Machine Learning for Computational Linguistics

Intorduction to artificial neural networks

Çağrı Çöltekin

University of Tübingen
Seminar für Sprachwissenschaft

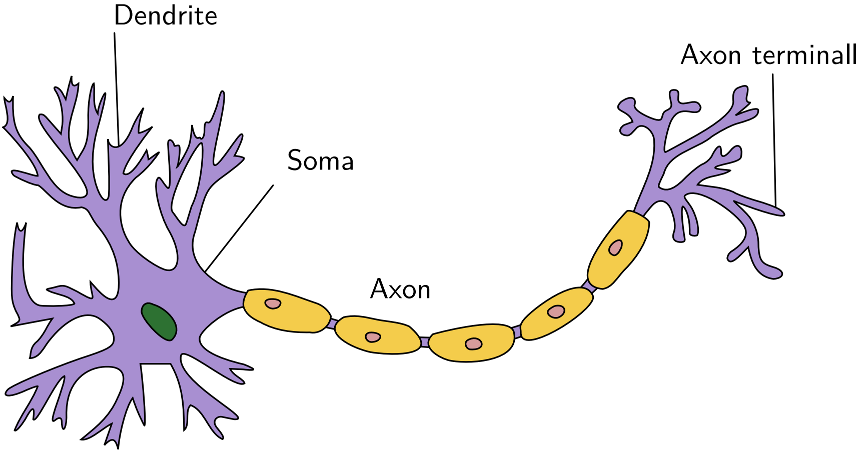
June 7, 2016

Practical stuff

- Reminder: proposal due June 13 (next week)
- Homework 2: FAQ
  - Training/test split is only necessary for the last question. In other questions use training data as the test set
  - While calculating precision, recall, f-measure, make sure that 'German' is your positive class
  - You are encouraged to use a (public) version control system like GitHub. If you do so, send me the repository address only (no need to send the files). If you use private repositories, make sure that I can access them.
  - You can submit your homework until Wednesday morning 10:00.

The biological neuron

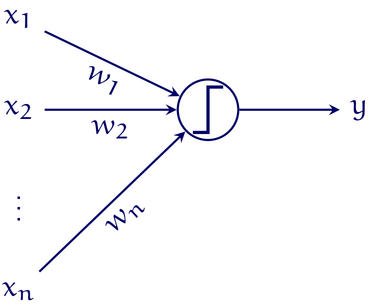
(showing a picture of a real neuron is mandatory in every ANN lecture)



Artificial neural networks (ANNs)

- Artificial neural networks are machine learning models *inspired* by the biological neural networks.
- Although some strong claims have been made about the link between the two, for our purposes, ANNs are just another statistical method in machine learning
- The founding blocks of ANNs are simple units (like biological neurons) that carry out simple calculations in parallel
- ANNs have many similarities to the linear models we discussed so far, but allow non-linearities that are often useful in practice
- The recent 'deep learning' methods are variations of ANN architectures.

The perceptron

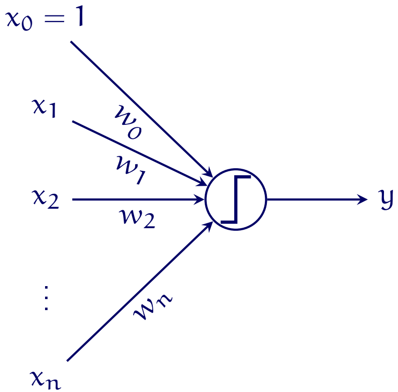


$$y = f\left(\sum_i^n w_i x_i\right)$$

where

$$f(x) = \begin{cases} +1 & \text{if} \quad \sum_i^n w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

The perceptron



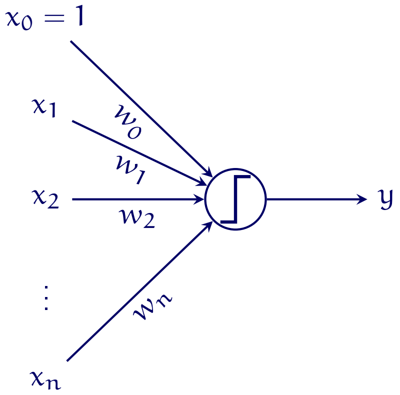
$$y = f\left(\sum_i^n w_i x_i\right)$$

where

$$f(x) = \begin{cases} +1 & \text{if} \quad \sum_i^n w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Similar to the *intercept* in linear models, an additional input x_0 which is always set to one is often used (called bias in ANN literature.)

The perceptron: in plain words

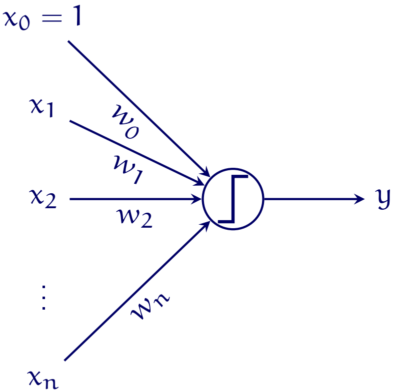


- Sum all input x_i weighted with corresponding weight w_i
- Pass it through a threshold function
- Classify the input a

positive: if the perceptron fires (the sum is larger than 0),

negative: otherwise

The perceptron: in plain words



- Sum all input x_i weighted with corresponding weight w_i
- Pass it through a threshold function
- Classify the input a

positive if the perceptron fires (the sum is larger than 0),
negative otherwise

The perceptron can solve *linearly separable* classification problems.

The perceptron algorithm

- For correctly classified examples, we do not update the parameters
- For misclassified example, we try to minimize

$$E(w) = -\sum_{i} \boldsymbol{w}\boldsymbol{x}_i y_i$$

where i ranges over all misclassified examples.

- For each misclassified example, we update the weights

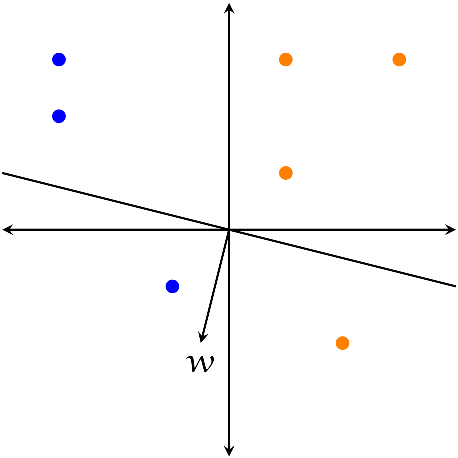
$$\begin{aligned} \boldsymbol{w} &\leftarrow \boldsymbol{w} + \eta \nabla E(\boldsymbol{w}) \\ \boldsymbol{w} &\leftarrow \boldsymbol{w} + \boldsymbol{x}_i y_i \end{aligned}$$

note that with every update the set of misclassified examples change.

The perceptron algorithm

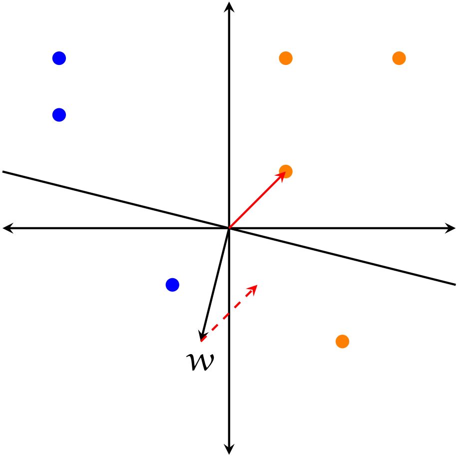
- The perceptron algorithm (eventually) converges to the global minimum if the classes are linearly separable.
- If the classes are not linearly separable, the perceptron algorithm will not stop
- We do not know whether the classes are linearly separable or not before the algorithm converges

Perceptron learning: demonstration



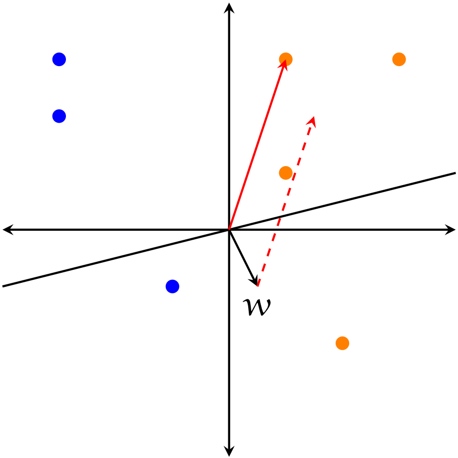
1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add it to $\boldsymbol{w}$.
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$, repeat step 2 until convergence

Perceptron learning: demonstration



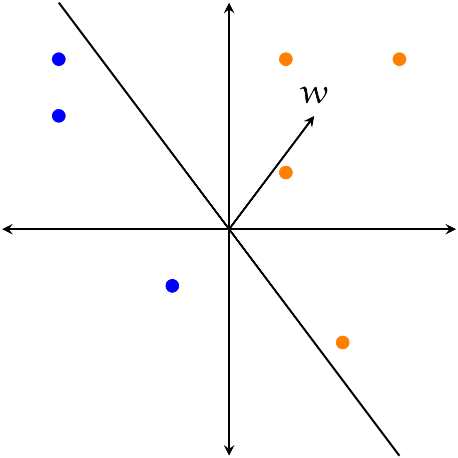
1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add it to $\boldsymbol{w}$.
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$, repeat step 2 until convergence

Perceptron learning: demonstration



1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add it to $\boldsymbol{w}$.
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$, repeat step 2 until convergence

Perceptron learning: demonstration



1. Randomly initialize $\boldsymbol{w}$ the decision boundary is orthogonal to $\boldsymbol{w}$
2. Pick a misclassified example $\boldsymbol{x}_i$ add it to $\boldsymbol{w}$.
3. Set $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i$, repeat step 2 until convergence

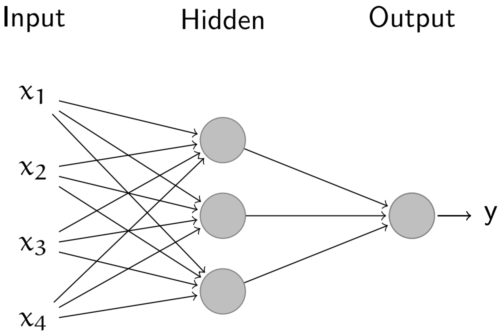
Perceptron: a bit of history

- The perceptron was developed in late 1950's and early 1960's (Rosenblatt 1958)
- It caused some excitement in many fields including computer science, artificial intelligence, cognitive science
- The excitement (and funding) died away in early 1970's (after the criticism by Minsky and Papert 1969)
- The main issue was the fact that perceptron cannot handle problems that are not linearly separable.

Multi-layer perceptron

- Multi-layer perceptron (MLP) is a fully connected feed-forward network consisting of perceptron-like units
- The units in an MLP use a continuous activation function, unlike threshold threshold function used in perceptron
- The MLP can be trained using gradient-based methods
- The MLP can represent many interesting machine learning problems. It can be used for both regression and classification

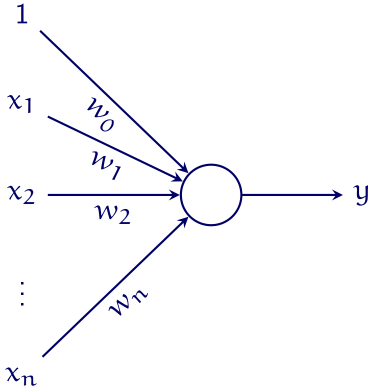
Multi-layer perceptron



Activation functions in MLP

- The activation functions in MLP are typically continuous (differentiable) functions
- For hidden units sigmoid functions (logistic sigmoid or tanh) are a common choice (more on this in a few weeks)
- The activation functions of the output units depends on the task
  - For regression, identity function
  - For binary classification, logistic sigmoid
  - For multi-class classification, softmax

Single neuron in an MLP



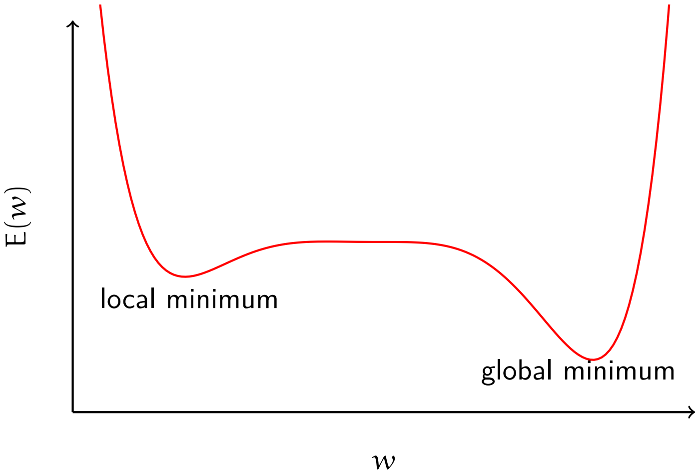
$$y = f\left(\sum_{i}^{n} w_i x_i\right)$$

- With logistic sigmoid as the activation function

$$y = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + \ldots + w_n x_n)}}$$

  it is equivalent to the logistic regression
- We can now use gradient-descent to estimate the parameters

Finding minima of the error function



Gradient descent: a refresher

- The general idea is to approach a minimum of the error function in small steps

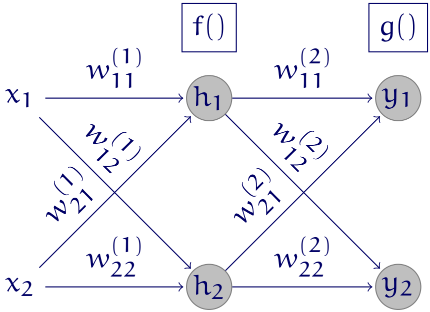
$$\boldsymbol{w} \leftarrow w - \alpha \nabla E(\boldsymbol{w})$$

  - ∇E is the gradient of the error function, it points to the direction of the maximum increase
  - α is the learning rate
- The updates can be made in

batch updates are performed for the complete training set

on-line updates are performed for each training instance. This verson is known as *stochastic gradient descent* (SGD)

MLP: a simple example

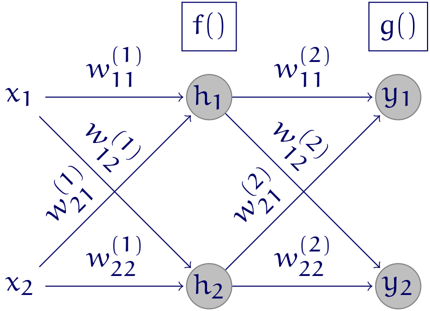


$$h_j = f\left(\sum_i w_{ij} x_i\right)$$

$$y_k = g\left(\sum_j w_{jk} h_j\right)$$

$$y_k = g\left(\sum_j w_{jk} f\left(\sum_i w_{ij} x_i\right)\right)$$

MLP: a simple example



- Alternatively, we can write the computations in matrix form

$$\mathbf{h} = f(W^{(1)}\mathbf{x})$$

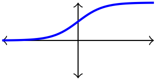
$$\begin{aligned}\mathbf{y} &= g(W^{(2)}\mathbf{h}) \\ &= g\left(W^{(2)}f(W^{(1)}\mathbf{x})\right)\end{aligned}$$

- This corresponds to a series of (non-linear) transformations of the input

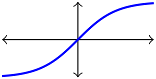
Activation functions

- Choice of activation functions depend on the application, and the type of unit
- For hidden units, common choices are sigmoid functions

  Logistic $\sigma(z) = \frac{1}{1+e^{z}}$

  Hyperbolic tangent $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

  (We will later introduce a few others)
- For regression the common choice is a linear function, most typically the identity function $I(z) = z$
- For classification, either logistic function (for binary classification), or softmax (for multi-class)
  $\mathrm{softmax}(z)_j = \frac{e^{z_j}}{\sum_k e^{z}_k}$

Error functions in ANN training

- If we assume Gaussian noise, a natural choice is the minimizing the sum of squared error

$$E(w) = \sum_i (y_i - \hat{y}_i)^2$$

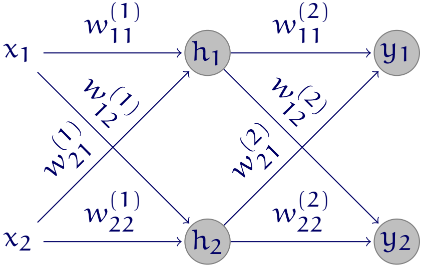
- For binary classification, we use *cross entropy*

$$E(w) = -\sum_i y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

- Similarly, for multi-class classification, also cross entropy

$$E(w) = -\sum_i \sum_k y_{i,k} \log \hat{y}_k$$

Learning in multi-layer networks: back propagation



- The final output of the network is computed by calculating the output each layer and passing it to the next (forward propagation)
- The weights are updated using a technique called back propagation
- Back-propagation algorithm makes use of chain rule of derivatives to efficiently propagate the error from output units to the input weights

Regularization in neural networks

- As in linear models we studied, we can use L1 and L2 regularization by adding a regularization term to the error function (known as weight decay). For example,

$$J(w) = E(w) + \|\boldsymbol{W}\|$$

- There are other ways to fight overfitting
  - With early stopping, one stops the training before it reaches to the smallest training error
  - With dropout, random units (with all of their connections) are dropped during training
  - Injecting noise at the output, as a way to (implicitly) model the noise in the target classes/values

How many layers, units

- A network with single hidden layer, is said to be *a universal approximator*: it can approximate any continuous function with arbitrary precision
- However, in practice multiple interconnected layers are useful and commonly used in modern ANN models
- The choice of layers, in general the architecture of the system, depends on the application

Another bit of history

- In 1980's ANNs became popular again
- One of the important developments that made this possible was the back propagation algorithm
- In 1990's the ANNs had again fallen 'out of fashion'. Mainly due to
  - From the engineering perspective: other, more successful algorithms (such as SVMs) performed generally better
  - From the cognitive science perspective: the fact that ANNs are complex systems that are difficult to interpret
- At present (after 2005 or so) they, once more, enjoy success stories and popularity with the name 'deep learning'
- We will study some aspects of the deep learning methods for the remainder of the course

References/Credits

- The neuron figure on slide 3 is adapted from the figure by Quasar Jarosz at English Wikipedia.

Minsky, Marvin and Seymour Papert (1969). *Perceptrons: An introduction to computational geometry*. MIT Press.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, pp. 386–408.