ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Machine Learning for DH

Mathieu Salzmann

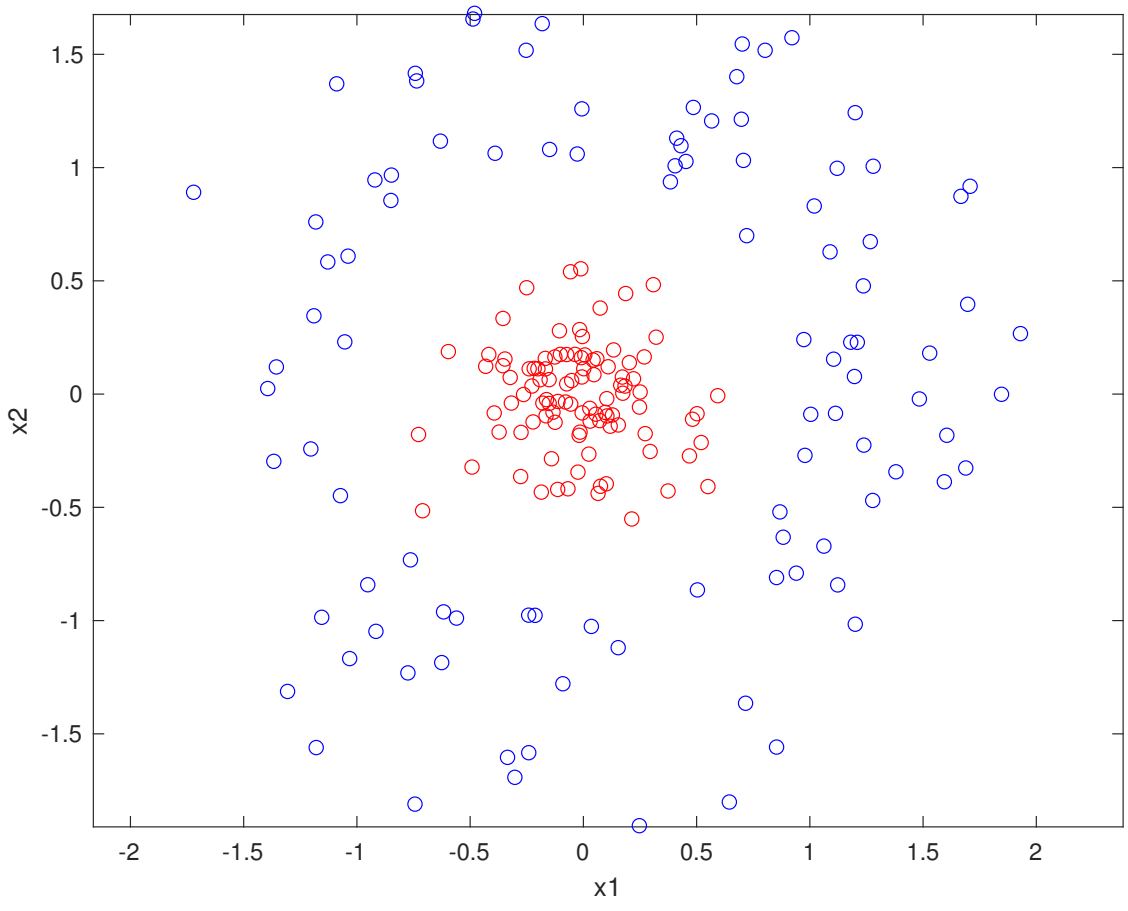
▶ Digital Humanities

Deep Learning

Learning goals:

- Characterize the Multi-Layer Perceptron;
- Characterize convolutional neural networks;
- Derive the principles of backpropagation;
- Analyze the differences between different models;
- Choose the right model for a given problem;
- Implement basic architectures within existing frameworks.

Recap: Nonlinearly-separable Data



Recap: Combining Models

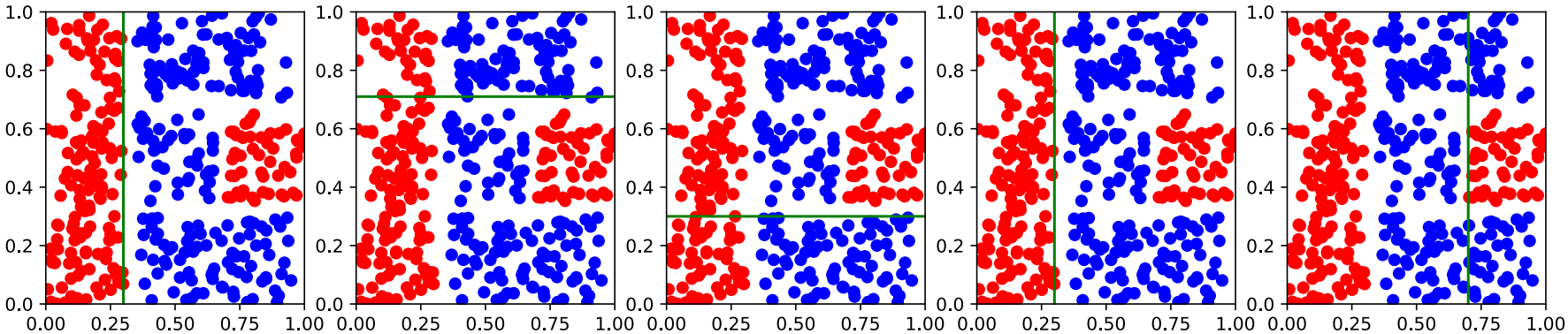
1. Learn one global model for the entire input space
   - E.g., boosting

2. Divide the input space to have local models
   - E.g., decision trees

Recap: AdaBoost

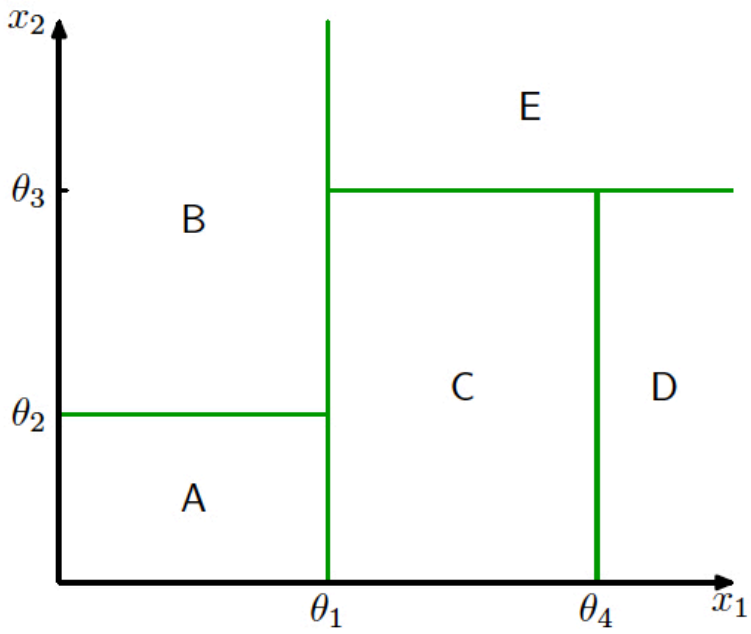
$$\hat{y}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \hat{y}_m(\mathbf{x})\right) ,$$

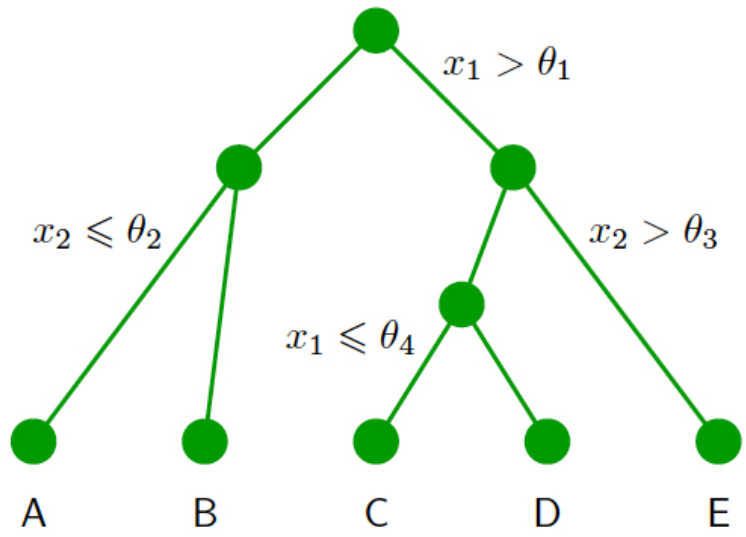
- Learning is sequential
  - Each weak learner aims to improve the current classifier
  - The samples are re-weighted according to their current prediction



Recap: Decision Trees

- Partition the input space recursively
- Each region is assigned one class (or one value for regression)





Figures from C. Bishop's book

Combining Models

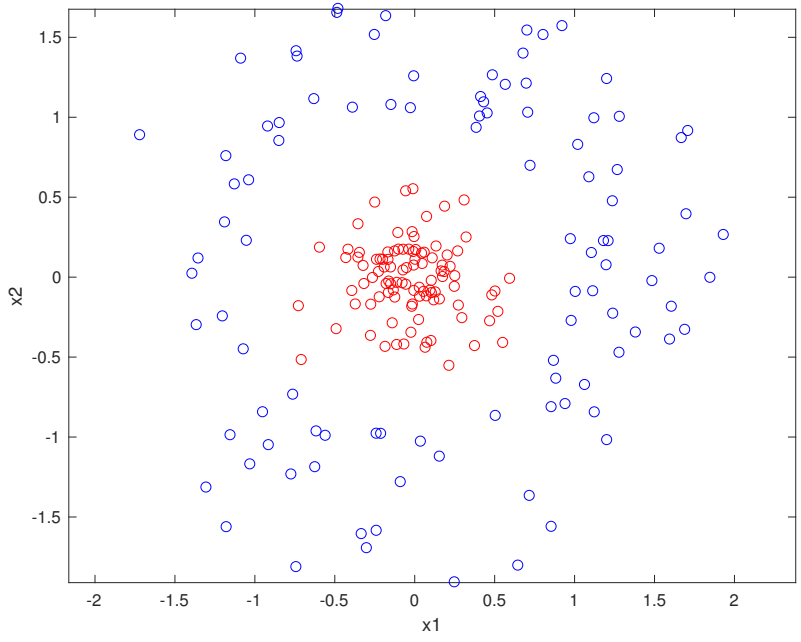
+ Can effectively handle nonlinearities

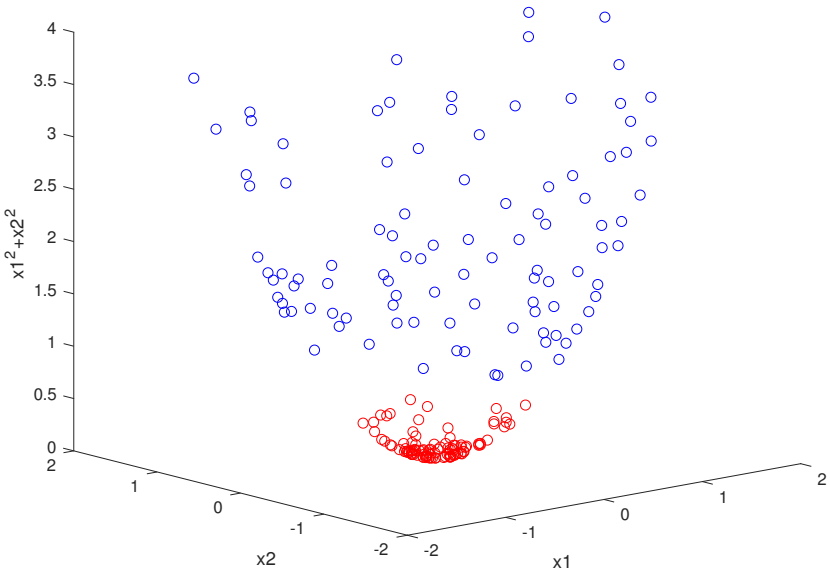
+ Intuitive solutions

- May overfit

Recap: Mapping to a Higher-dimensional Space

- $[x_1; x_2] \rightarrow \phi([x_1; x_2]) = [x_1; x_2; x_1^2 + x_2^2]$





Recap: The Kernel Trick

- An inner product between data point

$$\langle \mathbf{x}_i, \mathbf{x}_j \rangle = \mathbf{x}_i^T \mathbf{x}_j$$

  can be replaced by a kernel value

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

→ Kernel ridge regression

→ Kernel support vector machines

Kernel Methods

+ Can effectively handle nonlinearities
+ Often have nice solutions (e.g., closed-form, convex)

- Depend on the choice of kernel
- The complexity depends on the number of samples

Until Now...

All the methods we have seen

+ can effectively handle linear and nonlinear data

- depend on the choice of input representation
  - E.g., using raw pixel values or HOG features will not give the same solution

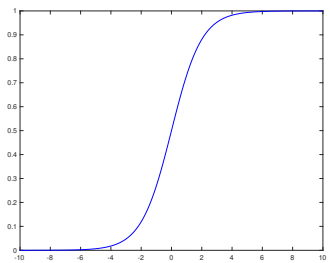
→ How about learning the representation itself?

Before we get there...

- Let us look back at logistic regression

$$\hat{y}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x} + b) \ ,$$

  with $\sigma(\cdot)$ the sigmoid function



- Lesson learned: make use of nonlinearity

The Perceptron Algorithm

you just check that there is something

- In fact, this idea was already exploited by the *Perceptron* algorithm (Rosenblatt 1962), which uses

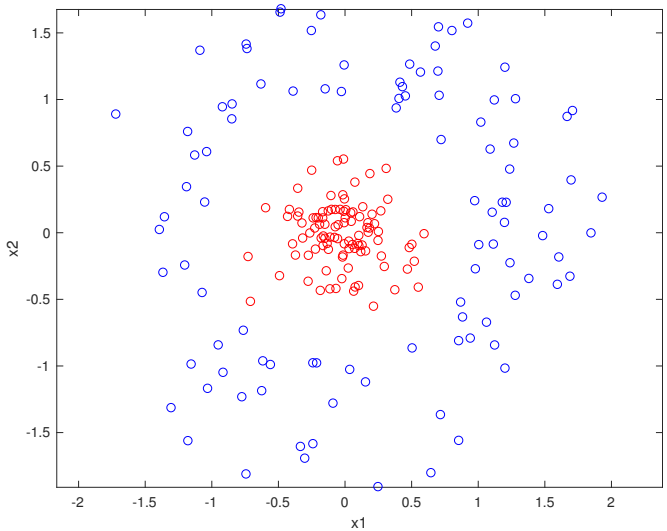
$$\hat{y}(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0 \text{ (or } -1), & \text{otherwise} \end{cases}$$

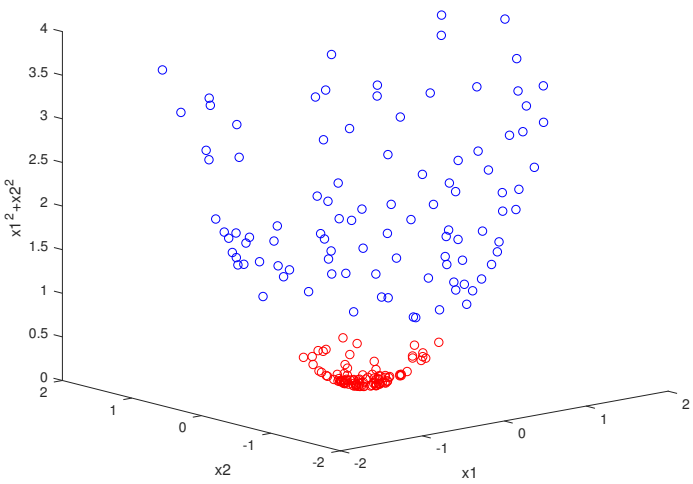
- $f(\cdot)$ was dubbed an *activation function*

And in a different direction...

- Kernel methods inherently map the data to a different representation

- Lesson learned: transforming the input data can help





Now let's put this all together...

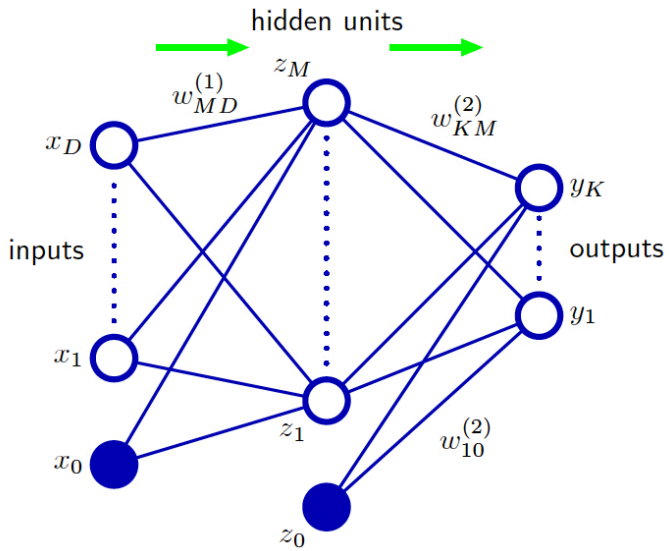
- A simple artificial neural network
  - We map the data to a different representation (*hidden units*)

$$\mathbf{z}_i = f^{(1)}(\mathbf{W}_i^{(1)}\mathbf{x} + \mathbf{b}_i^{(1)})$$

  - The outputs are obtained by another mapping

$$\hat{\mathbf{y}}_j = f^{(2)}(\mathbf{W}_j^{(2)}\mathbf{z} + \mathbf{b}_j^{(2)})$$

  - Each layer relies on an activation function

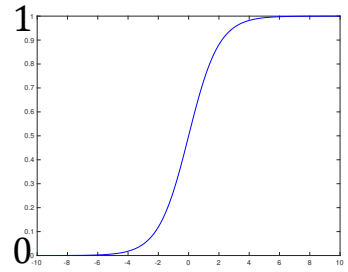


every zi will depend on every input x

Activation Functions: Hidden Layer

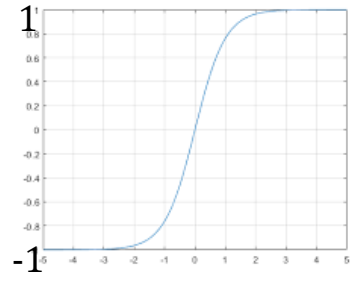
- Sigmoid

$$f(a) = \sigma(a) = \frac{1}{1 + \exp(-a)}$$



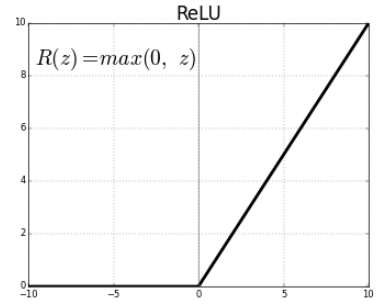
- Hyperbolic tangent

$$f(a) = \tanh(a)$$



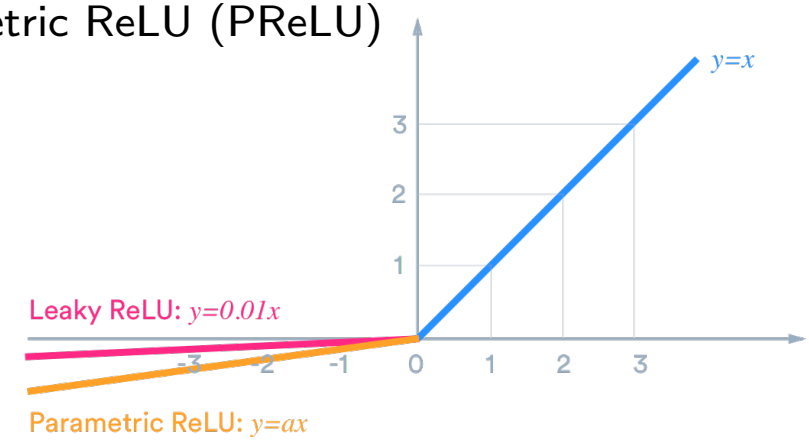
- Rectified linear unit (ReLU)

$$f(a) = \begin{cases} a, & \text{if } a \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

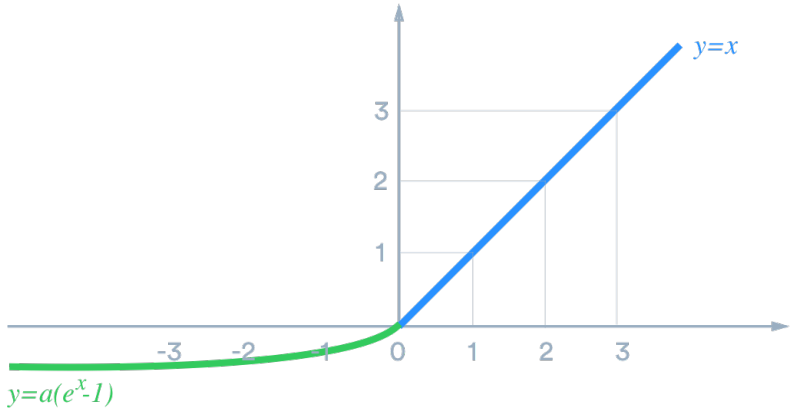


Activation Functions: Variants of ReLU

- Leaky ReLU & Parametric ReLU (PReLU)



- Exponential Linear (ELU, SELU)



Activation Functions: Final Layer (Output)

softmax => probabilities (as for handwritten numbers)

- For classification, the last layer typically relies on the softmax function

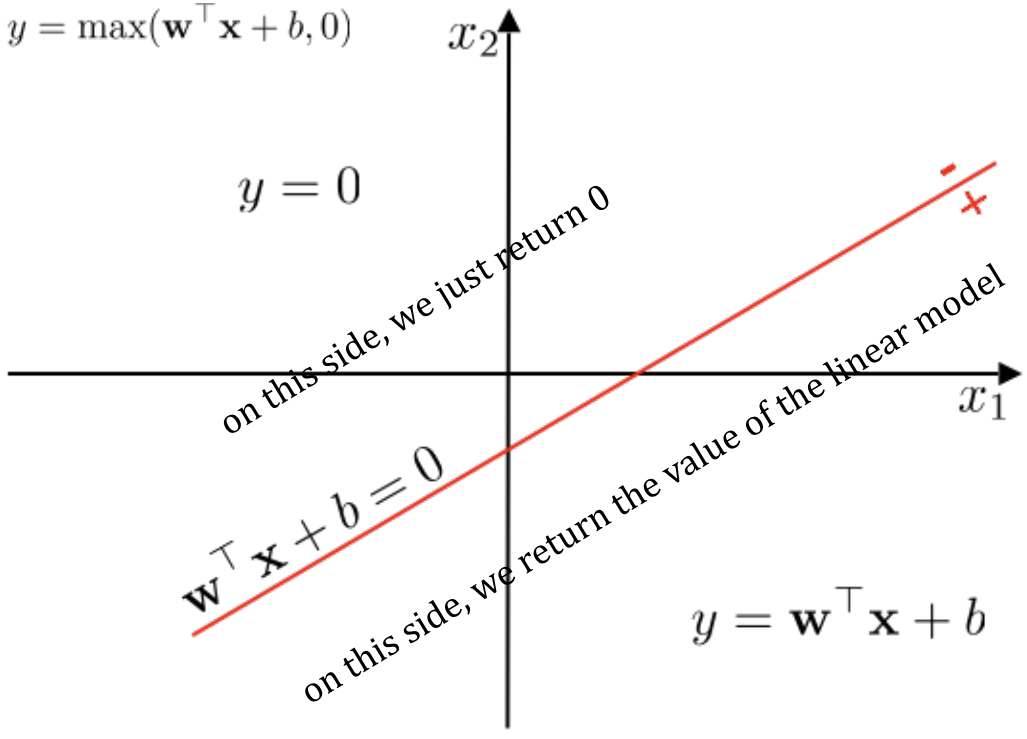
$$\hat{\mathbf{y}}_j = \frac{\exp(\mathbf{W}_j^{(2)}\mathbf{z} + \mathbf{b}_j^{(2)})}{\sum_{k=1}^{K} \exp(\mathbf{W}_k^{(2)}\mathbf{z} + \mathbf{b}_k^{(2)})}$$

- For regression, a linear activation (i.e., no activation function) is usually employed

Simple Network: Visualization

Single hidden layer

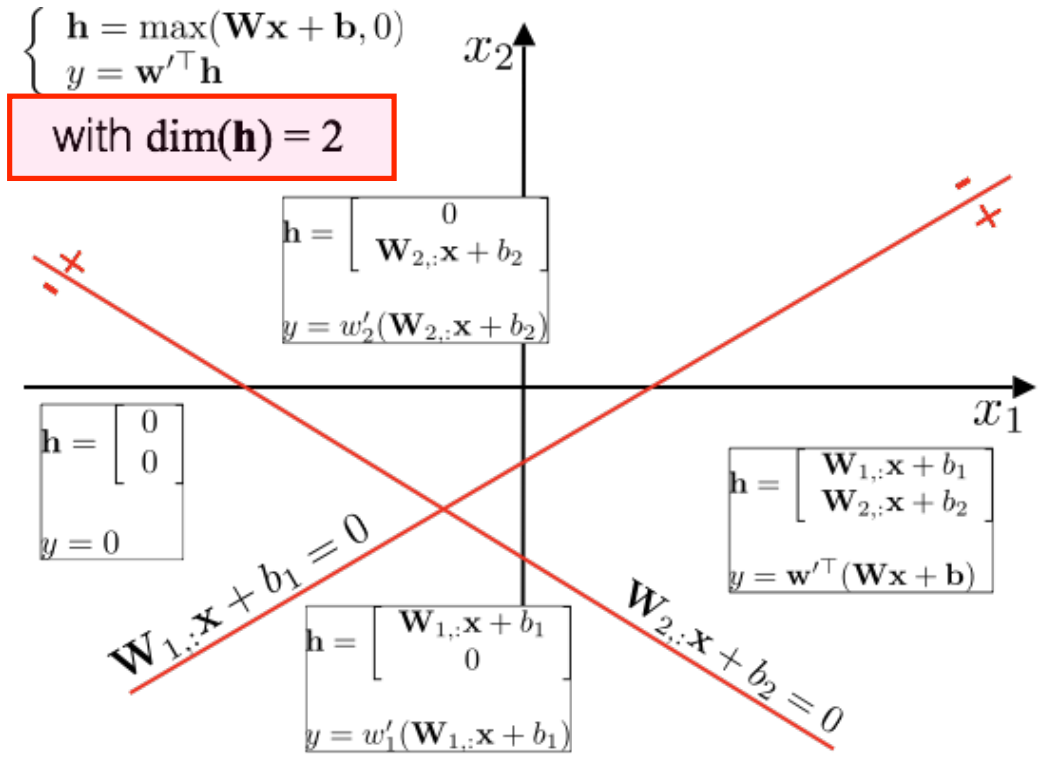
- Single hidden unit with ReLU activation



Simple Network: Visualization

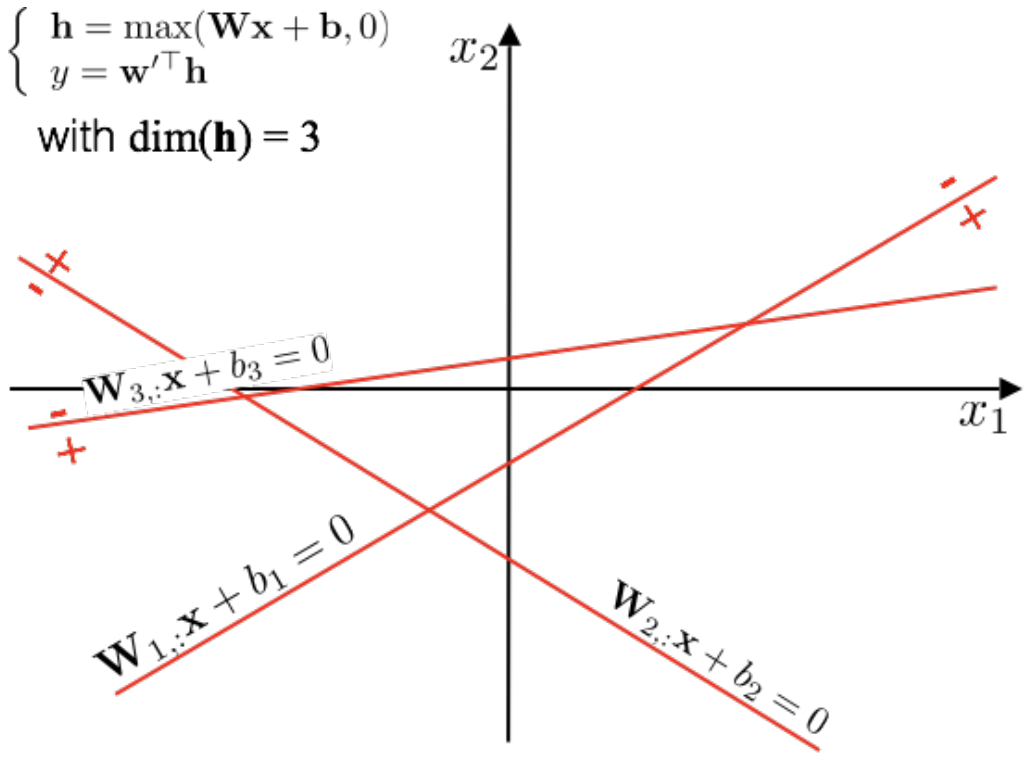
Single hidden layer

- Two hidden units with ReLU activation



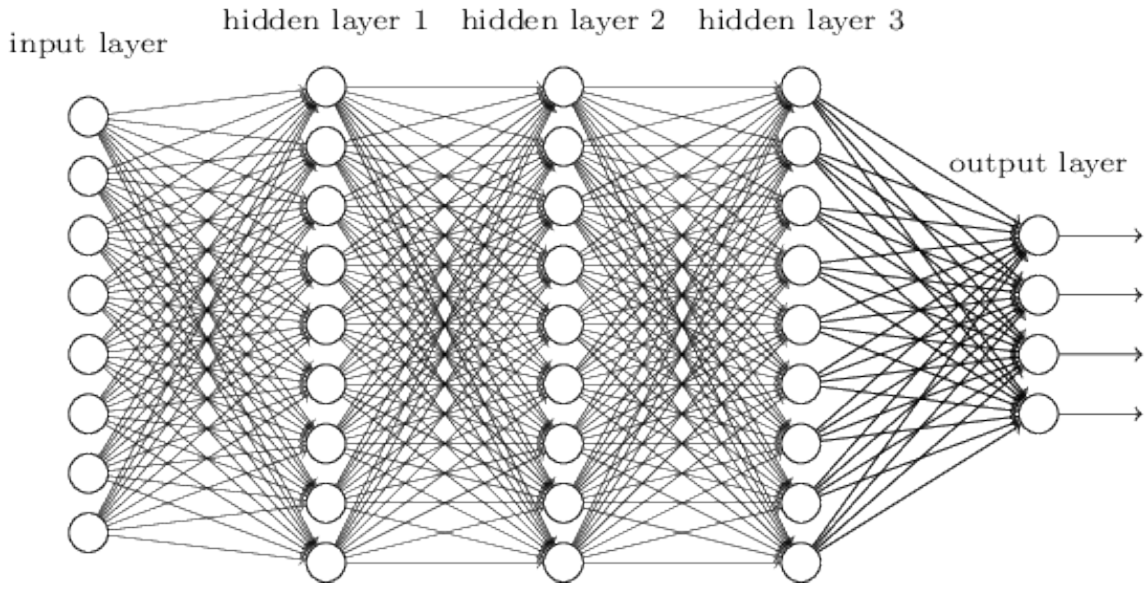
Simple Network: Visualization

- Three hidden units with ReLU activation



Multilayer Perceptron

- Why stop with a single hidden layer?

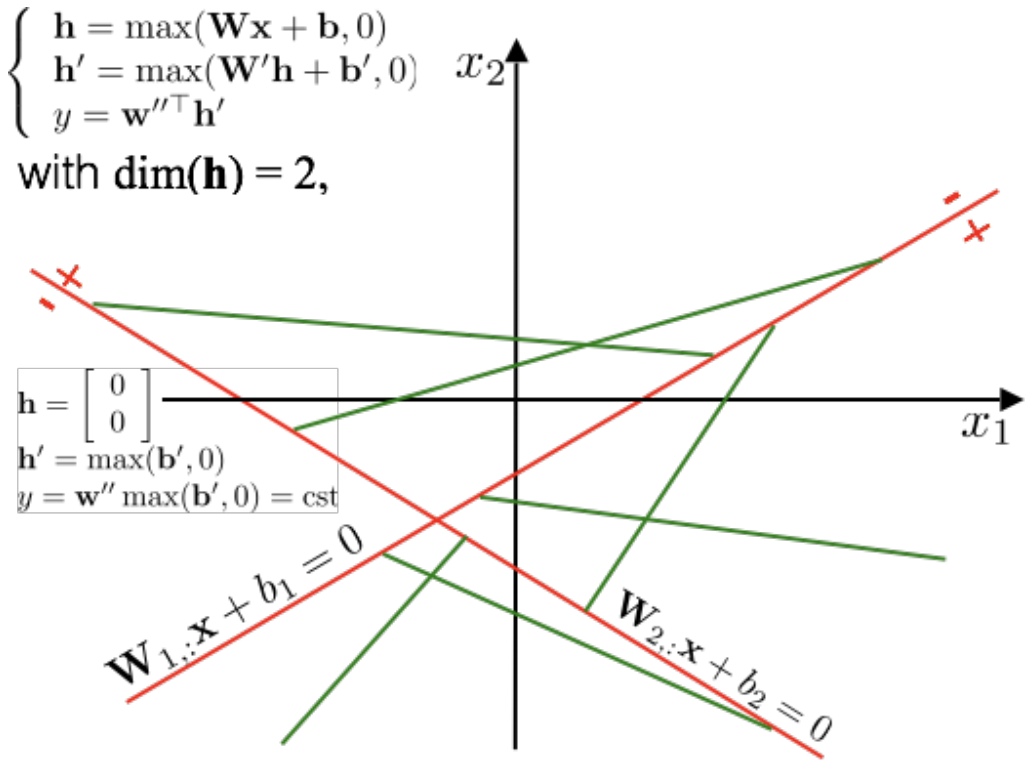


- Slightly misleading name
  - The Perceptron relies on a step function as activation function
  - MLPs typically rely on continuous activation functions (e.g., sigmoid)

e.g sigmoid, tanh, etc.

Two-Layer Network: Visualization

- Two layers with two units each with ReLU activation



Training a Neural Network

- As usual, training is done by minimizing an empirical risk R
  - For regression, one typically uses the square loss

$$R = \frac{1}{N} \sum_{i=1}^{N} \|\hat{\mathbf{y}}_i - \mathbf{y}_i\|_2^2$$

  - For classification, the standard choice is the cross-entropy

$$R = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbf{y}_i^k \ln \hat{\mathbf{y}}_i^k$$

when y_hat is linear, the model is convex

- Now, however, the problem is not convex anymore
  - particularly because of the multiple nonlinear activations

→ Learning is typically done by gradient descent

Gradient-based Learning

- Computing the gradient of a multilayer architecture is seemingly complicated
- However, we can make use of the chain rule

- First, since the empirical risk has the form

$$R(\theta) = \sum_{i=1}^{N} E_i(\theta) \ ,$$

  we can consider the error of a single sample $E_i(\cdot)$

- Furthermore, here, we ignore the bias $\mathbf{b}$ of every layer for simplicity
  - As before, the bias can be thought as one more parameter in $\mathbf{W}$ with value 1 as a corresponding feature

Gradient-based Learning

- Recall that, we can write the output of any layer ℓ as

$$\mathbf{z}_k^{(\ell)} = f(\mathbf{a}_k^{(\ell)}) = f\left(\sum_j \mathbf{W}_{kj}^{(\ell)} \mathbf{z}_j^{(\ell-1)}\right)$$

- Following the chain rule, we have

$$\frac{\partial E_i}{\partial \mathbf{W}_{kj}^{(\ell)}} = \frac{\partial E_i}{\partial \mathbf{a}_k^{(\ell)}} \frac{\partial \mathbf{a}_k^{(\ell)}}{\partial \mathbf{W}_{kj}^{(\ell)}}$$

- Furthermore, from the definition above, we have

$$\frac{\partial \mathbf{a}_k^{(\ell)}}{\partial \mathbf{W}_{kj}^{(\ell)}} = \mathbf{z}_j^{(\ell-1)}$$

Gradient-based Learning

- Let us define

$$\delta_k^{(\ell)} = \frac{\partial E_i}{\partial \mathbf{a}_k^{(\ell)}}$$

- Now, we have

$$\frac{\partial E_i}{\partial \mathbf{W}_{kj}^{(\ell)}} = \delta_k^{(\ell)} \mathbf{z}_j^{(\ell-1)}$$

- For the last layer (layer L), $\delta_k^{(L)}$ can be computed explicitly. E.g., for the square loss,

$$\delta_k^{(L)} = \hat{\mathbf{y}}_k - \mathbf{y}_k$$

Gradient-based Learning

- Let us now look at layer L-1
- Following the chain rule, we have

$$\delta_k^{(L-1)} = \frac{\partial E_i}{\partial \mathbf{a}_k^{(L-1)}} = \sum_m \frac{\partial E_i}{\partial \mathbf{a}_m^{(L)}} \frac{\partial \mathbf{a}_m^{(L)}}{\partial \mathbf{a}_k^{(L-1)}} = \sum_m \delta_m^{(L)} \frac{\partial \mathbf{a}_m^{(L)}}{\partial \mathbf{a}_k^{(L-1)}}$$

- Now, recall that

$$\mathbf{a}_m^{(L)} = \mathbf{W}_m^{(L)} f(\mathbf{a}^{(L-1)}) = \sum_k \mathbf{W}_{mk}^{(L)} f(\mathbf{a}_k^{(L-1)})$$

- This means that

$$\frac{\partial \mathbf{a}_m^{(L)}}{\partial \mathbf{a}_k^{(L-1)}} = \mathbf{W}_{mk}^{(L)} f'(\mathbf{a}_k^{(L-1)})$$

where $f'(\cdot)$ is the derivative of the activation function w.r.t. its argument

Backpropagation

- This means that

$$\delta_k^{(L-1)} = \sum_m \delta_m^{(L)} \frac{\partial \mathbf{a}_m^{(L)}}{\partial \mathbf{a}_k^{(L-1)}} = f'(\mathbf{a}_k^{(L-1)}) \sum_m \mathbf{W}_{mk}^{(L)} \delta_m^{(L)}$$

- This derivation is true for any layer

→ Given $\delta^{(\ell+1)}$, one can automatically compute $\delta^{(\ell)}$

- This gives us an automatic way to compute the gradient

$$\frac{\partial E_i}{\partial \mathbf{W}_{kj}^{(\ell)}} = \delta_k^{(\ell)} \mathbf{z}_j^{(\ell-1)}$$

  at any layer, starting from the last one

- This process is known as *backpropagation*

Backpropagation: Algorithm

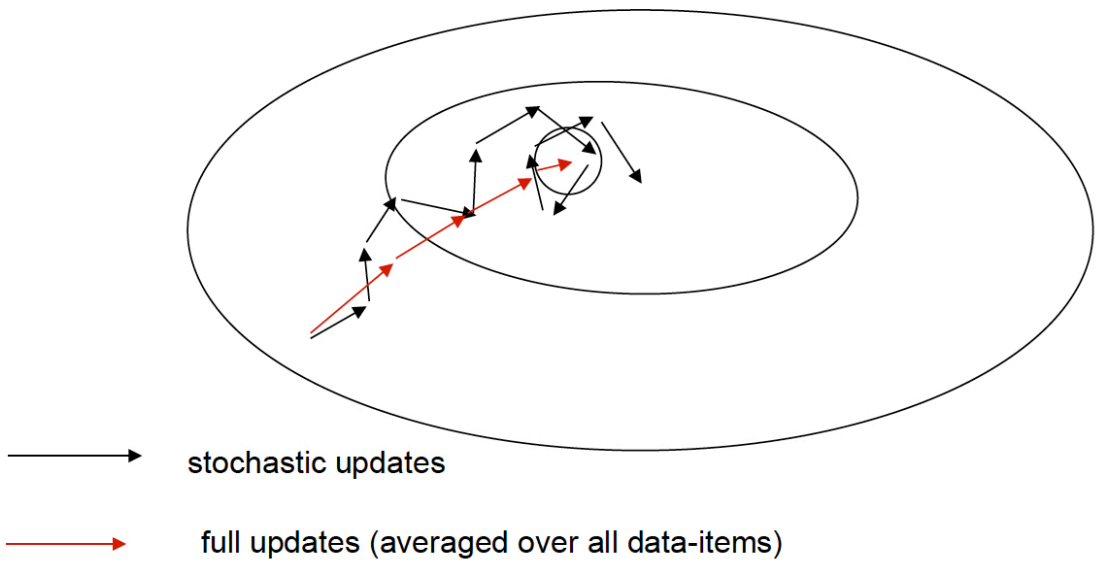
1. Propagate sample i forward through the network
2. Compute $\delta^{(L)}$ depending on the loss
3. Propagate δ backward to get each $\delta^{(\ell)}$
4. At each layer, compute the partial derivatives as

$$\frac{\partial E_i}{\partial \mathbf{W}_{kj}^{(\ell)}} = \delta_k^{(\ell)} \mathbf{z}_j^{(\ell-1)}$$

where $\mathbf{z}^{(\ell-1)}$ has been obtained from the forward pass

Stochastic Gradient Descent

- Standard gradient descent relies on all samples at every iteration
  - This can be expensive for large training sets
- *Stochastic Gradient Descent (SGD)* approximates gradient descent by only considering one, or a few, samples at a time
  - This dances around the minimum, and thus requires decreasing the learning rate
  - It can be effective at avoiding bad local minima



SGD: Extensions

- Momentum
  - Prevents the weights from changing in an inconsistent manner

$$\begin{aligned} \nu &\leftarrow \gamma\nu + \eta\nabla_\theta R(\theta) \\ \theta &\leftarrow \theta - \nu \end{aligned}$$

- Adaptive learning rates
  - Adagrad (Duchi et al., JMLR 2011)
  - Adadelta (Zeiler, 2012)
  - RMSprop (Hinton, 2012)
  - ADAM (Kingma et al., ICLR 2015)

Regularization

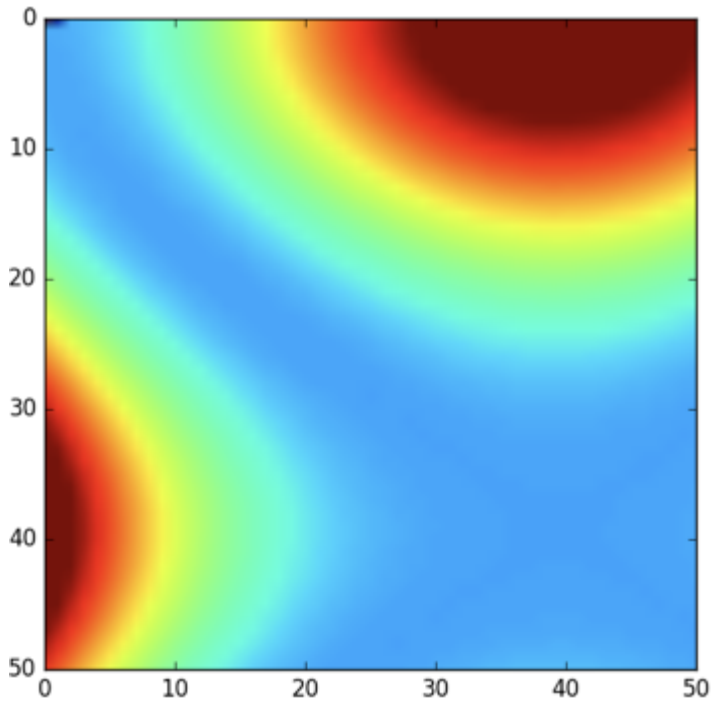
- Deep networks have many parameters and can therefore overfit
- Several strategies can be used. E.g.,
  - Early stopping
  - Weight decay

$$\tilde{R}(\theta) = R(\theta) + \lambda \|\theta\|_2^2$$

  - Dropout
    - ★ Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR 2014
    - ★ Randomly remove units and their connections during training

Single-Layer Network: Example

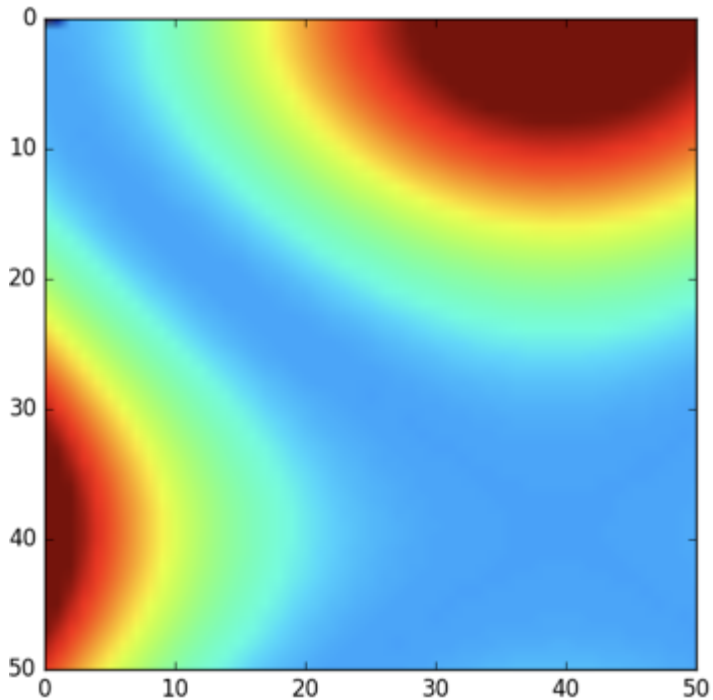
- Interpolating a surface
  - Input: 2D coordinates (x,y)
  - Output: 1D value ($z = 100(y - x^2)^2 + (1 - x)^2$, represented as color)



3 hidden units

Single-Layer Network: Example

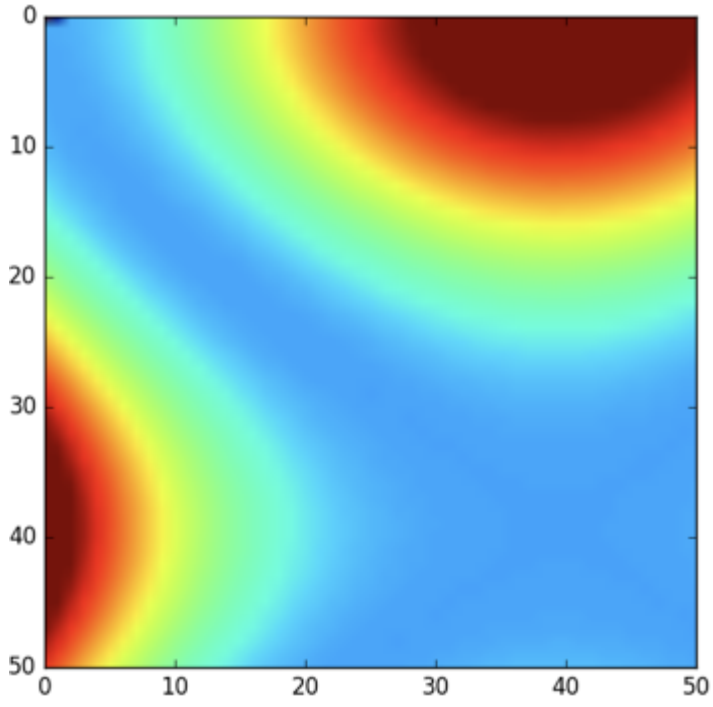
- Interpolating a surface
  - Input: 2D coordinates (x,y)
  - Output: 1D value $(z = 100(y - x^2)^2 + (1 - x)^2$, represented as color)

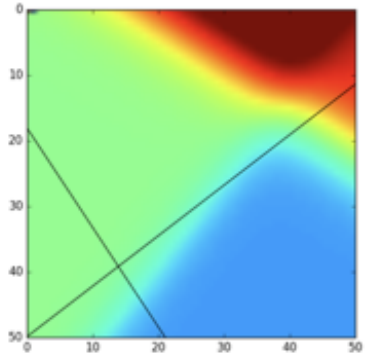


4 hidden units

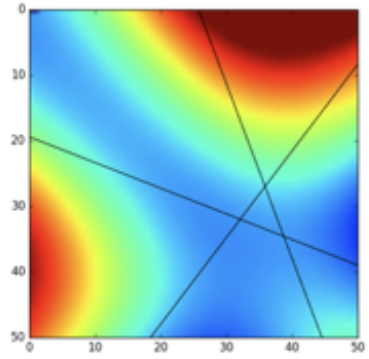
Single-Layer Network: Example

- Interpolating a surface
  - Input: 2D coordinates (x,y)
  - Output: 1D value ($z = 100(y - x^2)^2 + (1 - x)^2$, represented as color)

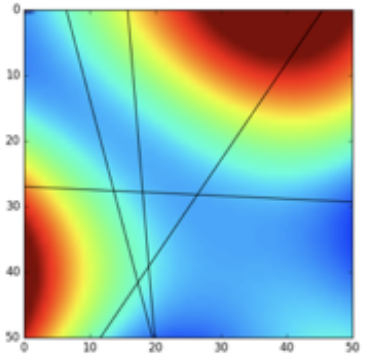




2 nodes -> loss 3.02e-01



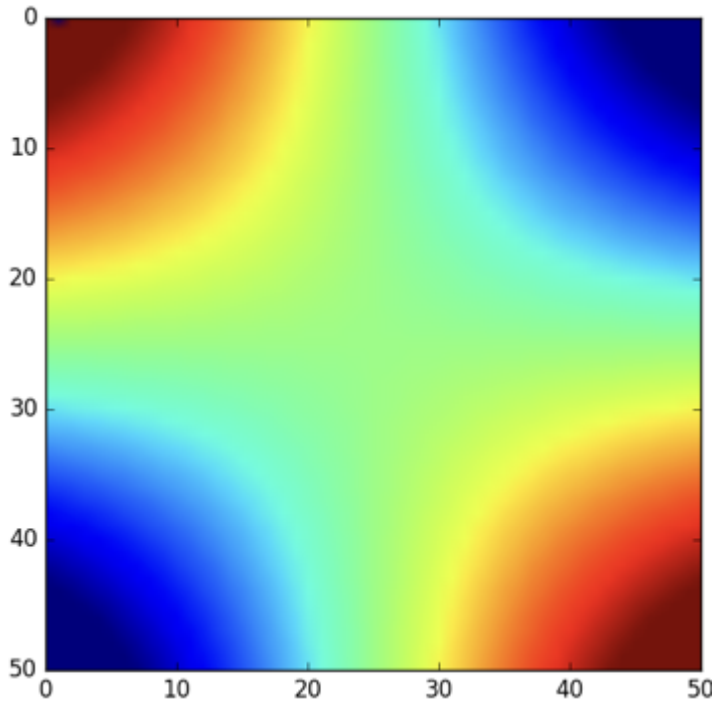
3 nodes -> loss 2.08e-02

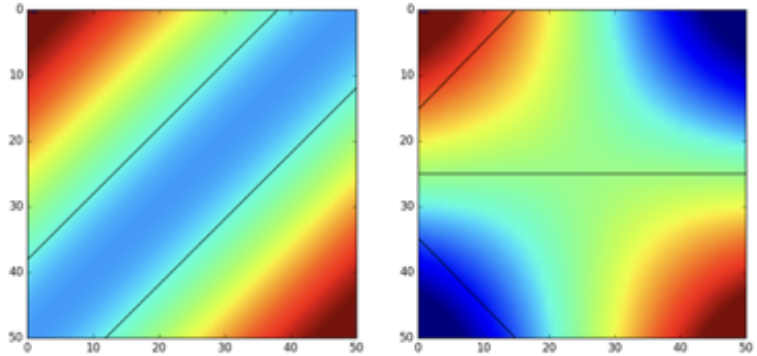


4 nodes -> loss 8.27e-03

Single-Layer Network: Example

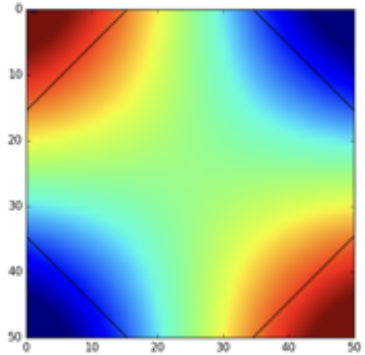
- Interpolating a surface
  - Input: 2D coordinates (x,y)
  - Output: 1D value ($z = \sin(x)\sin(y)$, represented as color)





2 nodes -> loss 2.61e-01

3 nodes -> loss 2.51e-04



4 nodes -> loss 3.07e-07

MLP Demos

- http://playground.tensorflow.org/
- http://scs.ryerson.ca/ aharley/vis/

MLP Tutorial

- Let us try to replicate the last example with pytorch

MLP: Exercise

- Music genre recognition

Image-based Recognition

- Treating images as big vectors seems counter-intuitive
  - An image of reasonable size yields a huge vector
  - Small image regions have similar properties
  - A translation should not affect the results

- Can we reduce the number of parameters and improve robustness?

1D Convolutions

2D Convolutions

Convolutional Layer

- Fewer parameters (the same parameters are shared across different spatial locations)
- Invariance to translation

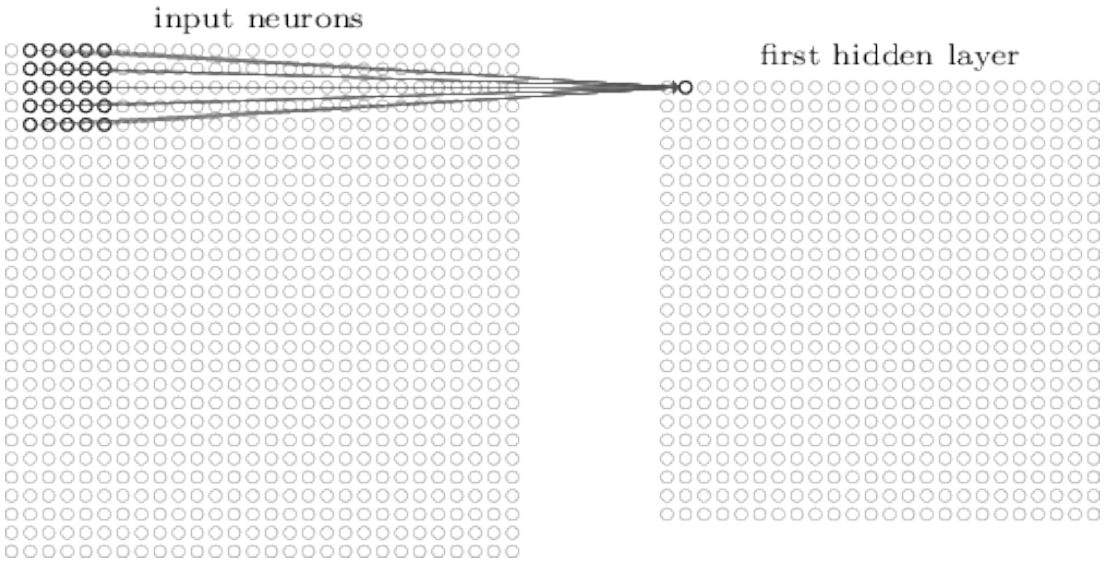


Figure from M. Nielsen's online book
http://neuralnetworksanddeeplearning.com

Convolutional Layer

- Formally, for a $K \times K$ filter, the output at a location (j, k) is computed as

$$\mathbf{z}_{j,k} = f\left(\sum_{l=1}^{K}\sum_{m=1}^{K} \mathbf{w}_{l,m}\mathbf{x}_{j+l,k+m} + b\right),$$

where $\mathbf{x}$ are the input neurons and $f(\cdot)$ is the activation function

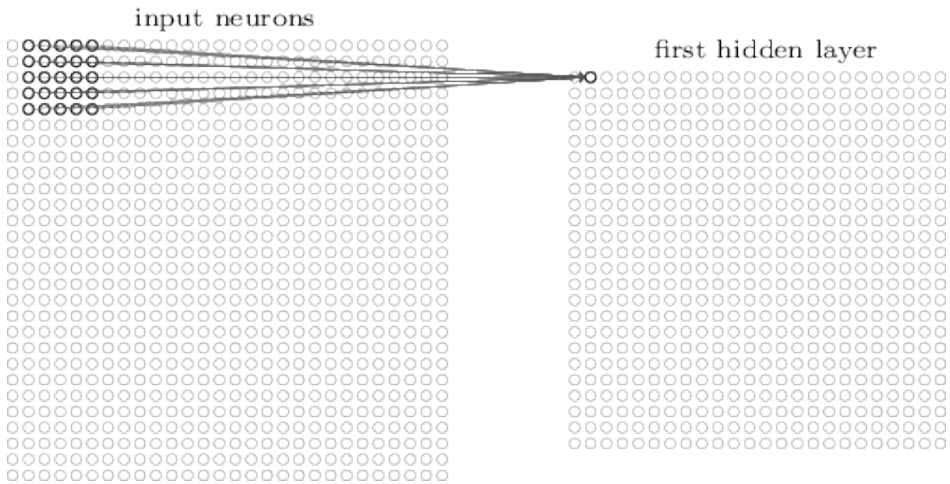


Figure from M. Nielsen's online book
http://neuralnetworksanddeeplearning.com

Convolutional Layer

- We can then use multiple filters to create multiple channels (3 in this example)

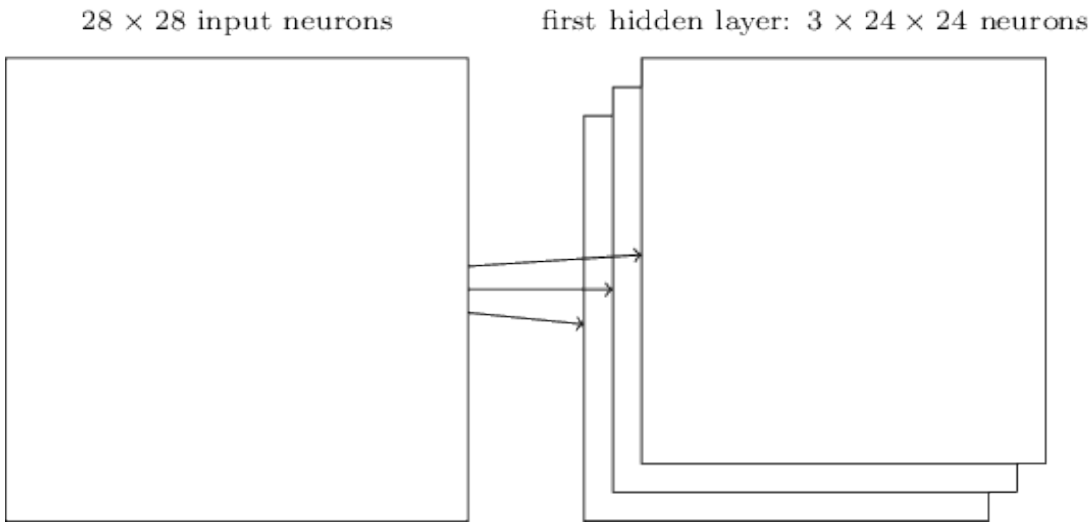


Figure from M. Nielsen's online book
http://neuralnetworksanddeeplearning.com

Pooling Layer

- From local information to a more global representation
- Different pooling strategies, e.g.,
  - max pooling
  - average pooling

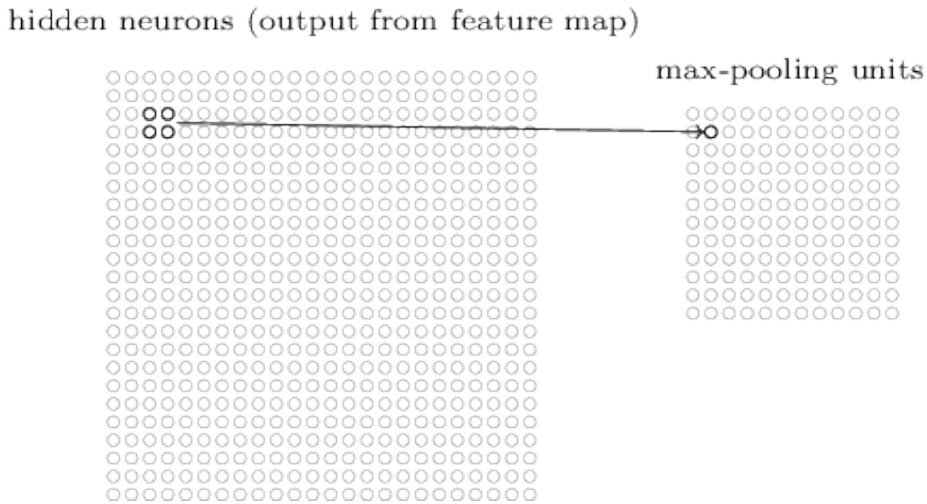


Figure from M. Nielsen's online book
http://neuralnetworksanddeeplearning.com

Convolutional Block

- Altogether...

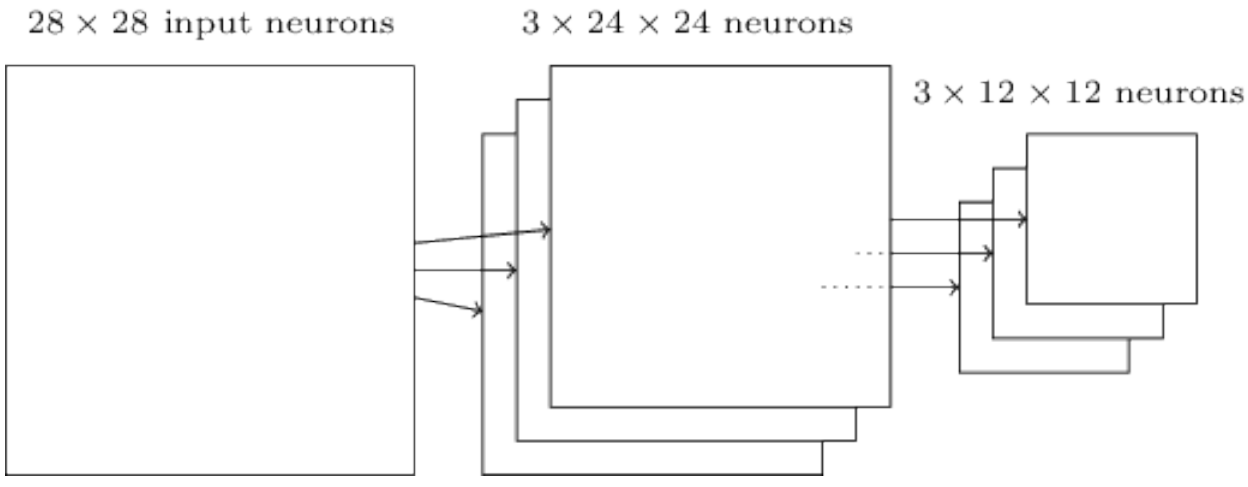
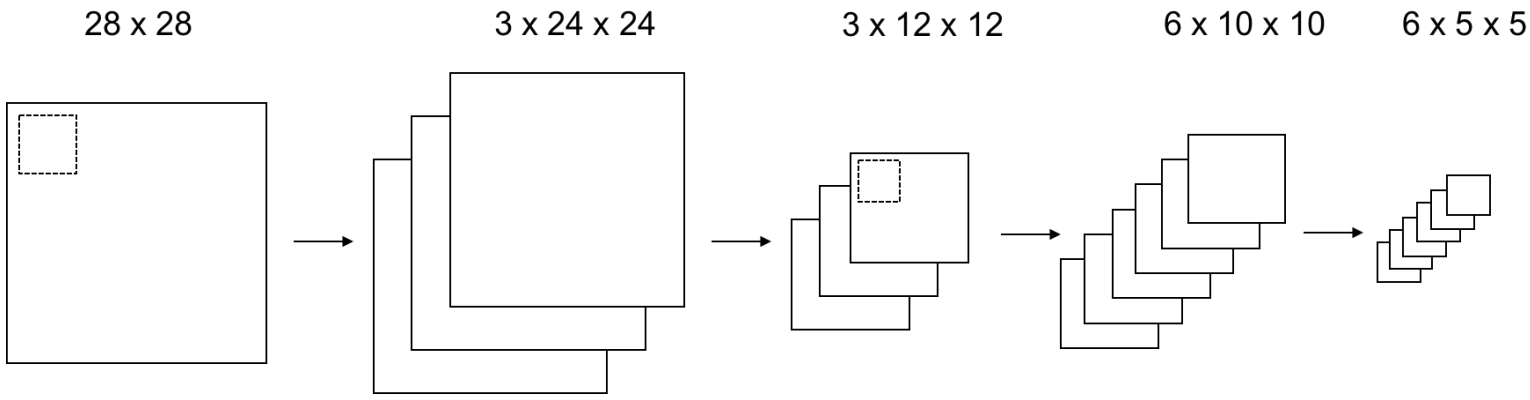


Figure from M. Nielsen's online book
http://neuralnetworksanddeeplearning.com

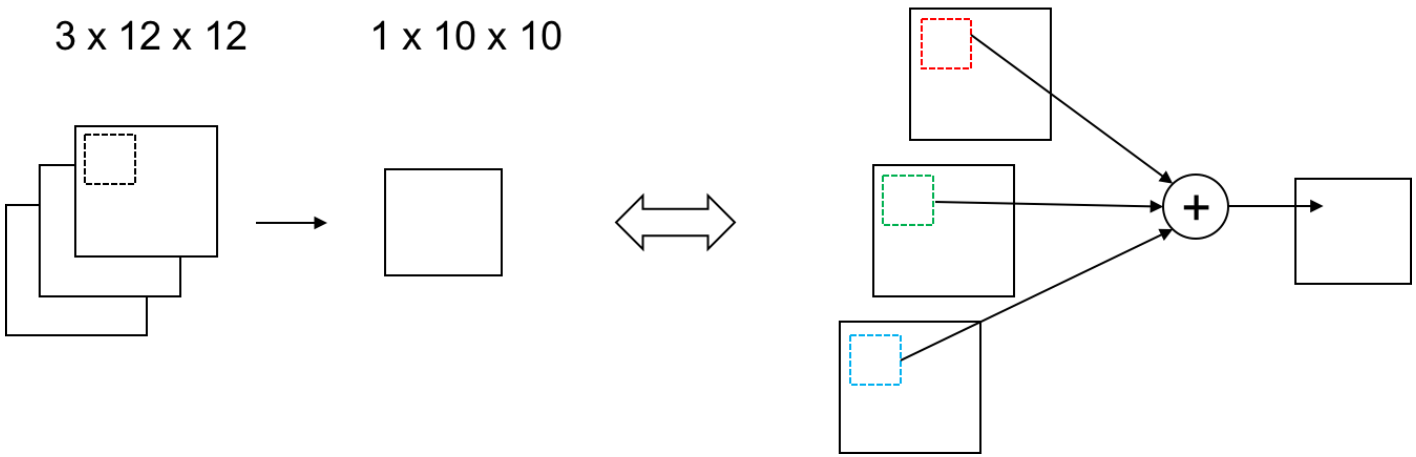
Convolutional Neural Networks

- One can then stack multiple such convolutional blocks
  - The number of channels can vary for different layers (here, 3 at first and then 6)
  - The size of the filters can vary for different layers (here, 5×5 at first and then 3×3)



Convolutional Neural Networks

- Note that with multiple input channels, the convolutions are in fact done in 3D



Convolutional Neural Networks

- One can also append one or more fully-connected (perceptron-like) layers
  - This requires vectorizing the output of the convolutional block

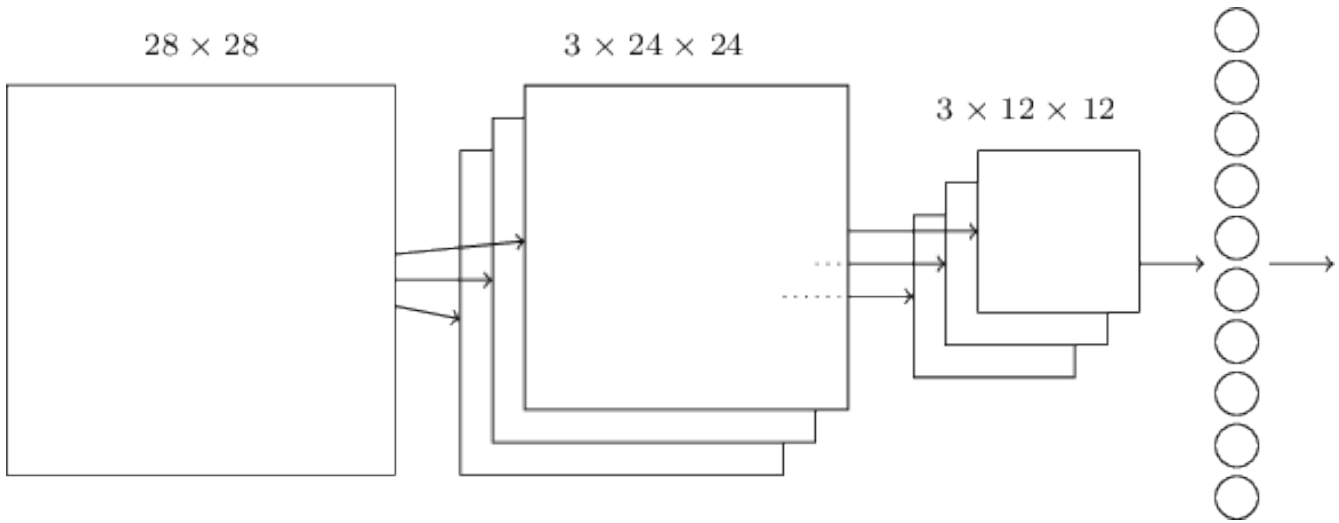
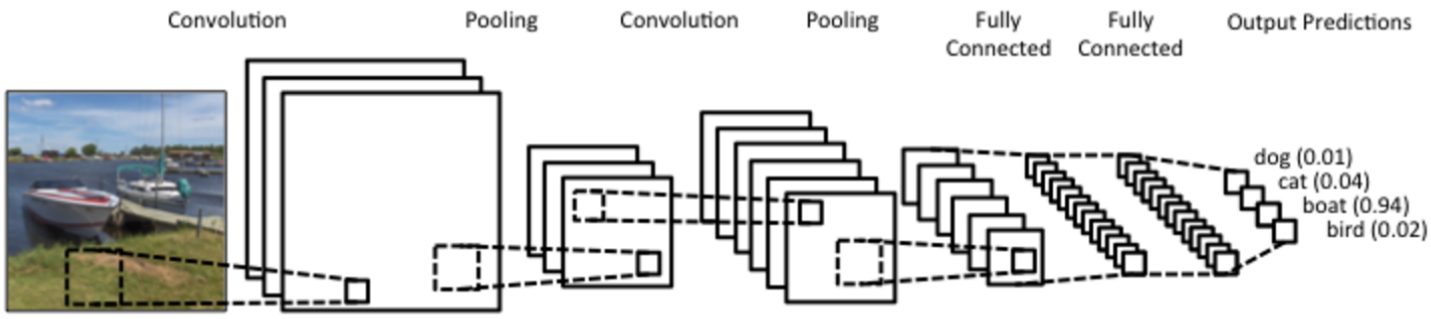


Figure from M. Nielsen's online book
http://neuralnetworksanddeeplearning.com

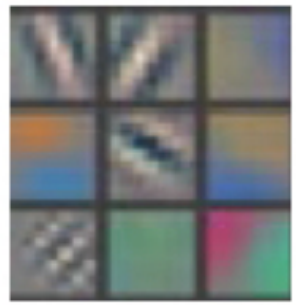
Convolutional Neural Networks: Interpretation

- A CNN can be thought of as learning the representation and the classifier (regressor)



Convolutional Neural Networks: Interpretation

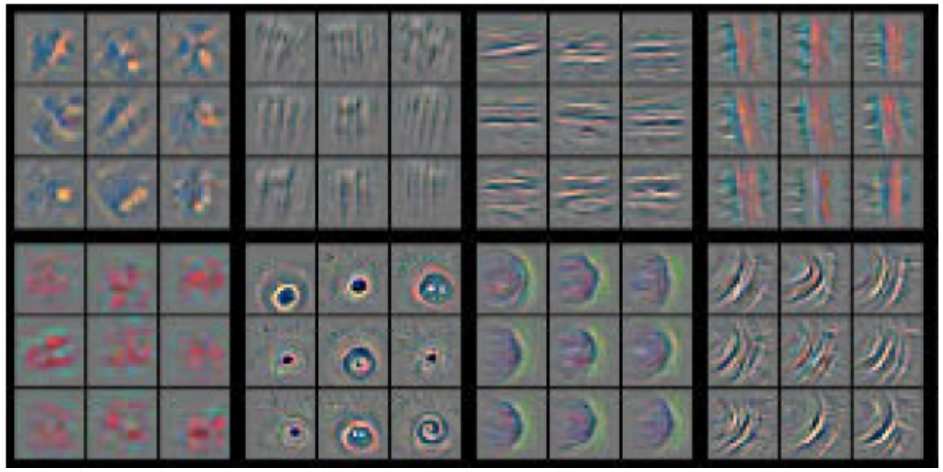
- As a matter of fact, visualizing the features of different layers...
  - From Zeiler & Fergus, ECCV 2014



Layer 1

Convolutional Neural Networks: Interpretation

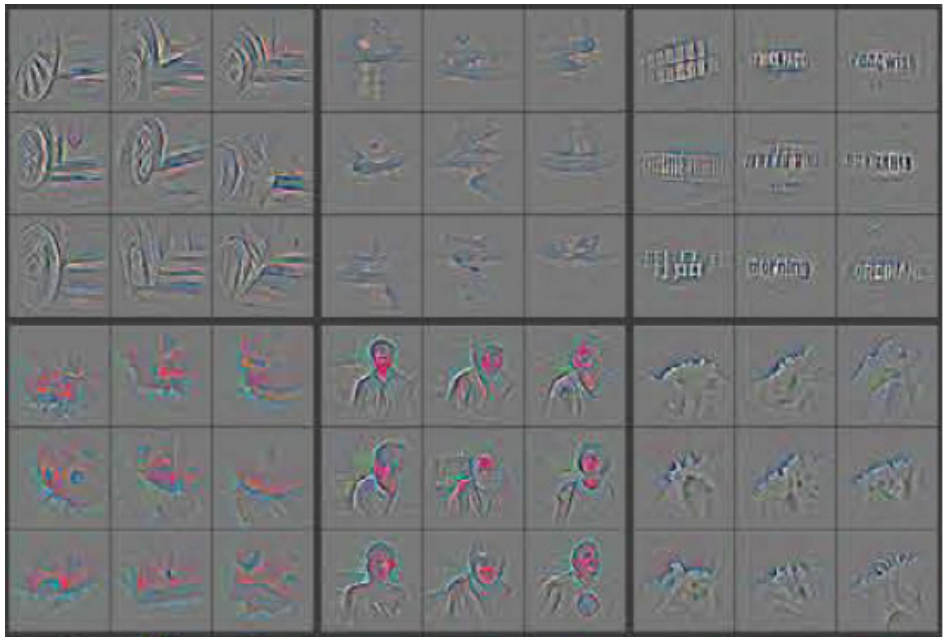
- As a matter of fact, visualizing the features of different layers...
  - From Zeiler & Fergus, ECCV 2014



Layer 2

Convolutional Neural Networks: Interpretation

- As a matter of fact, visualizing the features of different layers...
  - From Zeiler & Fergus, ECCV 2014



Layer 3

CNN Demo

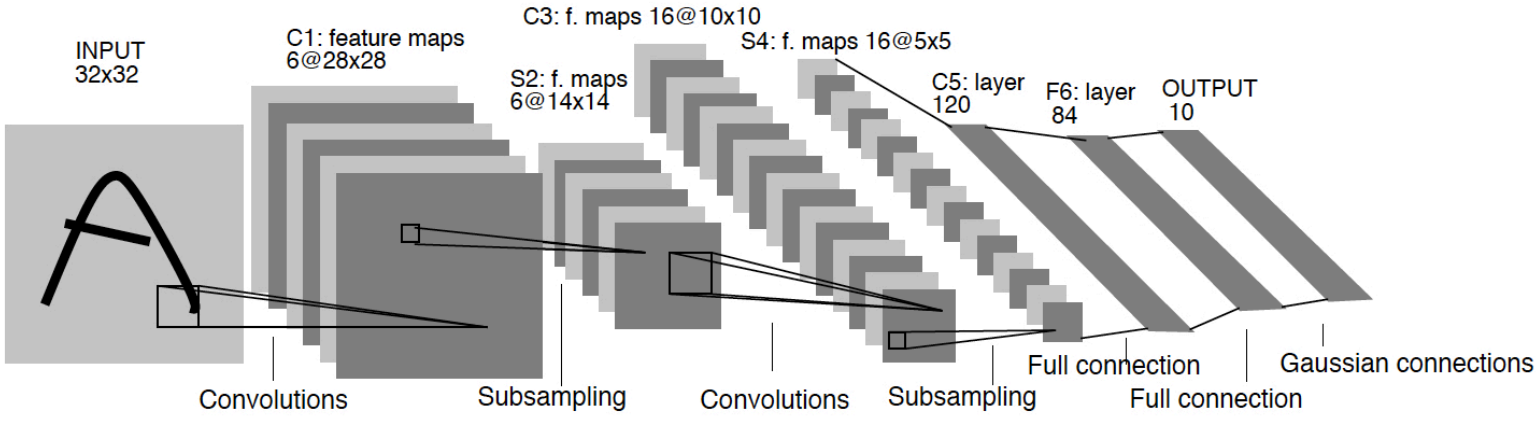
- http://scs.ryerson.ca/ aharley/vis/

CNN Tutorial

- Let us replicate the previous example with pytorch

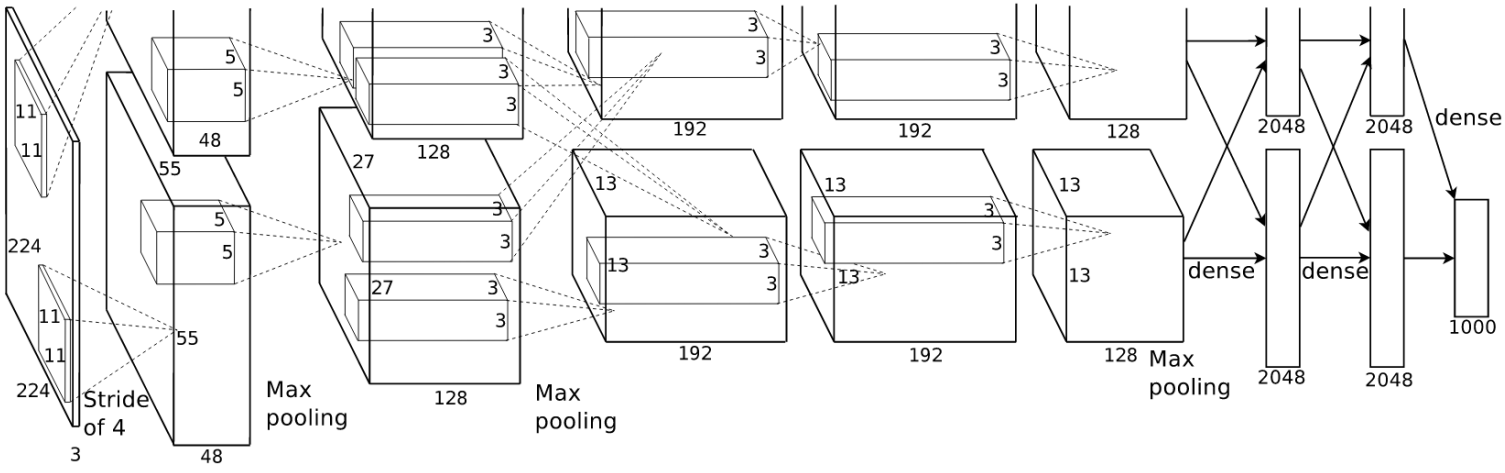
Standard Architectures

- LeNet-5
  - LeCun et al., Gradient-Based Learning Applied to Document Recognition, 1998



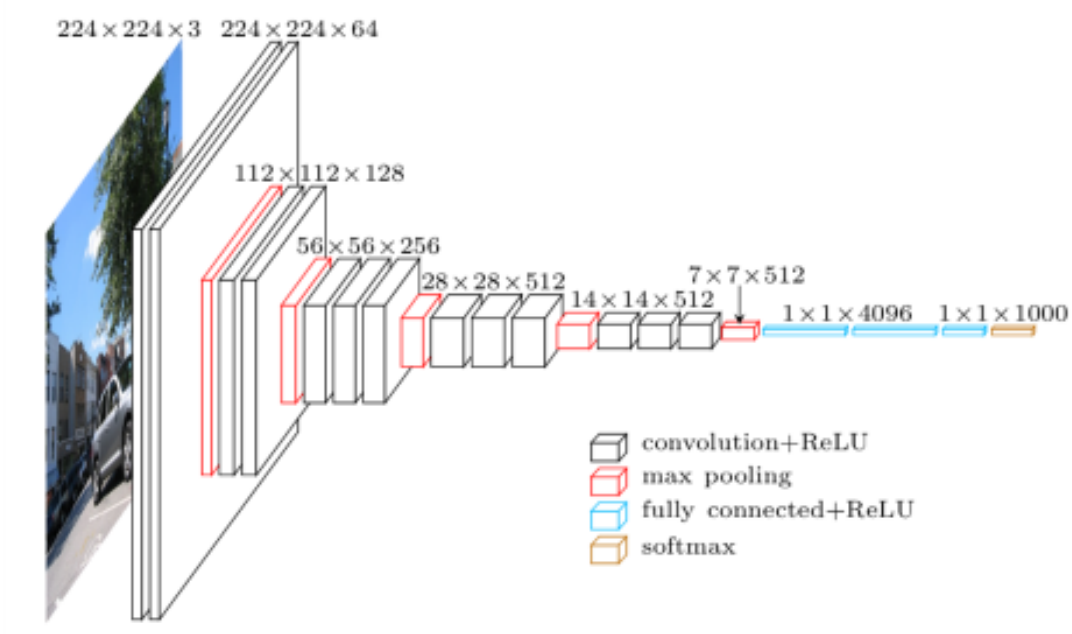
Standard Architectures

- AlexNet
  - Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012



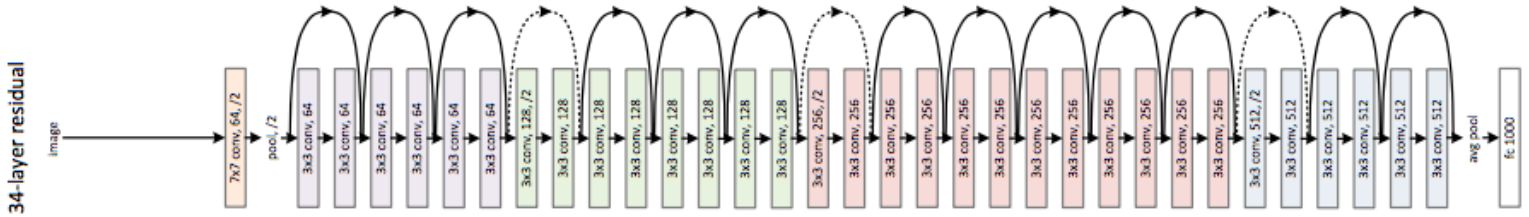
Standard Architectures

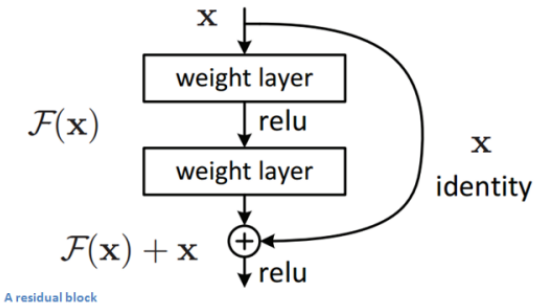
- VGG16
  - Simonyan & Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014



Standard Architectures

- ResNet-34
  - He et al., Deep Residual Learning for Image Recognition, CVPR 2016

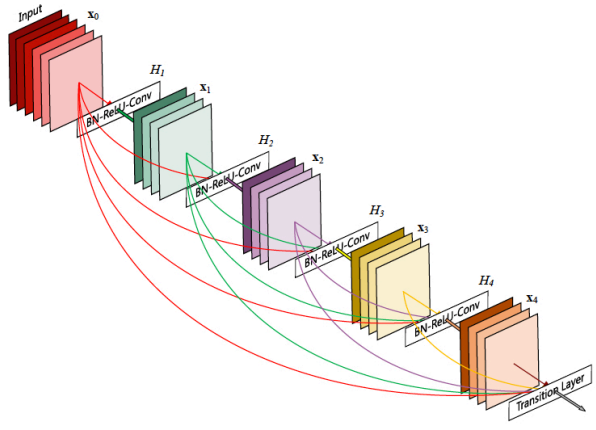


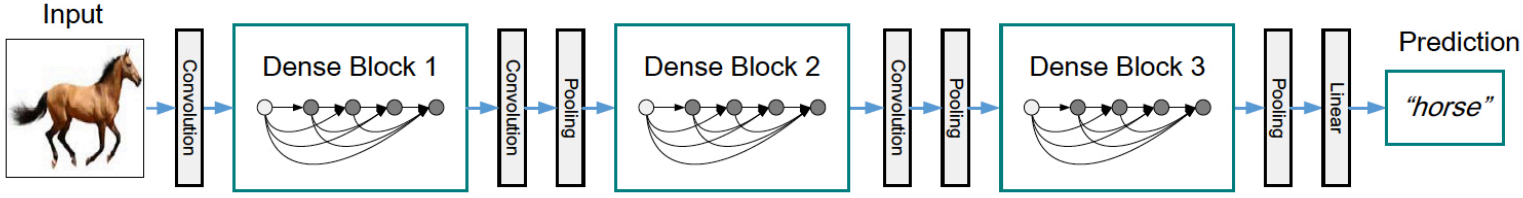


A residual block

Standard Architectures

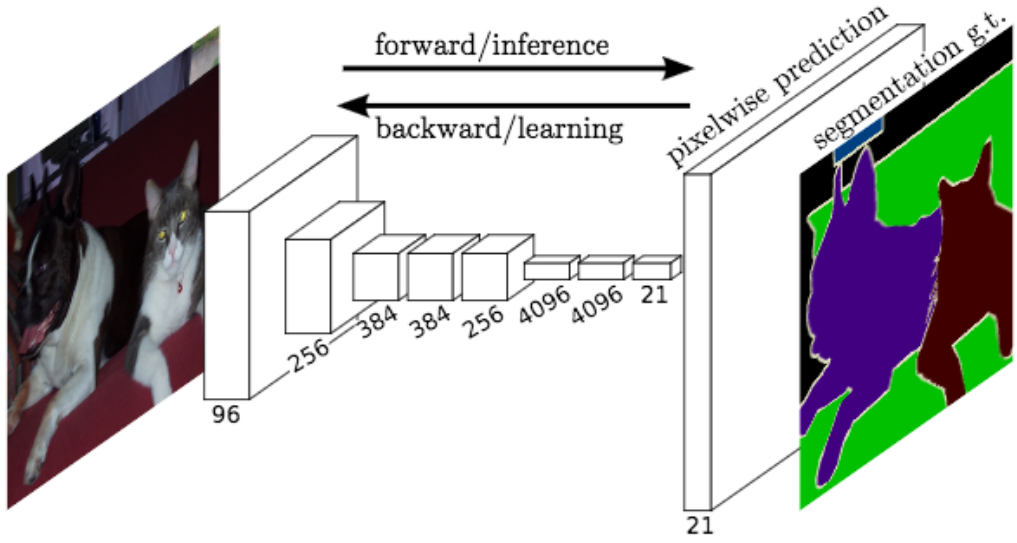
- DenseNet
  - Huang et al., Densely Connected Convolutional Networks, CVPR 2017 (best paper award)





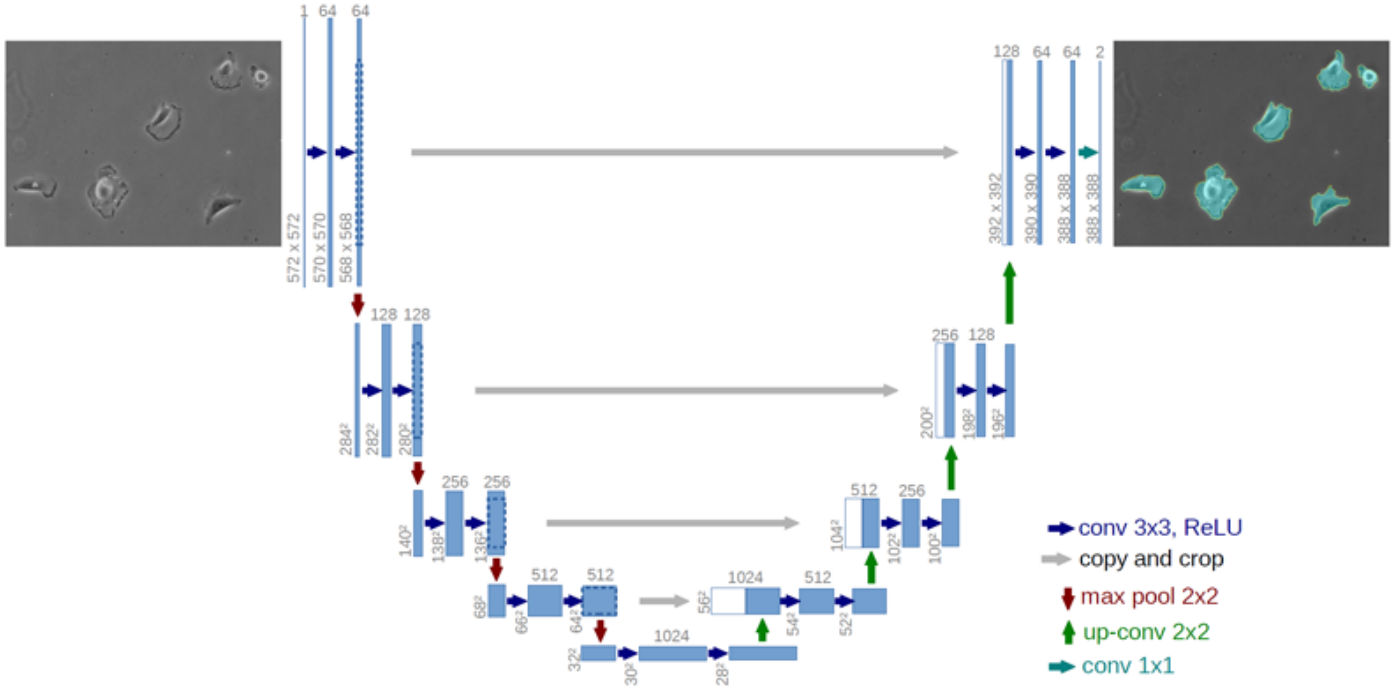
Different Models for Different Tasks

- Fully convolutional models
  - Long et al., Fully Convolutional Networks for Semantic Segmentation, CVPR 2015



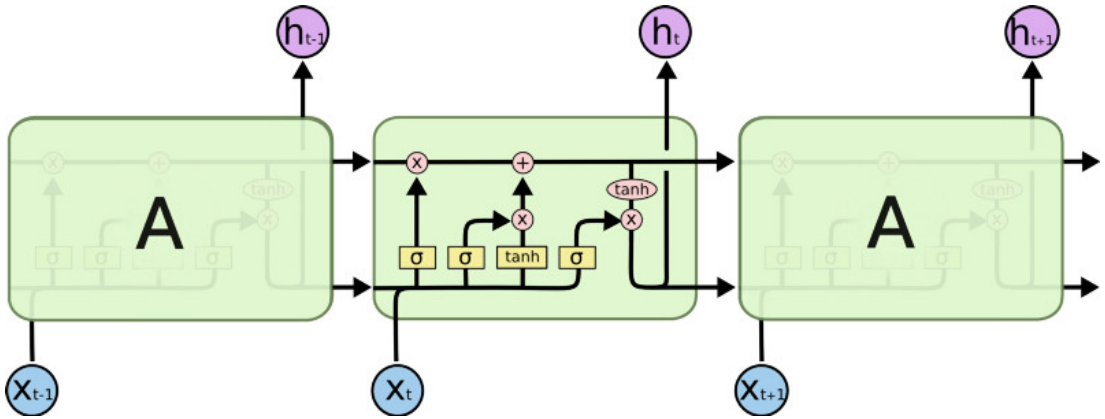
Different Models for Different Tasks

- Fully convolutional models
  - Ronneberger et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI 2015



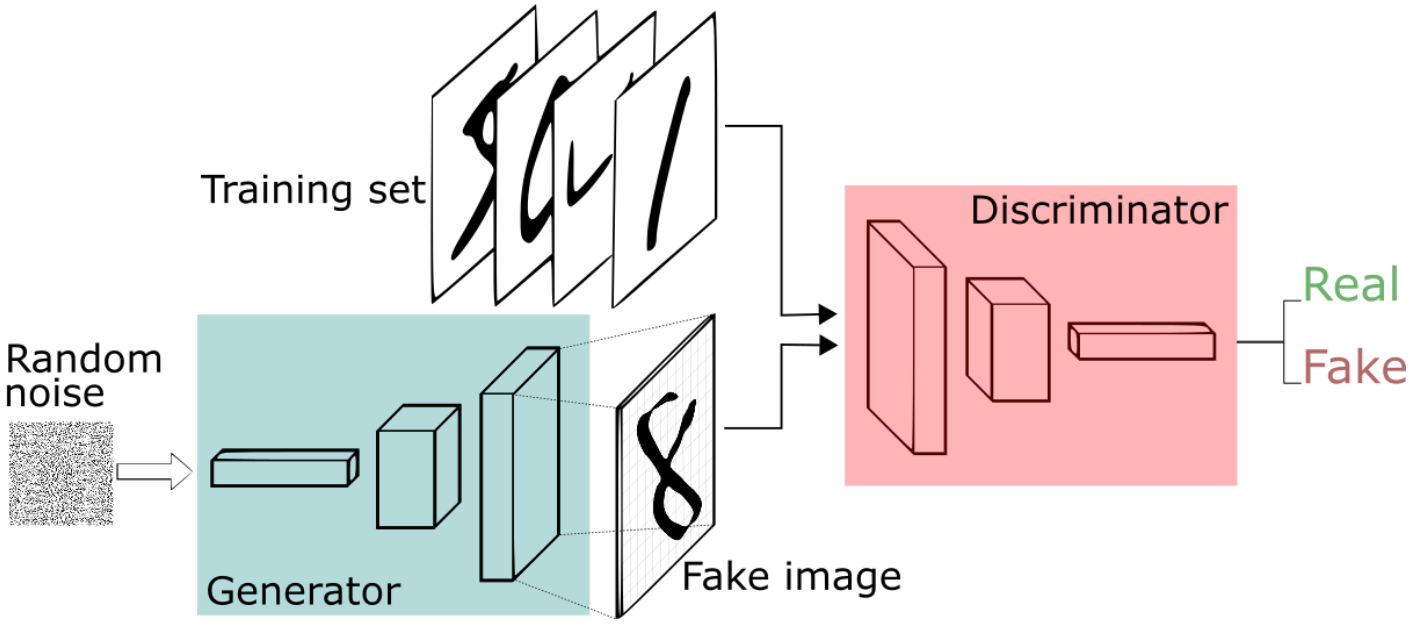
Different Models for Different Tasks

- Recurrent neural networks (RNNs)
  - Hochreiter & Schmidhuber, Long Short-term Memory, Neural Computations 1997
  - Often used to model temporal signals



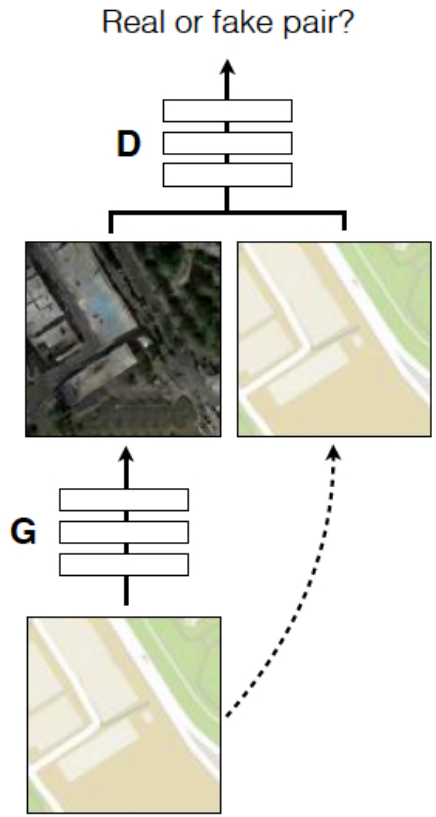
Different Models for Different Tasks

- Generative Adversarial Networks (GANs)
  - Goodfellow, Generative Adversarial Nets, NIPS 2014



Different Models for Different Tasks

- Conditional GANs
  - Isola et al., Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017
  - https://affinelayer.com/pixsrv/



The Key to Success

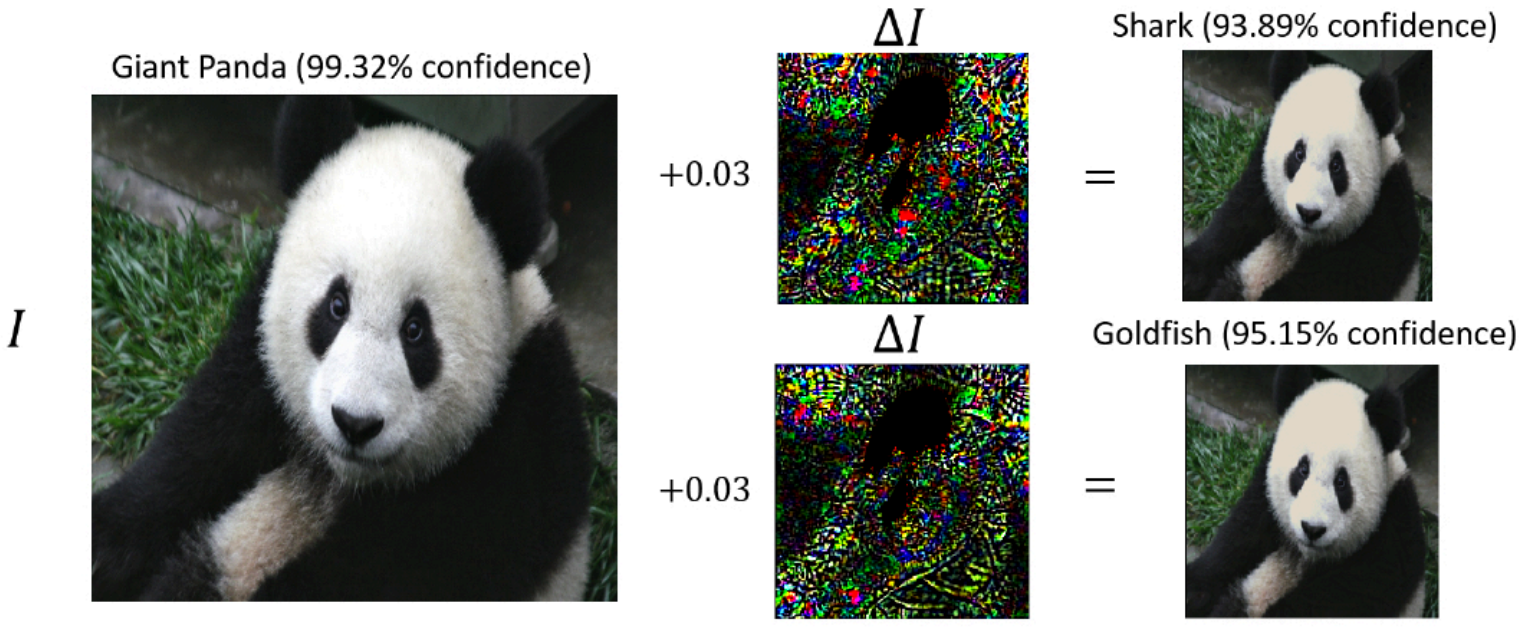
- Big Data: ImageNet
  - Russakovsky et al., 2015
- GPUs

Tricks of the Trade

- Pre-training
  - People typically start with a standard architecture trained on ImageNet
- Data augmentation
  - E.g., for images, flip, rotate, affine transform,...
- Batch normalization
  - Ioffe & Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015
  - Normalize the activations of the mini-batches during training

Despite the Success

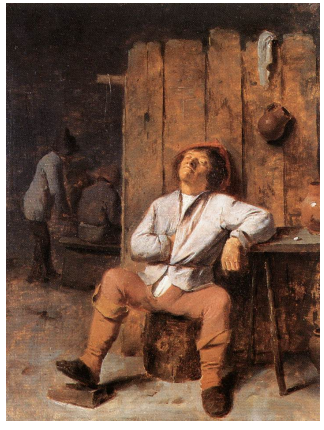
- Figure from Li & Li, Adversarial Examples Detection in Deep Networks with Convolutional Filter Statistics, 2016

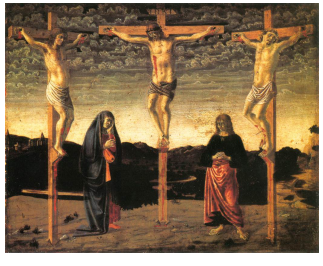


Exercise: Painting Style Recognition

- Evaluate deep architectures on the wikiart data





Next Time

- **Dimensionality reduction (for visualization)**