Introductory Overview Lecture
The Deep Learning Revolution
Part II: Optimization, Regularization

Russ Salakhutdinov

Machine Learning Department
Carnegie Mellon University
Canadian Institute for Advanced Research

Carnegie Mellon University

MACHINE LEARNING DEPARTMENT

Used Resources

- Some material and slides for this lecture were borrowed from
  - Hugo Larochelle's class on Neural Networks: https://sites.google.com/site/deeplearningsummerschool2016/
  - Grover and Ermon IJCA-ECA Tutorial on Deep Generative Models https://ermongroup.github.io/generative-models/

Supervised Learning

- Given a set of labeled training examples: $\{\mathbf{x}^{(t)}, y^{(t)}\}$, we perform Empirical Risk Minimization

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \lambda \underbrace{\Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

where

- $f(\mathbf{x}^{(t)}; \theta)$ is a (non-linear) function mapping inputs to outputs, parameterized by θ -> Non-convex optimization
- $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$ is the loss function
- $\Omega(\boldsymbol{\theta})$ is a regularization term

Supervised Learning

- Given a set of labeled training examples: $\{\mathbf{x}^{(t)}, y^{(t)}\}$, we perform Empirical Risk Minimization

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \lambda \underbrace{\Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

- Loss Functions:
  - For classification tasks, we can use Cross-Entropy Loss
  - For regression tasks, we can use Squared Loss

Training

- Empirical Risk Minimization

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \lambda \underbrace{\Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

- To train a neural network, we need:
  - Loss Function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - A procedure to compute its gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
  - Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

Stochastic Gradient Descent (SGD)

- 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 $(\mathbf{x}^{(t)}, y^{(t)})$

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

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

Learning rate: Difficult to set in practice

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
  - More accurate estimate of the gradient
  - Leverage matrix/matrix operations, which are more efficient

- Momentum: 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

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

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

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

  - Adam: essentially combines RMSProp with momentum

(Douchi et. al, 2011, Kingma and Ba, 2014)

Regularization

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \lambda \Omega(\boldsymbol{\theta})$$

- L2 regularization:

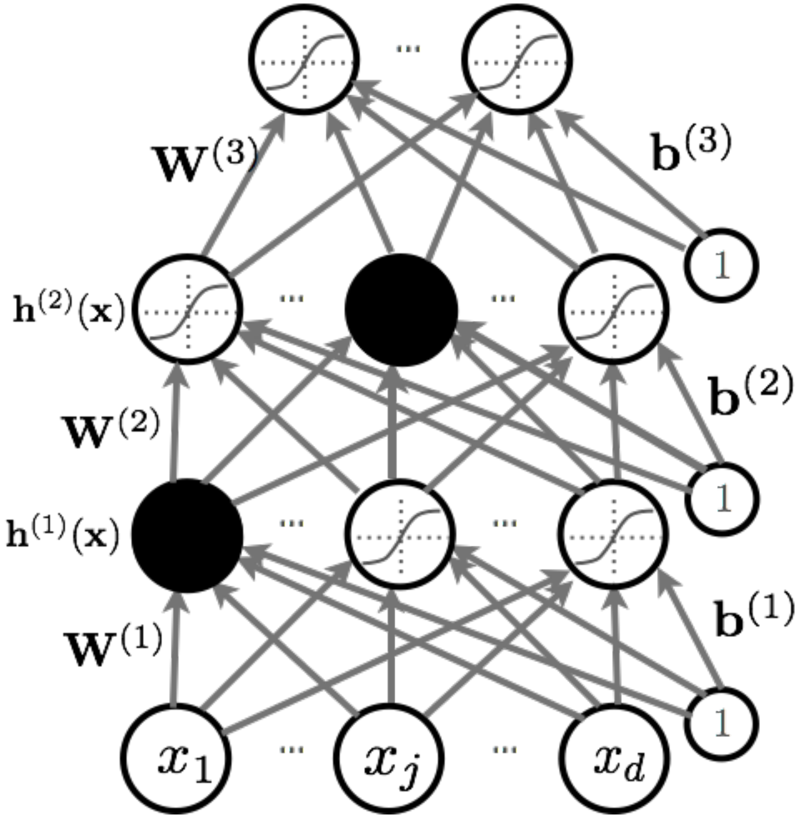
$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k ||\mathbf{W}^{(k)}||_F^2$$

- L1 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

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



Srivastava et al., JMLR 2014

Dropout

- Use random binary masks $\mathrm{m}^{(k)}$
  - Layer pre-activation for k>0

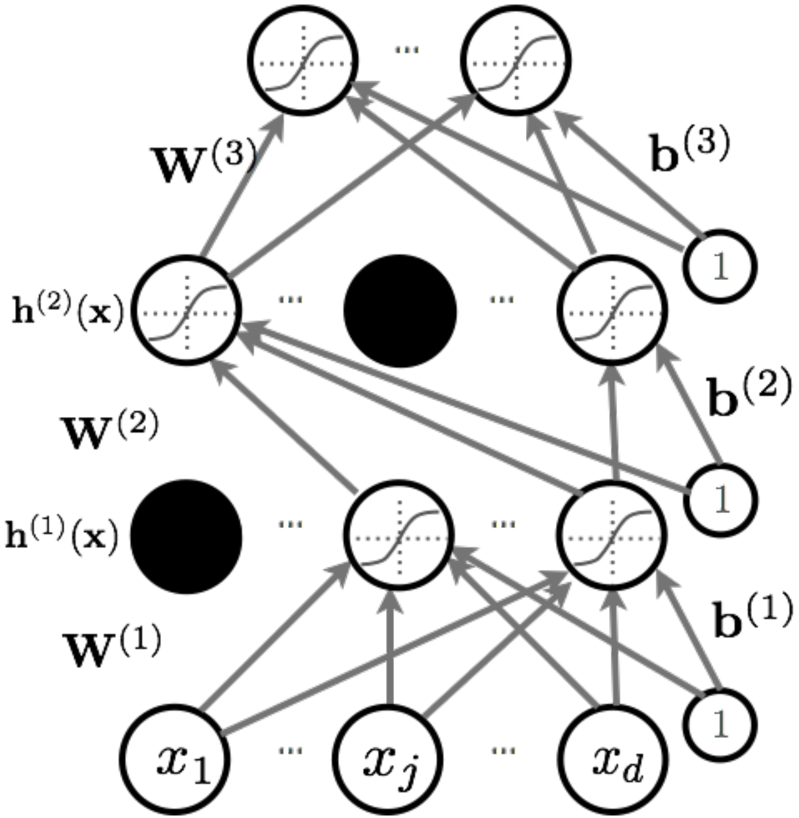
    $$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

  - hidden layer activation (k=1 to L):

    $$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x})) \odot \mathrm{m}^{(k)}$$

  - Output activation (k=L+1)

    $$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



Srivastava et al., JMLR 2014

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
- Beats regular backpropagation on many datasets and has become a standard practice
- Ensemble: Can be viewed as a geometric average of exponential number of networks.

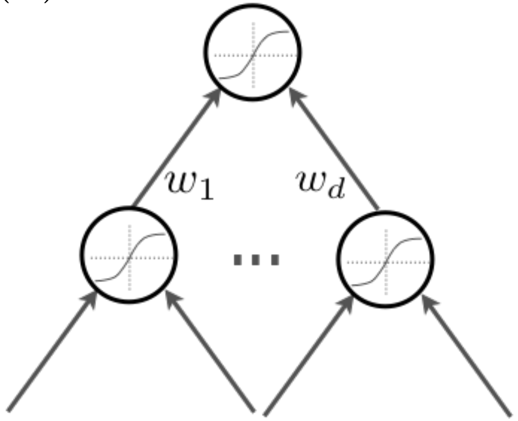
Batch Normalization

- Normalizing the inputs will speed up training (Lecun et al. 1998)
  - Could normalization be useful at the level of the hidden layers?

- Batch normalization is an attempt to do that (Ioffe and Szegedy, 2015)
  - each hidden unit's pre-activation is normalized (mean subtraction, stddev division)
  - during training, mean and stddev is computed for each mini-batch
  - backpropagation takes into account the normalization
  - at test time, the global mean and stddev is used

- Why normalize the pre-activation?
  - helps keep the pre-activation in a non-saturating regime
    → helps with vanishing gradient problem

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$



Batch Normalization

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

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

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

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

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

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

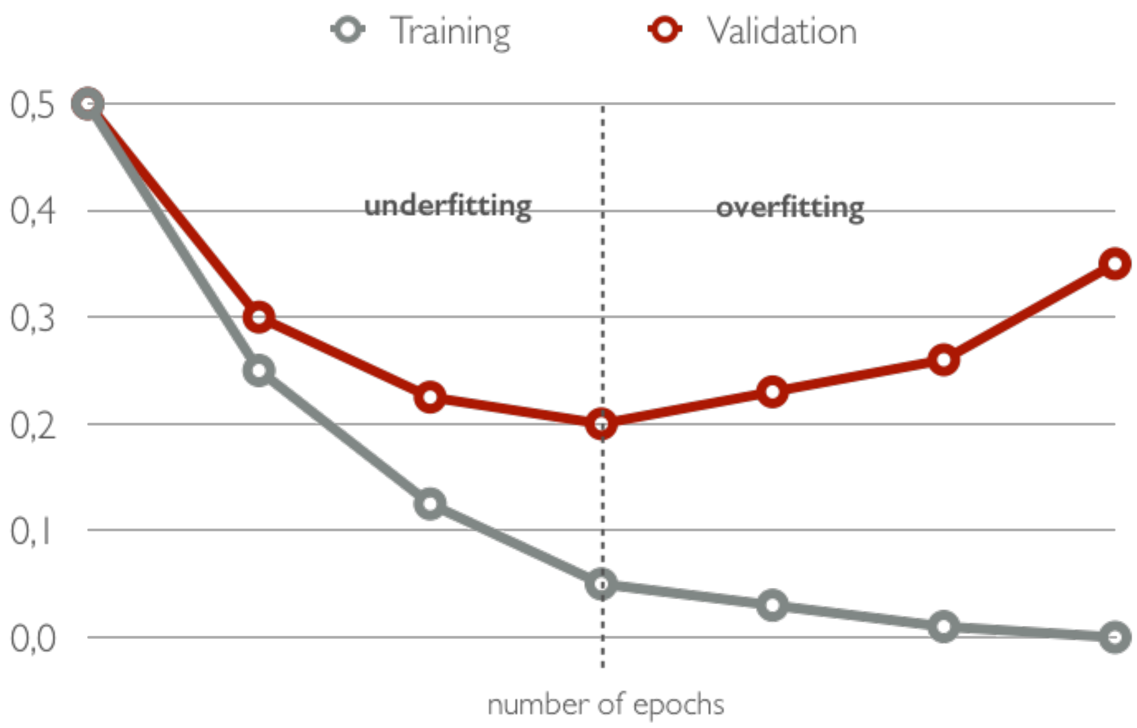
Learned linear transformation to adapt to non-linear activation function (γ and β are trained)

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, etc.*
  - Estimate generalization performance using the Test Set $\mathcal{D}^{\text{test}}$
- Generalization is the behavior of the model on **unseen examples**.

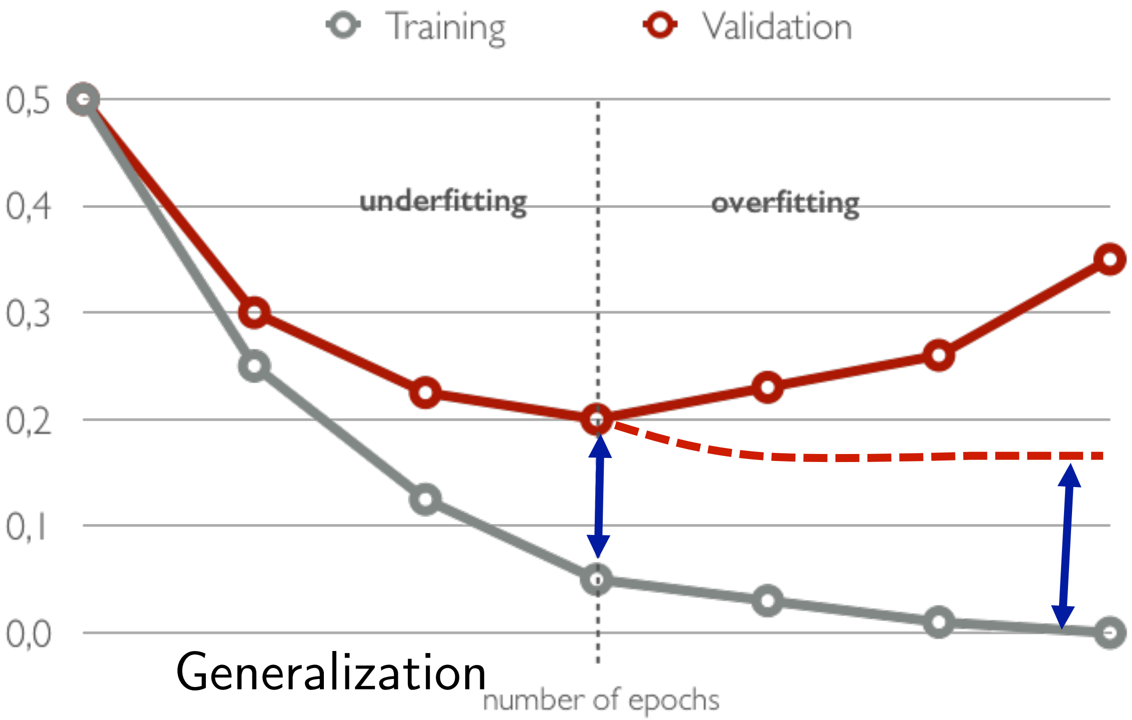
Early Stopping

- To select the number of epochs, stop training when validation set error increases → Large Model can Overfit



But in Practice

- To select the number of epochs, stop training when validation set error increases → Large Model can Overfit



Generalization Error

Implicit Regularization

- Optimization plays a crucial role in generalization
- Generalization ability is not controlled by network size but rather by some other implicit control

Behnam Neyshabur, PhD thesis 2017
Neyshabur et al., Survey Paper, 2017

Best Practice

- Given a dataset D, pick a model so that:
  - You can achieve 0 training error → Overfit on the training set.
- Regularize the model (e.g. using Dropout).
- Initialize parameters so that each feature across layers has similar variance. Avoid units in saturation.
- SGD with momentum, batch-normalization, and dropout usually works very well.

Choosing Architecture

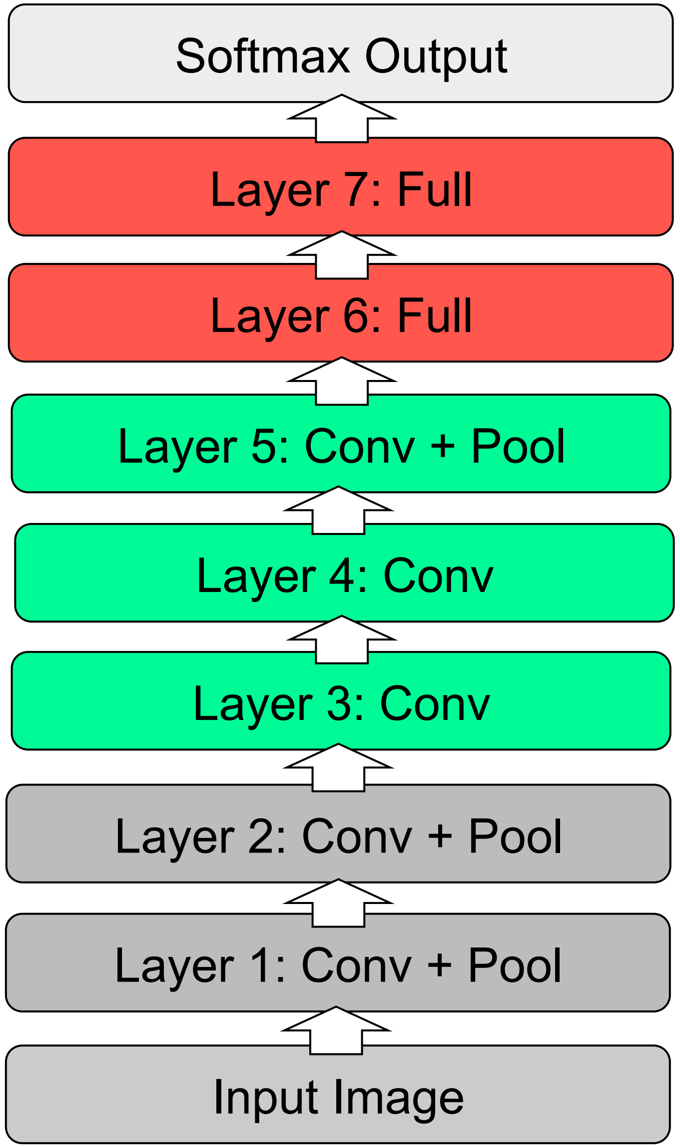
- How can we select the right architecture:
  - Manual tuning of features is now replaced with the manual tuning of architectures

- Many hyper-parameters:
  - Number of layers, number of feature maps

- Cross Validation
- Grid Search (need lots of GPUs)
- Smarter Strategies
  - Bayesian Optimization

AlexNet

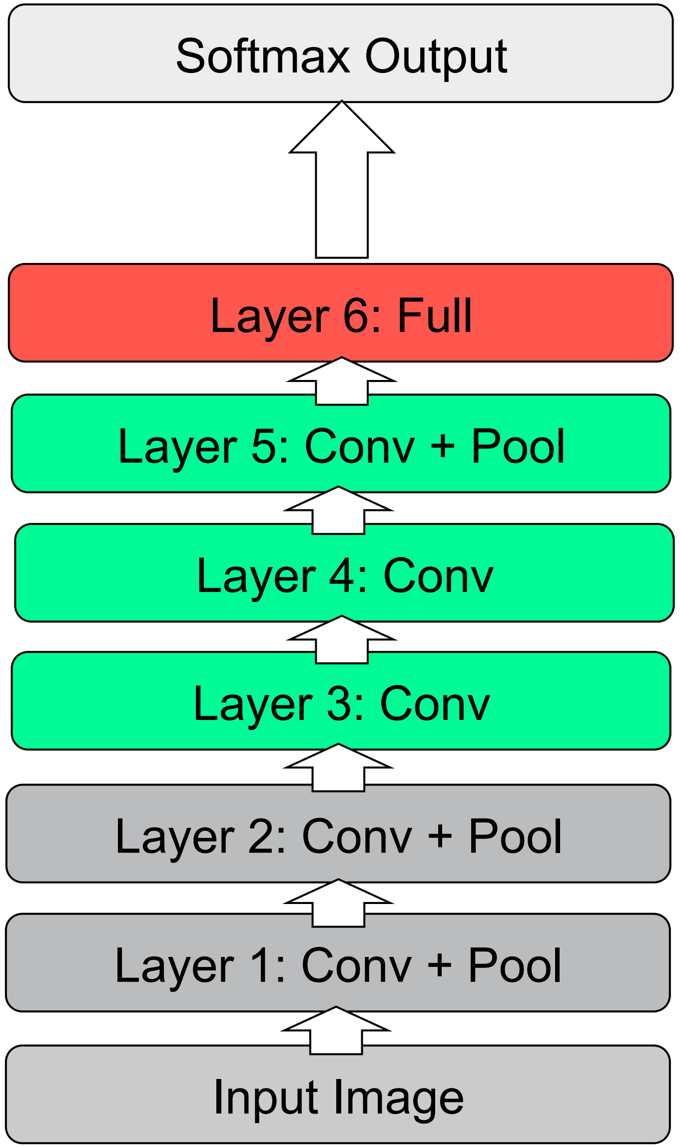
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error



[From Rob Fergus' CIFAR 2016 tutorial] Krizhevsky et al., NIPS 2012

AlexNet

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

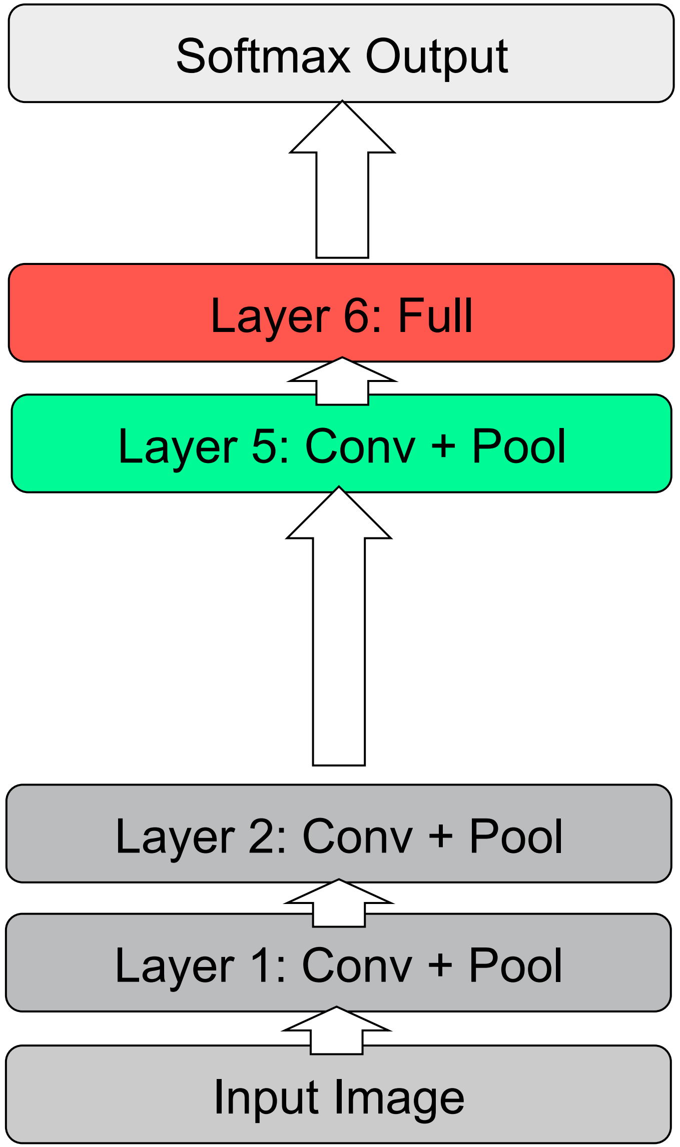


[From Rob Fergus' CIFAR 2016 tutorial] Krizhevsky et al., NIPS 2012

AlexNet

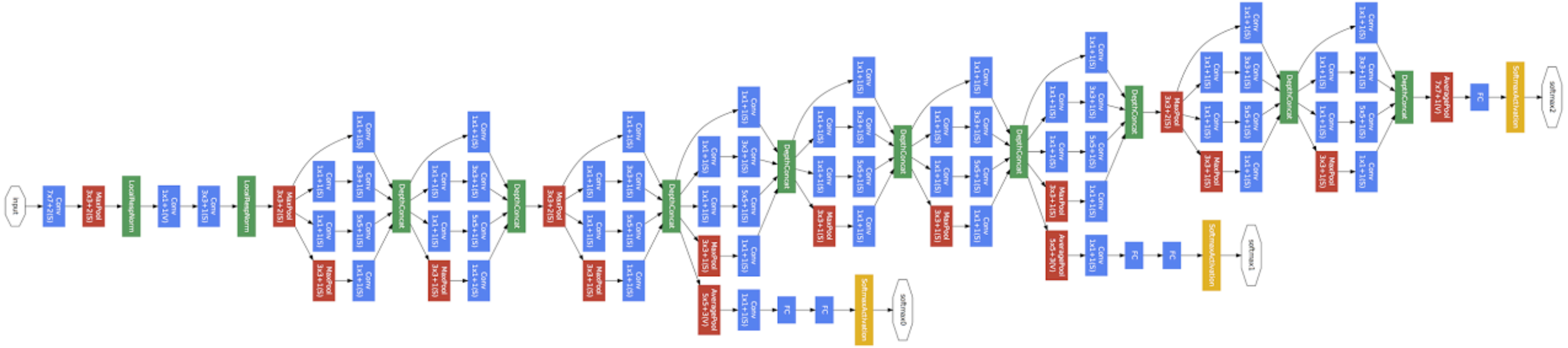
- Remove layers 3 4,6 and 7
- Drop ~50 million parameters
- **33.5%** drop in performance!

- Depth of the network is the key



[From Rob Fergus' CIFAR 2016 tutorial] Krizhevsky et al., NIPS 2012

GoogleNet



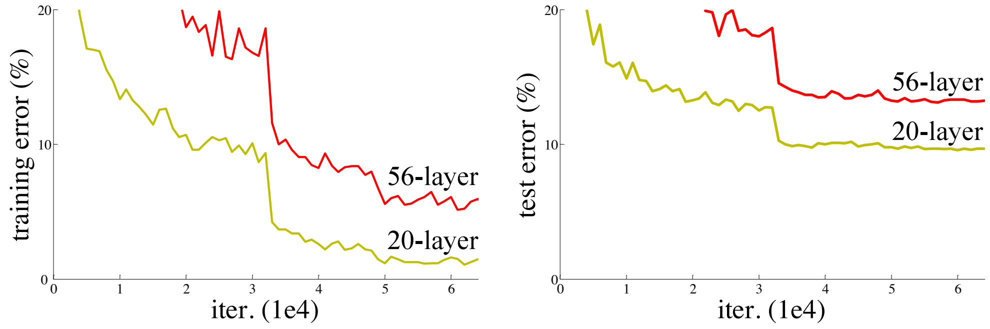
- 24 layer model

Convolution
Pooling
Softmax
Other

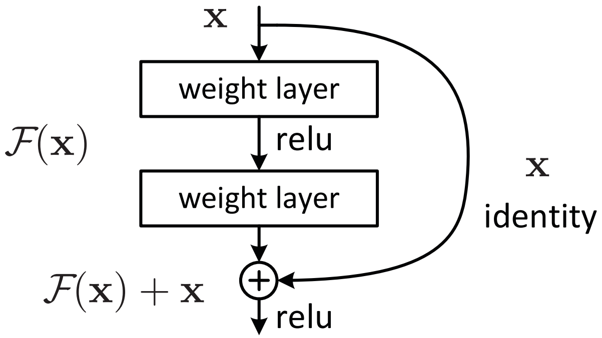
(Szegedy et al., Going Deep with Convolutions, 2014)

Residual Networks

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



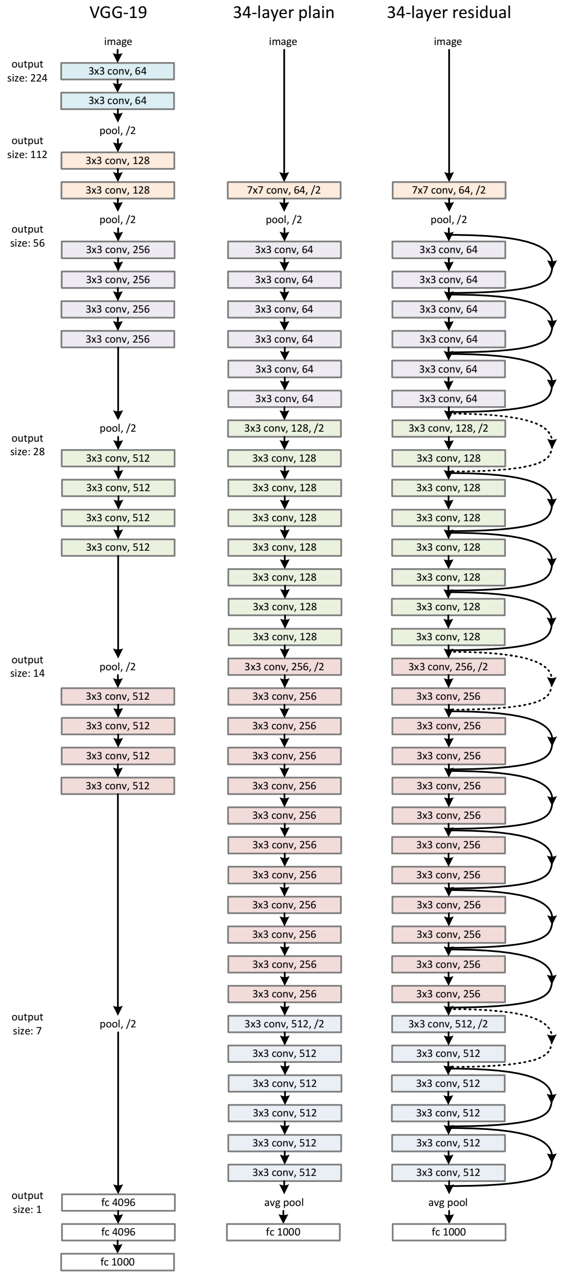
- Key idea: introduce "pass through" into each layer
- Thus only residual now needs to be learned:



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)