Universal Approximator Theorem: Let $\phi(\cdot)$ be a non-constant, bounded and monotonically increasing fn. $\forall \epsilon > 0$ and any continuous fn $\in \mathbb{R}^m$, there exists an integer N, real constants $v_i b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}$ where $i = 1, \dots N$ such that: $F(\vec{x}) = \sum_{i=1}^{N} v_i \phi(\vec{w}_i^T \vec{x} + b_i) \quad \text{with } |F(\vec{x}) - f(\vec{x})| < \epsilon$ where ϕ is a sensible activation function. Problems: ϵ can be very large in practice, making approximation less useful, and curse of dimensionality.

Shift Invariance: The unchanging response when the input is shifted. For classifier f and shift operator S_v, $f(\vec{x}) = f(S_v \vec{x})$ (no matter how the input is transformed, the output should remain constant) generalizes for unseen data.

Shift Equivariance: applying the shift operator after the function yields the same results as applying the function after the shift. i.e. $S_v \circ f(\vec{x}) = f(\vec{x}) \circ S_v$. It is about consistent transformation.

Translation Invariance: shift in input should have a predictable shift in hidden representation (location shouldn't matter)
Locality: we should not have to move far from location (i, j) to learn valuable information to asses what the area contains.

Fully connected net: every input feature n in an image influences ever neuron in the next layer: $n \times n$. **Sparsely connected net**: each neuron n is connected to a subset of neurons k: $k \times n$. **Weight sharing net**: weights are reused in a network

Curse of Dimensionality:
Sample Explosion: As the number of features or dimensions grows, the amount of data we need to generalize accurately grows exponentially To approximate a (Lipschitz) continuous function $f : \mathbb{R}^d \to \mathbb{R}$ with ϵ accuracy one needs $O(\epsilon^{-d})$ samples.
Sparseness: The more features we use, the more sparse the data becomes such that accurate estimation of the classifier's parameters (i.e. its decision boundaries) becomes more difficult, this sparseness is not uniformly distributed over the search space; the higher dimensions you have the higher probability that a data-point will sit in its own distinct corner in the hypercube.
Math: $V_{rind}^n = (1 - \alpha^n) V_{original} \Rightarrow \frac{V_{rind}}{V_{original}} = 1 - \alpha^n \Rightarrow \frac{d(1-\alpha^n)}{d\alpha} = -n\alpha^{n-1}$ $\iff d(1 - \alpha^n) = -n\alpha^{n-1} d\alpha$. this shows that the volume of the rind initially grows much faster, n times faster than the rate at which the object is being shrunk (when $\alpha = 1$ and $d\alpha < 0$ then $d(1 - \alpha^n) = n|d\alpha|$); In higher dims, small changes in distance lead to vast changes in vol.

Convolution: $(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \quad for f, g : [0, \infty] \to \mathbb{R}$
Correlation: $(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t + \tau) d\tau \quad for f, g : [0, \infty] \to \mathbb{R}$
commutative: $f * g = g * f$, associative $(f * g) * h = f * (g * h)$
distributive: $f_1 * (f_2 + f_3) = f_1 * f_2 + f_1 * f_3$

$$M = \left\lfloor \frac{M + 2 \times P - D \times (K-1) - 1}{S} + 1 \right\rfloor$$

standard CNNs are not naturally equivariant to rotations; Harmonic Networks/H-Nets; replacing kernel w/ 'circular harmonics'; rotation results in a proportionate rotation in the output.

Factorized conv: 2 3x3 convolutions can act as an approx for 5x5 conv trading expressiveness for efficiency. Inserting a non-linearity between the 3x3s lets it capture more complex features.
Separable conv: approximate a 5x5 conv with a 5x1 and 1x5 reducing params (as above). Lossy.

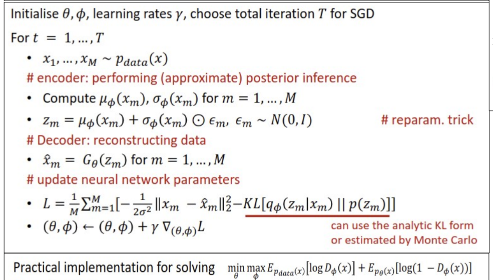
Pooling: smaller res, hierarchal features (concentrates abstract features), shift/deform Invariance. Break shift-equivariance by blurring sample to avoid the shifting pooling issue.

Approximate Deformation Invar: $||f(\vec{x}) - f(D_\tau \vec{x})|| \approx ||\nabla \tau||$ deform img τ=deform factor

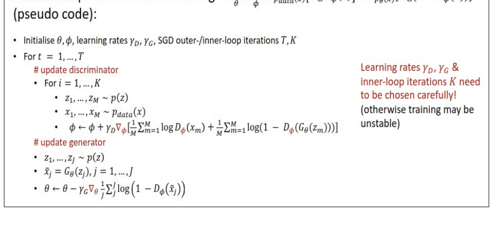
Activation Fns: introduce non-linearity for more complex data learning. Unbounded outputs lead to numerical instability in training. High values cause issues with gradients leading to problems like exploding gradients.

Linear: $f(x) = c \cdot x$ network behaves single-layered model. Regression.
Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$ more analogue continuous output than step, smooth gradient between (-2,2) rapid learning between there. Vanishing gradient problem. Good for n-classifiers.
Tanh: $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ stronger gradient, outputs are zero-centred on avg. Vanishing gradient problem, requires good normalization.
ReLU: $f(x) = \max(0, x)$ produces sparse activations, comp. Efficient (less dense activations), Dying ReLU problem often outputting 0.
Leaky ReLU: f(x) $= \begin{cases} x & for\ x \geq 0 \\ 0.01 \cdot x & for\ x < 0 \end{cases}$ like ReLU – dying problem
PreLU: as above only $0.01 = a$, adaptability during training, scale invar.
SoftPlus: $f(x) = \frac{1}{\beta} \cdot \log(1 + e^{\beta \cdot x})$ differentiable ReLU, if beta high, closer to ReLU, positive output only, non-linear.
ELU: CELU with $\frac{x}{\alpha} = x$ element-wise, can be -ve, avg activation pushes to 0 → helps converge faster.
CELU: $\max(0, x) + \min(0, \alpha \cdot (e^{\frac{x}{\alpha}} - 1))$ element-wise, when $\alpha \neq 1$ continuously differentiable
SELU: $scale \cdot (\max(0, x) + \min(0, \alpha \cdot (e^x - 1)))$ predefined scale and α for mean 0 and var 1, internal normalizaiton, robust (no dying unit)
GELU: $x \cdot \phi(x)$ cdf for gaussian, introduces probabilistic, regularization.
ReLU6: $\min(\max(0, x), 6)$ saturates activations,
LogSigmoid: $\log(\frac{1}{1+e^{-x}})$ element-wise, better for cost functions,
SoftMin: $\frac{e^{-x_i}}{\sum_j e^{-x_j}}$ rescale inputs to sum to 1, multi-dimensional non-linearity, resemble a probability distribution. Emphasises smallest values.
SoftMax: $softmax(z_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$ rescale to $\sum = 1$, multi-label class.
LogSoftmax: $\log(\frac{e^{x_i}}{\sum_j e^{x_j}})$ possible to simplify mathematical calculations and potentially improve the training process
Period activation: **SIREN:** $\Phi(\vec{x}) = \vec{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ \ldots \circ \phi_0)(\vec{x}) + \vec{b}_n$ $\vec{x}_i \mapsto \phi_i(\vec{x}_i) = \sin(\vec{W}_i \vec{x}_i + \vec{b}_i)$ deals with implicit representation involving finding a continuous function that represents sparse input data, such as images.

Loss: how well your network is doing, quantifying Δ(predicted outputs, ground truth). Backprob updates model parameters, aiming to min. error.
L2-norm: $\ell(x, y) = l_n = (x_n - y_n)^2$ regression.
L1-norm: $\ell(x, y) = l_n = |x_n - y_n|$ ↓sensitive to outliers, $\neg$ differentiable
Smooth L1: $\frac{1}{n}\sum_i z_i, \quad z_i = \begin{cases} 0.5(x_i - y_i)^2, & if |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, & otherwise \end{cases}$ for errors close to zero, it behaves like the L2 loss, else L1. Robust to outliers.
Negative Log-likelihood: $\ell(x, y) = \vec{\mathcal{L}} = \{l_1, \ldots, l_N\}^T, l_n = -w_{y_n} x_{n, y_n}$, $w_c = weight[c] \cdot 1\{c \neq ignore_{index}\} \ell(x, y) = \begin{cases} \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n}} l_n, & if\ reduction = mean \\ \sum_{n=1}^{N} l_n, & if\ reduction = sum \end{cases}$ element-wise, weights assign importance,
Cross-entropy: $loss(x, class) = -\log(\frac{e^{x[class]}}{\sum_j e^{x[j]}}) = -x[class] + \log(\sum_j e^{x[j]})$ n-classification, weighted classes optional
Binary cross-entropy: $\ell(x, y) = \mathcal{L} = \{l_1, \ldots, l_N\}^T$ $l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$, x_n predicted positive, y_n ground truth (0,1), w_n weight. Binary/Multi classification
Kullback-Libeler $\ell(x, y) = \mathcal{L} = \{l_1, \ldots, l_N\}^T, \quad l_n = y_n \cdot (\log y_n - x_n)$ suitable when target is 1-hot, suffers with numerical stability issues
Margin Ranking Loss/Ranking Losses/Contrastive Loss: $loss(x, y) = \max(0, -y \cdot (x_1 - x_2) + margin)$ useful to push classes as far away as possible
Triplet Margin Loss: $l_n(x_a, x_p, x_n) = \max(0, m + |f(x_a) - f(x_p)| - |f(x_a) - f(x_n)|$ make samples from same classes close and different classes far away e.g. Siamese Networks
Cosine Embedding Loss loss(x,y)$= \begin{cases} a - cos(x_1, x_2), & if y = 1 \\ \max(0, cos(x_1, x_2) - margin), & if y = -1 \end{cases}$
Measure whether two inputs are similar or dissimilar, 1 sim (tries to minimize the angle between the vectors), -1 not sim (aims to ensure that the cosine similarity is smaller than a specified margin).



```
Initialise $\theta, \phi$, learning rates $\gamma$, choose total iteration $T$ for SGD
For $t = 1, \dots, T$
    • $x_1, \dots, x_M \sim p_{data}(x)$
    # encoder: performing (approximate) posterior inference
    • Compute $\mu_\phi(x_m), \sigma_\phi(x_m)$ for $m = 1, \dots, M$
    • $z_m = \mu_\phi(x_m) + \sigma_\phi(x_m) \odot \epsilon_m,\ \epsilon_m \sim N(0, I)$        # reparam. trick
    # Decoder: reconstructing data
    • $\hat{x}_m = G_\theta(z_m)$ for $m = 1, \dots, M$
    # update neural network parameters
    • $L = \frac{1}{M}\sum_{m=1}^{M}[-\frac{1}{2\sigma^2}\|x_m - \hat{x}_m\|_2^2 - \underline{KL[q_\phi(z_m|x_m) \,||\, p(z_m)]}]$
    • $(\theta, \phi) \leftarrow (\theta, \phi) + \gamma \nabla_{(\theta,\phi)} L$
                                                  can use the analytic KL form
                                                  or estimated by Monte Carlo
```



Practical implementation for solving $\min_\theta \max_\phi E_{p_{data}(x)}[\log D_\phi(x)] + E_{p_\theta(x)}[\log(1 - D_\phi(x))]$ (pseudo code):

```
• Initialise $\theta, \phi$, learning rates $\gamma_D, \gamma_G$, SGD outer-/inner-loop iterations $T, K$
• For $t = 1, \dots, T$
    # update discriminator
    • For $i = 1, \dots, K$
        • $z_1, \dots, z_M \sim p(z)$
        • $x_1, \dots, x_M \sim p_{data}(x)$
        • $\phi \leftarrow \phi + \gamma_D \nabla_\phi[\frac{1}{M}\sum_{m=1}^{M}\log D_\phi(x_m) + \frac{1}{M}\sum_{m=1}^{M}\log(1 - D_\phi(G_\theta(z_m)))]$
    # update generator
    • $z_1, \dots, z_J \sim p(z)$
    • $\tilde{x}_j = G_\theta(z_j), j = 1, \dots, J$
    • $\theta \leftarrow \theta - \gamma_G \nabla_\theta \frac{1}{J}\sum_j^J \log(1 - D_\phi(\tilde{x}_j))$

Learning rates $\gamma_D, \gamma_G$ & inner-loop iterations $K$ need to be chosen carefully!
(otherwise training may be unstable)
```

LeNet: convolutions + avg pooling + fully connected last layer (small number of classes make this ok) performs object localization + recognition.
AlexNet: + dropout after 2 layers for robustness/regularization, ReLU, maxpool (more shift invariance, keep salient features, discard less useful), softmax for classification, increased kernel size, data augmentation, model ensemble, originally split into two streams due to hardware limitations.
VGG: bigger, didn't add more dense layers, didn't add more convolutional layers, but grouped layers into parametrized blocks. Using lots of narrow convolutions outperforms few wider ones, slower than others but performance is much better
Inception: introduces parallel dataflow, improves performance with width and depth. ↑size=↑params=↑overfitting=↑comp resource. Patch alignment issues fixed with 1x1, 3x3 (1-padding) and 5x5 (2-padding). Naive version also includes a 3x3 maxpool in parallel for additional benefit. Detailed approach includes 1x1 convolutions to compute reductions before expensive 3x3 and 5x5. Channel size: 1x1 = 64 ch, (1x1=96ch→ 3x3=128ch), (1x1=16ch→ 5x5=32ch), (3x3 maxpool→ 1x1=32ch) because big kernels already come with large parameter count. √ It has low param count and FLOPs : $\overset{fixed}{\boxed{k^2}} \times c_{in} \times c_{out} \times \overset{fixed}{\boxed{m_h \times m_w}}$ implies that $\overset{fixed}{\boxed{c_{in} \times m_h \times m_w}} \times \sum_{j \in paths}(k_j^2 \times c_{out,j})$ By varying no. chan and k we optimize network performance.
ResNet: parameterizes around an identity function instead of 0 function; you don't have to learn the identity function from scratch, allows for the addition of new layers without disrupting the outputs of previous layers. 3X3→Batchnorm→RelU → 3x3→BatchNorm→combine with identity/1x1 → ReLU.
DenseNet: each layer gets the feature maps from all the preceding layers (taylor expansion style), efficient and less parameters, alleviate the vanishing gradient problem by improving the gradient flow through the network, making optimization easier, scalable, since you can easily adjust the number of layers. Occasionally need to reduce resolution (transition layer)
SqueezeNet: uses attention, *Squeeze* global average pool, fed into a *Descriptor* (small fully-connected NN) capture which channels are more important given the current global context of the image. Use descriptors to *Excite* and scale the original channels *Re-weight channels* if certain channels need to emphasis certain features
Outcomes: Dense layers are computationally and memory intensive. Real-world problems with big input tensors and many classes will prohibit their use, 1x1 convolutions act like pixel-wise muli-layer perceptron,

BatchNorm: during GD, Each layer's learning destabilizes the next, causing a slow convergence process that takes a long time for all layers to adapt properly. BatchNorm normalizes the features within each mini-batch, thereby stabilizing the training process and speeding up convergence. $\mu_B = \frac{1}{|B|}\sum_{i \in B} x_i \quad \sigma_B^2 = \frac{1}{|B|}\sum_{i \in B}(x_i - \mu_B)^2 + \epsilon$ therefore $x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$ where γ, β are learnable parameters var and mean to scale and shift. BatchNorm is effectively acting as a form of regularization by introducing noise into the model. Insert after conv but before activation. Perform one batchnorm per channel, and in FCL one normalization for all. Not recommended for use with dropout.

U-Net: combination of 3x3 convs with ReLU x2 + maxpool 2x2 … up-conv 2x2 + conv 3x3 x 2 + skip connections + BatchNorm.
3D-UNet: H-DenseUNet: specific segmentation task Unet++: more powerful medical imaging+uses densely connected subnetworks nnU-Net auto

DataAugmentation: increases size and diversity of training set through artificial upsampling, e.g. random (flip, scale, rot, intensity/contrast var, cropping/padding, noise, affine/perspective transformations). Useful for anomaly detection, latent space measurement, input reconstruction,

Supervised: learn function $x \to y$ given $p_{data}(x, y)$, **Unsupervised:** learn a probabilistic model which describe hidden structure $p_\theta(x) = p_{data}(x)$
Generative Latent Models: $z \approx p_\theta(z), x \approx p_\theta(x|z) \to p_\theta(x) = \int p_\theta(x|z) p_\theta(z) dz$ where z is unobserved latent var and x is observed var.
PCA: explain the variability in data as k orthogonal directions that capture most of the variance in the data. **Probabilistic PCA:** $p(z) = \mathcal{N}(z; 0; I)$ $z \in \mathbb{R}^K, K < d$. $p_\theta(x|z) = \mathcal{N}(x; Wz, \theta^2 I), x \in \mathbb{R}^d$ and optimise $\theta = W \in \mathbb{R}^{d \times K}$ by training with max log-likelihood, W contains K eigenvectors
Clustering: $p_\theta(z) = Categorical(\pi), \pi = (\pi_1, \ldots, \pi_k), \pi_i = p_\theta(z = 1)$ and $\sum_{i=1}^{K} \pi_i = 1, p_\theta(x|z) = \mathcal{N}(x; \mu_z; \Sigma_z)$

$$\underbrace{p(\theta|x)}_{\text{posterior}} = \frac{\overbrace{p(x|\theta)}^{\text{likelihood}} \cdot \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(x)}_{\text{evidence}}}$$

Jensen's Inequality: if f is $\frac{d^2 y}{dx^2} \geq 0$ then $\forall p(x)$ $\mathbb{E}_{p(x)}[f(x)] \geq f(\mathbb{E}_{p(x)}[x])$

VAE: we want to minimise divergence between prediction and actual: $\theta^* = \arg\min D[p_{data}(x) || p_\theta(x)]$. Divergence is valid iff when D=0 → p=q and ≥ 0 otherwise. We set D = Kullback-Libeler divergence: $KL[p||q] = \int p(x) \log_e \frac{p(x)}{q(x)} dx$ $= \mathbb{E}_{p(\vec{x})}\left[\log_e \frac{p(\vec{x})}{q(\vec{x})}\right]$ and remove the constant term. Since the data distribution is approximated by $\{x_n\}_{n=1}^N$ we achieve $\theta^* = \arg\max \frac{1}{N}\sum_{n=1}^{N} \log p_\theta(x_n)$ by using empirical risk (computing expectation w.r.t empirical measure). Fit $p(z) = \mathcal{N}(z; 0; I) \wedge p_\theta(x|z) = \mathcal{N}(x; G_\theta(z), \sigma^2 I)$ where G is the neural network transform. Since $p_\theta(x) = \int p_\theta(x|z) p_\theta(z) dz$ we need to calculate the integral over the join distribution over the latent variable z; not tractable. Optimize the Variational Lower Bound by $\log p_\theta(x) \geq \mathbb{E}_{q(z)}[\log p_\theta(x|z)] - KL[q(z)||p(z)] := \mathcal{L}(x, q, \theta)$ which is what we try to maximise. The gap between the two is the KL divergence; therefore the lowerbound improves as q(z) approaches to the exact posterior. Variational Auto Encoders Since the exact posterior $p_\theta(z|x)$ depends on x, the variational lower-bound is tight when $q_n(z_n) \approx p_\theta(z_n|x_n)$. Therefore, we define q to be the encoder: $q(z) := q_\phi(z|x)$. Therefore the objectives become: $\theta^*, \phi^* = \arg\max L(\phi, \theta)$ and $L(\phi, \theta) := \mathbb{E}_{p_{data}(x)}[\mathbb{E}_{p_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||p(z)]]$ where q is defined by a NN as $q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), diag(\sigma_\phi^2(x)))$ where $\mu_\phi(x), \log \sigma_\phi(x) = NN_\phi(x)$. KL term has an analytical solution: $KL[q_\phi(z|x)||p(z)] = \frac{1}{2}(||\mu_\phi(x)||_2^2 + ||\sigma_\phi(x)||_2^2 - 2\langle \log \sigma_\phi(x), 1 \rangle - d$.
Reparametarization: this is still expensive to calculate because $\mathbb{E}_{p_\phi(z|x)}[\log p_\theta(x|z)]$ requires passing every z through the generative network. Therefore, use Monte-Carlo estimation to approximate the expected log likelihood. $\mathbb{E}_{p_\phi(z|x)}[\log p_\theta(x|z)] \approx \log p_\theta(x|z), \text{ where } z \sim q_\phi(z|x)$ thus $\nabla_\phi L(x, \phi, \theta) \approx \nabla_\phi E_{q_\phi(z|x)}[\log p_\theta(x|z)] - \nabla_\phi KL[q_\phi(z|x)||p(z)]$ and $z \sim q_\phi(z|x) \iff z = \mu_\phi + \sigma_\phi \odot \epsilon, \epsilon \sim \mathcal{N}(\epsilon; 0; I)$ then set $z = T_\phi(x, \epsilon)$ to make $\mathbb{E}_{p_\phi(z|x)}[\log p_\theta(x|z)] \approx \log p_\theta(x|T_\phi(x, \epsilon))$

GAN: constructs a binary classification task to assist learning generative model distribution $p_\theta(x)$ to fit the data distribution $p_{data}(x)$. Objective is: $\min_\theta \max_\phi L(\theta, \phi) := \mathbb{E}_{p_{data}(x)}[\log D_\phi(x)] + E_{p_\theta(x)}[\log(1 - D_\phi(x))]$ i.e. log(real 1) + log(fake 0). Assuming the discriminator network D_ϕ has infinite capacity with fixed θ: $\phi^* = \max_\phi L(\theta, \phi) \quad \text{satisfies} \quad D_{\phi^*}(x) = \frac{p_{data}(x)}{p_{data}(x) + p_\theta(x)}$ and substituting you get the Jensen-Shannon divergence $KL[p_{data}(x)||\tilde{p}(x)] + KL[p_\theta(x)||\tilde{p}(x)] - 2\log 2$ where $\tilde{p}(x) = \frac{1}{2}p_{data}(x) + \frac{1}{2}p_\theta(x)$ and thus $2JS[p_{data}(x)||p_\theta(x)] - 2\log 2$. We need to solve the 2 player game with a Double loop optimisaition: with fixed theta optimise phi, $\max_\phi \mathbb{E}_{p_{data}(x)}[\log D_\phi(x)] + \mathbb{E}_{p_\theta(x)}[\log(1 - D_\phi(x))]$. With fixed phi, optimize theta $\min_\theta \mathbb{E}_{p_\theta(x)}[\log(1 - D_\phi(x))]$. loop until convergence. Usually, these are batched: $\mathbb{E}_{p_{data}(x)}[\log D_\phi(x)] \approx \frac{1}{M}\sum_{m=1}^{M} \log D_\phi(x_m), \quad x_m \sim p_{data}(x)$ and $\mathbb{E}_{p_\theta(x)}[\log(1 - D_\phi(x))] \approx \frac{1}{K}\sum_{k=1}^{K} \log(1 - D_\phi(G_\theta(z_k))), \quad z_k \sim p(z)$.
Initialisation issue: generator very bad in beginning, so loss is 0. Instead, use an alternative "non-saturate" loss: $\min_\theta -\mathbb{E}_{p_\theta(x)}[\log D_\phi(x)]$ this maximises the probability of making a wrong decision on fake data. **Wasserstein GAN**: instead scores data . The original approach ($\min_\theta \max_\phi \mathbb{E}_{p_{data}(x)}[D_\phi(x)] - \mathbb{E}_{p_\theta(x)}[D_\phi(x)]$) doesn't consider infinite capacity of the discriminator network (which should return infinity if a real image). This makes a spiky function. We therefore constrain the lipschitz constraint $||D_\phi(\cdot)||_L \leq 1$. Once again, after intractibility issues: $\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[D_{\boldsymbol{\phi}}(\boldsymbol{x})] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[D_{\boldsymbol{\phi}}(\boldsymbol{x})] + \lambda \mathbb{E}_{\hat{p}(\boldsymbol{x})}[(||\nabla_{\boldsymbol{x}} D_{\boldsymbol{\phi}}(\boldsymbol{x})||_2 - 1)^2]$.