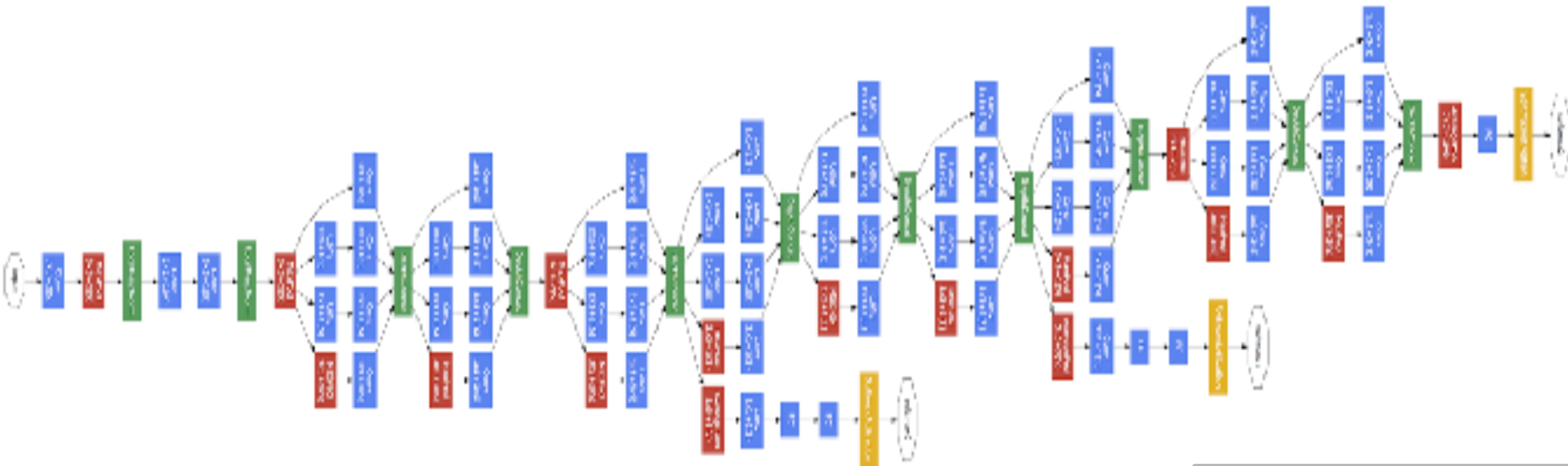
- **33.5 drop in performance!**

- Depth of the network is the key.



[From Rob Fergus' CIFAR 2016 tutorial]

GoogLeNet



- 24 layer model that uses so-called inception module.

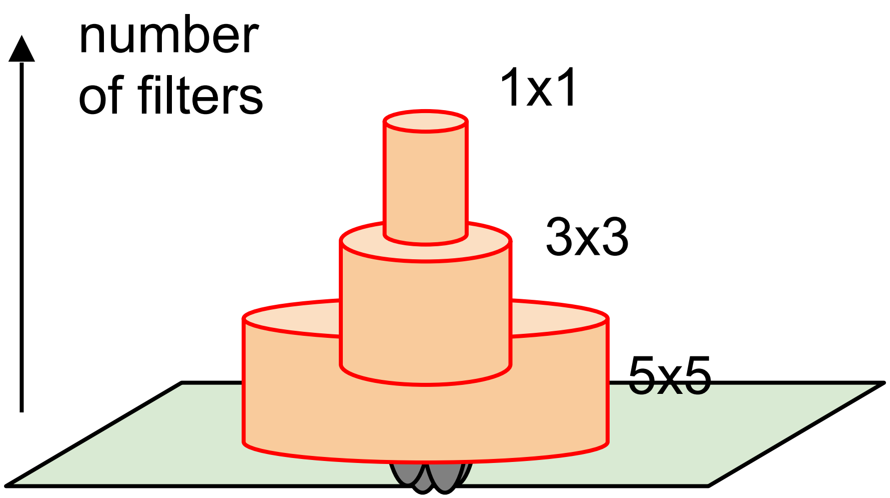
Convolution
Pooling
Softmax
Other

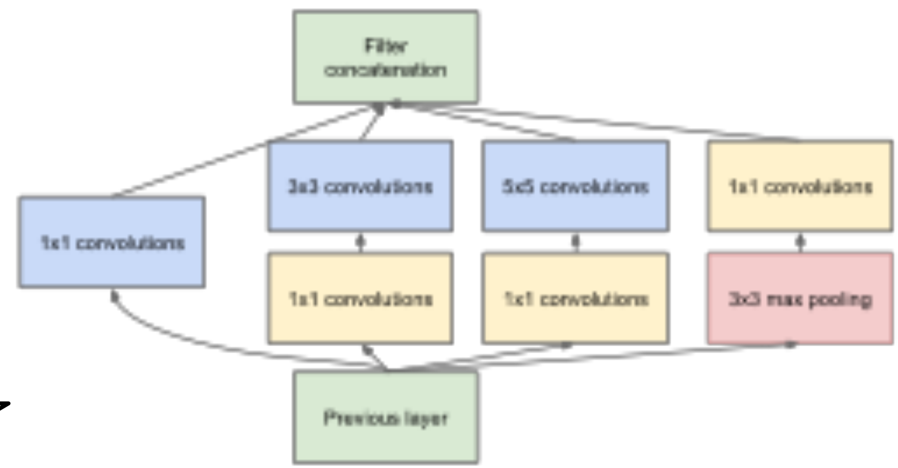
[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

GoogLeNet

- GoogLeNet inception module:

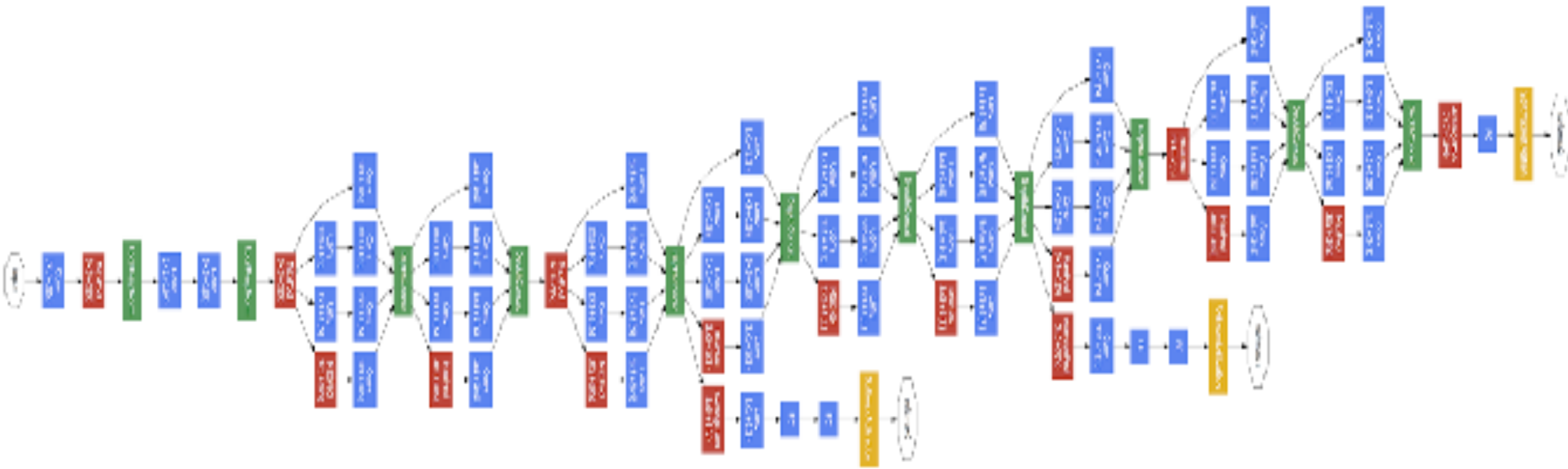
  - ➢ Multiple filter scales at each layer
  - ➢ Dimensionality reduction to keep computational requirements down





[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

GoogLeNet

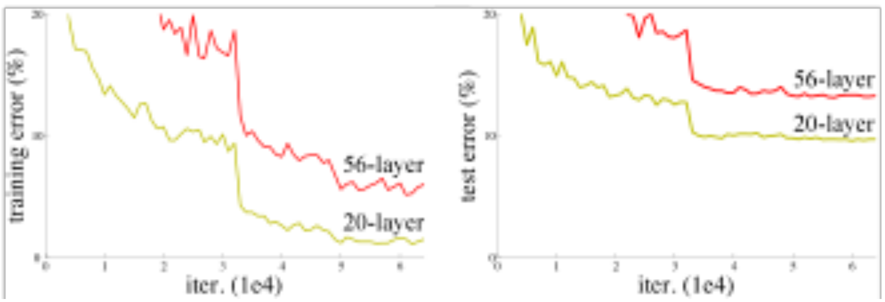


- Width of inception modules ranges from 256 filters (in early modules) to 1024 in top inception modules.
- Can remove fully connected layers on top completely
- Number of parameters is reduced to 5 million
- 6.7% top-5 validation error on Imagnet

[Going Deep with Convolutions, Szegedy et al., arXiv:1409.4842, 2014]

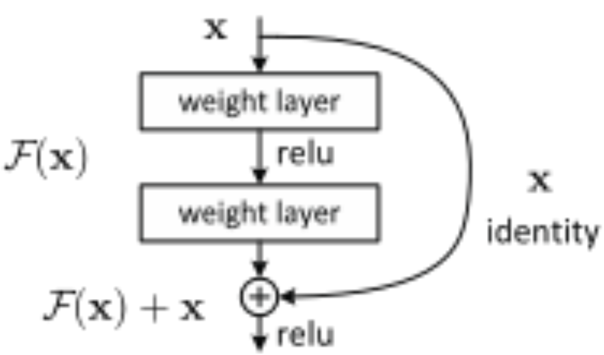
Residual Networks

Really, really deep convnets do not train well,
E.g. CIFAR10:



Key idea: introduce "pass through" into each layer

Thus only residual now needs to be learned



method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

With ensembling, 3.57% top-5 test error on ImageNet

[He, Zhang, Ren, Sun, CVPR 2016]