

Deep Learning Essentials

Supervised Learning

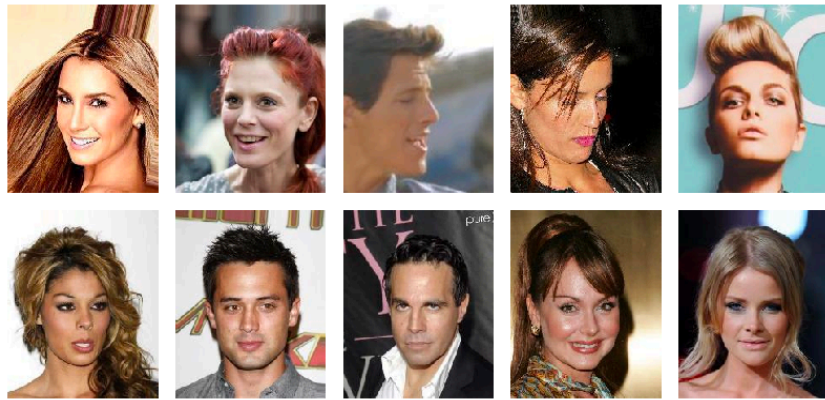
Russ Salakhutdinov

Machine Learning Department
Carnegie Mellon University

Impact of Deep Learning

- ▶ Speech Recognition
- ▶ Computer Vision
- ▶ Recommender Systems
- ▶ Language Understanding
- ▶ Drug Discovery and Medical Image Analysis

Statistical Generative Models



Training Data(CelebA)



Model Samples (Karras et.al., 2018)

4 years of progression on Faces



Brundage et al., 2017

Conditional Generation

- ▶ Conditional generative model $P(\text{zebra images} | \text{horse images})$



- ▶ Style Transfer



Input Image



Monet



Van Gogh

Conditional Generation

- ▶ Conditional generative model $P(\text{zebra images} | \text{horse images})$

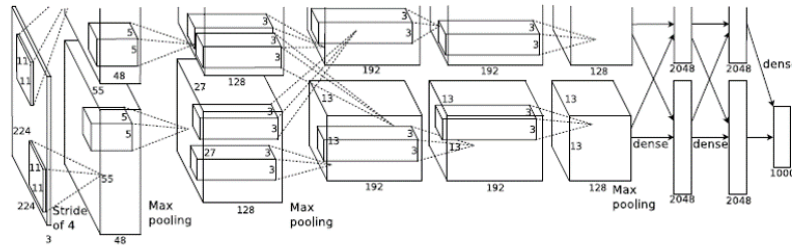


- ▶ Failure Case



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



- ▶ Deep Nets for Speech (Supervised)
 - ▶ Hinton et. al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, IEEE Signal Processing Magazine. 2012.

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/>

Outline

- ▶ Definition of Neural Networks
 - ▶ Forward propagation, Types of units, Capacity of neural networks
- ▶ Training Neural Networks
 - ▶ Loss function, Backpropagation algorithm
- ▶ Optimization/Regularization techniques
 - ▶ Dropout, Batch normalization, Best Practices
- ▶ Convolutional Neural Networks
 - ▶ Definition, Architecture Search
- ▶ Unsupervised Learning, Statistical Generative Models
 - ▶ Variational Autoencoders
 - ▶ Generative Adversarial Networks

Artificial Neuron

- ▶ Neuron pre-activation (or input activation):

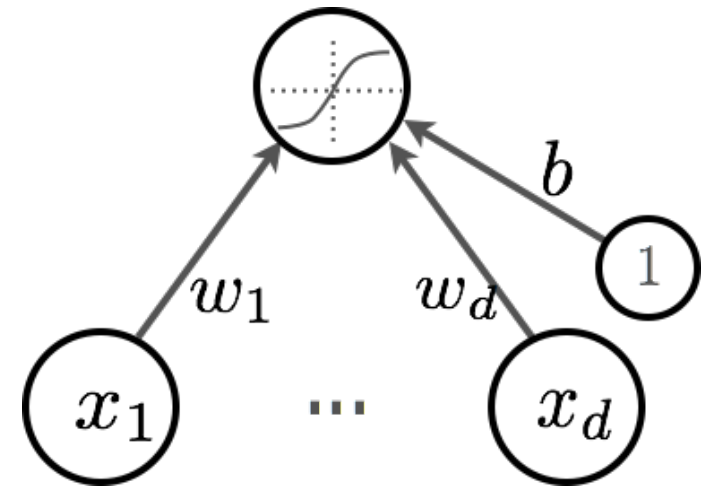
$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- ▶ Neuron output activation:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

- ▶ where

- ▶ \mathbf{W} are the weights (parameters)
- ▶ b is the bias term
- ▶ $g(\cdot)$ is called the activation function

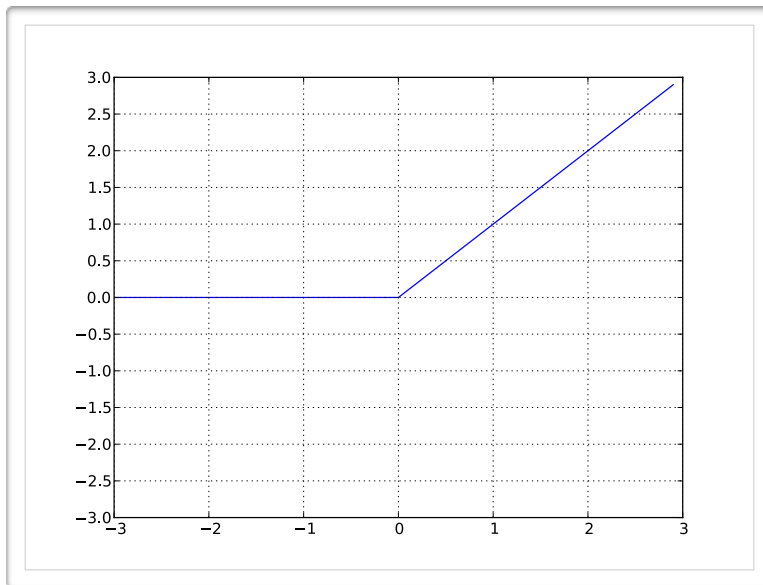


Activation Functions

- Neuron output activation: $h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$

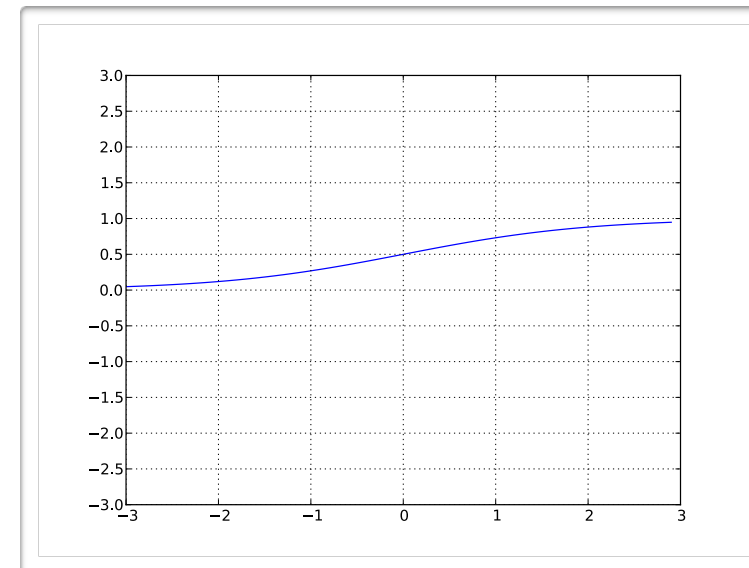
Rectified Linear Unit (ReLU)

$$g(a) = \text{reclin}(a) = \max(0, a)$$



Sigmoid Activation Function

$$g(a) = \text{sigm}(a) = \frac{1}{1 + \exp(-a)}$$



Neural Networks

- ▶ Hidden layer pre-activation:

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

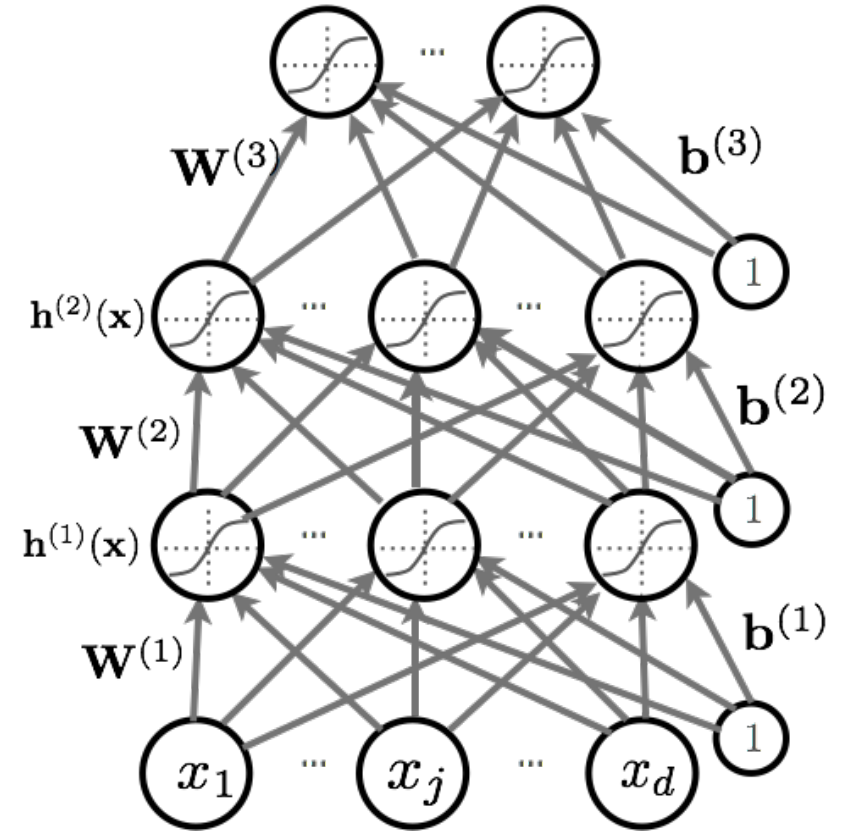
- ▶ Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- ▶ Output layer activation:

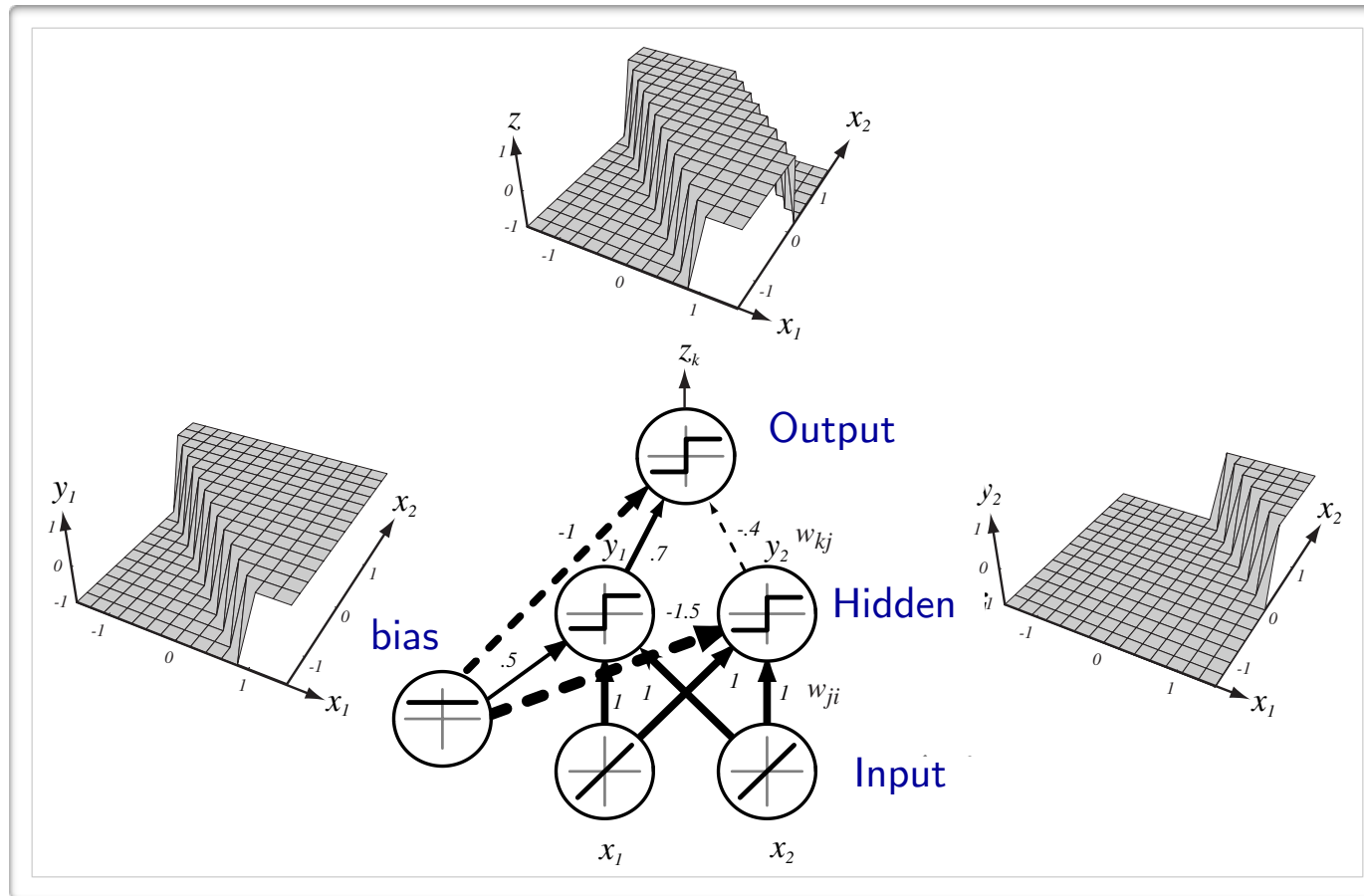
$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top} \mathbf{h}^{(1)}\mathbf{x}\right)$$

Output activation
function



Capacity of Neural Nets

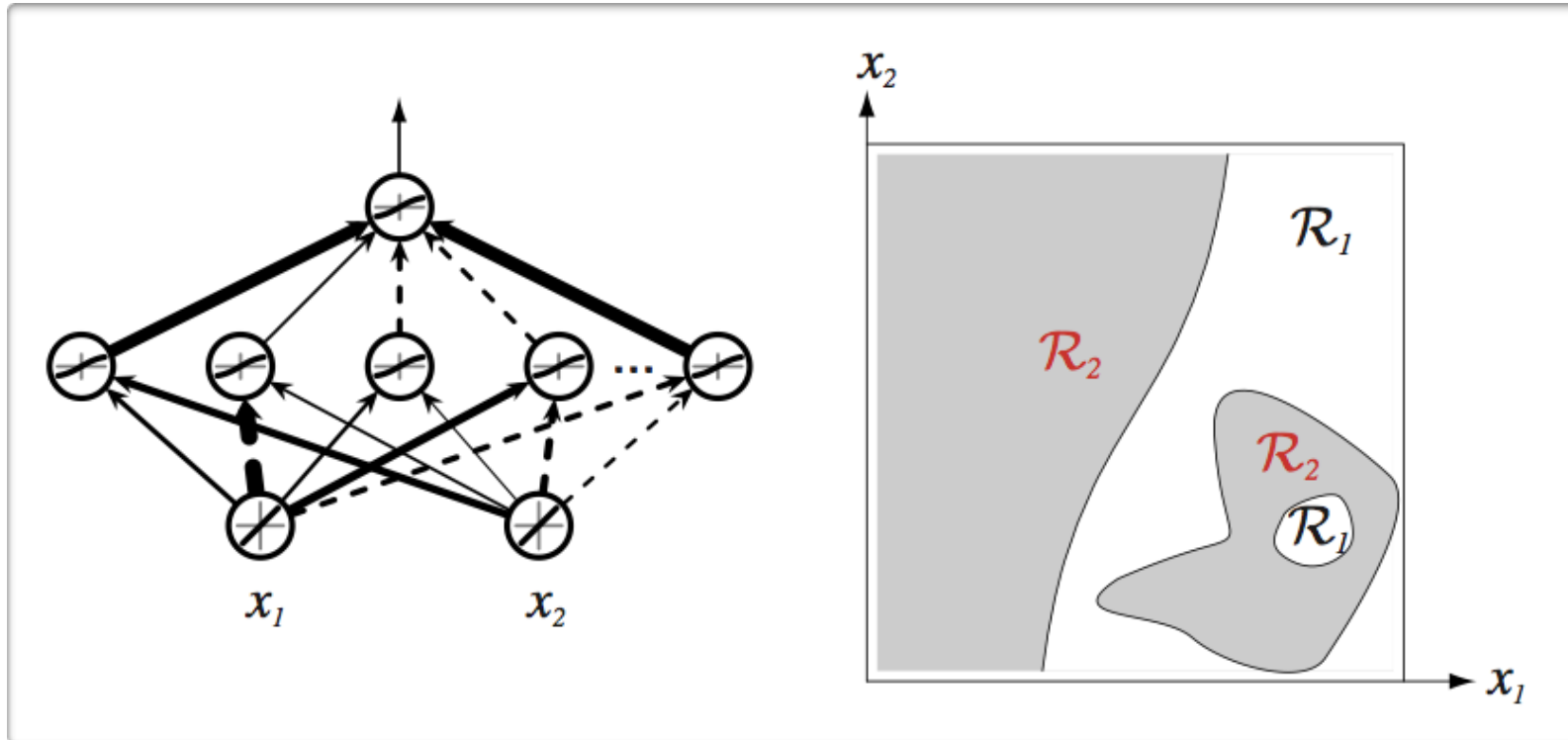
- Consider a single layer neural network:



(from Pascal Vincent's slides)

Capacity of Neural Nets

- Consider a single layer neural network:



(from Pascal Vincent's slides)

Outline

- ▶ Definition of Neural Networks
 - ▶ Forward propagation, Types of units, Capacity of neural networks
- ▶ Training Neural Networks
 - ▶ Loss function, Backpropagation algorithm
- ▶ Optimization/Regularization techniques
 - ▶ Dropout, Batch normalization, Best Practices
- ▶ Convolutional Neural Networks
 - ▶ Definition, Architecture Search
- ▶ Unsupervised Learning, Statistical Generative Models
 - ▶ Variational Autoencoders
 - ▶ Generative Adversarial Networks

Supervised Learning

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

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

where

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

Supervised Learning

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

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

where

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

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

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

Training

► Empirical Risk Minimization

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

► 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: $\theta \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
 - For $t=1:T$
 - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

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

$$\theta \leftarrow \theta + \alpha \Delta$$



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:

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

- ▶ 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

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

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

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

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

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

- **Adam**: essentially combines RMSProp with momentum

Outline

- ▶ Definition of Neural Networks
 - ▶ Forward propagation, Types of units, Capacity of neural networks
- ▶ Training Neural Networks
 - ▶ Loss function, Backpropagation algorithm
- ▶ Optimization/Regularization techniques
 - ▶ Dropout, Batch normalization, Best Practices
- ▶ Convolutional Neural Networks
 - ▶ Definition, Architecture Search
- ▶ Unsupervised Learning, Statistical Generative Models
 - ▶ Variational Autoencoders
 - ▶ Generative Adversarial Networks

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

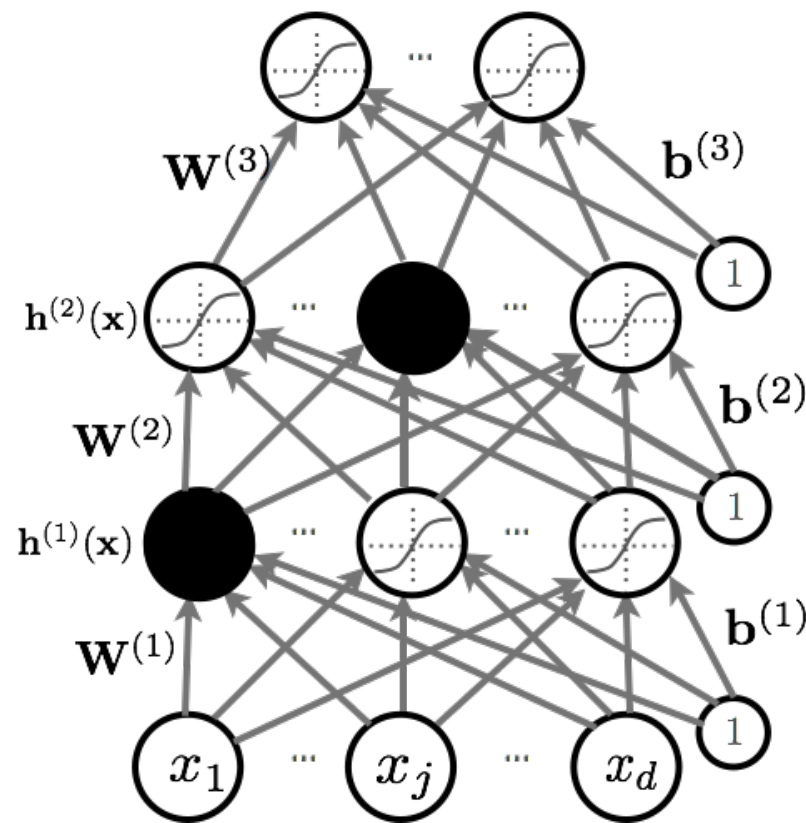
- ▶ **Key idea:** Cripple neural network by removing hidden units stochastically

- ▶ 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



Dropout

- ▶ Use random binary masks $\mathbf{m}^{(k)}$

- ▶ Layer pre-activation for $k > 0$

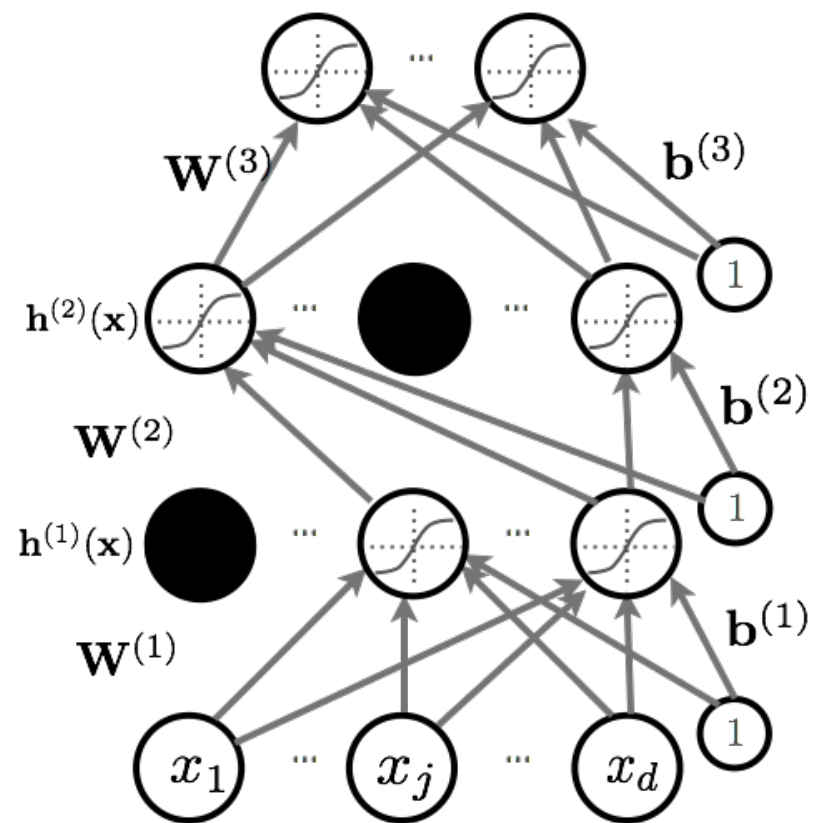
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)} \mathbf{h}^{(k-1)}(\mathbf{x})$$

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

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

- ▶ Output activation ($k=L+1$)

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



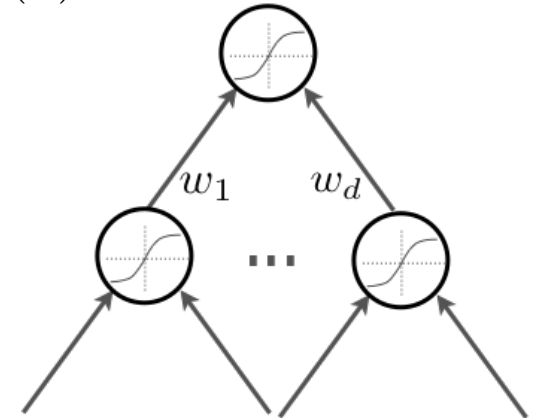
Dropout at Test Time

- ▶ 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: $\mathcal{B} = \{x_1 \dots x_m\}$;

Parameters to be learned: γ, β

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

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

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

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

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

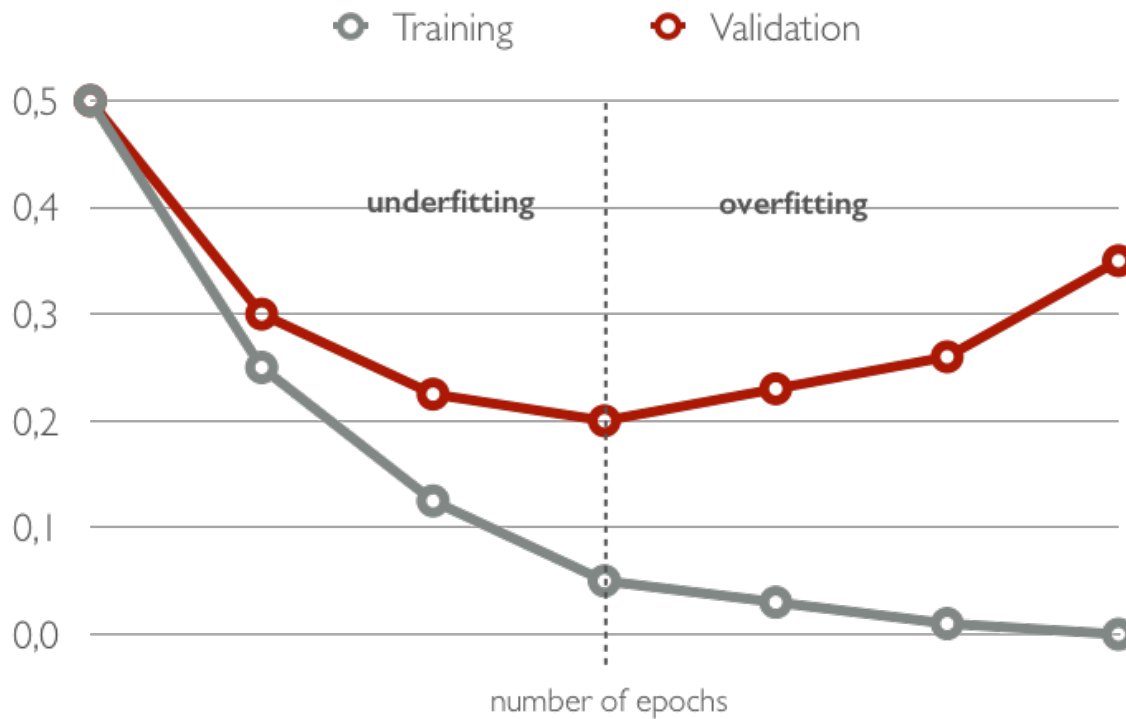
Learned linear transformation to adapt to non-linear activation function (γ and β are trained)

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**.

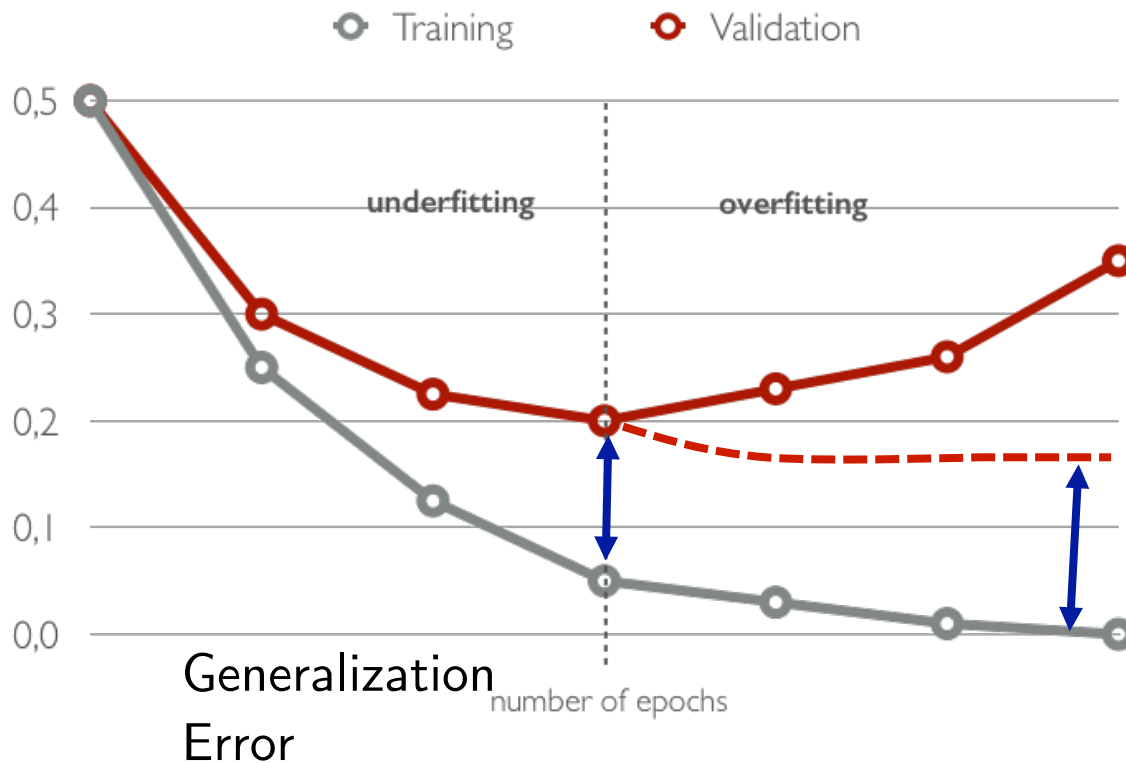
Early Stopping

- ▶ 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



Implicit Regularization

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

Best Practice

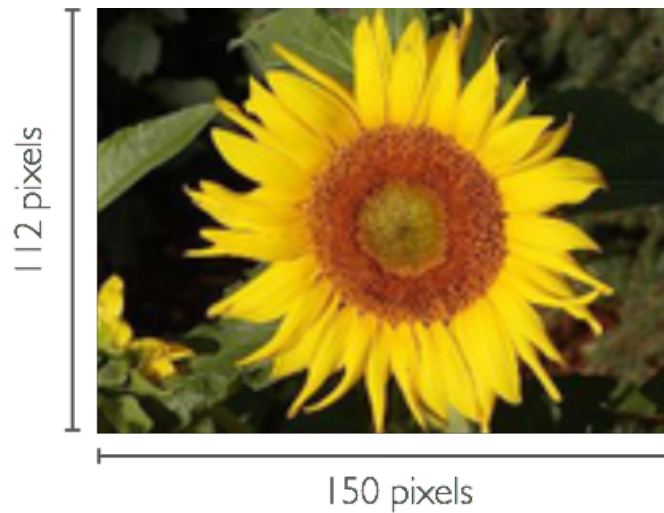
- ▶ Given a dataset D , pick a model so that:
 - ▶ You can achieve 0 training error \rightarrow Overfit on the training set.
- ▶ Regularize the model (e.g. using Dropout).
- ▶ Initialize parameters so that each feature across layers has similar variance. **Avoid units in saturation.**
- ▶ SGD with **momentum,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

Outline

- ▶ Definition of Neural Networks
 - ▶ Forward propagation, Types of units, Capacity of neural networks
- ▶ Training Neural Networks
 - ▶ Loss function, Backpropagation algorithm
- ▶ Optimization/Regularization techniques
 - ▶ Dropout, Batch normalization, Best Practices
- ▶ Convolutional Neural Networks
 - ▶ Definition, Architecture Search
- ▶ Unsupervised Learning, Statistical Generative Models
 - ▶ Variational Autoencoders
 - ▶ Generative Adversarial Networks

Computer Vision

- **Object recognition:** Given an input image, identify which object it contains



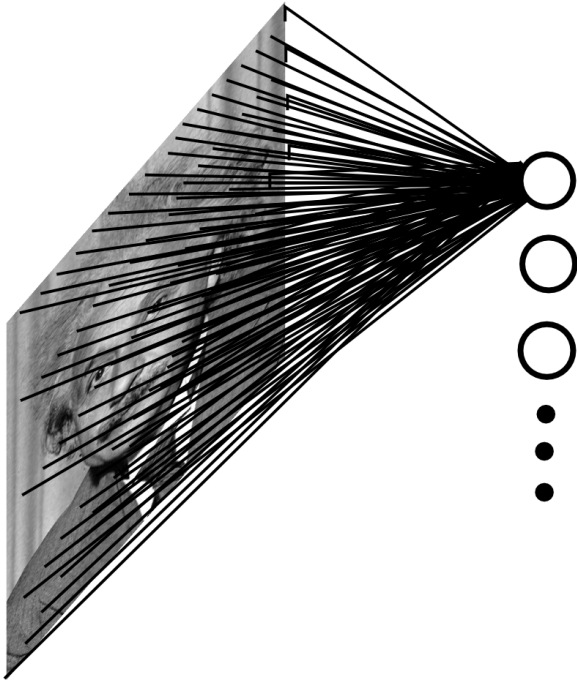
"sun flower"

Computer Vision

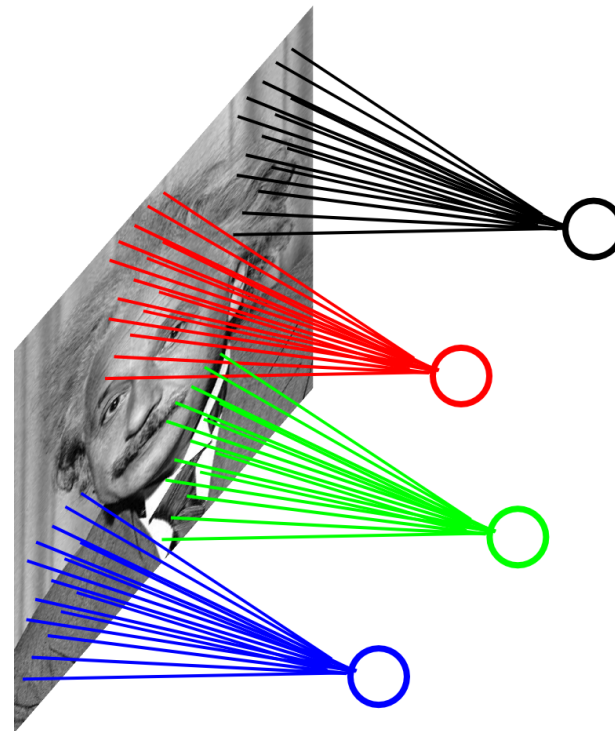
- ▶ Design neural networks that are specifically adapted for such problems:
 - ▶ Must deal with very **high-dimensional inputs**: 150×150 pixels = 22500 inputs, or 3×22500 if RGB pixels
 - ▶ Can exploit the **2D topology** of pixels (or 3D for video data)
 - ▶ Can build in **invariance** to certain variations: translation, illumination, etc.
- ▶ **Convolutional networks** leverage these ideas
 - ▶ Local connectivity
 - ▶ Parameter sharing
 - ▶ Convolution
 - ▶ Pooling / subsampling

Local Connectivity

- ▶ **Local connectivity** of hidden units
 - ▶ Each hidden unit is connected only to a sub-region (patch) of the input image
 - ▶ Spatial correlation is local



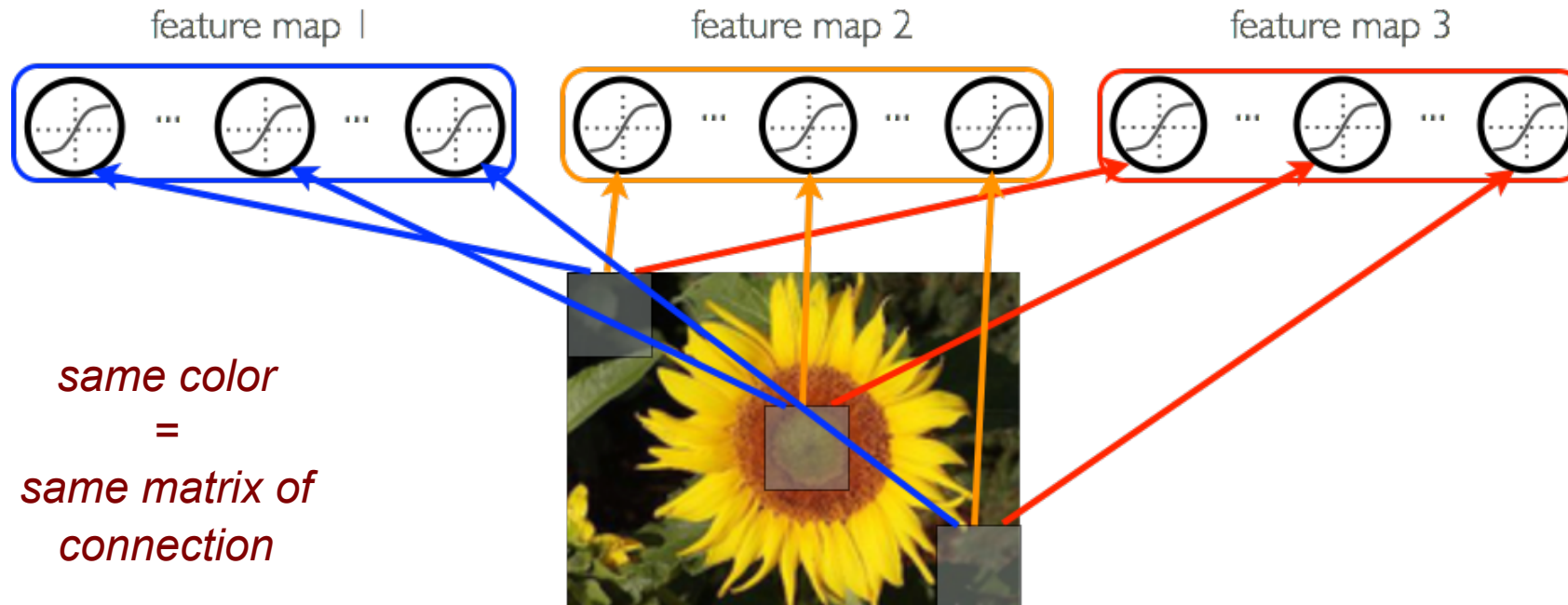
Fully Connected: 200x200 image, 40K hidden units, **~2B parameters**



Locally Connected: 200x200 image, filter size 10x10, **4M parameters!**

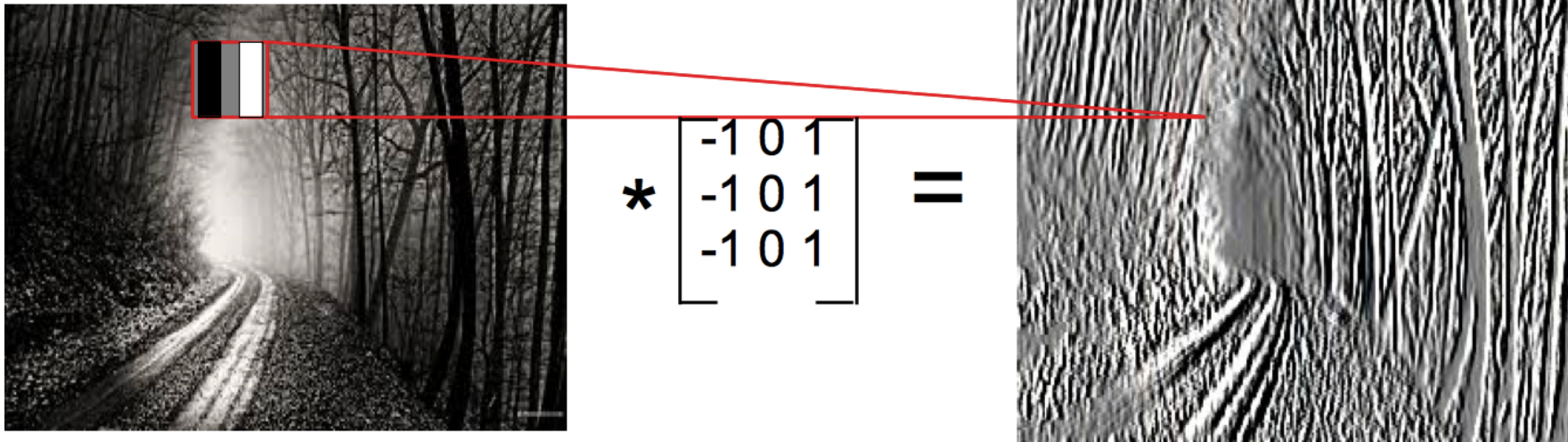
Parameter Sharing

- ▶ Share matrix of parameters across some units
 - ▶ Units that share parameters represent “feature map”
 - ▶ Units within a feature map cover different positions in the image



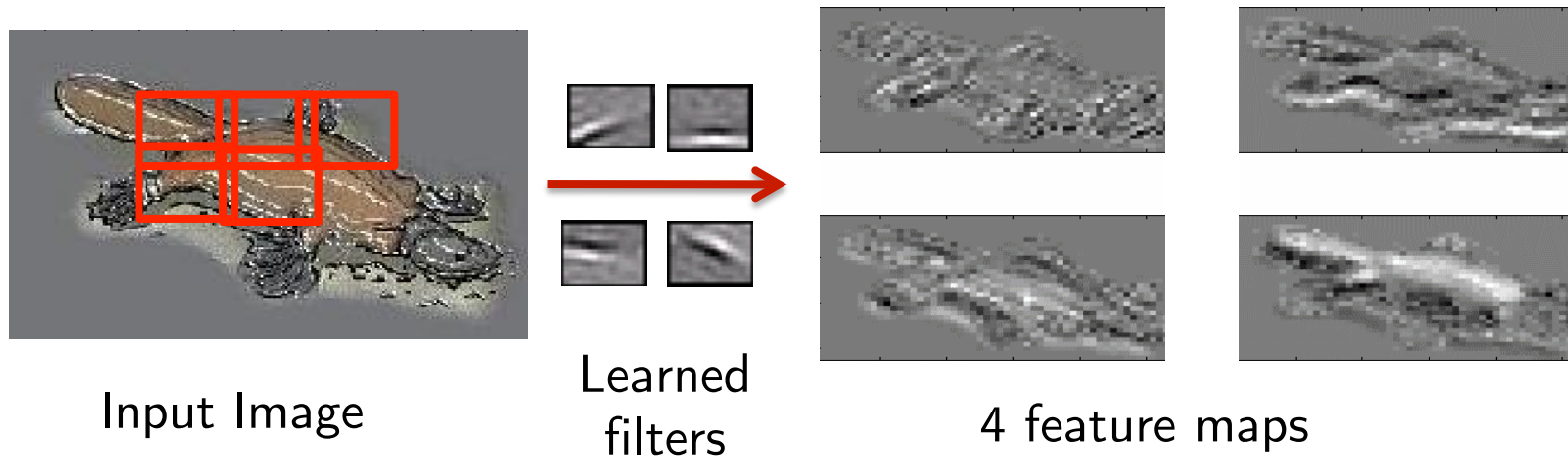
Convolution

- ▶ Each feature map forms a 2D grid of features
 - ▶ Can be computed with a discrete convolution of a **kernel matrix** K_{ij} which is the weights matrix W_{ij} with its rows and columns flipped



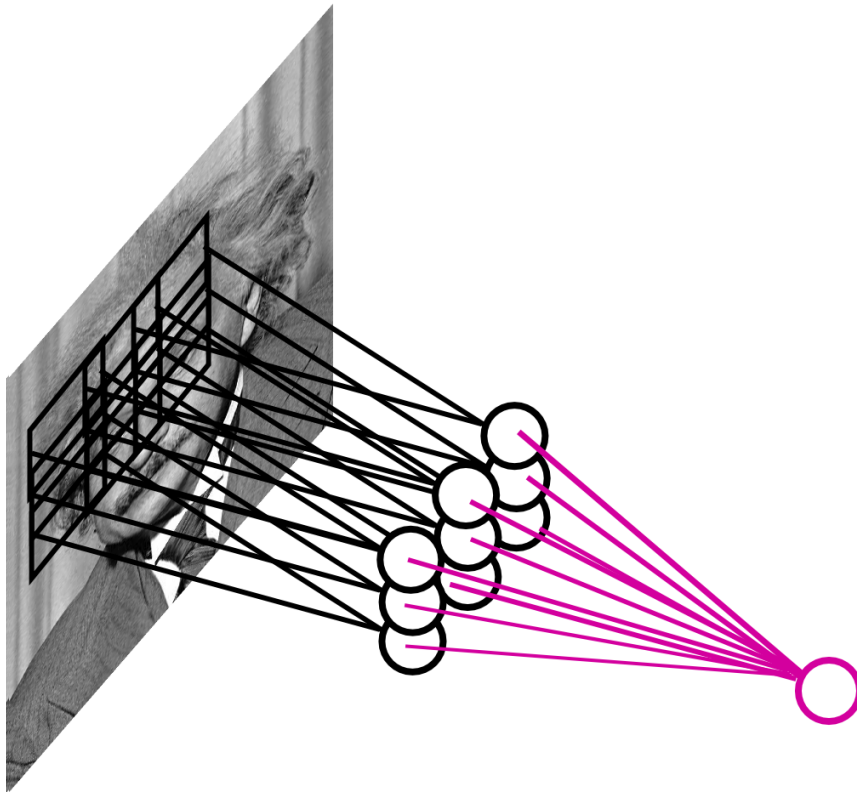
Convolution

- ▶ Each feature map forms a 2D grid of features
 - ▶ Can be computed with a discrete convolution of a **kernel matrix** k_{ij} which is the weights matrix W_{ij} with its rows and columns flipped

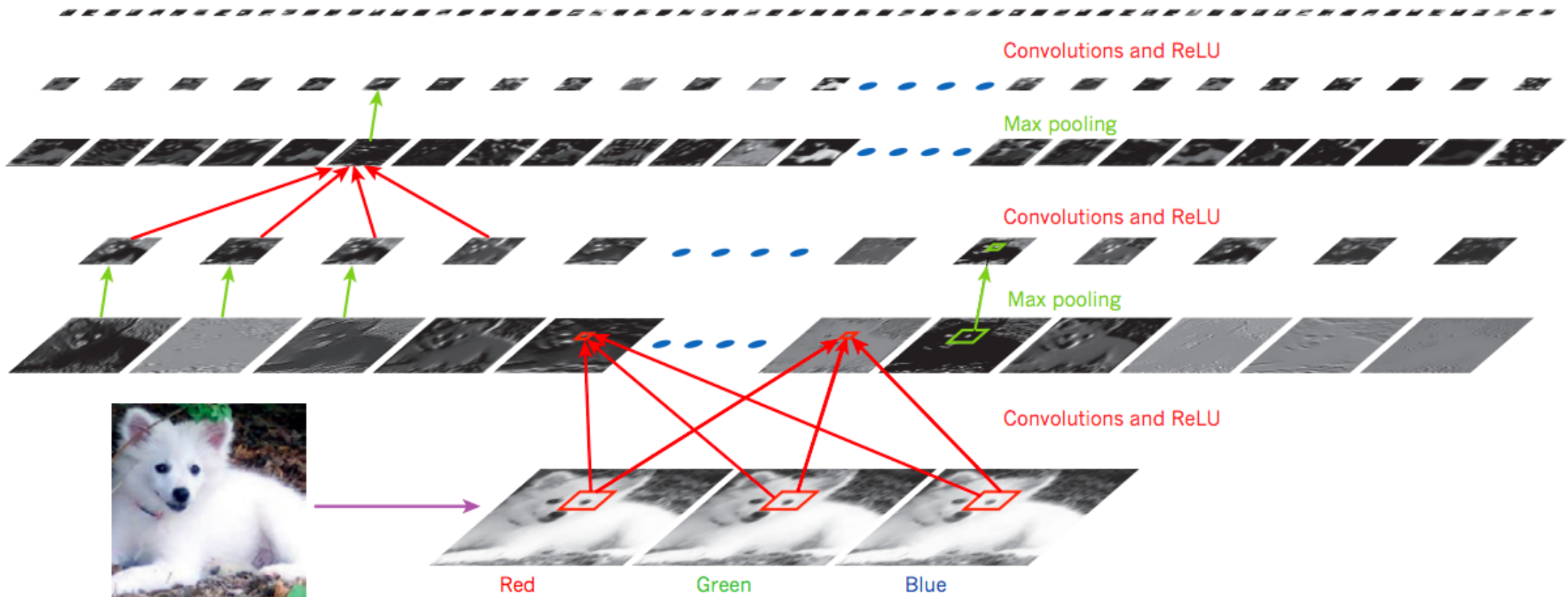


Pooling

- Make the detection robust to the exact location of the eye



Convolutional Neural Network (ConvNet)



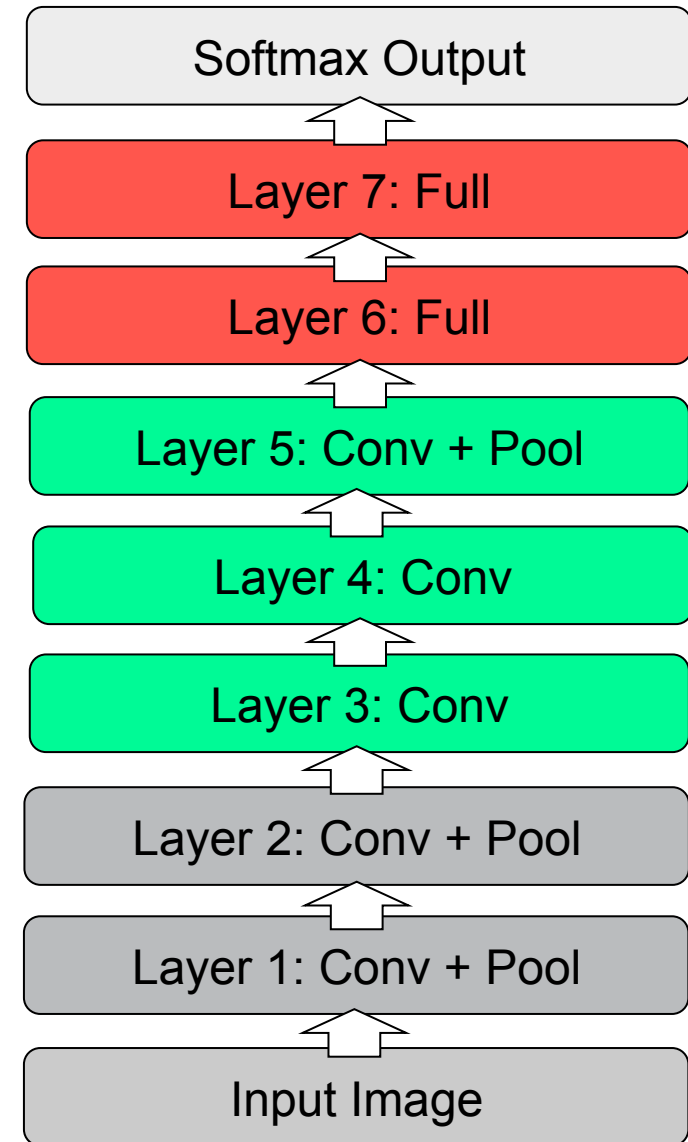
The outputs (not the filters) of each layer (horizontally) of a typical convolutional network architecture applied to the image of a Samoyed dog

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

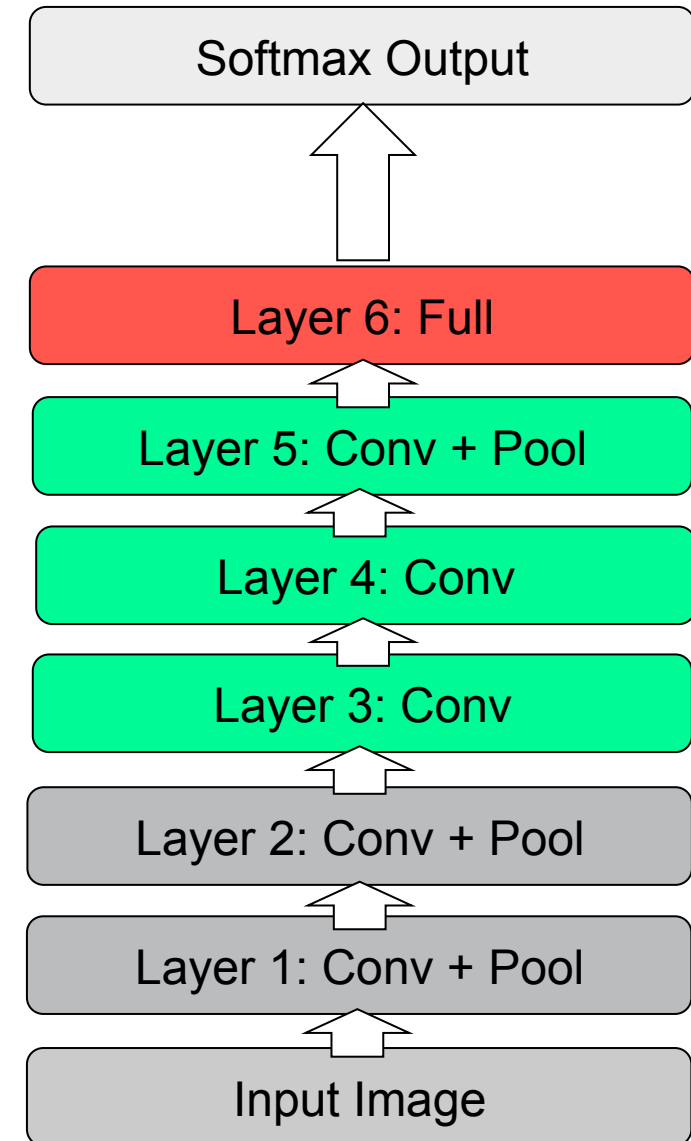
AlexNet

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



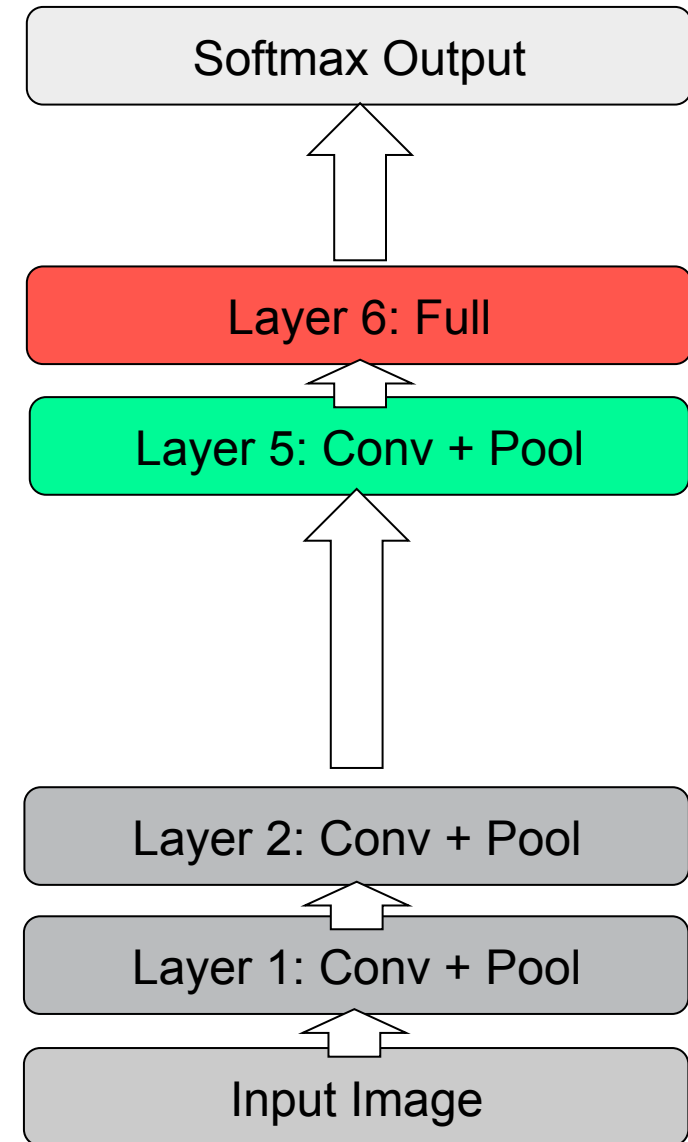
AlexNet

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

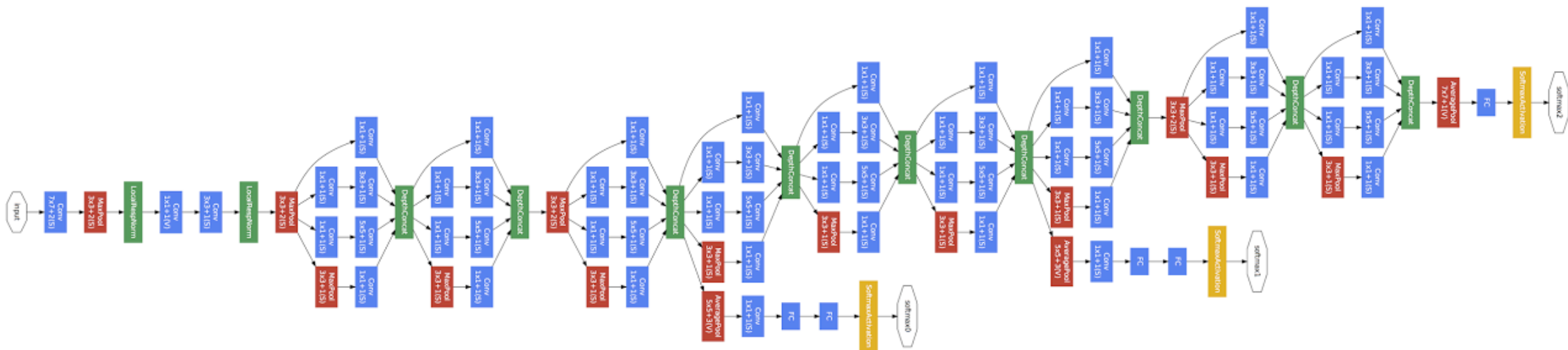


AlexNet

- ▶ Remove layers 3 4,6 and 7
- ▶ Drop ~50 million parameters
- ▶ 33.5% drop in performance!
- ▶ Depth of the network is the key



- ▶ 24 layer model

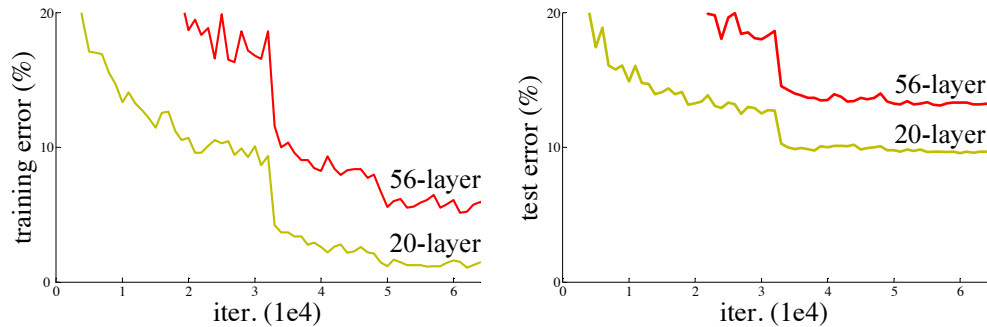


Convolution
Pooling
Softmax
Other

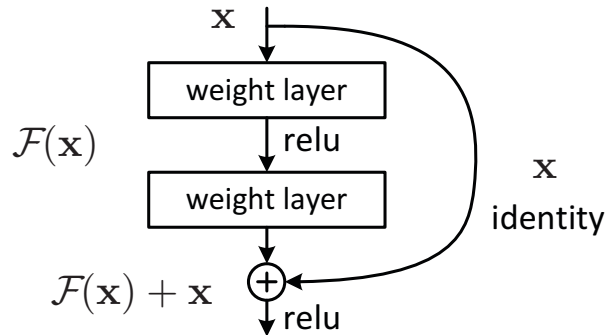
(Szegedy et al., Going Deep with Convolutions, 2014)

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:

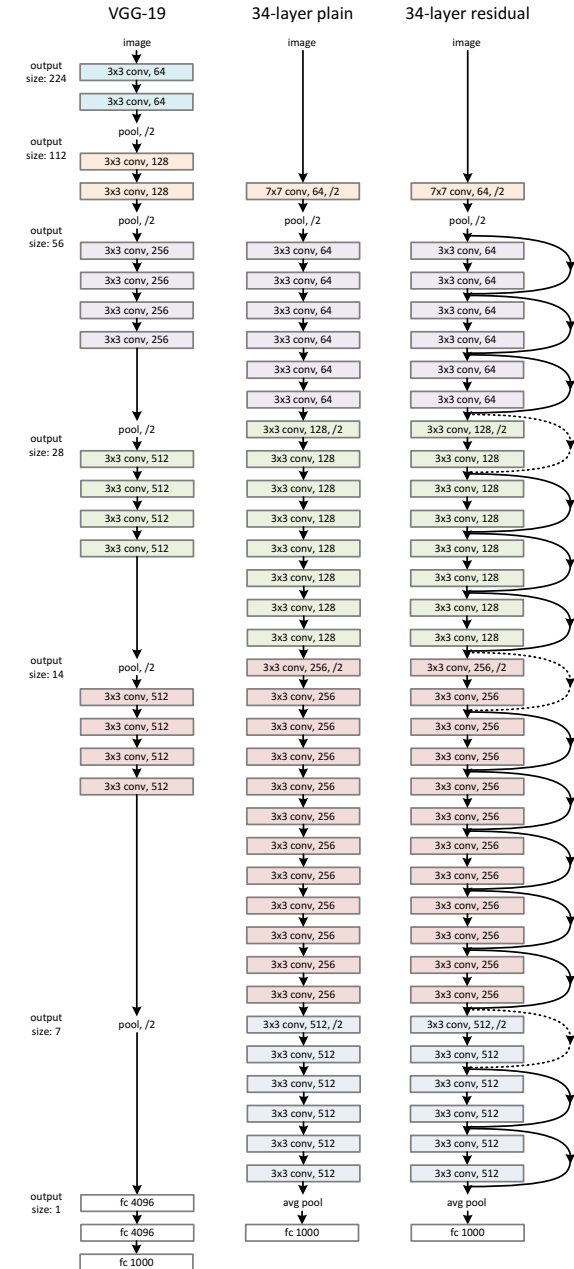


(He, Zhang, Ren, Sun, CVPR 2016)

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



Outline

- ▶ Definition of Neural Networks
 - ▶ Forward propagation, Types of units, Capacity of neural networks
- ▶ Training Neural Networks
 - ▶ Loss function, Backpropagation algorithm
- ▶ Optimization/Regularization techniques
 - ▶ Dropout, Batch normalization, Best Practices
- ▶ Convolutional Neural Networks
 - ▶ Definition, Architecture Search
- ▶ Unsupervised Learning, Statistical Generative Models
 - ▶ Variational Autoencoders
 - ▶ Generative Adversarial Networks