

Advanced Concepts in Deep Learning

Jian Zhou

TA & Practice session lead: James Elder

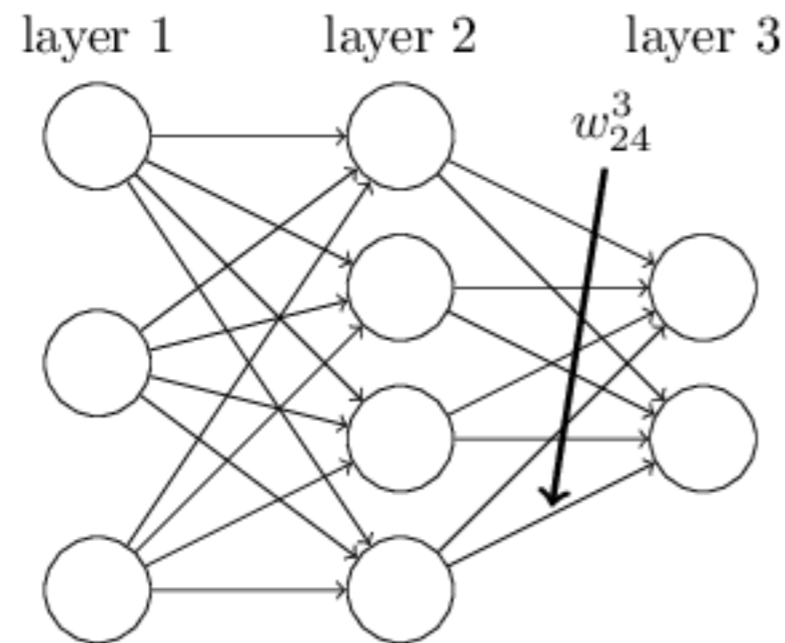
Day 1

- **Foundations of deep learning**
- **DL architectures: Transformers, Graph Networks, and more**
- Practice session: Tensorflow intro + Transformer

Day 2

- **Probabilistic / generative deep learning**
- **Deep reinforcement learning + case study: AlphaGo (if time permits)**
- **Case study: AlphaFold2 (if time permits)**
- Practice session: Diffusion model

The archetype



w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

What is deep learning?

What can it do

Flexible function approximation
capable of fitting complex functions

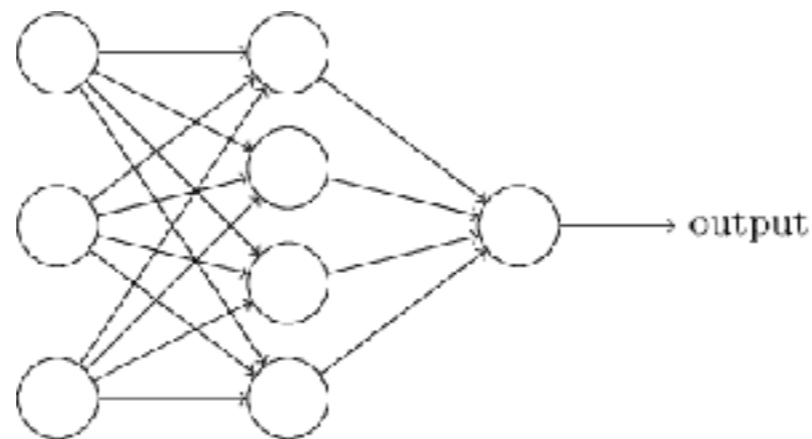
How to train it

Computable gradient
function *largely* smooth

Flexibility

- Universal representation theorem:

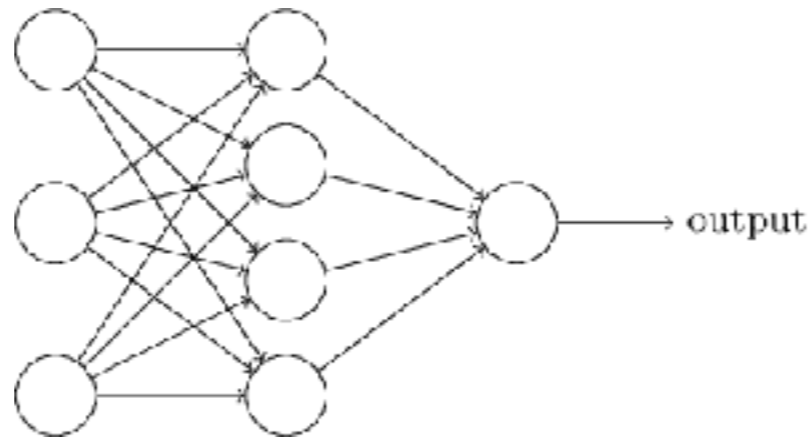
Any continuous function in finite dimensions can be approximated arbitrarily well with a two-layer neural network with *finite number of hidden unit*



Flexibility

- Universal representation theorem (improved):

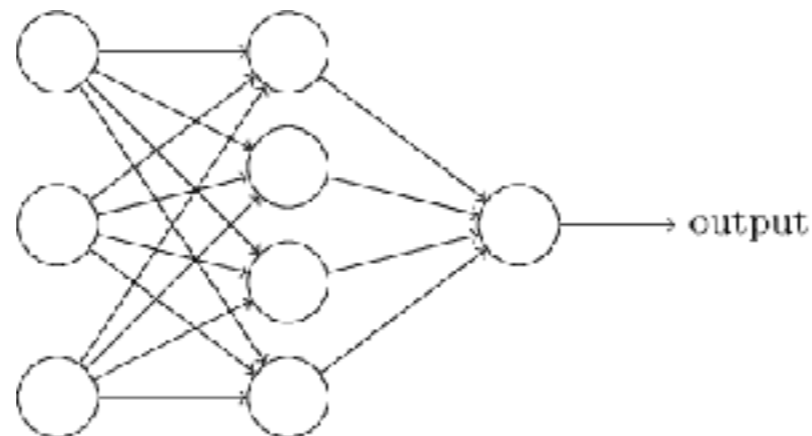
There exists a two-layer neural network with ReLU activations and $2n+d$ weights that can represent any function on a sample of size n in d dimensions.



Flexibility

- **Depth efficiency hypothesis**
(widely held belief + proof for certain models):

Some functions expressed in multi-layer models requires super-polynomial sized units to express in shallow models



Flexibility

- Flexible model does not generalize?

Rademacher complexity-based generalization bound

$$\hat{R}_m(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^m \sigma_i f(z_i) \right) \right] \quad \begin{array}{l} \text{Test Error} \\ \mathbb{E}_D[f(z)] \end{array} \leq \begin{array}{l} \text{Training Error} \\ \hat{\mathbb{E}}_S[f(z)] \end{array} + 2R_m(\mathcal{F}) + \sqrt{\frac{\ln(1/\delta)}{m}}.$$

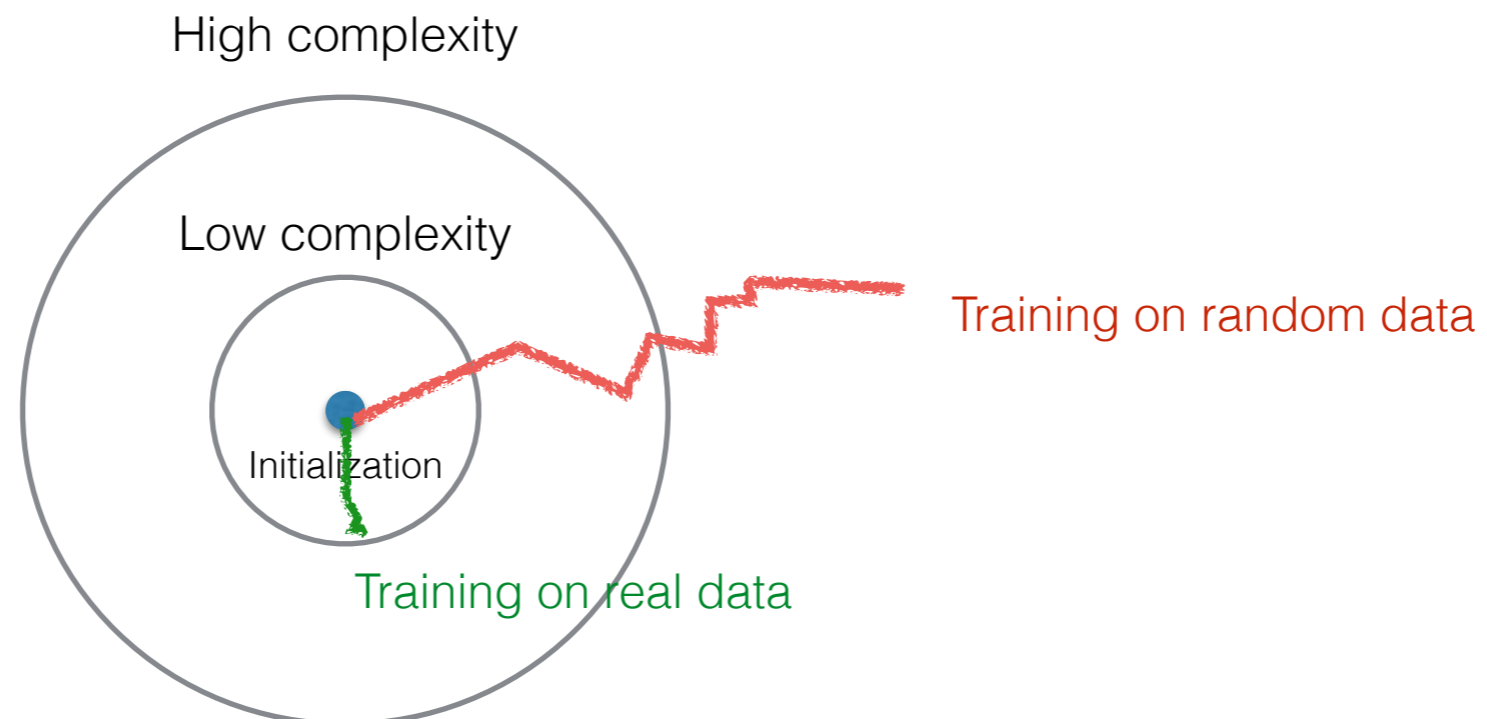
with probability at least $1 - \delta$

Fun fact: neural network usually has the capacity to memorize random labels perfectly

Flexibility

- Flexible model does not generalize?

In practice, models are never trained to obtain the minimal training loss



Notion of generalization based on the 'length' of training path?

Gradient

- Implicit assumption is that deep learning models can be learned by simply gradient descent

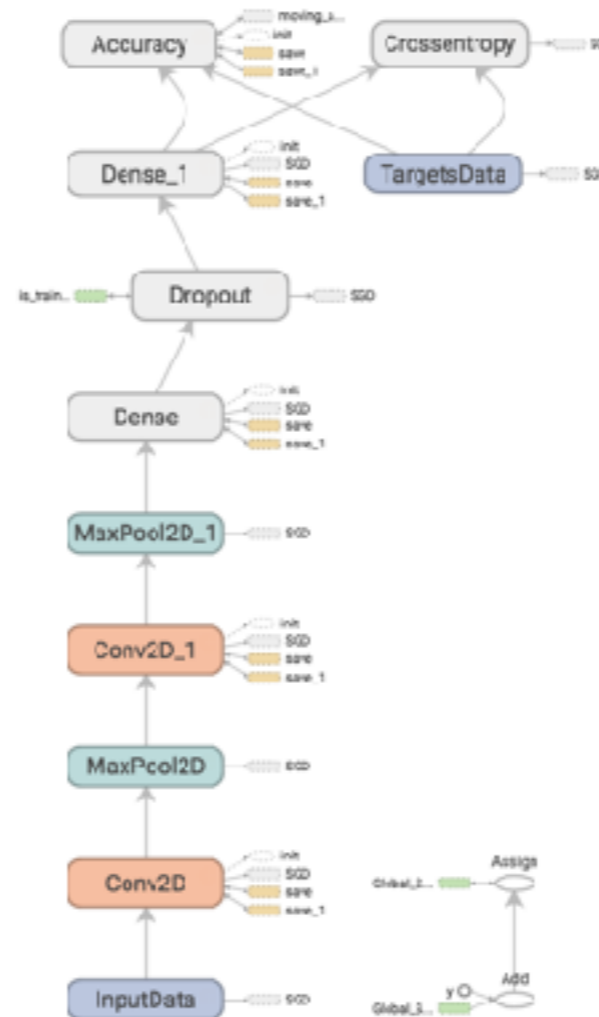
It will be interesting to understand when this assumption fails (e.g., prime factorization?)

Computation of Gradient: Automatic differentiation

Allow trivial solution to complex models /
changing model structure dynamically (data-dependent)

- The basics: $\frac{dy}{dx} = \frac{dy}{dw} \frac{dw}{dx}$

- Computational graph:



Computation of Gradient: Automatic differentiation

Allow trivial solution to complex models /
changing model structure dynamically (data-dependent)

- The basics: $\frac{dy}{dx} = \frac{dy}{dw} \frac{dw}{dx}$
- Two modes: forward mode and backward mode

(optimal traversal path for arbitrary computational graph is NP-complete)

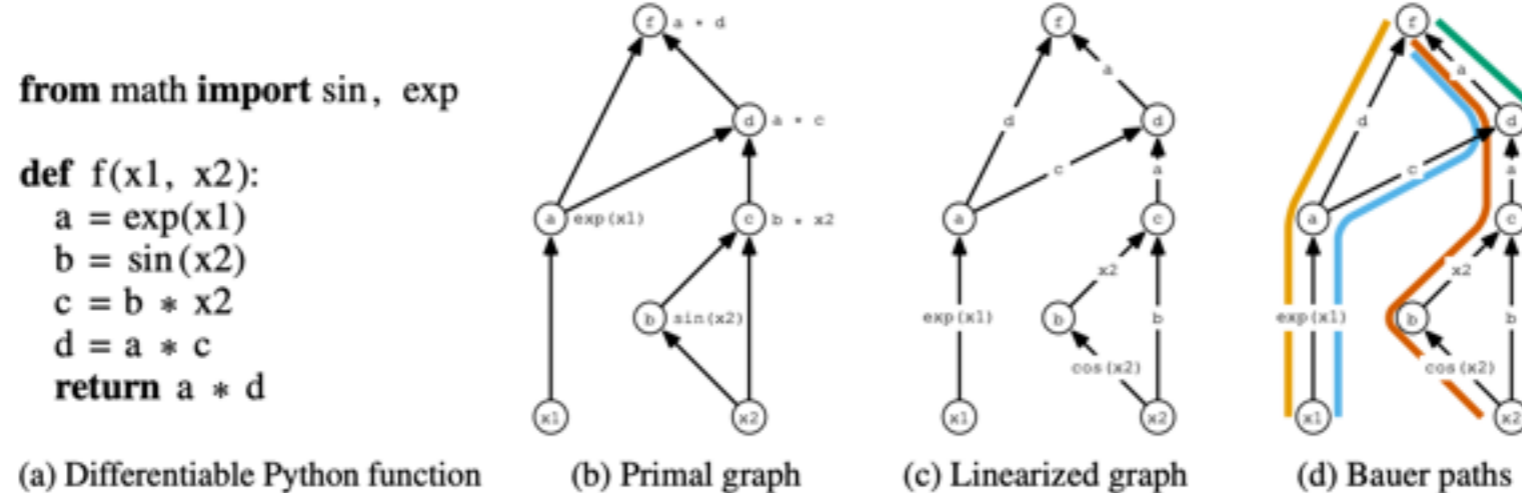
- Further improvement:
 - Compiler for mathematical expressions that achieves acceleration and numeric stability (JIT in pytorch, JAX)
 - Mixing programming language with computational graph (conditionals, loops, etc with mathematical functions)
 - Higher-order derivative (e.g. Hessian)

Computation of Gradient: Automatic differentiation

We only need stochastic gradient, so why not **randomized** automatic differentiation?

Unbiased estimator of gradient

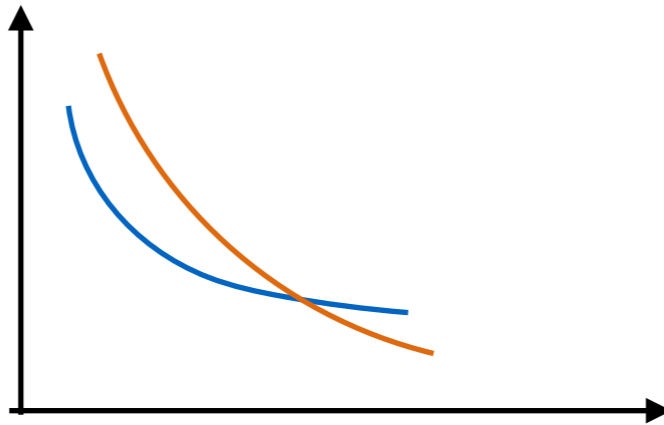
True gradient is sum of gradient through each computational paths, so subsampling the path leads to unbiased estimator



Sparse implementation similar to dropout in backward pass

Use gradient efficiently: Stochastic gradient descent

$1/\sqrt{t}$ error rate (stochastic) vs $1/t$ error rate (batch)



‘High optimization error’ is tolerable:

No need to optimize beyond the statistical limit

Is SGD adaptive to the data uncertainty?

Connection between Stochastic Gradient Descent and Bayesian inference

SGD as MCMC

Stochastic gradient Langevin dynamics, Welling and Teh, 2011

SGD

$$\Delta\theta_t = \frac{\epsilon_t}{2} \left(\nabla \log p(\theta_t) + \frac{N}{n} \sum_{i=1}^n \nabla \log p(x_{ti}|\theta_t) \right)$$

MCMC by Stochastic gradient Langevin dynamics

$$\Delta\theta_t = \frac{\epsilon_t}{2} \left(\nabla \log p(\theta_t) + \frac{N}{n} \sum_{i=1}^n \nabla \log p(x_{ti}|\theta_t) \right) + \eta_t$$
$$\eta_t \sim N(0, \epsilon_t) \quad (4)$$

MCMC by Langevin dynamics

$$\Delta\theta_t = \frac{\epsilon}{2} \left(\nabla \log p(\theta_t) + \sum_{i=1}^N \nabla \log p(x_i|\theta_t) \right) + \eta_t$$
$$\eta_t \sim N(0, \epsilon) \quad (3)$$

Connection between Stochastic Gradient Descent and Bayesian inference

SGD as VI

Stochastic Gradient Descent as Approximate Bayesian Inference, Mandt, 2017

$$\hat{g}_S(\theta) \approx g(\theta) + \frac{1}{\sqrt{S}} \Delta g(\theta), \quad \Delta g(\theta) \sim \mathcal{N}(0, C(\theta)). \quad C(\theta) \approx C = BB^\top$$

S is mini-batch size

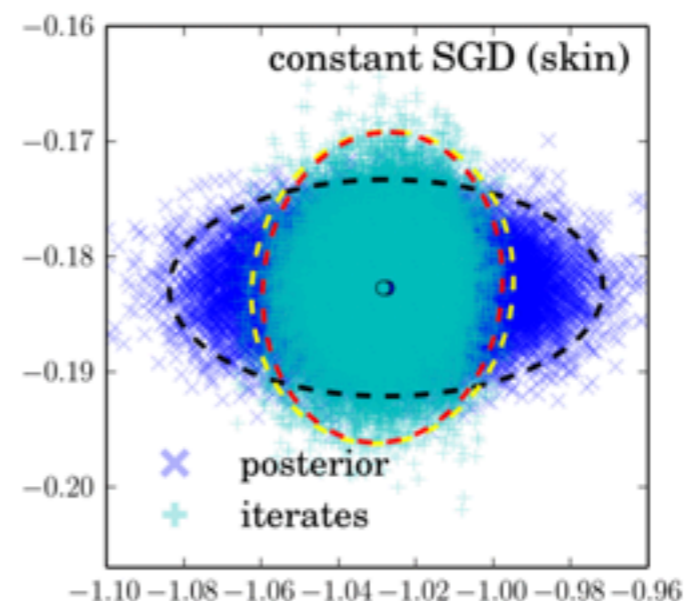
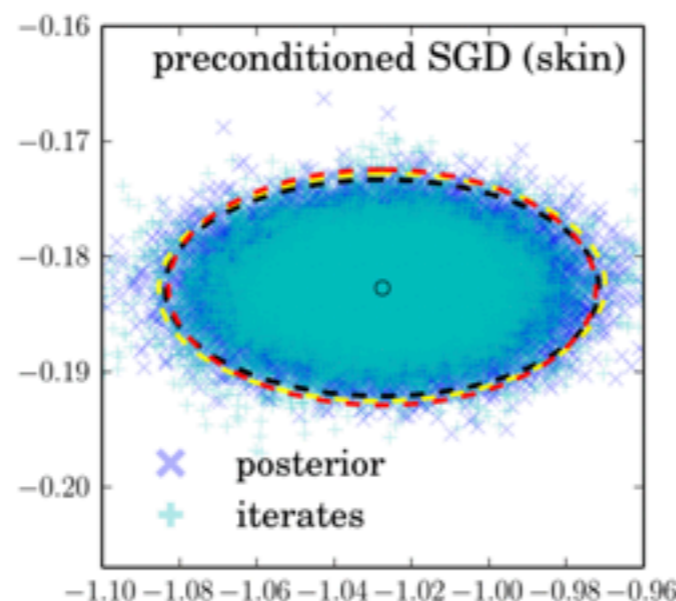
SGD is then equivalent to stochastic process $d\theta(t) = -\epsilon g(\theta) dt + \frac{\epsilon}{\sqrt{S}} B dW(t)$

which converge to Gaussian stationary distribution with covariance

Optimal preconditioning matrix

$$\theta_{t+1} = \theta_t - H \hat{g}_S(\theta(t)).$$

$$H^* = \frac{2S}{N} (BB^\top)^{-1}$$

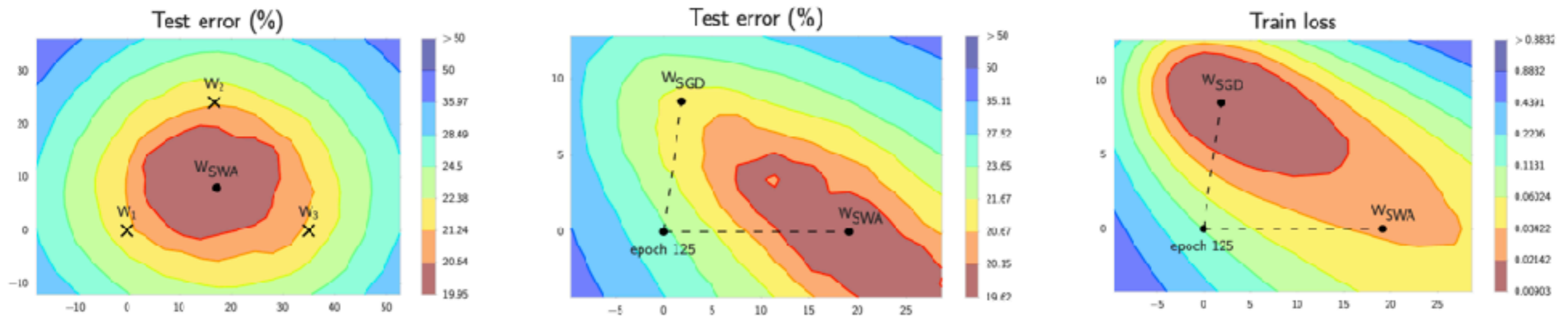


Optimal learning rate

$$\epsilon^* = 2 \frac{S}{N} \frac{D}{\text{Tr}(BB^\top)}$$

SGD should not be considered simply as approximate gradient descent

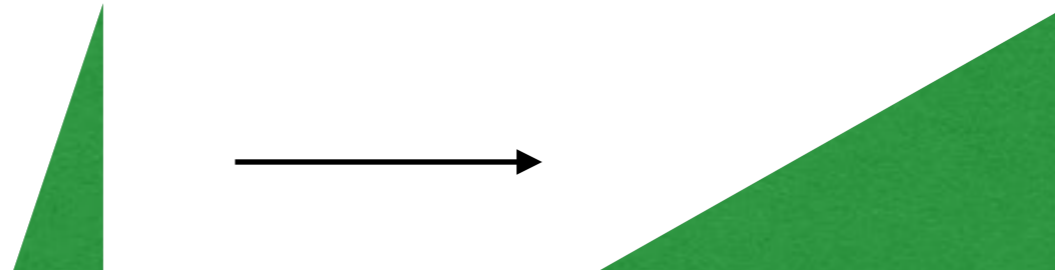
Find the center of the posterior: Stochastic weight averaging



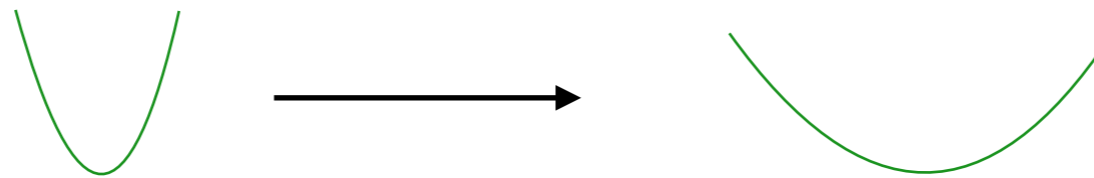
SWA can be seen as a particular type of learning rate decay $1-N/N_{max}$

Optimization: scale invariance

Naive gradient descent is not scale-invariant



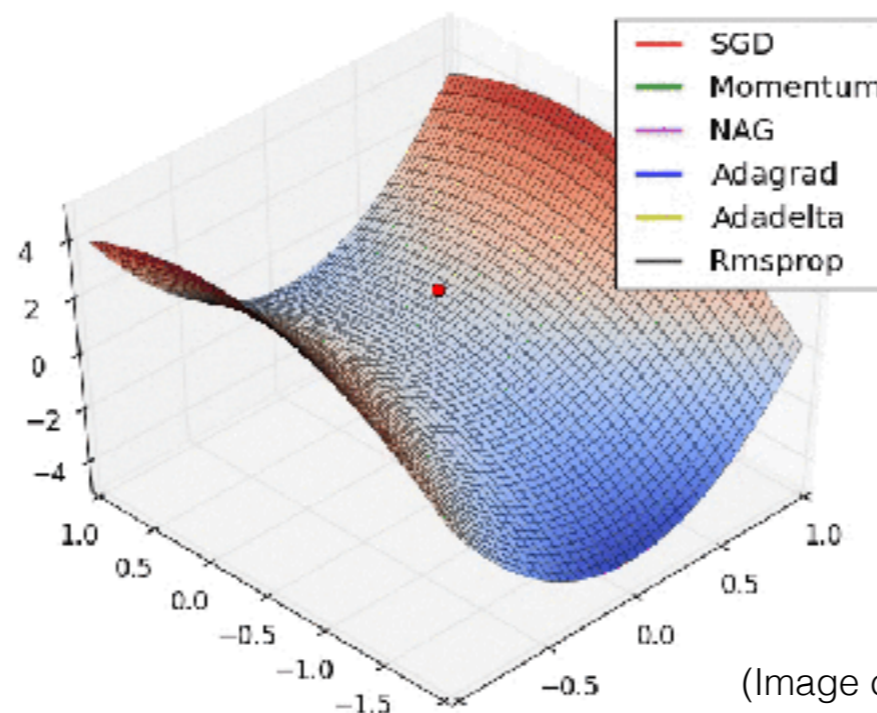
Known solution: use curvature of the surface (second order methods)



The exact way: compute Hessian matrix (second order derivatives) / Newton's method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{f}''(\mathbf{x}_k)]^{-1} \mathbf{f}'(\mathbf{x}_k)$$

The cheap way : approximation using the **history of gradients**



(Image credit: Alec Radford)

Optimization: variance reduction and scale invariance

SGD+momentum

$$g_t = 0.9 * g_{t-1} + 0.1 * g$$

RMSprop

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t.$$

Adam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t.$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2.$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t.$$

Training neural network without gradient: zero-th order optimization

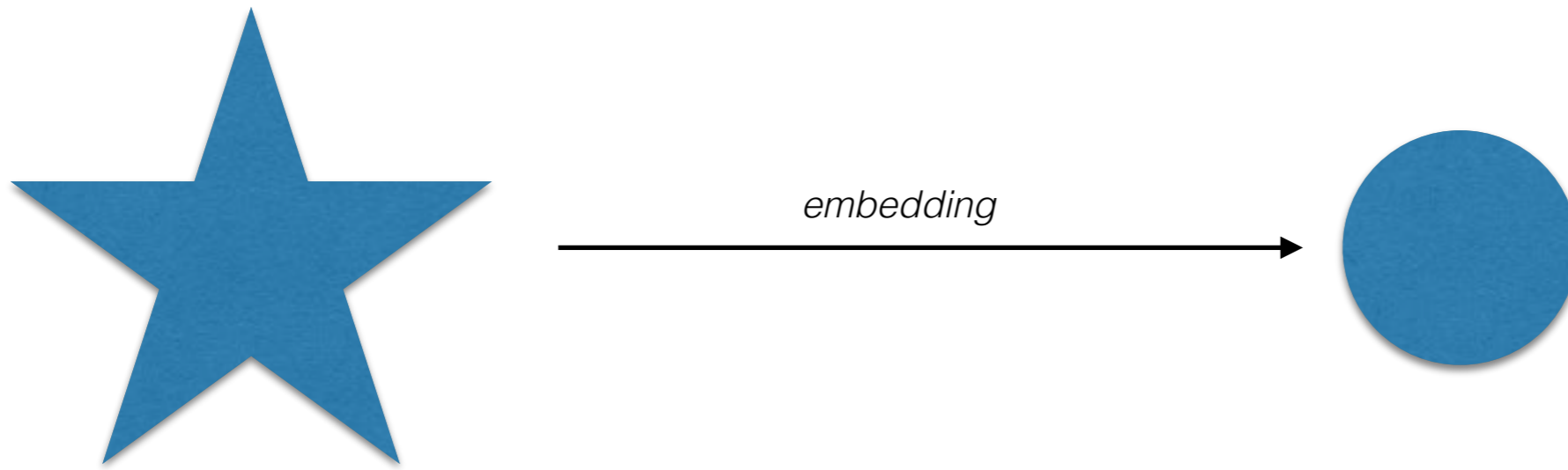
Gradient approximation via random-perturbation

$$\hat{\nabla} \mathcal{L}(\boldsymbol{\theta}; \mathcal{B}) = \frac{\mathcal{L}(\boldsymbol{\theta} + \epsilon \mathbf{z}; \mathcal{B}) - \mathcal{L}(\boldsymbol{\theta} - \epsilon \mathbf{z}; \mathcal{B})}{2\epsilon} \mathbf{z} \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}_d)$$

Forward-pass only optimization is possible with significant memory advantage

Learning representations

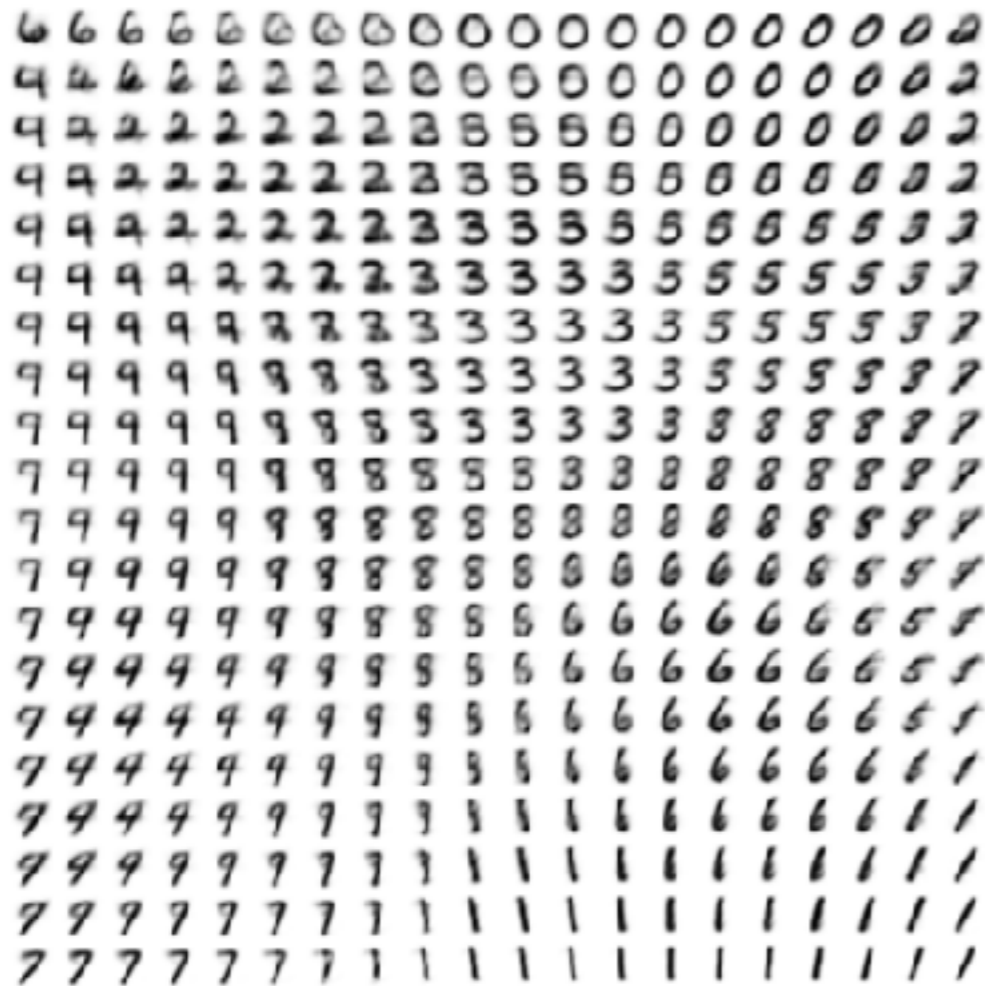
Raw data that lives in some arbitrary (high-dimensional) space



Representation space with
smooth and linear structure

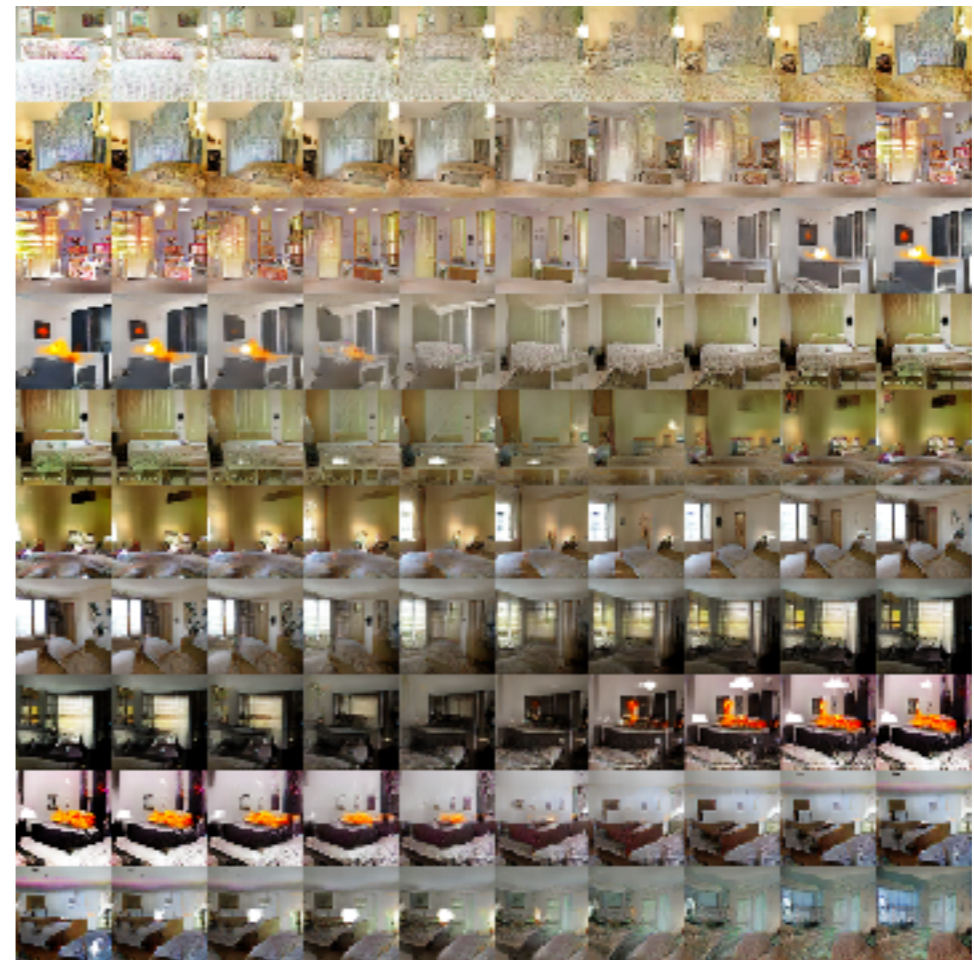
Representation: smoothness

Digits (MNIST)



Embedding learned by
variational autoencoder (VAE)

Bedroom (LSUN)



Embedding learned by
generative adversarial networks (GAN)

Representation: smoothness

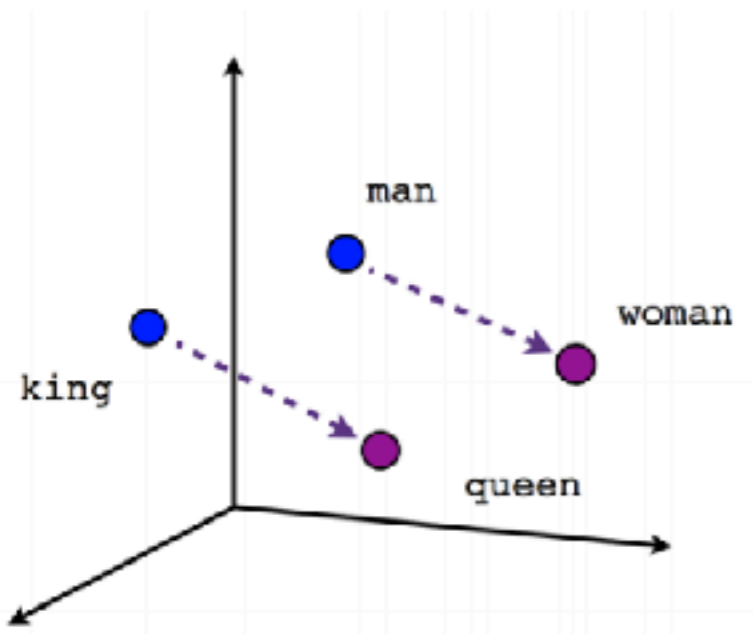


RNN autoencoder <https://arxiv.org/abs/1704.03477>

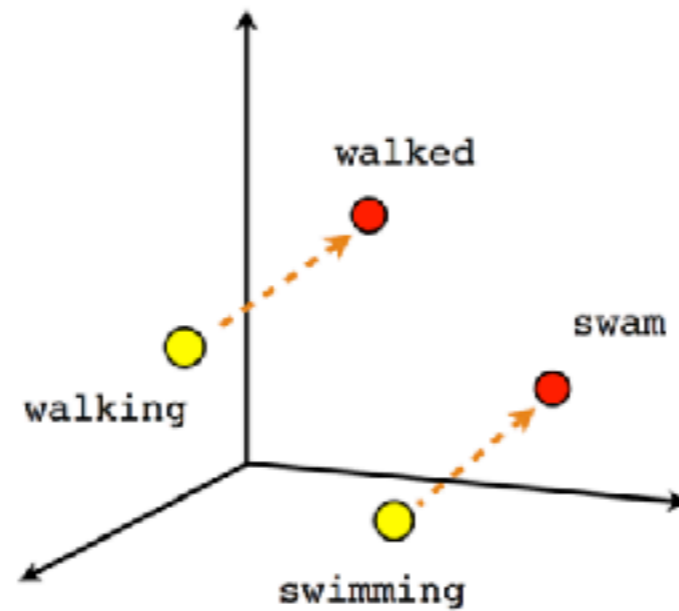
字 種 成 東 字 推
符 利 對 亞 型 斷
到 用 抗 語 進 的
字 條 網 言 行 新
符 件 絡 字 自 方
一 生 對 體 動 法

GAN github.com/kaonashi-tyc/zi2zi

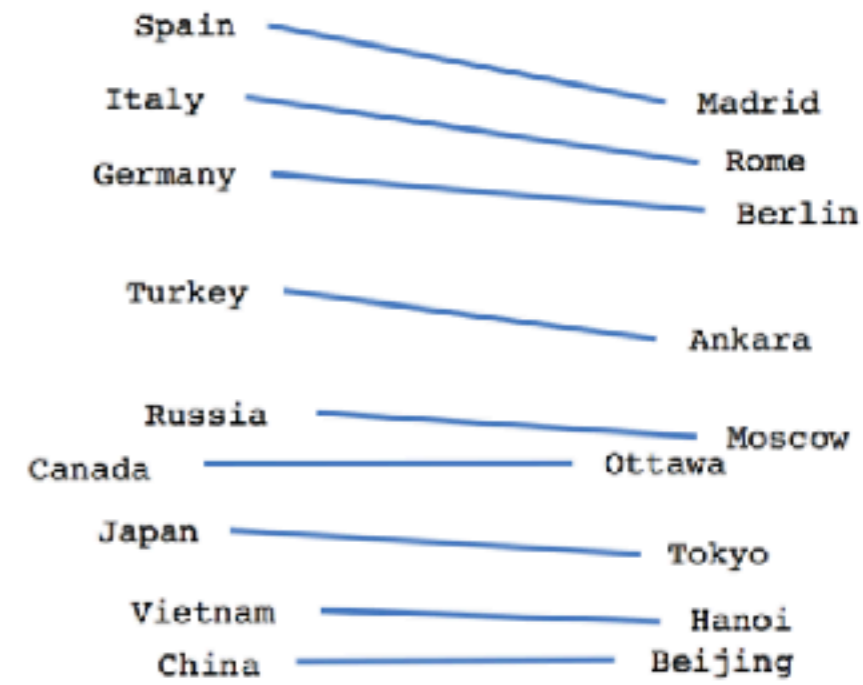
Representation: linearity



Male-Female



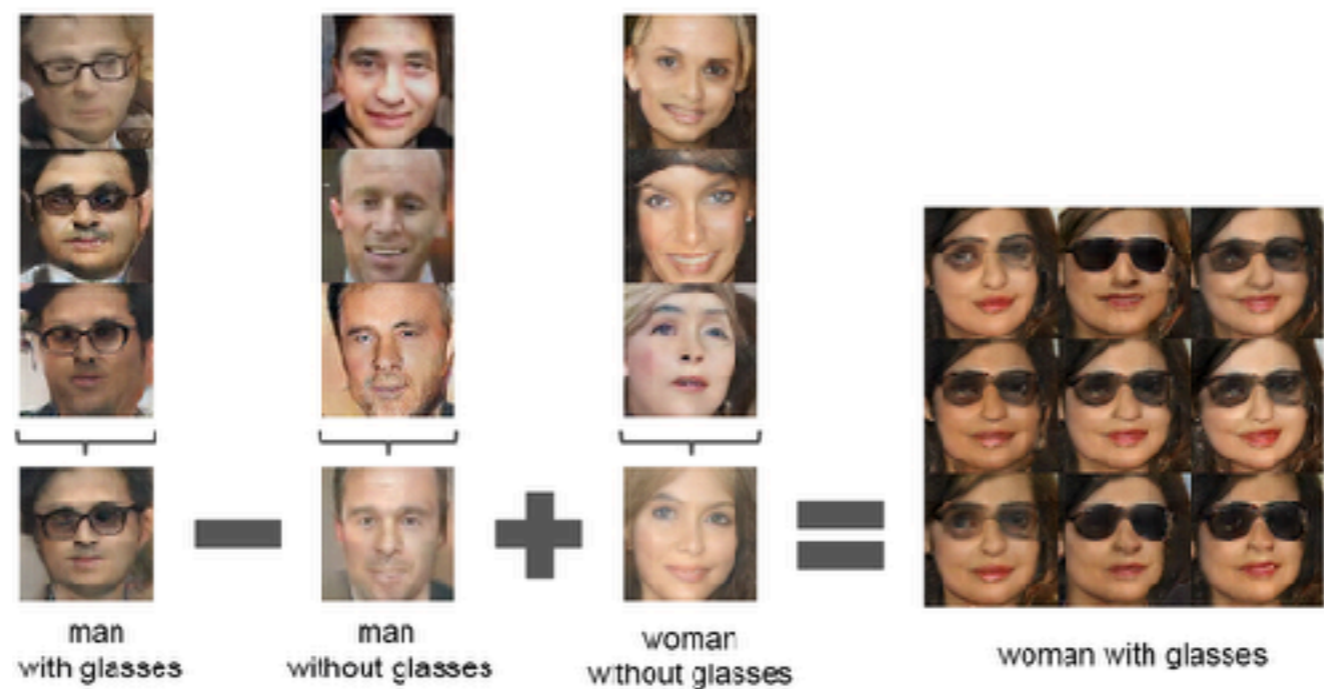
Verb tense



Country-Capital

Pretrained word vectors for >70 languages are publicly available

Representation: linearity



Representation learning

This is one of Crichton's best books. The characters of Karen Ross, Peter Elliot, Munro, and Amy are beautifully developed and their interactions are exciting, complex, and fast-paced throughout this impressive novel. And about 99.8 percent of that got lost in the film. Seriously, the screenplay AND the directing were horrendous and clearly done by people who could not fathom what was good about the novel. I can't fault the actors because frankly, they never had a chance to make this turkey live up to Crichton's original work. I know good novels, especially those with a science fiction edge, are hard to bring to the screen in a way that lives up to the original. But this may be the absolute worst disparity in quality between novel and screen adaptation ever. The book is really, really good. The movie is just dreadful.

“Sentiment neuron”

Trained on 82 million Amazon reviews to
predict the next character

method: multiplicative LSTM

Part 2. deep learning model architectures

Transformer: state-of-the-art architecture for NLP and beyond

[nature](#) > [articles](#) > [article](#)

Article | [Open Access](#) | [Published: 07 June 2023](#)

Faster sorting algorithms discovered using deep reinforcement learning

[Daniel J. Mankowitz](#) , [Andrea Michi](#), [Anton Zhernov](#), [Marco Gelmi](#), [Marco Selvi](#), [Cosmin Paduraru](#), [Edouard Leurent](#), [Shariq Iqbal](#), [Jean-Baptiste Lespiau](#), [Alex Ahern](#), [Thomas Köppe](#), [Kevin Millikin](#), [Stephen Gaffney](#), [Sophie Elster](#), [Jackson Broshear](#), [Chris Gamble](#), [Kieran Milan](#), [Robert Tung](#), [Minjia Hwang](#), [Taylan Cemgil](#), [Mohammadamin Berekatain](#), [Yujia Li](#), [Amol Mandhane](#), [Thomas Hubert](#), ... [David Silver](#) 

[Nature](#) **618**, 257–263 (2023) | [Cite this article](#)

Original

```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P // P = A
mov Memory[1] Q // Q = B
mov Memory[2] R // R = C

mov R S
cmp P R
cmovg P R // R = max(A, C)
cmovl P S // S = min(A, C)
mov S P // P = min(A, C)
cmp S Q
cmovg Q P // P = min(A, B, C)
cmovg S Q // Q = max(min(A, C), B)

mov P Memory[0] // = min(A, B, C)
mov Q Memory[1] // = max(min(A, C), B)
mov R Memory[2] // = max(A, C)
```

AlphaDev

```
Memory[0] = A
Memory[1] = B
Memory[2] = C

mov Memory[0] P // P = A
mov Memory[1] Q // Q = B
mov Memory[2] R // R = C

mov R S
cmp P R
cmovg P R // R = max(A, C)
cmovl P S // S = min(A, C)

cmp S Q
cmovg Q P // P = min(A, B)
cmovg S Q // Q = max(min(A, C), B)

mov P Memory[0] // = min(A, B)
mov Q Memory[1] // = max(min(A, C), B)
mov R Memory[2] // = max(A, C)
```

Transformer: state-of-the-art architecture for NLP and beyond

10

the following is a compiled version of a sorting algorithm in assembly. I think it can be improved, can you indicate in the following lines, with ******* which instructions could be removed, or changed? If not don't do anything, take it step by step and explain the reasoning, and go back and verify that it was correct

```
Memory[0] = A
Memory[1] = B
Memory[2] = c
```

```
mov Memory[0] P
mov Memory[1] Q
mov Memory[2] R
```

```
mov R S
cmp P R
cmovg P R // this is equivalent to R = max(A, C)
cmovl P S // this is equivalent to S = min(A, C)
mov S P // this is equivalent to P = min(A, C)
cmp S Q
cmovg Q P // this is equivalent to P = min(A, B, C)
cmovg S Q // this is equivalent to Q = max(min(A, C), B)
```

```
mov P Memory[0] // this is equivalent to = min(A, B, C)
mov Q Memory[1] // this is equivalent to = max(min(A, C), B)
mov R Memory[2] // this is equivalent to = max(A, C)
```

go over the above instructions in steps that make sense, don't say as a first pass if they can be removed or changed, just look at them and express some written thoughts that may help you in the second step.

First step first, then you ask me to move on to step two. Be very detailed, and VERY careful

Let's go through the code step by step, analyzing what each instruction does, and making some observations that might be helpful for future optimization. At this stage, I won't make any recommendations for removal or changes, I'll just analyze the purpose of each instruction.

```
css
Memory[0] = A
Memory[1] = B
Memory[2] = c
```

These lines just indicate that there are three values stored in memory locations 0, 1, and 2. They represent the initial state of the memory, and not instructions in assembly language.

```
css
mov Memory[0] P
mov Memory[1] Q
mov Memory[2] R
```

These instructions move the values stored in memory locations 0, 1, and 2 into registers P, Q, and R respectively. It's a common practice to load data into registers for faster access and manipulation.

```
mov R S
```

This instruction copies the value of register R into register S. At this point, both R and S contain the value of C (Memory[2]). This might be useful for comparisons that are coming next.

Transformer: State-of-the-art architecture for NLP (and beyond)

10

the following is a compiled version of a sorting algorithm in assembly. I think it can be improved, can you indicate in the following lines, with ******* which instructions could be removed, or changed? If not don't do anything, take it step by step and explain the reasoning, and go back and verify that it was correct

```
Memory[0] = A
Memory[1] = B
Memory[2] = c
```

```
mov Memory[0] P
mov Memory[1] Q
mov Memory[2] R
```

```
mov R S
cmp P R
cmovg P R // this is equivalent to R = max(A, C)
cmovl P S // this is equivalent to S = min(A, C)
mov S P // this is equivalent to P = min(A, C)
cmp S Q
cmovg Q P // this is equivalent to P = min(A, B, C)
cmovg S Q // this is equivalent to Q = max(min(A, C), B)
```

```
mov P Memory[0] // this is equivalent to = min(A, B, C)
mov Q Memory[1] // this is equivalent to = max(min(A, C), B)
mov R Memory[2] // this is equivalent to = max(A, C)
```

go over the above instructions in steps that make sense, don't say as a first pass if they can be removed or changed, just look at them and express some written thoughts that may help you in the second step.

First step first, then you ask me to move on to step two. Be very detailed, and VERY careful

Let's go through the code step by step, analyzing what each instruction does, and making some observations that might be helpful for future optimization. At this stage, I won't make any recommendations for removal or changes, I'll just analyze the purpose of each instruction.

```
css
Memory[0] = A
Memory[1] = B
Memory[2] = c
```

These lines just indicate that there are three values stored in memory locations 0, 1, and 2. They represent the initial state of the memory, and not instructions in assembly language.

```
css
mov Memory[0] P
mov Memory[1] Q
mov Memory[2] R
```

These instructions move the values stored in memory locations 0, 1, and 2 into registers P, Q, and R respectively. It's a common practice to load data into registers for faster access and manipulation.

```
css
mov R S
```

This instruction copies the value of register R into register S. At this point, both R and S contain the value of C (Memory[2]). This might be useful for comparisons that are coming next.

Transformer: State-of-the-art architecture for NLP (and beyond)

@DimitrisPapail



Let's move on to step two, focusing on optimization:

```
css
mov Memory[0] P
mov Memory[1] Q
mov Memory[2] R
```

These lines are essential for loading values from memory.

```
css
mov R S
```

This line is necessary to have an additional register (S) for comparison and swapping.

```
css
cmp P R
```

This line is essential for comparing the values in P and R.

```
css
cmovg P R
```

This line is essential for swapping the values if $P > R$, ensuring R will hold the $\max(A, C)$.

```
css
cmovl P S
```

```
css
* * * mov S P
```

This instruction can be optimized out. We know that after the previous two instructions, S holds the minimum of A and C. Instead of moving it to P, we can directly use S in the following comparisons, saving one instruction.

```
css
cmp S Q
```

This line is essential for comparing the values in S and Q.

```
css
cmovg Q P
```

This line is essential for finding the minimum among A, B, and C.

```
css
cmovg S Q
```

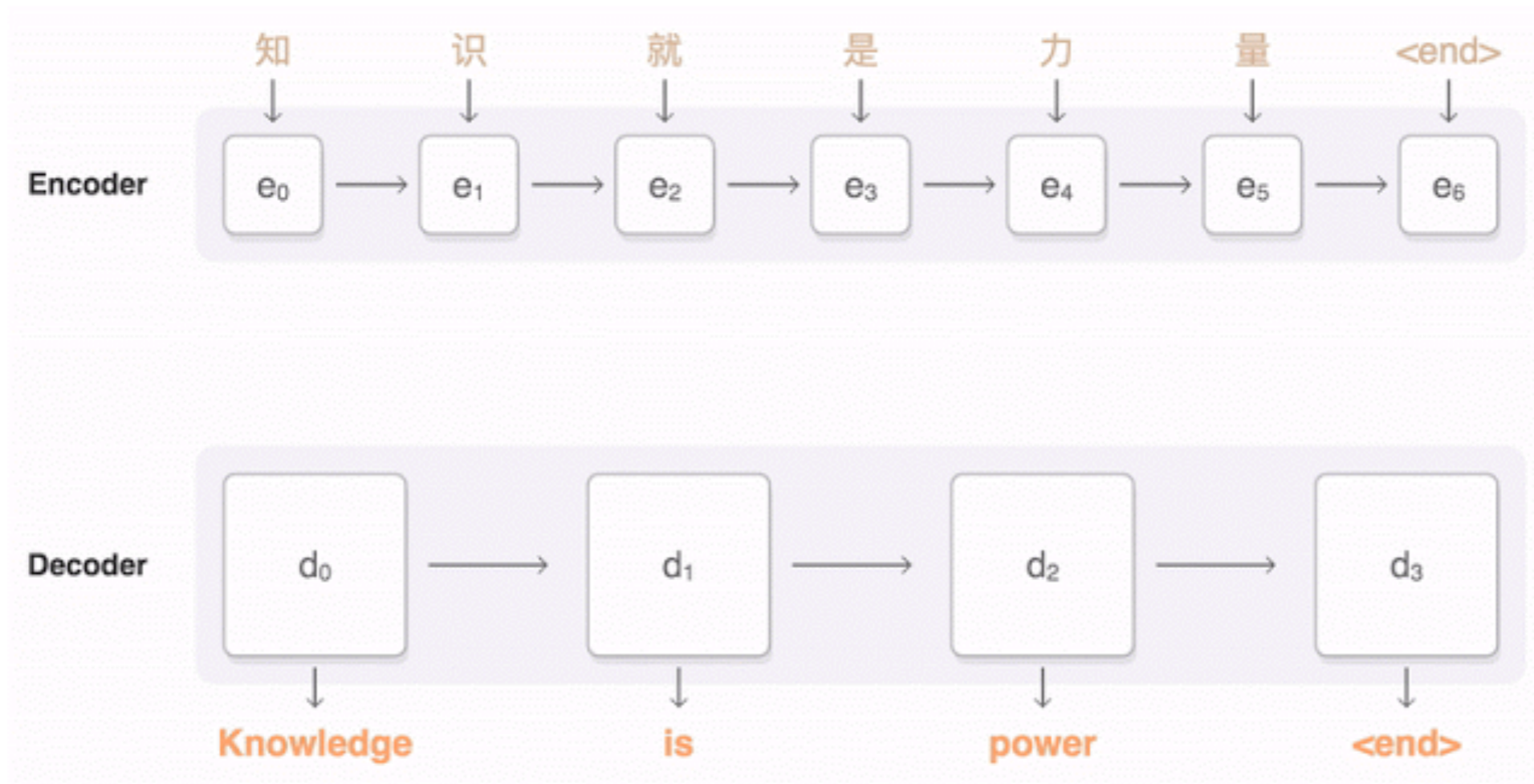
This line is essential for determining the middle value.

```
css
mov P Memory[0]
mov Q Memory[1]
mov R Memory[2]
```

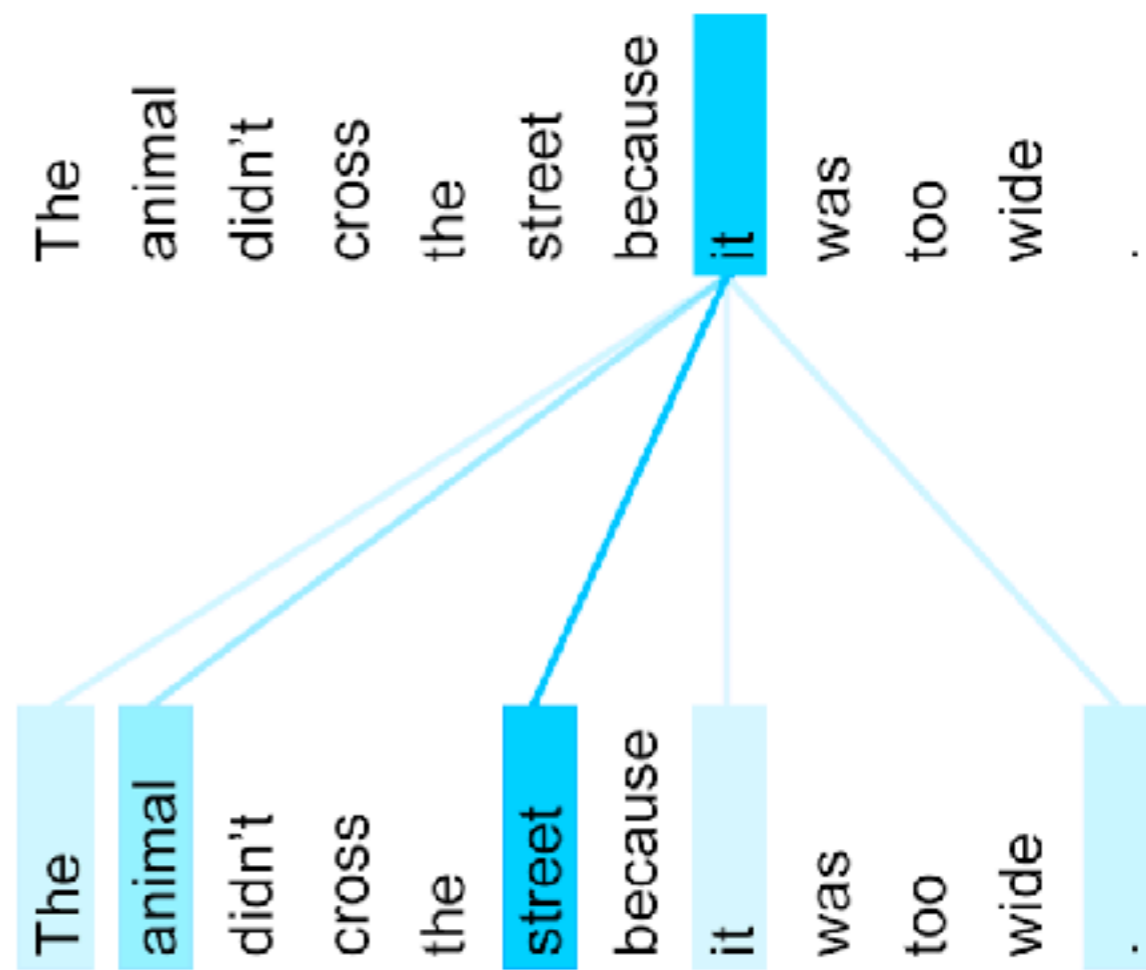
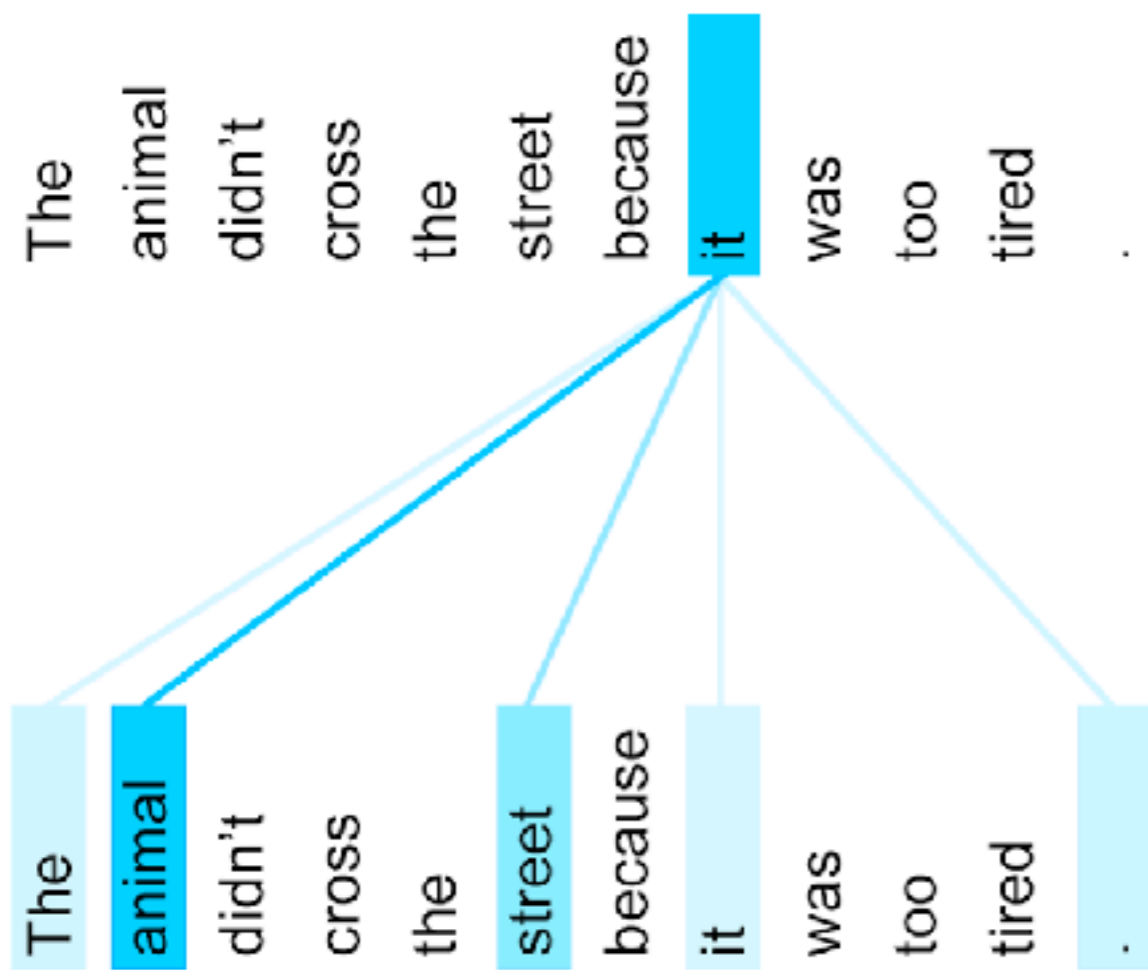
These lines are essential for storing the sorted values back into memory.

To summarize, we found that the instruction `*mov S P` can be removed. The rest of the instructions are necessary for the algorithm to function correctly. However, in the lines following the removal, P should be replaced with S to maintain correctness.

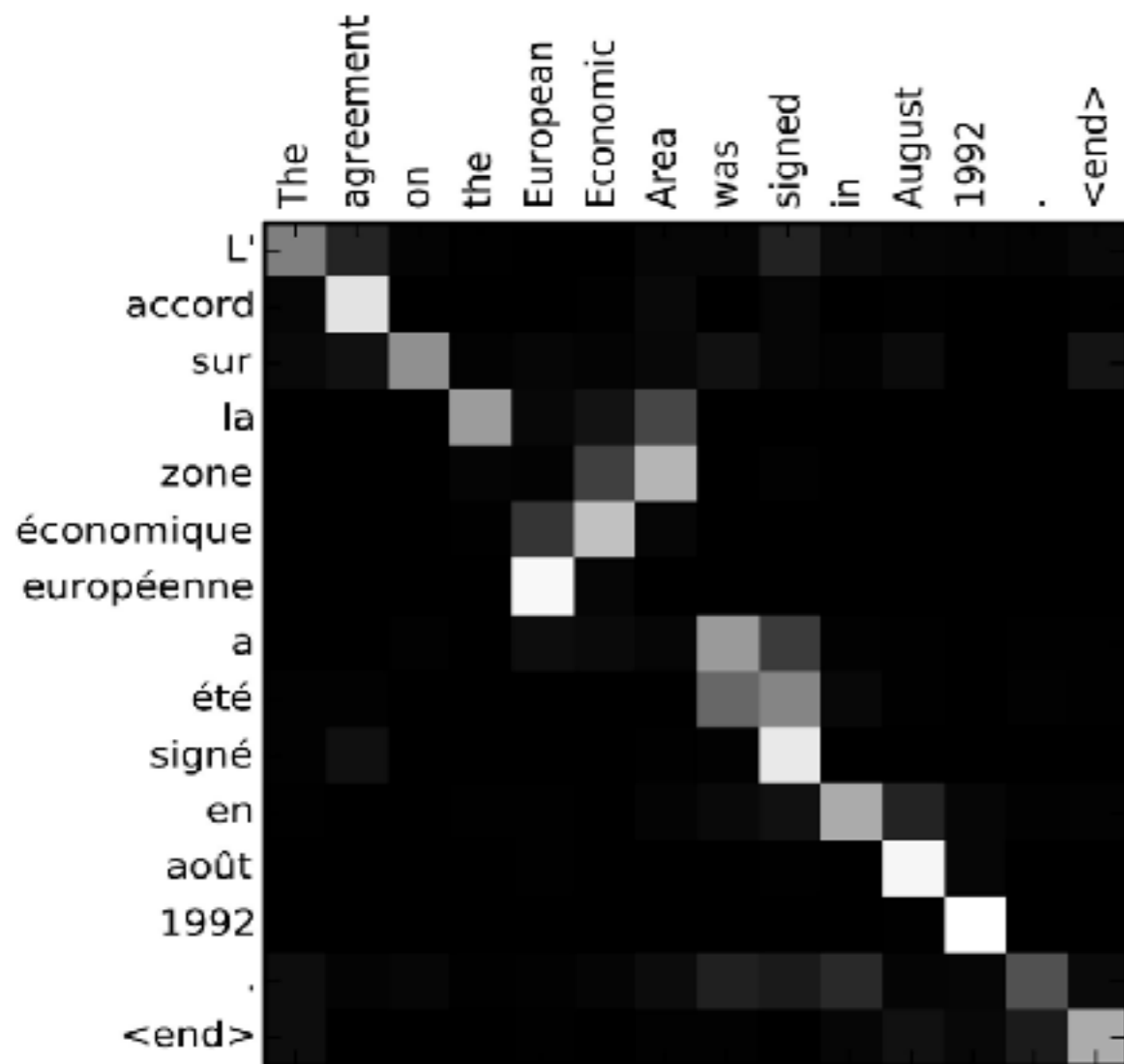
Attention mechanism: input-dependent dynamic weighting



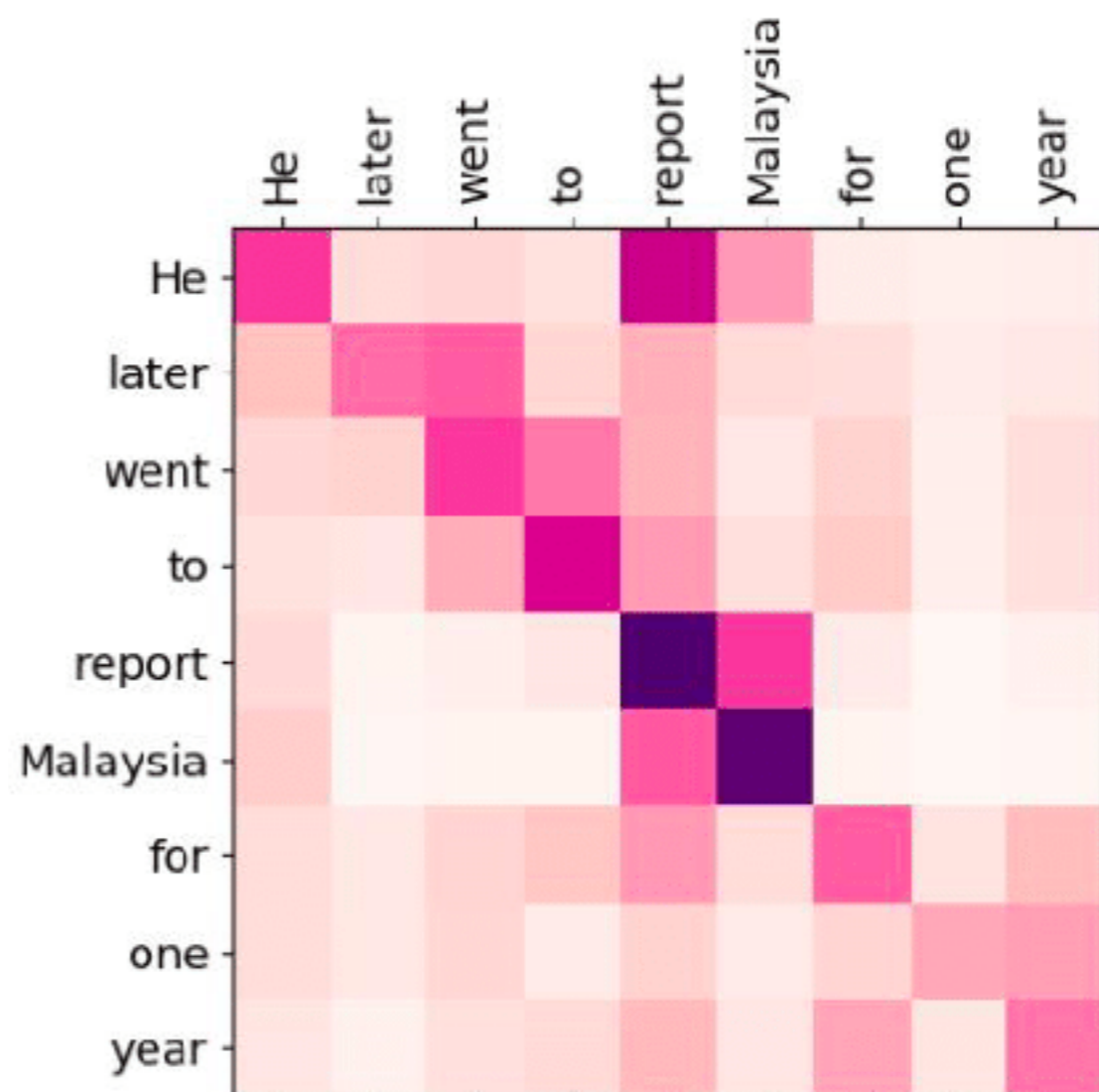
Attention mechanism: input-dependent dynamic weighting



Examples of attention matrices



Cross attention

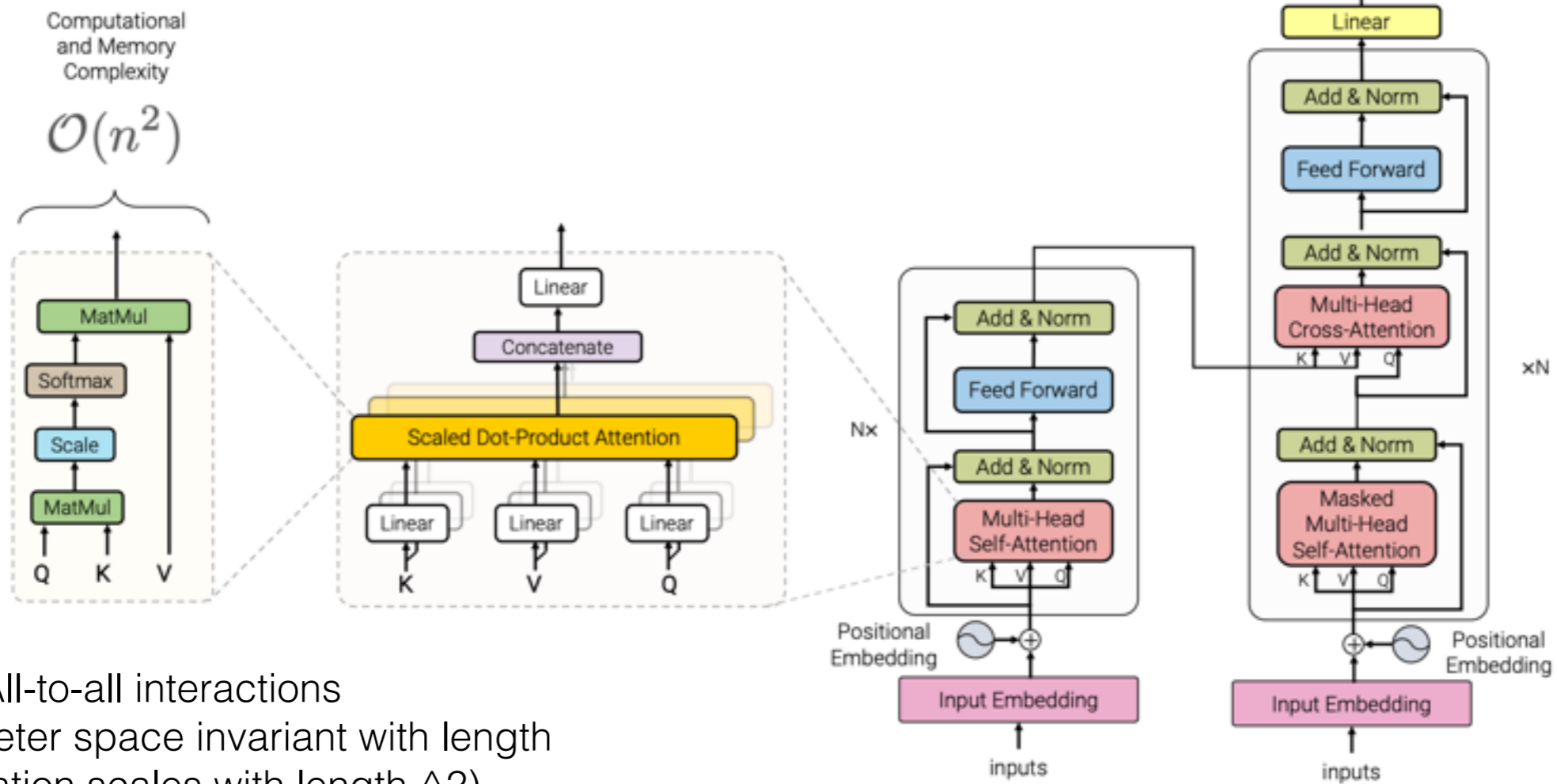


Self-attention

Transformers -attention is all you need?

Encoder-Decoder transformer architecture

Multihead dot product attention



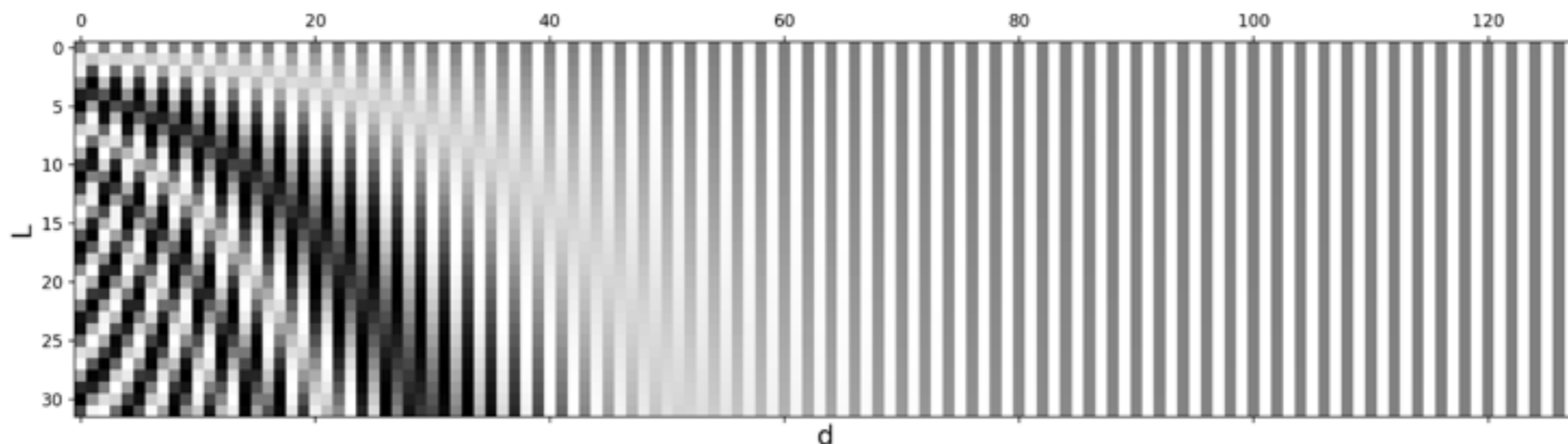
All-to-all interactions
Small parameter space invariant with length
(computation scales with length 2)

At least you also need positional encoding!

Pre-specified positional encoding / embedding: the original transformer

$$\text{PE}(i, \delta) = \begin{cases} \sin\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' \\ \cos\left(\frac{i}{10000^{2\delta'/d}}\right) & \text{if } \delta = 2\delta' + 1 \end{cases}$$

Important: this assumes input length $\ll 10000$
Increase the number your input is long (len^2)



Note that this ensures a large number of dimensions have near constant positional embedding

Fig. 3. Sinusoidal positional encoding with $L = 32$ and $d = 128$. The value is between -1 (black) and 1 (white) and the value 0 is in gray.

or, learned positional encoding (absolute or relative)

What does learned positional embedding learn?

BERT is trained on length-128 sentences in the first stage and extend to 512 in the second stage

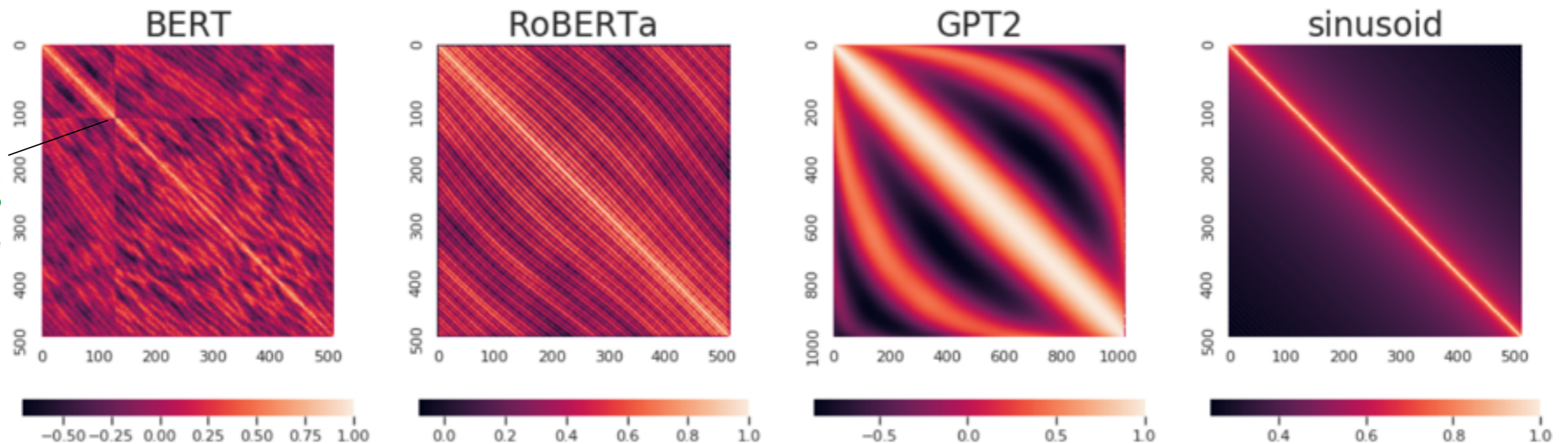


Figure 1: Visualization of position-wise cosine similarity of different position embeddings. Lighter in the figures denotes the higher similarity.

Hypothesis: Bidirectional language models (BERT/RoBERTa) are less good at learning positions compared to autoregressive language model (GPT2) (both with unsupervised training / language modeling task)

| Type | PE | MAE |
|-------------|----------|-------|
| Learned | BERT | 34.14 |
| | RoBERTa | 6.06 |
| | GPT-2 | 1.03 |
| Pre-Defined | sinusoid | 0.0 |

Predict position from embedding with Linear regression

| Type | PE | Error Rate |
|-------------|----------|------------|
| Learned | BERT | 19.72% |
| | RoBERTa | 7.23% |
| | GPT-2 | 1.56% |
| Pre-Defined | sinusoid | 5.08% |

Predict the order of two positions with Logistic regression

Table 1: Mean absolute error of the reversed mapping function learned by linear regression.

Table 2: Error rate of the relative position regression.

Relative position encoding encodes only relative distances

Depends on relative distances but not absolute positions

$$A = \text{Softmax} \left(\begin{array}{c} \text{Dark Gray Box} \\ QK^T \end{array} + \begin{array}{c} \text{Light Gray Box} \\ \Omega(m - n) \end{array} \right)$$

Ω can be implemented in different ways:
Lookup table,
MLP,
Rotary positional encoding,...

Relative positional encoding is more natural for data with translational invariance

Rotary Positional Embedding (RoPE)

Inner product of input with positional embedding should only be sensitive to the relative distance $m-n$

$$\begin{aligned}
 \text{RoPE}(x, m) &= x e^{mi\varepsilon} \\
 \langle \text{RoPE}(q_j, m), \text{RoPE}(k_j, n) \rangle &= \langle q_j e^{mi\varepsilon}, k_j e^{ni\varepsilon} \rangle \\
 &= q_j k_j e^{mi\varepsilon} \overline{e^{ni\varepsilon}} \\
 &= q_j k_j e^{(m-n)i\varepsilon} \\
 &= \text{RoPE}(q_j k_j, m - n)
 \end{aligned}$$

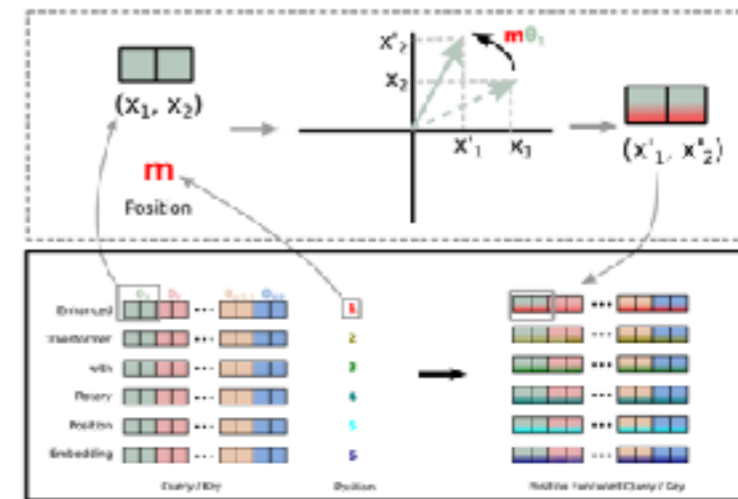


Figure 1: Implementation of Rotary Position Embedding(RoPE).

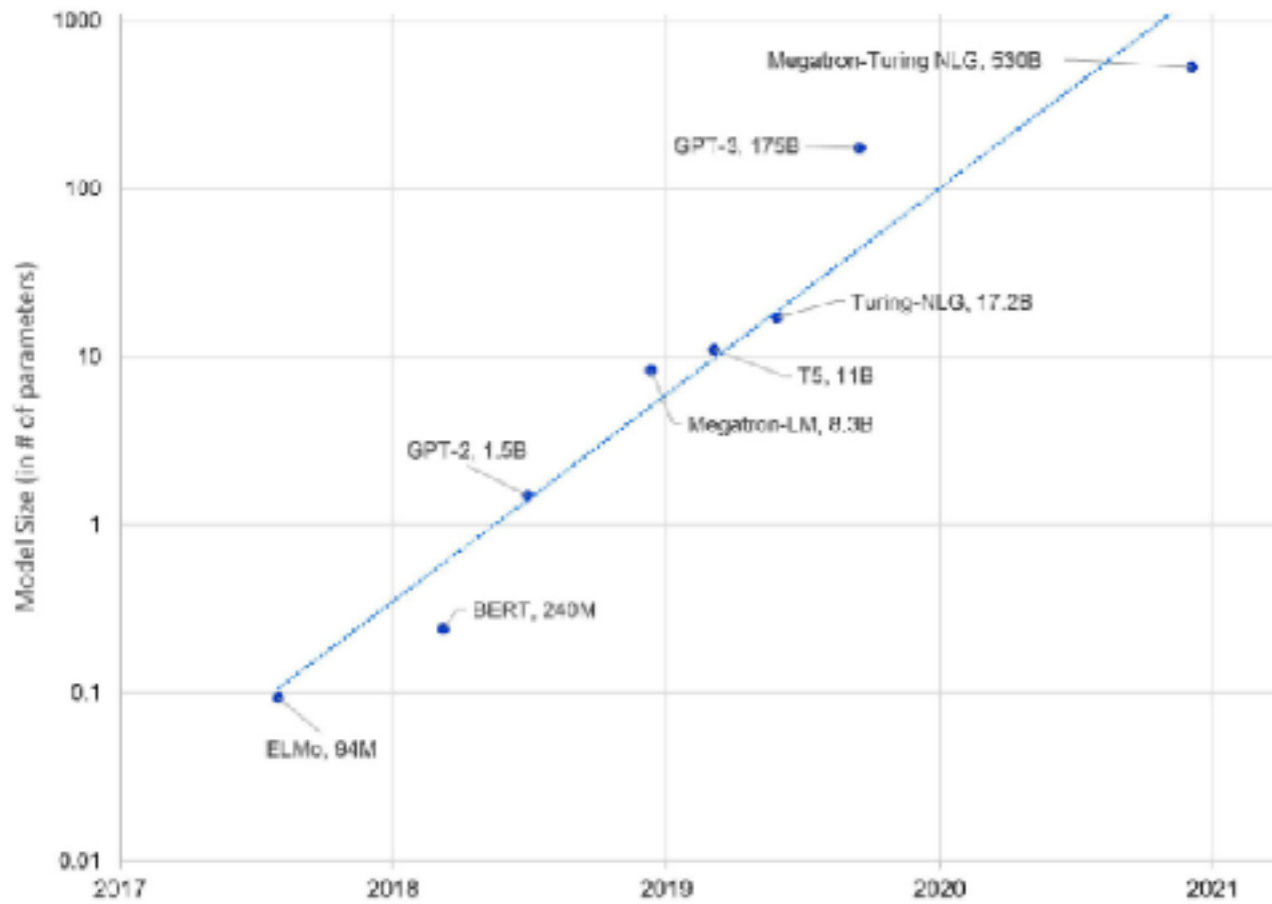
Rotation matrix

$$Rv = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}.$$

$$\underbrace{\begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & 0 & 0 & \dots & 0 & 0 \\ \sin m\theta_0 & \cos m\theta_0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos m\theta_1 & -\sin m\theta_1 & \dots & 0 & 0 \\ 0 & 0 & \sin m\theta_1 & \cos m\theta_1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos m\theta_{d/2-1} & -\sin m\theta_{d/2-1} \\ 0 & 0 & 0 & 0 & \dots & \sin m\theta_{d/2-1} & \cos m\theta_{d/2-1} \end{pmatrix}}_{R_m} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d-2} \\ q_{d-1} \end{pmatrix}$$

Emergent capabilities from predicting the next token at scale

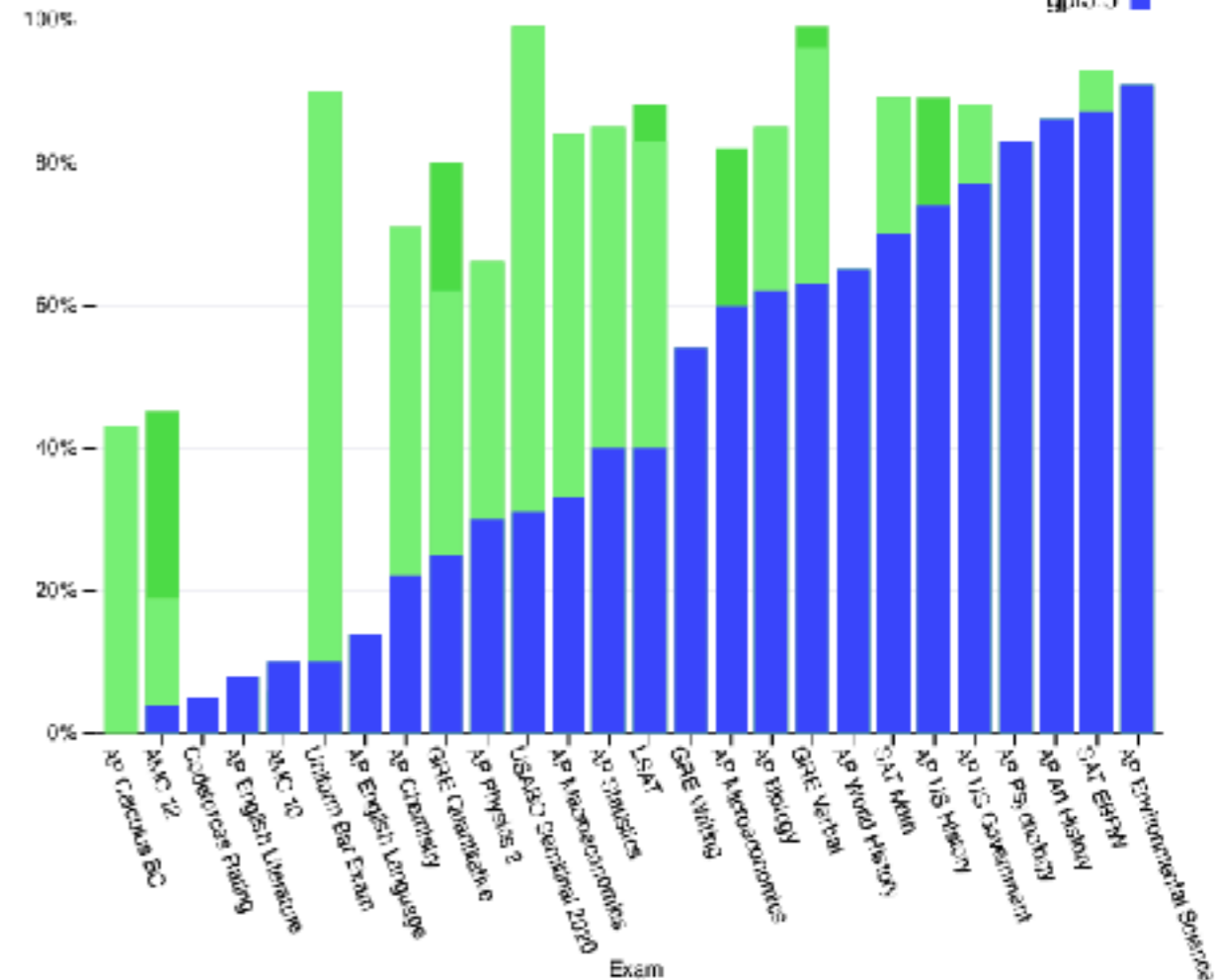
Growth in Large Language Model Size



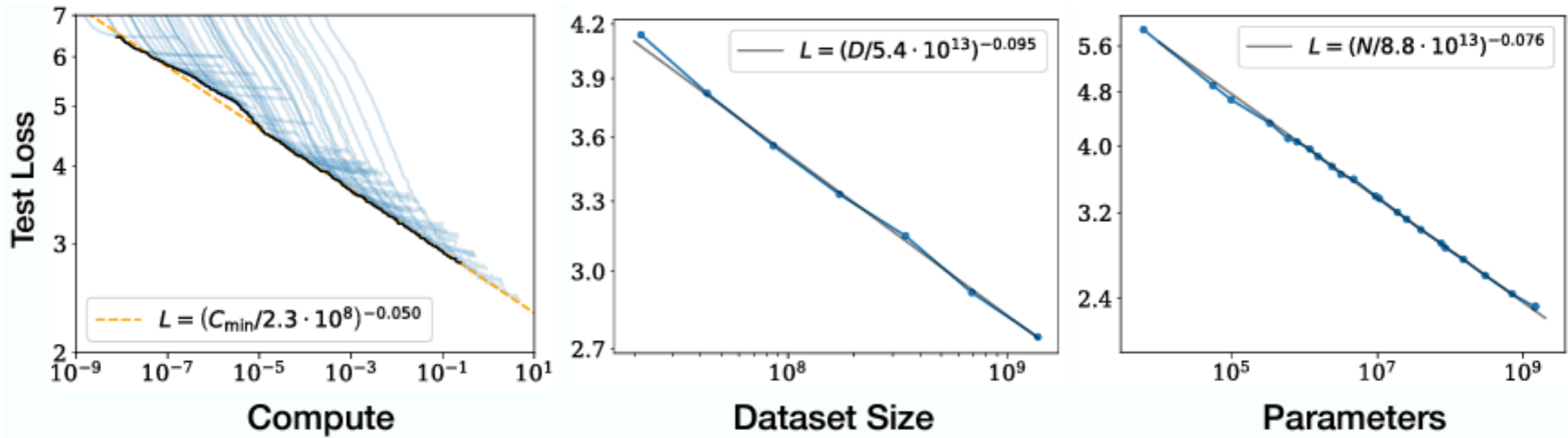
Growth in LLM model size, measured in # of parameters, from ELMo (94M) in 2017 to Megatron-Turing NLG (530B) in 2021. ©

Exam results (ordered by GPT-3.5 performance)

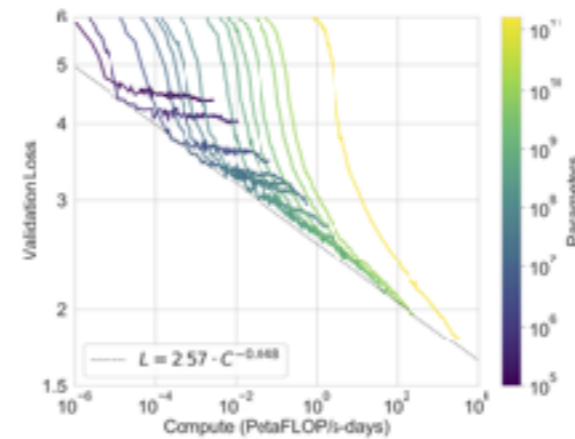
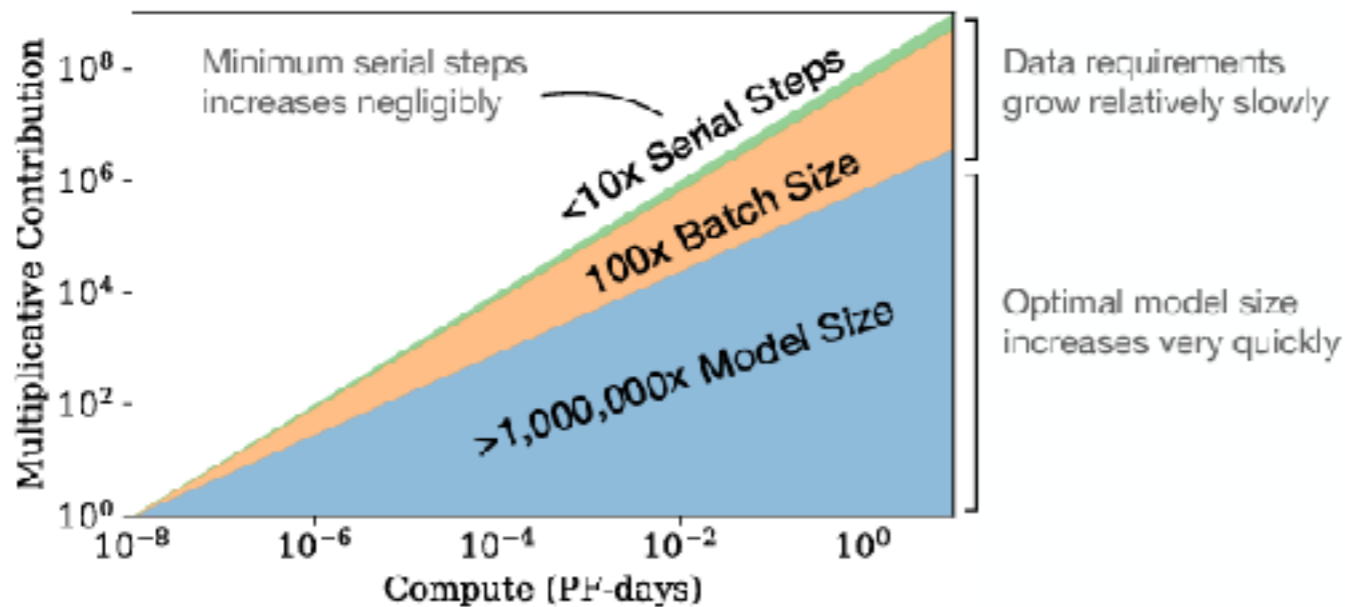
Estimated percentile lower bound (among test takers)



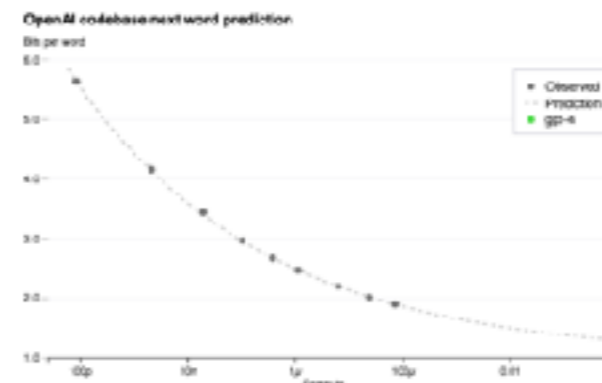
The vision of openAI : scale is all you need



Kaplan et al. 2020, Scaling Laws for Neural Language Models



GPT3 white paper, 2020



GPT4 white paper, 2023

Alignment of language model with custom training data and human feedback: ChatGPT

Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



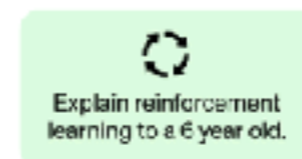
This data is used to fine-tune GPT-3.5 with supervised learning.



Step 2

Collect comparison data and train a reward model.

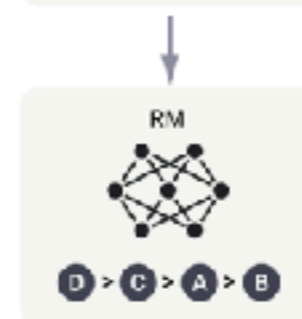
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



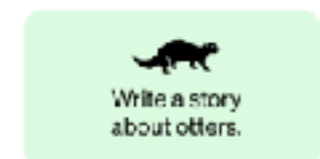
This data is used to train our reward model.



Step 3

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

A new prompt is sampled from the dataset.



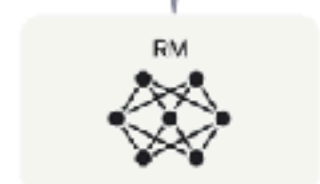
The PPO model is initialized from the supervised policy.



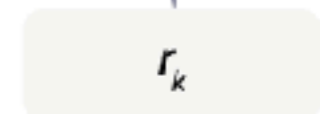
The policy generates an output.



The reward model calculates a reward for the output.

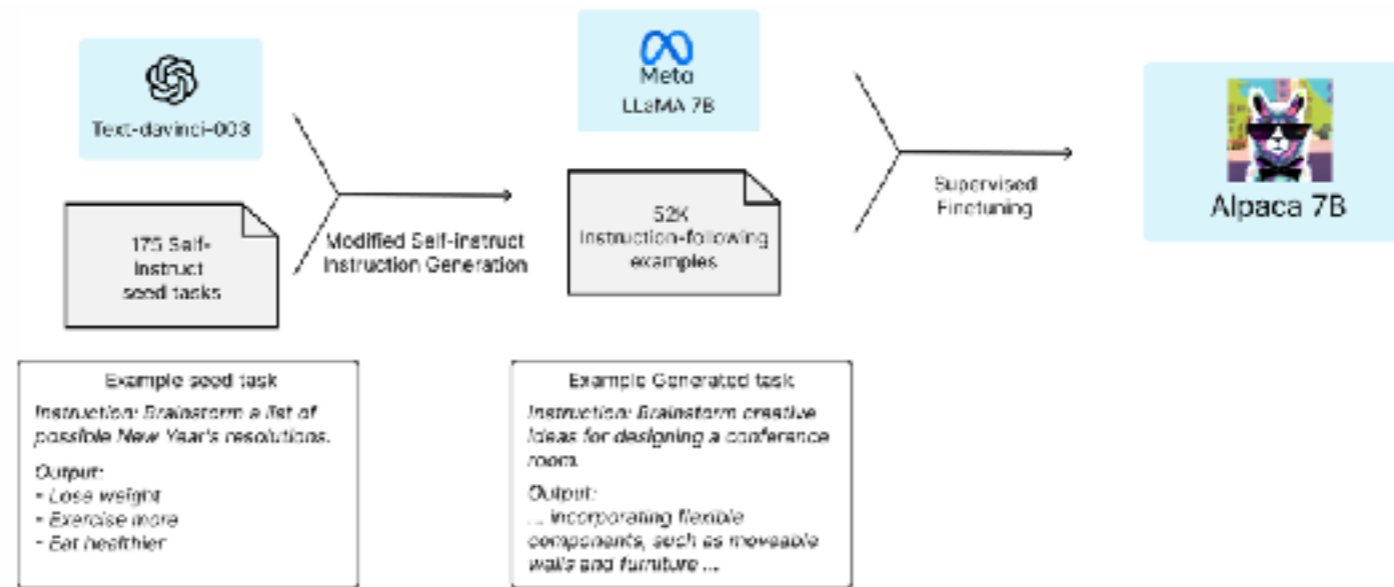


The reward is used to update the policy using PPO.

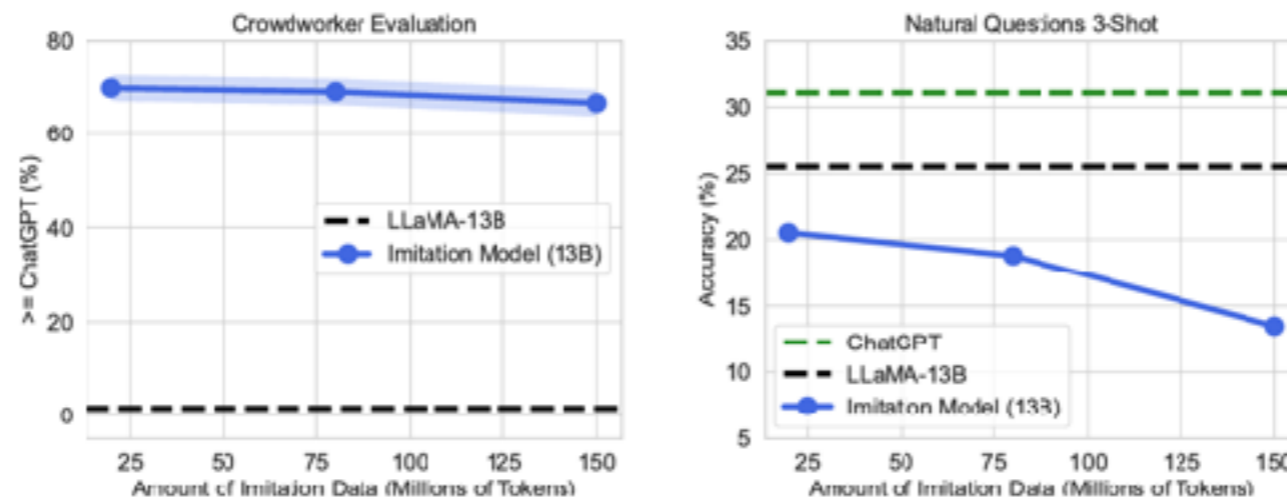


Is there hope for academics who don't have 10,000 GPUs?

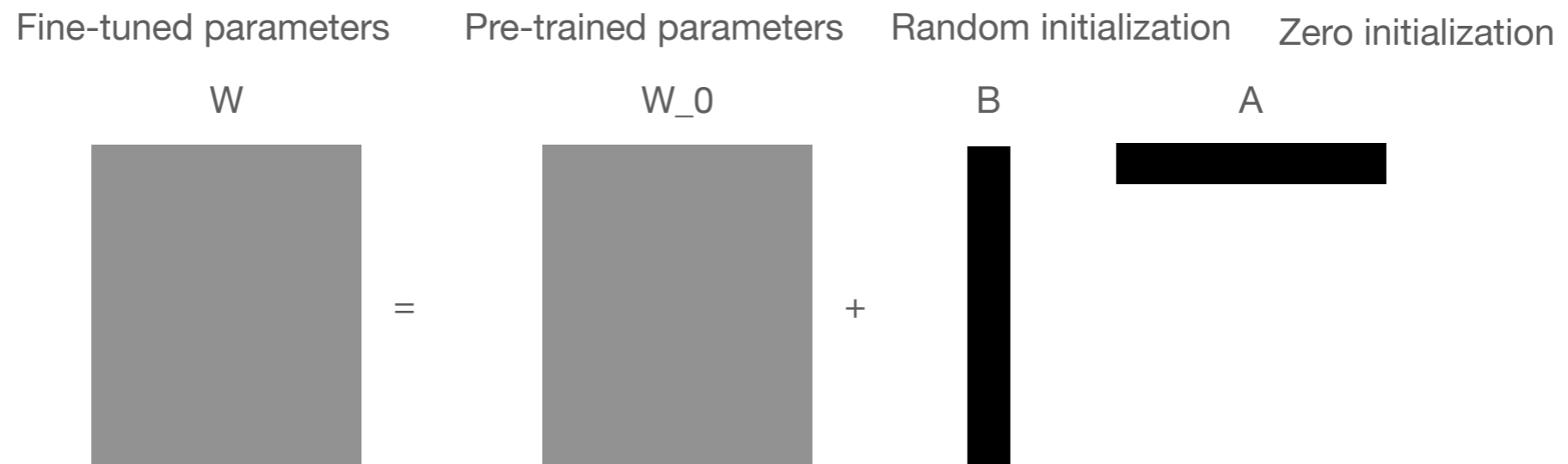
Alpaca: “\$600” ChaGPT: Open source LLM trained on dialogs generated by ChatGPT



“imitation models are adept at mimicking ChatGPT’s style but not its factuality”

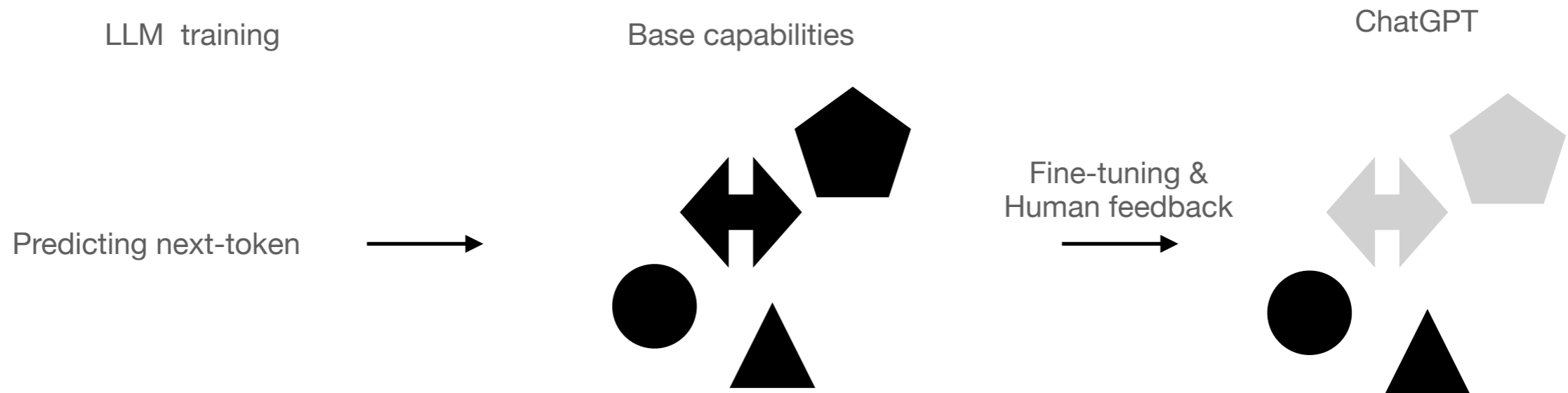


LoRA: fast finetuning of large models with efficient computation

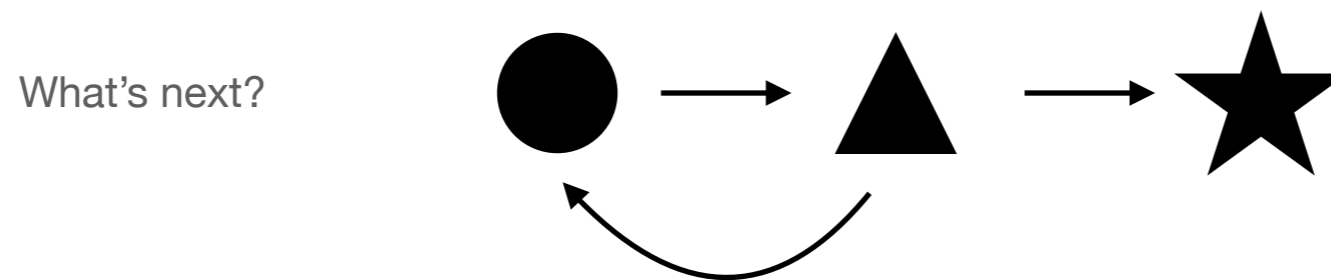


$$h = W_0x + \Delta Wx = W_0x + BAx$$

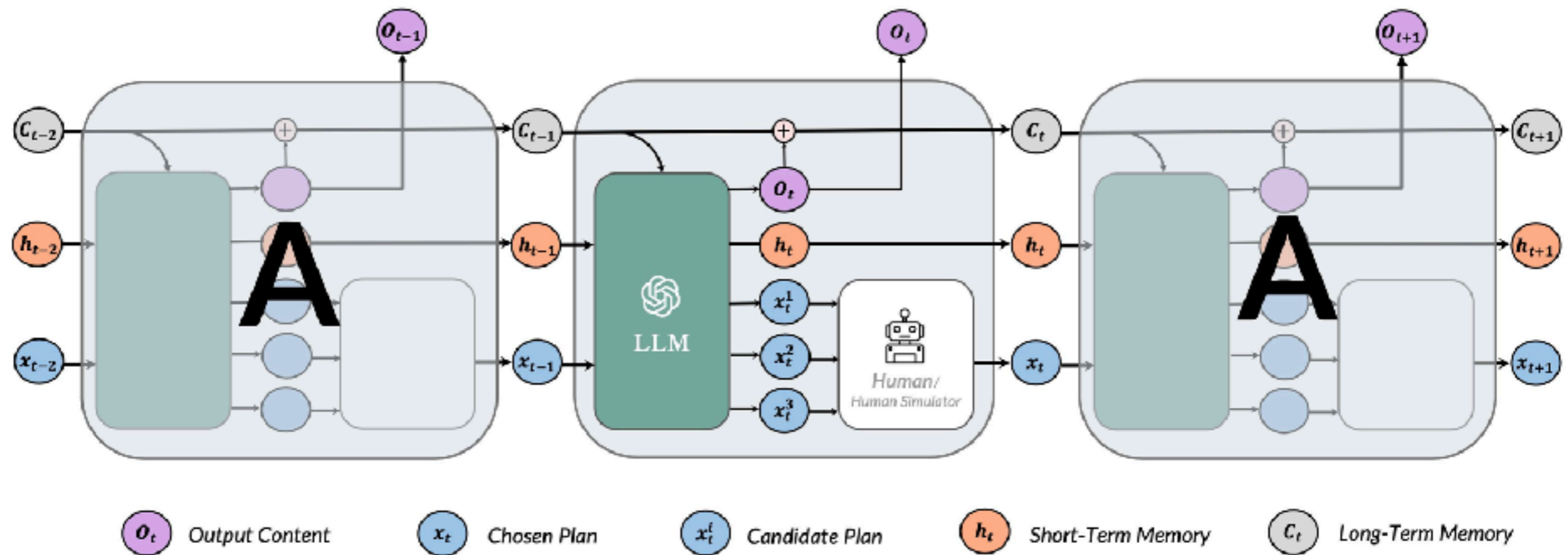
Will LLM lead to AGI?



Can artificial general intelligence be constructed from these capabilities?



“Programming” LLM to do complex tasks with language



<https://github.com/aiwaves-cn/RecurrentGPT>

“Programming” LLM to do complex tasks with language

I need you to help me write a novel. Now I give you a memory (a brief summary) of 400 words, you should use it to store the key content of what has been written so that you can keep track of very long context. For each time, I will give you your current memory (a brief summary of previous stories. You should use it to store the key content of what has been written so that you can keep track of very long context!), the previously written paragraph, and instructions on what to write in the next paragraph. I need you to write:

1. **Output Paragraph:** the next paragraph of the novel. The output paragraph should contain around 20 sentences and should follow the input instructions.
2. **Output Memory:** The updated memory. You should first explain which sentences in the input memory are no longer necessary and why, and then explain what needs to be added into the memory and why. After that you should write the updated memory. The updated memory should be similar to the input memory except the parts you previously thought that should be deleted or added. The updated memory should only store key information. The updated memory should never exceed 20 sentences!
3. **Output Instruction:** instructions of what to write next (after what you have written). You should output 3 different instructions, each is a possible interesting continuation of the story. Each output instruction should contain around 5 sentences

Here are the inputs:

Input Memory:
{short_memory}

Input Paragraph:
{input_paragraph}

Input Instruction:
{input_instruction}

Input Related Paragraphs:
{input_long_term_memory}

Now start writing, organize your output by strictly following the output format as below:

Output Paragraph:
<string of output paragraph>, around 20 sentences.

Output Memory:
Rational: <string that explain how to update the memory>;
Updated Memory: <string of updated memory>, around 10 to 20 sentences

Output Instruction:
Instruction 1: <content for instruction 1>, around 5 sentences
Instruction 2: <content for instruction 2>, around 5 sentences
Instruction 3: <content for instruction 3>, around 5 sentences

Very important: The updated memory should only store key information. The updated memory should never contain over 500 words! Finally, remember that you are writing a novel. Write like a novelist and do not move too fast when writing the output instructions for the next paragraph. Remember that the chapter will contain over 10 paragraphs and the novel will contain over 100 chapters. And this is just the beginning. Just write some interesting stuffs that will happen next. Also, think about what plot can be attractive for common readers when writing output instructions. You should first explain which sentences in the input memory are no longer necessary and why, and then explain what needs to be added into the memory and why. After that, you start rewrite the input memory to get the updated memory.

Timestep $t - 1$

Timestep t

Short-Term Memory h_{t-1} : As a renowned interstellar explorer, Elias had traveled to countless planets and encountered unimaginable wonders...

Chosen Plan x_{t-1} : Elias, a renowned interstellar explorer, embarks on a new mission to explore a distant planet that has been recently discovered. The planet is believed to have a unique ecosystem and Elias is eager to uncover its secrets.

Output Content O_t : Elias spent days studying the artifact, trying to unlock its secrets. He worked tirelessly, using all of his knowledge and skills to decipher the hieroglyphics and understand the technology. As he delved deeper into the artifact's mysteries, he began to realize that it was not of this world. It was a relic from a long-lost civilization that had once ruled the galaxy. Elias knew that he had stumbled upon something truly extraordinary, but he also knew that it was dangerous.
As he worked on the artifact, Elias began to notice that the creatures and the aliens were acting strangely. They seemed to be watching him, studying him, and waiting for something. Elias couldn't shake the feeling that something was not right. He knew that he had to be careful, but he also knew that he couldn't stop now. He was too close to unlocking the artifact's secrets.

Long-Term Memory C_t :

1. Elias sat in the cockpit of his spaceship, staring out at the vast expanse of space before him. He had been traveling for months, but the excitement of discovering a new planet kept him going. As he approached the planet, he couldn't help but feel a sense of anticipation. What wonders lay ahead?
2. The planet was unlike anything Elias had ever seen before. The sky was a deep shade of purple, and the trees were a vibrant shade of blue. As he stepped out of his spaceship, he was greeted by a warm breeze that carried the scent of exotic flowers. Elias couldn't wait to explore this new world.
3. As Elias continued to explore the planet, he couldn't help but feel a sense of wonder and excitement. The creatures he encountered were unlike anything he had ever seen before. Some were small and furry, while others were massive and had multiple limbs. Elias was fascinated by their unique features and behavior. He spent hours observing them, taking notes, and sketching them in his journal.
4. As he ventured deeper into the planet, Elias stumbled upon ancient ruins that suggested that there was once an advanced civilization on the planet. The ruins were unlike anything he had ever seen before. The architecture was intricate and beautiful, and the technology was far more advanced than anything he had ever encountered.

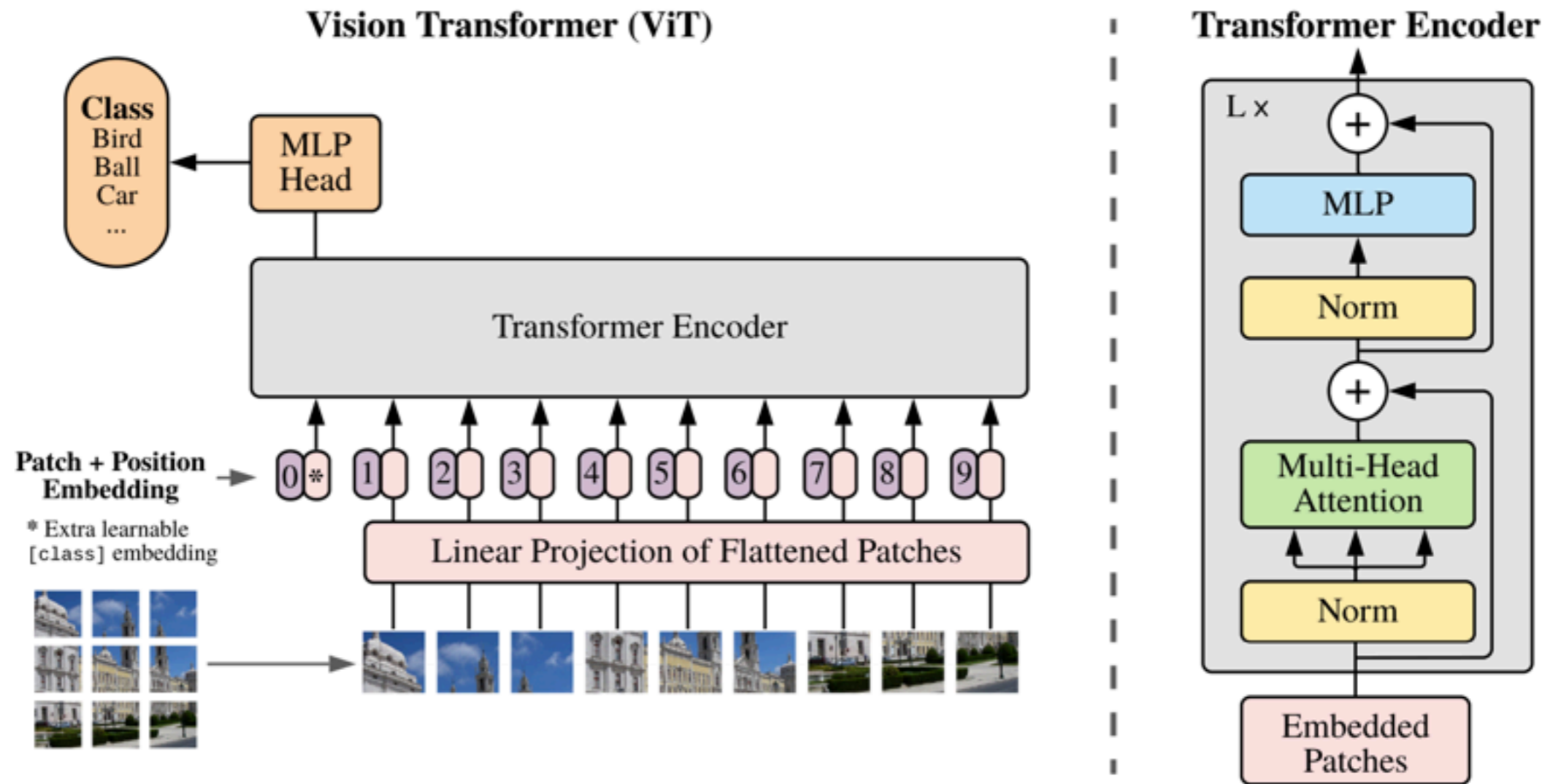
Short-Term Memory h_t : Elias and his team discovered that the artifact they had found was part of a larger network of artifacts scattered throughout the galaxy. Elias and his team were determined to find the other artifacts and stop the powerful entity that sought to use them for their own gain....

Candidate Plan x_t^1 : Elias and his team must race against time to find the next artifact before the rival team does. They travel to a planet that is home to a dangerous creature that guards the artifact. They must outsmart the creature and retrieve the artifact before the rival team arrives.

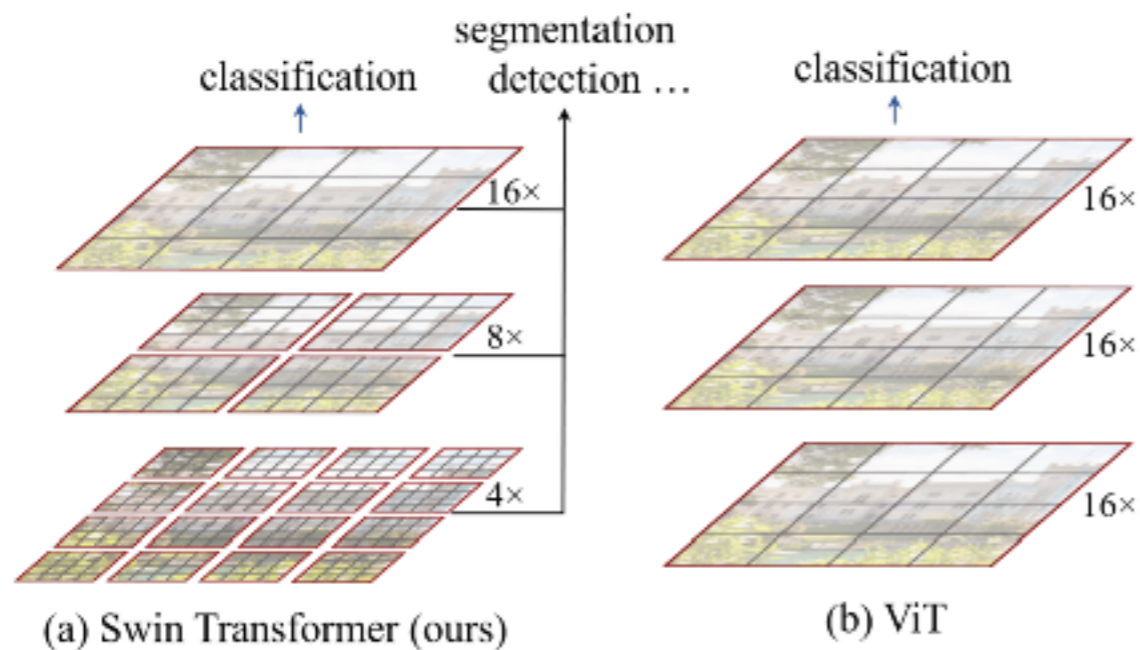
Candidate Plan x_t^2 : Elias and his team discover that the next artifact is located on a planet that is home to a peaceful alien race. However, they soon realize that the alien race is being threatened by a group of space pirates. They must team up with the aliens to defeat the space pirates and retrieve the artifact.

Candidate Plan x_t^3 : Elias and his team receive a distress signal from a planet that is being attacked by a powerful entity. They discover that the entity is after an artifact that is located on the planet. Elias and his team must fight against the entity and retrieve the artifact before it's too late.

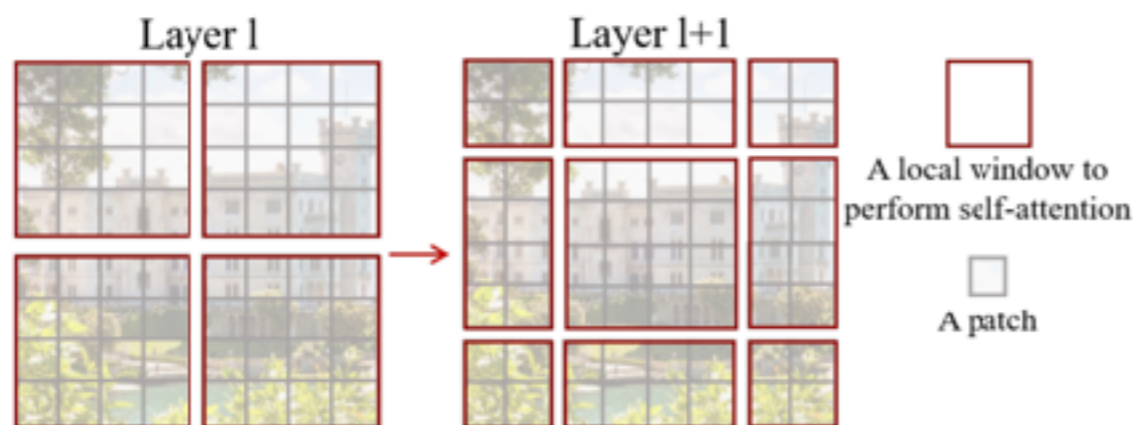
Vision transformer for image recognition



Swin transformer: improving ViT



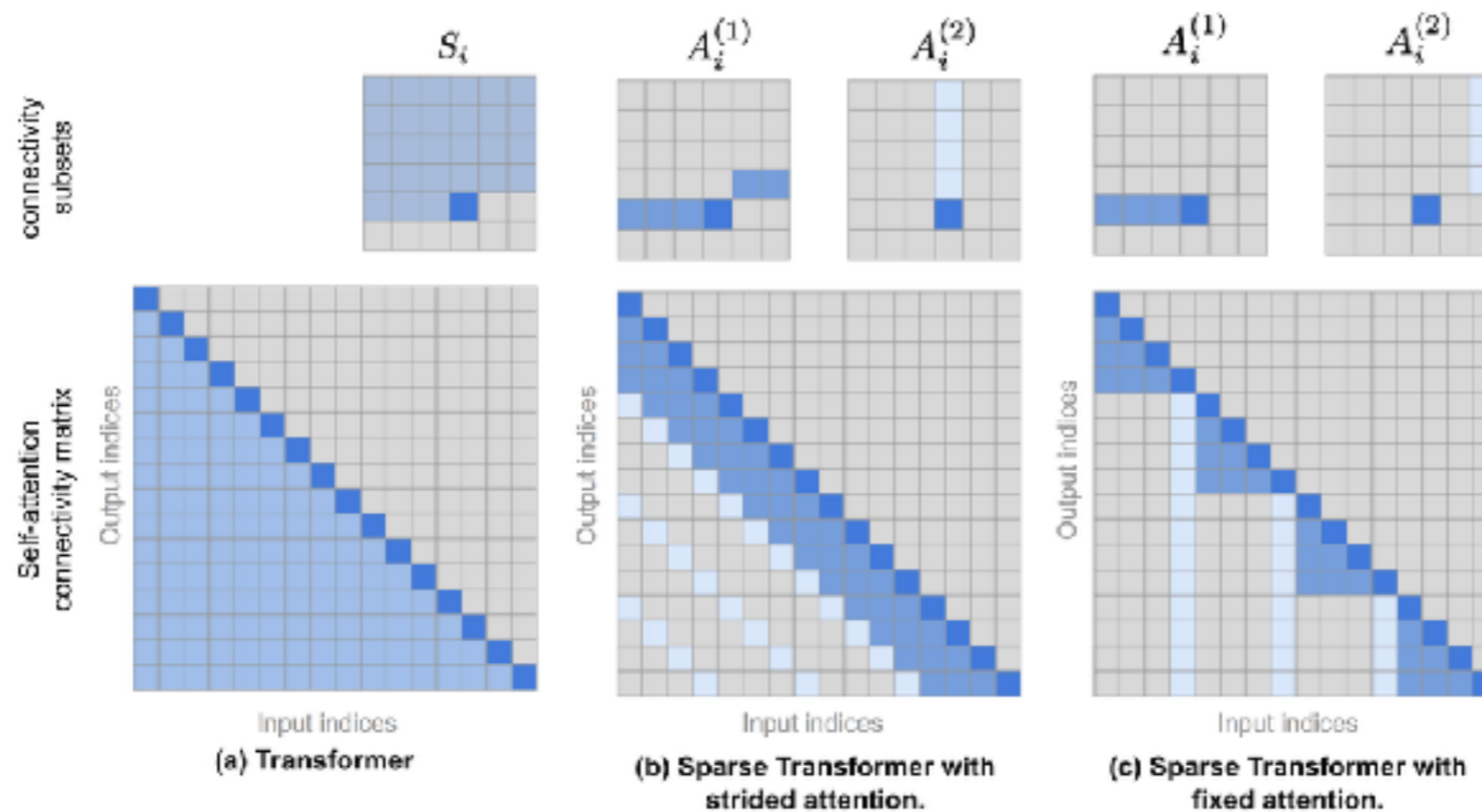
Hierarchical structure



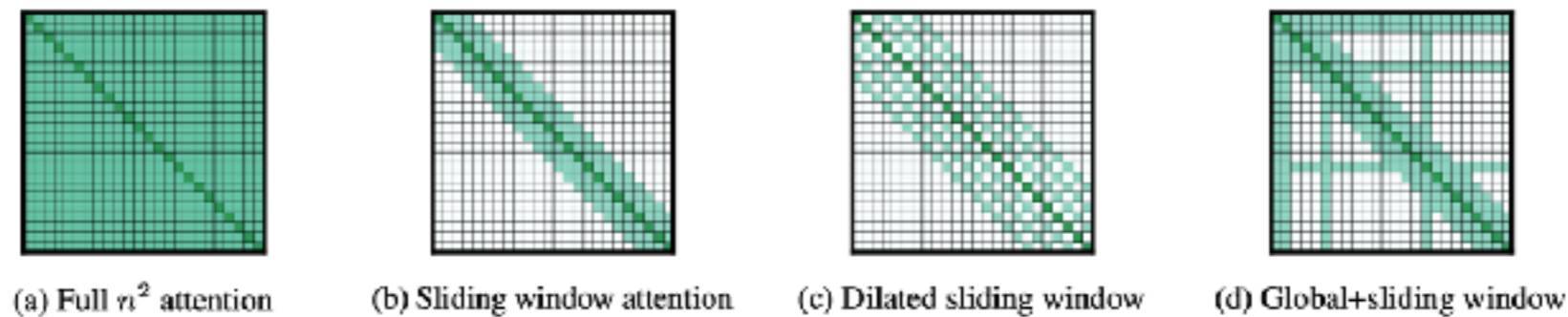
Shifted non-overlapping windows
(Swin means shifted windows)

Scalable transformer for long sequences

Sparse factorized attention

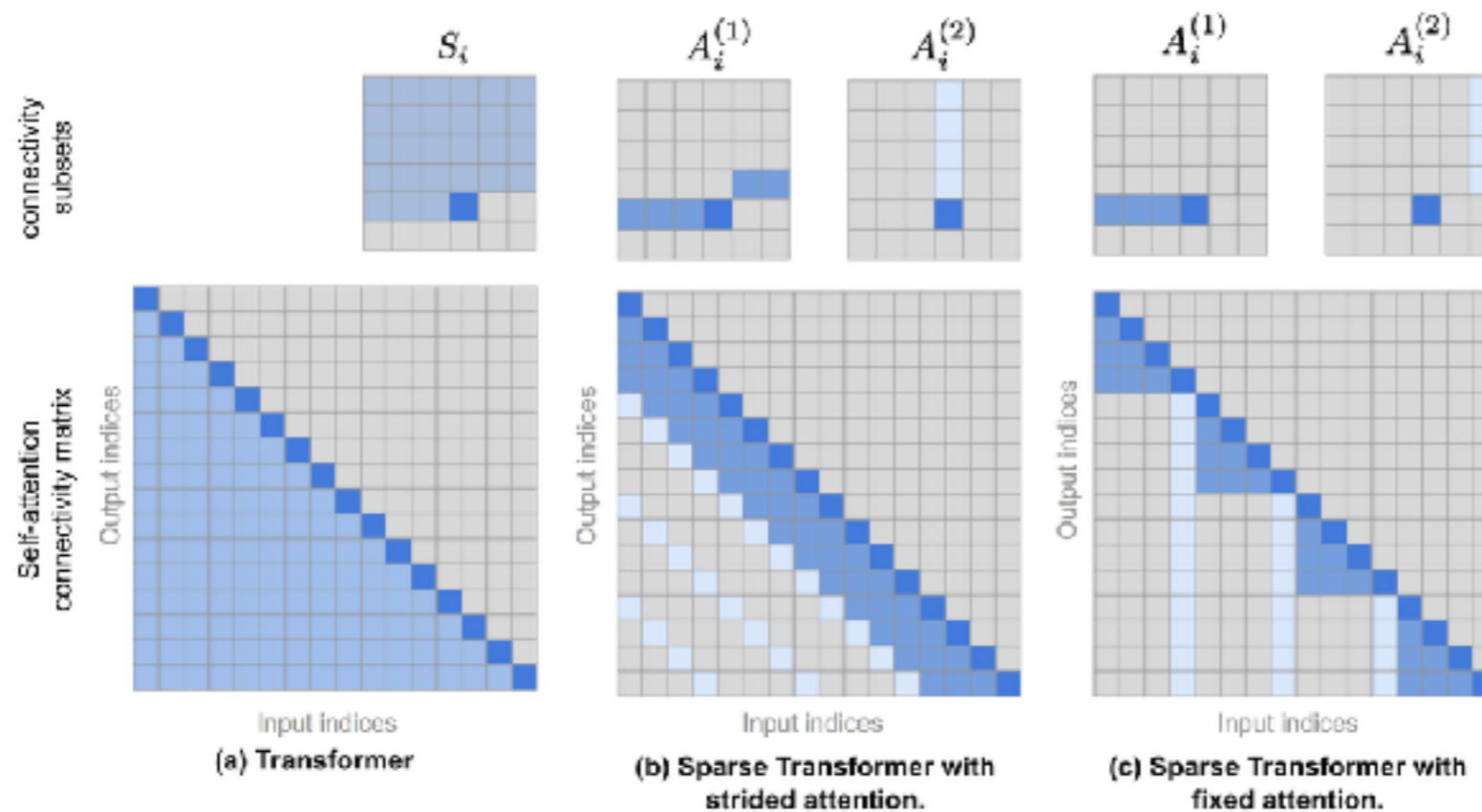


Generating Long Sequences with Sparse Transformers

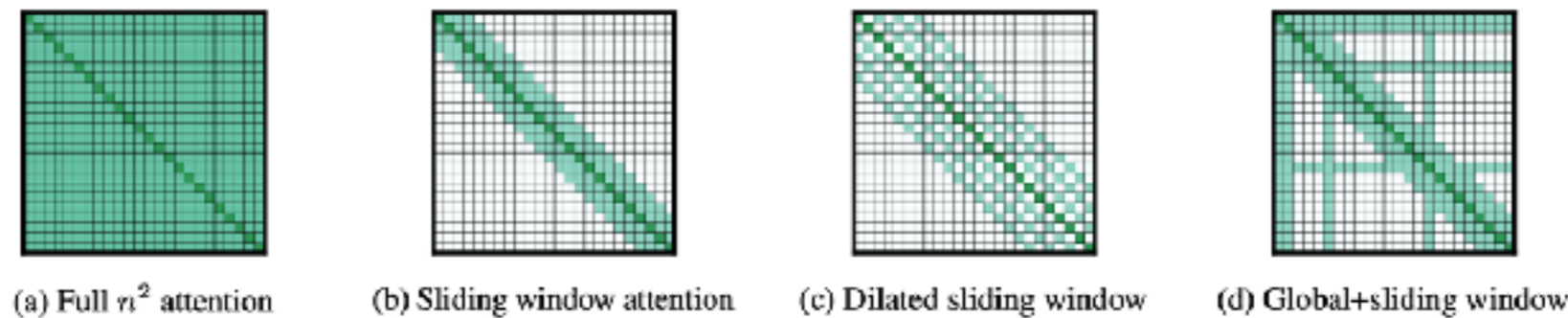


Scalable transformer for long sequences

Sparse factorized attention



Generating Long Sequences with Sparse Transformers



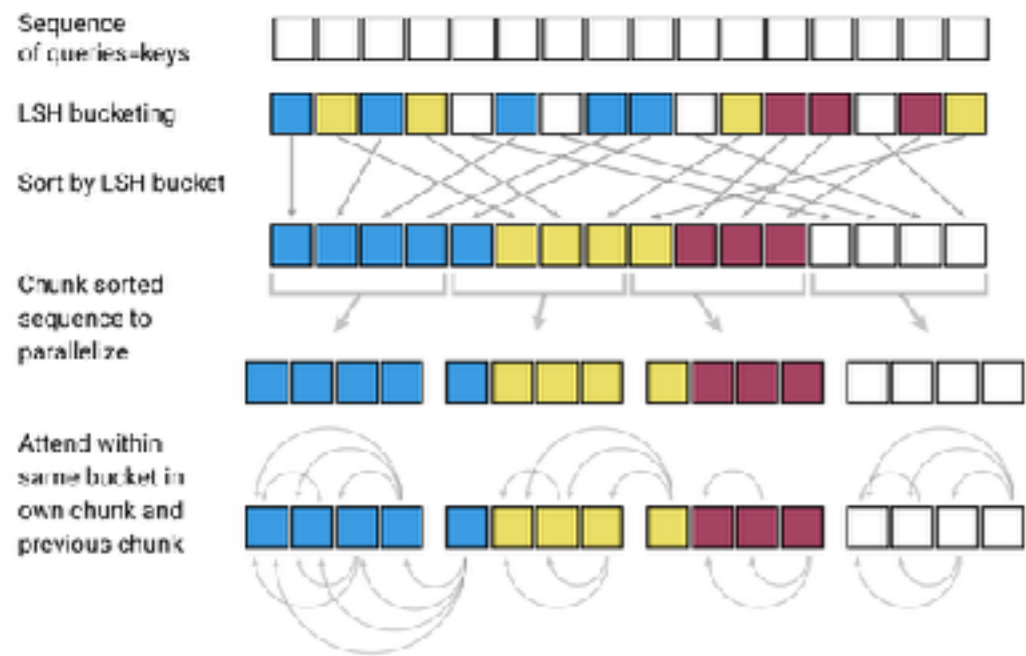
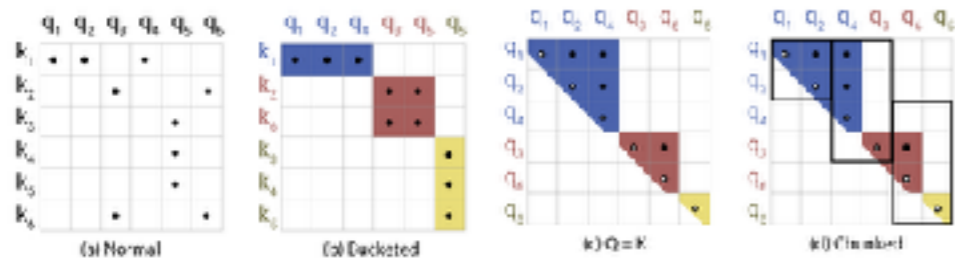
Scalable transformer for long sequences

Restrict attention to be within buckets (or within nearby buckets)

Reformer (LSH)

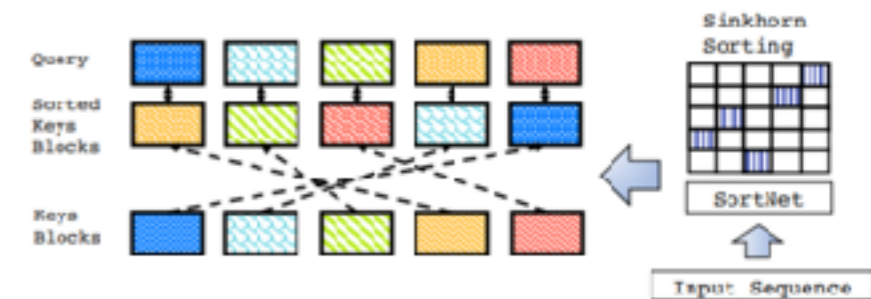
Routing transformer (k-means)

Sinkhorn transformer (Sinkhorn Sorting)



(c) Routing attention

Sorting (learned-ordering) as matrix multiplication

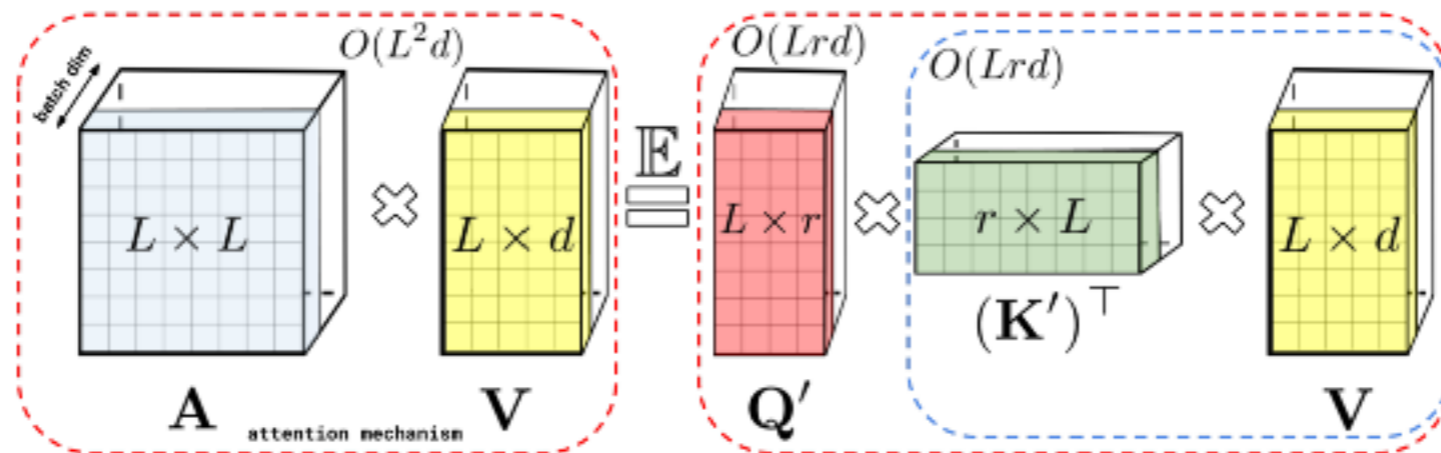


Sinkhorn-knopp algorithm output a sorting matrix-like matrix via differentiable iterations

Blocks are still predefined, algorithm is still n^2 wrt number of blocks and only determines neighbor of the blocks

Scalable transformer for long sequences

Low-rank approximation of attention (FAVOR+)



Kernel $\mathbf{K}(\mathbf{x}, \mathbf{y}) = \mathbb{E}[\phi(\mathbf{x})^\top \phi(\mathbf{y})].$

Feature map decomposition (can need infinite-dimensions though)

Most kernels can be approximated with random feature maps where w is random variable

$$\phi(\mathbf{x}) = \frac{h(\mathbf{x})}{\sqrt{m}} (f_1(\omega_1^\top \mathbf{x}), \dots, f_1(\omega_m^\top \mathbf{x}), \dots, f_l(\omega_1^\top \mathbf{x}), \dots, f_l(\omega_m^\top \mathbf{x})),$$

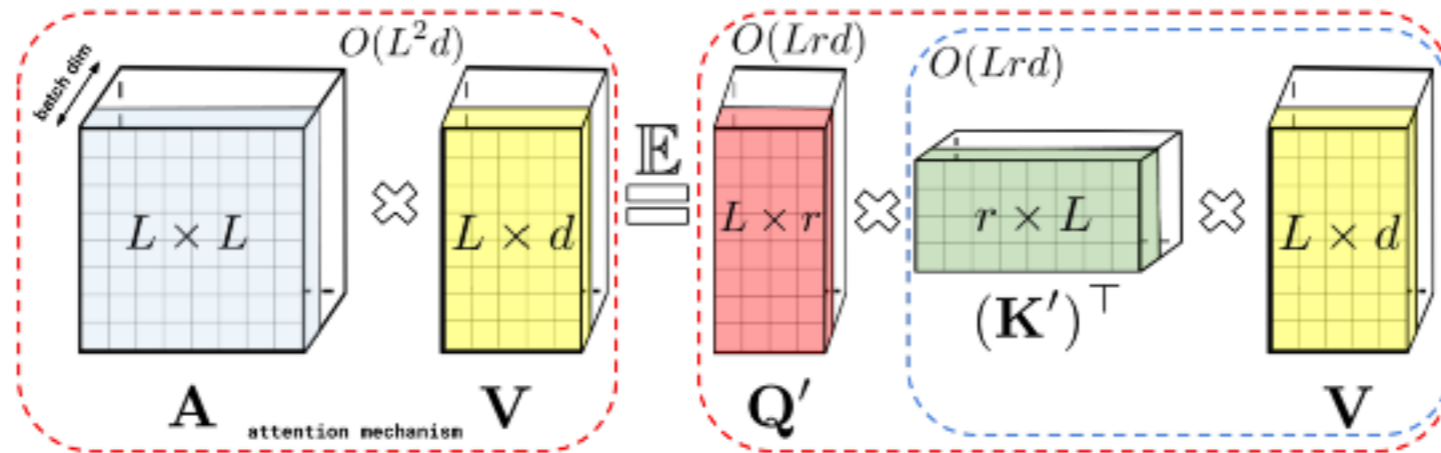
f

w

FAVOR+: Use **Nonlinear, random orthogonal** feature maps to replace full attention

Scalable transformer for long sequences

Low-rank approximation of attention (FAVOR+)



$$\text{SM}(\mathbf{x}, \mathbf{y}) = \exp(\mathbf{x}^\top \mathbf{y})$$

$$\Lambda = \exp\left(-\frac{\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2}{2}\right)$$

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\mathbf{z} = \mathbf{x} + \mathbf{y}$$

$$\text{SM}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{\omega \sim \mathcal{N}(0, \mathbf{I}_d)} \left[\exp\left(\omega^\top \mathbf{x} - \frac{\|\mathbf{x}\|^2}{2}\right) \exp\left(\omega^\top \mathbf{y} - \frac{\|\mathbf{y}\|^2}{2}\right) \right] = \Lambda \mathbb{E}_{\omega \sim \mathcal{N}(0, \mathbf{I}_d)} \cosh(\omega^\top \mathbf{z})$$

Proof:

$$\text{SM}(\mathbf{x}, \mathbf{y}) = \exp(\mathbf{x}^\top \mathbf{y}) = \exp(-\|\mathbf{x}\|^2/2) \cdot \exp(\|\mathbf{x} + \mathbf{y}\|^2/2) \cdot \exp(-\|\mathbf{y}\|^2/2).$$

$$\exp(\|\mathbf{x} + \mathbf{y}\|^2/2) = (2\pi)^{-d/2} \exp(\|\mathbf{x} + \mathbf{y}\|^2/2) \int \exp(-\|\mathbf{w} - (\mathbf{x} + \mathbf{y})\|^2/2) d\mathbf{w}$$

$$= (2\pi)^{-d/2} \int \exp(-\|\mathbf{w}\|^2/2 + \mathbf{w}^\top (\mathbf{x} + \mathbf{y}) - \|\mathbf{x} + \mathbf{y}\|^2/2 + \|\mathbf{x} + \mathbf{y}\|^2/2) d\mathbf{w}$$

$$= (2\pi)^{-d/2} \int \exp(-\|\mathbf{w}\|^2/2 + \mathbf{w}^\top (\mathbf{x} + \mathbf{y})) d\mathbf{w}$$

$$= (2\pi)^{-d/2} \int \exp(-\|\mathbf{w}\|^2/2) \cdot \exp(\mathbf{w}^\top \mathbf{x}) \cdot \exp(\mathbf{w}^\top \mathbf{y}) d\mathbf{w}$$

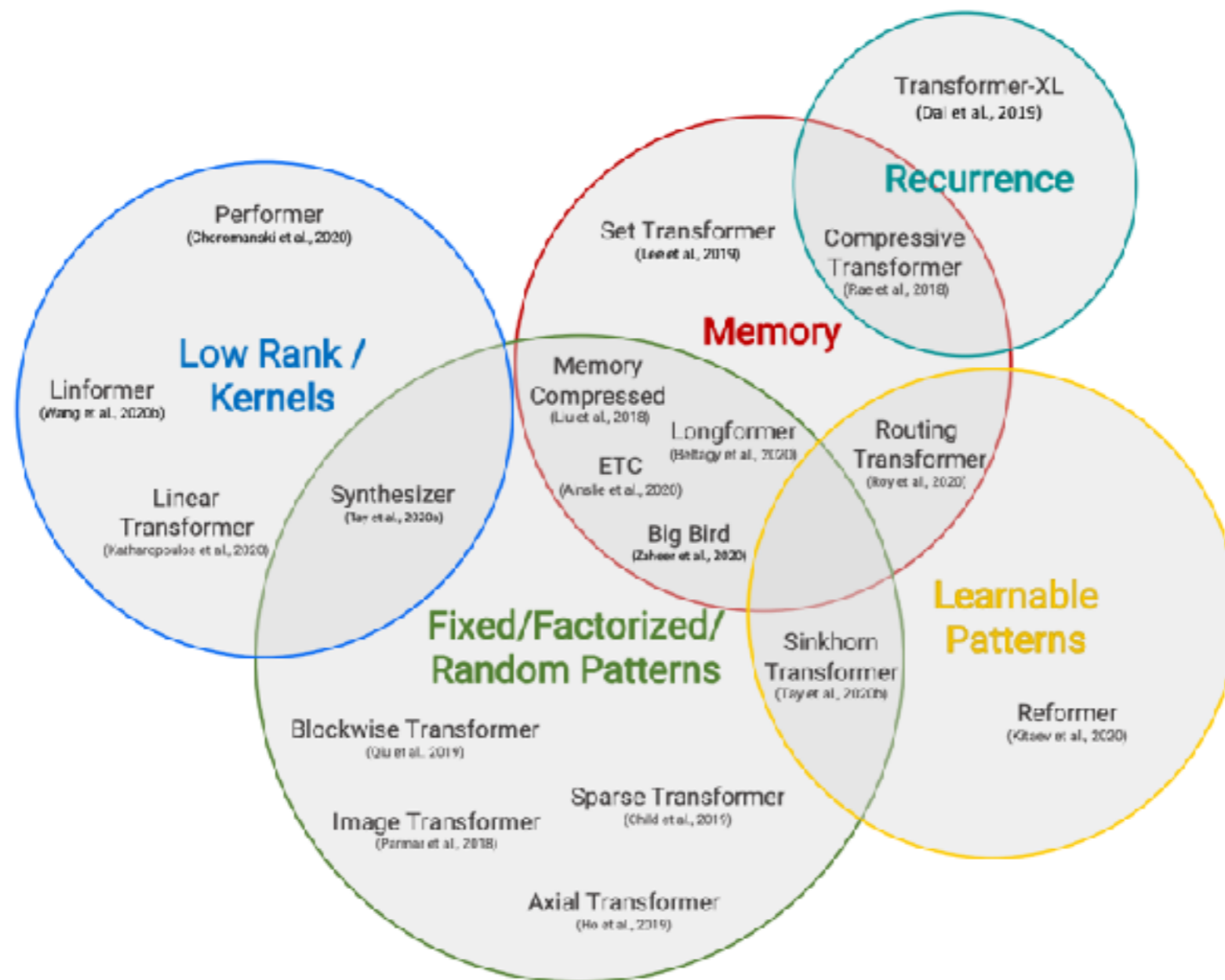
$$= \mathbb{E}_{\omega \sim \mathcal{N}(0, \mathbf{I}_d)} [\exp(\omega^\top \mathbf{x}) \cdot \exp(\omega^\top \mathbf{y})].$$

Exp can be replaced with ReLU for better performance in practice

No free lunch?: this approximation can be inefficient in high dimensions (r required $\gg L$)

Despite so, this attention-free formulation can be an alternative to transformer (with learnable instead of random w)

Summary of existing “efficient” transformers



A Hopfield-network interpretation of transformer

Classical Hopfield network:
Store and retrieval of binary patterns

$$W = \sum_i^N x_i x_i^T$$

$$\xi^{t+1} = \text{sgn}(W\xi^t - b)$$

$$E = -\frac{1}{2}\xi^T W\xi + \xi^T b$$



Fixed-point update

Discrete modern Hopfield network:

$$E = -\sum_{i=1}^N \exp(x_i^T \xi)$$

$$\text{update } \xi^{\text{new}}[l] = \text{sgn} \left[-E(\xi^{(l+)}) + E(\xi^{(l-)}) \right]$$

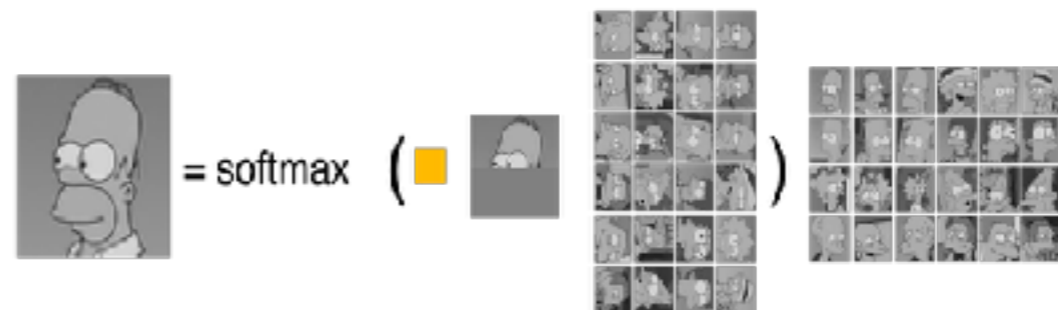
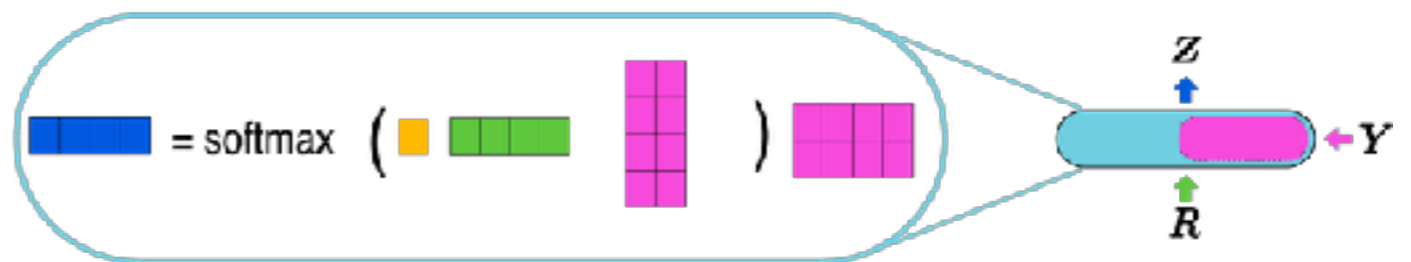
Continuous Hopfield network:

$$E = -\text{lse}(\beta, X^T \xi) + \frac{1}{2}\xi^T \xi$$

$$\text{update } \xi^{t+1} = X \text{softmax}(\beta X^T \xi^t)$$

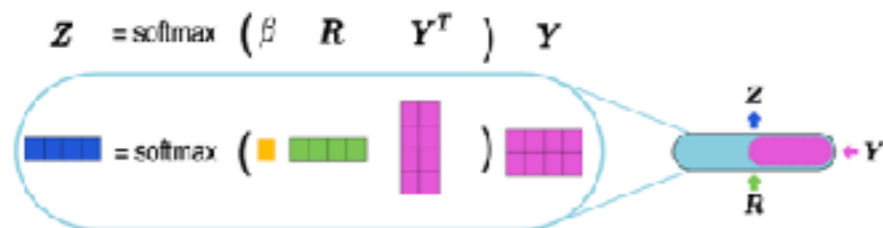
Query Key Value

$$Z = \text{softmax} \left(\beta \quad R \quad Y^T \right) Y$$

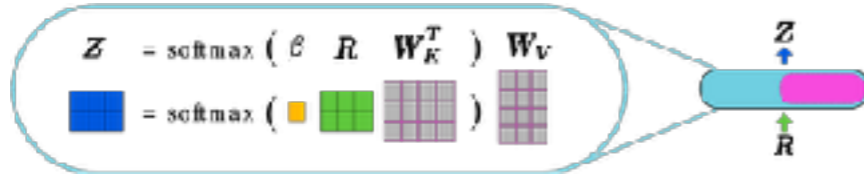


A Hopfield-network interpretation of transformer

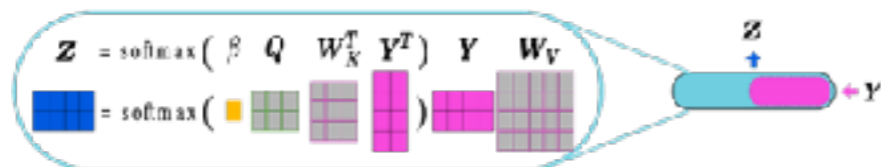
No internal parameters (similar pattern retrieval)



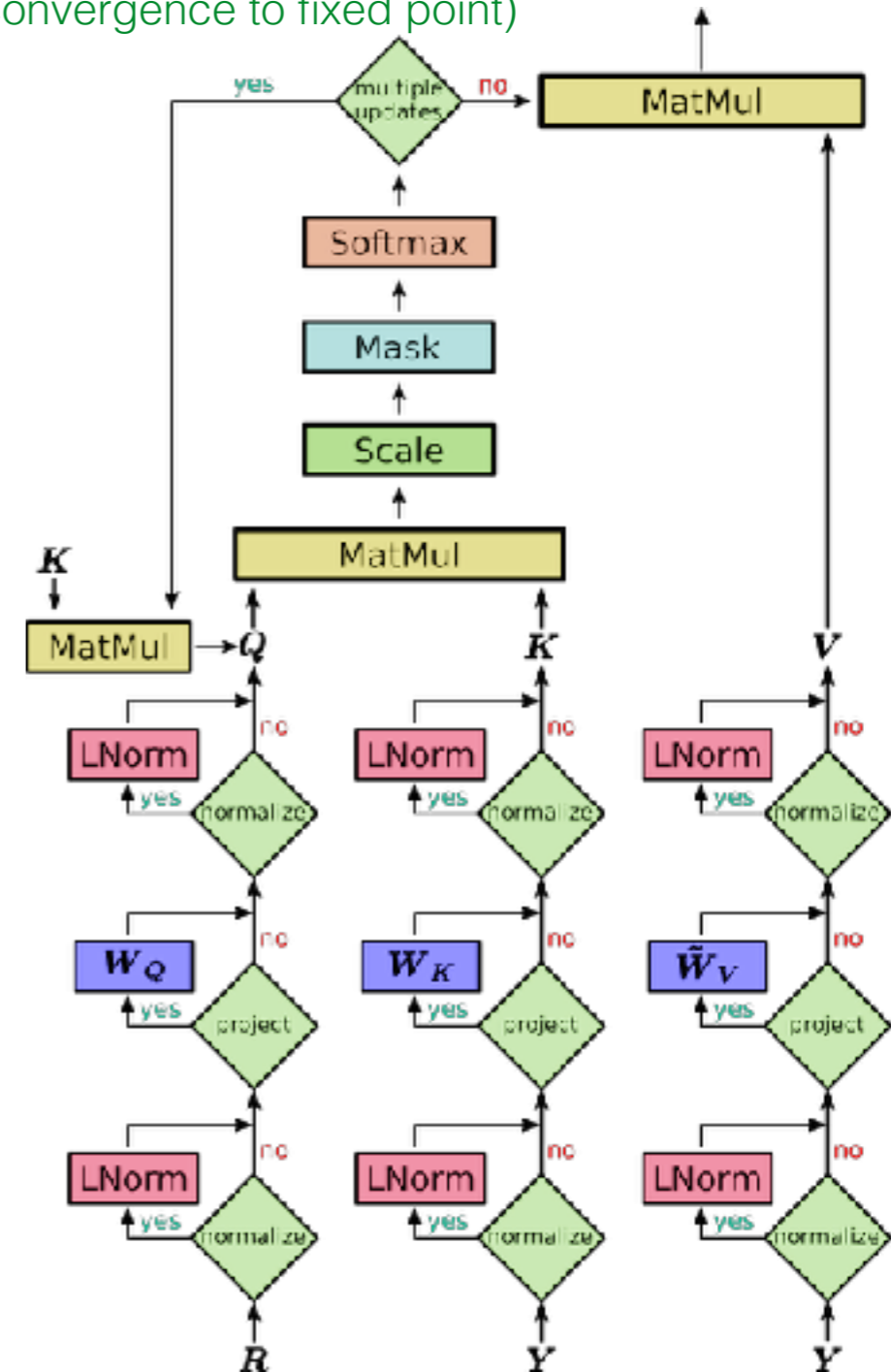
Stored patterns (key) and projection (value) are parameters



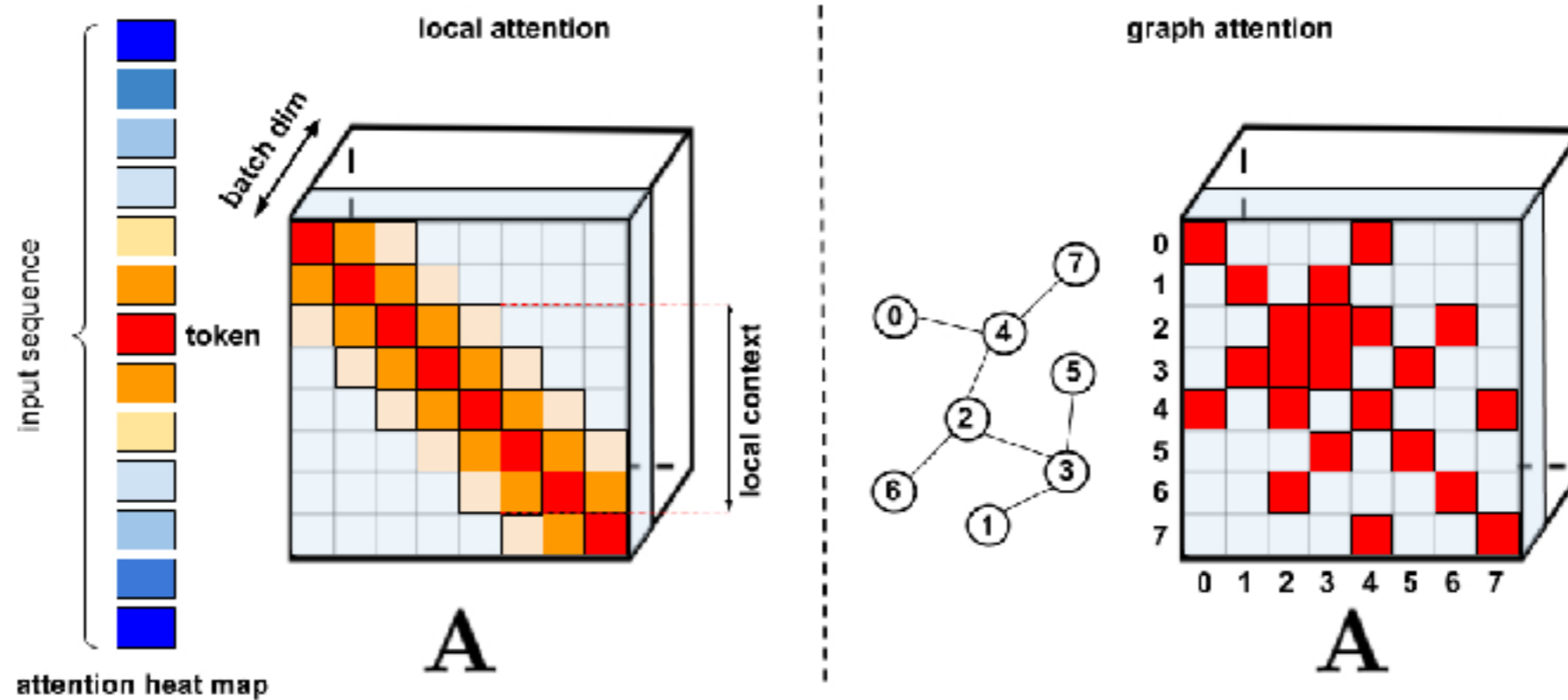
Query and projection are parameters



Motivating multi-step update
(better convergence to fixed point)



From transformer to graph network

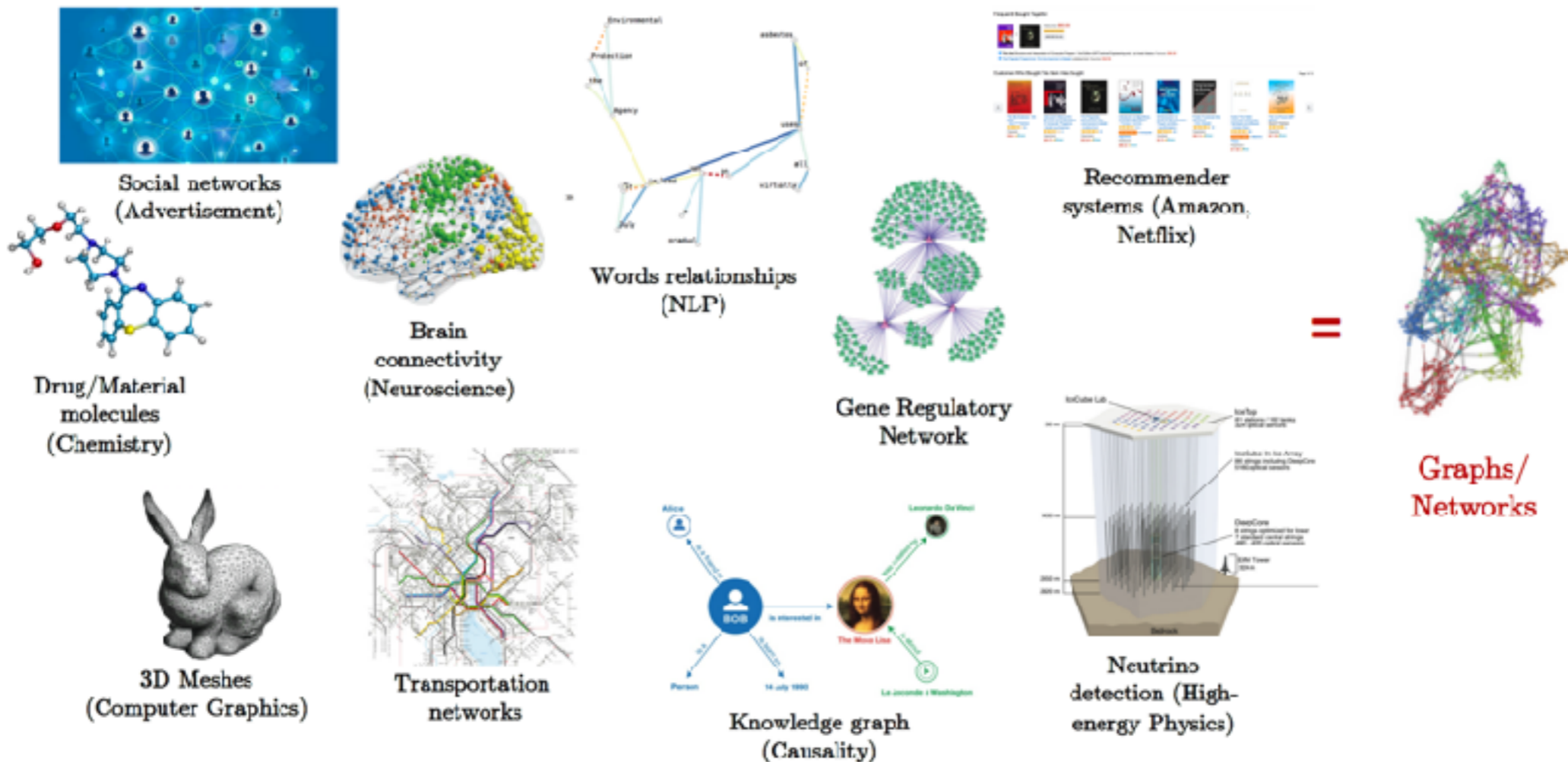


<https://ai.googleblog.com/2020/10/rethinking-attention-with-performers.html>

Graph Neural Network

- Graph is an extremely flexible abstraction for both data and models

Graph-structured data



A general form of Graph Network (node-centric)

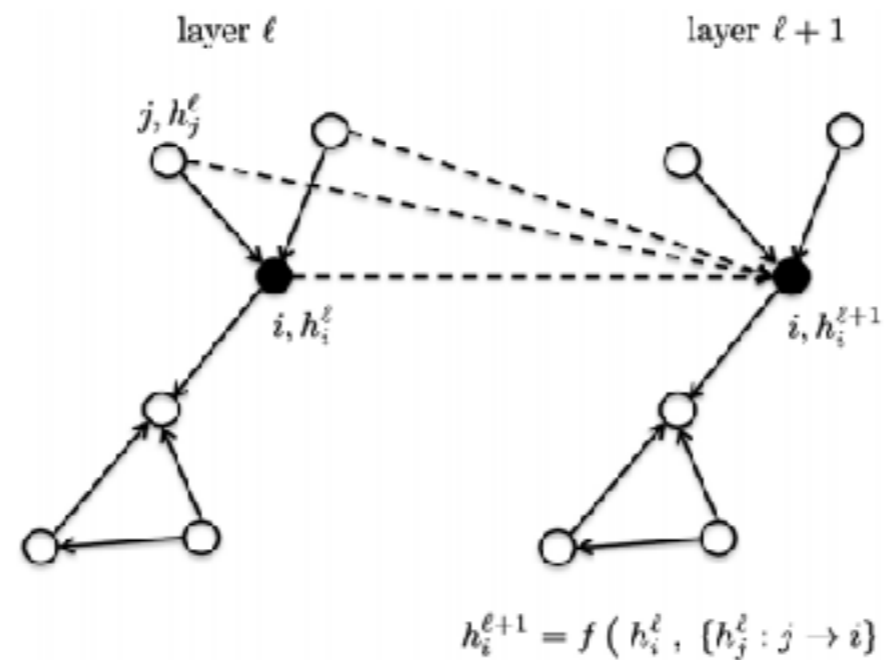


Figure 5: A generic graph neural network layer. Figure adapted from [11].

A general form of Graph Network (node-centric)

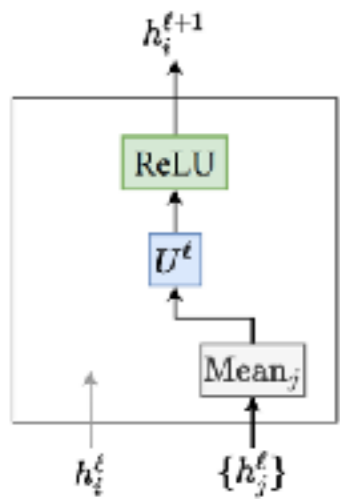


Figure 6: GCN Layer

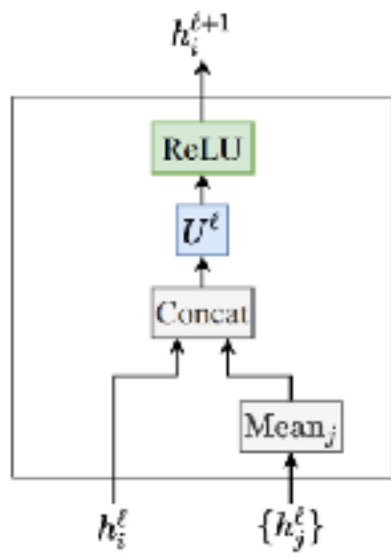


Figure 7: GraphSage Layer

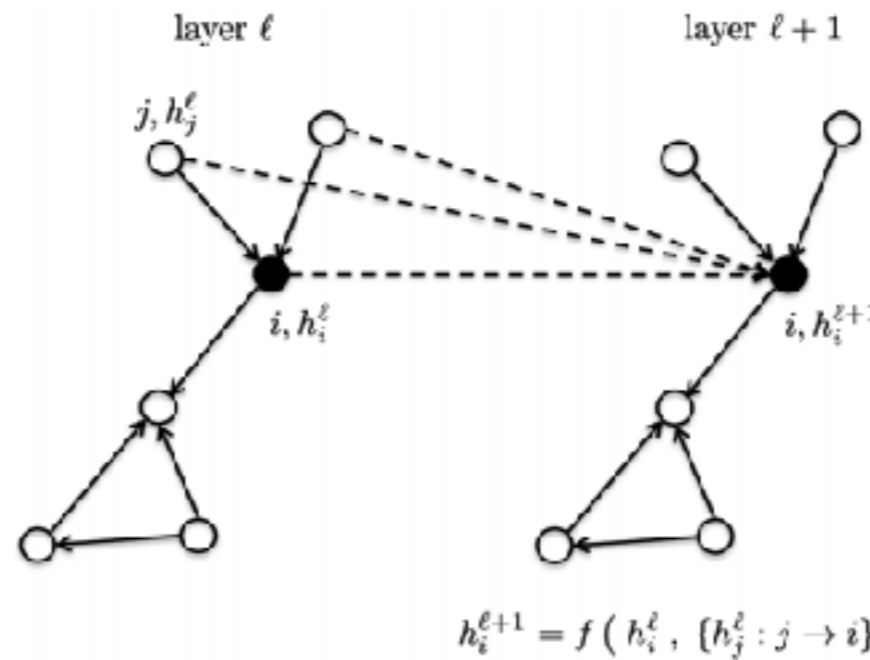


Figure 5: A generic graph neural network layer. Figure adapted from [11].

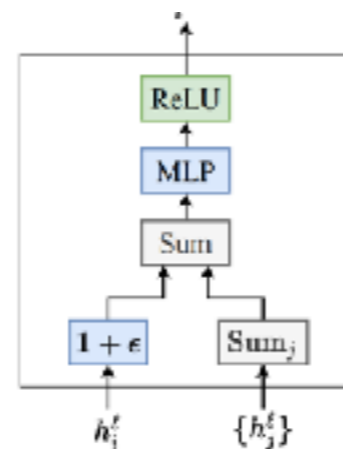


Figure 11: GIN Layer



Figure 10: GatedGCN Layer

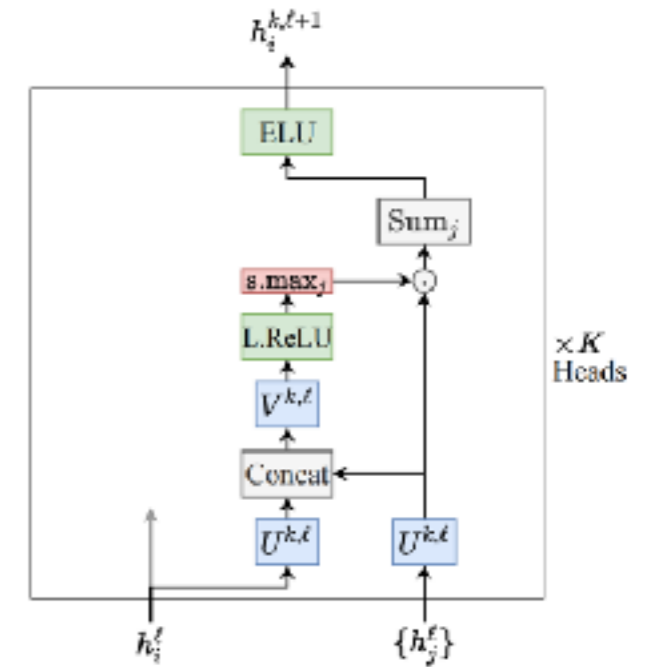


Figure 8: GAT Layer

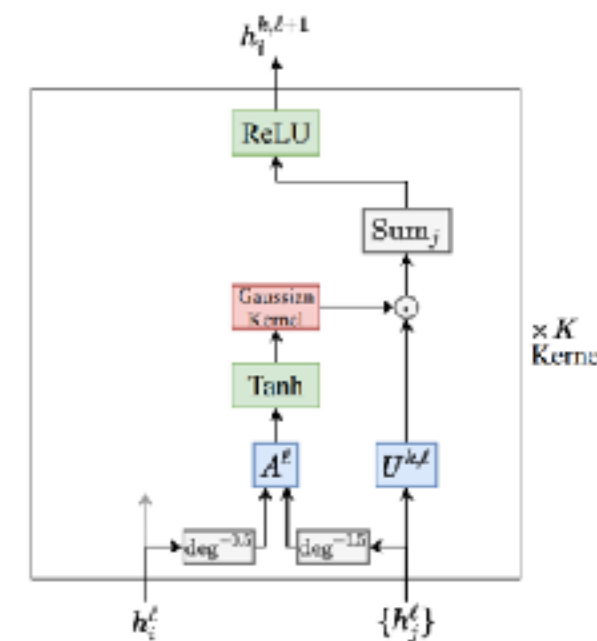
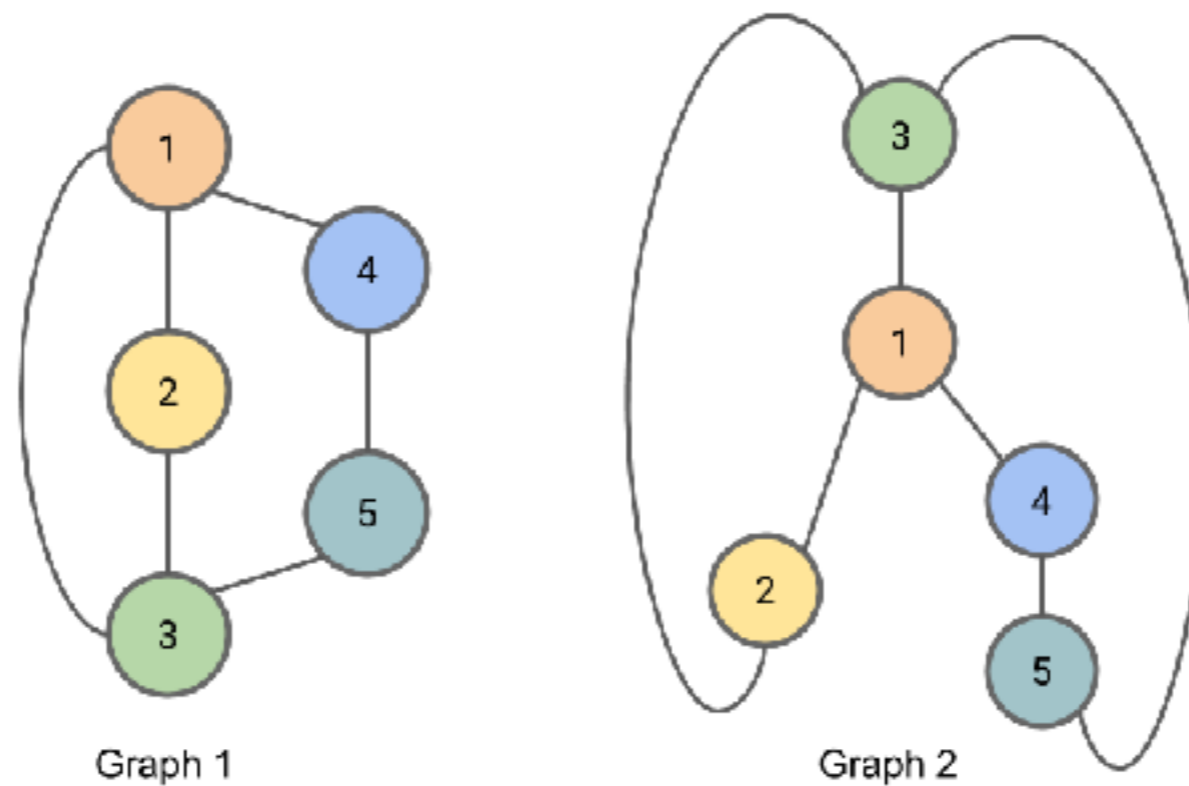


Figure 9: MoNet Layer

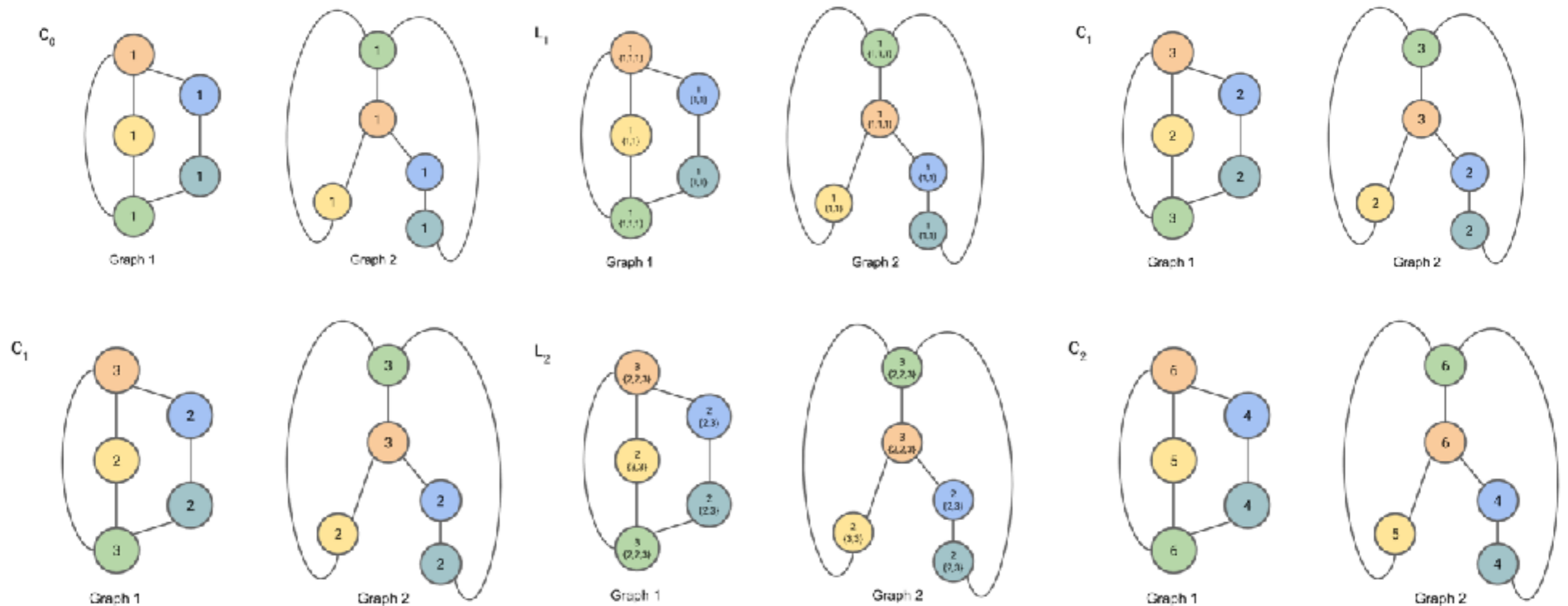
Expressiveness of Graph networks: **The Weisfeiler-Lehman Isomorphism Test**

If a mapping that preserves node adjacency exists,
two graphs are isomorphic



Expressiveness of Graph networks: The Weisfeiler-Lehman Isomorphism Test

If a mapping that preserves node adjacency exists,
two graphs are isomorphic



Is my GNN as powerful as WL test?

Sum is more expressive than mean...than max

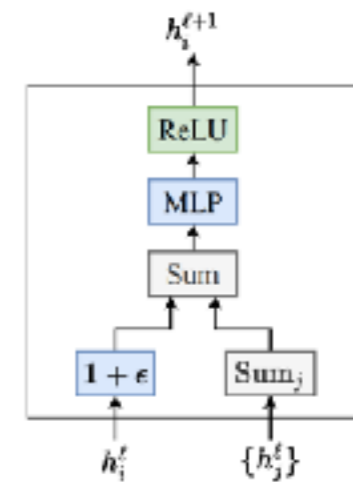
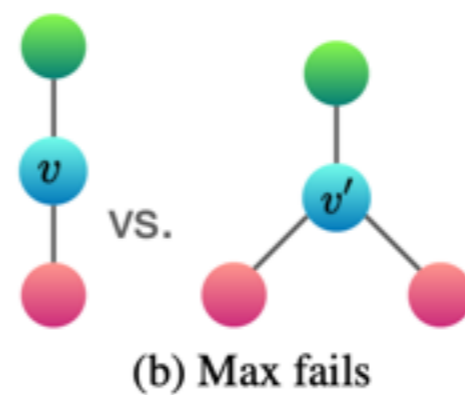
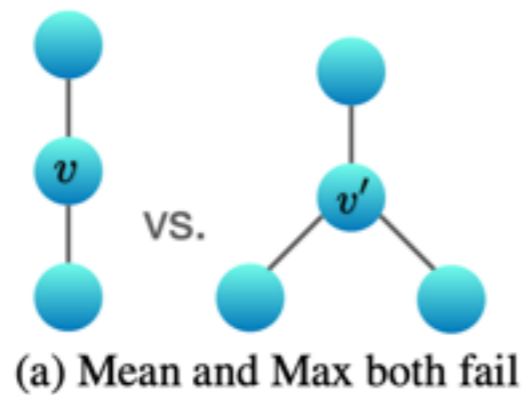
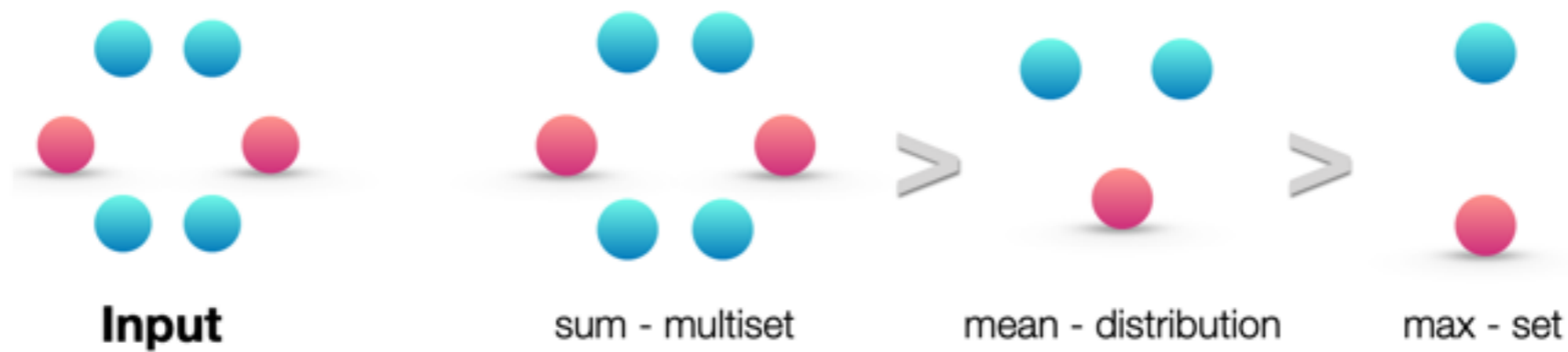
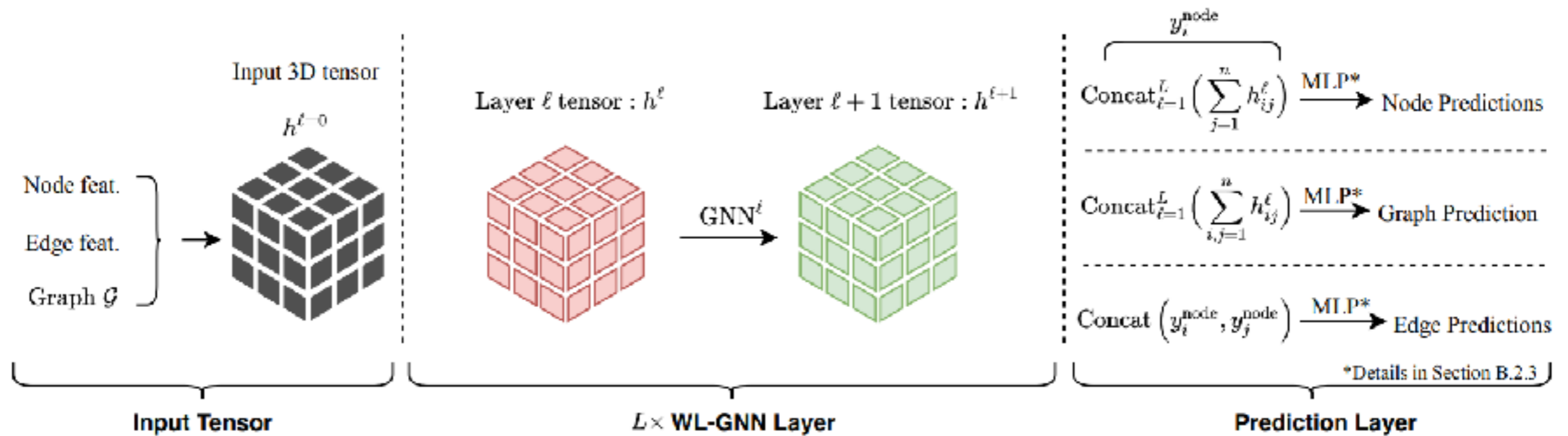


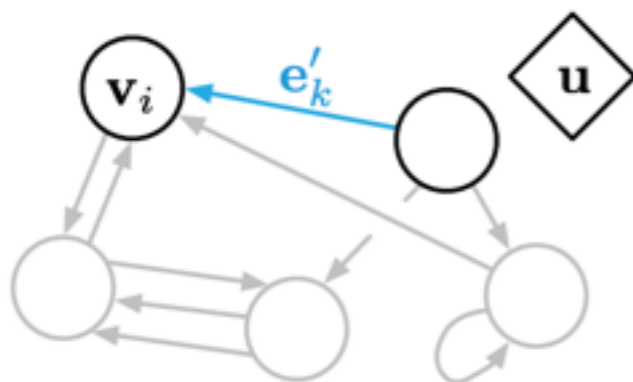
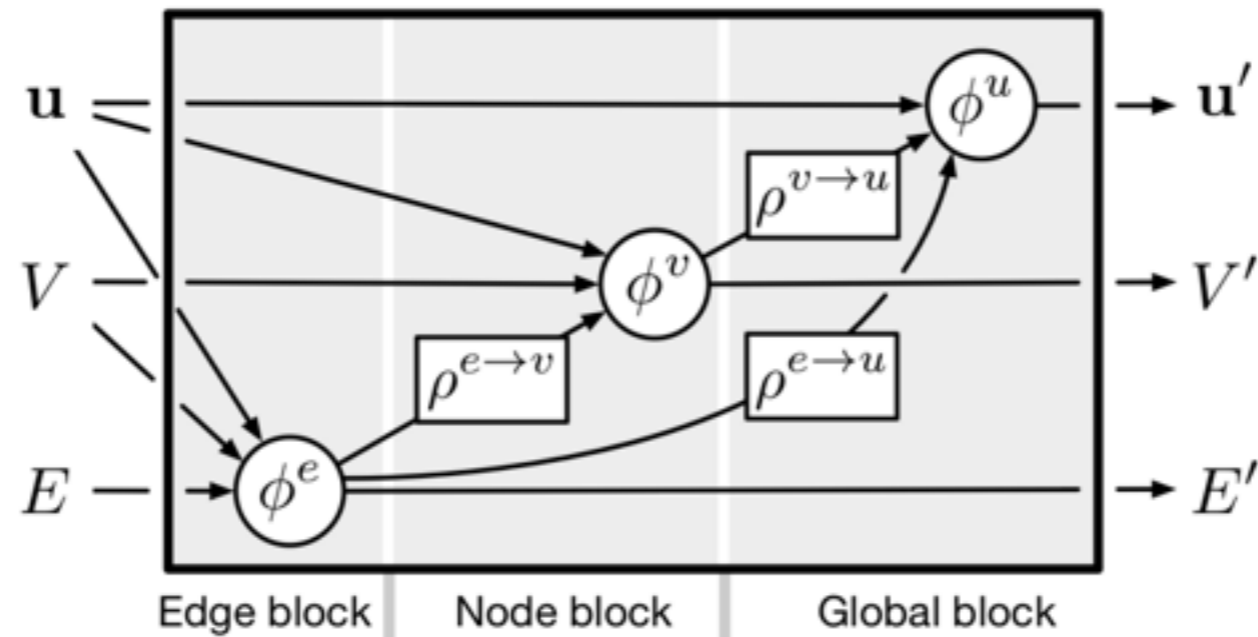
Figure 11: GIN Layer

1-WL
GNN

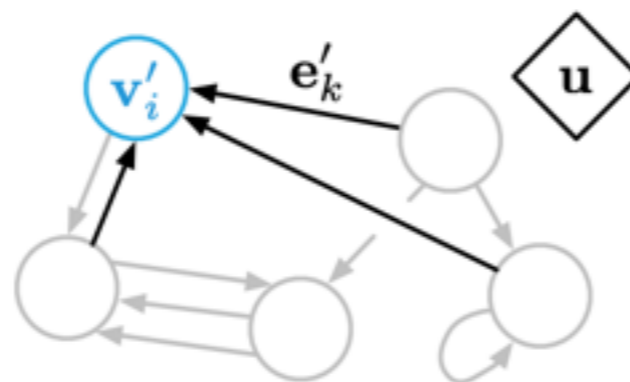
A general form of WL-Graph Network



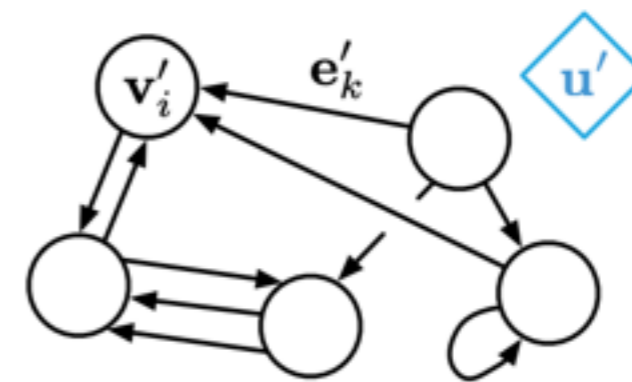
Toward a general form of Graph Network



(a) Edge update



(b) Node update



(c) Global update

Learning to Simulate Complex Physics with Graph Networks

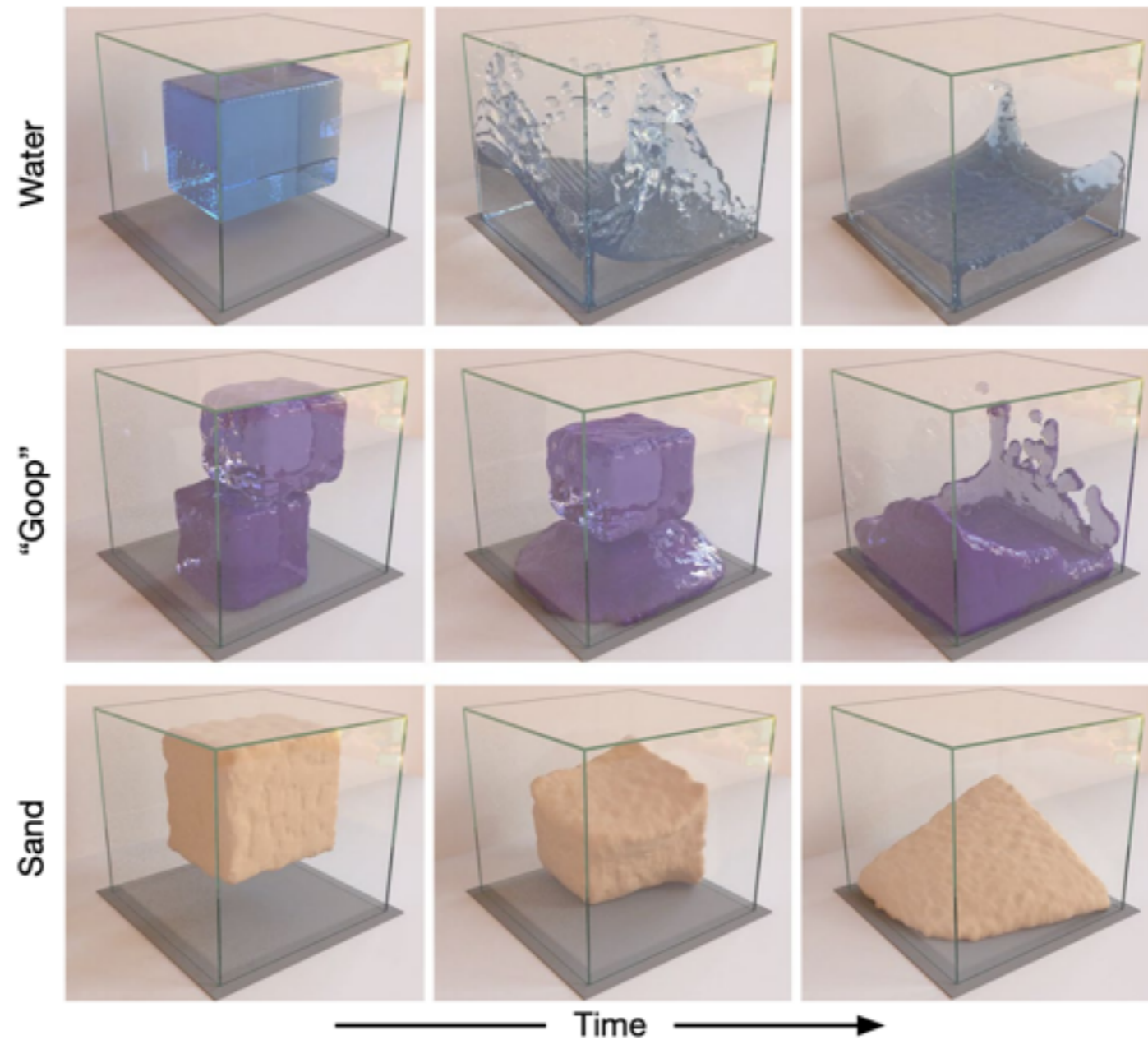


Figure 1. Rollouts of our GNS model for our WATER-3D, GOOP-3D and SAND-3D datasets. It learns to simulate rich materials at resolutions sufficient for high-quality rendering [\[video\]](#).

Convolution + Pooling is a general technique for enforcing **invariance** in representations

Can be extended to introduce **translation, rotation, or scaling** invariance etc.

Mathematical perspective: invariant transformations as symmetry groups

Cohen and Welling, 2016 Group Equivariant Convolutional Networks

Mallat, 2012

Group Invariant Scattering

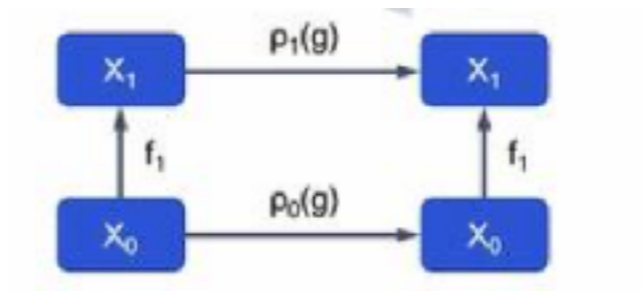
Computational challenge: how to compute efficiently?

Possible transformations grow multiplicatively if we stack invariances

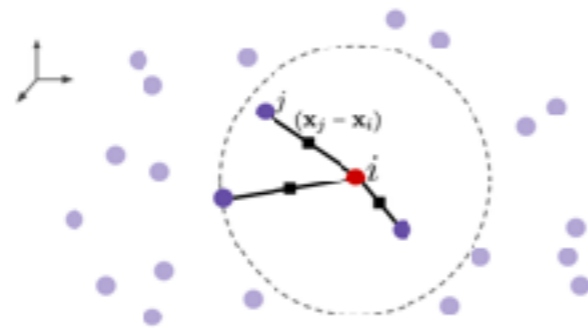
Stochastic approximation (one random transformation at a time)?

SE(3) equivariant transformer

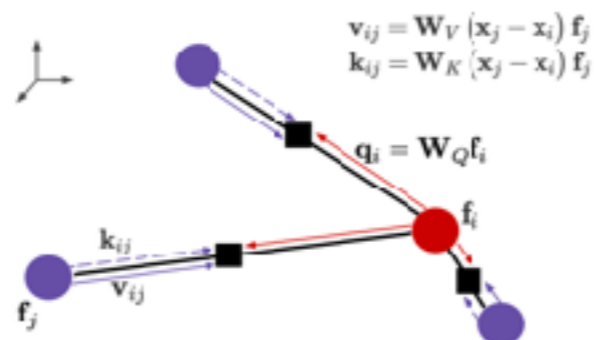
equivariant vs invariant



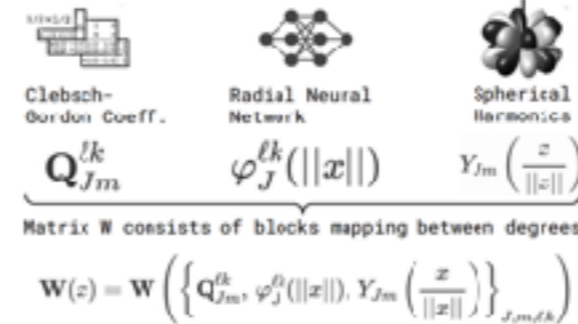
Step 1: Get nearest neighbours and relative positions



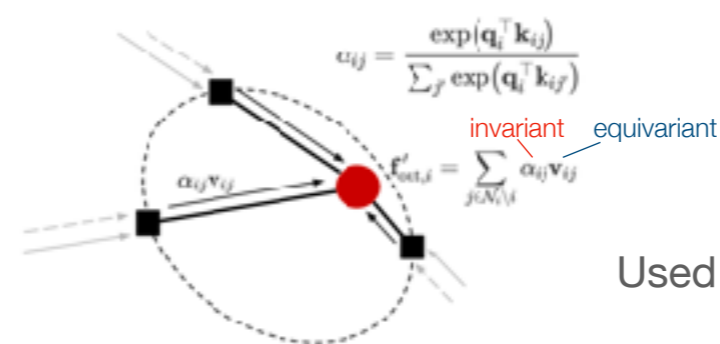
Step 2: Propagate queries, keys, and values to edges



Step 1: Get SO(3)-equivariant weight matrices



Step 2: Compute attention and aggregate



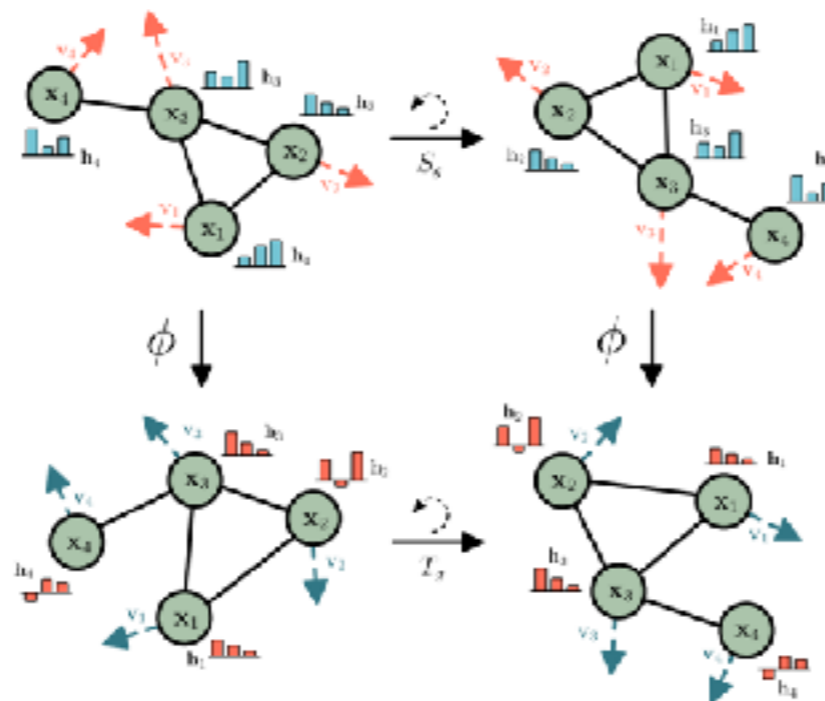
Used in RoseTTAFold & RoseTTAFold2

Design graph network for spatial coordinates

equivariant-GNNs

E(n) Equivariant Graph Neural Networks

| | GNN | Radial Field | TFN | Schnet | EGNN |
|------|---|---|--|--|---|
| Edge | $\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{a}_{ij})$ | $\mathbf{m}_{ij} = \phi_{rf}(\ \mathbf{r}_{ij}^l\)\mathbf{r}_{ij}^l$ | $\mathbf{m}_{ij} = \sum_k \mathbf{W}^{lk} \mathbf{r}_{ji}^l \mathbf{h}_i^{lk}$ | $\mathbf{m}_{ij} = \phi_{cf}(\ \mathbf{r}_{ij}^l\)\phi_s(\mathbf{h}_j^l)$ | $\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \ \mathbf{r}_{ij}^l\ ^2, \mathbf{a}_{ij})$ $\hat{\mathbf{m}}_{ij} = \mathbf{r}_{ij}^l \phi_x(\mathbf{m}_{ij})$ |
| Agg' | $\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}$ | $\mathbf{m}_i = \sum_{j \neq i} \mathbf{m}_{ij}$ | $\mathbf{m}_i = \sum_{j \neq i} \mathbf{m}_{ij}$ | $\mathbf{m}_i = \sum_{j \neq i} \mathbf{m}_{ij}$ | $\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_i = \mathcal{C} \sum_{j \neq i} \hat{\mathbf{m}}_{ij}$ |
| Node | $\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$ | $\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{m}_i$ | $\mathbf{h}_i^{l+1} = w^{ll} \mathbf{h}_i^l + \mathbf{m}_i$ | $\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$ | $\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$ $\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \hat{\mathbf{m}}_i$ |
| | Non-equivariant | E(n)-Equivariant | SE(3)-Equivariant | E(n)-Invariant | E(n)-Equivariant |



$$\mathbf{r}_{ij} = (\mathbf{x}_i - \mathbf{x}_j)$$

ϕ MLP

Day 2

Deep learning for probabilistic models

Why Toward tractable inference for more expressive probabilistic models

- Tractable inference for intractable distributions (unnormalized density)

$$\begin{array}{ll} \text{Posterior distribution} & p(\theta | \mathbf{X}, \alpha) = \frac{p(\mathbf{X} | \theta)p(\theta | \alpha)}{p(\mathbf{X} | \alpha)} \propto p(\mathbf{X} | \theta)p(\theta | \alpha) \\ \text{Energy-based models} & P(x) = \frac{1}{Z} \exp f(x) \end{array}$$

- Design more expressive tractable probability model
- Complex generative tasks / sampling

Similar to deep learning, Bayesian inference methods are often gradient based

- Variational inference
- MCMC (e.g. Hamiltonian Monte Carlo uses gradient to speed up sampling)

Deep learning for probabilistic models

Why Toward tractable inference for more expressive probabilistic models

- **Tractable inference for intractable distributions** (unnormalized density)

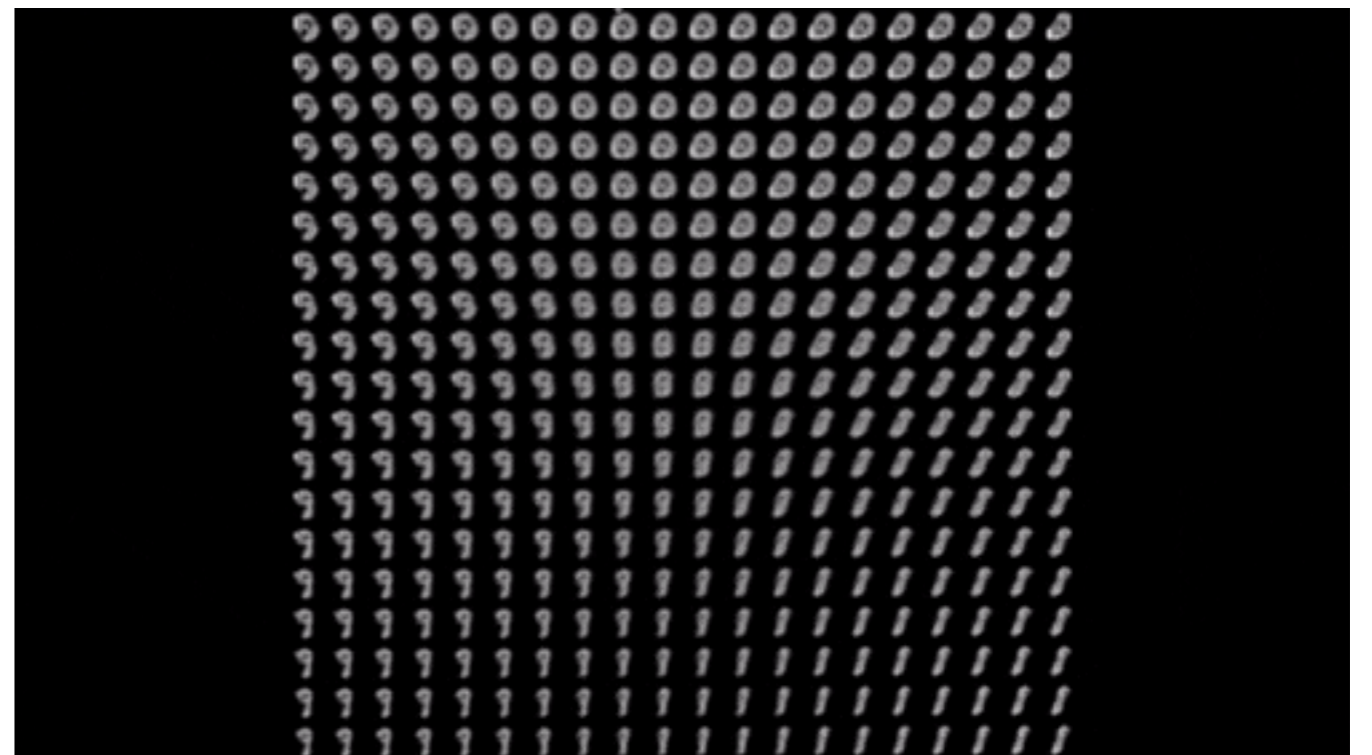
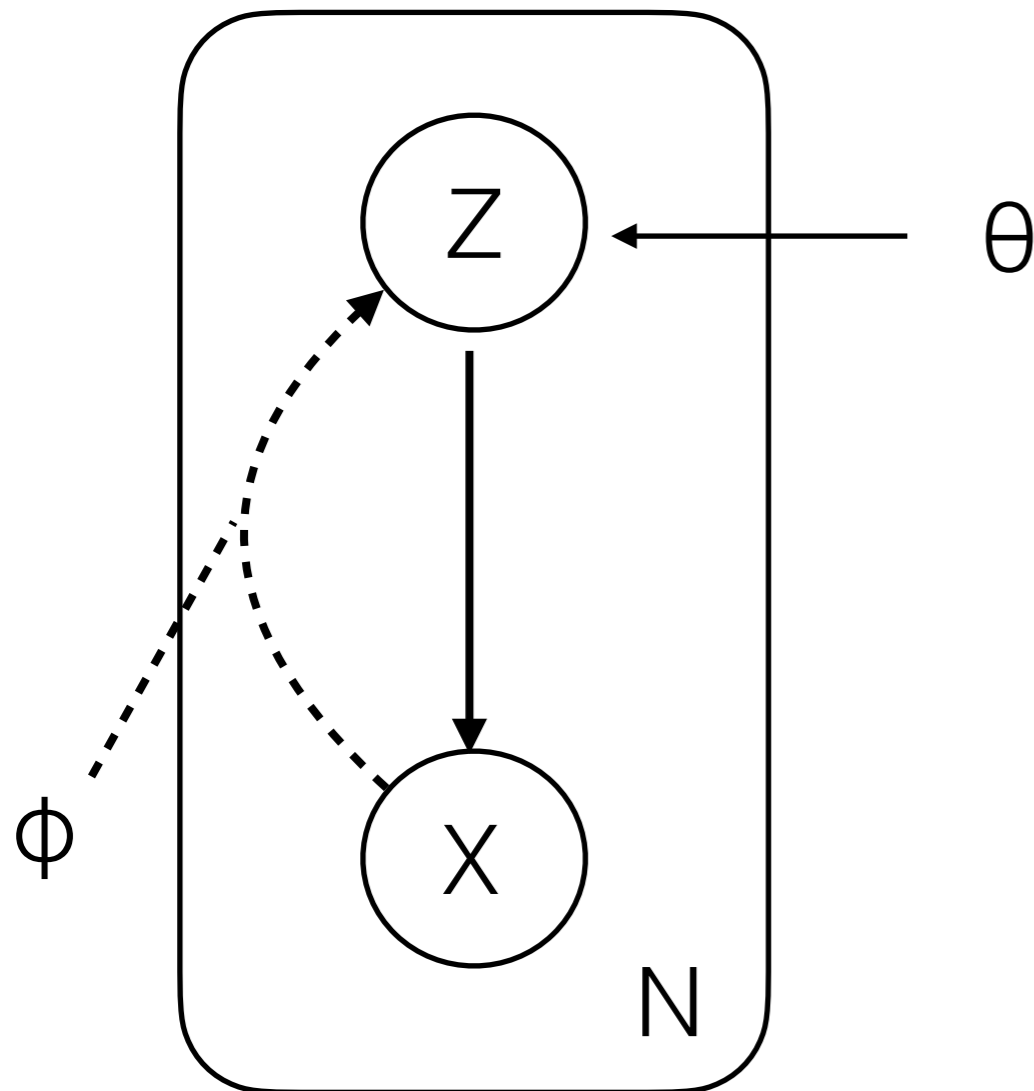
$$\begin{array}{ll} \text{Posterior distribution} & p(\theta | \mathbf{X}, \alpha) = \frac{p(\mathbf{X} | \theta)p(\theta | \alpha)}{p(\mathbf{X} | \alpha)} \propto p(\mathbf{X} | \theta)p(\theta | \alpha) \\ \text{Energy-based models} & P(x) = \frac{1}{Z} \exp f(x) \end{array}$$

Potential approaches for NN-assisted inference

- Neural variational inference (variational autoencoder)
- Neural MCMC sampler
- **Design probability model with tractable & flexible distribution**
 - Neural autoregressive model (e.g. transformer language model)
 - Normalizing flow
 - Neural ODE (continuous normalizing flow)
- **Implicit probability model with sampling capability**
 - Generative adversarial network
 - Diffusion probability models*

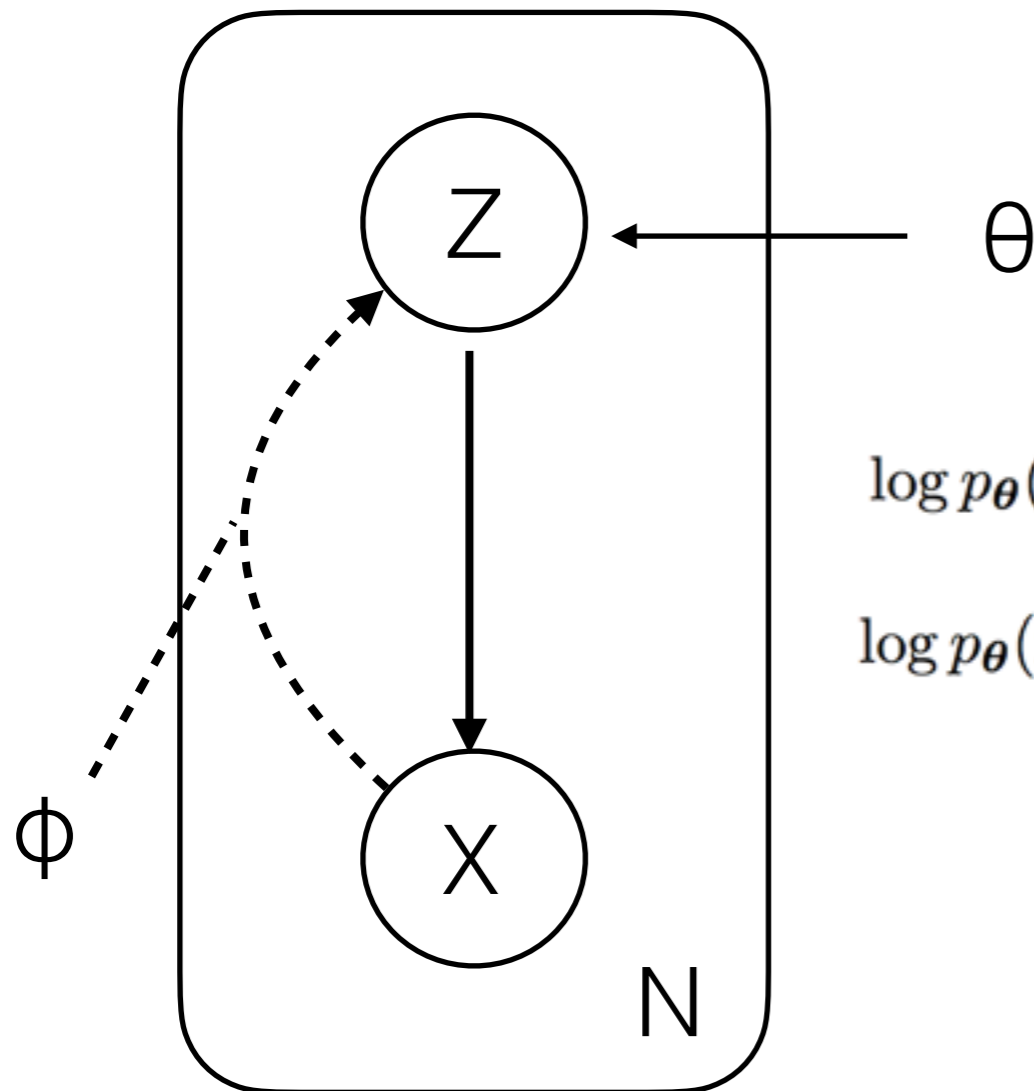
Neural variational inference

Use neural network for describing $P(X|Z)$ or $Q(Z|X)$



Neural variational inference

Use neural network for describing $P(X|Z)$ or $Q(Z|X)$



$$\log p_{\theta}(\mathbf{x}^{(i)}) = D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta}(\mathbf{z}|\mathbf{x}^{(i)})) + \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)})$$

$$\log p_{\theta}(\mathbf{x}^{(i)}) \geq \mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})} [-\log q_{\phi}(\mathbf{z}|\mathbf{x}) + \log p_{\theta}(\mathbf{x}, \mathbf{z})]$$

$$= -D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta}(\mathbf{z})) + \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})} [\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z})]$$

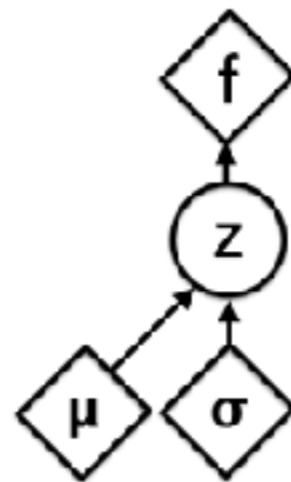
The variational objective

Backpropagation over stochastic units: Reparametrization trick

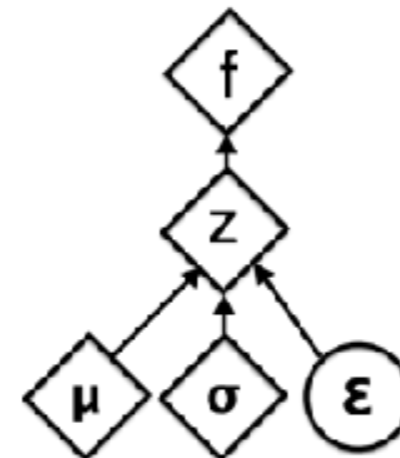
How to compute good gradient estimate of

$$-D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta}(\mathbf{z})) + \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})} \left[\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z}) \right]$$

Gradient of expectation -> expectation of stochastic gradient



Original



Reparametrized

$$\nabla_{\mu, \sigma} E_{z \sim p(z|\mu, \sigma)} [f(z)] = E_{z \sim p(z|\mu, \sigma)} [f(z) \nabla_{\mu, \sigma} \log(p(z|\mu, \sigma))]$$

$$\nabla_{\mu, \sigma} E_{\epsilon \sim p(\epsilon)} [f(z)] = E_{\epsilon \sim p(\epsilon)} [\nabla_{\mu, \sigma} f(g(\mu, \sigma, \epsilon))]$$

Backpropagation over stochastic units:

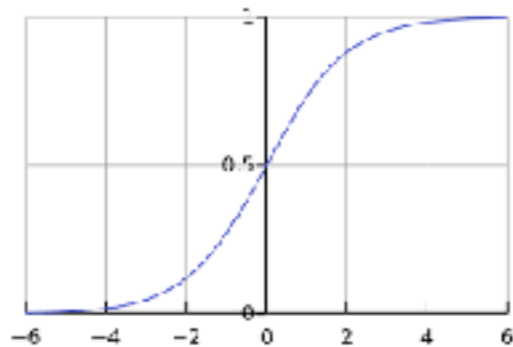
Reparametrization trick for **discrete variables**

The Gumbel trick for sampling from discrete distributions $P(X = k) \propto \alpha_k$

$$G = -\log(-\log(U)) \text{ with } U \sim \text{Unif}[0, 1]$$

$$X = \arg \max_k (\log \alpha_k + G_k).$$

Softmax function for approximating the max operation with a differentiable function



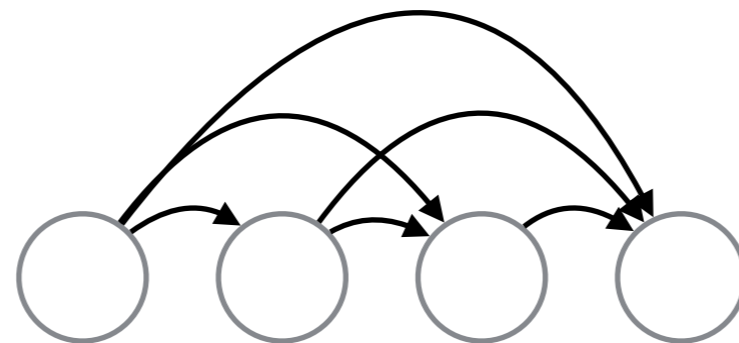
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

Discrete variables can always be represented by binary vectors

Toward flexible and normalized density models

$$-\underbrace{D_{KL}(q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta}(\mathbf{z}))} + \mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x}^{(i)})} \left[\underbrace{\log p_{\theta}(\mathbf{x}^{(i)}|\mathbf{z})} \right]$$

1. Fully factorized models

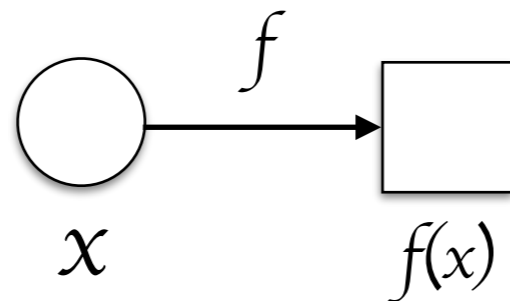


Neural autoregressive models (e.g. GPT)

Probability function is fully factorized
However, it has to commit to a certain order

Toward flexible and normalized density models

2. Invertible transformations (Flow models)



$$p_X(x) = p_H(f(x)) \left| \det \frac{\partial f(x)}{\partial x} \right|.$$

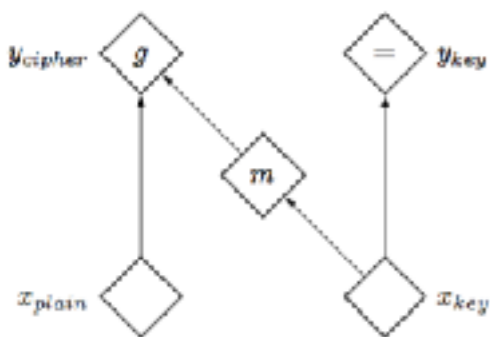
Examples:

NICE

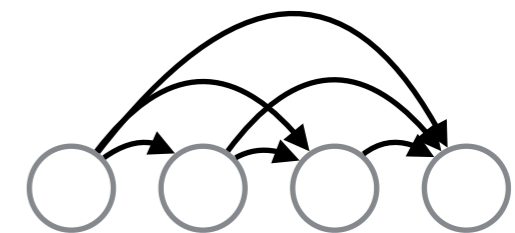
Normalizing flow

Invertible autoregressive flow

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_2 + \mathcal{F}(x_1) \end{aligned}$$



$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^\top \mathbf{z} + b)$$



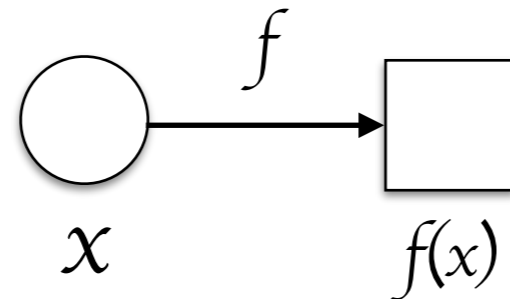
determinant fixed

determinant $O(D)$ time

determinant $O(D)$ time

Toward flexible and normalized density models

2. Invertible transformations (Flow models)



$$p_X(x) = p_H(f(x)) \left| \det \frac{\partial f(x)}{\partial x} \right|.$$

More Examples (free-form Jacobian):

Invertible ResNet

$$x_{t+1} \leftarrow x_t + g_{\theta_t}(x_t)$$

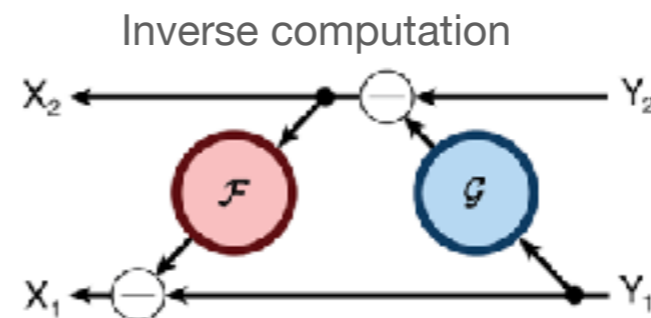
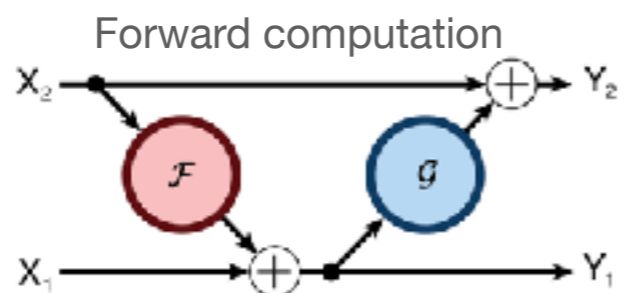
$$\text{Lip}(g_{\theta_t}) < 1, \text{ for all } t = 1, \dots, T,$$

Algorithm 1. Inverse of i-ResNet layer via fixed-point iteration.

Input: output from residual layer y , contractive residual block g , number of fixed-point iterations n
 Init: $x^0 := y$
for $i = 0, \dots, n$ **do**
 $x^{i+1} := y - g(x^i)$
end for

Invertible Residual Networks (2019)

RevNet

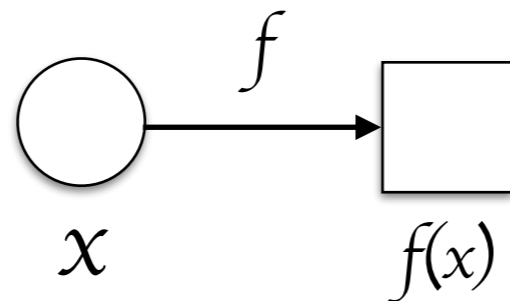


The Reversible Residual Network: Backpropagation Without Storing Activations

Hidden variables are equal in dimensionality.

Toward flexible and normalized density models

2. Invertible transformations (Flow models)

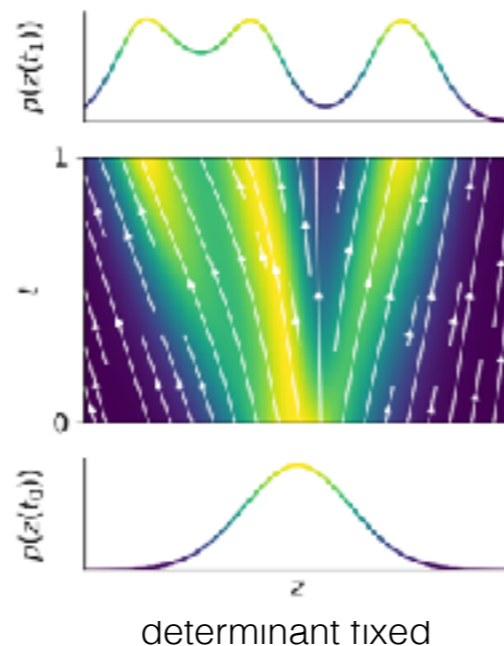
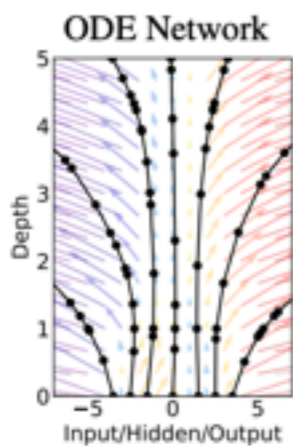
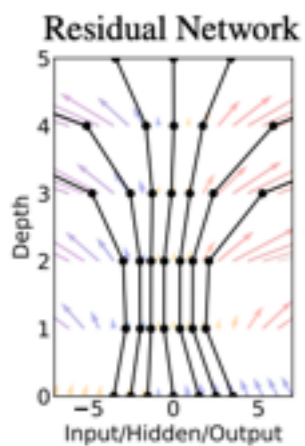


$$p_X(x) = p_H(f(x)) \left| \det \frac{\partial f(x)}{\partial x} \right|.$$

More Examples (Neural ODE):

ODE

Continuous change of variable formula



Log likelihood

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \text{Tr} \left(\frac{\partial f}{\partial \mathbf{z}(t)} \right) dt.$$

FFJORD: Unbiased estimate of $\text{Tr}(\frac{\partial f}{\partial \mathbf{z}})$ with $\epsilon^T \frac{\partial f}{\partial \mathbf{z}} \epsilon$

Sampling-focused* deep generative models: diffusion models and GAN



Score-matching allows generative model fitting without computing the partition function

Energy-based models $P(x) = \frac{1}{Z} \exp f(x)$

Score function $\mathbf{s}_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) + \underbrace{\nabla_{\mathbf{x}} \log Z_\theta}_{=0} = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x})$

Minimize Fisher divergence $\mathbb{E}_{p(\mathbf{x})} [\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|_2^2]$

However, we don't know $\nabla_{\mathbf{x}} \log p(\mathbf{x})$

Denoising score-matching

$$\frac{1}{2} \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p_{\text{data}}(\mathbf{x})} [\|\mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} | \mathbf{x})\|_2^2].$$

Sample from $p(X)$ using its gradient: Langevin dynamics

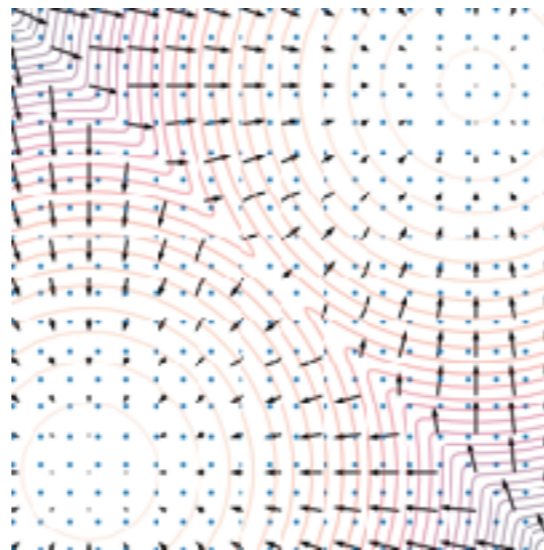
Initialize \mathbf{x} from arbitrary distribution

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon} \mathbf{z}_i, \quad i = 0, 1, \dots, K,$$

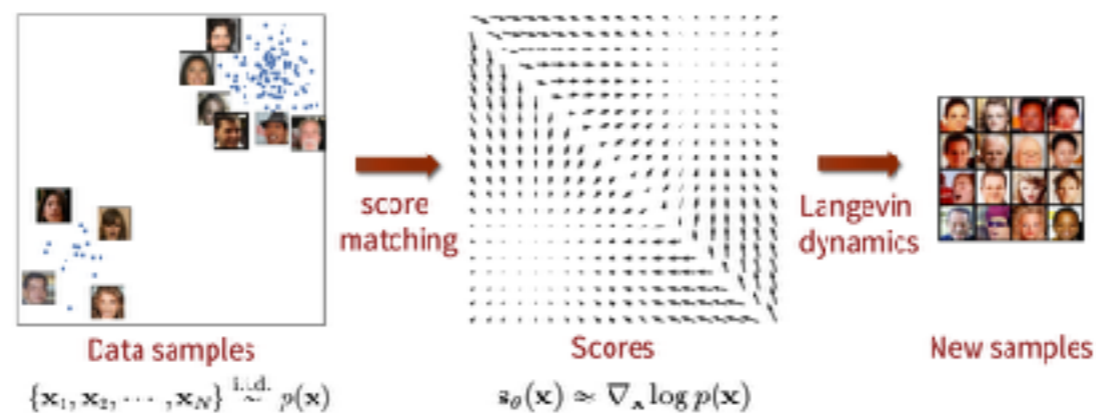
$$\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, I).$$

$$\epsilon \rightarrow 0;$$

$$K \rightarrow \infty$$



i.e. once we learned the score function, we can sample from $p(X)$,



Sample from $p(X)$ using its gradient: Langevin dynamics

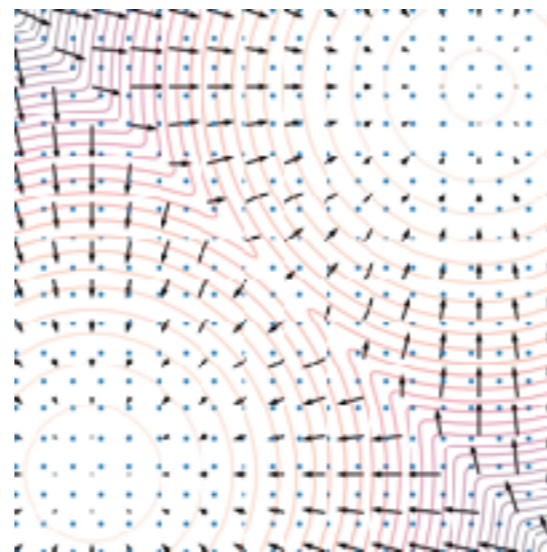
Initialize \mathbf{x} from arbitrary distribution

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon} \mathbf{z}_i, \quad i = 0, 1, \dots, K,$$

$$\mathbf{z}_i \sim \mathcal{N}(\mathbf{0}, I).$$

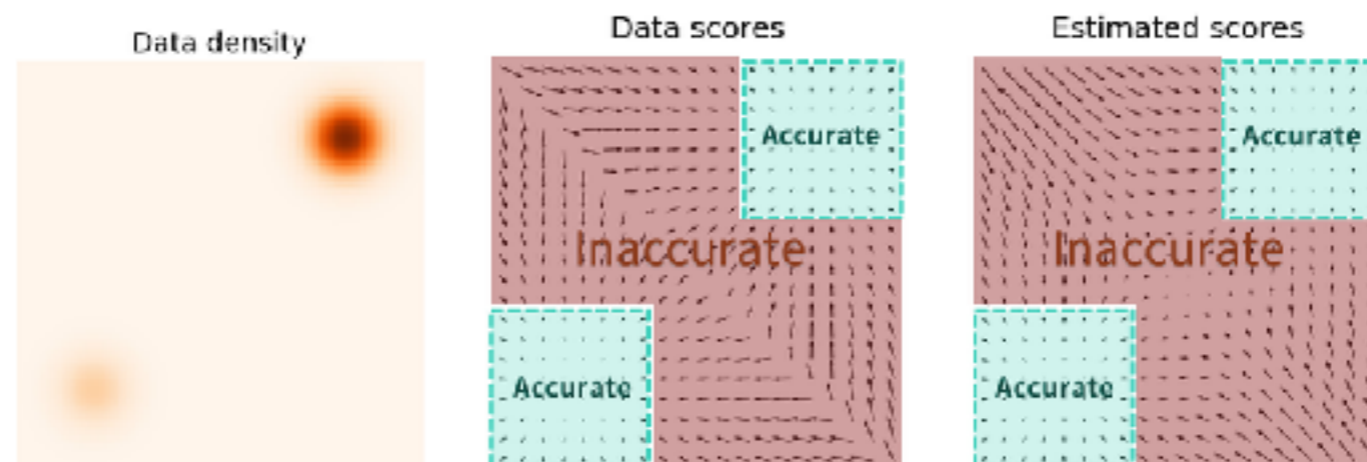
$$\epsilon \rightarrow 0;$$

$$K \rightarrow \infty$$

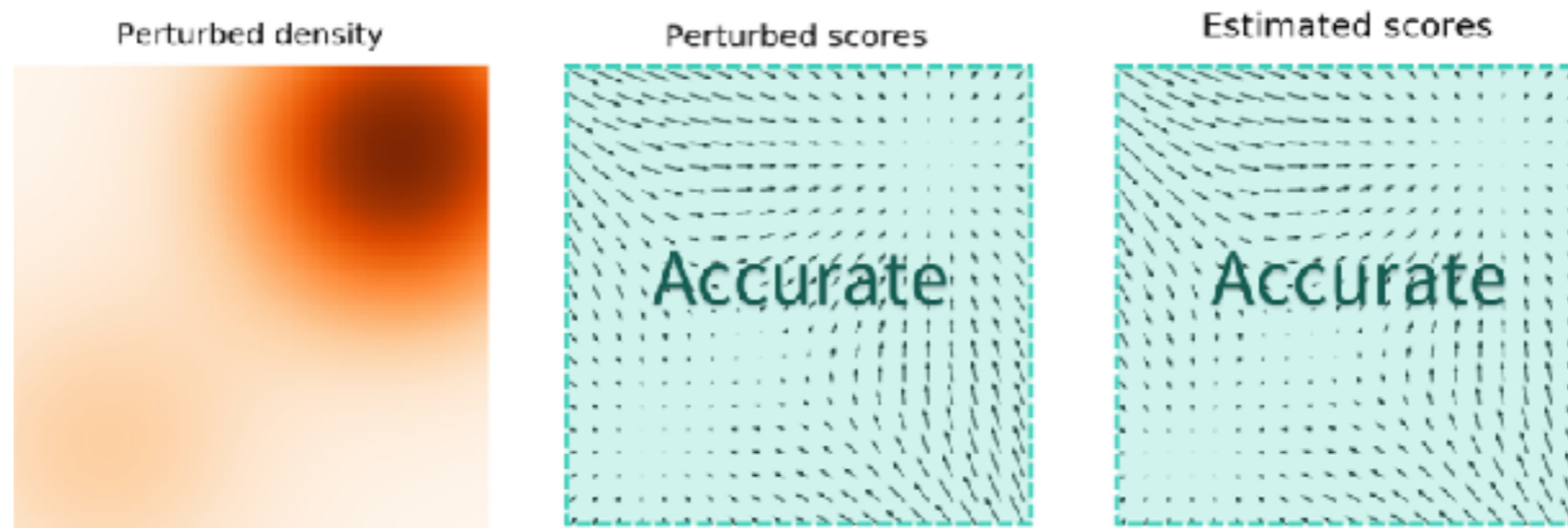


i.e. once we learned the score function, we can sample from $p(X)$,

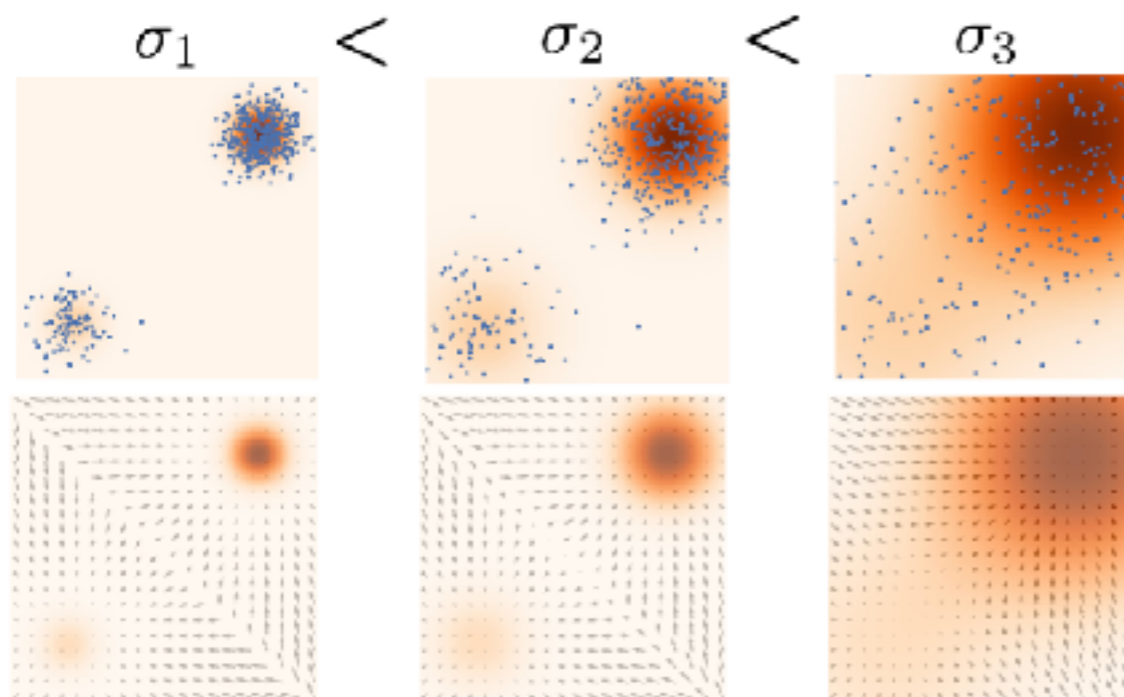
however...



Learning the score function with data + noise



What noise level? Use multiple!



Annealed Langevin dynamics

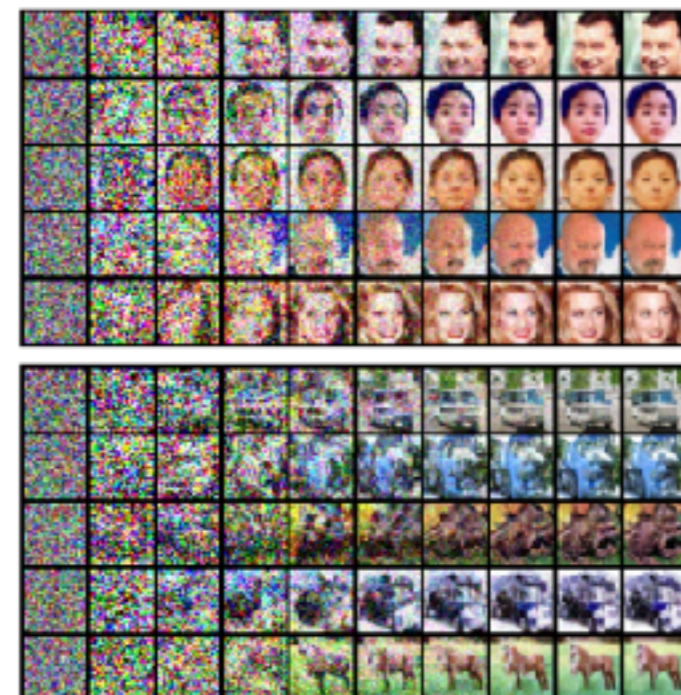
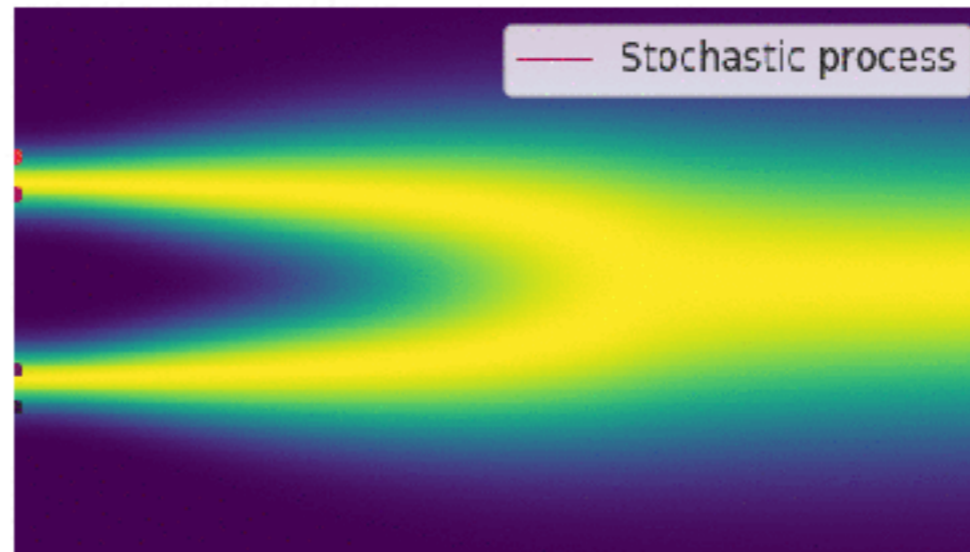


Figure 4: Intermediate samples of annealed Langevin dynamics.

Score-based generative modeling with stochastic differential equations (SDEs)

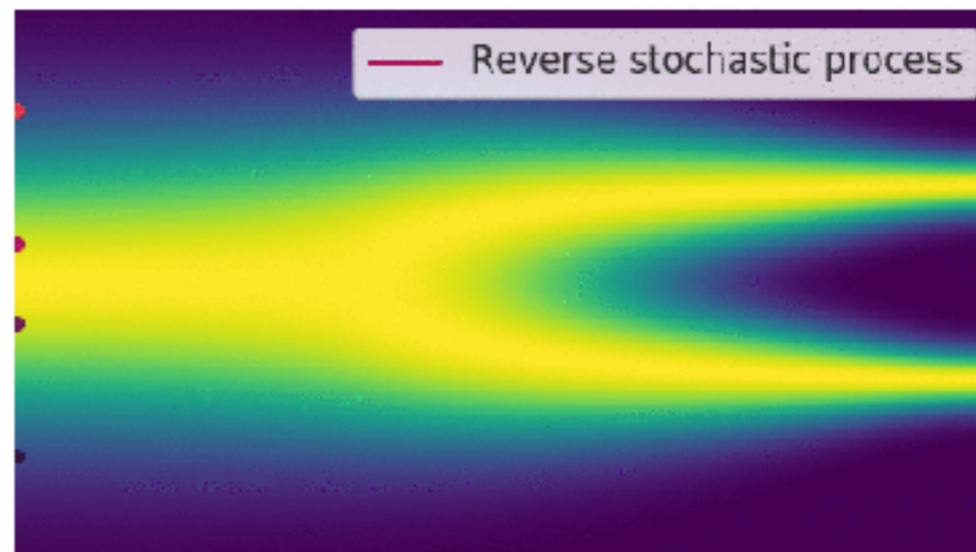
Multiple noise-levels -> infinite noise levels (SDE)



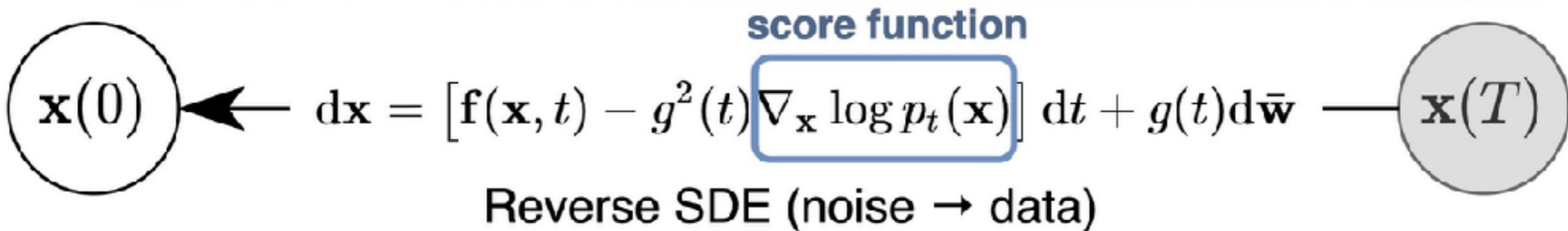
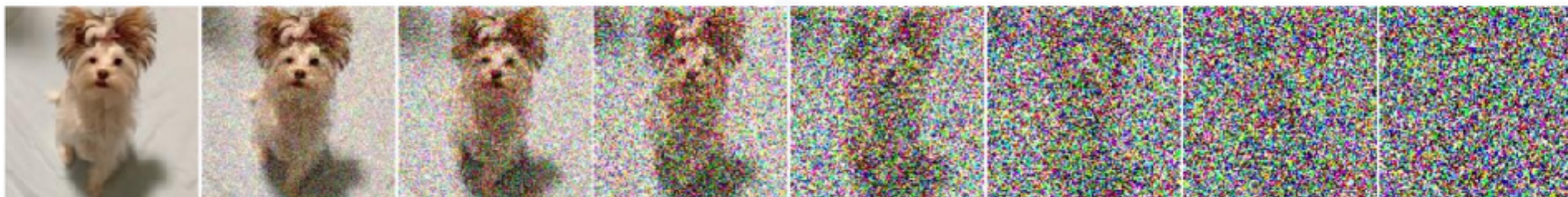
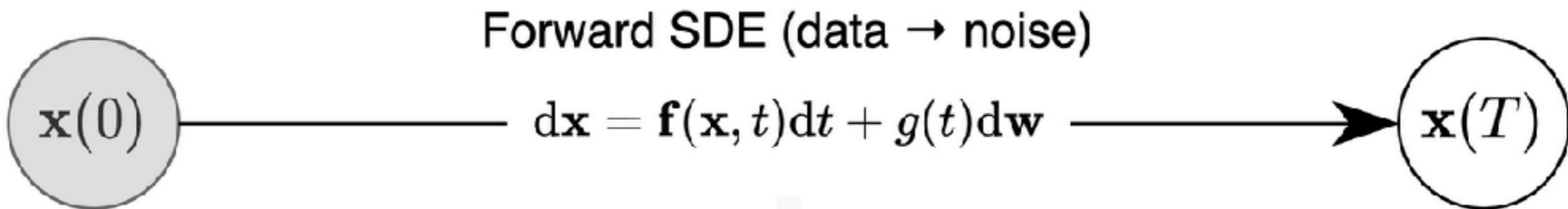
Converge to a static distribution (prior distribution)

$$dx_t = -\theta x_t dt + \sigma dW_t$$

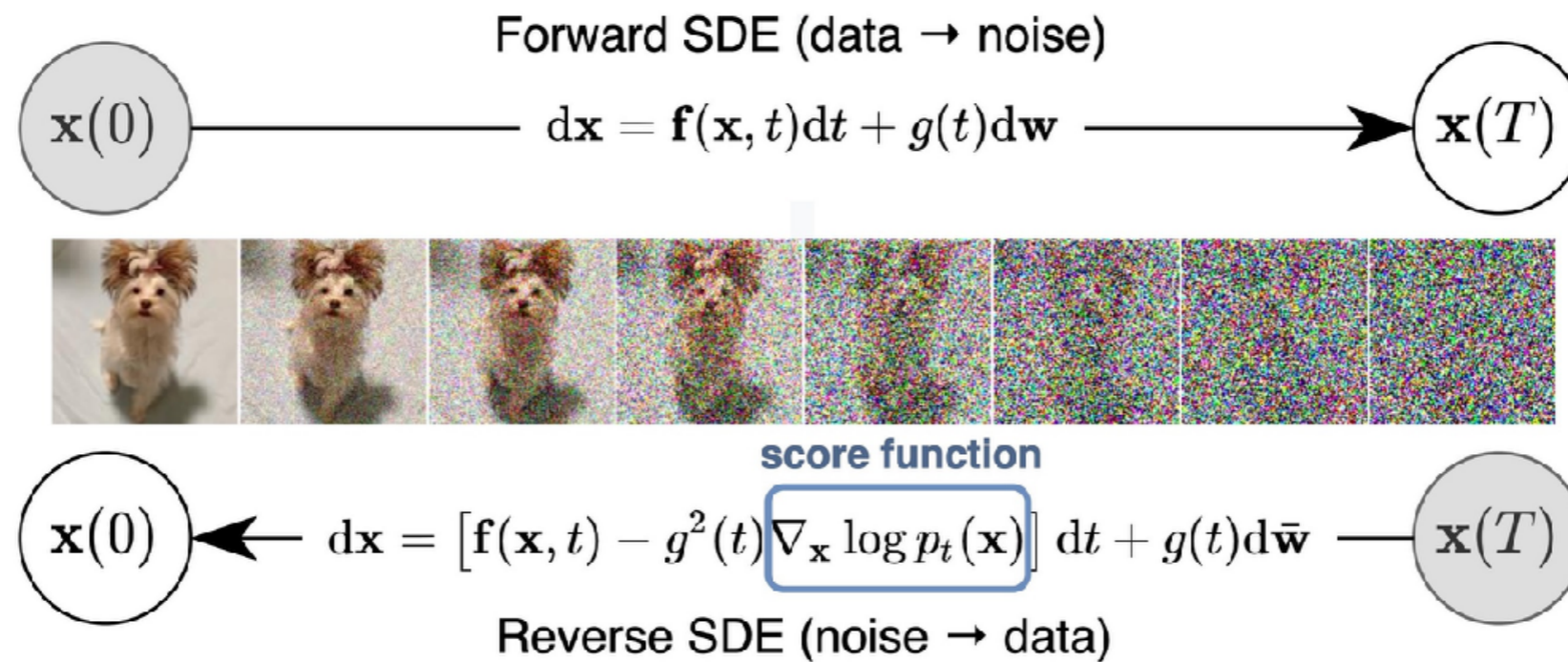
Reverse SDE is equivalent to sampling!



Score-based generative modeling with stochastic differential equations (SDEs)



Score-based generative modeling with stochastic differential equations (SDEs)



Learning the score function with infinite noise levels (SDE)

score-matching

$$\mathbb{E}_{p(\mathbf{x})} [\|\nabla_{\mathbf{x}} \log p(\mathbf{x}) - \mathbf{s}_{\theta}(\mathbf{x})\|_2^2]$$

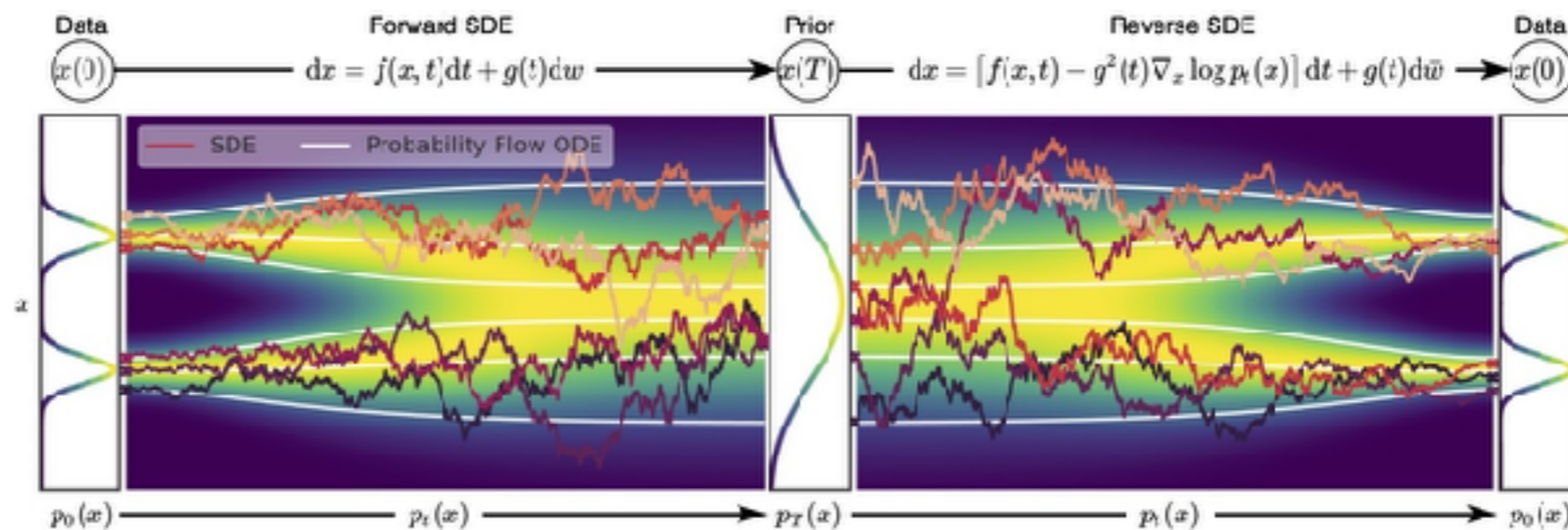
SDE score-matching

$$\mathbb{E}_{t \in \mathcal{U}(0, T)} \mathbb{E}_{p_t(\mathbf{x})} [\lambda(t) \|\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - \mathbf{s}_{\theta}(\mathbf{x}, t)\|_2^2]$$

Score-based generative modeling with stochastic differential equations (SDEs)

Convert learned SDE to and ODE with the same distribution (probability flow ODE): allows computing likelihood!

$$dx = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g^2(t) \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) \right] dt.$$



Score-matching for solving **inverse problems**

Given $P(Y|X)$ Solve $P(X|Y)$

Inverse problems are typically a family of problems, which is easy to compute in one direction, but hard to compute in the reversed direction

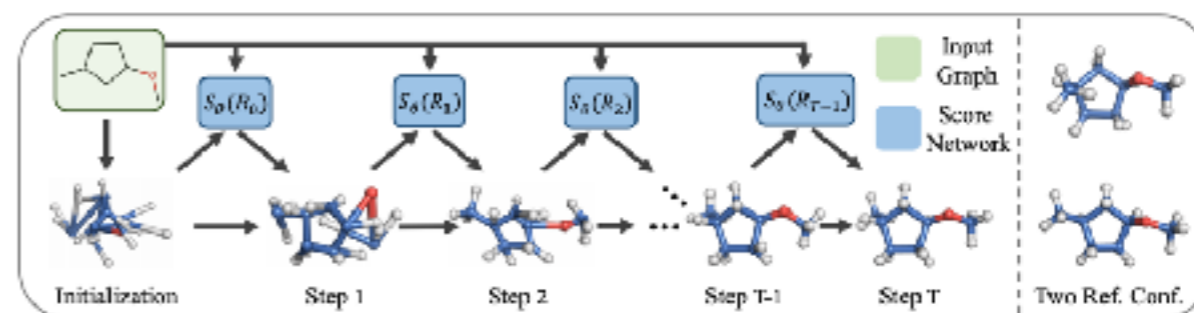
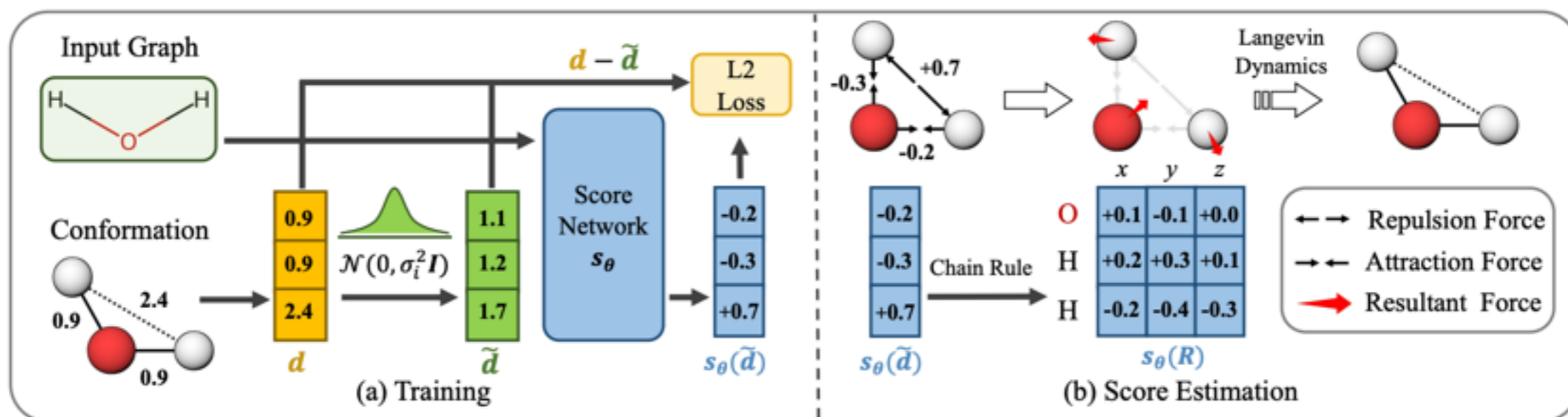
$$\nabla_{\mathbf{x}} \log p(\mathbf{x} | \mathbf{y}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \nabla_{\mathbf{x}} \log p(\mathbf{y} | \mathbf{x}).$$

Image colorization (x: color image, y: b/w image)

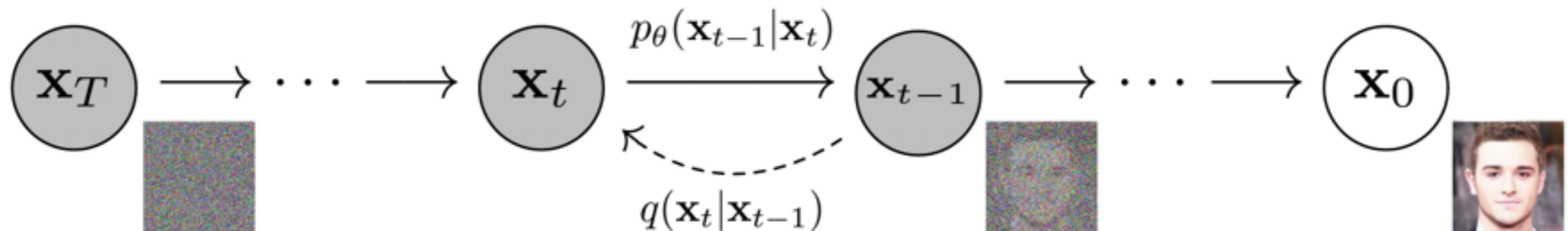


Application example: predicting 3D molecular structure

1. 3D **equivariant representation** of molecular structure with **distances**
2. Learn a **conditional score network** for distances with denoising score-matching
3. Sample by back-propagating **gradient from distance to coordinates**



Denoising diffusion probabilistic model



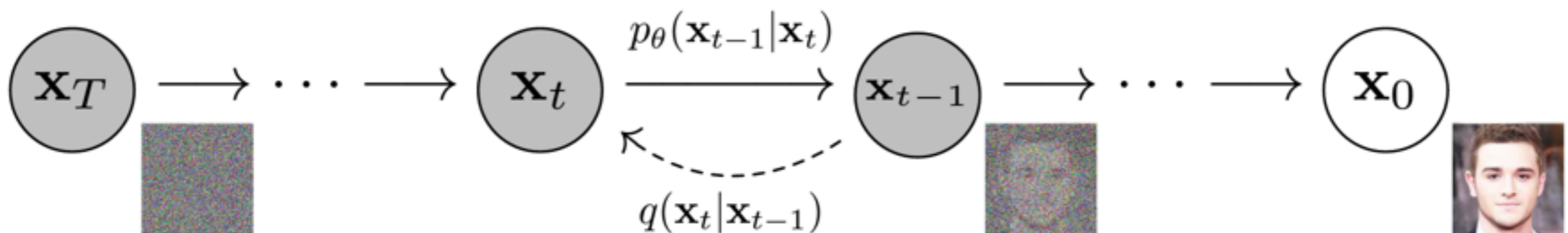
Forward “diffusion” process gradually add noise until reaching unit Gaussian distribution

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Multiple steps of diffusion is still described by Gaussian distribution

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}) \quad \bar{\alpha}_t := 1 - \beta_t \quad \bar{\alpha}_t := \prod_{s=1}^t \alpha_s$$

Denoising diffusion probabilistic model



Variational ELBO objective

$$\mathbb{E}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] = \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right]$$

Which simplifies to
$$\mathbb{E}_{\mathbf{x}_0, \epsilon} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t) \right\|^2 \right] + \log p_\theta(x_0 | x_1)$$

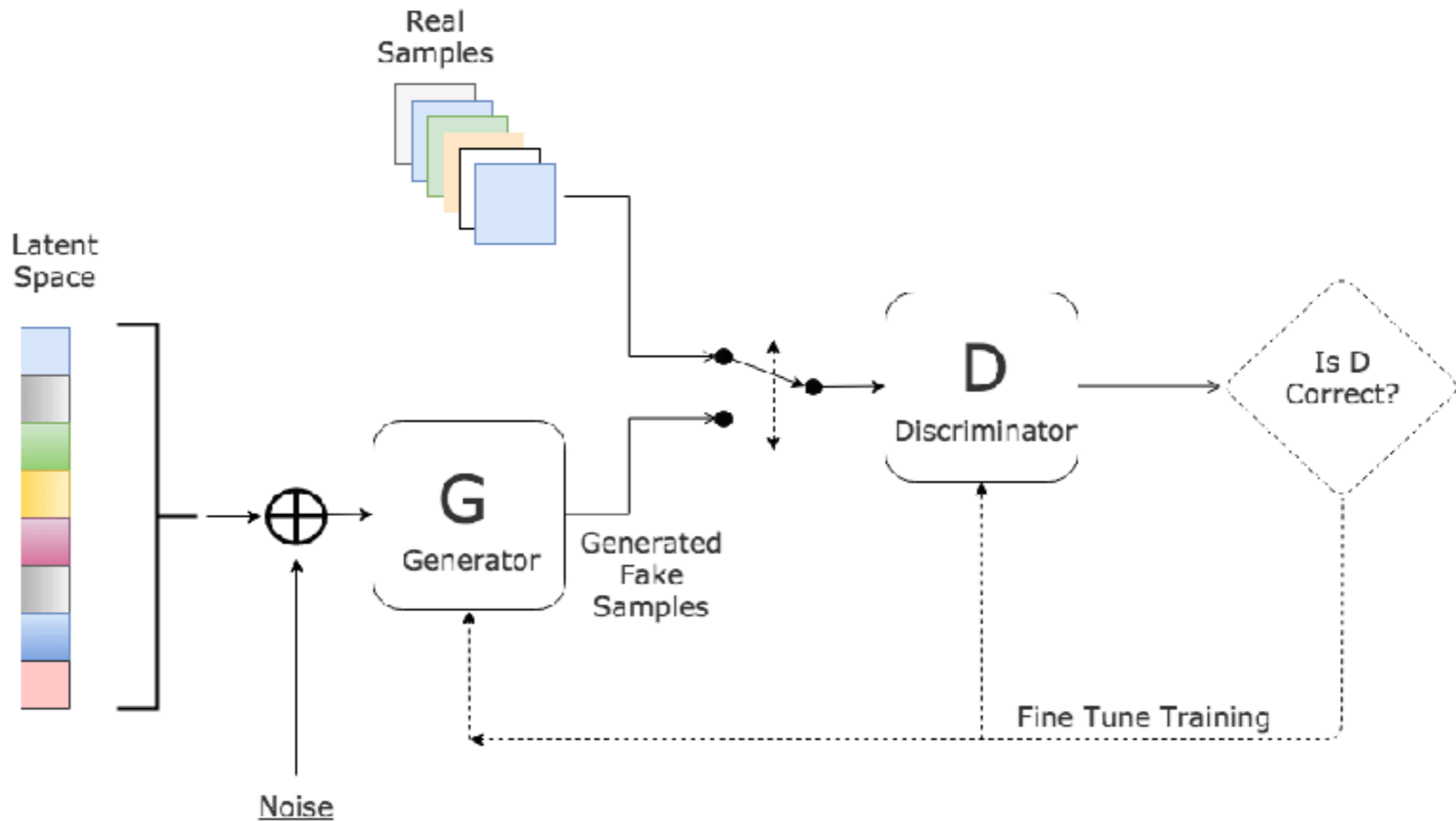
Simplified objective typically works better

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \epsilon} \left[\left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t) \right\|^2 \right]$$

p_θ is typically defined to be Gaussian and with variance matching the forward diffusion process

Probabilistic modeling with neural networks: Learn to sample

Generative adversarial networks



$$\min_{\theta} \max_{\phi} V(G_{\theta}, D_{\phi}) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log D_{\phi}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_{\phi}(G_{\theta}(\mathbf{z})))]$$

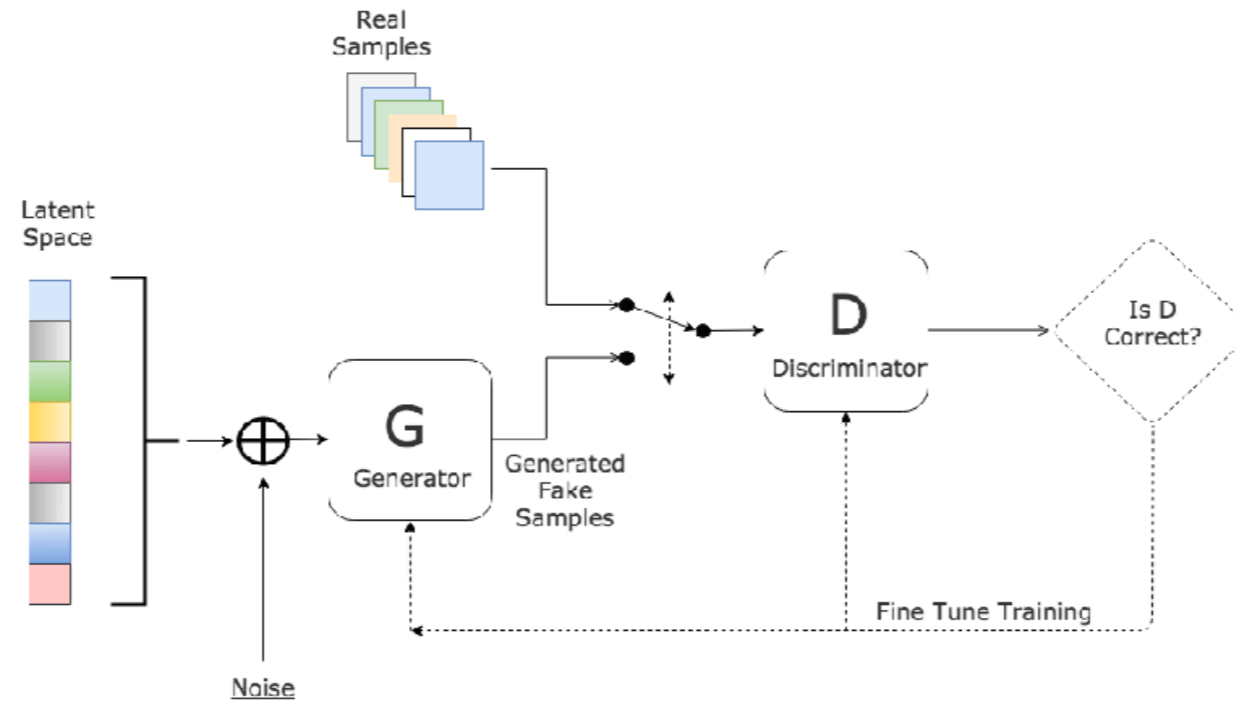
Probabilistic modeling with neural networks: Learn to sample

Generative adversarial networks



StyleGAN

Formulating Generative adversarial networks as a probabilistic model



$$P(x) = \frac{1}{Z} \exp f(x)$$

$$\frac{\partial \log P}{\partial \theta} = E_{x \sim \text{data}} \left(\frac{\partial f(x)}{\partial \theta} \right) - E_{x \sim \text{model}} \left(\frac{\partial f(x)}{\partial \theta} \right)$$

Generator network: use $x \sim \text{Generator}$ instead of $x \sim \text{model}$

Discriminator network: $f(x)$

Wasserstein GAN objective: $E_{x \sim \text{data}} f(x) - E_{x \sim \text{generator}} f(x)$

Sentence-guided generation: VQGAN + CLIP

"Planetary City C" from VQ-GAN+CLIP (source: [@RiversHaveWings](#) on Twitter)



Aran Kowalski
@arankowalski

When you generate images with VQGAN + CLIP, the image quality dramatically improves if you add "unreal engine" to your prompt.

People are now calling this "unreal engine trick" lol

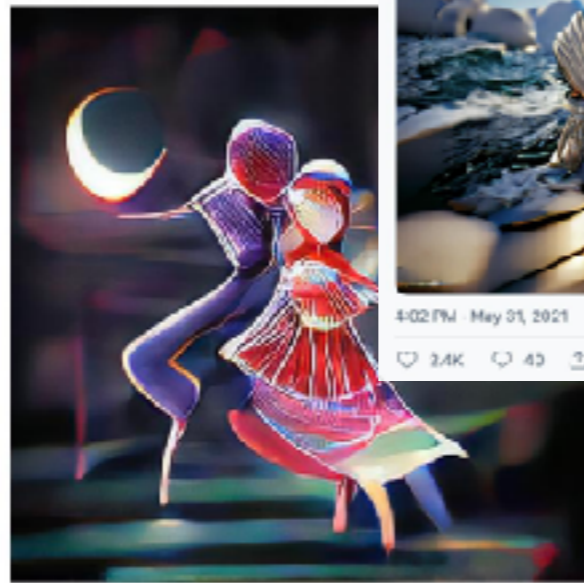
e.g. "the angel of air, unreal engine"



4:02 PM · May 01, 2021

2.4K 43 Share this Tweet

"Dancing in the moonlight" from VQ-GAN+CLIP (source: [@advadnoun](#) on Twitter)



"re" from VQ-GAN+CLIP (source: [@RiversHaveWings](#) on Twitter)

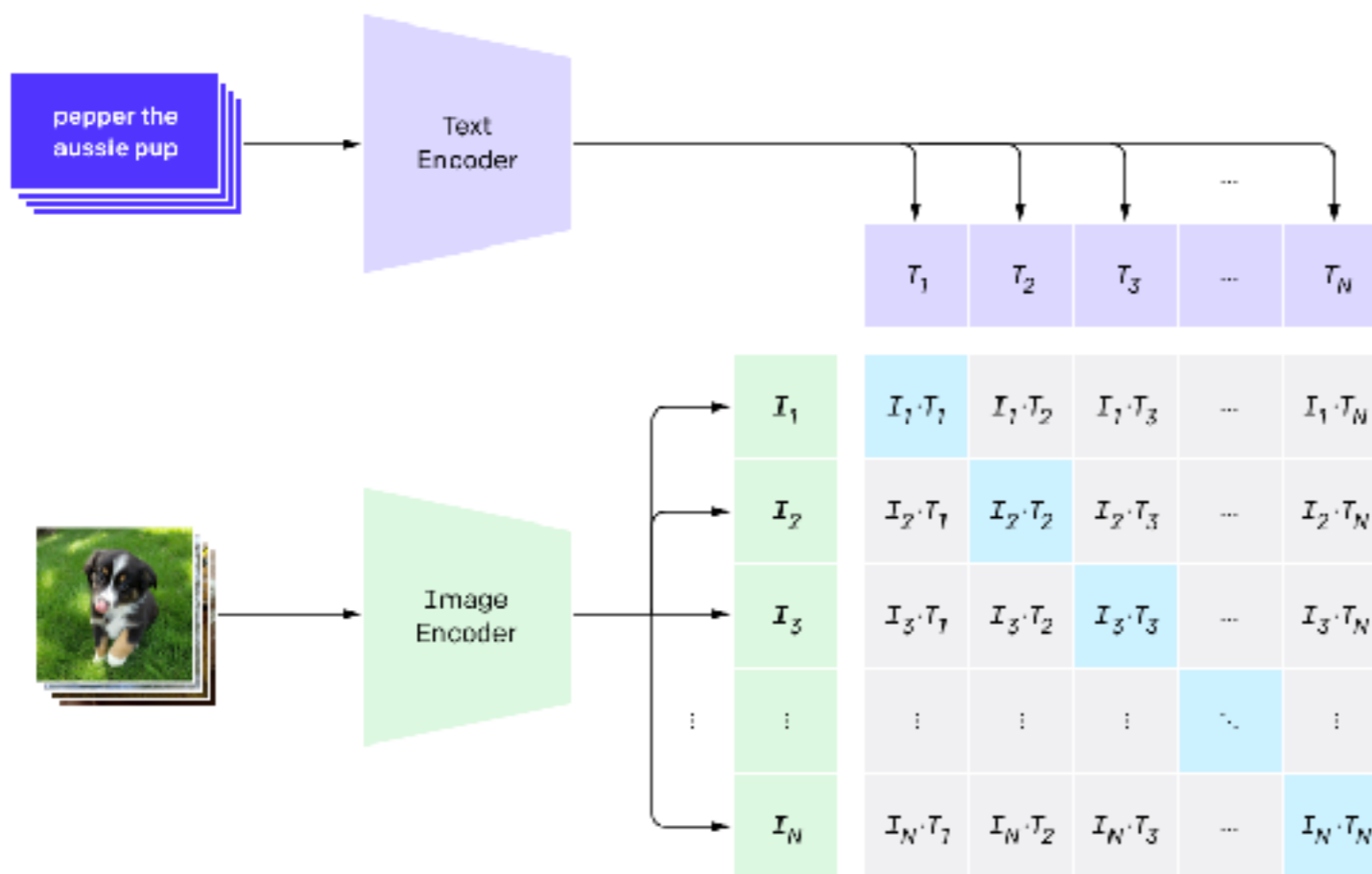


Sentence-guided generation: VQGAN + CLIP

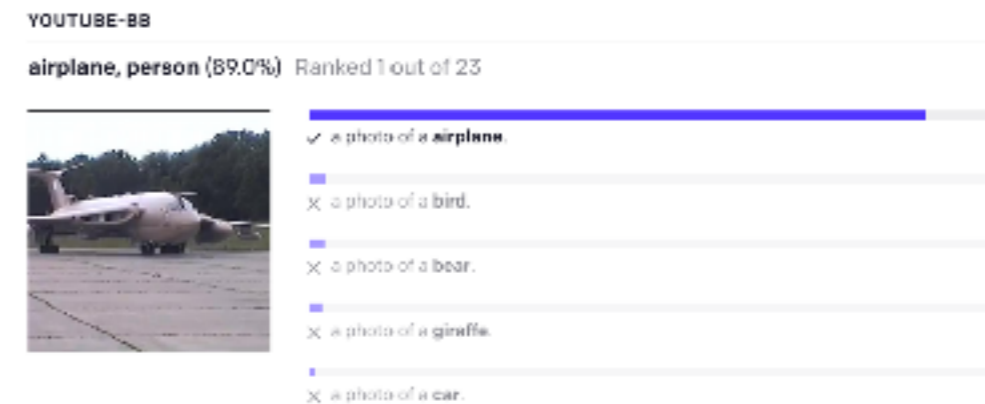
CLIP: embed sentence and image to the same space

1. Contrastive pre-training

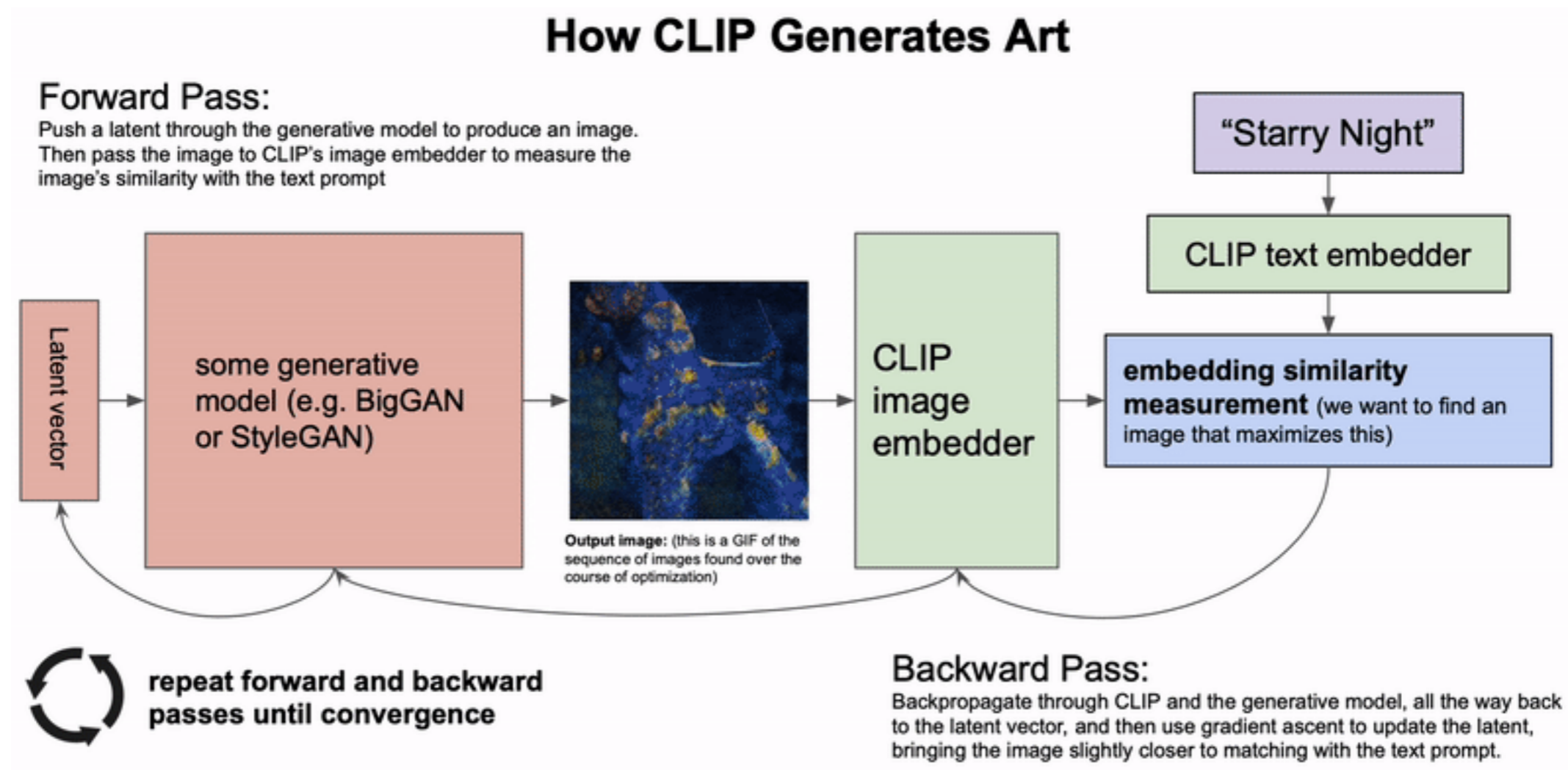
Correct pairs vs incorrect pairs



Original application:
Text choices

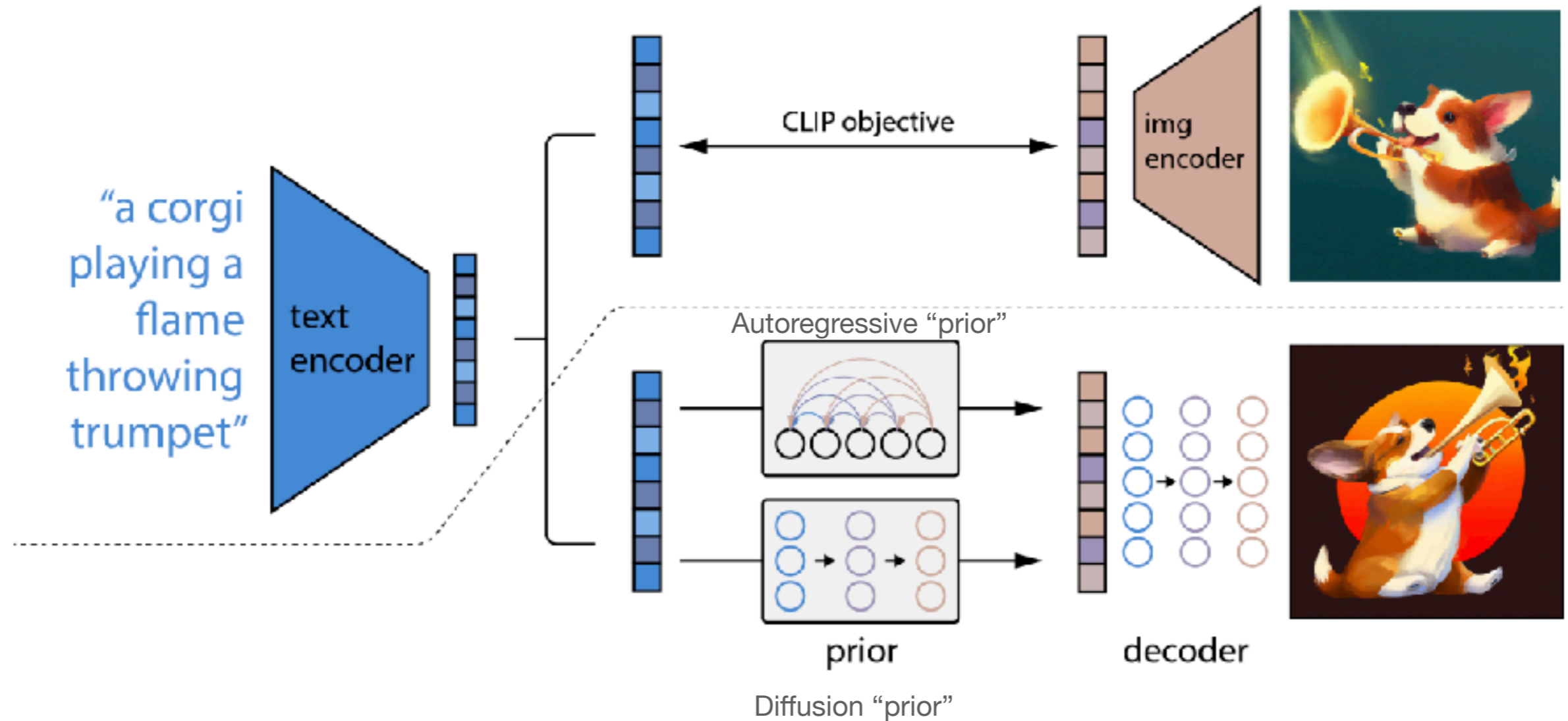


Sentence-guided generation: VQGAN + CLIP



<https://ml.berkeley.edu/blog/posts/clip-art/>

DALLE-2 Replace optimization-based generation with “prior” + decoder



Stable diffusion is almost equivalent to DALLE-2 with diffusion in design

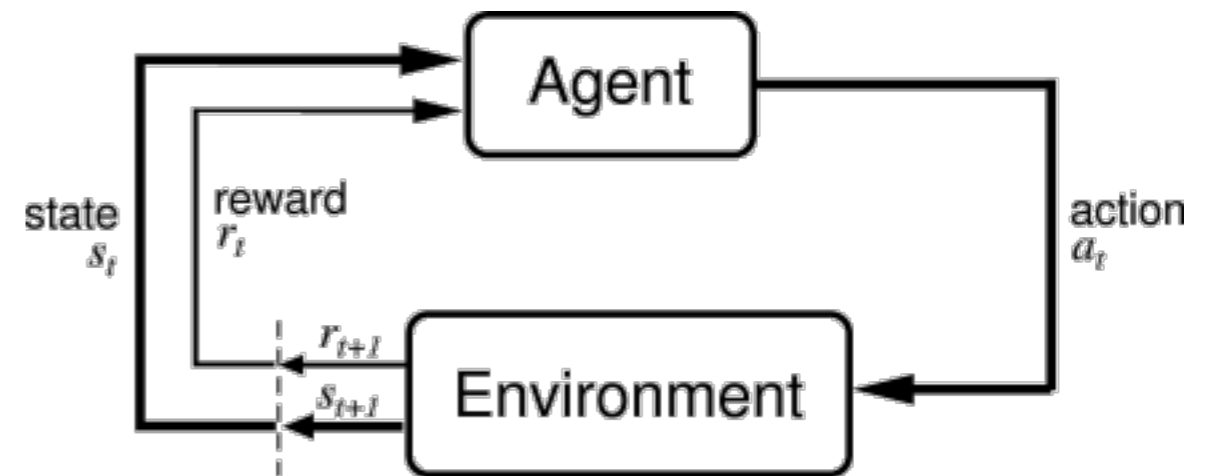
DALLE-2 Replace optimization-based generation with “prior” + decoder



Reinforcement learning



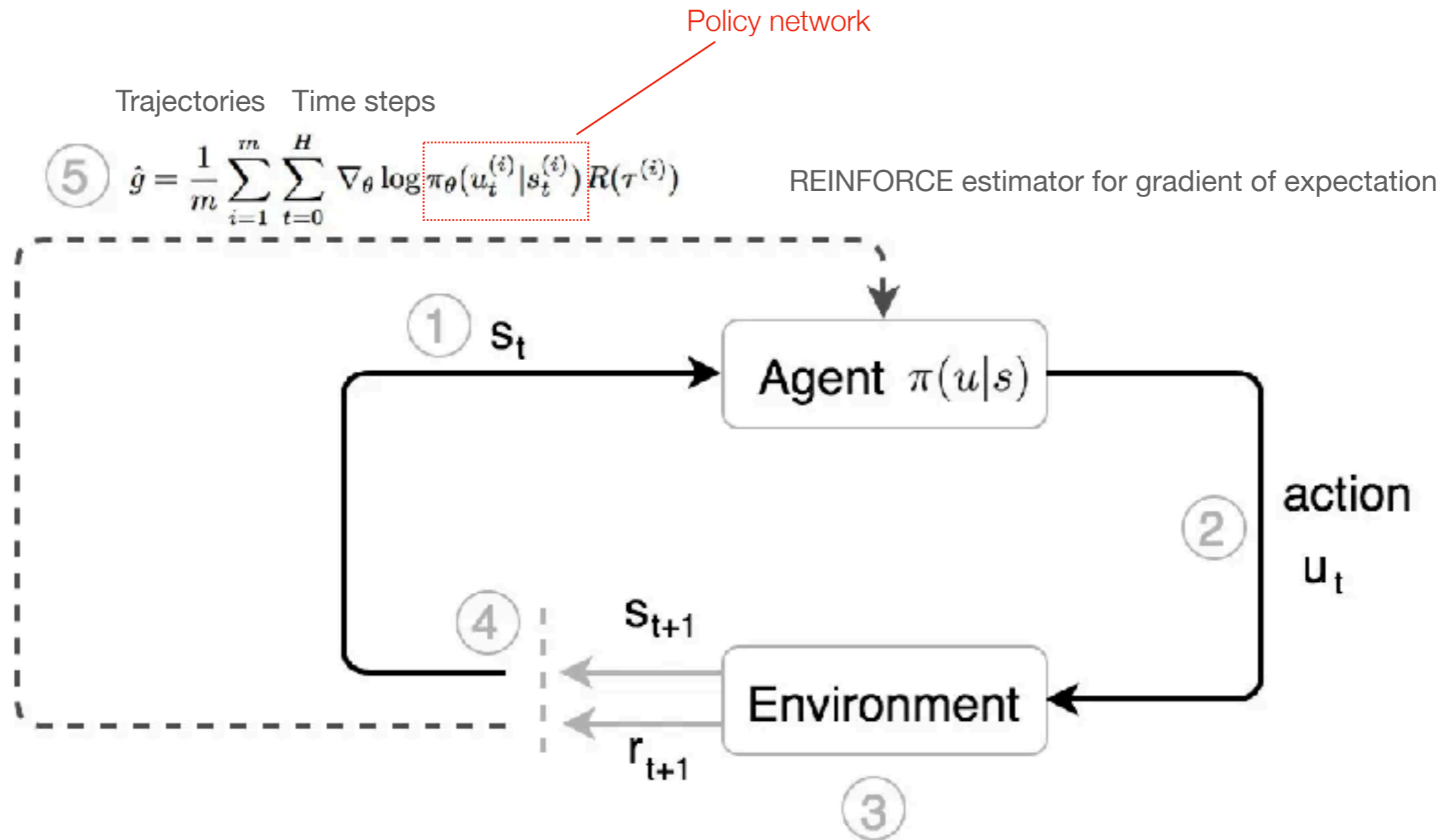
Image credit: daily.doodl @ instagram



Given **state**, choose **action**, get **reward**



Policy gradient: optimize for actions that leads to higher rewards



Improvement to naive policy gradient (Variance reduction) :

subtract a baseline reward (depending on only the state) from the observed reward (Advantage = Observed Reward - Baseline)

Policy gradient is only correct when the training data is generated from the current model weights

on-policy policy gradient: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

off-policy policy gradient: $\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

New updated policy $(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \sim \pi_{\theta}(\mathbf{s}_t, \mathbf{a}_t)$

Old policy used to generate data

+ preventing the policy from moving too far away from the old policy

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_k}(a_t | s_t)} A^{\tau_{\theta_k}}(s_t, a_t), g(t, A^{\tau_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_{\phi}(s_t) - \hat{R}_t)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**

Deep Q learning: Predict future rewards with deep networks

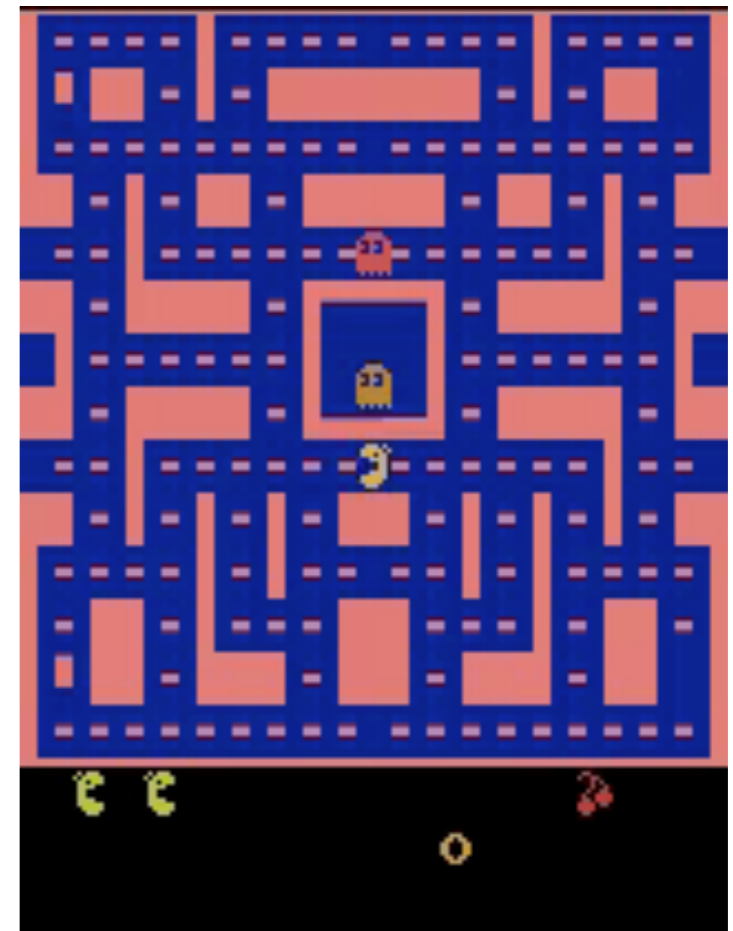
Q Learning

$Q(\text{state}, \text{action}) = \text{maximal future rewards (with the optimal actions)}$

Bellman equation

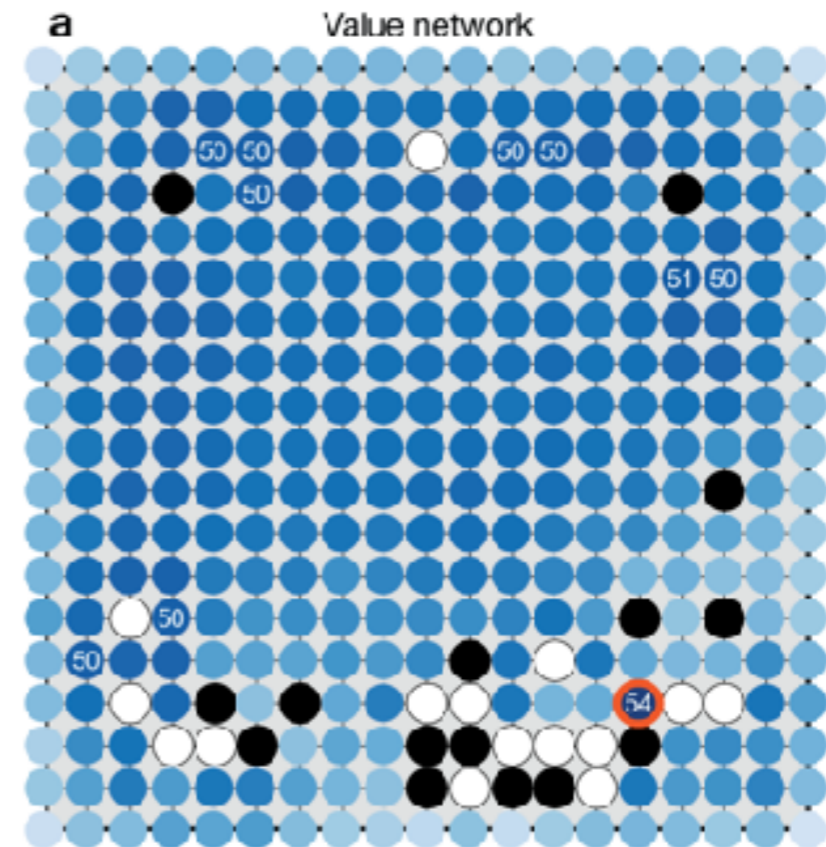
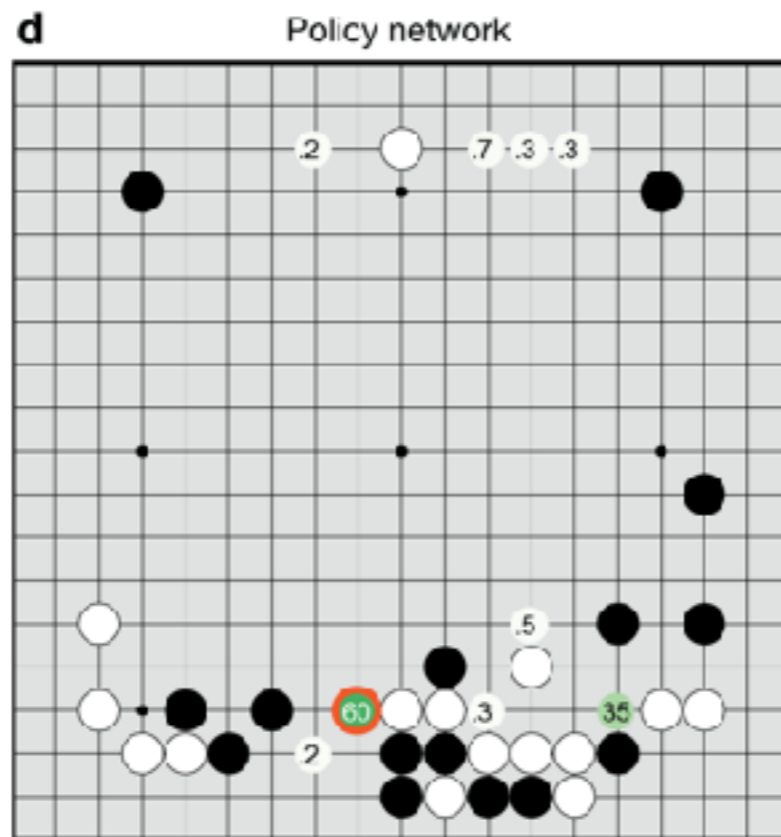
$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

Training: minimize MSE



AlphaGo - surpass human-level game playing in Go

(the nature publication version)



SL policy network: predict expert human moves
convnet / GLM



RL policy network: optimized by self-play
convnet

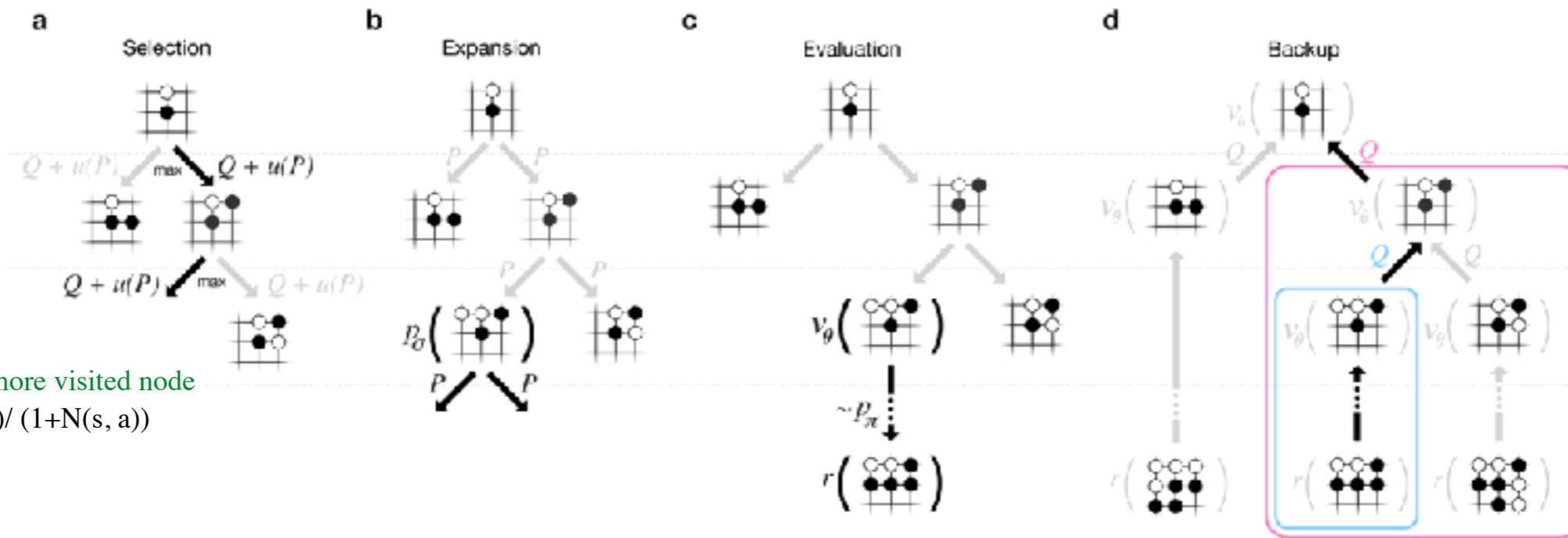
Value network: predict outcome of self-play
convnet



REINFORCE algorithm (Williams, 1992)

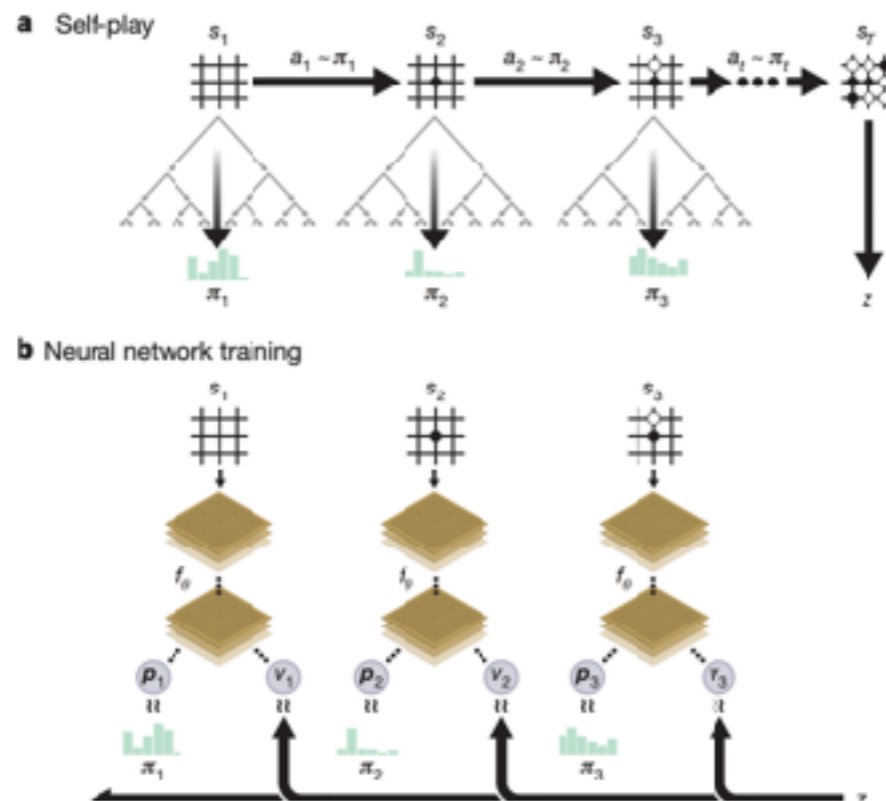
Silver et al., 2016, Mastering the game of Go with deep neural networks and tree search

AlphaGo - Monte carlo tree search



Discounting more visited node
 $u(s, a) \propto P(s, a) / (1 + N(s, a))$

Final game play:
 $\pi_a \propto N(s, a)^{1/\tau}$



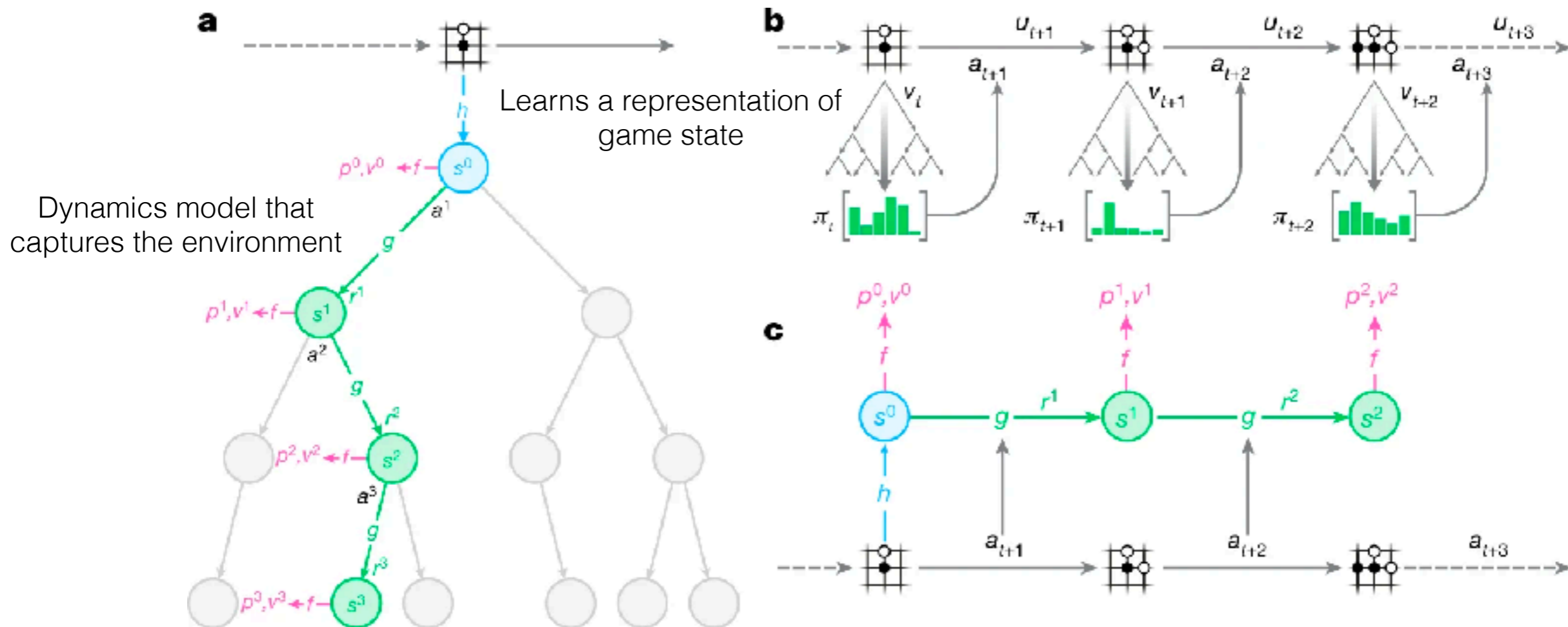
AlphaGo Zero:

Train policy network using MCTS policy

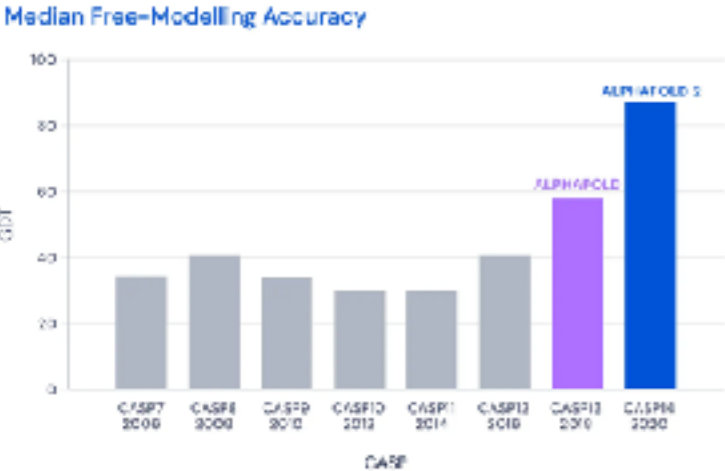
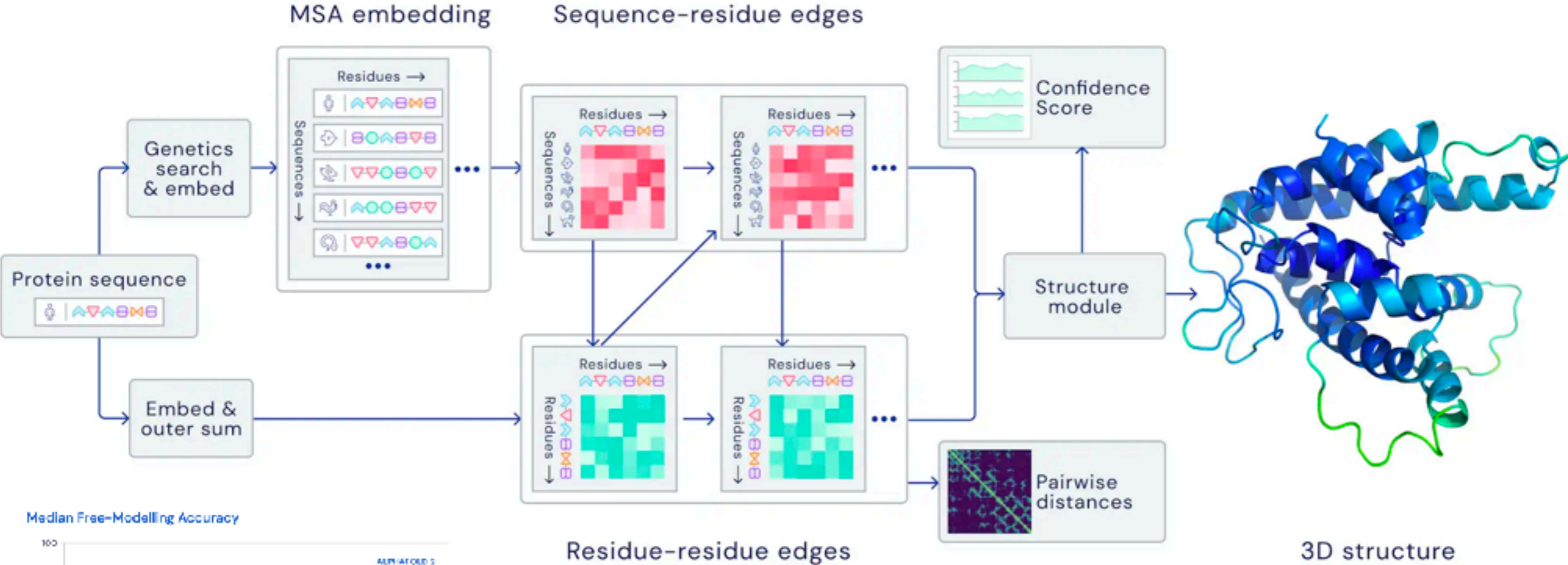
Learning without access to environment during planning (MCTS)

MuZero:

Training model without access to the environment during MCTS

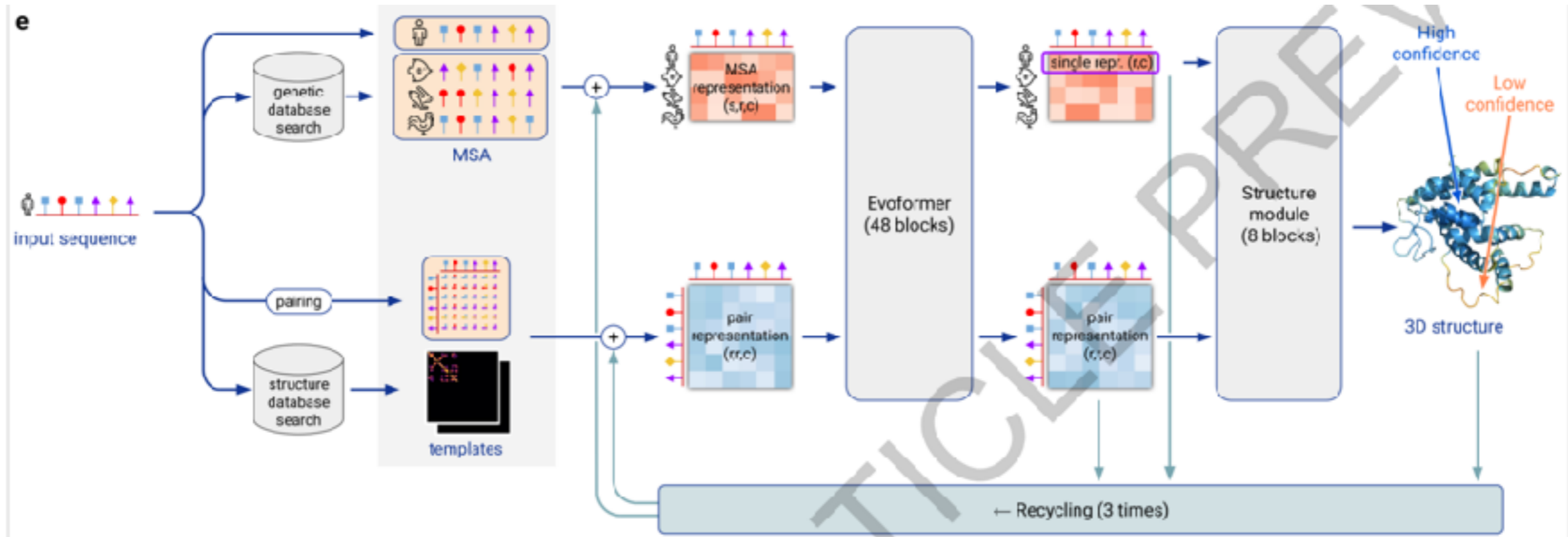


AlphaFold2 - X-ray level atomic resolution prediction

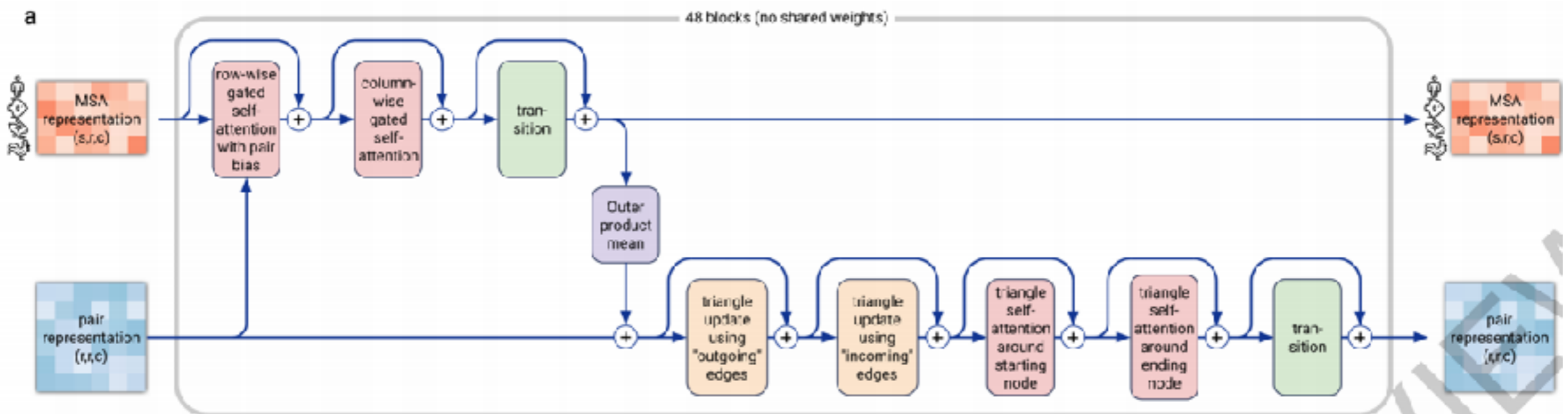


AlphaFold v2.0

Overall structure

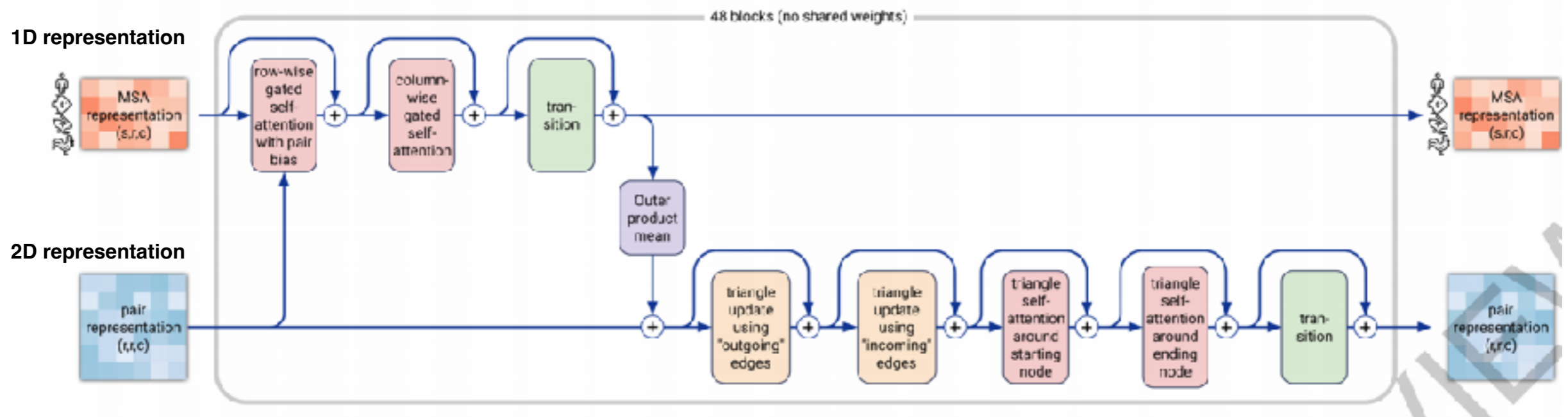


Sequence model structure



AlphaFold v2.0

Sequence model structure

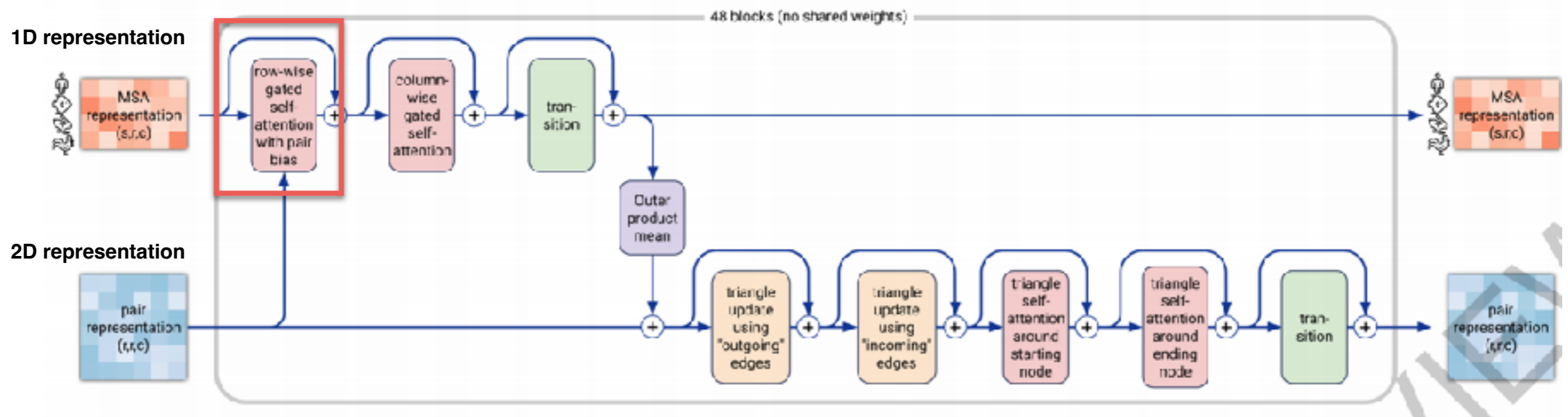


Template protein data also included
(torsion angles)

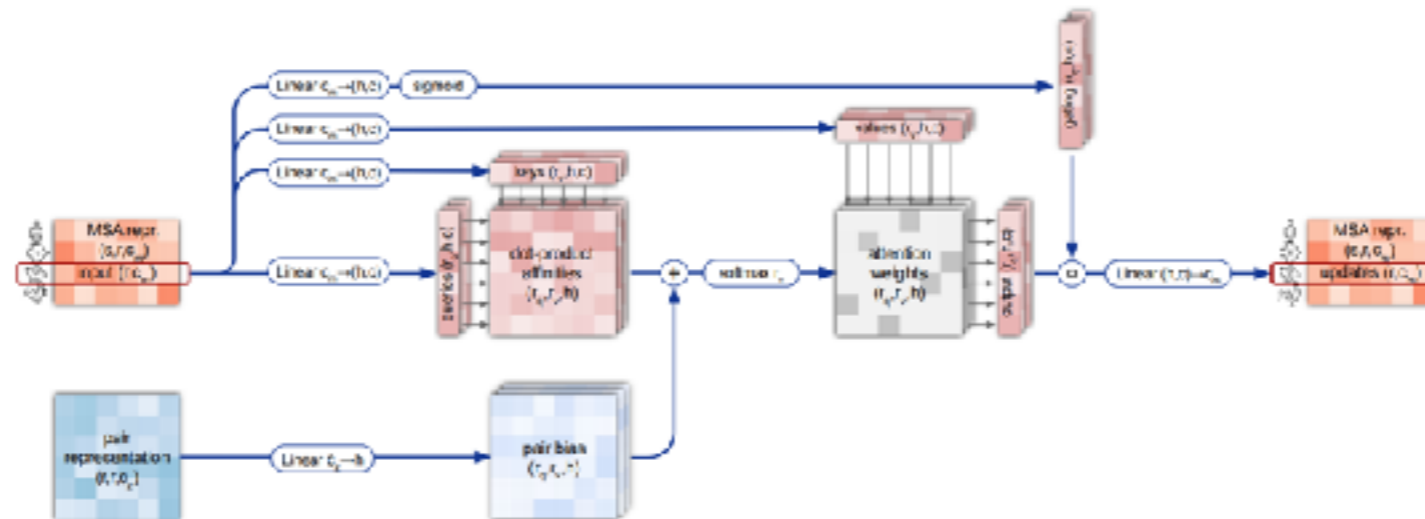
+

AlphaFold v2.0 : model structure

Sequence model structure



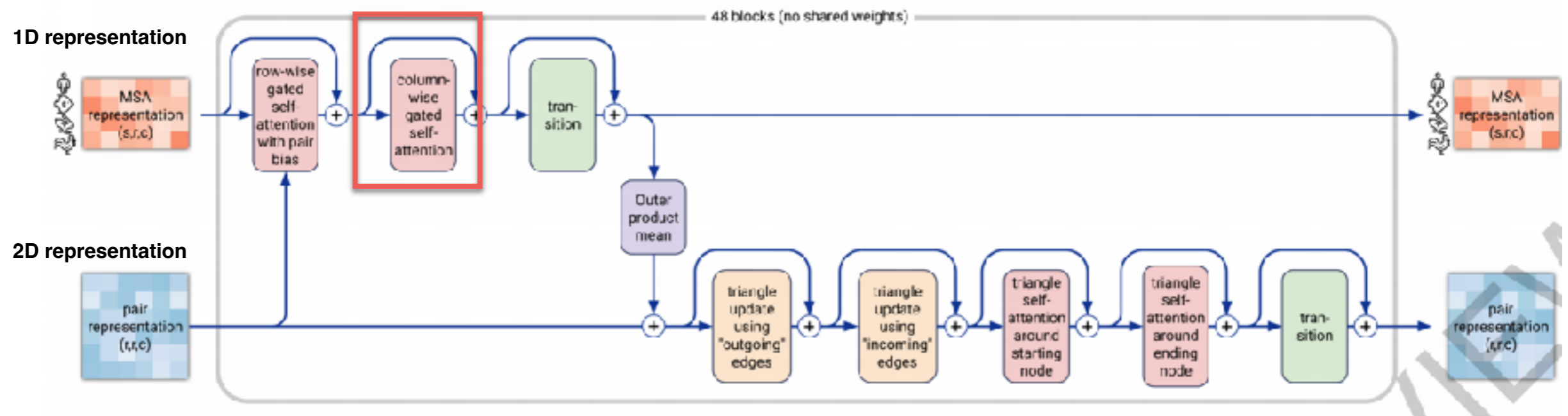
Gated transformer + linear transformed 2D bias



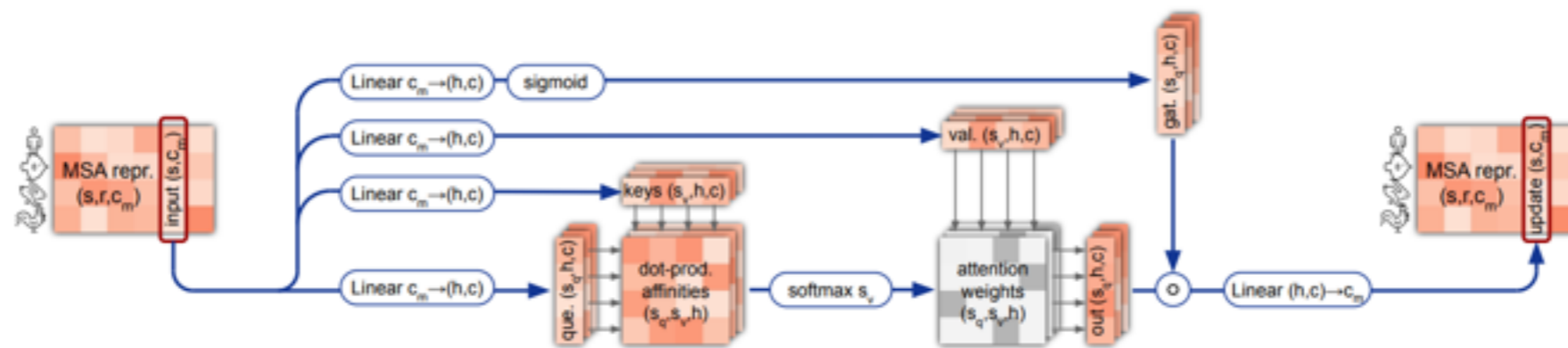
Supplementary Figure 2 | MSA row-wise gated self-attention with pair bias. Dimensions: s: sequences, r: residues, c: channels, h: heads.

AlphaFold v2.0 : model structure

Sequence model structure



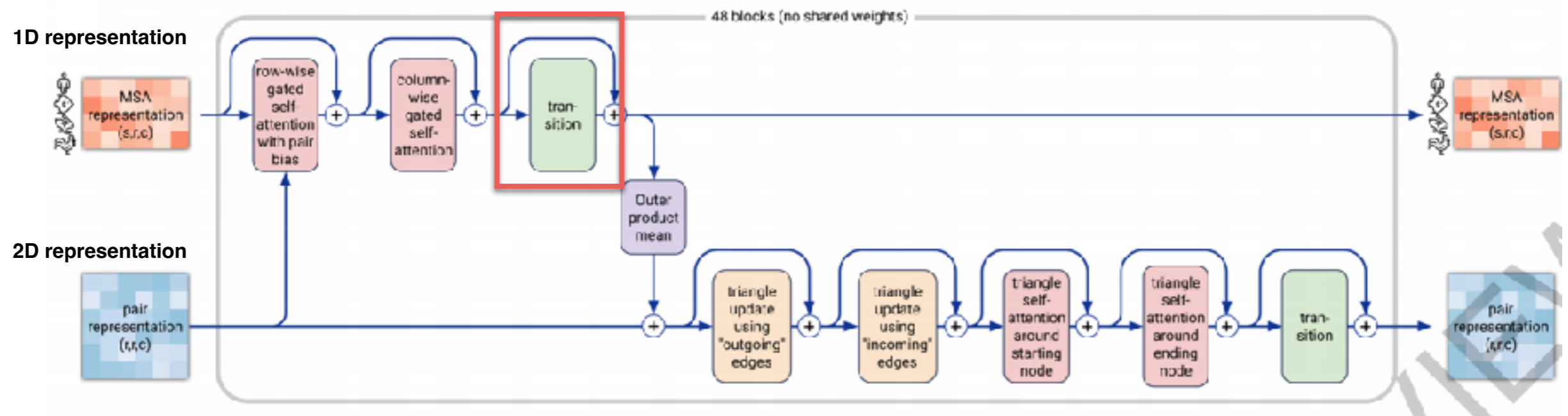
Gated transformer



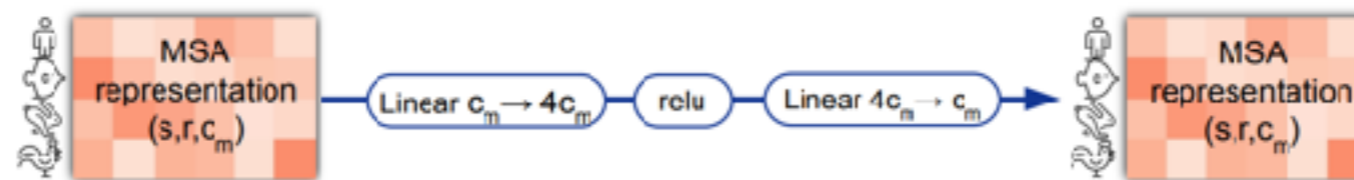
Supplementary Figure 3 | MSA column-wise gated self-attention. Dimensions: s: sequences, r: residues, c: channels, h: heads.

AlphaFold v2.0 : model structure

Sequence model structure



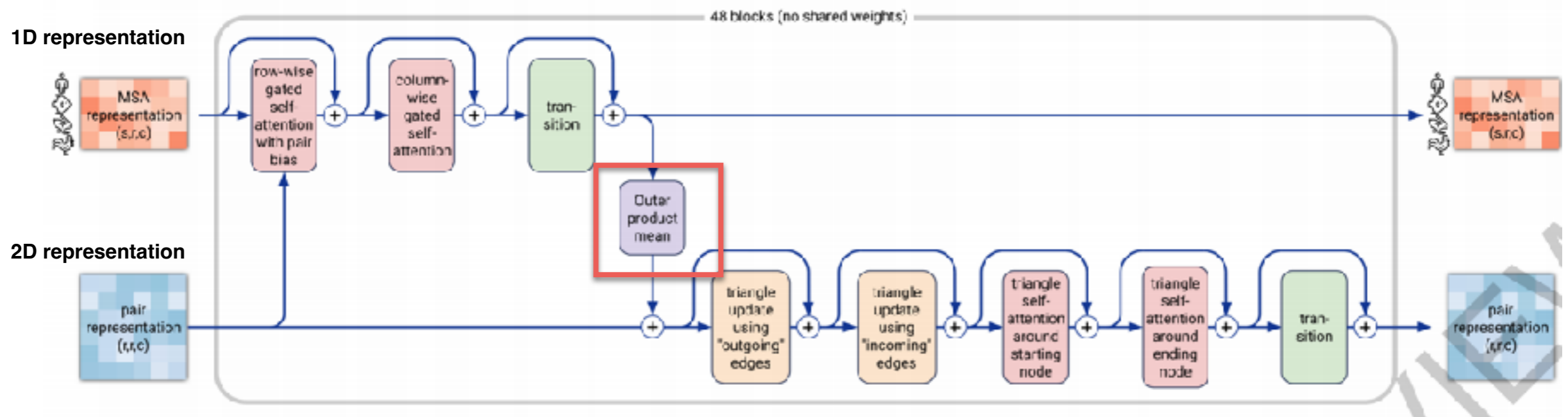
Linear - ReLU- Linear (with LayerNorm)



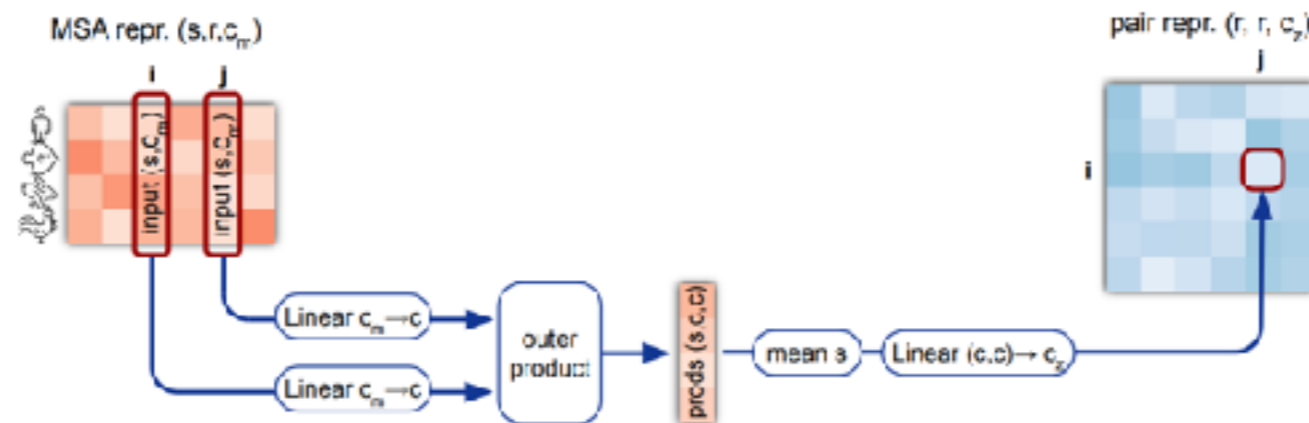
Supplementary Figure 4 | MSA transition layer. Dimensions: s: sequences, r: residues, c: channels.

AlphaFold v2.0 : model structure

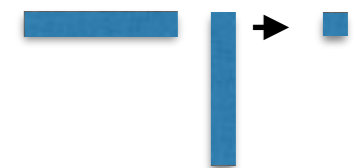
Sequence model structure



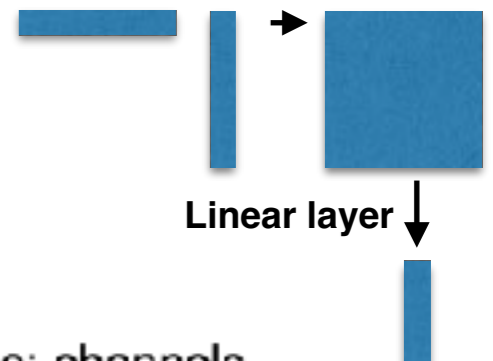
Outer product -> linear : more flexible than inner product



Dot product (inner product)



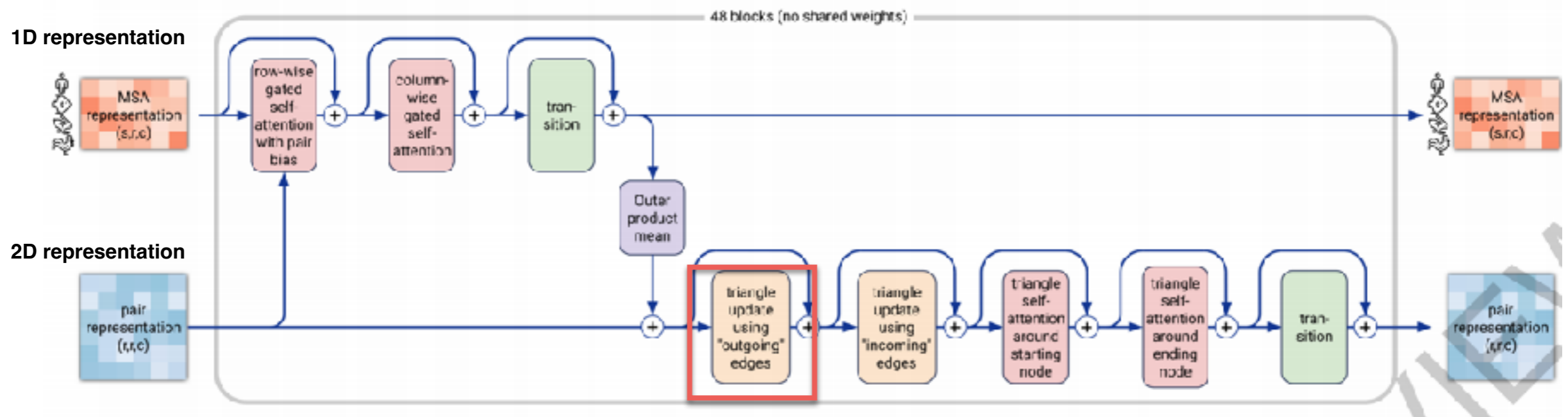
Outer product



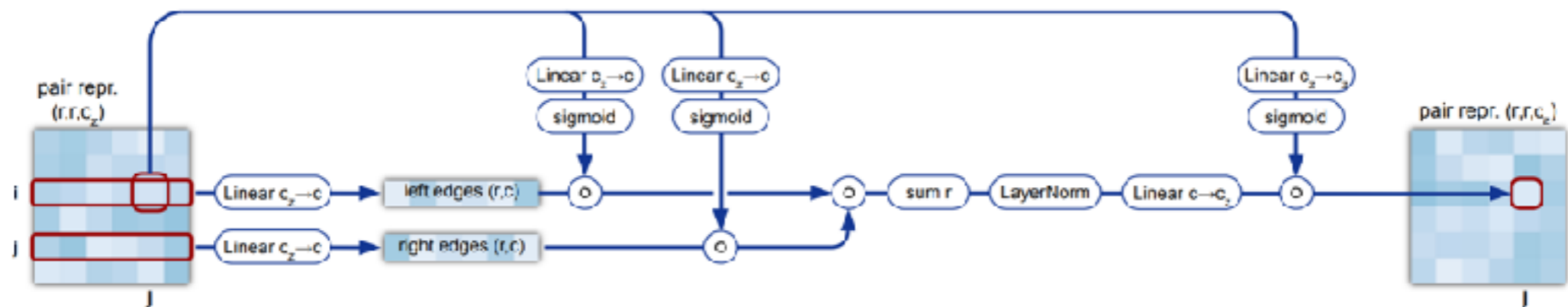
Supplementary Figure 5 | Outer product mean. Dimensions: s: sequences, r: residues, c: channels.

AlphaFold v2.0 : model structure

Sequence model structure



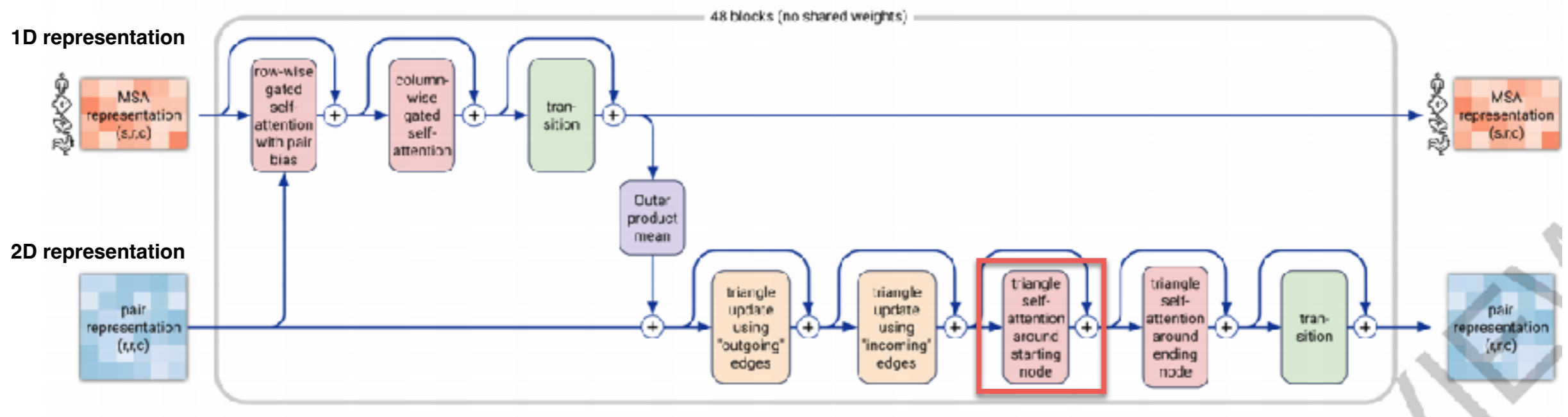
Similar to row-wise gated self attention



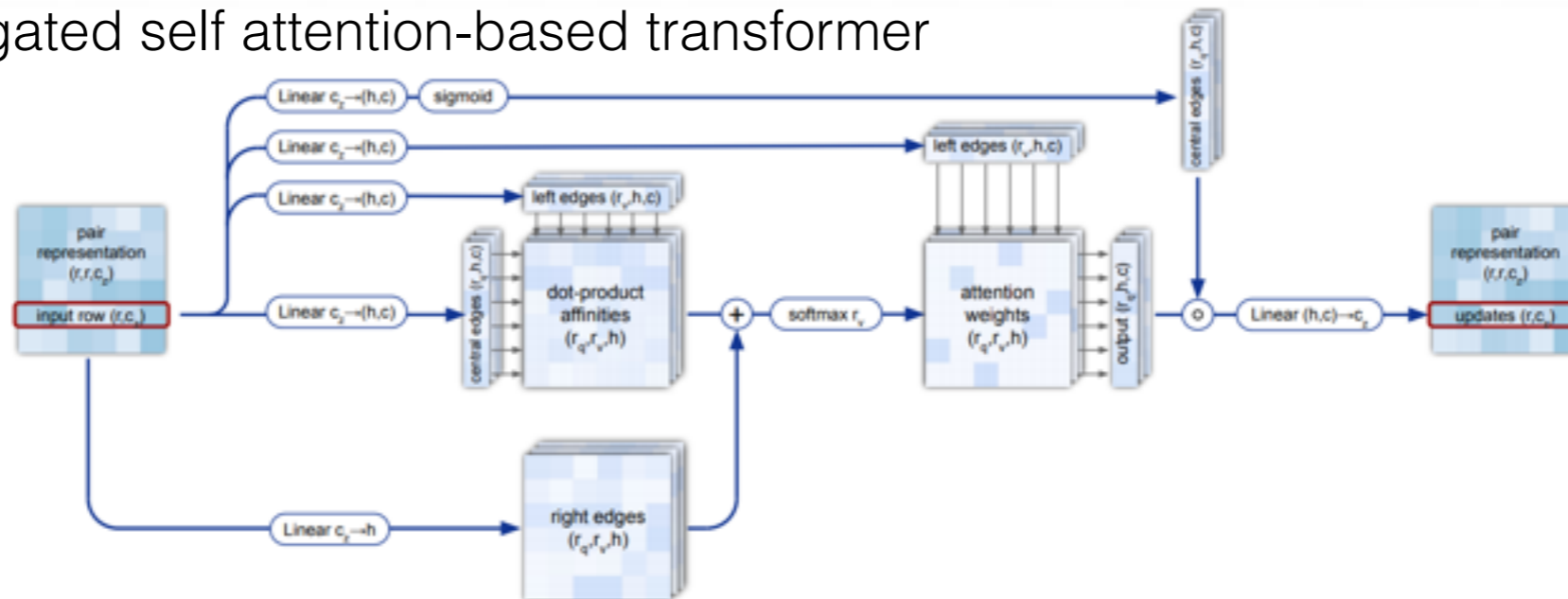
Supplementary Figure 6 | Triangular multiplicative update using "outgoing" edges. Dimensions: r: residues, c: channels.

AlphaFold v2.0 : model structure

Sequence model structure



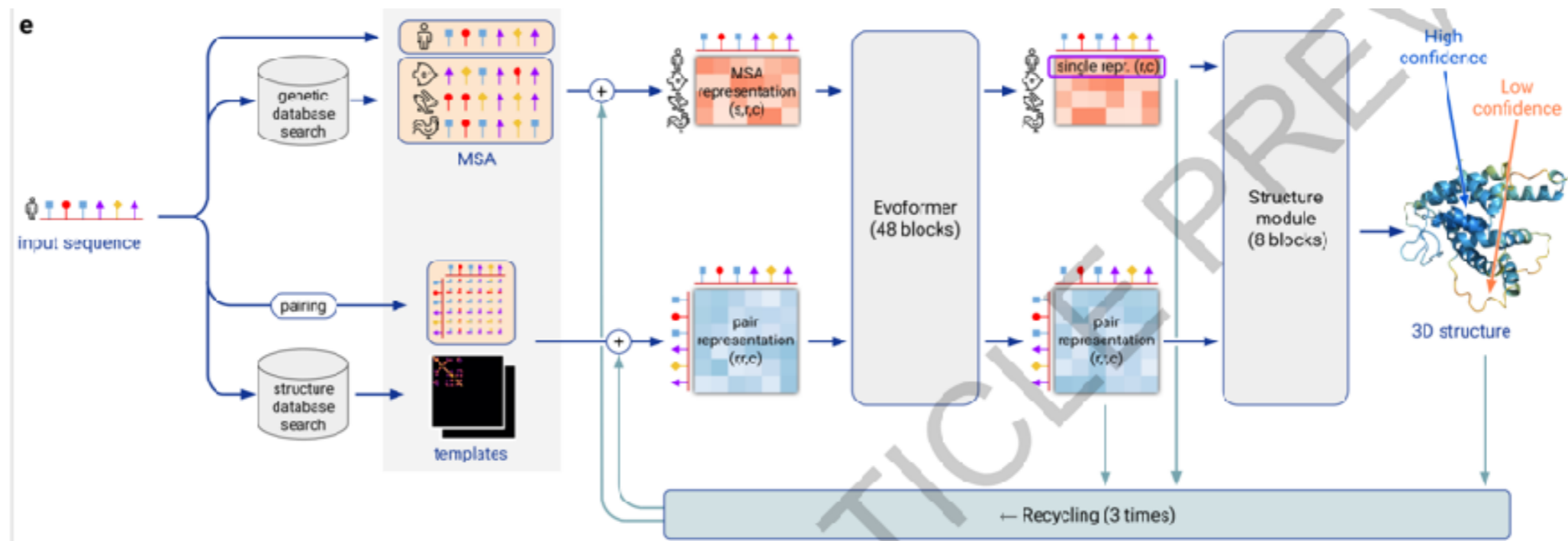
Similar to row-wise gated self attention-based transformer



Supplementary Figure 7 | Triangular self-attention around starting node. Dimensions: r: residues, c: channels, h: heads

AlphaFold v2.0 : Recycling mechanism

Learn to iteratively refine rather than jumping right at the results



Simple approach:

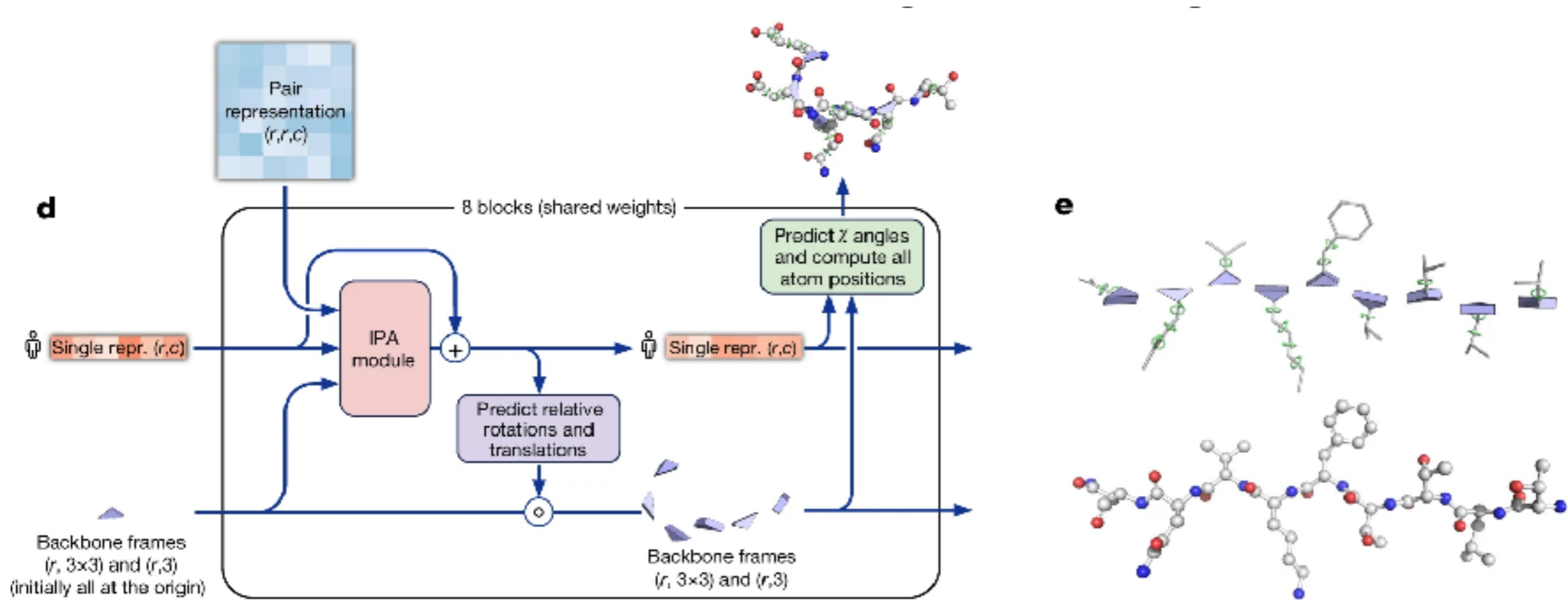
$$X \rightarrow Y$$

Recycled / recurrent prediction:

$$X + Y^* \rightarrow Y$$

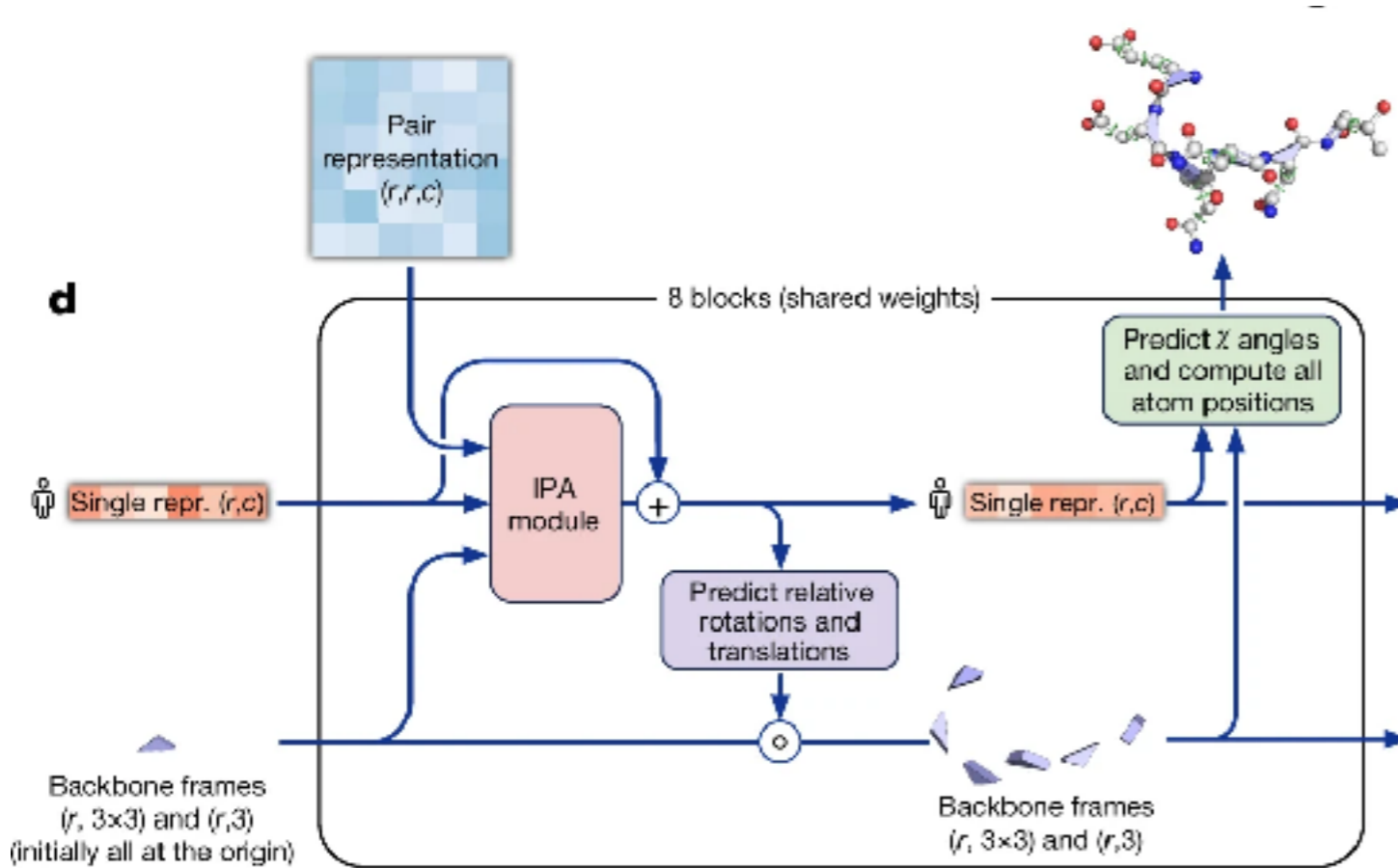
AlphaFold v2.0 : Structure module

From intermediate representations to 3D coordinates

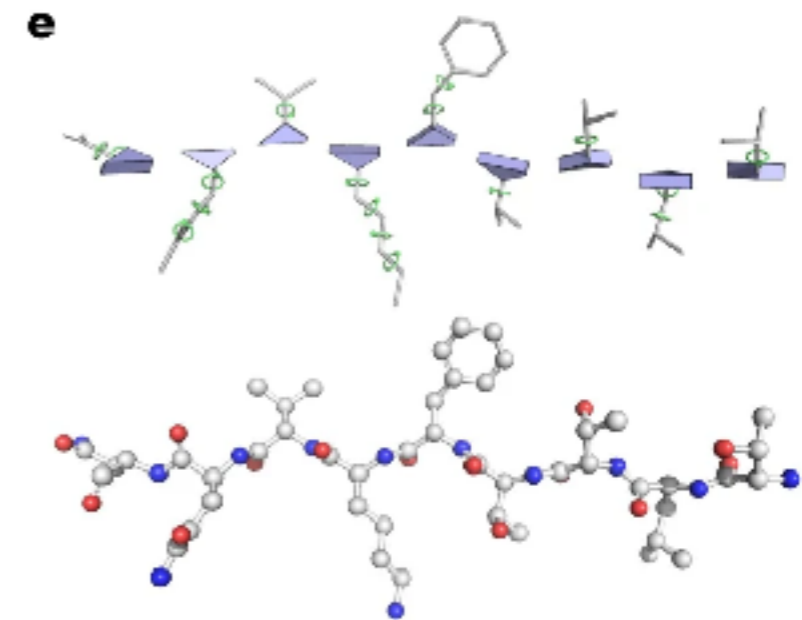


AlphaFold v2.0 : Structure module

From intermediate representations to 3D coordinates

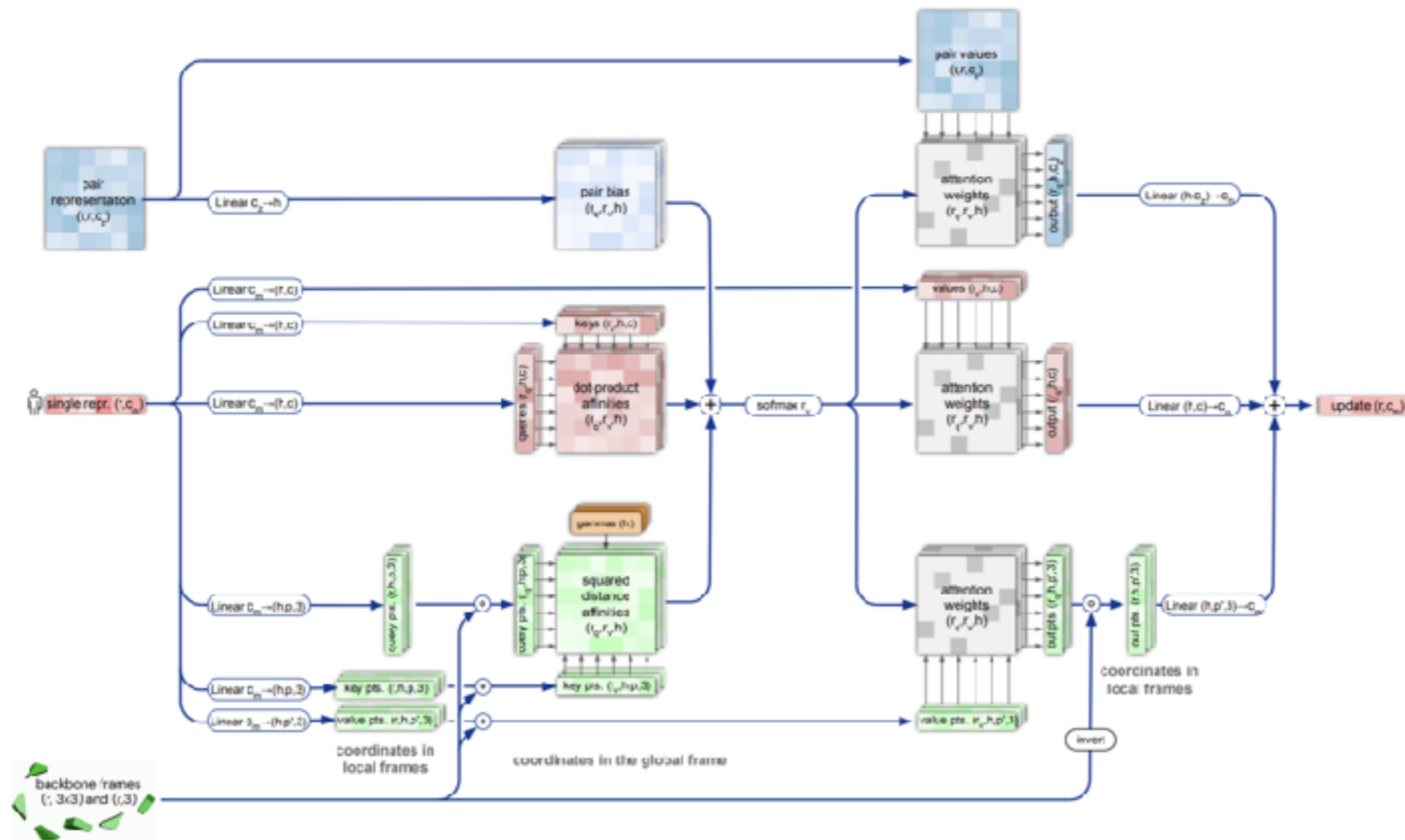


Side chain angles are computed by per-residue network



AlphaFold v2.0 : Structure module

Invariant Point Attention module



equivariant to the rotation of backbone frames

Supplementary Figure 8 | Invariant Point Attention Module. (top, blue arrays) modulation by the pair representation. (middle, red arrays) standard attention on abstract features. (bottom, green arrays) Invariant point attention. Dimensions: r: residues, c: channels, h: heads, p: points.