



MACHINE LEARNING BASICS

Tutorial at Machine Learning Conference

April 10th 2024 | Stefan Häusler, Marina Ganeva | JCNS

Preparation

The screenshot shows a configuration interface for a JupyterLab instance. The top part lists fields: Name (ML Tutorial), Version (JupyterLab - 3.6), System (JUSUF), Account (haeusler1), Project (training2407), and Partition (batch). Below these fields is a preview of a browser window titled "Jupyter-JSC". The browser displays a message: "Jupyter-JSC is now running or We have moved Jupyter-JSC to the JSC-Cloud. A You can configure your existing JupyterLabs by exp...". The browser also shows a sidebar with a "ML Tutorial" section containing tabs for "Lab Config", "Resources", "System", "Kernels and Extensions", "Account", and "Logs". The "Lab Config" tab is selected, showing the same configuration values as the interface above. At the bottom of the browser window are "Save", "Reset", and "Delete" buttons.

github.com/neutron-simlab/ml-basics-gpt2

Preparation

The image shows two side-by-side browser windows. The left window is the Jupyter-JSC interface, which displays a message stating "Jupyter-JSC is now running on JSC-Cloud" and "We have moved Jupyter-JSC to the JSC-Cloud. All data from HDF-Cloud was moved to JSC-Cloud." It includes a table for configuring existing JupyterLabs and a sidebar for managing resources like kernels and extensions. The right window is a JupyterLab interface showing a list of available kernels: Python 3 (system), R 4.2.1, Julia 1.8.3, C++17, Jupyter-0.18.0, Octave-6.2.0, PyData2023, Python-2023, PyGromacs-2023, PyVizualization-2023, Python 3 (system), R 4.2.1, Julia 1.8.3, C++17, Jupyter-0.18.0, Octave-6.2.0, PyData2023, Python-2023, PyGromacs-2023, PyVizualization-2023, and Other.

github.com/neutron-simlab/ml-basics-gpt2

Outline

Introduction to machine learning

Neural networks

— lunch break —

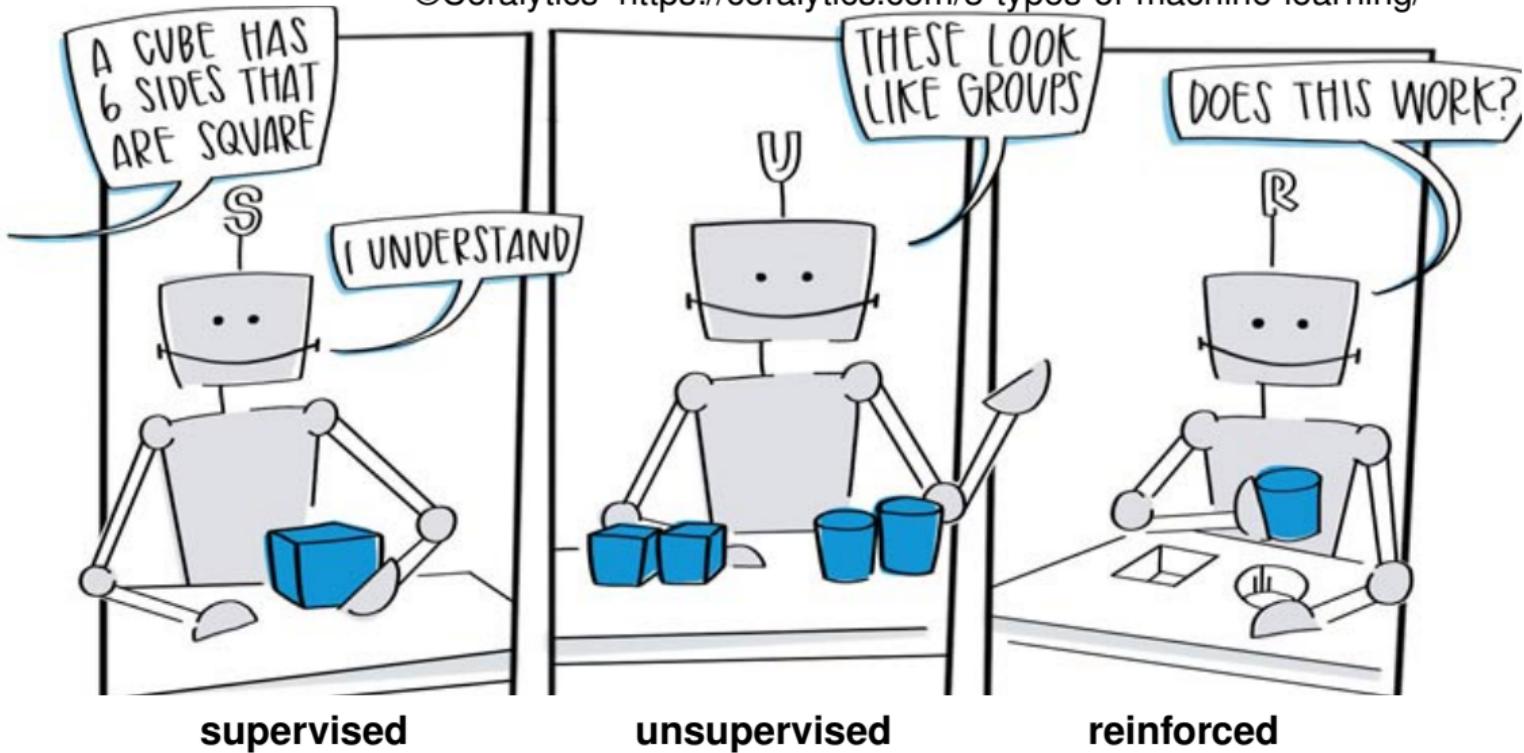
Introduction to language models

Generative pre-trained transformer

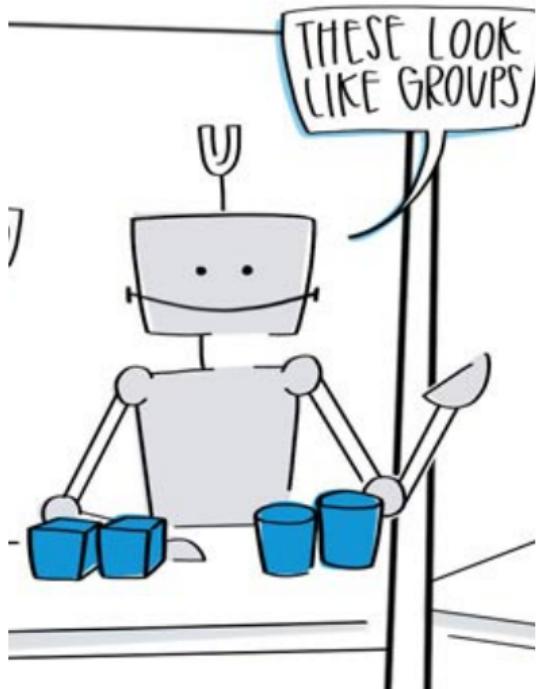
What do you understand under “Machine Learning?”

Machine Learning

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



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

Clustering

Organize similar items into groups

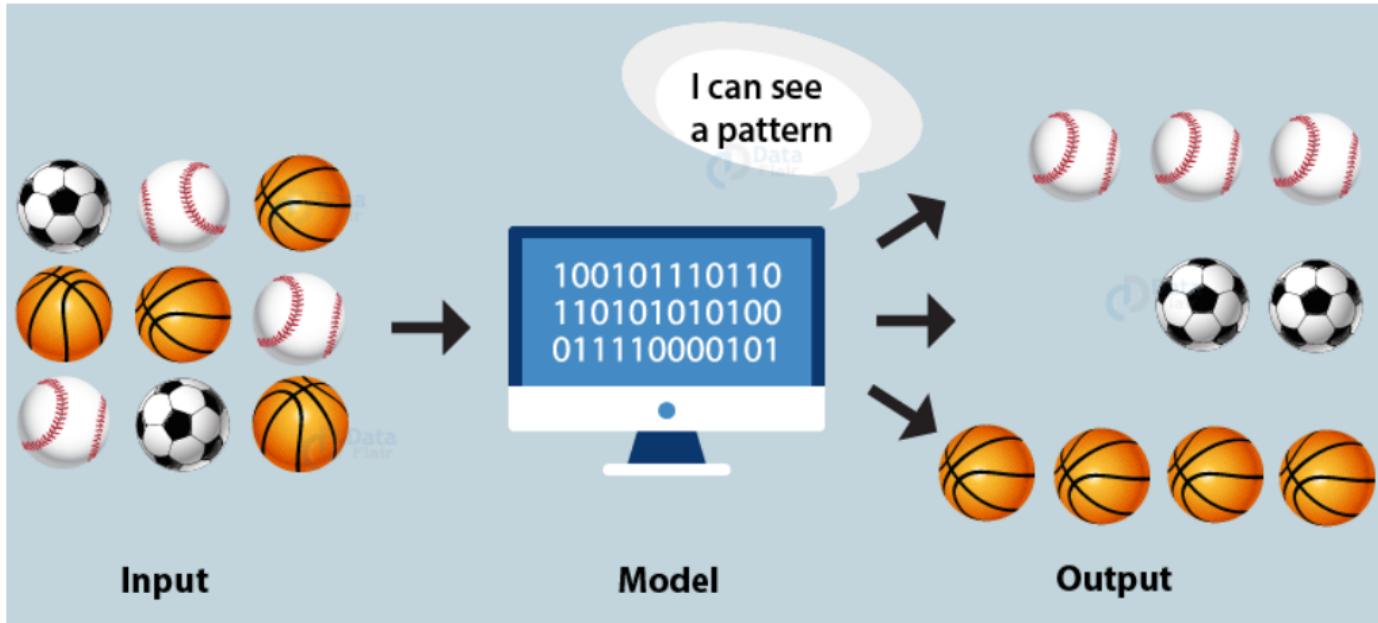
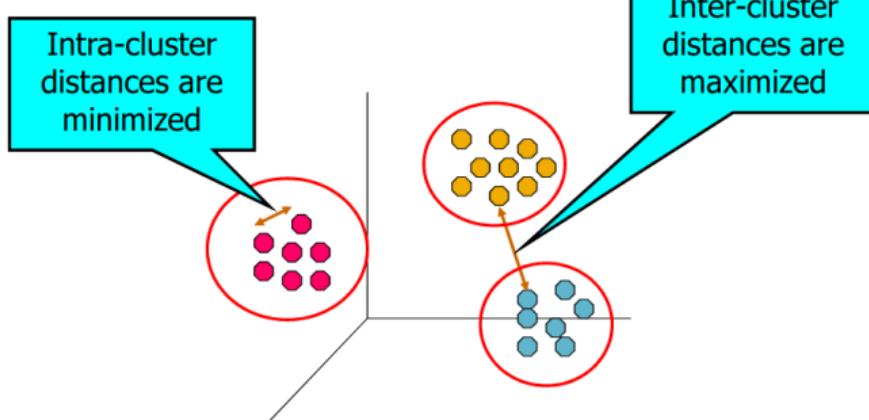


Image from <https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/08/introduction-to-clustering.png>

Clustering



Problem statement

- **Given:** X , but no y .
- **Desired:** separate X into K groups of similar objects.
- **Required:** defined distance $\rho(x_i, x_j)$ between objects.

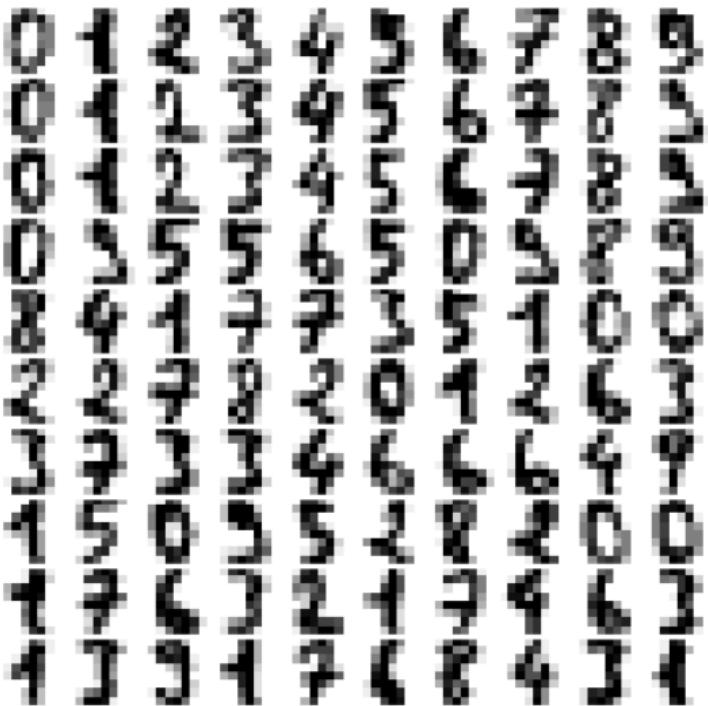
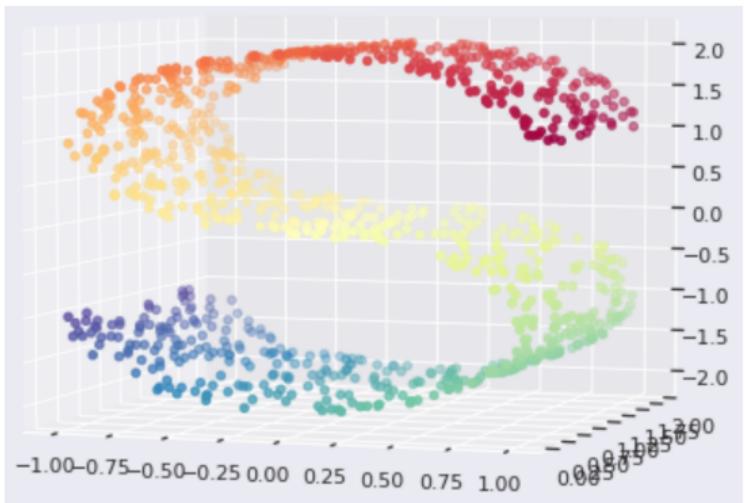
What to fit?

For example, model parameters can be chosen to minimize the ratio:

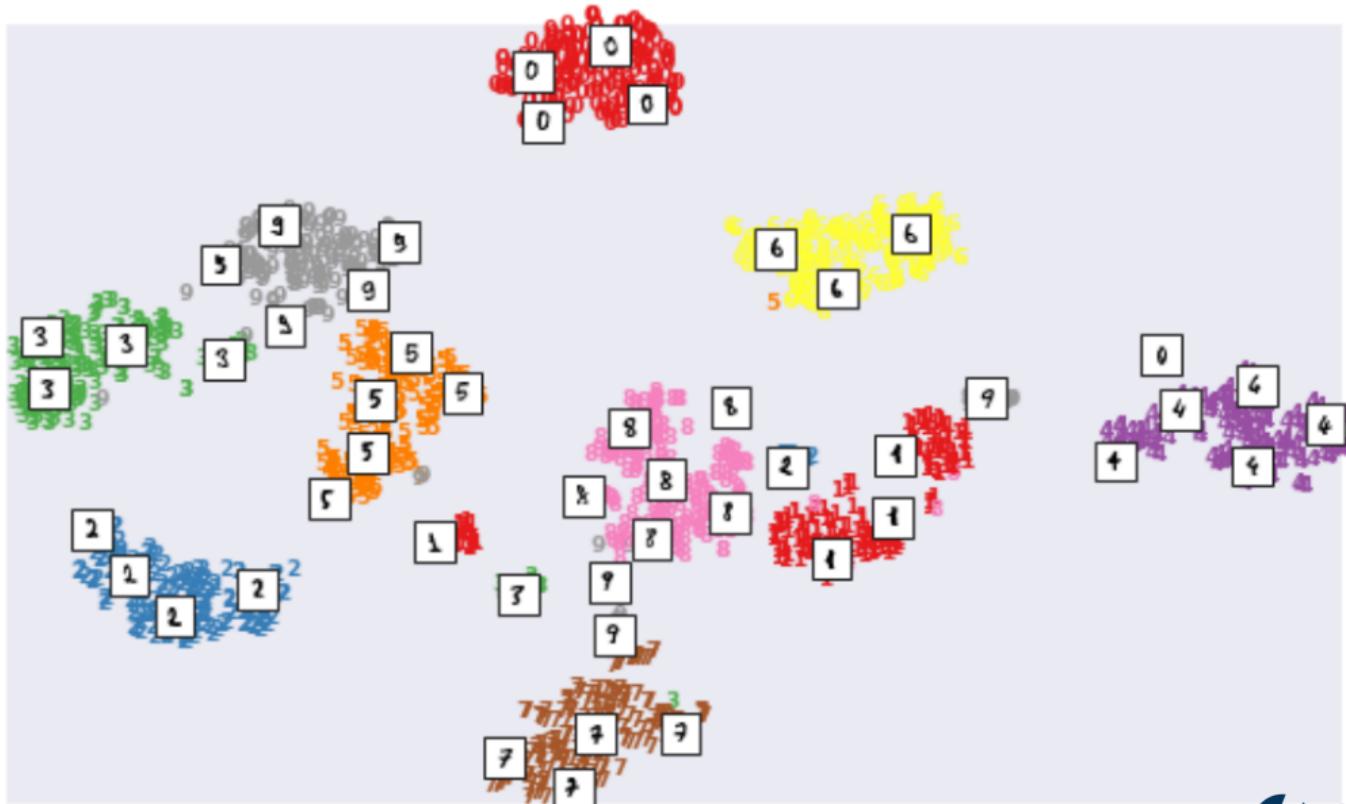
$$\frac{\text{distance between objects in cluster}}{\text{distance between clusters}}$$

Manifold learning

How many dimensions would be enough to represent the data shown in figures below?



Manifold learning



t-distributed Stochastic Neighbor Embedding (t-SNE)

Problem statement: convert a high-dimensional dataset X into low-dimensional data Y . Distance between neighboring data points should not increase.

t-SNE Algorithm starts by converting the high-dimensional Euclidean distances between data points x_i into conditional probabilities p_{ij} that represent similarities:

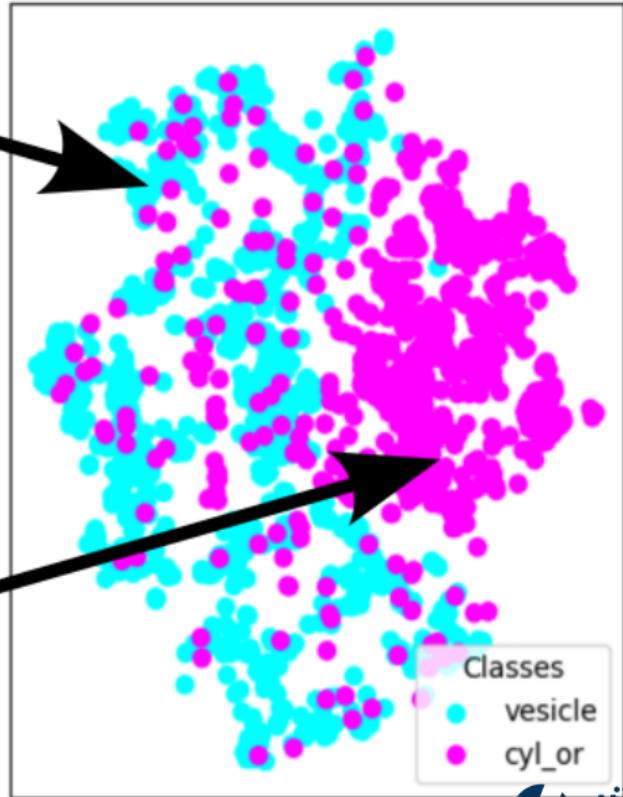
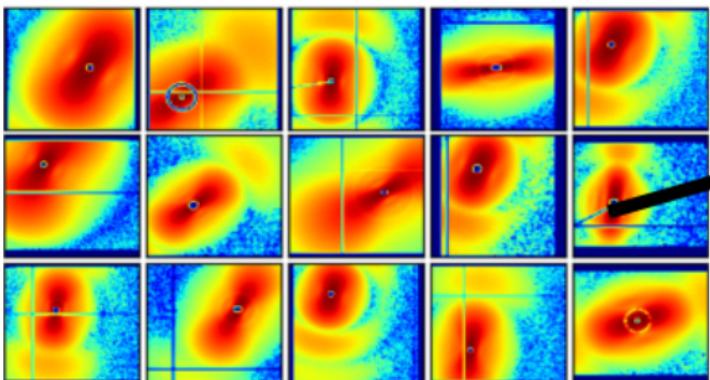
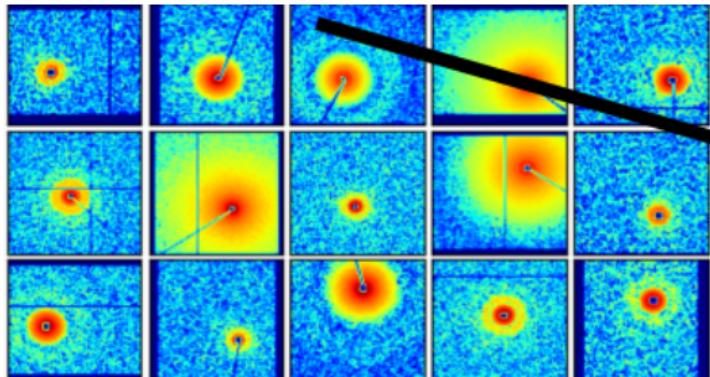
$$p_{ij} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

For the low-dimensional counterparts y_i and y_j of the high-dimensional data points x_i and x_j , it is possible to compute a similar symmetrized conditional probability, q_{ij} :

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq j} (1 + ||y_k - y_j||^2)^{-1}}$$

Thus, t-SNE looks for the **conditional probability distribution** q in space Y , which contains amount of information as close as possible to the initial distribution p in space X .

t-SNE: reducing dimensionality of SAS data

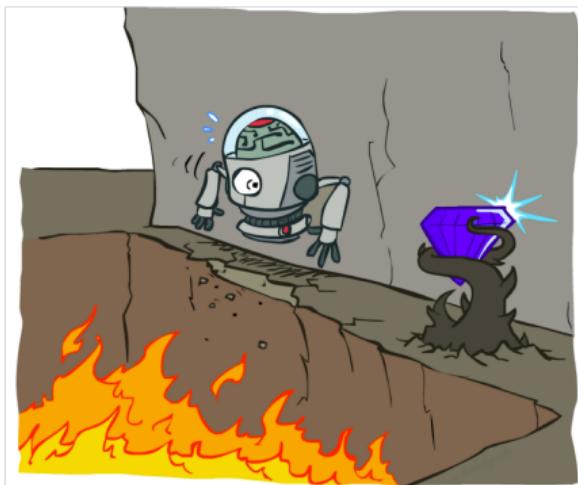


Reinforcement learning

Generally speaking, it is a computational approach to **learning from interaction with the environment**.

Used everywhere where **autonomous decision making** is required. For example:

- Teach robot to walk
- Play board games (chess, go), play computer games
- For Autonomous Control of a Particle Accelerator
- Teach helicopter to fly autonomously and make maneuvers
- ...



Reinforcement learning: how it works

House (environment)

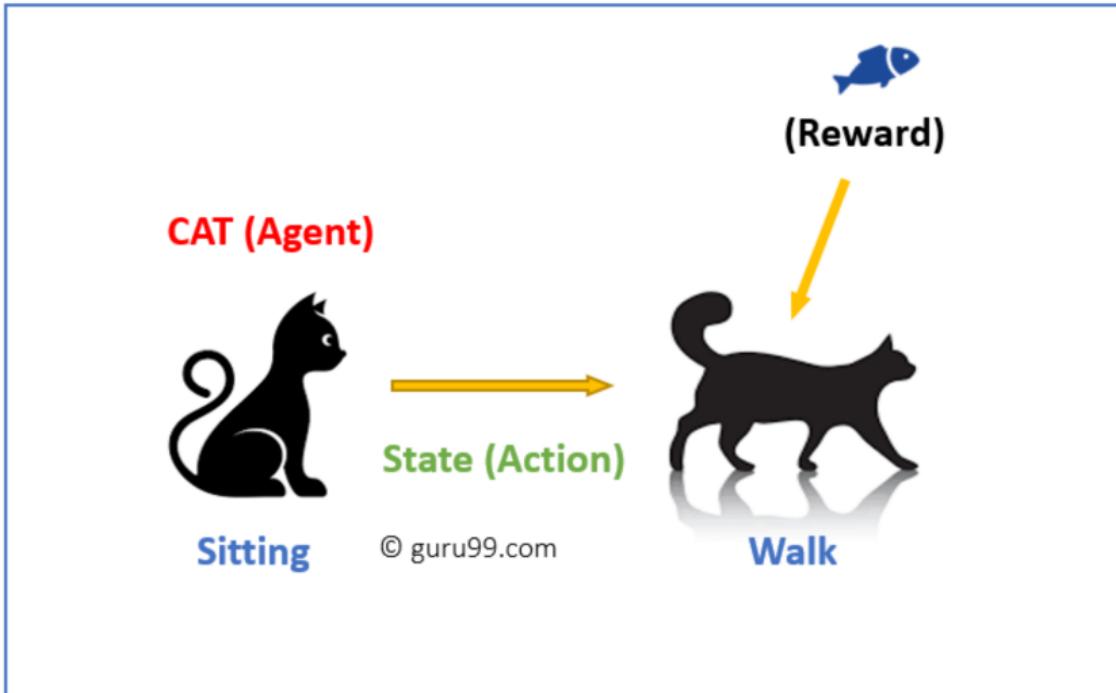
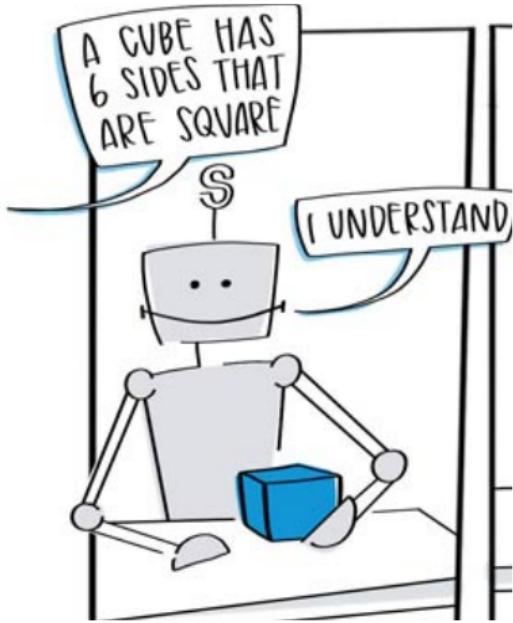


Image from <https://www.guru99.com/reinforcement-learning-tutorial.html>

Reinforcement learning vs. unsupervised learning

Unsupervised learning	Reinforcement learning
Learning underlying data structure	Learning optimal strategy by trial and error
No feedback required	Needs feedback on agent's own actions
Model does not affect the input data	Agent can affect its own observations
Time does not play a role	Time is important

Supervised learning



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

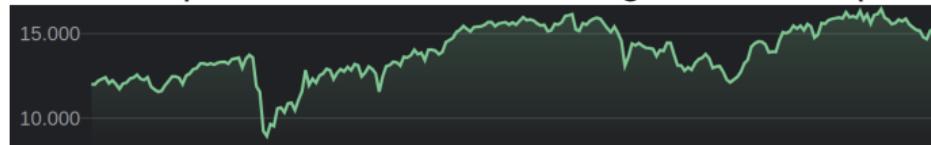
Classification

Goal is to assign the input data to specific categories.



Regression

Goal is to predict a numerical value given some inputs.



Learning data underlying patterns and relationships

Supervised learning: problem statement

Given $X = \begin{bmatrix} 1 & x_{11} & \dots & x_{1m} \\ 1 & x_{21} & \dots & x_{2m} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \dots & x_{nm} \end{bmatrix}$ and $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$ find $w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$

such that the cost function $J(w)$ is minimized

Example cost function for linear regression:

$$J(w) = \frac{1}{n} ||X \cdot w - y||^2 \rightarrow \min_w J(w)$$

Machine Learning — what do we learn?

Outline

Introduction to machine learning

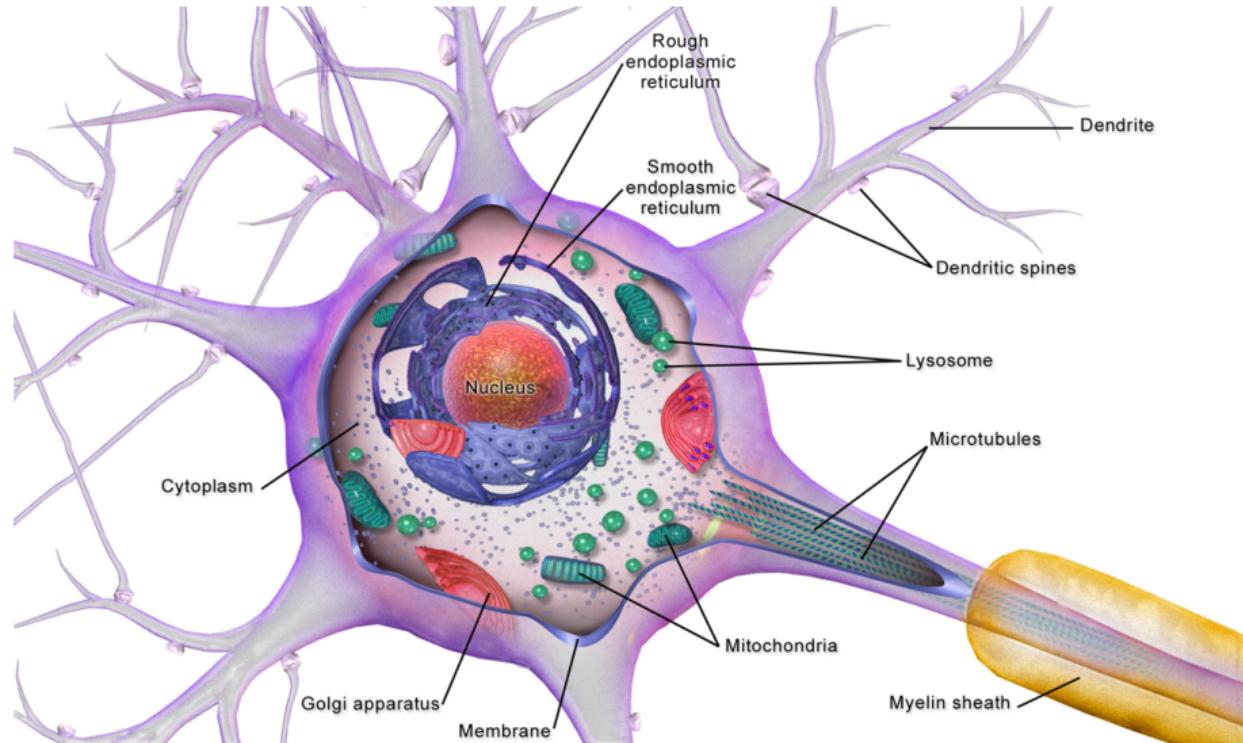
Neural networks

— lunch break —

Introduction to language models

Generative pre-trained transformer

Brain neuron



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

Neuron model

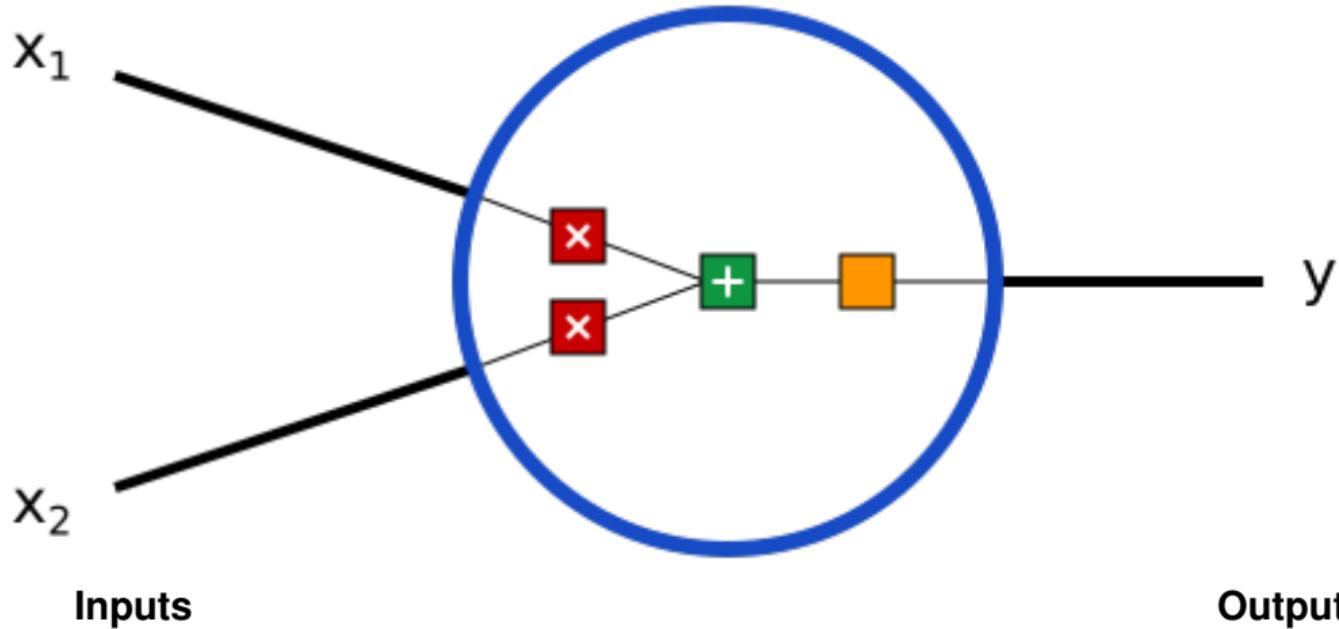


Image from <https://victorzhou.com/blog/intro-to-neural-networks/>

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,

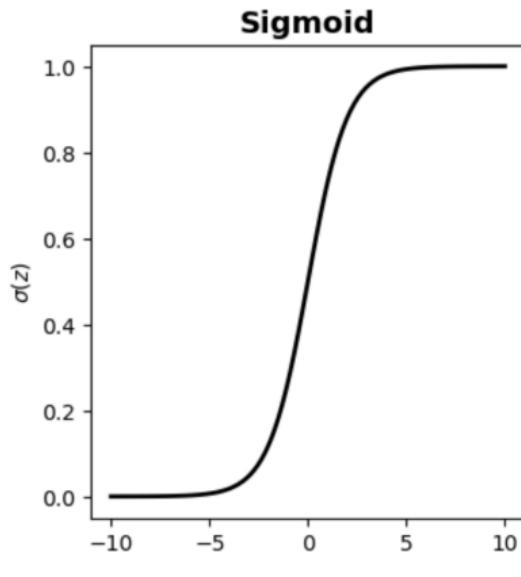
w — vector of weights (model parameters),

x — features vector, $x_0 = 1$

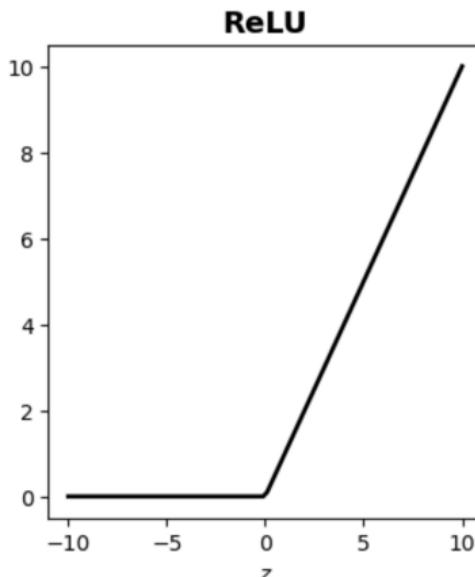
For $\sigma = id$, i.e. $a(x, w) = w^T x$ — *linear regression*.

Activation functions

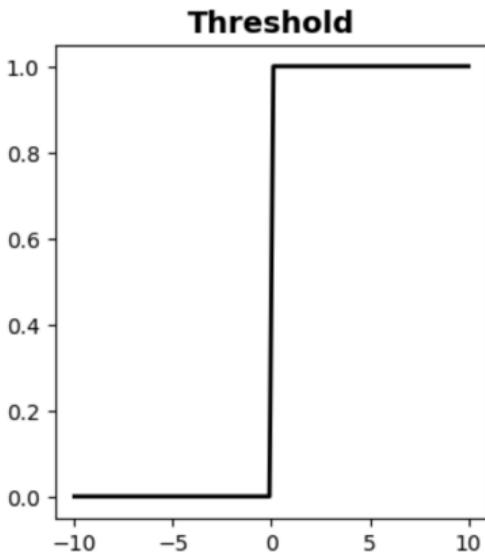
Activation functions are real monotonic functions, preferably differentiable



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

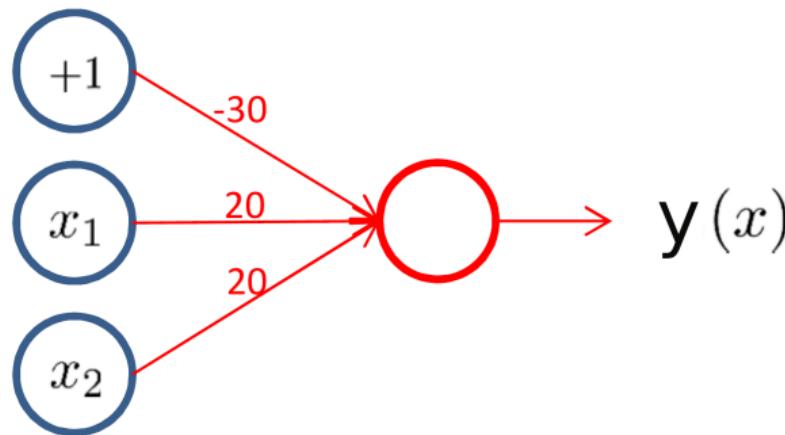


$$\sigma(z) = \max(0, z)$$



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

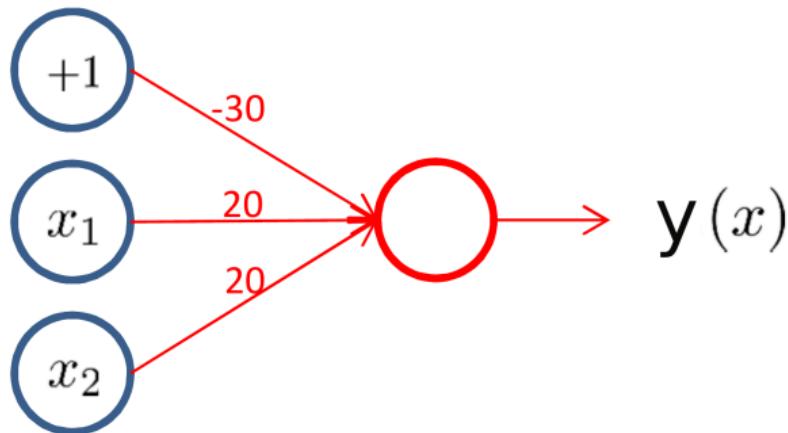
Example: x_1 AND x_2



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

where σ is a threshold activation function

Example: x_1 AND x_2

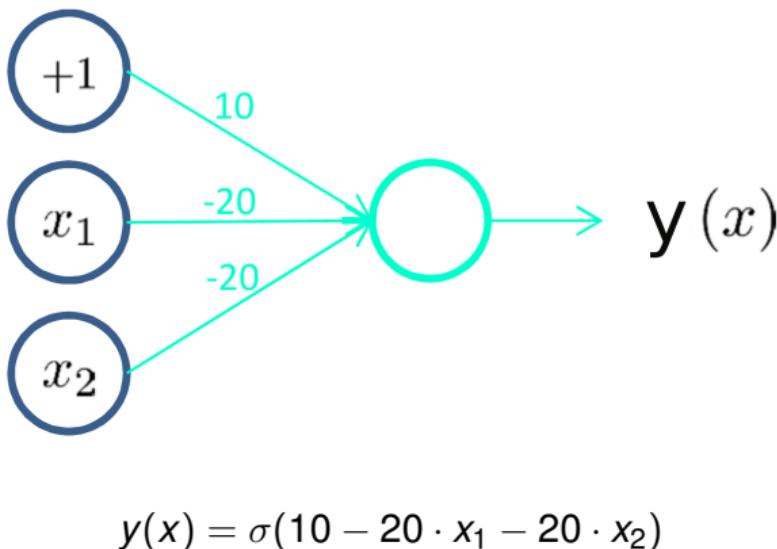


$$y(x) = \sigma(-30 + 20 \cdot x_1 + 20 \cdot x_2)$$

where σ is a threshold activation function

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

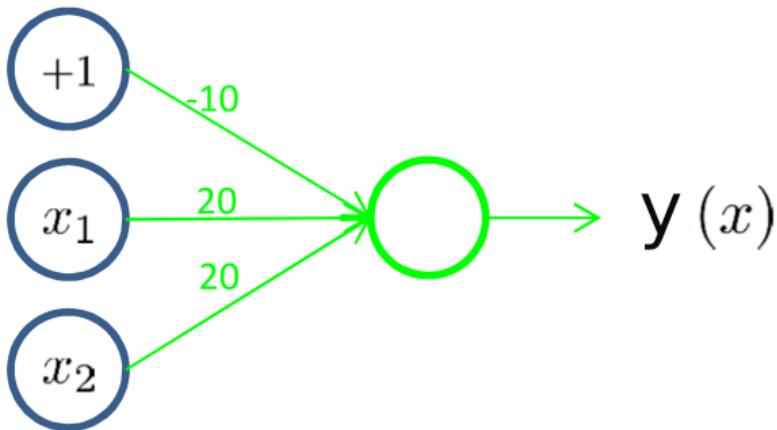
Example: (NOT x_1) AND (NOT x_2)



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

where σ is a threshold activation function

Example: x_1 OR x_2

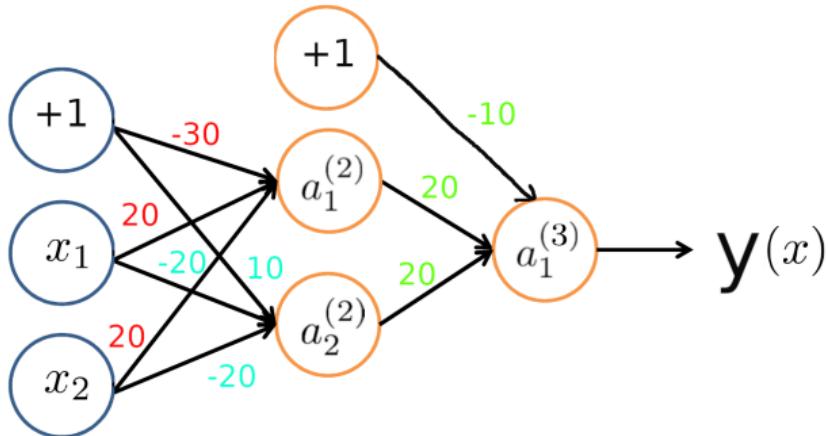


$$y(x) = \sigma(-10 + 20 \cdot x_1 + 20 \cdot x_2)$$

where σ is a threshold activation function

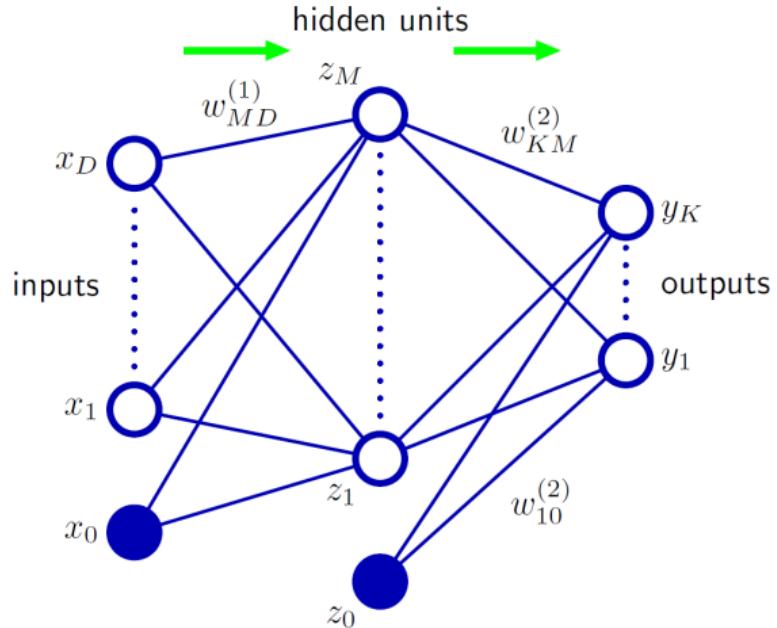
x_1	x_2	$y(x)$
0	0	$\sigma(-10) = 0$
0	1	$\sigma(+10) = 1$
1	0	$\sigma(+10) = 1$
1	1	$\sigma(+30) = 1$

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

Neural networks



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, $x_0 = 1$

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

Universal function approximation

- **Boolean networks:**

Every Boolean function $f : \{0, 1\}^D \rightarrow \{0, 1\}^K$ 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 $f : \{0, 1\}^D \rightarrow \{0, 1\}^K$ 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 $f : [0, 1]^D \rightarrow \mathbb{R}^K$ 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 $f : \{0, 1\}^D \rightarrow \{0, 1\}^K$ 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 $f : [0, 1]^D \rightarrow \mathbb{R}^K$ 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:**

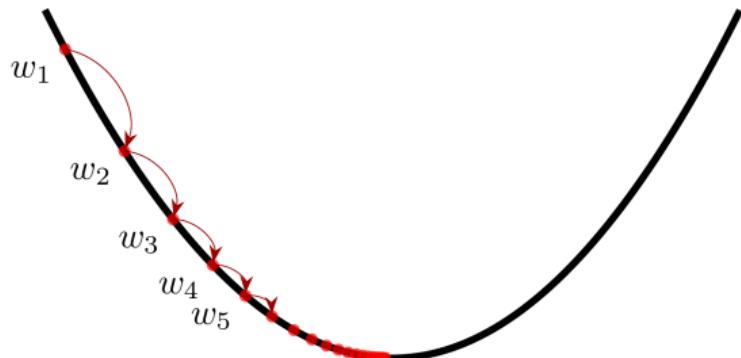
For every continuous function $f : [0, 1]^D \rightarrow \mathbb{R}^K$ for $D, K \in \mathbb{N}$ and ϵ exists a neural network of width $K + N$ with **multiple hidden layers** and **ReLU** activations such that $\|f(\mathbf{x}) - N(\mathbf{x})\| \leq \varepsilon$ for all $\mathbf{x} \in [0, 1]^D$.

Gradient descent

The view from the Wank to Garmisch-Partenkirchen. ©Wolfgang Zwanzger



Gradient descent



Repeat:

$$w = w - \alpha \nabla_w J(w)$$

where

$$\nabla_w J(w) = \frac{2}{n} X^T \cdot (X \cdot w - y)$$

and α is a learning parameter.

For 1D linear problem

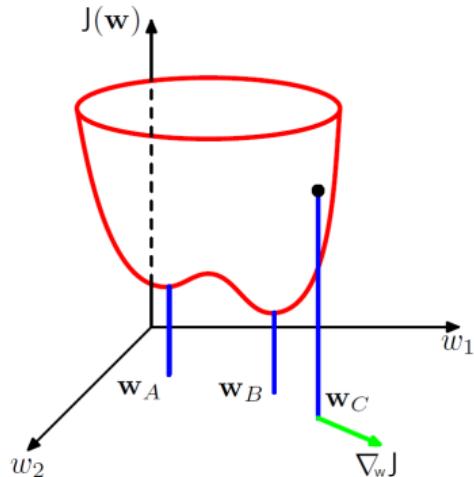
$$w_0 = w_0 - \alpha \frac{\partial J(w_0, w_1)}{\partial w_0},$$

$$\frac{\partial J(w_0, w_1)}{\partial w_0} = \frac{2}{n} \sum_{i=1}^n (w_0 + w_1 x_i - y_i)$$

$$w_1 = w_1 - \alpha \frac{\partial J(w_0, w_1)}{\partial w_1},$$

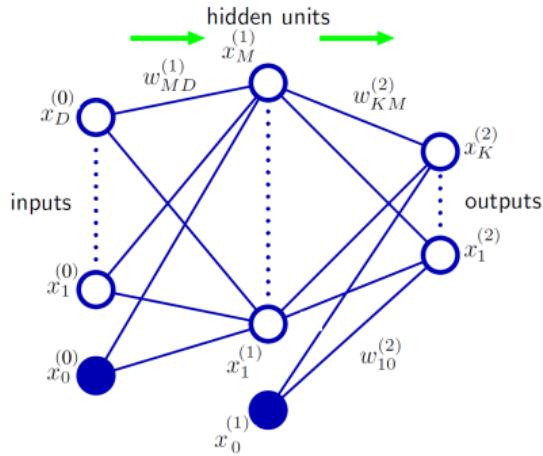
$$\frac{\partial J(w_0, w_1)}{\partial w_1} = \frac{2}{n} \sum_{i=1}^n (w_0 + w_1 x_i - y_i) x_i$$

Gradient descent optimization



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

Gradient descent optimization



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

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

Gradient descent optimization

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

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

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

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

Efficient implementation: Backpropagation algorithm

Backpropagation algorithm

Gradients of the weights in layer 2

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

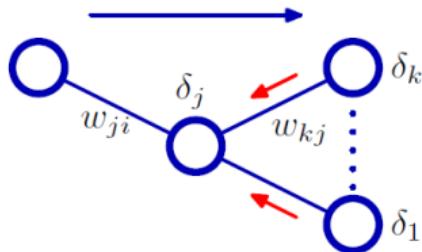
and layer 1 after application of the "**chain rule**"

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

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 δ s 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 δ s 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 δ s 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 .

Preparation

The screenshot shows a web-based interface for managing JupyterLab environments. At the top, there's a header with tabs for 'Jupyter-JSC' and 'JupyterLab - 3.6'. Below the header, there are several input fields and dropdown menus:

- Name:** ML Tutorial
- Version:** JupyterLab - 3.6
- System:** JUSUF
- Account:** haeusler1
- Project:** training2407
- Partition:** batch

Below these settings, there's a detailed configuration panel for the 'ML Tutorial' lab:

Lab Config	ML Tutorial
Name	ML Tutorial
Version	JupyterLab - 3.6
Resources	30
System	JUSUF
Kernels and Extensions	haeusler1
Account	training2407
Logs	batch
Project	None
Partition	None
Reservation	None

At the bottom of the configuration panel are three buttons: 'Save' (green), 'Reset' (red), and 'Delete' (red).

github.com/neutron-simlab/ml-basics-gpt2

Preparation

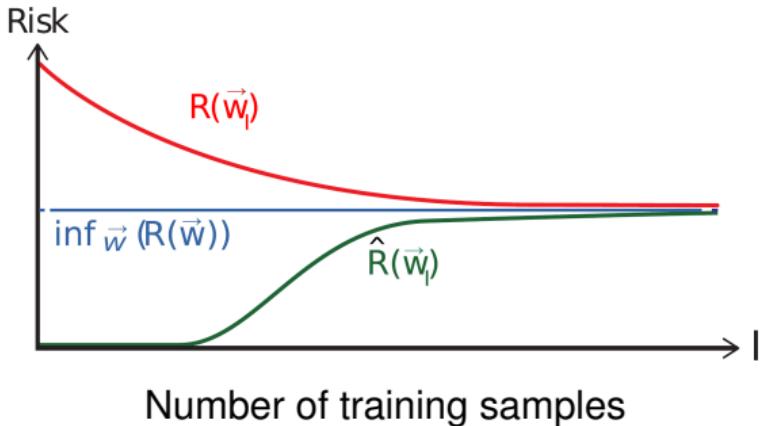
The image shows two side-by-side browser windows. The left window is the Jupyter-JSC interface, which displays a message stating "Jupyter-JSC is now running on JSC-Cloud" and "We have moved Jupyter-JSC to the JSC-Cloud. All data from HDF-Cloud was moved to JSC-Cloud." It includes a table for configuring existing JupyterLabs and a sidebar for managing resources like kernels and extensions. The right window is a JupyterLab interface showing a list of available kernels: Python 3 (system), R 4.2.1, Julia 1.8.3, C++17, Jupyter-0.18.0, Octave-6.2.0, PyData2023, Python-2023, PyGromacs-2023, PyVizualization-2023, Python 3 (system), R 4.2.1, Julia 1.8.3, C++17, Jupyter-0.18.0, Octave-6.2.0, PyData2023, Python-2023, PyGromacs-2023, PyVizualization-2023, and Other.

github.com/neutron-simlab/ml-basics-gpt2

Exercise 1: backprop/regression

Learning objective

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

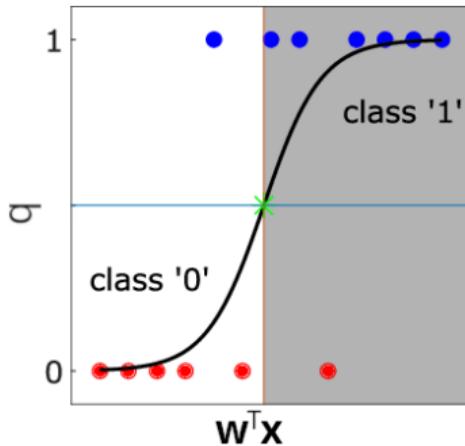


- The risk R is estimated by the empirical risk \hat{R} given samples.

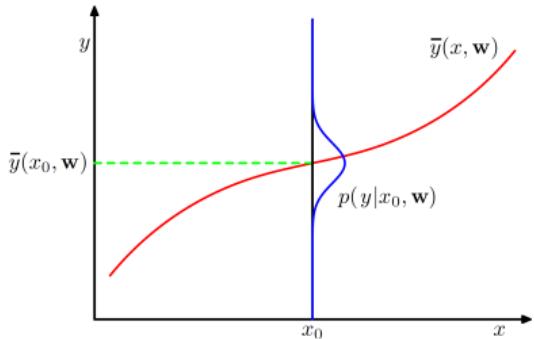
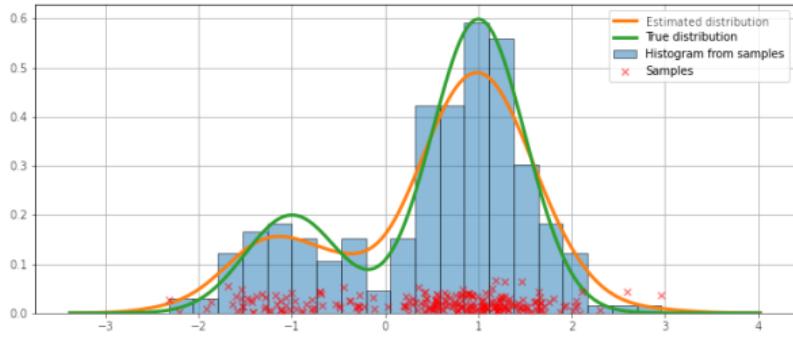
Logistic regression

Binary classification

- Two-class classification problem for $y \in \{0, 1\}$
- Parametric estimate $q(y = 1 | \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x})$, where σ is the logistic sigmoid function.



Density estimation



- 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 cost function that enables gradient descent?

Density estimation

Kullback–Leibler (KL) divergence:

Measures how one probability distribution p is different from a second, reference probability distribution q . For densities p and q

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

$D_{\text{KL}}(p \parallel q) \geq 0$ 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 $D_{\text{KL}}(p \parallel q)$.

Logistic regression

Cross-entropy error function

For n i.i.d. samples y_i and \mathbf{x}_i and abbreviation $q_i = q(y_i = 1 | \mathbf{x}_i, \mathbf{w})$.

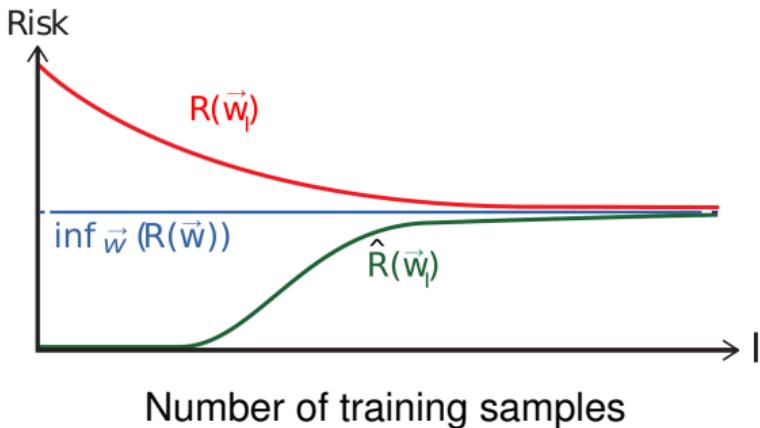
$$q(y_i | \mathbf{x}_i, \mathbf{w}) = q_i^{y_i} (1 - q_i)^{1 - y_i}$$

$$J(\mathbf{w}) = -\langle \ln q(y | \mathbf{x}, \mathbf{w}) \rangle = -\frac{1}{n} \sum_{i=1}^n [y_i \ln q_i + (1 - y_i) \ln(1 - q_i)]$$

Exercise 2: classification

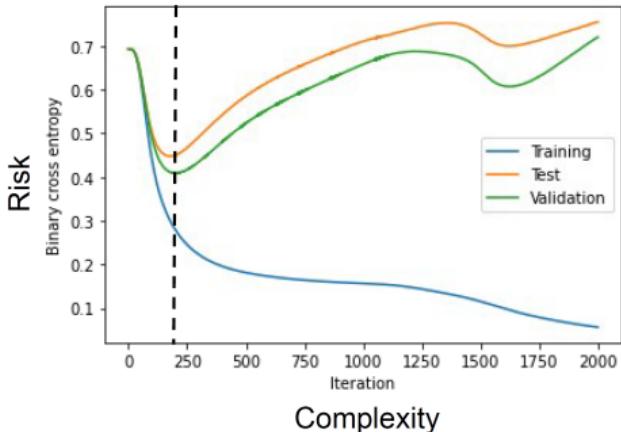
Learning objective

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.

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..

Outline

Introduction to machine learning

Neural networks

— lunch break —

Introduction to language models

Generative pre-trained transformer

Preparation

The screenshot shows the Jupyter-JSC web interface for configuring a JupyterLab instance. The configuration fields are as follows:

Name	ML Tutorial
Version	JupyterLab - 3.6
System	JUSUF
Account	haeusler1
Project	training2407
Partition	gpus
ML Tutorial	
Lab Config	Name: ML Tutorial Version: JupyterLab - 3.6
Resources	System: JUSUF Account: haeusler1 Project: training2407 Partition: gpus Reservation: None
Kernels and Extensions	
Logs	

At the bottom, there are buttons for Save, Reset, and Delete.

github.com/neutron-simlab/ml-basics-gpt2

Preparation

The image shows two side-by-side screenshots. On the left is the Jupyter-JSC interface, which displays a message stating "Jupyter-JSC is now running on JSC-Cloud" and "We have moved Jupyter-JSC to the JSC-Cloud. All data from HDF-Cloud was moved to JSC-Cloud." Below this, there is a table titled "You can configure your existing JupyterLabs by expanding the corresponding table row." The table has columns "Name" and "Configuration". A single row is expanded, showing "ML Tutorial" with "System: JUSUF" and "Runtime (min): 30". A sidebar on the left contains tabs for "Lab Config", "Resources", "Kernels and Extensions", "Account", "Logs", and "Project". At the bottom are "Save" and "Reset" buttons. On the right is a screenshot of the JupyterLab launcher window. It shows a grid of icons for various kernels and tools, including Python 3, R, Julia 1.8.3, C++11, Octave 8.1.0, PyTorch 2023.3, PyTesseract 2023.3, PyGuanche 2023.3, PyEvaluation 2023.3, Jupyter Notebook, Jupyter Console, and various file types like JSON, LaTeX, Test File, Typicode File (Playground), Markdown File, Julia File, Python File, R File, and View Continuous Help. The status bar at the bottom indicates "User: j40.80 / 257187.30 MB" and "English (United States) Launcher".

github.com/neutron-simlab/ml-basics-gpt2

Outline

Introduction to machine learning

Neural networks

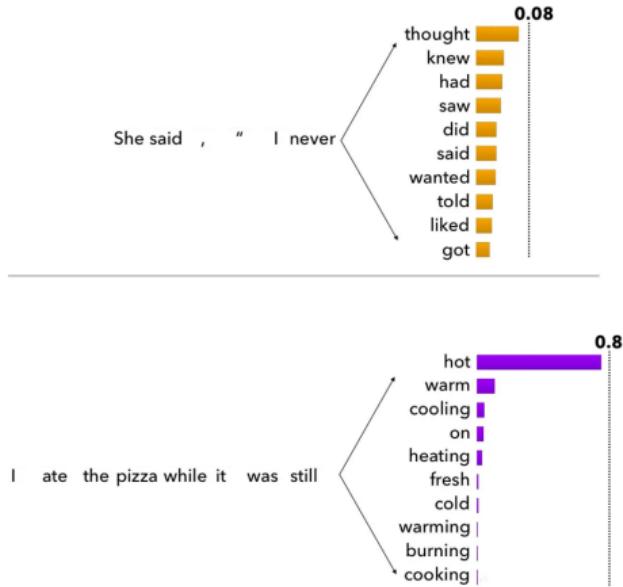
— lunch break —

Introduction to language models

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.

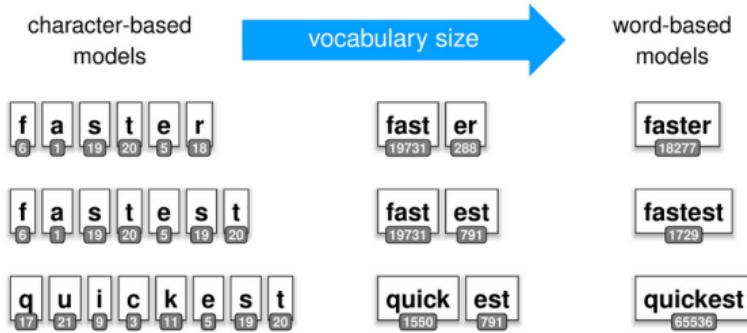
First Citizen:

Let us kill him

↑

predict

Preprocessing: Tokenization



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

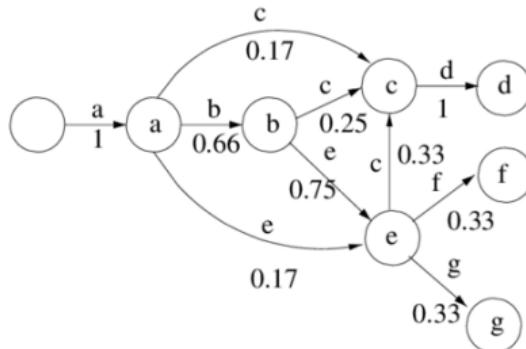
Bi-gram model

Training samples (L)

abcd
acd
abef
abeg
abe
aec

Bigram

$p(al < s>) = 1$	$p(dlc) = 1$
$p(bla) = 0.66$	$p(cle) = 0.33$
$p(cla) = 0.17$	$p(fle) = 0.33$
$p(ela) = 0.17$	$p(gle) = 0.33$
$p(clb) = 0.25$	
$p(elb) = 0.75$	



Multinomial logistic regression

- Generalization of the logistic function to K possible outcomes (predicted token).

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. $\mathbf{y} = (0, 1, 0, \dots, 0)$

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

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. $\mathbf{y} = (0, 1, 0, \dots, 0)$

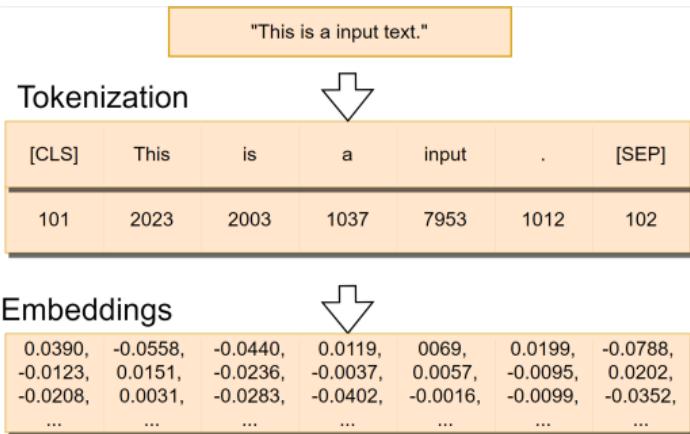
$$q(y_k = 1 | \mathbf{x}, \mathbf{W}) = \text{softmax}(\mathbf{w}_k^T \mathbf{x}) = \frac{e^{\mathbf{w}_k^T \mathbf{x}}}{\sum_{i=1}^K e^{\mathbf{w}_i^T \mathbf{x}}}$$

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

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

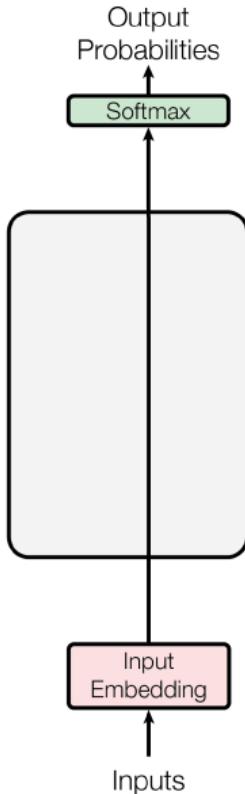
$$J = -\langle \ln q(\mathbf{y} | \mathbf{x}, \mathbf{W}) \rangle = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K y_{ik} \ln(q_{ik})$$

Preprocessing: Embedding

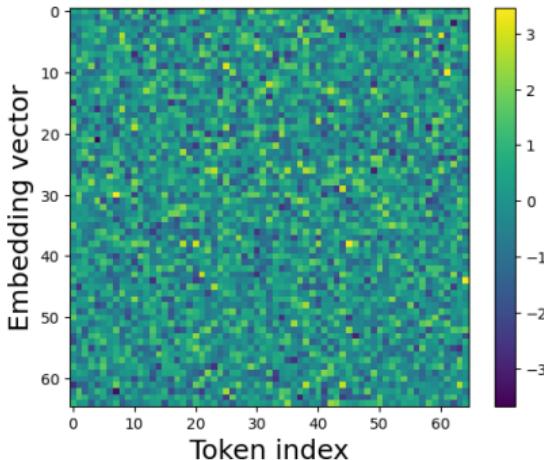


- Every token is replaced by an n_{embd} dimensional embedding vector.
- Embedding vectors are learned.
- Fixed distances between input characters are replaced by learned ones.

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

Tri-gram

Four-gram

Query 1

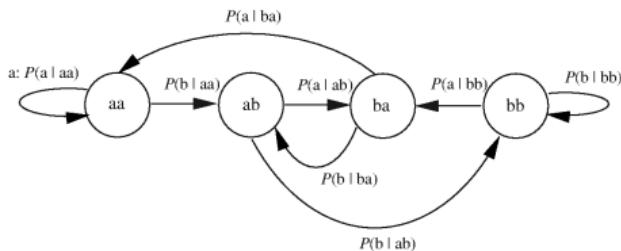
	position	token
1	1	a
2	2	b
3	3	c
4	4	d
5	5	e
6	6	f

Query 2

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

Query 3

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



n-gram model

Tokenization of abcdef

Bi-gram

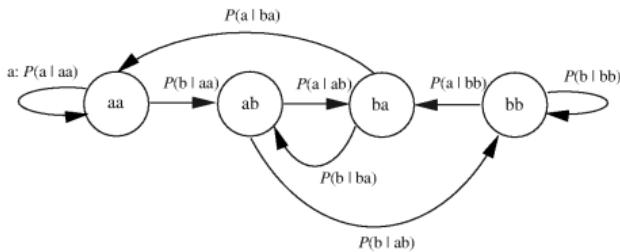
Query 1	
position	token
1	a
2	b
3	c
4	d
5	e
6	f

Tri-gram

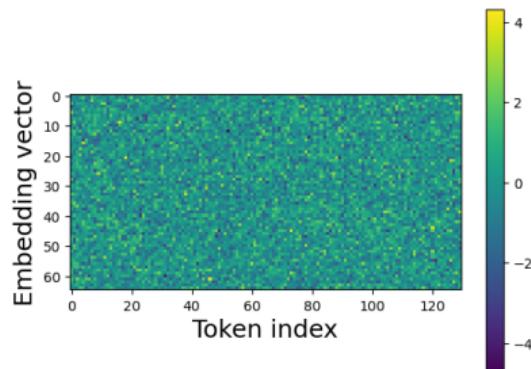
Query 2	
position	token
1	ab
2	bc
3	cd
4	de
5	ef

Four-gram

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



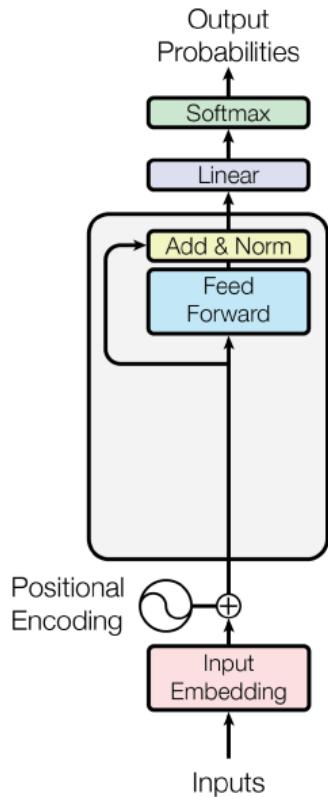
- Predictions based on the last n characters.
- No. tokens = (No. of characters) n
- Sufficient to learn embedding vectors.
For the tri-gram model and $n = 2$



Exercise 3: bigram and n-gram model

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

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.

Outline

Introduction to machine learning

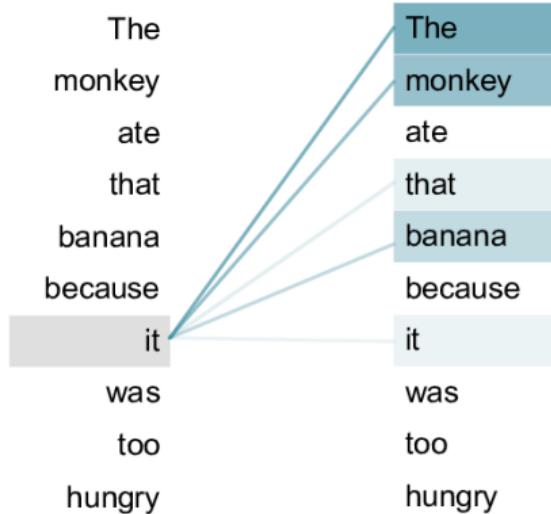
Neural networks

— lunch break —

Introduction to language models

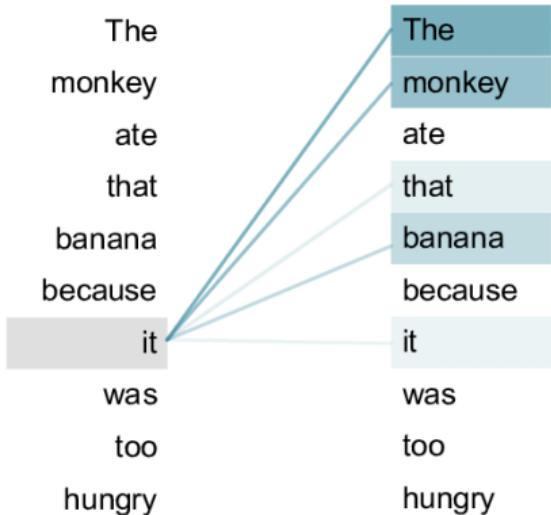
Generative pre-trained transformer

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.

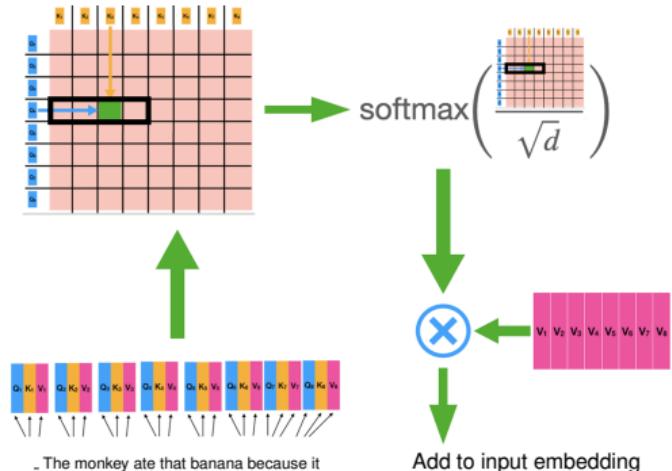
Generative Pre-trained Transformer



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.

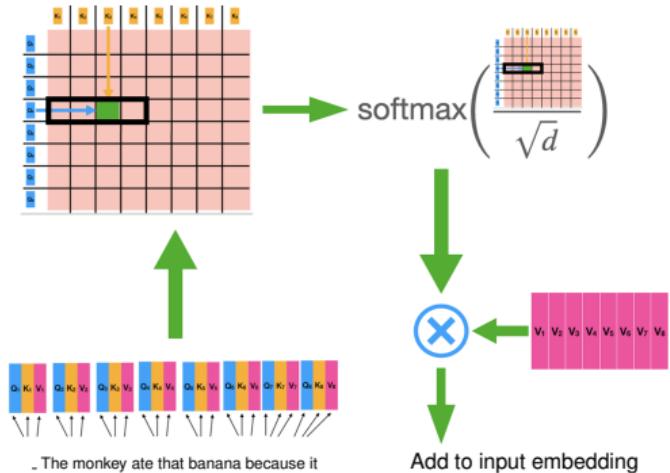
Generative Pre-trained Transformer



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.

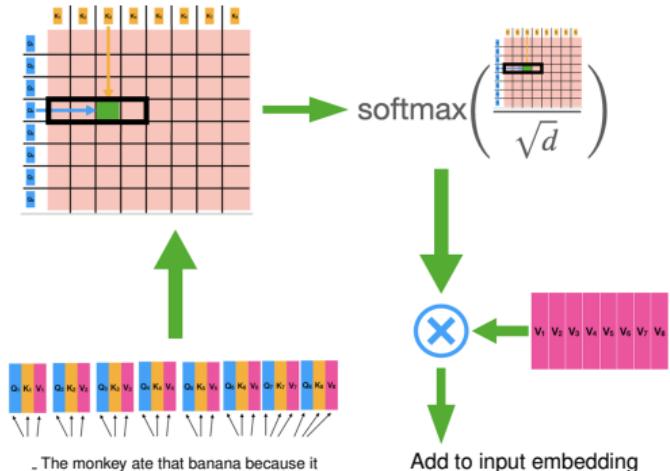
Generative Pre-trained Transformer



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 $j \leq i$

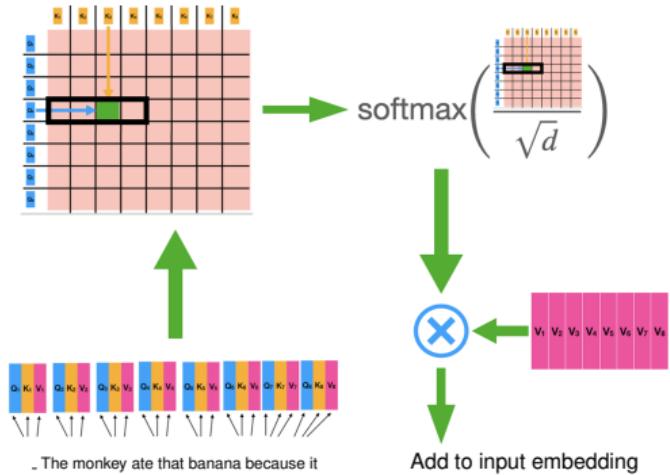
Generative Pre-trained Transformer



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 $j \leq i$.

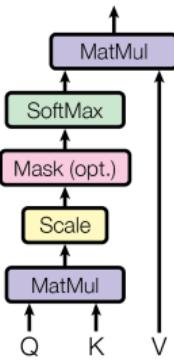
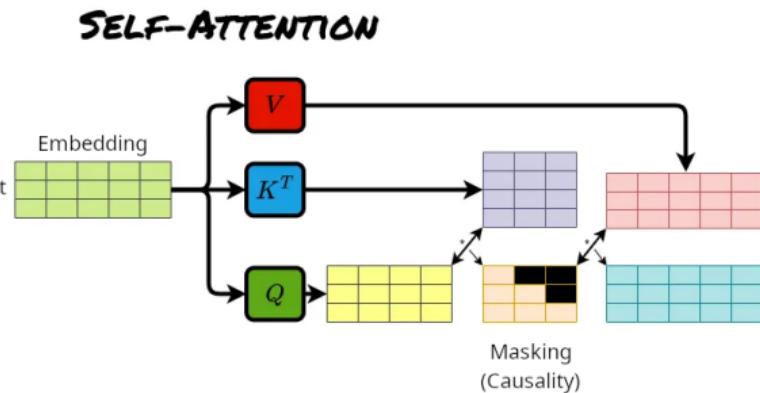
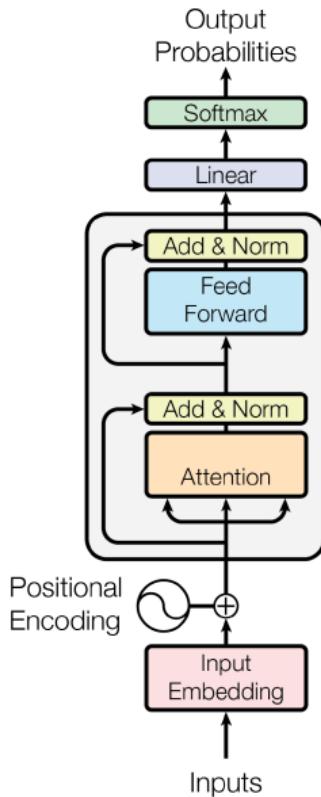
Generative Pre-trained Transformer



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 $j \leq i$.
- $\text{Attention}(Q_i, K_j, V_j) = \text{softmax}\left(\frac{Q_i^T K_j}{\sqrt{d}}\right) V_j$

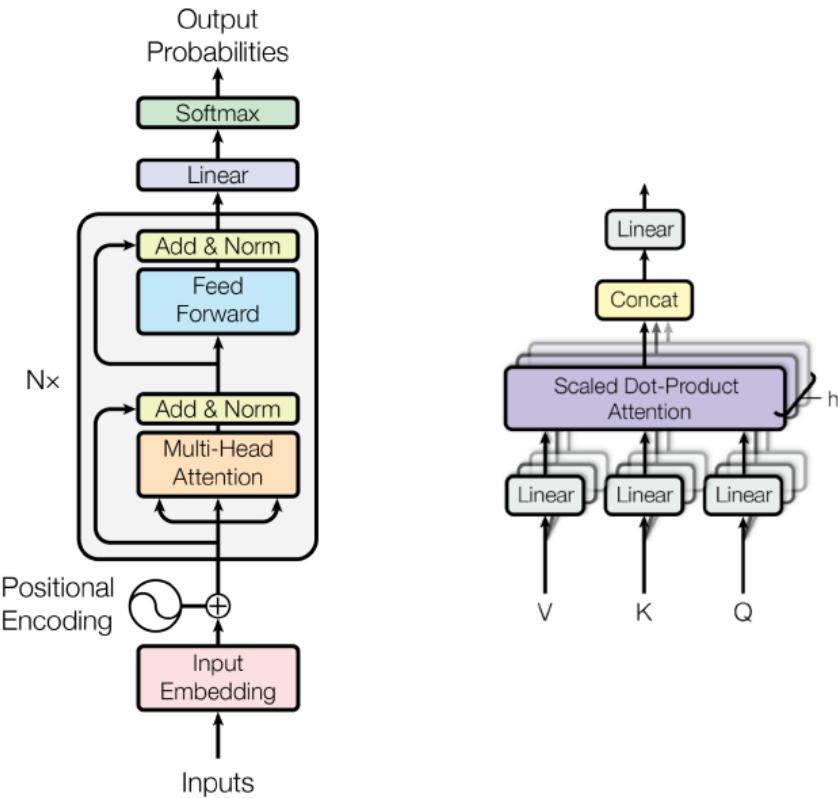
Self-attention mechanism



Exercise 4: 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 5: GPT2

Optimize the full GPT2 model.
(investigate the performance without dropout)

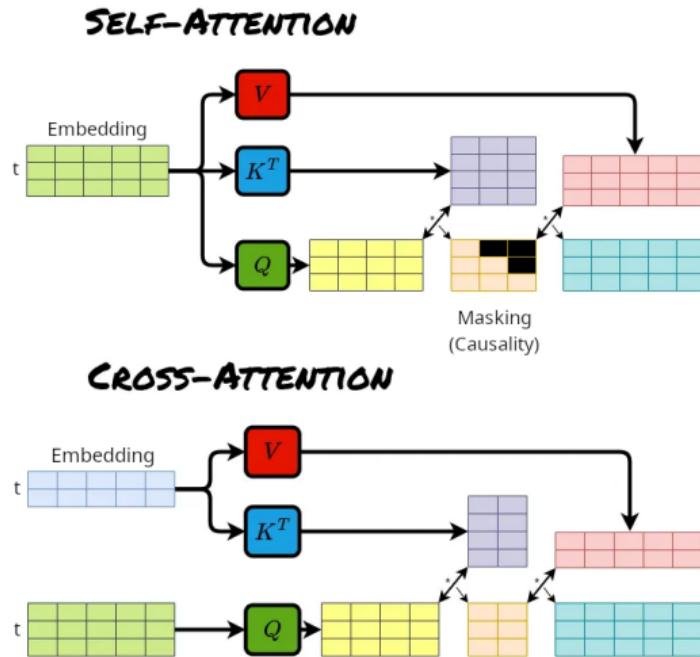
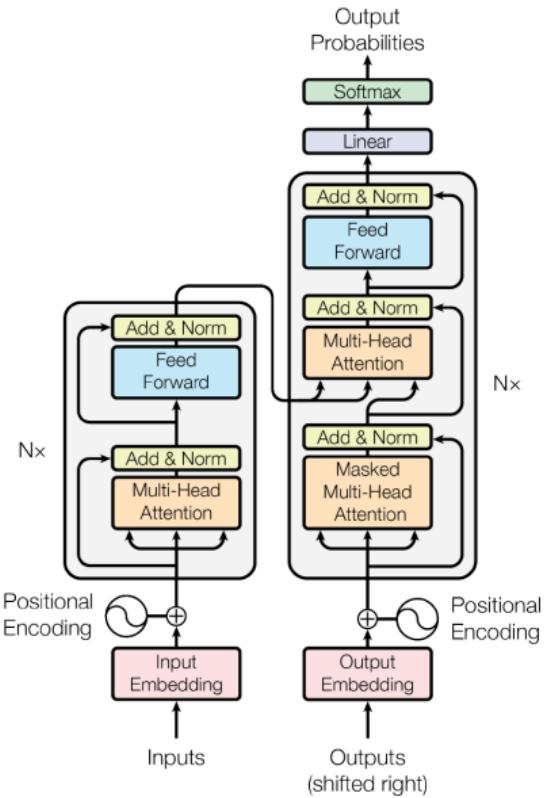
GPT3 model

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}

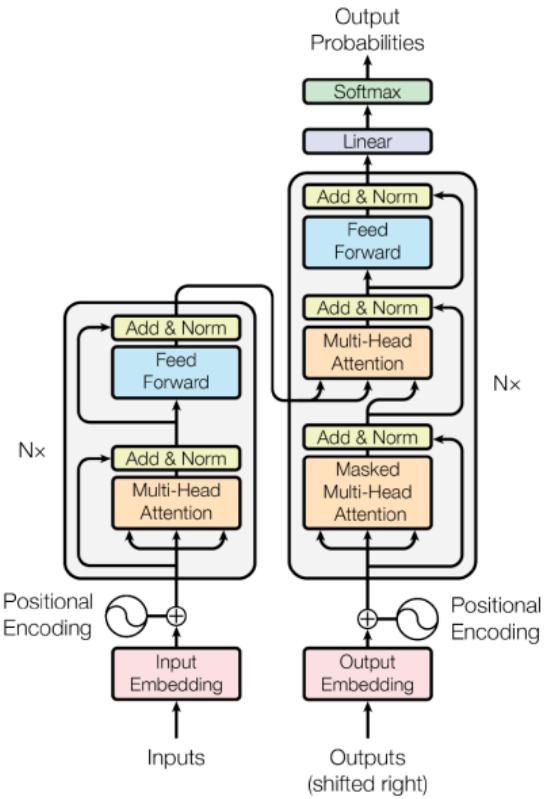
Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

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.

Explain reinforcement learning to a 6 year old.

A labeler demonstrates the desired output behavior.

We give treats and punishments to teach...

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

SFT
↙
DOC

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

Explain reinforcement learning to a 6 year old.

A labeler ranks the outputs from best to worst.

A
B
C
D
In reinforcement learning, the agent is...
Explains rewards...
In machine learning...
We give treats and punishments to teach...

This data is used to train our reward model.

D > C > A > B
RM
DOC

Reinforcement learning

Step 3

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

A new prompt is sampled from the dataset.

Write a story about otters.

The PPO model is initialized from the supervised policy.

PPO

The policy generates an output.

Once upon a time...

The reward model calculates a reward for the output.

RM

The reward is used to update the policy using PPO.

r_k

How to get started?

The screenshot shows the Hugging Face website. At the top, there's a navigation bar with links for Models, Datasets, Spaces, Posts, Docs, Solutions, Pricing, and a sign-in option. Below the navigation is a search bar with the placeholder "Search models, datasets, users...". A large central banner features a yellow emoji of a smiling face with hands clasped, followed by the text "The AI community building the future." and a subtitle about the platform being the place where the machine learning community collaborates on models, datasets, and applications.

The main content area displays a list of AI models. The first few models listed are:

- meta-llama/Llama-2-70B
- stabilityai/stable-diffusion-xl-base-0.9
- openchat/openchat
- ilyasviel/ControlNet-v1.1
- corspense/zeroscope_v2_XL
- meta-llama/Llama-2-13b
- tiiuae/falcon-40b-instruct
- WizardLM/WizardCoder-15B-V1.0
- CompVis/stable-diffusion-v1-4
- stabilityai/stable-diffusion-2-1
- Salesforce/CodeGen-2B-8k-int

At the bottom right of the page, the URL <https://huggingface.co/> is displayed.

Thank you for your attention!