Linear Least Squares

$$\hat{y}_i = X_i \theta$$

$$J(\theta) = \frac{1}{n} \sum_{i=0}^{n} (\hat{y}_i - y_i)^2 = \frac{1}{n} \sum_{i=0}^{n} (X_i \theta - y_i)^2$$

matrix notation: $J(\theta) = (X\theta - y)^T (X\theta - y)$

$$\theta = (X^T X)^{-1} X^T y$$

Sigmoid

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

Logistic Regression

$$\hat{y}_i = \sigma(X_i, \theta)$$

$$p(y \mid X, \theta) = \hat{y} = p(y=1 \mid X, \theta) = \prod_{i=1}^{n} p(y_i = 1 \mid X_i, \theta)$$

$$\hat{y} = \prod_{i=1}^{n} \hat{y}_i^{y_i} (1 - \hat{y}_i)^{(1-y_i)}$$

BCE (binary cross entropy)

argmax

$$\sum_{i=1}^{n} y_i \log \hat{y}_i + (1-y_i) \log (1 - \hat{y}_i)$$

$$\mathcal{L}(\hat{y}_i, y_i) = y_i \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i)$$

2 casos

$$\begin{cases} y_i = 1 \rightarrow \mathcal{L}(\hat{y}_i, 1) = \log \hat{y}_i \\ y_i = 0 \rightarrow \mathcal{L}(\hat{y}_i, 0) = \log (1 - \hat{y}_i) \end{cases}$$

minimization $\rightarrow$ $$-\frac{1}{n} \sum_{i=0}^{n} \mathcal{L}(\hat{y}_i, y_i)$$

BCE general case $\rightarrow$ $$\mathcal{L}(\hat{y}_i, y_i) = \sum_{i=1}^{N} \sum_{i=1}^{C} y_{i,j} \log \hat{y}_{i,j}$$

Activation Functions

- Sigmoid → $\sigma(x) = \frac{1}{1+e^{x}}$
- tanH → $\tanh(x)$
- ReLU → $\max(0, x)$
- Leaky ReLU → $\max(0.1x, x)$
- Parametric ReLU → $\max(ax, x)$

L1 Loss

$$L(y, \hat{y}, \theta) = \frac{1}{n} \sum_{i=1}^{n} \| y_i - \hat{y}_i \|_1$$

MSE (Mean-Squared-Error)

$$\| y_i - \hat{y}_i \|_2^2 = (y_i - \hat{y}_i)^2$$

$$L(y, \hat{y}, \theta) = \frac{1}{n} \sum_{i=1}^{n} \| y_i - \hat{y}_i \|_2^2$$

BCE (Binary Cross Entropy) → binary classification

$$L(y, \hat{y}, \theta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

Cross Entropy → multi-class classification

$$L(y, \hat{y}, \theta) = -\sum_{i=1}^{n} \sum_{K=1}^{K} (y_{iK} \cdot \log \hat{y}_{iK})$$

Stochastic Gradient Descent (SGD)

$$\frac{1}{n}\left(\sum_{i=1}^{n} L_i(\theta, x_i, y_i)\right) = E_{i\sim[1\ldots n]}\left[L_i(\theta, x_i, y_i)\right]$$

Bias → Difference between average prediction of our model and the correct value which we are trying to predict

High error on train and test data UNDERFITTING

High Variance → OVERFITTING

Multiclass classification: Softmax

$$p(y_i|x_i,\theta) = \frac{e^{s_{y_i}}}{\sum_{k=1}^{c} e^{s_k}} = \frac{e^{x_i\theta_{y_i}}}{\sum_{k=1}^{c} e^{x_i\theta_k}}$$

- Cross Entropy Loss: $L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_k e^{s_k}}\right)$ → Always want to improve (Loss never going to 0)
- Hinge loss: $L_i = \sum_{k\neq y_i} \max(0, s_k - s_{y_i} + 1)$ → saturates

Xavier Initialization

- Gaussian with 0 mean, with standard deviation:

$$Var(s) = \sum_{i}^{n} Var(x_i) Var(w_i) = n\,[Var(w)\,Var(x)]$$

Ensure the variance of the output is the same as the input

$$Var(s) = Var(x) \rightarrow \underbrace{n \cdot Var(w)}_{1}\, Var(y) = Var(x)$$

$$\rightarrow Var(w) = \frac{1}{n}$$

- With ReLU (that kills half of the data)

$$Var(w) = \frac{1}{n/2} = \frac{2}{n}$$

CNN

- Without padding: (Valid convolution)

Input: N×N

filter: F×F

Stride: S

$$\text{Output} \rightarrow \left(\frac{N-F}{S}+1\right) \times \left(\frac{N-F}{S}+1\right)$$

- With padding: (Same convolution → output = input size)

  ↓ setting $P = \frac{F-1}{2}$

Padding: P

$$\text{Output} \rightarrow \left(\left\lfloor \frac{N+2P-F}{S} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{N+2P-F}{2} \right\rfloor + 1\right)$$

- Pooling

  - Spatial filter F×F
  - Stride S
  - Output → $W_{out} \times H_{out} \times D_{out}$

$$W_{out} = \frac{W_{in}-F}{S}+1$$

$$H_{out} = \frac{H_{in}-F}{S}+1$$

$$D_{out} = D_{in}$$

CNN Architectures

- LeNet → Digit recognition (10 classes)

  Conv → avg pool → Conv → avg pool → FC → FC → $\hat{y}$

  tanh / Sigmoid as activation function

  60k parameters

- AlexNet (Same Conv) (1000 classes)

  Conv → max pool → Same conv. → Max pool

  → Same conv → Same conv → Same conv → Max pool → FC → FC → FC

  ReLU

  ~1000 time bigger than LeNet (softmax)

  60 M parameters

- VGGNet

  - Striving for simplicity
  - Conv: 3x3 filters with stride 1 (Same conv) } just repeting that
  - Max pool: 2x2 with stride 2
  - FC, FC, Softmax

  137 M parameters

- ResNet
  - Xavier/2 Init.
  - SGD + Momentum
  - Learing rate 0.1 / 10 when plateau
  - Mini-batch 256
  - Weight decay 1e-5
  - No dropout

60M parameters

deeper networks ResNet → better performance

- Xception Net (extreme version of Inception)

  - 36 conv layers (with skip connections)
  - Applies depthwise separable Convolutions instead of normal conv

- U-Net
  - Left Side → Contraction Path (Encoder)
  - Right Side → Expansion Path (Decoder)

LSTM

. Gates:

current input ↑ | previous input ↑

1. Forget gate $f_t = \text{sigmoid}(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)$

   - 0 → forget info (erase cell)
   - 1 → keep info (keep cell)

2. Input gate $i_t = \text{sigmoid}(\theta_{xi} x_t + \theta_{hi} h_{t-1} + b_i)$

   decides which values will be updated

$\tanh(-1, 1)$

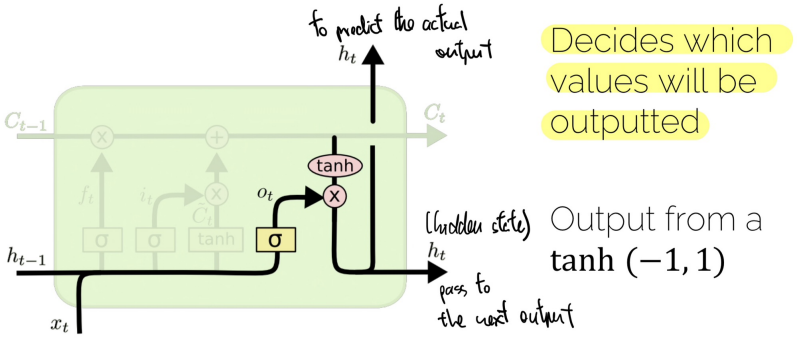
add ↓ subtract ↓

$C_t = f_t \odot C_{t-1} + i_t \odot g_t$

3. Output gate → Decides which values will be outputted

$h_t = o_t \odot \tanh(C_t)$

- Output gate $\boldsymbol{h_t} = \boldsymbol{o_t} \odot \tanh(\boldsymbol{C_t})$



- Forget gate $\boldsymbol{f}_t = sigm(\boldsymbol{\theta}_{xf}\boldsymbol{x}_t + \boldsymbol{\theta}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f)$
- Input gate $\boldsymbol{i}_t = sigm(\boldsymbol{\theta}_{xi}\boldsymbol{x}_t + \boldsymbol{\theta}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$
- Output gate $\boldsymbol{o}_t = sigm(\boldsymbol{\theta}_{xo}\boldsymbol{x}_t + \boldsymbol{\theta}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o)$
- Cell update $\boldsymbol{g}_t = tanh(\boldsymbol{\theta}_{xg}\boldsymbol{x}_t + \boldsymbol{\theta}_{hg}\boldsymbol{h}_{t-1} + \boldsymbol{b}_g)$

exactly the same in each time step

- Cell $\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t$

weights

- Output $\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$

identity function

Learned through backpropagation

1 for important info → so not vanishing gradient

$C_t = f_t \odot C_{t-1} + i_t \odot g_t$

weights

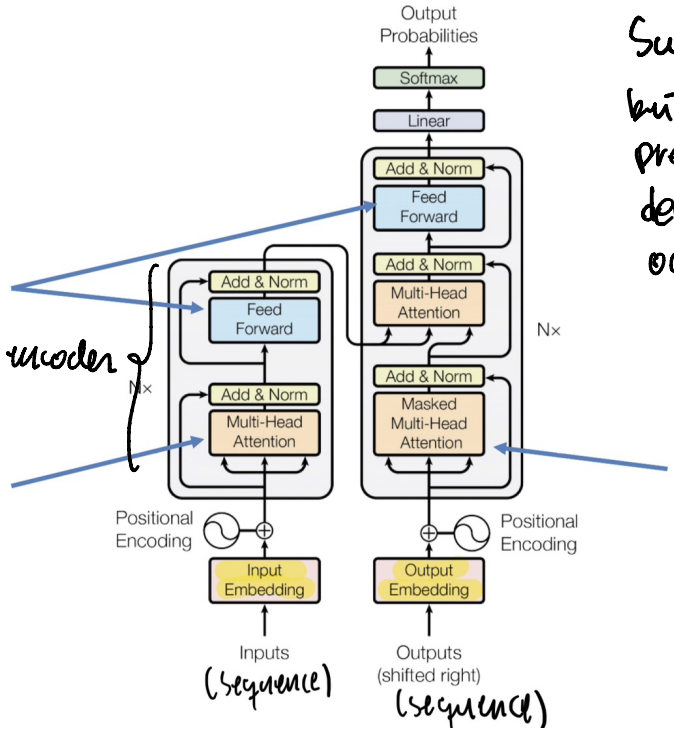
Transformers

Transformers

Same as multi-head attention, but masked. Ensures that the predictions for position i can depend only on the known outputs at positions less than i

Fully connected layer

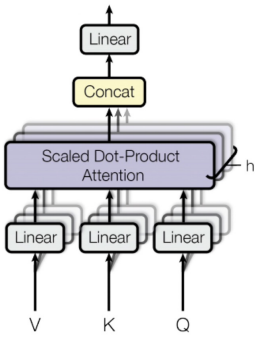
Multi-Head Attention on the "encoder"



Masked Multi-Head Attention on the "decoder"

I2DL: Prof. Dai

Multi-Head Attention



Intuition: Take the query Q, find the most similar key K, and then find the value V that corresponds to the key.

In other words, learn V, K, Q where:
V – here is a bunch of interesting things.
K – here is how we can index some things.
Q – I would like to know this interesting thing.

Loosely connected to Neural Turing Machines (Graves et al.).

Multiple queries with keys (similarities)

Get the values

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

"normalize"

What is missing from self-attention?

- Convolution: a different linear transformation for each relative position. Allows you to distinguish what information came from where.
- Self-attention: a weighted average.

Self-attention: complexity

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

where n is the sequence length, d is the representation dimension, k is the convolutional kernel size, and r is the size of the neighborhood.

Considering that most sentences have a smaller dimension than the representation dimension (in the paper, it is 512), self-attention is very efficient.

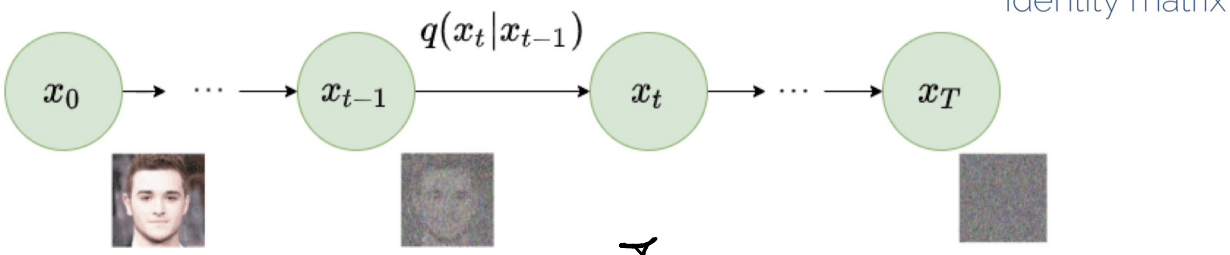
Diffusion :

1. Forward Diffusion

- Markov chain of T steps
  - Each step depends only on previous
- Adds noise to x_0 sampled from real distribution $q(x)$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \mu_t = \sqrt{1-\beta_t}x_{t-1}, \Sigma_t = \beta_t \mathbf{I})$$

mean — variance — identity matrix



$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Not efficient

Reparameterization

- Define $\alpha_t = 1 - \beta_t$, $\overline{\alpha_t} = \prod_{s=0}^{t} \alpha_s$, $\epsilon_0, \ldots, \epsilon_{t-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$$\begin{aligned} x_t &= \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\epsilon_{t-1} \\ &= \sqrt{\alpha_t}x_{t-2} + \sqrt{1-\alpha_t}\epsilon_{t-2} \\ &= \cdots \\ &= \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1-\overline{\alpha_t}}\epsilon_0 \end{aligned}$$

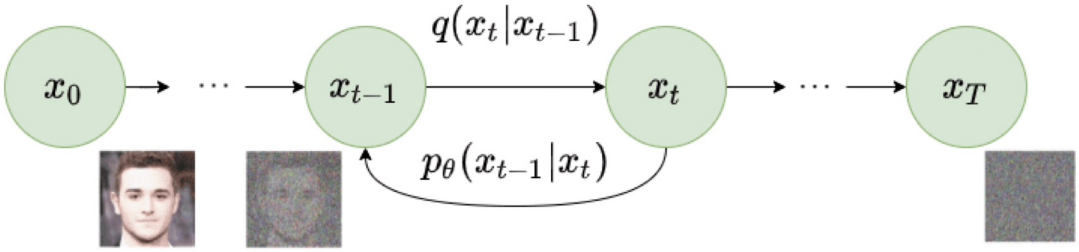
$$x_t \sim q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}x_0, (1-\overline{\alpha_t})\mathbf{I})$$

2. Approximate Reverse Process

- Approximate $q(x_{t-1}|x_t)$ with parameterized model p_θ (e.g., deep network)

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$



Loss:

Training a Diffusion Model

- Optimize negative log-likelihood of training data

$$L_{VLB} = \mathbb{E}_q[D_{KL}(q(x_T|x_0||p_\theta(x_T))] \quad L_T$$

$$+ \sum_{t=2}^{T} D_{KL}(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) \quad - \log p_\theta(x_0|x_1)]$$

L_{t-1} L_0

- Input and output dimensions must match
- Highly flexible to architecture design
- Commonly implemented with U-net architecture