Machine learning:
SVM, ANN, ensembles, active learning, practical issues

Agenda

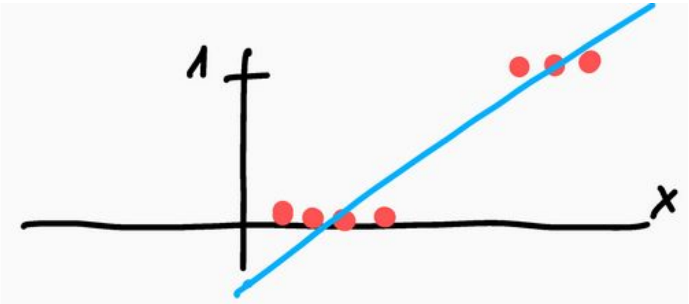
- SVM
- Neural networks
- Ensemble methods
- Active learning
- Practical issues

Agenda

- Logistic regression → SVM
- Neural networks
- Ensemble methods
- Active learning
- Practical issues

Logistic Regression (not yet)

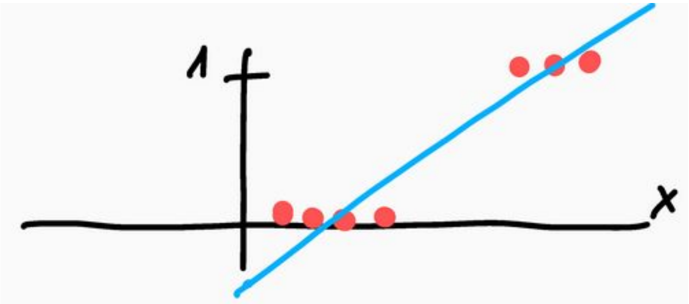
- We could use a linear function to **classify** examples

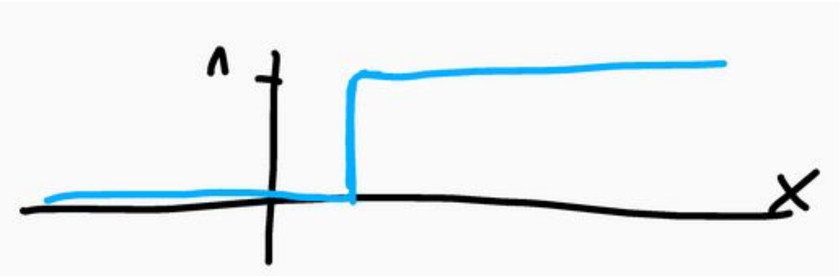


Logistic Regression (not yet)

- We could use a linear function to **classify** examples

  (we would need: a linear function; and a step function for threshold)

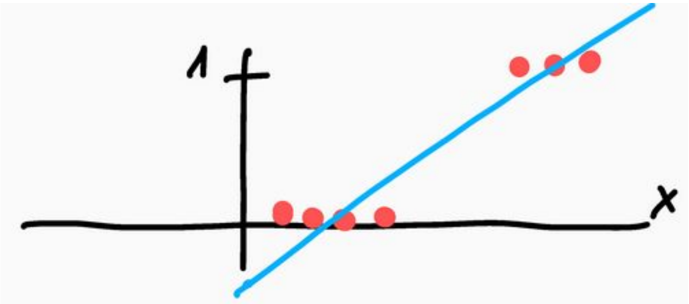


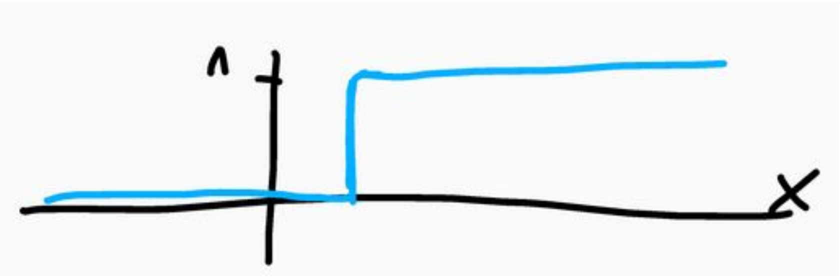


Logistic Regression (not yet)

- We could use a linear function to **classify** examples

  (we would need: a linear function; and a step function for threshold)



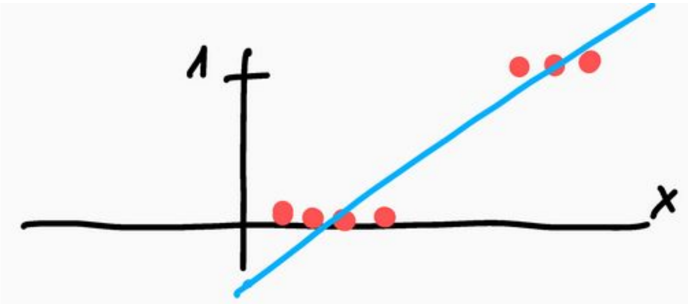


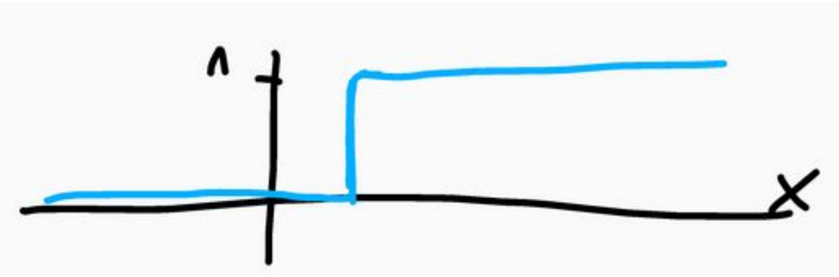
- But this has issues
  - Sensitive to non-important examples in extremes
  - We could optimize both functions together to alleviate this, BUT
    - Step function is not differentiable, so usual optimization approaches cannot be used
    - Values close to the cut-off and far from it have the same value

Logistic Regression (not yet)

- We could use a linear function to **classify** examples

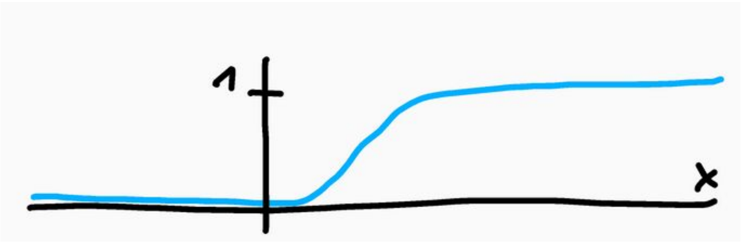
  (we would need: a linear function; and a step function for threshold)





- But this has issues
  - Sensitive to non-important examples in extremes
  - We could optimize both functions together to alleviate this, BUT
    - Step function is not differentiable, so usual optimization approaches cannot be used
    - Values close to the cut-off and far from it have the same value
- There are better functions for such purposes

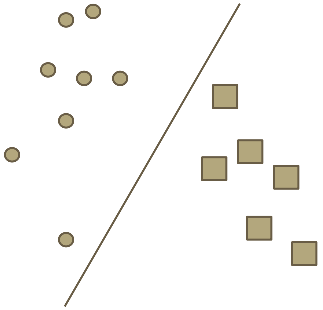
Logistic Regression



- Probabilistic linear classifier
- Logistic (sigmoid) function $$f(x) = \frac{1}{1+e^{-x}}$$
  - Where: $x = w_0 + \sum_i w_i x_i$
  - $f(x) = P(C=1|X)$

- $w_0 + \sum_i w_i x_i = 0$ defines a (linear) decision boundary
  - a hyperplane where $P(C=1|X) = 0.5$ and $P(C=0|X) = 0.5$

in case of two variables:



and $w_0 + \sum_i w_i x_i$ is proportional to the distance from the hyperplane

Logistic Regression

- Learning
  - no closed form solution - optimization, e.g., with gradient descent
  - definition of a cost function (several options);
    - $cost(y', y) = \sum_i -y_i \log(y_i') - (1-y_i) \log(1-y_i')$; y_i', y_i in $\{0,1\}$
  - updating of weights (according to optimization results)

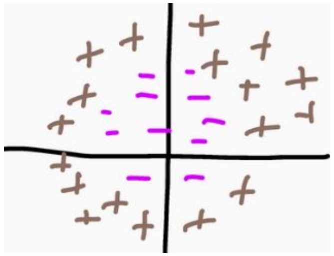
    $$w_j = w_j - \alpha \sum_i (y'_i - y_i)x_{ij}$$

    for all instances, multiple times

- Fast, usually performs well, common choice

Logistic Regression...

- Also non-linear decision boundaries can be modelled



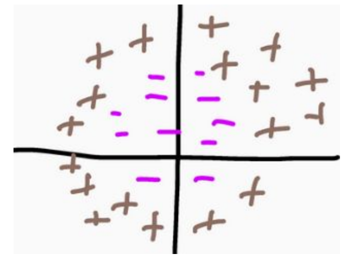
  - We expand the attribute space with synthetic higher-order attributes:

    $y' = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2$

    -1 0 0 1 1 0 gives $x_1^2 + x_2^2 = 1$

Logistic Regression...

- Also non-linear decision boundaries can be modelled



  - We expand the attribute space with synthetic higher-order attributes:

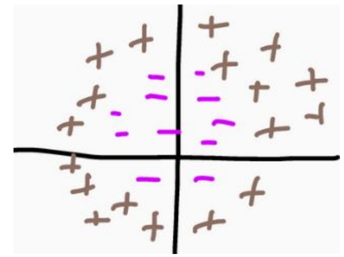
    $$y' = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2$$

    $-1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0$ gives $x_1^2 + x_2^2 = 1$

  - But this also causes problems
    - Computational complexity (many more parameters to learn, additional computing)
    - Overfitting

Logistic Regression...

- Also non-linear decision boundaries can be modelled



  - We expand the attribute space with synthetic higher-order attributes:

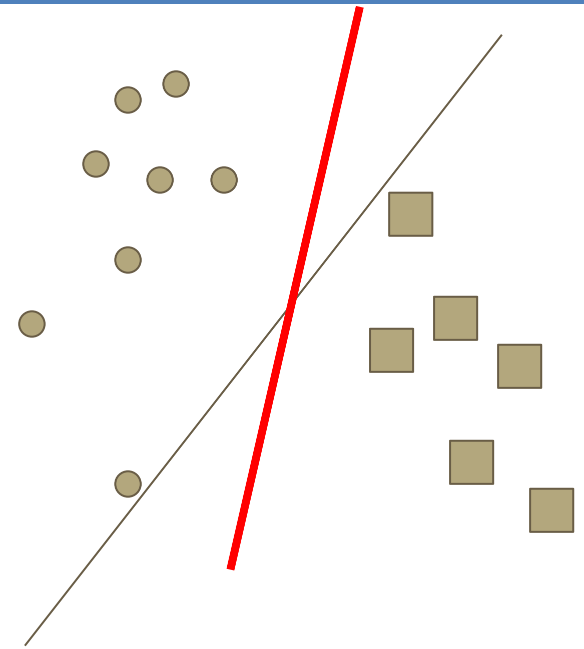
$$y' = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2$$

-1 0 0 1 1 0 gives $x_1^2 + x_2^2 = 1$

  - But this also causes problems
    - Computational complexity (many more parameters to learn, additional computing)
    - Overfitting

- SVM tackles these (max. margin and support vectors, kernel trick)

SVM - max. margin



- Linear binary classifier (not probabilistic)
- Model (linear, *hyperplane*) for separation of data by using the *maximal margin* principle (max margin: robustness) based on *support vectors* (SV: stability)

- Learning: maximal margin (optimal hyperplane) optimization problem
- Soft margin to allow misclassifications
  - Distance on the wrong side: ξ_i
  - Parameter *C* (misclassification cost) - set with experimentation!
  - Penalty: $C \cdot \xi^r_i$

SVM - kernel trick

- Use of higher dimensions for linearly non-separable data
  - https://www.youtube.com/watch?v=3liCbRZPrZA
  - https://www.youtube.com/watch?v=9NrALgHFwTo

- Learning (optimization) involves dot products in the term to maximize:

$$L_D = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \overline{X_i} \cdot \overline{X_j}$$

Dot product of training data points is needed, (not feature values) ~similarity

We can avoid representing W

classification too:

$$F(\overline{Z}) = \text{sign}\{\overline{W} \cdot \overline{Z} + b\} = \text{sign}\{(\sum_{i=1}^{n} \lambda_i y_i \overline{X_i} \cdot \overline{Z}) + b\}$$

SVM - kernel trick, here it is

- We do not need the feature values, just dot products
- Transformation to another (higher dimensional) feature space would mean:

  $\Phi(x_i) \cdot \Phi(x_i)$

  calculation of transformations, then the lengthy dot products...

- Instead, we can use a function such that: $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_i)$
  - And $K(x_i, x_j)$ is in original space!

SVM - kernel trick, here it is

- We do not need the feature values, just dot products
- Transformation to another (higher dimensional) feature space would mean:

  $\Phi(x_i) \cdot \Phi(x_i)$

  calculation of transformations, then the lengthy dot products...

- Instead, we can use a function such that: $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_i)$
  - And $K(x_i, x_j)$ is in original space!
    - EXAMPLE

$$\Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \qquad K(x,z) = (x \cdot z)^2$$

$$\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$x = (x_1, x_2)$$

$$z = (z_1, z_2)$$

$$\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\Phi(z) = (z_1^2, \sqrt{2}z_1z_2, z_2^2)$$

$$\Phi(x) \cdot \Phi(z) = x_1^2 z_1^2 + 2x_1x_2z_1z_2 + x_2^2 z_2^2$$

$$\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$x = (x_1, x_2)$$

$$z = (z_1, z_2)$$

$$\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\Phi(z) = (z_1^2, \sqrt{2}z_1z_2, z_2^2)$$

$$\Phi(x) \cdot \Phi(z) = x_1^2 z_1^2 + 2x_1x_2z_1z_2 + x_2^2 z_2^2$$

$$K(x,z) = (x \cdot z)^2$$

$$x = (x_1, x_2)$$

$$z = (z_1, z_2)$$

$$K(x,z) = (x \cdot z)^2 = (x_1z_1 + x_2z_2)^2 =$$

$$= x_1^2 z_1^2 + 2x_1z_1x_2z_2 + x_2^2 z_2^2$$

$$\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$x = (x_1, x_2)$

$z = (z_1, z_2)$

$$\Phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\Phi(z) = (z_1^2, \sqrt{2}z_1z_2, z_2^2)$$

$$\Phi(x) \cdot \Phi(z) = x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2$$

$x = (1, 2)$

$z = (4, 5)$

$$\Phi(x) = (1\cdot 1, \sqrt{2}\cdot 1\cdot 2, 2\cdot 2) = (1, \sqrt{2}\cdot 2, 4)$$

$$\Phi(z) = (4\cdot 4, \sqrt{2}\cdot 4\cdot 5, 5\cdot 5) = (16, \sqrt{2}\cdot 20, 25)$$

$$\Phi(x) \cdot \Phi(z) = 1\cdot 16 + \sqrt{2}\cdot 2\cdot \sqrt{2}\cdot 20 + 4\cdot 25 = 196$$

$$K(x,z) = (x \cdot z)^2$$

$x = (x_1, x_2)$

$z = (z_1, z_2)$

$$K(x,z) = (x \cdot z)^2 = (x_1z_1 + x_2z_2)^2 =$$

$$= x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2$$

$x = (1, 2)$

$z = (4, 5)$

$$K(x,z) = (1\cdot 4 + 2\cdot 5)^2 = 14\cdot 14 = 196$$

SVM - kernel trick, here it is

- We do not need the feature values, just dot products
- Transformation to another (higher dimensional) feature space would mean:

  $\Phi(x_i) \cdot \Phi(x_i)$

  calculation of transformations, then the lengthy dot products...

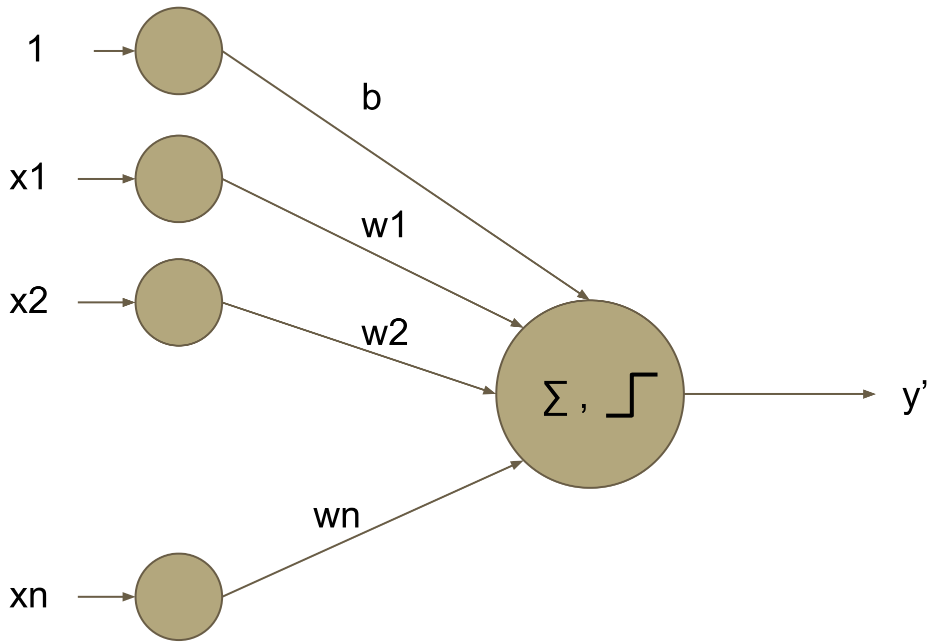
- Instead, we can use a function such that: $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_i)$
  - And $K(x_i, x_j)$ is in original space!
    - EXAMPLE
  - We can only calculate kernels (polynomial, Gaussian RBF, ...)
  - The mapping Φ can now be only implicitly used
  - Simetric, positive semi-definite; similarity ; even for strings, graphs

SVM - practical note

- It is important to normalize the attributes!
    - otherwise the ones with large values dominate in influence

Perceptron

- Inspired by (simulation of) the human nervous system



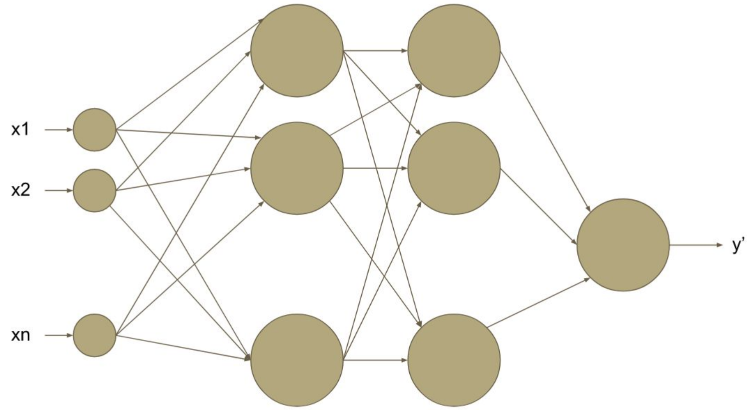
$$y' = sign(\sum_i w_i x_i + b)$$

Learning (iterative process):

- Initialize weights
- For each training item (**x**,**y**)
  - compute y’
  - update all weights
    $w_i' = w_i + \alpha(y_i - y'_i)x_i$
- Until convergence

- Can learn (converge) in linearly separable situations
- Finds (some!) linear separation

Neural networks with hidden layers



- Very powerful in capturing arbitrary functions
  - having non-linear activation functions; careful selection to facilitate learning
- Automatic generation of (higher-level) features!
  - last level is similar to logreg on generated (relevant) high-level features, not all quadratic, cubic, … which easily go into hundreds of thousands.
- Drawbacks
  - computationally demanding learning (recently alleviated)
  - more layers - more power - more prone to overfitting
  - black-box models

Neural network - use (forward propagation)

Use of a neural network

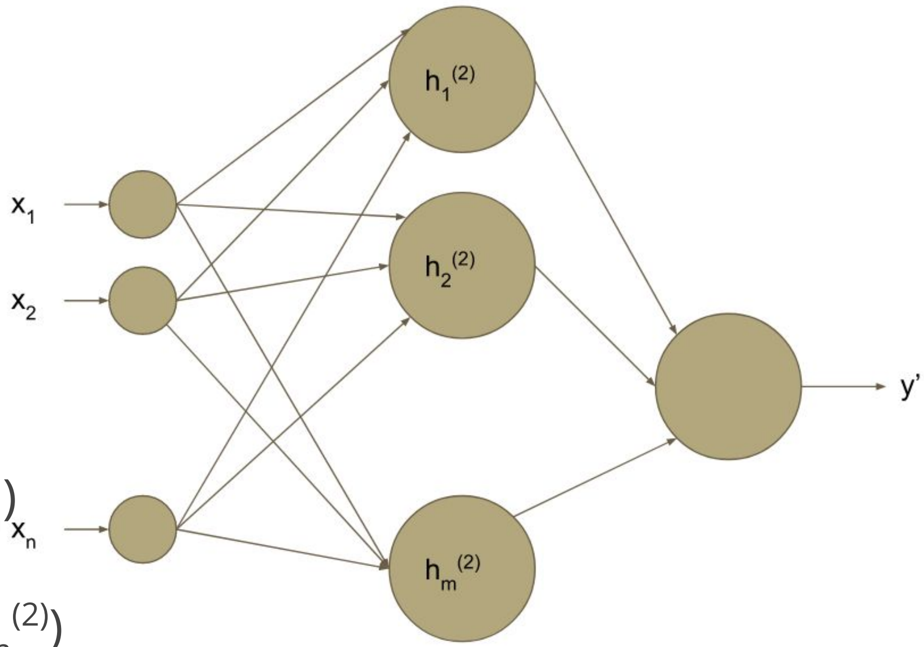
$h_1^{(2)} = g(w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + \dots + w_{n1}^{(1)}x_n)$

$h_2^{(2)} = g(w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + \dots + w_{n2}^{(1)}x_n)$

…

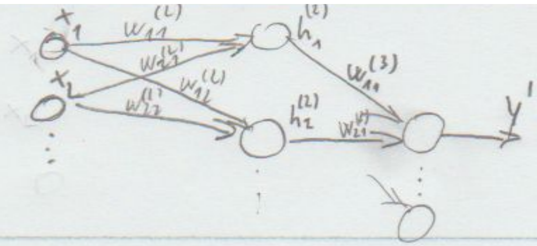
$h_m^{(2)} = g(w_{1m}^{(1)}x_1 + w_{2m}^{(1)}x_2 + \dots + w_{nm}^{(1)}x_n)$

$y' = g(w_{11}^{(2)}h_1^{(2)} + w_{21}^{(2)}h_2^{(2)} + \dots + w_{m1}^{(2)}h_m^{(2)})$



Neural networks - learning

- Two things to learn:
  - Structure: expert knowledge and experimentation
  - Parameters/weights : backpropagation (and other optimization approaches)
    - Gradient descent (consequence: step → sigmoid; error 0/1 → $(y-y')^2$)
      - Optimum can be local !
      - Weights must be initialized to random values
    - Can be done in a batch or online mode
      - One epoch : one learning iteration over training data
    - Overfitting problem - stop on check with holdout, ...
    - Computationally demanding
    - EXAMPLE



$x_j^{(l)}$ = input to node j at l

$w_{ij}^{(l)}$ = weight from i in $l-1$ to j in l

$h_j^{(l)}$ = output of hidden node j at level l.

$$E = \sum_{k \in K} \frac{1}{2} (y_k' - y_k)^2$$

$$\sigma_{(z)}' = \sigma_{(z)} (1 - \sigma_{(z)})$$

I) weights for the output layer

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \frac{1}{2} \sum_{k \in K} (y_k' - y_k)^2 = \rightarrow (y_k' - y_k)^2 + (y_{k'}' - y_{k'})^2 + \ldots$$

$$= \frac{\partial}{\partial w_{jk}} \frac{1}{2} (y_k' - y_k)^2 =$$

CONST.

$$= (y_k' - y_k) \frac{\partial}{\partial w_{jk}} y_k' = (y_k' - y_k)\, \sigma(x_k) \, (1 - \sigma(x_k)) \frac{\partial x_k}{\partial w_{jk}} =$$

$\rightarrow w_{1k} h_1 + w_{2k} h_2 + \ldots w_{jk} h_j + \ldots$

$$= (y_k' - y_k)\, y_k' (1 - y_k') \cdot h_j$$

II) weights for hidden layers

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \frac{1}{2} \sum_{k \in K} (y_k' - y_k)^2 = \sum_{k \in K} (y_k' - y_k) \frac{\partial}{\partial w_{ij}} y_k' =$$

$\rightarrow w_{1k} h_1 + w_{2k} h_2 + \dots + w_{jk} h_j + \dots$

$$= \sum_{k \in K} (y_k' - y_k)\, \sigma(x_k) \left(1 - \sigma(x_k)\right) \cdot \frac{\partial x_k}{\partial w_{ij}} =$$

$$= \sum_{k \in K} (y_k' - y_k)\, y_k' (1 - y_k') \cdot \frac{\partial x_k}{\partial h_j} \cdot \frac{\partial h_j}{\partial w_{ij}} =$$

$h_j = \sigma(x_j)$

$\rightarrow x_1 \cdot w_{1j} + x_2 w_{2j} + \dots x_i w_{ij} + \dots$

$$= \sum_{k \in K} (y_k' - y_k)\, y_k' (1 - y_k') \cdot w_{jk} \cdot \sigma(x_j) \cdot \left(1 - \sigma(x_j)\right) \cdot \frac{\partial x_j}{\partial w_{ij}} =$$

$$= \sum_{k \in K} (y_k' - y_k)\, y_k' (1 - y_k')\, w_{jk}\, h_j (1 - h_j)\, x_i^{(1)}$$

$$W = W - \alpha \frac{\partial E}{\partial W}$$

Neural networks - learning of the structure

- Fully connected
  - Number of layers, number of nodes in layers
  - Experiment & select
- Not fully connected
  - Optimal brain damage
    - Create a fully connected ANN
    - Remove a connection (or a node)
    - Retrain & test
    - If not worse, keep and repeat
  - Constructive approaches: sequential adding of units (e.g., to tackle misclassified examples)
- ! Very large networks can memorize all the training data
- Specific structures: recurrent (internal state, dynamics, memory), convolutional, ...

Neural networks - multiple classes