Autoencoders

Comparison to PCA: Autoencoders have an encodec scheme where encoder -> lower latent, decoder -> recon input. Like PCA where lower dim projection (principal coordinates) by using eigenvectors to capture the directions of most variance, recon the original input through a linear combo of the principal coordinates and their respective principal coordinates. Deep Autoencoders = pseudo-invertible nonlinear dimensionality reduction

What distribution should the latent follow? We can enforce a prior distribution by adding a probabilistic distance error (e.g. KLD or JSD). **Kullback-Leibler (KL) Divergence**: $D_{KL}(p||q) = \mathbb{E}_x\left[\log \frac{p(x)}{q(x)}\right]$ But this is not defined in non-overlapping support, i.e. when q(x) = 0 Not symmetric. The Jensen-Shannon (JSD) Divergence fixes this by essentially repeating the KL divergence twice to make it symmetric. Formula: $D_{JSD}(p||q) = \mathbb{E}_x\left[-\log \frac{p(x)}{q(x)}\right] + \mathbb{E}_x\left[-\log \frac{q(x)}{p(x)}\right]$ How do we calculate a Divergence metric when we don't have access to the data distribution? We're are only given access to a batch of samples at training time.

Adversarial Networks

One way we can enforce the latent distribution to follow the prior distribution is through adversarial networks by using a discriminator (adversary) to judge whether a generated output is real or fake. This training method actually allows us to provably measure JSD at an individual sample level.

Loss: $\min_{\theta_g,\theta_\phi} \max_\eta \mathbb{E}_{x\sim p_X}[\|x - \phi(\theta(x))\|^2] + \lambda \log(1-\eta(\phi(x))) + \mathbb{E}_{z\sim q_z}[\log(\eta(z))]$ where first term is MSE and other terms are JSD.

Proof deriving JSD:

$\int_z q_z(z)\log(\eta(z))dz + \int_z p_z(z)\log(1-\eta(z))dz \overset{L}{\Longrightarrow} \frac{\partial L}{\partial \eta} = \frac{q_z(z)}{\eta(z)} - \frac{p_z(z)}{1-\eta(z)}dz = 0 \Rightarrow \eta^* = \frac{q_z}{q_z+p_z}$, $\int_z q_z(z)\log\left(\frac{q_z}{q_z+p_z}\right)dz + \int_z p_z(z)\log\left(1-\frac{q_z}{q_z+p_z}\right)dz =$ $\int_z q_z(z)\log\left(\frac{q_z}{q_z+p_z}\right)dz + \int_z p_z(z)\log\left(\frac{p_z}{q_z+p_z}\right)dz =$ $\int_z q_z(z)\log\left(\frac{q_z}{q_z+p_z}\right)dz + \int_z p_z(z)\log\left(\frac{p_z}{q_z+p_z}\right)dz - 2\log(2) = 2JS(q_z,p_z) - 2\log(2)$ Extra term $-2\log(2)$ acts as a lower bound on the loss if p(z) = q(z) so if discriminator gives p = 0.5, meaning that $\eta(z) = 0.5$ from the initial distance formula, then we get $-\log(2) + -\log(2) = -2\log(2)$.

Mode Collapse: However, we may get mode collapse when the generator starts producing a limited variety of outputs, failing to capture the full diversity of the target distribution.

Wasserstein GAN Also known as Earth Mover's distance. Quantifies the amount of "work" moving a probability distribution to another. This better quantifies probabilistic distance than other metric, but I won't get into the details here. For the dual formulation for 1-Wasserstein distance: $W_1(p,q) = \max_{\|\eta\|_L\le 1}\left(\int_X \eta(x)p(x)dx - \int_Y \eta(y)q(y)dy\right)$ $X, Y \subseteq \mathbb{R}^d$ For the Wasserstein Adversarial Networks: $D(p_z,q_z) = \max_{\|\eta\|_L\le 1}\mathbb{E}_{z\sim q_z}[\ln(\eta(z))] - \mathbb{E}_{z'\sim p_z}[\ln(\eta(z'))]$ For the Loss function: $\min_{\theta_g,\theta_\phi}\max_{\|\eta\|_L\le 1}\mathbb{E}_{x\sim p_X}[\|x - \psi(\phi(x))\|^2] + \lambda\left(2\eta(\phi(x)) - \mathbb{E}_{z\sim q_z}[\ln(\eta(z))]\right)$

Variational Autoencoders

The prior $p(z)$ here is a simple Gaussian, which is mathematically convenient to work with, while the conditional output $p(x|z)$ is complex (image generated). Thus, we would ideally use MLE to estimate the parameters: $p_\theta(x) = \int p_\theta(x)p_\theta(x|z)dz$ However, marginalizing across all z is simply not practically possible, which makes this MLE intractable. Thus, the posterior density $p_\theta(z|x)$ is also intractable.

Solution: we also need to estimate an encoder $q_\phi(z|x)$ model. We can derive the loss terms based on the Evidence-based Lower Bound (ELBO) that we want to maximize: $\log p_\theta(x^{(i)}) = \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right]$ (Does not depend on z) $= \mathbb{E}_z\left[\log \frac{p_\theta(x^{(i)})}{p_\theta(z)}p_\theta(z|x^{(i)})\right]$ (Bayes' Rule) $= \mathbb{E}_z\left[\log p_\theta(x^{(i)}|z)p_\theta(z)\frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}\right]$ (Multiply by constant) $= \mathbb{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - \mathbb{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right] + \mathbb{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right]$ (Logarithms) $= \mathbb{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}\left(q_\phi(z|x^{(i)})\|p_\theta(z)\right) +$ positive KL term (intractable, ignore). The first 2 terms are the ELBO, term 1 is decoder distribution estimated through sampling w/ reparam trick (MSE), 2nd term is KLD loss.

The first term is the reconstruction loss and the second term is to make the approximate posterior close to the Gaussian prior.

Reparam Trick: To take a forward pass through the VAE, we have to sample from the latent distribution (Gaussian in this case). However, we can't backprop through such a stochastic operation, so the reparameterization trick is used. Since we estimate μ and σ through the encoder model, instead of drawing z from $N(\mu,\sigma^2)$, we can instead draw ϵ from N(0, 1) and then get z = $\mu + \sigma * \epsilon$, and thus gradient calculation isn't affected.

Generating Data: Once the VAE trained, sample data from Gaussian as the latent z and use the decoder model for output image. Since z is a multivariate Gaussian, the diagonal prior on z are independent latent variables that cause different factors of variation (think principal comp in PCA).

Denoising Autoencoders: It seems that $\psi(\phi(x)) - x$ would give us a vector that projects data from outside the manifold onto the manifold: Through the Bengio paper "What Regularized Auto-Encoders Learn from the Data-Generating Distribution", they find that: $\nabla_x \log(p(x)) = -\nabla E(x)$ In other words, the score function is the gradient of the log probability of the data! If we have the score function, we can sample from the distribution by setting an initial x drawn from an arbitrary prior distribution and then following the gradient of the data to eventually converge at a sample from p(x). This is called Langevin Dynamics: $x_{i+1} \leftarrow x_i + \epsilon\nabla \log p(x) + \sqrt{2\epsilon}\, z_i,\ i \in 0\dots K$ where the "learning rate" ϵ is sufficiently small and number of steps K is sufficiently large.

Diffusion Models

For the training algorithm: Repeat $x_0 \sim q(x_0)$ $t \sim$ uniform$\{1,\dots,T\}$ $\epsilon \sim \mathcal{N}(0,I)$ $x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon$ $\theta \leftarrow \theta + \eta\nabla_\theta\|\epsilon - \epsilon_\theta(x_t,t)\|^2$ Until convergence For the testing algorithm: $x_T \sim \mathcal{N}(x_0,I)$ For $t = T,\dots,1$ do $\epsilon \sim \mathcal{N}(0,I)$ if $t > 1$, else $\epsilon = 0$ $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}}\epsilon_\theta(x_t,t)\right) + \sqrt{\alpha_t}\epsilon$ Return x_0

Adversarial Attacks

White-Box: architecture and weights are known Black-Box: only access to input and output (e.g. API) Gray-Box: architecture known but not weights

We can attain an adversarial generally by adding some perturbation that maximizes the loss: $max_\delta\ loss(\theta, x+\delta, y)$ We want δ that's small w.r.t. l_p norm, Rotation/Translation, VGG feature perturbation or any other perturbation

White Box Attack Methods

Fast Gradient Sign Method (FGSM) Attack: Use pretrained classifier like ResNet50: $\tilde{y} = f(\theta, x)$ Find adversarial example that maximizes the loss: $\mathcal{L}(x', y) = \mathcal{L}(f(\theta, x'), y)$ Bounded perturbation s.t.: $||x' - x||_\infty \le \epsilon$, where ϵ is attack strength Optimal adversarial image: $x' = x + \epsilon sign(\nabla_x \mathcal{L}(x, y))$

Iterative FGSM Attack: Let m be number of iterations $x^{(m)} = x^{(m-1)} + \epsilon sign(\nabla_x \mathcal{L}(x^{(m-1)}, y))$ Both (I)FGSM are fix-perturbation attacks

(Iterative) Least Likely Attack: Similar to FSGM but y_{LL} is the least likely (LL) class predicted by the network on clean image x $x' = x - \epsilon sign(\nabla_x \mathcal{L}(x, y_{LL}))$ Strong attack as it emphasizes the least likely class

Projected Gradient Descent Attack: We can take a gradient step and then project it back to the feasible set Δ since the perturbed input may not lie on the data manifold (similar to denoising AutoEncoder logic) $\delta := \mathcal{P}_\Delta[\delta + \nabla_\delta Loss(x+\delta), y; \theta]$ For example, the projected gradient descent applied to l_∞ ball, repeat: $\delta := Clip_\epsilon[\delta + \alpha\nabla_\delta J(\delta)]$ Slower than FGSM but typically able to find better optima

Carlini and Wagner (CW) L2 Attack: zero-confidence attack for all $t \neq y$, find the adversarial image that will be classified by t as solving the problem: $min_\delta||\delta||_2^2$ subject to $f(x+\delta) = y, x+\delta \in [0,1]^n$ Finding the exact solution is difficult so we use relaxed version $min_\delta||\delta||_2^2 + c\cdot g(x+\delta)$ subject to $x+\delta \in [0,1]^n, c \ge 0$ Let $Z(x)$ be the NN activations before the logit output layer, also called the embeddings $g(x) = max\left(max_{i\neq t}(Z(x)_i - Z(x)_t), 0\right)$ Let $\delta = \frac{1}{2}(tanh(w)+1) - x$ With the following constrained optimization problem: $min_w||\frac{1}{2}(tanh(w)+1) - x||_2^2 + c, ReLU\{max_{i\neq t}Z\left(\left(\frac{1}{2}tanh(w)+1\right)_i\right) - Z\left(\frac{1}{2}\left(tanh(w)+1\right)_t\right)\}$ Powerful attack method that resists many defenses

Universal Adversarial Perturbation Attacks: A single perturbation on any image with high probability (e.g. 0.8+) Generalize well across different models

Single Pixel Attack: self-explanatory. Only modify one pixel

Poisoning Attacks: manipulating the training data itself rather than during inference Maintain accuracy but hamper generalization due to outliers through poisoning

Adversarial Training

Do training with adversarial samples **Outer Minimization**: $min_\theta \sum_{x,y\in S} max_{\delta\in\Delta} Loss(x+\delta, y; \theta)$

We can do gradient descent, but how do we find the gradient of the inner maximization term?

Danskin's Theorem: A fundamental result from optimization is: $\nabla_\theta max_{\delta\in\Delta} Loss(x+\delta, y; \theta) = \nabla_\theta Loss(x=\delta^*, y; \theta)$ where $\delta^* = max_{\delta\in\Delta} Loss(x+\delta, y; \theta)$ which means we can optimize through the max by finding its maximum value, but this only applies when max is found exactly.

Steps: 1. Select mini-batch B 2. For each sample, calculate the corresponding adversarial sample 3. Update parameters where learning rate is divided by batch size It's also common to mix adversarial samples and normal samples for stability reasons

Continual Learning

Data is not assumed to be IID and distribution will shift over time so need continual training.

Terminology: Transfer: from task A to B -> improved perf in B after learning A (gradient aligned, cosim >0) Interference: negative transfer (gradients cancel out, cosim < 0)

Rehearsal based solutions: make data close to IID by replaying from old task while learning new task, ideally we'd like to sub losses from each task together. Ideas: 1. learn coreset of data from task A and use that for rehearsal, 2. learn generative model for task A and generate samples for rehearsal **Elastic Weight Consolidation (Metaplastic Networks)**: latent distribution preservation. $\hat{L}_B(\theta) = L_B(\theta) + \lambda\mathbb{E}[d(f(x;\theta), f(x;\theta_A^*))]$. Assume that $\Delta\theta = \theta - \theta_A^*$ is small; we can write second order Taylor expansion of regularization term: $d(f(x;\theta), f(x;\theta_A^*)) = d(f_{\theta_A^*}, f_{\theta_A^*}) + \Delta\theta^T\nabla d(f_{\theta_A^*}, f_{\theta_A^*}) + \frac{1}{2}\Delta\theta^T Hd(f_{\theta_A^*}, f_{\theta_A^*})\Delta\theta$. First two terms become 0 so we have $\hat{L}_B(\theta) = L_B(\theta) + \lambda\mathbb{E}[Hd(f_{\theta_A^*}, f_{\theta_A^*})\Delta\theta]$. EWC uses KLD for d() when f is a pmf (otherwise use Wasserstein, Sliced-Cramer, or MMD) and assumes that H is diagonal so $\hat{L}_B(\theta) = L_B(\theta) + \lambda\sum_{i=1}^{nparam} H_{ii}([\theta]_i - [\theta_A^*]_i)^2$, where H_{ii} is importance weight of each parameter and summation is independent of data from task A. The second term can be seen as regularization, penalizing the change in "important" parameters or you could project the gradient into a subspace to ensure no interference with prior tasks. **Gradient Projection**: assumptions: activation subspace is low-rank; otherwise, projection into null space wouldn't lead to informative updates (can't learn new tasks efficiently). There isn't backward transfer. Solution: sparsity leads to low-rank activation subspace. Instead of training dense network with ReLU, train sparse with k-Winner take all activation (set not top-k activations to 0). Non-overlapping subnetworks are highly effective in overcoming forgetting. Keep track of activation freq for each neuron and down-modulate frequently activation neurons. Mecha-

nism: heterogenous dropout.

Meta-Learning

Meta-Learning: Loss function is a sum across task losses where each task loss is w.r.t test set. The training data for each task is called the "support" and the relevant test data is called the "query". To optimize this loss function, if it's differentiable then use GD, else use RL or evo approach. Iterating across all test set in tasks is called an episode. **Model-Agnostic Meta-Learning (MAML)**: find init s.t. a few steps of GD on NEW task can solve the task. Let operator $U_t^k(\phi)$ be k steps of GD on L_t with init at $\theta_0 = \phi$, so $U_t^k(\phi) = \theta_k$. The optimization in MAML is then $\min_\phi \mathbb{E}_t[L_t(U_t^k(\phi))]$, practically, inner loop uses train set A and outer loop uses val set B: $\min_\phi \mathbb{E}_{t,B}[L_t(U_{t,A}^k(\phi))]$. We can optimize ϕ with $\frac{\partial}{\partial\phi} L_{t,B}(U_{t,A}(\phi))) = DU_{t,A}(\phi)\nabla L_{t,B}(\tilde{\phi})$ where $\tilde{\phi} = U_{t,A}(\phi)$ and the first term is the Jacobian matrix of $U_{t,A}$. Let's look at k=1 GD step of ϕ : $U_{t,A}(\phi) = \phi - \epsilon\nabla L_{t,B}(\phi) \rightarrow DU_{t,a}(\phi) = I - \epsilon H(\phi)$. $g_{MAML} = \mathbb{E}_t\left[\frac{\partial}{\partial\phi} L_{t,B}(U_{t,A}(\phi))\right] = \mathbb{E}_t\left[(I - \epsilon H)\nabla L_t(U_{t,A}(\phi))\right]$. By a first order approximation, the first half inside the expectation is identity, so just $\mathbb{E}_t\left[\nabla L_t(U_{t,A}(\phi))\right]$, basically take the average gradient across all tasks. This 1-step approach using the expected gradient helps with computation efficiency and generalization. **Reptile**: Another MAML, For each iteration, sample task t, calculate $\tilde{\phi} = U_t^k(\phi)$ with optimizer, and update $\phi \leftarrow \phi + \epsilon(\tilde{\phi} - \phi)$. Uses first-order update (rather than 2nd order in MAML) so faster MAML is good bc Almost No Inner Loop due to feature reuse and optimizer can also be learned, but not preferred bc optimizer is specific to that task and not generalizable like SGD. You can also do this on NAS or data augs.