MACHINE LEARNING BASICS

May 21st–22nd 2025 | Stefan Häusler, Marina Ganeva | JCNS

Day I: Organization

10:00 – 12:00 Tutorial

12:00 – 13:00 Lunch

13:00 – 15:00 Tutorial

15:00 – 15:30 Coffee

15:30 – 17:30 Q&A and/or project

Outline

DAY I: Introduction to machine learning

— lunch break —

Neural networks

Day II: Introduction to language models

— lunch break —

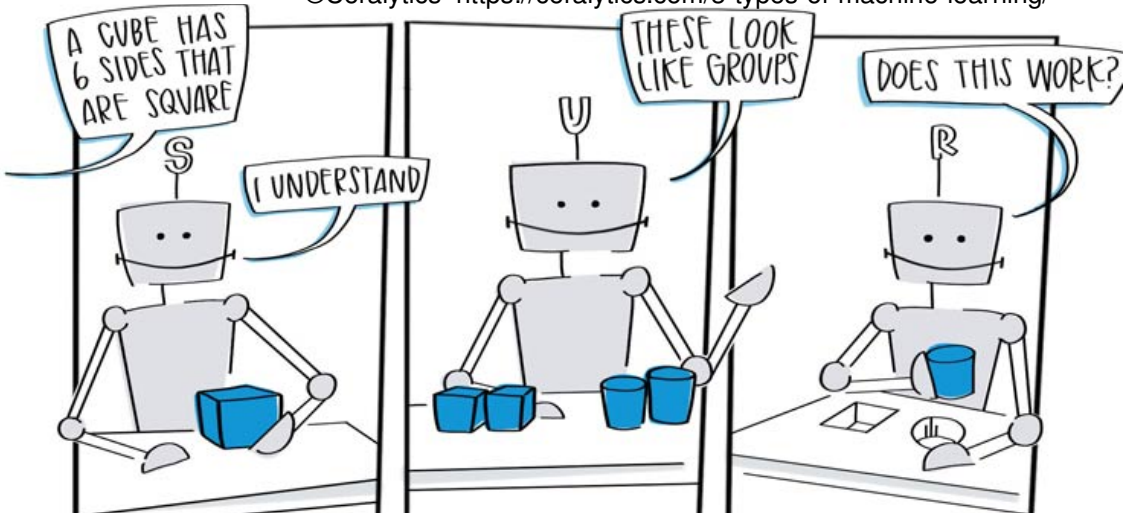
Huggingface & GPT2

Generative pre-trained transformer

What is *Machine Learning*?

Machine Learning

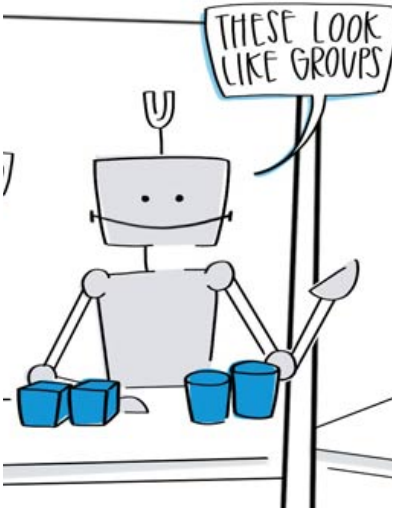
©Ceralytics https://ceralytics.com/3-types-of-machine-learning/



supervised **unsupervised** **reinforced**

JÜLICH Forschungszentrum

Unsupervised learning



Clustering
Organize similar items into groups.

Manifold learning
Reduce the dimensionality of a dataset while maintaining the essential relationships between the points.

Anomaly detection
Find out whether the new point belongs to the same distribution as existing ones.

Generative modelling
Generate text, music, images using neural networks.

Learning data underlying structure

Example: Clustering

Organize similar items into groups

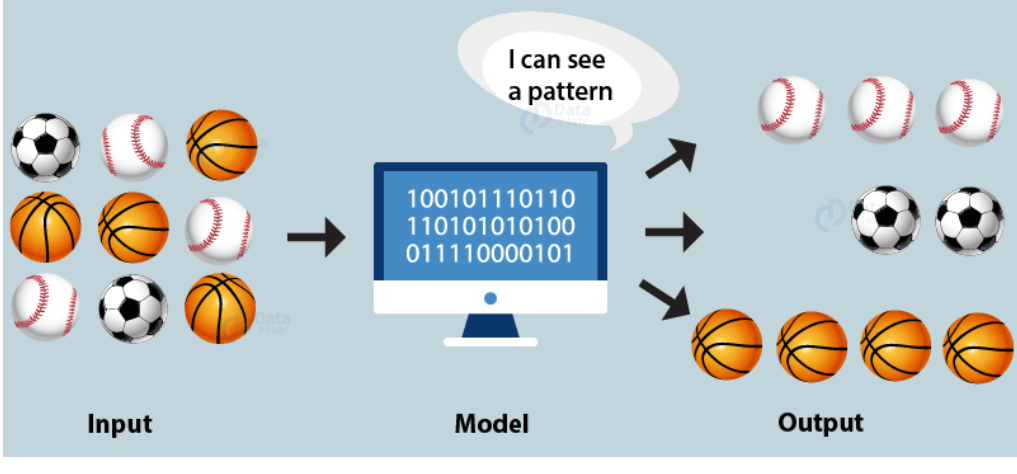
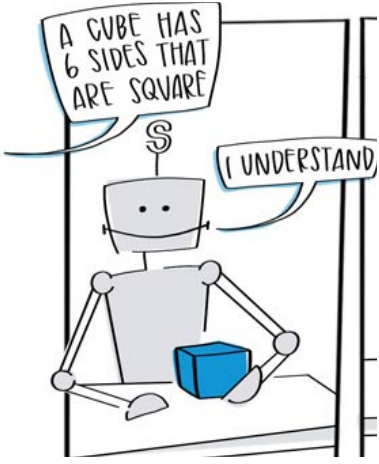


Image from https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/08/introduction-to-clustering.png

Supervised learning



Uses **labeled datasets** to train algorithms to predict outcomes and recognize patterns.

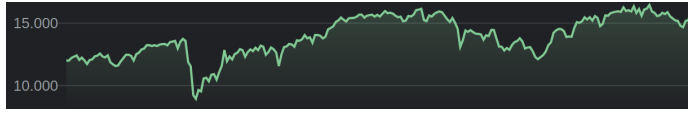
Classification

Goal is to assign the input data to specific categories.

3 1 2 0 3 2 0 1

Regression

Goal is to predict a numerical value given some inputs.



Learning data underlying patterns and relationships

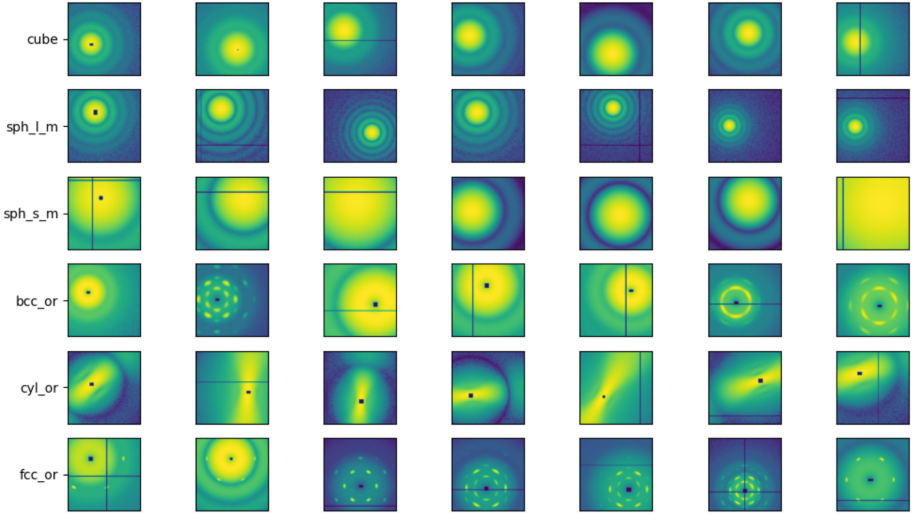
Discussion

Machine Learning — What and how do we learn?

Supervised Machine Learning in a Nutshell

- **Data:** List of data samples $D = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N))$ for $D \in \mathcal{D}$

Example: SANS pattern classification

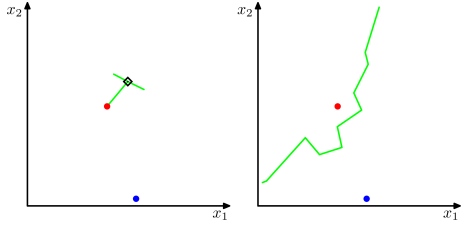


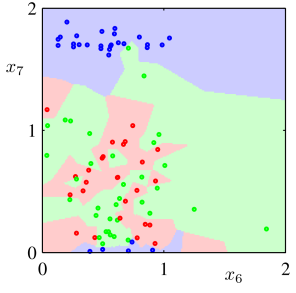
Supervised Machine Learning in a Nutshell

- **Data:** List of data samples $D = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N))$ for $D \in \mathcal{D}$

- **Machine learning models**: Make predictions $H : X \to Y$ for $H \in \mathcal{H}$

Classification example: k-nearest neighbor algorithm





Supervised Machine Learning in a Nutshell

- **Data:** List of data samples $D = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N))$ for $D \in \mathcal{D}$
- **Machine learning models**: Make predictions $H : X \to Y$ for $H \in \mathcal{H}$
- **Machine learning algorithm**: Return models given samples $\mathcal{D} \to \mathcal{H}$

Supervised Machine Learning in a Nutshell

- **Data:** List of data samples $D = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N))$ for $D \in \mathcal{D}$

- **Machine learning models**: Make predictions $H : X \to Y$ for $H \in \mathcal{H}$

- **Machine learning algorithm**: Return models given samples $\mathcal{D} \to \mathcal{H}$

  The algorithm minimizes the true error E_P estimated by the empirical error E_D.

$$E_P = \langle E(y, H(\mathbf{x})) \rangle_{(\mathbf{x},y)\sim P} \qquad E_D = \frac{1}{N} \sum_{n=1}^{N} E(y_n, H(\mathbf{x}_n))$$

  E.g. for classification: The error $E(y_n, H(\mathbf{x}_n)) = 0$ if $y_n = H(\mathbf{x}_n)$ and 1 otherwise.

Training and testing

Training: The learning algorithm selects a model based on the empirical error of the training data (e.g. select models parameters).

Testing: The quality of the selected model is estimated based on the empirical error of the test data, also known as generalization.

$$\text{Training data} \cap \text{Test data} = \emptyset$$

Supervised Machine Learning in a Nutshell

- **Data:** List of data samples $D = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_N, y_N))$ for $D \in \mathcal{D}$
- **Machine learning models**: Make predictions $H : X \to Y$ for $H \in \mathcal{H}$
- **Machine learning algorithm**: Return models given samples $\mathcal{D} \to \mathcal{H}$

  The algorithm minimizes the true error E_P estimated by the empirical error E_D.

  $$E_P = \langle E(y, H(\mathbf{x})) \rangle_{(\mathbf{x},y)\sim P} \qquad E_D = \frac{1}{N} \sum_{n=1}^{N} E(y_n, H(\mathbf{x}_n))$$

  E.g. for classification: The error $E(y_n, H(\mathbf{x}_n)) = 0$ if $y_n = H(\mathbf{x}_n)$ and 1 otherwise.

- **Model selection**: Compare machine learning algorithms and model classes. For example, select hyperparameters.

Training, testing and validation

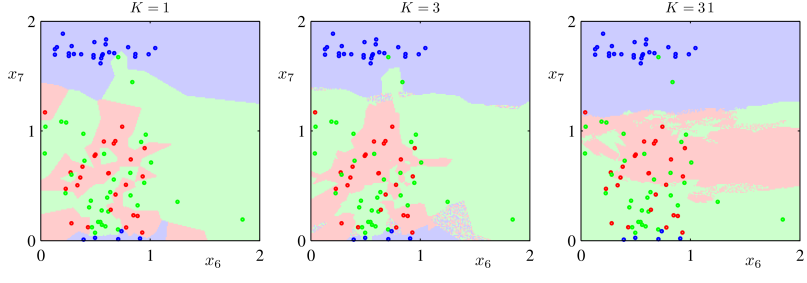
Training: The learning algorithm selects a model based on the empirical error of the training data (e.g. select models parameters).

Testing: The quality of the selected model is estimated based on the empirical error of the test data, also known as generalization.

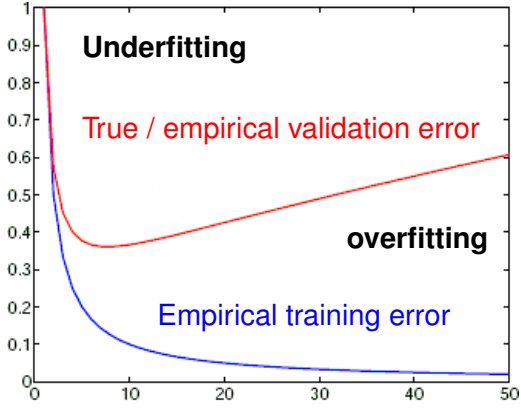
Validation: The empirical error of the validation data is used to select a learning algorithm or model class (e.g. select hyperparameters).

$$\text{Training data} \cap \text{Test data} = \emptyset$$
$$\text{Training data} \cap \text{Validation data} = \emptyset$$
$$\text{Test data} \cap \text{Validation data} = \emptyset$$

Example: k-nearest neighbor algorithm



Model selection and overfitting



Exercise 1: Model selection and overfitting

Outline

DAY I: Introduction to machine learning

— lunch break —

Neural networks

Day II: Introduction to language models

— lunch break —

Huggingface & GPT2

Generative pre-trained transformer

Outline

DAY I: Introduction to machine learning

— lunch break —

Neural networks

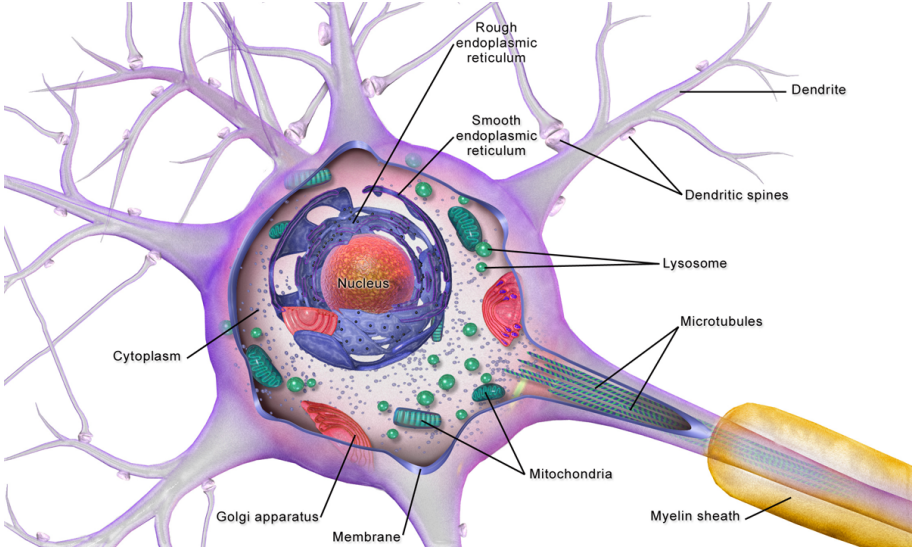
Day II: Introduction to language models

— lunch break —

Huggingface & GPT2

Generative pre-trained transformer

Brain neuron



https://en.wikipedia.org/wiki/Neuron

Neuron model

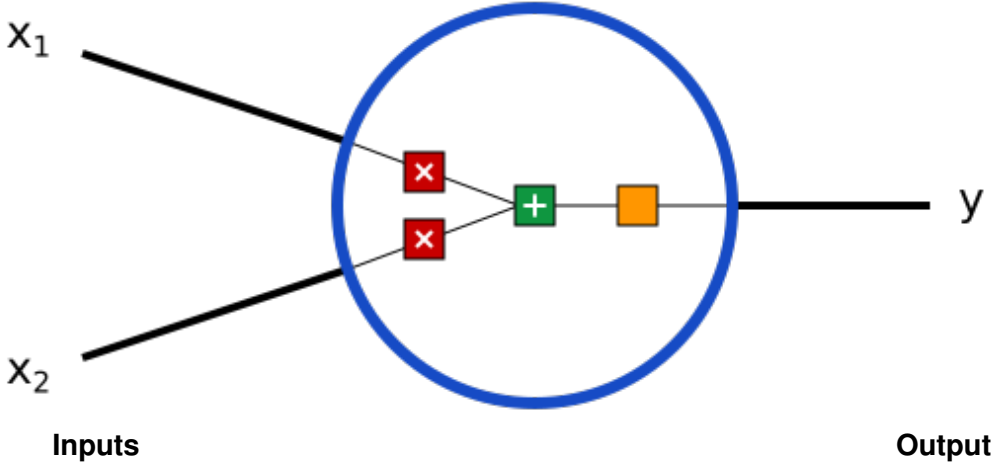


Image from https://victorzhou.com/blog/intro-to-neural-networks/

Neuron model

Single layer neural network

$$a(x, w) = \sigma(w^T x) = \sigma\left(\sum_{j=1}^{d} w_j^{(1)} x_j + w_0^{(1)}\right)$$

where

σ is an activation function,

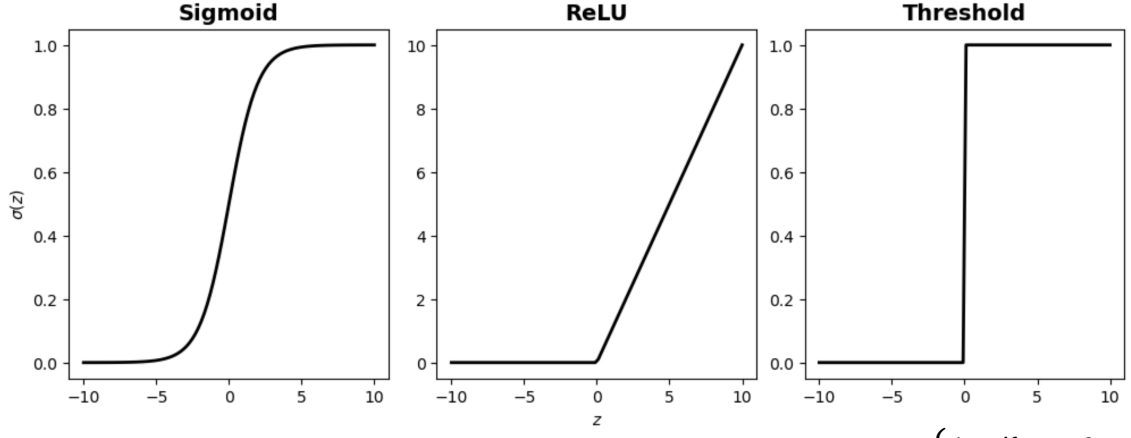
w — vector of weights (model parameters),

x — features vector, $x_0 = 1$

For $\sigma = id$, i.e. $a(x, w) = w^T x$ — *linear regression*.

Activation functions

Activation functions are real monotonic functions, preferably differentiable

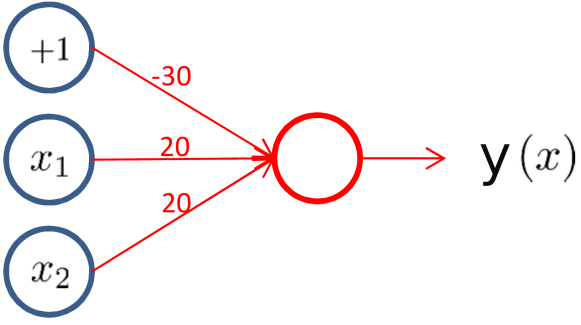


$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\sigma(z) = \max(0,\, z)$$

$$\sigma(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z < 0 \end{cases}$$

Example: x_1 AND x_2

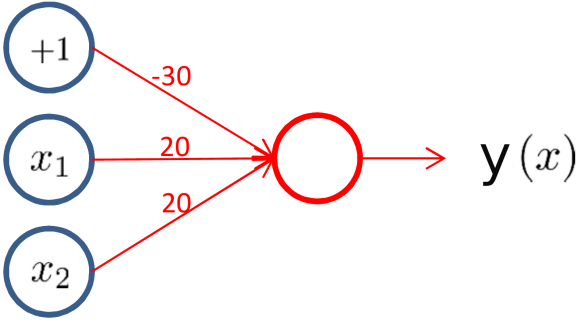


x_1	x_2	$y(x)$
0	0	$\sigma(-30) = \mathbf{0}$
0	1	$\sigma(-10) = \mathbf{0}$
1	0	$\sigma(-10) = \mathbf{0}$
1	1	$\sigma(+10) = \mathbf{1}$

$$y(x) = \sigma(-30 + 20 \cdot x_1 + 20 \cdot x_2)$$

where σ is a threshold activation function

Example: x_1 AND x_2

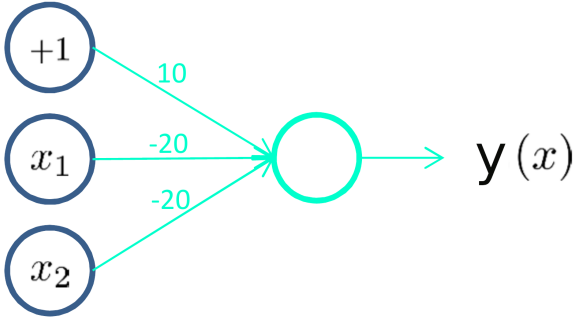


x_1	x_2	$y(x)$
0	0	$\sigma(-30) = \mathbf{0}$
0	1	$\sigma(-10) = \mathbf{0}$
1	0	$\sigma(-10) = \mathbf{0}$
1	1	$\sigma(+10) = \mathbf{1}$

$$y(x) = \sigma(-30 + 20 \cdot x_1 + 20 \cdot x_2)$$

where σ is a threshold activation function

Example: (NOT x_1) AND (NOT x_2)

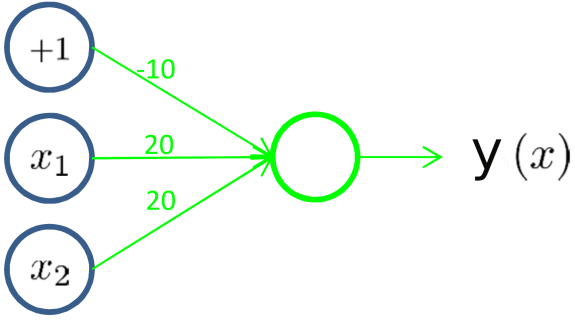


x_1	x_2	$y(x)$
0	0	$\sigma(+10) = \mathbf{1}$
0	1	$\sigma(-10) = \mathbf{0}$
1	0	$\sigma(-10) = \mathbf{0}$
1	1	$\sigma(-30) = \mathbf{0}$

$$y(x) = \sigma(10 - 20 \cdot x_1 - 20 \cdot x_2)$$

where σ is a threshold activation function

Example: x_1 OR x_2

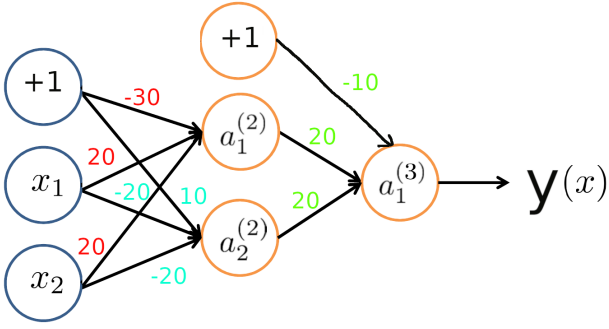


x_1	x_2	$y(x)$
0	0	$\sigma(-10) = \mathbf{0}$
0	1	$\sigma(+10) = \mathbf{1}$
1	0	$\sigma(+10) = \mathbf{1}$
1	1	$\sigma(+30) = \mathbf{1}$

$$y(x) = \sigma(-10 + 20 \cdot x_1 + 20 \cdot x_2)$$

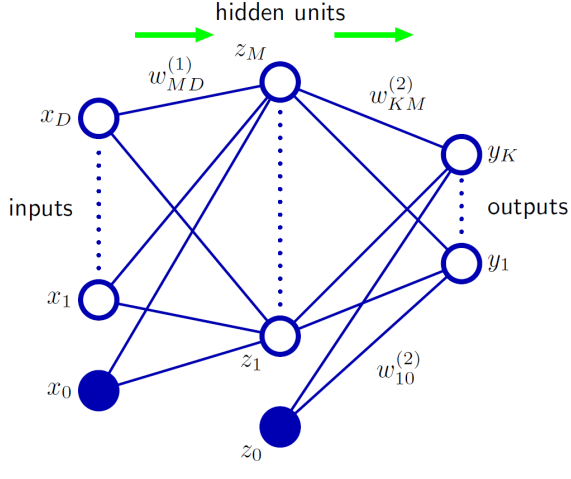
where σ is a threshold activation function

Example: x_1 XNOR x_2



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$y(x)$
0	0	0	1	**1**
0	1	0	0	**0**
1	0	0	0	**0**
1	1	1	0	**1**

Feedforward Neural Networks



Feedforward Neural Networks

$$y_k(x, W) = \sigma^{(2)} \left(\sum_{i=1}^{M} w_{ki}^{(2)} \cdot \sigma^{(1)} \left(\sum_{j=1}^{D} w_{ij}^{(1)} x_j + w_{i0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

where

$\sigma^{(k)}$ are the activation functions for k-th layer,

$w^{(k)}$ — vectors of weights (model parameters) for k-th layer,

x — features vector, $x_0 = 1$

$W = \left\{ w_{ki}^{(2)}, w_{k0}^{(2)}, w_{ij}^{(1)}, w_{i0}^{(1)} \right\}$ vector of model parameters for the whole neural network

Universal function approximation

- **Boolean networks:**
  Every Boolean function $f : \{0,1\}^D \to \{0,1\}^K$ for $D, K \in \mathbb{N}$ can be implemented by a neural network with one hidden layer consisting exclusively of threshold functions.

Universal function approximation

- **Boolean networks:**
  Every Boolean function $f : \{0,1\}^D \to \{0,1\}^K$ for $D, K \in \mathbb{N}$ can be implemented by a neural network with one hidden layer consisting exclusively of threshold functions.

- **Shallow networks:**
  For every continuous function $f : [0,1]^D \to \mathbb{R}^K$ for $D, K \in \mathbb{N}$ and ε exists a neural network N with one hidden layer consisting exclusively of sigmoidal activations such that $||f(\mathbf{x}) - N(\mathbf{x})|| \leq \varepsilon$ for all $\mathbf{x} \in [0,1]^D$.

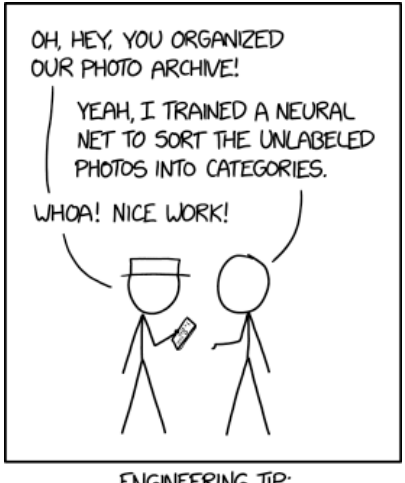
Universal function approximation

- **Boolean networks:**
  Every Boolean function $f : \{0,1\}^D \to \{0,1\}^K$ for $D, K \in \mathbb{N}$ can be implemented by a neural network with one hidden layer consisting exclusively of threshold functions.

- **Shallow networks:**
  For every continuous function $f : [0,1]^D \to \mathbb{R}^K$ for $D, K \in \mathbb{N}$ and ε exists a neural network N with one hidden layer consisting exclusively of sigmoidal activations such that $||f(\mathbf{x}) - N(\mathbf{x})|| \leq \varepsilon$ for all $\mathbf{x} \in [0,1]^D$.

- **Deep networks:**
  For every continuous function $f : [0,1]^D \to \mathbb{R}^K$ for $D, K \in \mathbb{N}$ and ϵ exists a neural network N of width $K + D + 2$ with multiple hidden layers and ReLU activations such that $||f(\mathbf{x}) - N(\mathbf{x})|| \leq \varepsilon$ for all $\mathbf{x} \in [0,1]^D$.

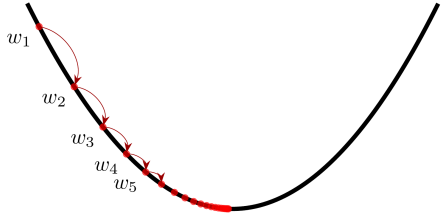
https://xkcd.com/2173/



ENGINEERING TIP:
WHEN YOU DO A TASK BY HAND, YOU CAN TECHNICALLY SAY YOU TRAINED A NEURAL NET TO DO IT.

Gradient descent

Gradient descent



Principle: repeat small updates

$$w_i = w_i - \alpha \nabla_{w_i} J(\mathbf{w})$$

where α is a learning parameter and J an error function. E.g. for 1D regression

$$J(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (H(x_n) - y_n)^2$$

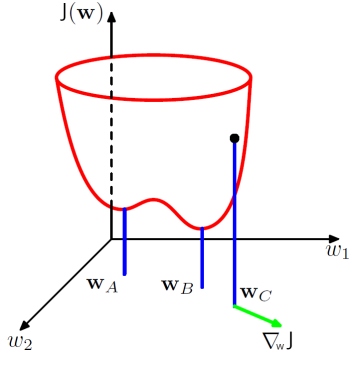
for the linear model $H(x_n) = w_0 + w_1 x_n$.

For 1D linear regression:

$$w_0 = w_0 - \alpha \frac{\partial J(w_0, w_1)}{\partial w_0}, \qquad \frac{\partial J(w_0, w_1)}{\partial w_0} = \frac{2}{n} \sum_{n=1}^{N} (w_0 + w_1 x_n - y_n)$$

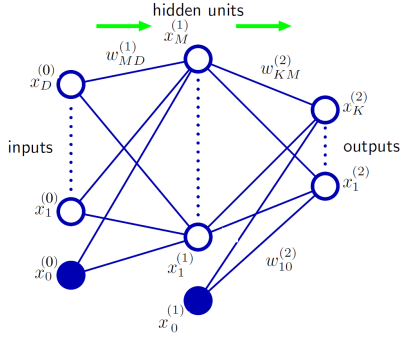
$$w_1 = w_1 - \alpha \frac{\partial J(w_0, w_1)}{\partial w_1}, \qquad \frac{\partial J(w_0, w_1)}{\partial w_1} = \frac{2}{n} \sum_{n=1}^{N} (w_0 + w_1 x_n - y_n)\, x_n$$

Gradient descent optimization



$$\Delta w_i = -\alpha \frac{\partial J}{\partial w_i} \qquad \Delta \mathbf{w} = -\alpha \nabla_{\mathbf{w}} J$$

Gradient descent



Output of neuron *a* with sigmoidal activation function σ on the last layer (2):

$$x_a^{(2)} = \sigma \underbrace{\left(\sum_b w_{ab}^{(2)} x_b^{(1)} \right)}_{h_a^{(2)}} = \sigma \left(\sum_b w_{ab}^{(2)} \sigma \underbrace{\left(\sum_c w_{bc}^{(1)} x_c^{(0)} \right)}_{h_b^{(1)}} \right)$$

Gradient descent optimization

Output of neuron a with sigmoidal activation function σ on the last layer (2):

$$x_a^{(2)} = \sigma\underbrace{\left(\sum_b w_{ab}^{(2)} x_b^{(1)}\right)}_{h_a^{(2)}} = \sigma\left(\sum_b w_{ab}^{(2)} \sigma\underbrace{\left(\sum_c w_{bc}^{(1)} x_c^{(0)}\right)}_{h_b^{(1)}}\right)$$

Learning rule: For learning rate η and mean squared error J

$$\Delta w_{ij}^{(l)} = -\alpha \frac{\partial J}{\partial w_{ij}^{(l)}} \qquad J = \frac{1}{2}\sum_a \left(y_a - x_a^{(2)}\right)^2$$

Efficient implementation: Backpropagation algorithm

Backpropagation algorithm

Gradients of the weights in layer 2

$$\frac{\partial J}{\partial w_{ij}^{(2)}} = \underbrace{(y_i - x_i^{(2)})\sigma'(h_i^{(2)})}_{\delta_i^{(2)}} x_j^{(1)} = \delta_i^{(2)} x_j^{(1)}$$

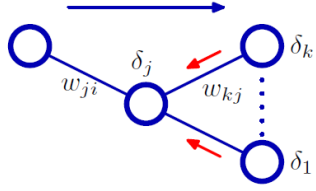
and layer 1 after application of the **"chain rule"**

$$\frac{\partial J}{\partial w_{ij}^{(1)}} = \underbrace{\sum_a \delta_a^{(2)} w_{ai}^{(2)} \sigma'(h_i^{(1)})}_{\delta_i^{(1)}} x_j^{(0)} = \delta_i^{(1)} x_j^{(0)}$$

The gradient is always a product of a neuronal output $x_j^{(l)}$ and an error factor $\delta_i^{(l+1)}$.

Backpropagation algorithm

The error δ in layer l can be computed from the δs in the next downstream layer $l+1$ by



$$\delta_i^{(l)} = \sum_a \delta_a^{(l+1)} w_{ai}^{(l+1)} \sigma'(h_i^{(l)})$$

The δs are back propagated from the output to the input layer coining the term **error backpropagation**.

Backpropagation algorithm

Recipe for a feed-forward network consisting of L layers:

1. Forward pass:
   For given input $\mathbf{x}^{(0)}$ compute the outputs $\mathbf{x}^{(l)}$ for $0 < l \leq L$ of all neurons.

2. Backward pass:
   Compute the total error J and all δs in the backward direction.

3. Update the network weights according to the gradient descent rule.

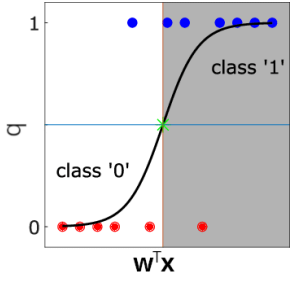
4. Go back to 1 until convergence to a (local) minimum of the error function J.

Exercise 2: backprop/regression

Binary classification

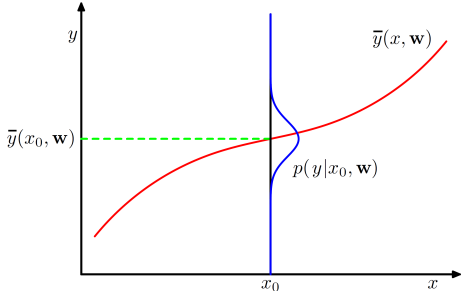
Logistic regression

- Two-class classification problem for $y \in \{0, 1\}$
- Learning a conditional distribution $q(y = 1|\mathbf{x}, \mathbf{w})$ parameterized by $\mathbf{w}$.
- For logistic regression $q(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x})$, where σ is the logistic sigmoid.



Density estimation

Other examples, regression



- Instead of a function $y = f(x)$ a conditional distribution $p(y|x)$ is learned.
- A model distribution $q(y|x, \mathbf{w})$ with parameters $\mathbf{w}$ is optimized.
- What is the equivalent of the error function that enables gradient descent?

Density estimation

Error function

Kullback–Leibler (KL) divergence:
Measures how one probability distribution p is different from a second, reference probability distribution q. For densities p and q

$$D_{\mathrm{KL}}(p \parallel q) = \int_{-\infty}^{\infty} p(\mathbf{x}, \mathbf{y}) \ \ln \frac{p(\mathbf{x}, \mathbf{y})}{q(\mathbf{x}, \mathbf{y})} \, \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{y}$$

$D_{\mathrm{KL}}(p \parallel q) \geq 0$ and 0 if and only if $p = q$.

$$0 \leq \left\langle \ln \frac{p}{q} \right\rangle = \langle \ln(p) - \ln(q) \rangle = \langle \ln(p) \rangle - \langle \ln(q) \rangle$$

Maximizing $\langle \ln q(\mathbf{y}|\mathbf{x}, \mathbf{w}) \rangle$ w.r.t to $\mathbf{w}$ results in the best estimate minimizing $D_{\mathrm{KL}}(p \parallel q)$.

Logistic regression

Binary classification with cross-entropy error function

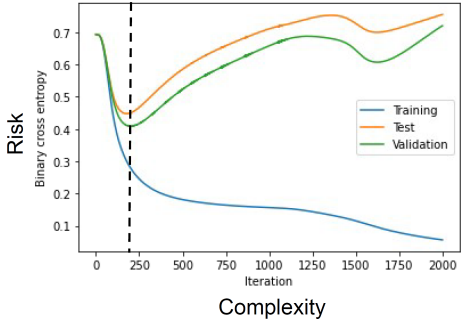
For N i.i.d. samples y_n and $\mathbf{x}_n$ and abbreviation $q_n = q(y_n = 1|\mathbf{x}_n, \mathbf{w}) = \sigma(\mathbf{w}^T\mathbf{x}_n)$.

$$q(y_n|\mathbf{x}_n, \mathbf{w}) = q_n^{y_n}(1 - q_n)^{1-y_n}$$

$$J(\mathbf{w}) = -\langle \ln q(y|\mathbf{x}, \mathbf{w})\rangle \approx -\frac{1}{N}\sum_{n=1}^{N}[y_n \ln q_n + (1 - y_n)\ln(1 - q_n)]$$

Exercise 3: classification

Generalization and overfitting

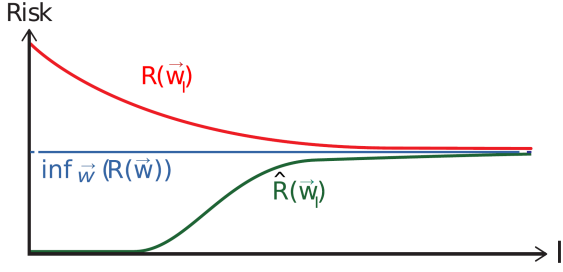


- Training data: Estimate model parameters
- Test data: Estimate risk, i.e. empirical risk
- Validation data: Model selection

The complexity of the model must match the learning problem and the sample size.

The number of training samples affects overfitting

The objective is to predict the target values for new samples, i.e. to minimize the risk $R = \langle J \rangle$.



- The risk R is estimated by the empirical risk $\hat{R}$ given samples.

Final project

Apply the code from example 1 to classify your own dataset with a classification algorithm of your choice.

Day II: Organization

10:00 – 12:00 Tutorial

12:00 – 13:00 Lunch

13:00 – 15:00 Tutorial

15:00 – 15:30 Coffee

15:30 – 17:30 Q&A and/or project

Outline

DAY I: Introduction to machine learning

— lunch break —

Neural networks

Day II: Introduction to language models

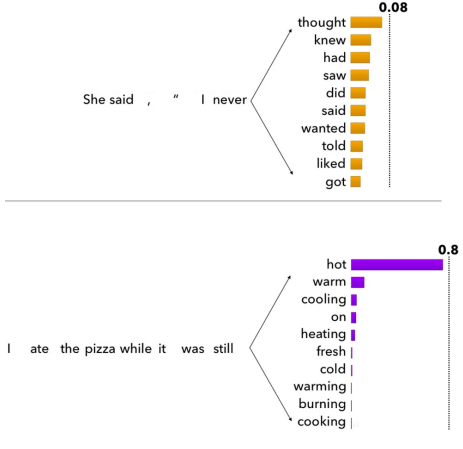
— lunch break —

Huggingface & GPT2

Generative pre-trained transformer

Language models

A probabilistic generative model of natural language.



Language models

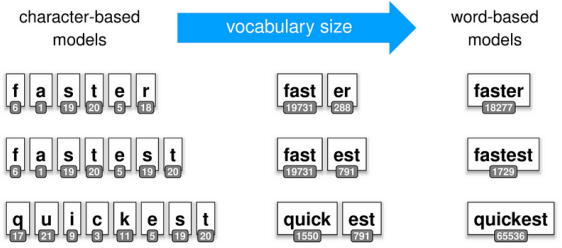
Example: Works of William Shakespeare (`input.txt`)

```
First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him
             ↑
          predict
```

Preprocessing: Tokenization



- Word tokenization
- Sub-word tokenization
- **Character tokenization**

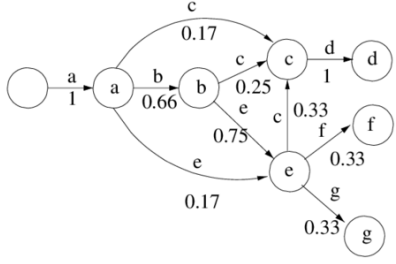
Bi-gram model (Bi-token model)

Training samples (L)

abcd
acd
abef
abeg
abe
aec

Bigram

p(a|<s>)=1
p(b|a)=0.66
p(c|a)=0.17
p(e|a)=0.17
p(c|b)=0.25
p(e|b)=0.75
p(d|c)=1
p(c|e)=0.33
p(f|e)=0.33
p(g|e)=0.33



Multinomial logistic regression

- Generalization of the logistic function to K possible outcomes (predicted token).
- For 1-of-K coding for the next token, e.g. $\mathbf{y} = (0, 1, 0, ..., 0)$

$$q(y_k = 1|\mathbf{x}, \mathbf{W}) = \text{softmax}(\mathbf{w}_k^T \mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{i=1}^{K} e^{\mathbf{w}_i^T \mathbf{x}}}$$

- For N i.i.d. samples $\mathbf{y}_n$ and $\mathbf{x}_n$ and abbreviation $q_{nk} = q(y_{nk} = 1|\mathbf{x}_n, \mathbf{W})$

$$q(\mathbf{y}_n|\mathbf{x}_n, \mathbf{W}) = \prod_{k=1}^{K} q_{nk}^{y_{nk}}$$

$$J = -\langle \ln q(\mathbf{y}|\mathbf{x}, \mathbf{W}) \rangle \approx -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln(q_{nk})$$

Multinomial logistic regression

- Generalization of the logistic function to K possible outcomes (predicted token).
- For 1-of-K coding for the next token, e.g. $\mathbf{y} = (0, 1, 0, ..., 0)$

$$q(y_k = 1|\mathbf{x}, \mathbf{W}) = \text{softmax}(\mathbf{w}_k^T \mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{i=1}^{K} e^{\mathbf{w}_i^T \mathbf{x}}}$$

- For N i.i.d. samples $\mathbf{y}_n$ and $\mathbf{x}_n$ and abbreviation $q_{nk} = q(y_{nk} = 1|\mathbf{x}_n, \mathbf{W})$

$$q(\mathbf{y}_n|\mathbf{x}_n, \mathbf{W}) = \prod_{k=1}^{K} q_{nk}^{y_{nk}}$$

$$J = -\langle \ln q(\mathbf{y}|\mathbf{x}, \mathbf{W}) \rangle \approx -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln(q_{nk})$$

Multinomial logistic regression

- Generalization of the logistic function to K possible outcomes (predicted token).
- For 1-of-K coding for the next token, e.g. $\mathbf{y} = (0, 1, 0, ..., 0)$

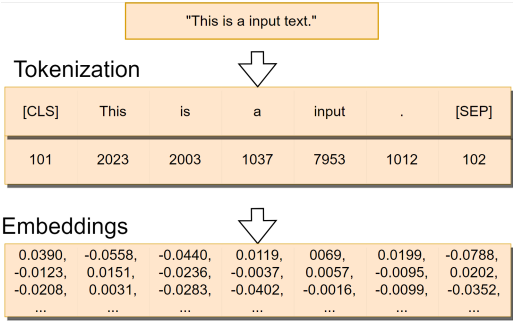
$$q(y_k = 1|\mathbf{x}, \mathbf{W}) = \text{softmax}(\mathbf{w}_k^T \mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{i=1}^{K} e^{\mathbf{w}_i^T \mathbf{x}}}$$

- For N i.i.d. samples $\mathbf{y}_n$ and $\mathbf{x}_n$ and abbreviation $q_{nk} = q(y_{nk} = 1|\mathbf{x}_n, \mathbf{W})$

$$q(\mathbf{y}_n|\mathbf{x}_n, \mathbf{W}) = \prod_{k=1}^{K} q_{nk}^{y_{nk}}$$

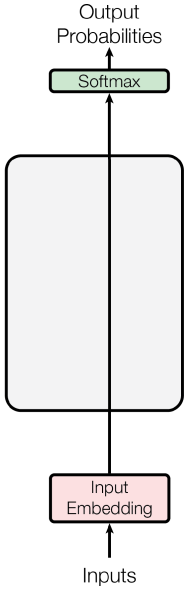
$$J = -\langle \ln q(\mathbf{y}|\mathbf{x}, \mathbf{W}) \rangle \approx -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln(q_{nk})$$

Preprocessing: Embedding

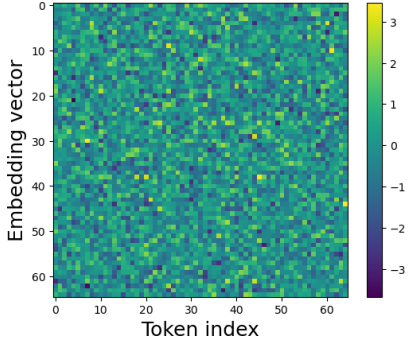


- Every token is replaced by an n_{embd} dimensional embedding vector.
- Embedding vectors are learned.
- Fixed distances between input characters are replaced by learned ones.

Bi-gram model implementation



- Embedding dimension = No. of tokens
- Feed embedding directly into softmax
- Sufficient to learn embedding vectors



n-gram model

Tokenization of `abcdef`

Bi-gram

Query 1

	position	token
1	1	a
2	2	b
3	3	c
4	4	d
5	5	e
6	6	f

Tri-gram

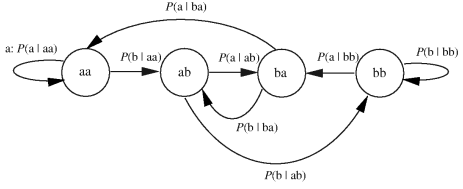
Query 2

	position	token
1	1	ab
2	2	bc
3	3	cd
4	4	de
5	5	ef

Four-gram

Query 3

	position	token
1	1	abc
2	2	bcd
3	3	cde
4	4	def



- Predictions based on the last $n-1$ characters.
- No. tokens = (No. of characters)$^{n-1}$
- Sufficient to learn embedding vectors.

n-gram model

Tokenization of `abcdef`

Bi-gram

Query 1

	position	token
1	1	a
2	2	b
3	3	c
4	4	d
5	5	e
6	6	f

Tri-gram

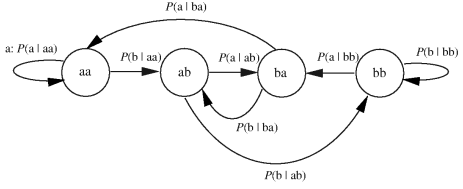
Query 2

	position	token
1	1	ab
2	2	bc
3	3	cd
4	4	de
5	5	ef

Four-gram

Query 3

	position	token
1	1	abc
2	2	bcd
3	3	cde
4	4	def



- Predictions based on the last $n-1$ characters.
- No. tokens = (No. of characters)$^{n-1}$
- Sufficient to learn embedding vectors.

Exercise 4: bigram and n-gram model

Explore overfitting as a function of the number of training samples and n.

Outline

DAY I: Introduction to machine learning

— lunch break —

Neural networks

Day II: Introduction to language models

— lunch break —

Huggingface & GPT2

Generative pre-trained transformer

Outline

DAY I: Introduction to machine learning

— lunch break —

Neural networks

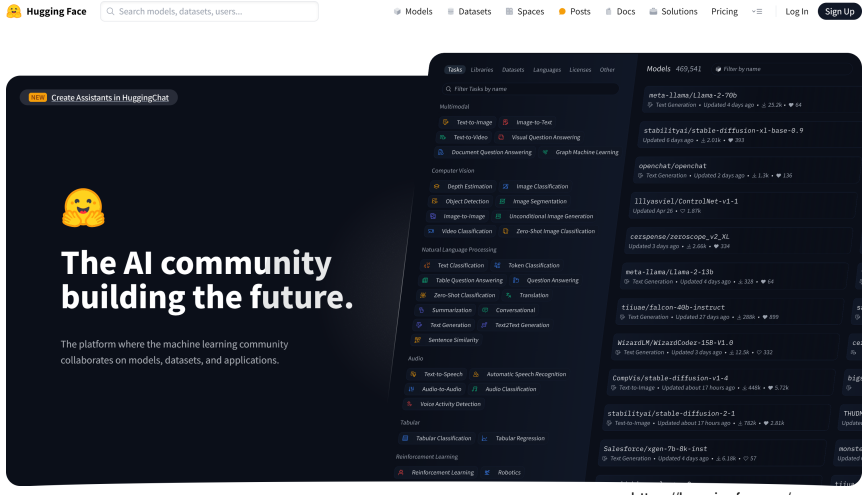
Day II: Introduction to language models

— lunch break —

Huggingface & GPT2

Generative pre-trained transformer

How to get started?



https://huggingface.co/

Working with Hugging Face models

Basic components

- **Tokenizer:** Converts raw text into tokens and input IDs.
- **Model:** Processes input IDs to produce predictions.
- **Pipeline:** A high-level interface for common tasks like text generation.

Working with Hugging Face models

Tokenizer

`AutoTokenizer` automatically loads the right tokenizer for a given model. It handles ...

- Lowercasing (if needed)
- Byte Pair Encoding (BPE), WordPiece, SentencePiece, etc.
- Padding, truncation, special tokens

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("openai-community/gpt2")
tokens = tokenizer("To be, or not to be, that is the question.")
```

Working with Hugging Face models

Models

- Every model on `https://huggingface.co/models` has a **model card**. It tells you ...
  - What the model was trained on
  - Supported tasks
  - Limitations and licenses
  - Example usage
- `AutoModel` and `AutoModelFor...`:
  - `AutoModel` gives you the base model that outputs raw hidden states (useful for feature extraction, embeddings).
  - `AutoModelForCausalLM`, `AutoModelForSequenceClassification`, etc., include task-specific heads.
  - Example: `AutoModelForCausalLM` adds a language modeling head on top of GPT-2.

```
from transformers import AutoModel
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
```

Working with Hugging Face models

Models

- Every model on `https://huggingface.co/models` has a **model card**. It tells you ...
  - What the model was trained on
  - Supported tasks
  - Limitations and licenses
  - Example usage
- `AutoModel` and `AutoModelFor...`:
  - `AutoModel` gives you the base model that outputs raw hidden states (useful for feature extraction, embeddings).
  - `AutoModelForCausalLM`, `AutoModelForSequenceClassification`, etc., include task-specific heads.
  - Example: `AutoModelForCausalLM` adds a language modeling head on top of GPT-2.

```
from transformers import AutoModel
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
```

Working with Hugging Face models

Pipelines

- Pipelines are a great and easy way to use models for inference. Pipelines are objects that abstract most of the complex code from the library, offering a simple API dedicated to several tasks.
- Great for:
  - text-generation
  - sentiment-analysis
  - translation
  - question-answering

```
from transformers import pipeline
generator = pipeline("text-generation", model=model, tokenizer=tokenizer)
print(generator("Long text goes here..."))
```

Exercise 5: Pretrained models from Hugging Face

Investigate the implementation.

Outline

DAY I: Introduction to machine learning

— lunch break —

Neural networks

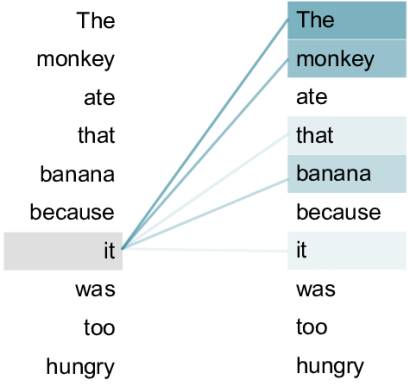
Day II: Introduction to language models

— lunch break —

Huggingface & GPT2

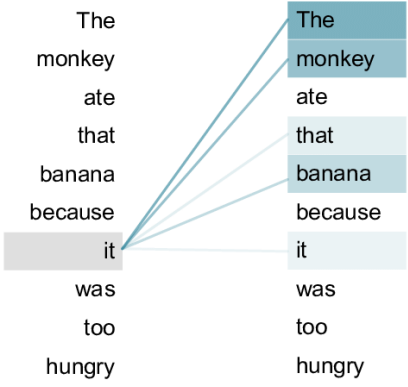
Generative pre-trained transformer

Generative Pre-trained Transformer



- Search for previous tokens (here words) in the sequence that carry information about the next token.
- Provide this information as additional input to the feed-forward network.

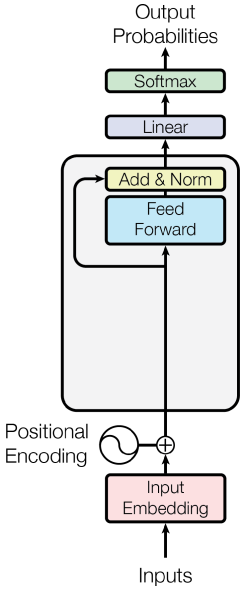
Generative Pre-trained Transformer



Self-attention mechanism:

- Classify which of the previous tokens is important using multinomial logistic regression.
- Provide this information by adding a vector to the input embedding that encodes previous tokens and their importance.

Feed-forward neural network



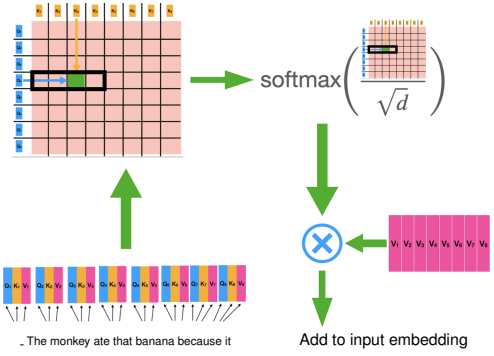
Modifications to the previous logistic regression example:

- Multinomial logistic regression
- ReLU activation functions for hidden (feed-forward) layer neurons.
- Linear skip layer
- Positional encoding of time

Code: Feed-forward neural network

Investigate the implementation of exercise 6.

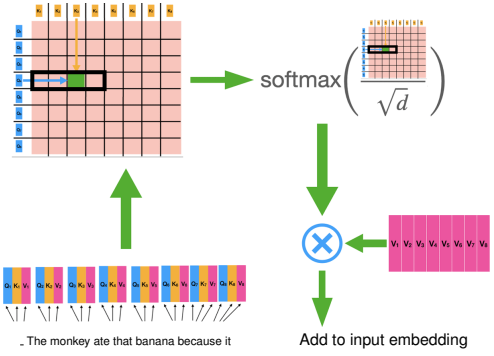
Generative Pre-trained Transformer



Self-attention mechanism:

- For each time step i, three vectors Q_i, K_i, V_i are calculated by three different learned linear transformations of the embedded token.
- $\text{softmax}\left(\frac{Q_i^T K_j}{\sqrt{d}}\right)$ encodes the attention to time step $j \leq i$
- V_j encodes the token in time step $j \leq i$.
- $\text{Attention}(Q_i, K_j, V_j) = \text{softmax}\left(\frac{Q_i^T K_j}{\sqrt{d}}\right) V_j$

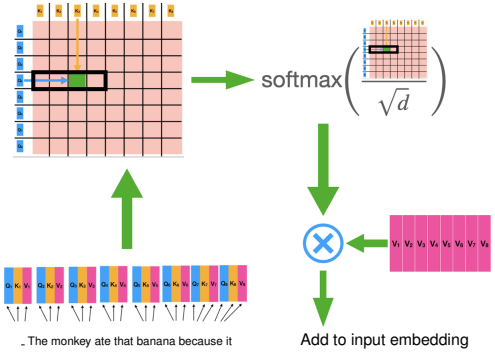
Generative Pre-trained Transformer



Self-attention mechanism:

- For each time step i, three vectors Q_i, K_i, V_i are calculated by three different learned linear transformations of the embedded token.
- $\text{softmax}\left(\frac{Q_i{}^T K_j}{\sqrt{d}}\right)$ encodes the attention to time step $j \leq i$
- V_j encodes the token in time step $j \leq i$.
- $\text{Attention}(Q_i, K_j, V_j) = \text{softmax}\left(\frac{Q_i{}^T K_j}{\sqrt{d}}\right) V_j$

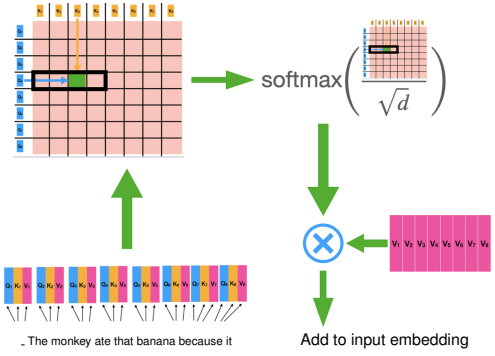
Generative Pre-trained Transformer



Self-attention mechanism:

- For each time step i, three vectors Q_i, K_i, V_i are calculated by three different learned linear transformations of the embedded token.
- $\text{softmax}\left(\frac{Q_i{}^T K_j}{\sqrt{d}}\right)$ encodes the attention to time step $j \leq i$
- V_j encodes the token in time step $j \leq i$.
- $\text{Attention}(Q_i, K_j, V_j) = \text{softmax}\left(\frac{Q_i{}^T K_j}{\sqrt{d}}\right) V_j$

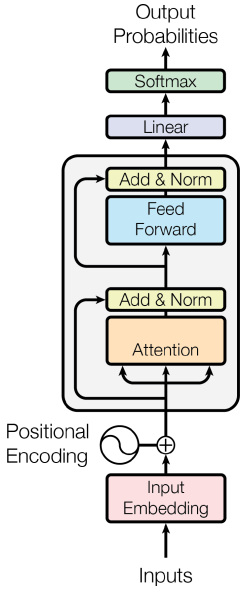
Generative Pre-trained Transformer

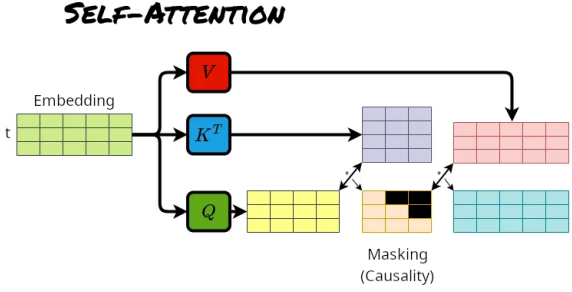


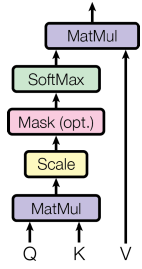
Self-attention mechanism:

- For each time step i, three vectors Q_i, K_i, V_i are calculated by three different learned linear transformations of the embedded token.
- $\text{softmax}\left(\frac{Q_i{}^T K_j}{\sqrt{d}}\right)$ encodes the attention to time step $j \leq i$
- V_j encodes the token in time step $j \leq i$.
- $\text{Attention}(Q_i, K_j, V_j) = \text{softmax}\left(\frac{Q_i{}^T K_j}{\sqrt{d}}\right) V_j$

Self-attention mechanism



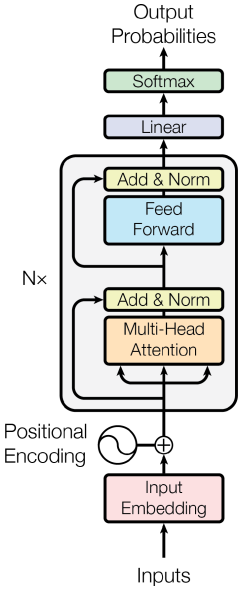


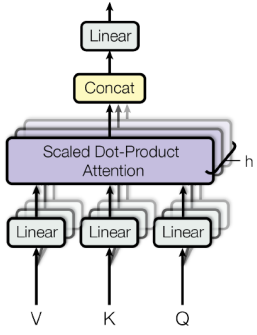


Exercise 6: self-attention

Explore the model performance with and without self-attention and different block sizes.

GPT2





Modifications:

- *h* self-attention heads
- *N* sequential layers
- Dropout regularization

Exercise 6b: GPT-2

Optimize the full GPT-2 model.
(investigate the performance without dropout)

Exercise 7: Fine-tune LLMs with Huggingface

Use the provided code template to fine-tune the GPT-2 model on Huggingface with a text of your choice. The template implements the following steps.

- Load and Preprocess Shakespeare Text
- Load GPT-2 Model and Tokenizer
- Tokenize the Dataset
- Set Up Data Collator and Training Arguments
- Train the Model
- Save the Fine-Tuned Model
- Load the Fine-Tuned Model for Text Generation
- Generate Shakespearean Text

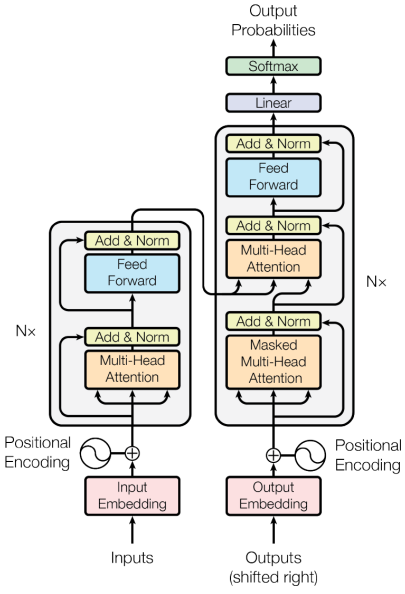
GPT3 model

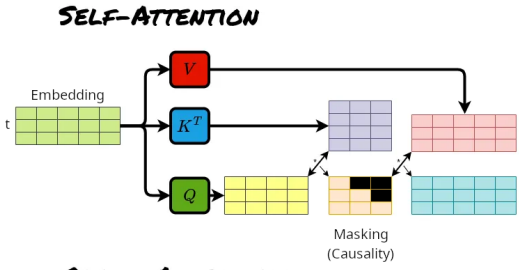
Model Name	n_{params}	n_{layers}	d_{embd}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

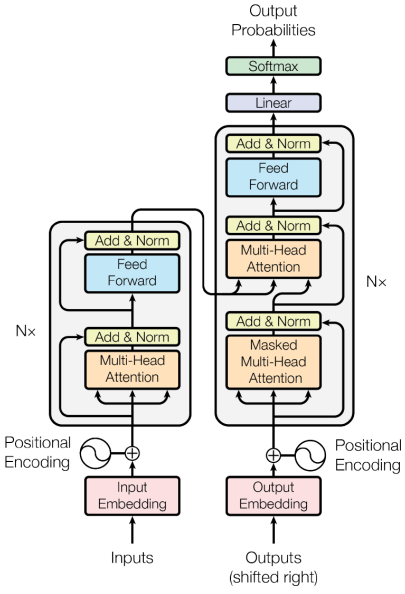
Brown, Tom B., et al. "Language Models are Few-Shot Learners." arXiv preprint:2005.14165 (2020).

Different transformer architectures





Original transformer design



- **Encoder & decoder:** BART
- **Encoder only:** BERT
- **Decoder only:** GPT3

How is ChatGPT trained?

Fine-tuning

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Reinforcement learning

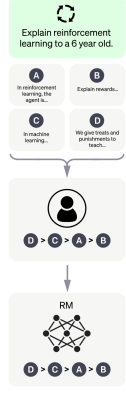
Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

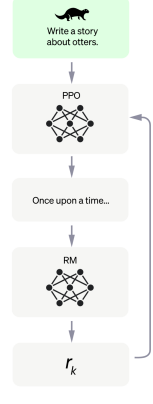
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



https://online-chatgpt.com/

Thank you for your attention!