

Deep Learning and Generative AI

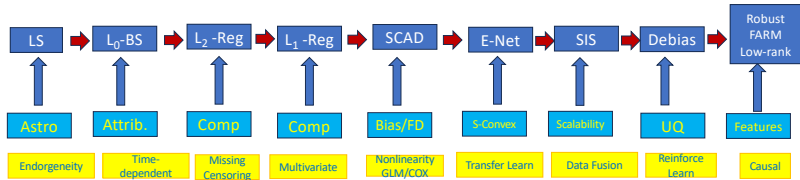
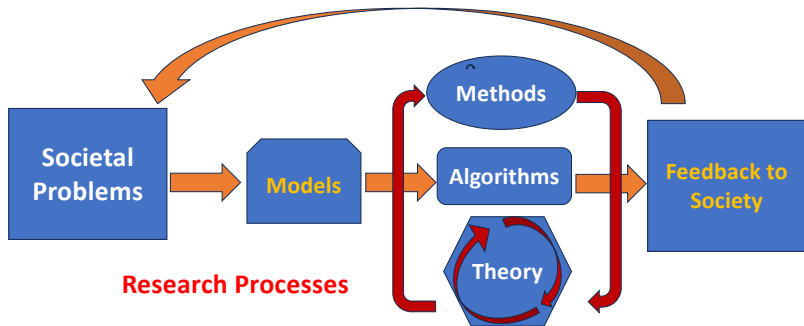
Jianqing Fan

Princeton University

<https://fan.princeton.edu>



Research Processes



1. Introduction to Deep Learning and Generative AI

1.1. Introduction

1.3. Transformers & LLMs

1.5. Generative Adversary Nets

1.7. Flow matching

1.2. FNN, ResNet, CNN, RNN

1.4. Watermarking

1.6 Diffusion Models

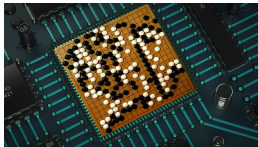
1.8. Training algorithms

★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

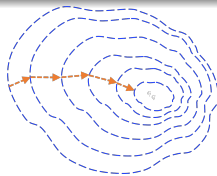
- **Games**: Atari, Go, Starcraft, Texas hold'em, ...
- **Control**: robotic hand, automatic driving, fleet management, ...
- **Prediction**: peotein folding, LLM, generative AI ...
- **Causality**: Molecular mechnism, policy eval, ...



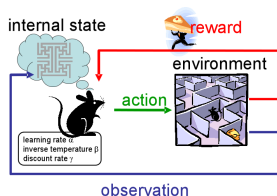
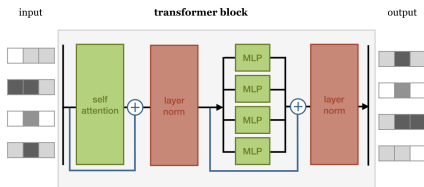
Drivers of successes

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-driven** way.



What is deep neural network?

■ High-dim function approximation using deep neural nets.

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

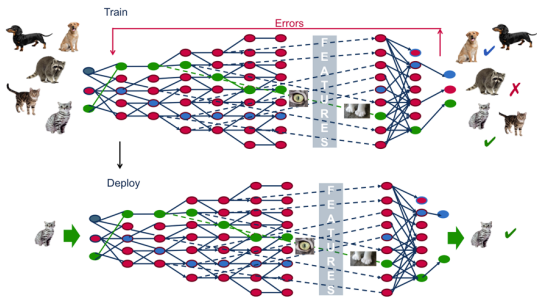
$$\mathbf{h}^{(1)} = \sigma(\mathbf{W}^{(1)}\mathbf{h}^{(0)} + \mathbf{b}^{(1)});$$

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

$\vdots$

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

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.

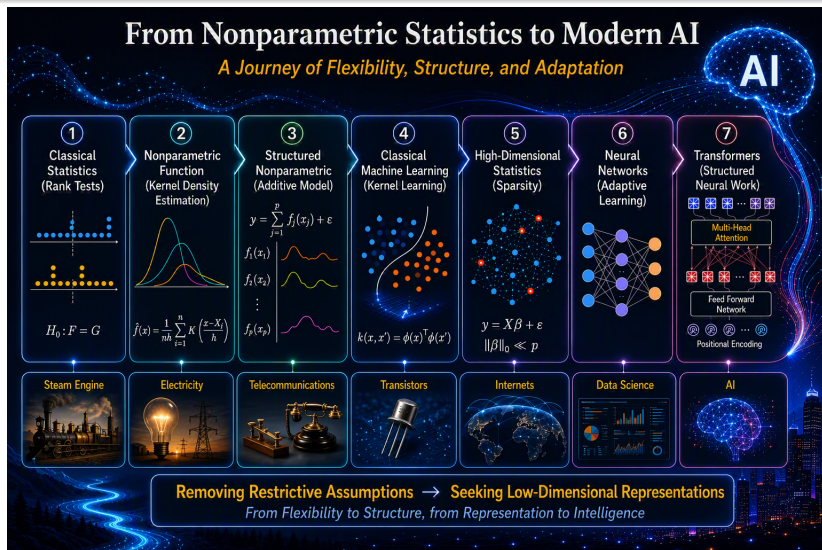
Great for bias-var trade-off!

- ★ Deep nets make high-d function approx (representation) flexible
- ★ Arrivals of Big Data reduce variance
- ★ Modern computation 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

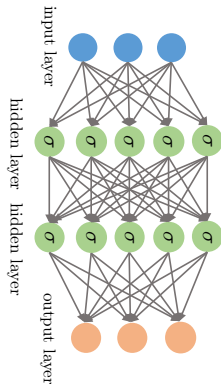
★ **Feed-Forward Neural Networks**
= **Multi Layer Perceptron (MLP)**

Feed-forward neural networks

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

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

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



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

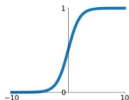
$$f_k(\mathbf{x}) \triangleq \frac{e^{z_k}}{\sum_k e^{z_k}}, \quad \forall k \in [K], \quad \text{where} \quad \mathbf{z} = \mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)} \in \mathbb{R}^K.$$

Common activation functions

Activation Functions

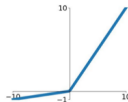
Sigmoid

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



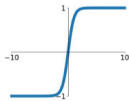
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

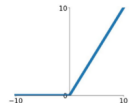


Maxout

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

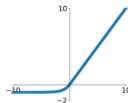
ReLU

$$\max(0, x)$$



ELU

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



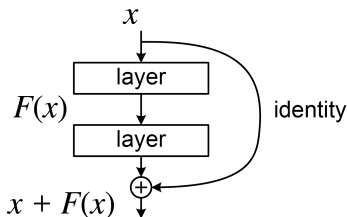
■ ReLU (Rectified Linear Unit) function: $[\sigma(\mathbf{z})]_j = \max\{z_j, 0\}$.

■ 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 general
 $\mathbf{x} + \mathbf{MF}(\mathbf{x})$ with trainable $\mathbf{M}$ that makes
the output dimension comparable.



Regression and Classification

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

Last-layer Output: $f(\mathbf{x}_i; \theta) \triangleq (f_1(\mathbf{x}_i; \theta), \dots, f_K(\mathbf{x}_i; \theta))^T$, $\theta \triangleq \{\mathbf{w}^{(\ell)}, \mathbf{b}^{(\ell)} : 1 \leq \ell \leq L+1\}$.

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 accommodates **multivariate nonparametric** prediction.

Convolutional Neural Networks

Drawbacks of general feed-forward neural networks:

- ★ Number of parameters grows quickly with 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

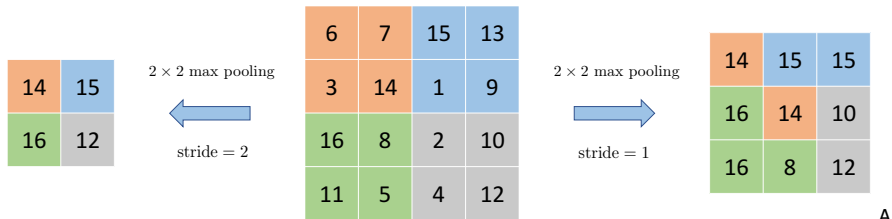
Convolution in 2D

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

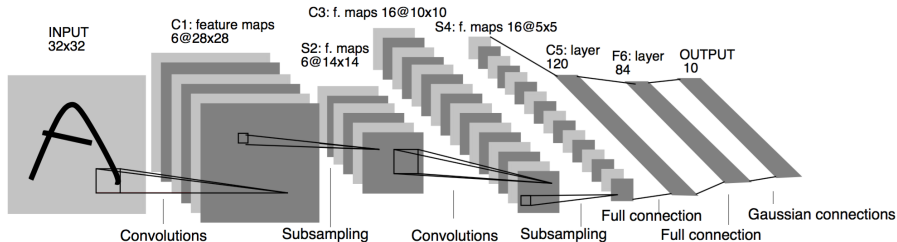


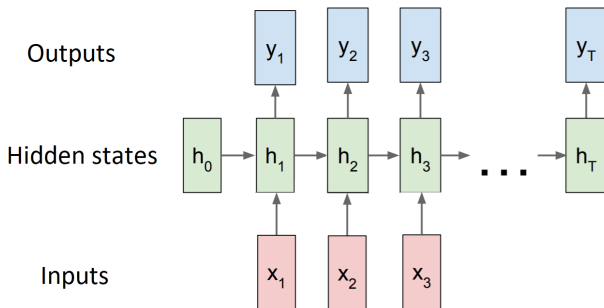
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- **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



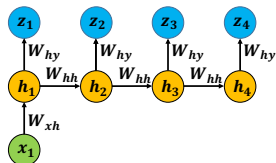
★ Recurrence: $\mathbf{h}_t = \mathbf{f}_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$, e.g.

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

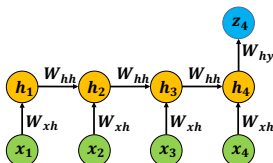
$$\mathbf{y}_t = \sigma(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y),$$

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

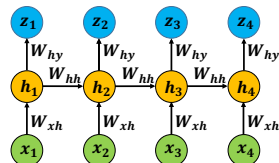
Variations of networks



(a) One-to-many



(b) Many-to-one



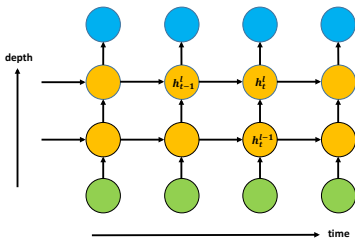
(c) Many-to-many

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

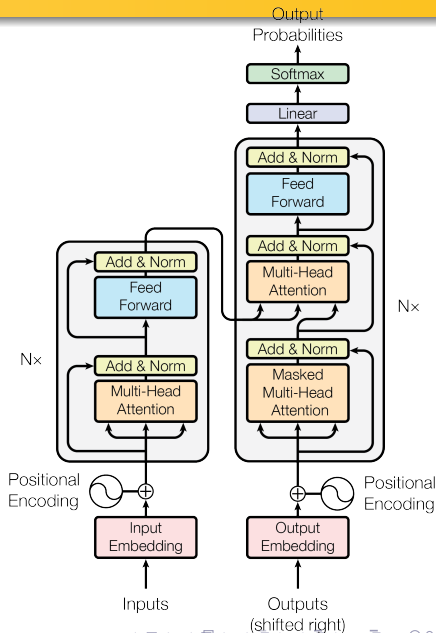


1.3 Transformers

Transformers have revolutionized the field of modern AI

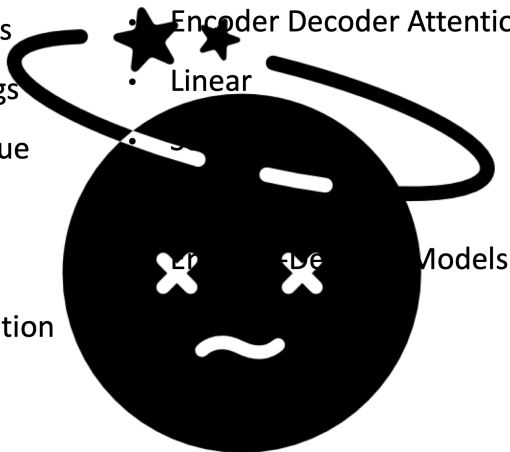
★ Transformer model architecture
in Vaswani, et al. (2017, NeurIPS).

"Attention is all you need".



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



Tokenization

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

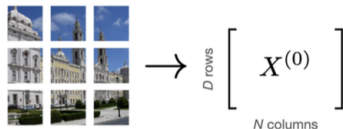
Tokens	Characters
8	41

"Statistical Foundations of Data Science"

Token ID: [1, 16257, 40596, 99612, 315, 2956, 10170, 1]

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

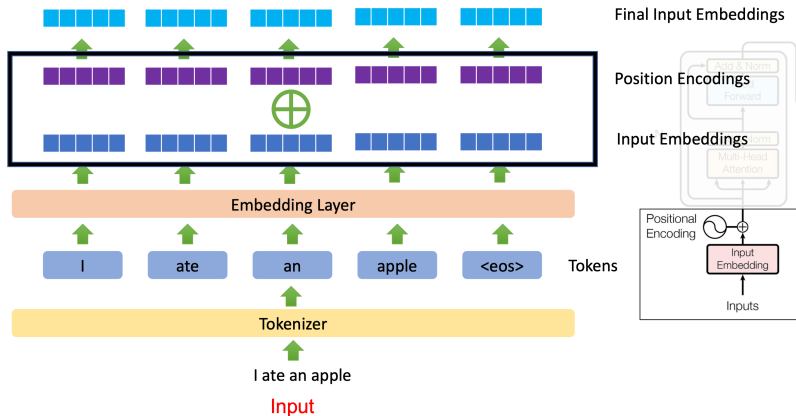
Taken from Richard Turner (2023)



$$\mathbf{x}_n^{(0)} = W \text{vec} \left(\begin{array}{c} \text{image patch} \\ \text{n}^{\text{th}} \text{ patch} \end{array} \right)$$

Embedding

Token embedding: map a token to a vector: $\mathbf{A} Token \rightarrow \mathbb{R}^d$, $\mathbf{A}$ trainable, $d \approx 10^4$.



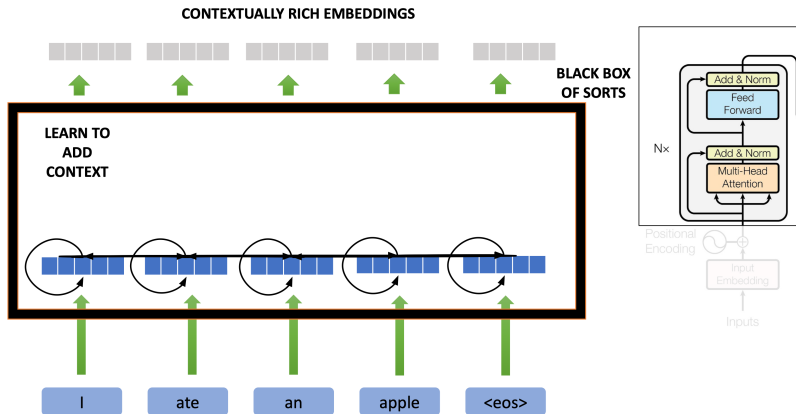
Position Encoding: "I ate an apple ." $\neq$ "ate apple I an ."

Transformer Input: Input Embedding Matrix (light blue rows)

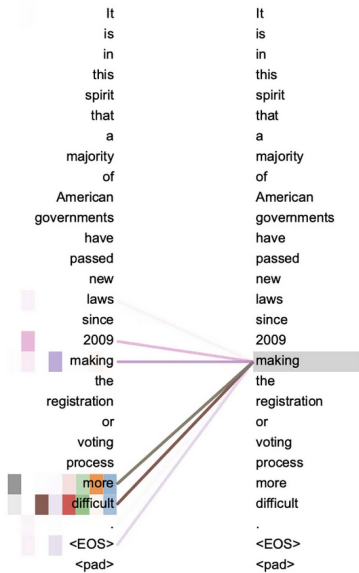
Transformer block

Goal: Capture contextual interrelationship between embedded tokens

In translation, word by word translation is not correct; the key is to determine which words are most important in a specific context.



An Illustrative Example



It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more

Transformer block—one head

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

- self-attention w/ residual skip connection (asymmetric similarity, “hammer” is strongly associated with “tools”, not vice versus)

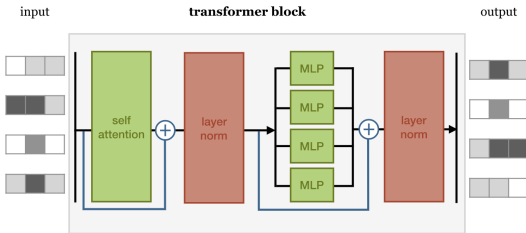
$\mathbf{H} = \mathbf{Q}^\top \mathbf{K}$ is **low-rank** factorization.

$$\text{Attn}(\mathbf{Z}) = \mathbf{Z} + \underbrace{\mathbf{VZ}}_{\text{Value}} \text{softmax} \left[\underbrace{(\mathbf{QZ})^\top}_{\text{Query}} \underbrace{\mathbf{KZ}}_{\text{Key}} \right]$$

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

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

- layer normalization standardizes each column/token (optional)



Multi-head self-attention transformers

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

$$\text{Attn}_{\theta_1}(\mathbf{H}) = \mathbf{H} + \sum_{m=1}^M (\mathbf{V}_m \mathbf{H}) \sigma \left((\mathbf{Q}_m \mathbf{H})^\top (\mathbf{K}_m \mathbf{H}) \right),$$

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

$$\text{FC}_{\theta_2}(\mathbf{H}) := \mathbf{H} + \mathbf{W}_2 \sigma(\mathbf{W}_1 \mathbf{H}).$$

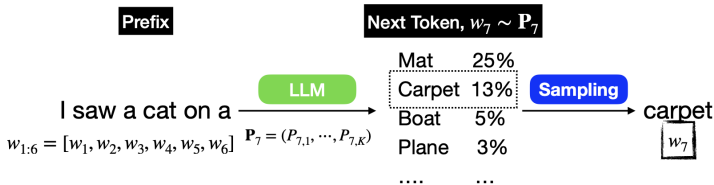
Transformer: L -layer, output dim d ,

$$\text{TF}_{\theta}(\mathbf{H}) := \tilde{\mathbf{W}}_0 \times \text{FC}_{\theta_2^L}(\text{Attn}_{\theta_1^L}(\cdots \text{FC}_{\theta_2^1}(\text{Attn}_{\theta_1^1}(\mathbf{H}))) \times \tilde{\mathbf{W}}_1,$$

where $\tilde{\mathbf{W}}_0 \in \mathbb{R}^{d_1 \times d}$ and $\tilde{\mathbf{W}}_1 \in \mathbb{R}^{N \times d_2}$.

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)

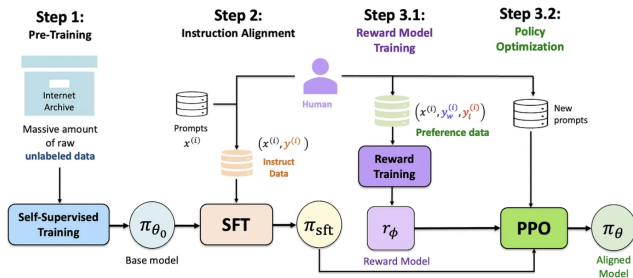


Training loss: $-\sum_{i \in \text{doc}} \sum_{t=2}^{\text{length}_i} \log P_{\theta}(w_{i,t} | w_{i,1:t-1})$

Predictive: To be predictive, attention matrices should be upper triangular.

Fine-Tuning

Fine-Tuning in LLM Training. **Base model** (expensive) $\rightarrow$ **SFT** $\rightarrow$ **PPO**



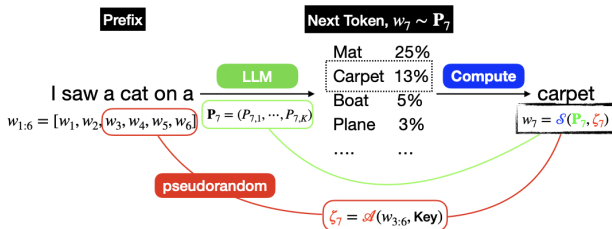
1.4 Watermarking

Basic Idea

Pseudorandomness:

★ $\zeta_t = \mathcal{A}(w_{1:t-1}, \text{Key})$ for $t = 1, \dots, n$ are iid, independent of $w_{1:t-1}$.

★ 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.

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

• $\zeta_t \sim_{i.i.d.} \text{Unif}(0, 1)$ and $w_t = \mathcal{S}(\mathbf{P}_t, \zeta_t) = I(\zeta_t \leq P_{0,t})$.

• Detection statistics: $\sum_{i=1}^n (2w_t - 1)(2\zeta_t - 1)$ (large $\implies$ generated text)

Inverse Transform and Gumbel Watermarks

■ A watermark corresponds to sampling from a multinomial distribution

Inverst Transform Watermarking:

(Kirchenbauer, et al, 23)

Partition $(0, 1)$ into K intervals according to $\mathbf{P}_t$.

(Generalize Bernoulli)

Gumbel-max Watermarking :

(Aaronson, 2023)

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

where $\zeta = (U_1, \dots, U_K)$ i.i.d. $\operatorname{uniform}(0, 1)$.

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

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

Power of Pivotal Test

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

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

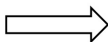
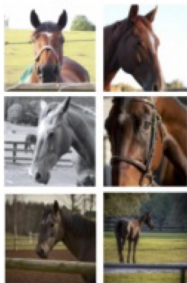
$$R_{\mathcal{P}}(h) = -\inf_{\theta \geq 0} \left\{ \theta \mathbb{E}_0 h(Y) + \log \phi_{\mathcal{P}, h}(\theta) \right\}, \quad \phi_{\mathcal{P}, h}(\theta) = \sup_{P \in \mathcal{P}} \mathbb{E}_{1, P} e^{-\theta h(Y)}.$$

- ★ $R_{\mathcal{P}}(h) = 0$ for any h if $\mathcal{P}$ includes $(0, \dots, 0, 1, 0, \dots, 0)$.
- ★ 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. (2005). A Statistical Framework of Watermarks for Large Language Models: Pivot, Detection Efficiency and Optimal Rules

1.5 Generative Adversarial Networks

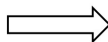
Generative modeling

training data

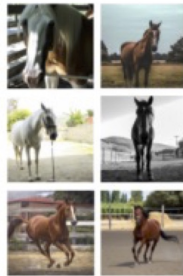


Generative
modeling

$\hat{p}_{\text{data}}$



new samples



- 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 $X \sim \hat{p}_{\text{data}}$

Sampling as a zero-sum game

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

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 $\{\mathbf{x}_i\}_{1 \leq i \leq n}$
- 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 can't tell

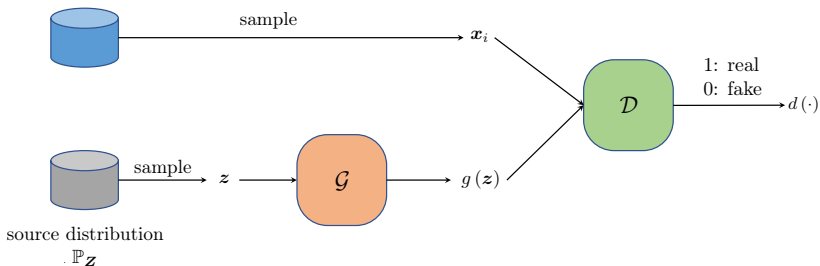
Generative Adversary Network

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

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

training samples

$\{\mathbf{x}_i\}_{1 \leq i \leq n}$



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

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**). $\mathbf{Z} \sim$ **Gaussian**

Remarks

- 1 The empirical minimax problem is to solve

$$\min_{\theta_G} \max_{\theta_D} \frac{1}{n} \sum_{i=1}^n \log(d_{\theta_D}(\mathbf{x}_i)) + \frac{1}{n} \sum_{i=1}^n \log(1 - d_{\theta_D}(\mathbf{g}_{\theta_G}(\mathbf{z}_i))) .$$

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

$$\min_{\theta_G} \left[\mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathbf{x}}} \log \frac{\mathbb{P}_{\mathbf{x}}(\mathbf{x})}{\mathbb{P}_{\mathbf{x}}(\mathbf{x}) + \mathbb{P}_G(\mathbf{x})} + \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_G} \log \frac{\mathbb{P}_G(\mathbf{x})}{\mathbb{P}_{\mathbf{x}}(\mathbf{x}) + \mathbb{P}_G(\mathbf{x})} \right],$$

★ $\mathbb{P}_G$ = density of $\mathbf{g}_{\theta_G}(\mathbf{Z})$.

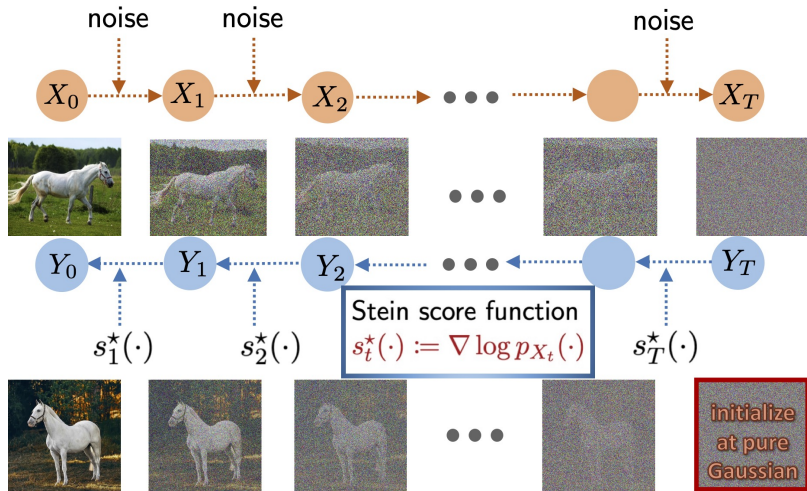
★ unconstrained min over $\mathbb{P}_G = \mathbb{P}_{\mathbf{x}}$

- 3 This generalizes to other divergences. Wasserstein GAN

$$\min_{\theta_G} \sup_{f: f \text{ 1-Lipschitz}} \underbrace{\mathbb{E}_{\mathbf{x} \sim \mathbb{P}_{\mathbf{x}}} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_G} [f(\mathbf{x})]}_{\text{dist}(\mathbb{P}_{\mathbf{x}}, \mathbb{P}_G)} \quad \text{min dist}$$

1.6 Diffusion Models

New Image Generations



- **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

$$dX_t = -\beta_t X_t + \sqrt{2\beta_t} dB_t \quad \text{with} \quad X_0 \sim \mathbf{p}_0, \quad \text{target on } \mathbb{R}^d$$

◆ 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)

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

◆ Score function $\mathbf{s}^*(t, \mathbf{x}) = \nabla_{\mathbf{x}} \log \mathbf{p}_t(\mathbf{x})$ where $X_t \sim p_t$
learning object

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

An illustration



★ ImageNet Data with the same initialization

Score Matching

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

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

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

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

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

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

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 $\mathbf{u} : \mathbb{R}^d \times [0, 1] \rightarrow \mathbb{R}^d$ so the ODE

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

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 $\partial_t p_t(\mathbf{x}) + \nabla \cdot (p_t(\mathbf{x})\mathbf{u}(\mathbf{x}, t)) = 0$, where $\nabla \cdot F = \sum_{j=1}^d \frac{\partial F_j}{\partial x_j}$.
- ★ **Affine Gaussian path:** $\mathbf{X}_t = \alpha_t \mathbf{X}_1 + \beta_t \boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, I)$
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

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

★ It can easily be shown that

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

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

Affine Gaussian Path

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

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

★ Since $\epsilon = (X_t - \alpha_t X_1) / \beta_t$, 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 velocity field by

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

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

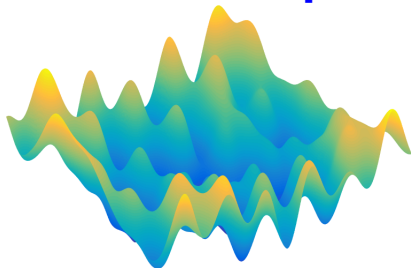
1.8. Training Algorithms

Empirical risk minimization

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

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 $\ell(\hat{\theta})$, 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

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

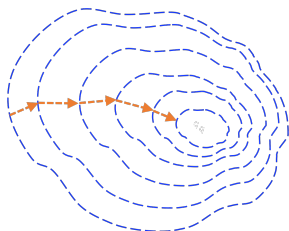
Gradient Descent: $\theta^{t+1} \triangleq \theta^t - \eta_t G(\theta^t)$, where

★ η_t learning rate

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

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 $\{1, 2, \dots, n\}$, compute (Robbins & Monro, 1951)

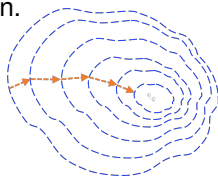
$$G(\theta^t) \triangleq \nabla \mathcal{L}(f(\mathbf{x}_{i_t}; \theta^t), y_{i_t})$$

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

■ 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 $\{(\mathbf{x}_{i_t^k}, y_{i_t^k})\}_{k=1}^K$ at a time and compute

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

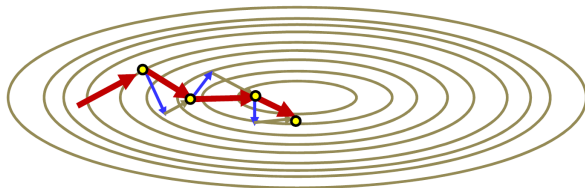
Advantages:

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

★ RMSProp: $\mathbf{P}_t = \left\{ \text{diag} \left(\rho \mathbf{P}_{t-1} + (1 - \rho) G(\theta^t) G(\theta^t)^\top \right) \right\}^{1/2}$ 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.