Objective Functions and Regularization

AIML 2021

Victoria University of Wellington

July 31, 2021

Overview

- Goal: to be able to select and use objective (cost) functions.
  - Material not all in the book (is too old).
- Objective functions for classification.
- Objective functions for regression and some generation methods.
- Regularization.
- Enforcing a density function.

Perspective

- Optimization of arbitrary objective functions was difficult before the advent of computers.
- Problems quadratic in the variables facilitate analytic solutions (setting the gradient to zero leads to a set of linear equations, which are easily solved). Hence, such problems have dominated history.
- Now we can use any reasonably behaved objective function:
  - If the objective function is convex it is nicer as then there is only one minimum.

Classification and objective functions 1

- Let us say we have a set classes and index them with v.
- You write a program and your prediction is $\hat{v}$.
- A *naive* objective function tells us if $\hat{v} = v$ (right) or $\hat{v} \neq v$ (wrong):
  - But a two-valued objective function is not differentiable.
  - A small change in the parameters θ does not lead to a small change in such a naive objective function. It either does nothing or it changes from 0 to 1 or from 1 to 0.
  - We need something less naive, something that provides us with a continuous representation of how close we are to getting it right.

Classification and objective functions 2

- Instead of a yes/no output and a hard yes/no decision on whether that is right, we can output a *class-is-observed probability* given the input, leading to a soft, differentiable objective function:
  - For a single class, our model then provides the probability of the class being observed in the input x: $q_\theta(v|x)$ with $v \in \{0, 1\}$ (class not present, class present), with parameters θ (we may omit the subscript).
  - For a set of classes we can write v as a vector: $y = [y_1, y_2, \cdots, y_d]$.
  - The network outputs d numbers that are class observation probabilities.
  - For a five-class (classes a, b, c, d, e) problem the desired output when groundtruth is item c is $[p(y_a|x), p(y_b|x), \cdots] = [0, 0, 1, 0, 0]$. The network output is $[q(y_a|x), q(y_b|x), q(y_c|x), q(y_d|x), q(y_e|x)]$. We then want the θ that gets q to match the groundtruth p over all data.
  - Easy to define a differentiable (to θ) objective function for training.
- In contras, in a regression problem the network has as output a variable (scalar, vector); an image, or a speech signal segment.

Two types of classification

- Each class gets an output unit (neuron).
- Network output: probabilities that each class has occurred in input x.
- We can distinguish two cases based on prior information provided.
- Type 1: here is one item and it is of one particular class:
  - Training data labels are one-hot vectors, e.g.,
    $[p(y_1 = 1|x), p(y_2 = 1|x), p(y_3 = 1|x)] = [0, 1, 0]$.
  - It can be only one of the possible classes: a car or a bicycle, or a person.
- Type 2: multiple items possible; sigmoidal output:
  - Training data labels are of the form $[p(y_1 = 1|x), \cdots] = [0, 1, 1]$.
  - There may be a car and/or a bicycle and/or person in the image.
- Often convenient: for training data $p(y_i = 1|x) = y_i$ with $y_i \in \{0, 1\}$.
- A natural objective function for both cases is cross entropy.
- This web page provides another view on the two cases.
- This web page discusses the entire classification setup.

Type 1: review of common notation

- We use V for standard notation of class:
  - For set of classes $C = \{1, 2, 3\}$ we have $v = 1$ or $v = 2$, or $v = 3$.
- We use Y for one-hot notation for the class:
  - For set of classes $C = \{1, 2, 3\}$ we have $y = [1, 0, 0]$ or $y = [0, 1, 0]$ or $y = [0, 0, 1]$. This notation may seem inefficient/redundant but is nice as each y_i can be a network output neuron.
  - $y_2 = 1$ then indicates "class 2 is observed" and is the same as $y = [0, 1, 0, \cdots]$.
- Scalar v and vector y different ways to represent the same class label.
- Each desired output training data point is a probability distribution p that is one-hot (of the form $[0, 1, 0]$). Note the following equivalencies:

$$p(v = 2|x) = p(y_2|x) = y_2. \tag{1}$$

- In the real world out there papers do not use V but use Y for both notation methods.

Type 1: softmax

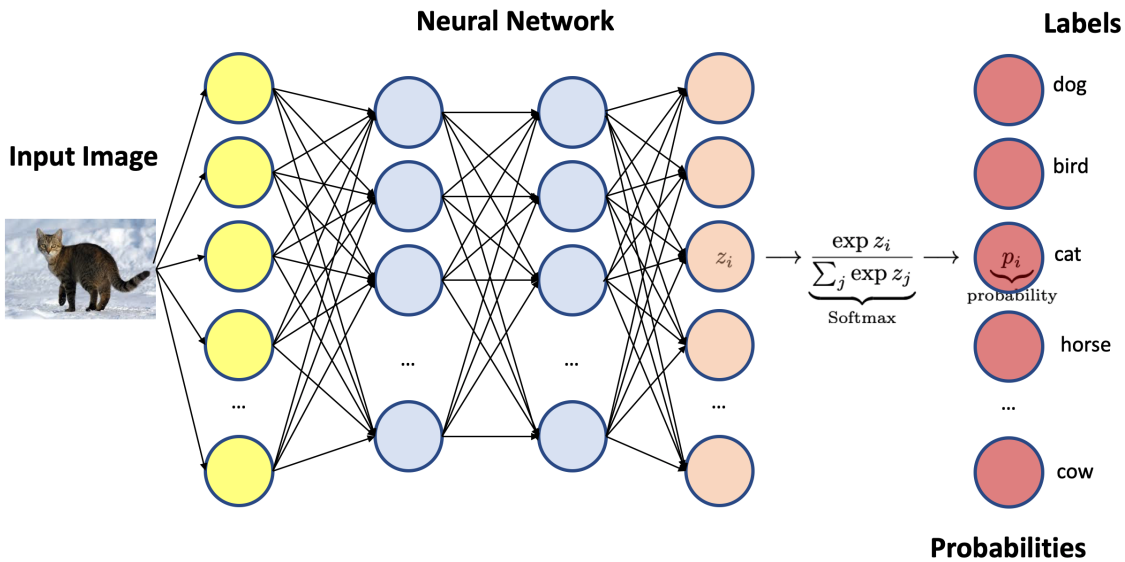
- Class probabilities $q(y_j)$ must sum to one:
  - Each input vector x of one class.
  - Example: either a car or a bicycle or a person.
- Output units always provide a proper probability mass function by means of *softmax* mapping:

$$q(y_i|x) = \frac{\exp(f_i(x))}{\sum_{j \in \mathbf{C}} \exp(f_j(x))} \tag{2}$$

  where $f(x)$ is the vector of $|C|$ activations in layer before output layer.
- Need $f_j(x)$ for all j to compute each $q(y_i|x)$: special computation.
- Recognize that $\exp$ is a trick to keep the probabilities positive.
- Natural objective functions: Kullback-Leibler divergence, max likelihood, cross entropy. We will shown again that they are equivalent.

Type 1: softmax

- Training: compare softmax output probabilities with one-hot vectors:
  - Proper measure described in next slides.
- Inference: select the highest probability class as output.



Type 1: Kullback-Leibler divergence and cross entropy

- This web page gives another view of this (note different notation).
- p short for ground-truth joint distribution, p_{XY} (X input, Y output).
- q is short for our model for the joint distribution, $q_{XY;\theta}$:

$$q_{XY;\theta}(x, y) = q_{Y|X;\theta}(y|x)\, p_X(x) \tag{3}$$

  - $q_{Y|X;\theta}(y|x)$ is the deep network (our model).
  - $p_X(x)$ distribution of input data (not modelled; is what it is).
- Objective: find θ that minimizes KL divergence p with model distribution q_θ.
- KL objective is same as minimizing cross entropy:

$$\inf_\theta D(p\|q) = \inf_\theta \mathrm{E}_p[\log \frac{p}{q}] = \mathrm{E}_p[\log p] - \inf_\theta \mathrm{E}_p[\log q] \tag{4}$$

$$= \inf_\theta -\mathrm{E}_p[\log q] \tag{5}$$

  - ($\inf$ is almost the same thing as "minimum of").

Type 1: cross entropy formulation

- Replace E_p by averaging over input data $x \in \mathcal{A}$:

$$\theta_{\text{opt}} = \arg\min_\theta -\mathrm{E}_p \log q \tag{6}$$

$$= \arg\min_\theta -\mathrm{E}_{p_X} \mathrm{E}_{p_{Y|X}} \log q(X, Y) \tag{7}$$

$$= \arg\min_\theta -\mathrm{E}_{p_X} \mathrm{E}_{p_{Y|X}} \log(q(Y|X)p(X)) \tag{8}$$

$$= \arg\min_\theta -\mathrm{E}_{p_X} \mathrm{E}_{p_{Y|X}} \log q(Y|X) \tag{9}$$

$$\approx \arg\min_\theta -\frac{1}{|\mathcal{A}|} \sum_{x \in A} \mathrm{E}_{p_{Y|X}} \log q(Y|x) \tag{10}$$

$$= \arg\min_\theta -\sum_{x \in A} \sum_y p(y|x) \log q(y|x) \tag{11}$$

- Set $p(y|x) = 1$ for correct class y for x and $p(y|x) = 0$ for other labels.
- $-\log q(y|x)$ is minus log likelihood loss. (11) maximizes a weighted average of class log likelihoods. Weighting depends on number of data available for each class (is $p(x)$).
- This derivation does not depend on y being one-hot, it could be our v; one-hot makes it work with softmax setup.

Type 1: binary logistic regression formulation

- Two-class classification commonly based on *logistic regression*:
  - Note we can get by with only a single output neuron.
  - Really nice description.
- Standard notation: $h(x) = q_{V|X;\theta}(1|x)$, where $v \in \{0, 1\}$ is the class.
- The likelihood of θ for the database A can be written as:

$$L = \prod_{i \in \mathcal{A}} (h(x^{(i)}))^{v^{(i)}} (1 - h(x^{(i)}))^{1 - v^{(i)}} \tag{12}$$

$$LL = \sum_{i \in \mathcal{A}} v^{(i)} \log h(x^{(i)}) + (1 - v^{(i)}) \log(1 - h(x^{(i)})) \tag{13}$$

$$= \sum_{i \in \mathcal{A} | v=1} \log h(x^{(i)}) + \sum_{i \in \mathcal{A} | v=0} \log(1 - h(x^{(i)})) \tag{14}$$

- Note that classes can be rewritten as $p(v^{(i)} = 1|x) = v^{(i)}$ and $p(v = 0|x) = 1 - v^{(i)}$, where p is ground truth. Then (14) is:

$$LL = \sum_{i \in \mathcal{A}} p(v^{(i)} = 1|x) \log(h(x^{(i)})) + p(v^{(i)} = 0|x) \log(1 - h(x^{(i)})) \tag{15}$$

- Binary logistic regression also equivalent to cross-entropy and KL div.

Type 1: binary logistic regression formulation

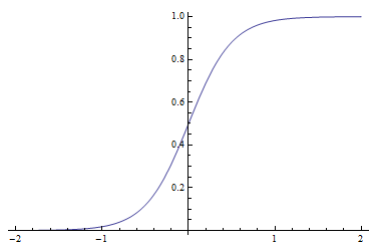
- We can get by with only a single output neuron:

$$h(x) = q(v = 1|x) = \frac{\exp(f_1(x))}{\exp(f_0(x) + \exp(f_1(x))} \tag{16}$$

$$= \frac{1}{1 + \exp(f_0(x) - f_1(x))} \tag{17}$$

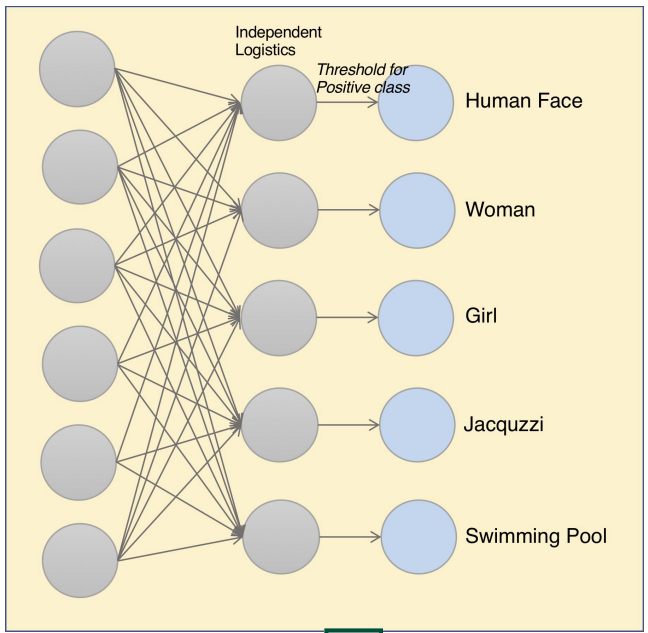
$$= \frac{1}{1 + \exp(f(x))} \tag{18}$$

- Tells us if $v = 1$, or not.



Type 2: sigmoid for binary classification

- Multiple classes can occur simultaneously:
  - Example: a car and/or a bicycle.
- For each class a separate binary classification problem: present/not+present.
- Training: compare output probabilities with binary vectors.
- Inference: compare value with threshold.



Type 2: sigmoid for binary classification

- $y_j \in \{0, 1\}$: class j happening yes/no.
- Output unit j provides the probability that class j happened.
- For each class a binary classification; (14) holds for each class = unit j:

$$LL_j = \sum_{i \in \mathcal{A}} y_j^{(i)} \log(h_j(x^{(i)})) + (1 - y_j^{(i)}) \log(1 - h_j(x^{(i)})) \tag{19}$$

- Note y_j (yes/no for j) is appropriate here, and not for (14).
- Can use form (18) for each class.
- Optimizing it all:

$$LL = \sum_{j \in \mathbf{C}} LL_j = \sum_{i \in \mathcal{A}} \sum_{j \in \mathbf{C}} y_j^{(i)} \log(h_j(x^{(i)})) + (1 - y_j^{(i)}) \log(1 - h_j(x^{(i)})) \tag{20}$$

Prediction/regression

- Learning to predict a $y \in \mathbb{R}^{d_1}$ given an $x \in \mathbb{R}^{d_2}$.
- Model: y is something you can predict from x, plus noise from a known distribution family:

$$y = \mu(x) + \eta \tag{21}$$

  - Assuming distribution is symmetric, then $\mu(x)$ is the mean for input x.
  - For prediction/regression we usually are interested only in the mean: for inference we want the network to provide $\mu(x)$.
- We will consider two cases:
  - Multi-variate Gaussian distribution.
  - Laplacian distribution.
    - Easily generalized to independent.
    - General multi-variate case is too complicated and not used.

Multi-variate Gaussian noise

- Predict multiple related variables y from a vector x.
- Examples:
  - Predicting the weather.
  - Tracking the coordinates of a missile trajectory from its past.
- Model: $p(y|x;\theta) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(y-\mu(x;\theta))^T\Sigma^{-1}(y-\mu(x;\theta))\right)$:
  - $\mu(x;\theta)$ is the network map with network parameters θ.
- Log likelihood

$$LL = -\frac{|\mathcal{A}|k}{2}\log(2\pi) - \frac{|\mathcal{A}|}{2}\log|\Sigma| - \frac{1}{2}\sum_{i\in\mathcal{A}}(y^{(i)}-\mu(x^{(i)};\theta))^T\Sigma^{-1}(y^{(i)}-\mu(x^{(i)};\theta) \tag{22}$$

- Attributes of method:
  - Emphasizes contributions of outliers, fewer large prediction errors.
  - Assumes short-tailed noise distribution.

Multi-variate Gaussian noise: why it works

- Only for insight, let us look at optimal solutions for θ and Σ:
  - Differentiate to θ, set to zero: optimal θ^* does not depend on Σ.
  - Differentiate to Σ, set to zero:

    $\hat{\Sigma} = \frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{A}} (y^{(i)} - \mu(x^{(i)}))^T (y^{(i)} - \mu(x^{(i)}))$.
- Is maximizing likelihood same as minimizing the noise in the model?
  - Yes, a tighter distribution increases the likelihood of the observations.
- Is the covariance matrix Σ relevant for optimizing network?
  - No, because our "insight only" shows they all lead to the same θ.
  - Yes, because it weights the importance of the errors during optimization, see (22).
  - But we see in practice setting $\Sigma = I$ is easy and not so bad.

Laplacian noise

- Predict variable y from a vector x.
- Scalar model: $p(y|x;\theta) = \frac{1}{2b}\exp\left(-\frac{|y-\mu(x;\theta)|}{b}\right)$.
- Log likelihood: $LL = -\log(2b) - \frac{1}{b}\sum_{i\in\mathcal{A}}|y^{(i)} - \mu(x^{(i)};\theta)|$.
  - To make it (independent) multi-variate, sum over output units j.
- For insight only:
  - Differentiating to μ, setting to zero not dependent on b.
  - Differentiating to b, set to zero: $\hat{b} = \frac{1}{|\mathcal{A}|}\sum_{i\in\mathcal{A}}|y^{(i)} - \mu(x^{(i)})|$.
- Attributes of method:
  - Equal footing to MSE for neural nets: analytic tractability not an issue.
  - Finding θ has only weak dependency on b.
  - De-emphasizes the effect of outliers: allows a few large errors.
  - Assumes long-tailed noise distribution.
  - Often very different results from Gaussian noise assumption.

Regularization

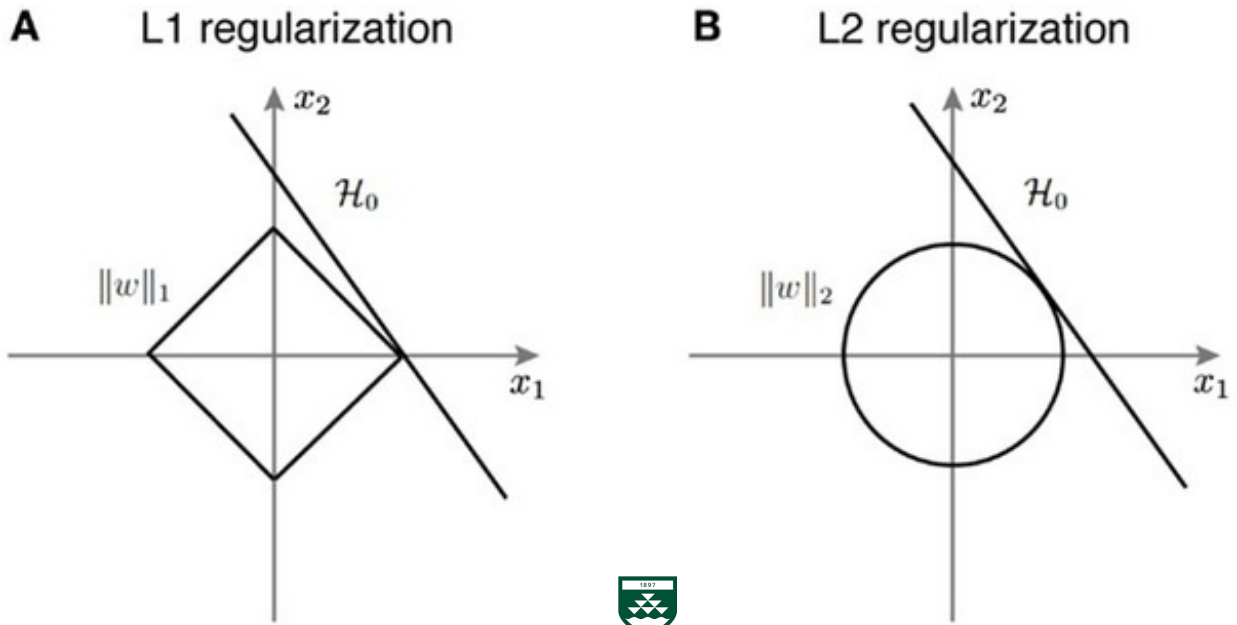
- Exploit *prior* knowledge about θ?
- Reduce the freedom of the network to get better generalization.
- Regularization by penalty term in the objective function:
  - (Yes, a bit "primitive" compared to Lagrange multiplier method.)
  - The weights are sparse (many are zero).
  - Spectral norm regularization.
  - Gradient penalty.
- Approaches that modify network operation:
  - Batch normalization.
  - Spectral normalization.
  - Weight clipping.
  - Orthonormal regularization.
  - Drop out.

L2 weight regularization

- The more specialized the weights, the more overfitting.
- Let W be the network weights
- We write baseline objective function as $J(W; x, y)$.
- L2 regularized objective function $J(W; x, y) + \lambda \|W\|_2^2$.
  - We wrote all weights together as a vector W.
  - Figure out a good λ by trial and error.
- Penalizes outliers in the weights since everything is squared.
- The weight decay perspective:
  - Updating with the baseline objective f: $W_{t+1} = W_t - \alpha \nabla_W J$.
  - L2 regularized objective f: $W_{t+1} = W_t - \alpha \nabla_W J - 2\alpha\lambda W$.
  - The additional term pushes the weights down to zero (weight decay).

L1 weight regularization

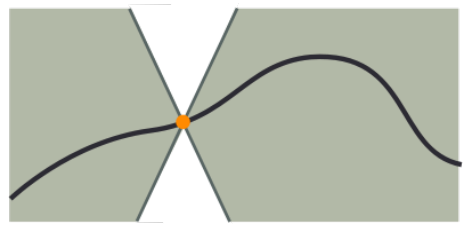
- We write baseline objective function as $J(W; x, y)$.
- L1 regularized objective function $J(W; x, y) + \lambda \|W\|_1$.
  - We wrote all weights together as a vector W.
  - $\|W\|_1$ is the sum of the absolute values of the weights.
  - Figure out a good λ by trial and error.
- Does not penalize outliers.
- Tends to give sparse weights: handwaving argument: let $\mathcal{H}_0$ be weights for which $J(W; x, y)$ is constant. Then the point on $\mathcal{H}_0$ with the smallest L1 norm tends to have many zeros and the point with the smallest L2 norm does not.



Lipschitz continuity and gradient penalty

- Good if the network is a continuous, nicely behaved mapping.
- A function f is k-Lipschitz continuous with metric $\|\cdot\|$ if

$$\|f(z_1) - f(z_0)\| \leq k\|z_1 - z_0\| \tag{23}$$



  - A nice animation of k-Lipschitz is on the wikipedia page.
  - Simple approximation: bound the gradient. (Exact if differentiable.)
- Practical implementations:
  - Crude methods: clip the weights (obsolete); restrict L2 of weights.
  - Two different gradient penalty terms; let f be the network:

$$G1 = \mathrm{E}_X(\|\nabla_x f(X)\| - 1)^2 \tag{24}$$

$$G2 = \mathrm{E}_X \max(0, \|\nabla_x f(X)\| - 1)^2 \tag{25}$$

Lipschitz continuity and spectral normalization

- Directly control the Lipschitz constant in each layer.
- The Lipschitz norm of a matrix A is defined as $\sigma(A) = \sup_z \frac{\|Az\|}{\|z\|}$
  - The direction of the largest gain (largest singular value).
  - All gains together form a "spectrum".
- As the maximum of the product of the gains of all network layers is the product of the max gains, we can look at layers individually.
- Control of the spectrum also means you control the gradient:
  - Single network layer maps $f^{(i)} : z_{i-1} \mapsto z_i$ with $f^{(i)} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$:
  - The gradient $\nabla f^{(i)}(z)$ is a Jacobian matrix that depends on z.
  - Spectral normalization constrains $\sup_z \sigma(\nabla f^{(i)}(z))$ for each layer.
  - Now ignore the gain of the nonlinearity (ok for ReLU). Then $\sup_z \sigma(\nabla f^{(i)}(z)) = \sigma(W)$, where W are the weights of the layer.
  - $\sigma(W)$ is just largest singular value of W.
- Definition spectral normalization is: $W := W/\sigma(W)$ for each layer.
- Naturally must use a fast method to implement $\sigma(W)$.

Drop out

- Simply set the output of a percentage of the units (e.g., 50%) of a layer to zero during training.
- Makes the network more robust: reduces overfitting.
- Figure 1 in the the original paper illustrates the method. You can also find illustrations of the performance improvement in that paper.

Encouraging densities of a particular form

- Example applications:
  - Output distribution that corresponds to faces or bedrooms.
  - Bottleneck latent variable that facilitates coding or classification.
- Compare two densities I:
  - Kullback-Leibler (not a metric / distance):
  - Jensen-Shannon (square-root is a metric).
- Compare two densities II (good for empirical distributions):
  - Integral probability metrics:
    - Maximum mean discrepancy (MMD).
    - Earth-mover's (Wasserstein) distance.
- Metric=distance: zero if identical, symmetric, triangle inequality holds.

Kullback-Leibler and Jensen-Shannon divergences

- Kullback-Leibler: $E_p \log \frac{p}{q}$,
  - Natural when distributions are parametric.
  - Nice:
    - Valued in bits.
    - Minimimizing it $=$ minimizing cross-entropy, works even when p is empirical, then $H(p,q) = -\sum_{i \in \mathcal{A}} \log q(x_i)$; good for classification.
  - Not nice:
    - Not a metric.
    - Cumbersome if both distributions are empirical.
    - Problematic when support q does not overlap with that of p (e.g., at initialization) as $\log 0 = -\infty$.
- Jensen-Shannon: $\frac{1}{2}\mathrm{E}_p[\log \frac{2p}{p+q}] + \frac{1}{2}\mathrm{E}_q[\log \frac{2q}{p+q}]$,
  - Corrects some of the problems of Kullback-Leibler.
  - Its square-root is a metric (distance).
  - Not convenient for empirical distributions.
  - Support problem no longer exists.

Maximum Mean Discrepancy (MMD)

- Distance between distributions, p_X and p_Y:
  - Nice approximation if only known through sets of observations (so for *empirical* distributions).
  - We can always sample a distribution if it is explicitly known.
- Application: you want distribution p_Y of Y to equal p_X of X:
  - Use MMD as penalty.
- Requires kernel $k(x, y)$:
  - Gaussian kernel: $k(x, y) = \exp(-\frac{\|x-y\|^2}{r})$, where r is chosen by designer.
  - Many others possible: a list of kernels.
- Empirical MMD for a set of samples of X and of Y:

$$\mathrm{MMD}(\{x^{(i)}\}_{i=1,\cdots,m}, \{y^{(j)}\}_{j=1,\cdots,n}) =$$
$$\frac{1}{m(m-1)} \sum_{i \neq j} k(x^{(i)}, x^{(j)}) + \frac{1}{n(n-1)} \sum_{i \neq j} k(y^{(i)}, y^{(j)}) - 2\frac{1}{mn} \sum_{i,j} k(x^{(i)}, y^{(j)})$$

- Intuition: first two terms larger than last one when p_X and p_Y differ.

Maximum Mean Discrepancy (MMD): the theory

- Uses reproducing kernel Hilbert space (RKHS).
- Let $h(\cdot)$ live in an RKHS, with kernel k: then $h(x) = \langle k(x, \cdot), h(\cdot) \rangle$.
- Define mean in RKHS $\mu_q(x) = \mathrm{E}_q[k(x, \cdot)]$, a function. Then:

$$\mathrm{E}_q[h(x)] = \mathrm{E}[\langle k(x, \cdot), h(\cdot) \rangle] = \langle \mathrm{E}[k(x, \cdot)], h(\cdot) \rangle = \langle \mu_q(x), h \rangle \tag{26}$$

  Hence this mean fully characterizes the distribution q.

- Take function to be the most telling one (witness function):

$$\sup_{\|h\| \leq 1} \mathrm{E}_p[h(x)] - \mathrm{E}_q[h(y)] = \sup_{\|h\| \leq 1} \mathrm{E}_p \langle k(x, \cdot), h \rangle - \mathrm{E}_q \langle k(x, \cdot), h \rangle \tag{27}$$

$$= \sup_{\|h\| \leq 1} \langle \mu_p - \mu_q, h \rangle = \langle \mu_p - \mu_q, \mu_p - \mu_q \rangle \tag{28}$$

$$= \mathrm{E}_{p,p} \langle k(x, \cdot), k(x', \cdot) \rangle - 2\mathrm{E}_{p,q} \langle k(x, \cdot), k(x', \cdot) \rangle + \mathrm{E}_{q,q} \langle k(x, \cdot), k(x', \cdot) \rangle$$

- Replace expectations with empirical averages: the empirical MMD.
- Witness function = critic.

Earth mover's / 1-Wasserstein distance

- A distance between distributions, p_X and p_Y.
- Let π be the set of joint distributions with marginals p_X and p_Y, then:

$$W_1(p_X, p_Y) = \inf_{p_{XY} \in \pi} \int \|x - y\| \, p_{XY}(x, y) dx dy \tag{29}$$

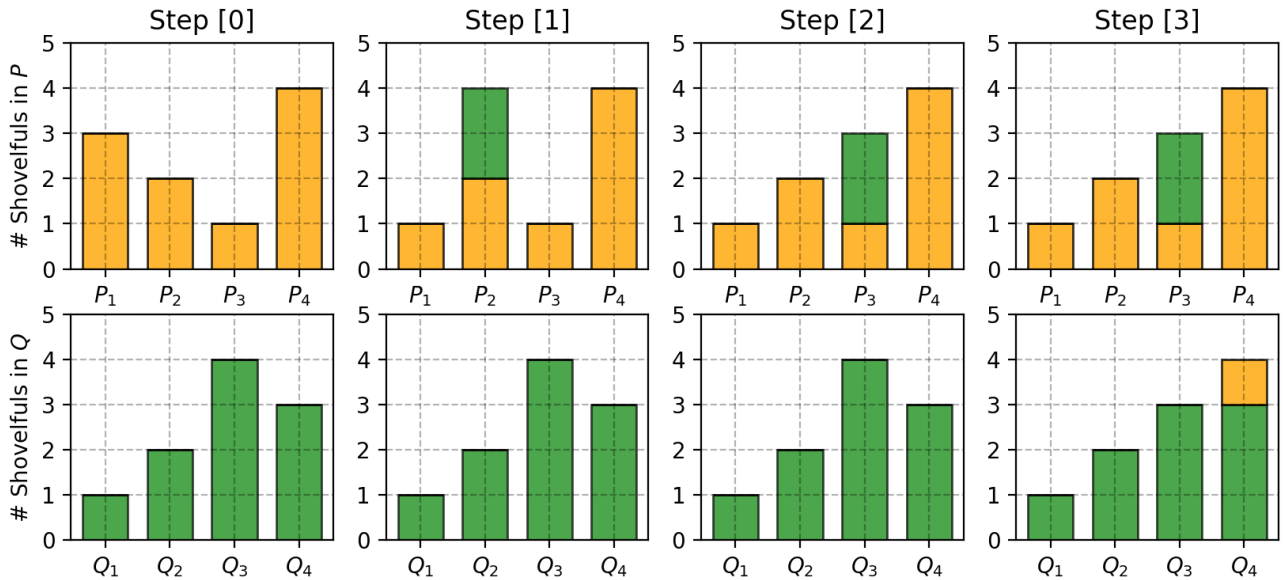
$$= \inf_{p_{XY} \in \pi} \mathrm{E}_{p_{XY}} \|X - Y\| \tag{30}$$

- (Subscript points to particular case of moment 1; we usually omit it.)
- $\|x - y\|$ is "travel" distance.
- The Kantorovich-Rubinstein duality shows that ($\| \cdot \|_{L \leq k}$ is k-Lipschitz):

$$W_1(p_X, p_Y) = \sup_{\|f\|_{L \leq 1}} \mathrm{E}_{p_X}[f(X)] - \mathrm{E}_{p_Y}[f(Y)] \tag{31}$$

- As with MMD, the expectations facilitate empirical distributions.
- A near-optimal f (the critic) can be found with a neural network!
- Compare two empirical distributions (groundtruth vs artificial faces):
  - Interlaced optimization f and generation of faces $\rightarrow$ GAN.
- A nice, more complete, description is on this page.

Illustration earthmoving / Wasserstein distance



Variance constraint

- Typical application: prevent latent variable from diverging.
- Example: constrain information passing through a layer:
  - Use results from information theory / communication theory.
  - Add gaussian noise N with known variance σ_N^2, constrain variance of input.
  - Can show information traveling through is bound by $I = \log_2(1 + \frac{\sigma_Z^2}{\sigma_N^2})$.
- Variance constraints are easy.
- Penalty function can be $\mathrm{E}_Z|\|Z\|^2 - 1|$, for example.

ELBO, evidence lower bound: preamble

- Scenario: observe X, have model $p(x, z)$ with hidden Z:
  - Want posterior $p(z|x)$.
  - Example: mixture distribution $p(x) = \sum_k p(z_k)p_k(x|z_k)$, $z \sim \text{Mult}(K)$.
  - $p(z|x) = \frac{p(x,z)}{\int p(x,z)dz}$ denominator problematic, as x are many data.
  - Solution: evidence lower bound (ELBO). More detail is here.
- Idea: find surrogate $q(z|x)$ for $p(z|x)$ maximizing lower bound on $p(x)$ (evidence):

$$\log p(x) = \text{E}_{q_{Z|X}} \log p(x) = \text{E}_{q_{Z|X}} \log \frac{p(Z, x)}{p(Z|x)} \tag{32}$$

$$= \text{E}_{q_{Z|X}} [\log p(x, Z) + \log \frac{q(Z|x)}{q(Z|x)p(Z|x)}] \tag{33}$$

$$= \text{E}_{q_{Z|X}} [\log p(x, Z) - \log q(Z|x) + \log \frac{q(Z|x)}{p(Z|x)}] \tag{34}$$

$$\geq \text{E}_{q_{Z|X}} [\log p(x, Z) - \log q(Z|x)] \tag{35}$$

$$= \text{ELBO} = \text{E}_{q_{Z|X}} [\log p(x|Z)] - D_{KL}(q(Z|x)\|p(Z)) \tag{36}$$

- With additional effort, (35) solves the original problem; we use (36).
- $q(z|x)$ is *variational* approximation of $p(z|x)$.

ELBO, evidence lower bound: VAE

- Maximize ELBO to get a generative model; write ELBO as:

$$\text{ELBO} = \mathrm{E}_{q_{Z|X}}[\log p(x|Z)] - D_{KL}(q(Z|x)||p(Z)) \tag{37}$$

- Note first term ELBO is reconstruction error, second measure of difference posterior and prior, this is the variational autoencoder (VAE):
  - Optimize both $p(x|z)$ and $q(z|x)$.
  - $p(x|z)$ typically a deterministic map and then we add Gaussian noise.
  - Gaussian noise is zero-mean and has fixed variance (not trained).
  - $p(z)$ is known from the start and chosen for ease to sample from.
  - Sample from z, and given a z we can create x.
  - z Gaussian noise, x a human face, for example.
- Comments on training:
  - $\mathrm{E}_{q_{Z|X}}$ can be replaced with summation over data at output of encoder.
  - $\log p(x|z)$ is tractable if deterministic decoder with noise at output.
  - $p(z)$ usually a Gaussian distribution.
  - Algorithm: maximize ELBO (optimize parameters of $p(x|z)$ and $q(z|x)$).

Problems I

1. Classification on MNIST (code needs to explictly compute gradients):
   - 1.1 Down-load, then preprocess the MNIST data base (both train and test) so that 1/3 of the images are rotated by 90 degrees left or and 1/3 are rotated by 90 degrees right. Separate training data into training and validation data.
   - 1.2 Train a network that has as objective to recognize if the images are rotated left, right, or not rotated.
   - 1.3 Provide a plot that shows training and validation accuracy as a function of epoch and report your final accuracy on the test data.

Problems II

2. The basic principle of generative models. *Your code must explictly compute gradients and your training method should also apply when only empirical desired output distributions are available (e.g., faces).*
   - 2.1 Train a simple fully connected neural network f_1 of your design that converts a two-dimensional (2D) $Z \sim \mathcal{N}(0, I)$ into a 2D uniform Y. That is, Y has uniform density in a 2D box (2-cube) with edges of length 1 and has zero probability outside the box.
   - 2.2 Train a second network f_{2a} that converts the 2D Y into a gaussian 2D Z.
   - 2.3 Next train f_1 and f_{2a} with different levels of L2 regularization on the weights. You may include one more sophisticated regularization method (gradient penalty or spectral normalization) for extra credit.
   - 2.4 Using 2D color plots, discuss qualitatively what happens to input data if we concatenate two networks $f_{2a} \circ f_1$, thus mapping Gaussian to Gaussian, for the different levels of regularization. For example, discuss how the movement of adjacent points is related.
   - 2.5 Train a third network f_{2b} that converts a 1D uniform Y into a gaussian 2D Z. Again use 2D color plots that show how points are mapped from the input Y to the output Z.