Generative Models

Desired properties:
1. Easy to sample $x_{new} \sim p(x)$
2. Easy to evaluate likelihood $p(x)$
3. Optionally extract latent features

Classical distribution (eg gaussian) doesn't capture the complexity of real-world data. Mixtures (GMM) can but the number of components grows exponentially as the dimension increases.

1. Normalizing flows

NF can model complex distribution by applying a transformation on a simple distribution $f(z) = x$

Probability in output space should be same as input space by normalizing $\left|\frac{\partial z}{\partial x}\right|$

Change of variable

f valid and $x = f(z)$

$$p_2(x) = p_1(z) \left| \det \left(\frac{\partial z}{\partial x}\right) \right| = p_1(f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{\partial x}\right) \right|$$

volume: normalizes probs

$z = f^{-1}(x)$

Jacobian of f^{-1}

valid
- invertible: input and output same dimension, monotonic
- differentiable: (continuously differentiable) Jacobian exists at any point

Stacking:

$$p_k(x) = p_0(z_0) \prod_{i=1}^{k} \left| \det \left(\frac{\partial f_i^{-1}(z_i)}{\partial z_i}\right) \right|$$

Log prob (numerical stability):

$$\log p_k(x) = \log p_0(z) + \sum_{i=1}^{n} \log \left| \det \left(\frac{\partial f_i^{-1}(z_i)}{\partial z_i}\right) \right|$$

Reverse Parametrization

Goal: Evaluate $p_2(x)$ of any point x. x domain

$$g_\varphi(x) = f^{-1}(x) \Rightarrow p_2(x) = p_1(g_\varphi(x)) \left| \det \left(\frac{\partial g_\varphi(x)}{\partial x}\right) \right|$$ parametrize f^{-1}.

Once learned g_φ's, given a x we can go back to z_0. Then compute $p_0(z_0)$ and the dets to have $p_k(x)$

ϕ learned by maximizing likelihood with given dataset.

Forward Parametrization

Goal: sample from $p_2(x)$. Given z, sample $x \sim p_2(x)$ and also give $p_2(x)$

$f_\theta(z) = x \Rightarrow p_2(x) = p_1(z) \left| \det \left(\frac{\partial f_\theta(z)}{\partial z} \right) \right|^{-1}$ z domain

Once learned f_θ, sample from $z_0 \sim p_0(z_0)$ and compute $z_i = f_{\theta i}(z_{i-1})$ until k to have z_k. We can also have the density for this particular sample by computing intermediate $\left| \det \left(\frac{\partial f_{\theta i}(z_{i-1})}{\partial z_{i-1}} \right) \right|$

Det Jacobian

Diagonal: g is applied element-wise, $\det(J) = \prod \frac{\partial g_i(x_i)}{\partial x_i}$ $O(D)$

Triangular: $g(x) = \begin{bmatrix} g_1(x_1) \\ g_2(x_1, x_2) \\ \vdots \end{bmatrix}$, $\det(J) = \prod \frac{\partial g_i(x_i)}{\partial x_i}$ $O(D)$

Full: $g(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \end{bmatrix}$, decompose $J = LU$, $\det(J) = \det(L)\det(U)$ $O(D^3)$

2. Variational Inference

Latent Variable Model

$p_\theta(x)$ where x high dim but it can be described with latent z. That is $p(x|z)$ simple but $p(x)$ complex.

$p_\theta(x) = \int p_\theta(x, z)\, dz = \int p_\theta(x|z)\, p_\theta(z) = E_{z \sim p(z)}[p_\theta(x|z)]$.

Like in GMM: 1. $z \sim p_\theta(z)$
2. $x \sim p_\theta(x|z)$

Tasks:

Inference

Given a sample x, find posterior distribution over z:

$p_\theta(z|x) = \frac{p_\theta(x|z)\, p_\theta(z)}{p_\theta(x)}$

Learning

Given dataset X, find θ that maximizes log likelihood

$$\max_\theta \log p_\theta(x) = \max_\theta \frac{1}{N} \sum^N \log p_\theta(x_i)$$

↳ usually intractable (integral), can't compute gradients

special case: NF. Tractable with reverse parametrization. But somewhat constrained.

ELBO

We can define a lower bound g of f (or a collection of lower bounds) and try to maximize it.

$$\max_\theta f(\theta) \geq \max_\theta g(\theta).$$

for $p_\theta(x)$ we can also find a lower bound:

$$\log p_\theta(x) = \underbrace{E_{z\sim q(z)}\left[\log \frac{p_\theta(x,z)}{q(z)}\right]}_{\text{ELBO} = L(\theta,q)} + KL(q(z) \| p_\theta(z|x))$$

which is tight if $KL(\cdot) = 0 \Leftrightarrow q(z) = p_\theta(z|x)$

EM is an alternating optimization based on this property:

- E step: set $q(z) = p_\theta(z|x)$, with fixed θ from last iteration
- M-step: $\theta^{new} = \arg\max_\theta L(\theta, q)$, with q fixed

Optimizing ELBO

parametrize $q(z) \to q_\phi(z)$ to be able to optimize it.

$\max_{\theta,\phi} L(\theta,\phi)$: need to compute $\nabla_\theta L(\theta,\phi)$ and $\nabla_\phi L(\theta,\phi)$

$\nabla_\theta L(\theta,\phi)$

Assuming ϕ known and fixed: $E_{z\sim q_\phi(z)}[\log p_\theta(x,z)] = \frac{1}{S}\sum_i^S \log p_\theta(z_i)$. $z_i \sim q_\phi(z)$

Expectation is approximated with Monte Carlo

$\nabla_\theta \mathbb{E}_{z\sim q_\phi(z)}[f_\theta(z)] = \nabla_\theta \int q_\phi(z) f_\theta(z)\,dz = \int q_\phi(z) \nabla_\theta f_\theta(z)\,dz = \mathbb{E}_{z\sim q}[\nabla_\theta f_\theta(z)] = \frac{1}{S}\sum_1^S \nabla_\theta f_\theta(z_i)$

$\nabla_\phi L(\theta,\phi)$

Assuming θ known and fixed: $\mathbb{E}_{z\sim q_\phi(z)}[h_\phi(z)] = \int q_\phi(z) h_\phi(z)\,dz$

$h_\phi(z) = \log p_\theta(x,z) - \log q_\phi(z)$

here we can't push gradient inside integral as in ∇_θ because the expectation depends on ϕ.

Reparametrization Trick: sampling from some base distribution that doesn't depend on ϕ and then apply transformation

$\Rightarrow \mathbb{E}_{z\sim q_\phi(z)}[h_\phi(z)] = \int q_\phi(z) h_\phi(z)\,dz = \int b(\epsilon) h_\phi(T(\epsilon,\phi))\,d\epsilon = \mathbb{E}_{\epsilon\sim b(\epsilon)}[h_\phi(T(\epsilon,\phi))]$.

Now as in ∇_θ we can push gradients inside and approximate expectation with MC.

Mean field assumption

When optimizing over the entire dataset X, to lower bound $p_\theta(X)$ we need to consider $q(z)$ $z\in\mathbb{R}^{N\times L}$

In practice we often simplify $q(z) = \prod q(z_i)$ which simplifies the ELBO and allows to approximate gradient with mini-batch for efficiency.

Implications:

- we can't capture dependencies between dimensions (not important if data iid)
- less expressive but easier to optimize and infere

3. Variational Autoencoder

$ELBO = \mathbb{E}_{z\sim q(z)}[\log p_\theta(x|z) + \log p_\theta(z) - \log q_\phi(z)]$

many choices for $p_\theta(x|z)$, $p_\theta(z)$, $q_\phi(z)$

Prior $p_\theta(z)$

$z\in\mathbb{R}^L$, pick simple distributions for prior.

$p(z) = N(z|0,I)$

Conditional likelihood $p_\theta(x|z)$

$z \in \mathbb{R}^L$, for every z we have to produce a different distribution. Then pick a parametric distribution whose parameters are produced by some function f_ψ that takes z as input.

e.g. $f_\psi : \mathbb{R}^L \to \mathbb{R}^D$

$p(x|z) = \mathcal{N}(x | \mu = f_\psi(z), I) \quad x \in \mathbb{R}^D$

f_ψ is the decoder in neural networks that converts latent z into params θ of $p_\theta(x|z)$

Variational distribution $q_\phi(z)$

Dataset X consist of N $x^{(i)} \in \mathbb{R}^D$ each corresponding to different latent $z^{(i)}$. With mean field assumption:

$$q_\phi(z^{(1)}, .., z^{(n)}) = \prod_i q_\phi(z^{(i)})$$

For simplicity $q_\phi^{(i)}(z^{(i)}) = \mathcal{N}(z^{(i)} \mid \mu^{(i)}, \Sigma^{(i)}) \quad \phi^{(i)} = \{\mu^{(i)}, \Sigma^{(i)}\}$.

The choice of $q_\phi(z)$ should be similar to $p_\theta(z)$ as it minimize ELBO (KL) when $q_\phi(z)$ is close to $p_\theta(z)$. Additionally, KL between normals have closed form solution.

Every $x^{(i)}$ should map to the optimal $\phi^{(i)}$. We train an encoder g_λ that finds the optimal mapping.

Summary

$L(\psi, \lambda) = E_{z \sim q_\phi(z)}[\log p_\theta(x|z)] - KL(q_\phi(z) \| p(z)) \quad \theta = f_\psi(z) \quad \phi = g_\lambda(x)$.

1. Compute $\phi = g_\lambda(x)$
2. Compute ML estimate (often done with a single sample):
   - draw $z' \sim q_\phi(z)$ reparametrization
   - Compute $\theta = f_\psi(z')$
   - Compute ELBO $L(\psi, \lambda) = \log p_\theta(x|z') - KL(q_\phi(z) \| p(z))$
3. Backprop gradients $(\nabla_\psi, \nabla_\lambda)$
4. Update weights

Once trained (got optimal ψ of decoder) we can generate samples:

- $z' \sim p(z) = \mathcal{N}(z | 0, I)$
- $x \sim p_\theta(x|z) = \mathcal{N}(x | f_\psi(z'), I)$

4. Generative Adversarial Network

Two competing network models:

Generator: takes noise sample $z \sim p(z)$ and trasform it to $x = f_\theta(z)$. More flexible than NF as $f_\theta(z)$ can be any function

Discriminator: distinguish generated data and real data

Problem: Sampling is easy but density (likelihood) evaluation is hard.

Density Estimation by Comparison

Test hypothesis that true data distribution $p^*(x)$ and generated data distribution $p_\theta(x)$ are equal

$$H_0: p^* = p_\theta \text{ vs } p^* \neq p_\theta \quad L(\theta, \phi)$$

Density Difference Approaches

Moment Matching

$p^* = p_\theta$ if their expectation of any statistical tests $s(x)$ equals. Approximated by MC.

eg. $s(x) = \begin{bmatrix} x \\ x^2 \\ e^x \\ \vdots \end{bmatrix}$ $E_{p^*}(x) = E_{p_\theta}(x)$? $E_{p^*}(x^2) = E_{p_\theta}(x^2)$? ...

Max Mean Discrepancy

Generalization of moment matching

Ratio-based Approaches

$r^*(x) = \frac{p^*(x)}{p_\theta(x)}$ but true $r^*(x)$ hard to compute, instead approximate by $r_\phi(x)$.

Class Probability Estimation

Estimate density ratio via classifier which can distinguish between real and fake data.

- $Y = 1 \Rightarrow$ true, $Y = 0 \Rightarrow$ fake $\Rightarrow p(x|Y=1) = p^*(x)$, $p(x|Y=0) = p_\theta(x)$.
- $p(Y=1) = \pi$

$$r^*(x) = \frac{p^*(x)}{p_\theta(x)} = \frac{p(x|Y=1)}{p(x|Y=0)} = \frac{\frac{p(Y=1|x)\,p(x)}{p(Y=1)}}{\frac{p(Y=0|x)\,p(x)}{p(Y=0)}} = \frac{p(Y=1|x)\,p(Y=0)}{p(Y=0|x)\,p(Y=1)} = \frac{p(Y=1|x)}{p(Y=0|x)} \frac{(1-\pi)}{\pi}$$

Discriminator $D_\phi(x) = p(Y=1|x)$. CE loss for learning:

$L_{\theta,\phi} = \mathbb{E}_{(x,y)\sim p(x,y)}[-y \log[D_\phi(x)] - (1-y)\log[1-D_\phi(x)]]$

1. Optimize discriminator:

$\phi^*(\theta) = \arg\min_\phi L_{\theta,\phi}$. wlog $\pi = 0.5 \Rightarrow r^*(x) \approx \frac{D_{\phi^*(\theta)}(x)}{1-D_{\phi^*(\theta)}(x)}$

Ratio loss $= \frac{L_{\theta,\phi}(x,1) + L_{\theta,\phi}(f_\theta(z),0)}{2}$

2. Optimize generator:

drive $p(Y=1|x) = p(Y=0|x)$ ie $r^*(x) = 1$

$\theta^* = \arg\max_\theta L_{\theta,\phi^*(\theta)}$. Generative loss $= L_{\theta,\phi^*}(f_\theta(z), \underset{\downarrow}{y=0})$

Minimax game (bilevel optimization) $\min_\phi \max_\theta \pi(-L_{\theta,\phi})$

alternating optimization ϕ θ

Robustness

ML models are optimized following simple metrics (CE, L2...) => prone to specifically handcrafted inputs (attacks)

It's a counterexample to the hypotheses that neural networks can learn meaningful representations capturing the semantics

Adversarial Examples

Perturbation set $P(x)$ which doesn't change the semantic of x (small perturbation) but changes the classification result.

$$\tilde{x} \in P(x) \Rightarrow f(x) = y \text{ and } f(\tilde{x}) \neq y$$

$$P_{\epsilon,p}(x) = \{\tilde{x} : \|\tilde{x} - x\|_p < \epsilon\}$$

Attack variants

Poisoning Attack

Modify training set so classifier has some desired property by the attacker after training

Targeted Attack

Attacker wants some sample classified as some specific class

Untargeted Attack

Attacker wants a sample misclassified as any class different than the correct one.

Construction of Adversarial Examples

$$\tilde{x}_x^* = \underset{\tilde{x} \in P(x)}{\operatorname{argmax}} \; l_{0/1}(f(\tilde{x}), y)$$

$l_{0/1}$ zero/one loss which could have undef gradient.

$$\downarrow$$

$$\tilde{x}_x^* = \underset{\tilde{x} \in P(x)}{\operatorname{argmax}} \; L(f(\tilde{x}), y)$$

Use CE loss as alternative

Projected Gradient Descent

After each gradient step update the sample and project it back to the valid domain defined by $P(x)$.

$x_{t+1} = \Pi(x_t + \eta \epsilon \nabla_x L(f(x_t), y))$ if $f(\tilde{x})$ not convex, cannot find global minimum

Fast Gradient-Sign Method

$\hat{x} = \Pi(x + \eta \cdot \text{sign}(\nabla_x L(f(x), y))$ when norm defined on the radius ϵ, setting $\eta = \epsilon$ yields valid perturbation with only one step

Alternative formulation

$\min_{\tilde{x}} D(x, \tilde{x})$ s.t. $l_{0/1}(f(\tilde{x}), y) > 0$. D = L_p dist between $x, \tilde{x}$

↓ Lagrange

$\min_{\tilde{x}} D(x, \tilde{x}) + \lambda \cdot L(\tilde{x}, y)$ find min perturbed sample which yields misclassification

Robust Training

Training process that aims to produce models that are robust to adversarial attacks.

Optimize robust loss (worst-case loss)

$$R_{rob} = \mathbb{E}_{(x,y) \in P_{data}} \left[\sup_{\tilde{x} \in B(x)} l(f(\tilde{x}), y) \right]$$

Adversarial training

Use adversarial examples as proxy for worst-case perturbations to train robust classifiers.

Perform SGD on R_{rob}. $\Rightarrow$ $\nabla_\theta R_{rob} = \mathbb{E}_{(x,y) \in P_{data}} [\underbrace{\nabla_\theta (\sup_{\tilde{x} \in B(x)} l(f_\theta(\tilde{x}), y))}_{\text{gradient of the worst case loss}}]$

Danskin's Theorem

$\Delta(\theta)$ set of $\tilde{x}$ where sup is obtained. If $|\Delta(\theta)| = 1$ then sup is differentiable:

$\nabla_\theta (\sup_{\tilde{x} \in B(x)} l(f_\theta(\tilde{x}), y)) = \nabla_\theta l(f_\theta(\tilde{x}_\theta^*), y)$ i.e. use adv. example as proxy

Training outline:
1. Sample $(x_i, y_i) \sim$ Pdata
2. Find $\tilde{x}_i$ with FGSM which has high loss $\ell(f_\theta(\tilde{x}_i), y_i)$
3. Update weights $\theta \leftarrow \theta - \eta \nabla_\theta \ell(f_\theta(\tilde{x}_i), y_i)$

Cons:
- 10x slow down if we use powerful attack on step 2.
- Lower accuracy on resulting model
- No theoretical guarantee (certificate)

Certifiable Robustness

Prove that the classifier's prediction doesn't change within a radius (adversarial-free)

Exact Verification

Algorithm that return YES $\Leftrightarrow$ no adversarial examples.
Designed for NN with ReLU activations.

However exact verification for ReLU activation is NP-complete. But still tractable if network size is small-medium

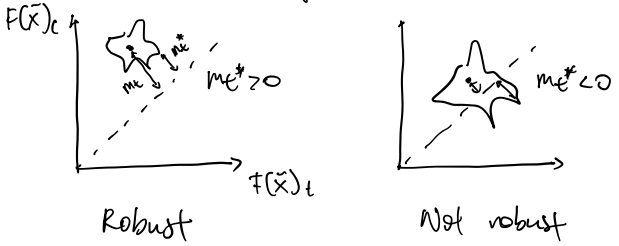
As MILP

Exact certification of ReLU network is possible with mixed integer linear programming. LP polynomial but MILP NP-complete.

$m_t = F(x)_{c^*} - F(x)_t$ $\quad F(x)_c$ class with highest prob (logit).

worst case margin: $m_t^* = \min_{\tilde{x}} F(\tilde{x})_{c^*} - F(\tilde{x})_t$ s.t. $\|\tilde{x} - x\|_p \leq \epsilon$

- $m_t^* > 0$: classification can't be changed from c^* to t
- if $\forall t \neq c$ we have $m_t^* > 0 \rightarrow$ certified robustness
- if $\exists t \neq c$ with $m_t^* < 0 \rightarrow$ exist adversarial example $\tilde{x} \in P(c)$



$m_t^* = \min_{\bar{x}} [\hat{x}^{(L)}]_{c^*} - [\hat{x}^{(L)}]_t$ $\quad \hat{x}^{(l)}$ pre-relu at layer l

s.t. $\|\bar{x} - x\|_p \leq \epsilon$

$y^{(0)} = \bar{x}$

$\hat{x}^{(l)} = W_l y^{(l-1)} + b_l \quad \forall l$

$y^{(l)} = \text{ReLU}(\hat{x}^{(l)}) \quad \forall l = 1 \dots L-1$

Encoding LP

$p=\infty \rightarrow x_i - \bar{x}_i \leq \epsilon \ \forall i$ $\quad$ $p=2$ straight forward

$\bar{x}_i - x_i \leq \epsilon \ \forall i$ $\quad$ $p=1$ quadratic programming

Encoding ReLU

Add binary variable a: $y_i \leq a_i x_i$ and $y_i \geq 0$ and $y_i \geq x_i$

But this adds an extra variable, making it a quadratic programming $(a_i x_i)$

If we have an upper and lower bound $[l, u]$ on the input x:

$y_i = \text{ReLU}(x) \iff$ 1) $y_i \leq x_i - l_i(1-a_i) \quad \forall i$ still linear and integer

2) $y_i \leq a_i \cdot u_i$

3) $y_i \geq x_i$

$y_i \geq 0$

$a_i \in \{0,1\}$

1) $a_i = 1 \Rightarrow y_i = x_i \quad a_i = 0 \Rightarrow y_i \leq x_i - l_i$

2) $a_i = 1 \Rightarrow y_i \leq u_i \quad a_i = 0 \Rightarrow y_i = 0$

3) 

To get $[l^{(l)}, u^{(l)}]$ on every layer with interval arithmetic:

$u^{(l)} = [W_l]_+ u^{(l-1)} - [W_l]_- l^{(l-1)} + b_l \qquad [W]_+ = \max(W, 0) \qquad h^{(0)} = x_i + \epsilon$

$l^{(l)} = [W_l]_+ l^{(l-1)} - [W_l]_- u^{(l-1)} + b_l \qquad [W]_- = \max(-W, 0) \qquad l^{(0)} = x_i - \epsilon$

$W = \begin{bmatrix} -1 & 4 \\ 3 & -2 \end{bmatrix} \quad W_+ = \begin{bmatrix} 0 & 4 \\ 3 & 0 \end{bmatrix} \quad W_- = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}$

We only need the case where $l^{(l)} \leq 0 \leq u^{(l)}$ as we don't know a_i beforehand. Otherwise it's $a_i = 1$ for $0 \leq l^{(l)} \leq u^{(l)}$ or $a_i = 0$ for $l^{(l)} \leq u^{(l)} \leq 0$.

Tighter the interval, faster the convergence

Convex Relaxation

Relax the question from exact certification to allow the answer MAYBE. Therefore instead of solving for true M_t^*, we solve for $\underline{M_t^*}$ which is a lower bound.

$0 \leq \underline{M_t^*} \leq M_t^*$ if $\underline{M_t^*} > 0 \rightarrow$ Adv free
else $\rightarrow$ Maybe

ReLU was the cause of NP-Complete, if we relax it, the algorithm becomes polynomial.

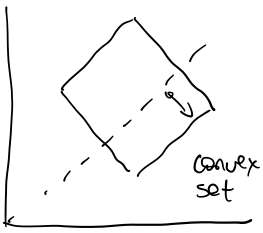
$y^{(l)} = \text{ReLU}(x^{(l)}) \longrightarrow y^{(l)} = \text{triangle}(x^{(l)})$

Convex relaxation

1) $y_i \geq 0$
$y_i \geq x$
$(u-l)\, y_i - u_i x_i \leq u_i l_i$

2) $y = x$ (for $l \geq 0$)

3) $y = 0$ (for $u \leq 0$)



MAYBE

For the case of MAYBE, we actually have $\bar{x} = \text{argmin}$ which we can feed it into the model and see output to prove non-robustness

The tightness of $[l, u]$ has an impact on the frequency of MAYBE

Robust Training

M_t actually depends on the weight θ. $\rightarrow M_t(\theta)$
We can use this for robust training

$$\ell(f_\theta(\bar{x}), y) = \max_{t \neq y} \max\Big(\underbrace{f_\theta(\bar{x})_t - f_\theta(\bar{x})_y}_{-M_t^*(\theta)}, 0\Big)$$

correctly classified $\rightarrow \ell = 0$
else $\rightarrow \ell = \underline{M_t^*}(\theta)$

However even with $-\underline{M_t^*}(\theta)$ we have to solve an LP and computing $\nabla\theta$ is expensive. But we can take advantage of the duality, each α in the dual is a lower bound for the primal problem and the gradient is easily computable. $\underline{M_t^{dual}}(\theta) = g(\alpha')$
(as it's just some analytical fnc)

$h_\theta(x') \geq h(x^*) = g(\alpha^*) \geq g(\alpha')$.

Lipschitz Continuity

A function is said to be k-lipschitz continuous if:

$$D_Y(f(x_1), f(x_2)) \leq k \cdot D_X(x_1, x_2) \quad \text{for any } x_1, x_2 \in X. \qquad k: \text{lipschitz constant}$$

The change of output space (logits) is less than k times change in input space.

Lipschitz in NN

For a stacking of functions (NN), we can have an upper bound for the lipschit constant.

$$k = L(F) \leq \prod_{\ell}^{L} L(f_\ell) \qquad f_\ell \text{ functions in the stack.}$$

For a fully connected layer $f(x) = Wx + b$:

$$L(f) = \sup_{a \neq 0} \frac{\|Wa\|_p}{\|a\|_p} \qquad a = x_1 - x_2 \qquad \text{(easily deduced from Lipschitz formula and FC)}$$

if $p=1 \rightarrow L(f)$ max L_1 between colums of W
$p=\infty \rightarrow$ " " rows "
$p=2 \rightarrow$ max singular value of W

(for conv layers it can be approximated efficiently)

Most activation functions (ReLU, sigmoid ...) have lipschitz constant of almost 1. ReLU has exactly 1.

Therefore, for the entire network:

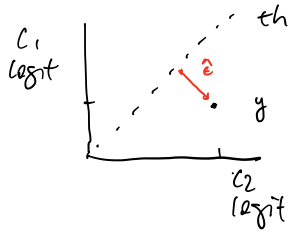
$$L(f) = \prod_{\ell}^{L} \|W_\ell\|_p$$

So by regularizing the norm of the weights we can enforce any desired constant

Certifying Robustness

With know lipschitz constant of the NN.
for an input x and logit $\hat{y} = F(x)$. We can compute maximum norm ϵ of perturbation δ: $\|\delta\|_p \leq \epsilon$ for which we can guarantee robustness.

1. Compute min perturbation norm $\hat{\epsilon} = \|\hat{\delta}\|_p$ in logit space to change the classification



2. Compute perturbation norm $\epsilon = \|\delta\|_p$ in input space based on Lipschitz constant.

$$\epsilon = \|x - \bar{x}\|_p \leq \frac{\hat{\epsilon}}{k} \iff k \cdot \|x - \bar{x}\|_p \leq \hat{\epsilon} \Rightarrow \|F(x) - f(\bar{x})\|_p \leq \hat{\epsilon}$$

Certifying that for any $\bar{x} \in P(x) = \{\bar{x} : \|x - \bar{x}\|_p \leq \frac{\hat{\epsilon}}{k}\}$ the robustness is guaranteed.

For smaller k, the network is more robust. However, it's a trade off with expressiveness.

Randomized Smoothing

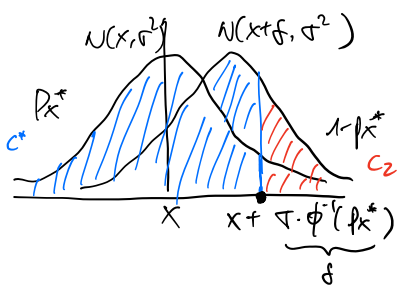
The base classifier is transformed into a smoothed one via adding gaussian noise $\epsilon \sim N(0, \sigma^2 I)$.

$f(x) = \arg\max_c F(x)_c$ the class prediction of base classifier.

$g(x)_c = \mathbb{E}_\epsilon(\mathbb{I}(f(x+\epsilon) = c)) = P_\epsilon(f(x+\epsilon) = c)$ logit of class c for smoothed classifier.

With $c^* = \arg\max_c g(x)_c$ and $p_x^* = g(x)_{c^*}$, we want to certify:

$$\arg\max_c g(x+\delta)_c = c^* \quad \forall \|\delta\|_2 \leq r$$



Worst case scenario when one class are concentrated at the sides.

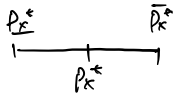
Φ^{-1} inverse gaussian CDF

For any $\|\delta\|_2 < \sigma \cdot \Phi^{-1}(p_x^*)$ the classifier is robust (prediction still the same). Larger σ leads to larger radius, but reduces p_x^* by introducing too much noise.

p_x^*

$A = \mathbb{1}[f(x+\epsilon) = c^*] \in \{0,1\}$ Bernoulli random variable of observing c^* with prob p_x^*

We can compute an one-sided confidence interval $(1-\alpha)$ (5%) for p_x^*.
If we sample n times and $A=1$ in m/n then $P(A=m) = \text{Binomial}(n, p_x^*)$.
To be conservative we take the lower bound $\underline{p_x^*}$ from the confidence interval for the Binomial



At practice we first use small set of samples to estimate c^* and then use large set to compute $\underline{p_x^*}$ with confidence interval

Robust Training

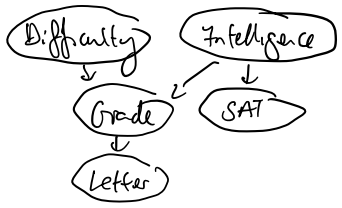
Randomized smoothing lends itself to robust training by adding data augmentation at training. The training set is augmented with random gaussian noise added to the original samples. This way, the model is more robust to noise.

Sequential Data

Bayesian Network

Naively to model sequential data $Pr(X_1, \dots, X_T)$ $X_t \in \{1, \dots, k\}$ we would need $k^T - 1$ parameters.

However, with Bayesian networks we can make the model more compact with assumptions about the data. The model can be represented in a directed graph where each node is a conditional distribution.



$Pr(X_1, \dots, X_M) = \prod_i^M Pr(X_i \mid Pa(X_i))$ Pa = parents

Then each $Pr(X_i \mid Pa(X_i))$ is parametrized by θ_i.

And the overall # variables is reduced from $k^T - 1$ to $\sum_i^T |\theta_i|$

$Pr(D, I, G, S, L) = P(D) \cdot P(I) \cdot P(G \mid D, I) \cdot P(S \mid I), P(L \mid G)$

This defines a generative process if we start sampling from the parent and following the graph get the joint probability

Autoregressive

AR(p): $X_t = c + \sum_i^p \varphi_i X_{t-i} + \varepsilon_t$ (X_{t-i}: lagged value)

φ_i params, c constant, $\varepsilon_t \sim N(0, \sigma)$.
The variable X_t are non-iid

$P(X_t \mid X_{t-1}, \dots X_{t-p}) \sim N(c + \sum \varphi_i X_{t-i}, \sigma)$ φ, c shared through time

$\mu(t) = E[X_t]$ (mean), $\gamma(t,i) = cov(X_t, X_{t-i})$ (autocov) $\rightarrow$ $\rho(t,i) = \frac{cov(X_t, X_{t-i})}{\sqrt{var(X_t)}\sqrt{var(X_{t-i})}}$ $\in [-1, 1]$ Pearson

Stationary

It's said to be stationary if:

1. $E[X_t] = E[X_{t-i}] = \mu$ $\forall t, i$ — mean is constant
2. $cov(X_t, X_{t-i}) = \gamma_i$ $\forall t, i$ — cov only depends on the lagged value
3. $\mathbb{E}[|X_t|^2] < \infty$ $\forall t$ — finite

mean and autocov can be estimated by averaging measures over time

$E(X_t) = \mu = \frac{c}{1-\sum_{i=1}^{p}\varphi_i}$ $\quad var(X_t) = \gamma_0 = \sum_{j=1}^{p}\varphi_j \gamma_j + \sigma^2$ $\quad cov(X_t, X_{t-i}) = \gamma_i = \sum_{j=1}^{p}\varphi_j \gamma_{i-j}$ $\quad \rho_i = \frac{\gamma_i}{\gamma_0}$

$\gamma_{-j} = \gamma_j$

AR(p) is stationary iff roots of characteristic polynomial $\phi(L) = 1 - \sum_{i=1}^{p}\varphi_i L^i$ lie outside the unit circle.

for $p=1 \rightarrow |\varphi_1| < 1$

$p=2 \rightarrow \varphi_1 + \varphi_2 < 1$, $\varphi_1 - \varphi_2 < 1$, $|\varphi_2| < 1$

...

The value doesn't explode (go to ∞) as the sequence goes due to the restrictions on φ_i

Learning

Opt 1. Parameters can be learned with least squares regression (normal equation):

$$\begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_p \end{bmatrix} = (X^T X)^{-1} X^T y \qquad X = \begin{bmatrix} x_{p-1} & \cdots & x_0 \\ x_p & \cdots & x_1 \end{bmatrix} \qquad Y = \begin{bmatrix} x_p \\ x_{p+1} \\ \vdots \end{bmatrix}$$

Opt 2. Use Yules walker equations:

1. Estimate moments $\gamma_0, \gamma_1, \dots \gamma_p$ (cov)
2. Inverse Yules-Walker matrix to estimate $\varphi_1, .., \varphi_p$
3. Use γ_0 equation to estimate σ

$$\begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_p \end{bmatrix} = \begin{bmatrix} \gamma_0 & \cdots & \gamma_{1-p} \\ \vdots & & \vdots \\ \gamma_{p-1} & \cdots & \gamma_0 \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \vdots \\ \varphi_p \end{bmatrix}$$

Yules-Walker matrix

Note: data spliting (test, train, val) should perserve time order

Markov Chain

Markov assumption:

$P(X_t | X_{t-1}, .. X_1) = P(X_t | X_{t-1})$ $\quad X_t$ discrete $\in \{1, .. K\}$

$$P(X_1 = i_1, X_1 = i_T) = P(X_1 = i_1) \prod_{t}^{T-1} P(X_{t+1} = i_{t+1} | X_t = i_t)$$

$$= \pi_{i_1} \cdot A_{i_1, i_2} \cdots A_{i_{T-1}, i_T} \qquad (A^{(t)} = A \text{ time homogenous assumption})$$

π $\quad t$ $\quad A$

$\pi \in \mathbb{R}^K$ prior probs $\quad A \in \mathbb{R}^{K \times K}$ transition matrices . #params: $(K-1) + (T-1)K(K-1)$

Therefore it can be interpreted as a state machine

$A_{ij}(n) = P(X_{t+n}=j \mid X_t=i)$ prob of getting from i to j in n steps
$= A^n$

$\pi_j(t) = Pr(X_t=j) = \pi A^{(t-1)}$ prob of getting to j in step t

Learning

We can use MLE to learn π and A (by counting).

$$A_{ij} = \frac{N(i,j)}{\sum_{j'} N(i,j')} \qquad \pi_k = \frac{L(k)}{\sum_{k'} L(k')}$$

Hidden Markov Chain

Markov model not expressive enough:

1. Markov assumption doesn't fit real world
2. Latent variable

HMM is a sequence composed of latent variables:

$$P(X_{t+1} \mid z_1 \dots z_T, X_1 \dots X_T) = P(X_{t+1} \mid z_{t+1})$$
$$P(z_{t+1} \mid z_1 \dots z_T) = P(z_{t+1} \mid z_t)$$

$$P(z_1=z_1, z_T=z_t, X_1=x_1 \dots) = P(z_1=z; \pi) \prod_t^{T-1} P(z_{t+1}=z_{t+1} \mid z_t; \theta^{(t+1)}) \prod_{t=1}^{T} P(X_t=x_t \mid z_t=z_t; \phi^{(t)})$$

$P(z_1=i) = \pi_i$
$P(z_{t+1}=j \mid z_t=i) = A_{ij}^{(t+1)}$
$P(X_{t+1} \mid z_{t+1}=i) = B_{ij}^{(t+1)}$

params: $k + k^2 + kk'$ (π, A, B) As variables can share params

Inference

Seek to find information about posterior $P(z_{1:T} \mid X_{1:T})$. 3 types:

- Filtering: computes $P(z_t \mid X_{1:t})$ incrementally as data streams (online). → Forward
- Smoothing: " based on past and future data (offline). → Backward
- MAP: computes $\underset{z_{1:T}}{\operatorname{argmax}} P(z_{1:T} \mid X_{1:T})$, mode of posterior. → Viterbi

Forward Algorithm

Incrementally compute $P(z_t | x_{1:t})$, online. With Bayes Rule

$$P(z_t = k | x_{1:t}) = \frac{P(z_t = k, x_{1:t})}{\sum_j^K P(z_t = j, x_{1:t})} = \frac{\alpha_t(k)}{\sum_j \alpha_t(j)} \qquad \alpha_t(k) = P(z_t = k, x_{1:t})$$

1. Compute α_1 (initialization)

$$\alpha_1(k) = P(z_1 = k) P(x_1 | z_1 = k) = \pi_k B_{k x_1}$$

2. Given α_t, compute α_{t+1} (recursion)

$$\alpha_{t+1}(k) = P(x_{t+1} | z_{t+1} = k) \sum_j^K P(z_{t+1} = k | z_t = j) P(z_t = j, x_{1:t}) = B_{k(x_{t+1})} \sum_j^h A_{jk} \alpha_t(j)$$

$$\alpha_{t+1} = B_{:(x_{t+1})} \odot (A' \alpha_t)$$

Backward Algorithm

Incrementally compute $P(z_t | x_{1:T})$, offline. With Bayes Rule:

$$P(z_t = k | x_{1:T}) = \frac{P(z_t = k, x_{1:t}) P(x_{t+1:T} | z_t = k)}{\sum_j P(z_t = j, x_{1:T})} \propto \alpha_t(k) \beta_t(k) \qquad \beta_t(k) = P(x_{t+1:T} | z_t = k)$$

1. Compute β_T (initialization)

$$\beta_t(k) = 1$$

2. Given β_{t+1} compute β_t (recursion)

$$\beta_t(j) = \sum_k^h A_{jk} B_{k x_{t+1}} \beta_{t+1}(k) \qquad \beta_t = A(B_{:(x_{t+1})} \odot \beta_{t+1})$$

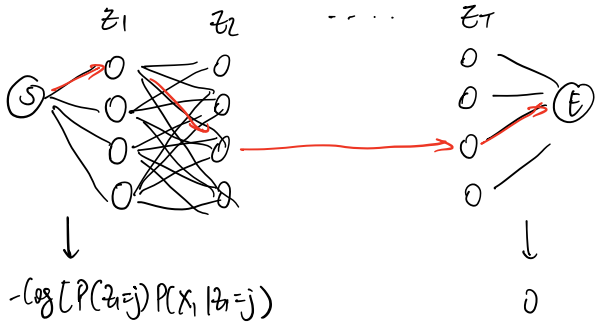
Viterbi

MAP inference in hmm. Given $x_{1:T}$, find $z_1, \ldots, z_T$.

$$\arg\max_z P(z_{1:T} | x_{1:T}) = \arg\max_z \log(P(z_1) P(x_1 | z_1)) + \sum \log(P(z_t | z_{t-1}) P(x_t | z_t)).$$

$P(z_{1:T}, x_{1:T})$

Each $\log[P(z_t | z_{t-1}) P(x_t | z_t)]$ depends on z_{t-1} and z_t. This can be viewed as a bipartite graph with edge weight $(i \to j) = -\log[P(z_t = j | z_{t-1} = i) P(x_t | z_t = j)]$



MAP inference = shortest-path

Learning

We want to learn $\theta = \{\pi, A, B\}$ from observed sequence x. By solving $\max_\theta \log P_\theta(x)$ via variational inference (ELBO optimization).

In a EM-fashion, as we can set $q(z) = p_\theta(z|x)$ for a fixed θ.

$$\mathbb{E}_{P(z_{1:T}|x_{1:T},\theta^{old})}[\log P(x_{1:T}, z_{1:T}, \theta)] = \sum_u P(z_1=u|x_{1:T},\theta^{old}) \log \pi_u +$$
$$\sum_{i,j}\sum_t P(z_t=i, z_{t+1}=j|x_{1:T},\theta^{old}) \log A_{ij} +$$
$$\sum_i \sum_t P(z_t=i|x_{1:T},\theta^{old}) \mathbb{1}(x_t=j) \log B_{ij}$$

Thanks to forward-backward algorithm, we have a closed-form solution for the blue terms. Thus, θ^{new} can easily be computed as the argmax

We don't pose mean field assumption as we want to keep dependencies introduced by the sequence

Continuous Data

X can also be continuous instead of discrete. The only difference is that the previous categorical distribution $P(x_{t+1}=x|z_{t+1}=i) = B_{ij}$ becomes gaussian $\mathcal{N}(x|\mu_i, \sigma_i^2 I)$

The algorithms for inference stays the same but with this change to gaussian.

For learning, we want μ_i, σ_i^2 instead of B_{ij}. The setting is very similar to GMM

Relation to other models:

X \ Z	NO	DISC	CONT
DISC	Markov	HMM	/
CONT	AR	Hmm	Kalman Filter

Neural Network Approaches

Use NN to solve word sequence tasks. But need meaningful representation for words.

Word2Vec

Naive one-hot encoding is highly dimensional, sparse and assumed independence between words.

However, we want similar words to have similar embedding vectors. To be aware of the context, a solution is to have a sliding window around the word $\{x_{i-l}, .. x_{i-1}, x_{i+1}, .., x_{i+l}\}$ and this will be the context. We can count the times 2 words appear in the same context and this defines the co-ocurrence matrix (symmetric) which has the property of similar words to have similar vectors, but still high-dim. → SVD dim-reduction.

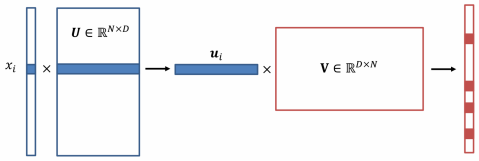
A different way to get word vector is with NN. Two approaches:

CBOW

predict words from its surrounding. Not good for rare words

Skip-gram

predict context from current word. Slower to train but better for rare words



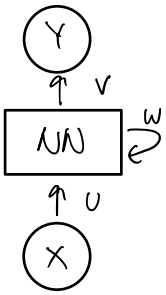
x_i one-hot u_i embedding ↳ prob of other words being context of x_i (softmaxed)

Train to find optimal params U and V. But can choose U=V for less params (also less expressiveness)

NN Architectures

For sequences we need layers that support input of dynamic lenght.

RNN



$P(y^{(t)} \mid h^{(t-1)}) = \text{softmax}(o^{(t)}) \qquad o^{(t)} = Vh^{(t)}$

Hidden state $h^{(t)} = \{x^{(1)}, \dots, x^{(t)}\} = \tanh(z^{(t)})$

$z^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b$

input at step $t-1$

params $\theta = \{U, W, V\}$ train with negative log likelihood loss.

Backprop:

$$\frac{\partial L}{\partial V} = \sum (\hat{y}^{(t)} - y^{(t)})(h^{(t)})^T \qquad \frac{\partial L}{\partial W} = \sum \text{diag}(1 - h^{(t)^2}) \frac{\partial L}{\partial h^{(t)}} (h^{(t-1)})^T$$

$$\frac{\partial L}{\partial U} = \sum \text{diag}(1 - h^{(t)^2}) \frac{\partial L}{\partial h^{(t)}} X^{(t)^T}$$

can't retain large dependencies → vanishing/exploding gradient

GRU

to alleviate vanishing gradient problem. We all information should be fully taken into account → gating mechanism. Adds gate (cell) that allows gradient to flow

$$z^{(t)} = \sigma(W_z[h^{(t-1)}, x^{(t)}])$$
$$r^{(t)} = \sigma(W_r[h^{(t-1)}, x^{(t)}])$$
$$\tilde{h}^{(t)} = \tanh(W[r^{(t)} \odot h^{(t-1)}, x^{(t)}])$$
$$h^{(t)} = (1 - z^{(t)}) \odot h^{(t-1)} + z^{(t)} \odot \tilde{h}^{(t)}$$

Gates

Simple RNN update – gives candidate state

How much to take from previous state vs. candidate state

LSTM

Improved version of GRU.

$$f^{(t)} = \sigma(W_f[h^{(t-1)}, x^{(t)}])$$
$$i^{(t)} = \sigma(W_i[h^{(t-1)}, x^{(t)}])$$
$$o^{(t)} = \sigma(W_o[h^{(t-1)}, x^{(t)}])$$
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tanh(W[h^{(t-1)}, x^{(t)}])$$
$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

Forget gate

Input gate

Output gate

Previous cell state

Simple RNN update – LSTM treats it as an input

Update hidden state (now the output) using a cell state

Convolutions

Sometimes we only need the information around the current word, ie a sliding window or context. In this case, convs comes handy.

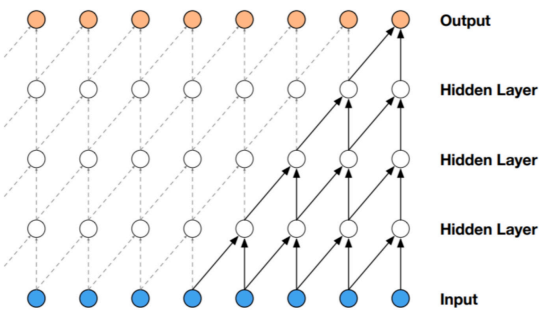
$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t-\tau) d\tau$$

Different design of kernel (filter) have different effects: smoothing, edge detection... often this is learned by training.

WaveNet

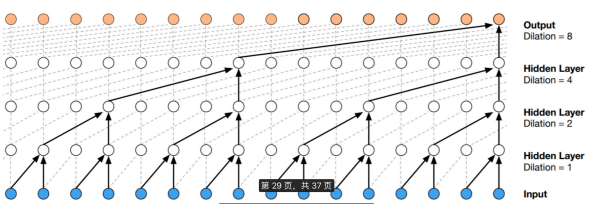
Architecture that uses 1-D conv to model speech. It uses spatial conv to ensure causality and increase receptive field.

Causal conv



only depends on the past

Dilated conv



skip some inputs to increase receptive field.

Transformer

Use attention mechanism to learn where to give attention. Faster than recurrent architectures as it's parallelizable. It's a stack of autoencoder using self-attention.

The weighting is computed by softmaxed dot prod of the embeddings.

$$Z = \underbrace{\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)}_{\text{weights}} V$$

Q: query (current embedding)
K: key (all embeddings)
V: values (embedding)
↝ same dims

Self-attention: current embedding also taken into account

Positional encoding: describe position/index of a word by a vector. The result is a positional encoding matrix. It is a way that transformers use to represent order.

Temporal Point Process

We are interested in event data which are discrete events in continuous time and irregular interval

TTP models the distribution of these events $t = \{t_1, .., t_N\}$ on the interval $[0, T]$ and gives the likelihood $p(\{t_1, .., t_N\})$

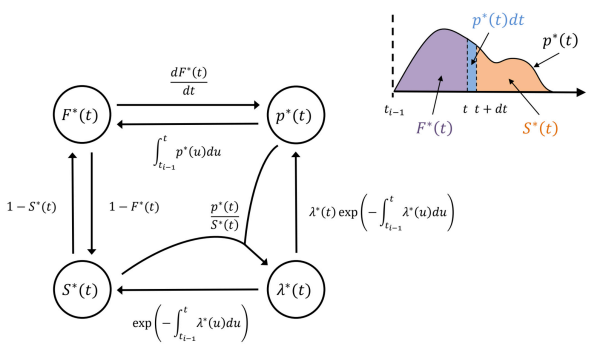
$p^*(t) = p(t|H(t))$ conditional density $\quad H(t) = \{t_j < t\}$

↳ prob of next event happening in $[t, t+dt]$ and no event $[t_{i-1}, t)$

$F^*(t) = \int_{t_{i-1}}^{t} p^*(u)du$ CDF. Prob that next event happens in $[t_{i-1}, t)$

$S^*(t) = 1 - F^*(t)$ Survival fnc. Prob next event doesn't happen before t

$\lambda^*(t)\,dt = \frac{p^*(t)dt}{S^*(t)}$ Conditional intensity. Prob event in $[t, t+dt)$ given no event in $[t_{i-1}, t)$. # event / time unit



$p(\{t_1, t_2, t_3\}) = p^*(t_1)\,p^*(t_2)\,p^*(t_3)\,S^*(T) = \lambda^*(t_1)\lambda^*(t_2)\,\lambda^*(t_3)\,\exp\left(-\int_0^T \lambda^*(u)du\right)$

Models

Define TTP in terms of $\lambda^*(t)$:

1. Easy pre-defined behavior
2. Easy to combine different TTP
3. Efficient sampling

Homogeneous Poisson Process

Simplest model with constant intensity $\lambda^*(t) = \mu$.

$p^*(t) = \mu \exp(-\mu(t - t_{i-1}))$ inter-event time follows exponential distribution

Easy to simulate HPP by sampling inter-event time $T \sim \exp(\mu)$

$T = F^{-1}(\mu) = \frac{1}{\mu} \log(1-u)$ $u \sim \text{Uni}(0,1)$

Inhomogeneous Poisson Process

Intensity changes over time $\lambda^*(t) = g(t) \geq 0$ dependent of the history.

Number of event in $[a,b] \subseteq [0,T]$ follows Poisson distribution:

$$N([a,b]) \sim \text{Poisson}\left(\underbrace{\int_a^b g(t)dt}_{\text{expected \#events} \;=\; \text{total intensity over } [a,b]}\right)$$

IPP with different intensities can be combined and it's still IPP

Simulating IPP is hard, but we can:

1. Find upper bound $\mu_0 \geq g(t)$ $\forall t$
2. Simulate by HPP with μ_0
3. keep each t_i with prob $g(t_i)/\mu_0$

Hawkes Process

$\lambda^*(t) = \mu + \alpha \sum_{t_j \in H(t)} k_w(t - t_j)$ self-exciting (burst) event occurrences

Learning

We can learn the (parametric) intensity function $\lambda_\theta^*(t)$ of the models by maximizing the log-likelihood of observed sequences.

$$\log P_\theta(\{t_1, \ldots, t_N\}) = \sum_{i=1}^{N} \log \lambda^*(t_i) - \int_0^T \lambda^*(u)\,du$$

HPP has closed form solution. Hawkes / IPP can be optimized with convex optimization or grad. descend

RNN

We could also model $p^*(t)$ directly instead of $\lambda^*(t)$ with RNN, as it gives more flexibility.

We feed the inter-event time τ_i as inputs and the hidden state h_i will keep track of the history. Then the output θ_i is a parameter for the distribution $p^*(t)$. $p^*(t) = p(t \mid H(t)) = p(t \mid h_i)$

Instead of t_i, p^* can be defined over τ_i. This way we only need an non-negative distribution (Exp, Gamma ...), a mixture or NF

Graphs

Graph $G=(V,E)$ V nodes $E \subseteq V \times V$ edges, can be represented by adjacency matrix $A \in \{0,1\}^{M \times |V|}$ (instead of binary could also be weights)

Properties

Real-word graph are not random (Erdös-Renyi model):

N nodes, each edge has same prob p of being added.

Instead it's better explained with a power law distribution $y(x) = Ax^{-r}$ over the degree of nodes. In log(degree) vs log(rank) it forms a line.

Clustering coefficient

Social network has a lot of triangles.

Clustering coeff $C = \frac{3 \, \#\text{triangles}}{\#\text{connected triplets}}$

Efficient triangle counting

$\#\text{triangles} = \frac{1}{6} \text{trace}(A^3) = \frac{1}{6} \sum_i \lambda_i^3$ λ_i eigenvals of A

λ follows power law, so we only need the top few. These can be computed with power iteration:

until convergence: $V = \frac{Xv}{\|Xv\|}$ v = eigenvector, initialized randomly

for the next eigenvec: $X = X - \lambda vv^T$ and repeat

Small word Phenomenon

$E \ll N^2$ $E = O(N^\alpha)$ $\alpha \approx 1.4$ real-word graph very sparse.
The average length of path to reach other person is 6

Characteristic path length

starting from a node, compute min path to reach other nodes. Take avg
Consider path length for all starting nodes. Take median of the averages

$$\underset{v \in V}{\text{median}} \left\{ \frac{1}{|V|} \sum_{j \in V} \text{len}(P_{min}(v, v_j)) \right\}$$

Average diameter

$\frac{1}{|V|}\sum_{v \in V} \frac{1}{|V|}\sum_{v_j \in V} \text{len}(p_{min}(v_i, v_j))$ mean istead of median

Hop-plot

$N_h(u)$ the number of nodes reachable from u via h hops
$N_h = \sum_{u \in V} N_h(u)$

plot N_h vs h

Effective diameter (Eccentricity)

$\min\{k \in \mathbb{N} \mid N_k \geq 0.9\,|V|^2\}$ min number of hops in which some fraction (90%) of nodes (connected) can reach each other

Generative Models

Erdös - Renyi

$p \in [0,1]$ $A_{ij} \sim Ber(p)$

Degree distribution

Prob of vertex having degree $k \Rightarrow p_k = \binom{N-1}{k} p^k (1-p)^{N-1-k} \approx \frac{z^k e^{-z}}{k!}$ $z = p(N-1)$
follow a Poisson distribution, but real-world power law

Diameter

$\log(N)/\log(z)$ z: avg degree of nodes.
Grows slowly with N, but real-world almost constant

Clustering coefficient

$p = z/(N-1)$ connection probability.
But real-world has community structures.

Planted Partition Model

Generalization of ER, nodes from same community have higher connectivity than outer nodes.

$Pr(A_{ij}=1 \mid z_i, z_j) \begin{cases} p & z_i = z_j \\ q & z_i \neq z_j \end{cases}$ $p > q$

Stochastic Block Model

Generalization of PPM with arbitrary num and size of communities, and varying edge densities

$\Pi = [\pi_1, .., \pi_k]$ community proportions $p(z_i = k) = \pi_k$

$\eta = \begin{bmatrix} & \end{bmatrix}$ edge prob between communities $P(A_{ij} | z_i, z_j) = Ber(\eta_{z_i z_j})$

Preferential Attachment Model

Generate the network continuously. Start with a subset of nodes and grow by adding new nodes and edges. Prob connecting nodes is proportional to the current degree of nodes (rich gets richer)

Initial Attractiveness

Implementation of PAM. When a new node is added, m new edges are added each (u,v) with prob $A_v = A + in\text{-}deg(v)$.

A constant for all nodes (the attractiveness)

Degree Distribution

power law with $\gamma = 2 + \frac{A}{m}$

Diameter

for $m \geq 2 \rightarrow O(\frac{\log N}{\log \log N})$ matches small world effect but still increasing

Average degree

Constant over time. But real-world increasing.

Configuration Model

predefine degree for each node (as stubs), randomly select stub pair and connect them.

Deep Models

Learn real-world graph properties from data.

NetGAN

Use GAN to learn distribution over random walks with LSTMs, initialize

states with $z \sim N(0, 1\alpha)$ and generate sequence of nodes. Each step's outputs are softmaxed and fed into categorical dist to pick the node.

Radon walk: $P(w_{j+1} = v_u \mid w_j) = \begin{cases} \frac{1}{d_{w_j}} & v_u \in N(w_j) \\ 0 \end{cases}$

Reparametrization trick to make the sampling differentiable for the categorical distr.

1. Draw Gumble noise $g_k = -\log(-\log(u))$ $u \sim Uni(0,1)$
2. Sum it to $\log \pi_k$ (softmaxed LSTM output) $g_k + \log \pi_k$
3. Take k with highest val. $v = \underset{k}{\text{argmax}} [g_k + \log \pi_k]$

Argmax from 3 still not differentiable. → Replace by softmax + temperature τ

$v \approx v^* = \nabla\left(\frac{\log \pi + g}{\tau}\right)$ π, g vectors

Clustering

Find cluster by graph data instead of attributes. Vertices in same cluster are connected by many edges and few for different cluster.
Can be categorized in overlapping or non-overlapping

But NP-hard problem if formulated as optimization over some objective function $Q(C)$, st. $C_1 \cup .. \cup C_n = V$ and $C_i \cap C_j = \emptyset$ (discrete)

Spectral Clustering

Cuts

Minimize edge/weights between clusters → min cut (Ford Fulkerson in poly).
But min cut tends to cut small vertex sets → only consider inter-cluster edges.

Cut: $\min_{C_1,..,C_n} \sum_{i}^{n} cut(C_i, V \setminus C_i)$

$cut(C_1, C_2) = \sum_{\substack{v_1 \in C_1 \\ v_2 \in C_2}} w(v_1, v_2)$

Ratio cut: $\min_{C_1,..,C_n} \sum_{i}^{n} \frac{cut(C_i, V \setminus C_i)}{|C_i|}$

Normalized ratio cut: $\min_{C_1,..,C_n} \sum_{i}^{n} \frac{cut(C_i, V \setminus C_i)}{vol(C_i)}$ $\quad vol(C_i) = \sum_{v \in C_i} deg(v)$

Finding optimal for various cuts is NP-hard

Graph Laplacian

$L = D - W$ $\quad D$ = degree matrix $\quad W$ = adjacency (weighted) matrix

$L_{uv} = \begin{cases} -W_{uv} & \text{if } (u,v) \in E \\ deg(v) & \text{if } u = v \\ 0 & \text{else} \end{cases}$ for any vector f: $f^T L f = \frac{1}{2} \sum_{(u,v) \in E} W_{uv}(f_u - f_v)^2$

- L is symmetric and PSD. With n non-negative eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq .. \lambda_n$

- Laplacian is additive $L_{G\cup H} = L_G + L_H$ G, H graphs on same vertex set
- Laplacian of graph is sum of laplacian of edges. $L_G = \sum_{(u,v)\in E} L_{(u,v)}$
- The number of eigenvalues $=0$ correspond to the number of connected components.
- Algebraic connectivity of the graph is $\lambda_2(L)$. λ_2 big $\rightarrow$ graph well connected

The spectrum of L encodes useful information. However, different graphs can have same spectrum (co-spectral graphs)

Spectral Clustering

With $f_{c_1}[i] = \begin{cases} 1 & v_i \in C_1 \\ -1 & \text{else} \end{cases}$ we can get min cut by minimizing $f_{c_1}^T L f_{c_1}$.
But min cut is not ideal. $\rightarrow$ ratio cut

For $k=2$

$$f_{c_1}[i] = \begin{cases} +\sqrt{\frac{|C_1|}{|\bar{C}_1|}} & \text{if } v_i \in C_1 \\ -\sqrt{\frac{|\bar{C}_1|}{|C_1|}} & \text{else} \end{cases}$$

1. $\sum f_{c_1}[i] = 0 \rightarrow f_{c_1} \perp \vec{1}$
2. $f_{c_1}^T f_{c_1} = \|f_{c_1}\|_2^2 = |V| \rightarrow$ constant norm
3. $f_{c_1}^T L f_{c_1} = |V| \cdot$ ratio cut

Need to drop discretness of f_{c_1} (relaxation) to make the optimization viable.

$$\min_{f_{c_1} \in \mathbb{R}^{|V|}} \{f_{c_1}^T L f_{c_1}\} \quad \text{s.t. } f_{c_1} \perp \vec{1} \text{ and } \|f_{c_1}\| = \sqrt{|V|}$$

$\lambda_1(L)$ is always 0 with eigenvect $\vec{1}$. $\rightarrow$ The second smallest eigenvect is f_{c_1}. And the sign of the elements in f_{c_1} tells the cluster. Alternatively, k-means could be performed on these values.

For $k \geq 2$

$$h_c[i] = \begin{cases} \frac{1}{\sqrt{|C|}} & \text{if } v_i \in C \\ 0 & \text{else} \end{cases} \qquad H = [h_{c_1}; h_{c_2}; \dots] \quad h_{c_i} \text{ as colums}$$

1. $H^T H = I$ (orthonormal)
2. $h_{c_i}^T \cdot L \cdot h_{c_i} = \frac{\text{cut}(C_i, V \setminus C_i)}{|C_i|}$, $h_{c_i}^T L h_{c_i} = (H^T L H)_{ii}$

$\Rightarrow$ Ratio Cut $(C_1, \dots, C_k)$ = trace $(H^T L H)$

$\min_{H \in \mathbb{R}^{V \times k}}$ trace $(H^T L H)$ s.t. $H^T H = I$. optimal H = first k smallest eigenvect of L.

$$H = \begin{bmatrix} \uparrow & & \uparrow \\ x_1 & \cdots & x_k \\ \downarrow & & \downarrow \end{bmatrix} \begin{matrix} v_1 \\ \vdots \\ v_n \end{matrix}$$

v_1 doesn't cut, as it's constant anyways

The rows gives the embedding vector for each node. Performing k-means on these gives the clustering result.

Remarks

Could change $L \to L_{sym} = D^{-1/2} L D^{-1/2}$ for normalized Laplacian.
In general spectral clustering gives good result although no guarantee to optimal due to relaxation.

For numerical data, can build similarity matrix (based on some similarity function) instead of L.

Probabilistic Approaches

Perform inference on generative models to find cluster assignment. Advantages: noise handling, capture uncertainty and generation of new data.

Planted Partition Model

$$P(A|z) = \prod_{i<j} \underbrace{\left[p^{A_{ij}}(1-p)^{1-A_{ij}}\right]^{\mathbb{1}(z_i=z_j)}}_{\text{same cluster}} \underbrace{\left[q^{A_{ij}}(1-q)^{1-A_{ij}}\right]^{\mathbb{1}(z_i \neq z_j)}}_{\text{different cluster}}$$ Likelihood function

Inference: $z^* = \underset{z \in \{-1,1\}^N}{\operatorname{argmax}} P_\theta(A|z)$ MLE.

No closed form solution. But if we assume $|C_1| = |C_2|$ then:

$$P(A|z) \propto \left(\frac{(1-p)q}{(1-q)p}\right)^{E_{cut}(z)} \qquad E_{cut}(z) = \#\text{ inter-cluster edges}$$

Maximizing $P(A|z) \iff$ minimizing $E_{cut}(z) \iff$ spectral clustering

Stochastic Block Model

Inference:

$$P(z|A,\eta,\pi) = \frac{P(A|z,\eta,\pi)\,P(z|\pi)}{P(A|\eta,\pi)}$$ But $P(A|\eta,\pi)$ intractable $\to$ variational inference. $\approx q(z|\psi)$

Learning (η, π):

If z, A observed $\Rightarrow$ simple counting $\quad \pi_k = \frac{N_k}{N} \quad \eta_{uv} = \frac{\#\text{observed edges}}{\#\text{possible edges}} = \frac{\sum \mathbb{1}(z_i=u)\mathbb{1}(z_j=v)}{P_{uv}}$

$$P_{uv} = \begin{cases} \binom{N_u}{2} & u=v \\ N_u N_v & u \neq v \end{cases}$$

If only A observed: optimize ELBO wrt $q(z)$
and with fixed z optimize η, π

Node Embedding

Raw graph data not suitable for standard ML models → learn function $\phi: V \to \mathbb{R}^d$ map discrete nodes to vector

Applications: clustering, semi-supervised classification, link prediction.

Methods

Spectral clustering

As seen before, the matrix built with the top k eigenvectors of L gives the embedding of the nodes preserving cluster structure. Spectral embedding

DeepWalk

Train word2vec (graph ≈ NLP) by sampling many random walks from the graph. Each random walk can be treated as a sentence in NLP to train the network. Nodes close to each other in the graph have similar embedding

Graph2Gauss

Learn mapping from nodes to a gaussian distribution $f_\theta(v) \to \mathcal{N}(\mu_\theta(v), \Sigma_\theta(v))$.
Loss: $\forall u \sum_{(v,w)} e^{E_{uv}} + e^{-E_{uw}}$ $E_{uv} = KL(f_\theta(u) \| f_\theta(v)) \to$ ranking loss: v is closer to u than w

Distance in embedding space relates to distance in graph, and variance tells the certainty

Graph Embedding

Aggregate all node embeddings from the graph via avg, sum... and thats the graph embedding which transforms a discrete graph to embedding.

Ranking

The web is a graph, a page is important if many important pages point to it.

PageRank

Each node's importance is the sum of its in-votes. Each node's vote is proportional to it's importance and divided by the number of votes made $r_j = \sum_{(i,j)} \frac{r_i}{d_i}$ d_i = out-degree

To get the importance of each node we can set the system of equations and solve it. To make the system unique we can force uniqueness by adding $\sum r_i = 1$.

To solve it efficiently:

- form the transition matrix M (similar to adjacency matrix but divided by out-degree).
- " rank vector r (the unknown importances)
- $r = Mr \longrightarrow r$ is eigenvector of M with eigenvalue = 1 (the largest one, since M column stochastic)

Markov Chain

A random walk in the web can be considered a the sequence generated in the Markov chain

score of page i = $r_i = \lim_{t\to\infty} P(X_t = i)$ ↗ transition matrix

$r = \lim_{t\to\infty} \pi(t)$ limiting distribution — if B row stochastic $\Rightarrow \lim_{t\to\infty} \pi(t) = \lim_{t\to\infty} \pi B^{(t-1)} \Rightarrow$ at $t=\infty$ it doesn't matter from where we start

$\pi(\infty) = \pi(\infty)B$ stationary distribution — if exists it's equal to rank vector r

It has to meet some properties:

- Irreducible: possible to get any state from any state
- Aperiodic: $Pr(X_{n'} = i \mid X_i = i) > 0 \quad \forall n : n' \geq n$ (able get back to the node of any time point) — the chain is aperiodic if all nodes aperiodic

Real-word web might not satisfy these. → Allow random teleport.
With prob β follow link, with prob $1-\beta$ jump random page

$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$ // $A = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$ in matrix notation

$\Rightarrow$ solution $r = Ar$ ($B = A^T$, $r = \pi B^\infty$)

Topic Specific PageRank

Generic popularity biased against specific topics. Therefore instead of the generic popularity we can measure the popularity within a topic.

$r = \beta M r + (1-\beta)\pi$ $\quad \sum \pi_i = 1$. Now teleport π_i is only restricted in a subset S.

solution: $r = (1-\beta)(I - \beta M)^{-1}\pi$ $\quad O(N^2)$

↳ only change π to match a different query

We are now measuring the rank against the subset S (could be the user query for ex.)

Trust Rank

topic-sensitive page rank with S as a set of trusted pages

Semi-Supervised Learning

Label Propagation

If's the scenario where only a set of nodes have known label.
We need to assume homophily (same labels nodes are likely connected)

Binary case

$V = S \cup U$ S = labeled set U = unlabeled set
$W \in \mathbb{R}^{|V| \times |V|}$ weighted adjacency matrix

$\min_y E(y) = \min_y \frac{1}{2} \sum_{ij} w_{ij}(y_i - y_j)^2 \equiv$ MAP inference $p(y_U | W, y_L) = \frac{1}{Z} \exp(-E(y))$

$Z = \sum_y \exp(-E(y))$ normalizing const.

$\Rightarrow \min_{y \in \{0,1\}^{|V|}} \underbrace{\frac{1}{2} \sum_{ij} w_{ij}(y_i - y_j)^2}_{\text{smoothness}}$ s.t. $\underbrace{y_i = \hat{y}_i \ \forall i}_{\text{matching}}$ $\hat{y}$ = gt labels

$\Rightarrow \min_{y \in \mathbb{R}^{|V|}} y^T L y$ s.t. $y_i = \hat{y}_i \ \forall i$ $L = D - W$

$\begin{bmatrix} \hat{y}_s \\ y_u \end{bmatrix}^T \begin{bmatrix} L_{ss} & L_{su} \\ L_{us} & L_{uu} \end{bmatrix} \begin{bmatrix} \hat{y}_s \\ y_u \end{bmatrix}$ $\nabla_{y_u} = 2 L_{us} \hat{y}_s + 2 L_{uu} y_u \Leftrightarrow y_u = -L_{uu}^{-1} L_{us} \hat{y}_s$

Generalization

k labels $\rightarrow$ y_{iu} $\begin{cases} 1 & i \in \text{class } k \\ 0 \end{cases}$ $E(y) = \sum_{i,j} w_{ij} (y_i - y_j)^T (y_i - y_j)$

Network effect $\rightarrow$ compatibility matrix H $E(y) = \sum_{i,j} w_{ij} (y_i - y_j)^T H (y_i - y_j)$

Non-graph data $\rightarrow$ construct graph with feature vector

LP is an instance of transductive learning, where we only try to predict the label of the set U, in contrast with inductive learning where we learn function mapping.

Semi-supervised learning is a more generic principle, the idea is that we use both labeled and unlabeled data for learning (exploit connexity as few data is labeled)

Graph Neural Network

Exploit graph structure as well as attributes by differentiable message passing

For each node: 1. Gather message from neighbors $m_v^{(u)} = \sum_{u \in N(v)} M(h_v^{(u-1)}, h_u^{(u-1)}, E_{uv})$
(hidden state, edge weight)

2. Update $h_v^{(u)} = U(h_v^{(u-1)}, m_v^{(u)})$

M, U mlps

Network weights are shared across the whole graph
Message passing steps (propagation) ≈ layers of standard network

For classification task, we perform softmax on the last layer (step) of the node $h_v^{(k)}$ (it has to be k-dim). Then the loss is just cross entropy:

$$\{W^{(k)}, B^{(k)}, p^{(k)}, b^{(k)}\}_{k=1..K} \min - \sum_{v \in V_L} \sum_{c \in C} y_{vc} \log p_{vc}$$

y_{vc} = one-hot gt, p_{vc} = softmaxed output

↳ labeled nodes

Relation with CNN

It's similar to performing convolution in images where each pixel (node) is connected to the neighbors.

$$X^{(t+1)} = X^t + AX^t = (A+1)X^t \quad \text{Conv in matrix form}$$

$$H^{(l+1)} = \sigma(\underbrace{\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}}_{\text{message passing}} \underbrace{H^l W^l}_{\text{feature transform}}) \quad \text{graph conv}$$

Multi-graph

Setup where were multiple graphs are input and we have one output per graph, i.e. molecule classification.

The idea is very similar to graph embedding. We get the hidden state of all nodes $H_{Gi} = [h_i^{(k)}, \dots, h_{|V|}^{(k)}]$ and aggregate via mlp $R(H_{Gi})$, the simplest case could be avg/sum.

Limitations of GNN

Expressiveness, Robustness (adversarial manipulation), Scalability (mpn expensive)

Overview

Isomorphism Problem

Best algorithm to determine if structure is the same for two graphs is NP.

Weisfeiler-Lehman Test (WL):
1. Label all nodes 1
2. Collect neighboring labels (concat)
3. Relabel each node by hashing collected labels
4. Compare result of each node.

If not 4 → not isomorphic
Else → might be or not

GNN is known to not solve the isomorphism problem. In fact, it's less powerful than WL (case of mean, max aggregation fnc). With more complex fnc it could get same power as WL.

Oversmoothing

Influence between the nodes is proportional to prob of visiting from a random walk of length k (k = upper length). The more central → more influence → smoother

If $k \to \infty$: the influence becomes propo. to stationary distribution in MC → Deep GNN unable to preserve local neighborhood info.

$r^{(t+1)} = AD^{-1} r^{(t)}$ Page-rank

$H^{(l+1)} = \sigma(\cdot)$ $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}\sigma(H^l)W^l$ GNN

if we push the non-linearity one level up, the similarity becomes higher.

To preserve neighborhood → as in pr, add teleport mechanism. → PPNP

Personalized Propagation of Neural Networks

1. Separate transformation and propagation.

$H_{i,:}^{(0)} = f_\theta(X_{i,:}) \in \mathbb{R}^K$ $\quad H^{(0)} \in \mathbb{R}^{N \times K}$

2. Introduce personalized teleportation

$$R = \alpha\underbrace{(I - (1-\alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})^{-1}}_{\in \mathbb{R}^{N\times N} \text{ weightings}}\underbrace{H^{(0)}}_{\in \mathbb{R}^{N\times K} \text{ embeddings}} \in \mathbb{R}^{N\times K}$$ exact sol.

3. $Z = \text{softmax}(R)$

$$H^{(l+1)} = (1-\alpha)\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^l + \alpha H^{(0)}$$

Approximate exact sol $Z \approx H^{(k)}$. Otherwise it's not scalable.
Now $\alpha H^{(0)}$ is fixed, we only need to iteratively compute $(1-\alpha)(\cdot)$ term, as it's sparse this is doable, propagation is done over $\mathbb{R}^{N\times K}$ matrix instead of high dim hidden state.

Robustness

Adversarial attack now can happen both on node attributes or on connectivity (structural attacks) X A

Adversarial Attack

Perturbations measured via non-convex L_0-norm.

$(A', X') \approx (A, X)$ hard to find measure of perturbation that do not change semantics

Unlabeled data also used for training → poisoning attack.

Nettack

Linearize $Z = \cancel{\text{softmax}}(\hat{A}\cancel{\sigma}(\hat{A}XW^{(1)})W^{(2)}) \rightarrow \hat{A}^2XW'$

For: 1. Structure perturbation $\max_{\hat{A}} L(\hat{A}^2 C)$ } pick optimal
2. Feature perturbation $\max_{X} L(C_1 X C_2)$ } perturbation at each step

Poisoning attack 1% correctness from 90% with clean data.

Certificates

Convex relaxation as before → answer YES / MAYBE

Certify attribute perturbations via LP
Certify structure perturbations via jointly constraint bilinear program
Certify PPNP model (structure perturbation) via quadratically constrained linear program. Polynomial if local budget (max x perturbations per node) but NP if global budget (max x perturbations overall)

Randomized Smoothing

We want to certify the smooth classifier: $\forall \tilde{x}\ \|\tilde{x} - x\|_0 \leq r : c^* = \arg\max_c g(x)_c$
→ L_0-norm radius certification.

But how to add noise to adjacency matrix? → n^2 Bernoulli random variables

$$P(\tilde{x}_i | x) = \begin{cases} p & \tilde{x}_i = 1 - x \quad \text{flip} \\ 1-p & \tilde{x}_i = x \quad \text{not flip} \end{cases}$$

But still problems: real-world graph are sparse ($m \ll n^2$) therefore picking right p almost impossible.

→ Sparsity-aware random sampling: each elem of A modelled by Bernoulli with data dependent probability.

$$P(\tilde{x}_i | x) = \begin{cases} p_d^{x_i} p_a^{1-x_i} & \tilde{x}_i = 1 - x \\ (1-p_d)^{x_i}(1-p_a)^{1-x_i} & \tilde{x}_i = x \end{cases}$$

If $x_i = 0$ flip with prob p_a else with prob p_b.

Now $g(x)_c = (\cdots) \overset{\text{divide}}{=} \sum_{\tilde{x} \text{ s.t. } f(\tilde{x}) = c} P(\tilde{x}|x) \overset{\text{iid}}{=} \sum_{\tilde{x}_1 \cdots \tilde{x}_{n^2}} \prod_i^{n^2} P(\tilde{x}_i | x)$ sum of probs. of $\tilde{x}$ that leads to class c.

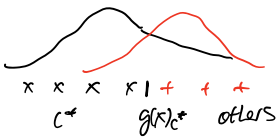
To find the radius we need to calculate the worst-case base classifier.
This can be done with linear programming:

$$\min_h \sum_i h_i P(\bar{x}^{(i)} | x') \quad \text{s.t.} \quad \sum_i h_i P(\bar{x}^{(i)} | x) = g(x) c^* \quad \text{and} \quad 0 \le h_i \le 1$$

h is the worst-case classifier and h_i indicates whether $h(\bar{x}^{(i)})$ outputs c^*, $g(x)c^*$ the budget. h is in the set H which produce the lowest prob.

The LP can be solved optimally by greedy approach:

1. Initialize all $h_i = 0$
2. Compute likelihood ratio $\eta_i = P(\bar{x}^i | x) / P(\bar{x}^i | x')$ and sort → indices $j_1, j_2 \dots$ st $\eta_{j_1} \ge \eta_{j_2} \ge$.
3. Set $h_{ju} = 1$ while $g(x)c^*$ not used up.



Now we are in the same situation as gaussian variables.

Technically we don't need to compute all h (as there are infinite), instead we can group them by some η_i