Deep Learning meets Computer Vision

Open Lecture at Mid Sweden University

Faisal Qureshi

Faculty of Science, Ontario Tech University

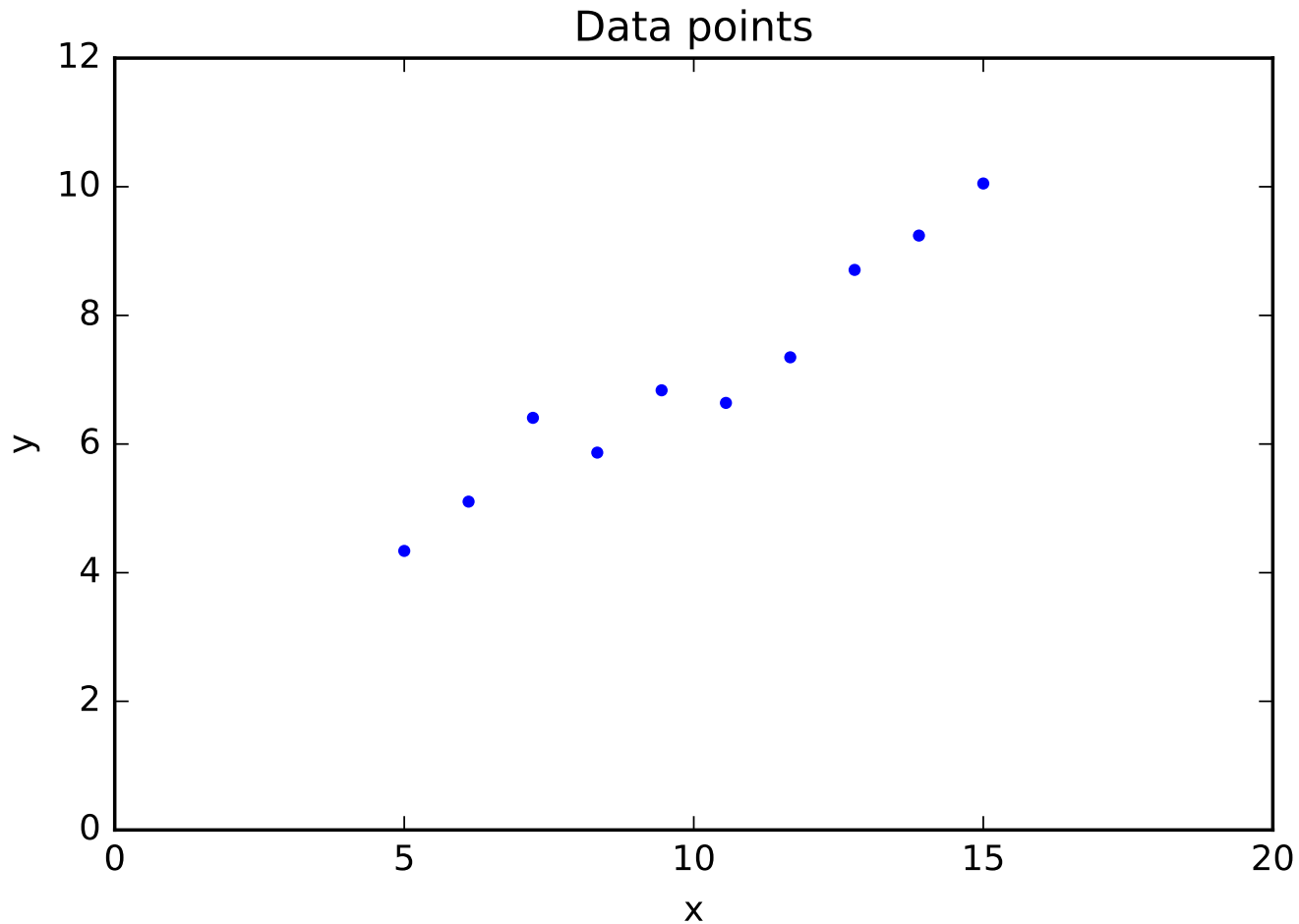
2 December 2019

Talk abstract

In recent years computer vision has seen a seismic shift from classical statistical methods to deep learning. In this talk I will introduce deep learning and showcase some of its successes at many computer vision tasks. I will start with basic line fitting and use our understanding of it to construct more complex Artificial Neural Networks (ANNs) models. I will then discuss convolutional neural networks and see how these have rendered hand-crafted features of old, e.g., SIFT, redundant. I will conclude this talk with some recent applications of deep learning to long-standing computer vision problems.

Regression

Consider data points $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(N)}, y^{(N)})$. Our goal is to learn a function $f(x)$ that returns (predict) the value y given an x.



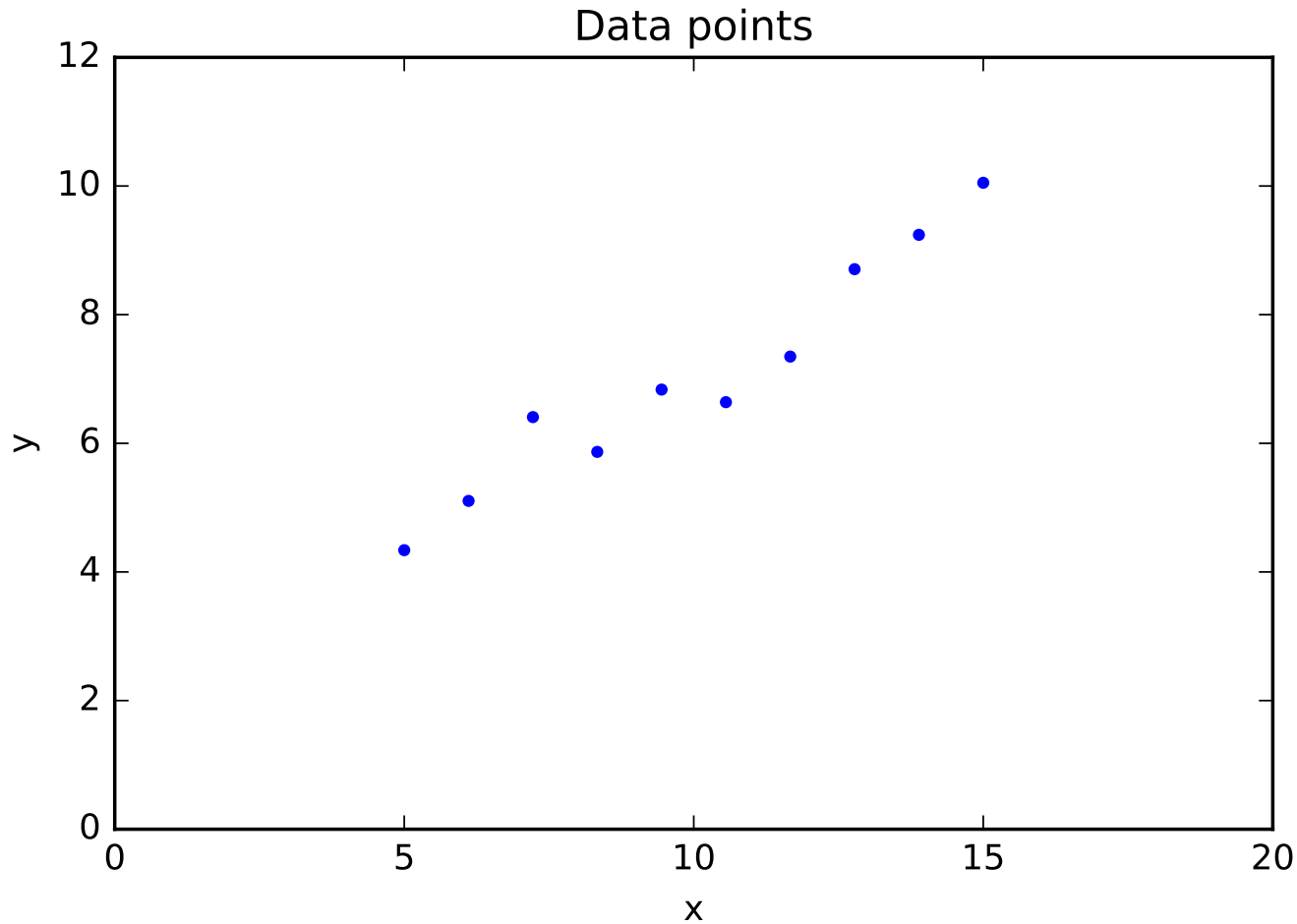
Regression

Given data $D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(N)}, y^{(N)})\}$, learn function $y = f(x)$.

- x is the input feature. In the example above, x is 1-dimensional, however, in practice x is often an M-dimensional vector.
- y is the target output. We assume that y is continuous. y is 1-dimensional (why?)

Linear regression

We assume that a linear model of the form $y = f(x) = \theta_0 + \theta_1 x$ best describe our data.



How do we determine the degree of "fit" of our model?

Least squares error

Loss-cost-objective function measures the degree of fit of a model to a given data.

A simple loss function is to sum the squared differences between the actual values $y^{(i)}$ and the predicted values $f(x^{(i)})$. This is called the *least squares error*.

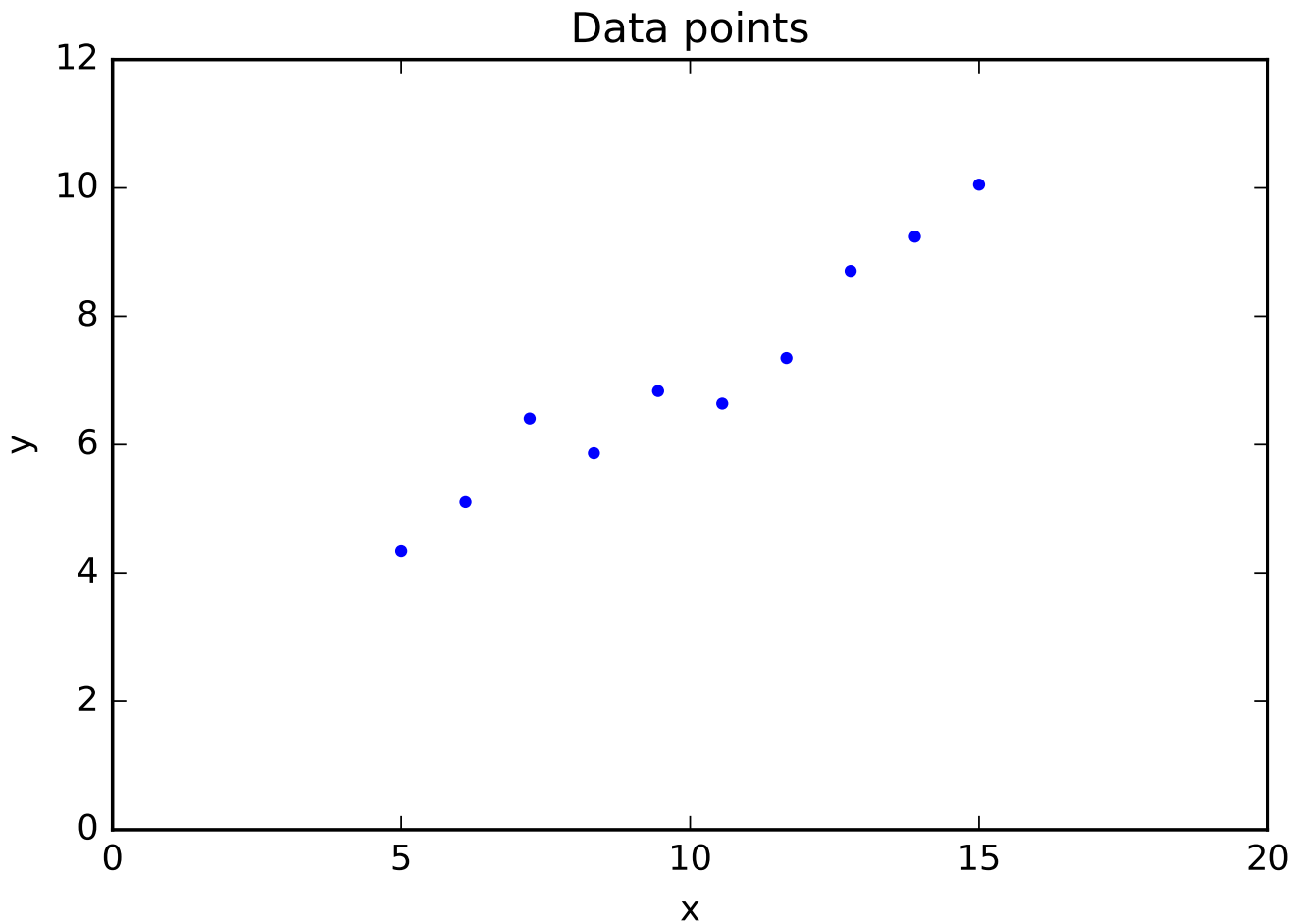
$$C(\theta_0, \theta_1) = \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$

Our task is to find values for θ_0 and θ_1 (model parameters) to minimize the cost.

We often refer to the predicted value as $\hat{y}$. Specifically, $\hat{y}^{(i)} = f(x^{(i)})$.

Least squares error

$$C(\theta_0, \theta_1) = \sum_{i=1}^{N} \left(y^{(i)} - f(x^{(i)}) \right)^2$$



Linear least squares in higher dimensions

Input feature: $\mathbf{x}^{(i)} = \left(1, x_1^{(i)}, x_2^{(i)}, \cdots, x_M^{(i)}\right)^T$.

For this discussion, we assume $x_0^{(i)} = 1$ (just to simplify mathematical notation).

Target feature: $y^{(i)}$

Parameters: $\theta = (\theta_0, \theta_1, \cdots, \theta_M)^T \in \mathbb{R}^{(M+1)}$

Model: $f(\mathbf{x}) = \mathbf{x}^T\theta$

Linear least squares in higher dimensions

Loss:

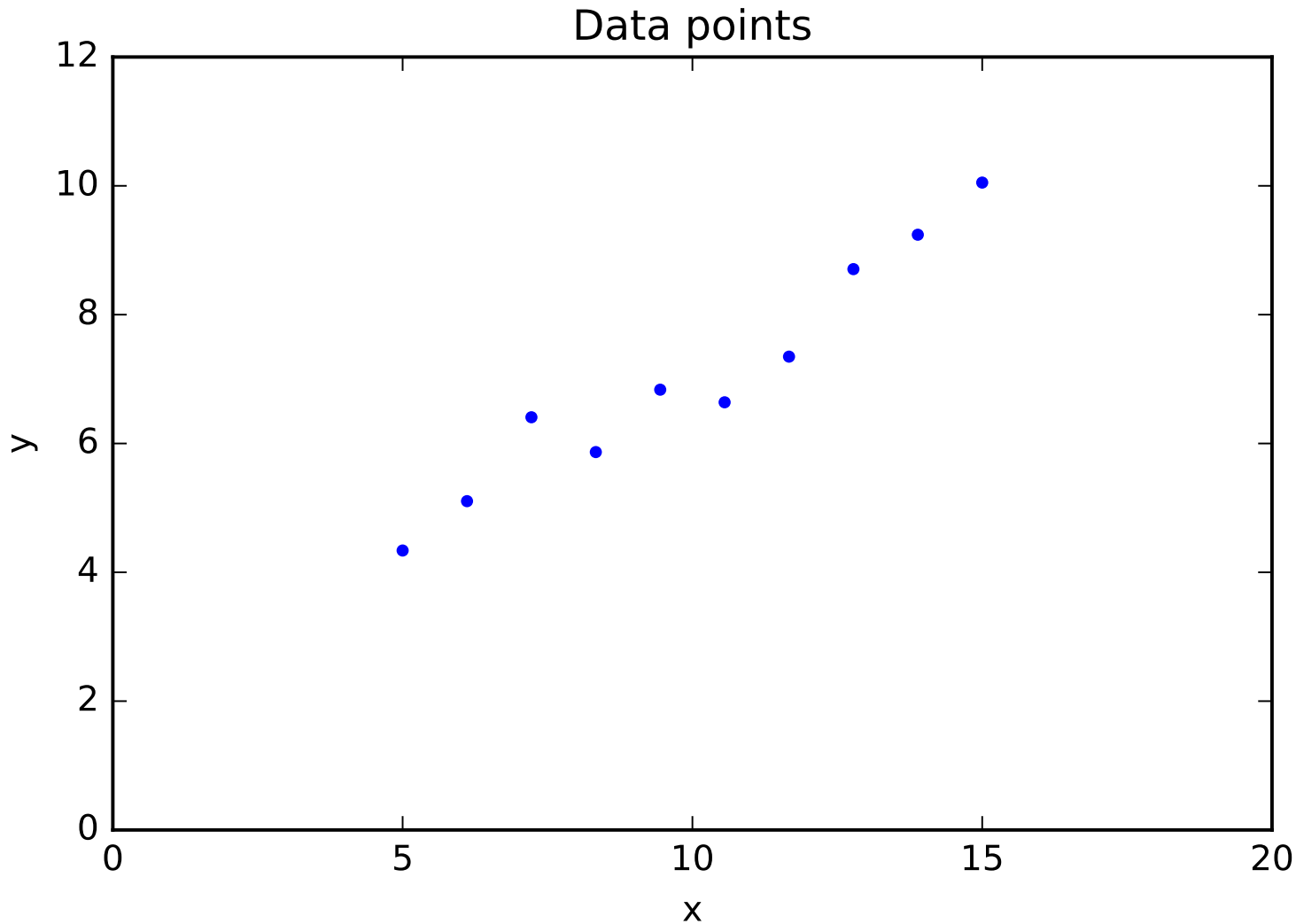
$$C(\theta) = (\mathbf{Y} - \mathbf{X}\theta)^T(\mathbf{Y} - \mathbf{X}\theta)$$

$$\mathbf{X} = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ - & \mathbf{x}_2^T & - \\ & \vdots & \\ - & \mathbf{x}_N^T & - \end{bmatrix} \in \mathbb{R}^{N\times(M+1)} \qquad \mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^{N\times 1}$$

$\mathbf{X}$ is referred to as the *design matrix*.

A Probabilistic View of Linear regression

Consider data points $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(N)}, y^{(N)})$. Our goal is to learn a function $f(x)$ that returns (predict) the value y given an x.



The likelihood for linear regression

Let's assume that targets $y(i)$ are corrupted by Gaussian noise with 0 mean and σ^2 variance

$$\begin{aligned} y^{(i)} &= \theta^T x^{(i)} + \mathcal{N}(0, \sigma^2) \\ &= \mathcal{N}\left(\theta^T x^{(i)}, \sigma^2\right) \end{aligned}$$

In higher dimensions, we write:

$$y^{(i)} = \mathcal{N}\left(\theta^T \mathbf{x}^{(i)}, \sigma^2\right)$$

Why assume Gaussian noise?

- Mathematically convenient
- A reasonably accurate assumption in practice
- Central Limit Theorem

The likelihood for linear regression

Under the assumption that each $y^{(i)}$ is i.i.d., we can write the likelihood of $\mathbf{y}$ given data $\mathbf{X}$ as follows:

$$
\begin{aligned}
p(\mathbf{y}|\mathbf{X};\theta,\sigma) &= \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)};\theta,\sigma) \\
&= \prod_{i=1}^{n} \left(2\pi\sigma^2\right)^{-1/2} e^{-\frac{1}{2\sigma^2}\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2} \\
&= \left(2\pi\sigma^2\right)^{-n/2} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2} \\
&= \left(2\pi\sigma^2\right)^{-n/2} e^{-\frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{X}\theta)^T(\mathbf{y}-\mathbf{X}\theta)}
\end{aligned}
$$

Aside: the ";" above indicate that we are following the *frequentist* approach, and we do not treat θ as a random variable. Rather we view θ as having some true value that we are trying to estimate.

Likelihood example

Consider the points: $y_1 = 1$, $y_2 = 0.5$ and $y_3 = 1.5$. The points are drawn from a Gaussian with unknown *mean* θ and $\sigma^2 = 1$.

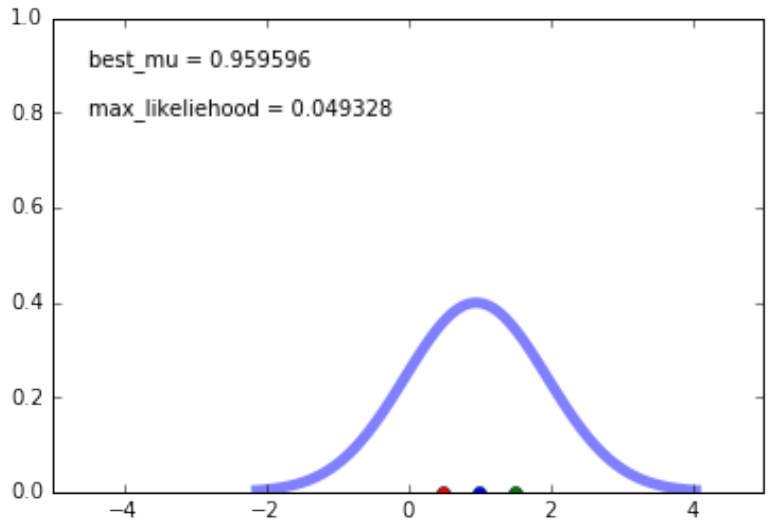
$$y_i \sim \mathcal{N}(\theta, 1)$$

.

Points are independent so

$$P(y_1, y_2, y_3|\theta) = P(y_1|\theta)P(y_2|\theta)P(y_3|\theta)$$

Our goal is to find the Gaussian (i.e., find its mean, since variance is already given) that maximizes the *likelihood* of this data.



From Nando de Freitas

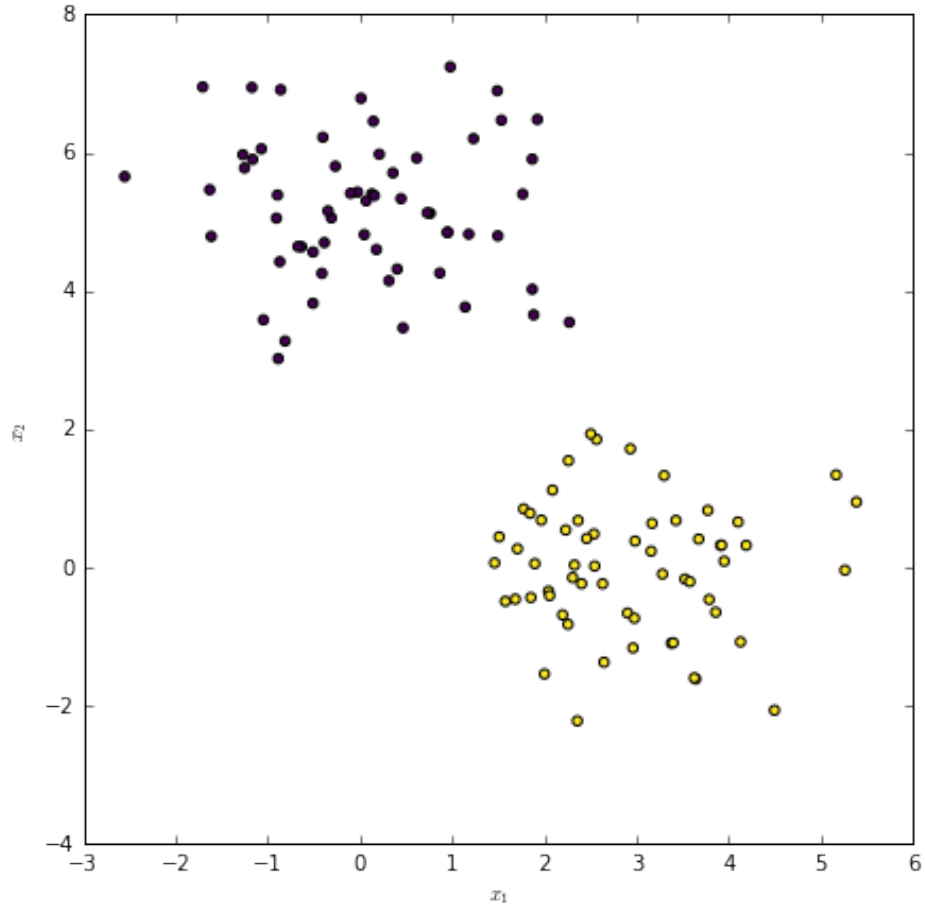
Probability of data given parameters

Loss for linear regression

$$C(\theta) = (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta)$$

Probability of data given parameters is related to the loss for linear regression that we obtained before.

Binary Classification



In binary classification, the target variable y takes on values in $\{0, 1\}$.
Given a point $\mathbf{x} = (x_1, x_2)$, we want to predict whether y is 0 or 1.

Binary classification

The goal of binary classification is to learn $h_\theta(\mathbf{x})$, which can be used to assign a label $y \in \{0, 1\}$ to the input $\mathbf{x}$. Label y takes values in $\{0, 1\}$, so we can use Bernoulli distribution to specify its probability distribution. Specifically

$$\Pr(y = 1) = h_\theta(\mathbf{x})$$
$$\Pr(y = 0) = 1 - h_\theta(\mathbf{x})$$

Or more succinctly

$$\Pr(y) = h_\theta(\mathbf{x})^y \left(1 - h_\theta(\mathbf{x})\right)^{1-y}$$

Likelihood for binary classification

Under the assumption that data is i.i.d.

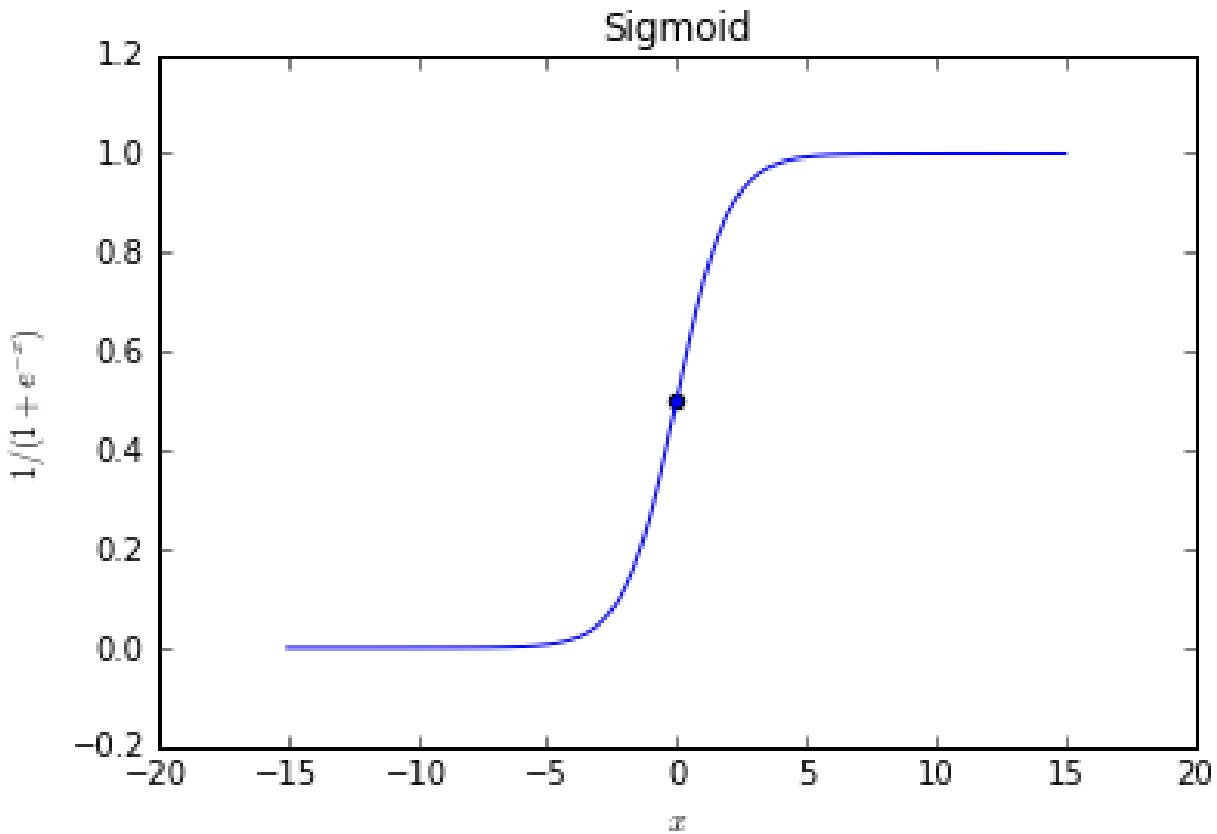
$$\Pr(y|\mathbf{X},\theta) = \prod_{i=1}^{N} h_\theta(\mathbf{x}^{(i)})^{y^{(i)}} \left(1 - h_\theta(\mathbf{x}^{(i)})\right)^{1-y^{(i)}}$$

Lets consider a simple 1D case for binary classification

Sigmoid function

$\mathrm{sigm}(x)$ refers to a *sigmoid* function, also known as the *logistic* or *logit* function.

$$\mathrm{sigm}(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



Lets consider a simple 1D case for binary classification

Logistic regression

$$\Pr(y|x,\theta) = \left[\frac{1}{1+e^{-(\theta_0+\theta_1 x)}}\right]^y \left[1 - \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}\right]^{1-y}$$

$\theta = (\theta_0, \theta_1)$ are model parameters.

θ_0 controls the shift.

θ_1 controls the scale (how steep is the slope of the sigmoid function).

Logistic regression (in higher dimensions)

For logistic regression, we set $h_\theta(\mathbf{x}) = \text{sigm}(\mathbf{x}^T\theta)$. So

$$\Pr(y|\mathbf{X},\theta) = \prod_{i=1}^{N}\left[\frac{1}{1+e^{-\mathbf{x}^{(i)T}\theta}}\right]^{y^{(i)}}\left[1-\frac{1}{1+e^{-\mathbf{x}^{(i)T}\theta}}\right]^{1-y^{(i)}}$$

where

$$\mathbf{x}^T\theta = \theta_0 + \sum_{i=1}^{M}\theta_i\mathbf{x}_i$$

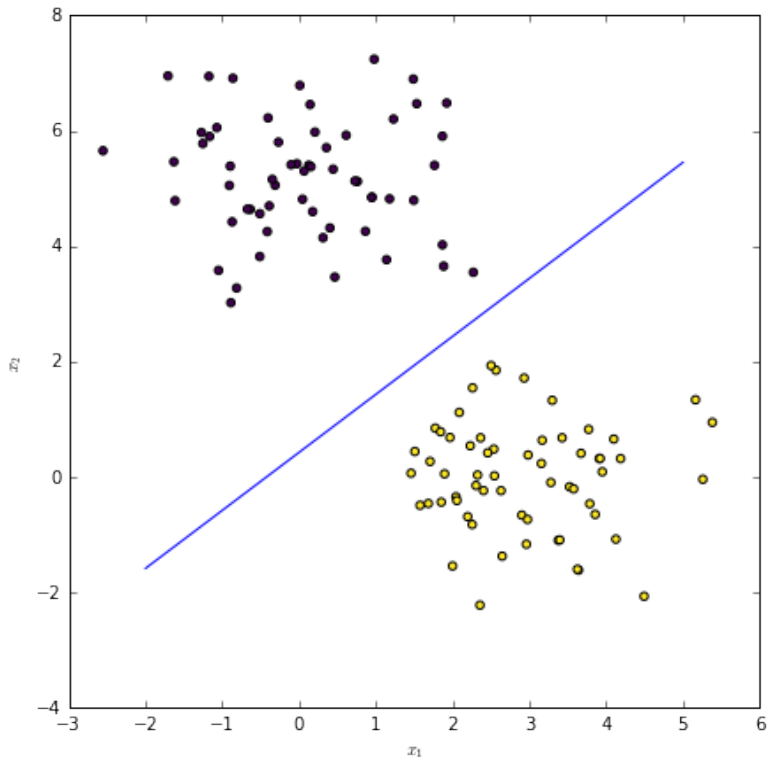
.

Logistic regression for binary classification

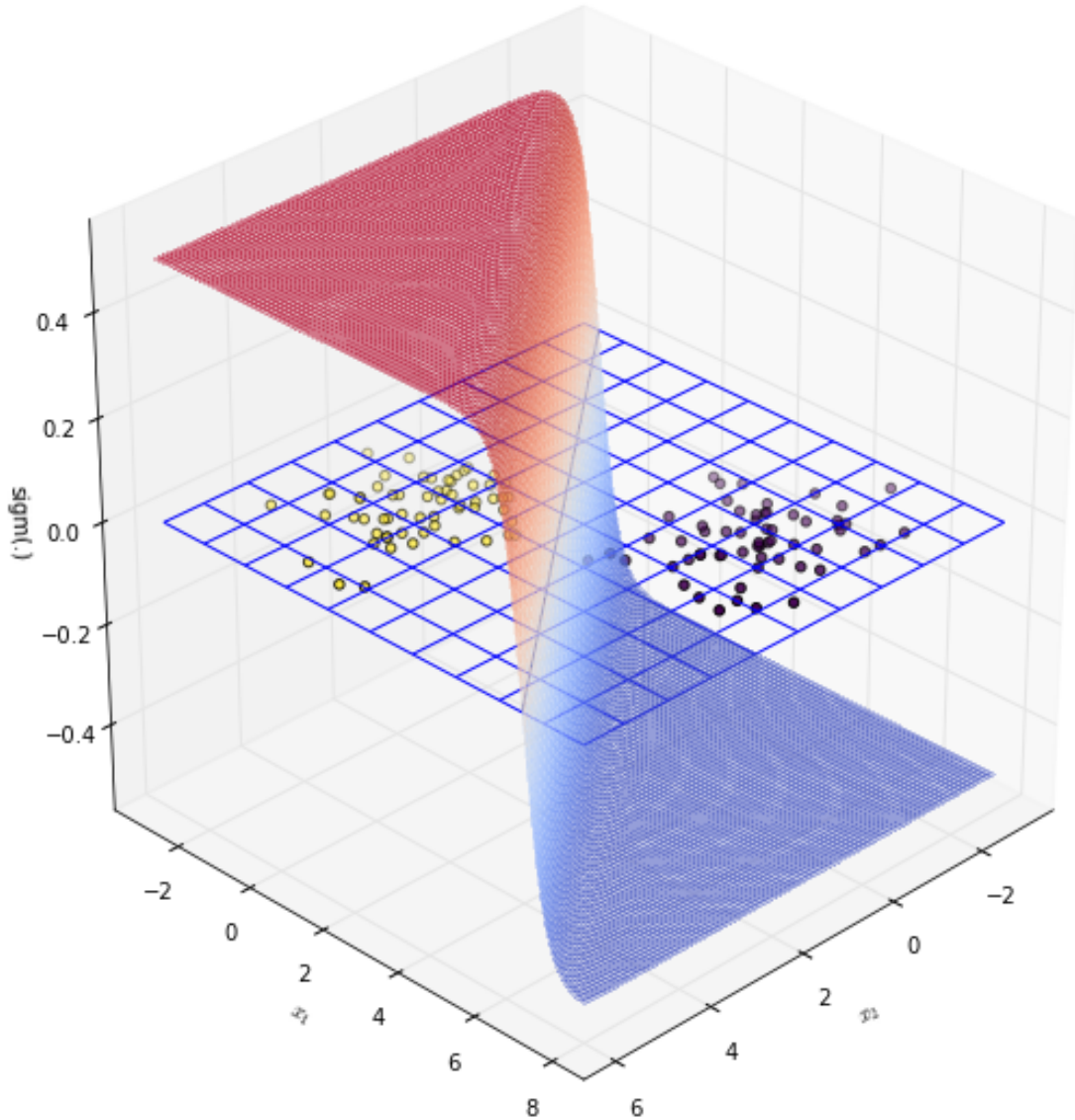
Given a point $\mathbf{x}^{(*)}$, classify using the following rule

$$y^{(*)} = \begin{cases} 1 & \text{if } \Pr(y|\mathbf{x}^{(*)}, \theta) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

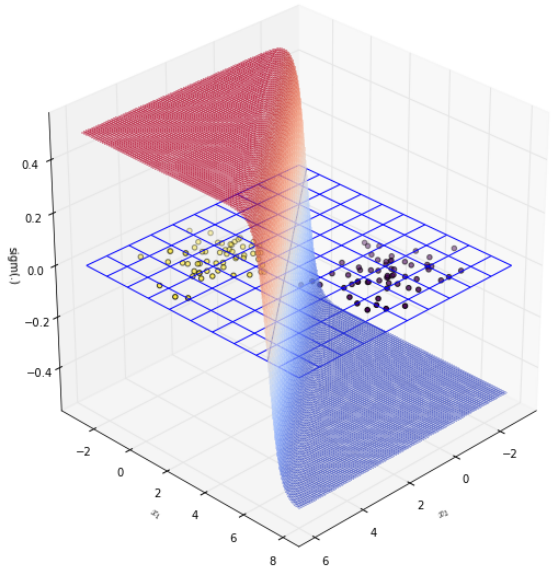
The decision boundary is $\mathbf{x}^T\theta = 0$.
Recall that this is where the sigmoid function is 0.5.

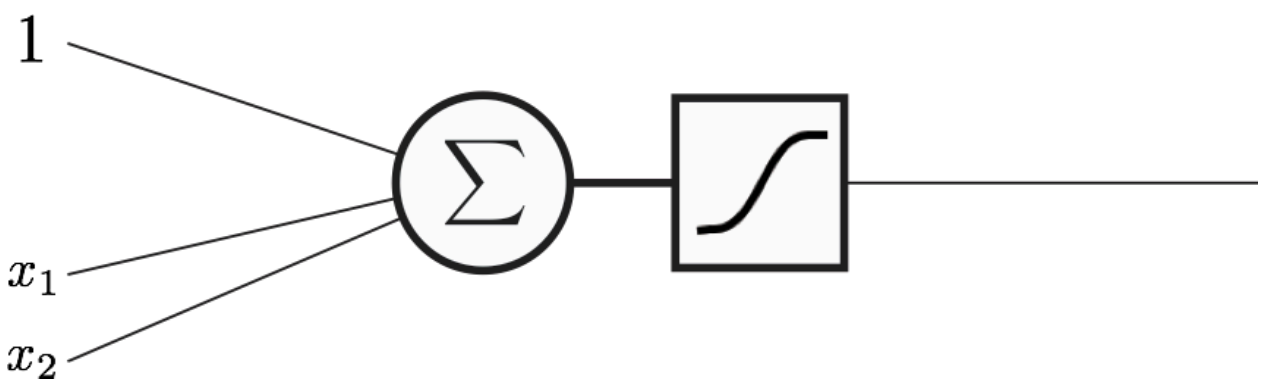


Logistic regression for binary classification

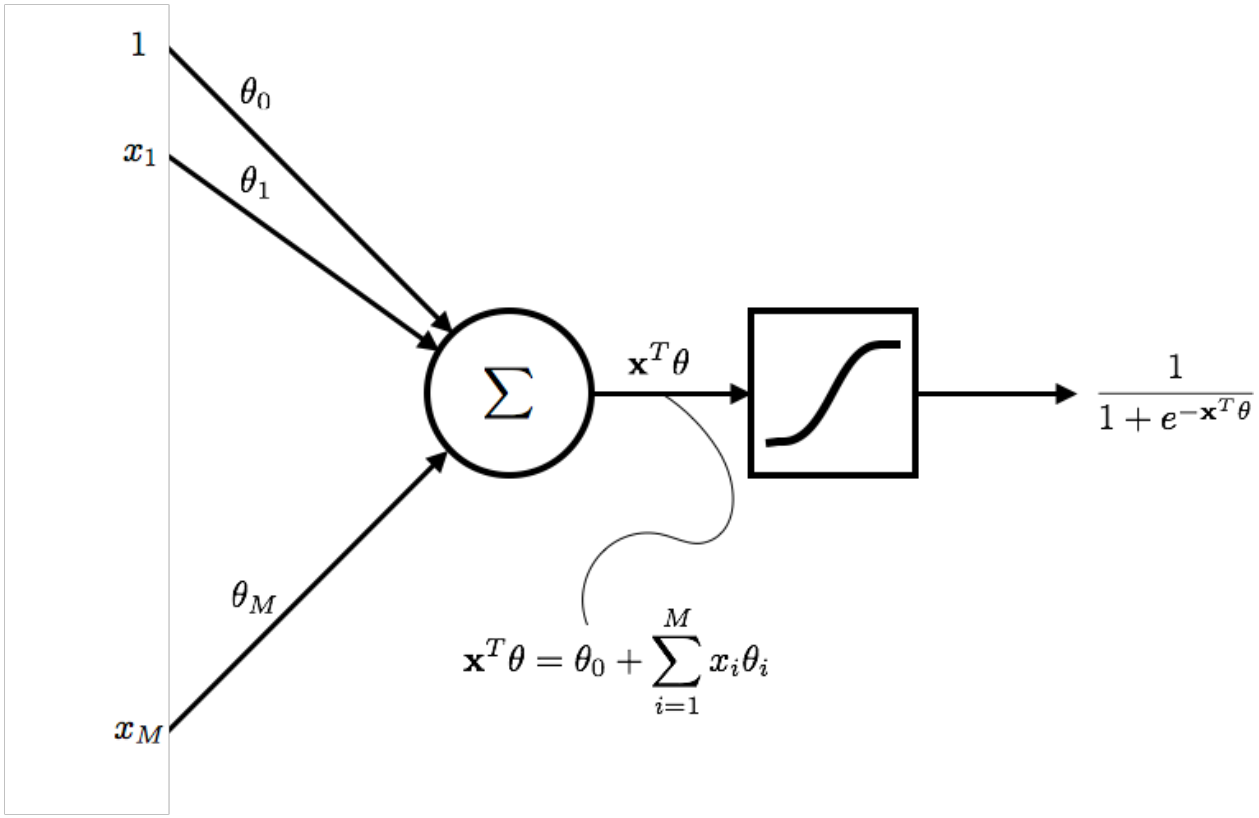


Pictorial view of logistic regression

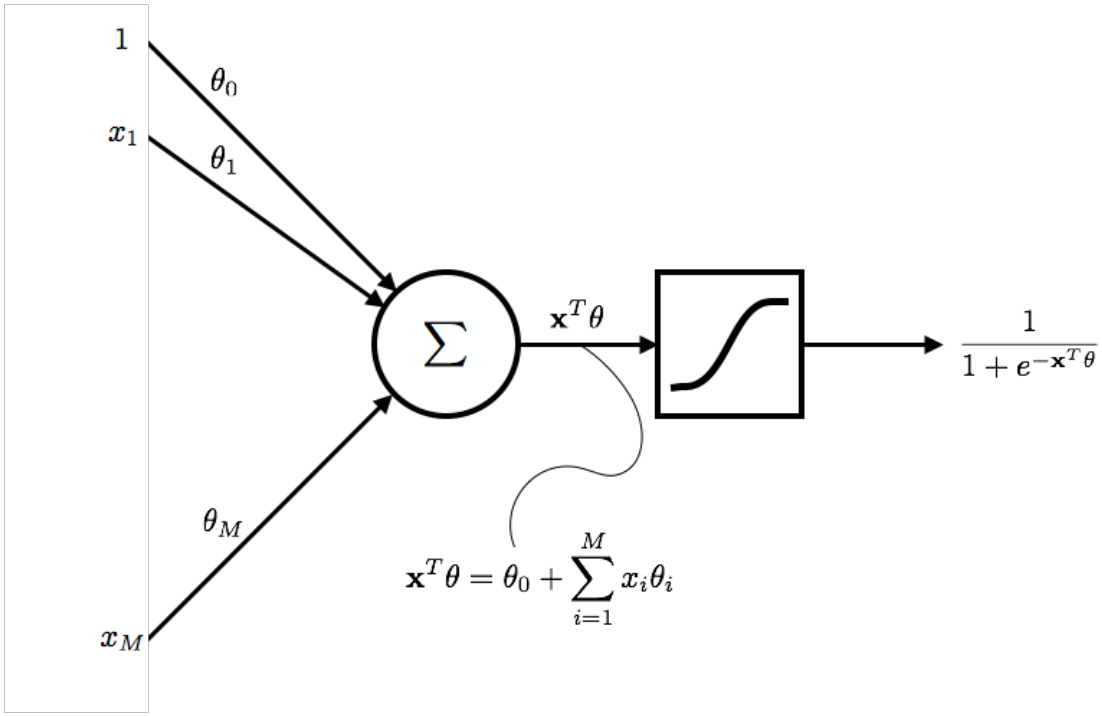


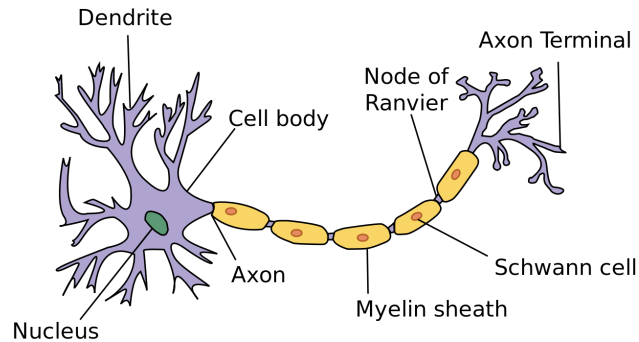


Pictorial view of logistic regression (in higher dimensions)



Artificial Neuron



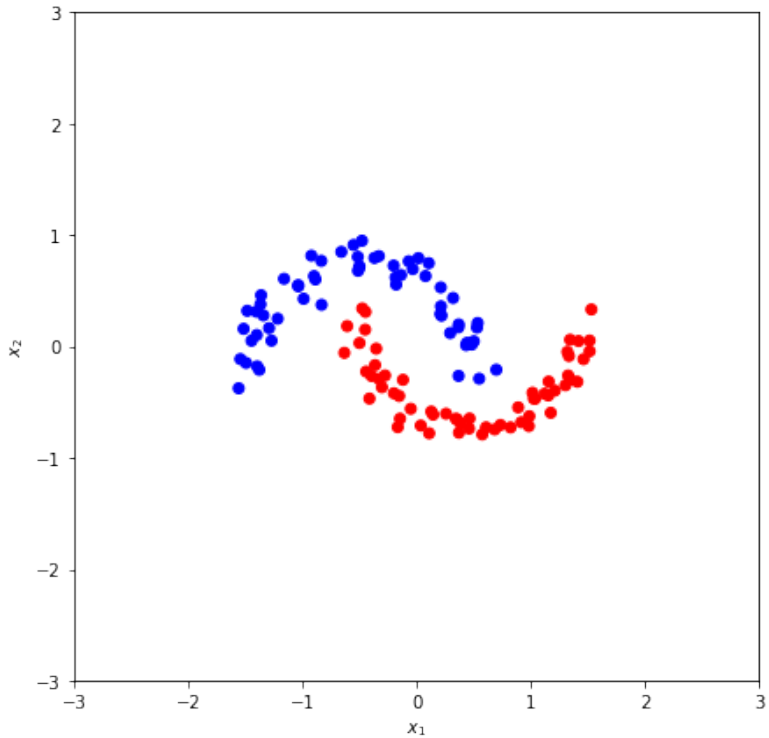


From wikipedia.

- McCulloch and Pitts (neuron), 1943.
- Hebb (Hebb's rule), 1949.
- Rosenblatt (perceptron), 1962.

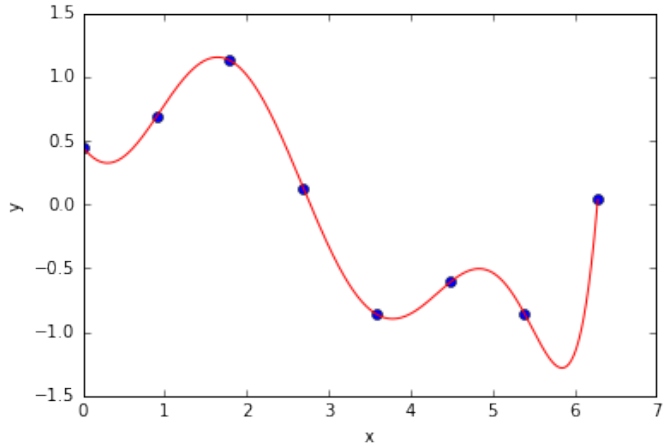
Another Binary Classification Problem

Can we use logistic regression to solve this binary classification problem?



Lets Go Back to Regression for a Moment

Can we use the least square fitting technique that we used to fit a line to a set of data points to fit a curve?



We need to change our model to be able to deal with non-linearities.

Basis functions

It is possible to introduce non-linearity in the system by using basis functions $\phi(.)$ as follows:

$$f(\mathbf{x}) = \phi(\mathbf{x})^T \theta$$

Example:

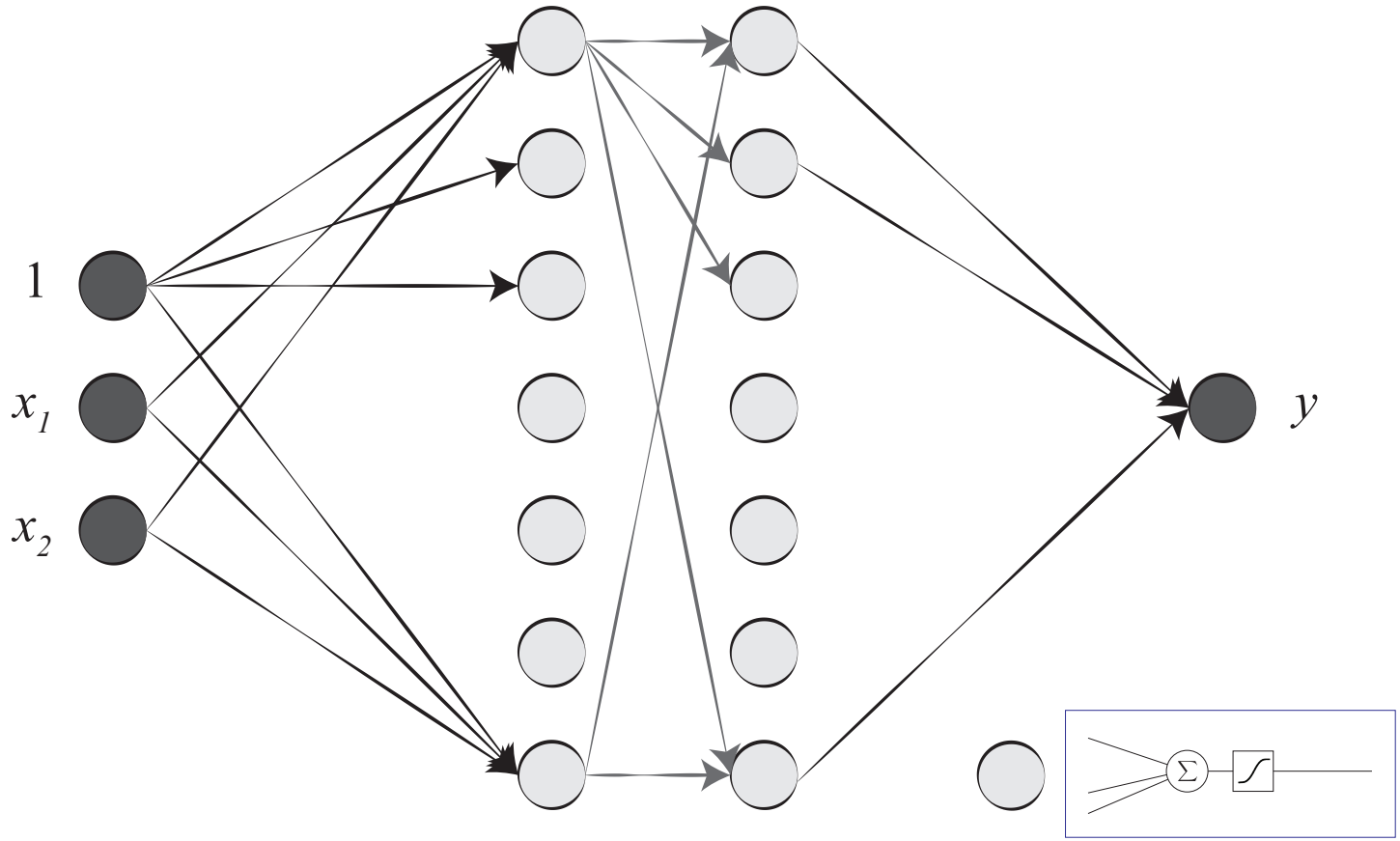
For a cubic polynomial (in 1D)

$\phi_0(x) = 1$
$\phi_1(x) = x$
$\phi_2(x) = x^2$
$\phi_3(x) = x^3$

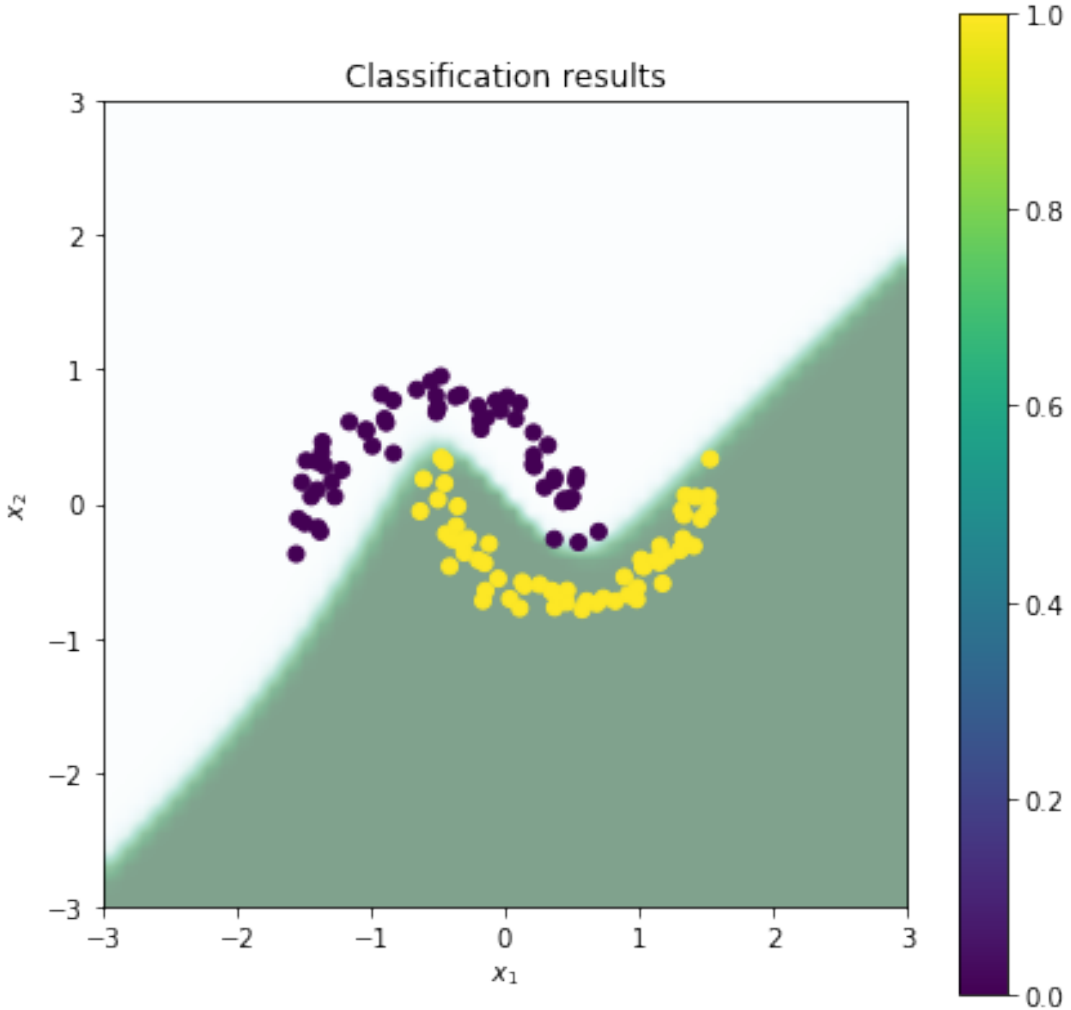
Then

$$f(x) = \begin{bmatrix} \phi_0(x) & \phi_1(x) & \phi_2(x) & \phi_3(x) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

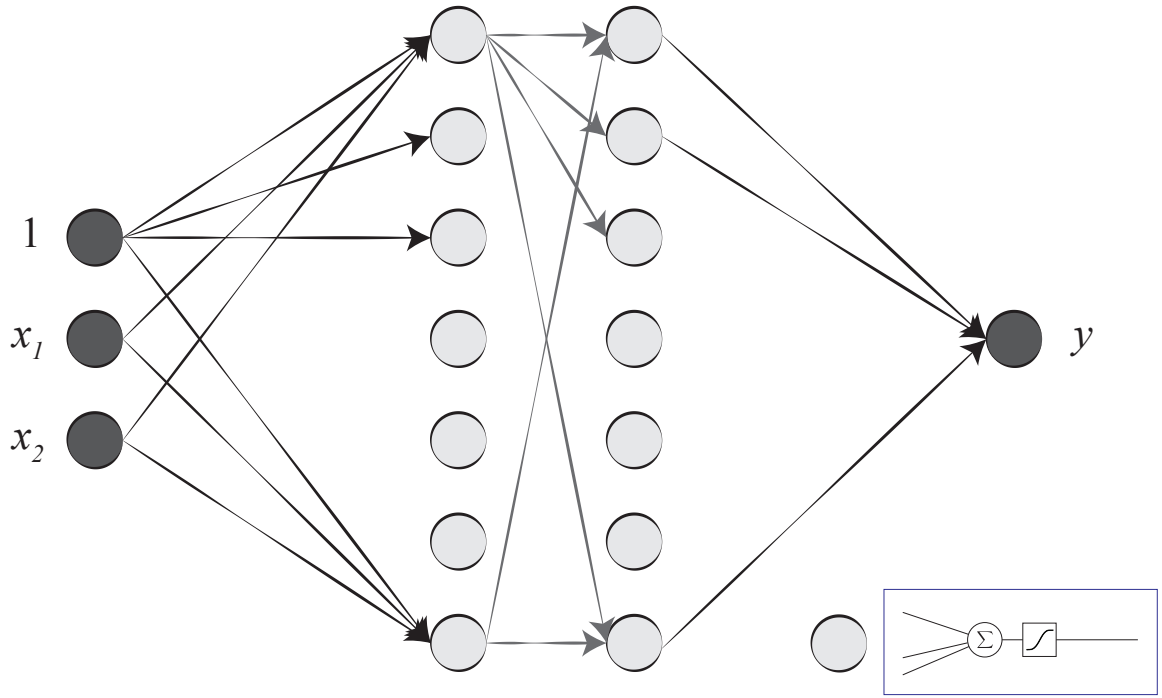
Artificial Neural Networks (ANN)



Binary Classification using ANN



Artificial Neural Networks (ANN)

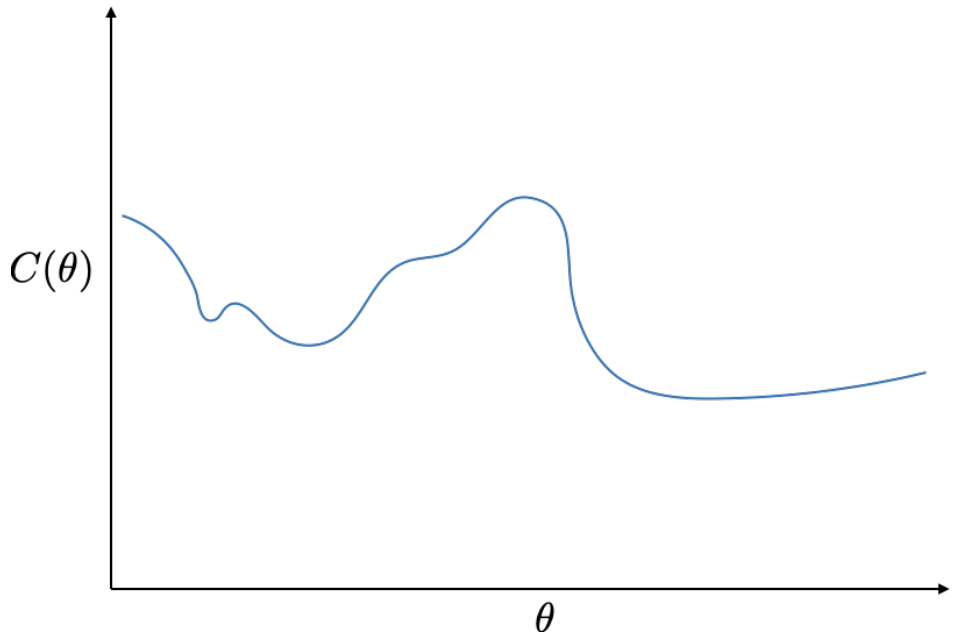


How do we fit this model to data?

How many parameters does this model have?

Gradient descent

A very powerful method of training the model parameters by minimizing the loss function. One of the simplest optimization methods. It is also referred to as *steepest descent*.



Gradient descent Recipe

1. Initialize model parameters randomly (in our case θ)
2. Compute gradient of the loss function
3. Take a step in the direction of negative gradient (decreasing loss function), and update parameters
4. Repeat steps 2 to 4 until cannot decrease loss function anymore

Cauchy, 1847.

See Petrova and Solov'ev, 1996 for a brief histroy of gradient descent.

Gradient

Let $\theta \in \mathbb{R}^D$ and $f(\theta)$ is a scalar-valued function. The gradient of $f(.)$ with respect to θ is:

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \frac{\partial f(\theta)}{\partial \theta_3} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_D} \end{bmatrix} \in \mathbb{R}^D$$

Gradient descent

Gradient descent update rule is:

$$\begin{aligned}\theta^{(k+1)} &= \theta^{(k)} - \eta\frac{\partial C}{\partial \theta} \\ &= \theta^{(k)} - \eta\nabla_\theta C \\ &= \theta^{(k)} + \Delta\theta^{(k+1)}\end{aligned} \tag{1}$$

η is referred to as the *learning rate*. It controls the size of the step taken at each iteration.

When using *momentum*, we select $\Delta\theta^{(k+1)}$ as follows:

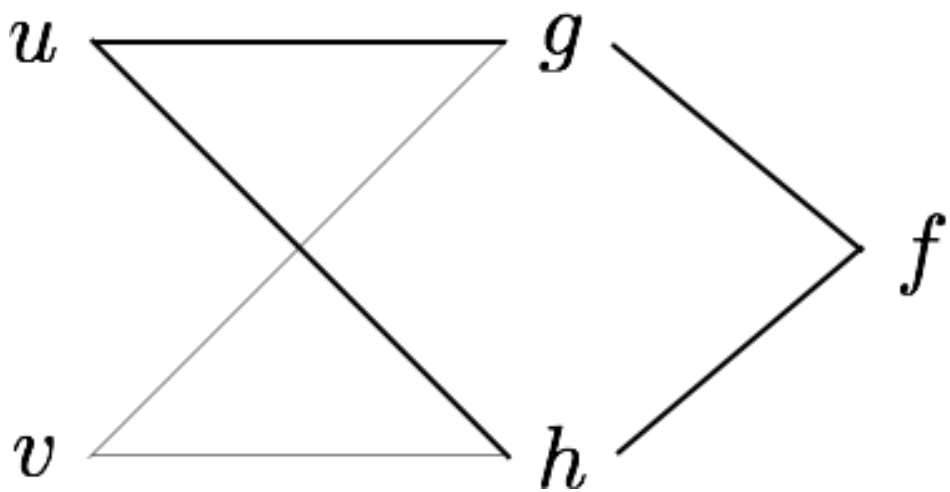
$$\Delta\theta^{(k+1)} = \alpha\Delta\theta^{(k)} + (1-\alpha)(-\eta\nabla_\theta C)$$

α refers to momentum.

Chain rule

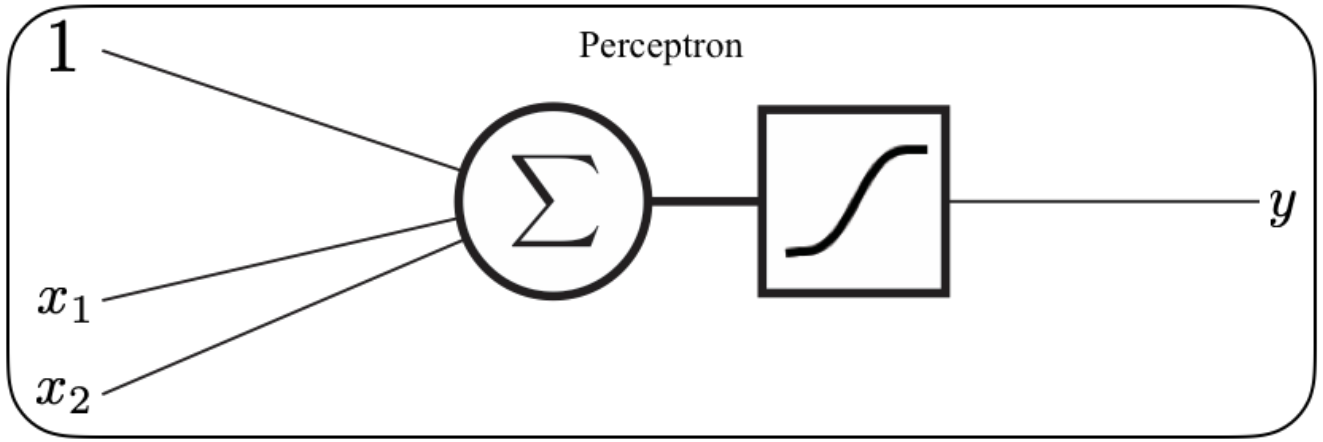
In calculus, a formula for computing the derivative of the composition of more than one function.

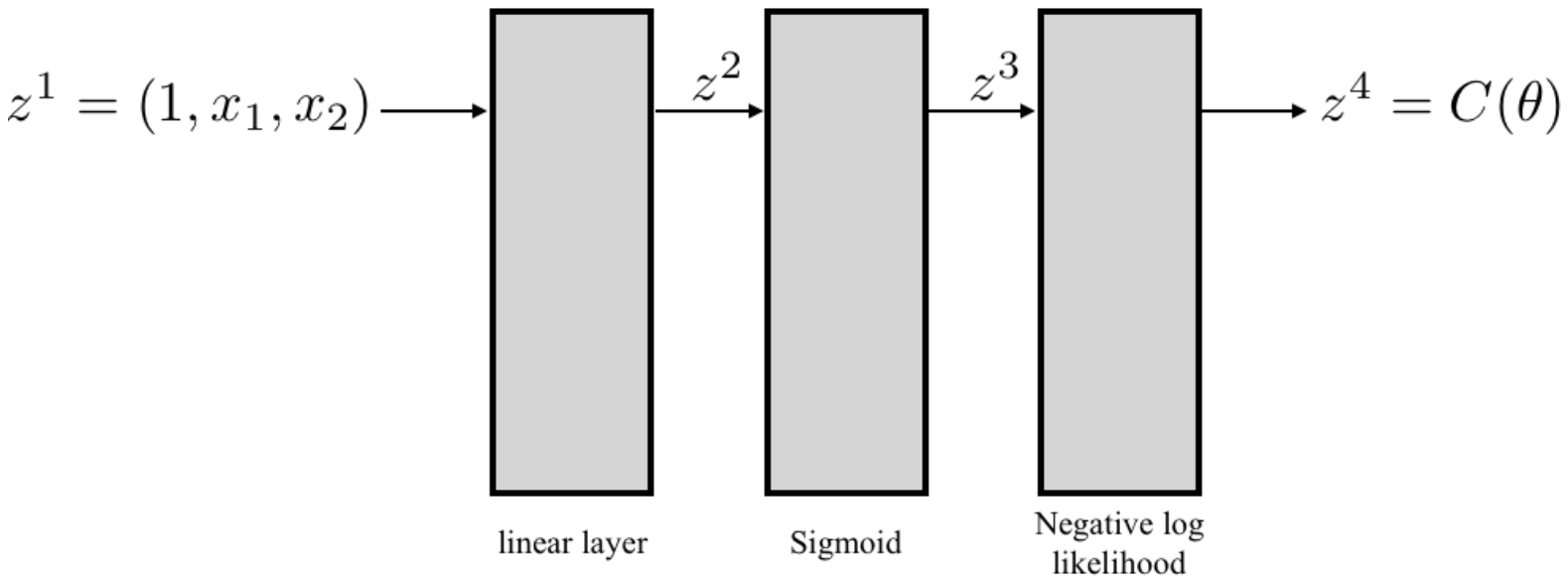
$$\frac{\partial f(g(u,v),h(u,v))}{\partial u} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial u} + \frac{\partial f}{\partial h}\frac{\partial h}{\partial u}$$



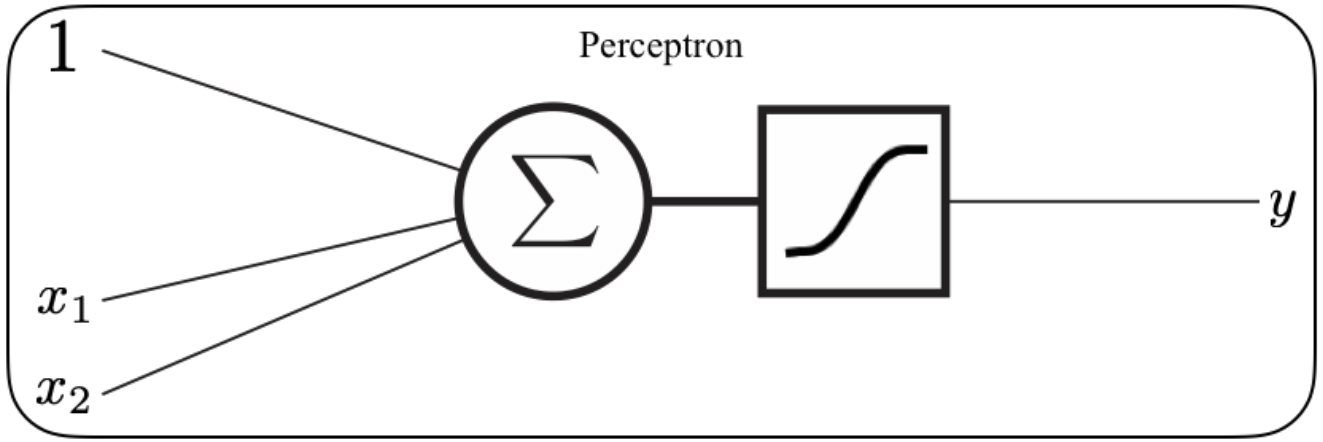
Leibniz and Newton, 17th century.

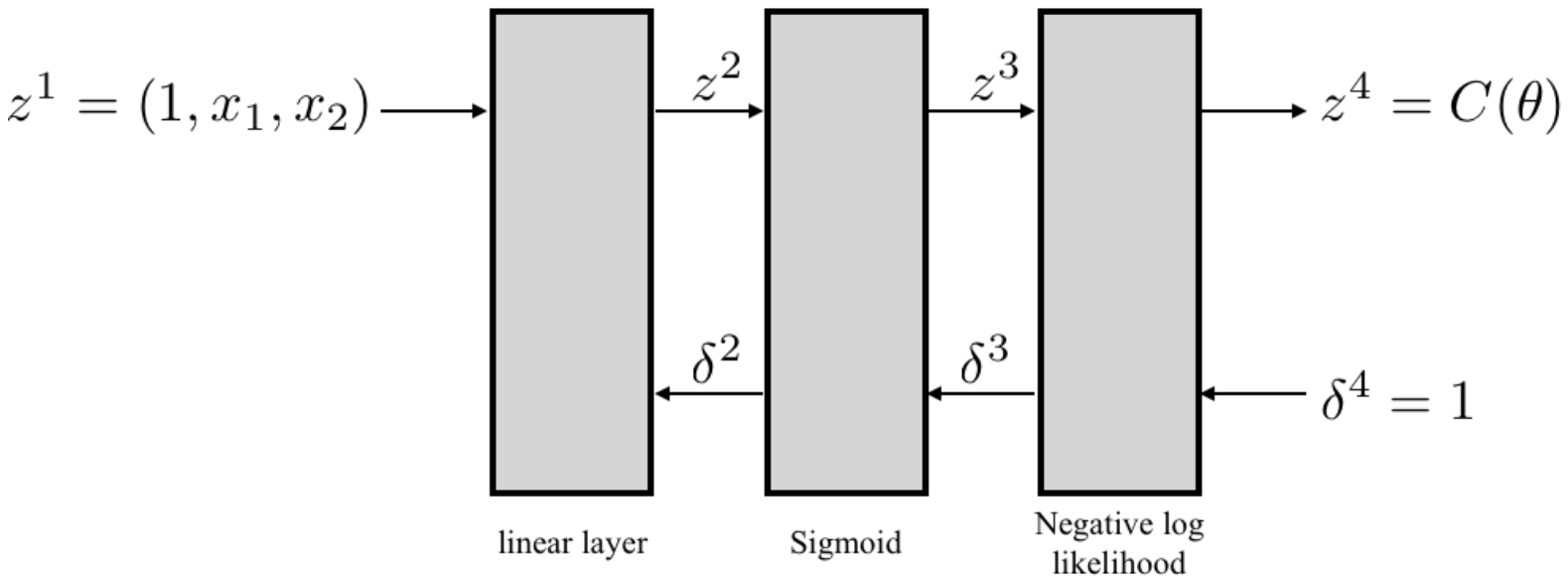
A layered view (1)



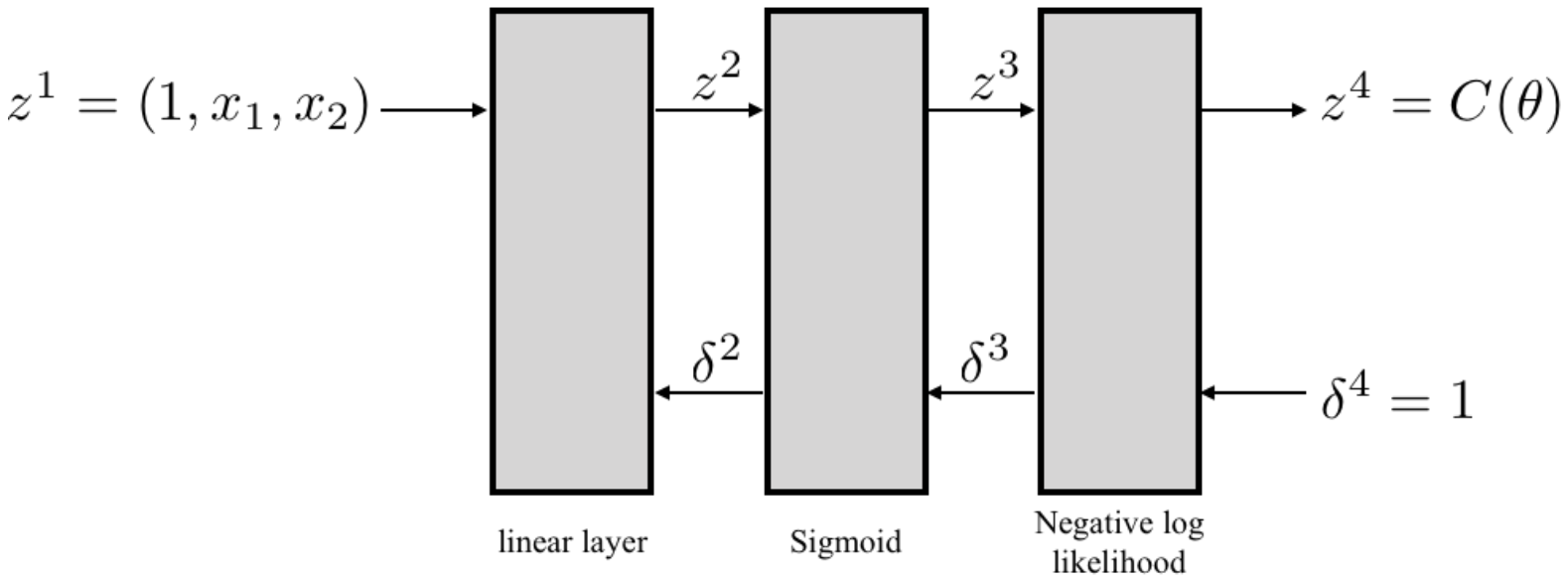


A layered view (2)





A layered view (3)

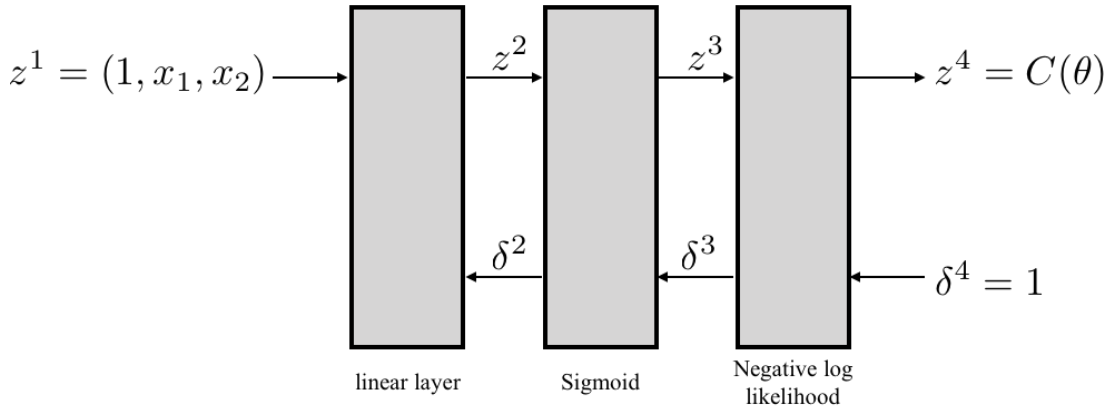


z^l = Input to layer l

z^{l+1} = Output of layer l

$\delta^l = \frac{\partial C(\theta)}{\partial z^l}$

Computing δ^l



$$\delta^4 = \frac{\partial C(\theta)}{\partial z^4} = \frac{\partial z^4}{\partial z^4} = 1$$

$$\delta_1^3 = \frac{\partial C(\theta)}{\partial z_1^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_1^3} = \delta^4 \frac{\partial z^4}{\partial z_1^3}$$

$$\delta_2^3 = \frac{\partial C(\theta)}{\partial z_2^3} = \frac{\partial C(\theta)}{\partial z^4}\frac{\partial z^4}{\partial z_2^3} = \delta^4 \frac{\partial z^4}{\partial z_2^3}$$

$$\delta_1^2 = \frac{\partial C(\theta)}{\partial z_1^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_1^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_1^2}$$

$$\delta_2^2 = \frac{\partial C(\theta)}{\partial z_2^2} = \sum_k \frac{\partial C(\theta)}{\partial z_k^3}\frac{\partial z_k^3}{\partial z_2^2} = \sum_k \delta_k^3 \frac{\partial z_k^3}{\partial z_2^2}$$

For any differentiable layer l

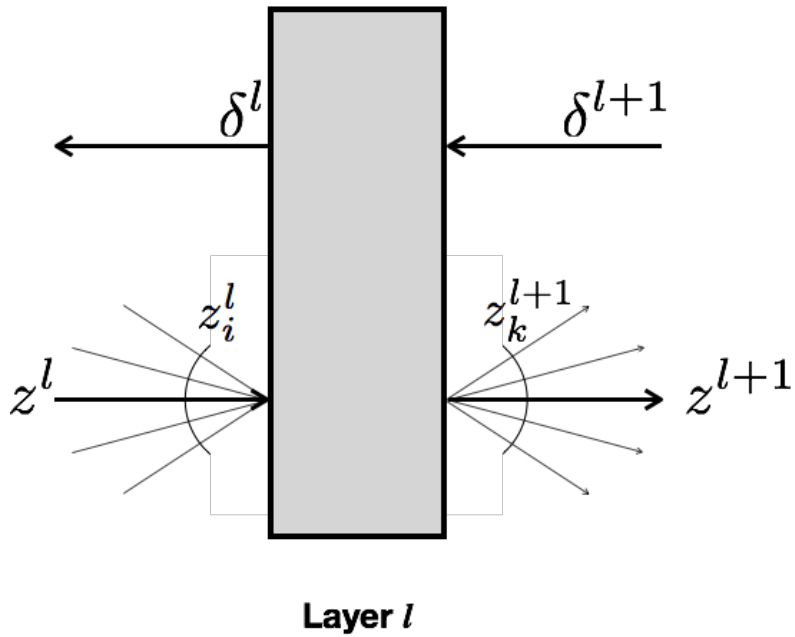
For a given layer l, with inputs z_i^l and outputs z_k^{l+1}

$$\delta_i^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_i^l}$$

Similarly, for layer l that depends upon parameters θ^l,

$$\frac{\partial C(\theta)}{\partial \theta^l} = \sum_k \frac{\partial C(\theta)}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial \theta^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial \theta^l}$$

In our 2-class softmax classifier only layer 1 has parameters (θ_0 and θ_1).



Backpropagation

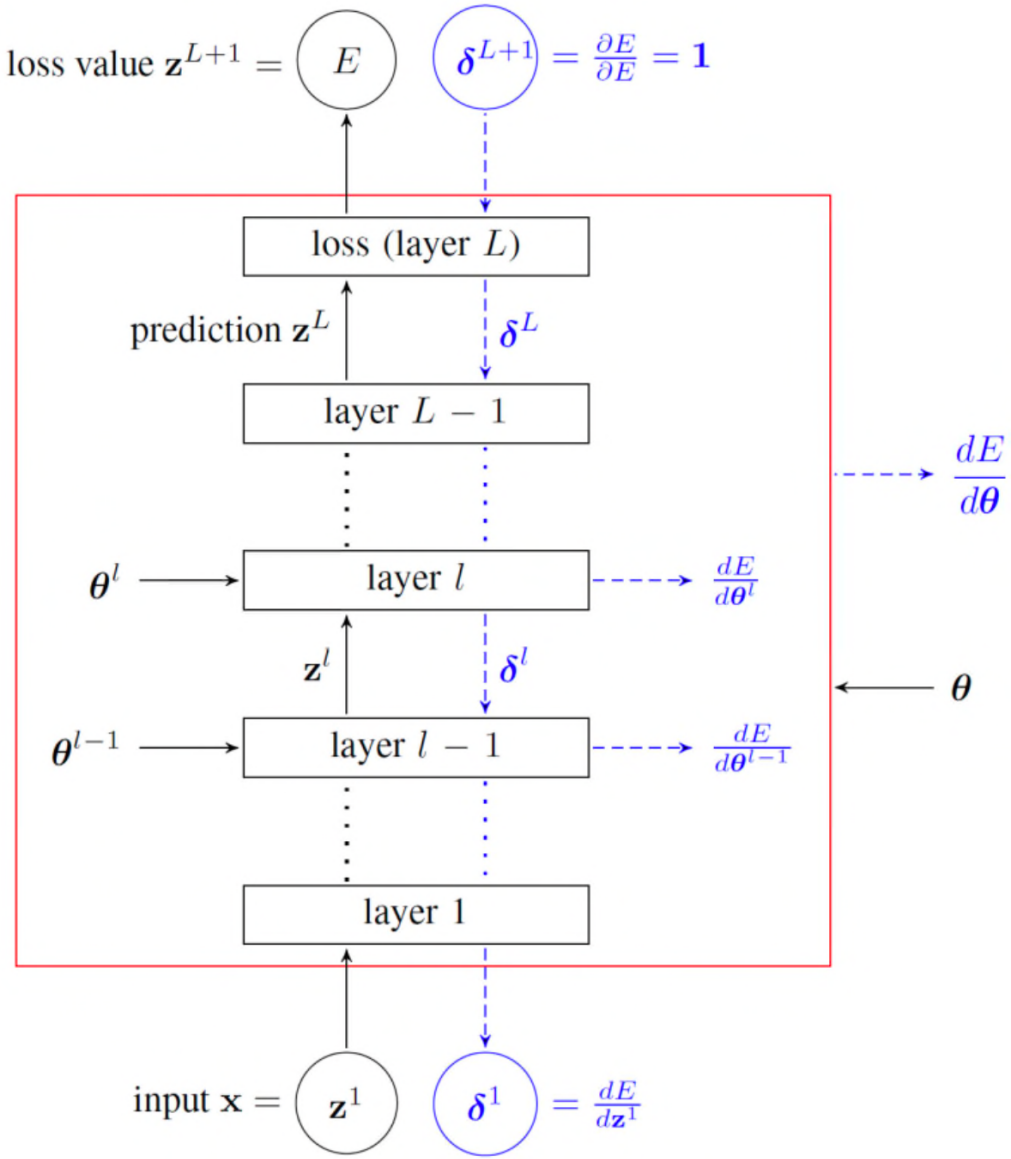
- Set z^1 equal to input $\mathbf{x}$.
- Forward pass: compute $z^2, z^3, \ldots$ layers $1, 2, \ldots$ activations.
- Set δ at the last layer equal to 1
- Backward pass: backpropagate δs all the way to first layer.
- Update θ
- Repeat

Dreyfus, 1962, 1973.

Rumelhart et al., 1986.

See also, automatic differentiation.

Deep learning: backpropagation



From Nando de Freitas

$$\mathbf{z}^{l+1} = \mathbf{f}^l(\mathbf{z}^l; \theta^l)$$

$$\delta^l = \delta^{l+1} \frac{\partial \mathbf{f}^l(\mathbf{z}^l; \theta^l)}{\partial \mathbf{z}^l}$$

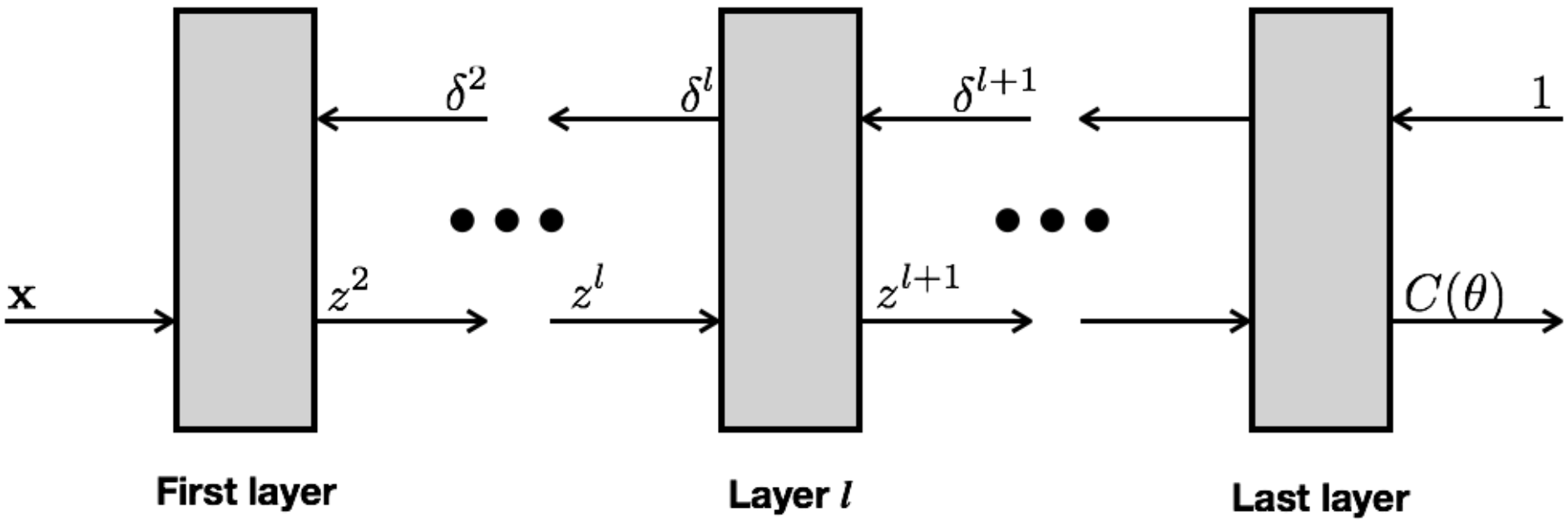
$$\delta_i^l = \sum_j \delta_j^{l+1} \frac{\partial f_j^l(\mathbf{z}^l; \theta^l)}{\partial z_j^l}$$

$$\frac{\partial E}{\partial \theta^l} = \delta^{l+1} \frac{\partial \mathbf{f}^l(\mathbf{z}^l; \theta^l)}{\partial \theta^l}$$

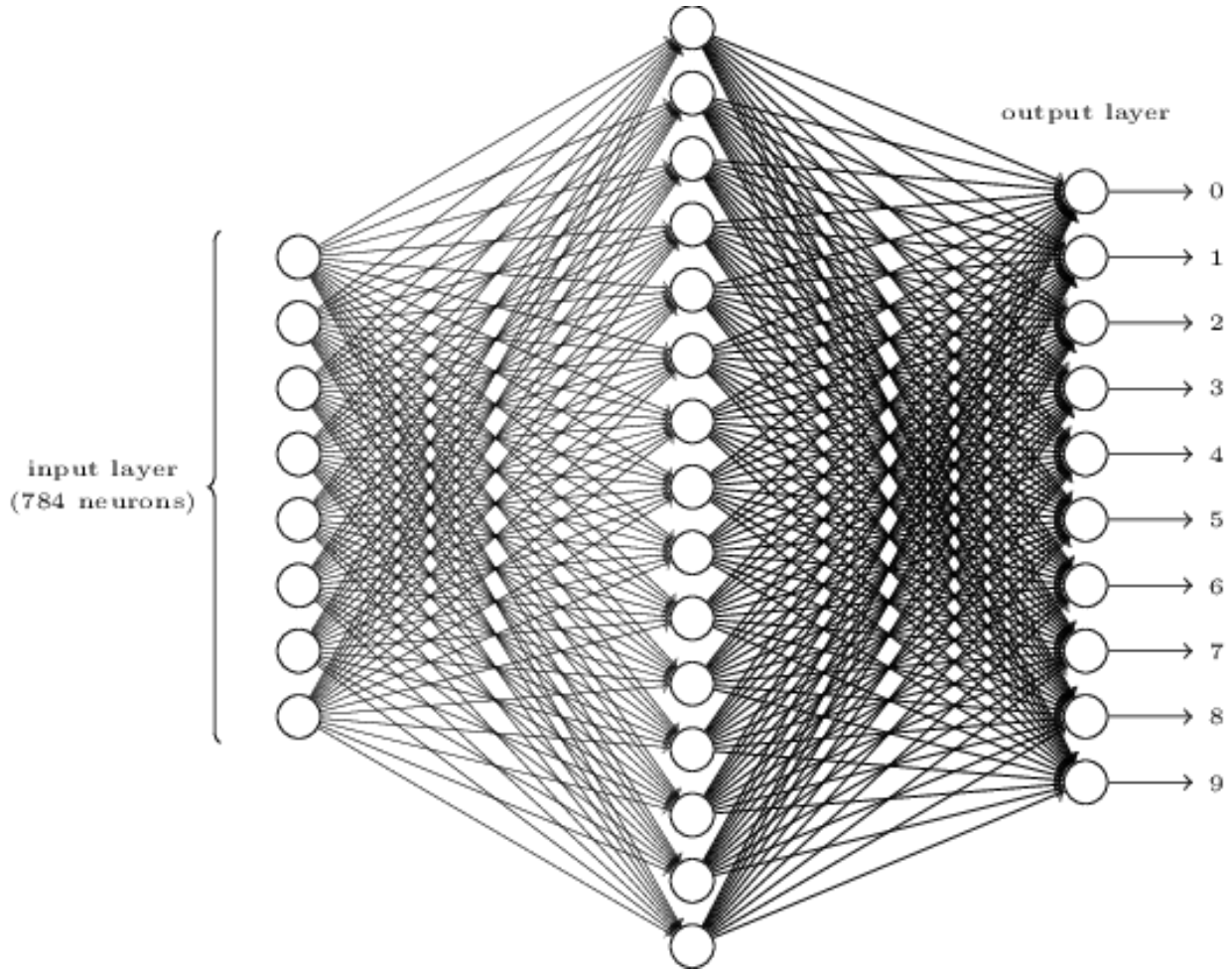
$$\frac{\partial E}{\partial \theta_i^l} = \sum_j \delta_j^{l+1} \frac{\partial f_j^l(\mathbf{z}^l; \theta^l)}{\partial \theta_i^l}$$

Layered architectures

As long as we have differentiable layers, i.e., we can compute $\frac{\partial z_k^{l+1}}{\partial z_i^l}$, we can use *backpropagation* to update the parameters θ to minimize the cost $C(\theta)$.



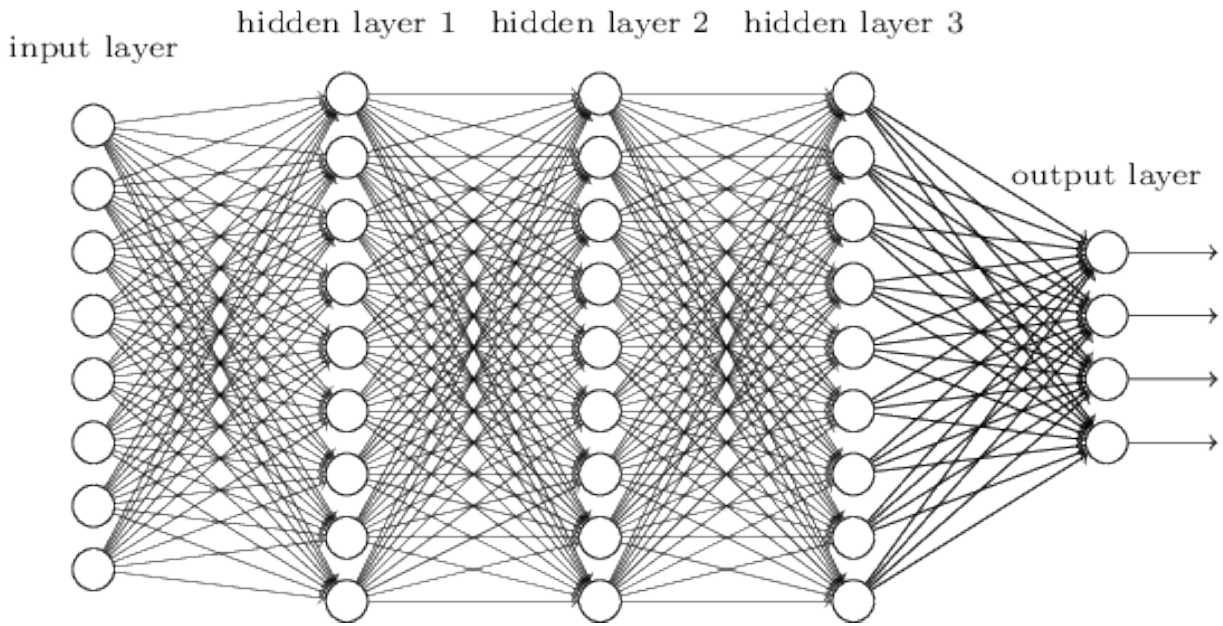
The time before deep networks



- Shallow and wide
- One hidden layer can represent any function
- Focus was on efficient ways to optimize (train)

From http://neuralnetworksanddeeplearning.com

Deep learning (1)



- Deep networks - multi-layer networks
- Access to large amount of labeled data
- Advances in computing hardware

From http://neuralnetworksanddeeplearning.com

Deep learning (2)

- Excels at supervised learning
- Forms representations at various level of abstractions, where higher level abstractions are formed by the composition of lower level features

Artificial Neural Network

Deep Neural Network

Hinton, 2006.

Deep learning (3)

- Multilayer Perceptron Networks.
- Convolutional Neural Networks.
- Long Short-Term Memory Recurrent Neural Networks.

Gradient-based learning in neural networks

- Non-linearities of neural networks render most cost functions non-convex
- Use iterative gradient based optimizers to drive cost function to lower values
- Gradient descent applied to non-convex cost functions has no guarantees, and it is sensitive to initial conditions
  - Initialize weights to small random values
  - Initialize biases to zero or small positive values

Stochastic gradient descent

In stochastic (or "on-line") gradient descent, the true gradient of $\nabla_\theta C$ is approximated by a gradient at a single example: $\nabla_\theta C^{(i)}$

Stochastic gradient descent update rule

1. Choose an initial vector of parameters θ and a learning rate η
2. Repeat:
   - Randomly shuffle examples in the training set.
   - for $i = 1, 2, 3, \cdots, n$, do:
     - $\theta := \theta - \eta \nabla_\theta C^{(i)}$

Commonly used stochastic gradient descent methods

- AdaGrad (Duchi, *et al.*, 2011)
- RMSProp (Tijmen and Hinton, 2012)
- Adam (Diederik and Ba, 2014) – widely used for deep networks training

Output units

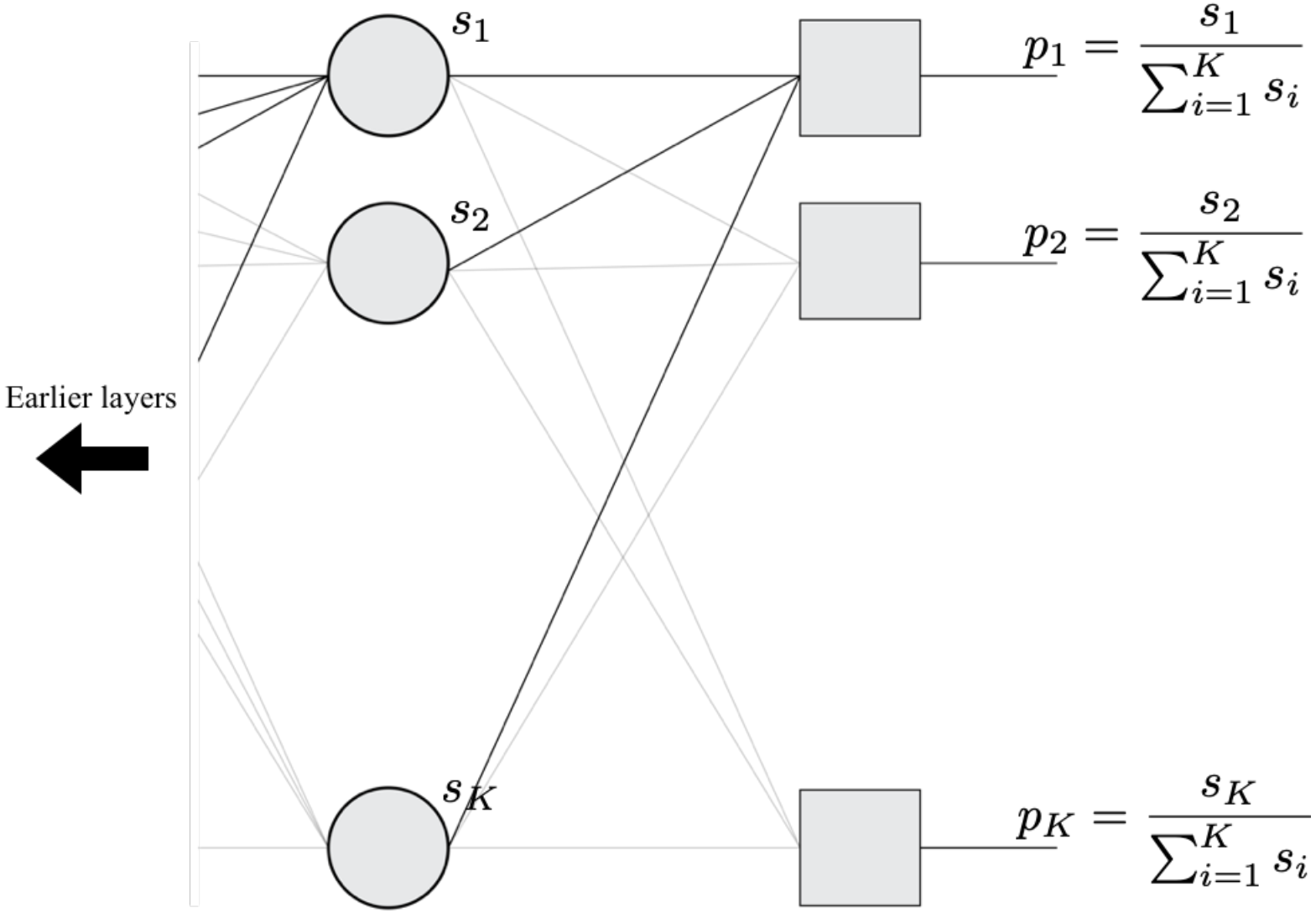
The role of the output units is to provide some additional transformations from the features computed by the hidden layers to complete the task at hand:

$$y = f(\mathbf{h}),$$

where $\mathbf{h} = g(\mathbf{x}; \theta)$ are the features computed by the hidden layer(s).

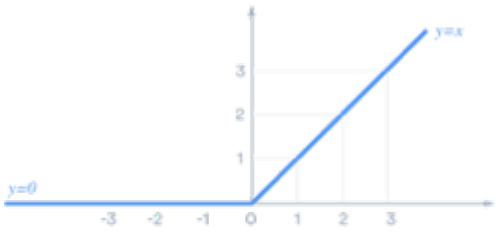
- Linear units (Regression tasks)
- Sigmoid units (Binary classification)
- Softmax units (Multi-class classification)

Softmax

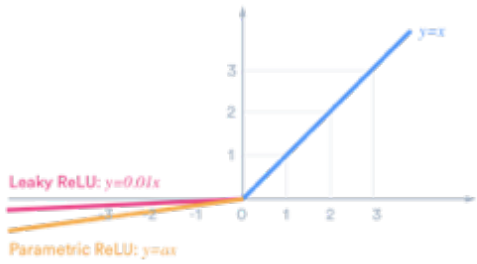

s_1
s_2
s_K
Earlier layers
$p_1 = \frac{s_1}{\sum_{i=1}^{K} s_i}$
$p_2 = \frac{s_2}{\sum_{i=1}^{K} s_i}$
$p_K = \frac{s_K}{\sum_{i=1}^{K} s_i}$

Hidden units (1)

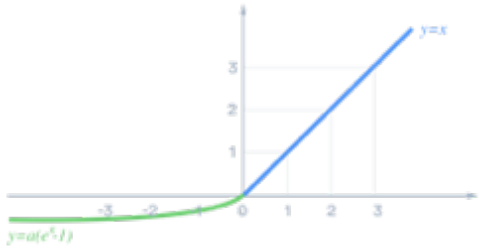
- ReLU and its variants
- Logistic, sigmoid, hyperbolic tangent
- Maxout
- Dropout



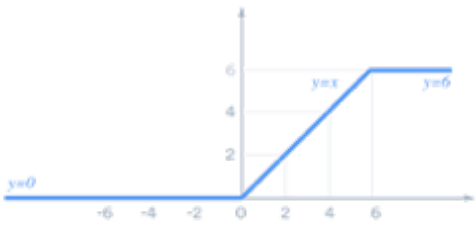
ReLU



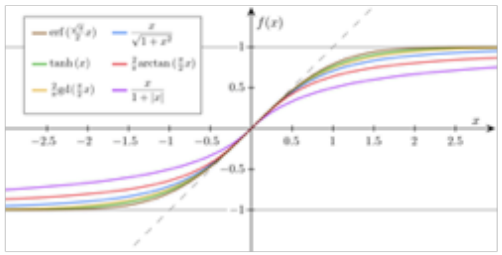
Leaky ReLU



Exponential Linear



Concatenated ReLU



Sigmoid like functions

From Liu, Medium.com

Hidden units (2)

- Use ReLU
- Select a small learning rate, and monitor the fraction of *dead* neurons in your network
- If this fails, give Leaky ReLU a try
- Do *not* use sigmoid
- For recurrant neural networks, try tanh
- No harm in trying maxout

You need to set up a mechanism for evaluating the performance of your network. This allows you to evaluate different design choices.

Regularization for deep networks (1)

Regularization — any modification to reduce generalization error but not the training errors:

- extra constraints and penalties
- prior knowledge

Deep learning is applied to extremely complex tasks. Consequently, regularization is not as simple as controlling the number of parameters

Regularization for deep networks (2)

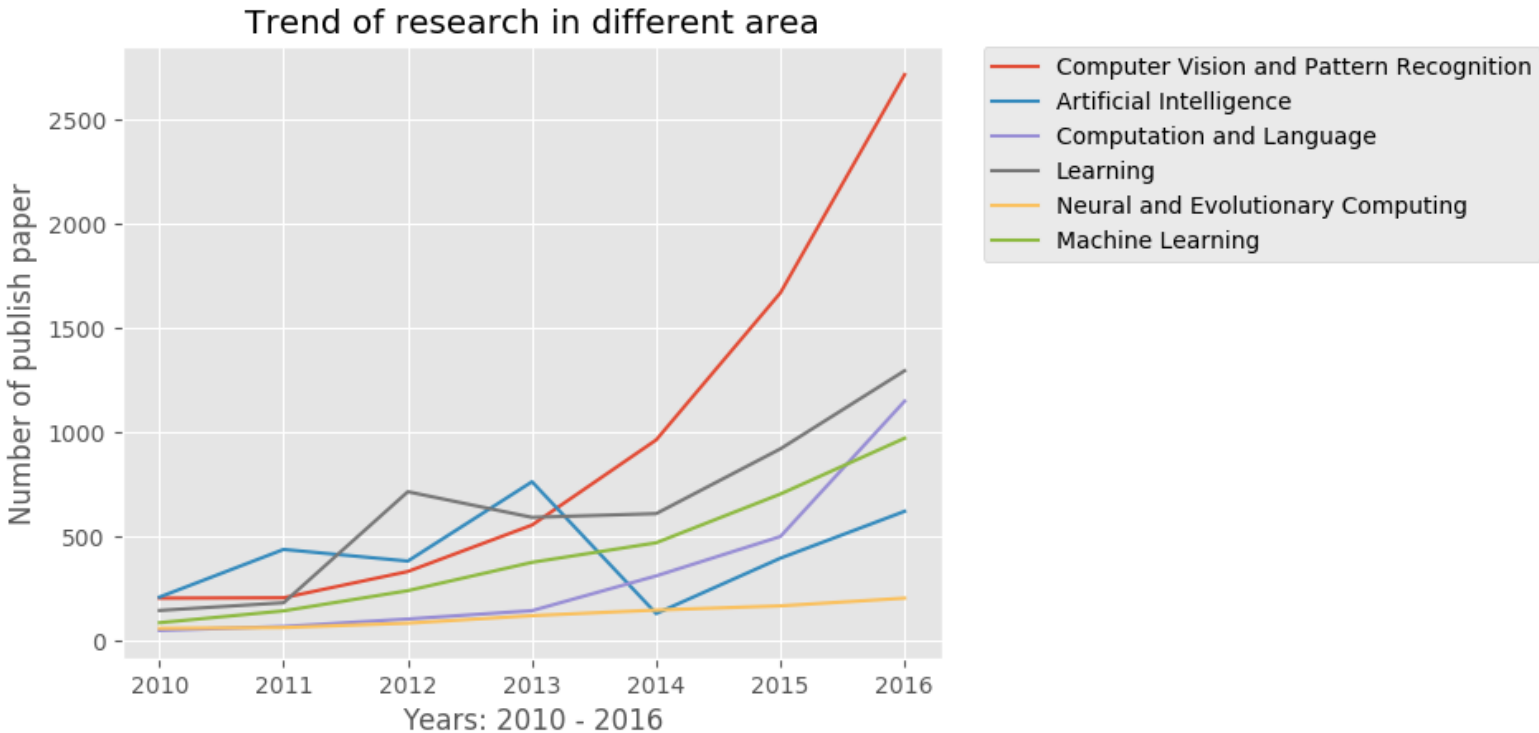
- Parameter norm penalties
- Data augmentation
  - Fake data
  - Successful in classification/object recognition tasks
- Noise injection
  - Applying random noise to the inputs
  - Applying random noise to hidden layers' inputs
    - Data augmentation at multiple levels of abstraction
  - Data augmentation almost always improves the performance of a neural network
  - Noise added to the weights
    - Recurrent neural networks
    - A practical stochastic implementation of Bayesian inference over weights
  - Noise can also bve added to target outputs

Regularization for deep networks (3)

- Dropout (Srivastava, *et al.*, 2014)
  - Turn off random units
  - Prevents over-fitting and co-dependencies
- BatchNorm (Ioffe and Szegedy, 2015)
  - Improves gradient flow, used on very deep models (Resnet need this)
  - Allow higher learning rates
  - Reduce dependency on initialization
  - No need for L2 regularization if using BatchNorm+Dropout
  - Use *after linear* layers and *before non-linear* layers

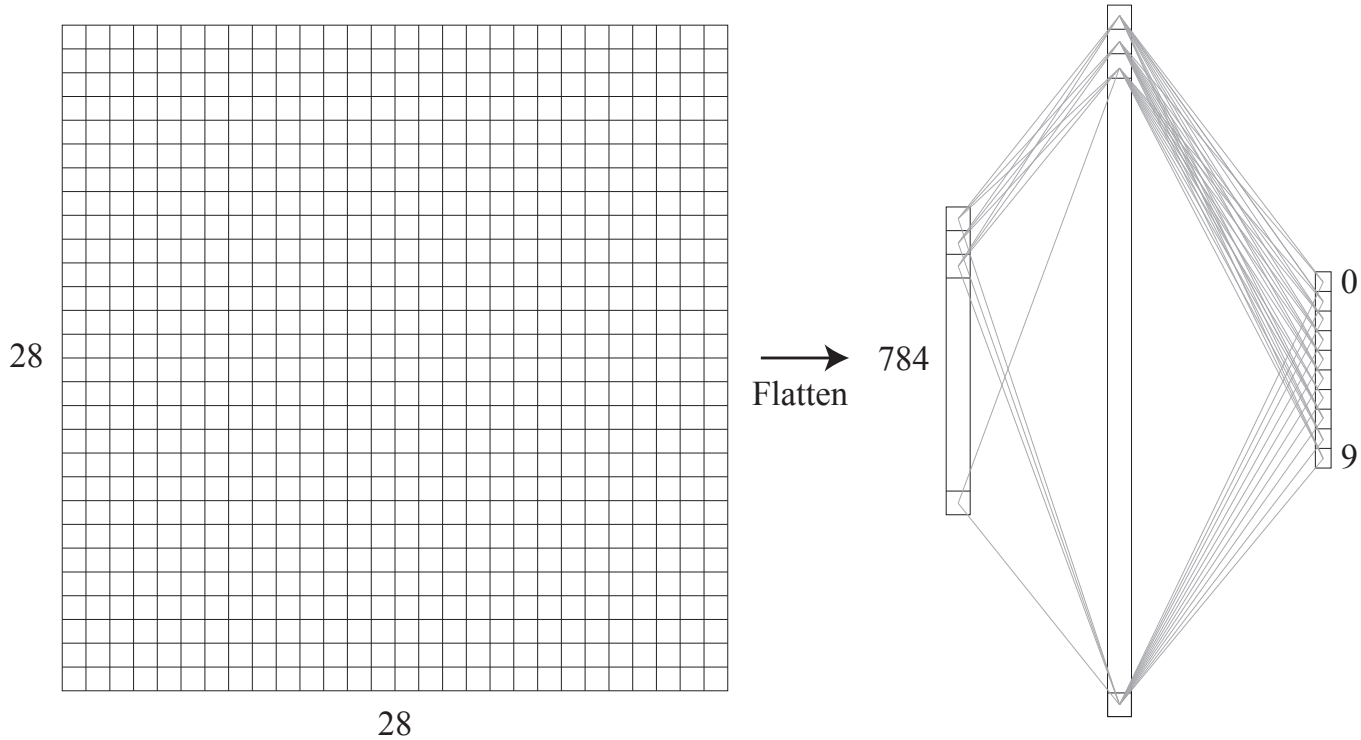
Deep learning and computer vision

- Krizhevsky *et al.*, 2012. "ImageNet Classification with Deep Convolutional Networks" won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC).



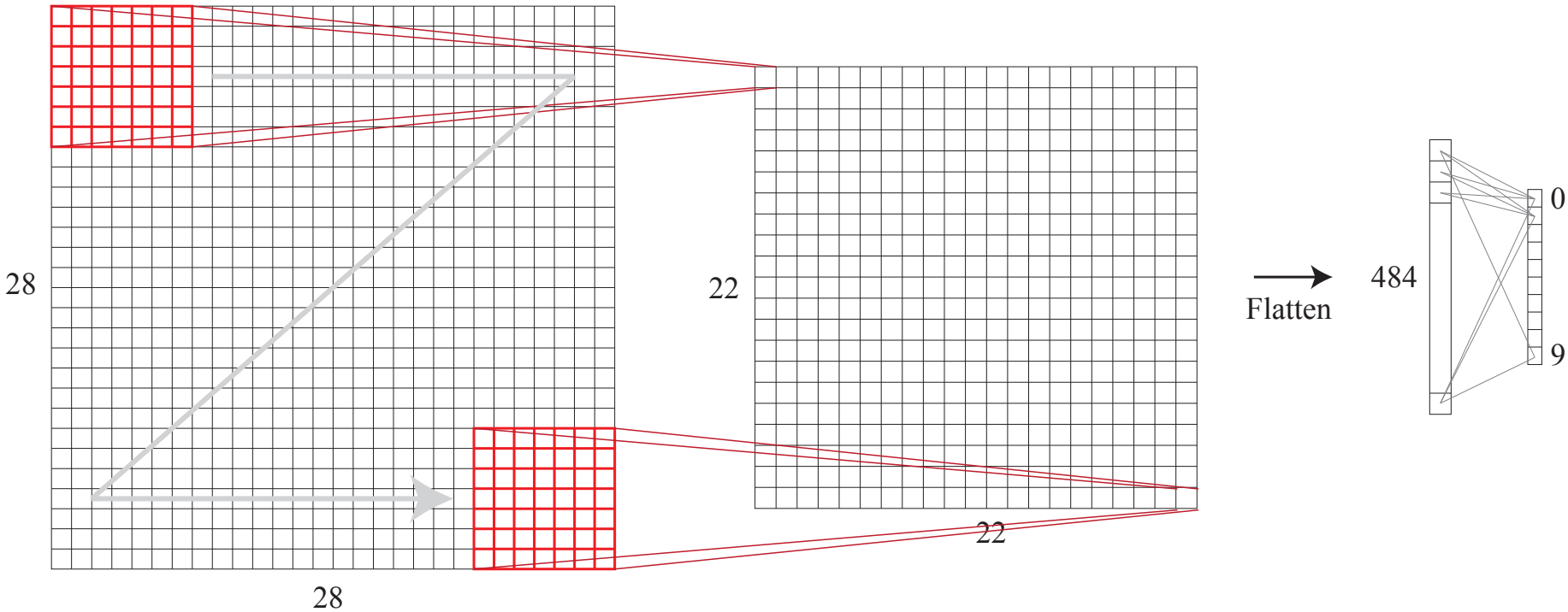
From N. Agarwal, Medium.com.

Character recognition (1)



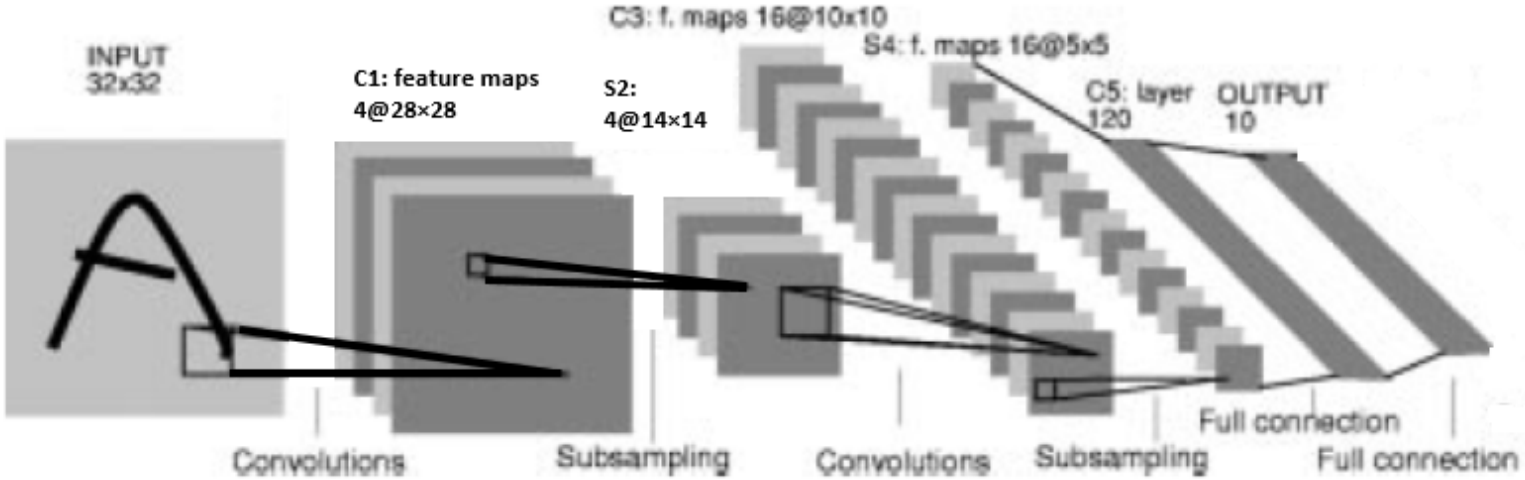
- Treats images as a high-dimensional data points
- Curse of dimensionality
  - A 100×100 image is a $1,000,000$ data point
- Spatial structure is lost

Character recognition (2)



- Convolutional neural networks
  - A trainable convolutional layer
  - Maintains spatial structure
  - Far fewer parameters
- Suggested by Kunihiko Fukushima, 1980

Le Net



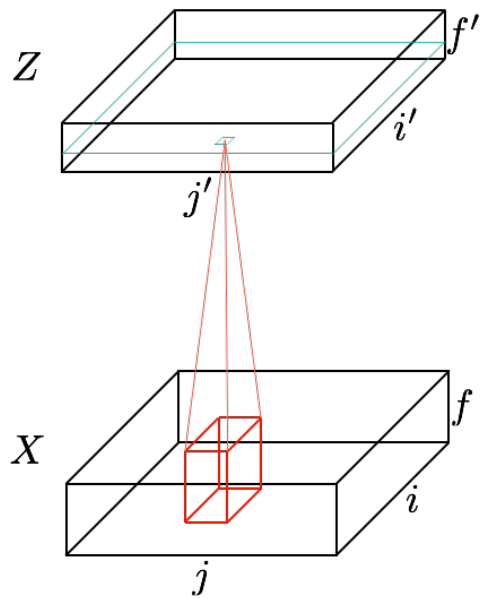
Le Cun *et al.*, 1989

Examples of digits correctly recognized by Le Net.

Figures from Sik-Ho Tsang, Medium.com and Le Cun, *et al.*, 1989

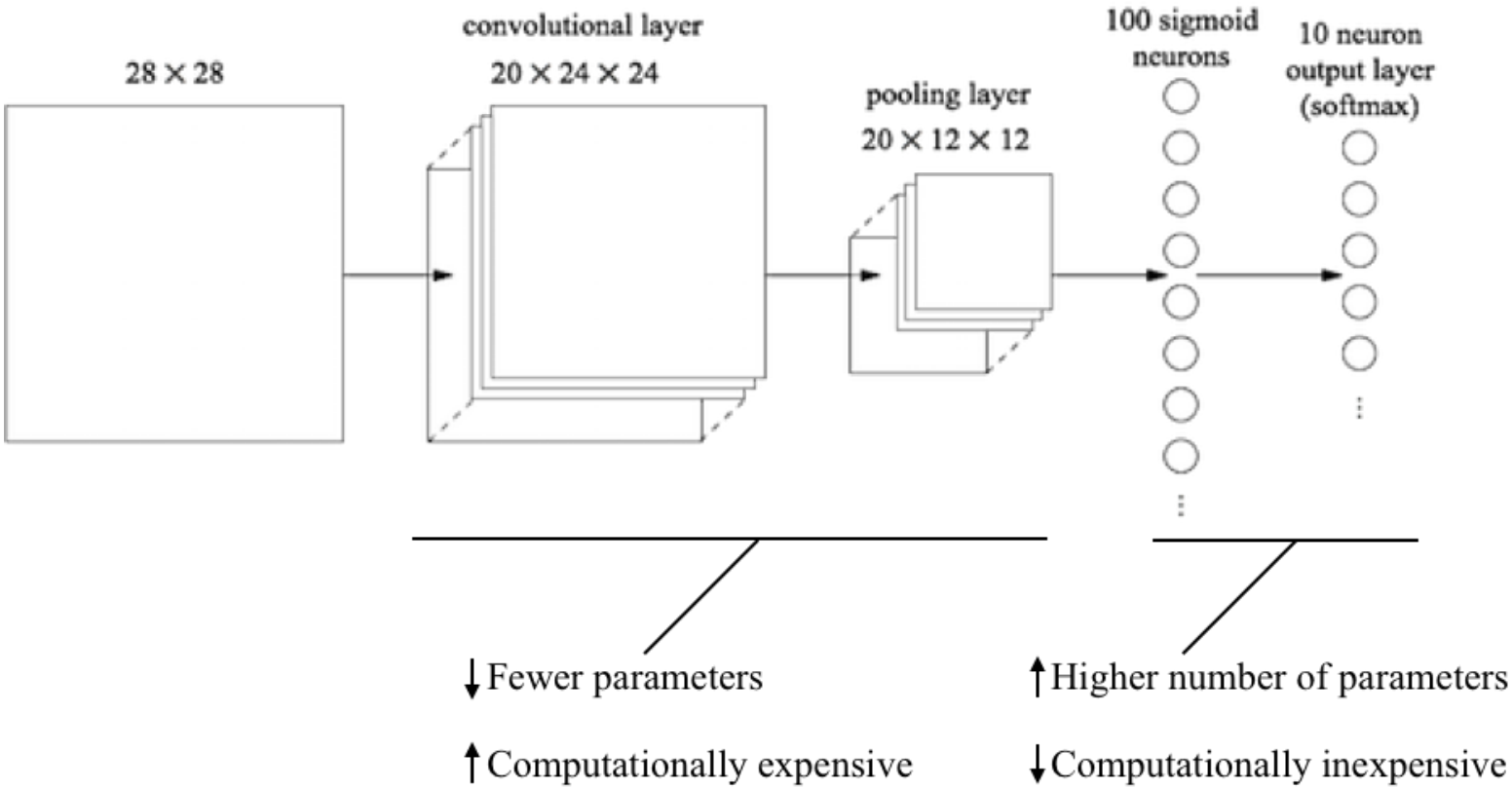
Differentiable convolutional layer

$$\mathbf{y}_{i',j',f'} = b_{f'} + \sum_{i=1}^{H_f} \sum_{j=1}^{W_f} \sum_{f=1}^{F} \mathbf{x}_{i'+i-1,j'+j-1,f} \theta_{ijff'}$$



Yet another digit classifier

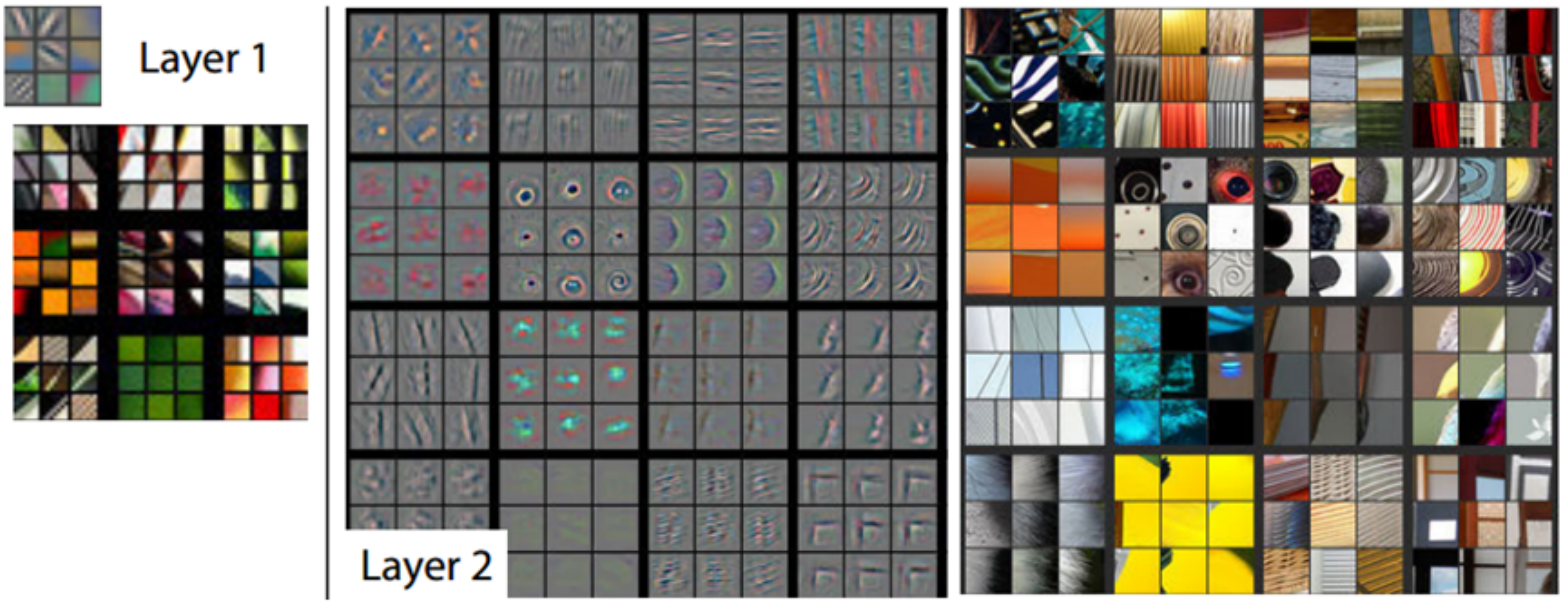
We can connect these layers in various configurations to design new networks



From http://neuralnetworksanddeeplearning.com

Convolution layers learning feature representations

Convolutional layers for a network trained on ImageNet.

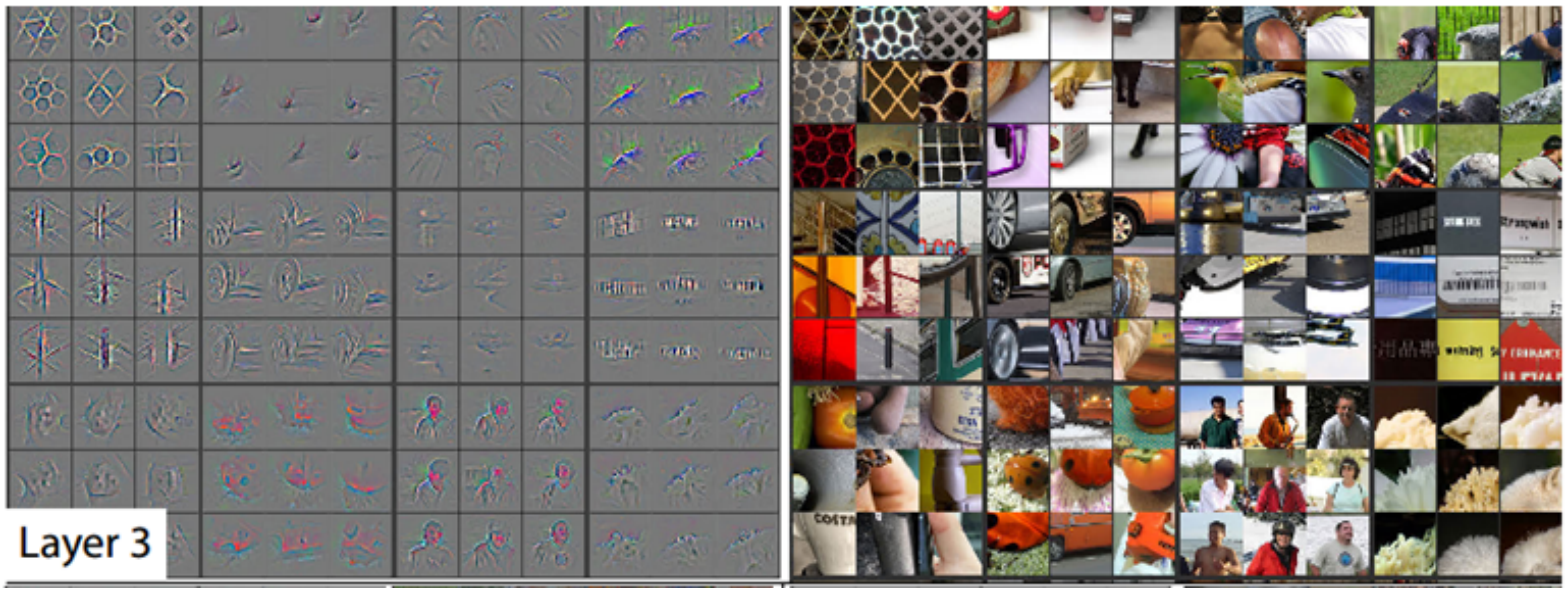


Visualizations for layers 1 and 2: Filters alongwith part of images that respond strongly to the filter.

Zeiler and Furgus, 2014.

Convolution layers learning feature representations

Convolutional layers for a network trained on ImageNet.

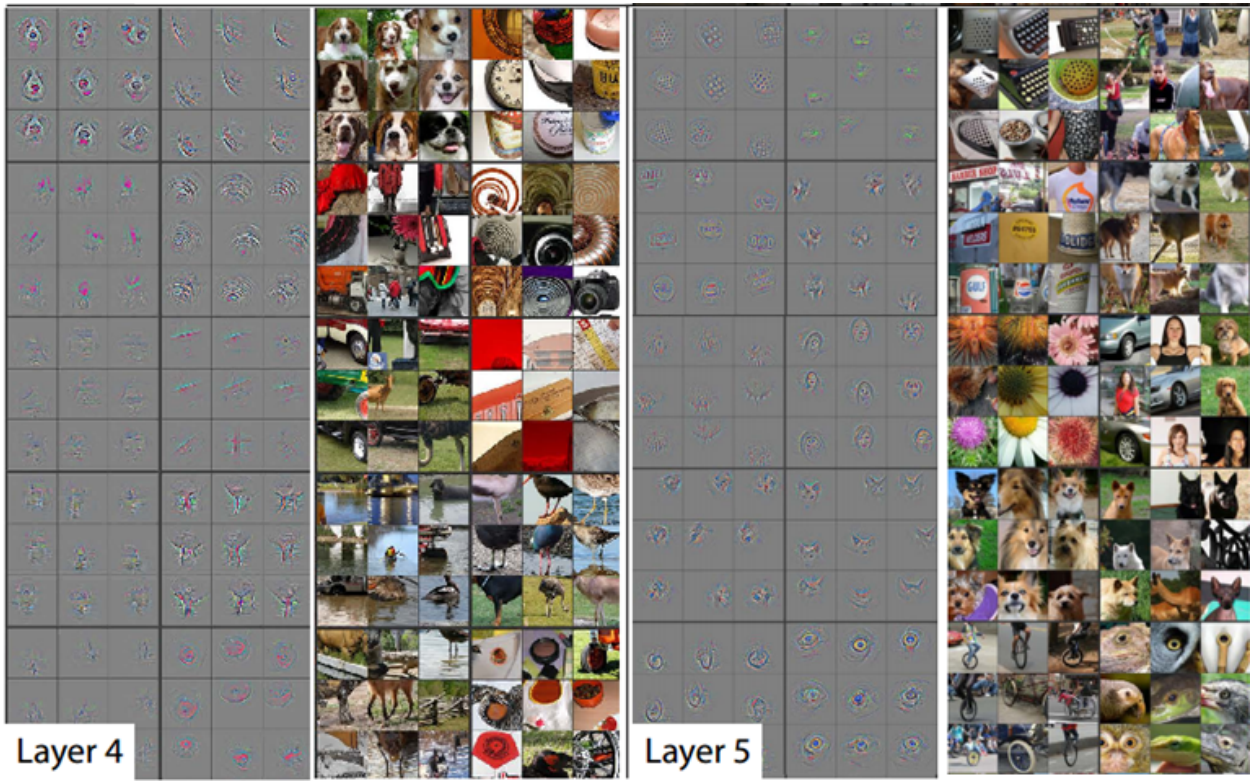


Visualizations for layer 3: Filters alongwith part of images that respond strongly to the filter.

Zeiler and Furgus, 2014.

Convolution layers learning feature representations

Convolutional layers for a network trained on ImageNet.

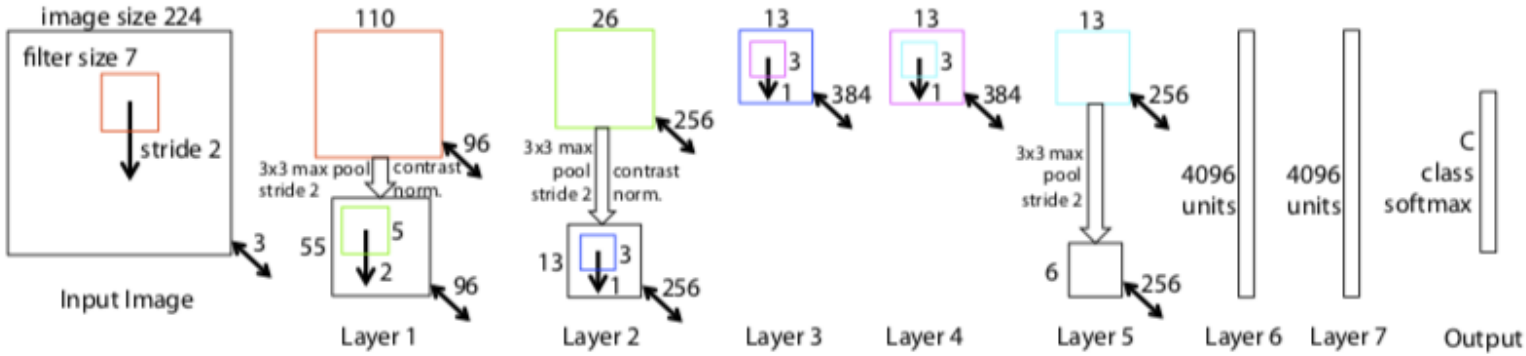


Visualizations for layers 4 and 5: Filters alongwith part of images that respond strongly to the filter.

Zeiler and Furgus, 2014.

Convolution layers learning feature representations

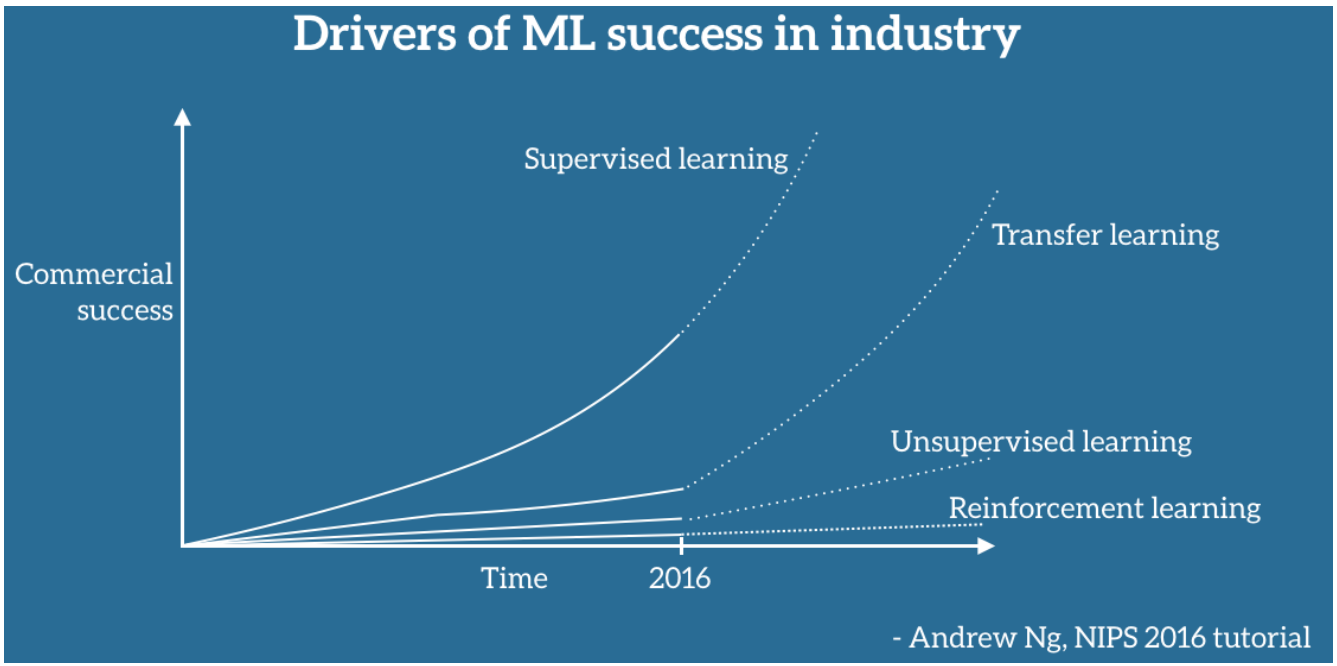
The network used for visualizing the feature maps in the previous slides.



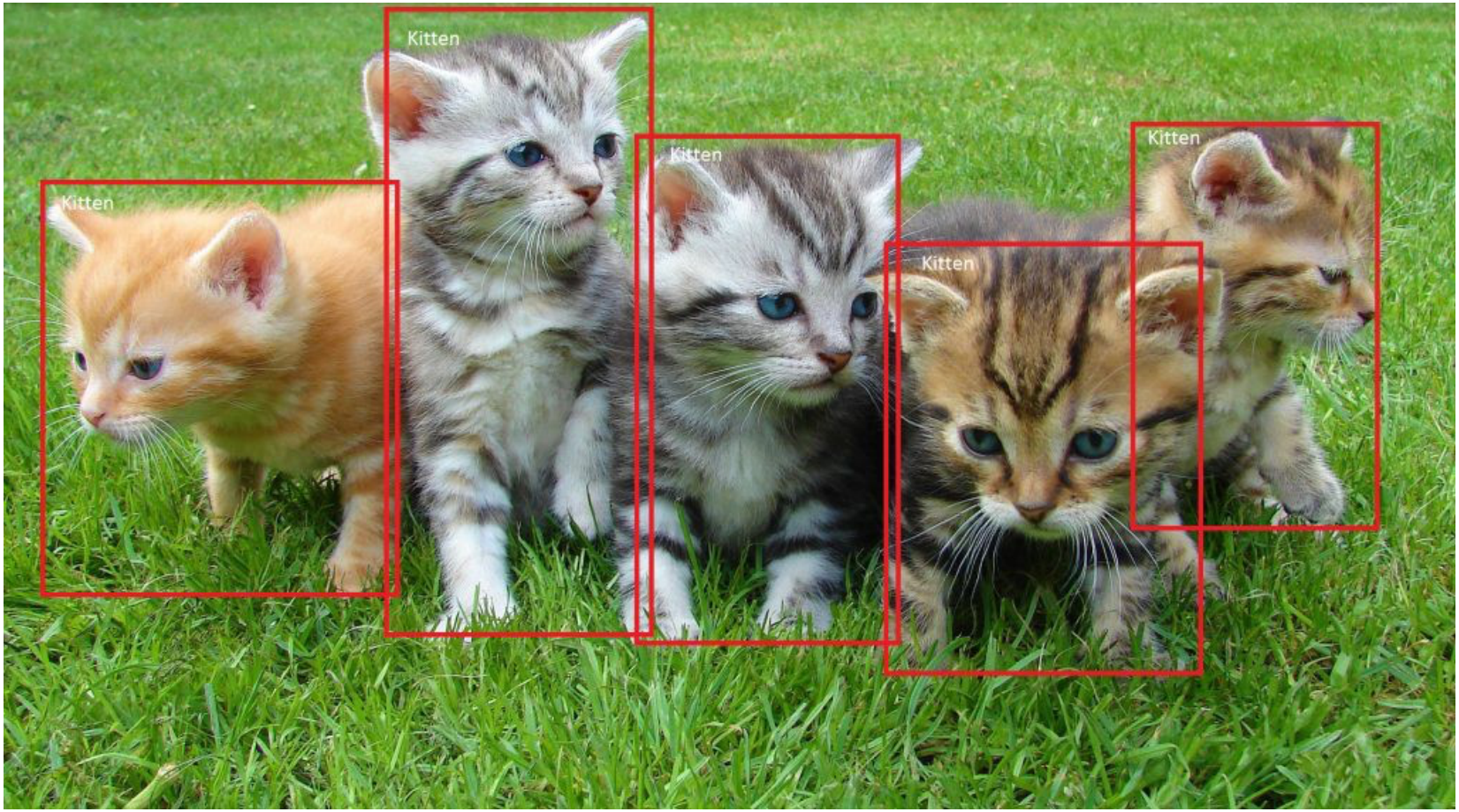
Zeiler and Furgus, 2014.

Transfer learning

- Transferring of knowledge from on model (trained on a large dataset) to another model trained on another dataset exhibiting similar characteristics
- It is always to recommended to use transfer learning in practice
- Very few situations where a convoluational neural network (for computer vision) is trained from scratch.

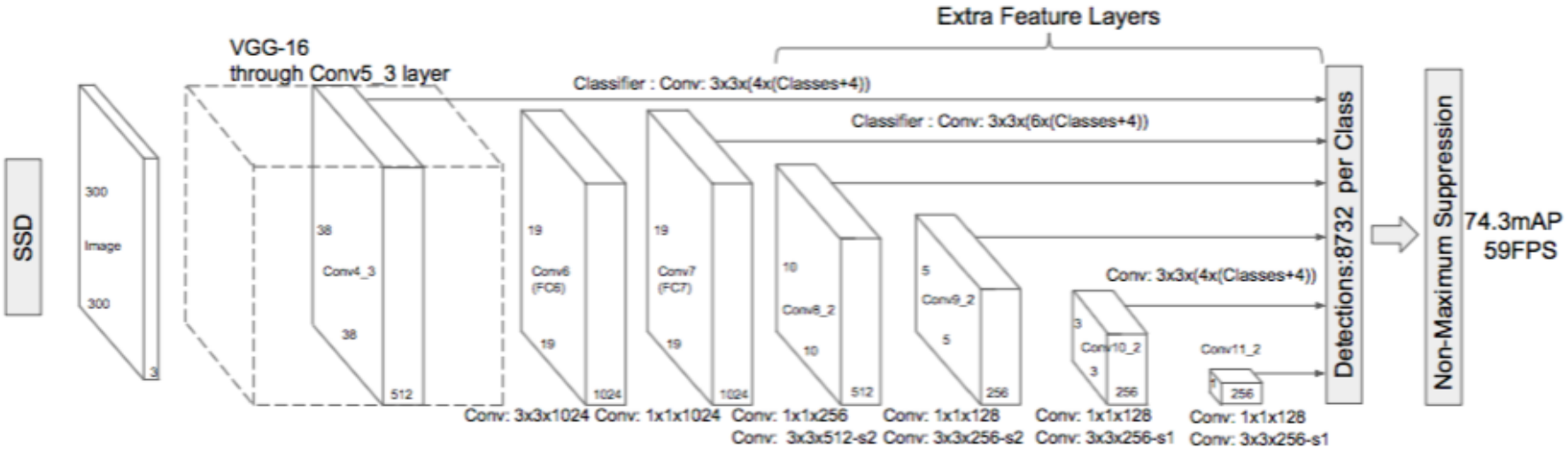


Single Shot Detector

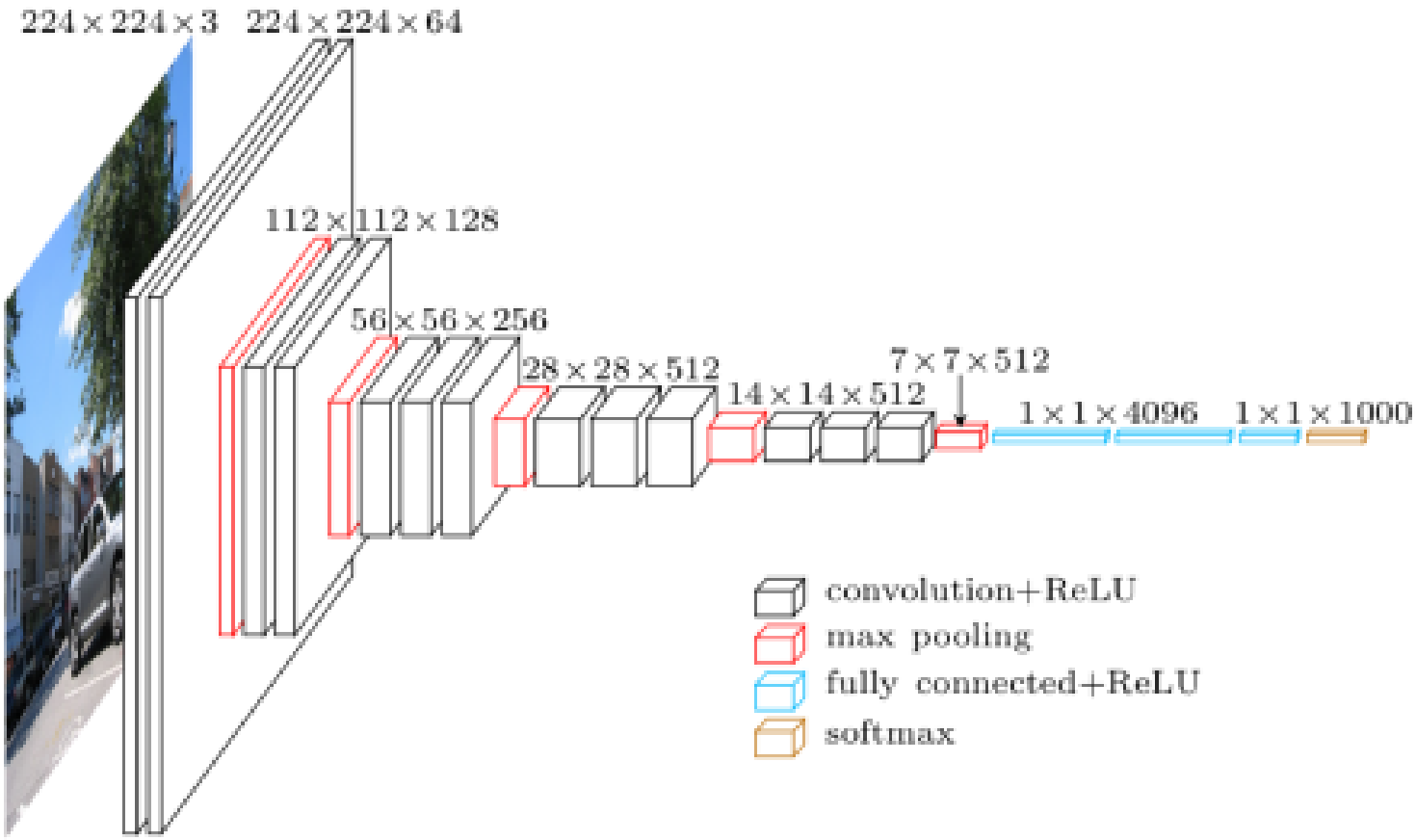


Liu *et al.*, 2016, "SSD: Single Shot Multibox Detector"

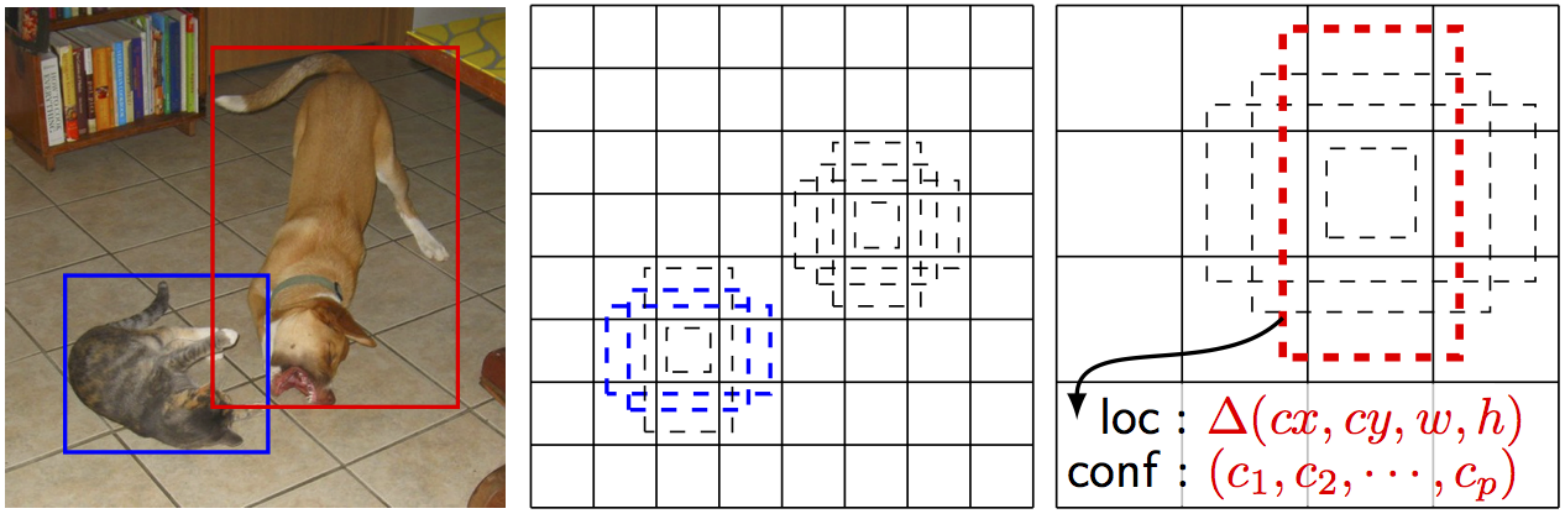
SSD Architecture



SSD feature extraction



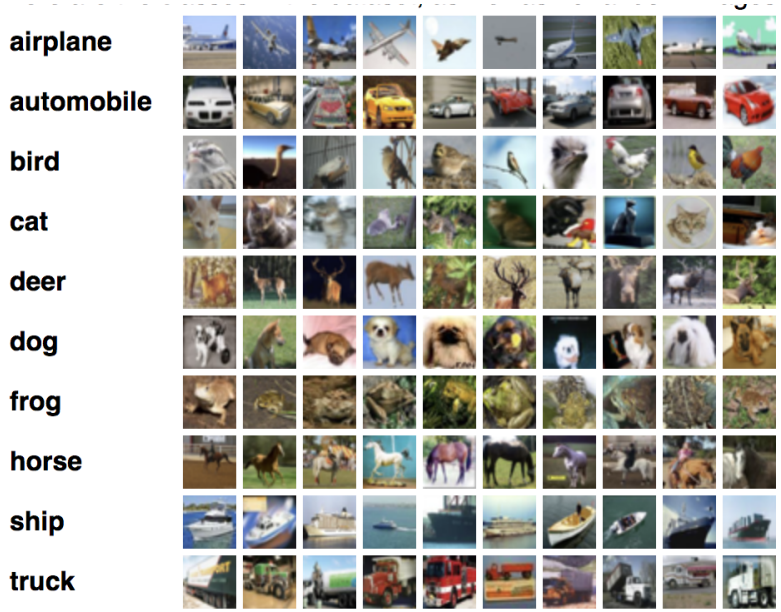
SSD bounding box computation



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

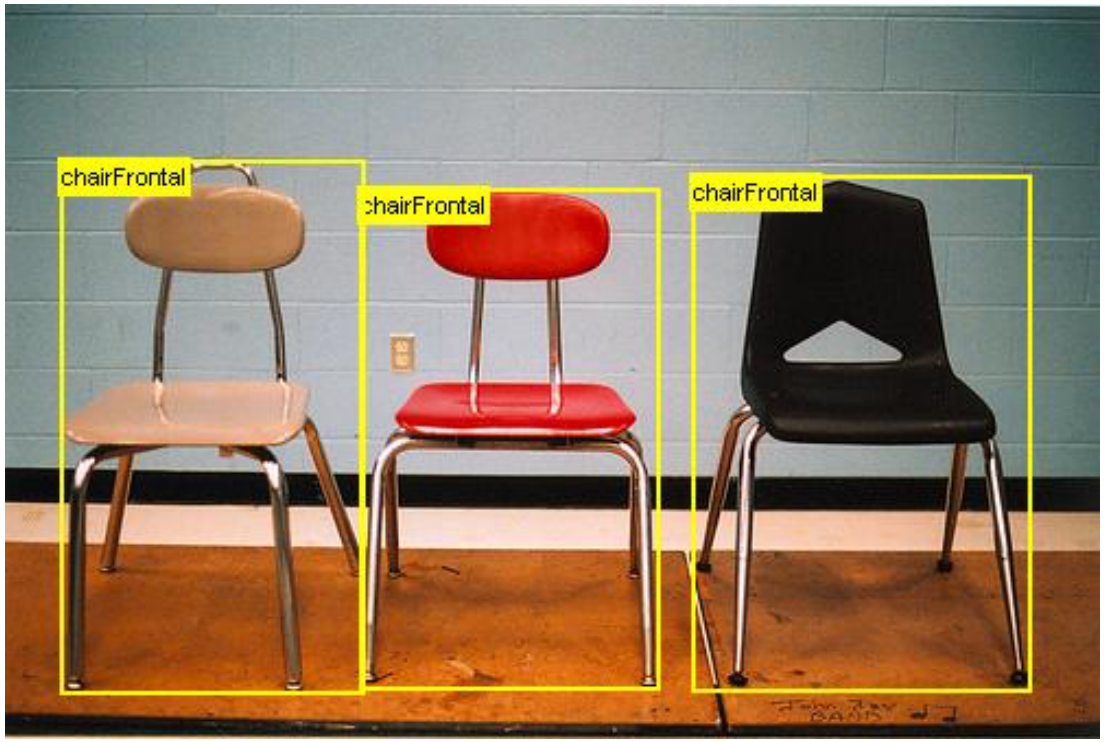
Deep learning and computer vision: success stories

Image classification



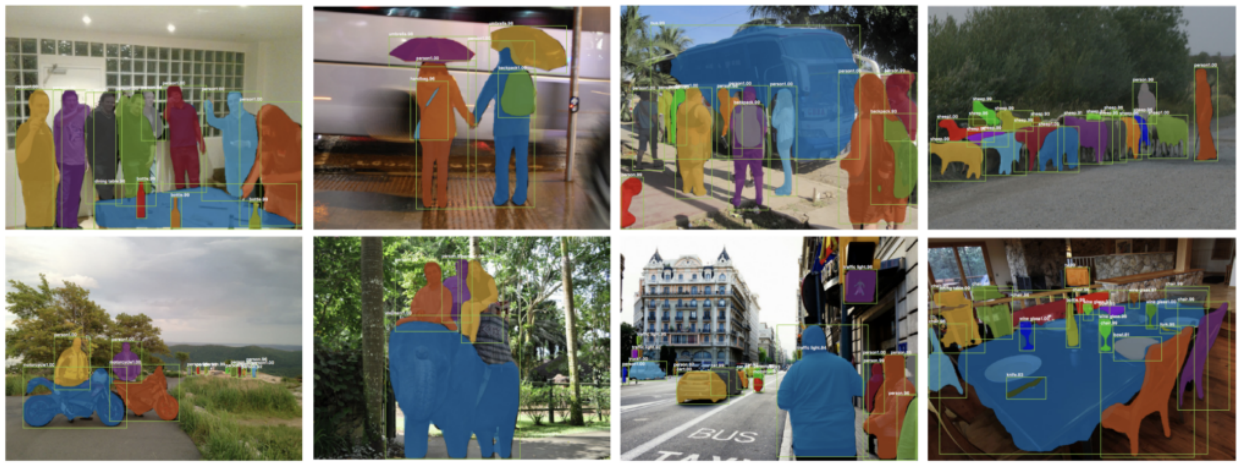
- ImageNet Classification With Deep Convolutional Neural Networks, 2012.
- Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014.
- Going Deeper with Convolutions, 2015.
- Deep Residual Learning for Image Recognition, 2015.

Object detection and localization



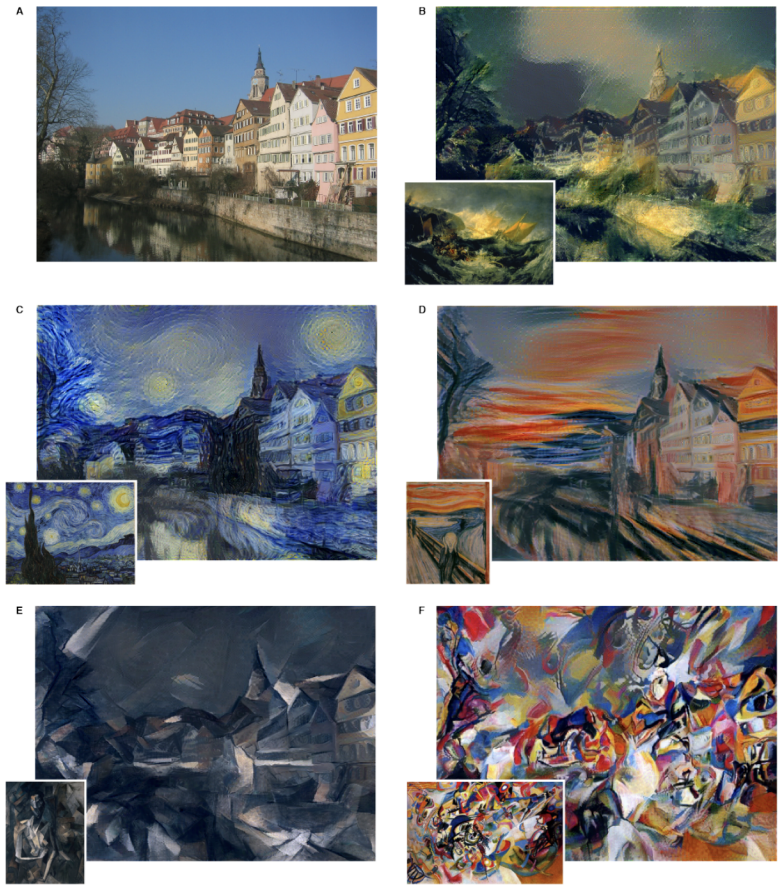
- Selective Search for Object Recognition, 2013.
- Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.
- Fast R-CNN, 2015.

Object segmentation



- Simultaneous Detection and Segmentation, 2014.
- Fully Convolutional Networks for Semantic Segmentation, 2015.
- Hypercolumns for Object Segmentation and Fine-grained Localization, 2015.
- SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, 2016.
- Mask R-CNN, 2017.

Style transfer

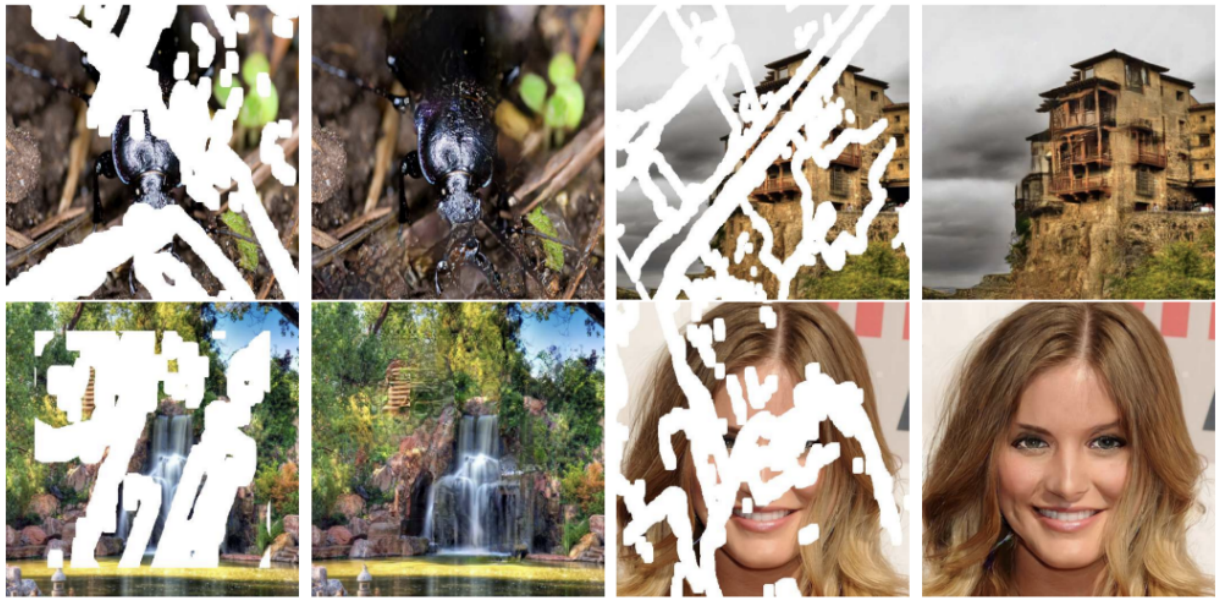


- A Neural Algorithm of Artistic Style, 2015.
- Image Style Transfer Using Convolutional Neural Networks, 2016.

Image Colorization

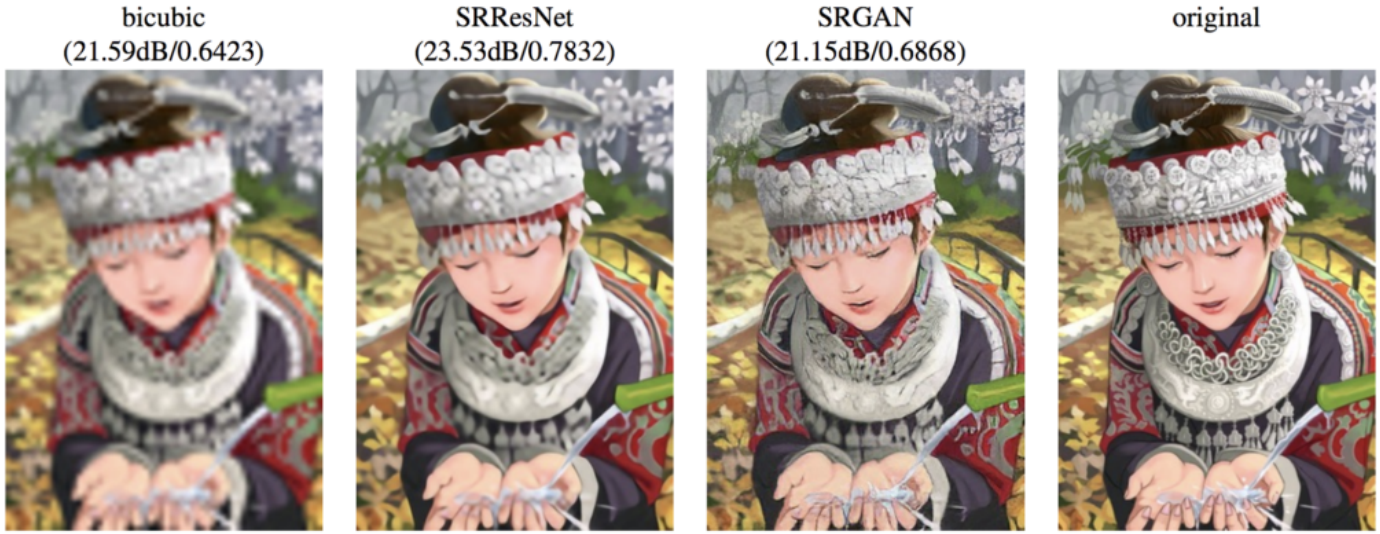
- Colorful Image Colorization, 2016.
- Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification, 2016.
- Deep Colorization, 2016.

Image reconstruction and inpainting



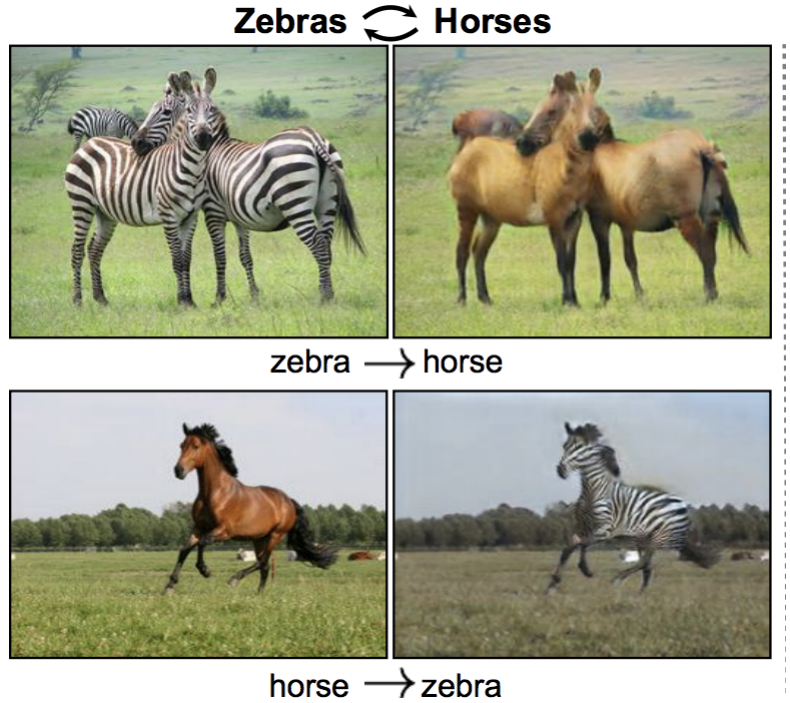
- Pixel Recurrent Neural Networks, 2016.
- Image Inpainting for Irregular Holes Using Partial Convolutions, 2018.
- Highly Scalable Image Reconstruction using Deep Neural Networks with Bandpass Filtering, 2018.

Image superresolution



- Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, 2017.
- Deep Laplacian Pyramid Networks for Fast and Accurate Super-Resolution, 2017.
- Deep Image Prior, 2017.

Image synthesis



- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015.
- Conditional Image Generation with PixelCNN Decoders, 2016.
- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017.

Images and text

- Show and Tell: A Neural Image Caption Generator, 2014.
- Deep Visual-Semantic Alignments for Generating Image Descriptions, 2015.
- AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks, 2017.

Video: summarization, pose estimation and activity recognition

- Deep Recurrent Neural Networks for Human Activity Recognition, 2017.
- 3D Convolutional Neural Networks for Human Action Recognition, 2010.
- Long-term Temporal Convolutions for Action Recognition, 2016.

and many more . . .

Summary

- Feature engineering has given way to architecture engineering
- Know your *calculus*
- Know your *linear algebra*
- Layers, layers, every where
- End-to-end training
- Transfer learning
- Regularization
- Familiarize yourself with one of many deep learning frameworks: PyTorch, TensorFlow, Keras, etc.
  - Learn to write efficient data loaders
- Use scientific method to converge upon a network
  - Ablative study
- Do not forget to compare your work with approaches that *do not* use deep learning

Reading material

Books to develop a theoretical understanding of machine learning and deep learning

- "Deep Learning" by Goodfellow *et al.*
- "Foundations of Machine Learning" by Mohri *et al.*
- "Machine Learning" by Murphy (read it only if you have to)
- "Neural Networks for Pattern Recognition" by Bishop (an old book, but still highly relevant to develop a deeper understanding of artificial neural networks)

If you are in a hurry, the following books can help you use deep learning frameworks for your projects

- "Deep Learning with Python" by Chollet
- "Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Geron

Acknowledgements

- Mittuniversitetet
- The Swedish Foundation for International Cooperation in Research and Higher Education
- Mattias O'Nils
- Special thanks to
  - Hiba Alqaysi
  - Igor Fedorov
  - Najeem Laval
  - Benny Thornberg