CS 1675 Introduction to Machine Learning
Lecture 13

Multilayer neural networks

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Midterm exam

Midterm Thursday, October 19, 2017

- **in-class (75 minutes)**
- **closed book**

- **Covers material from the beginning of the semester including lecture today**

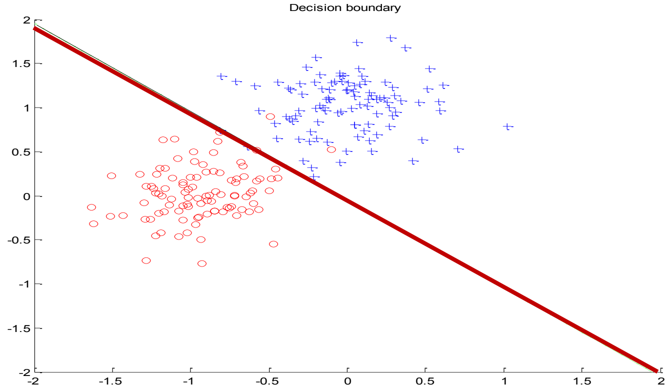
Multilayer neural networks

Or another way of modeling nonlinearities for regression and classification problems

Classification with the linear model.

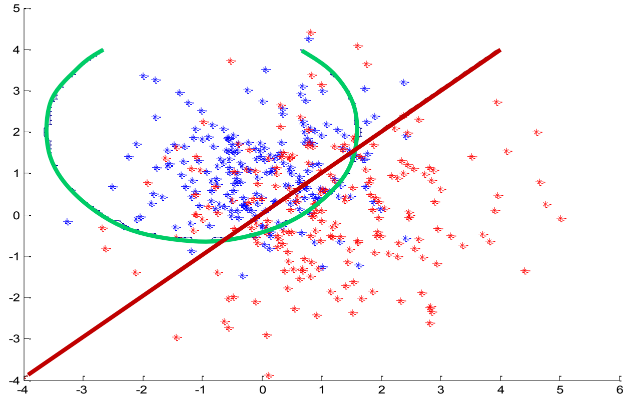
Logistic regression model defines a linear decision boundary

- Example: 2 classes (blue and red points)



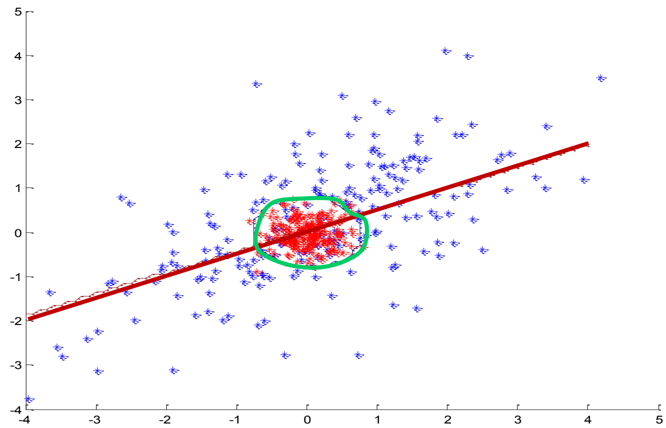
Linear decision boundary

- logistic regression model is not optimal, but not that bad



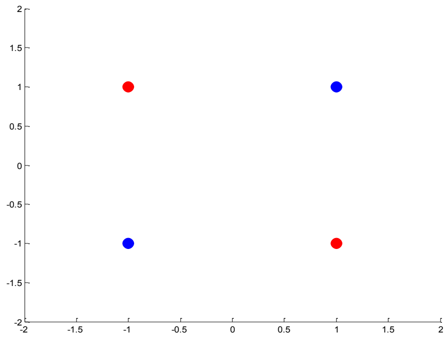
When logistic regression fails?

- Example in which the logistic regression model fails



Limitations of linear units

- Logistic regression does not work for **parity functions**
  - no linear decision boundary exists



Solution: a model of a non-linear decision boundary

Extensions of simple linear units

- **Feature (basis) functions** to model **nonlinearities**

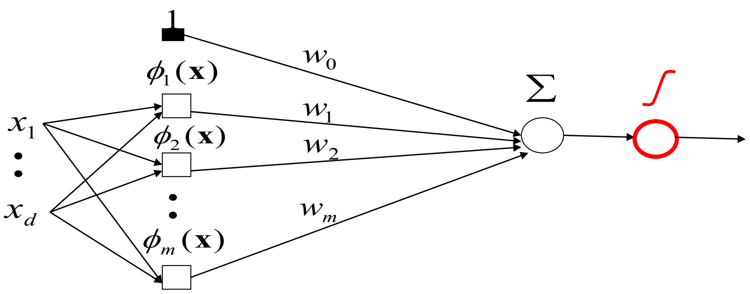
Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

Logistic regression

$$f(\mathbf{x}) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}))$$

$\phi_j(\mathbf{x})$ - an arbitrary function of $\mathbf{x}$



Learning with extended linear units

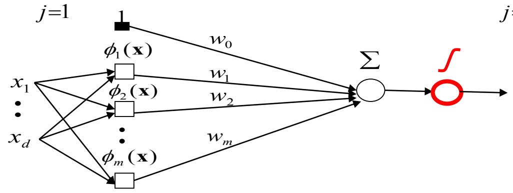
Feature (basis) functions model **nonlinearities**

Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

Logistic regression

$$f(\mathbf{x}) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}))$$



Advantage:

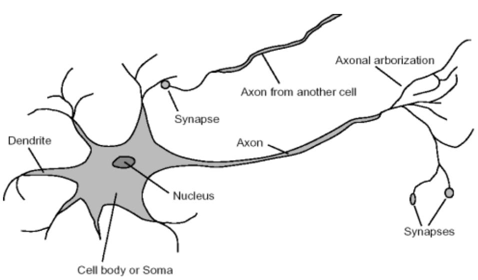
- The same problem as learning of the weights of linear units

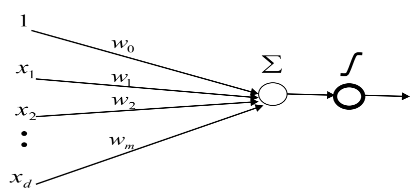
Limitations/problems:

- How to define the right set of basis functions
- Many basis functions → many weights to learn

Multi-layered neural networks

- An alternative way to model **nonlinearities for regression /classification problems**
- **Idea:** Cascade several simple nonlinear models (e.g. logistic units) to approximate nonlinear functions for regression /classification. Learn/adapt these simple models.
- **Motivation:** neuron connections.



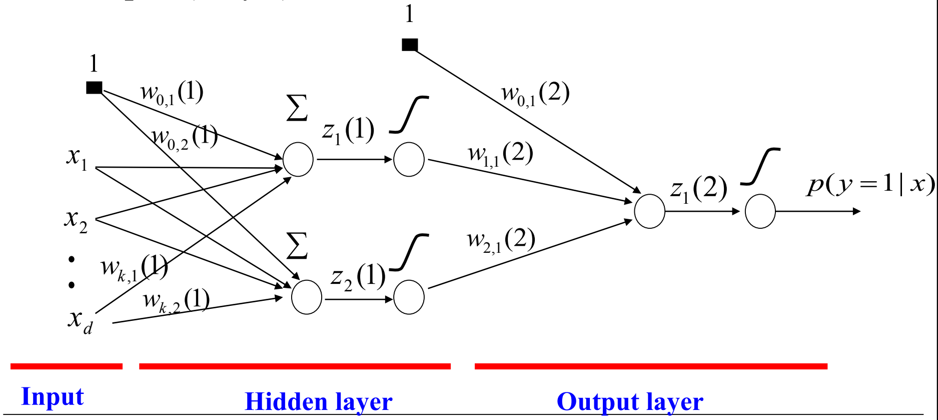


Multilayer neural network

Also called a **multilayer perceptron (MLP)**

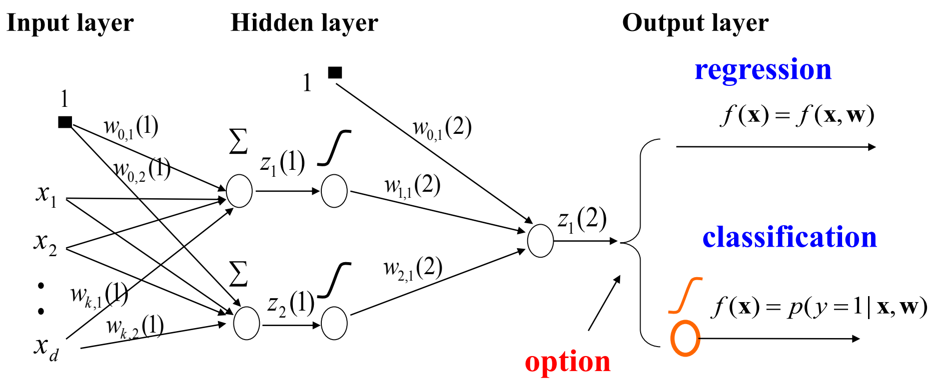
Cascades multiple **logistic regression** units

Example: (2 layer) classifier with non-linear decision boundaries



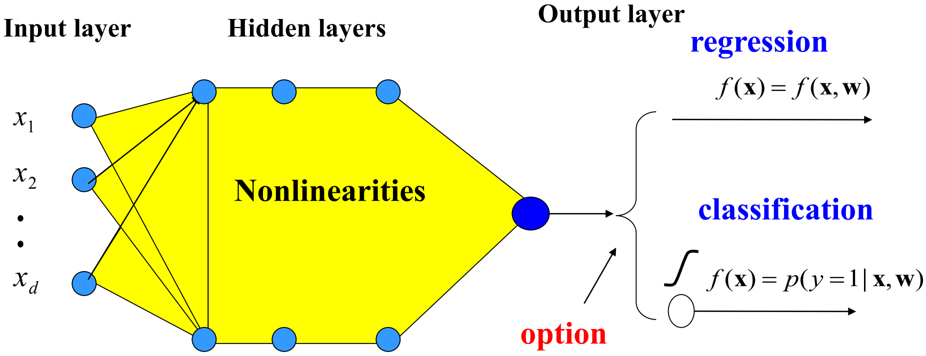
Multilayer neural network

- Models **non-linearity through nonlinear switching units**
- Can be applied to both **regression and binary classification problems**



Multilayer neural network

- **Non-linearities are modeled using multiple hidden nonlinear units (organized in layers)**
- The output layer determines whether it is a **regression or a binary classification problem**

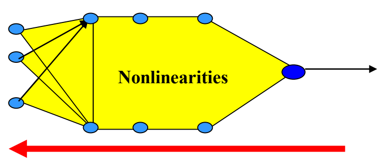


Learning with MLP

- How to learn the parameters of the neural network?
- **Gradient descent algorithm**
  - Weight updates based on the error: $J(D, \mathbf{w})$

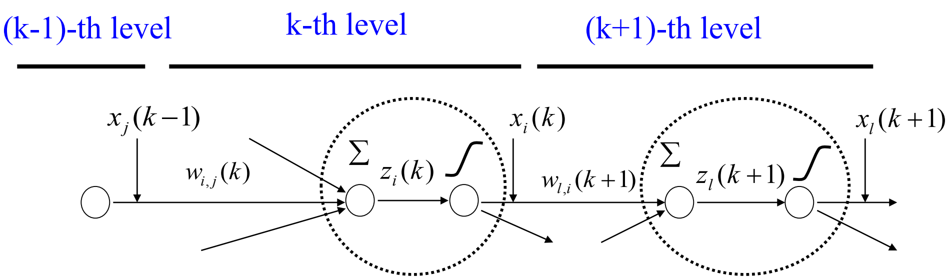
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(D, \mathbf{w})$$

- We need to **compute gradients for weights in all units**
- **Can be computed in one backward sweep through the net !!!**



- The process is called **back-propagation**

Backpropagation



$x_i(k)$ - output of the unit i on level k

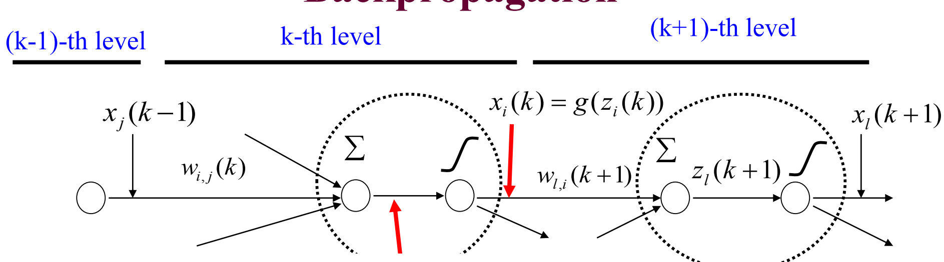
$z_i(k)$ - input to the sigmoid function on level k

$w_{i,j}(k)$ - weight between units j and i on levels (k-1) and k

$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

$$x_i(k) = g(z_i(k))$$

Backpropagation



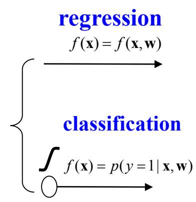
$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Error function**: $J(D, \mathbf{w})$ (online) error where D is a data point
  - **Regression**

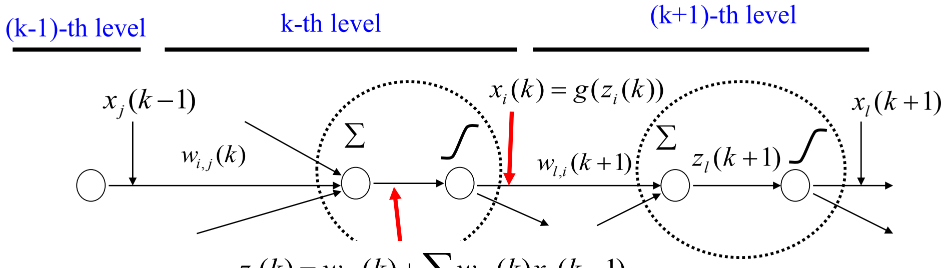
$$J(D, \mathbf{w}) = (y_u - f(\mathbf{x}_u))^2$$

  - **Classification**

$$J(D, \mathbf{w}) = -\log p(y_u \mid f(\mathbf{x}_u))$$



Backpropagation



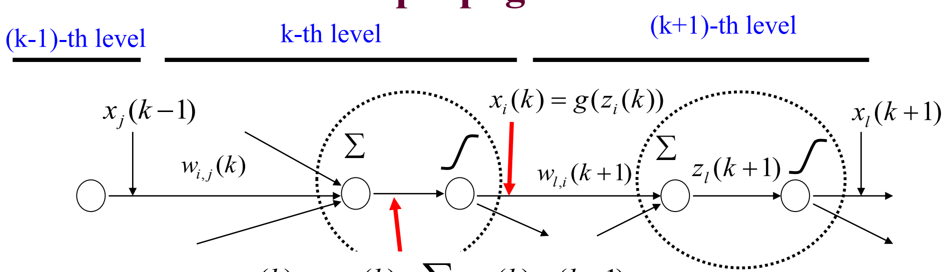
$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Gradient descent:** $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w})$

$$\frac{\partial}{\partial w_{i,j}(k)} J(D,\mathbf{w}) = \frac{\partial J(D,\mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$$

$$\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w}) \qquad x_j(k-1)$$

Backpropagation



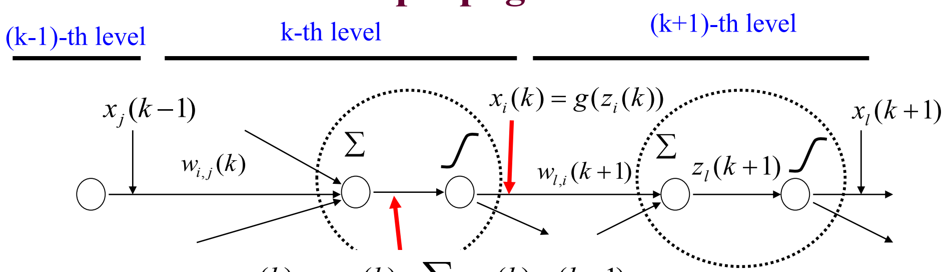
$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Derivation:** $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D,\mathbf{w}) = \frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) * \frac{\partial x_i(k)}{\partial z_i(k)}$

$$\frac{\partial}{\partial x_i(k)} J(D,\mathbf{w}) = \sum_l \underbrace{\frac{\partial}{\partial z_l(k+1)} J(D,\mathbf{w})}_{\delta_l(k+1)} * \underbrace{\frac{\partial z_l(k+1)}{\partial x_i(k)}}_{w_{l,i}(k+1)} \qquad \frac{\partial x_i(k)}{\partial z_i(k)} = x_i(k)(1 - x_i(k))$$

$$\delta_i(k) = \left[\sum_l \delta_l(k+1) w_{l,i}(k+1) \right] x_i(k)(1 - x_i(k))$$

Backpropagation



$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

- **Gradient:**

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \left[\delta_i(k) x_j(k-1)\right]$$

$$\delta_i(k) = \left[\sum_l \delta_l(k+1) w_{l,i}(k+1)\right] x_i(k)(1 - x_i(k))$$

- **Last unit** (is the same as for the regular linear units),

  E.g. for regression:

$$\delta_i(K) = -(y_u - f(\mathbf{x}_u, \mathbf{w}))$$

Backpropagation

Update weight $w_{i,j}(k)$ using data D $\quad D = \{<\mathbf{x}, y>\}$

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D, \mathbf{w})$$

Let $\quad \delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D, \mathbf{w})$

Then: $$\frac{\partial}{\partial w_{i,j}(k)} J(D, \mathbf{w}) = \frac{\partial J(D, \mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$$

S.t. $\delta_i(k)$ is computed from $x_i(k)$ and the next layer $\delta_l(k+1)$

$$\delta_i(k) = \left[\sum_l \delta_l(k+1) w_{l,i}(k+1)\right] x_i(k)(1 - x_i(k))$$

Last unit (is the same as for the regular linear units):

$$\delta_i(K) = -(y_u - f(\mathbf{x}_u, \mathbf{w}))$$

It is the same for the classification with the log-likelihood measure of fit and linear regression with least-squares error!!!

Learning with MLP

- **Online gradient descent algorithm**
  - Weight update:

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J_{\text{online}}(D_u, \mathbf{w})$$

$$\frac{\partial}{\partial w_{i,j}(k)} J_{online}(D_u, \mathbf{w}) = \frac{\partial J_{online}(D_u, \mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$$

$$\boxed{w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)}$$

$x_j(k-1)$ - j-th output of the (k-1) layer
$\delta_i(k)$ - derivative computed via backpropagation
α - a learning rate

Online gradient descent algorithm for MLP

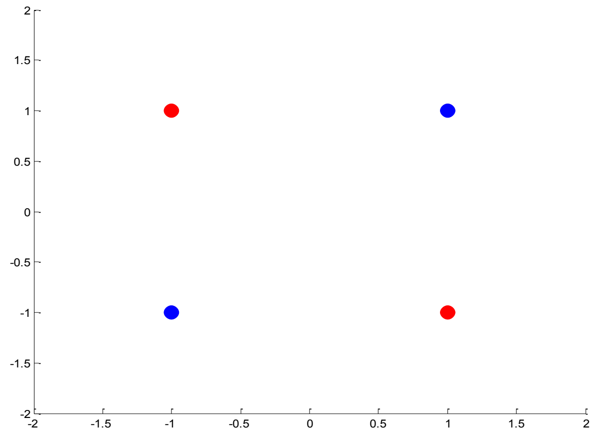
Online-gradient-descent (*D, number of iterations*)
  Initialize all weights $w_{i,j}(k)$
  for *i*=1:1*: number of iterations*
    do **select** a data point D_u=<**x**,*y*> from *D*
      set learning rate α
      compute outputs $x_j(k)$ for each unit
      compute derivatives $\delta_i(k)$ via **backpropagation**
      update all weights (in parallel)

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)$$

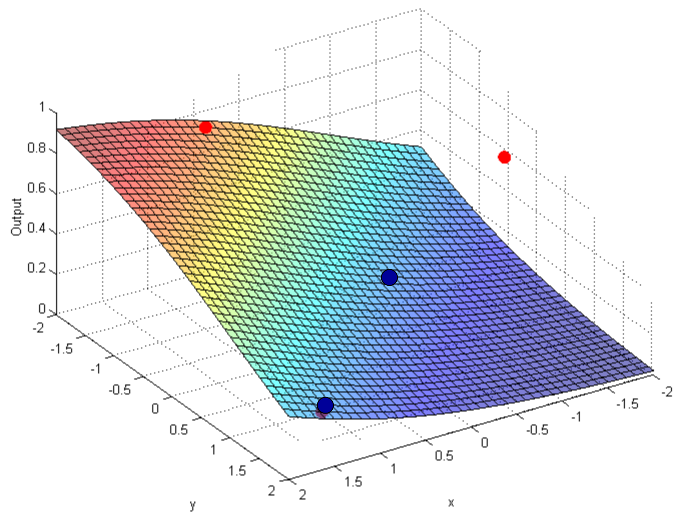
  end for
  return weights **w**

Xor Example.

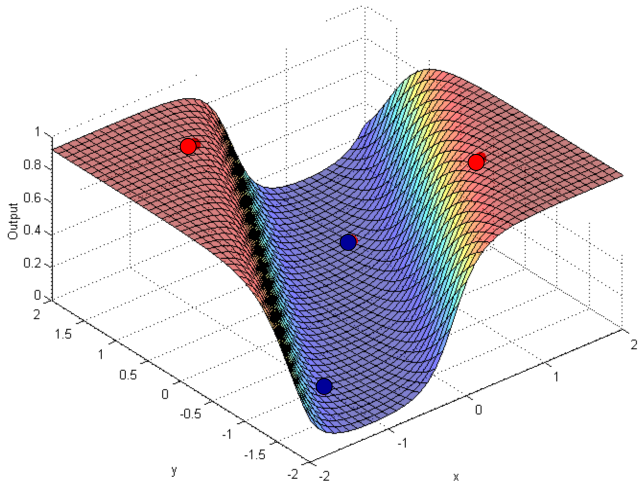
- linear decision boundary does not exist



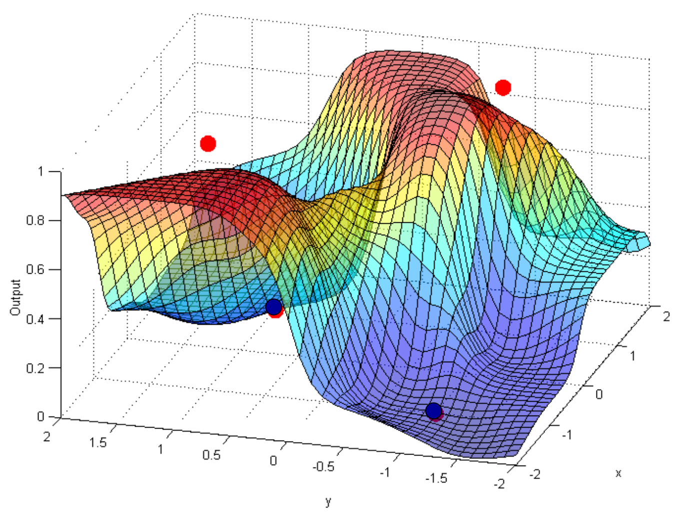
Xor example. Linear unit



Xor example.
Neural network with 2 hidden units



Xor example.
Neural network with 10 hidden units



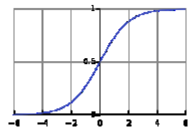
Neural networks

Activation (transfer) functions

- **Determine how inputs are transformed to output**

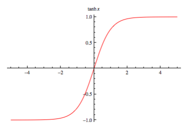
Possible choices of nonlinear transfer functions:

- **Logistic function**



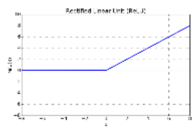
$$f(z)=\frac{1}{1+e^{-z}} \qquad f(z)'=f(z)(1-f(z))$$

- **Hyperbolic tangent**



$$f(z)=\tanh(z)=\frac{2}{1+e^{-2z}}-1 \qquad f(z)'=1-f(z)^2$$

- **Rectified linear function**



$$f(z)=\begin{cases}0 & z<0\\ z & z\geq 0\end{cases}$$

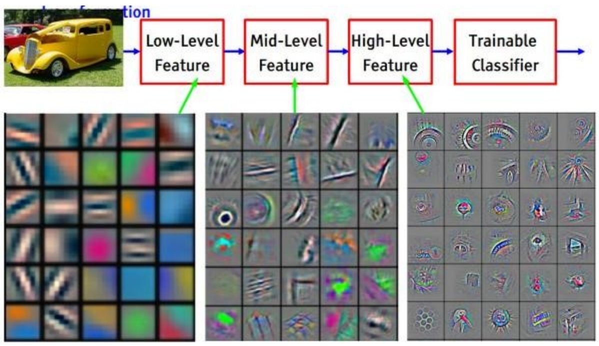
Limitation of standard NNs

Standard NN:

- **do not scale well to high dimensional data (e.g. images)**
  - 100x100 image + 100 hidden units = 1 million parameters.
  - Overfitting;
  - Tremendous requirements of computation and storage.
- **Sensitive to small translation of inputs**
  - Images: objects can have size, slant or position variations
  - Speech: varying speed, pitch or intonation.
- **Ignores the topology of the input**
  - i.e. the input variables can be presented in any order without affecting the outcome of training.
  - However, images or speech has a strong local structure.
    - E.g. pixels nearby are highly correlated.

Deep learning

- **Deep learning**. Machine learning algorithms based on learning multiple levels of representation / abstraction. More than one layer of non-linear feature transformation.

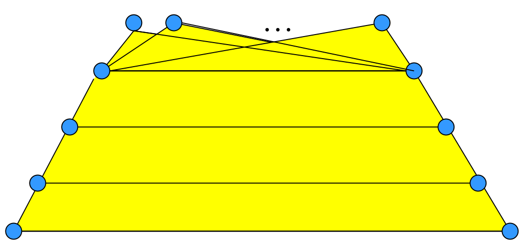


Deep neural networks

Early efforts

- **Optical character recognition** – digits 20x20
  - Automatic sorting of mails
  - 5 layer network with multiple output functions and somewhat restricted topology

10 outputs (0,1,…9)



20x20 = 400 inputs

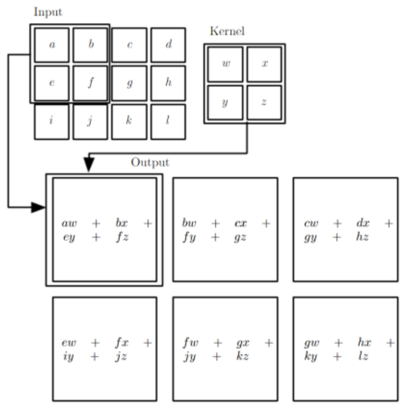
layer	Neurons	Weights
5	10	3000
4	300	1200
3	1200	50000
2	784	3136
1	3136	78400

Convolutional NN

Take advantage of the local structure of the data (image, speech)

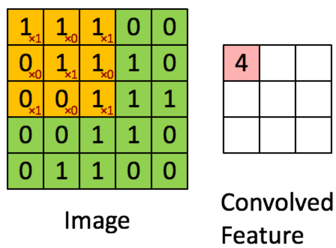
Convolution in Machine Learning

- the **input** array
  - e.g. image pixels.
- a **kernel** or **filter.**
  - a smaller (local) matrix of parameters
- Output: a **feature map**
  - Filter applied to the image



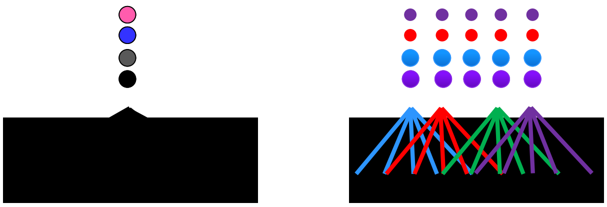
Feature Extraction using Convolution

- The statistics of one part of the image are the same as any other part.
- Meaning that different parts of an image can share the same feature parameters (**kernel**).
- Use this kernel to **convolve** a set of features.
- This is called one feature mapping.



Feature Extraction using Convolution

4 features on full data (image) 4 features on the local data



Fully connected layer / Locally connected layer

9 weights per hidden unit
9 x 4 = 36 weights

5 weights per hidden unit
5 x 4 = 20 weights

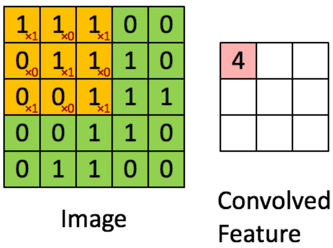
Increased #input, #hidden unit, but fewer weights

Pooling (Subsampling, Down-sampling)

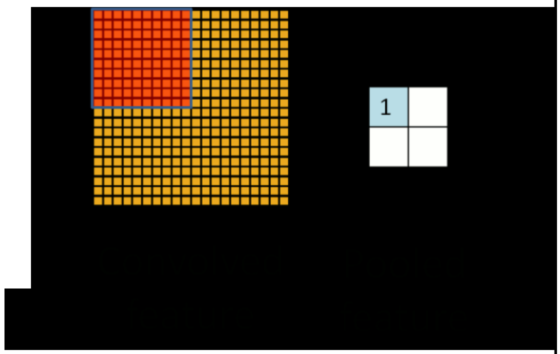
- **Assumption:** Features useful in one region are likely to be useful for other regions.
- To describe a large image, statistics can be **aggregated.**
- For example, one can calculate mean or max of a particular feature over a region.
  - Called **mean pooling**, **max pooling** respectively.
- These summary statistics are much lower in dimension.
- Also can improve results (less-overfitting).

Convolution and Pooling

Convolution

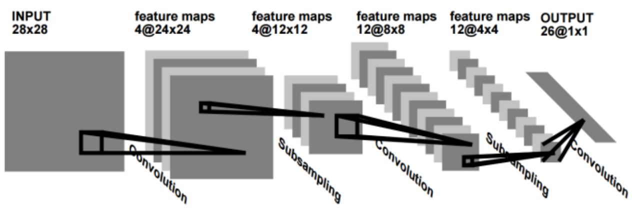


Pooling



Convolutional NN

- CNN = (≥ 1) convolution layer(s) + standard NN
- **One convolution layer is:**
  - Convolution operation + activation function + pooling
- You can view the convolution layer(s) as a feature extractor.
  - Input: raw image pixels, raw time series
  - Output: summarized features.



CNN vs. NN

- NN is sensitive to local distortions of unstructured data.
  - NN can theoretically be trained to be invariant to these distortions, probably resulting in multiple units with identical weights.
  - But such a training task requires a large number of training instances.
- CNN with pooling can be invariant to small translations:
  - Shifts (automatically)
  - Rotation (with extra mechanism)

Object Recognition Task

- ImageNet Data (2009 - 2016)

ImageNet 2012

Data

– Size:
  - Number of images
    – 1.2 million training images
    – 50K validation images
    – 150K testing images
  - Variable image size

– Supervised task
  - Labeled using Amazon's Mechanical Turk

– Categories:
  - 1000 categories (objects)
    – Approximately 1000 in each categor

– RGB pictures

Goal

Provide a probability for different categories that an image can belong to

Object Recognition

- **ImageNet**
  – Achieves state-of-the-art on many object recognition tasks.