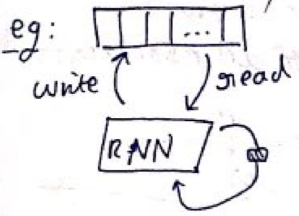
* Memory Networks

- NNs capture implicit knowledge but not explicit knowledge.
- Hence, there were invented to work on explicit

Weston 2014.

eg:



- read chap 11 — (Goodfellow et. al)
- read chap 12 — Practical methodologie, Application,

* Auto encoder

Encoder f

Decoder g

x: input

$f(x) = h$

$g(h) = \tilde{x}$

$\min L(x, \tilde{x})$

$\min L(x, g(f(x)))$

when $L = 0$

- $g(f(x)) = x$ (identity)

i.e $x \longrightarrow h \longrightarrow x$
code

Uses:- ① Dimensionality redn (h is low dim)
$\dim(h) << \dim(x)$

② Information retrieval
→ easier to search in lower dimension
→ efficient to retrieve similar info.

③ Anomaly detection

④ Image denoising

// 1. Undercomplete autoencoder vs overcomplete → not useful

2. purposefully alter the model so that we get $\tilde{x}$ which is similar to x but not same.

// $L(x, g(f(x))) + \Omega(h)$ Regularized autoencoder

$\Omega(h) = \lambda \sum_i |h_i|$ Sparse autoencoders

// Denoising auto encoder (DAE)

$\underbrace{x + Z}_{\tilde{x}} \longrightarrow AE \longrightarrow x$ $L(x, g(f(\tilde{x})))$

$c(x) = \tilde{x}$ (adding noise)

$\in N(x, \sigma^2 I)$

$x \xrightarrow[c]{} \tilde{x} \xrightarrow[f]{} h \xrightarrow[g]{} x$ It force AE to learn salient feature of data.

$\tilde{x} = x + z$

where $z \sim N(0, \Sigma)$ (Gaussian).

// Regularization based on restricting derivative

$\cdot L(x, g(f(x))) + \Omega(x, h)$ $h = f(x)$

$\cdot \Omega(x, h) = \lambda \sum_i (\nabla_x h_i)^2$ $\nabla_x h = \dfrac{\partial f(x)}{\partial x}$

Contractive Auto Encoder (CAE) $\left\| \dfrac{\partial f(x)}{\partial x} \right\|_F^2$: Frobenius norm

// Chap 13 - Good Fellow

// Deep Generative Models

- Variational autoencoder (VAE)
- Generative adversarial networks (GAN)

① Need to assume some structure

② Inference and estimation technique are not efficient

· MCMC

· Variational inference

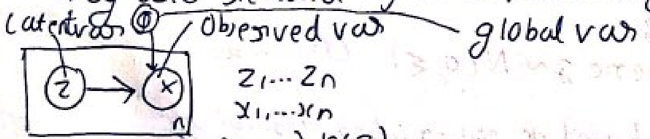
$(\lambda-1)^2 = \alpha^2$ $\lambda - 1 = \pm\alpha$ ... $\lambda - 1 = \pm \alpha$

③ Approximation are required, which lead to suboptimal solutions.

- NN can approximate complex function
- Backpropagation can learn those function efficiently over large scale data.

- Deep feature consistent variational autoencoder 2016
- Hierarchical variational autoencoder 2017
- Auto-encoding variational Bayes, Kingma, Welling ICLR 2014
- Tutorial on VAE Doersch 2016

// Hierarchical models
- multiple random vars
- they are related by a dependency gph



$$P(x) = \sum_z P(x|z)\, P(z)$$

Algo: ① get θ // all params
② for $i = 1$ to n
a. sample z_i from $P(z)$
b. sample x_i from $P(x|z=z_i)$

// - Gaussian mixture model
- MCMC
- VI

// Inference: finding value of latent variable
Estimation: Finding value of parameter

* $p(x) = \int p(x|z) \cdot p(z)\, dz$ (we want to learn this)

* so we use params θ

$p(x|\theta) = \int p(x|z,\theta)\, p(z|\theta)\, dz$

① How to find a suitable model for z?

② How to compute the integration

- $z \sim \mathcal{N}(0, I_1)$ (example)
- ~~[illegible]~~ $x \sim \mathcal{N}(f(z,\theta), \sigma^2 I)$
- f can be any complicated function

// core idea: 1. $z \sim \mathcal{N}(0, I_1)$
2. implement f through a NN
3. $x \sim \mathcal{N}(f(z), \sigma^2 I)$

Apply backprop to learn f

// ~~P(x|z)~~ · maximize $P(x|z)$
· But $P(z|x)$ is intractable

- Need z so that when sampled from $P(x|z)$, x is similar to real data.
- $D(\varphi(z) \| P(z|x))$ is small

KL divergence

- $D(\varphi(z) \| P(z|x)) = E_{z\sim\varphi}[\log \varphi(z) - \log P(z|x)]$

$= E_{z\sim\varphi}[\log \varphi(z) - \log P(x|z) - \log P(z) + \log P(x)]$

$\left(P(z|x) = \frac{P(x|z)\, P(z)}{P(x)} \right)$

- $\log P(x) - D(\varphi(z) \| P(z|x))$

$= E_{z\sim\varphi}[\log P(x|z)] - D(\varphi(z) \| P(z))$

= core eqn of VAE

// $\varphi(z|x)$: Recognition model

$\log P(x) - D(\varphi(z|x) \| P(z|x)) = E_{z\sim\varphi}[\log P(x|z)] - D(\varphi(z|x) \| P(z))$

$\phi(z|x) = \mathbb{N}(\mu(x), \Sigma(x))$

$$\begin{tikzcd} & \mu \arrow[dr] & & \\ x \arrow[r] & \boxed{E} \arrow[u] \arrow[d] & \boxed{D} \arrow[r] & \tilde{x} \\ & \Sigma \arrow[ur] & & \end{tikzcd}$$

* GAN

- $x \sim p$
- p is unknown,
- so we use some ϕ and sample from it.

↳ $\boxed{D} \longrightarrow$ real or fake

Discriminator

- If D is accurate, then we would know when ϕ is close to p when D gives 'real' for generated samples.
- G is the generator, $G \longrightarrow \tilde{x}$
- G and D would be NNs

$$\begin{tikzcd} & & x \arrow[dr] & & \\ z \arrow[r] & \boxed{G} \arrow[r] & \tilde{x} \arrow[r] & \boxed{D} \arrow[r] & \text{score} \in [0,1] \end{tikzcd}$$

If score is high if D says real o/w score is low

- z is sampled from a simple dist.

$z \sim \mathbb{N}(0, I)$, or $U(a,b)$

- $G(z, \theta_g)$, $D(x, \theta_d)$

Need to find θ_g and θ_d

*- Find θ_d s.t D is acc

- Find θ_g s.t G is acc

/// D always penalizes output of G, so that G should always try to be better.

- Goal of D: $D(x)\uparrow$ and $D(G(z))\downarrow$
- Goal of G: $D(G(z))\uparrow$

/// $V(D,G) = E_{x \sim p_{data}}[\log D(x)] + E_{z \sim p_z}[\log(1 - D(G(z)))]$

2 players - D and G

$$\min_G \max_D V(D, G)$$

* Algo:-

For a number of iteration do

for k step do

- sample $\{z^1, \ldots, z^m\}$ from P_z
- sample $\{x^1, \ldots, x^m\}$ from P_{data}
- update discriminator $D(x; \theta_d)$

end for

- sample $\{z^1, \ldots, z^m\}$ from P_z
- update generator $G(z; \theta_g)$

end for

$$E_{x \sim p_{data}}[\log D(x)] = \int_{-\infty}^{\infty} \log D(x)\, p_{data}(x)\, dx \quad -①$$

$$E_{z \sim p_z}[\log(1 - D(G(z)))] = \int_{-\infty}^{\infty} \log(1 - D(G(z)))\, p_z(z)\, dz \quad -②$$

$z \sim p_z$

$\tilde{x} \sim p_g$ — represent $G(z)$

hence, $E_{\tilde{x} \sim p_g}[\log(1 - D(\tilde{x}))] = E_{z \sim p_z}[\log(1 - D(G(z)))]$

so,

$$V(D, G) = \int \log D(x)\, p_{data}(x)\, dx + \int \log(1 - D(\tilde{x}))\, p_g(\tilde{x})\, d\tilde{x}$$

max $a \log y + b \log(1-y)$, here $p = p_{data} \cup p_g$

D will attain max at $\frac{a}{a+b}$

$$D(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \qquad D \to 1/2 \text{ when } p_{data} = p_g$$

$$C(G) = \max_D V(G, D)$$

$$= E_{x \sim p_d}[\log D^*_G(x)] + \cancel{E_{z \sim p_z}[\ldots]}$$

$$+ E_{z \sim p_z}[\log(1 - D^*_G(x))]$$

$$= E_{x \sim p_d}[\log D^*_G(x)] + E_{\tilde{x} \sim p_g}[\log(1 - D^*_G(\tilde{x}))]$$

- $D^*_{(x)} = \frac{p_d(x)}{p_d(x) + p_g(x)}$ where $D_* = \max(\ldots)$

$$\Rightarrow C(G) = E_{x \sim p_d}\left[\log \frac{p_d(x)}{p_d(x) + p_g(x)}\right]$$

$$+ E_{\tilde{x} \sim p_g}\left[\log \frac{p_g(\tilde{x})}{p_d(\tilde{x}) + p_g(\tilde{x})}\right]$$

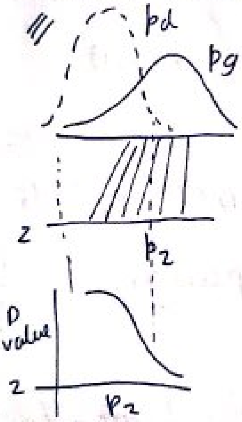
$$= KL\left[p_d \,||\, \frac{(p_d + p_g)}{2}\right] + KL\left[p_g \,||\, \frac{(p_d + p_g)}{2}\right]$$

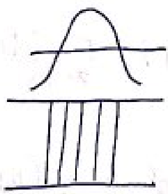
we want min $C(G)$

/// later: $D^*(x) = 1/2 \;\; \forall x$

$$C(G) = \log\frac{1}{2} + \log\frac{1}{2} = 2\log(1/2)$$

///





\\\ imp: Transpose convolution
= inv of ~~down~~ upsampling
down

- Radford et al. – architecture
- mode collapse
- unrolled GAN