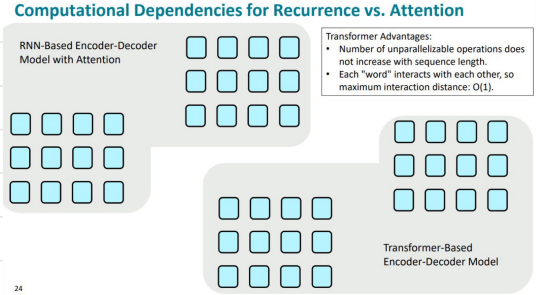
# TRANSFORMER

## What Is a Transformer?

The Transformer architecture revolutionised NLP by eliminating recurrence and convolution entirely and relying solely on attention mechanisms.

| Traditional RNN/LSTM | Transformer |
| --- | --- |
| Sequential processing | Fully parallelizable |
| Bottleneck in information | Simultaneous access to all tokens |
| Difficult to train for long sequences | Better with long-range dependencies |

### Computational Dependencies for Recurrence vs. Attention




24

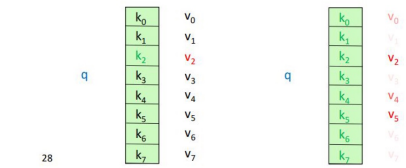
### Attention

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.


- Sequential models ingest the input one word or one token at the time. And so, as as if each unit was like a bottleneck to the flow of information.
- Transformer ingest an entire sentence all at the same time.
- Attention mechanism + CNN like parallelism.

### Intuition for Attention Mechanism

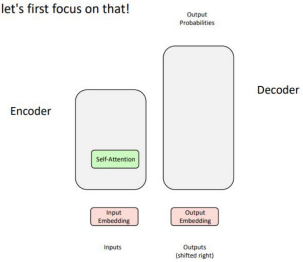
- Let's think of attention as a "fuzzy" or approximate hashtable:
  - To look up a **value**, we compare a **query** against **keys** in a table.
  - In a hashtable (shown on the bottom left):
    - Each **query** (hash) maps to exactly one **key**-**value** pair.
  - In (self-)attention (shown on the bottom right):
    - Each **query** matches each **key** to varying degrees.
    - We return a sum of **values** weighted by the **query**-**key** match.



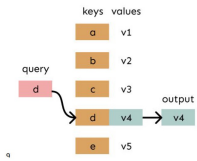

28

### Encoder: Self-Attention

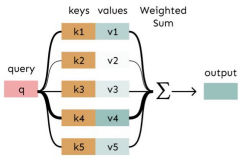
Self-Attention is the core building block of Transformer, so let's first focus on that!




In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.




In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.




9

## Recipe for Self-Attention in the Transformer Encoder

- Step 1: For each word $x_i$ , calculate its **query**, **key**, and **value**.

  $q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$

- Step 2: Calculate attention score between **query** and **keys.**

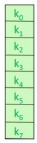
  $e_{ij} = q_i \cdot k_j$

- Step 3: Take the softmax to normalize attention scores.

  $\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}$

- Step 4: Take a weighted sum of **values**.

  $Output_i = \sum_j \alpha_{ij} v_j$




## Recipe for (Vectorized) Self-Attention in the Transformer Encoder

- Step 1: With embeddings stacked in X, calculate **queries**, **keys**, and **values**.

  $Q = XW^Q \quad K = XW^K \quad V = XW^V$

- Step 2: Calculate attention scores between **query** and **keys.**

  $E = QK^T$

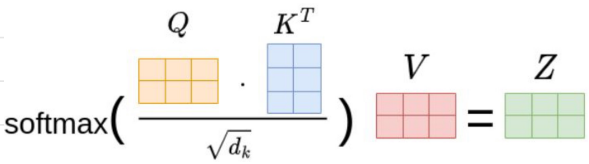
- Step 3: Take the softmax to normalize attention scores.

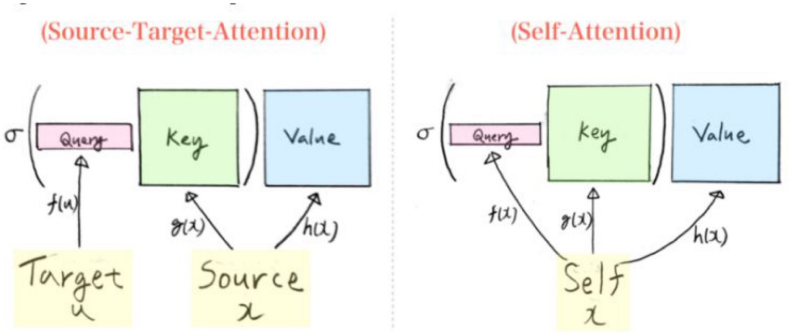
  $A = softmax(E)$

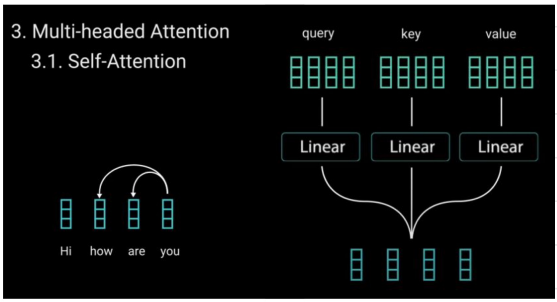
- Step 4: Take a weighted sum of **values**.

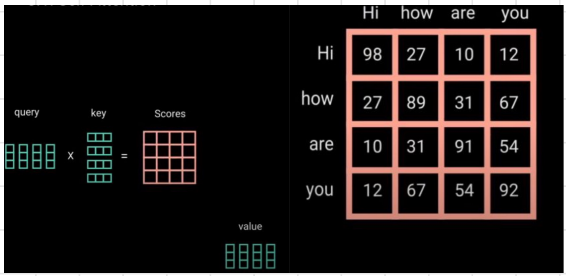
  $Output = AV$

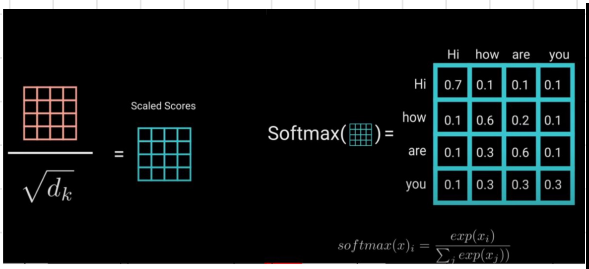
$$Output = softmax\left(QK^T\right)V$$

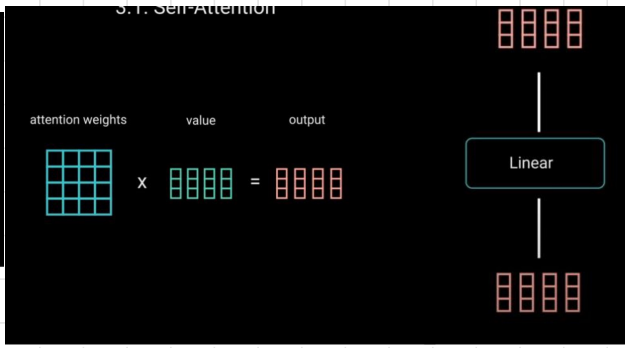


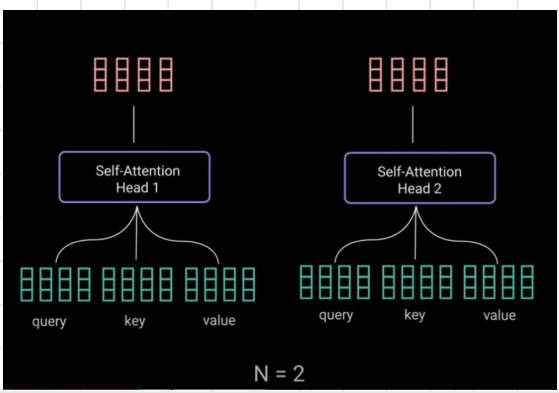



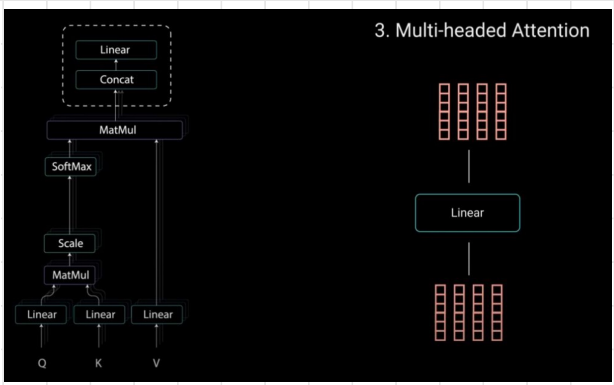

3. Multi-headed Attention
3.1. Self-Attention
query
key
value
Linear
Linear
Linear
Hi
how
are
you


query
key
Scores
x
=
value
Hi how are you
Hi 98 27 10 12
how 27 89 31 67
are 10 31 91 54
you 12 67 54 92


Scaled Scores
=
Softmax( ) =
Hi how are you
Hi 0.7 0.1 0.1 0.1
how 0.1 0.6 0.2 0.1
are 0.1 0.3 0.6 0.1
you 0.1 0.3 0.3 0.3
$\sqrt{d_k}$
$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j))}$


3.1. Self-Attention
attention weights
value
output
x
=
Linear


Self-Attention Head 1
Self-Attention Head 2
query
key
value
query
key
value
N = 2


Linear
Concat
MatMul
SoftMax
Scale
MatMul
Linear
Linear
Linear
Q
K
V
3. Multi-headed Attention
Linear

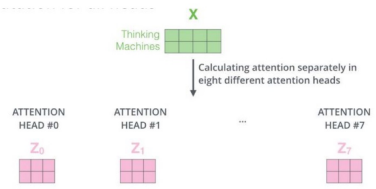
**Multi-Head Attention**

Multi-head attention = performing self-attention multiple times with different learned weight matrices (i.e., different "heads").

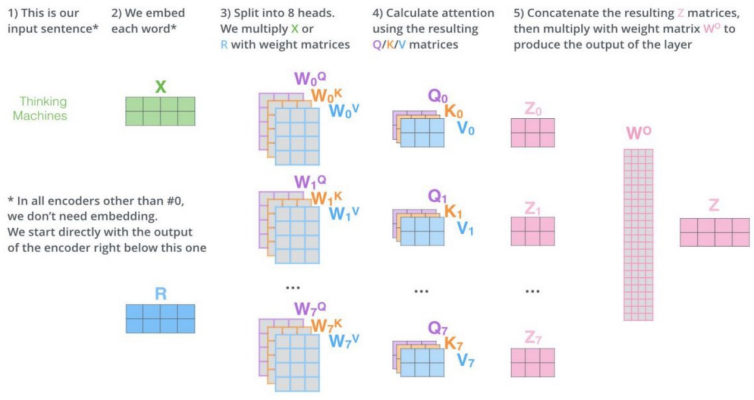
Each head:
Looks at the input from a different representation subspace.
Learns diverse relationships (e.g., syntax, semantics, position).
These are concatenated and passed through another linear layer.

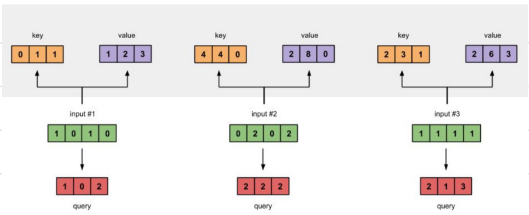
Think of each head as asking a different question about the same sentence.

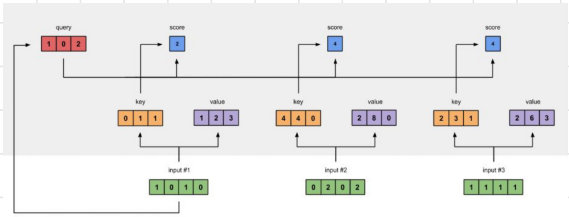
Multi-Head Attention

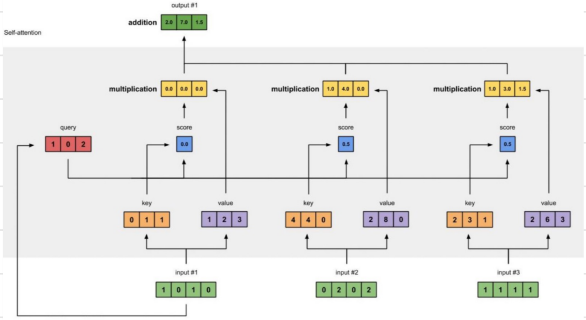


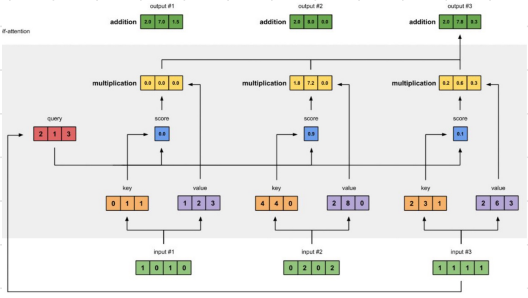

- Doing self-Attention multiple times (Multi-Head).
- As if you are asking a different query about the input multiple times.
- Parallel computation for all heads



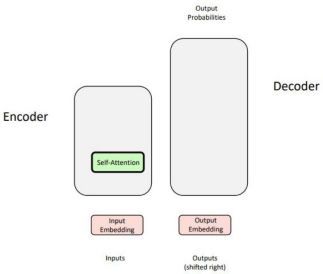

key
value
input #1
input #2
input #3
query


query
score
key
value
input #1
input #2
input #3


Self-attention
output #1
addition
multiplication
query
score
key
value
input #1
input #2
input #3


output #1
output #2
output #3
addition
multiplication
query
score
key
value
input #1
input #2
input #3

## What We Have So Far: (Encoder) Self-Attention!

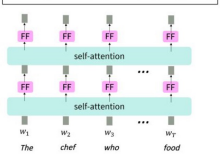


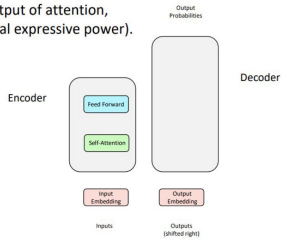

## But attention isn't quite all you need!

- **Problem:** Since there are no element-wise non-linearities, self-attention is simply performing a re-averaging of the value vectors.
- **Easy fix:** Apply a feedforward layer to the output of attention, providing non-linear activation (and additional expressive power).

Equation for Feed Forward Layer

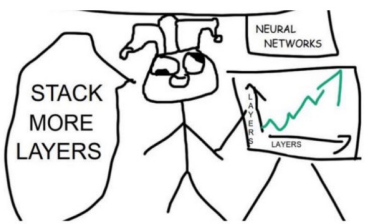
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



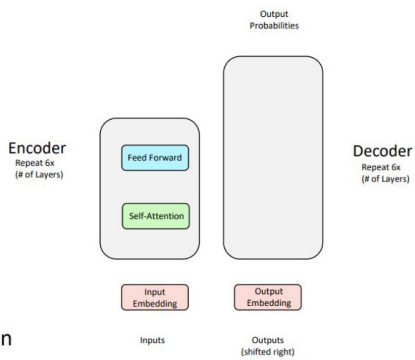




32

## But how do we make this work for deep networks?




Training Trick #1: Residual Connections
Training Trick #2: LayerNorm
Training Trick #3: Scaled Dot Product Attention

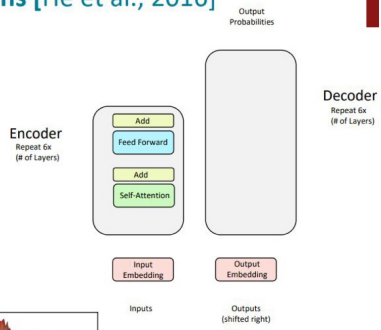


## Training Trick #1: Residual Connections [He et al., 2016]

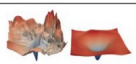
- Residual connections are a simple but powerful technique from computer vision.
- Deep networks are surprisingly bad at learning the identity function!
- Therefore, directly passing "raw" embeddings to the next layer can actually be very helpful!

$$x_\ell = F(x_{\ell-1}) + x_{\ell-1}$$

- This prevents the network from "forgetting" or distorting important information as it is processed by many layers.




Residual connections are also thought to smooth the loss landscape and make training easier!



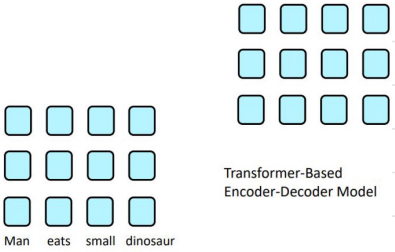

[Loss landscape visualization, Li et al., 2018, on a ResNet]



## Major issue!

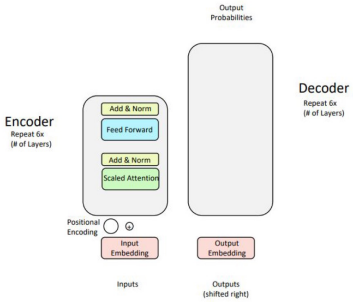
- We're almost done with the Encoder, but we have a major problem! Has anyone spotted it?
- Consider this sentence:
  - "Man eats small dinosaur."

$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$




Transformer-Based Encoder-Decoder Model

## **Solution:** Inject Order Information through Positional Encodings!



## Fixing the first self-attention problem: **sequence order**

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

  $p_i \in \mathbb{R}^d$, for $i \in \{1,2,\ldots,T\}$ are position vectors

- Don't worry about what the $p_i$ are made of yet!
- Easy to incorporate this info into our self-attention block: just add the $p_i$ to our inputs!
- Let $\tilde{v}_i\ \tilde{k}_i, \tilde{q}_i$ be our old values, keys, and queries.

$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...

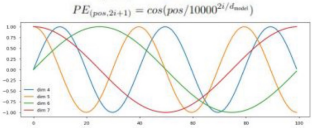
**Positional Encoding**
Since transformers don't process input sequentially (no recurrence), they add position information manually.

Positional Encoding (usually sine/cosine) is added to each input embedding to capture word order.

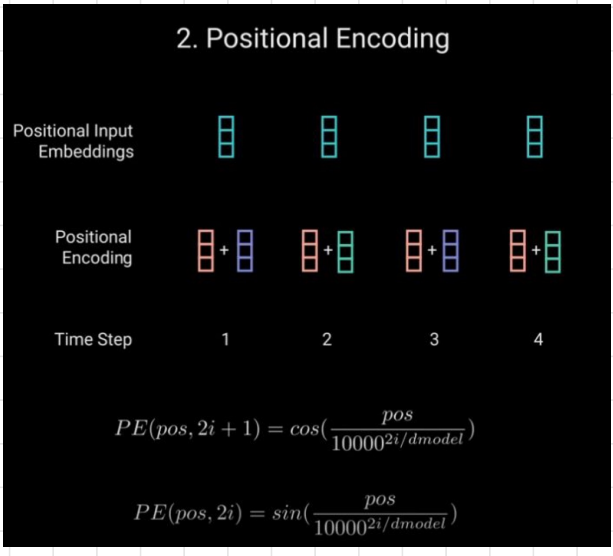
## Positional Encoding

- Account for the order of the words in the input sequence.
- Adds a vector to each input embedding. These vectors follow a specific pattern that helps determine the position of each word.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

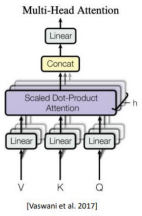


$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \vdots \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$



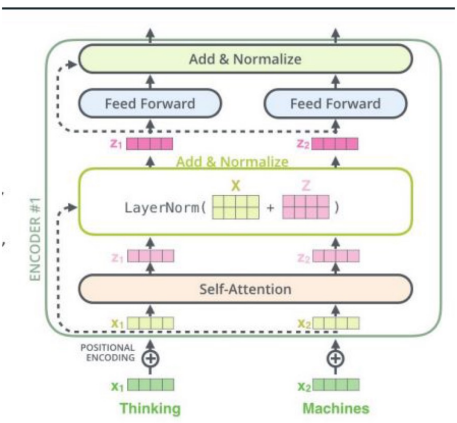
## Multi-Headed Self-Attention: k heads are better than 1!

- **High-Level Idea:** Let's perform self-attention multiple times in parallel and combine the resu

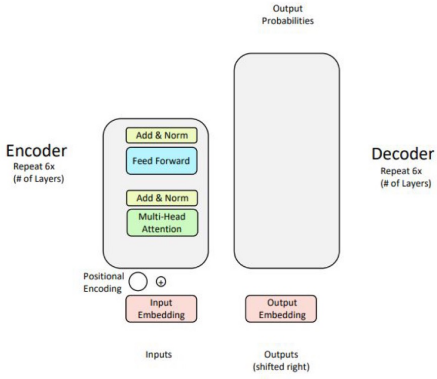


[Vaswani et al. 2017]


Wizards of the Coast, Artist: Todd Lockwood

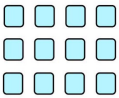


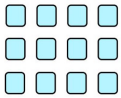

## Yay, we've completed the Encoder! Time for the Decoder...



## Decoder: Masked Multi-Head Self-Attention

- **Problem:** How do we keep the decoder from "cheating"? If we have a language modeling objective, can't the network just look ahead and "see" the answer?
- **Solution:** Masked Multi-Head Attention. At a high-level, we hide (mask) information about future tokens from the model.



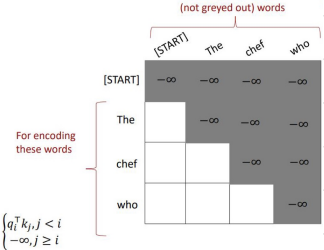


Transformer-Based Encoder-Decoder Model

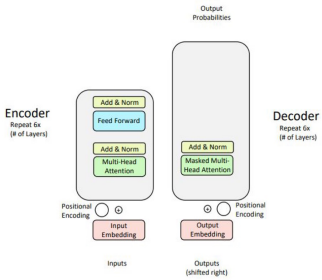
## Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^T k_j, j < i \\ -\infty, j \geq i \end{cases}$$

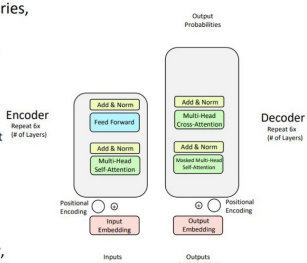



## Decoder: Masked Multi-Headed Self-Attention



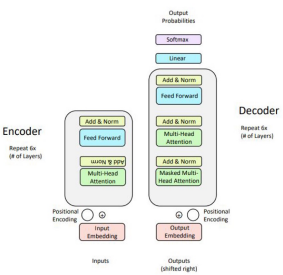

## Encoder-Decoder Attention

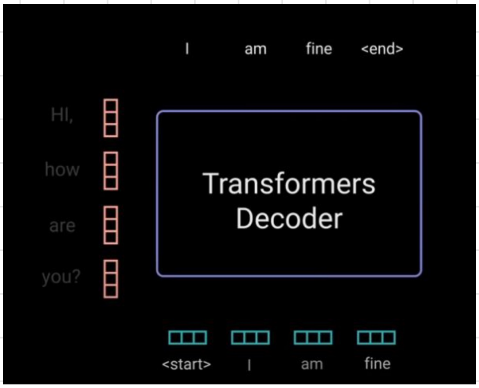
- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \ldots, h_T$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \ldots, z_T$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.

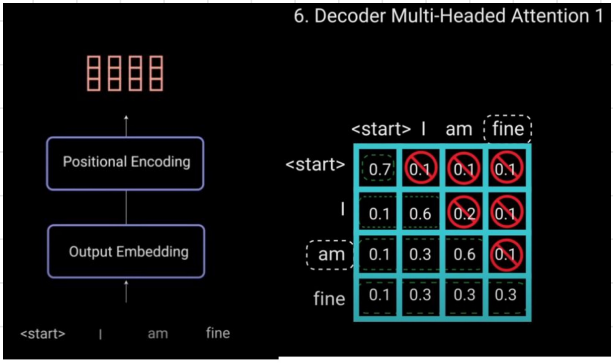


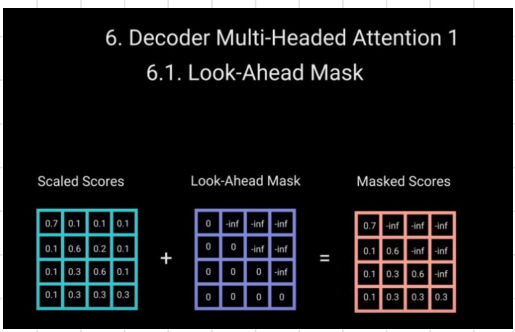

## Decoder: Finishing touches!

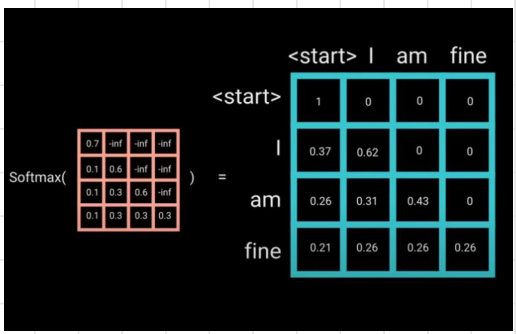
- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate a probability distribution of possible next words!

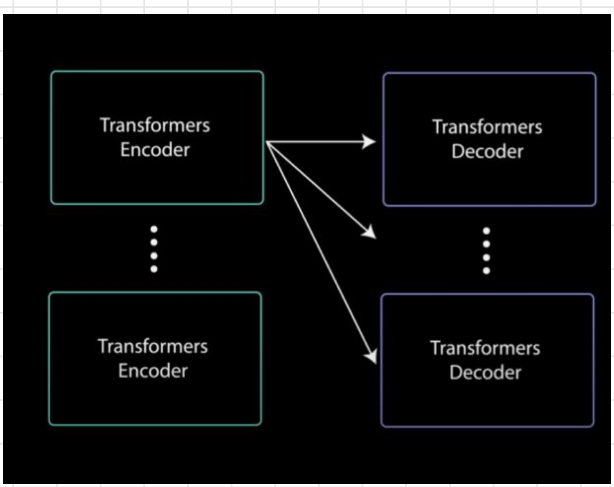




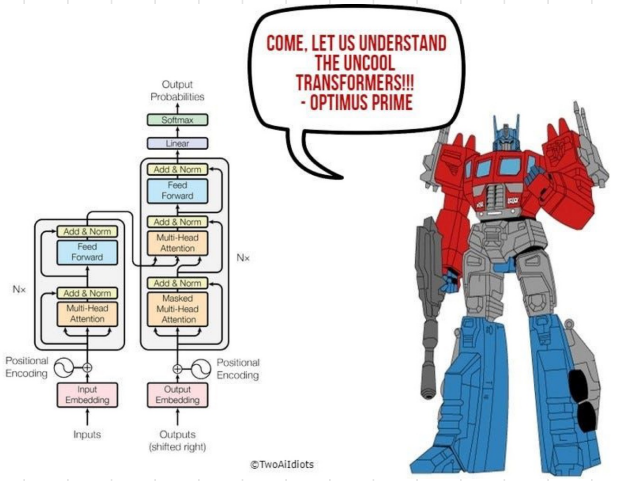




**Transformer Architecture**
A typical Transformer block contains:

🔷 Encoder Block:
Input embeddings + positional encoding
Multi-head self-attention
Feedforward neural network
Layer normalization and residual connections

🔷 Decoder Block:
Masked multi-head self-attention
Cross-attention: attends over encoder output
Feedforward + layer norm + residuals




## What would we like to fix about the Transformer?

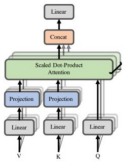
- **Quadratic compute in self-attention (today)**:
  - Computing all pairs of interactions means our computation grows **quadratically** with the sequence length!
  - For recurrent models, it only grew linearly!
- **Position representations**:
  - Are simple absolute indices the best we can do to represent position?
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

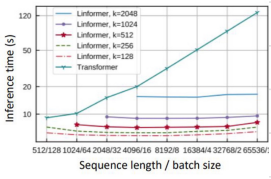
## Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*

- For example, **Linformer** [Wang et al., 2020]

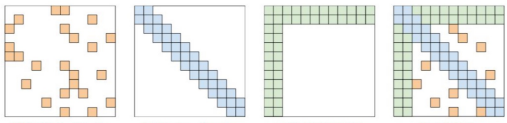
Key idea: map the sequence length dimension to a lower-dimensional space for values, keys




- For example, **BigBird** [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows**, **looking at everything**, and **random interactions**.



**Pre-Training Objective**

🔷 Pre-training:

Done unsupervised on massive text data like Wikipedia, QA forums, books.
Learns general language representations: grammar, semantics, context.

🔷 Fine-tuning:

Supervised learning on specific downstream tasks (like sentiment analysis, QA, NER).
Requires labeled data and task-specific modifications.

🚩 Problem: Fine-tuning is labor-intensive and domain-specific.

**Pre-training vs Fine-tuning Unification**

Modern goal: Unify all tasks under one pre-training objective to reduce reliance on fine-tuning.

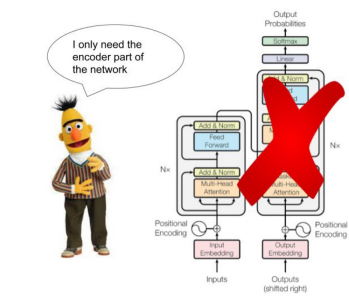
| Type | Goal | Strengths | Models |
| --- | --- | --- | --- |
| **AR (Autoregressive)** | Predict next word (left to right) | Great for text generation | GPT series |
| **AE (Autoencoding)** | Reconstruct og text from corrupted data | Great for language understanding | BERT, BART, T5 |

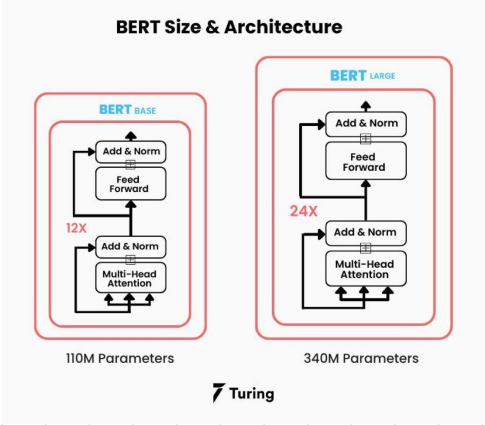
# Encoder only Transformer

## BERT (2018) - Bidirectional Encoder Representations from Transformers

Uses only the encoder stack of the transformer.
Unlike GPT (left-to-right), BERT is bidirectional — sees both left and right context.
Outperformed GPT-1 with fewer parameters.

Google
BERT




## Transformer VS BERT

Transformer: I am a ___

BERT: I am a ___. I like playing Cricket

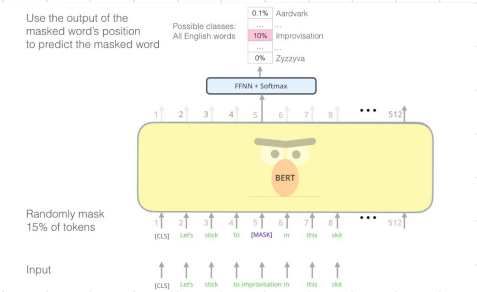
# Pre-training Objectives in BERT

**1. MASKED IM (MLM)**
The idea here is "simple":
Randomly mask out 15% of the words in the input
- replacing them with a [MASK] token
- run the entire sequence through the BERT attention based encoder and then predict only the masked words, based on the context provided by the other non-masked words in the sequence.

The model only tries to predict when the [MASK] token is present in the input, while we want the model to try to predict the correct tokens regardless of what token is present in the input.

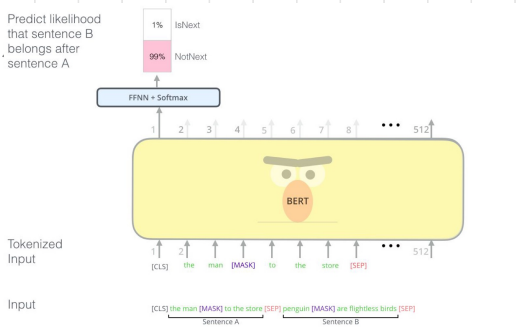
To deal with this issue, out of the 15% of the tokens selected for masking:
• 80% of the tokens are actually replaced with the token [MASK].
• 10% of the time tokens are replaced with a random token.
• 10% of the time tokens are left unchanged.

While training the BERT loss function considers only the prediction of the masked tokens and ignores the prediction of the non-masked ones.
This results in a model that converges much more slowly than left-to-right or right-to-left models.




**2. NEXT SENTENCE PREDICTION (NSP)**
In order to understand relationship between two sentences, BERT training process also uses next sentence prediction.
During training the model gets as input pairs of sentences and it learns to predict if the second sentence is the next sentence in the original text as well.
• 50% of the time the second sentence comes after the first one.
• 50% of the time it is a random sentence from the full corpus.

Given sentence pairs:
50%: sentence B is next after A
50%: sentence B is a random sentence
Model learns sentence relationships.




## 2. Next Sentence Prediction (NSP)

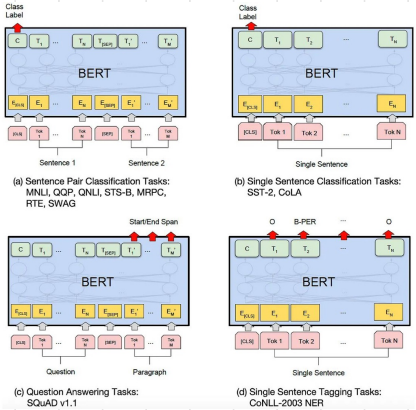
```
Input = [CLS] the man went to [MASK] store [SEP]
        he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
        penguin [MASK] are flight ##less birds [SEP]
Label = NotNext
```

# Fine Tuning

Real-life language tasks are not denoising tasks, leading to a discrepancy between pre-training and fine-tuning for BERT models. Therefore, fine-tuning is necessary for individual downstream tasks.

BERT categorizes downstream tasks into four types:

- Sentence Pair Classification Tasks (e.g., semantic similarity between two sentences)
- Single Sentence Classification Tasks (e.g., sentiment analysis)
- SQuAD (Question-Answering)
- Named Entity Tagging.




## Architecture details

BERT-Base: 12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters

BERT-Large: 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters

Fun fact: BERT-Base was trained on 4 cloud TPUs for 4 days

BERT-Large was trained on 16 TPUs for 4 days!

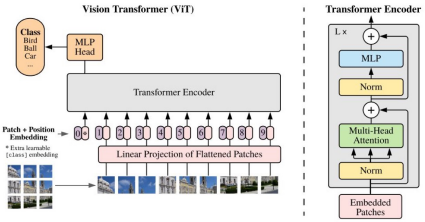



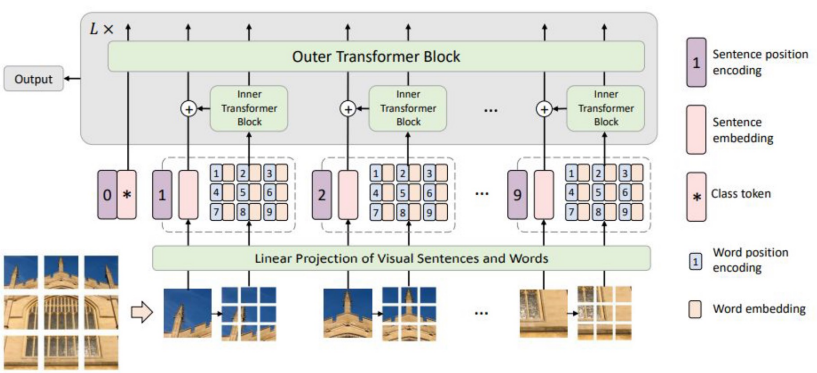

Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

## Transformer in Transformer

**Kai Han**[1,2] **An Xiao**[2] **Enhua Wu**[1,3*] **Jianyuan Guo**[2] **Chunjing Xu**[2] **Yunhe Wang**[2*]
[1]State Key Lab of Computer Science, ISCAS & UCAS
[2]Noah's Ark Lab, Huawei Technologies
[3]University of Macau
{hankai,weh}@ios.ac.cn, yunhe.wang@huawei.com

### Abstract

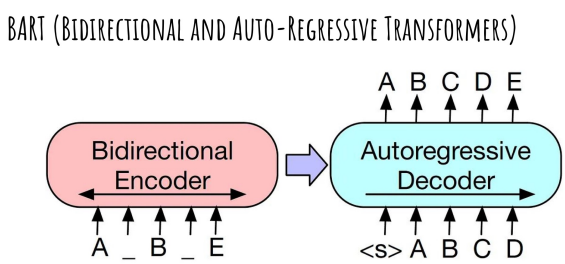
Transformer is a new kind of neural architecture which encodes the input data as powerful features via the attention mechanism. Basically, the visual transformers first divide the input images into several local patches and then calculate both representations and their relationship. Since natural images are of high complexity with abundant detail and color information, the granularity of the patch dividing is not fine enough for excavating features of objects in different scales and locations. In this paper, we point out that the attention inside these local patches are also essential for building visual transformers with high performance and we explore a new architecture, namely, Transformer iN Transformer (TNT). Specifically, we regard the local patches (*e.g.*, 16×16) as "visual sentences" and present to further divide them into smaller patches (*e.g.*, 4×4) as "visual words". The attention of



# Encoder Decoder Transformers

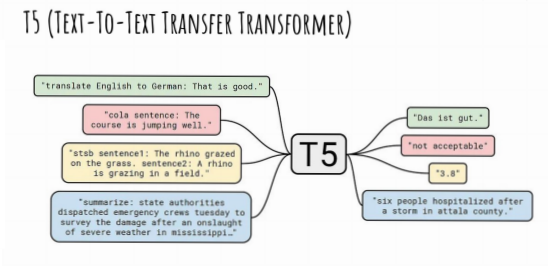
**BART (2019) – Facebook**
Combines:
- BERT-like encoder (bidirectional)
- GPT-like decoder (autoregressive)

Uses denoising objectives:
Masking 30% tokens
Permuting sentence orders




BART (2019) proposed by Facebook greatly resembles T5 in terms of the denoising pre-training objective and the encoder-decoder architecture, with the only difference being that 30% of the tokens are masked and sentence permutation is used in the pre-training text.

**T5 (2020) – Text-to-Text Transfer Transformer (Google)**
Unifies all NLP tasks into a text-in, text-out format.
15% tokens masked using sentinel tokens (special tokens that represent blanks).
Learns both understanding + generation.




In 2020, Google proposed T5 as a unified model capable of transforming all downstream tasks into text generative tasks, even classification problems.

T5 uses an encoder-decoder architecture and a denoising objective

During pre-training, 15% of the tokens fed into the encoder are randomly masked.

It is a modified version of BERT's masking and denoising algorithm: consecutive masked tokens are replaced by, a sentinel and are treated as a new single token added to the original vocabulary.

This helps the model learn to predict how many words are missing in the blank.

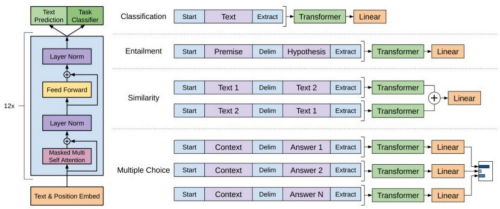
Later, generative capability is learned by randomly splitting the input text in the dataset, with the first part fed into the encoder and the second part treated as the output to be auto-regressively regenerated.

# Decoder Only Transformers

Used for generative tasks, especially autoregressive text generation.

## Generative Pretrained Transformer (GPT) [Radford et al., 2018]

How do we format inputs to our decoder for **finetuning tasks?**



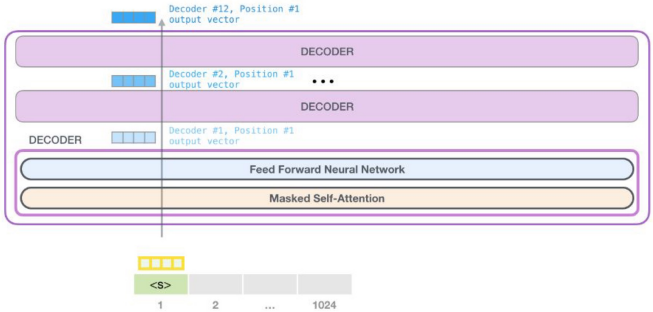

The linear classifier is applied to the representation of the [EXTRACT] token.

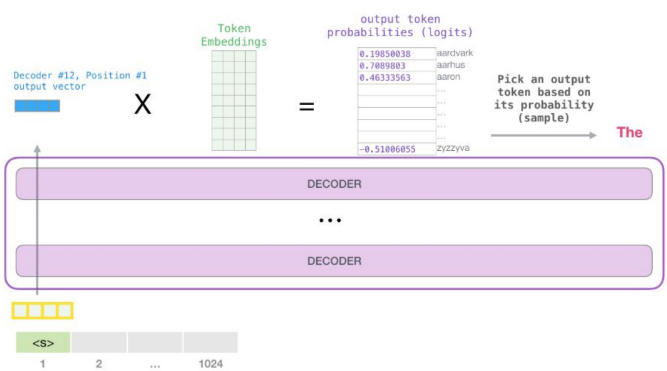
**Improving Language Understanding by Generative Pre-Training**


**Alec Radford** OpenAI alec@openai.com
**Karthik Narasimhan** OpenAI karthikn@openai.com
**Tim Salimans** OpenAI tim@openai.com
**Ilya Sutskever** OpenAI ilyasu@openai.com


**Abstract**


Natural language understanding comprises a wide range of diverse tasks such as textual entailment, question answering, semantic similarity assessment, and document classification. Although large unlabeled text corpora are abundant, labeled data for learning these specific tasks is scarce, making it challenging for discriminatively trained models to perform adequately. We demonstrate that large gains on these tasks can be realized by *generative pre-training* of a language model on a diverse corpus of unlabeled text, followed by *discriminative fine-tuning* on each specific task. In contrast to previous approaches, we make use of task-aware input transformations during fine-tuning to achieve effective transfer while requiring minimal changes to the model architecture. We demonstrate the effectiveness of

# GPT - 1

OpenAI proposed the GPT-1-4 models that only used decoder stacks, making them left-to-right autoregressive models.

GPT-1 (2018, 117 million parameters) did not exhibit emergent capabilities and heavily relied on fine-tuning for individual downstream tasks.

# GPT - 2

GPT-2 (2019, 1.5 billion parameters) introduced the phenomenon of in-context learning for a few tasks, and improved its tokenizer by using Byte-level Encoding (BLE.

The GPT-2 was trained on a massive 40GB dataset called WebText that the OpenAI researchers crawled from the internet as part of the research effort.

The smallest variant of the trained GPT-2, takes up 500MBs of storage to store all of its parameters.

The largest GPT-2 variant is 13 times the size so it could take up more than 6.5 GBs of storage space.

# GPT - 3

GPT-3 (2020, 175 billion parameters) has surprisingly demonstrated strong in-context learning capabilities, including zero-shot and few-shot learning abilities.

In few-shot or zero-shot settings without further fine-tuning, GPT-3 can achieve comparable performance with other fine-tuned state-of-the-art (SOTA) models.

# Zero-Shot, One-Shot, and Few-Shot Learning

These describe how much help (examples or guidance) we give to a model before it has to perform a task

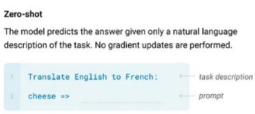
## Zero-Shot

"Hey model, just do it—no examples."

The model performs a task without seeing any task-specific examples. It relies purely on its pretrained knowledge.

Is the following sentence +ve or -ve?

"I absolutely love this movie!"




## One-Shot

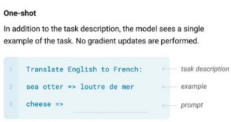
"Here's one example — now you try."

The model is shown one example of the task before doing it.

Classify the sentiment:

Example: "This is terrible." → Negative

Now classify: "This is fantastic."




## Few-Shot

"Here's a few examples — learn the pattern."

The model is given a few examples (2 to ~10) before the real task.

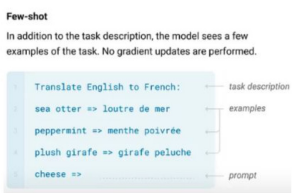
Classify the sentiment:

"Awful experience." → Negative

"I loved it!" → Positive

"Not great, not terrible." → Neutral

Now classify: "Best day ever!"




| Type | Description |
| --- | --- |
| Zero-shot | No examples given, model uses pretraining alone |
| One-shot | One example shown before task |
| Few-shot | 2–10 examples shown before task |

# ChatGPT

It is a variant of the popular GPT-3 (Generative Pertained Transformer 3) model.

Chat GPT was modified and improved using both supervised and reinforcement learning methods, with the assistance of human trainer (RLHF).Chat GPT also has 176 billion parameters same as GPT -3 model. The learning includes 3 Steps.

- Supervised fine tuning of GPT 3.5 Model
- Reward Model
- Proximal Policy Optimization (PPO)

ChatGPT is a pre-trained GPT-3 model that has been fine-tuned using the InstructGPT method.

InstructGPT method was used for fine-tuning, which combines supervised learning of demonstration texts from labelers, then with reinforcement learning of generation text scoring and ranking, which are referred to as Reinforcement Learning from Human Feedback (RLHF).

## Supervised fine tuning (Step1)

In first Step a pretrained GPT-3 model is used and it will be fine tuned with the help of labelers by creating a supervised dataset. Input Queries were collected from the actual user entries and model generated different responses with respect to that input prompts. The labelers then wrote an appropriate response to the input prompt's (how they want to see that prompt to be answered).The GPT-3 model was then fine-tuned using this new supervised dataset, to create GPT-3.5 model.

## Reward Model (Step 2)

In this step SFT model is used and different input/prompts queries fed to the finetuned model and different responses were generated (4 to 7)for every input/prompt. Then labeler determines a reward for each of these outcomes and this reward is proportional to the quality of response with respect to initial prompt.


## Proximal Policy Optimization (PPO) RL algo- Step3

In this step we pass unseen input sequences to the clone SFT model we got in step1. The model will generate response with respect to the input prompt. We pass the response to our reward model which we got in step 2 to understand, how high quality was this response for that input prompt and the output reward will be used to finetune the parameters of our SFT model .This is how our SFT model will incorporate more human like characteristics and behavior's via Reinforcement Learning.

**ChatGPT (based on GPT-3.5)**

✅ Fine-tuning Process (RLHF):

**Step 1: Supervised Fine-Tuning (SFT)**
Human labelers write ideal responses to prompts.
GPT-3 is fine-tuned on this supervised dataset → GPT-3.5.

**Step 2: Reward Model**
Several outputs are generated for each prompt.
Human labelers rank the responses.
Model learns to associate higher rewards with better answers

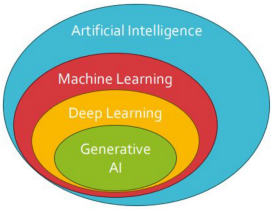
**Step 3: Proximal Policy Optimization (PPO)**
Reinforcement Learning step.
Uses reward signal to fine-tune GPT-3.5 further.
Introduces human-like behaviors and preferences.

| **Model** | **Params** | **Key Features** |
| --- | --- | --- |
| GPT-1 | 117M | Required fine-tuning |
| GPT-2 | 1.5B | Introduced **in-context learning** |
| GPT-3 | 175B | Powerful **zero/few-shot learning** |
| ChatGPT | 175B | Fine-tuned GPT-3 using **RLHF** |

| Model | Architecture | Type | Objective | Good For |
| --- | --- | --- | --- | --- |
| BERT | Encoder-only | AE | MLM + NSP | Understanding |
| GPT | Decoder-only | AR | Predict next token | Text generation |
| BART | Encoder-Decoder | AE+AR | Denoising | Summarization, Translation |
| T5 | Encoder-Decoder | AE+AR | Text-to-text | Universal NLP |
| ChatGPT | Decoder-only | AR + RLHF | Instruction-following | Conversations, reasoning tasks |

# Generative AI






## What is Generative AI (GenAI)?

GenAI creates new content—text, images, code, music—based on patterns learned from data. Unlike traditional AI, it generates something new instead of analyzing existing data.

## How Does GenAI Work?

GenAI models learn from vast datasets and generate new outputs using:

- **Neural Networks** — Deep learning models for content creation.
- **Pattern Recognition** — Predicts and generates sequences.
- **Pre-trained Models** — Fine-tuned for specific tasks.

## Applications of GenAI

- **Marketing** — AI-generated ads & content.
- **Gaming** — AI-generated characters & environments.
- **Healthcare** — AI-powered drug discovery.
- **Education** — AI tutors & content summarization.

## Challenges & Ethical Concerns

- **Bias & Misinformation** — Risk of false data.
- **Deepfakes & Manipulation** — AI-created fake media.
- **Copyright Issues** — Ownership of AI content.
- **Privacy Risks** — AI models trained on personal data.

## Types of Generative AI Models

### Transformer Models (Text & Code)

Used for chatbots, writing, coding assistants.
📌 Examples: GPT-4, Gemini, Llama, Claude

### Diffusion Models (Images & Videos)

Turn random noise into realistic visuals.
📌 Examples: Stable Diffusion, DALL·E, Midjourney

### GANs (Realistic Media)

Generative Adversarial Network

Two competing AIs create lifelike images & videos.

📌 Examples: DeepFaceLab, This Person Does Not Exist

### VAEs (Data Compression & Enhancement)

Variational Autoencoders

Used in creative design and data reconstruction.

📌 Examples: AI-assisted image variations

### RNN & LSTM (Sequential Data)

Recurrent Neural Network — Long short-term memory

Used for music, speech, and handwriting generation.

📌 Examples: AI-generated poetry, song compositions

# Generative Adversarial networks (GAN)

## GAN

GANs are generative models: they create new data instances that resemble your training data

For example, GANs can create images that look like photographs of human faces, even though the faces don't belong to any real person

Images generated by a GAN created by NVIDIA

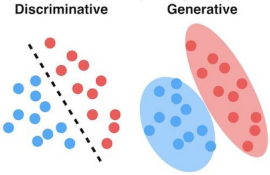
## GAN

GANs achieve this level of realism by pairing a generator and discriminator

Generator learns to produce the target output, with a discriminator, which learns to distinguish true data from the output of the generator

The generator tries to fool the discriminator, and the discriminator tries to keep from being fooled


## Background

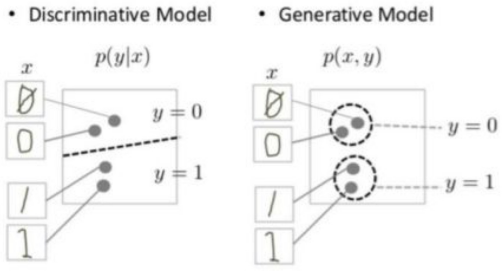



Generative models can generate new data instances

Discriminative models discriminate between different kinds of data instances

More formally, given a set of data instances X and a set of labels Y:

- Generative models capture the joint probability p(X, Y), or just p(X) if there are no labels
- Discriminative models capture the conditional probability p(Y| X)



# Overview of GAN Structure

A generative adversarial network (GAN) has two parts:

The generator learns to generate plausible data. The generated instances become negative training examples for the discriminator

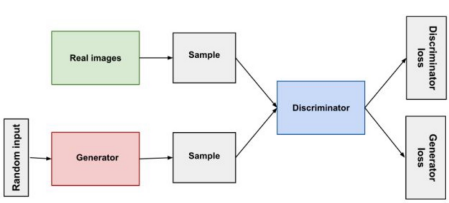
The discriminator learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results

When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake:


As training progresses, the generator gets closer to producing output that can fool the discriminator:


Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases


# GAN Block Diagram




Both the generator and the discriminator are neural networks. The generator output G(z) is connected directly to the discriminator input D(G(z))

Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights

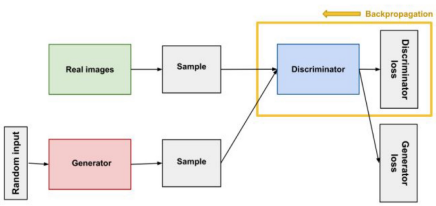
# Discriminator

The discriminator in a GAN is simply a classifier

It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying

The discriminator's training data comes from two sources:

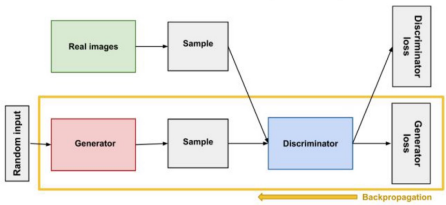
- Real data instances,x, such as real pictures of people. The discriminator uses these instances as positive examples during training
- Fake data instances created by the generator, G(z). The discriminator uses these instances as negative examples during training



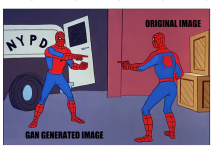

# Generator

The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator

It learns to make the discriminator classify its output as real




# Generator Training




- Sample random noise
- Produce generator output from sampled random noise
- Get discriminator "Real" or "Fake" classification for generator output
- Calculate loss from discriminator classification
- Backpropagate through both the discriminator and generator to obtain gradients
- Use gradients to change only the generator weights

# Discriminator Training

The discriminator connects to two loss functions

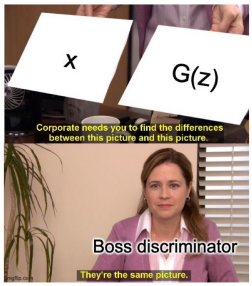
During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss.

During discriminator training:

- The discriminator classifies both real data and fake data from the generator
- The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real
- The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network

# GAN Training

1. The discriminator trains for one or more epochs
2. The generator trains for one or more epochs
3. Repeat steps 1 and 2 to continue to train the generator and discriminator networks
4. Careful with Convergence as not to overtrain the generator otherwise generator will start training with less meaningful discriminator




# Loss Function

Original GAN: minimax Loss

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

- D(x) is the discriminator's estimate of the probability that real data instance x is real
- Ex is the expected value over all real data instances
- G(z) is the generator's output when given noise z
- D(G(z)) is the discriminator's estimate of the probability that a fake instance is real
- Ez is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances G(z))

The formula derives from the cross-entropy between the real and generated distributions

The generator can't directly affect the log(D(x)) term in the function, so, for the generator, minimizing the loss is equivalent to minimizing log(1 - D(G(z)))

| Component | Original GAN | WGAN |
|---|---|---|
| Discriminator | Classifies real (1) vs fake (0) | Called a **Critic** — outputs **a score** (no sigmoid) |
| Output Range | [0, 1] (probability) | $\mathbb{R}$ (real numbers; no restriction) |
| Loss Type | Binary Cross-Entropy | Wasserstein (EM distance approximation) |
| Stability | Can be unstable (saturates, vanishes) | More stable gradients |

# Modified MiniMax Loss

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

The original GAN paper notes that the above minimax loss function can cause the GAN to get stuck in the early stages of GAN training when the discriminator's job is very easy

The paper therefore suggests modifying the generator loss so that the generator tries to maximize log D(G(z))

# Wasserstein Loss

Here no classification for discriminator.

Discriminator becomes “Critic”: Critic Loss: D(x) - D(G(z))

The discriminator tries to maximize this function

In other words, it tries to maximize the difference between its output on real instances and its output on fake instances

Generator Loss: D(G(z))

The generator tries to maximize this function. In other words, It tries to maximize the discriminator's output for its fake instances

# Advantages of Wasserstein Loss

Wasserstein GANs are less vulnerable to getting stuck than minimax based GANs and avoid problems with vanishing gradients

The earth mover distance also has the advantage of being a true metric: a measure of distance in a space of probability distributions

Cross-entropy is not a metric in this sense.

# Common problems in GAN

Problem #1: Vanishing Gradient: Research has suggested that if your discriminator is too good, then generator training can fail due to vanishing gradients

In effect, an optimal discriminator doesn't provide enough information for the generator to make progress

Sol: Wasserstein loss and Modified minimax loss

Problem #2: Mode Collapse: When the generators rotate through a small set of output types, this form of GAN failure is called Mode Collapse

In effect, an optimal discriminator doesn't provide enough information for the generator to make progress

Sol: Wasserstein loss and Modified minimax loss

Problem #3: Failure To Converge: Often problem in GAN

Sol:

(i) Adding noise to discriminator inputs

(ii) Penalizing discriminator weights

# Auto Encoders

## Unsupervised Learning

“We expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object.” - LeCun, Bengio, Hinton, Nature 2015

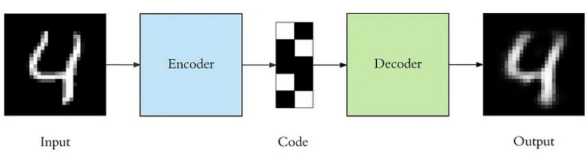
If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake. - Yann LeCun, March 14, 2016 (Facebook)

## AutoEncoders

Autoencoders are a specific type of feedforward neural networks where the input is the same as the output.

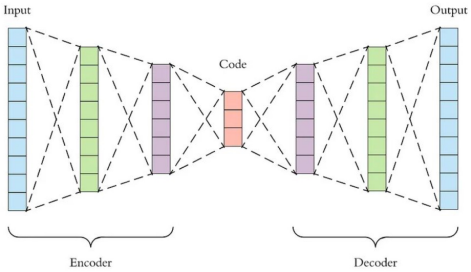
An autoencoder consists of 3 components: encoder, code and decoder.

Autoencoders are mainly a dimensionality reduction (or compression) algorithm.




## Unsupervised????

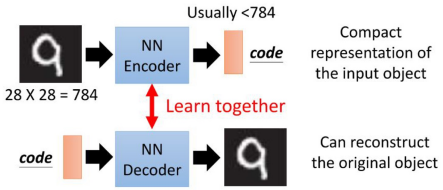
Autoencoders are considered an unsupervised learning technique since they don’t need explicit labels to train on. But to be more precise they are self-supervised because they generate their own labels from the training data.




There are 4 hyperparameters that we need to set before training an autoencoder:

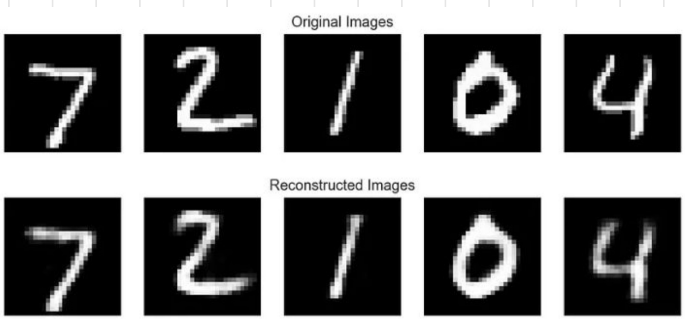
1. Code size
2. Number of layers
3. Number of nodes per layer
4. Loss function

# AutoEncoders




# Implementation

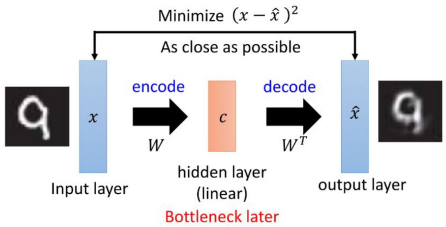
```
input_size = 784
hidden_size = 128
code_size = 32

input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu')(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)

autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train, epochs=5)
```



## A BIT OF ADVICE

We should be careful to not make it too powerful. Otherwise the autoencoder will simply learn to copy its inputs to the output, without learning any meaningful representation. It will just mimic the identity function. The autoencoder will reconstruct the training data perfectly, but it will be overfitting without being able to generalize to new instances, which is not what we want.

## Recap: PCA

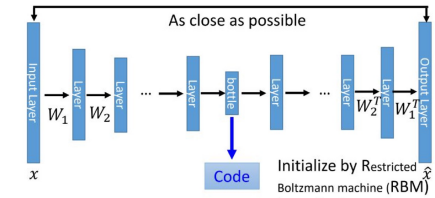



Output of the hidden layer is the code

## Deep Auto-encoder

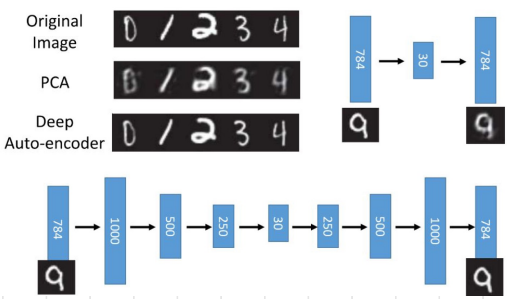
Symmetric is not necessary.

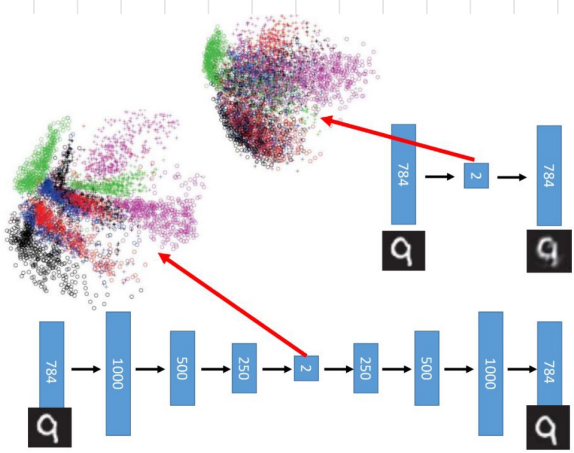
- Of course, the auto-encoder can be deep




Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

## Deep Auto-encoder



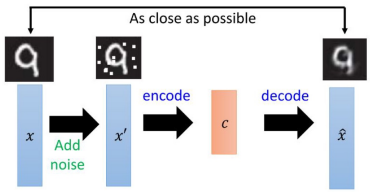



# Auto-encoder

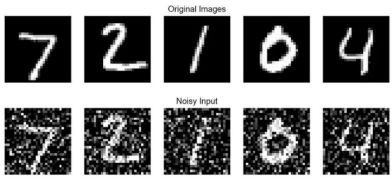
More: Contractive auto-encoder

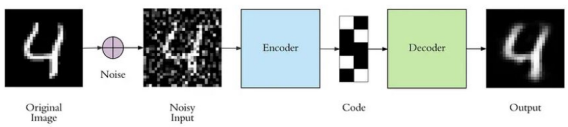
Ref: Rifai, Salah, et al. "Contractive auto-encoders: Explicit invariance during feature extraction." *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.

- De-noising auto-encoder




Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *ICML*, 2008.

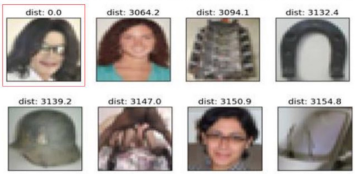



```
Denoising autoencoder is trained as:
autoencoder.fit(x_train_noisy, x_train)
```

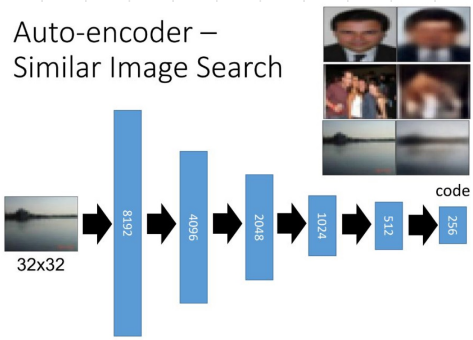
# Auto-encoder – Similar Image Search

Retrieved using Euclidean distance in pixel intensity space



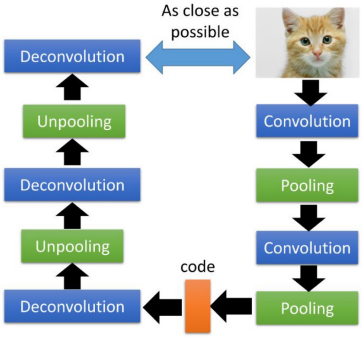

retrieved using 256 codes

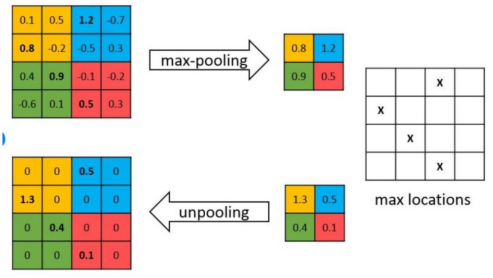

# Auto-encoder – Similar Image Search




(crawl millions of images from the Internet)

Auto-encoder for CNN

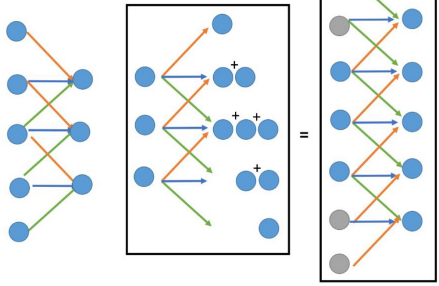



Pooling and unpooling layers. For each pooling layer, the max locations are stored. These locations are then used in unpooling layer.

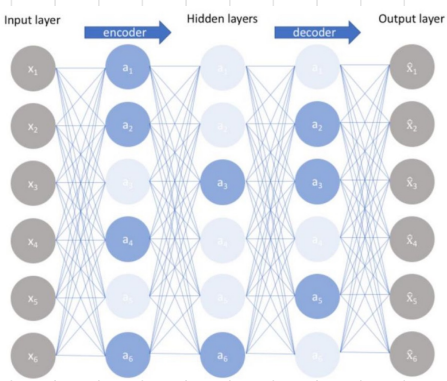
CNN
- Deconvolution

Actually, deconvolution is convolution.



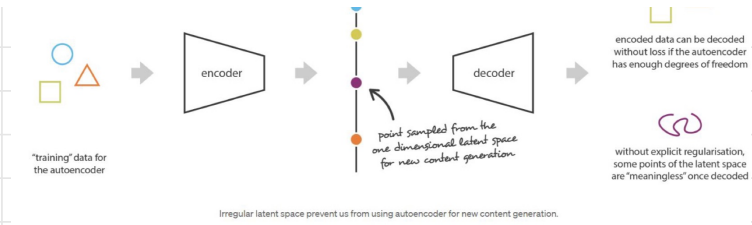
# Sparse AutoEncoder

- Use of Regularization (For better generalization)
- Introducing sparsity constraint such that only a fraction of the nodes would have nonzero values called active nodes
- Add a penalty term to the loss function such that only a fraction of nodes become active
- This forces the autoencoder to represent each input as a combination of small number of nodes and demands to discover interesting structure in the data

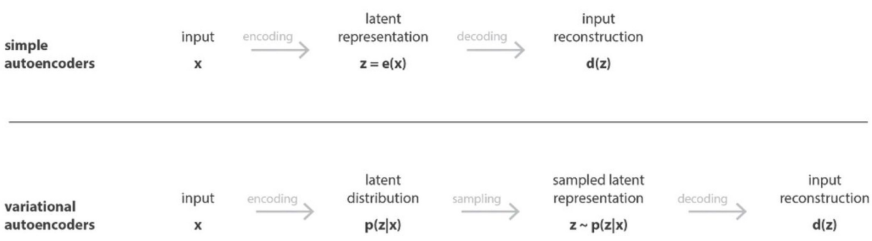


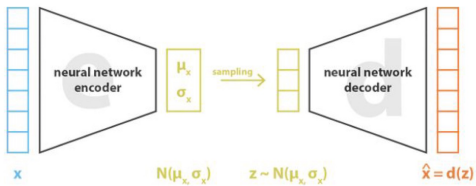
# VARIATIONAL AUTOENCODER (VAE)

A variational autoencoder can be defined as being an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process.

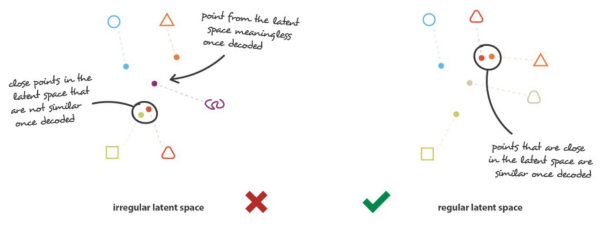


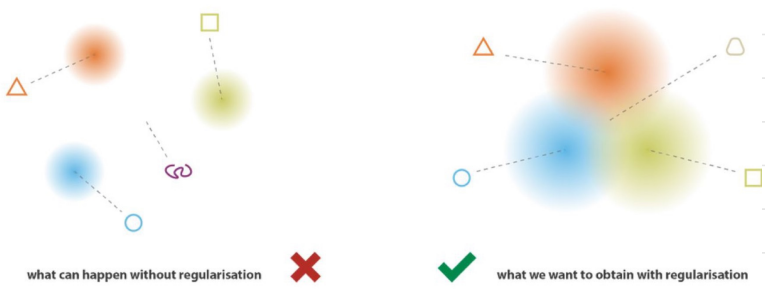

Irregular latent space prevent us from using autoencoder for new content generation.






$$\text{loss} = ||x - \hat{x}||^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$



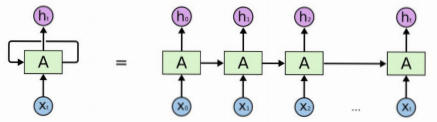

The returned distributions of VAEs have to be regularised to obtain a latent space with good properties.

# RNN NUMERICAL

Sentence:
"I like to eat pizza."

Vocabulary: POS Tags
"I" - Pronoun
"like" - Verb
"to" - Preposition
"eat" - Verb
"pizza" - Noun

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$




8) **W_xh = [[0.4, -0.3, 0.1, -0.2, 0.5],**
**[0.1, -0.2, 0.3, 0.2, -0.4],**
**[0.2, -0.1, 0.5, 0.4, -0.3]]**

**W_hy = [[0.2, 0.6, -0.1],**
**[0.3, -0.2, 0.4],**
**[-0.4, 0.1, 0.5],**
**[0.1, 0.2, 0.3]]**

**W_hh = [[0.2, -0.1, 0.3],**
**[-0.1, 0.4, -0.2],**
**[0.4, -0.3, 0.5]]**
**h_0 = [0.1, -0.1, 0.2]**

Word "I":
x_t = [1, 0, 0, 0, 0] # One-hot encoding for "I" in the vocabulary
h_t = tanh(W_xh * x_t + W_hh * h_(t-1))
h_t = tanh([[0.4, -0.3, 0.1, -0.2, 0.5],
[0.1, -0.2, 0.3, 0.2, -0.4],
[0.2, -0.1, 0.5, 0.4, -0.3]] * [1, 0, 0,0,0]^T //3x5 *5x1

\+ [[0.2, -0.1, 0.3],
[-0.1, 0.4, -0.2],
[0.4, -0.3, 0.5]] * [0.1, -0.1, 0.2]^T) //3x3 *3*1

h_t =tanh([0.49, 0.01, 0.37])
h_t=[0.4621, 0.0099, 0.3584]

*Tanh*

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

pos_scores = W_hy * h_t
pos_scores =[[0.2, 0.6, -0.1],
[0.3, -0.2, 0.4],
[-0.4, 0.1, 0.5],
[0.1, 0.2, 0.3]] * [0.4621, 0.0099, 0.3584]

pos_scores = [0.06252, 0.28001, 0.19535, 0.15571]

After softmax [0.2229, 0.2777, 0.2547, 0.2446]
Pronoun verb preposition noun

Cross-Entropy Loss (L) = -Σ(T_i * log(P_i))
L = -(1 * log(0.2229) + 0 * log(0.2777) + 0 * log(0.2547) + 0 * log(0.2446))

L = -(log(0.2229))

L ≈ 1.5020

# RNN NUMERICAL

Sentence:
"I like to eat pizza."

Vocabulary: POS Tags
"I" - Pronoun
"like" - Verb
"to" - Preposition
"eat" - Verb
"pizza" - Noun

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = \tanh(a^{(t)})$$
$$o^{(t)} = c + Vh^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}(o^{(t)})$$

(hidden bias; hidden to hidden weights; input to hidden weights; hidden to output weights; output bias)

$U_h$ ← W_xh = [[0.4, -0.3, 0.1, -0.2, 0.5],
[0.1, -0.2, 0.3, 0.2, -0.4],
[0.2, -0.1, 0.5, 0.4, -0.3]]

$V_h$ ← W_hy = [[0.2, 0.6, -0.1],
[0.3, -0.2, 0.4],
[-0.4, 0.1, 0.5],
[0.1, 0.2, 0.3]]

$W_h$ ← W_hh = [[0.2, -0.1, 0.3],
[-0.1, 0.4, -0.2],
[0.4, -0.3, 0.5]]

$h^0$ ← h_0 = [0.1, -0.1, 0.2] (initial hidden state)

1. One hot vector encode

| | I | like | to | eat | Pizza |
|---|---|---|---|---|---|
| I: | 1 | 0 | 0 | 0 | 0 |
| like: | 0 | 1 | 0 | 0 | 0 |
| to: | 0 | 0 | 1 | 0 | 0 |
| eat: | 0 | 0 | 0 | 1 | 0 |
| Pizza: | 0 | 0 | 0 | 0 | 1 |

## Steps

1. One hot encode
2. Forward Pass of $X_1$
   - ↳ $a^1 = W_{hh} \cdot h^0 + W_{xh} \cdot x^1$
   - ↳ $h^1 = \tanh(a^1)$
   - ↳ $o^1 = W_{hy} \cdot h^1$
   - ↳ $y^1 = \text{softmax}(o^1) \cdot \frac{e^{o_i}}{\sum_j e^{o_j}}$
   - ↳ CEL $= -\sum(x^1 \cdot \log(y^1))$

2. forward pass for each word

$x^1$ → For I I: $[1\ 0\ 0\ 0\ 0]^T$ , $h_0 = [0.1, -0.1, 0.2]^T$

→ $a^1 = W_{hh} \cdot h^0 + W_{xh} \cdot x^1$

$$\begin{bmatrix} 0.2 & -0.1 & 0.3 \\ -0.1 & 0.4 & -0.2 \\ 0.4 & -0.3 & 0.5 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.1 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.1 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.49 \\ 0.01 \\ 0.37 \end{bmatrix}$$

(as I: $[1\ 0\ 0\ 0\ 0]$; select first column)

→ $h^1 = \tanh(a^1)$

$$h^1 = \begin{bmatrix} \tanh(0.49) \\ \tanh(0.01) \\ \tanh(0.37) \end{bmatrix} = \begin{bmatrix} 0.454 \\ 0.01 \\ 0.354 \end{bmatrix}$$

POS scores → $o^1 = W_{hy} \cdot h^1$

$$\begin{bmatrix} 0.2 & 0.6 & -0.1 \\ 0.3 & -0.2 & 0.4 \\ -0.4 & 0.1 & 0.5 \\ 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 0.454 \\ 0.01 \\ 0.354 \end{bmatrix} = \begin{bmatrix} 0.0614 \\ 0.2758 \\ -0.0036 \\ 0.1536 \end{bmatrix}$$

→ $y^1 = e^{o_j} / \sum_j e^{o_j}$ (softmax)

$e^{0.0614} = 1.063 = 1.063/4.542 = 0.234$

$e^{0.2758} = 1.317 = 1.317/4.542 = 0.29$

$e^{0.0036} = 0.996 = \ldots = 0.219$

$e^{0.1536} = 1.166 = \ldots = 0.257$

$\frac{}{4.542}$

↳ Cross-Entropy Loss: $-\sum(T_i \cdot \log(P_i))$

$= -(1 \cdot \log(0.234) + 0 \cdot \log(0.29) + 0 \cdot \log(0.219) + 0 \cdot \log(0.257))$

$= 1.5020$

# Transformer Numerical

Input sentence: "Transformers transforming our lives"
Output sentence: "For sure"

The embedding matrix for the words is represented as follows:

"Transformers": [0.1, 0.2, 0.3]
"transforming": [0.4, 0.5, 0.6]
"our": [0.7, 0.8, 0.9]
"lives": [1.0, 1.1, 1.2]
"For": [0.4, 0.1, 0.8]
"sure": [0.9, 0.7, 0.2]

Assuming the weight matrices:

$$W_q = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 2 \\ 7 & 1 & 9 \end{bmatrix} \quad W_k = \begin{bmatrix} 1 & 6 & 9 \\ 7 & 3 & 1 \\ 9 & 2 & 1 \end{bmatrix} \quad W_v = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 0 & 2 \\ 1 & 6 & 8 \end{bmatrix}$$

For simplicity, assume dk is 1.

1. Calculate masked self-attention for the above input sentence using the provided weight matrices. Show all steps clearly.
2. Calculate cross-attention for the above and output sentence using the provided weight matrices. Show all steps clearly.

$$X_1 = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 1.0 & 1.1 & 1.2 \end{bmatrix}$$ Transformers / transforming / our / lives

## Masked self attention

### 1. COMPUTE Q, K, V

→ $Q = X_1 W^Q$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 1.0 & 1.1 & 1.2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 2 \\ 7 & 1 & 9 \end{bmatrix} = \begin{bmatrix} 3 & 1.5 & 3.4 \\ 6.6 & 3.9 & 7.6 \\ 10.2 & 6.3 & 11.8 \\ 13.8 & 8.7 & 16 \end{bmatrix}$$

→ $K = X_1 W_K$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 1.0 & 1.1 & 1.2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 6 & 9 \\ 7 & 3 & 1 \\ 9 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 4.2 & 1.8 & 1.4 \\ 9.3 & 5.1 & 4.7 \\ 14.4 & 8.4 & 8 \\ 19.5 & 11.7 & 11.3 \end{bmatrix}$$

→ $V = X_1 W_V$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 1.0 & 1.1 & 1.2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 & 6 \\ 8 & 0 & 2 \\ 1 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 2.1 & 2.2 & 3.4 \\ 5.4 & 5.2 & 8.2 \\ 8.7 & 8.2 & 13 \\ 12 & 11.2 & 17.8 \end{bmatrix}$$

### 2. Compute attention score

$E = QK^T/\sqrt{dk}$ (it becomes 1, so ignore for now)

$$\begin{bmatrix} 3 & 1.5 & 3.4 \\ 6.6 & 3.9 & 7.6 \\ 10.2 & 6.3 & 11.8 \\ 13.8 & 8.7 & 16 \end{bmatrix} \cdot \begin{bmatrix} 4.2 & 9.3 & 14.4 & 19.5 \\ 1.8 & 5.1 & 8.4 & 11.7 \\ 1.4 & 4.7 & 8 & 11.3 \end{bmatrix} = \begin{bmatrix} 20.06 & 51.53 & 83 & 114.47 \\ 45.38 & 116.99 & 188.6 & 260.21 \\ 70.7 & 182.45 & 294.2 & 405.95 \\ 96.02 & 247.91 & 399.8 & 551.69 \end{bmatrix}$$

After applying masking

$$E = \begin{bmatrix} 20.06 & -\infty & -\infty & -\infty \\ 45.38 & 116.99 & -\infty & -\infty \\ 70.7 & 182.45 & 294.2 & -\infty \\ 96.02 & 247.91 & 399.8 & 551.69 \end{bmatrix}$$

### 3. Softmax E

Row wise softmax

$$\begin{bmatrix} 20.06 & -\infty & -\infty & -\infty \\ 45.38 & 116.99 & -\infty & -\infty \\ 70.7 & 182.45 & 294.2 & -\infty \\ 96.02 & 247.91 & 399.8 & 551.69 \end{bmatrix} = \begin{bmatrix} \frac{e^{20.06}}{e^{20.06}} & 0 & 0 & 0 \\ \frac{e^{45.38}}{e^{45.38}+e^{116.99}} & \frac{e^{116.99}}{e^{45.38}+e^{116.99}} & 0 & 0 \\ \frac{e^{70.7}}{e^{70.7}+e^{182.45}+e^{294.2}} & \frac{e^{182.45}}{e^{70.7}+e^{182.45}+e^{294.2}} & \frac{e^{294.2}}{e^{70.7}+e^{182.45}+e^{294.2}} & 0 \\ \frac{e^{96.02}}{e^{96.02}+e^{247.91}+e^{399.8}+e^{551.69}} & \frac{e^{247.91}}{e^{96.02}+e^{247.91}+e^{399.8}+e^{551.69}} & \frac{e^{399.8}}{e^{96.02}+e^{247.91}+e^{399.8}+e^{551.69}} & \frac{e^{551.69}}{e^{96.02}+e^{247.91}+e^{399.8}+e^{551.69}} \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 7.9\times10^{-52} & 1 & 0 & 0 \\ 8.6\times10^{-98} & 2.9\times10^{-49} & 1 & 0 \\ 1.27\times10^{-198} & 1.17\times10^{-132} & 1.08\times10^{-66} & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 4. Output = A . V

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2.1 & 2.2 & 3.4 \\ 5.4 & 5.2 & 8.2 \\ 8.7 & 8.2 & 13 \\ 12 & 11.2 & 17.8 \end{bmatrix} = \begin{bmatrix} 2.1 & 2.2 & 3.4 \\ 5.4 & 5.2 & 8.2 \\ 8.7 & 8.2 & 13 \\ 12 & 11.2 & 17.8 \end{bmatrix}$$ Transformers / transforming / our / lives

### Steps

1. Compute Q, K, V with $X_{input}$
2. Attention Score
   - $E = QK^T/\sqrt{dk}$
   - Apply mask
3. Softmax row wise → A
4. Output = A . V

Input sentence: "Transformers transforming our lives"
Output sentence: "For sure"

The embedding matrix for the words is represented as follows:

"Transformers": [0.1, 0.2, 0.3]
"transforming": [0.4, 0.5, 0.6]
"our": [0.7, 0.8, 0.9]
"lives": [1.0, 1.1, 1.2]
"For": [0.4, 0.1, 0.8]
"sure": [0.9, 0.7, 0.2]

Assuming the weight matrices:

$$W_q = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 2 \\ 7 & 1 & 9 \end{bmatrix} \quad W_k = \begin{bmatrix} 1 & 6 & 9 \\ 7 & 3 & 1 \\ 9 & 2 & 1 \end{bmatrix} \quad W_v = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 0 & 2 \\ 1 & 6 & 8 \end{bmatrix}$$

$$X_0 = \begin{bmatrix} 0.4 & 0.1 & 0.8 \\ 0.9 & 0.7 & 0.2 \end{bmatrix} \begin{matrix} \text{For} \\ \text{Sure} \end{matrix}$$

For simplicity, assume dk is 1.

1. Calculate masked self-attention for the above input sentence using the provided weight matrices. Show all steps clearly.
2. Calculate cross-attention for the above input and output sentence using the provided weight matrices. Show all steps clearly.

## Cross-attention

### 1. Compute Q, K, V

→ $Q = X_0 W_q$ ⟶ Using Output Sequence

$$= \begin{bmatrix} 0.4 & 0.1 & 0.8 \\ 0.9 & 0.7 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 2 \\ 7 & 1 & 9 \end{bmatrix} = \begin{bmatrix} 6.4 & 2.1 & 8.6 \\ 5.1 & 5.5 & 5.9 \end{bmatrix}$$

$2 \times 3$ $\quad 3 \times 3$ $\quad 2 \times 3$

→ $K = X_1 W_k$ ⟶ Using Input Sequence

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 1.0 & 1.1 & 1.2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 6 & 9 \\ 7 & 3 & 1 \\ 9 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 4.2 & 1.8 & 1.4 \\ 9.3 & 5.1 & 4.7 \\ 14.4 & 8.4 & 8 \\ 19.5 & 11.7 & 11.3 \end{bmatrix}$$

→ $V = X_1 W_v$ ⟶ Using Input Sequence

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \\ 0.7 & 0.8 & 0.9 \\ 1.0 & 1.1 & 1.2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 & 6 \\ 8 & 0 & 2 \\ 1 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 2.1 & 2.2 & 3.4 \\ 5.4 & 5.2 & 8.2 \\ 8.7 & 8.2 & 13 \\ 12 & 11.2 & 17.8 \end{bmatrix}$$

### 2. Attention Score

$E = QK^T / \sqrt{dk}$

$$\begin{bmatrix} 6.4 & 2.1 & 8.6 \\ 5.1 & 5.5 & 5.9 \end{bmatrix} \cdot \begin{bmatrix} 4.2 & 1.8 & 1.4 \\ 9.3 & 5.1 & 4.7 \\ 14.4 & 8.4 & 8 \\ 19.5 & 11.7 & 11.3 \end{bmatrix} = \begin{bmatrix} 42.7 & 110.65 & 178.6 & 246.5 \\ 39.58 & 103.21 & 166.8 & 230.5 \end{bmatrix}$$

$2 \times 4$ $\quad 4 \times 3$ $\quad 2 \times 3$

### 3. Softmax E

Row wise Softmax

$$\begin{bmatrix} 42.7 & 110.65 & 178.6 & 246.5 \\ 39.58 & 103.21 & 166.8 & 230.5 \end{bmatrix} = \begin{bmatrix} \frac{e^{42.7}}{e^{42.7}+e^{110.65}+e^{178.6}+e^{246.5}} & \frac{e^{110.65}}{e^{42.7}+e^{110.65}+e^{178.6}+e^{246.5}} & \frac{e^{178.6}}{e^{42.7}+e^{110.65}+e^{178.6}+e^{246.5}} & \frac{e^{246.5}}{e^{42.7}+e^{110.65}+e^{178.6}+e^{246.5}} \\ \frac{e^{39.58}}{e^{39.58}+e^{103.21}+e^{166.8}+e^{230.5}} & \frac{e^{103.21}}{e^{39.58}+e^{103.21}+e^{166.8}+e^{230.5}} & \frac{e^{166.8}}{e^{39.58}+e^{103.21}+e^{166.8}+e^{230.5}} & \frac{e^{230.5}}{e^{39.58}+e^{103.21}+e^{166.8}+e^{230.5}} \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### 4. Output = A.V

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2.1 & 2.2 & 3.4 \\ 5.4 & 5.2 & 8.2 \\ 8.7 & 8.2 & 13 \\ 12 & 11.2 & 17.8 \end{bmatrix} = \begin{bmatrix} 12 & 11.2 & 17.8 \\ 12 & 11.2 & 17.8 \end{bmatrix} \begin{matrix} \text{for} \\ \text{sure} \end{matrix}$$

### Steps

1. Compute Q, K, V with X
   - Q with $X_{output}$
   - K, V with $X_{input}$
2. Attention Score $E = QK^T / \sqrt{dk}$
3. Softmax row wise → A
4. Output = A.V

# SELF ATTENTION NUMERICAL

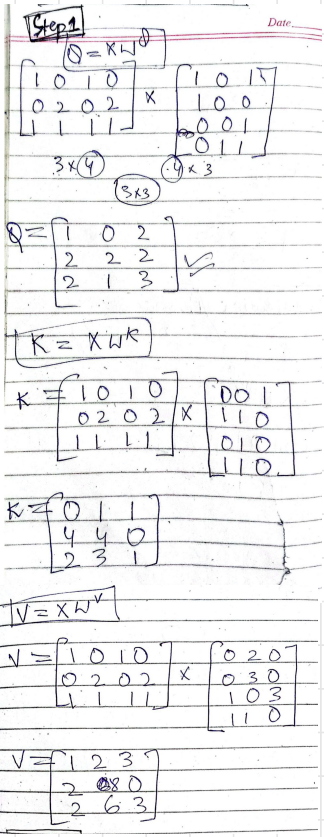
```
x = [
  [1, 0, 1, 0], # Input 1
  [0, 2, 0, 2], # Input 2
  [1, 1, 1, 1]  # Input 3
 ]
```

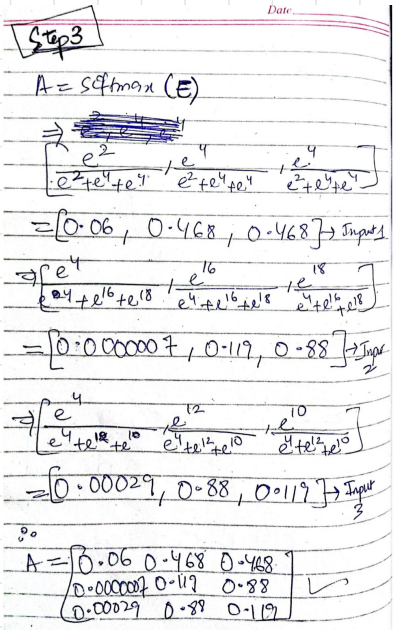
```
w_key = [
  [0, 0, 1],
  [1, 1, 0],
  [0, 1, 0],
  [1, 1, 0] ]
w_query = [
  [1, 0, 1],
  [1, 0, 0],
  [0, 0, 1],
  [0, 1, 1] ]
w_value = [
  [0, 2, 0],
  [0, 3, 0],
  [1, 0, 3],
  [1, 1, 0] ]
```

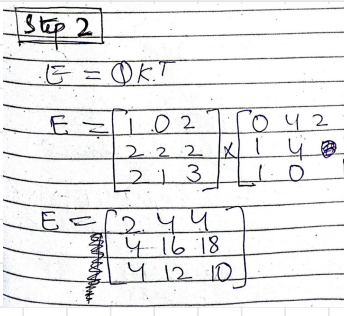
Solve the above example. For simplicity dk=1.

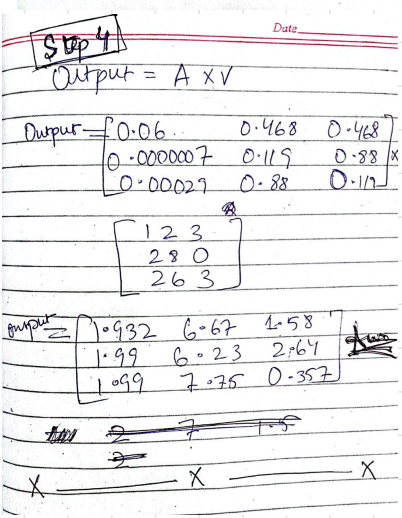
Round off answer =

```
[[2.        7.        1.5      ]
 [2.        8.        0.       ]
 [2.        7.7999997 0.3      ]]
```




## STEPS

1. Compute Q,K,V with X
2. Attention Score $E = QK^T/\sqrt{dk}$
3. Softmax row wise → A
4. Output: A.V

# Attention Numerical

Encoder | Decoder

Q) $h_1 = [1, 2, 3]$ $s_1 = [3, 4, 5]$

$h_2 = [3, 1, 1]$

$h_3 = [4, 2, 1]$

↳ Find attention scores $e^t$

$s_1^T \cdot h_1 = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} [1, 2, 3] = 3+8+15 = 26$

$s_1^T h_2 = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} [3, 1, 1] = 9+4+5 = 18$

$s_1^T h_3 = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} [4, 2, 1] = 12+8+5 = 25$

$e^t = [26, 18, 25]$

↳ Softmax $= \left[ \frac{e^{26}}{e^{26}+e^{18}+e^{25}}, \frac{e^{18}}{e^{26}+e^{18}+e^{25}}, \frac{e^{25}}{e^{26}+e^{18}+e^{25}} \right]$

$\alpha = [0.73, 0.0002, 0.2688]$

$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 1 \\ 4 & 2 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix}$

↳ Context vector

$a_t = \alpha \cdot h^T$

$= [0.73] \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + [0.0002] \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} + [0.26888] + \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = [1.8058, 1.9978, 2.459]$

## Steps

1. Attention score

$e_i^t = s_1^T \cdot h_i$ — h encoder, s decoder

2. Softmax $(e^t)$

$\alpha_i = \frac{e^t}{\sum_j e^{t_j}}$

3. Context vector

$a_t = \sum_{i=1} \alpha_i \cdot h_i$

input activation: $a_0 = \tanh(W_a \cdot x_0 + U_a \cdot out_{-1} + b_a)$ — no previous value so 0

input gate: $i_0 = \sigma(W_i \cdot x_0 + U_i \cdot out_{-1} + b_i)$

forget gate: $f_0 = \sigma(W_f \cdot x_0 + U_f \cdot out_{-1} + b_f)$

output gate: $O_0 = \sigma(W_o \cdot x_0 + U_o \cdot out_{-1} + b_o)$

internal state: $state_0 = (a_0 \times i_0) + (f_0 \times state_{-1})$

output: $out_0 = \tanh(state_0) \times O_0$

$$\delta out_t = \Delta_t + \Delta out_t$$

$$\delta state_t = \delta out_t \odot O_t \odot (1 - \tanh^2(state_t)) + \delta state_{t+1} \odot f_{t+1}$$

$$\delta a_t = \delta state_t \odot i_t \odot (1 - a_t^2)$$

$$\delta i_t = \delta state_t \odot a_t \odot i_t \odot (1 - i_t^2)$$

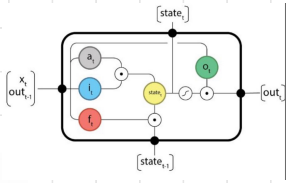
$$\delta f_t = \delta state_t \odot a_t \odot i_t \odot f_t \odot (1 - f_t^2)$$

$$\delta O_t = \delta state_t \odot \tanh(state_t) \odot O_t \odot (1 - O_t)$$

$$\delta_{n_t} = W^T \cdot \delta gates_t$$

$$\Delta out_{t-1} = U^T \cdot \delta gates_t$$

# LSTM Numerical




- Above ⊙ is the element-wise product or Hadamard product.
- Inner products will be represented as ·
- Outer products will be respresented as ⊗
- σ represents the sigmoid function

Lets consider sequence length of two to demonstrate the unrolling over time

(a) Show that for t=0; following were the values of $a_0$, $i_0$, $f_0$, $state_0$, and $out_0$

a) Internal weights

$W_a = \begin{bmatrix} 0.45 \\ 0.25 \end{bmatrix}$ $W_i = \begin{bmatrix} 0.95 \\ 0.8 \end{bmatrix}$, $W_f = \begin{bmatrix} 0.7 \\ 0.45 \end{bmatrix}$, $W_o = \begin{bmatrix} 0.6 \\ 0.4 \end{bmatrix}$

$U_a = [0.15]$ $U_i = [0.8]$, $U_f = [0.1]$, $U_o = [0.25]$

$b_a = [0.2]$ $b_i = [0.65]$, $b_f = [0.15]$, $b_o = [0.1]$

Input data

$x_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ label: 0.5

$x_1 = \begin{bmatrix} 0.5 \\ 3 \end{bmatrix}$ label: 1.25

## Forward at t=0

no previous value so 0

Input activation → $a_0 = \tanh(W_a \cdot x_0 + U_a \cdot out_{-1} + b_a)$

$= \tanh([0.45 \quad 0.25]\begin{bmatrix} 1 \\ 2 \end{bmatrix} + [0.15][0] + [0.2] = 0.81775$

Input gate → $i_0 = \sigma(W_i \cdot x_0 + U_i \cdot out_{-1} + b_i)$

$= \sigma([0.95 \quad 0.8]\begin{bmatrix} 1 \\ 2 \end{bmatrix} + [0.8][0] + [0.65] = 0.96083$ (≈3.2)

Forget gate → $f_0 = \sigma(W_f \cdot x_0 + U_f \cdot out_{-1} + b_f)$

$= \sigma([0.7 \quad 0.45]\begin{bmatrix} 1 \\ 2 \end{bmatrix} + [0.1][0] + [0.15] = 0.8519$

Output gate → $O_0 = \sigma(W_o \cdot x_0 + U_o \cdot out_{-1} + b_o)$

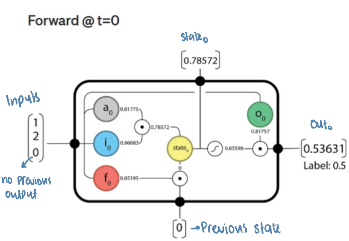
$= \sigma([0.6 \quad 0.4]\begin{bmatrix} 1 \\ 2 \end{bmatrix} + [0.25][0] + [0.1] = 0.81757$

Internal State → $state_0 = a_0 \odot i_0 + f_0 \odot state_{-1}$

$= 0.81775 \odot 0.96083 + 0.85190 \odot 0 = 0.78571$

Output → $out_0 = \tanh(state_0) \odot O_0$

$= \tanh(0.78571) \odot 0.81757 = 0.5363$




$\sigma = \frac{1}{1 + e^{-n}}$

$\tanh = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$

## Forward at t=1

Input activation → $a_1 = \tanh([0.45 \quad 0.25]\begin{bmatrix} 0.5 \\ 3 \end{bmatrix} + [0.15][0.5363] + [0.2] = 0.84980$

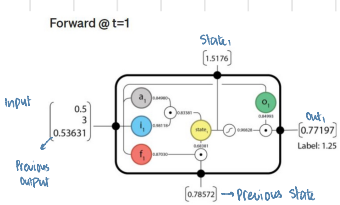
Input gate → $i_1 = \sigma([0.95 \quad 0.8]\begin{bmatrix} 0.5 \\ 3 \end{bmatrix} + [0.8][0.5363] + [0.65] = 0.98118$

Forget gate → $f_1 = \sigma([0.7 \quad 0.45]\begin{bmatrix} 0.5 \\ 3 \end{bmatrix} + [0.1][0.5363] + [0.15] = 0.87030$

Output gate → $O_1 = \sigma([0.6 \quad 0.4]\begin{bmatrix} 0.5 \\ 3 \end{bmatrix} + [0.25][0.5363] + [0.1] = 0.84993$

Internal State → $state_1 = 0.84980 \odot 0.98118 + 0.87030 \odot 0.78572 = 1.5176$

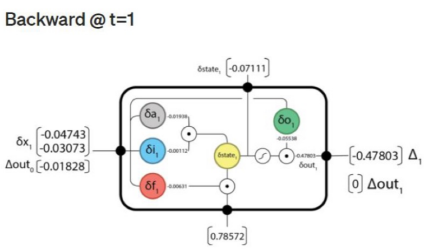
Output → $out_1 = \tanh(1.5176) \odot 0.84993 = 0.7719$




Then we will compute Backward for t = 1

(b) For Backward @t=0; show the steps to get the values of various derivatives e.g. derivate of state, $x, out_0$

Use L2 Loss




## Backward at t=1

Predicted output ↑ Actual output ↑

$\delta out_1 = \Delta_1 + \Delta out_1$

$= 0.77191 - 1.25 = -0.47803$

$\delta state_1 = \delta out_1 \odot o_1 \odot (1 - \tanh^2(state_1)) + \delta state_{t+1} \odot f_{t+1}$

$= -0.47803 \odot 0.84993 \odot (1 - \tanh^2(1.5176)) + 0 \odot 0 = -0.07111$

$\delta a_1 = \delta state_1 \odot i_1 \odot (1 - a_1^2)$

$0.07111 \odot 0.98118 \odot (1 - 0.84980^2) = -0.01938$

$\delta i_1 = \delta state_1 \odot a_1 \odot i_1 \odot (1 - i_1^2)$

$0.07111 \odot 0.84980 \odot 0.98118 \odot (1 - 0.98118^2) = -0.00112$

$\delta f_1 = \delta state_1 \odot a_1 \odot i_1 \odot f_1 \odot (1 - f_1^2)$

$0.07111 \odot 0.84980 \odot 098118 \odot 087030 \odot (1 - 0.87030^2) = -0.00630$

$\delta o_1 = \delta state_1 \odot \tanh(state_1) \odot o_1 \odot (1 - o_1)$

$0.07111 \odot \tanh(0.07111) \odot 0.84993 \odot (1 - 0.84993) = -0.05537$

$\delta x_1 = W^T \cdot \delta gates_1$

$$\begin{bmatrix} 0.45 & 0.95 & 0.7 & 0.6 \\ 0.25 & 0.8 & 0.45 & 0.4 \end{bmatrix} \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00630 \\ -0.5537 \end{bmatrix} = \begin{bmatrix} -0.04743 \\ -0.03073 \end{bmatrix}$$

$\Delta out_{t-1} = U^T \cdot \delta gates_1$

$$\begin{bmatrix} 0.15 & 0.8 & 0.1 & 0.25 \end{bmatrix} \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00630 \\ -0.5537 \end{bmatrix} = \begin{bmatrix} -0.01828 \end{bmatrix}$$

# Backward at t=0

$\delta_{out_0} = 0.03631 + (-0.01828) = 0.01803$

$\delta_{state_0} = -0.5349$

$\delta_{a_0} = -0.1702$

$\delta_{i_0} = -0.00165$

$\delta_{f_0} = 0$

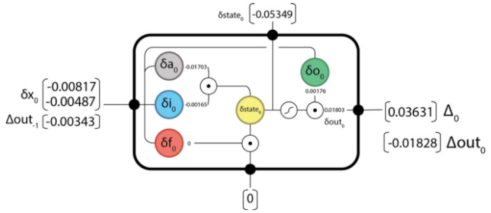
$\delta_{o_0} = 0.00176$

$$\delta_{n_0} = \begin{bmatrix} 0.45 & 0.95 & 0.7 & 0.6 \\ 0.25 & 0.8 & 0.85 & 0.4 \end{bmatrix} \begin{bmatrix} -0.01702 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = \begin{bmatrix} -0.00817 \\ -0.004871 \end{bmatrix}$$

$$\Delta out_0 = [0.15 \;\; 0.8 \;\; 0.1 \;\; 0.25] \begin{bmatrix} -0.01702 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} = [-0.003]$$

$\Delta_0 = \partial_x E = 0.53631 - 0.5 = 0.03631$

$\Delta out_0 = -0.01828$, passed back from T=1




**L2 loss**

$$E(n, \hat{n}) = \frac{(n - \hat{n})^2}{2}$$

$$\delta_n E(n, \hat{n}) = n - \hat{n}$$

**(c)** Use Forward t=0, Backward t=0 above, Forward t=1, Backward t = 1 below, compute $\delta W$, $\delta U$ and $\delta b$

Sol:

$$\delta W = \sum_{t=0}^{T} \delta gates_t \otimes x_t$$

$$= \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} [1.0 \;\; 2.0] + \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00631 \\ -0.05538 \end{bmatrix} [0.5 \;\; 3.0] = \begin{bmatrix} -0.02672 & -0.0922 \\ -0.00221 & -0.00666 \\ -0.00316 & -0.01893 \\ -0.02593 & -0.16262 \end{bmatrix} \rightarrow \text{old}$$

$x_0$, $gate_0$, $gate_1$, $x_1$

$$\delta U = \sum_{t=0}^{T-1} \delta gates_{t+1} \otimes out_t$$

$$= \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00631 \\ -0.05538 \end{bmatrix} [0.53631] = \begin{bmatrix} -0.01039 \\ -0.00060 \\ -0.00338 \\ -0.02970 \end{bmatrix}$$

$gate_1$, $out_0$

$$\delta b = \sum_{t=0}^{T} \delta gates_{t+1}$$

$$= \begin{bmatrix} -0.01703 \\ -0.00165 \\ 0 \\ 0.00176 \end{bmatrix} + \begin{bmatrix} -0.01938 \\ -0.00112 \\ -0.00631 \\ -0.05538 \end{bmatrix} = \begin{bmatrix} -0.03641 \\ -0.00277 \\ -0.00631 \\ -0.05362 \end{bmatrix}$$

$gate_0$ + $gate_1$

## Internal Parameters Update

$\delta W = \sum_{t=0}^{T} \delta gates_t \otimes n_t$

$\delta U = \sum_{t=0}^{T-1} \delta gates_{t+1} \otimes out_t$

$\delta b = \sum_{t=0}^{T} \delta gates_{t+1}$

**(d) Update parameters based on SGD update**

$$W_a = \begin{bmatrix} 0.45267 \\ 0.25922 \end{bmatrix}, U_a = [0.15104], b_a = [0.20364]$$

$$W_i = \begin{bmatrix} 0.95022 \\ 0.80067 \end{bmatrix}, U_i = [0.80006], b_i = [0.65028]$$

$$W_f = \begin{bmatrix} 0.70031 \\ 0.45189 \end{bmatrix}, U_f = [0.10034], b_f = [0.15063]$$

$$W_o = \begin{bmatrix} 0.60259 \\ 0.41626 \end{bmatrix}, U_o = [0.25297], b_o = [0.10536]$$

$W^{new} = W^{old} - \lambda \cdot \delta W^{old}$ $\quad \lambda = 0.1$

$W_a^{old} - \lambda \cdot \delta W_a^{old}$

$$\begin{bmatrix} 0.45 \\ 0.25 \end{bmatrix} - 0.1 \begin{bmatrix} -0.02672 \\ -0.0927 \end{bmatrix} = \begin{bmatrix} 0.45267 \\ 0.25922 \end{bmatrix}$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

# Positional Encoding

8) pos: 1, d: 4 so i:0 and i:1 → after every 2 dim i increases

1. Dim 0 → sin → even
   $PE(1,0) = \sin\left(1/10000^{0/4}\right) = 0.8415$
2. Dim 1 → cos → odd
   $PE(1,1) = \cos\left(1/10000^{0/4}\right) = 0.5403$
3. Dim 2 → sin
   $PE(1,2) = \sin\left(1/10000^{2/4}\right) = 0.0099$
4. Dim 3 → cos
   $PE(1,3) = \cos\left(1/10000^{2/4}\right) = 0.99995$

So final Positional Encoding Vector for Position 1 and d:4

(0.8415, 0.5403, 0.0099, 0.9995)