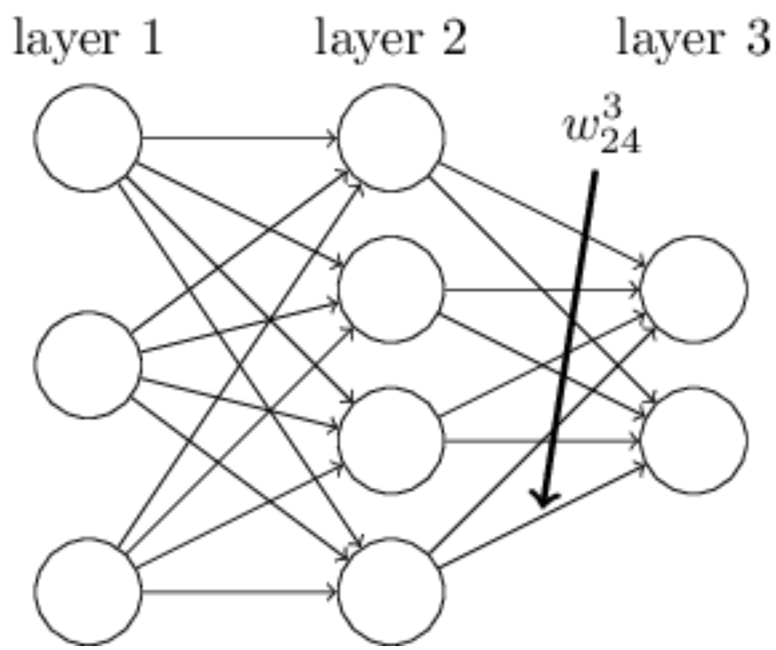
Advanced Concepts in Deep Learning

Plan for Day 1

- **Foundations of deep learning**
- **Emerging architectures: Transformers, Graph Networks, and more**
- Deep reinforcement learning + case study: AlphaGo (if time permits)
- Case study: AlphaFold (if time permits)
- Group discussion: Design deep learning approaches for your problems.

The archetype



w^l_{jk} 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

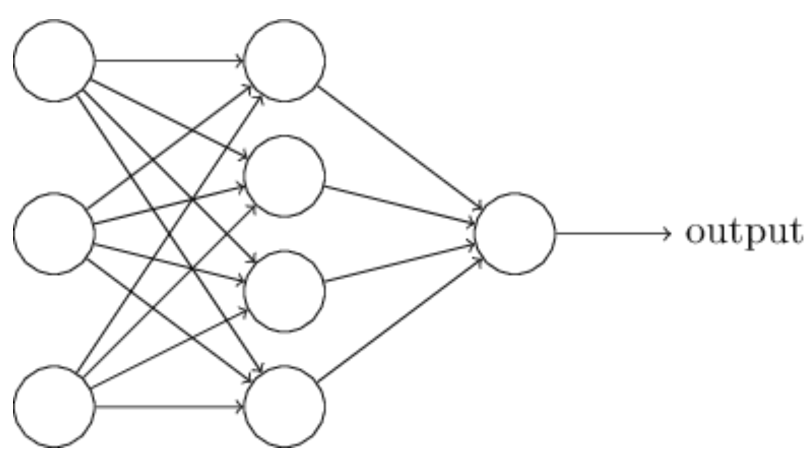
How to train it

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

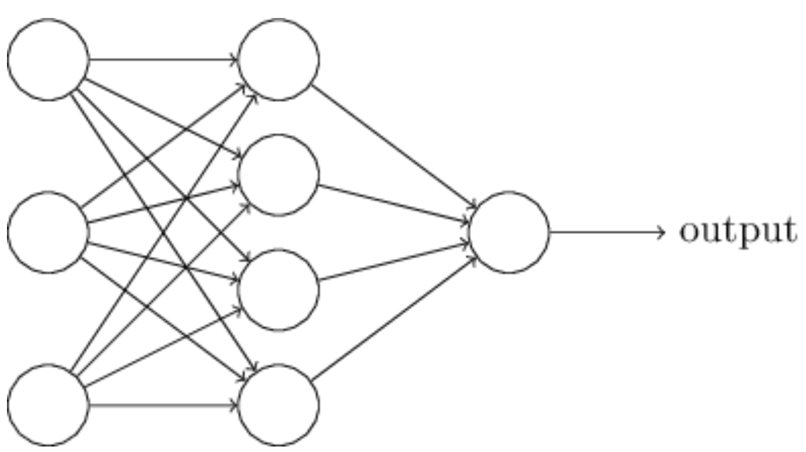


Universal Approximation Bounds for Superpositions of a Sigmoidal Function

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

$$\hat{R}_m(\mathcal{F}) = \mathsf{E}_\sigma\left[\sup_{f\in\mathcal{F}}\left(\frac{1}{m}\sum_{i=1}^{m}\sigma_i f(z_i)\right)\right]$$

Test Error Training Error

$$\mathsf{E}_D[f(z)] \le \hat{\mathsf{E}}_S[f(z)] + 2R_m(\mathcal{F}) + \sqrt{\frac{\ln(1/\delta)}{m}}.$$

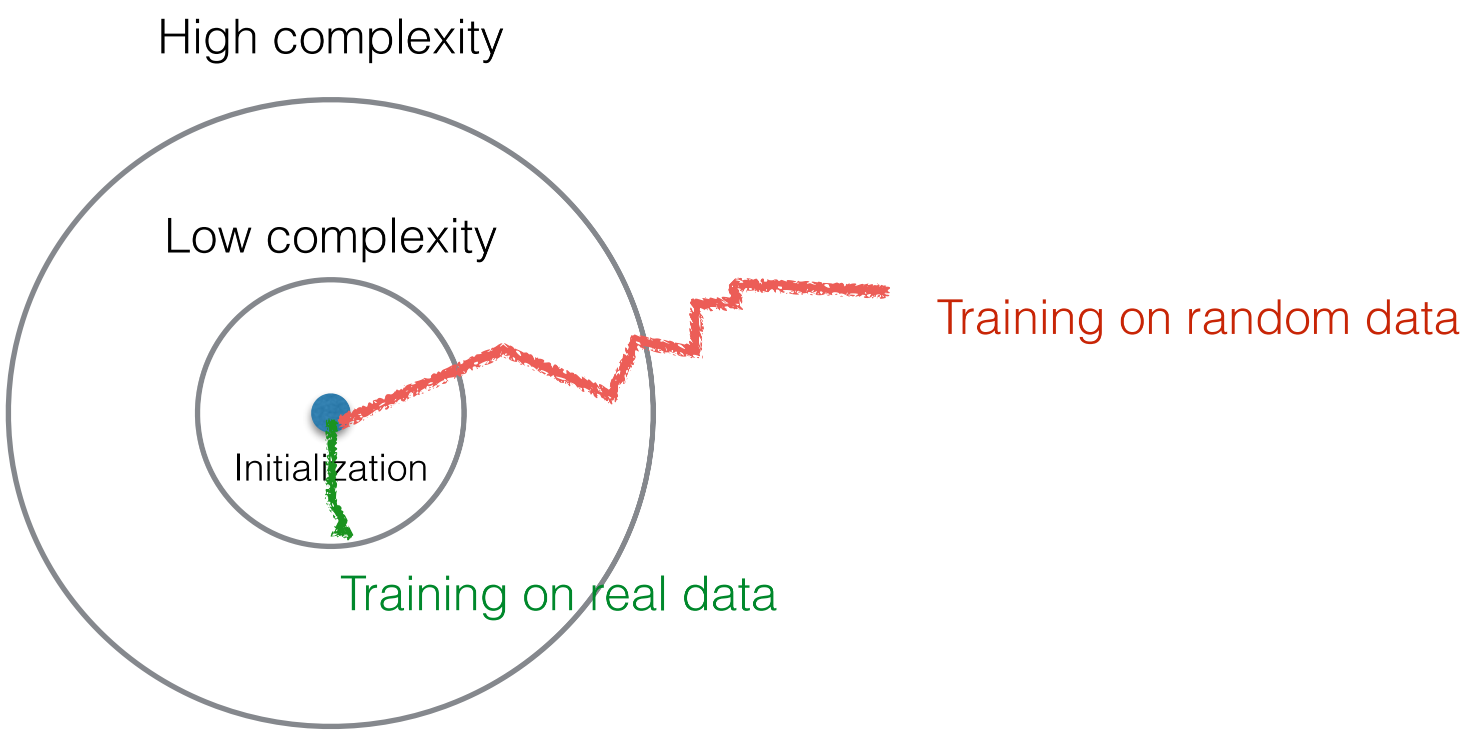
with probability at least 1- δ

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

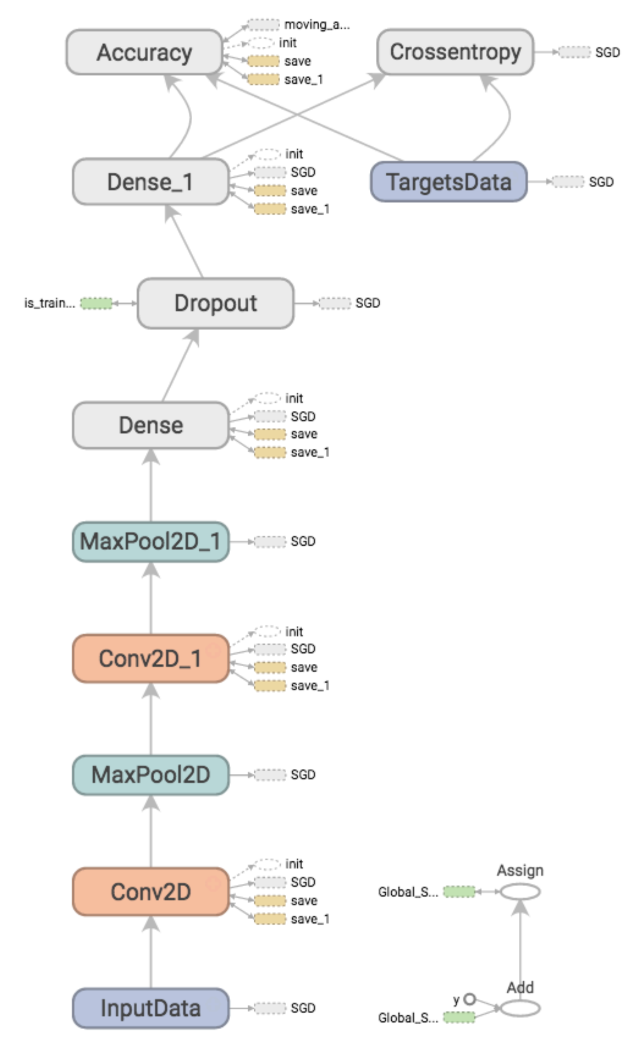
It will be interesting to understand when this assumption fails
(e.g., prime factorization)

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
  - Mixing programing language with computational graph (conditionals, loops, etc with mathematical functions)
  - Higher-order derivative (e.g. Hessian)

Computation of Gradient: Automatic differentiation

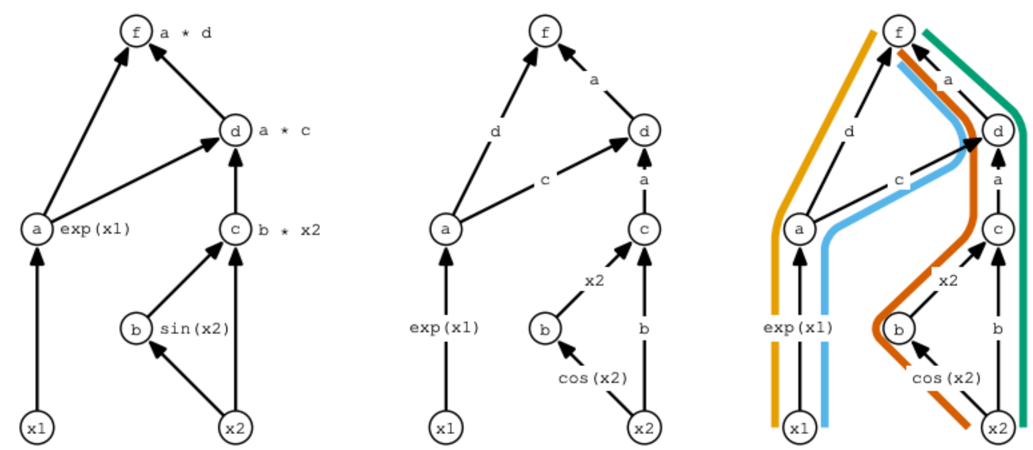
We only need stochastic gradient, so why not **randomized** automatic differentiation?

Unbiased estimator of gradient

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

```
from math import sin, exp

def f(x1, x2):
    a = exp(x1)
    b = sin(x2)
    c = b * x2
    d = a * c
    return a * d
```



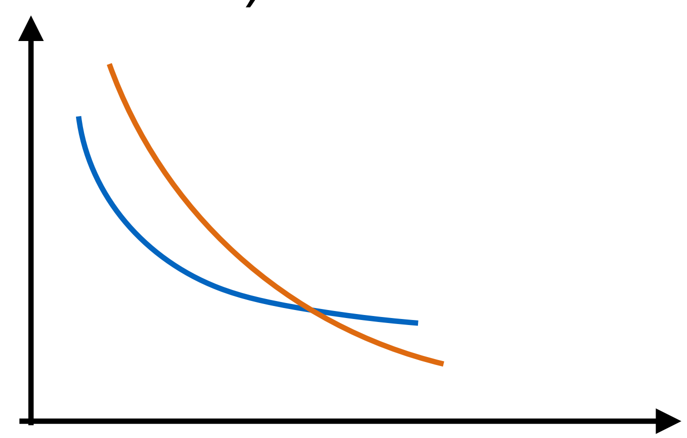
(a) Differentiable Python function (b) Primal graph (c) Linearized graph (d) Bauer paths

Sparse implementation similar to dropout in backward pass

Randomized Automatic Differentiation, Deniz Oktay, Nick McGreivy, Joshua Aduol, Alex Beatson, Ryan P. Adams

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

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

$$\eta_t \sim N(0, \epsilon_t) \qquad (4)$$

MCMC by Langevin dynamics

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

$$\eta_t \sim N(0, \epsilon) \qquad (3)$$

Connection between Stochastic Gradient Descent and Bayesian inference

SGD as VI

Stochastic Gradient Descent as Approximate Bayesian Inference, Mandt, 2017

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

$$C(\theta) \approx C = BB^\top$$

S is mini-batch size

SGD is then equivalent to stochastic process

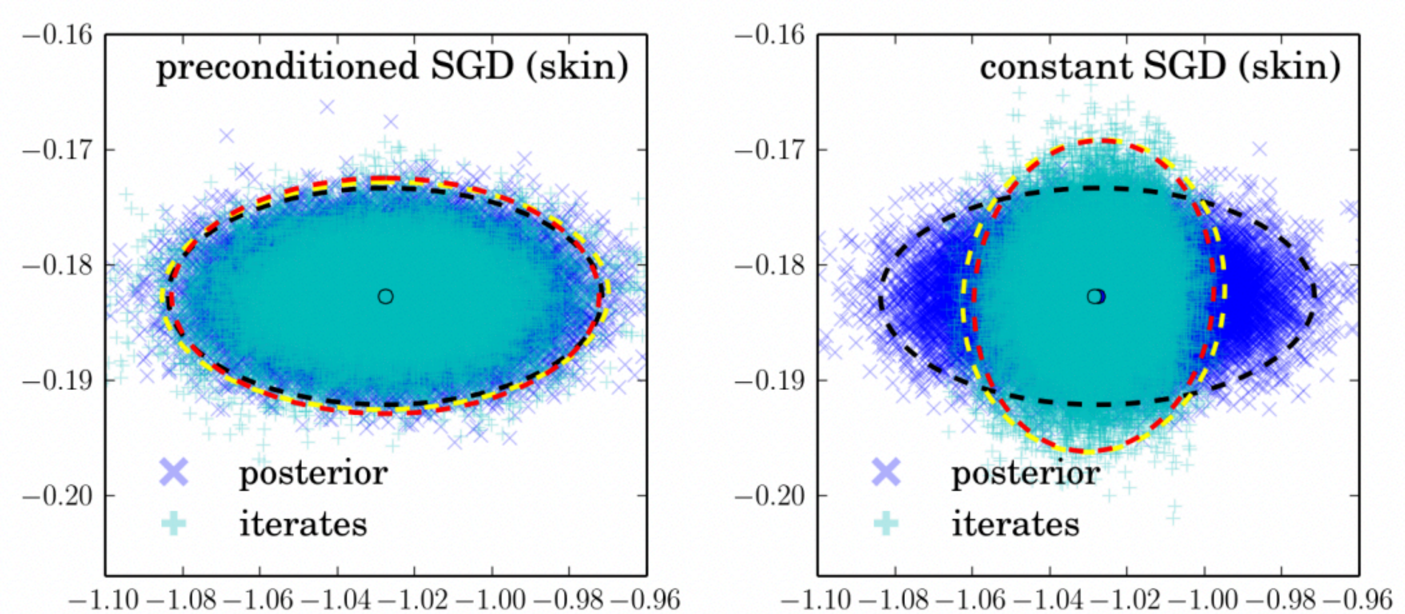
$$d\theta(t) = -\epsilon g(\theta)dt + \frac{\epsilon}{\sqrt{S}} B\, dW(t)$$

which converge to Gaussian stationary distribution with covariance

Optimal preconditioning matrix

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

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

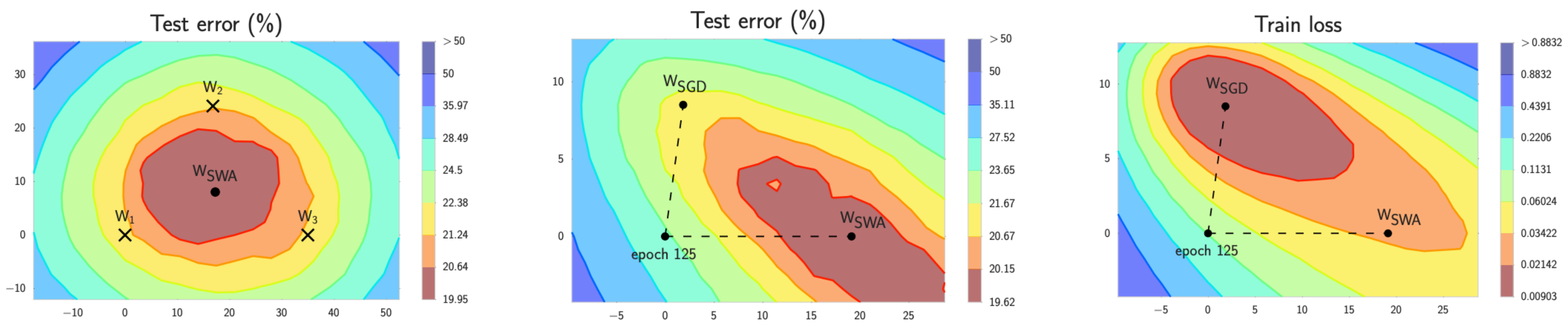


Optimal learning rate

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

SGD should not be considered simply as approximate gradient descent

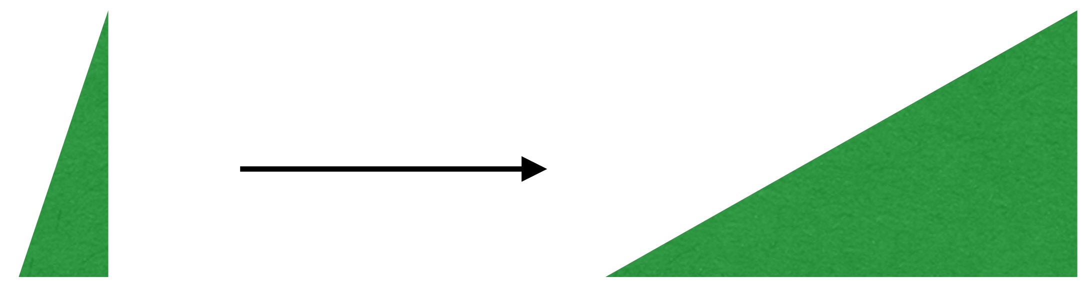
Find the center of the posterior:
Stochastic weight averaging



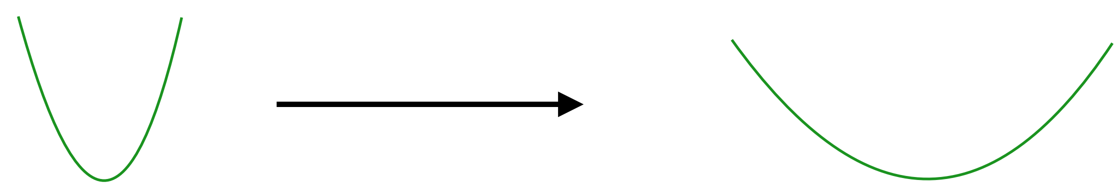
SWA can be seen as a particular type of learning rate decay 1-N/N_max

Optimization: scale invariance

Naive gradient descent is not scale-invariant



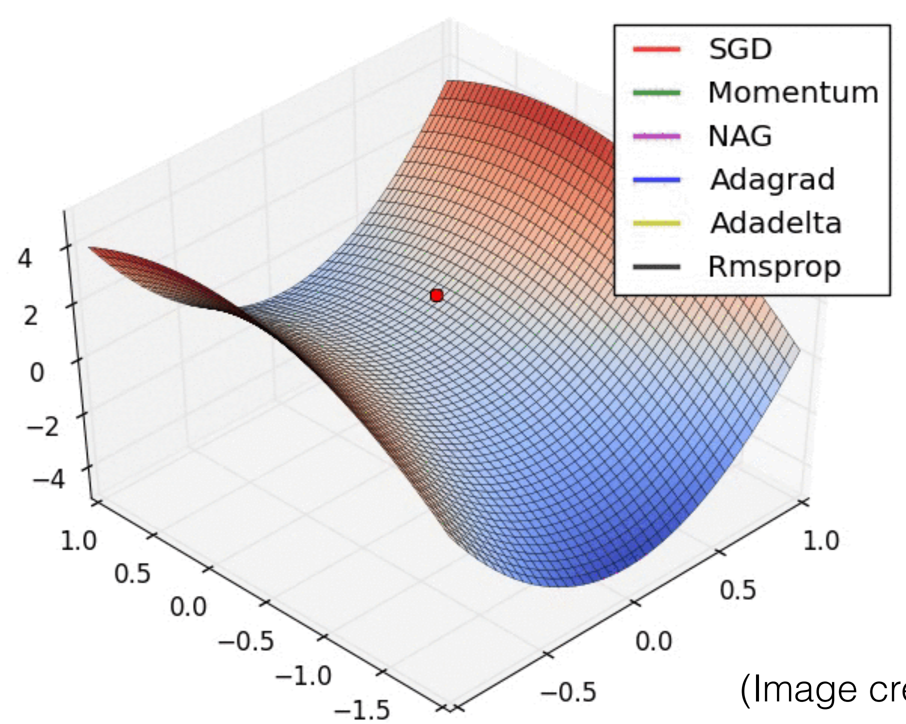
Known solution: use curvature of the surface (second order methods)



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

$$x_{k+1} = x_k - [f''(x_k)]^{-1} f'(x_k)$$

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

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

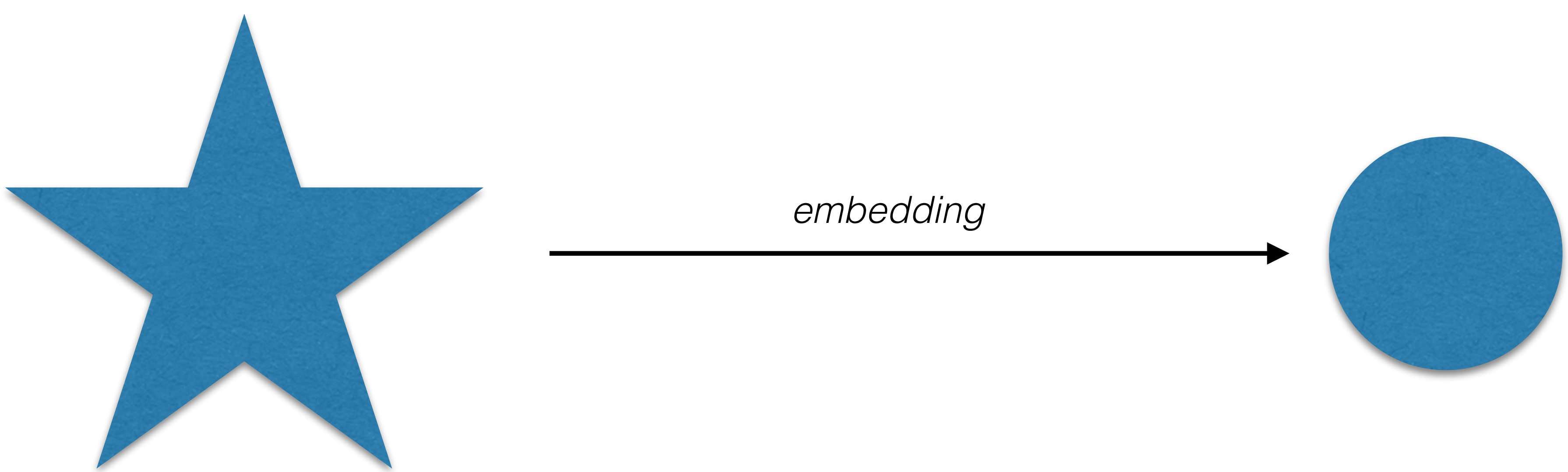
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2.$$

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

http://sebastianruder.com/optimizing-gradient-descent/

Learning representations

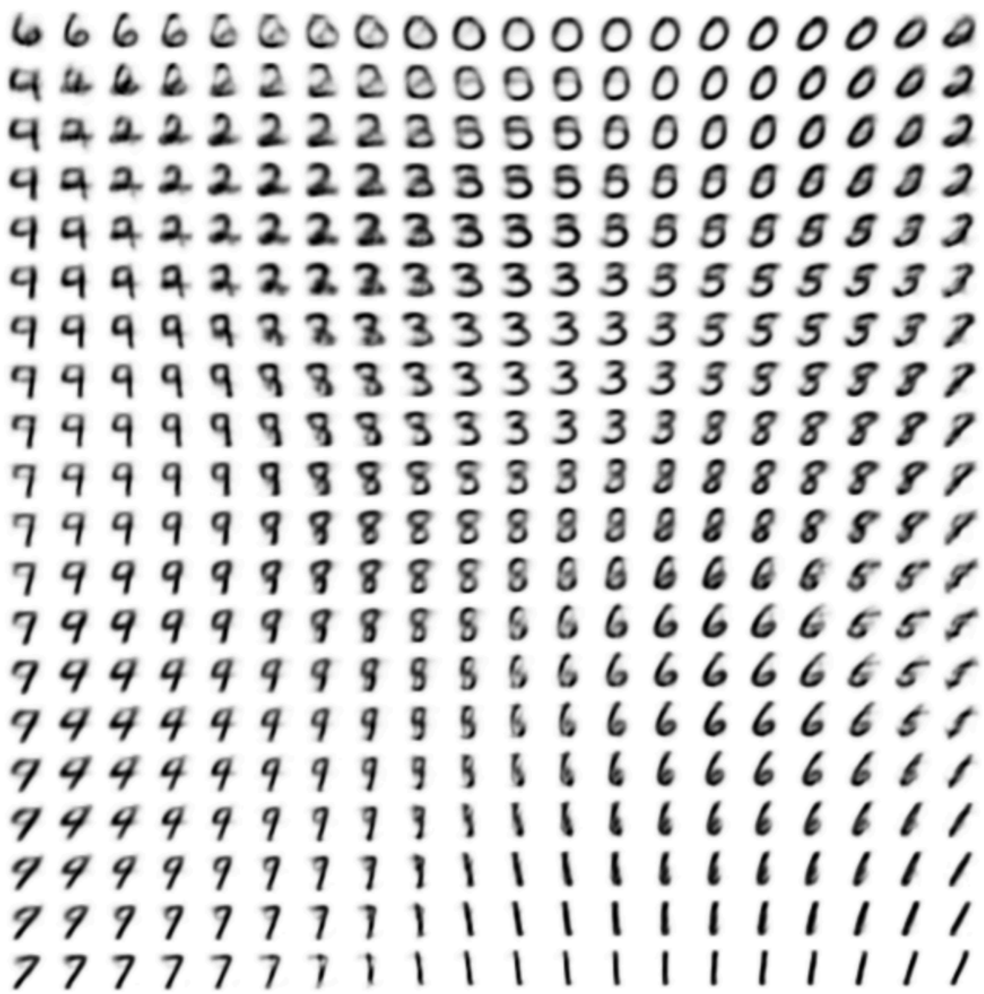
Raw data that lives in some arbitrary (high-dimensional) space



Representation space with
smooth and linear structure

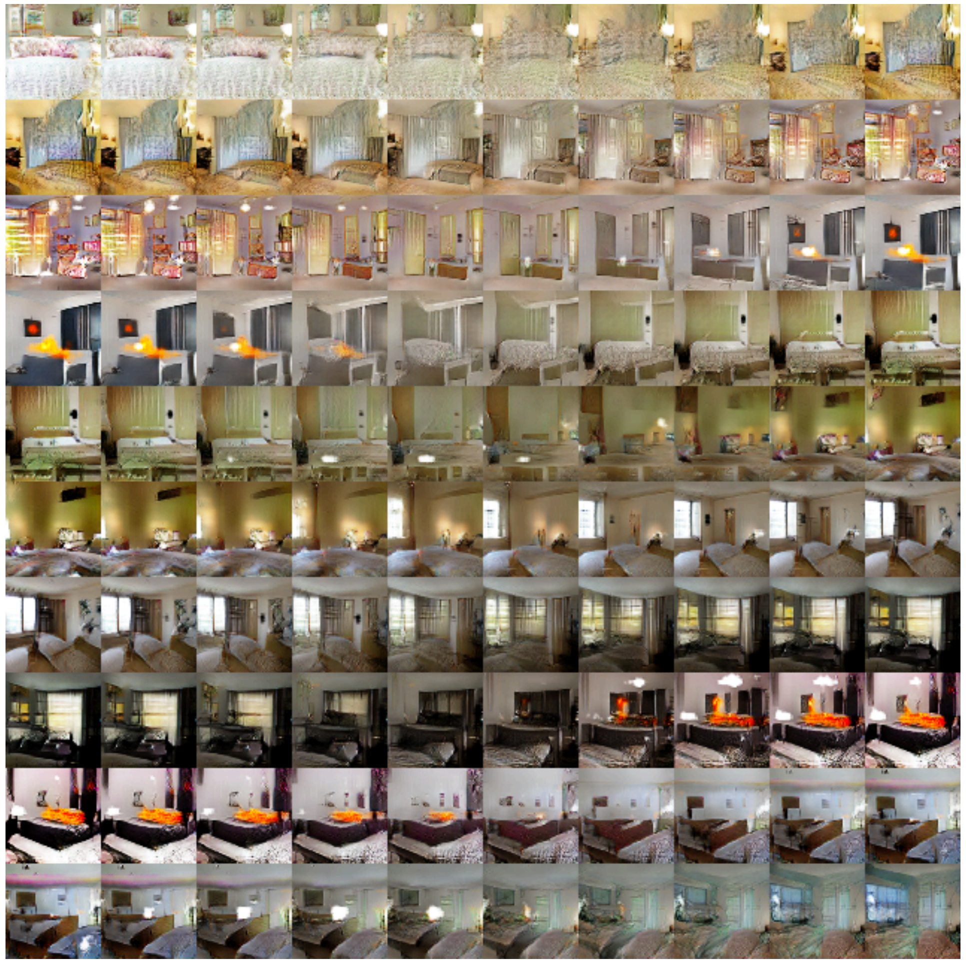
Representation: smoothness

Digits (MNIST)



Embedding learned by
variational autoencoder (VAE)

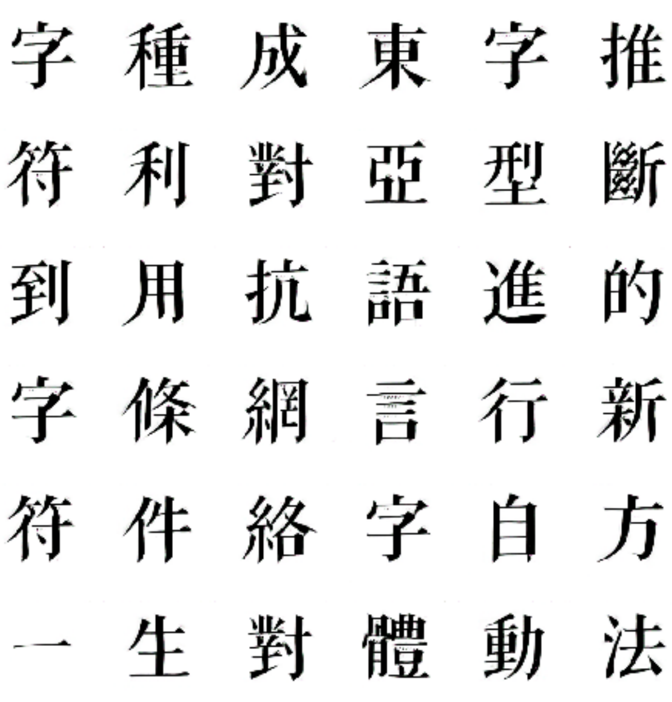
Bedroom (LSUN)



Embedding learned by
generative adversarial networks (GAN)

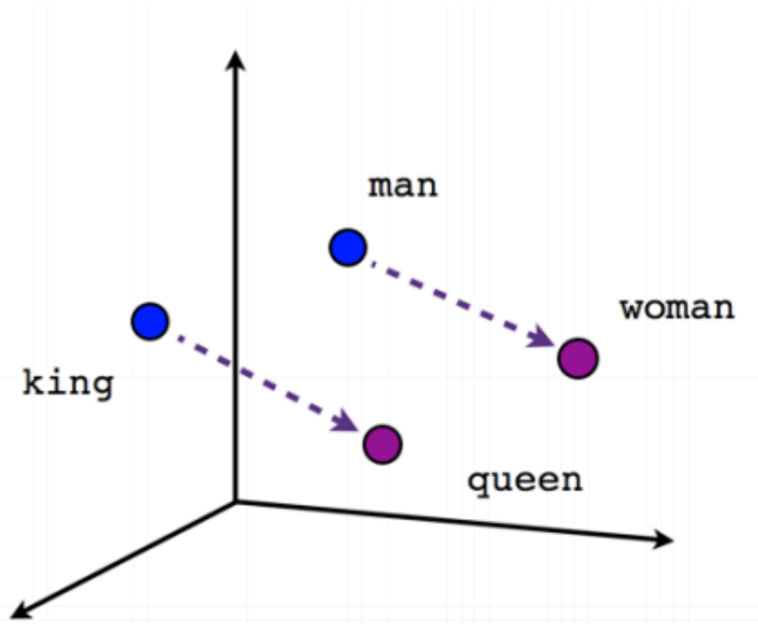
Representation: smoothness

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

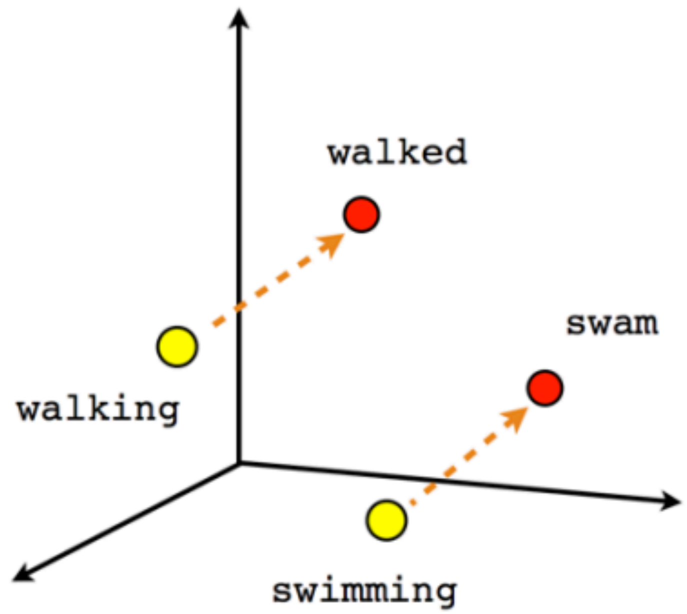


GAN github.com/kaonashi-tyc/zi2zi

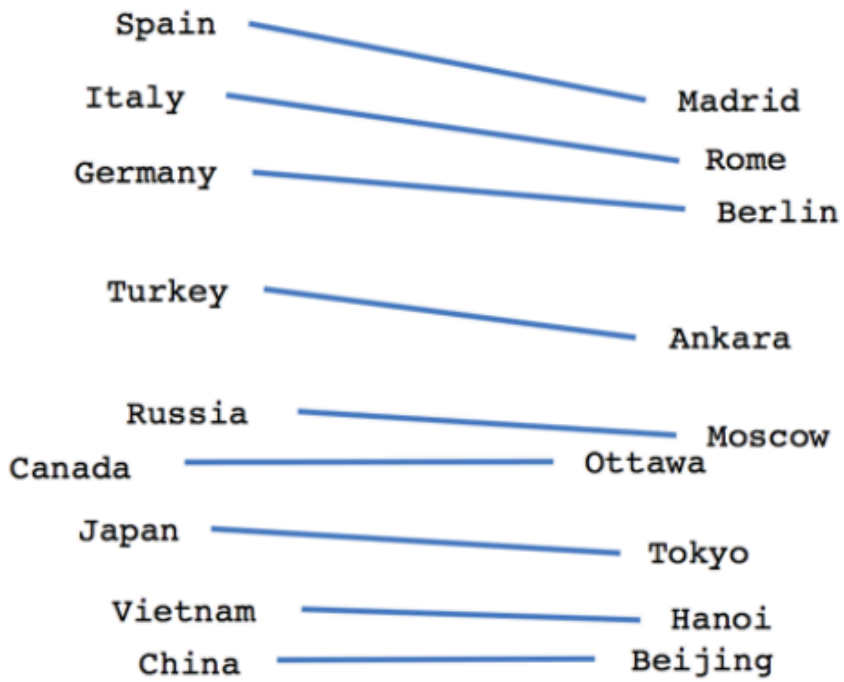
Representation: linearity



Male-Female



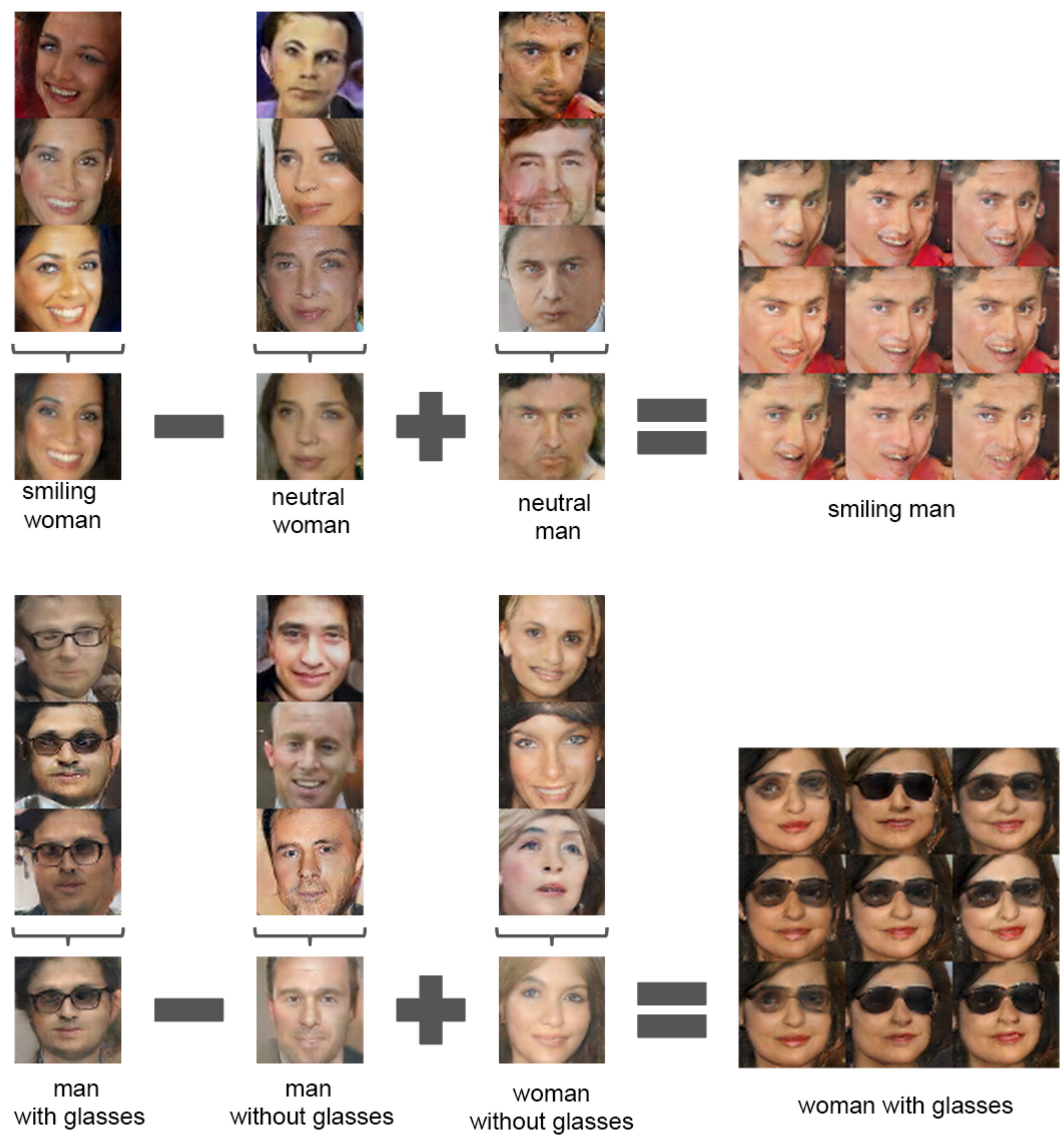
Verb tense



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

Reference: https://blog.openai.com/unsupervised-sentiment-neuron/

Part 2. Emerging deep learning architectures

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

Explaining a Joke

Input: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!
Model Output: TPUs are a type of computer chip that Google uses for deep learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

Logical Inference

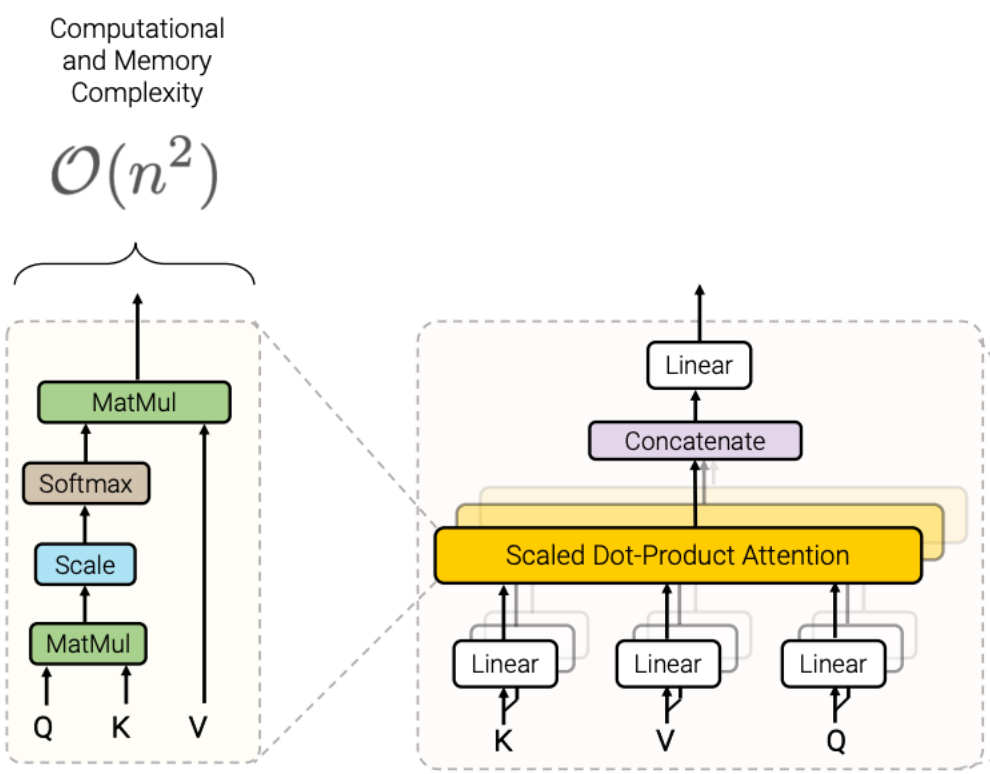
Input: Shelley is from Virginia, but is visiting that city with that famous market where they throw the fish. Going home next Tuesday!
Question: Is it likely that Shelley will be near the Pacific Ocean this weekend?
Model Output: The city with the famous market where they throw the fish is Seattle, Washington. Seattle is on the Pacific Ocean. Shelley is visiting Seattle, so she will be near the Pacific Ocean this weekend. The answer is "yes", it is likely that Shelley will be near the Pacific Ocean this weekend.

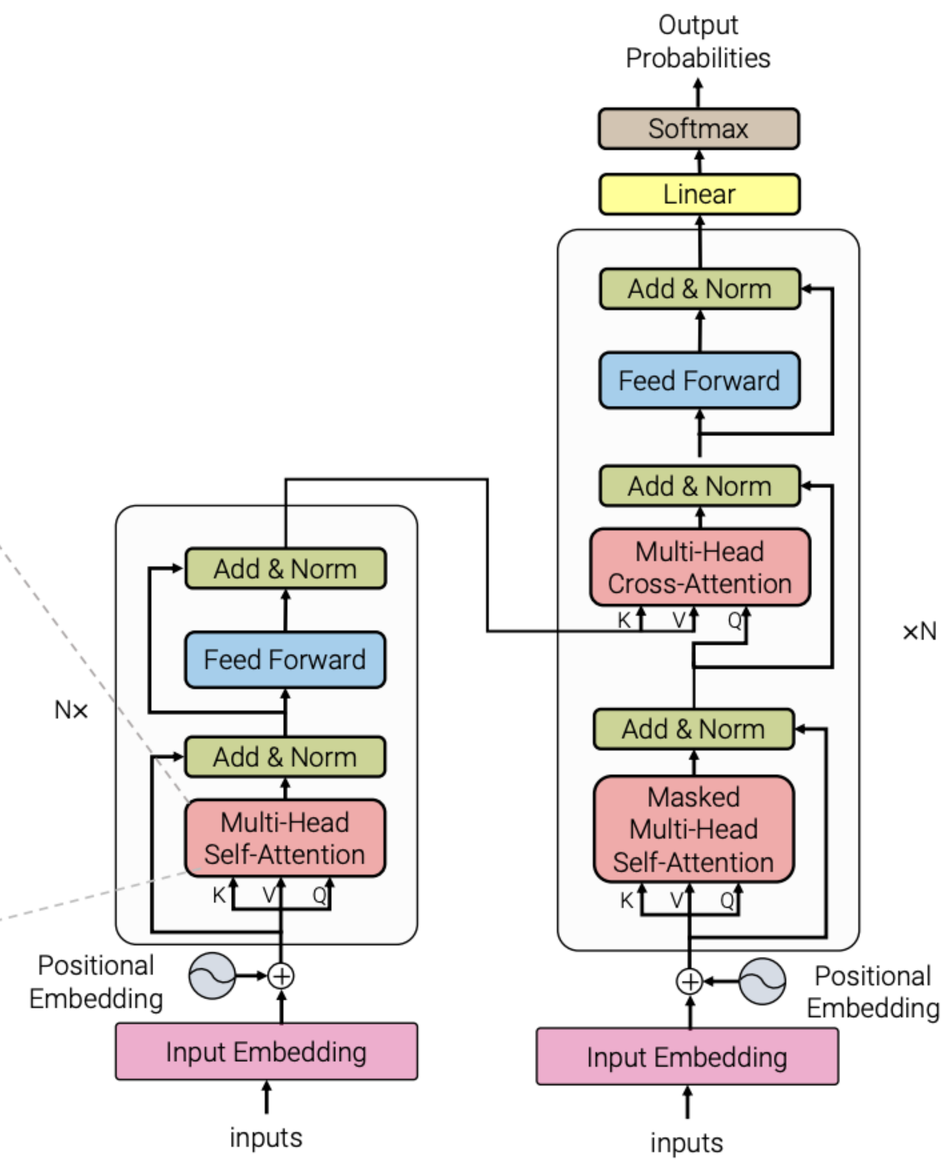
PaLM: Scaling Language Modeling with Pathways

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)

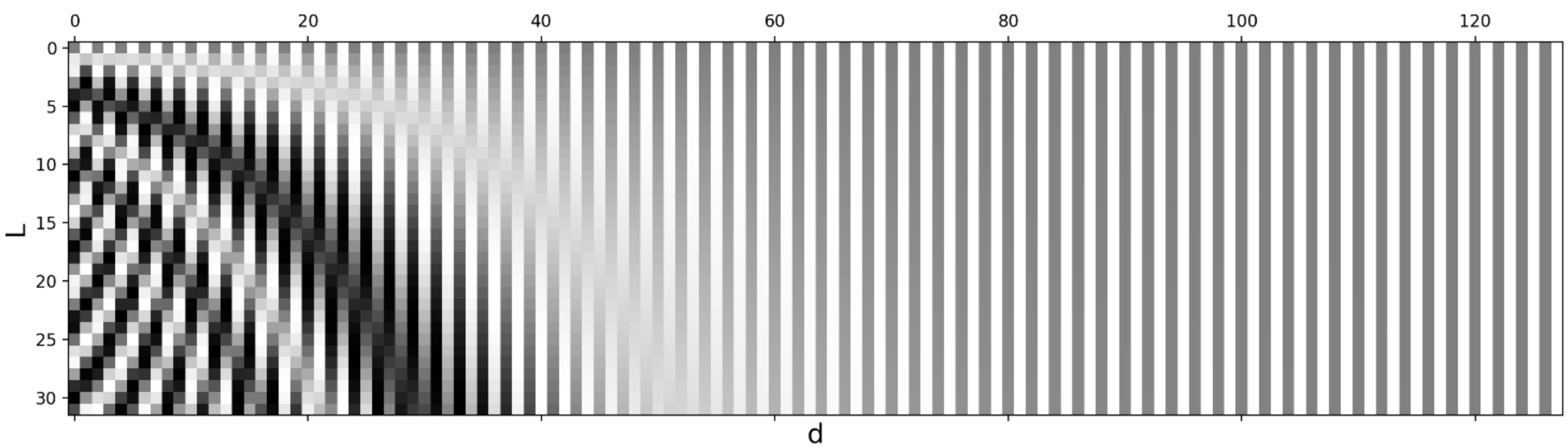
Ashish Vaswani, et al. "Attention is all you need." NIPS 2017.

At least you also need positional encoding!

Pre-specified positional encoding / embedding: the original transformer

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

Important: this assumes input length << 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)

https://lilianweng.github.io/lil-log/2020/04/07/the-transformer-family.html#longer-attention-span-transformer-xl

Ashish Vaswani, et al. "Attention is all you need." NIPS 2017.

What does learned positional embedding learn?

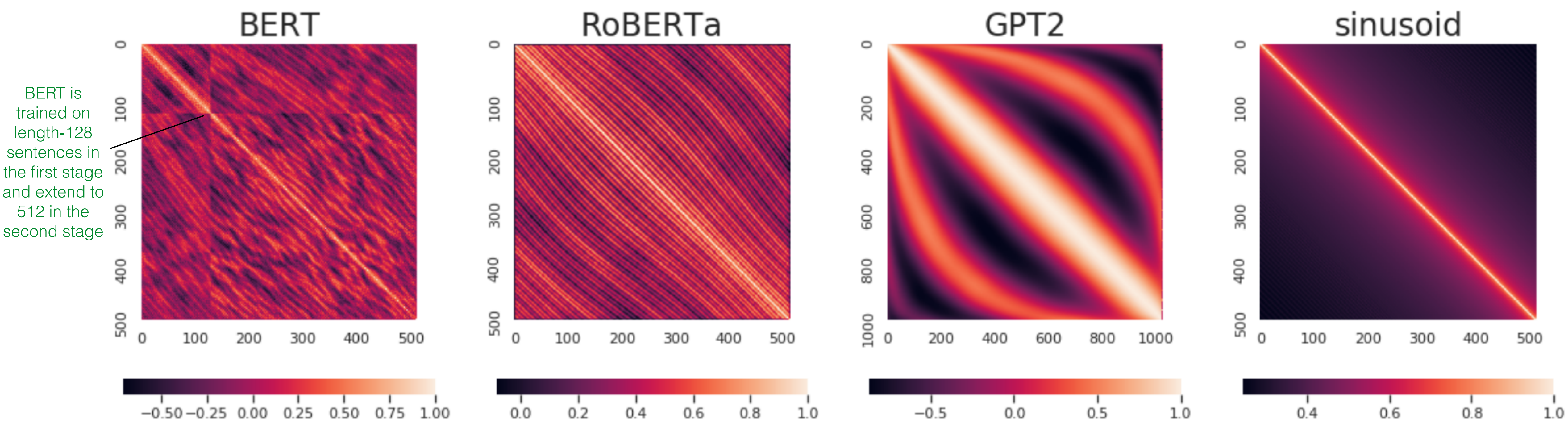


Figure 1: Visualization of position-wise cosine similarity of different position embeddings. Lighter in the figures denotes the higher similarity.

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

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

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

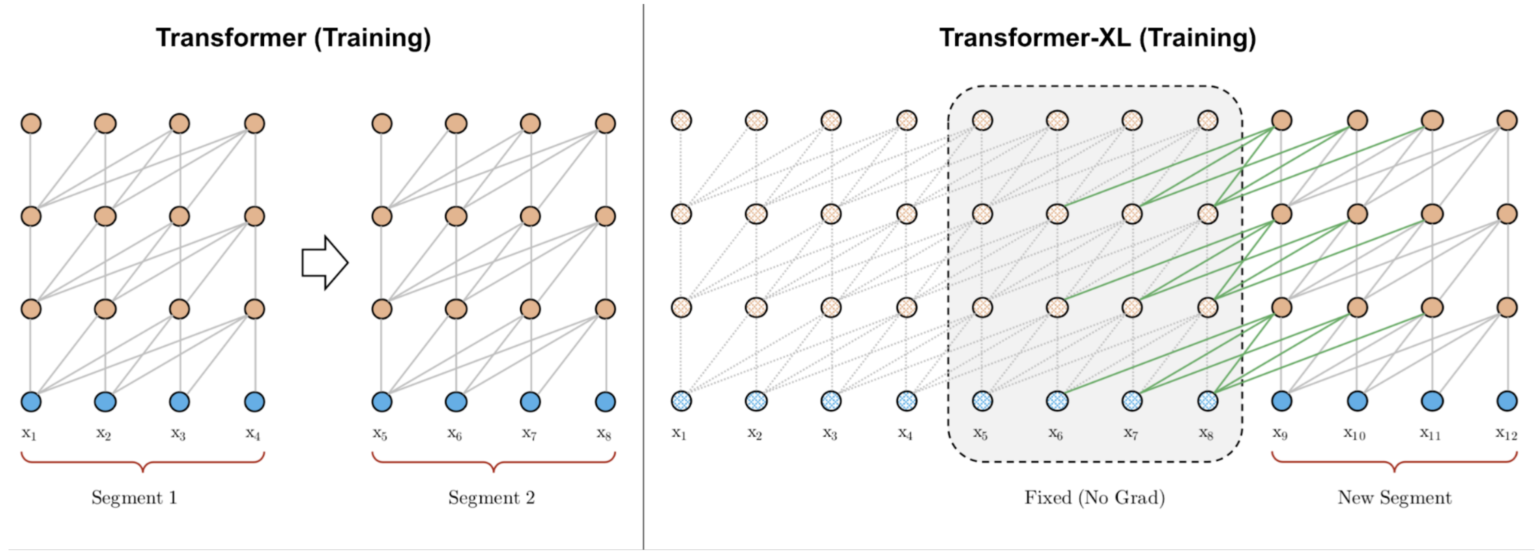
Predict the order of two positions with Logistic regression

Table 2: Error rate of the relative position regression.

What Do Position Embeddings Learn? An Empirical Study of Pre-Trained Language Model Positional Encoding

Relative positional embedding - better ways to encode position?

- Transformer XL



Motivation: Mimicking absolution positional embedding without absolution positional embedding

$$a_{ij} = \mathbf{q}_i \mathbf{k}_j^\top = (\mathbf{x}_i + \mathbf{p}_i)\mathbf{W}^q((\mathbf{x}_j + \mathbf{p}_j)\mathbf{W}^k)^\top$$
$$= \mathbf{x}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{x}_j^\top + \mathbf{x}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{p}_j^\top + \mathbf{p}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{x}_j^\top + \mathbf{p}_i \mathbf{W}^q {\mathbf{W}^k}^\top \mathbf{p}_j^\top$$

Transformer-XL reparameterizes the above four terms as follows:

Q' K' | Q' R' | u K' | v R'

$$a_{ij}^{\text{rel}} = \underbrace{\mathbf{x}_i \mathbf{W}^q {\mathbf{W}_E^k}^\top \mathbf{x}_j^\top}_{\text{content-based addressing}} + \underbrace{\mathbf{x}_i \mathbf{W}^q {\mathbf{W}_R^k}^\top \mathbf{r}_{i-j}^\top}_{\text{content-dependent positional bias}} + \underbrace{\mathbf{u} {\mathbf{W}_E^k}^\top \mathbf{x}_j^\top}_{\text{global content bias}} + \underbrace{\mathbf{v} {\mathbf{W}_R^k}^\top \mathbf{r}_{i-j}^\top}_{\text{global positional bias}}$$

- Replace $\mathbf{p}_j$ with relative positional encoding $\mathbf{r}_{i-j} \in \mathbf{R}^d$;
- Replace $\mathbf{p}_i \mathbf{W}^q$ with two trainable parameters $\mathbf{u}$ (for content) and $\mathbf{v}$ (for location) in two different terms;
- Split $\mathbf{W}^k$ into two matrices, $\mathbf{W}_E^k$ for content information and $\mathbf{W}_R^k$ for location information.

Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context

Q' K' Q' R'

$$e_{ij} = \frac{x_i W^Q (x_j W^K)^T + x_i W^Q (a_{ij}^K)^T}{\sqrt{d_z}}$$

Self-Attention with Relative Position Representations

Q' K' R'

$$\alpha_{ij}^{T5} = \frac{1}{\sqrt{d}} (x_i^l W^{Q,l})(x_j^l W^{K,l})^T + b_{j-i}.$$

Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

Rotary Positional Embedding (RoPE)

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

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

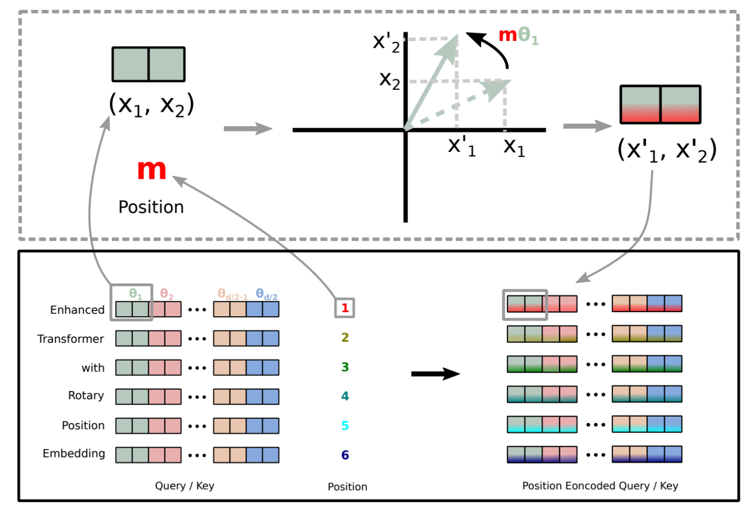


Figure 1: Implementation of Rotary Position Embedding(RoPE).

Rotation matrix

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

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

Requires even number of dimensions (can be interpreted as real and imaginary parts of a complex number that is rotated)

rotary embeddings must be applied at every layer (every Q and K), but computational cost is negligible compared to transformer

Can rotary positional embedding be combined with complex-valued neural networks (complex transformer?)

RoFormer: Enhanced Transformer with Rotary Position Embedding

Rotary Positional Embedding (RoPE)

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

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

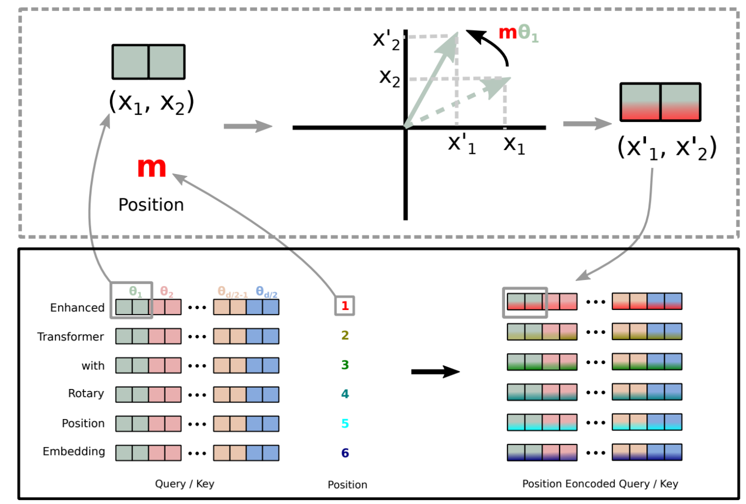


Figure 1: Implementation of Rotary Position Embedding(RoPE).

$$
\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d-2} \\ q_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_0 \\ \cos m\theta_0 \\ \cos m\theta_1 \\ \cos m\theta_1 \\ \vdots \\ \cos m\theta_{d/2-1} \\ \cos m\theta_{d/2-1} \end{pmatrix} + \begin{pmatrix} -q_1 \\ q_0 \\ -q_3 \\ q_2 \\ \vdots \\ -q_{d-1} \\ q_{d-2} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_0 \\ \sin m\theta_0 \\ \sin m\theta_1 \\ \sin m\theta_1 \\ \vdots \\ \sin m\theta_{d/2-1} \\ \sin m\theta_{d/2-1} \end{pmatrix}
$$

Rotation matrix

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

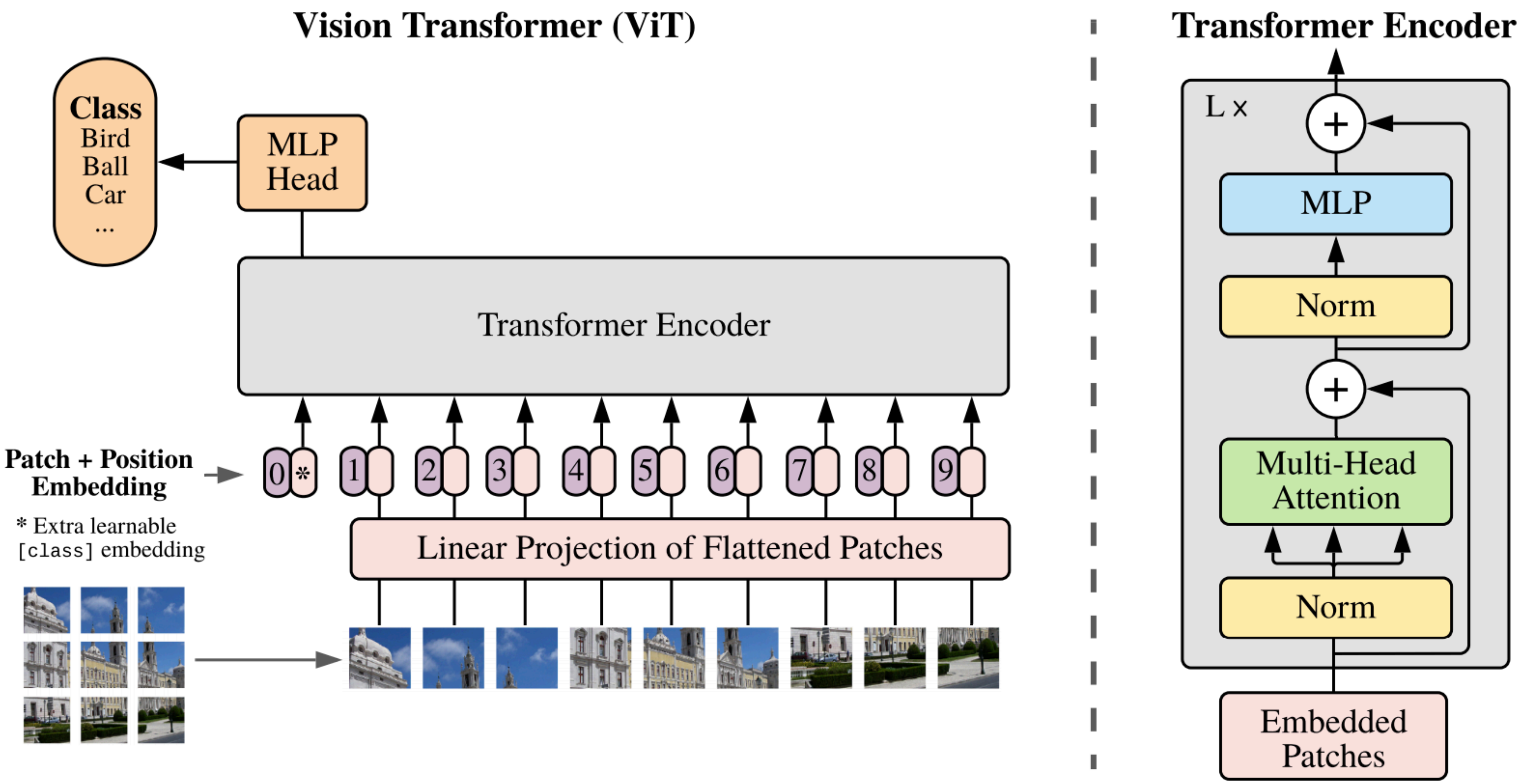
Requires even number of dimensions (can be interpreted as real and imaginary parts of a complex number that is rotated)

rotary embeddings must be applied at every layer (every Q and K), but computational cost is negligible compared to transformer

Can rotary positional embedding be combined with complex-valued neural networks (complex transformer?)

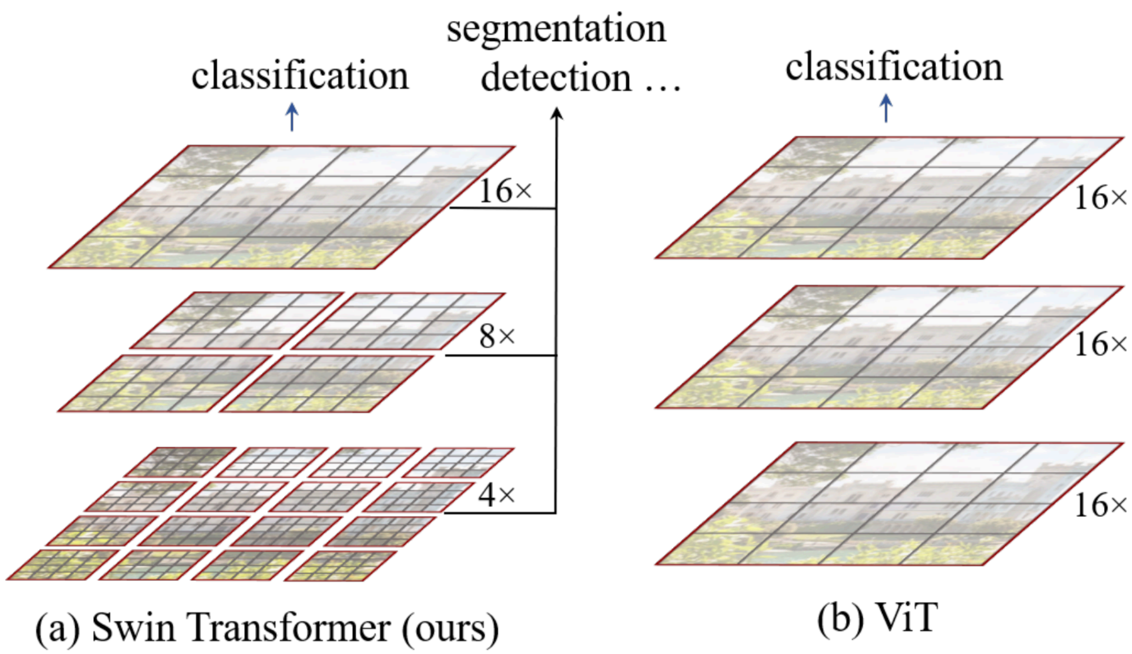
RoFormer: Enhanced Transformer with Rotary Position Embedding

Vision transformer for image recognition

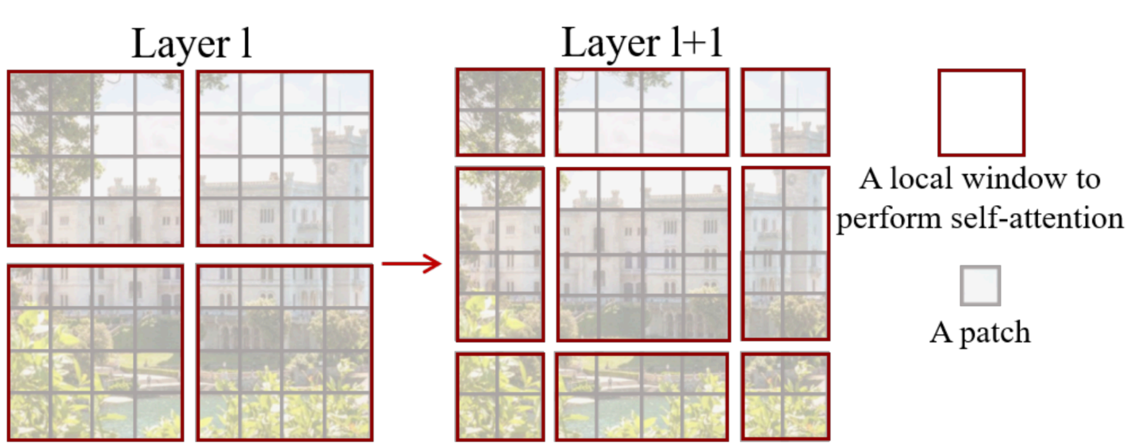


Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Swin transformer: improving ViT



Hierarchical structure

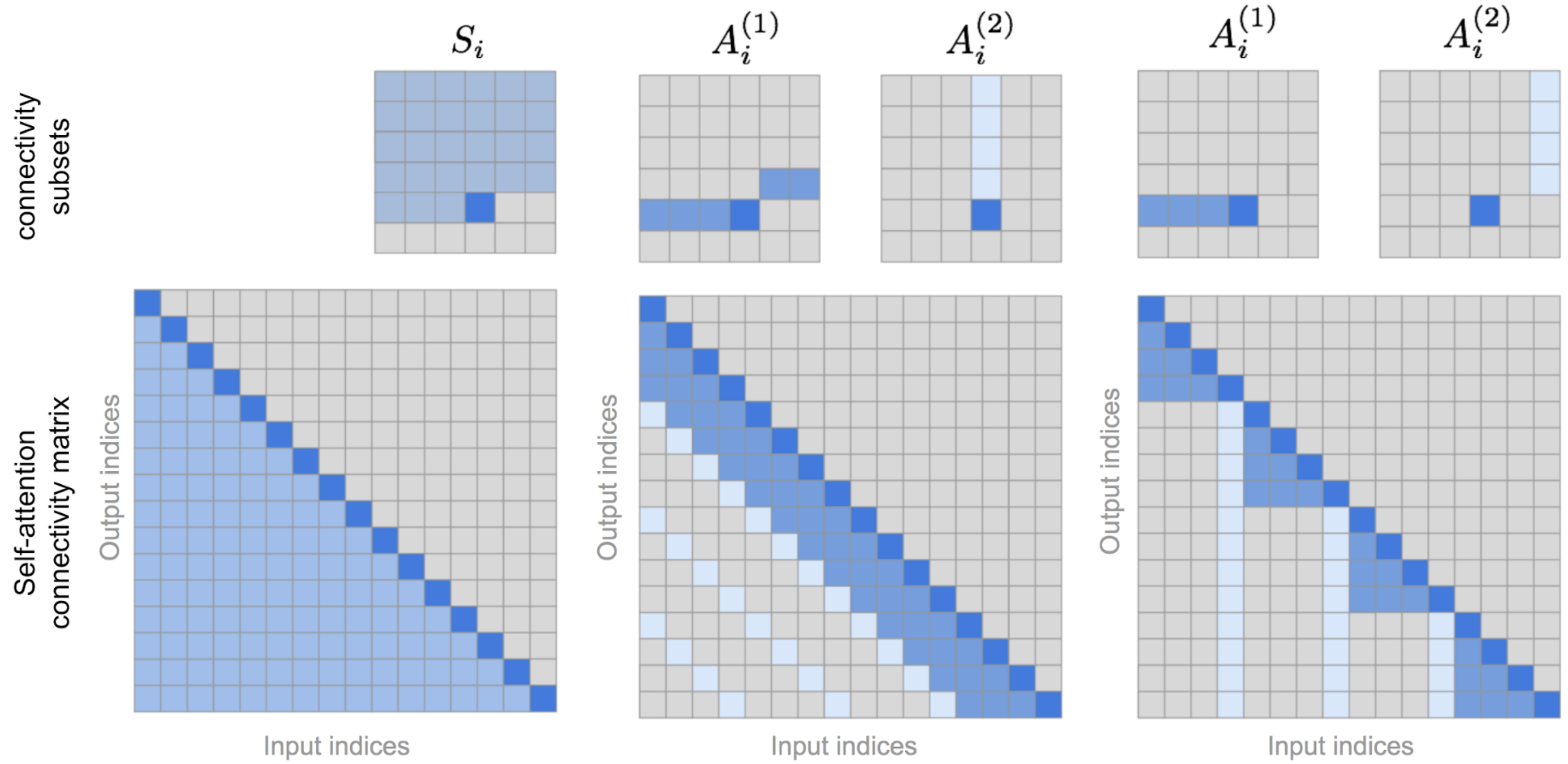


Shifted non-overlapping windows
(Swin means shifted windows)

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Scalable transformer for long sequences

Sparse factorized attention

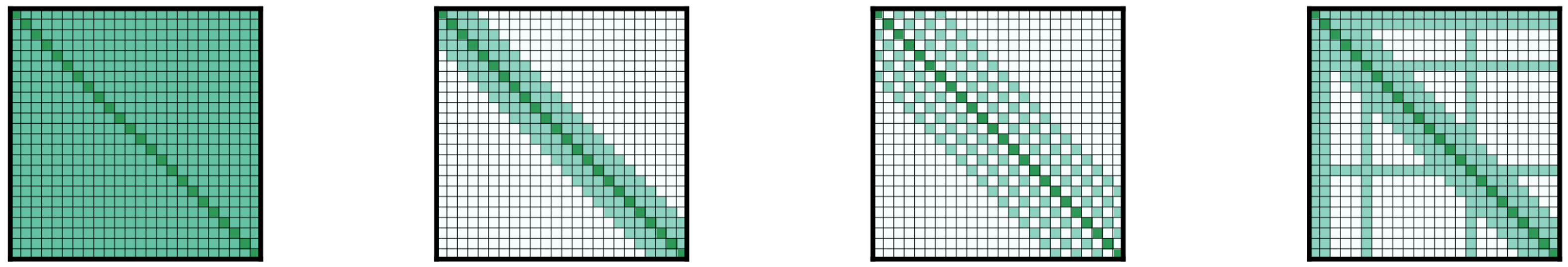


(a) Transformer

(b) Sparse Transformer with strided attention.

(c) Sparse Transformer with fixed attention.

Generating Long Sequences with Sparse Transformers

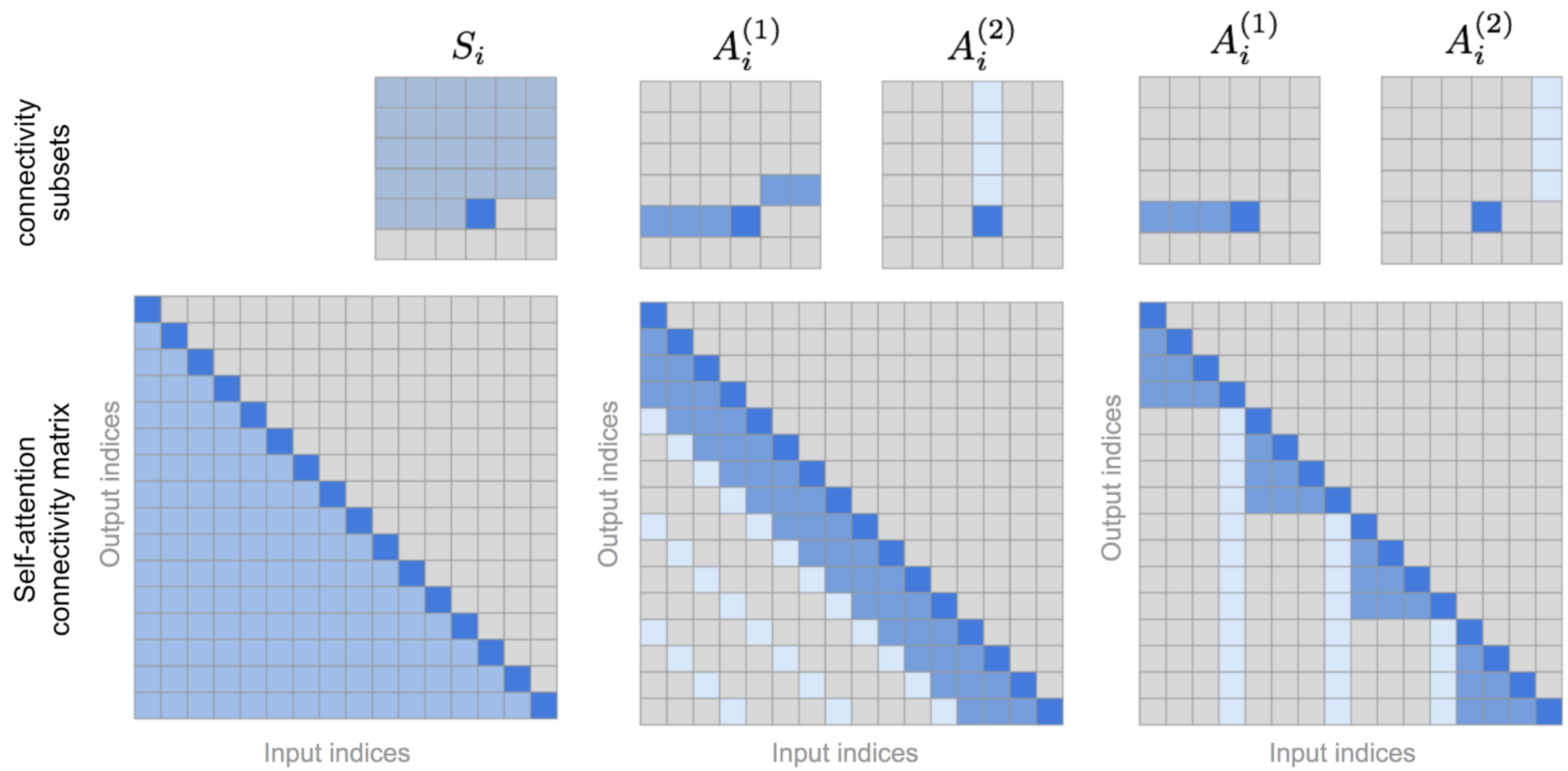


(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Longformer: The Long-Document Transformer

Scalable transformer for long sequences

Sparse factorized attention

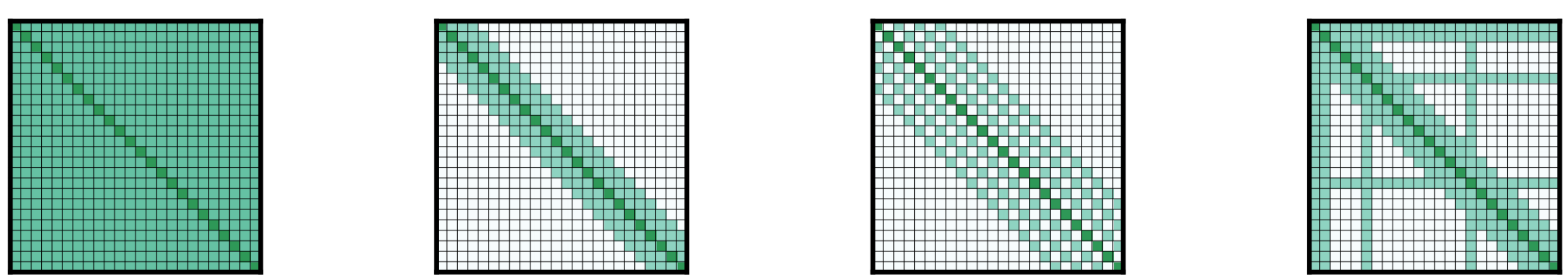


(a) Transformer

(b) Sparse Transformer with strided attention.

(c) Sparse Transformer with fixed attention.

Generating Long Sequences with Sparse Transformers



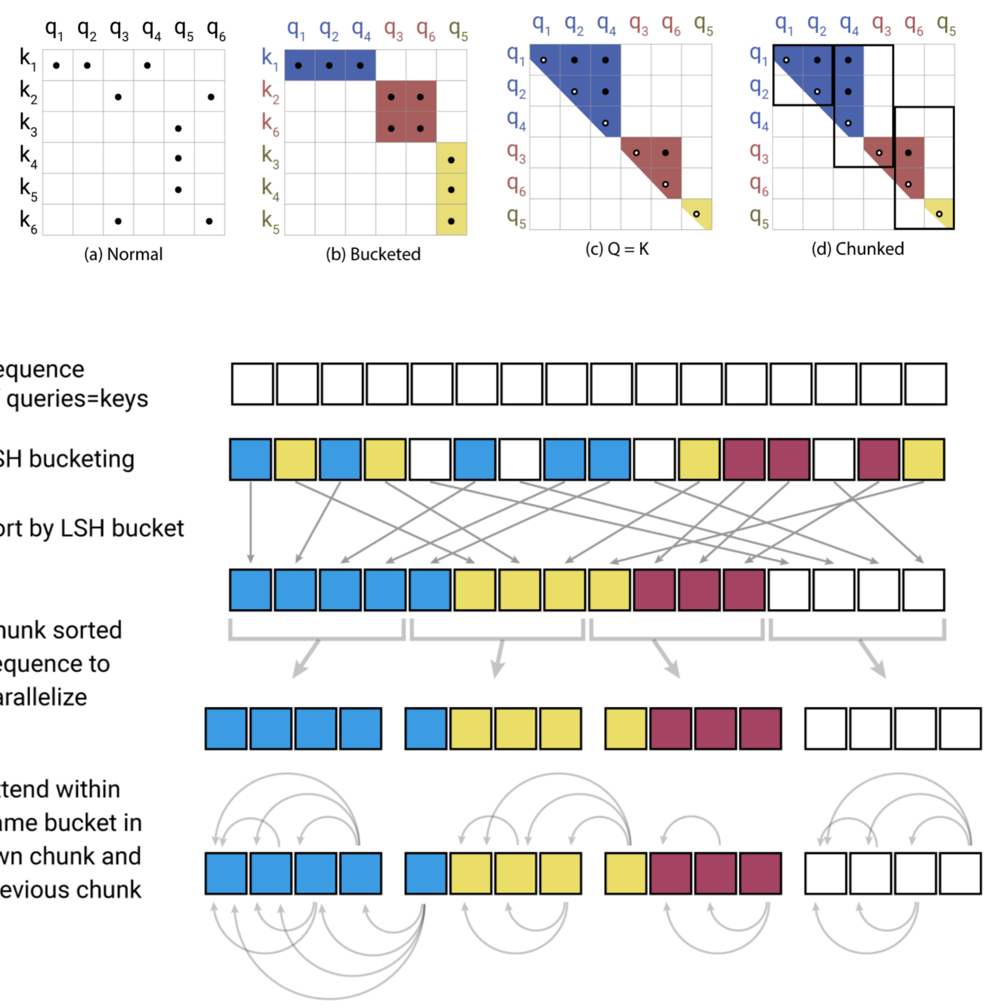
(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Longformer: The Long-Document Transformer

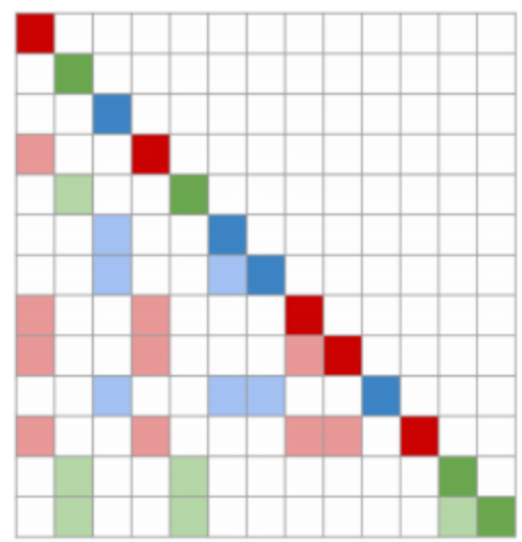
Scalable transformer for long sequences

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

Reformer (LSH)



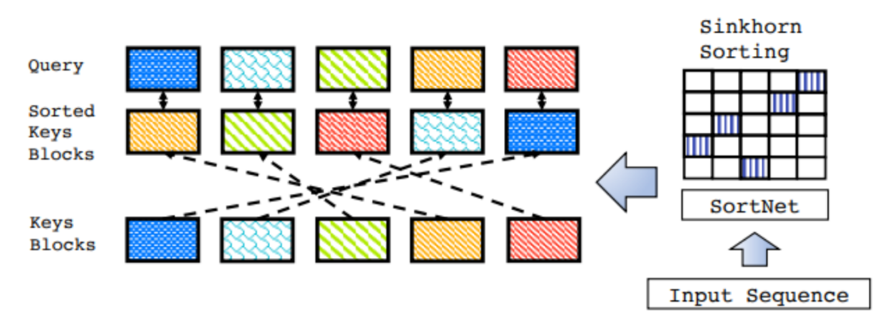
Routing transformer (k-means)



(c) Routing attention

Sinkhorn transformer (Sinkhorn Sorting)

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

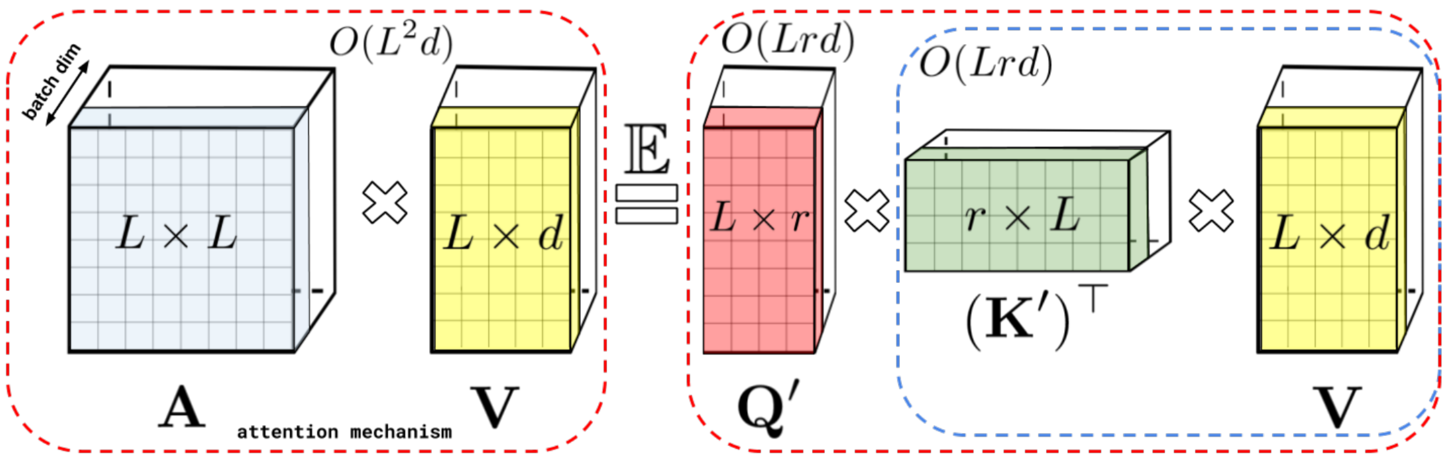
Nikita Kitaev, et al. "Reformer: The Efficient Transformer" ICLR 2020.

Efficient Content-Based Sparse Attention with Routing Transformers

Sparse Sinkhorn Attention

Scalable transformer for long sequences

Low-rank approximation of attention (FAVOR+)



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

Feature map decomposition (can need infinite-dimensions though)

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

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

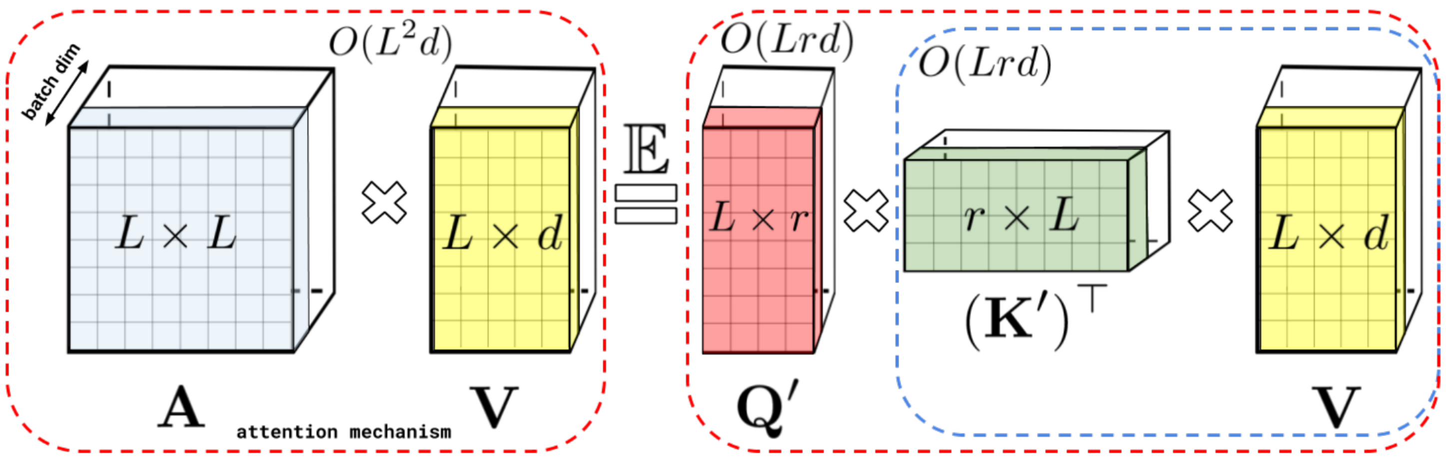
f w

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

Rethinking Attention with Performers

Scalable transformer for long sequences

Low-rank approximation of attention (FAVOR+)



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

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

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

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

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

Proof:

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

$$\begin{aligned}
\exp(\|\boldsymbol{x} + \boldsymbol{y}\|^2/2) &= (2\pi)^{-d/2} \exp(\|\boldsymbol{x} + \boldsymbol{y}\|^2/2) \int \exp(-\|\boldsymbol{w} - (\boldsymbol{x} + \boldsymbol{y})\|^2/2) d\boldsymbol{w} \\
&= (2\pi)^{-d/2} \int \exp(-\|\boldsymbol{w}\|^2/2 + \boldsymbol{w}^\top(\boldsymbol{x} + \boldsymbol{y}) - \|\boldsymbol{x} + \boldsymbol{y}\|^2/2 + \|\boldsymbol{x} + \boldsymbol{y}\|^2/2) d\boldsymbol{w} \\
&= (2\pi)^{-d/2} \int \exp(-\|\boldsymbol{w}\|^2/2 + \boldsymbol{w}^\top(\boldsymbol{x} + \boldsymbol{y})) d\boldsymbol{w} \\
&= (2\pi)^{-d/2} \int \exp(-\|\boldsymbol{w}\|^2/2) \cdot \exp(\boldsymbol{w}^\top \boldsymbol{x}) \cdot \exp(\boldsymbol{w}^\top \boldsymbol{y}) d\boldsymbol{w} \\
&= \mathbb{E}_{\omega \sim \mathcal{N}(\mathbf{0}_d, \mathbf{I}_d)}[\exp(\omega^\top \boldsymbol{x}) \cdot \exp(\omega^\top \boldsymbol{y})].
\end{aligned}$$

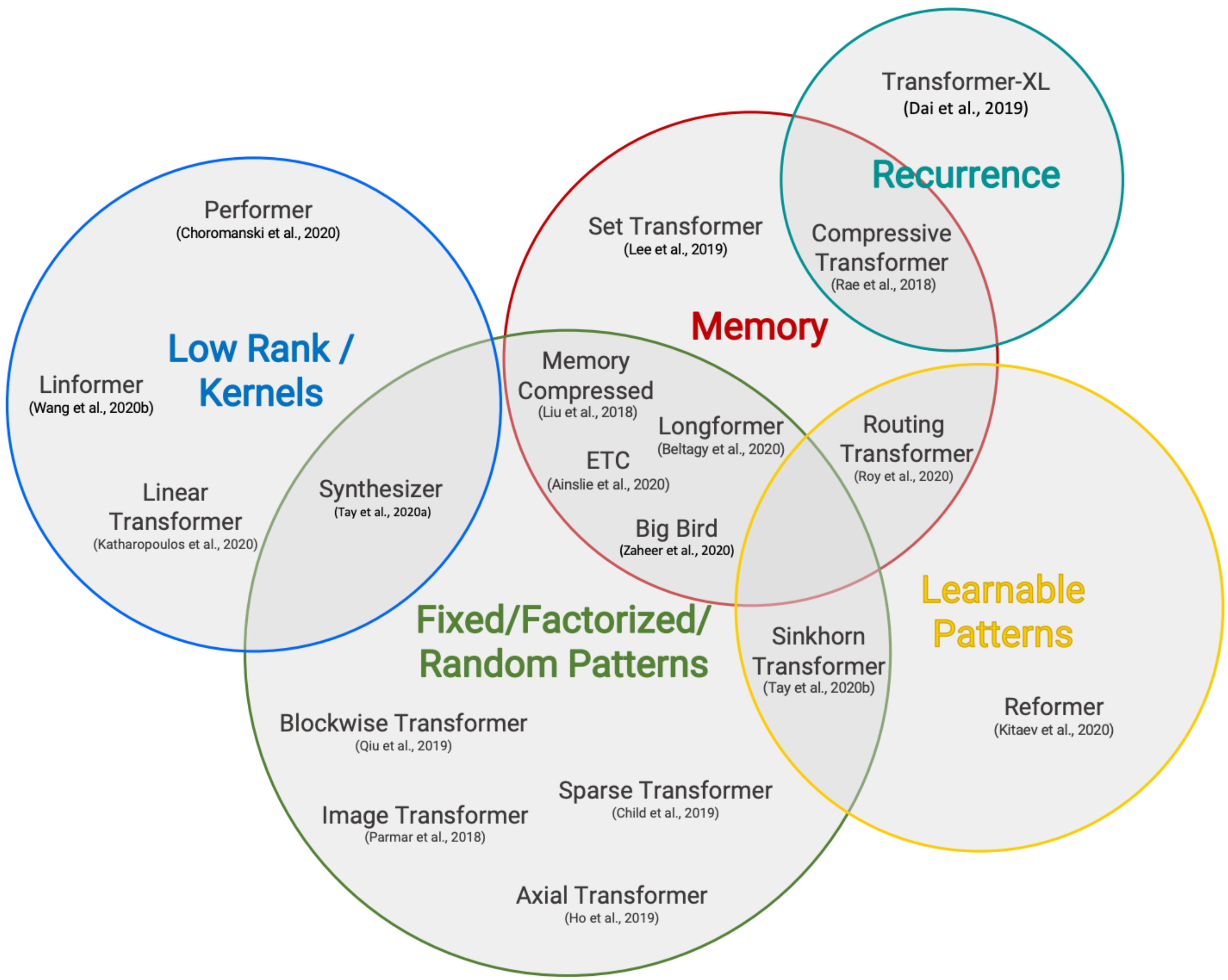
Exp can be replaced with ReLU for better performance in practice

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

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

Rethinking Attention with Performers

Summary of existing "efficient" transformers

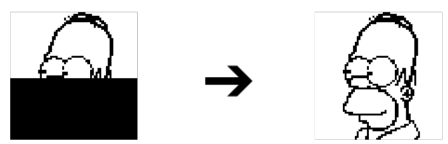


A Hopfield-network interpretation of transformer

Classical Hopfield network:
Store and retrieval of binary patterns

$$\boldsymbol{W} = \sum_i^N \boldsymbol{x}_i \boldsymbol{x}_i^T$$

Fixed-point update $\boldsymbol{\xi}^{t+1} = \operatorname{sgn}(\boldsymbol{W}\boldsymbol{\xi}^t - \boldsymbol{b})$



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

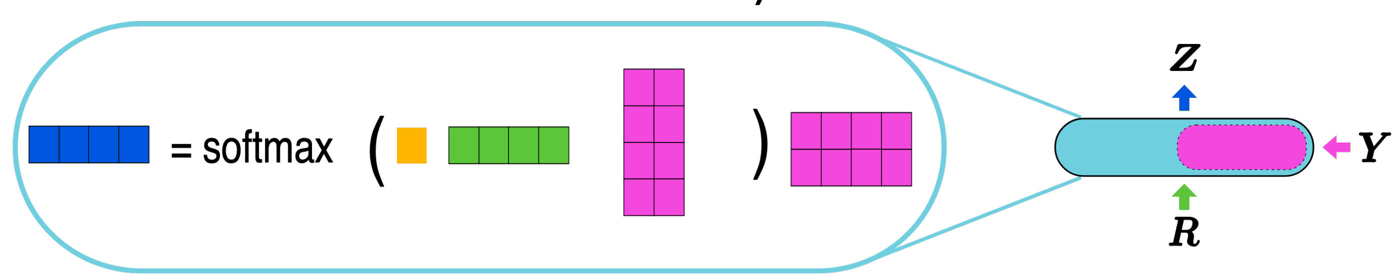
Continuous Hopfield network:

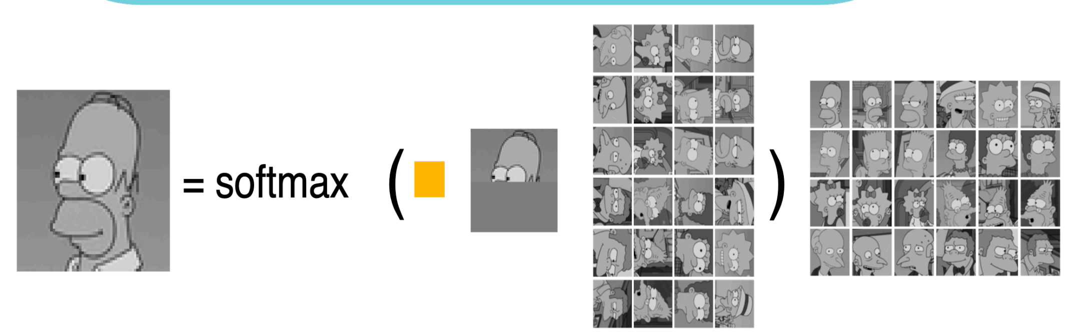
$$\mathrm{E} = -\mathrm{lse}\left(\beta, \boldsymbol{X}^T\boldsymbol{\xi}\right) + \frac{1}{2}\boldsymbol{\xi}^T\boldsymbol{\xi} \cdot$$

update $\boldsymbol{\xi}^{t+1} = \boldsymbol{X}\operatorname{softmax}\left(\beta \boldsymbol{X}^T \boldsymbol{\xi}^t\right)$

Query Key Value

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





Discrete modern Hopfield network:

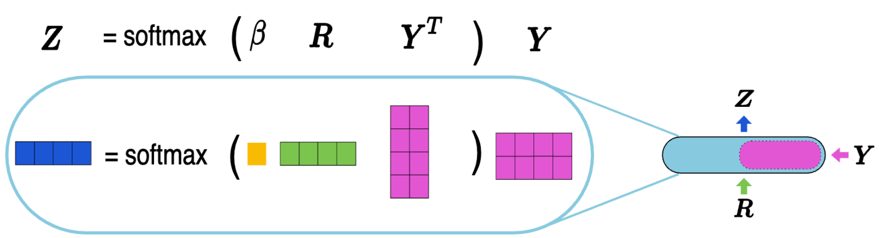
$$\mathrm{E} = -\sum_{i=1}^N \exp(\boldsymbol{x}_i^T \boldsymbol{\xi})$$

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

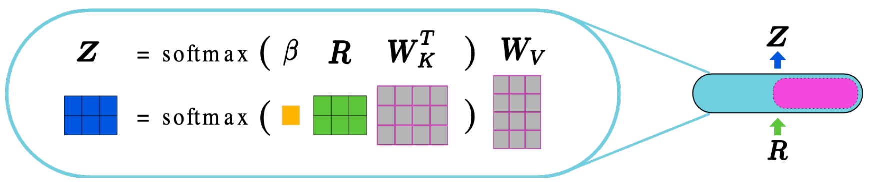
https://ml-jku.github.io/hopfield-layers/

A Hopfield-network interpretation of transformer

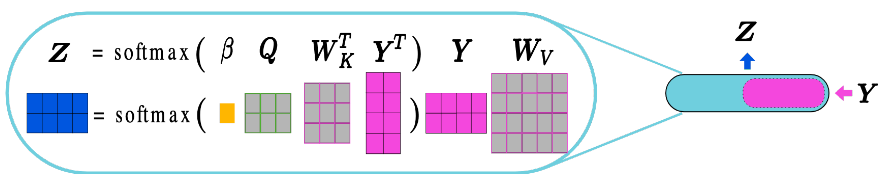
No internal parameters (similar pattern retrieval)



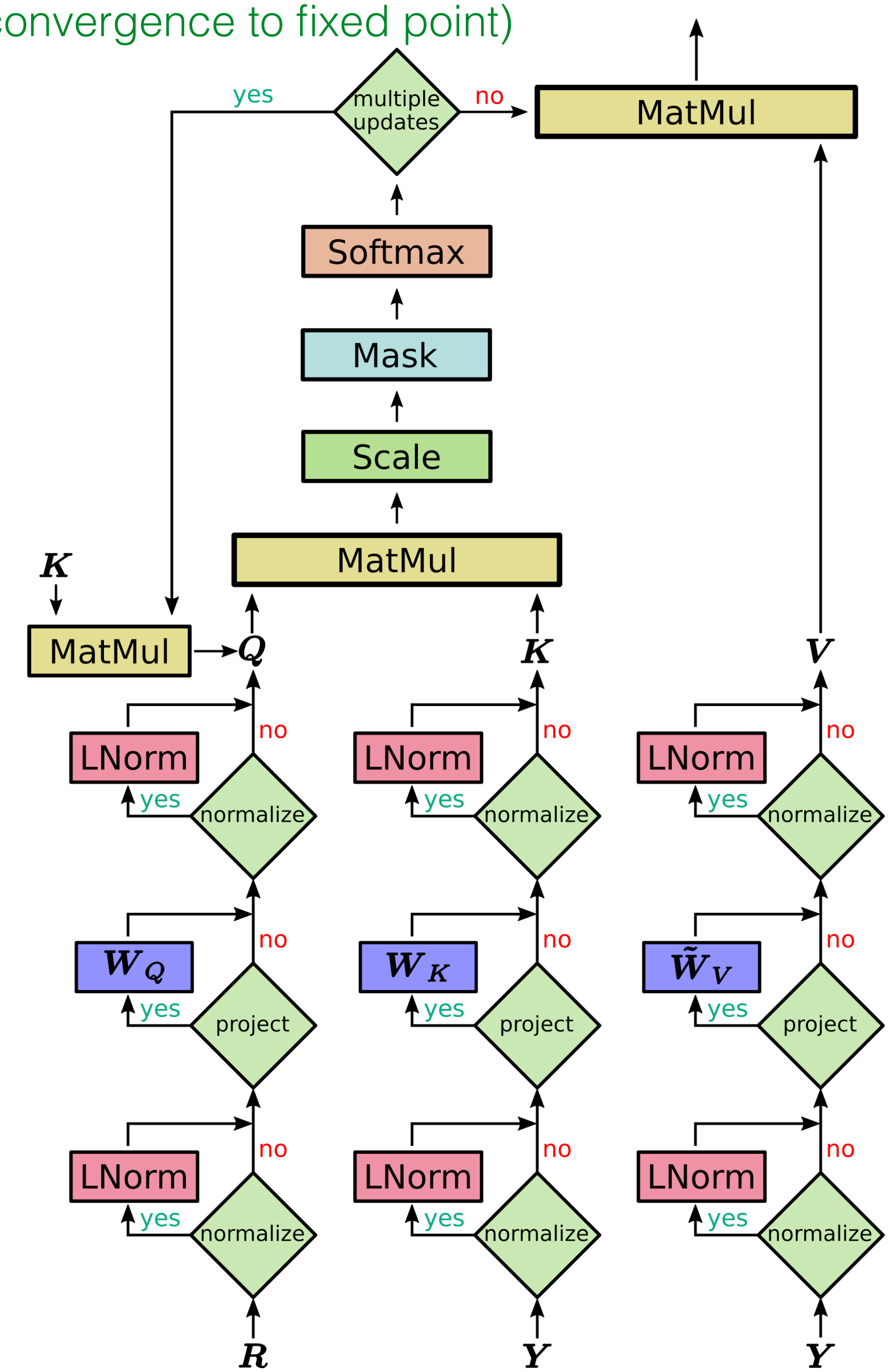
Stored patterns (key) and projection (value) are parameters



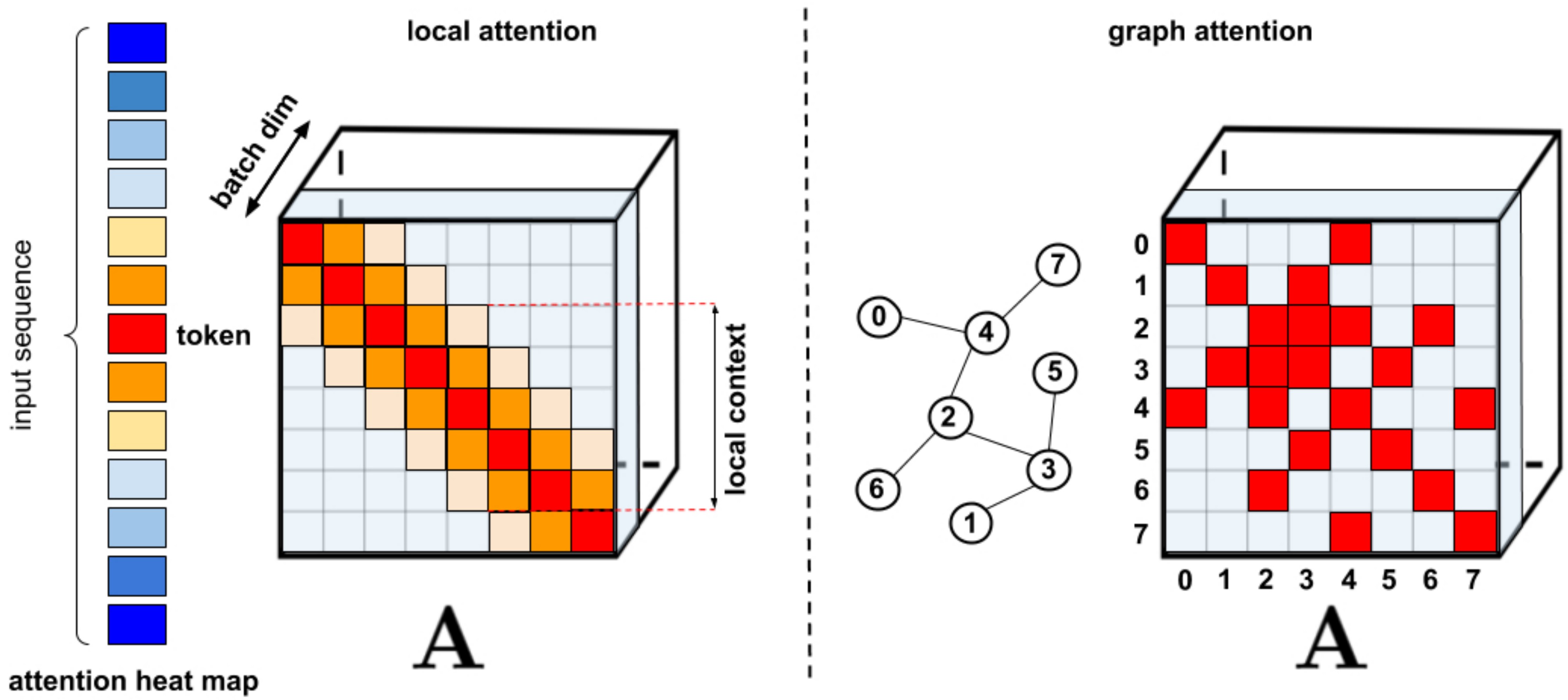
Query and projection are parameters



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



From transformer to graph network

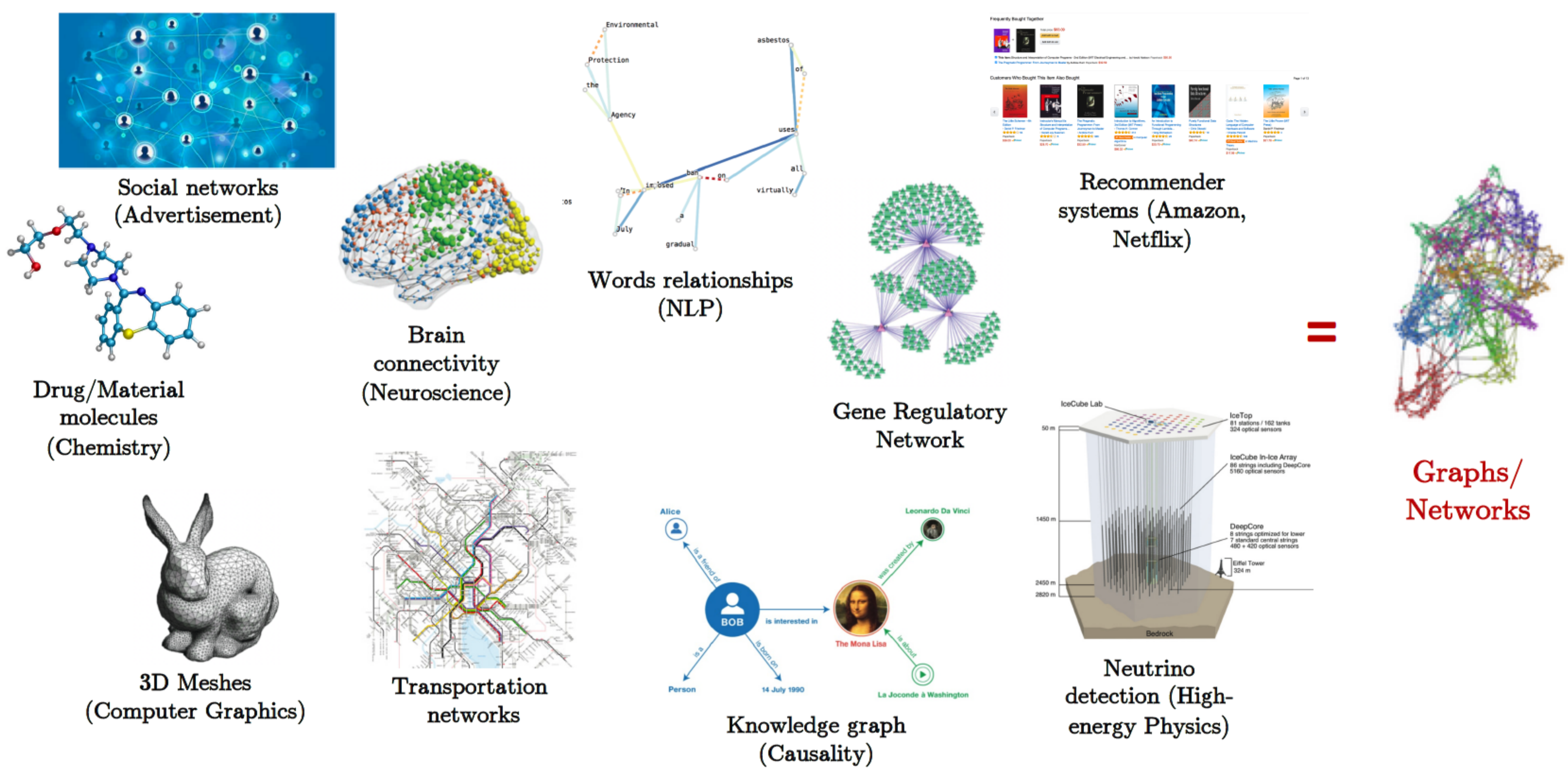


https://ai.googleblog.com/2020/10/rethinking-attention-with-performers.html

Graph Neural Network

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

Graph-structured data



https://graphdeeplearning.github.io/project/spatial-convnets/

A general form of Graph Network (node-centric)

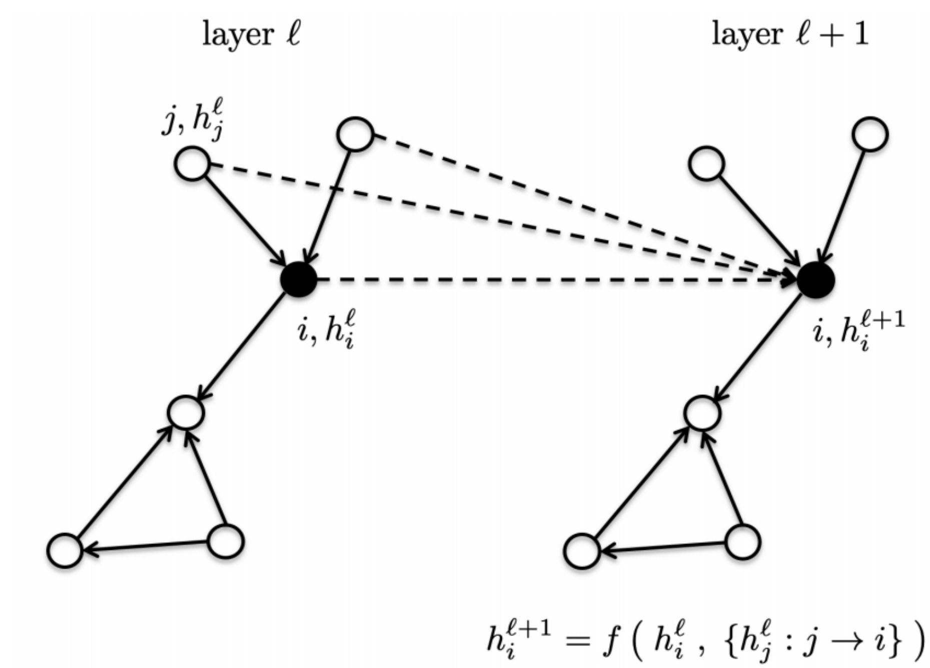


Figure 5: A generic graph neural network layer. Figure adapted from [11].

Benchmarking Graph Neural Networks
https://arxiv.org/pdf/2003.00982.pdf

A general form of Graph Network (node-centric)

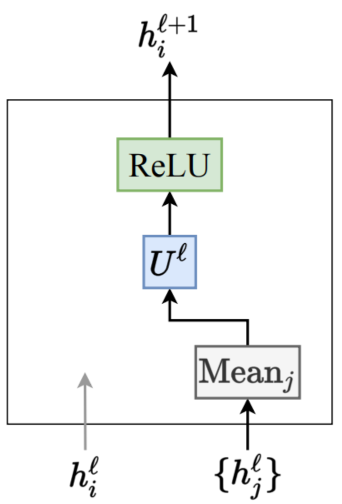


Figure 6: GCN Layer

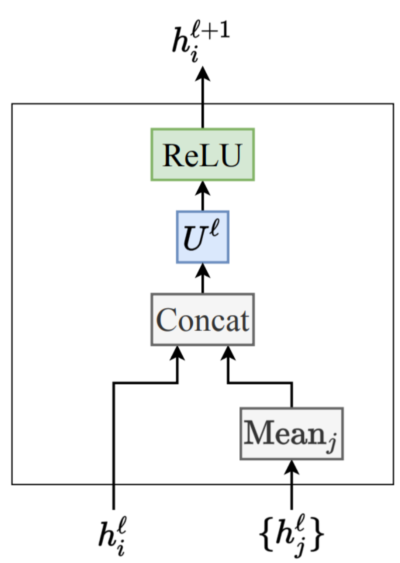


Figure 7: GraphSage Layer

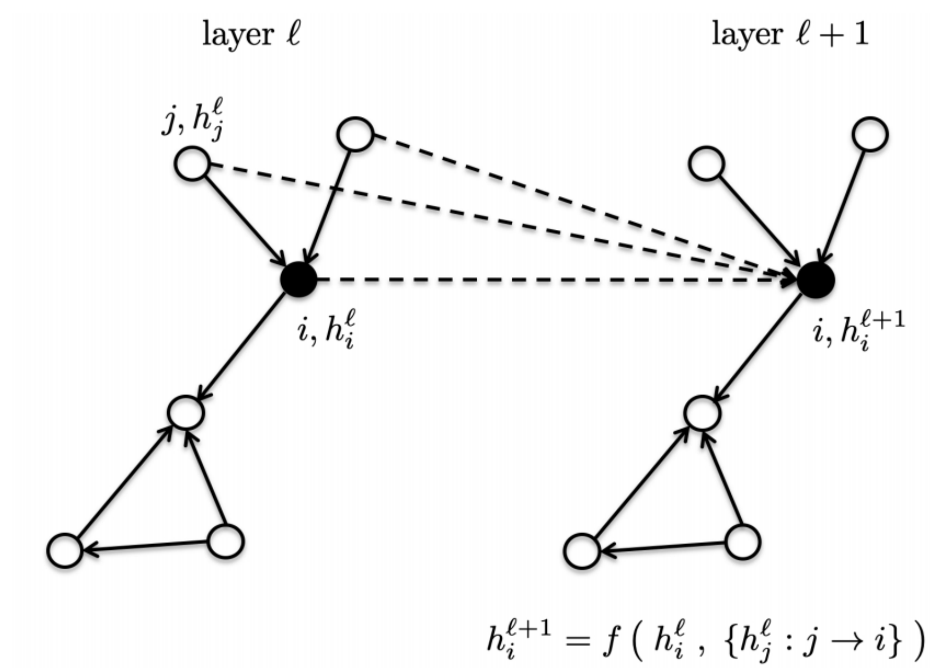


Figure 5: A generic graph neural network layer. Figure adapted from [11].

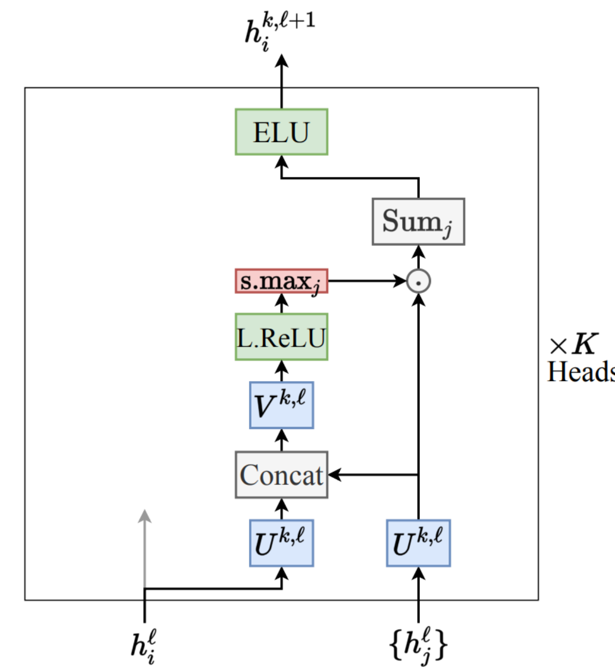


Figure 8: GAT Layer

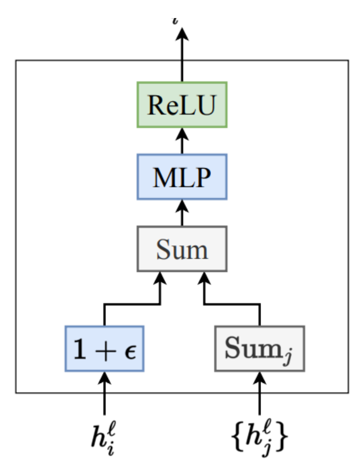


Figure 11: GIN Layer

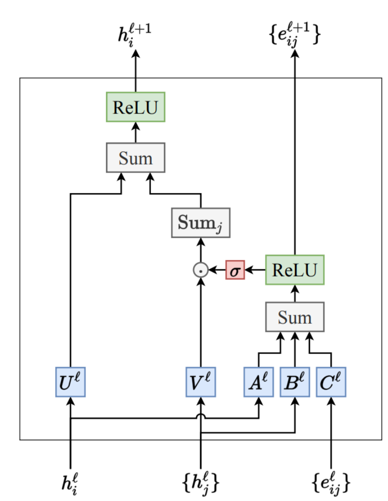


Figure 10: GatedGCN Layer

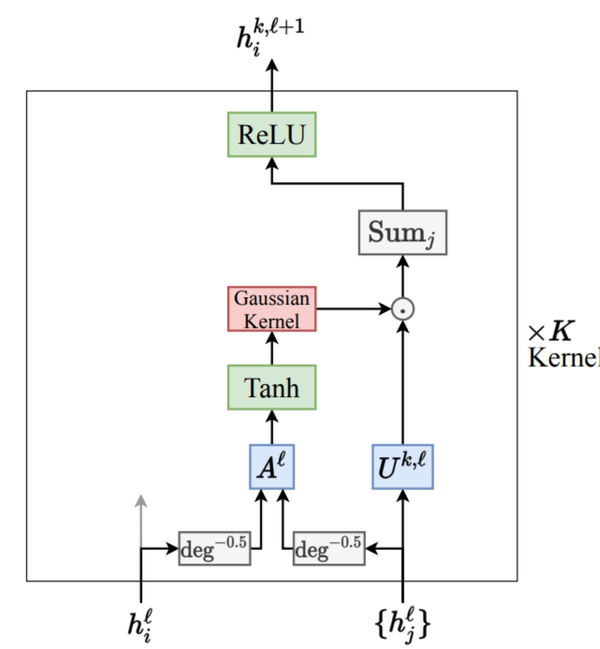


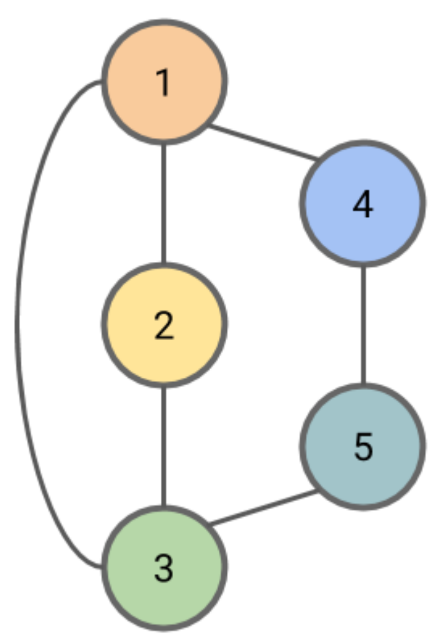
Figure 9: MoNet Layer

https://arxiv.org/pdf/2003.00982.pdf

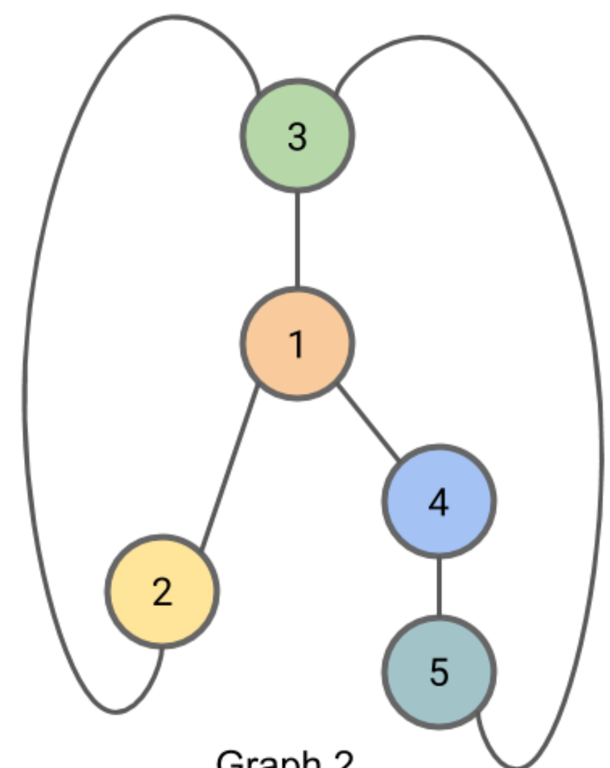
Expressiveness of Graph networks:

The Weisfeiler-Lehman Isomorphism Test

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



Graph 1

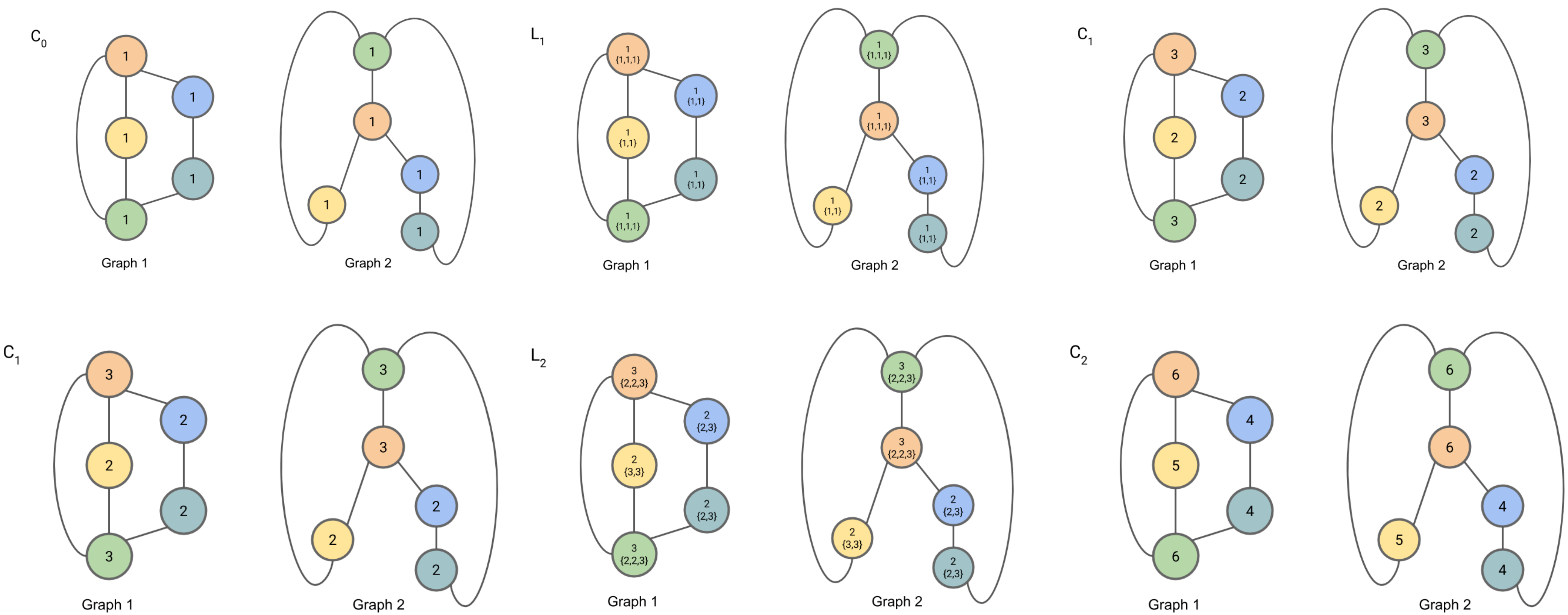


Graph 2

Expressiveness of Graph networks:

The Weisfeiler-Lehman Isomorphism Test

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

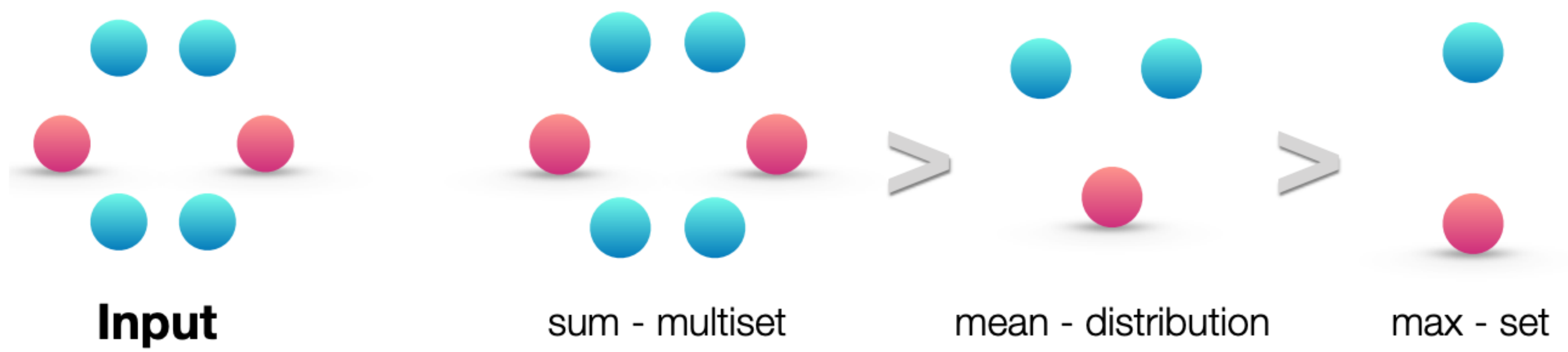


Is my GNN as powerful as WL test?

HOW POWERFUL ARE GRAPH NEURAL NETWORKS?
https://arxiv.org/pdf/1810.00826.pdf

https://davidbieber.com/post/2019-05-10-weisfeiler-lehman-isomorphism-test/

Sum is more expressive than mean…than max



(a) Mean and Max both fail   (b) Max fails

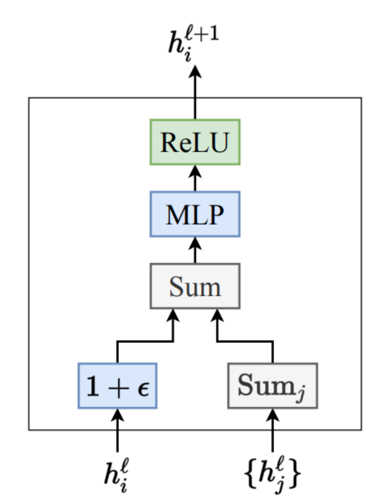
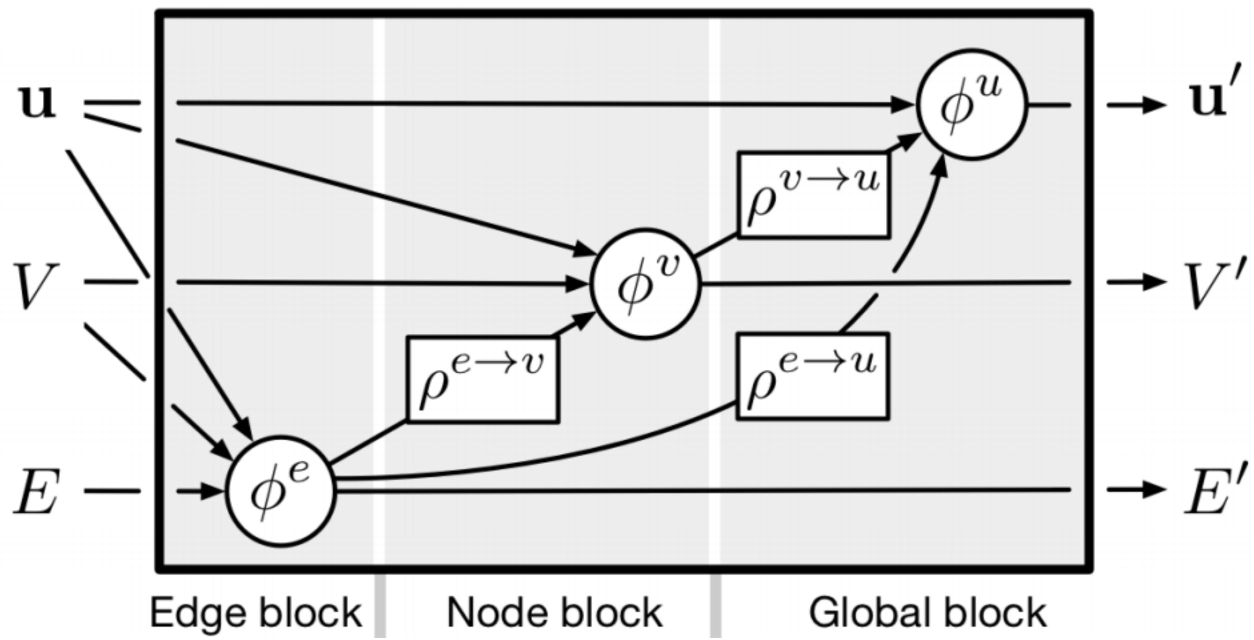
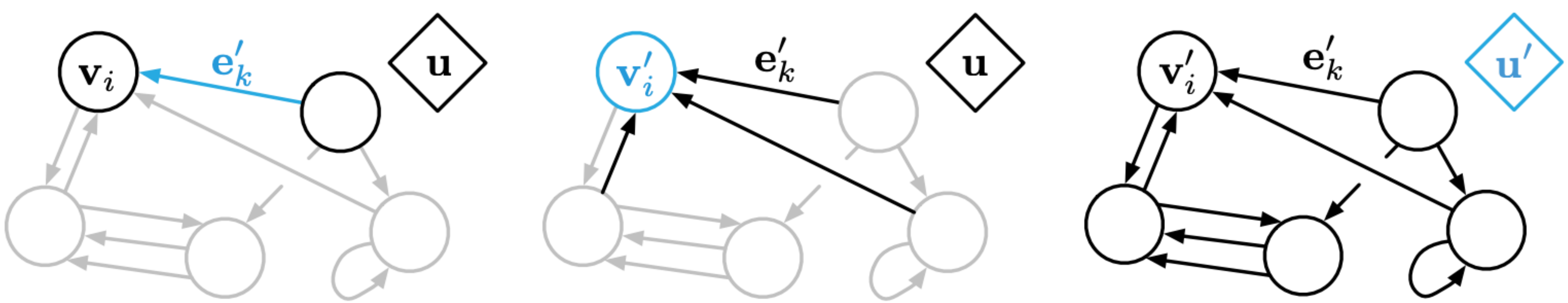


Figure 11: GIN Layer

1-WL
GNN

Toward a general form of Graph Network





(a) Edge update (b) Node update (c) Global update

Relational inductive biases, deep learning, and graph networks

https://arxiv.org/pdf/1806.01261.pdf

Learning to Simulate Complex Physics with Graph Networks

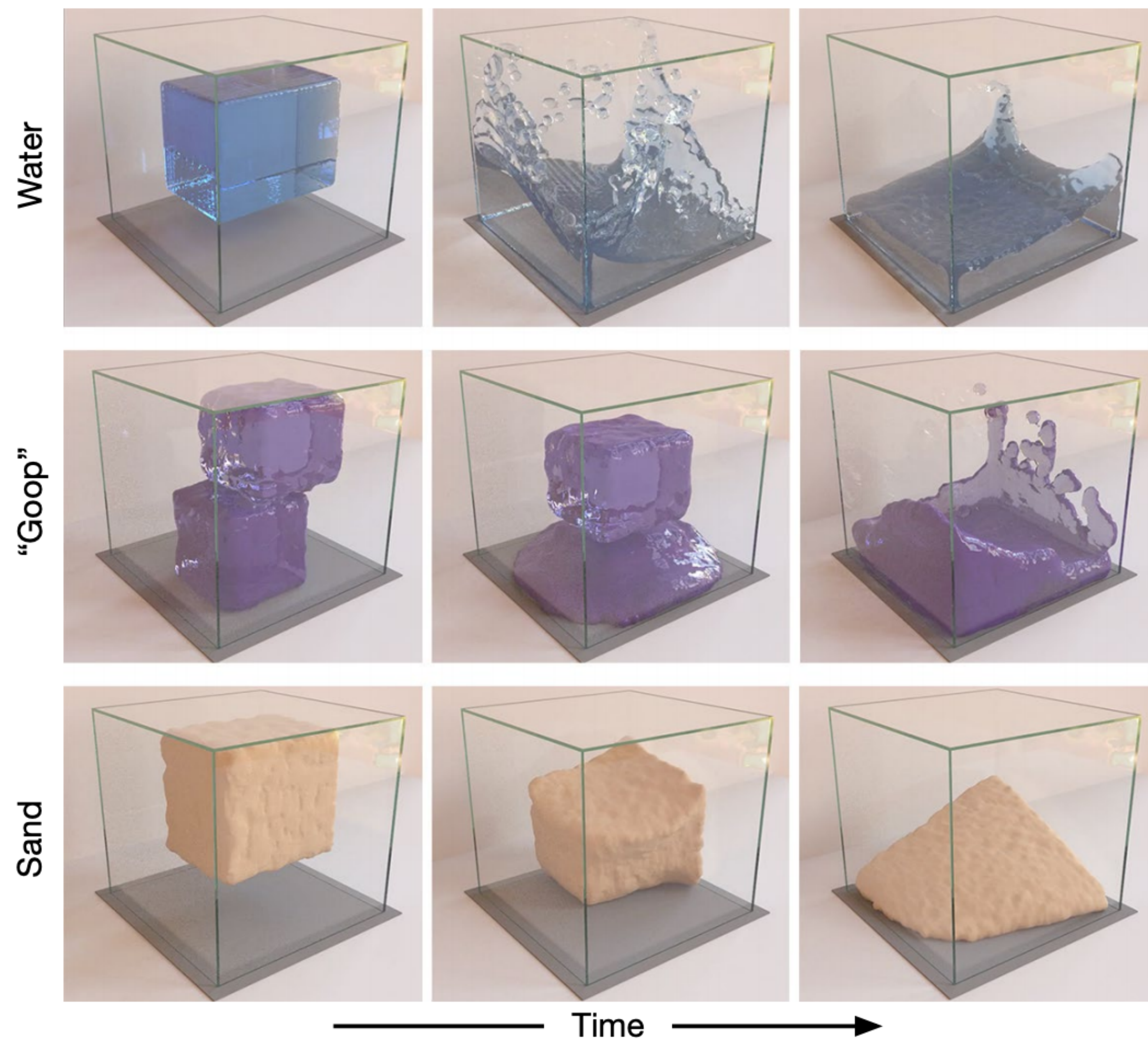


Figure 1. Rollouts of our GNS model for our WATER-3D, GOOP-3D and SAND-3D datasets. It learns to simulate rich materials at resolutions sufficient for high-quality rendering [video].

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

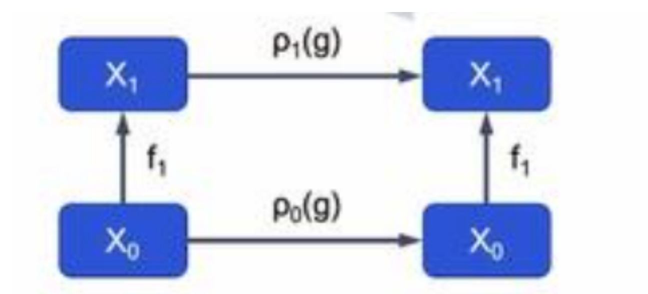
Computational challenge: how to compute efficiently?

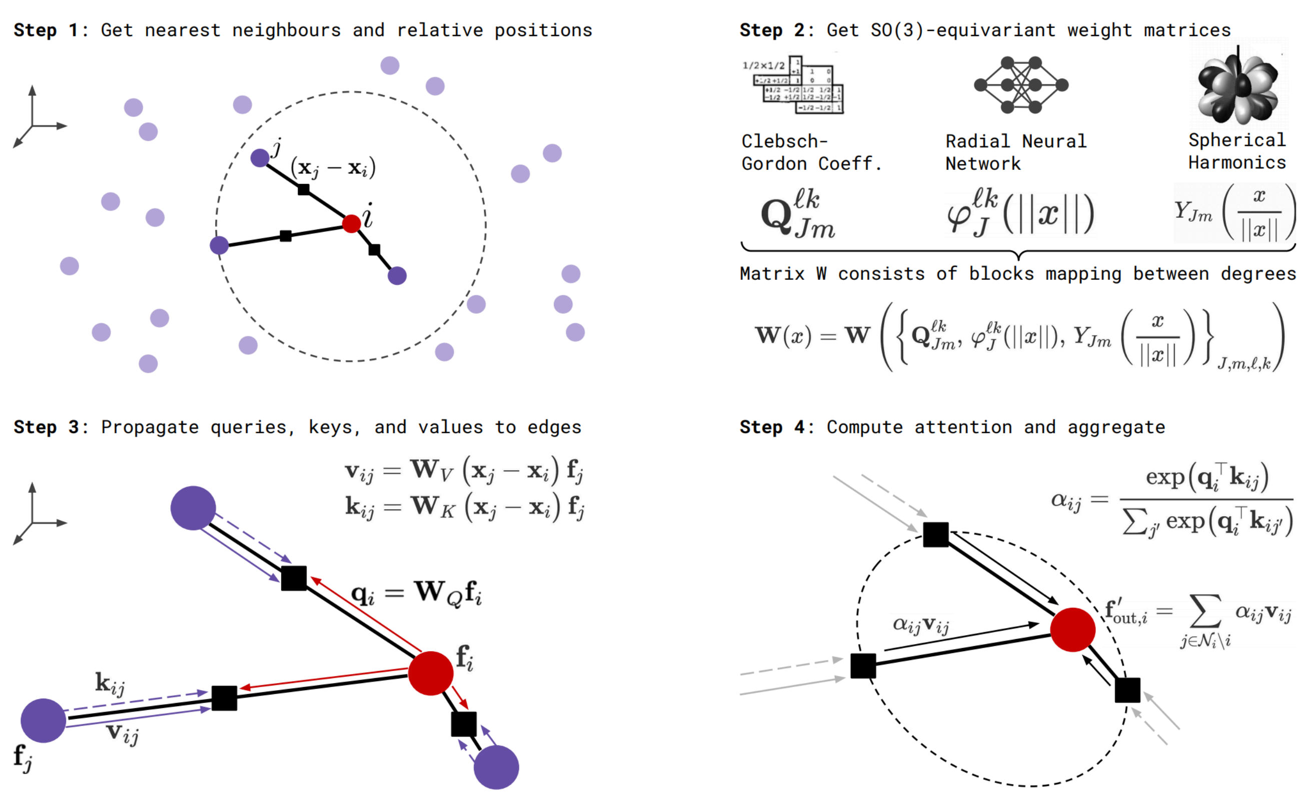
Possible transformations grow multiplicatively if we stack invariances

Stochastic approximation (one random transformation at a time)?

SE(3) equivariant transformer

equivariant vs invariant





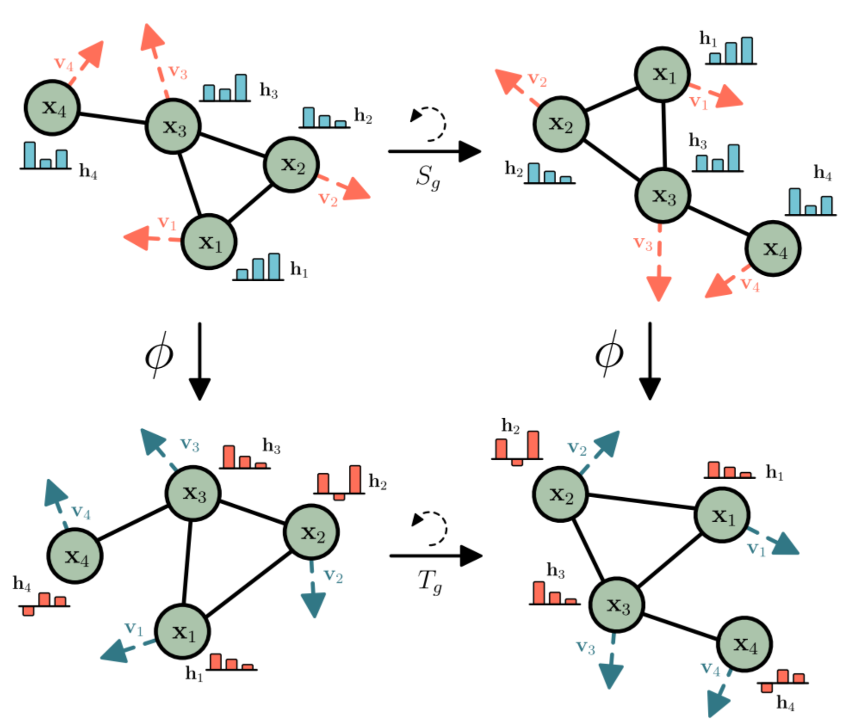
You can find the NeurIPS 2020 tutorial on equivariant networks

Fuchs et al., 2020

Design graph network for spatial coordinates
equivariant-GNNs

E(n) Equivariant Graph Neural Networks

	GNN	Radial Field	TFN	Schnet	EGNN
Edge	$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, a_{ij})$	$\mathbf{m}_{ij} = \phi_{\text{rf}}(\|\mathbf{r}_{ij}^l\|)\mathbf{r}_{ij}^l$	$\mathbf{m}_{ij} = \sum_k \mathbf{W}^{lk}\mathbf{r}_{ji}^l\mathbf{h}_i^{lk}$	$\mathbf{m}_{ij} = \phi_{\text{cf}}(\|\mathbf{r}_{ij}^l\|)\phi_{\text{s}}(\mathbf{h}_j^l)$	$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \|\mathbf{r}_{ij}^l\|^2, a_{ij})$ $\hat{\mathbf{m}}_{ij} = \mathbf{r}_{ij}^l\phi_x(\mathbf{m}_{ij})$
Agg.	$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_i = C\sum_{j\neq i} \hat{\mathbf{m}}_{ij}$
Node	$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{m}_i$	$\mathbf{h}_i^{l+1} = w^{ll}\mathbf{h}_i^l + \mathbf{m}_i$	$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right)$ $\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \hat{\mathbf{m}}_i$
	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