Deep Learning and Generative AI

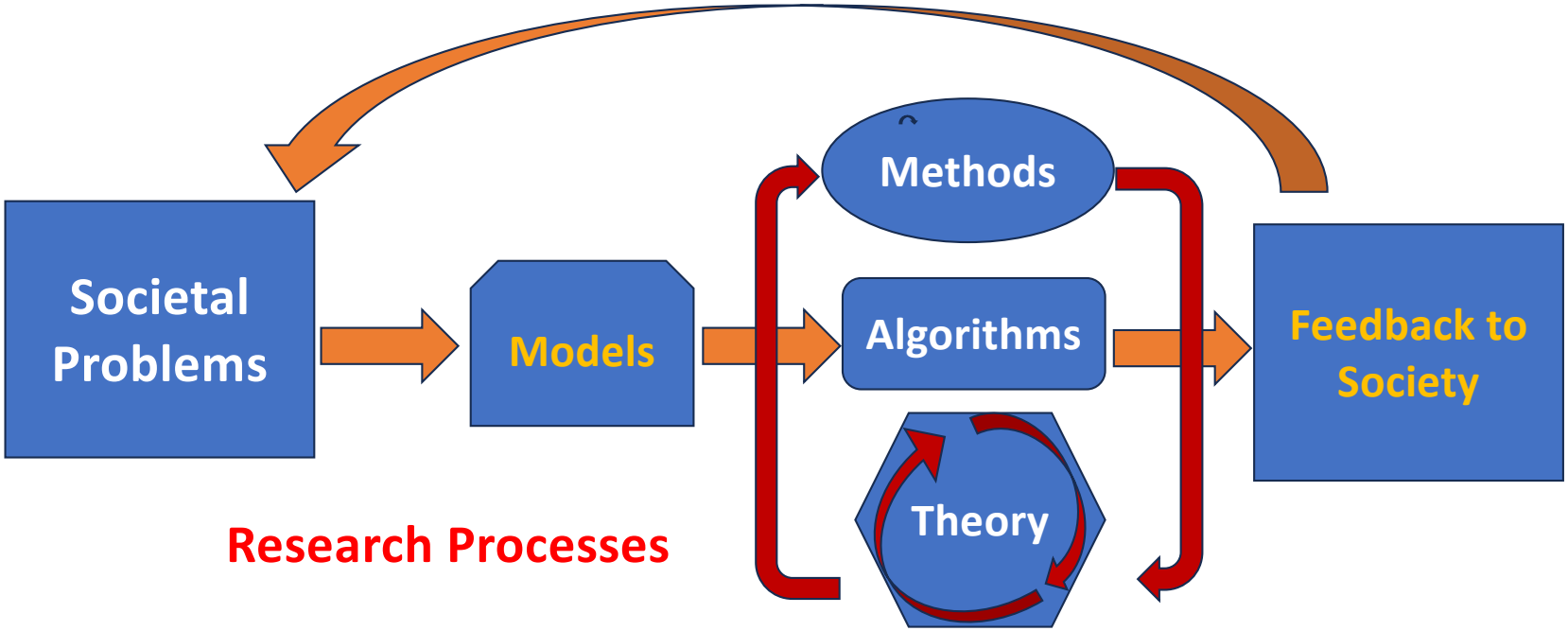
Jianqing Fan

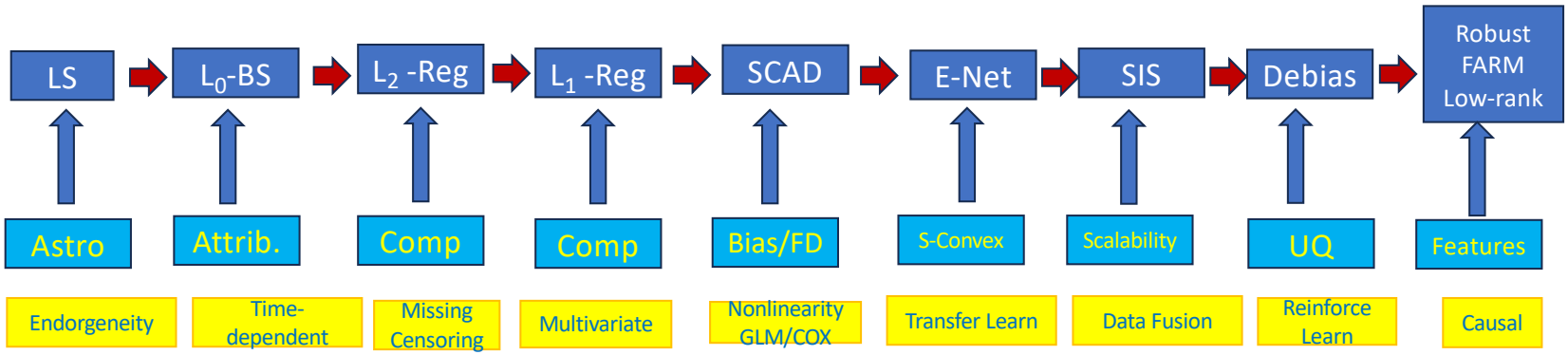
Princeton University

https://fan.princeton.edu



Research Processes





1. Introduction to Deep Learning and Generative AI

1.1. Introduction

1.2. FNN, ResNet, CNN, RNN

1.3. Transformers & LLMs

1.4. Watermarking

1.5. Generative Adversary Nets

1.6 Diffusion Models

1.7. Flow matching

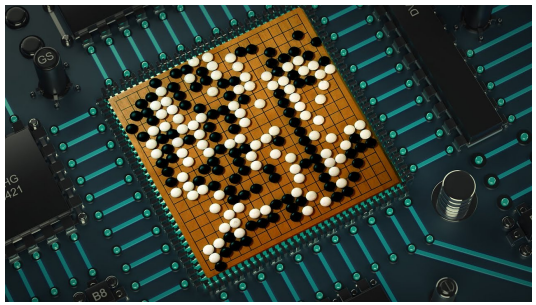
1.8. Training algorithms

★Fan, J., Ma, C. and Zhong, Y. (2021). A selective overview of deep learning. *Stat. Sci.*, 264-290.

1.1 Introduction

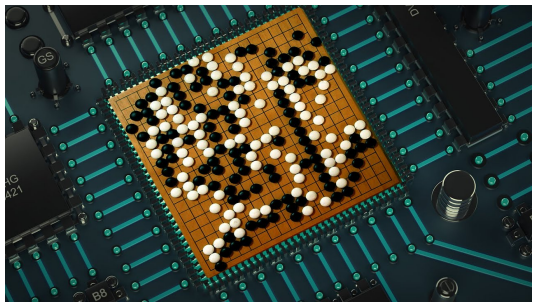
Interface of Learning and Decision

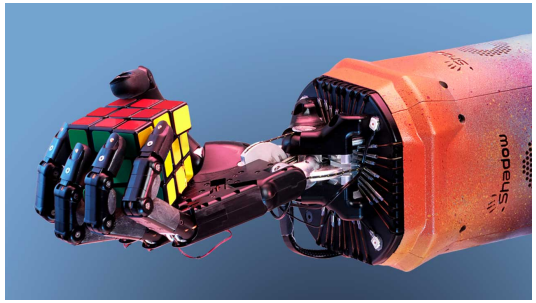
- **Games**: Atari, Go, Starcraft, Texas hold'em, ...
- **Control**: robotic hand, automatic driving, fleet management, ...
- **Prediction**: peotein folding, LLM, generative AI ...
- **Causality**: Molecular mechnism, policy eval, ...



Interface of Learning and Decision

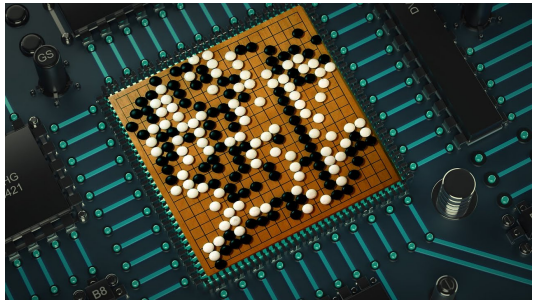
- **Games**: Atari, Go, Starcraft, Texas hold'em, ...
- **Control**: robotic hand, automatic driving, fleet management, ...
- **Prediction**: peotein folding, LLM, generative AI ...
- **Causality**: Molecular mechnism, policy eval, ...

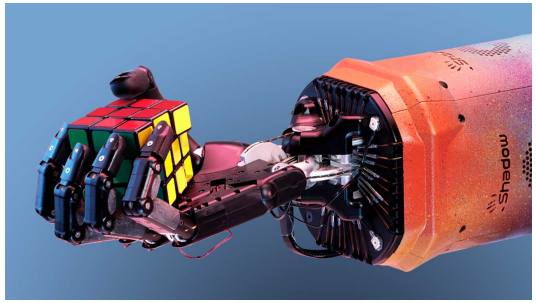




Interface of Learning and Decision

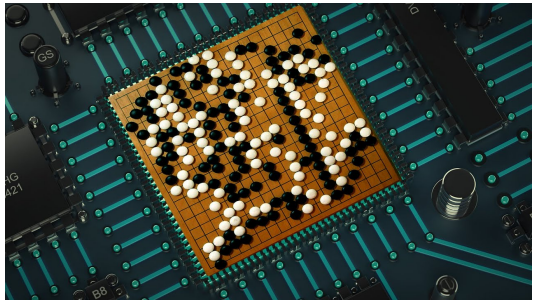
- **Games**: Atari, Go, Starcraft, Texas hold'em, ...
- **Control**: robotic hand, automatic driving, fleet management, ...
- **Prediction**: peotein folding, LLM, generative AI ...
- **Causality**: Molecular mechnism, policy eval, ...

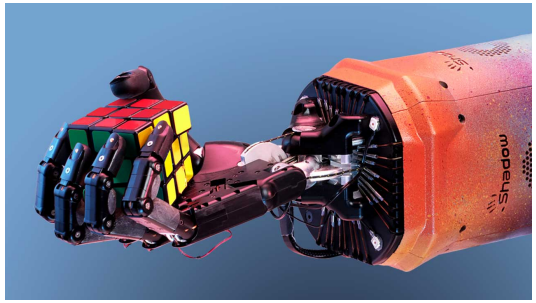




Interface of Learning and Decision

- **Games**: Atari, Go, Starcraft, Texas hold'em, ...
- **Control**: robotic hand, automatic driving, fleet management, ...
- **Prediction**: peotein folding, LLM, generative AI ...
- **Causality**: Molecular mechnism, policy eval, ...

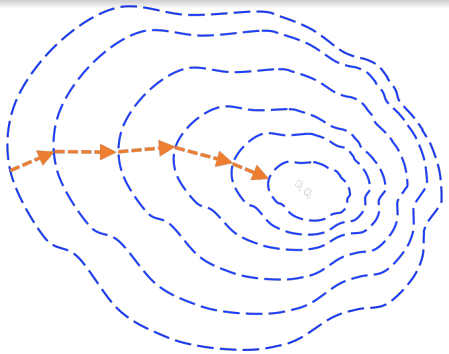




Drivers of successes

modern computation + **big-data**

Deep Q-Network 24/39 (Minh et al, 15)

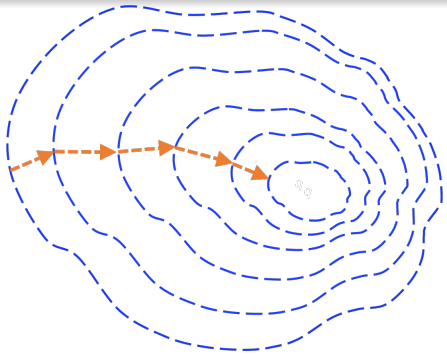


- ★ Representation power of deep neural nets + statist learning algorithms.
- ★ Policy optimization framework of Markov decision processes.
- ★ Help solve complex problems such as high-dim prediction, attribution, causality, mechnism analysis, treatment evaluation in a **data-diven** way.

Drivers of successes

DL + **modern computation** + **big-data**

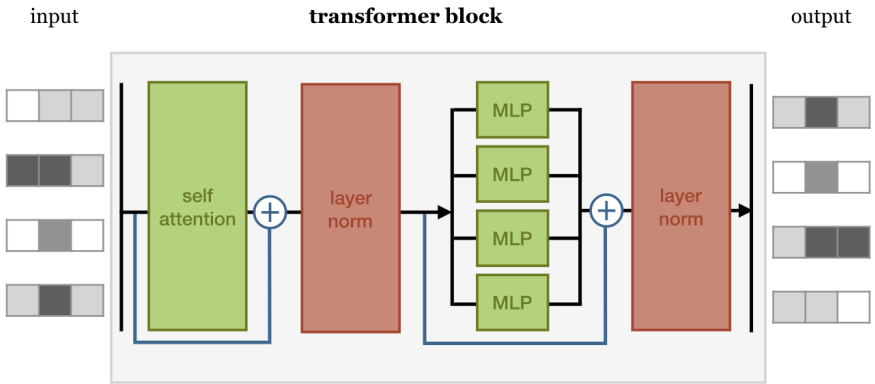
Deep Q-Network 24/39 (Minh et al, 15)



★ Representation power of deep neural nets + statist learning algorithms.

★ Policy optimization framework of Markov decision processes.

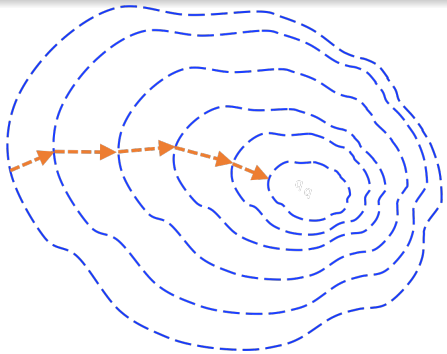
★ Help solve complex problems such as high-dim prediction, attribution, causality, mechnism analysis, treatment evaluation in a **data-diven** way.



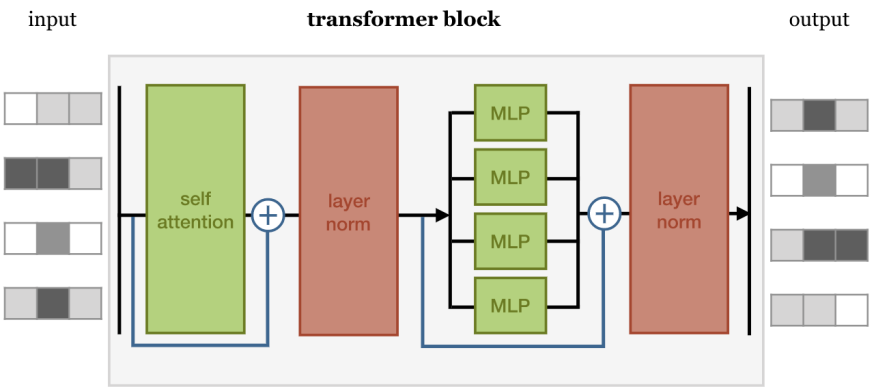
Drivers of successes

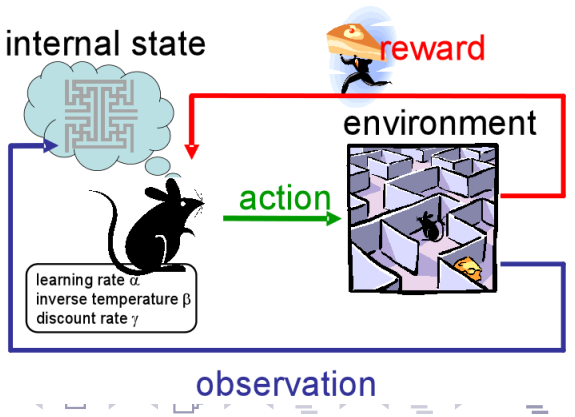
DL + **RL** + **modern computation** + **big-data**

Deep Q-Network 24/39 (Minh et al, 15)



- ★ Representation power of deep neural nets + statist learning algorithms.
- ★ Policy optimization framework of Markov decision processes.
- ★ Help solve complex problems such as high-dim prediction, attribution, causality, mechnism analysis, treatment evaluation in a **data-diven** way.

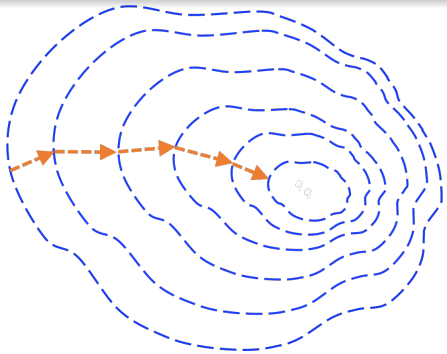




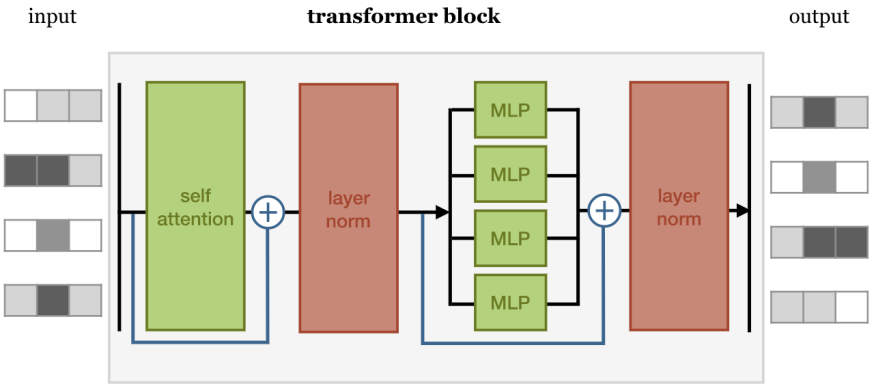
Drivers of successes

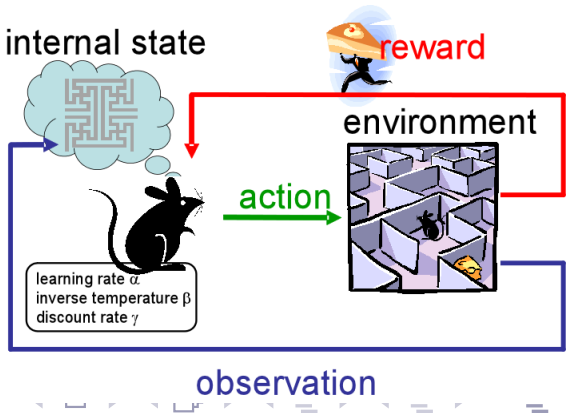
DL + **RL** + **modern computation** + **big-data**

Deep Q-Network 24/39 (Minh et al, 15)



- ★ Representation power of deep neural nets + statist learning algorithms.
- ★ Policy optimization framework of Markov decision processes.
- ★ Help solve complex problems such as high-dim prediction, attribution, causality, mechnism analysis, treatment evaluation in a **data-diven** way.





What is deep neural network?

■High-dim function approximation using deep neural nets.

Input: $\mathbf{h}^{(0)} = \mathbf{x} \in \mathbb{R}^p$;

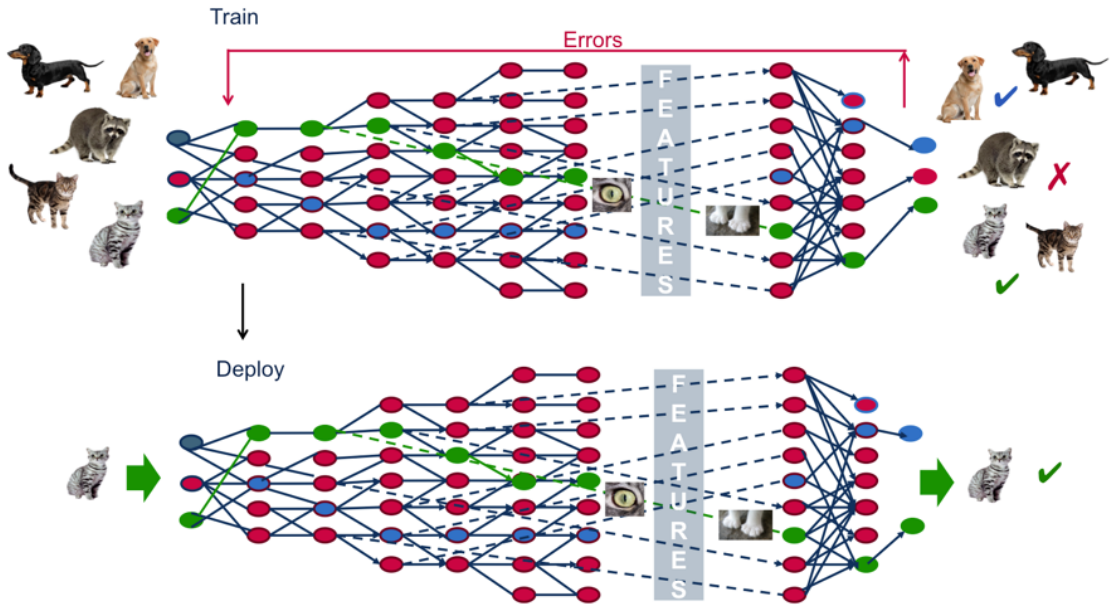
$$\mathbf{h}^{(1)} = \sigma\left(\mathbf{W}^{(1)}\mathbf{h}^{(0)} + \mathbf{b}^{(1)}\right);$$

$$\mathbf{h}^{(2)} = \sigma\left(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right);$$

$$\vdots$$

$$\mathbf{h}^{(L)} = \sigma\left(\mathbf{W}^{(L-1)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L-1)}\right)$$

Output: $\mathbf{y} = g(\mathbf{h}^{(L)}) \in \mathbb{R}^K$.



Output of last layer: $\mathbf{h}^{(L)}$ regarded as machine **learned features**

★$\sigma(\cdot)$ called activation function; ★$\mathbf{W}$ weight matrix; ★$\mathbf{b}$ bias.

Why So Successful Nowadays?

Great for bias-var trade-off!

- ★ Deep nets make high-d function approx (representation) flexible
- ★ Arrivals of Big Data reduce variance
- ★ Modern comp power and algorithms make large scale comp. feasible.
- ★ Adaptive to low dimensional structure.

  e.g. 2-d rate for $f(\mathbf{x}) = f_1(x_1) + f_2(x_2, x_3, x_8) + f_3(f_4(x_1, x_3), f_5(x_4, x_9))$

Scalable High-d Nonparam Methods

Why So Successful Nowadays?

Great for bias-var trade-off!

- ★ Deep nets make high-d function approx (representation) flexible
- ★ Arrivals of Big Data reduce variance
- ★ Modern comp power and algorithms make large scale comp. feasible.
- ★ Adaptive to low dimensional structure.

  e.g. 2-d rate for $f(\mathbf{x}) = f_1(x_1) + f_2(x_2, x_3, x_8) + f_3(f_4(x_1, x_3), f_5(x_4, x_9))$

Scalable High-d Nonparam Methods

Why So Successful Nowadays?

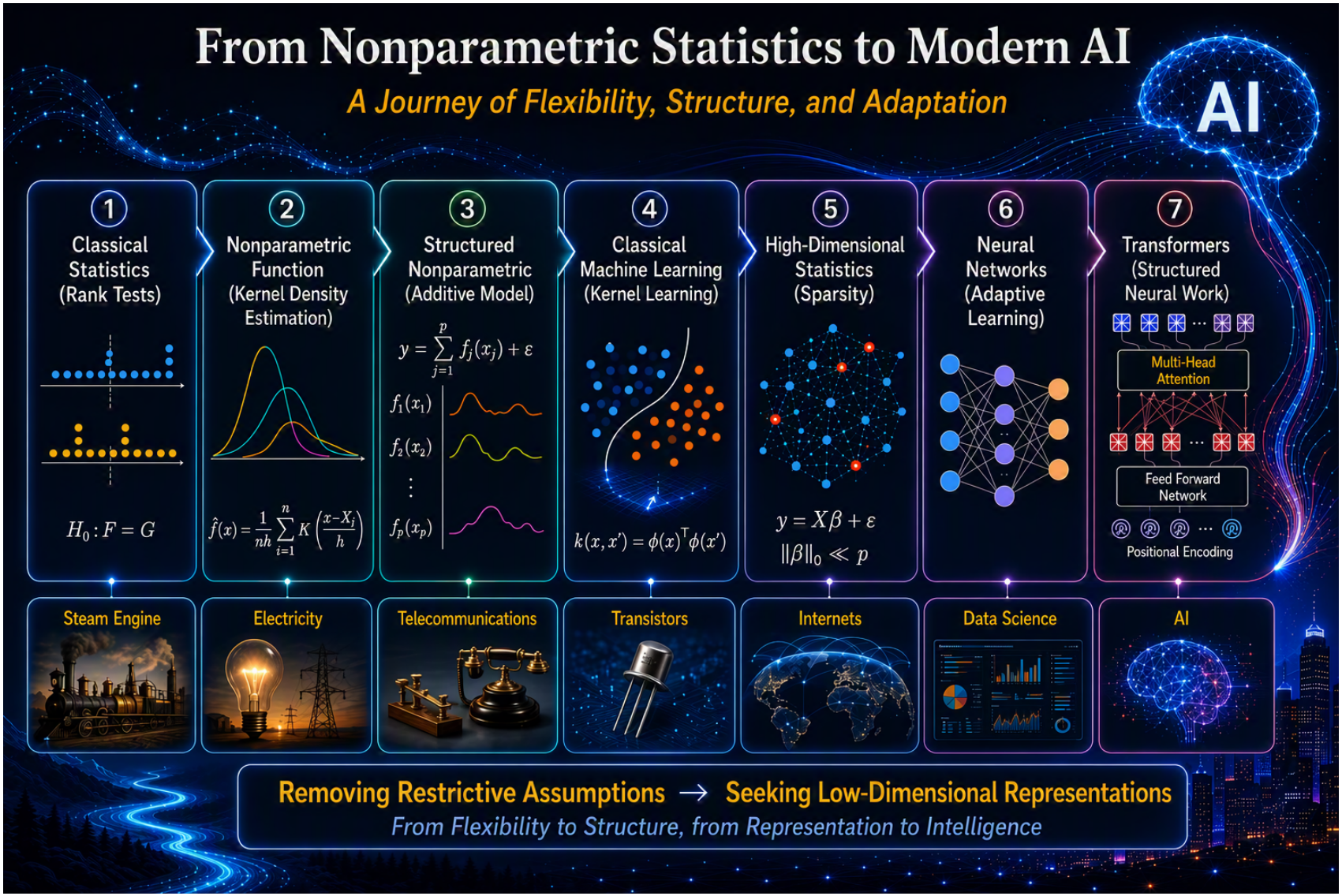
Great for bias-var trade-off!

- ★ Deep nets make high-d function approx (representation) flexible
- ★ Arrivals of Big Data reduce variance
- ★ Modern comp power and algorithms make large scale comp. feasible.
- ★ Adaptive to low dimensional structure.

  e.g. 2-d rate for $f(\mathbf{x}) = f_1(x_1) + f_2(x_2, x_3, x_8) + f_3(f_4(x_1, x_3), f_5(x_4, x_9))$

Scalable High-d Nonparam Methods

Evolution of Statistics



Bias–Variance, Flexibility–Learnability

1.2 FNN, ResNet, CNN, RNN

★**Feed-Forward Neural Networks**

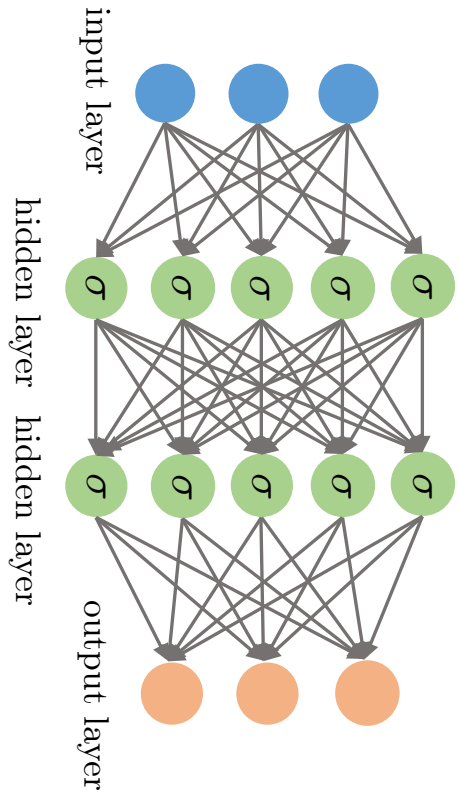
= Multi Layer Perceptron (MLP)

Feed-forward neural networks

Let $\mathbf{h}^{(0)} \triangleq \mathbf{x}$. **Feed-forward neural networks** define recursively

$$\mathbf{h}^{(\ell)} = \sigma\left(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right)$$

- $\mathbf{W}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ are **weight matrix** and **bias**.
- $\sigma(\cdot)$ is called **activation function**.
- Written as $f_{NN}(\mathbf{x};\theta) = \mathbf{W}_{L+1}\sigma(\mathbf{W}_L \ldots \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)+\mathbf{b}_2)\ldots+\mathbf{b}_L)$
- bias can be absorbed in $\mathbf{x}$



From output $\mathbf{h}^{(L)}$, run reg. or multinomial logistic reg:

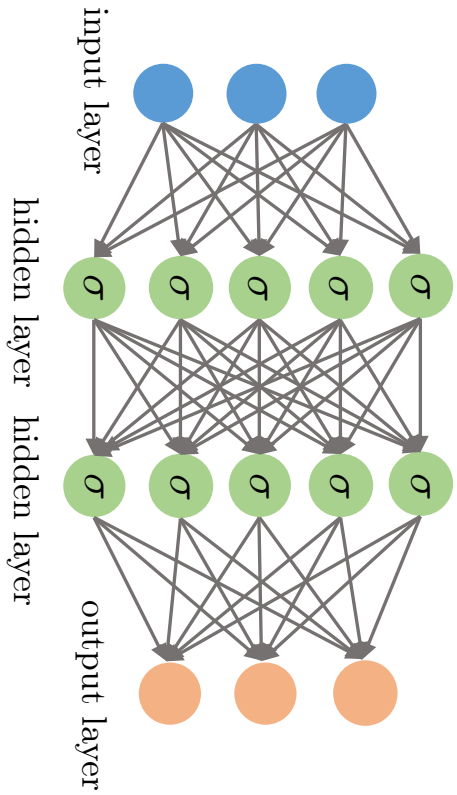
$$f_k(\mathbf{x}) \triangleq \frac{e^{z_k}}{\sum_k e^{z_k}}, \quad \forall k \in [K], \quad \text{where} \quad \mathbf{z} = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)} \in \mathbb{R}^K.$$

Feed-forward neural networks

- Let $\mathbf{h}^{(0)} \triangleq \mathbf{x}$. **Feed-forward neural networks** define recursively

$$\mathbf{h}^{(\ell)} = \sigma\left(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right)$$

  - ★ $\mathbf{W}^{(\ell)}$ and $\mathbf{b}^{(\ell)}$ are **weight matrix** and **bias**.
  - ★ $\sigma(\cdot)$ is called **activation function**.
  - ★ Written as $f_{NN}(\mathbf{x};\theta) = \mathbf{W}_{L+1}\sigma(\mathbf{W}_L \ldots \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)+\mathbf{b}_2)\ldots+\mathbf{b}_L)$
  - ★ bias can be absorbed in $\mathbf{x}$



- From output $\mathbf{h}^{(L)}$, run reg. or multinomial logistic reg:

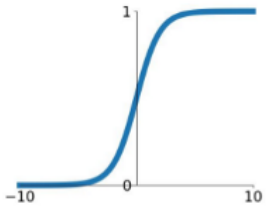
$$f_k(\mathbf{x}) \triangleq \frac{e^{z_k}}{\sum_k e^{z_k}}, \quad \forall k \in [K], \quad \text{where} \quad \mathbf{z} = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)} \in \mathbb{R}^K.$$

Common activation functions

Activation Functions

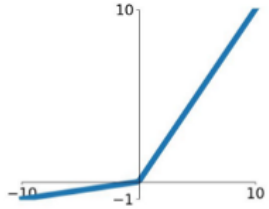
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



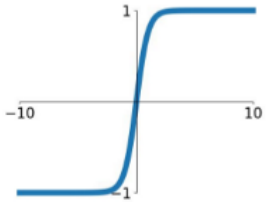
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

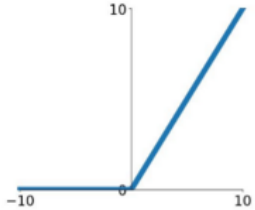


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

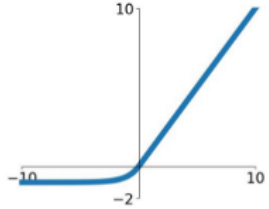
ReLU

$\max(0, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- ReLU (Rectified Linear Unit) function: $[\sigma(\mathbf{z})]_j = \max\{z_j, 0\}$.
- Hyperbolic tangent: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1} = 1 - \frac{2}{e^{2x}+1}$

Residual neural network

- ★ also called residual network or ResNet
- ★ stabilize the training and convergence of DNNs with hundreds of layers
- ★ used in transformer models (e.g., BERT and GPT models such as ChatGPT), AlphaGo Zero, AlphaStar, and AlphaFold systems.

It has a skip connection in the subnet.

The output is $\mathbf{x} + \mathbf{F}(\mathbf{x})$ or more generally

$\mathbf{x} + \mathbf{MF}(\mathbf{x})$ with trainable $\mathbf{M}$ that makes

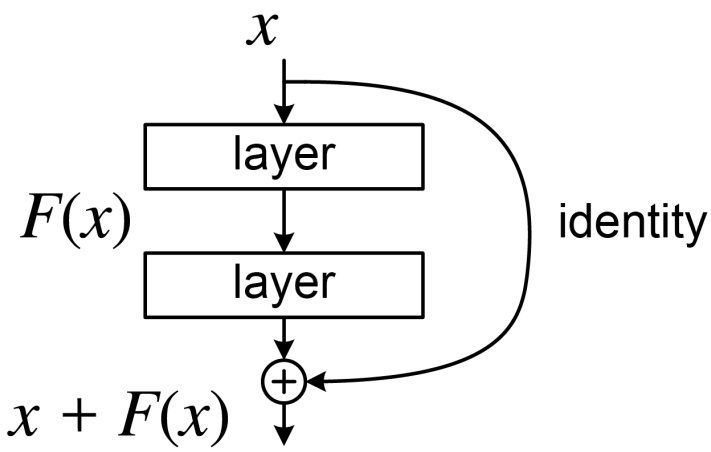
the output dimension compatible.

Residual neural network

- ★ also called residual network or ResNet
- ★ stabilize the training and convergence of DNNs with hundreds of layers
- ★ used in transformer models (e.g., BERT and GPT models such as ChatGPT), AlphaGo Zero, AlphaStar, and AlphaFold systems.

■It has a skip connection in the subnet

The output is $\mathbf{x} + \mathbf{F}(\mathbf{x})$ or more generall $\mathbf{x} + \mathbf{MF}(\mathbf{x})$ with trainable $\mathbf{M}$ that makes the output dimension compatible.



Regression and Classification

Training data: $\{(y_i, \mathbf{x}_i)\}_{i=1}^n$ — examples

Last-layer Output: $f(\mathbf{x}_i;\theta) \triangleq (f_1(\mathbf{x}_i;\theta),\ldots,f_K(\mathbf{x}_i;\theta))^\top$, $\quad \theta \triangleq \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)} : 1 \le \ell \le L+1\}$.

Regression: $\min \sum_{i=1}^n (y_i - f(\mathbf{x}_i;\theta))^2$

Classification: $\min \sum_{i=1}^n \mathcal{L}(f(\mathbf{x}_i;\theta), y_i)$ with cross entropy loss:

$$\mathcal{L}(f(\mathbf{x}_i;\theta), y_i) = -\sum_{k=1}^K 1\{y_i = k\} \log f_k(\mathbf{x}_i;\theta)$$

★the same as negative log-likelihood of multinomial distribution

★FNN accomodates **multivariate nonparametric** prediction.

Regression and Classification

Training data: $\{(y_i, \mathbf{x}_i)\}_{i=1}^n$ — examples

Last-layer Output: $f(\mathbf{x}_i;\theta) \triangleq (f_1(\mathbf{x}_i;\theta),\ldots,f_K(\mathbf{x}_i;\theta))^\top$, $\quad \theta \triangleq \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)} : 1 \le \ell \le L+1\}$.

Regression: $\min \sum_{i=1}^n (y_i - f(\mathbf{x}_i;\theta))^2$

Classification: $\min \sum_{i=1}^n \mathcal{L}(f(\mathbf{x}_i;\theta), y_i)$ with cross entropy loss:

$$\mathcal{L}(f(\mathbf{x}_i;\theta), y_i) = -\sum_{k=1}^K 1\{y_i = k\} \log f_k(\mathbf{x}_i;\theta)$$

★the same as negative log-likelihood of multinomial distribution

★FNN accomodates **multivariate nonparametric** prediction.

Convolutional Neural Networks

Drawbacks of general feed-forward neural networks:

- ★ Number of parameters grows quickly width and depth
- ★ Does not incorporate structures of the input (say images)

New characteristics of CNN:

- ★ Weight sharing: some parameters are identical across locations
- ★ Convolutional layer is designed for **images**

Convolutional Neural Networks

Drawbacks of general feed-forward neural networks:

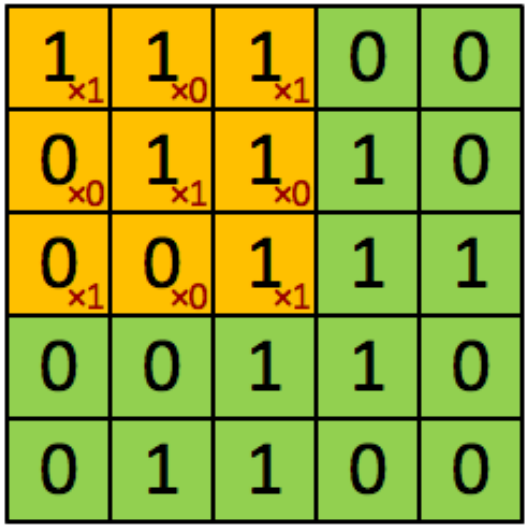
- ★ Number of parameters grows quickly width and depth
- ★ Does not incorporate structures of the input (say images)

New characteristics of CNN:

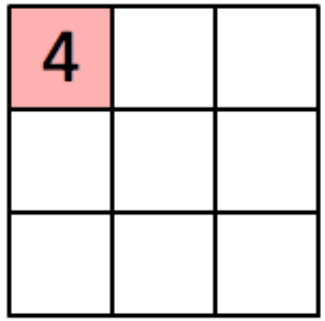
- ★ Weight sharing: some parameters are identical across locations
- ★ Convolutional layer is designed for **images**

Convolutional layer in 2D

Convolution in 2D



Image



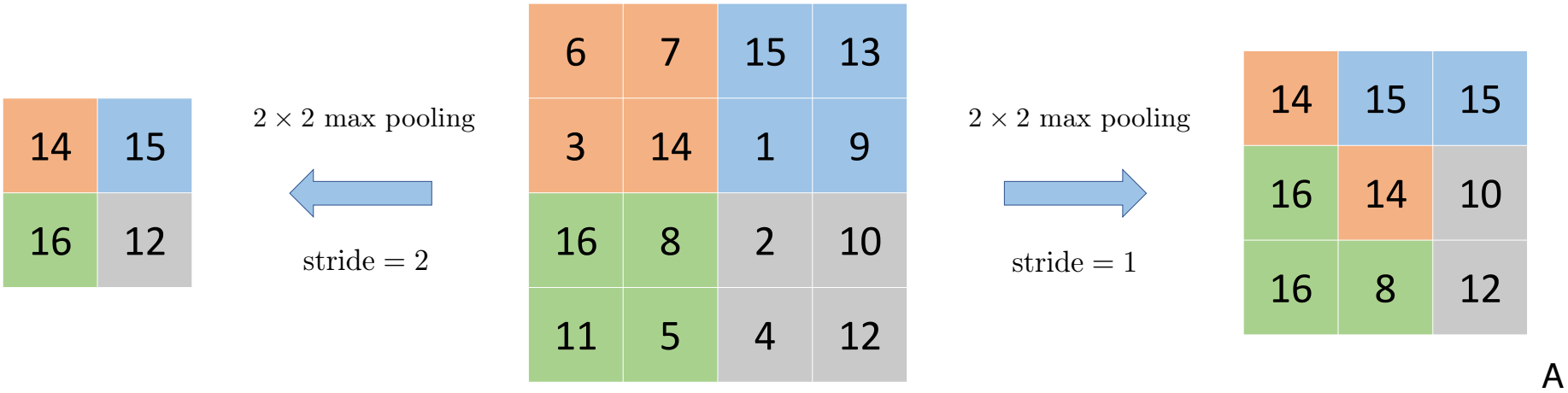
Convolved Feature

3×3 filter

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

- Filters are reused across locations (weight sharing)
- Can and usually have multiple filters on the same input
- Usually followed by the element-wise ReLU activation

Pooling layer / down-sampling layer



2×2 max pooling layer extracts the maximum of 2 by 2 neighboring pixels spatially.

- ★ Pooling can be done with different strides
- ★ Pooling is used to reduce features and computation

Example: LeNet-5

Architecture of LeNet-5 by LeCun et al. (1998) for digits recognition

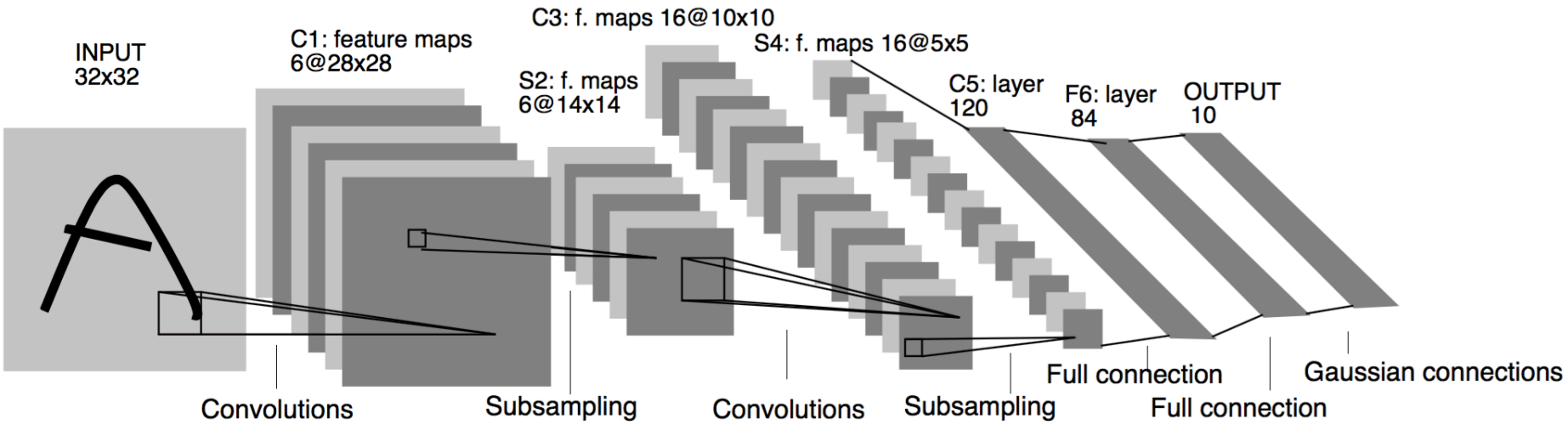


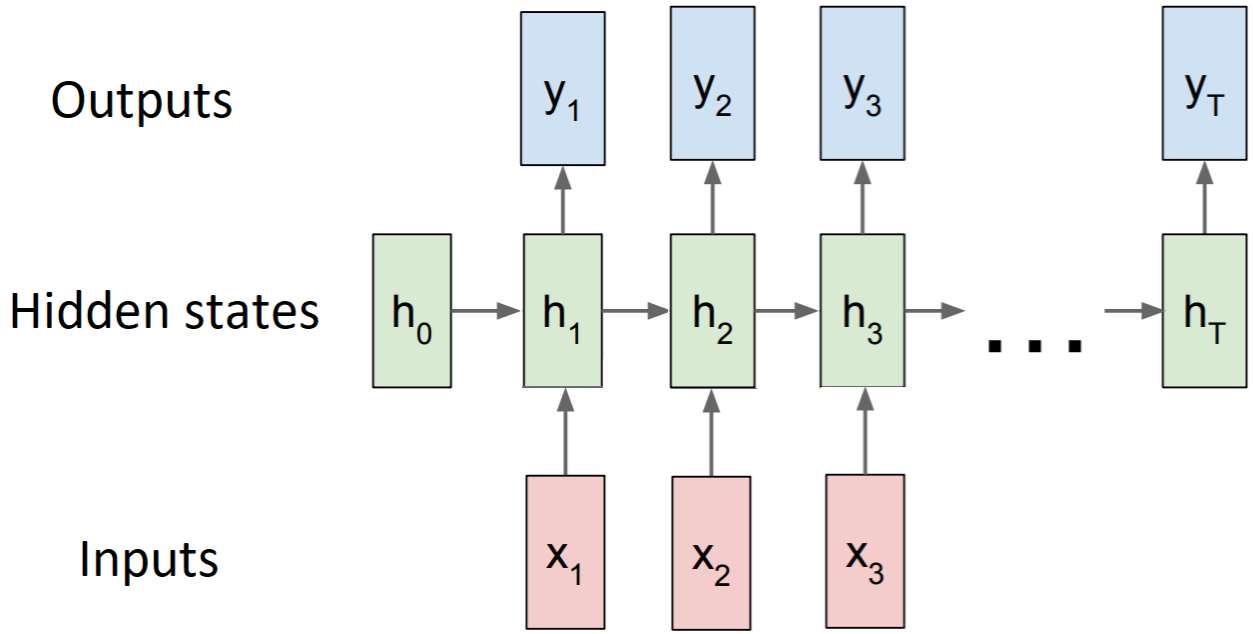
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- **Convolution**: feature extraction (1^{st} layer: (5*5+1)*6. 2^{nd} *layer*: (5*5*6+1)*16)
- **Subsampling**: data compression (**nonlinear gating**)

Deep architecture: high expressive power and low bias

Recurrent neural networks

■ **RNN** for time series, speech recognition, machine translation



★ Recurrence: $\mathbf{h}_t = \mathbf{f}_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$, e.g.

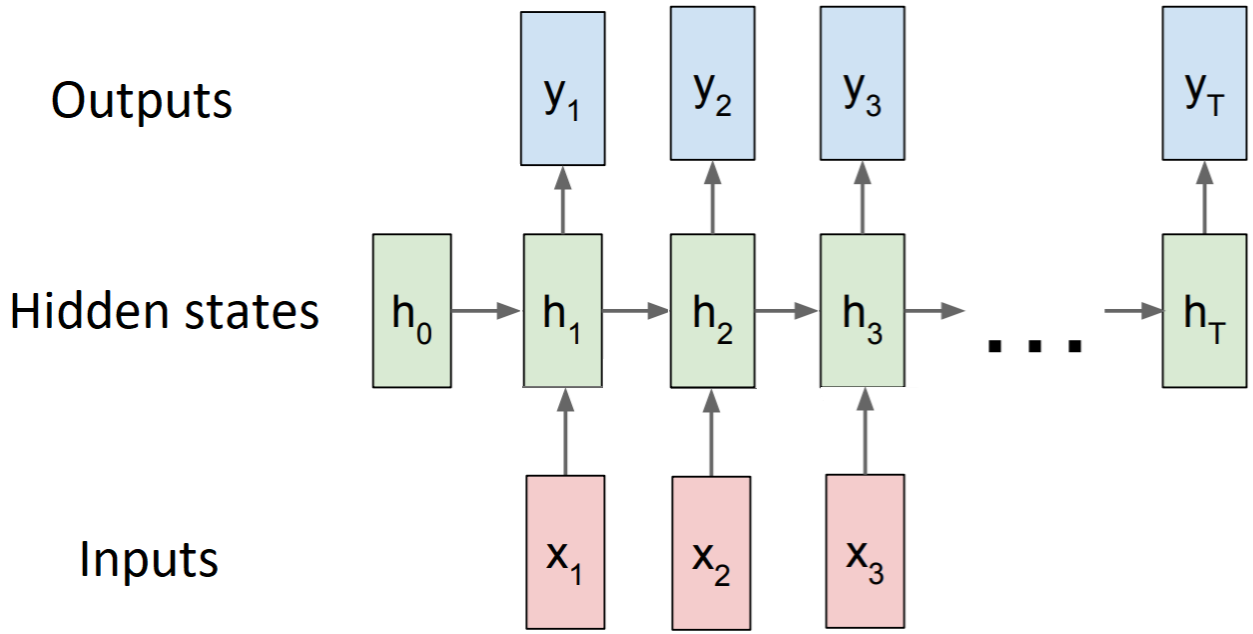
$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_\mathbf{h}\right), \qquad \tanh(u) = \frac{e^{2u}-1}{e^{2u}+1},$$

$$\mathbf{y}_t = \sigma\left(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_\mathbf{z}\right),$$

★ **Localization in time**: exploding/vanishing gradient $\prod_t(\partial \mathbf{h}_{t+1}/\partial \mathbf{h}_t)$

Recurrent neural networks

■ **RNN** for time series, speech recognition, machine translation



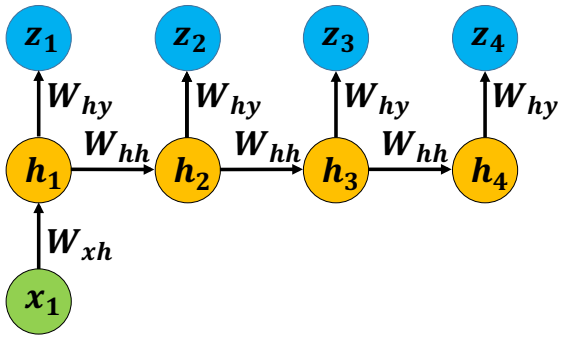
★ Recurrence: $\mathbf{h}_t = \mathbf{f}_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$, e.g.

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{b}_\mathbf{h}\right), \qquad \tanh(u) = \frac{e^{2u}-1}{e^{2u}+1},$$

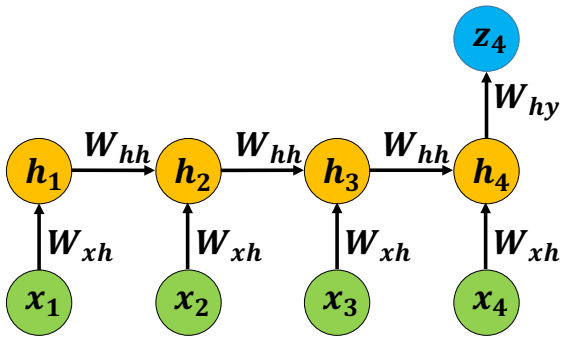
$$\mathbf{y}_t = \sigma\left(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_\mathbf{z}\right),$$

★ **Localization in time**: exploding/vanishing gradient $\prod_t (\partial \mathbf{h}_{t+1}/\partial \mathbf{h}_t)$

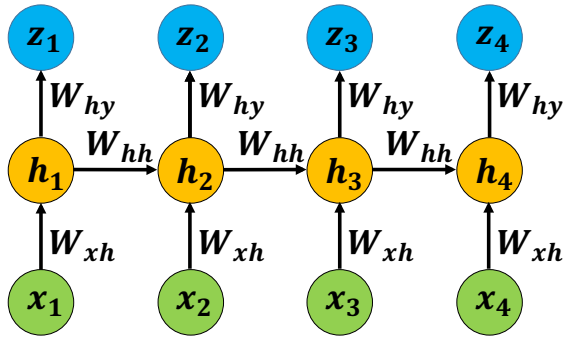
Variations of networks



(a) One-to-many



(b) Many-to-one



(c) Many-to-many

(a) **One-to-many**. Image captioning: input is an image and outputs are a series of words.

(b) **Many-to-one**. Text sentiment classification: input is words in a sentence and output is a label

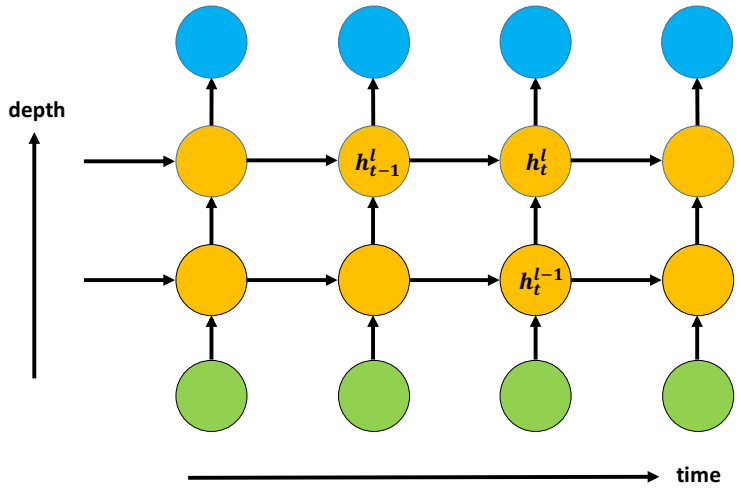
(c) **Many-to-many**. Machine translation: inputs are words in a language and outputs are words of another language

Additional Variations of networks

- **GRU**: Gated recurrent units introduce gates to decide how much old hidden states units to keep.

- **LSTM**: Long short-term memory introduces additional cell states, forget, input and output gates to control the evolution of $\mathbf{h}_t$.

- **Multilayer RNN**: $\mathbf{h}_t^\ell = \tanh\left[\mathbf{W}^\ell \begin{pmatrix} \mathbf{h}_t^{\ell-1} \\ \mathbf{h}_{t-1}^\ell \\ 1 \end{pmatrix}\right], \quad \ell \in [L], \qquad \mathbf{h}_t^0 \triangleq \mathbf{x}_t.$

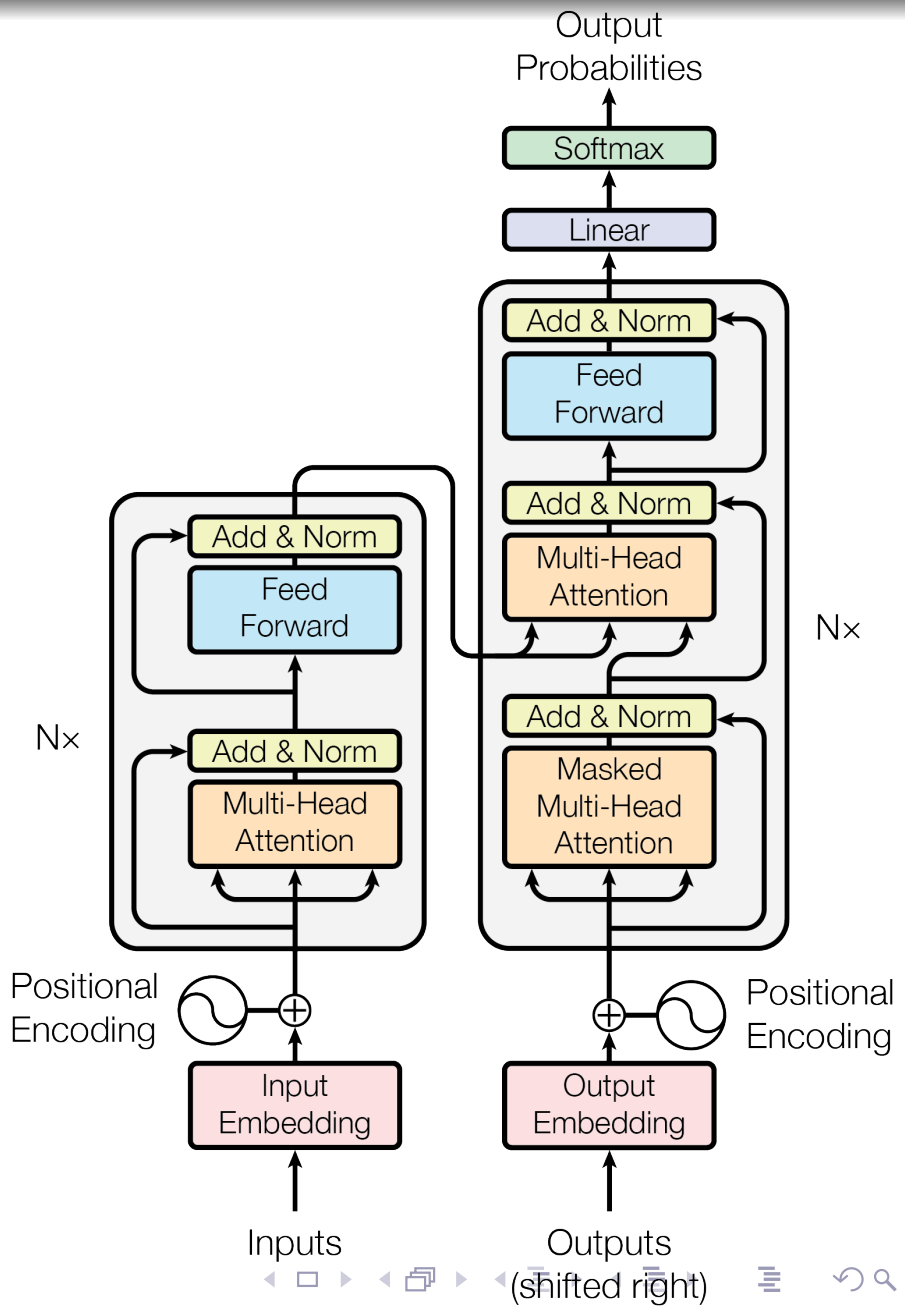


1.3 Transformers

Transformers have revolutionized the field of modern AI

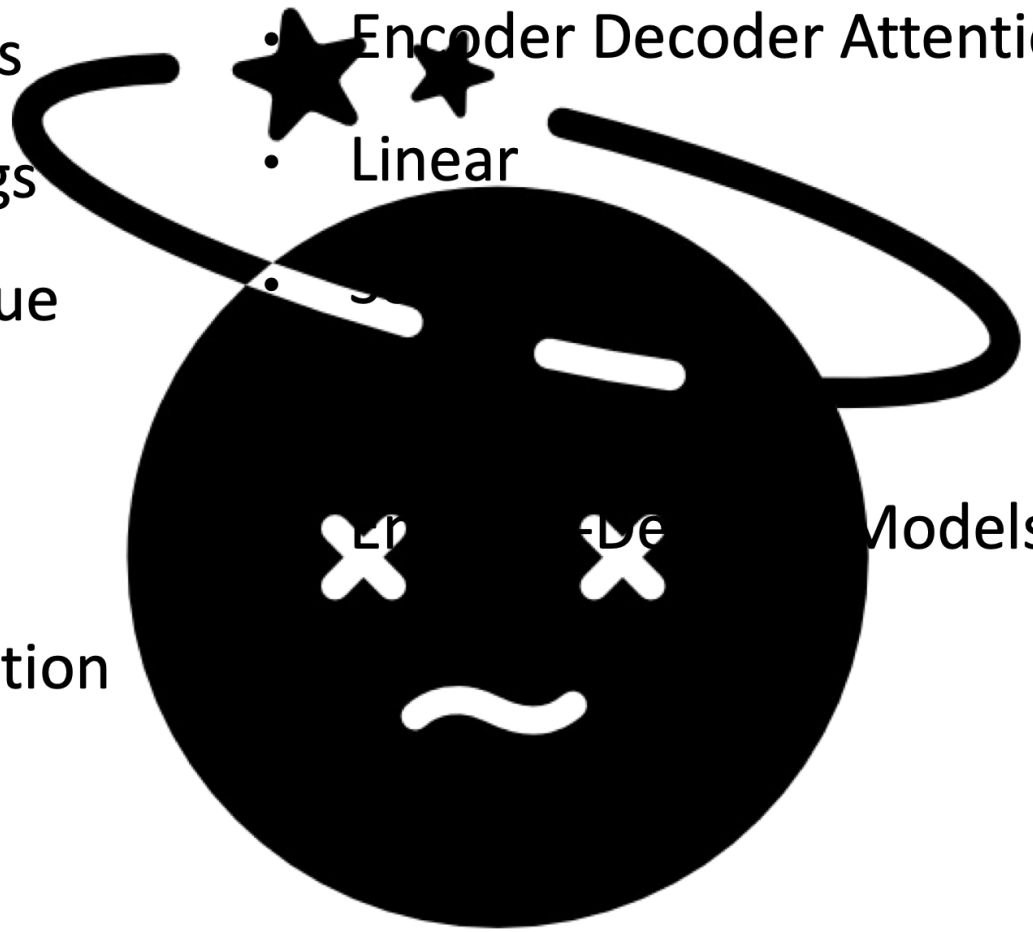
★Transformer model architecture in Vaswani, et al. (2017, NeurIPS). ”Attention is all you need”.

See more explanation



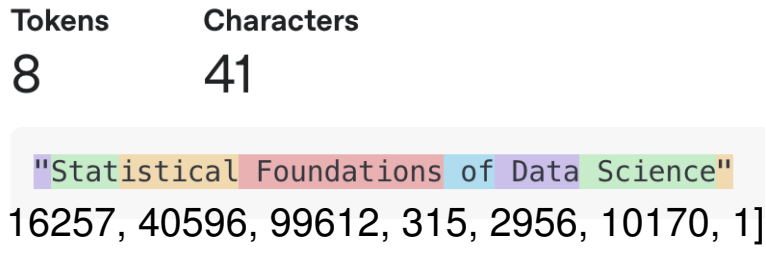
Basic structure

- Tokenization
- Input Embeddings
- Position Encodings
- Query, Key, & Value
- Attention
- Self Attention
- Multi-Head Attention
- Feed Forward
- Add & Norm
- Encoders
- Masked Attention
- Encoder Decoder Attention
- Linear
- S
- Models



Tokeninzation

- ★ A part of words: Use all basic symbols and successively add the most frequently pattern as new a token.
- ★ For GPT5, $\approx$ 100K-200 tokens, 4 characters or 0.75 words per token, one page (double space) = 500 tokens. See https://platform.openai.com/tokenizer.



- ★ There are limits on input tokens and output tokens. e.g. GPT5 has input / output limit 400K / 128K tokens.
- ★ Extended to images via patches

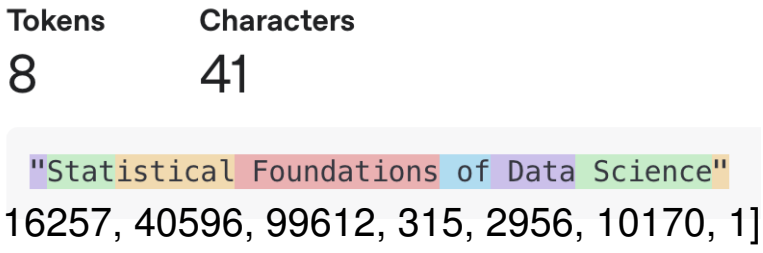
Taken from Richard Turner (2023)

$$\boldsymbol{x}_n^{(0)} = W \text{ vec}\left(\text{patch}\right)$$

n^{th} patch

Tokeninzation

- A part of words: Use all basic symbols and successively add the most frequently pattern as new a token.
- For GPT5, $\approx$ 100K-200 tokens, 4 characters or 0.75 words per token, one page (double space) = 500 tokens. See https://platform.openai.com/tokenizer.



- There are limits on input tokens and output tokens. e.g. GPT5 has input / output limit 400K / 128K tokens.
- Extended to images via patches

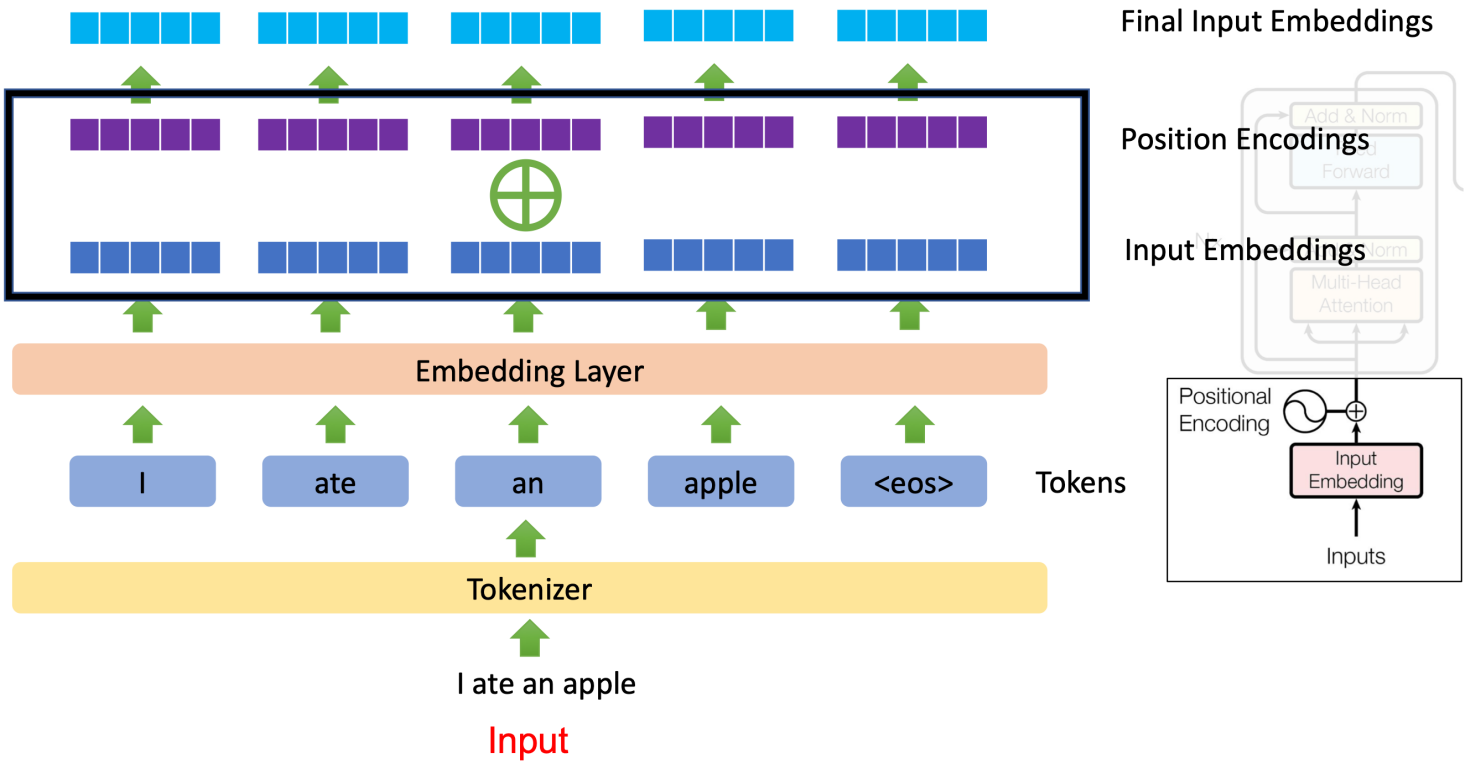
Taken from Richard Turner (2023)

$$\boldsymbol{x}_n^{(0)} = W \text{ vec}\left(\text{patch}\right)$$

n^{th} patch

Embedding

Token embedding: map a token to a vector: $\mathbf{A}\,Token \to \mathbb{R}^d$, **A** trainable (GPT3: 12,228, GPT2: 768).



Position Encoding: "I ate an apple ." $\neq$ "ate apple I an ."

$f(t) \in R^d$: $f(t) = \left(\exp(it/r^k)\right)_{k=0}^{d/2-1}$, $r = N^{2/d}$, $N =$ length of input

Transformer Input: Input Embedding Matrix (light blue rows)

An Illustrative Example

See more explanation

Outcome:

visualize	create	see	make	easily	quickly	explore	find
.5464	.2087	.1209	.0626	.0611	0	0	0

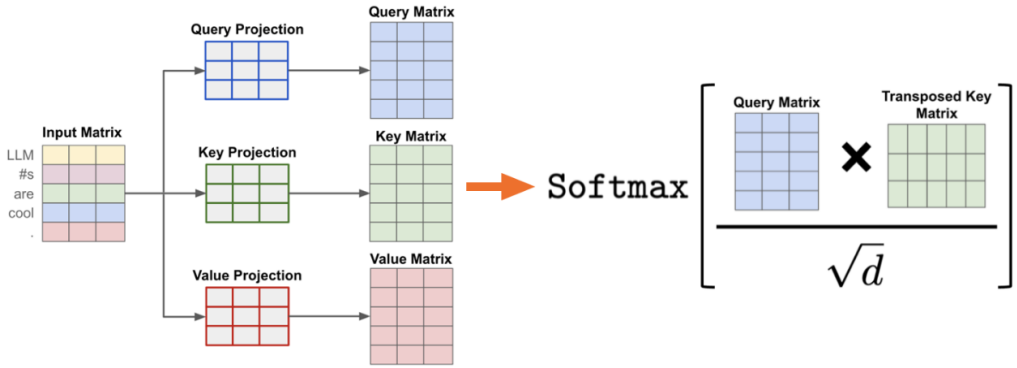
Transformer block – Self Attention

Matrix Input: $\mathbf{Z} \in \mathbb{R}^{N \times d}$, N tokens with feature dimension d

- self-attention w/ residual skip connection (asymmetric similarity, "hammer" is strongly associated with "tools", not vice versus) $\mathbf{H} = \mathbf{Q}\mathbf{K}^T$ can be **low-rank** factorization.

$$\mathrm{Attn}(\mathbf{Z}) = \mathbf{Z} + \underbrace{\mathbf{ZV}}_{\text{Value}} \mathrm{softmax}\left[(\underbrace{\mathbf{ZQ}}_{\text{Query}})(\underbrace{\mathbf{ZK}}_{\text{Key}})^\top \right]^\top$$

— $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d \times d} \Longrightarrow \mathrm{Attn}(\mathbf{Z}_i) = \mathbf{Z}_i + \mathrm{softmax}\left[(\mathbf{ZQ})(\mathbf{ZK})^\top \right] \mathbf{V}^\top \mathbf{Z}_i$ $\mathbf{Z}_i^T = i^{th}$ row



- layer normalization standardizes each column/token (optional):

$$\mathrm{LayerNorm}(x) = \alpha \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

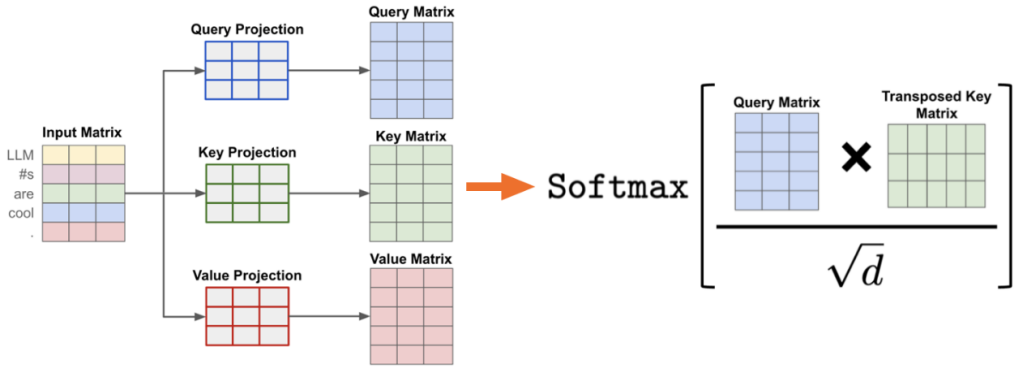
Transformer block – Self Attention

Matrix Input: $\mathbf{Z} \in \mathbb{R}^{N \times d}$, N tokens with feature dimension d

- self-attention w/ residual skip connection (asymmetric similarity, "hammer" is strongly associated with "tools", not vice versus) $\mathbf{H} = \mathbf{Q}\mathbf{K}^T$ can be **low-rank** factorization.

$$\mathrm{Attn}(\mathbf{Z}) = \mathbf{Z} + \underbrace{\mathbf{ZV}}_{\text{Value}} \mathrm{softmax}\left[(\underbrace{\mathbf{ZQ}}_{\text{Query}})(\underbrace{\mathbf{ZK}}_{\text{Key}})^\top \right]^\top$$

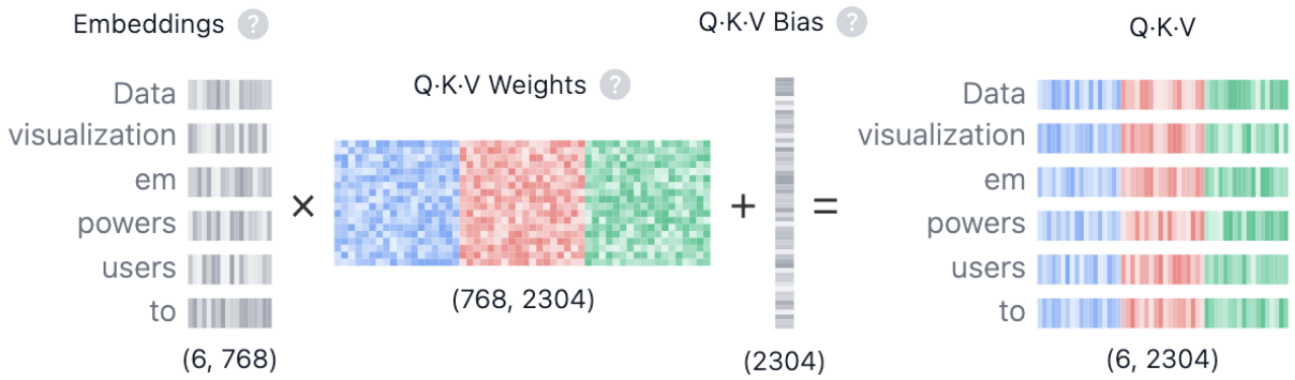
—$\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d \times d} \Longrightarrow \mathrm{Attn}(\mathbf{Z}_i) = \mathbf{Z}_i + \mathrm{softmax}\left[(\mathbf{ZQ})(\mathbf{ZK})^\top \right] \mathbf{V}^\top \mathbf{Z}_i$ $\mathbf{Z}_i^T = i^{th}$ row



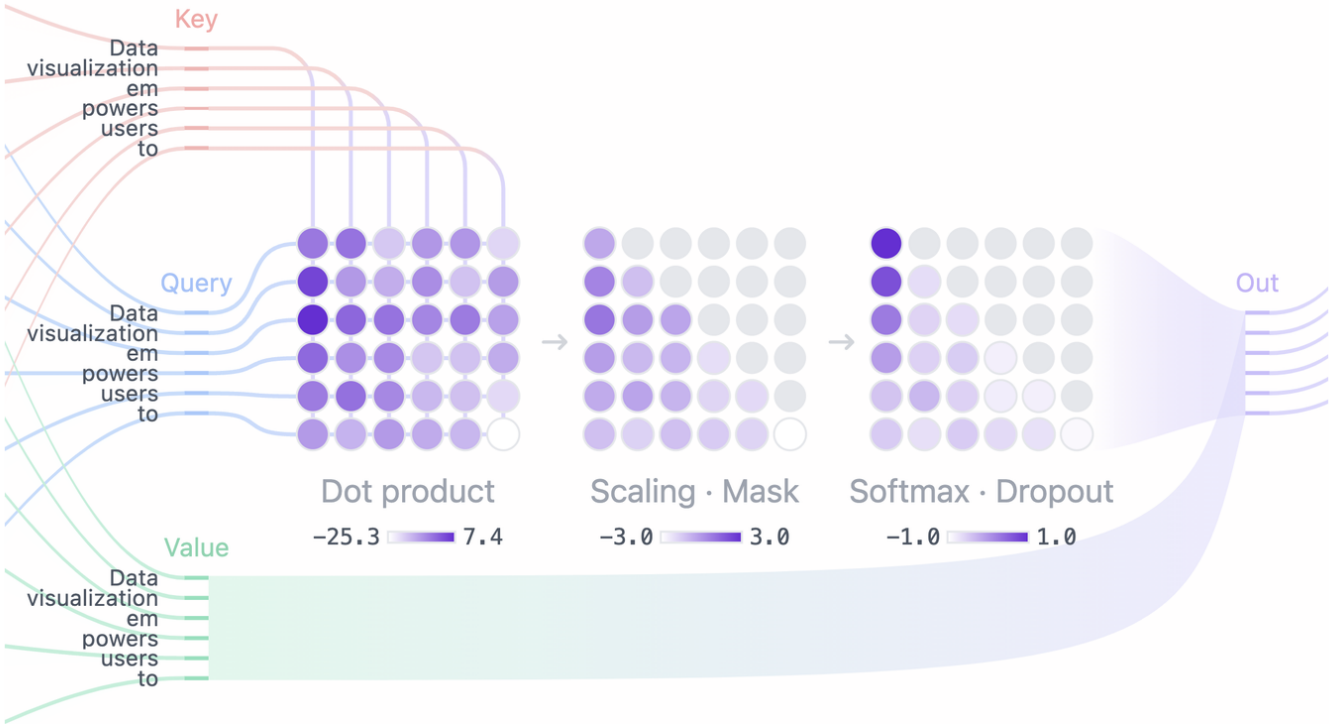
- layer normalization standardizes each column/token (optional):

$$\mathrm{LayerNorm}(x) = \alpha \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

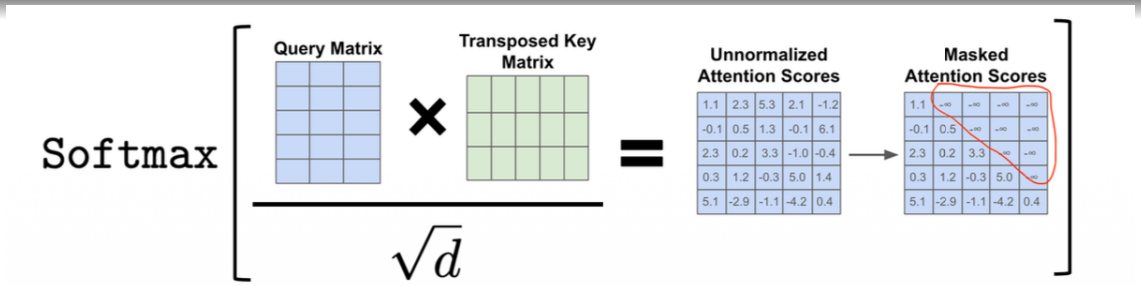
An Illustrative Example



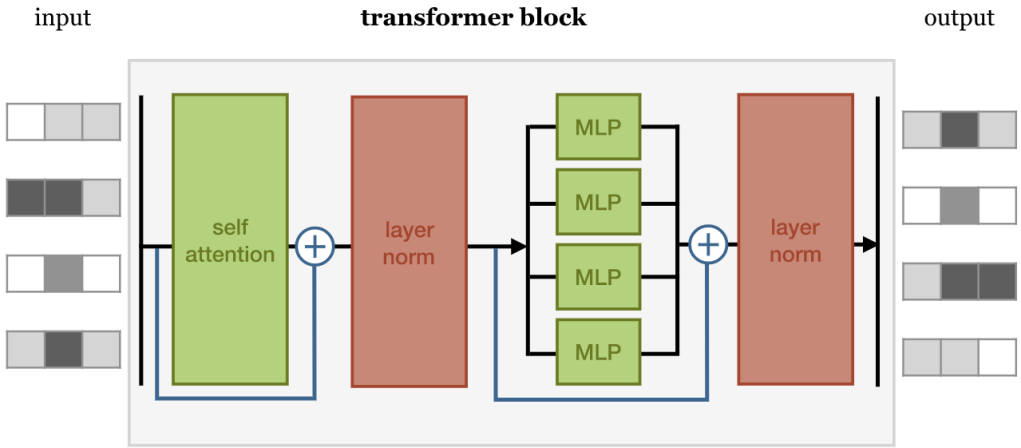
$$QKV_{ij} = (\sum_{d=1}^{768} \text{Embedding}_{i,d} \cdot \text{Weights}_{d,j}) + \text{Bias}_j$$



Transformer block – one-head



★ masked self attention: Not use future tokens– set upper triangle to zero ($-\infty$).



★ token-wise standard feed-forward neural network layer (MLP)

$$\mathrm{MLP}(\mathbf{Z}) = \mathbf{Z} + \mathrm{ReLU}(\mathbf{Z}\mathbf{W}_1)\mathbf{W}_2$$

$$\Longrightarrow \mathrm{MLP}(\mathbf{Z}_i) = \mathbf{Z}_i + \mathbf{W}_2^T \mathrm{ReLU}(\mathbf{W}_1^T \mathbf{Z}_i)$$

—$\mathbf{Z}_i^T = i^{th}$ row.

Multi-head self-attention transformers

Multi-head attention: With $\theta_1 = \{(\boldsymbol{V}_m, \boldsymbol{Q}_m, \boldsymbol{K}_m)\}_{m\in[M]} \subset \mathbb{R}^{d\times d}$, for input $\boldsymbol{Z} \in \mathbb{R}^{N\times d}$,

$$Attn_{\theta_1}(\boldsymbol{Z}) = \boldsymbol{Z} + \sum_{m=1}^{M} (\boldsymbol{Z}\boldsymbol{V}_m)\sigma\left((\boldsymbol{Z}\boldsymbol{Q}_m)(\boldsymbol{Z}\boldsymbol{K}_m)^\top\right)^\top,$$

—GPT1 uses 12, GPT 3 uses 96 heads

ResNet: Parameters $\theta_2 \in (\boldsymbol{W}_1, \boldsymbol{W}_2^\top) \in \mathbb{R}^{d\times d'}$, for input $\boldsymbol{Z} \in \mathbb{R}^{N\times d}$,

$$FC_{\theta_2}(\boldsymbol{Z}) := \boldsymbol{Z} + \sigma(\boldsymbol{Z}\boldsymbol{W}_1)\boldsymbol{W}_2.$$

Transformer: L-layer, output dim d,

$$TF_\theta(\boldsymbol{Z}) := \widetilde{\boldsymbol{W}}_0 \times FC_{\theta_2^L}(Attn_{\theta_1^L}(\cdots FC_{\theta_2^1}(Attn_{\theta_1^1}(\boldsymbol{Z}))) \times \widetilde{\boldsymbol{W}}_1,$$

where $\widetilde{\boldsymbol{W}}_0 \in \mathbb{R}^{d_1\times d}$ and $\widetilde{\boldsymbol{W}}_1 \in \mathbb{R}^{d\times d_2}$.

Multi-head self-attention transformers

Multi-head attention: With $\theta_1 = \{(\boldsymbol{V}_m, \boldsymbol{Q}_m, \boldsymbol{K}_m)\}_{m\in[M]} \subset \mathbb{R}^{d\times d}$, for input $\boldsymbol{Z} \in \mathbb{R}^{N\times d}$,

$$Attn_{\theta_1}(\boldsymbol{Z}) = \boldsymbol{Z} + \sum_{m=1}^{M} (\boldsymbol{Z}\boldsymbol{V}_m)\sigma\left((\boldsymbol{Z}\boldsymbol{Q}_m)(\boldsymbol{Z}\boldsymbol{K}_m)^\top\right)^\top,$$

—GPT1 uses 12, GPT 3 uses 96 heads

ResNet: Parameters $\theta_2 \in (\boldsymbol{W}_1, \boldsymbol{W}_2^\top) \in \mathbb{R}^{d\times d'}$, for input $\boldsymbol{Z} \in \mathbb{R}^{N\times d}$,

$$FC_{\theta_2}(\boldsymbol{Z}) := \boldsymbol{Z} + \sigma(\boldsymbol{Z}\boldsymbol{W}_1)\boldsymbol{W}_2.$$

Transformer: L-layer, output dim d,

$$TF_\theta(\boldsymbol{Z}) := \widetilde{\boldsymbol{W}}_0 \times FC_{\theta_2^L}(Attn_{\theta_1^L}(\cdots FC_{\theta_2^1}(Attn_{\theta_1^1}(\boldsymbol{Z}))) \times \widetilde{\boldsymbol{W}}_1,$$

where $\widetilde{\boldsymbol{W}}_0 \in \mathbb{R}^{d_1\times d}$ and $\widetilde{\boldsymbol{W}}_1 \in \mathbb{R}^{d\times d_2}$.

Multi-head self-attention transformers

Multi-head attention: With $\theta_1 = \{(\boldsymbol{V}_m, \boldsymbol{Q}_m, \boldsymbol{K}_m)\}_{m\in[M]} \subset \mathbb{R}^{d\times d}$, for input $\boldsymbol{Z} \in \mathbb{R}^{N\times d}$,

$$Attn_{\theta_1}(\boldsymbol{Z}) = \boldsymbol{Z} + \sum_{m=1}^{M} (\boldsymbol{Z}\boldsymbol{V}_m)\sigma\left((\boldsymbol{Z}\boldsymbol{Q}_m)(\boldsymbol{Z}\boldsymbol{K}_m)^\top\right)^\top,$$

—GPT1 uses 12, GPT 3 uses 96 heads

ResNet: Parameters $\theta_2 \in (\boldsymbol{W}_1, \boldsymbol{W}_2^\top) \in \mathbb{R}^{d\times d'}$, for input $\boldsymbol{Z} \in \mathbb{R}^{N\times d}$,

$$FC_{\theta_2}(\boldsymbol{Z}) := \boldsymbol{Z} + \sigma(\boldsymbol{Z}\boldsymbol{W}_1)\boldsymbol{W}_2.$$

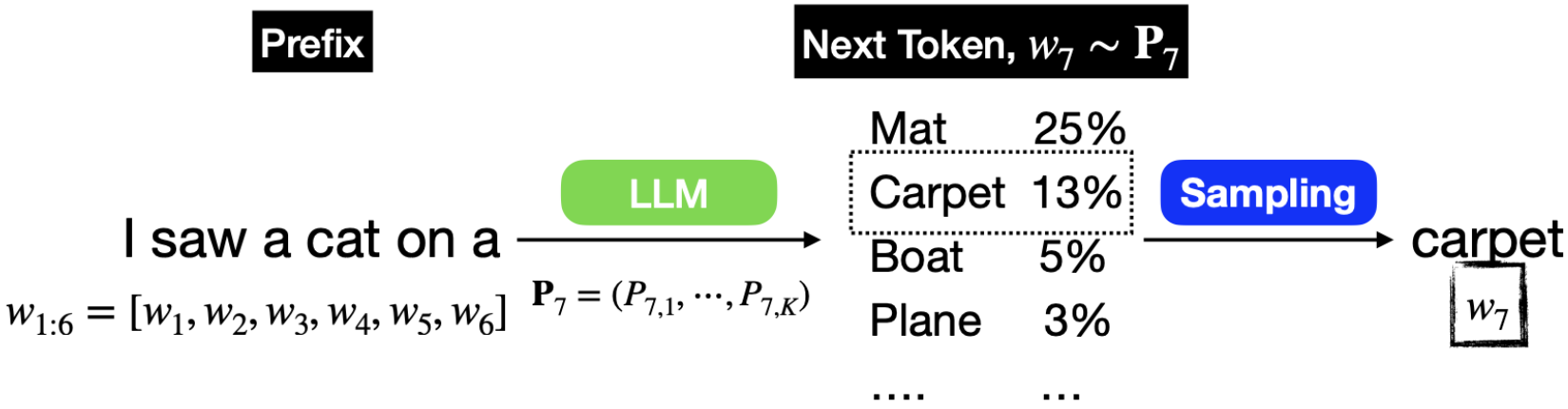
Transformer: L-layer, output dim d,

$$TF_\theta(\boldsymbol{Z}) := \widetilde{\boldsymbol{W}}_0 \times FC_{\theta_2^L}(Attn_{\theta_1^L}(\cdots FC_{\theta_2^1}(Attn_{\theta_1^1}(\boldsymbol{Z}))) \times \widetilde{\boldsymbol{W}}_1,$$

where $\widetilde{\boldsymbol{W}}_0 \in \mathbb{R}^{d_1\times d}$ and $\widetilde{\boldsymbol{W}}_1 \in \mathbb{R}^{d\times d_2}$.

Autoregressive language models

Model: $p(w_1 \cdots w_n) = p(w_1) \prod_{t=2}^{n} P(w_t | w_{1:t-1})$, by a transformer (e.g. last 5)

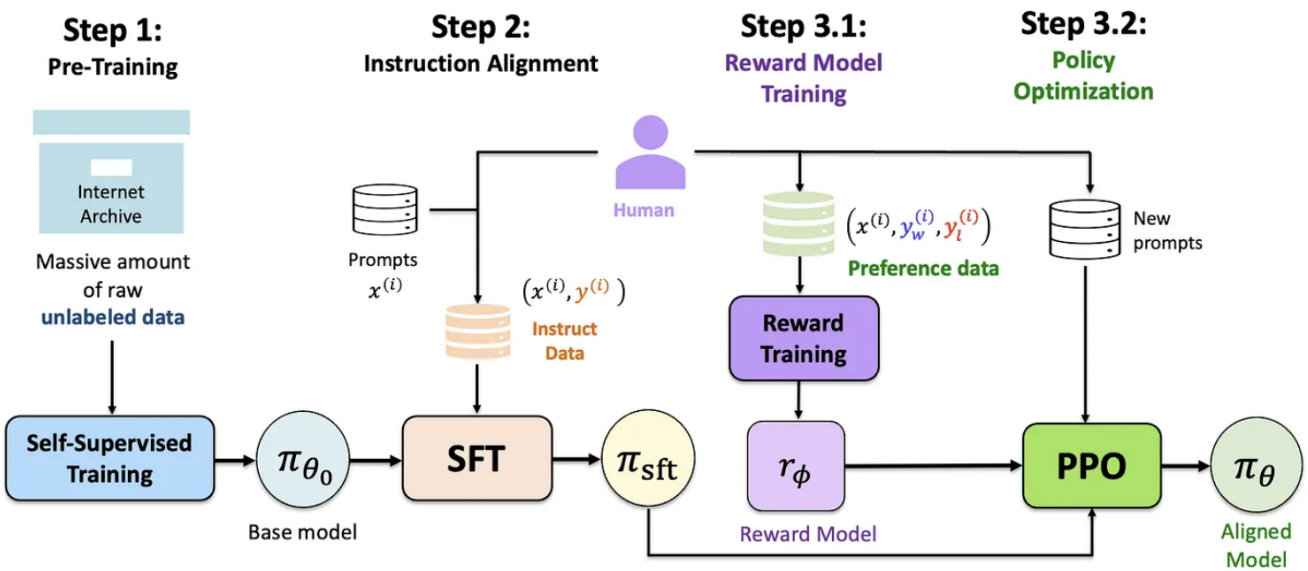


Training loss: $-\sum_{i \in \text{doc}} \sum_{t=2}^{\text{length}_i} \log P_\theta(w_{i,t} | w_{i,1:t-1})$

Predictive: To be predictive, attention matrices should be upper trangular (masked self-attention).

Fine-Tuning

Fine-Tuning in LLM Training. **Base model** (expensive) → **SFT** → **PPO**



1.4 Watermarking

★Embed signals into generated text that are invisible to humans but algorithmically detectable from a short span of tokens

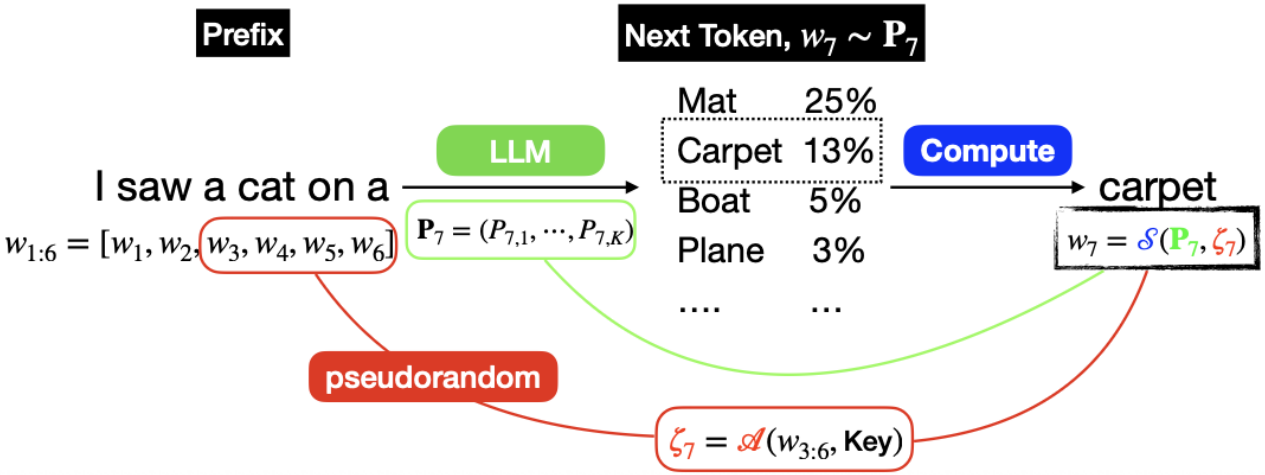
★Dathathri, et al (2024). Scalable watermarking for identifying large language model outputs. *Nature*, **634**, 818-823

Basic Idea

Psudorandomness: ζ_t and w_t are dependent

★Compute $\zeta_t = \mathcal{A}(w_{1:t-1}, \mathrm{Key})$ (**watermarks**)for $t = 1, \ldots, n$ are iid, indep of $\mathbf{w}_{1:t-1}$.

★Computationally infeasible to infer Key from ζ_t and $w_{1:t-1}$. random seed



★When human written data, ζ_t and w_t are independent; otherwise dep

Example: Two tokens (Bernoulli): $\mathcal{W} = \{0, 1\}$, $\mathbf{P}_t = (P_{0,t}, P_{1,t})$,

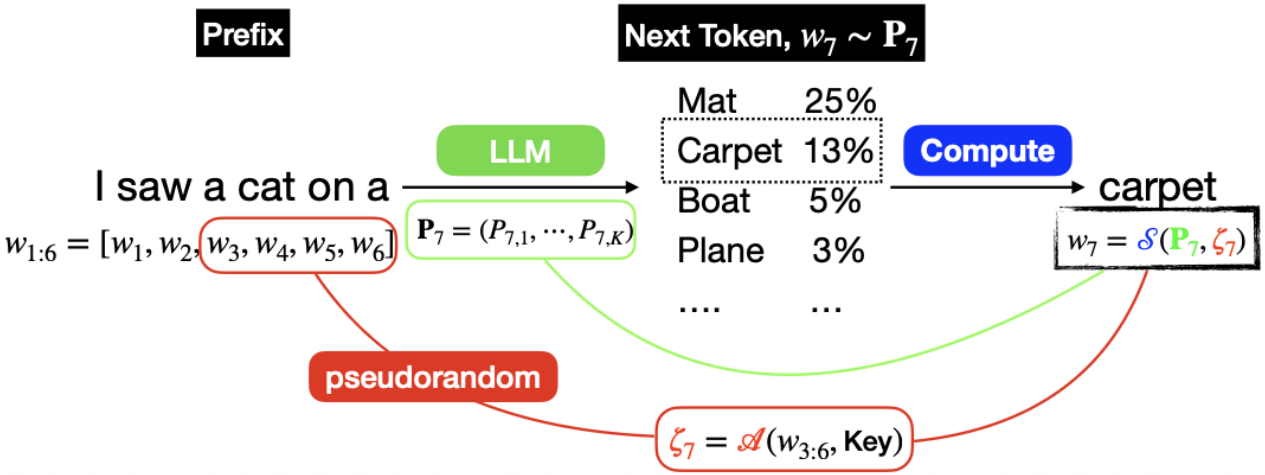
- $\zeta_t \sim_{i.i.d.}$ Unif(0, 1) and $w_t = \mathcal{S}(\mathbf{P}_t, \zeta_t) = I(\zeta_t \geq P_{0,t})$.
- Detection statistics: $\sum_{i=1}^{n}(2w_t - 1)(2\zeta_t - 1)$ (large $\Longrightarrow$ generated text)

Basic Idea

Psudorandomness: ζ_t and w_t are dependent

★Compute $\zeta_t = \mathcal{A}(w_{1:t-1}, \mathrm{Key})$ (**watermarks**)for $t = 1, \ldots, n$ are iid, indep of $\mathbf{w}_{1:t-1}$.

★Computationally infeasible to infer Key from ζ_t and $w_{1:t-1}$. random seed



★When human written data, ζ_t and w_t are independent; otherwise dep

Example: Two tokens (Bernoulli): $\mathcal{W} = \{0, 1\}$, $\mathbf{P}_t = (P_{0,t}, P_{1,t})$,

- $\zeta_t \sim_{i.i.d.}$ Unif(0, 1) and $w_t = \mathcal{S}(\mathbf{P}_t, \zeta_t) = I(\zeta_t \geq P_{0,t})$.
- Detection statistics: $\sum_{i=1}^{n}(2w_t - 1)(2\zeta_t - 1)$ (large $\Longrightarrow$ generated text)

Inverse Transform and Gumbel Watermarks

- A watermark corresponds to sampling method for a multinomial distribution

Inverst Transform Watermarking: $K = |\mathcal{W}|$ (*Kirchenbauer, et al, 23)*

Partition (0, 1) into K intenvals according to $\mathbf{P}_t$. (*Generalize Bernoulli)*

Gumbel-max Watermarking : (*Aaronson, 2023)*

$$\mathcal{S}(\mathbf{P}, \zeta) = \text{argmax}_{w \in \mathcal{W}} \frac{\log U_w}{P_w} \sim \text{Multi}(1, \mathbf{P}),$$

where $\zeta = (U_1, \cdots, U_K)$ i.i.d. uniform(0, 1).

Inverse Transform and Gumbel Watermarks

- A watermark corresponds to sampling method for a multinomial distribution

Inverst Transform Watermarking: $K = |\mathcal{W}|$ (*Kirchenbauer, et al, 23)*

Partition (0, 1) into K intenvals according to $\mathbf{P}_t$. (*Generalize Bernoulli)*

Gumbel-max Watermarking : (*Aaronson, 2023)*

$$\mathcal{S}(\mathbf{P}, \zeta) = \text{argmax}_{w \in \mathcal{W}} \frac{\log U_w}{P_w} \sim \text{Multi}(1, \mathbf{P}),$$

where $\zeta = (U_1, \cdots, U_K)$ i.i.d. uniform(0, 1).

Hypothesis Testing Formulation

H_0 : $w_{1:n}$ generated by Human vs H_0 : $w_{1:n}$ generated by LLMs

★Under H_0 :, $(w_t, \zeta_t)|(w_{1:t-1}, \zeta_{1:t-1}) \stackrel{d}{=} \mathbf{P}_t \times \zeta$ **$\mathbf{P}_t$ unknown**

★Under H_1, $(w_t, \zeta_t)|(w_{1:t-1}, \zeta_{1:t-1}) \stackrel{d}{=} (\mathcal{S}(\mathbf{P}_t, \zeta_t), \zeta_t)$

Pivotal Stat: Find $Y_t = Y(w_t, \zeta_t) \sim_{H_0} \mu_0$, regardless of $\mathbf{P}_t$. Use test stat

$$T_h = \sum_{i=1}^{n} T_h(Y_t)$$

Examples: ★Bernoulli: $Y_t = (2w_t - 1)(2\zeta_t - 1) \sim_{H_0} \text{Unif}(-1, 1)$

★Gumbel-max $Y_t^{gum} = U_{t,w_t} \sim_{H_0} \text{Unif}(0, 1)$ $\mathbb{P}_1(Y_t^{gum} \leq x|\mathbf{P}_t) = \sum_{k=1}^{K} P_{t,k} x^{1/P_{t,k}}$

Hypothesis Testing Formulation

H_0 : $w_{1:n}$ generated by Human vs H_0 : $w_{1:n}$ generated by LLMs

★Under H_0 :, $(w_t, \zeta_t)|(w_{1:t-1}, \zeta_{1:t-1}) \stackrel{d}{=} \mathbf{P}_t \times \zeta$ **$\mathbf{P}_t$ unknown**

★Under H_1, $(w_t, \zeta_t)|(w_{1:t-1}, \zeta_{1:t-1}) \stackrel{d}{=} (\mathcal{S}(\mathbf{P}_t, \zeta_t), \zeta_t)$

Pivotal Stat: Find $Y_t = Y(w_t, \zeta_t) \sim_{H_0} \mu_0$, regardless of $\mathbf{P}_t$. Use test stat

$$T_h = \sum_{i=1}^{n} T_h(Y_t)$$

Examples: ★Bernoulli: $Y_t = (2w_t - 1)(2\zeta_t - 1) \sim_{H_0}$ Unif$(-1, 1)$

★Gumbel-max $Y_t^{gum} = U_{t,w_t} \sim_{H_0}$ Unif(0, 1) $\mathbb{P}_1(Y_t^{gum} \leq x|\mathbf{P}_t) = \sum_{k=1}^{K} P_{t,k} x^{1/P_{t,k}}$

Power of Pivital Test

Fixing Type I error in $(0,1)$, the pivot-based test statistic has

$$\lim_{n\to\infty} \sup_{\mathbf{P}\in\mathcal{P}} \textit{Type II error}^{\frac{1}{n}} \leq \exp\{-R_{\mathcal{P}}(h)\},$$

$$R_{\mathcal{P}}(h) = -\inf_{\theta\geq 0}\left\{\theta\mathbb{E}_0 h(Y) + \log\phi_{\mathcal{P},h}(\theta)\right\}, \qquad \phi_{\mathcal{P},h}(\theta) = \sup_{P\in\mathcal{P}} \mathbb{E}_{1,P} e^{-\theta h(Y)}.$$

- $R_{\mathcal{P}}(h) = 0$ for any h if $\mathcal{P}$ includes $(0,\ldots,0,1,0,\ldots,0)$.
- For the class $\mathbb{P}_\Delta = \{\mathbf{P} : \max_j P_j \leq \Delta\}$, the rate $R_{\mathcal{P}}(h) = \Delta$.
- More explicit for Gumbel and Inverse Dist.
- Li, et al. (2025). A Statistical Framework of Watermarks for Large Language Models: Pivot, Detection Efficiency and Optimal Rules

Power of Pivital Test

Fixing Type I error in $(0,1)$, the pivot-based test statistic has

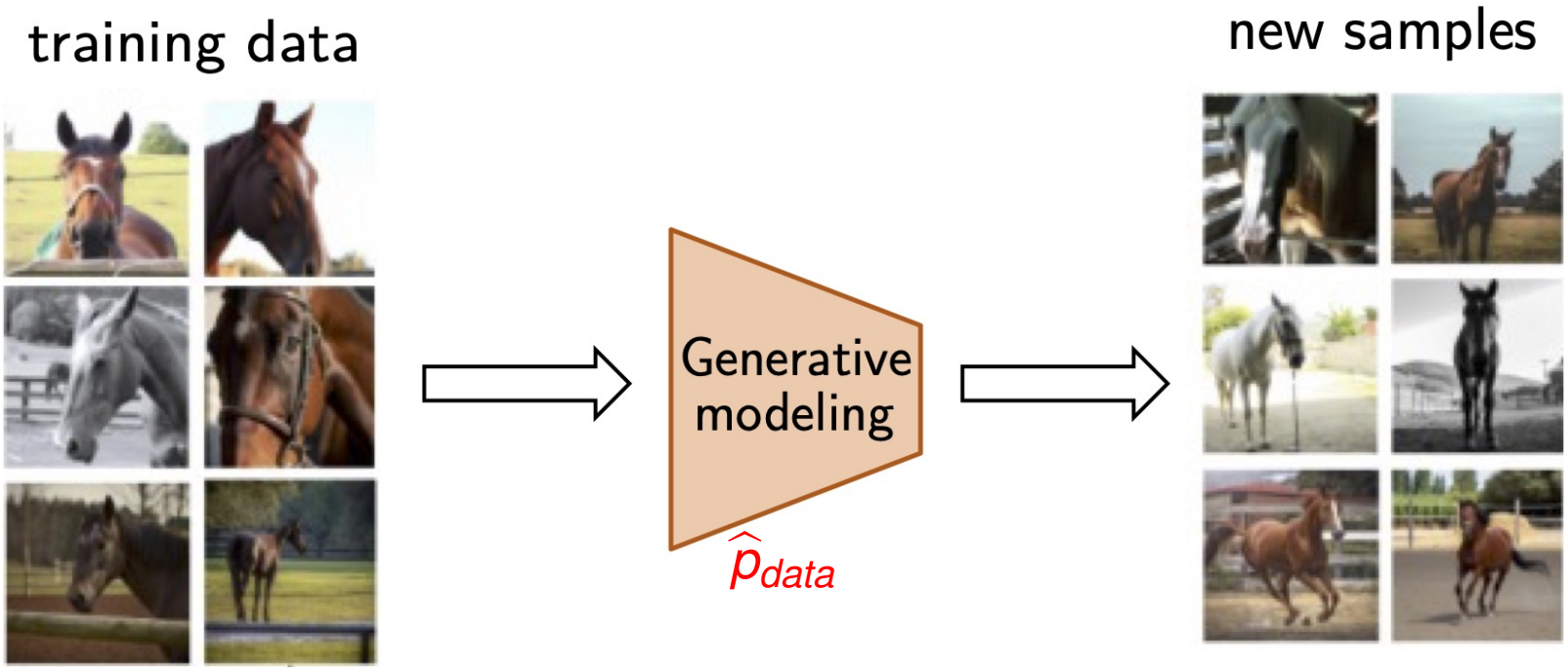
$$\lim_{n\to\infty} \sup_{\mathbf{P}\in\mathcal{P}} \textit{Type II error}^{\frac{1}{n}} \leq \exp\{-R_{\mathcal{P}}(h)\},$$

$$R_{\mathcal{P}}(h) = -\inf_{\theta\geq 0}\left\{\theta\mathbb{E}_0 h(Y) + \log\phi_{\mathcal{P},h}(\theta)\right\}, \qquad \phi_{\mathcal{P},h}(\theta) = \sup_{P\in\mathcal{P}} \mathbb{E}_{1,P} e^{-\theta h(Y)}.$$

- ★ $R_{\mathcal{P}}(h) = 0$ for any h if $\mathcal{P}$ includes $(0,\ldots,0,1,0,\ldots,0)$.
- ★ For the class $\mathbb{P}_\Delta = \{\mathbf{P} : \max_j P_j \leq \Delta\}$, the rate $R_{\mathcal{P}}(h) = \Delta$.
- ★ More explicit for Gumbel and Inverse Dist.
- ★ Li, et al. (2025). A Statistical Framework of Watermarks for Large Language Models: Pivot, Detection Efficiency and Optimal Rules

1.5 Generative Adversarial Networks

Generative modeling



- Given training data $\underbrace{X^{\text{train},i} \sim p_{\text{data}}}_{\text{from a general distribution}}$ $(1 \leq i \leq n)$ in $\mathbb{R}^d$
- Generate new samples $X \sim \widehat{\mathbf{p}}_{\text{data}}$

Sampling as a zero-sum game

Aim: Generate new data (**natural** images) like training $\{\mathbf{x}_i\}_{1\le i\le n}$.

GANs model a zero-sum game btwn generator $\mathcal{G}$ and discriminator $\mathcal{D}$

- The generator $\mathcal{G}$ (usually a CNN) tries to generate fake images akin to the true images $\{\mathbf{x}_i\}_{1\le i\le n}$
- The discriminator $\mathcal{D}$ (another CNN) aims at differentiating the fake ones from the true ones.
- Payoff of discriminator $d(\mathbf{x}) = \begin{cases} \log(2d(\mathbf{x})) & \text{if } \mathbf{x} \text{ real} \\ \log(2(1-d(\mathbf{x}))) & \text{if } \mathbf{x} \text{ fake} \end{cases}$

Goal: construct a generator that the best discriminator cann't tell

Sampling as a zero-sum game

Aim: Generate new data (**natural** images) like training $\{\mathbf{x}_i\}_{1\le i\le n}$.

GANs model a zero-sum game btwn generator $\mathcal{G}$ and discriminator $\mathcal{D}$

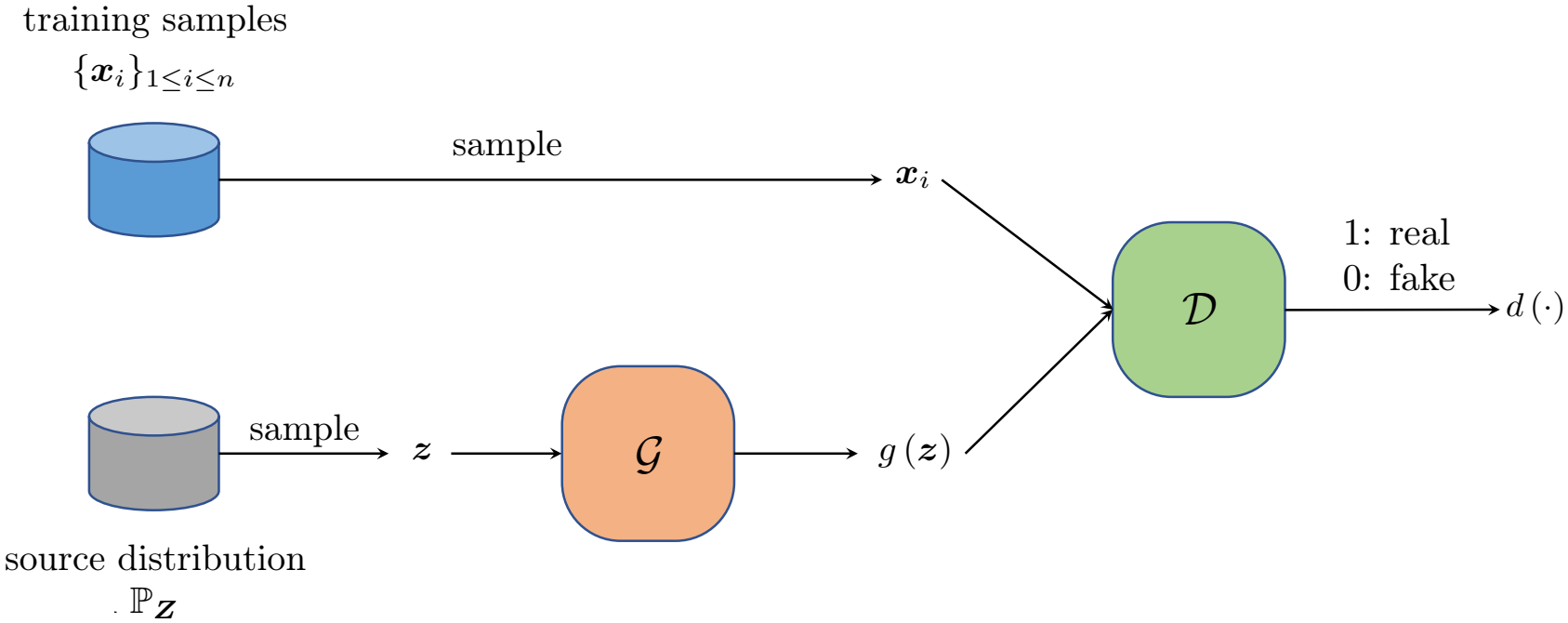
- The generator $\mathcal{G}$ (usually a CNN) tries to generate fake images akin to the true images $\{\mathbf{x}_i\}_{1\le i\le n}$
- The discriminator $\mathcal{D}$ (another CNN) aims at differentiating the fake ones from the true ones.
- Payoff of discriminator $d(\mathbf{x}) = \begin{cases} \log(2d(\mathbf{x})) & \text{if } \mathbf{x} \text{ real} \\ \log(2(1-d(\mathbf{x}))) & \text{if } \mathbf{x} \text{ fake} \end{cases}$

Goal: construct a generator that the best discriminator cann't tell

Generative Adversary Network

Aim: Generate from dist of data $\{\mathbf{x}_i\}$ *(Goodfellow et al., 14; 98K)*

Fact: Given r.v. **Z**, there exists a **g** such that **Dist(X) = Dist (g(Z))**.



$$\min_{\theta_{\mathcal{G}}} \max_{\theta_{\mathcal{D}}} \quad \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{X}}}\left[\log\left(d\left(\mathbf{x}\right)\right)\right] + \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_{\mathbf{Z}}}\left[\log\left(1-d\left(\mathbf{g}\left(\mathbf{z}\right)\right)\right)\right].$$

NN NN

★High-dim functions $d(\cdot)$ and $\mathbf{g}(\cdot)$ are approximated by NN.

★$\dim(\mathbf{Z}) \approx 100$ rather than $\dim(\mathbf{x}_i)$ (**dim reduction**). **Z ∼ Gaussian**

Remarks

1. The empirical minimax problem is to solve

$$\min_{\theta_{\mathcal{G}}} \max_{\theta_{\mathcal{D}}} \quad \frac{1}{n}\sum_{i=1}^{n} \log\left(d_{\theta_{\mathcal{D}}}(\mathbf{x}_i)\right) + \frac{1}{n}\sum_{i=1}^{n} \log\left(1 - d_{\theta_{\mathcal{D}}}\left(\mathbf{g}_{\theta_{\mathcal{G}}}(\mathbf{z}_i)\right)\right).$$

2. For unrestrict $d(\cdot)$, inner max obtained at $d^*(\mathbf{x}) = \frac{\mathbb{P}_{\mathbf{X}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}$. Problem reduces to min Jensen-Shannon divergence:

$$\min_{\theta_{\mathcal{G}}} \left[\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{X}}} \log \frac{\mathbb{P}_{\mathbf{X}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})} + \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathcal{G}}} \log \frac{\mathbb{P}_{\mathcal{G}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}\right],$$

★$\mathbb{P}_{\mathcal{G}}$ = density of $\mathbf{g}_{\theta_{\mathcal{G}}}(\mathbf{Z})$. ★unconstrained min over $\mathbb{P}_{\mathcal{G}} = \mathbb{P}_{\mathbf{X}}$

3. This generalizes to other divergences. Wasserstein GAN

$$\min_{\theta_{\mathcal{G}}} \underbrace{\sup_{f: f \text{ 1-Lipschitz}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{X}}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathcal{G}}}[f(\mathbf{x})]}_{\text{dist}(\mathbb{P}_{\mathbf{X}}, \mathbb{P}_{\mathcal{G}})} \qquad \text{min dist}$$

Remarks

1. The empirical minimax problem is to solve

$$\min_{\theta_{\mathcal{G}}} \max_{\theta_{\mathcal{D}}} \quad \frac{1}{n}\sum_{i=1}^{n} \log\left(d_{\theta_{\mathcal{D}}}(\mathbf{x}_i)\right) + \frac{1}{n}\sum_{i=1}^{n} \log\left(1 - d_{\theta_{\mathcal{D}}}\left(\mathbf{g}_{\theta_{\mathcal{G}}}(\mathbf{z}_i)\right)\right).$$

2. For unrestrict $d(\cdot)$, inner max obtained at $d^*(\mathbf{x}) = \frac{\mathbb{P}_{\mathbf{X}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}$. Problem reduces to min Jensen-Shannon divergence:

$$\min_{\theta_{\mathcal{G}}} \left[\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{X}}} \log \frac{\mathbb{P}_{\mathbf{X}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})} + \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathcal{G}}} \log \frac{\mathbb{P}_{\mathcal{G}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}\right],$$

★$\mathbb{P}_{\mathcal{G}}$ = density of $\mathbf{g}_{\theta_{\mathcal{G}}}(\mathbf{Z})$. ★unconstrained min over $\mathbb{P}_{\mathcal{G}} = \mathbb{P}_{\mathbf{X}}$

3. This generalizes to other divergences. Wasserstein GAN

$$\min_{\theta_{\mathcal{G}}} \underbrace{\sup_{f: f \text{ 1-Lipschitz}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{X}}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathcal{G}}}[f(\mathbf{x})]}_{\text{dist}(\mathbb{P}_{\mathbf{X}}, \mathbb{P}_{\mathcal{G}})} \qquad \text{min dist}$$

Remarks

1. The empirical minimax problem is to solve

$$\min_{\theta_{\mathcal{G}}} \max_{\theta_{\mathcal{D}}} \quad \frac{1}{n}\sum_{i=1}^{n} \log\left(d_{\theta_{\mathcal{D}}}(\mathbf{x}_i)\right) + \frac{1}{n}\sum_{i=1}^{n} \log\left(1 - d_{\theta_{\mathcal{D}}}\left(\mathbf{g}_{\theta_{\mathcal{G}}}(\mathbf{z}_i)\right)\right).$$

2. For unrestrict $d(\cdot)$, inner max obtained at $d^*(\mathbf{x}) = \frac{\mathbb{P}_{\mathbf{X}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}$. Problem reduces to min Jensen-Shannon divergence:

$$\min_{\theta_{\mathcal{G}}} \left[\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{X}}} \log \frac{\mathbb{P}_{\mathbf{X}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})} + \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathcal{G}}} \log \frac{\mathbb{P}_{\mathcal{G}}(\mathbf{x})}{\mathbb{P}_{\mathbf{X}}(\mathbf{x}) + \mathbb{P}_{\mathcal{G}}(\mathbf{x})}\right],$$

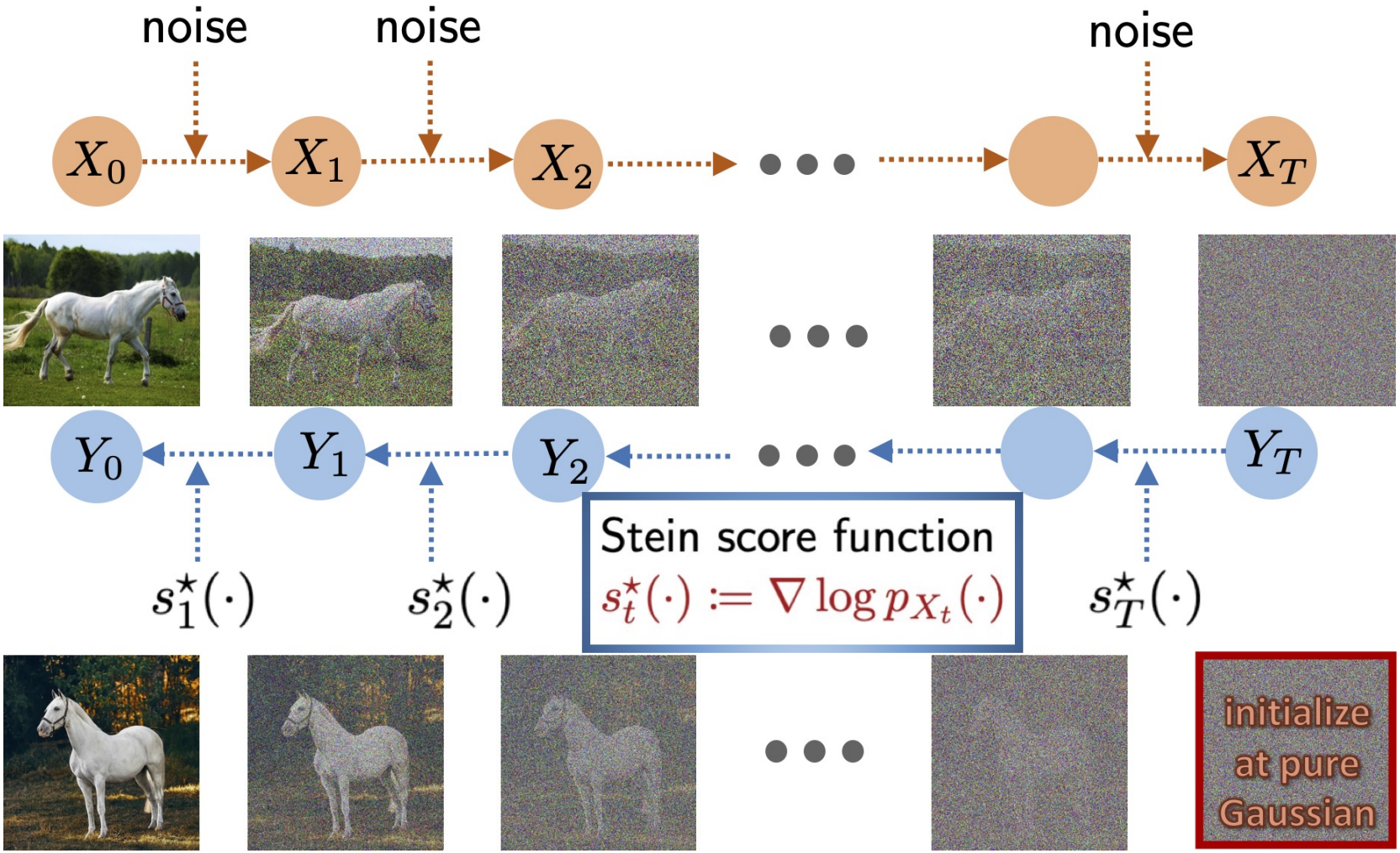
★$\mathbb{P}_{\mathcal{G}}$ = density of $\mathbf{g}_{\theta_{\mathcal{G}}}(\mathbf{Z})$. ★unconstrained min over $\mathbb{P}_{\mathcal{G}} = \mathbb{P}_{\mathbf{X}}$

3. This generalizes to other divergences. Wasserstein GAN

$$\min_{\theta_{\mathcal{G}}} \underbrace{\sup_{f: f \text{ 1-Lipschitz}} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathbf{X}}}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\mathcal{G}}}[f(\mathbf{x})]}_{\text{dist}(\mathbb{P}_{\mathbf{X}}, \mathbb{P}_{\mathcal{G}})} \qquad \textbf{min dist}$$

1.6 Diffusion Models

New Image Genereations



- **forward process:** diffuse data into noise
- **reverse process:** convert pure noise into data-like distributions

Basic Idea: Time-reversal SDE

★**Forward process:** Ornstein–Uhlenbeck process B_t = Brownian motion

$$dX_t = -\beta_t X_t + \sqrt{2\beta_t} dB_t \quad \text{with} \quad X_0 \sim \mathbf{p_0}, \quad \textbf{target on } \mathbb{R}^d$$

♦warping function β_t. Adopt $\beta_t \equiv 1$ w.l.o.g.

★**Backward process:** $\tilde{X}_t = X_{T-t}$ satisfies (Anderson, 82; Haussmann and Pardoux 86)

$$d\tilde{X}_t = -\beta_{T-t}\left(\tilde{X}_t + 2\nabla_{\mathbf{x}} \log \mathbf{p_{T-t}}\left(\tilde{\mathbf{X}}_\mathbf{t}\right)\right) dt + \sqrt{2\beta_{T-t}} dW_t \quad \text{with} \quad \tilde{X}_0 \sim p_T$$

♦Score function $\underbrace{\mathbf{s}^\star(\mathbf{t},\mathbf{x}) = \nabla_{\mathbf{x}} \log \mathbf{p_t}(\mathbf{x})}_{\text{learning object}}$ where $X_t \sim p_t$

♦$p_T \xrightarrow{d} N(0, I_d)$ as $T \to \infty$.

Basic Idea: Time-reversal SDE

★**Forward process:** Ornstein–Uhlenbeck process $B_t =$ Brownian motion

$$dX_t = -\beta_t X_t + \sqrt{2\beta_t} dB_t \quad \text{with} \quad X_0 \sim \mathbf{p_0}, \quad \textbf{target on } \mathbb{R}^d$$

♦warping function β_t. Adopt $\beta_t \equiv 1$ w.l.o.g.

★**Backward process:** $\tilde{X}_t = X_{T-t}$ satisfies (Anderson, 82; Haussmann and Pardoux 86)

$$\boxed{d\tilde{X}_t = -\beta_{T-t}\left(\tilde{X}_t + 2\nabla_{\mathbf{x}} \log \mathbf{p_{T-t}}\left(\tilde{\mathbf{X}}_\mathbf{t}\right)\right) dt + \sqrt{2\beta_{T-t}} dW_t} \quad \text{with} \quad \tilde{X}_0 \sim p_T$$

♦Score function $\underbrace{\mathbf{s}^\star(\mathbf{t},\mathbf{x}) = \nabla_{\mathbf{x}} \log \mathbf{p_t}(\mathbf{x})}_{\text{learning object}}$ where $X_t \sim p_t$

♦$p_T \xrightarrow{d} N(0, I_d)$ as $T \to \infty$.

An illustration

★ImageNet Data with the same initialization

Score Matching

Key identity (Vincent, 2011): For any $s(t, \mathbf{x})$,

$$\mathbb{E}_{\mathbf{X}_t}\left[\|s(t,\mathbf{X}_t) - \nabla_x \log p_t(\mathbf{X}_t)\|_2^2\right] = \mathbb{E}_{\mathbf{X}_0,\mathbf{X}_t}\left[\left\|s(t,\mathbf{X}_t) - \nabla_x \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0)\right\|_2^2\right] + C_{p_0}$$

$$\begin{aligned}\text{cross-term} &= \mathbb{E}_{\mathbf{x}} s(\mathbf{x}) \nabla_x \log p(\mathbf{x}) = \int s(\mathbf{x}) \nabla_x p(\mathbf{x}) d\mathbf{x} \\ &= \int\int s(\mathbf{x}) \nabla_x p(\mathbf{x}|\mathbf{x}_0) p_0(\mathbf{x}_0) d\mathbf{x} d\mathbf{x}_0 = \mathbb{E}_{\mathbf{x}_0,\mathbf{x}} s(\mathbf{x}) \nabla_x \log p(\mathbf{x}|\mathbf{x}_0)\end{aligned}$$

Estimating score function $s^\star(t,\cdot)$: Given $\{\mathbf{X}_{0,i}\}_{i=1}^n \overset{i.i.d.}{\sim} p_0$,

$$\widehat{s}(t,\cdot) \in \operatorname{argmin}_{s(t,\cdot)\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n \mathbb{E}_{\mathbf{X}_t\sim p_{t|0}(\cdot|\mathbf{X}_{0,i})}\left[\left\|\mathbf{s}(\mathbf{t},\mathbf{X}_t) - \frac{e^{-t}\mathbf{X}_{0,i} - \mathbf{X}_t}{1-e^{-2t}}\right\|_2^2\right]$$

where $(\mathbf{X}_t|\mathbf{X}_{0,i}) \sim N(e^{-t}\mathbf{X}_{0,i}, 1-e^{-2t})$.

Score Matching

Key identity (Vincent, 2011): For any $s(t, \mathbf{x})$,

$$\mathbb{E}_{\mathbf{X}_t}\left[\|s(t,\mathbf{X}_t) - \nabla_x \log p_t(\mathbf{X}_t)\|_2^2\right] = \mathbb{E}_{\mathbf{X}_0,\mathbf{X}_t}\left[\left\|s(t,\mathbf{X}_t) - \nabla_x \log p_{t|0}(\mathbf{X}_t|\mathbf{X}_0)\right\|_2^2\right] + C_{p_0}$$

$$\begin{aligned}\text{cross-term} &= \mathbb{E}_{\mathbf{x}} s(\mathbf{x}) \nabla_x \log p(\mathbf{x}) = \int s(\mathbf{x}) \nabla_x p(\mathbf{x}) d\mathbf{x} \\ &= \int\int s(\mathbf{x}) \nabla_x p(\mathbf{x}|\mathbf{x}_0) p_0(\mathbf{x}_0) d\mathbf{x} d\mathbf{x}_0 = \mathbb{E}_{\mathbf{x}_0,\mathbf{x}} s(\mathbf{x}) \nabla_x \log p(\mathbf{x}|\mathbf{x}_0)\end{aligned}$$

Estimating score function $s^\star(t,\cdot)$: Given $\{\mathbf{X_{0,i}}\}_{i=1}^n \overset{i.i.d.}{\sim} p_0$,

$$\widehat{s}(t,\cdot) \in \operatorname{argmin}_{s(t,\cdot)\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^n \mathbb{E}_{\mathbf{X}_t \sim p_{t|0}(\cdot|\mathbf{X_{0,i}})}\left[\left\|\mathbf{s(t, X_t)} - \frac{e^{-t}\mathbf{X_{0,i}} - \mathbf{X}_t}{1-e^{-2t}}\right\|_2^2\right]$$

where $(\mathbf{X}_t|\mathbf{X}_{0,i}) \sim N(e^{-t}\mathbf{X}_{0,i}, 1-e^{-2t})$.

1.7 Flow-matching

Flow Matching Setup

- Fix a base law P_0 that is easy to sample and a target law $P_1 = P_{\text{data}}$.
- Choose a probability path $\{P_t\}_{t\in[0,1]}$ connecting P_0 and P_1.
- Learn a velocity field $\mathbf{u} : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ so the ODE

$$\frac{d}{dt}\psi(\mathbf{x},t) = \mathbf{u}(\psi(\mathbf{x},t),t), \qquad \psi(\mathbf{x},0) = \mathbf{x}$$

  generates marginals P_t via $\mathbf{X}_t = \psi(\mathbf{X}_0, t)$ and **synthetic data** $\mathbf{X}_1 = \psi(\mathbf{X}_0, 1)$.
- ODE flow implies $\partial_t p_t(\mathbf{x}) + \nabla \cdot \big(p_t(\mathbf{x})\mathbf{u}(\mathbf{x},t)\big) = 0$, where $\nabla \cdot F = \sum_{j=1}^d \frac{\partial F_j}{\partial \mathbf{x}_j}$.
- **Affine Gaussian path**: $\mathbf{X}_t = \alpha_t \mathbf{X}_1 + \beta_t \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I)$
  with $(\alpha_0, \beta_0) = (0,1)$ and $(\alpha_1, \beta_1) = (1,0)$, e.g. $\alpha_t = t^2, \beta_t = \sqrt{1-t^2}$

Flow Matching Setup

- Fix a base law P_0 that is easy to sample and a target law $P_1 = P_{\text{data}}$.
- Choose a probability path $\{P_t\}_{t\in[0,1]}$ connecting P_0 and P_1.
- Learn a velocity field $\mathbf{u} : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ so the ODE

$$\boxed{\frac{d}{dt}\psi(\mathbf{x},t) = \mathbf{u}(\psi(\mathbf{x},t),t), \qquad \psi(\mathbf{x},0) = \mathbf{x}}$$

generates marginals P_t via $\mathbf{X}_t = \psi(\mathbf{X}_0,t)$ and **synthetic data** $\mathbf{X}_1 = \psi(\mathbf{X}_0,1)$.

- ODE flow implies $\partial_t p_t(\mathbf{x}) + \nabla\cdot\big(p_t(\mathbf{x})\mathbf{u}(\mathbf{x},t)\big) = 0$, where $\nabla\cdot F = \sum_{j=1}^d \frac{\partial F_j}{\partial \mathbf{x}_j}$.
- Affine Gaussian path: $\mathbf{X}_t = \alpha_t \mathbf{X}_1 + \beta_t \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,I)$
  with $(\alpha_0,\beta_0) = (0,1)$ and $(\alpha_1,\beta_1) = (1,0)$, e.g. $\alpha_t = t^2, \beta_t = \sqrt{1-t^2}$

Flow Matching Setup

- Fix a base law P_0 that is easy to sample and a target law $P_1 = P_{\text{data}}$.
- Choose a probability path $\{P_t\}_{t\in[0,1]}$ connecting P_0 and P_1.
- Learn a velocity field $\mathbf{u} : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ so the ODE

$$\boxed{\frac{d}{dt}\psi(\mathbf{x},t) = \mathbf{u}(\psi(\mathbf{x},t),t), \qquad \psi(\mathbf{x},0) = \mathbf{x}}$$

  generates marginals P_t via $\mathbf{X}_t = \psi(\mathbf{X}_0,t)$ and **synthetic data** $\mathbf{X}_1 = \psi(\mathbf{X}_0,1)$.

- ODE flow implies $\partial_t p_t(\mathbf{x}) + \nabla\cdot\big(p_t(\mathbf{x})\mathbf{u}(\mathbf{x},t)\big) = 0$, where $\nabla\cdot F = \sum_{j=1}^d \frac{\partial F_j}{\partial \mathbf{x}_j}$.
- **Affine Gaussian path**: $\mathbf{X}_t = \alpha_t \mathbf{X}_1 + \beta_t \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,I)$
  with $(\alpha_0,\beta_0) = (0,1)$ and $(\alpha_1,\beta_1) = (1,0)$, e.g. $\alpha_t = t^2, \beta_t = \sqrt{1-t^2}$

Conditional Flow Matching

- Introduce a conditional path $P_t(\cdot \mid \mathbf{X}_1)$ with endpoint $\mathbf{X}_1 \sim P_{\text{data}}$.
- If $\mathbf{u}(\mathbf{x}, t \mid \mathbf{X}_1)$ is the conditional velocity, then

$$\mathbf{u}(\mathbf{x}, t) = \mathrm{E}\left[\mathbf{u}(\mathbf{x}, t \mid \mathbf{X}_1) \mid \mathbf{X}_t = \mathbf{x}\right].$$

- It can easily be shown that

$$\int_0^1 \mathrm{E}\left[\|\mathbf{u}_\theta(\mathbf{X}_t, t) - \mathbf{u}(\mathbf{X}_t, t)\|^2\right] dt = \int_0^1 \mathrm{E}\left[\|\mathbf{u}_\theta(\mathbf{X}_t, t) - \mathbf{u}(\mathbf{X}_t, t \mid \mathbf{X}_1)\|^2\right] dt + C$$

Therefore, we need only to minimize the conditional flow matching.

Affine Gaussian Path

★ For a conditional path $X_t = \alpha_t X_1 + \beta_t \varepsilon$, it satisfies

$$\frac{d}{dt} X_t = \dot{\alpha}_t X_1 + \dot{\beta}_t \varepsilon.$$

★ Since $\varepsilon = (X_t - \alpha_t X_1)/\beta_t$, the conditional velocity can also be written as

$$\mathbf{u}(X_t, t \mid X_1) = \dot{\alpha}_t X_1 + \frac{\dot{\beta}_t}{\beta_t}(X_t - \alpha_t X_1).$$

Empirical Minimizer: $t_i \sim \text{Unif}(0, 1)$, learn volicity field by

$$\text{argmin}_{\mathbf{u}_\theta \in \mathcal{F}} \sum_{i=1}^n \Big(\underbrace{\dot{\alpha}_{t_i} X_{1,i} + \frac{\dot{\beta}_{t_i}}{\beta_{t_i}}(X_{t_i} - \alpha_{t_i} X_{1,i})}_{y_i} - \mathbf{u}_\theta(X_{t_i}, t_i) \Big)^2,$$

where $\{X_{1,i}\}_{i=1}^n$ are real images and $X_{t_i} = \alpha_{t_i} X_{1,i} + \beta_{t_i} \varepsilon_i$ an image on the path.

Affine Gaussian Path

★ For a conditional path $X_t = \alpha_t X_1 + \beta_t \varepsilon$, it satisfies

$$\frac{d}{dt} X_t = \dot{\alpha}_t X_1 + \dot{\beta}_t \varepsilon.$$

★ Since $\varepsilon = (X_t - \alpha_t X_1)/\beta_t$, the conditional velocity can also be written as

$$\mathbf{u}(X_t, t \mid X_1) = \dot{\alpha}_t X_1 + \frac{\dot{\beta}_t}{\beta_t}(X_t - \alpha_t X_1).$$

Empirical Minimizer: $t_i \sim \text{Unif}(0, 1)$, learn volicity field by

$$\text{argmin}_{\mathbf{u}_\theta \in \mathcal{F}} \sum_{i=1}^n \Big(\underbrace{\dot{\alpha}_{t_i} X_{1,i} + \frac{\dot{\beta}_{t_i}}{\beta_{t_i}}(X_{t_i} - \alpha_{t_i} X_{1,i})}_{y_i} - \mathbf{u}_\theta(X_{t_i}, t_i) \Big)^2,$$

where $\{X_{1,i}\}_{i=1}^n$ are real images and $X_{t_i} = \alpha_{t_i} X_{1,i} + \beta_{t_i} \varepsilon_i$ an image on the path.

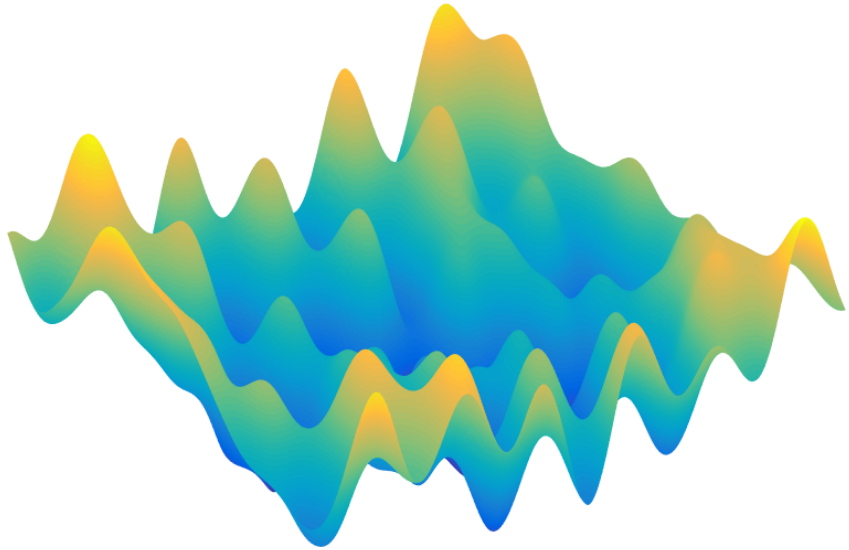
1.8. Training Algorithms

Empirical risk minimization

$$\text{minimize}_{\theta \in \mathbb{R}^p} \qquad \ell_n(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(\mathbf{x}_i; \theta\right), y_i\right)$$

Examples: ★Reg. loss; ★Classification loss (cross entropy).

What's new for deep learning?



Challenges in training deep NNs

★ **Scale**. Both n and p can be huge.

★ **Non-convexity**. $\ell_n(\theta)$ is highly nonconvex with scary landscapes. **Not clear** whether an algorithm can drive it small.

★ **Numerical stability.** "exploding gradients" or "vanishing gradients" arise during the training process.

★ **Generalization performance**. Out-of-sample error is $\ell(\widehat{\theta})$, where $\ell(\theta) \triangleq \mathbb{E}[\ell_n(\theta)]$. Multiple minima or local minima. ★Not sure able to find one. ★Need to guard against overfitting.

Challenges in training deep NNs

★ **Scale**. Both n and p can be huge.

★ **Non-convexity**. $\ell_n(\theta)$ is highly nonconvex with scary landscapes. **Not clear** whether an algorithm can drive it small.

★ **Numerical stability.** "exploding gradients" or "vanishing gradients" arise during the training process.

★ **Generalization performance**. Out-of-sample error is $\ell(\widehat{\theta})$, where $\ell(\theta) \triangleq \mathbb{E}[\ell_n(\theta)]$. Multiple minima or local minima. ★Not sure able to find one. ★Need to guard against overfitting.

Gradient descent method

$$\text{minimize}_{\theta \in \mathbb{R}^p} \qquad \ell_n(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(\mathbf{x}_i; \theta\right), \mathbf{y}_i\right)$$

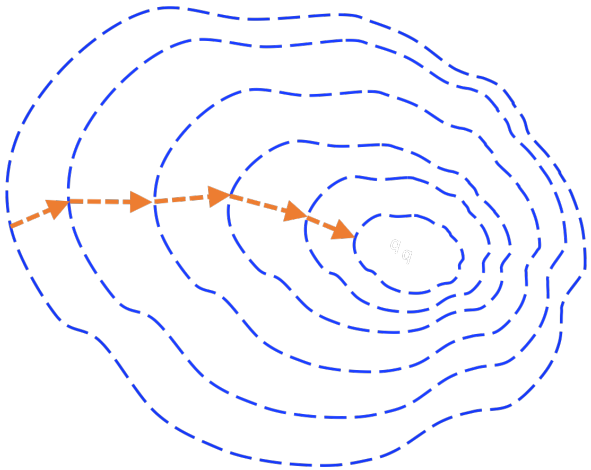
Gradient Descent: $\theta^{t+1} \triangleq \theta^t - \eta_t G(\theta^t)$, where

★η_t learning rate

$$G(\theta^t) \triangleq \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}\left(f\left(\mathbf{x}_i; \theta^t\right), \mathbf{y}_i\right)$$

Problems:

- ★ Heavy computation in each iteration (no need for high precision in each proposed direction of move)
- ★ Memory inefficient: data cannot fit in memory
- ★ Cannot handle online data



Gradient descent method

$$\text{minimize}_{\theta \in \mathbb{R}^p} \qquad \ell_n(\theta) \triangleq \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(\mathbf{x}_i; \theta\right), \mathbf{y}_i\right)$$

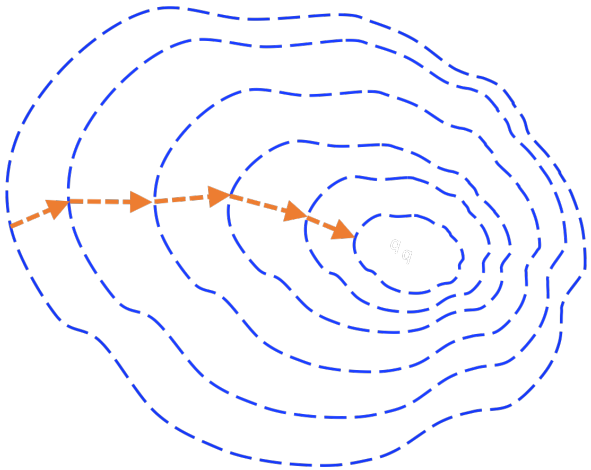
Gradient Descent: $\theta^{t+1} \triangleq \theta^t - \eta_t G(\theta^t)$, where

★η_t learning rate

$$G(\theta^t) \triangleq \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}\left(f\left(\mathbf{x}_i; \theta^t\right), \mathbf{y}_i\right)$$

Problems:

- ★ Heavy computation in each iteration (no need for high precision in each proposed direction of move)
- ★ Memory inefficient: data cannot fit in memory
- ★ Cannot handle online data



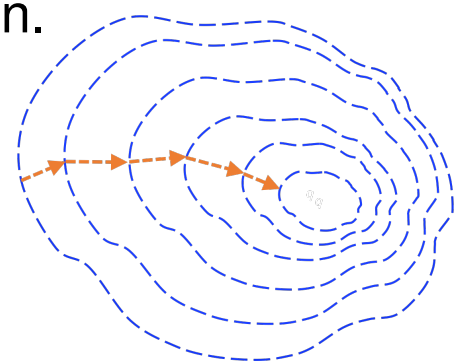
Stochastic gradient descent

Sample i_t from $\{1, 2, \cdots, n\}$, compute *(Robbins & Monro, 1951)*

$$G(\theta^t) \triangleq \nabla \mathcal{L}\left(f\left(\mathbf{x}_{i_t}; \theta^t\right), y_{i_t}\right)$$

$$\theta^{t+1} \triangleq \theta^t - \eta_t G(\theta^t)$$

- Use random initializations, usually independent Gaussian.



Benefits of SGD:

- ★ Use a single example in each update;
- ★ More efficient in computation and memory
- ★ Expected value = Gradient based on all the sample.

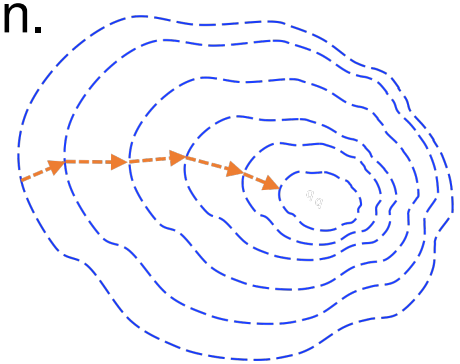
Stochastic gradient descent

Sample i_t from $\{1, 2, \cdots, n\}$, compute *(Robbins & Monro, 1951)*

$$G(\theta^t) \triangleq \nabla \mathcal{L}\left(f\left(\mathbf{x}_{i_t}; \theta^t\right), y_{i_t}\right)$$

$$\theta^{t+1} \triangleq \theta^t - \eta_t G(\theta^t)$$

- Use random initializations, usually independent Gaussian.



Benefits of SGD:

- ★ Use a single example in each update;
- ★ More efficient in computation and memory
- ★ Expected value = Gradient based on all the sample.

Mini-batch SGD

- Feed a batch of K data $\left\{\left(\mathbf{x}_{i_t^k}, y_{i_t^k}\right)\right\}_{k=1}^K$ at a time and compute

$$G_K(\theta^t) = \frac{1}{K}\sum_{k=1}^K \nabla \mathcal{L}\left(f\left(\mathbf{x}_{i_t^k}; \theta^t\right), y_{i_t^k}\right), \qquad \theta^{t+1} = \theta^t - \eta_t G_K(\theta^t).$$

Advantages:

- ★ using $1 \ll K \ll n$ examples to estimate the gradient reduces the variance and hence accelerates the convergence;
- ★ by taking the **batch size** K appropriately (say 64 or 128), the stochastic gradient $G_K(\theta^t)$ can be efficiently computed using GPUs.

- One epoch = a pass of all training data $\approx n/K$ gradient steps.

Mini-batch SGD

- Feed a batch of K data $\left\{\left(\mathbf{x}_{i_t^k}, y_{i_t^k}\right)\right\}_{k=1}^K$ at a time and compute

$$G_K(\theta^t) = \frac{1}{K}\sum_{k=1}^K \nabla \mathcal{L}\left(f\left(\mathbf{x}_{i_t^k}; \theta^t\right), y_{i_t^k}\right), \qquad \theta^{t+1} = \theta^t - \eta_t G_K(\theta^t).$$

Advantages:

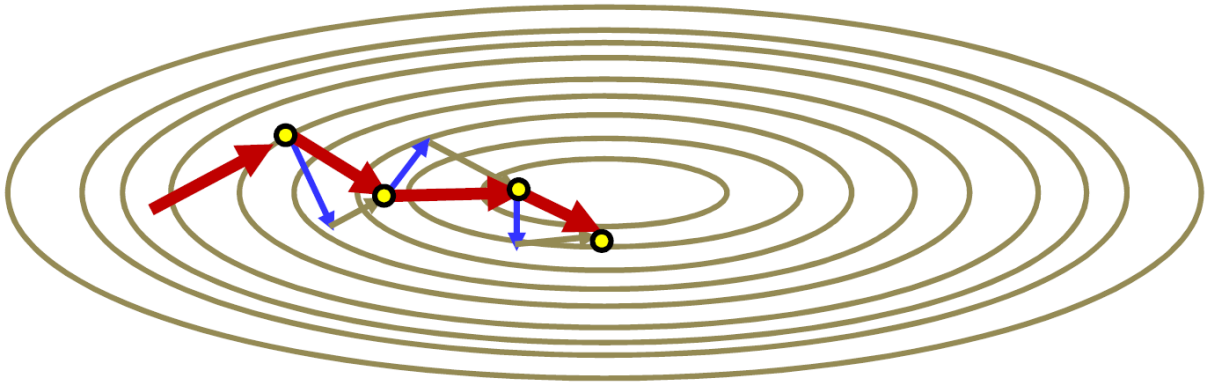
- ★ using $1 \ll K \ll n$ examples to estimate the gradient reduces the variance and hence accelerates the convergence;
- ★ by taking the **batch size** K appropriately (say 64 or 128), the stochastic gradient $G_K(\theta^t)$ can be efficiently computed using GPUs.

- One epoch = a pass of all training data $\approx n/K$ gradient steps.

SGD with momentums

Accelerating gradient descent with **momentum**

.



Credit: MLSP at CMU

Blue - gradient, Red - true moving direction

- Updating rule: $\mathbf{v}^t = \rho \mathbf{v}^{t-1} + (1-\rho) G(\theta^t)$ and $\theta^{t+1} = \theta^t - \eta_t \mathbf{v}^t$.
- $\rho \in (0,1)$: momentum term. Typical choice $\rho = 0.9$.
- Note that $\mathbf{v}^t = (1-\rho)\sum_{j=0}^{t} \rho^{t-j} G(\theta^j)$ is exponential smoothing

Adaptive learning rate

$$\theta^{t+1} = \theta^t - \eta \mathbf{P}_t^{-1} G(\theta^t)$$

- ★ **Newton**: $\mathbf{P}_t = \nabla^2 \ell_n(\theta^t)$, **too expensive**
- ★ **AdaGrad**: $\mathbf{P}_t = \left\{ \text{diag}\left(h^{-1} \sum_{j=t-h+1}^{t} G\left(\theta^j\right) G\left(\theta^j\right)^\top \right) + \delta \mathbf{I} \right\}^{1/2}$
- ★ **RMSProp**: $\mathbf{P}_t = \left\{ \text{diag}\left(\rho \mathbf{P}_{t-1} + (1-\rho) G\left(\theta^t\right) G\left(\theta^t\right)^\top \right) \right\}^{1/2}$ with $\rho = 0.9$. Adam becomes a default training algorithm in many applic.
- ★ **Random dropouts**: Randomly select a fraction of features at each layer to avoid over fitting(*Hinton et al., 2012, 62K)*, the same as setting columns of weights as zero.

Adaptive learning rate

$$\theta^{t+1} = \theta^t - \eta \mathbf{P_t^{-1}} G(\theta^t)$$

★ **Newton**: $\mathbf{P}_t = \nabla^2 \ell_n(\theta^t)$, **too expensive**

★ **AdaGrad**: $\mathbf{P}_t = \left\{ \text{diag}\left(h^{-1} \sum_{j=t-h+1}^{t} G\left(\theta^j\right) G\left(\theta^j\right)^\top \right) + \delta \mathbf{I} \right\}^{1/2}$

★ **RMSProp**: $\mathbf{P}_t = \left\{ \text{diag}\left(\rho \mathbf{P}_{t-1} + (1-\rho) G\left(\theta^t\right) G\left(\theta^t\right)^\top \right) \right\}^{1/2}$ with $\rho = 0.9$. Adam becomes a default training algorithm in many applic.

★ **Random dropouts**: Randomly select a fraction of features at each layer to avoid over fitting(*Hinton et al., 2012, 62K)*, the same as setting columns of weights as zero.