ML Tools for Precision Physics in HEP

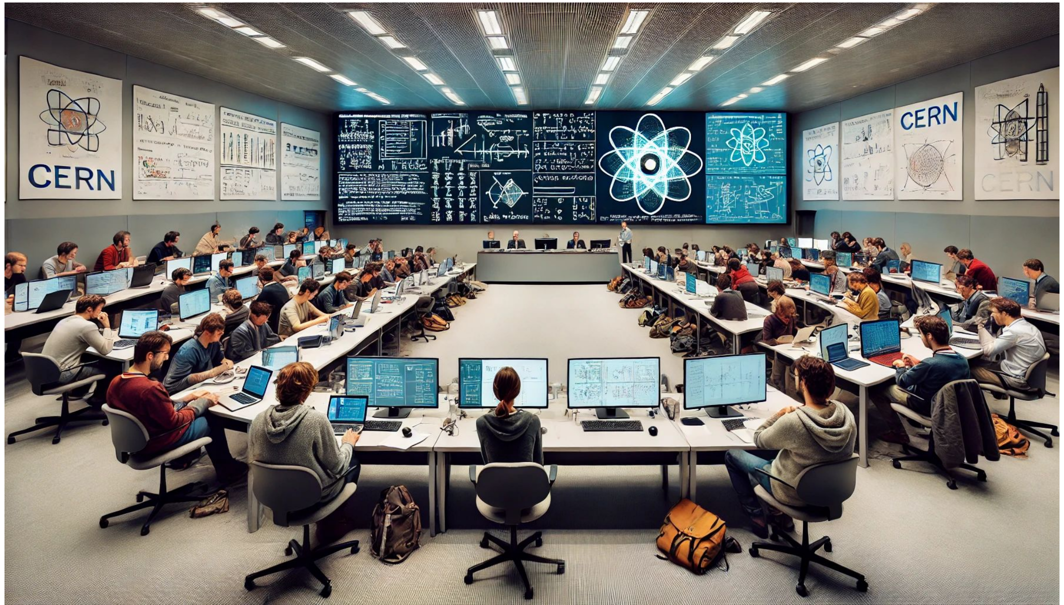
Dr. Davide Valsecchi (ETH Zurich)

10/10/2024

Intro

Machine Learning can enhance the performance of common tasks in HEP as reconstruction and analysis.

- we must always combine our Physics knowledge with ML techniques → better physics insight
- if we integrate our Physics bias in the structure of ML models → better performance
- it is possible to constraint model optimization using Physics biases → more control on the results
- not only regression and classification: probabilistic ML can be *very useful* for HEP analysis



ChatGPT prompt: generate an image of a room with PhD Students following a course about Machine Learning at CERN

PhD Course

What you will **NOT** find in this course:

- theory of ML architectures on the math side
- optimization recipes or common tricks recommendations (no *best* learning rate scheduler, no *best hyperparameters)*
- Complete physically meaningful examples
- A complete description of supervised ML theory from scratch

What you **will** find in this course:

- Examples of realistic applications of complex models to HEP problems
- Practical description of common ML architectures (Transformers, Normalizing Flows)
    - why they are useful for HEP
- Discussion of techniques to better control complex regressions
- A meaningful training dataset from WW Vector Boson Scattering (although simplified)

PhD Course - practicalities

GitHub repo for the code: https://github.com/valsdav/PhDCourse_MLForPrecisionPhysics_2024

- Setup instructions for CERN and outside.
- Dataset available on EOS or download here

Most of the course will be **hands-on**. We will explore ML architecture from notebooks, in PyTorch. We will use this slides for some conceptual introduction of different models.

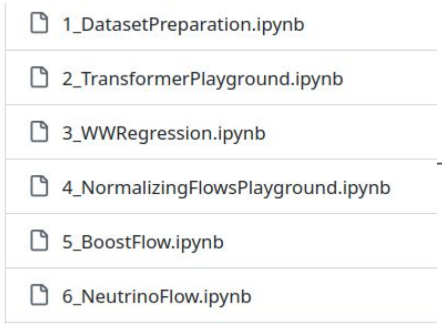
PhD Course “exam”: at the end of the course I will give you an exercise, on the line of the discussed models. You will have to expand on that and prepare a little report with your findings (~1month deadline)

Program

- **Dataset preparation (today)**
  - features scaling and normalization
  - data manipulation and formatting

- **Transformers (today)**
  - Intro and architecture
  - Full particles regression with transformers
    - Best losses for full particle regression
    - Constrained optimization with MDMM

- **Normalizing Flows (tomorrow)**
  - Intro and architecture
  - Example: Conditional probability for event boost
  - Application: Generative Transformers for neutrinos generation



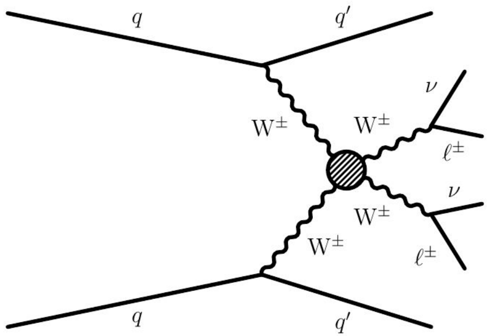
1_DatasetPreparation.ipynb
2_TransformerPlayground.ipynb
3_WWRegression.ipynb
4_NormalizingFlowsPlayground.ipynb
5_BoostFlow.ipynb
6_NeutrinoFlow.ipynb

Today

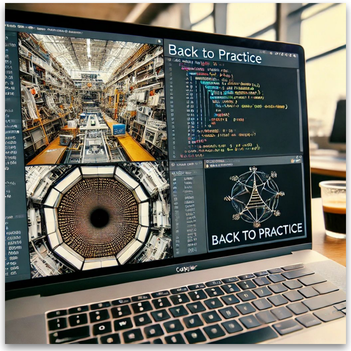
Tomorrow

Our dataset

- WW vector boson scatterting, fully leptonic events.
  - 2 leptons, 2 neutrino, MET, jets
- Augmented dataset with 10 random phi rotations
- Some plots: link



- DatasetPreparation notebook
- **Topics**:
  - how to handle the particle quadrimomenta when preparing features for ML models inputs
  - preprocessing pt
  - preprocessing phi
  - How to save the input datasets
-



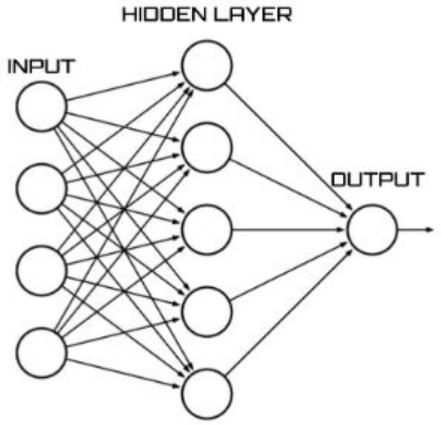
Neural networks

Feedforward networks are universal approximators of multivariate functions:

- theorem: 1 layer of hidden nodes (weights) is enough → but how many?
  - the quality of the approximation depends on the number of parameters (can be huge)
- from 1 layer to stack of multiple layer → **Deep Neural Network (DNN)**

Characteristics:

- input features order is **fixed** (no invariance)
- computational very fast
- Can have 1 or multiple outputs
- activation layer on the output is very important:
  - sigmoid R→[0,1] for classification



(I highly recommend 3Blue1Brown serie on Machine Learning and NN youtube)

Embedding

We often talk about “embedding”.

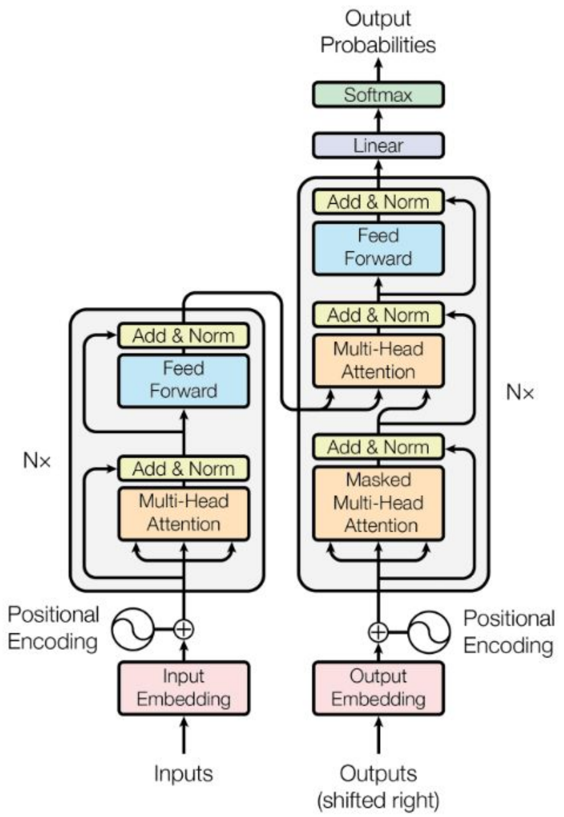
An embedding if a function f(x): $R^N \rightarrow R^M$ that we use to bring our **input** into a (usually) **larger dimensional** space.

The embedding is usually expressed with a DNN network:

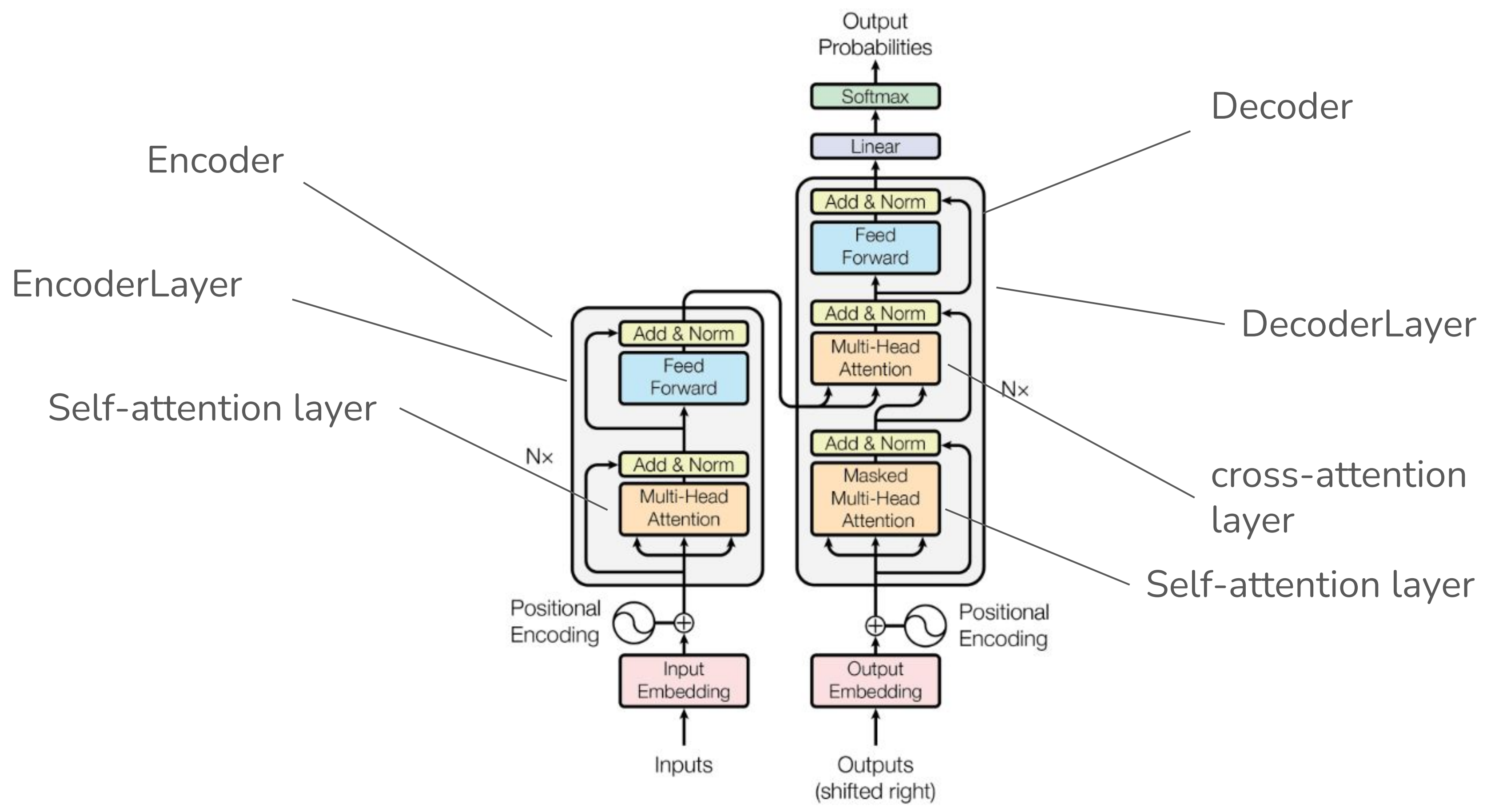
- optimized along size the ML model
- “extract” information from the input
- the higher dimensional space allows the ML model to have “more capacity” → find higher dimensional correlations and relations between the input features

Transformers

- **Fast forward**: our dataset is made of several *objects* and we want a model able to reason on the relation between the objects
- Transformer: special architecture able to elaborate on set of objects and their relations
  - input order invariance
  - based on the “attention mechanism” Attention is All you need https://arxiv.org/abs/1706.03762 paper
  - it seems very complex but it is made of simple building blocks and a math trick to analyse the “relation” between inputs



Structure


Output Probabilities
Softmax
Linear
Encoder
Decoder
EncoderLayer
DecoderLayer
Self-attention layer
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Add & Norm
Masked Multi-Head Attention
cross-attention layer
Self-attention layer
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

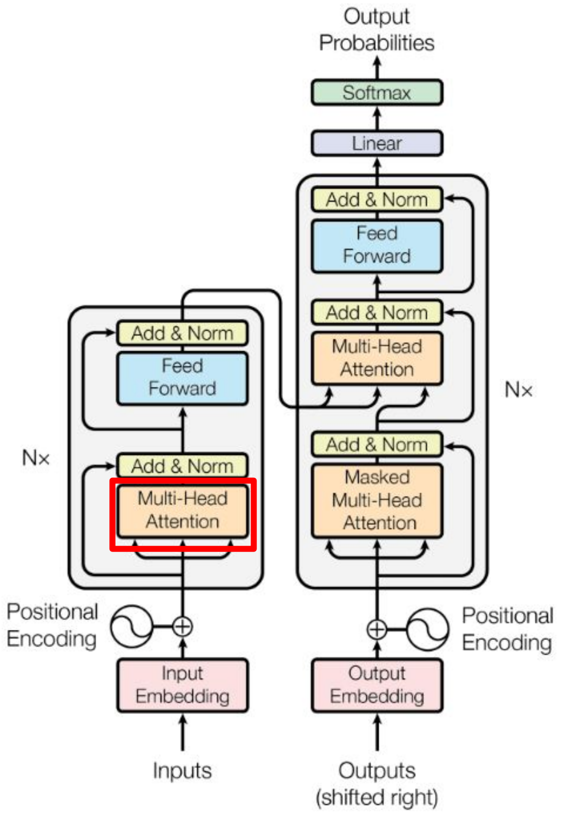
Attention computation

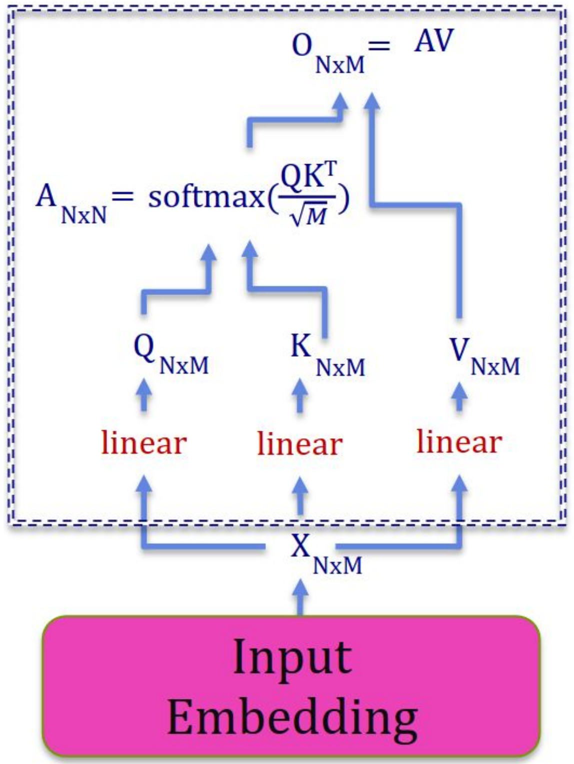
The attention mechanism is what makes the transformer so special.

It computes the relative importance between the input "tokens" (columns) using 3 matrices: **query, key and value.**

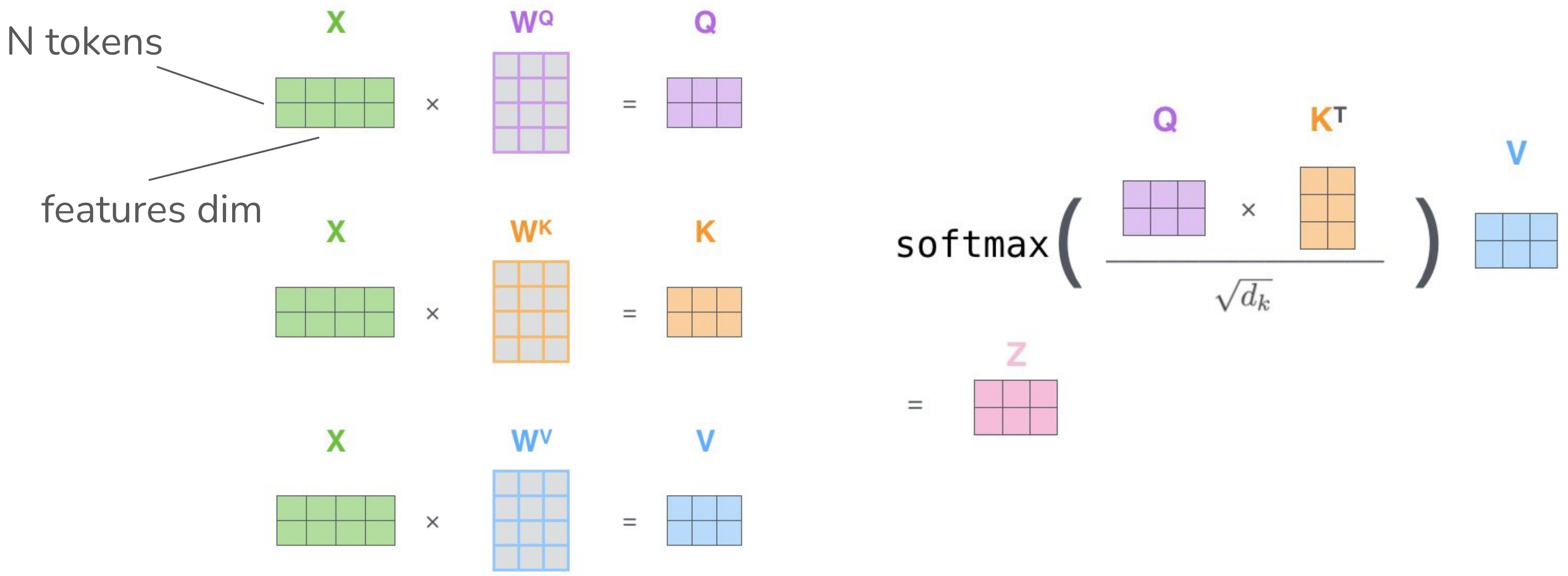
Two matrices are multiplied together to compute an "adjacency matrix", the third one is used to extract a vector for each input using this adjacency matrix.

Full visual explanation of transformers from 3Blue1brown youtube





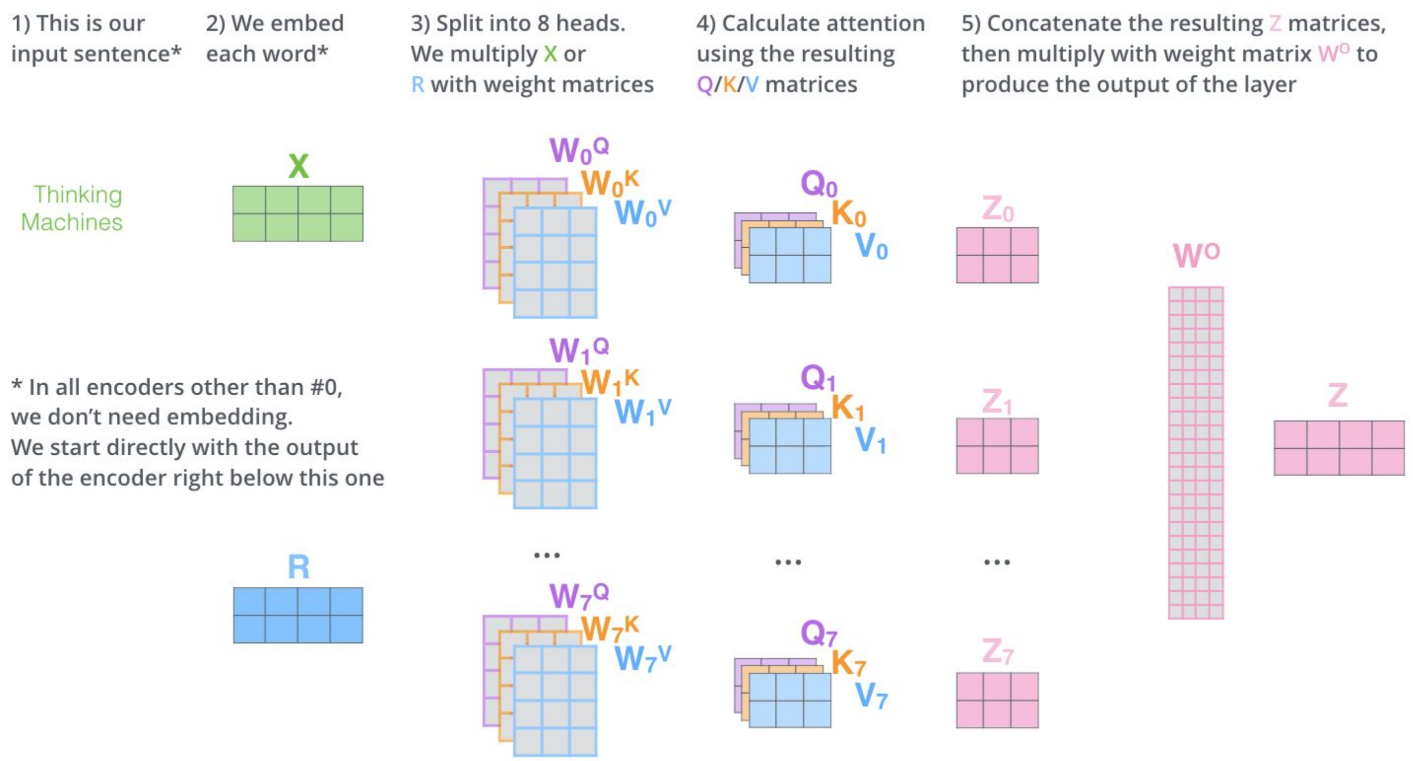
Attention: zoom in



The matrices Q, K, V are fixed dimensions, even if X can have different lengths
Q,K,V learns in a compact way how to extract information to mix and match the input tokens and generate an output that is effectively “mixing up” information from different tokens of the input.

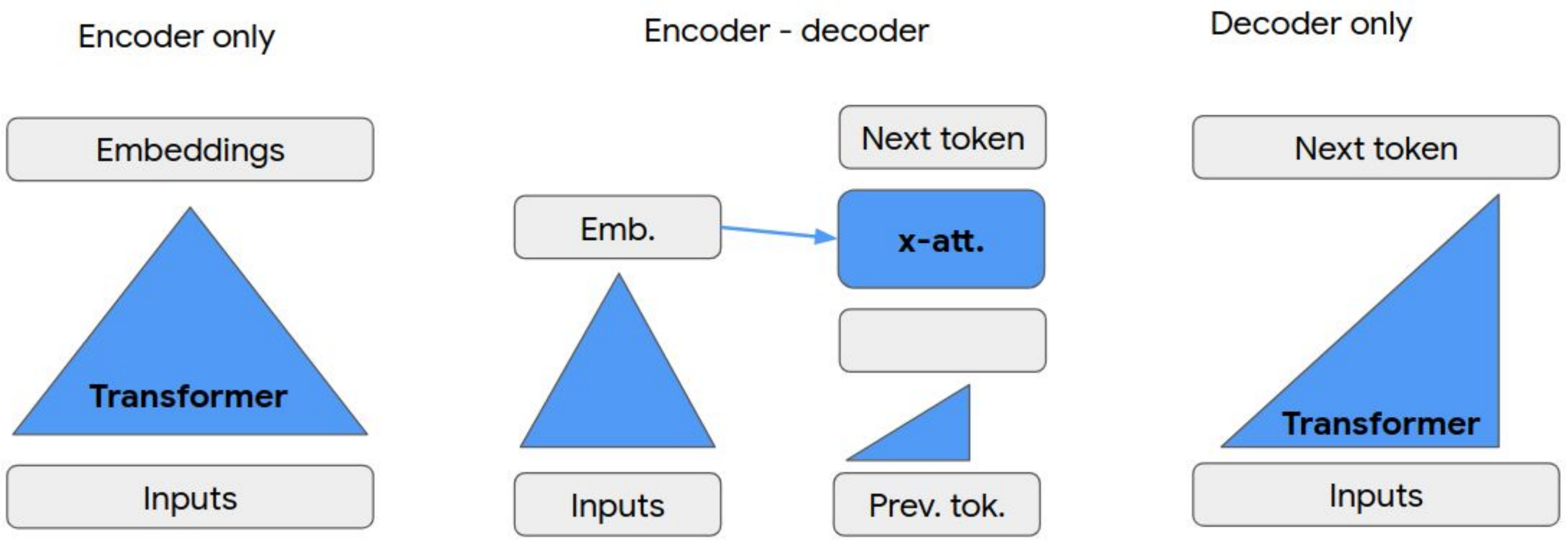
Multi-head attention

Now, just take this structure and let's scale it up → Multihead attention



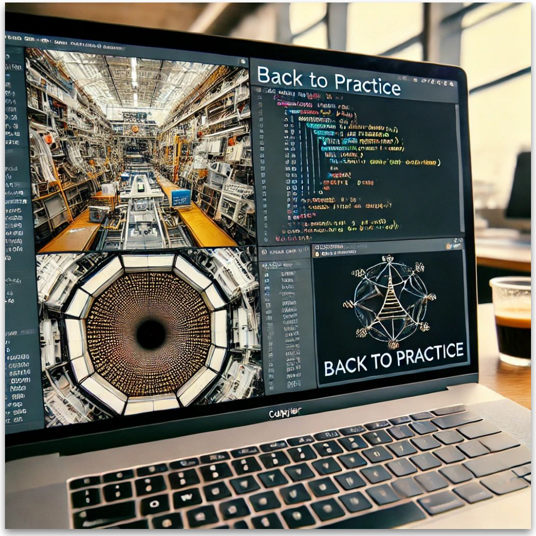
1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
W_0^Q W_0^K W_0^V
W_1^Q W_1^K W_1^V
W_7^Q W_7^K W_7^V
Q_0 K_0 V_0
Q_1 K_1 V_1
Q_7 K_7 V_7
Z_0 Z_1 Z_7
W^O
Z
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Transformers flavors



Practice - Transformers intro

1. TransformerPlayground notebook
   a. let’s pratice with pytorch transformers
   b. build encoder, decoders, full layers
   c. explore the input shapes, try to break it
   d. Check how the number of parameters scaled with the transformer dimensions (nhead, nlayers, d_model)



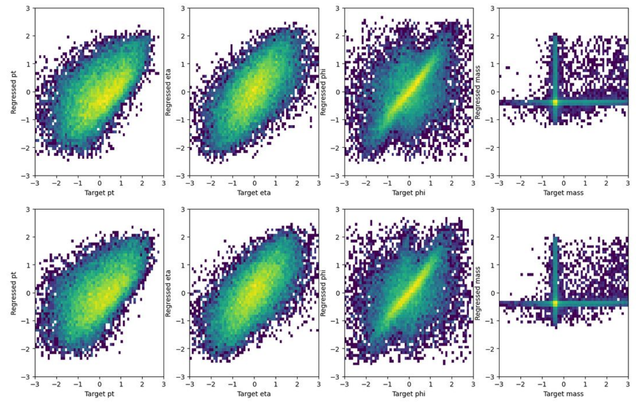
Practice - WW regression

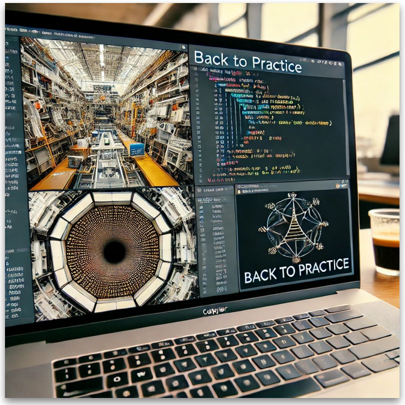
1. WWRegression notebook
   a. let’s pratice with a physics application

We want to regress the 4-momenta of the 2 W boson in WW VBS events.

Inputs: jets, leptons, MET in the event

Let’s use a transformer to decode the 2 W complete four momenta
   i. we discussed different losses
   ii. How to get the correct regressed distributions (MMD loss)
   iii. How to write the loss for stronger constraints in the optimization (mdmm method)

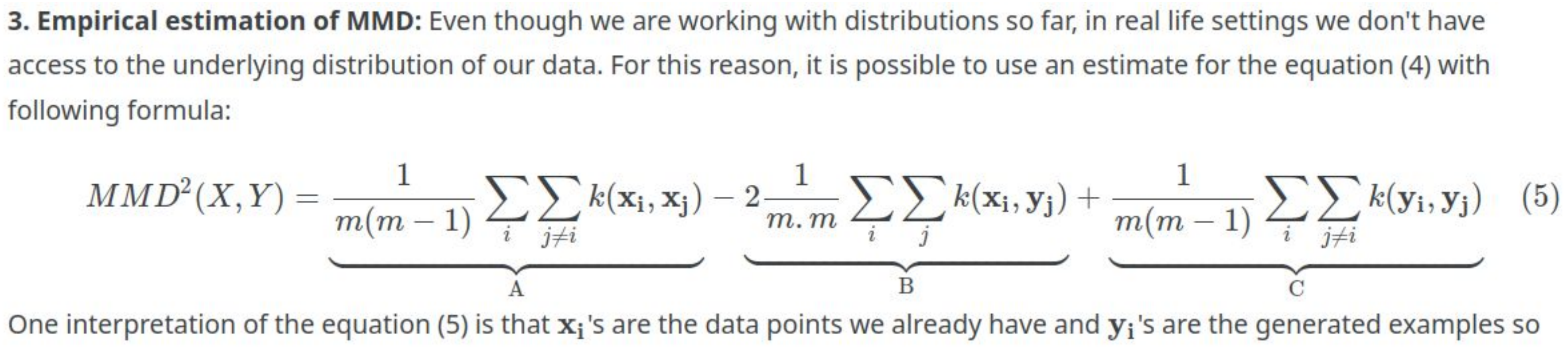




MMD loss

We need a loss measuring how different are two distributions: we just have samples from the distributions, so it will be an approximate computation.

Maximum Mean Discrepancy: a kernel two-sample test. Paper



3. Empirical estimation of MMD: Even though we are working with distributions so far, in real life settings we don't have access to the underlying distribution of our data. For this reason, it is possible to use an estimate for the equation (4) with following formula:

$$MMD^2(X,Y) = \underbrace{\frac{1}{m(m-1)}\sum_i\sum_{j\neq i} k(\mathbf{x_i},\mathbf{x_j})}_{\text{A}} - \underbrace{2\frac{1}{m.m}\sum_i\sum_j k(\mathbf{x_i},\mathbf{y_j})}_{\text{B}} + \underbrace{\frac{1}{m(m-1)}\sum_i\sum_{j\neq i} k(\mathbf{y_i},\mathbf{y_j})}_{\text{C}} \quad (5)$$

One interpretation of the equation (5) is that $\mathbf{x_i}$'s are the data points we already have and $\mathbf{y_i}$'s are the generated examples so

Different kernels can be used → practical implementation link

Constrained loss minimization

- Loss functions for ML models are typically theoretically motivated e.g., binary cross-entropy comes from the binomial likelihood
- Good guarantee that it will be well behaved
- Often though additional terms are added to the loss e.g., weight decay, latent norm, DISCO, …

$$\mathcal{L} \to \mathcal{L}_1 + \alpha \mathcal{L}_2$$

- How do we choose α? We make it trainable! Well…
- **There is no guarantee that the solution manifold of this new multi-objective problem is well behaved when we apply "vanilla" gradient descent**

$$\mathcal{L}(\theta|n,k) = \binom{n}{k} p(\theta)^k \left(1 - p(\theta)\right)^{n-k}$$

$$\Big\downarrow$$

$$\begin{aligned} &-\log \mathcal{L} \\ &= -k \log p(\theta) \\ &- (n-k) \log\big(1 - p(\theta)\big) \\ &+ C \end{aligned}$$

Material from Roberto Seidita ETH ML journal club

Back to analysis 2: the lagrangian multipliers

- The equation $\mathcal{L}(x,\alpha) = f(x) + \alpha g(x)$ is a Lagrange multiplier problem, minimizing f under the condition that $g = 0$
- This equation is minimized by solving

$$\begin{cases} \nabla_x \mathcal{L}(x,\alpha) = 0 \\ \partial_\alpha \mathcal{L}(x,\alpha) = g(x) = 0 \end{cases}$$

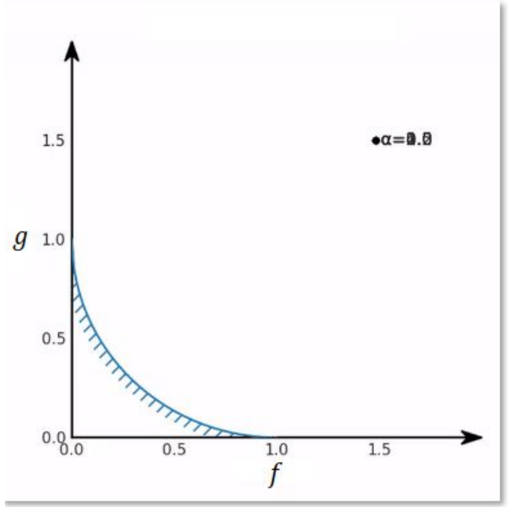
- The issue is that for fixed x, $\mathcal{L}$ can be decreased by sending α to $\pm\infty$
- Gradient descent will not be able to systematically find the optimum
- More formally: solutions are typically **saddle points** in (x,α) space or, equivalently, **minima of the constrained loss are not necessarily attractors for the set of differential equations**

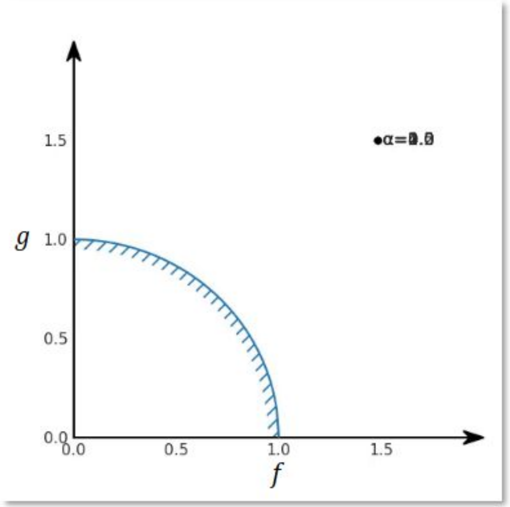
Material from Roberto Seidita ETH ML journal club

The Pareto front problem

Let's look at some animations here

What you want → What you (often) get





In many cases, regardless of the choice of α we cannot access the full set of equilibrium points

Material from Roberto Seidita ETH ML journal club