Slide 1

Introduction to artificial neural networks
Statistical Methods in NLP 2
ISCL-BA-08

Çağrı Çöltekin

ccoltekin@sfs.uni-tuebingen.de

University of Tübingen
Seminar für Sprachwissenschaft

Summer Semester 2025

Slide 2

Artificial neural networks

- Artificial neural networks (ANNs) are machine learning models inspired by biological neural networks
- ANNs are powerful non-linear models
- Power comes with a price: there are no guarantees of finding the global minimum of the error function
- ANNs have been used in ML, AI, Cognitive science since 1950's – with some ups and downs
- Currently they are the driving force behind the popular 'deep learning' methods
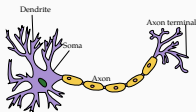

Slide 3

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

Dendrite

Soma

Axon

Axon terminal

*Image source: Wikipedia

Slide 4

Artificial and biological neural networks

- ANNs are inspired by biological neural networks
- Similar to biological networks, ANNs are made of many simple processing units
- Despite the similarities, there are many differences: ANNs do not mimic biological networks
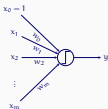- ANNs are practical statistical machine learning methods

---

Slide 5

Recap: the perceptron

$$y = f\left(\sum_i^m w_j x_i\right)$$

where

$$f(x) = \begin{cases} +1 & \text{if } wx > 0 \\ -1 & \text{otherwise} \end{cases}$$

In ANN-speak $f(\cdot)$ is called an activation function.

$x_0 = 1$

x_1

x_2

x_m

y

Slide 6

Recap: logistic regression

$$P(y) = f\left(\sum_i^m w_j x_i\right)$$

where

$$f(x) = \frac{1}{1 + e^{-wx}}$$

$x_0 = 1$

x_1

x_2

x_m

$P(y)$

Slide 7

Linear separability

- A classification problem is said to be linearly separable if one can find a linear discriminator
- A well-known counter example is the logical XOR problem

There is no line that can separate positive and negative classes.

Slide 8

Can a linear classifier learn the XOR problem?

- We can use non-linear basis functions

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 \phi(x_1, x_2)$$

is still linear in w for any choice of $\phi(\cdot)$
- For example, adding the product $x_1 x_2$ as an additional feature would allow a solution like: $x_1 + x_2 - 2x_1 x_2$

x_1	x_2	$x_1 + x_2 - 2x_1 x_2$
0	0	0
0	1	1
1	0	1
1	1	0

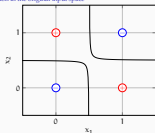- Choosing proper basis functions like $x_1 x_2$ is called feature engineering

---

Slide 9

Non-linear basis functions
solution in the original input space

The solution to

$$x_1 + x_2 - 2x_1 x_2 - 0.5 = 0$$

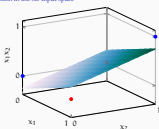is a (non-linear) discriminant that solves the problem

---

Slide 10

Non-linear basis functions
solution in the 3D input space

- The additional basis function maps the problem in 3D
- In the new, mapped space, the points are linearly separable

---

Slide 11

Where do non-linearities come from?
non-linearities are abundant in nature, it is not only the XOR problem

In a linear model, $y = w_0 + w_1 x_1 + \ldots + w_k x_k$
- The outcome is linearly-related to the predictors
- The effects of the inputs are additive

This is not always the case:
- Some predictors affect the outcome in a non-linear way
  - The effect may be strong or positive only in a certain range of the variable (e.g., reaction time change by age)
  - Some effects are periodic (e.g., many measures of time)
- Some predictors interact
  - 'not bad' is not 'not' + 'bad' (e.g., for sentiment analysis)

Slide 12

Multi-layer perceptron

- The simplest modern ANN architecture is called multi-layer perceptron (MLP)
- The MLP is a fully connected, feed-forward network consisting of perceptron-like units
- Unlike perceptron, the units in an MLP use a continuous activation function
- 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
the picture

Input   Hidden   Output

x_1
x_2
x_3
x_4

y
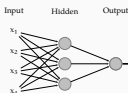
Each unit takes a weighted sum of their input,
and applies a (non-linear) *activation function*.

Artificial neurons

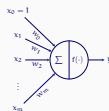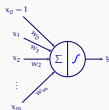- The unit calculates a weighted sum
  of the inputs

$$\sum_1^m w_i y_i = \boldsymbol{w}\boldsymbol{x}$$

$x_0 = 1$
x_1
x_2
$\vdots$
x_m

$\sum$ $f(\cdot)$ y

w_0
w_1
w_2
w_m

- result is a linear transformation
- a non-linear activation function $f(\cdot)$
  applied to the result
- Output is

$$y = f(\boldsymbol{w}\boldsymbol{x})$$

Artificial neurons
an example

$x_0 = 1$
x_1
x_2
$\vdots$
x_m

$\sum$ f y

w_0
w_1
w_2
w_m

- A common activation function is
  (was?) the *logistic sigmoid* function

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

- The output of the network becomes

$$y = \frac{1}{1 + e^{-\boldsymbol{w}\boldsymbol{x}}}$$

Activation functions in ANNs
hidden units

- The activation functions in MLP are typically continuous (differentiable)
  functions
- For hidden units common choices are

Sigmoid (logistic) $\frac{1}{1+e^{-x}}$

Hyperbolic tangent (tanh) $\frac{e^x - 1}{e^x + 1}$

Rectified linear unit (relu) $\max(0, x)$

Activation functions in ANNs
output units

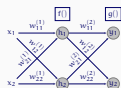- The activation functions of the output units depends on the task. Common
  choices are
  - For regression, the identity function ($y = x$)
  - For binary classification, logistic sigmoid

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-\boldsymbol{w}\boldsymbol{x}}} = \frac{e^{\boldsymbol{w}\boldsymbol{x}}}{1 + e^{\boldsymbol{w}\boldsymbol{x}}}$$

  - For multi-class classification, softmax

$$P(y = k \mid x) = \frac{e^{\boldsymbol{w}\boldsymbol{x}_k}}{\sum_j e^{\boldsymbol{w}\boldsymbol{x}_k}}$$

MLP: a simple example

$f(\cdot)$ $g(\cdot)$

x_1 $w_{11}^{(1)}$ h_1 $w_{11}^{(2)}$ g_1

x_2 $w_{22}^{(1)}$ h_2 $w_{22}^{(2)}$ g_2

$$h_i = f\left(\sum_i w_{ij}^{(1)} x_i\right)$$

$$y_k = g\left(\sum_j w_{jk}^{(2)} h_j\right)$$

$$y_k = g\left(\sum_j w_{jk}^{(2)} f\left(\sum_i w_{ij}^{(1)} x_i\right)\right)$$

MLP: a simple example

$f(\cdot)$ $g(\cdot)$

x_1 $w_{11}^{(1)}$ h_1 $w_{11}^{(2)}$ g_1

x_2 $w_{22}^{(1)}$ h_2 $w_{22}^{(2)}$ g_2

- Alternatively, we can write the
  computations in matrix form

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

$$y = g(W^{(2)} h)$$
$$= g\left(W^{(2)} f(W^{(1)} x)\right)$$

- This corresponds to a series of linear
  transformations followed by
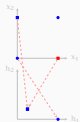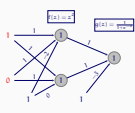 elementwise (non-linear) function
  applications

Solving non-linear problems with ANNs
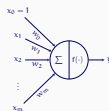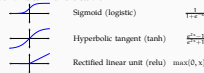a solution to XOR problem

$f(x) = x^2$ $g(x) = \frac{1}{1+e^{-x}}$

x_1 $w_{11}^{(1)}$ h_1

x_2 h_2

g

1

Solving non-linear problems with ANNs
a solution to XOR problem

$f(x) = x^2$ $g(x) = \frac{1}{1+e^{-x}}$

0   1
0

x_1 4   0

x_2 3

1   0   1

Solving non-linear problems with ANNs
a solution to XOR problem

$f(x) = x^2$ $g(x) = \frac{1}{1+e^{-x}}$

0   1

x_1 1   1

x_2

1   0   1

Solving non-linear problems with ANNs
a solution to XOR problem

$f(x) = x^2$ $g(x) = \frac{1}{1+e^{-x}}$

1   1

x_1 1

0   1

x_2

1   0   1

Solving non-linear problems with ANNs
a solution to XOR problem

$f(x) = x^2$ $g(x) = \frac{1}{1+e^{-x}}$

1   1

x_1 0

1   f

x_2 4

1   0   1
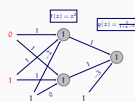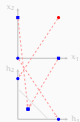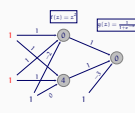
Is this different from non-linear basis functions?

Non-linear activation functions are necessary

Without non-linear activation functions, an ANN with any number of layers is equivalent to a linear model.

x_1 — h_1

y

x_2 — h_2
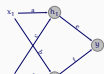
$$h_1 = ax_1 + cx_2$$
$$h_2 = bx_1 + dx_2$$
$$y = eh_1 + fh_2$$
$$= (ea + fb)x_1 + (ec + fd)x_2$$

y is still a linear function of x_1

Gradient descent: a refresher

- The general idea is to approach a minimum of the error function in small (or not so small) steps

$$w \leftarrow w - \eta \nabla J(w)$$

  - ∇J is the gradient of the loss function, it points to the direction of the maximum increase
  - η is the learning rate
- The updates can be performed

  batch for the complete training set
  on-line after every training instance
  - this is known as *stochastic gradient descent* (SGD)
  mini-batch after small fixed-sized batches

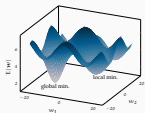## ANN objectives are not convex

---

Error functions in ANN training
depend on the task

- For regression, a natural choice is minimizing the sum of squared error

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

- For binary classification, we use *cross entropy*

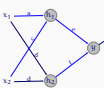$$E(w) = -\sum_i y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

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

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

In practice, the ANN loss functions will not be convex.

Learning in ANNs

- ANNs implement complex functions: we need to use optimization methods (e.g., gradient descent) to train them
- Typically error functions for ANNs are not convex, gradient descent will find a local minimum
- Optimization requires updating multiple layers of weights
- Assigning credit (or blame) to each weight during learning is not trivial
- An effective solution to the last problem is the *backpropagation* algorithm

---

Learning in multi-layer networks: the problem

x_1 $w_{11}^{(1)}$ $w_{11}^{(2)}$ $\hat{y}$

x_2 $w_{22}^{(1)}$ $w_{22}^{(2)}$ h_2

$E(y) = ?$ $E(y) = (y - \hat{y})^2$

We want a way to (efficiently) update non-final weights based on final error.

Calculating gradient on a neural network
(with some simplification)

x_1 — h_1

$E(y, \hat{y})$

y

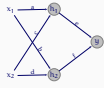x_2 — h_2

- We need to calculate the gradient:

$$\nabla E = \left(\frac{\partial E}{\partial a}, \frac{\partial E}{\partial b}, \frac{\partial E}{\partial c}, \frac{\partial E}{\partial d}, \frac{\partial E}{\partial e}, \frac{\partial E}{\partial f} \right)$$

we can use gradient descent directly
- $\frac{\partial E}{\partial e}$ and $\frac{\partial E}{\partial f}$ is easy, they do not depend on other variables
- We factor others using chain rule

$$\frac{\partial E}{\partial a} = \frac{\partial h1}{\partial a} \frac{\partial E}{\partial h1} \quad \text{and} \quad \frac{\partial E}{\partial c} = \frac{\partial h1}{\partial c} \frac{\partial E}{\partial h1}$$

Backpropagation

x_1 — h_1

y

x_2 — h_2

- So far, it is just math

$$\frac{\partial E}{\partial a} = \frac{\partial h1}{\partial a} \frac{\partial E}{\partial h1} \quad \text{and} \quad \frac{\partial E}{\partial c} = \frac{\partial h1}{\partial c} \frac{\partial E}{\partial h1}$$

- But a naive implementation does many repeated calculations
- Backpropagation is an efficient (dynamic programming) algorithm that avoids repeated calculations
- Backpropagation works for any *computation graph* without cycles

---

Preventing overfitting in neural networks

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

$$J(w) = E(w) + \|w\|$$

- 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

---

A bit of history

1950-60 ANNs (perceptron) became popular:
         lots of excitement in AI, cognitive science
1970s Not much interest
       - criticism on perceptron: linear separability
1980s ANNs became popular again
       - backpropagation algorithm
       - multi-layer networks
1990s ANNs had again fallen 'out of fashion'
       - Engineering: other algorithms (such as SVMs) performed generally better
       - From the cognitive science perspective: ANNs are difficult to interpret
present ANNs (again) enjoy a renewed popularity with the name 'deep learning'

Summary

- ANNs are powerful non-linear learners
  - based on some inspiration from biological NNs
  - using many simple processing units
  - built on linear models (logistic regression)
- For non-linear problems we need non-linear activation functions, and at least one hidden layer
- ANNs can be used for both regression and classification
- In general, ANN loss functions are not convex, what we find is a local minimum
- They (typically) are trained with *backpropagation* algorithm
- Reading: Jurafsky and Martin (2025, Chapter 7)
Next:
- Models for sequential data, reading: Jurafsky and Martin (2025, Chapter 17)

- Hastie, Tibshirani, and Friedman (2009, Chapter. 11) also includes an accessible introduction
- For a review of use of ANNs in NLP, including more advanced topics, see Goldberg, 2016

Goldberg, Yoav (2016). "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57, pp. 345–420.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. Springer series in statistics. Springer-Verlag New York. ISBN: 9780387848587. URL: http://web.stanford.edu/~hastie/ElemStatLearn/.

Jurafsky, Daniel and James H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

References & further reading

Goldberg, Yoav (2016). "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57, pp. 345–420.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. Springer series in statistics. Springer-Verlag New York. ISBN: 9780387848587. URL: http://web.stanford.edu/~hastie/ElemStatLearn/.

Jurafsky, Daniel and James H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.