Algorithms for protein structure

Oct 3, 2024

Ab initio prediction using energy functions

Physics based methods make use of energy functions

- Structure prediction
- Docking of proteins and small molecules
- Molecular dynamics
- Mutation effect prediction
- Protein design

Electrostatics

Van der waals

Hydrogen bonds

Solvent interactions and the hydrophobic effect

Electrostatics

$E_{elec} = C * q_1 q_2 / Dr$, where:

q1 and q2 are the charges
r is the distance
D is the dielectric constant (~80 for water, ~2-10 for protein)

For simple calculations, compare with the Bjerrum length l_B:

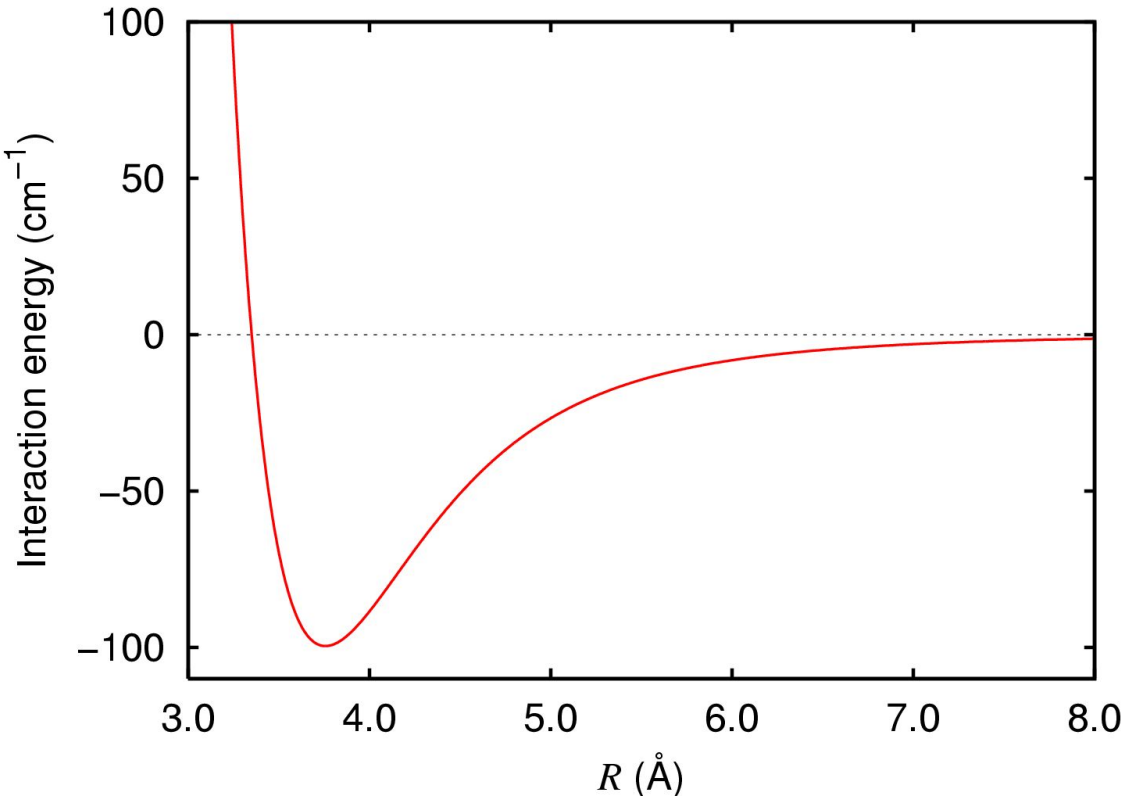
$E_{elec} = kT = 2.5$ kJ/mol at r=7A at 298K in water
$E_{elec} = 7/r * 2.5$ kJ/mol

Van der Waals / Lennard-Jones

Very short range (contact)

Interactions between induced dipoles in electron cloud

Seen for any atomic “surfaces” near each other



Hydrogen bonds

Quantum mechanical effect

Similar to, but weaker than, covalent bonds

5-10 kJ/mol

Bonded atom potentials

Explicit energies for:

- bond lengths
- bond angles
- dihedral angles

are used to enforce constraints between chemically bonded atoms

These are an approximation of the quantum mechanical energies that govern bond conformations

Some popular energy functions

AMBER

CHARMM

GROMOS

OPLS

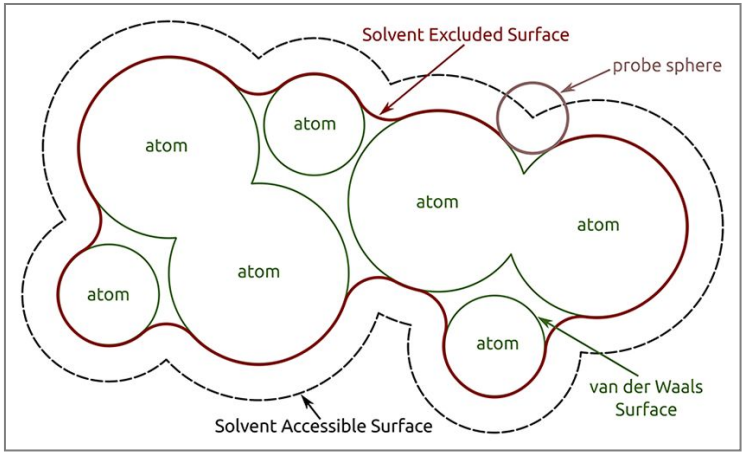
Can be used with explicit or implicit water models

Hydrophobic effect

Often approximated as proportional to solvent accessible surface area

Many other terms in energy function are approximately zero or positive because water makes good interactions with proteins (electrostatics, van der waals, H-bonds)

Mostly an entropic effect



How to interpret energies as probabilities

The Boltzmann distribution states that the energy of a state (eg, protein conformation) is related to the exponent of its energy

A more general form of the energy that has this property is referred to as the free energy

Boltzmann distribution

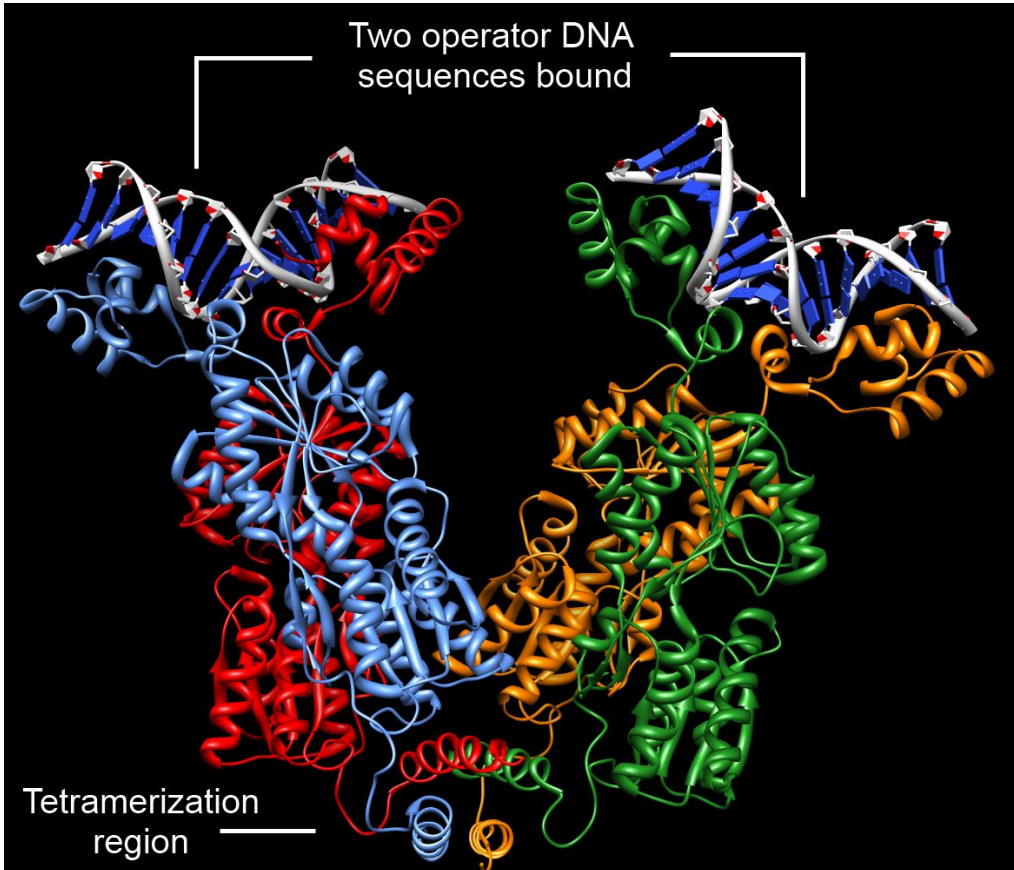
$$p_i = \frac{1}{Q} \exp\left(-\frac{\varepsilon_i}{kT}\right) = \frac{\exp\left(-\frac{\varepsilon_i}{kT}\right)}{\sum_{j=1}^{M} \exp\left(-\frac{\varepsilon_j}{kT}\right)}$$

where eps_i = energy of state i, kT = 2.5 kJ/mol at 298K

Thought problem

A protein has two states, open and closed. The open conformation of the protein has an energy 5 kJ/mol higher than the closed state. **What fraction of the protein molecules will be found in the open state?**

Higher order structure - DNA looping



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

Thought problem

Our LacI protein has two additional binding sites, O2 and O3. Imagine our protein binds O1 and one of either O2 or O3, and that it can bind each equally well. **If we mutate O3, so it no longer binds, how much stronger (lower in energy) would we need to make the interaction between LacI and O2 to compensate for the lost configuration?**

Bonus: How would you go about strengthening the LacI-O2 interaction?

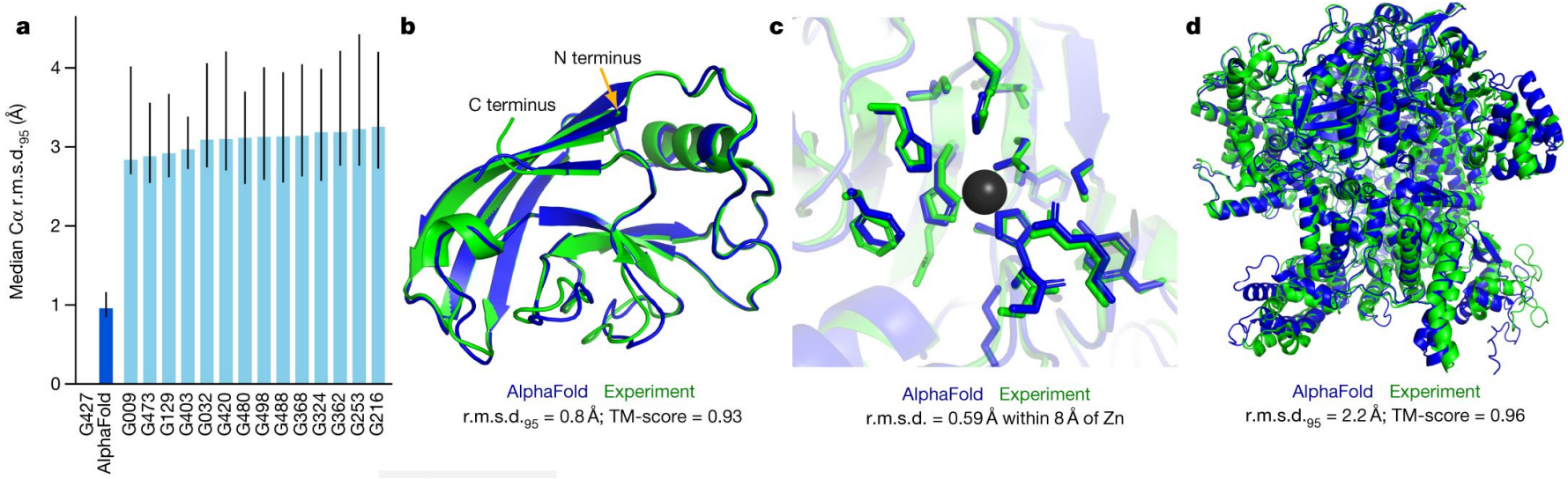
Deep learning and protein structure

Alphafold is a deep neural network approach to predict protein structure from sequence

Goals:

- Understand how “information” moves within a complex neural networks
- Appreciate how knowledge is captured in the network
- Understand the different types of data that inform prediction

AlphaFold 2 is considered a solution to the protein structure prediction problem



Jumper et al, Nature 2021 https://www.nature.com/articles/s41586-021-03819-2

Refresher: what is a neural net?

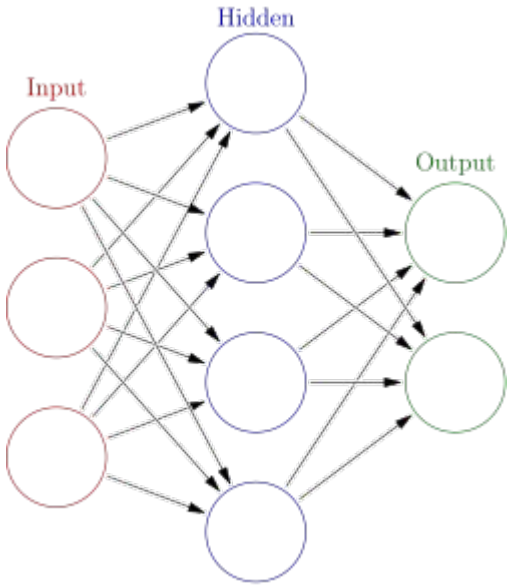
Neurons are nodes that take input together with **weights** to produce **activations**

Adjacent **layers** are connected by a matrix of **weights**

Each **layer** is a vector of numbers

Neurons emit activations by combining the input and **weights** with an optional vector of **biases**

Weights and **biases** are learned by training the network to produce a desired output



https://en.wikipedia.org/wiki/Neural_network_(machine_learning)

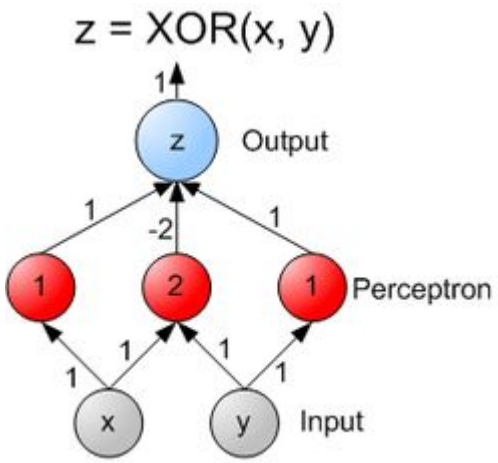
Non-linear activations allow complex reasoning

Each layer includes a linear (or affine) transformation

But this would be equivalent to a single linear layer if no non-linearity was applied between layers

Without non-linearities, non-linearly seperable problems like XOR can not be computed

ReLU, sigmoid, tanh are some common non-linear activation functions



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

Neural net output

Output from a neural net is a vector of logits

Logits can be thought of as log-likelihoods

The Softmax function converts logits to a distribution of probabilities

Softmax uses the Boltzmann distribution

Formally, the standard (unit) softmax function σ: $\mathbb{R}^K \to (0,1)^K$, where $K \geq 1$, takes a vecto
$\mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$ and computes each component of vector $\sigma(\mathbf{z}) \in (0,1)^K$ with

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} .$$

Softmax converts logits to probabilities

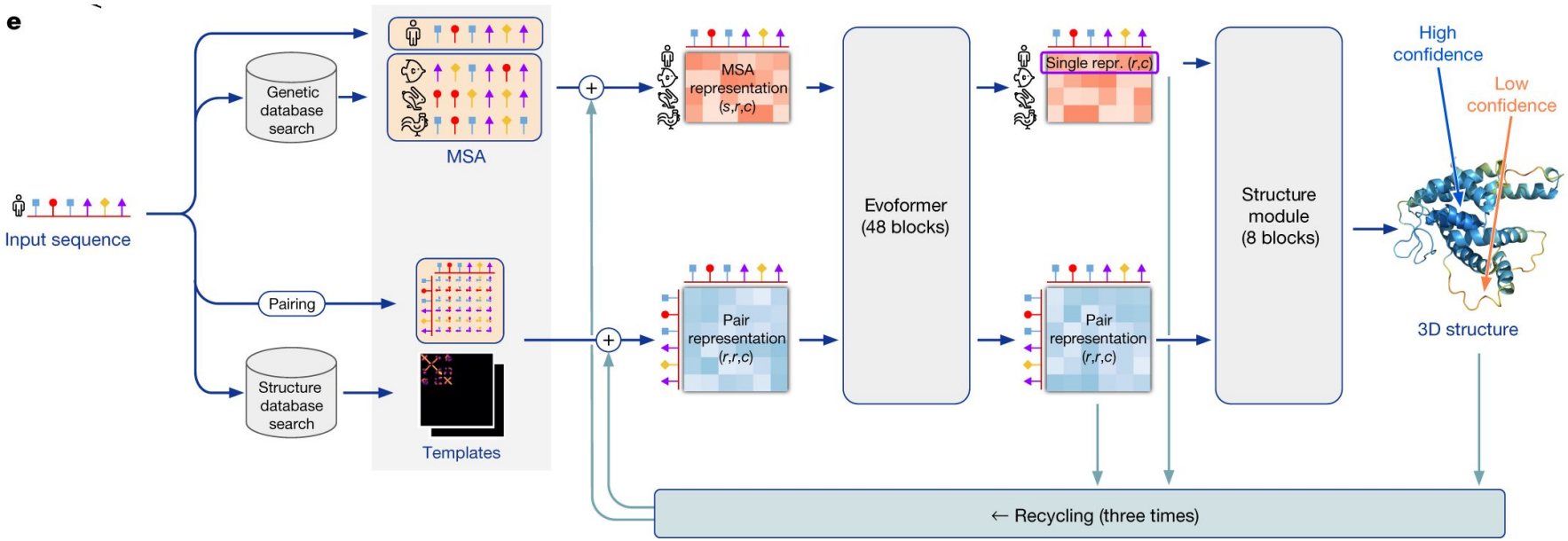
This is the same as the Boltzmann equation mathematically if we use E/kT as logits and take softmax

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

How AlphaFold 2 works

Input
MSA representation
Pair representation
Evoformer block
Structure module
Confidence measures

Overall workflow



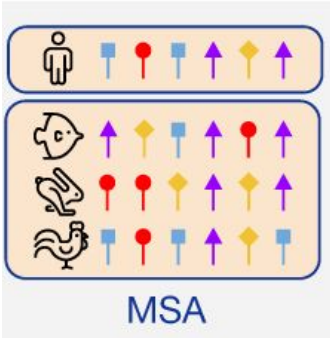
Jumper et al, Nature 2021 https://www.nature.com/articles/s41586-021-03819-2

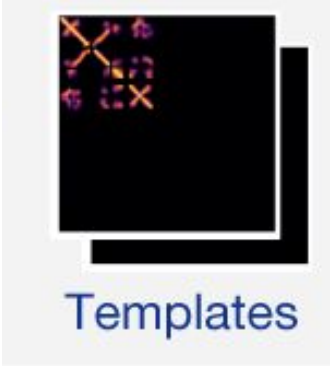
Input data

Primary sequence

MSA of homologs

Optional template



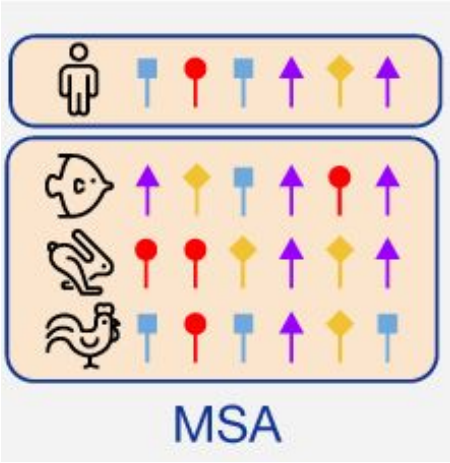


Evolutionary information

Hydrophobic core

Surface

interactions



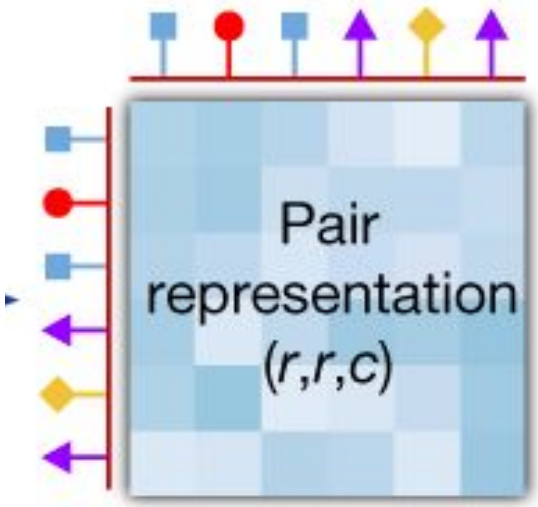
Position:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Seq1:	L	K	A	R	K	L	V	A	D	Q	W	G	D	R	D	Q
Seq2:	M	K	G	K	K	F	I	L	E	Q	W	D	N	K	D	E
Seq3:	R	E	P	Q	K	Y	L	A	E	Q	W	G	E	N	R	R
Seq4:	K	K	P	K	K	F	I	L	E	Q	W	D	N	K	D	E
Seq5:	H	E	T	R	K	W	M	A	E	Q	W	G	Q	K	R	D

Pair representation

Pair representation tensor is generated by combining features from each residue including their relative position and amino acid identity

Tensor has shape [length, length, 128]

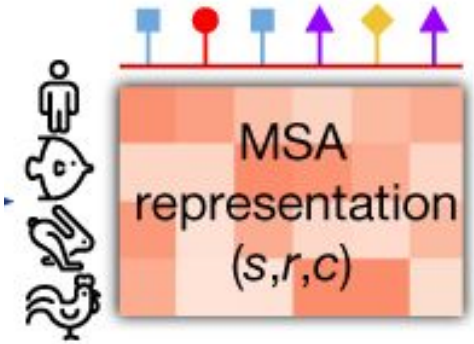
Interactions between the 2 residues are encoded as a 128 dimensional vector



MSA representation

MSA representation tensor is constructed by combining features derived from the MSA

The tensor has shape [species, length, 32]



Embedding discrete data as high dimensional continuous vectors

Both the MSA and pair representations embed amino acids (or pairs of AAs) as fixed dimensional vectors

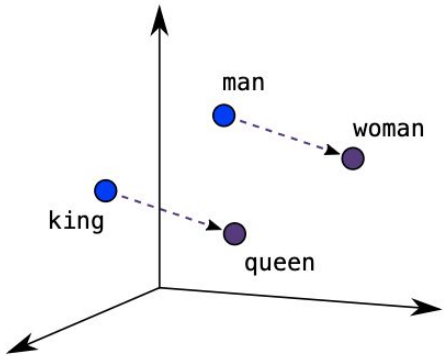
Embeddings are fixed dimensional continuous vectors that are learned for discrete data types

In NLP words are often converted to **embeddings** before being used in networks for other tasks **word2vec**

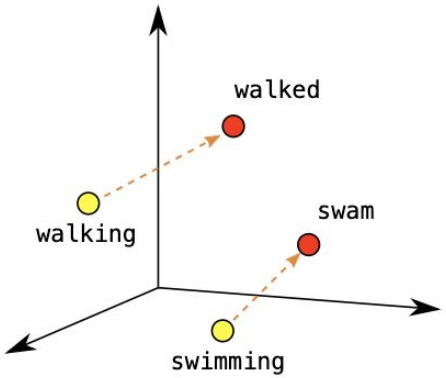
Embeddings are learned to optimize performance on a specified **supervised learning** task

Can be represented as a matrix mapping **one-hot encoding** to vectors

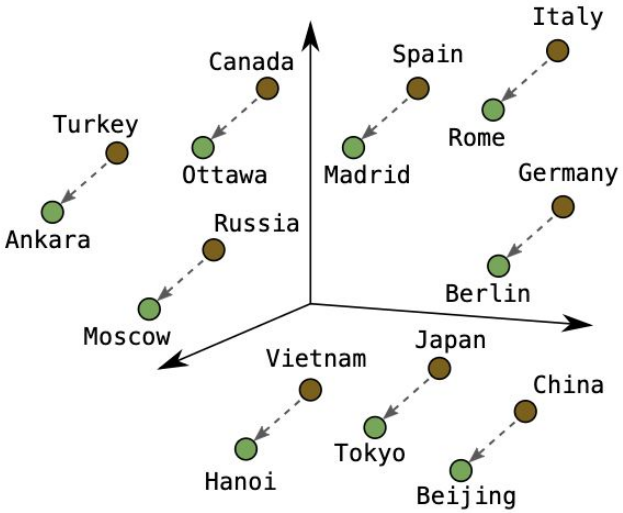
Embeddings often lead to surprising mathematical relationships between concepts



Male-Female



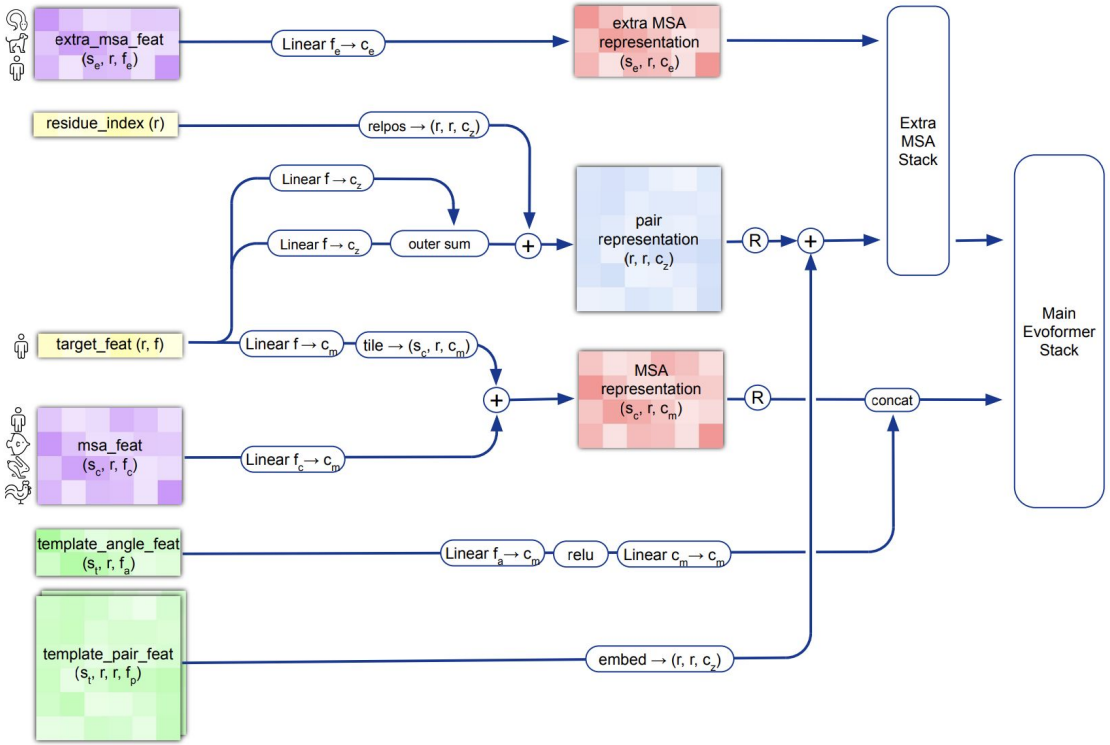
Verb Tense



Country-Capital

https://developers.google.com/machine-learning/crash-course/embeddings/embedding-space

Constructing the input to AlphaFold



Jumper et al, Nature 2021 https://www.nature.com/articles/s41586-021-03819-2

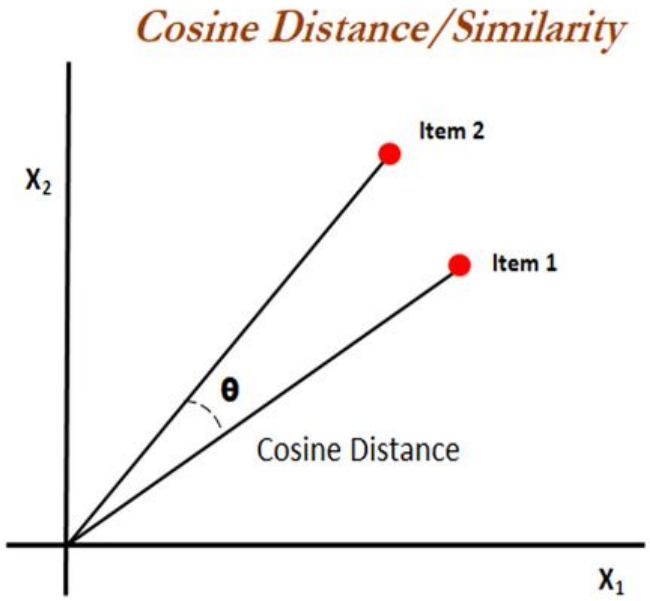
Intuition - adding vectors in a high-dimensional space

What is the prob of a randomly chosen vector having a modest similarity to any given vector?

1024 dimensional space

Pick arbitrary vector (1,0,0,0,0…0)

Take cos similarity of >0.1 as similar



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Intuition - adding vectors in a high-dimensional space

What is the prob of a randomly chosen vector having a modest similarity to any given vector?

1024 dimensional space

Pick arbitrary vector (1,0,0,0,0…0)

Take cos similarity of >0.1 as similar

Choose random vector y from 1024-dim normal distribution

norm(y) = 32

Scale y to unit norm (mult by 1/32)

y1 ~ Norm(0,0.03)

p(y1>0.1) = 0.0007

Intuition - adding vectors in a high-dimensional space

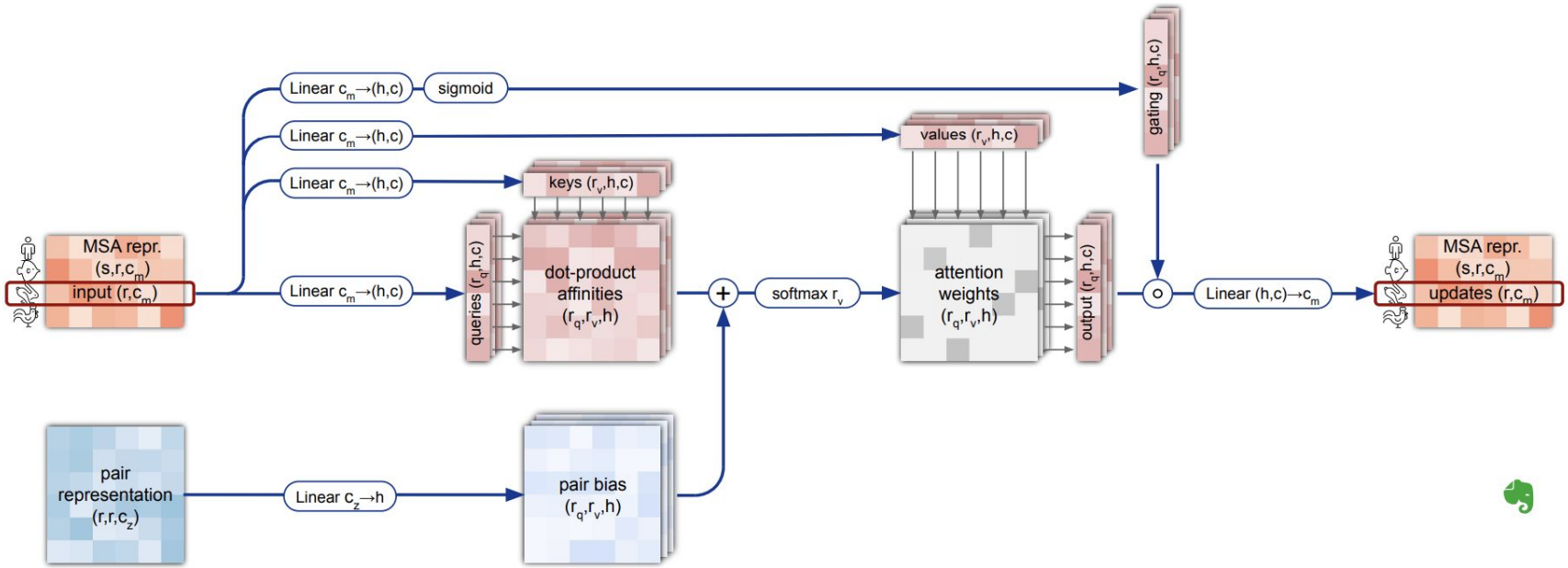
So we can encode a lot of information in high-dim vectors, and simply add them together without worrying about “forgetting” information

We can use a distance/similarity metric like cos similarity to extract information

For example, we could add a vector with the semantic meaning of a word to its position in a sequence without having to worry about

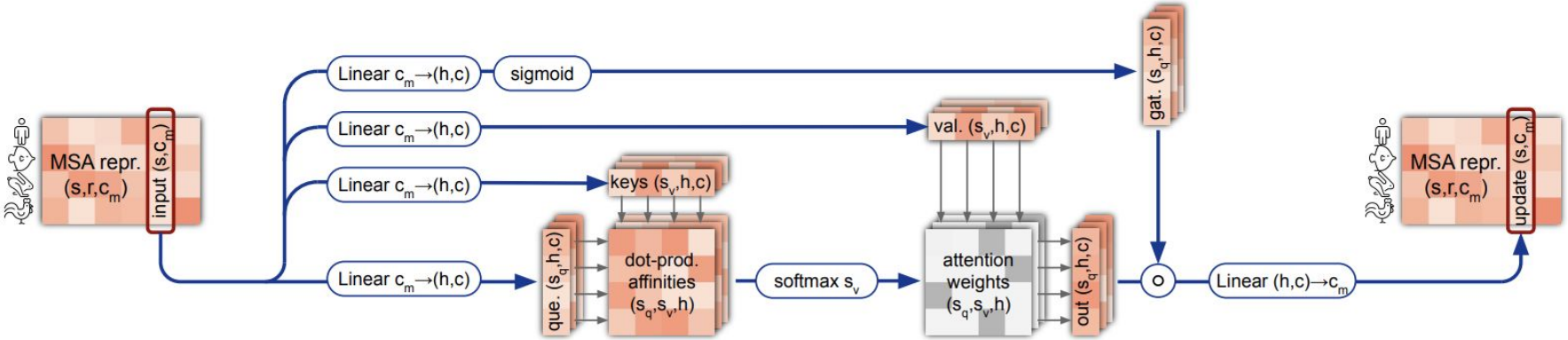
Or we could add multiple words together into a single vector, and still extract them later

Attention mechanism in AlphaFold - MSA rows



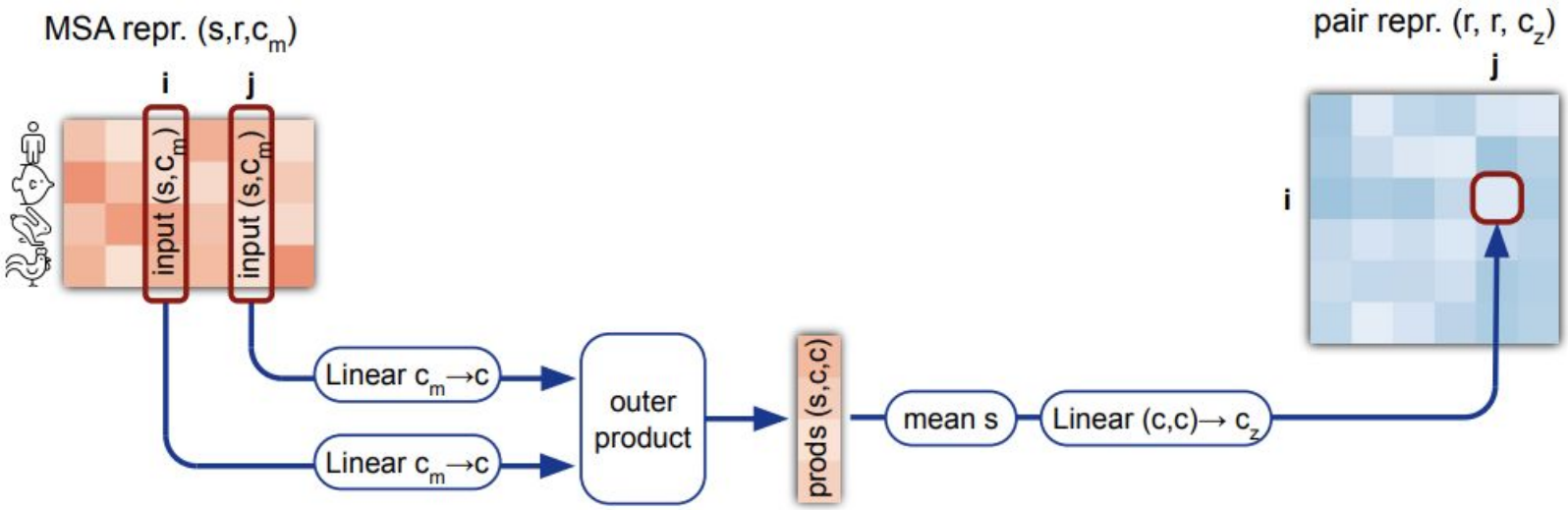
Jumper et al, Nature 2021 https://www.nature.com/articles/s41586-021-03819-2

Attention mechanism in AlphaFold - MSA columns



Jumper et al, Nature 2021 https://www.nature.com/articles/s41586-021-03819-2

Updating pair representation with MSA information



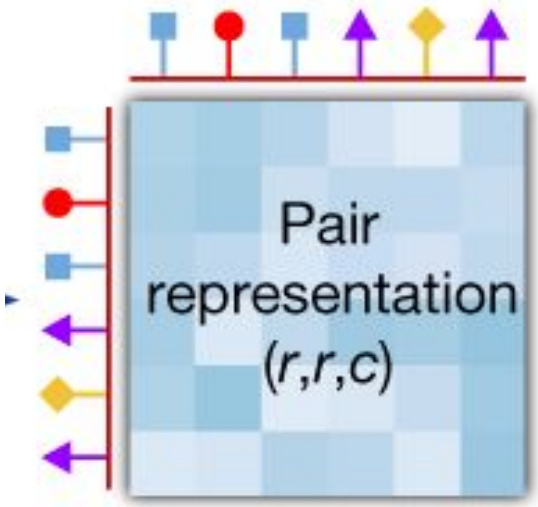
Jumper et al, Nature 2021 https://www.nature.com/articles/s41586-021-03819-2

How is 3D structure stored in the pair representation?

Each row,col represents an interaction bw 2 residues as a 128-vector

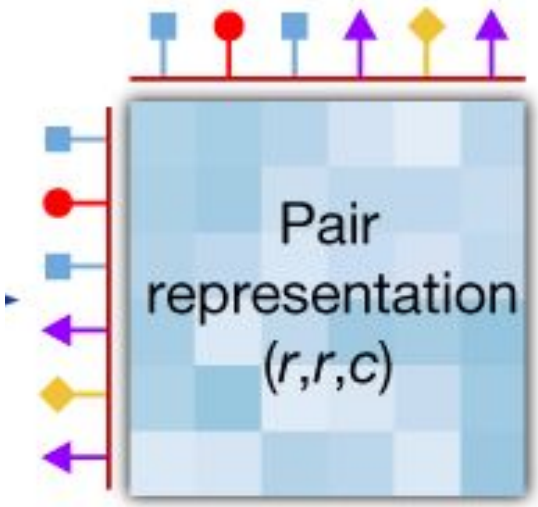
Triangle update module considers triplets of residues

Uses specialized attention mechanism and triangle multiplication for pairs that helps to enforce geometric constraints (eg, triangle inequality $d_{ij} <= d_{ik} + d_{kj}$)

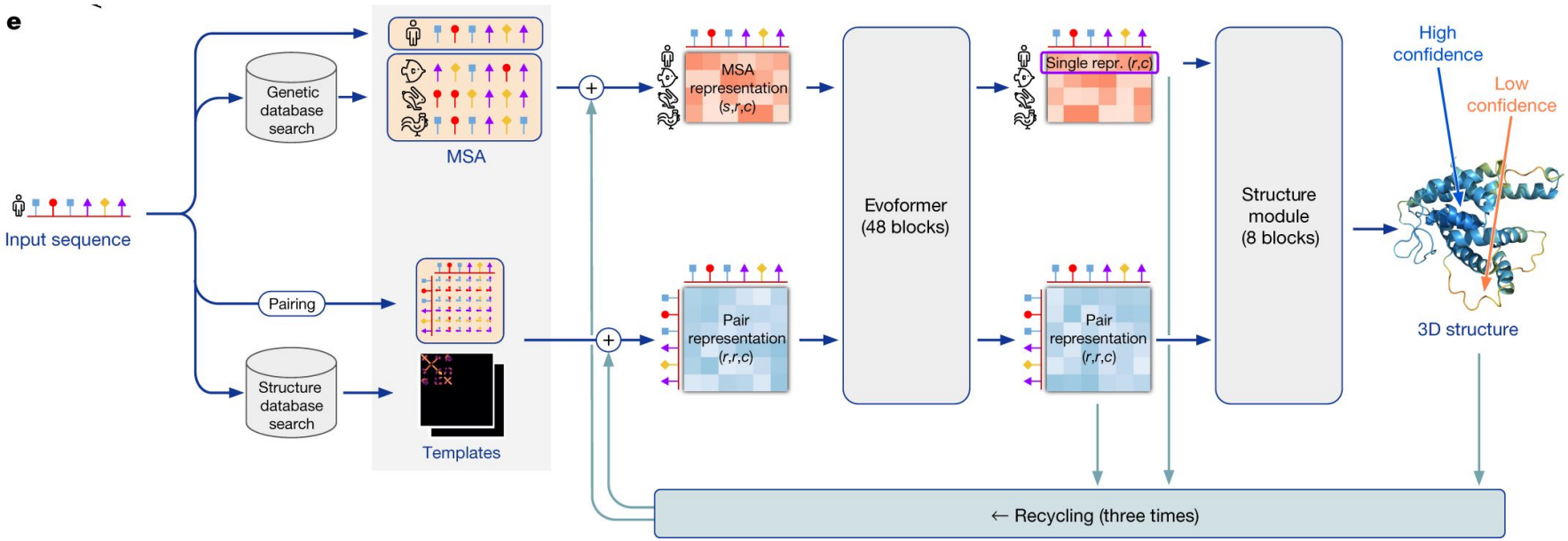


How is 3D structure stored in the pair representation?

Structure is stored in the pair representation tensor because it is trained to output pairwise distances (actually a distribution of distances) and orientation



Structure module - AlphaFold2



Jumper et al, Nature 2021 https://www.nature.com/articles/s41586-021-03819-2

Predicted LDDT

Predicted local distance difference test (ranges form 0-100)

pLDDT < 50 are not reliable

pLDDT > 70 backbone accurate but maybe not sidechains

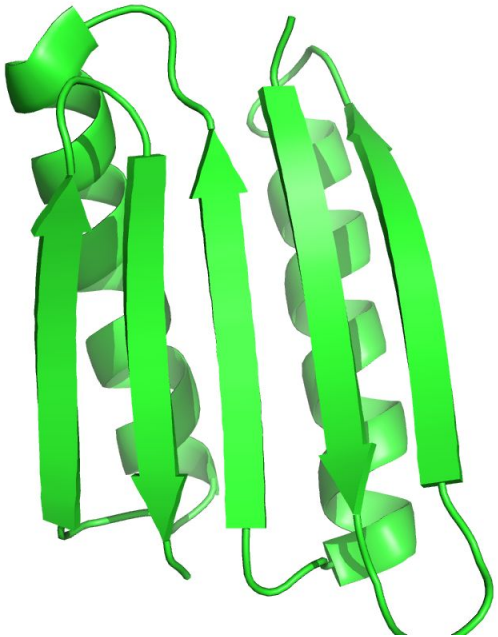
pLDDT > 90 suitable for docking, sidechains accurate

Protein design

Generate new proteins that fit some design criteria

- enzyme activity
- protein-protein binding
- molecular cage
- biosensors

Designed proteins can have novel folds, and stability greater than that of evolved proteins



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

Example protein design workflow

Generate target backbone

- Rational design
- RFDiffusion (diffusion model)

Generate example sequences eg, MPNN (uses GNNs)

Build sequence → structure using AlphaFold

Iterate at any step

State of the art - protein design

Protein target binders

Some small molecule binders

Novel protein folds

Quaternary interactions

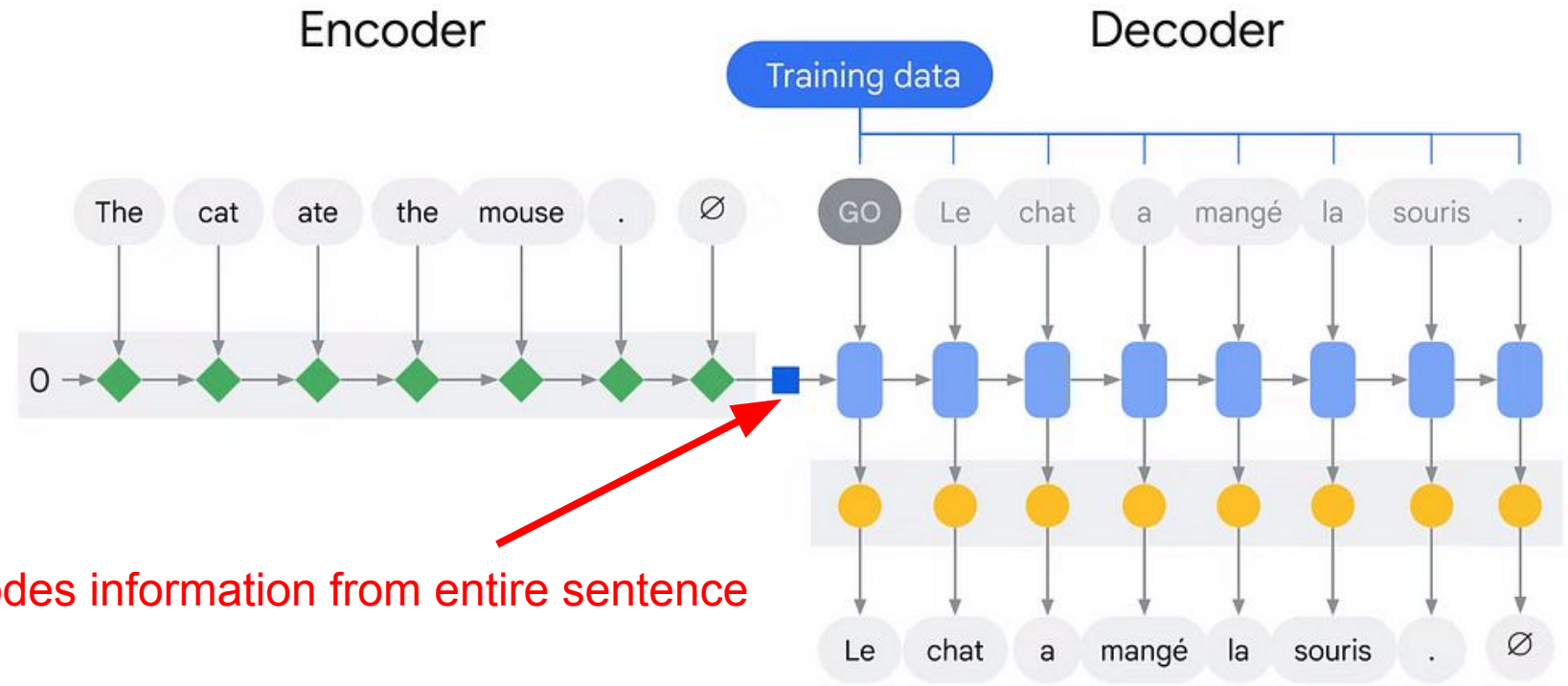
Horizons:

- Multiple conformations
- Enzyme activities
- Dynamics

Protein language models (PLMs)

PLMs are like LLMs (ChatGPT) but designed to generate the language of proteins

Encoder-decoder architecture



encodes information from entire sentence

https://www.cloudskillsboost.google/course_sessions/3415452/video/379211

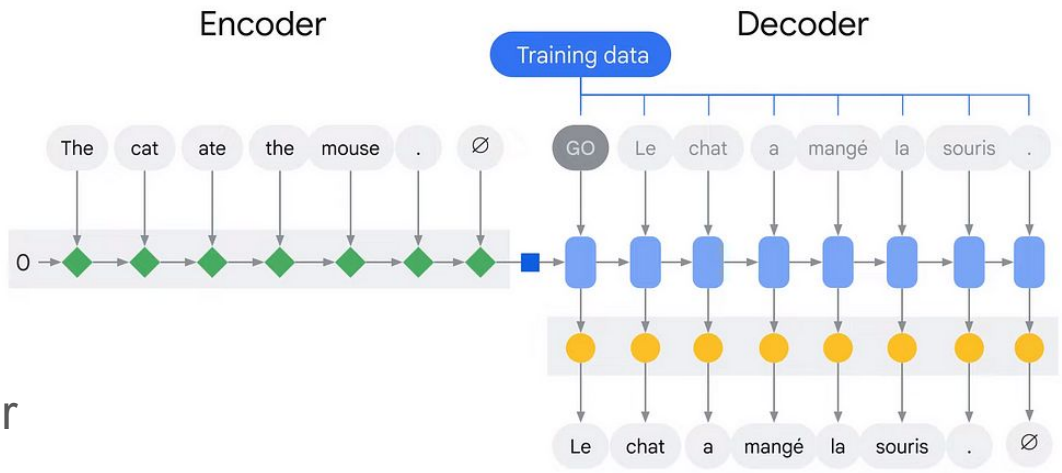
Encoder-decoder architecture

RNN processes sequence in order, so cat-ate-mouse != mouse-ate-cat

Difficult to train in practice

Idea: process entire sequence together so NN doesn't have to "remember" sequence

Let the model give "attention" to relevant parts of the sequence



https://www.cloudskillsboost.google/course_sessions/3415452/video/379211

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-

Overview of the Transformer achitecture

Encoder

Decoder

Embedding

Positional encoding

Attention mechanism

Residual connections

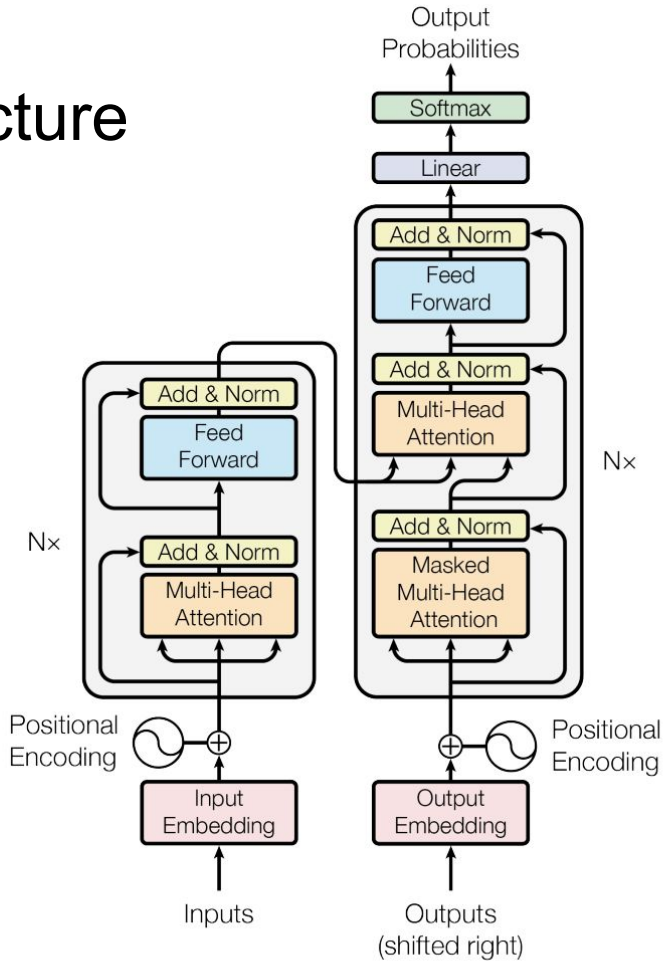
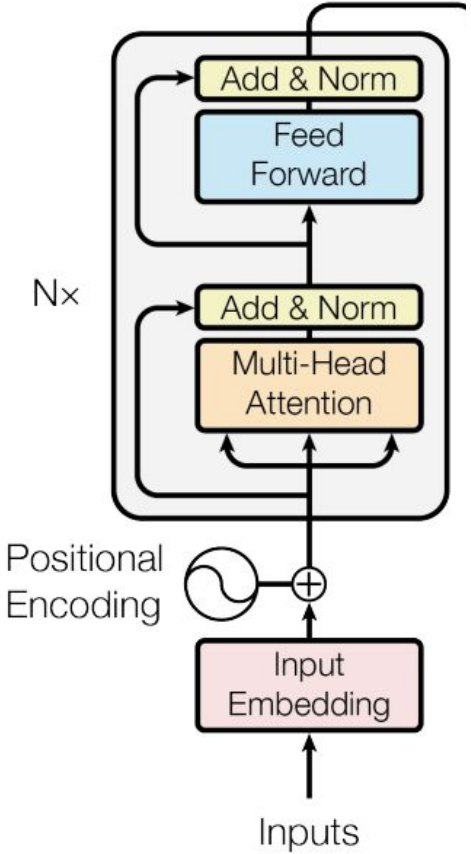


Figure 1: The Transformer - model architecture.

Encoder walkthrough

Input: sequence

Output: sequence in latent space with additional contextual information at each position



Tokenization

Convert input sequences to tensors (*e.g.*, words are not equal to tokens in GPT)

Include start, context separator, and padding tokens

Building a Transformers compatible tokenizer

```python
class DNATokenizer:
    def __init__(self, vocab=None, max_len=512):
        if vocab is None:
            self.vocab = {
                "A": 0, "C": 1, "G": 2, "T": 3,
                "[PAD]": 4, "[CLS]": 5, "[SEP]": 6,
"[UNK]": 7, "[MASK]": 8
            }
        else:
            self.vocab = vocab
        self.max_len = max_len
        self.pad_token_id = self.vocab["[PAD]"]

    def tokenize(self, sequence):
        """Converts a sequence of characters into a
list of tokens."""
        return list(sequence)

    def sequence_to_ids(self, sequence):
        """Converts a sequence to a list of IDs,
using the vocabulary."""
        return [self.vocab.get(token,
self.vocab["[UNK]"]) for token in sequence]
```

```python
    def pad_sequences(self, sequences_ids):
        """Pads sequences to the maximum length."""
        padded_sequences = [
            seq + [self.pad_token_id] * (self.max_len
- len(seq)) if len(seq) < self.max_len else
seq[:self.max_len]
            for seq in sequences_ids
        ]
        return padded_sequences

    def create_attention_masks(self,
padded_sequences):
        """Creates attention masks for sequences."""
        return [[0 if token_id == self.pad_token_id
else 1 for token_id in seq] for seq in
padded_sequences]

    def encode(self, sequences,
add_special_tokens=True):
        """Encodes a list of sequences into a format
compatible with Transformers models."""
        sequences_ids = []
        for sequence in sequences:
            tokens = self.tokenize(sequence)
            if add_special_tokens:
                tokens = ["[CLS]"] + tokens +
["[SEP]"]
```

```python
            sequence_ids =
self.sequence_to_ids(tokens)
            sequences_ids.append(sequence_ids)

        padded_sequences_ids =
self.pad_sequences(sequences_ids)
        attention_masks =
self.create_attention_masks(padded_sequences_ids)

        # Convert to PyTorch tensors
        padded_sequences_ids_tensor =
torch.tensor(padded_sequences_ids)
        attention_masks_tensor =
torch.tensor(attention_masks)

        return {
            "input_ids": padded_sequences_ids_tens
            "attention_mask": attention_masks_tens
        }
```

Input embedding

Convert tokens (int) to high dimensional embedding vector

For ESM2 protein language model, $n_{dim} = 1280$

Dimensionality is fixed throughout different layers

Size of protein is 1280 * seq_length features at each layer

Latent representation > input

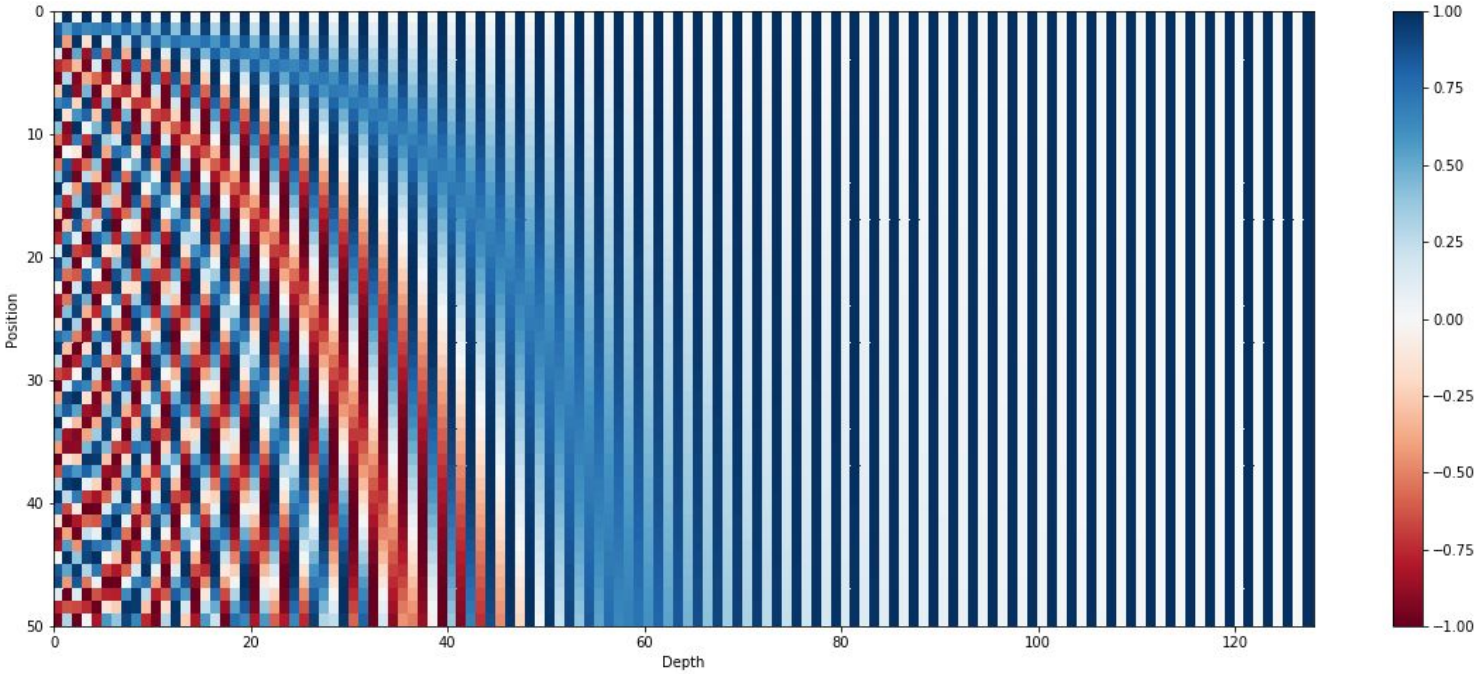
Positional embedding

Order matters, so we want to add positional data to each point in sequence

Initial paper described sinusoidal positional embeddings

Add vectors of sin/cos of varying frequency

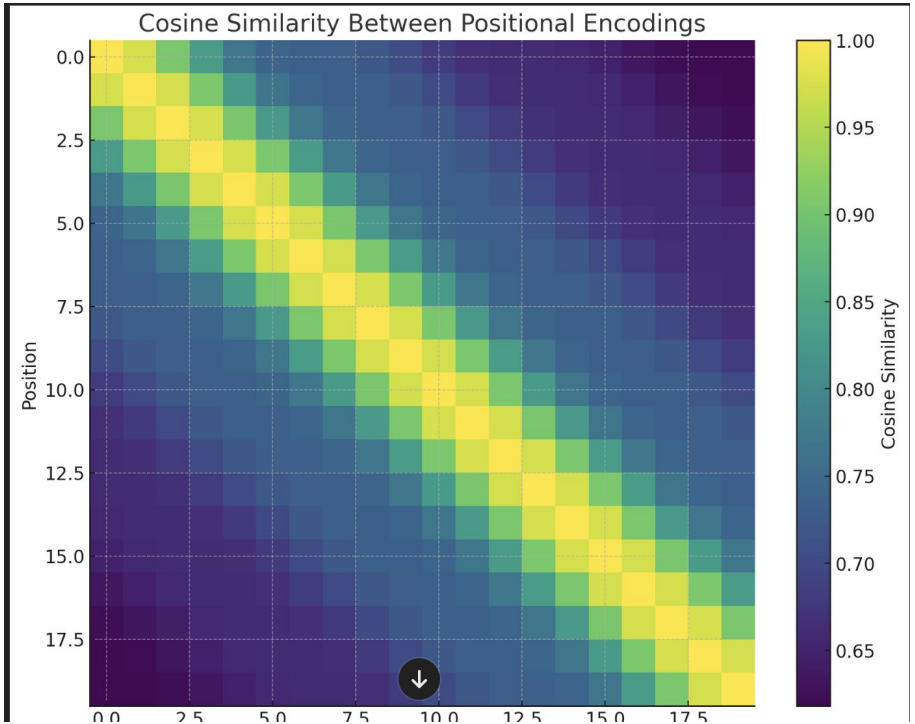
Sinusoidal encoding



Sinusoidal encoding

Mostly just measures pairwise distance

Most of signal is stored in the first few features



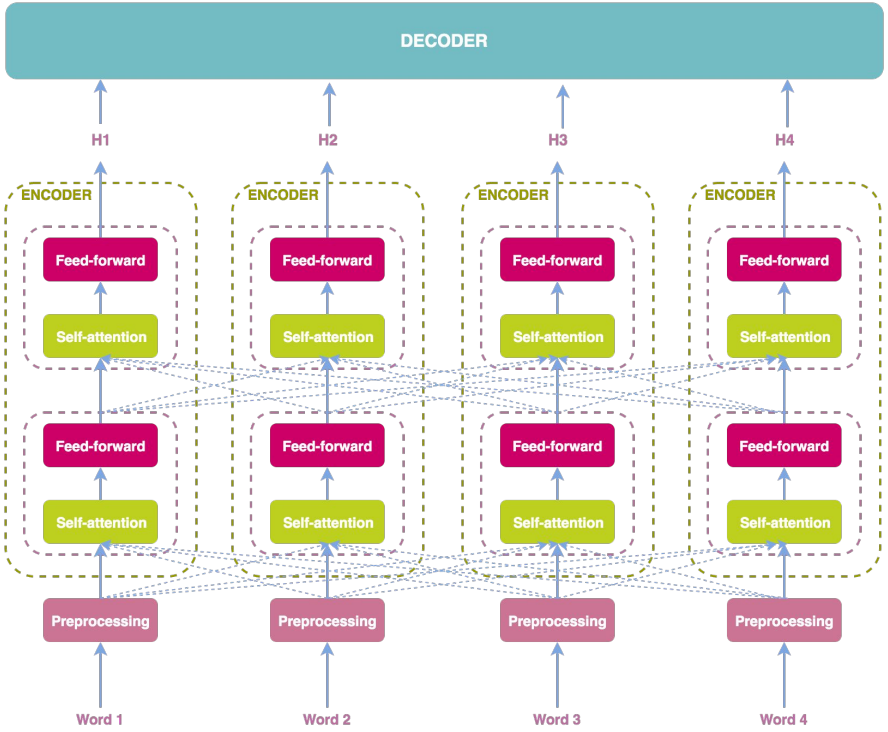
Information flow in transformer encoder is in parallel

Similar to RNN, each transformer layer is applied to each element in the sequence

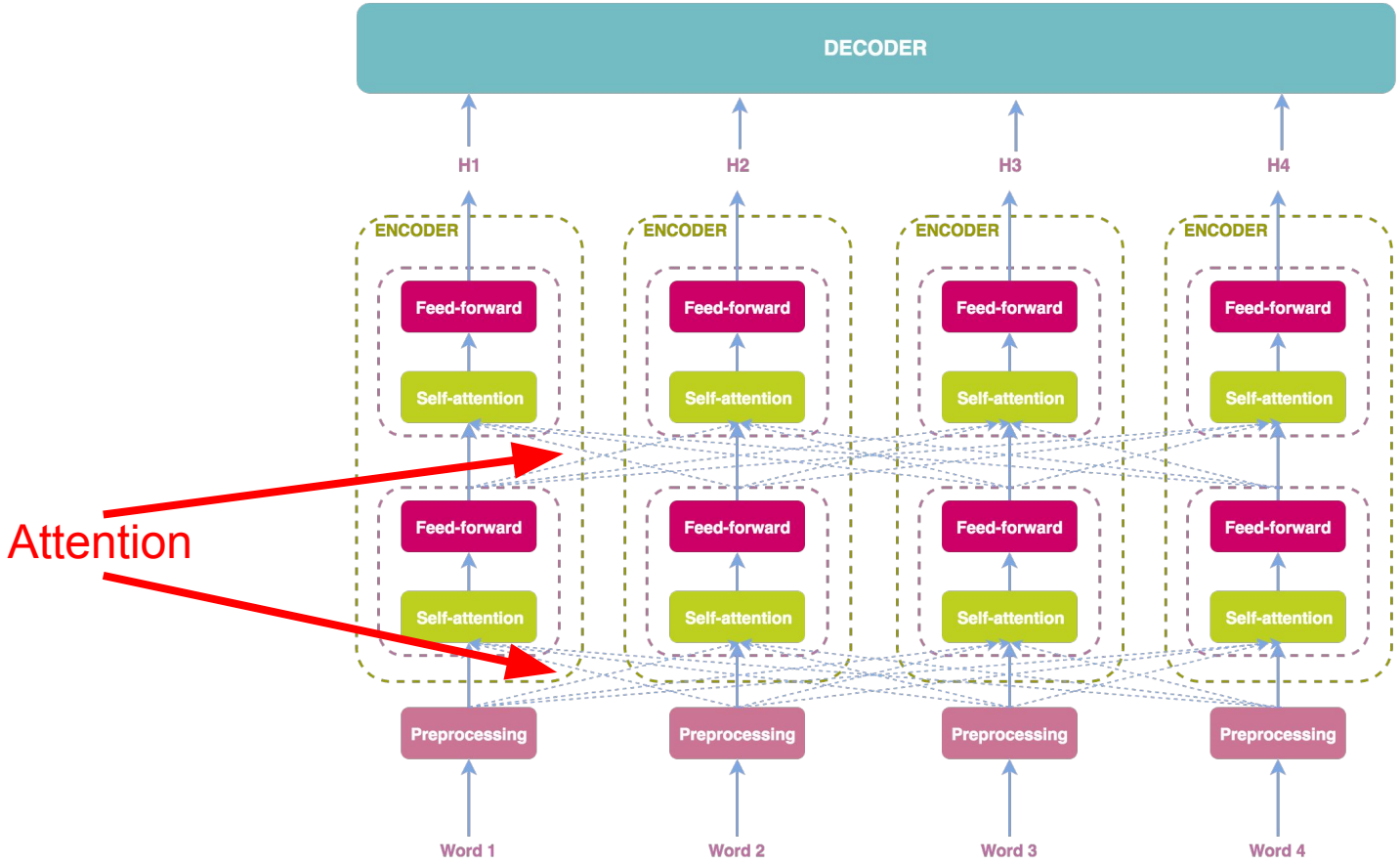
RNN: each element is passed in sequence

Transformer: all are passed in parallel

How is information shared? **Attention**



Information between words is shared using attention



Feed forward layer

2 fully connected linear layers with a non-linearity in between

```
d_model = 1280
d_ff = 2560
# Define the FFN using nn.Sequential
ffn = nn.Sequential(
    nn.Linear(d_model, d_ff),
    nn.ReLU(),
    nn.Linear(d_ff, d_model)

)
```

Layer Norm

Transformer architectures can be very deep

Batch norm is problematic when sequence length varies widely across samples

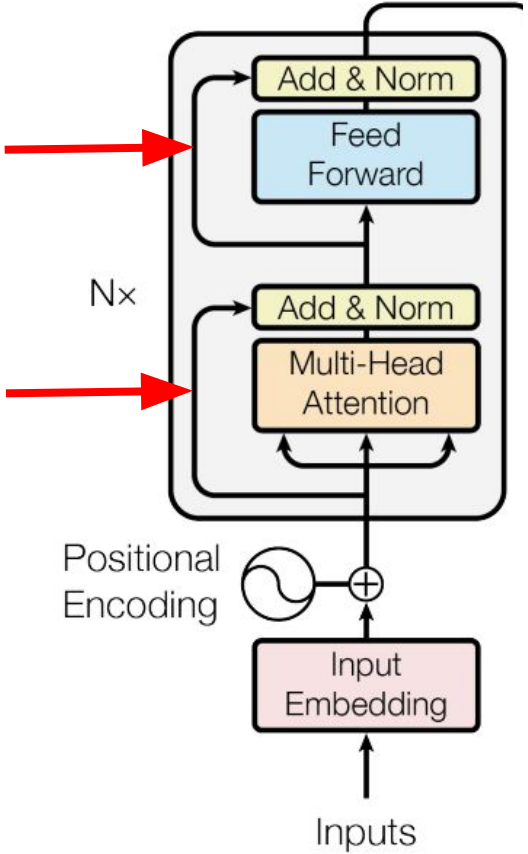
Layer norm: mean $\rightarrow$ 0 and variance $\rightarrow$ 1 for each element in the sequence

Residual connection

Also called skip connection

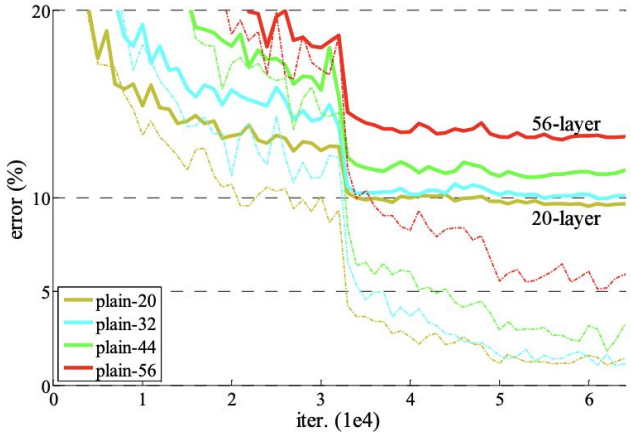
Adds input value to the output of the layer

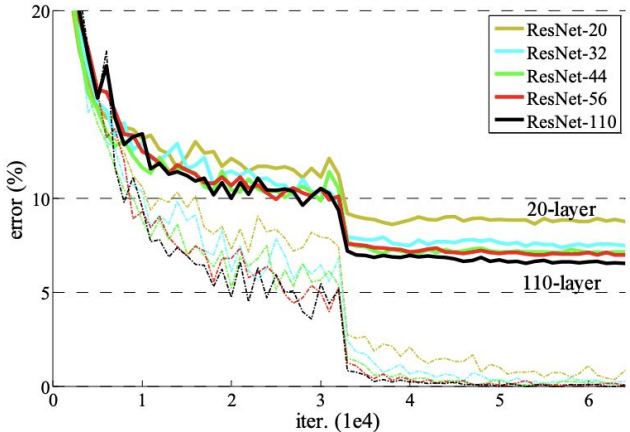
Allows gradients to pass through to avoid vanishing gradients



Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com





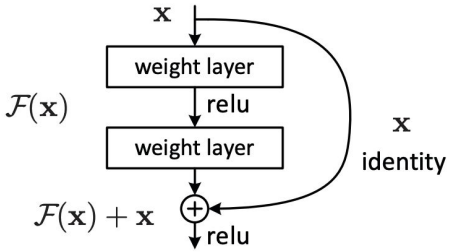


Figure 2. Residual learning: a building block.

Attention details

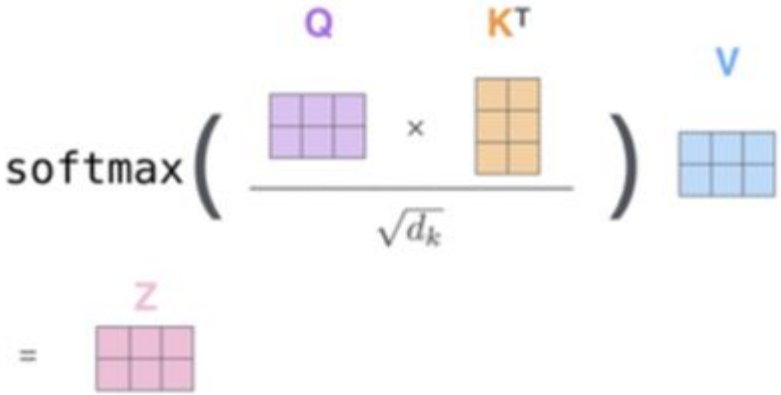
Data vectors in sequence are multiplied:

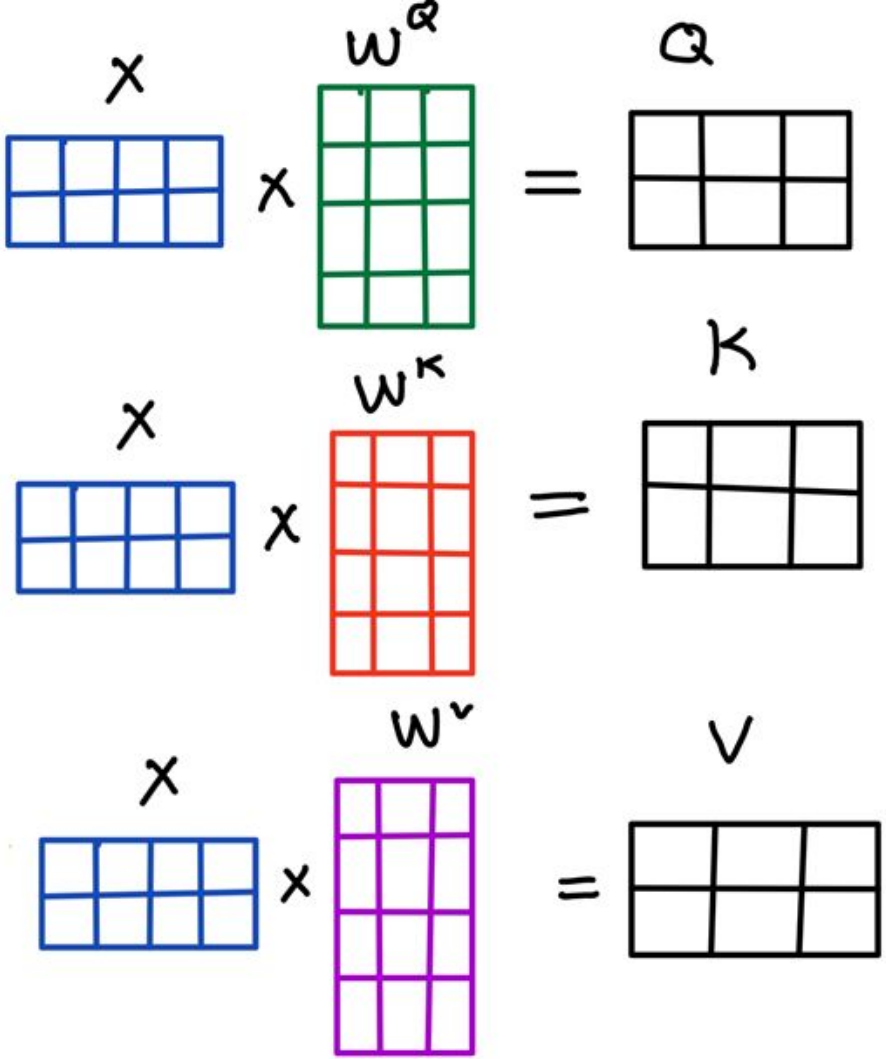
by a matrix W^Q to create query matrix, Q
by matrix W^K to create a matrix of keys, K
by matrix W^V to create a matrix of values, V

Attention is calculated as:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$

Attention details





Attention details

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$

units of probability

attention matrix

rescale ~1

Attention transformations

All transformations are linear layers - no activation functions

Maps square → parallelogram, hypercube → paralellpiped

Just decides which vectors are more or less important for each step, but doesn’t change proportions along any direction

Non-linearity comes afterwards in feed forward layer

Multihead attention

Typically 8 (or 16) attention heads

Each head has its own Q,K,V weight matrices

Each matrix produces a new representation in subspace (ndim/8), and these are recombined in a linear layer by a new matrix W^O

Positional encoding redux

Newer methods account for distance relationships without adding to semantic embedding vector

RoPE - rotate embedding vector depending on position in sequence

Alibi - compute pairwise distances independent of embedding vector

Alibi positional encoding

Add positional penalty to semantic similarity (QK^T)

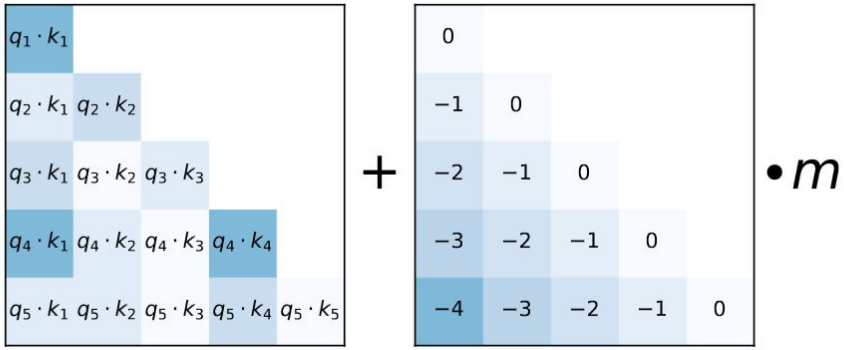
Constant m scales down geometrically across attention heads from ½ → $(½)^8$

Published as a conference paper at ICLR 2022

TRAIN SHORT, TEST LONG: ATTENTION WITH LINEAR BIASES ENABLES INPUT LENGTH EXTRAPOLATION

Ofir Press[1,2] **Noah A. Smith**[1,3] **Mike Lewis**[2]

[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Facebook AI Research
[3]Allen Institute for AI
ofirp@cs.washington.edu



Next up - transformer decoder and generative AI!

Transformers pt. 2

March 19, 2024

Review - transformer achitecture

Encoder used attention to enable information to pass bw different tokens

Decoder is similar w changes related to attention

Decoder is a generative model

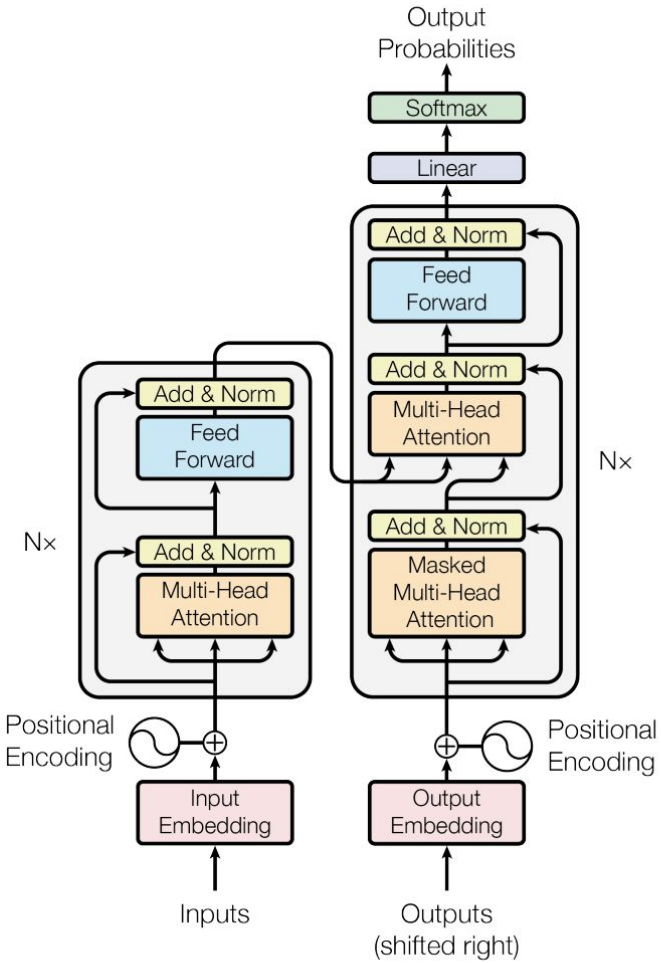


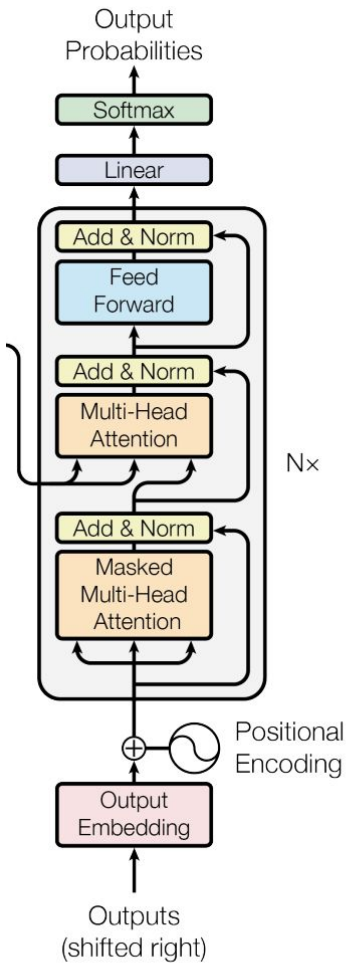
Figure 1: The Transformer - model architecture.

Transformer decoder vs. RNN

Both are autoregressive, and generate sequence one element at a time after feeding previous output back

Transformer processes all previous tokens simultaneously at each step

RNN passes hidden state between time steps



Transformer decoder walkthrough

Decoder only transformers are important generative tools, *e.g.*, ChatGPT

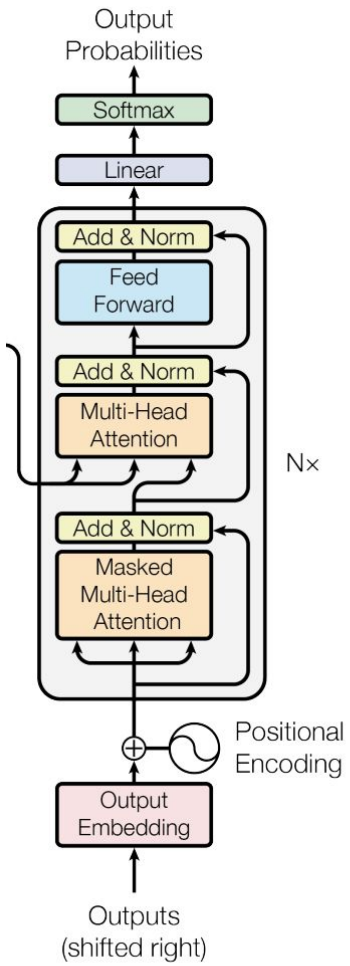
Semantic embedding

Positional embedding

Transformer layers:
- Self-attention
- Cross-attention
- Feed forward neural net

Linear output layer w Softmax

Attention maps themselves can be useful output (e.g., protein structure prediction)



Cross attention

Same as self-attention, except:

W^Q is applied to output from previous decoder layer
W^K is applied to final output from encoder

What about the value matrix, W^V?

W^O as before to combine output from multiple attention heads

Positional encoding generally used, but not clear that it is meaningful in cross attention

Which comes first self or cross attention?

Training a decoder

Train decoder to predict the next token in a sequence

Problem: if the first token doesn’t match training example, then the rest of the sequence will likely be wrong even if the model is making accurate predictions

Solution: force the model to select the correct answer, even if its predicted probability is low (teacher forcing)

This approach can be done very efficiently in the context of transformer architecture

Using attention to mask downstream tokens

Objective: train in parallel on an entire sequence rather than feeding in each token one by one

If we are predicting token i+1, then only consider tokens j, where j <= i

Computationally much more efficient than feeding tokens one by one

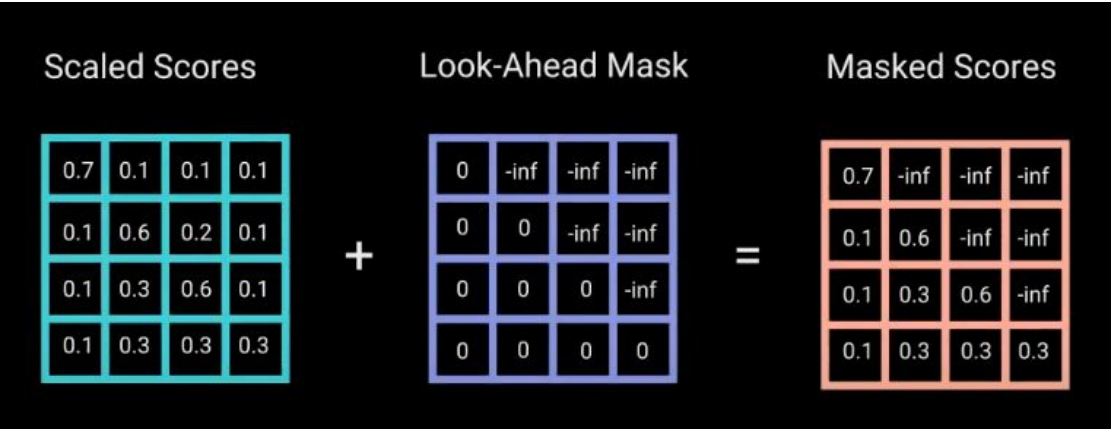
Attention masking

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$

units of probability

attention matrix

rescale ~1



Putting it all together - transformer Encoder-decoder models

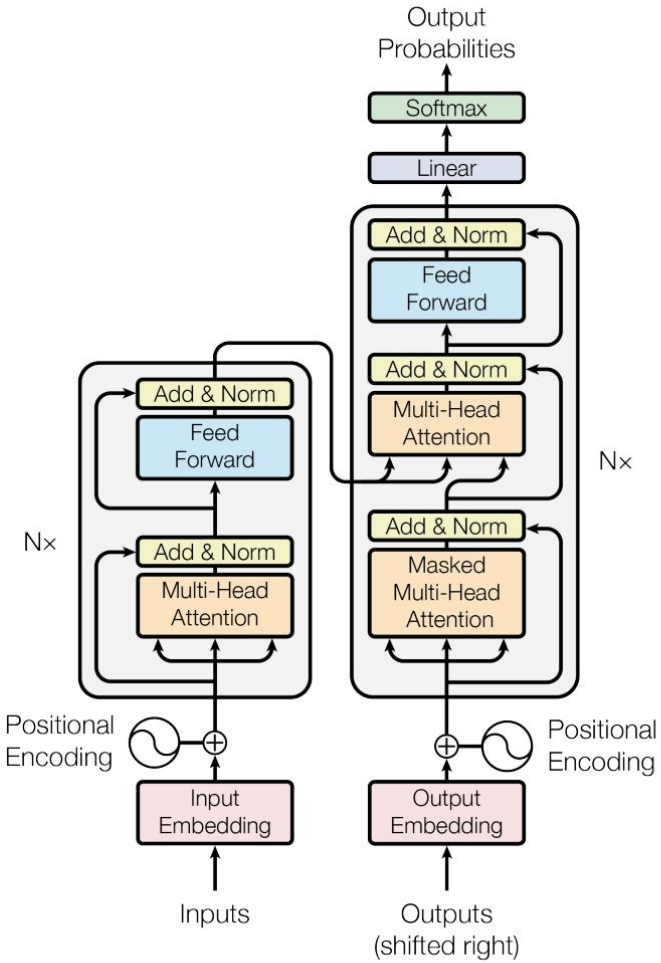


Figure 1: The Transformer - model architecture.

Encoder-decoder architectures

Encoder-decoder is like a game bw two players

Encode sequences of arbitrary length as fixed dimension feature vector (picture)

Decode feature vector into new example drawn from natural distribution

Machine translation

Image captioning

Sentiment analysis

Encoders

Dimensionality reduction - project complex data into a more meaningful and compact latent space (compression, noise reduction)

Dimensionality expansion - project data into a higher dimensional space to enable more complex patterns to be identified

Supervised vs. unsupervised training - can use known labels to help structure the latent space

Example: protein language models - generally encoder only, aimed at understanding the complexity of protein sequences (dimensionality expansion)

Decoders

Central role in generative AI

Encoders take complex data and encode it into meaningful latent space

Data generation - decoders use latent space to generate new instances of data

Sequence generation - can be called repeatedly for sequence data

Conditional generation - can be conditioned on specific context

Generative search algorithms

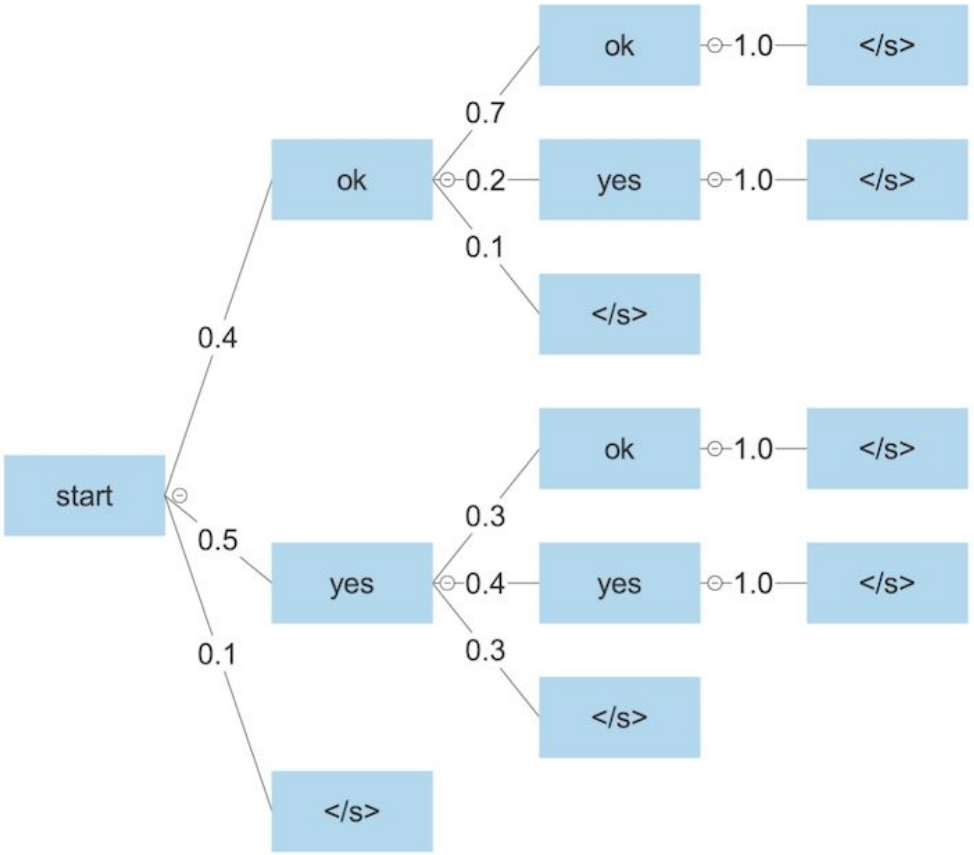
Greedy - take most likely output at each step
Heuristic - fast and compact in memory

Exhaustive - search all possible outputs at each step
Guaranteed to find optimal solution, but too expensive except for small problems

Beam search - follow k top paths at each step, where k is beam width
Heuristic compromise between greedy and exhaustive

Example search

Find greedy and beam search (k=2) solutions



https://builtin.com/software-engineering-perspectives/beam-search

Stopping condition

Stop each path when <EOS> token is predicted

Problem: each probability < 1, so shorter sequences are preferred

Normalize sequence length by dividing likelihood by number of tokens

What is generative AI?

Subset of AI focused on creating new data instances drawn from the distribution of the training data

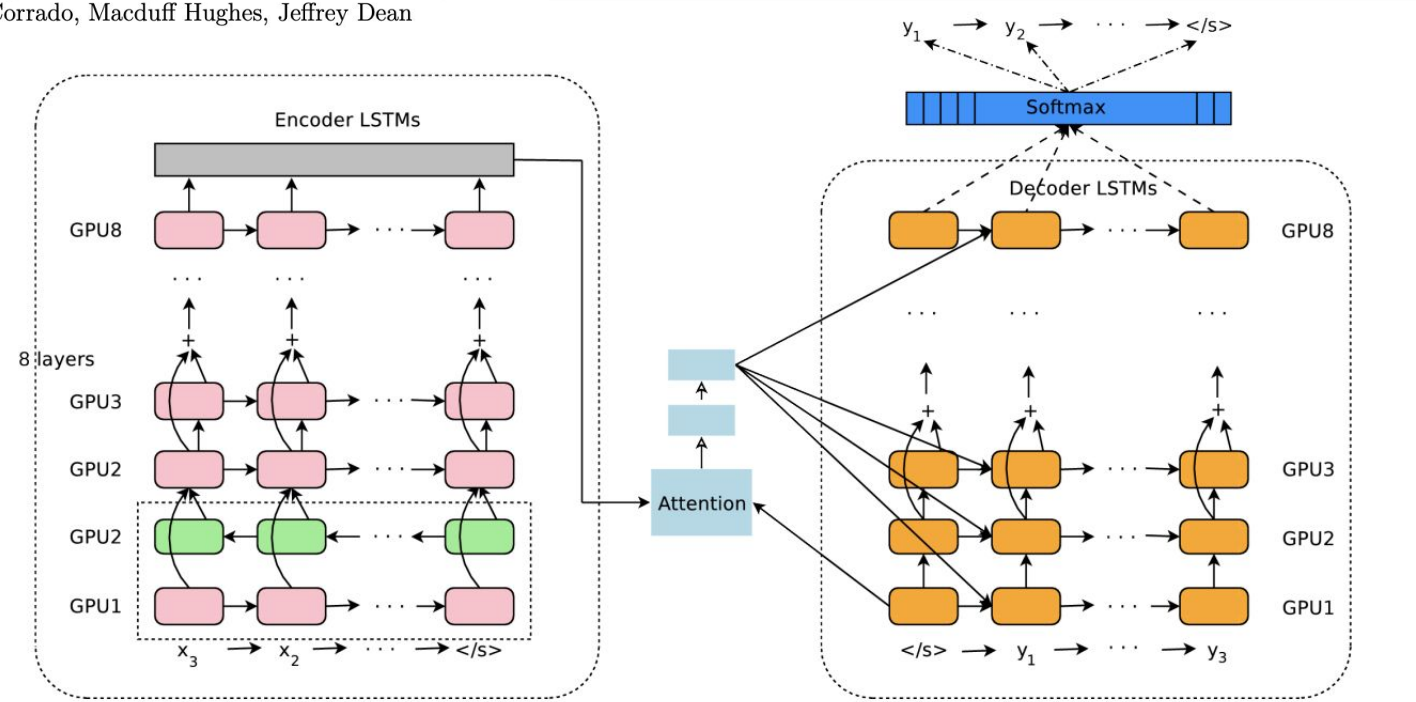
Generative vs discriminative models

ChatGPT, DALL-E, StableDiffusion, Tesla full self-driving model

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean



Problem: generate new compounds for *in silico* drug screening

Developing new lead compounds for drug screening is expensive and slow

in silico generation and screening of compounds would speed drug development

Potentially useful compounds may not exist in screening libraries, and therefore are never tested

10^{60} possible drug-like molecules, most screening libraries $<10^{6}$

Outline

Sample distribution of known compounds

Build a model of underlying distribution

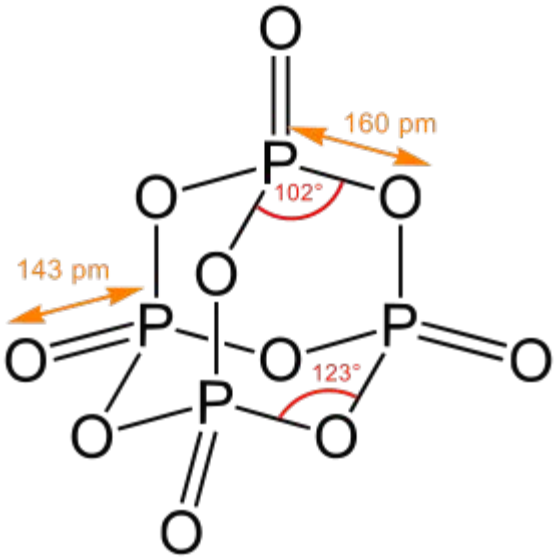
Decode samples from distribution into chemical structures

Representing small molecule drugs

3D graph representation
-captures spatial arrangement and conformation
-difficult to store, compute, search

SMILES (Simplified Molecular Input Line Entry System)
-compact and easy to read
-widely used
-multiple representations for given molecule
O=P(O)(O)OP(=O)(O)OP(=O)(O)OP(=O)(O)OP(=O)(O)O

InChi (International Chemical Identifier)
-unique identifier
-lengthy and hard to read, parse



Sampling the distribution of known compounds

ChEMBL

Search in ChEMBL

Current Release: ChEMBL 33

Provided under a Creative Commons Attribution-ShareAlike 3.0 Unported license

Last Update on 2023-05-31T00:00:00 | Release notes

15,398	2,399,743	20,334,684	88,630	215
Targets	Distinct compounds	Activities	Publications	Deposited Datasets

SMILES notation - 5 rules

1. **Atoms & bonds**
   Represented by their chemical symbol (Na,C,F)
   Lower case - aromatic (c1ccccc1)

   - Single bond
   = Double bond
   # Triple bond
   * Aromatic bond
   . Disconnected structures

SMILES notation - 5 rules

2. Simple chains

Hydrogens are usually suppressed

CC	CH3CH3	Ethane
C=C	CH2CH2	Ethene
CBr	CH3Br	Bromomethane
C#N	C=N	Hydrocyanic acid
Na.Cl	NaCl	Sodium chloride

SMILES notation - 5 rules

3. Branches

Branches are specified by parentheses, connect to preceding atom
Includes bond character after left parenthesis if needed

CC(O)C	2-Propanol
CC(=O)C	2-Propanone
CC(CC)C	2-Methylbutane
CC(C)CC(=O)	2-Methylbutanal
c1c(N(=O)=O)cccc1	Nitrobenzene
CC(C)(C)CC	2,2-Dimethylbutane

SMILES notation - 5 rules

4. Rings

Rings are specified by numbers. Same number indicates opening/closing atoms of ring. Bond type comes after atom but before number.

C=1CCCCC1	Cyclohexene
C*1*C*C*C*C*C1	Benzene
c1ccccc1	Benzene
C1OC1CC	Ethyloxirane
c1cc2ccccc2cc1	Naphthalene

SMILES notation - 5 rules

5. Charges
Charge is specified in {}

CCC(=O)O{-1} Ionized form of propanoic acid

CCC(=O)O{-}

c1ccccn{+1}1CC(=O)O 1-Carboxylmethyl pyridinium

c1cc2ccccc2cc1 Naphthalene

SMILES notation

Ambiguous names
[] used to separate individual atoms

Sc → sulfur + aromatic carbon

[Sc] → Scandium

SMILES Tokenizer

```python
def preprocess_smiles(smiles):
    # 1. Separate all characters
    smiles = ' '.join(list(smiles))

    # 2. Join characters that represent two-letter elements (e.g., 'Cl', 'Br', 'Na', etc.)
    smiles = re.sub(r'([A-Z]) ([a-z])', r'\1\2', smiles)

    # 3. Separate elements of the form '[A-Z]c', if necessary
    smiles = re.sub(r'([A-Z])c', r'\1 c', smiles)

    # Remove any extra spaces
    smiles = ' '.join(smiles.split())

    # Add termination character
    smiles += ' .'

    return smiles.split()
```

Simple model

Generate next character by looking at context_length of previous characters

Pad each molecule with context_length <PAD> characters in front and one <EOS> character at end

Simple model

```
context_len = 3

model = nn.Sequential(
    nn.Embedding(len(itos), 128),
    nn.Flatten(),
    nn.Linear(128*context_len,128*context_len),
    nn.ReLU(),
    nn.Linear(128*context_len, len(itos))
)
```

Protein docking