Statistical Machine Learning Revision Notes

Qinyu Skye Bai qinyub@student.unimelb.edu.au

Week 1 Revision

Vectors

- $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}' \cdot \boldsymbol{v} = \sum_{i=1}^{m} u_i v_i$
- $(k\boldsymbol{a})'\boldsymbol{b} = k(\boldsymbol{a}'\boldsymbol{b}) = \boldsymbol{a}'(k\boldsymbol{b})$
- $\boldsymbol{a}'(\boldsymbol{b} + \boldsymbol{c}) = \boldsymbol{a}'\boldsymbol{b} + \boldsymbol{a}'\boldsymbol{c}$
- $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u}' \cdot \boldsymbol{v} = ||\boldsymbol{u}||||\boldsymbol{v}||cos\theta$
- Scalar projection of $\boldsymbol{u}$ onto $\boldsymbol{v}$ is given by: $u_v = ||\boldsymbol{u}||cos\theta$
- If orthogonal: $\boldsymbol{u}'\boldsymbol{v} = 0$
- If parallel: $\boldsymbol{u}'\boldsymbol{v} = ||\boldsymbol{u}||||\boldsymbol{v}||$
- If anti-parallel: $\boldsymbol{u}'\boldsymbol{v} = -||\boldsymbol{u}||||\boldsymbol{v}||$
- $\boldsymbol{u}'\boldsymbol{u} = ||\boldsymbol{u}|| = \sqrt{u_1^2 + \cdots + u_m^2}$

Matrix Eigenspectrum

- $\boldsymbol{AV} = \lambda\boldsymbol{v}$, where $\boldsymbol{A}$ and λ simply stretches$\boldsymbol{v}$
- Eigenvalues of **symmetric matrices** are always real
- A matrix with linear dependent columns has some zero eigenvalues (rank deficient), i.e. no matrix inverse.

Positive (semi-)definite matrices

- A symmetric square matrix $\boldsymbol{A}$ is called positive semi-definite if for all vectors $\boldsymbol{v}$ we have $\boldsymbol{v}'\boldsymbol{A}\boldsymbol{v} \geq 0$, $\boldsymbol{A}$ has non-negative eigenvalues.
- If $\boldsymbol{v}'\boldsymbol{A}\boldsymbol{v} > 0$, $\boldsymbol{A}$ is positive definite. $\boldsymbol{A}$ has strictly positive eigenvalues.

Stochastic Convergence

- **Converges in probability**: A sequence $\{X_n\}$ of $r.v.$ converges in probability to $r.v.$ X if $\forall\epsilon > 0 : Pr(|X_n - X| > \epsilon) \rightarrow 0$
- **Converges almost surely**: A sequence $\{X_n\}$ of $r.v.$ converges almost surely to $r.v.$ X if $Pr(X_n \rightarrow X) = 1$
- Chain of implications:
  almost sure $\rightarrow$ in probability $\rightarrow$ in distribution

Frequentist Statistics

- Estimator bias: $B_\theta(\hat{\theta}) = E_\theta[\hat{\theta}(X_1, \cdots, X_n)] - \theta$
- Estimator variance: $Var_\theta(\hat{\theta}) = E_\theta[(\hat{\theta} - E_\theta[\hat{\theta}])^2]$
- Bias-variance decomposition of square-loss risk:
  $E_\theta[(\theta - \hat{\theta})^2] = [B(\hat{\theta})]^2 + Var_\theta(\hat{\theta})$
- **Consistence**: $\hat{\theta}(X_1, \cdots, X_n) \to \theta$ in probability.
- **Asymptotic efficiency**: $Var_\theta(\hat{\theta}(X_1, \cdots, X_n))$ converges to the smallest possible variance of any estimator of θ
- **Cramer-Rao lower bound**: $Var_\theta(\hat{\theta}) \geq \frac{1}{I(\theta)}$

Decision Theory

- Loss function:
  - Square loss of estimate: $(\hat{\theta} - \theta)^2$
  - Classification loss: $\mathcal{I}[y \neq \hat{y}]$
- Risk: $R_\theta[\delta] = E_{X \sim \theta}[l(\delta(X), \theta)]$
- Empirical Risk Minimisation (ERM): $\hat{R}_\theta[\delta] = \frac{1}{n}\sum_{i=1}^{n} l(\delta(X), \theta)$

Bayesian Statistics

- Parameters are also modeled like random variables. Bayes do not make point estimates.
- Bayes Rule: $P(\theta|X = x) = \frac{P(X=x|\theta)P(\theta)}{P(X=x)}$
- Marginalisation: $P(X = x) = \int_\Theta P(X = x, \theta \in \Theta)$
- Posterior mean: $\int \theta P(\theta|X) d\theta$
- Posterior mode (MAP): $argmax_\theta P(\theta|X)$

Parametric	Non-Parametric
Determined by fixed, finite number of parameters	Number of parameters grows with data, potentially infinite
Limited flexibility	More flexible
Efficient statistically and computationally	Less efficient

- Generative Model:
  - Model full joint $P(X, Y)$
  - Less accurate when conditional independence is not satisfied
  - Need less data to train but are bias because of assumptions
  - Suitable for known underlying distribution and for finding hidden parameters
  - Models: Bayesian Networks, Markov Random Fields, HMM
- Discriminative Model:
  - Model conditional $P(Y|X)$
  - Cannot work with missing data
  - Only useful for discriminating labels
  - Suitable to find the boundary that separates the data into different classes

    - Models: Logistic Regression, Traditional Neural Networks, KNN, CRF

Week 2 Linear & Logistic Regression, Iterative Optimisation

Linear Regression

- $\hat{y} = \sum_{i=0}^{m} x_i w_i = \boldsymbol{x'w}$ where $x_0 = 1$ and w_0 is the intercept.
- Sum of squared errors:

$$L = \sum_{i=1}^{n}(y_i - \sum_{j=0}^{m} X_{ij} w_j)^2$$

$$\frac{\partial L}{\partial w} = 2(X'(y - Xw)) = 0$$

$$X'y = X'Xw$$

$$\hat{w} = (X'X)^{-1}X'y$$

- Basis expansion: Marrying non-linear data to linear models by transforming the data, i.e. mapping the data to another feature space where data is linear.
    - **Radial basis function kernel**: $\psi(x) = \psi(||x - z||)$, z is a constant
    - **Advantage**: Basis expansion significantly increase the utility of methods, especially linear methods
    - **Disadvantage**:Transformations need to be pre-defined. For example, size of new feature set or z_i in RBFs.
    - We can learn transformation from data: Artificial NN
    - Use kernel trick

Optimisation formulations

- In general: $\hat{\theta} \in argmin_{\theta \in \Theta} L(data, \theta)$
- Approaches:
    - Analytic (Closed form solution): $\frac{\partial L}{\partial \theta_1} = ... = \frac{\partial L}{\partial \theta_p} = 0$
    - Approximate iterative solution: Gradient Descent
        * $\nabla L = [\frac{\partial L}{\partial \theta_1} = ... = \frac{\partial L}{\partial \theta_p}]'$
        * Choose $\theta^1 andsomeT$
        * For i from 1 to T^*: $\theta^{i+1} = \theta^i - \eta \nabla L(\theta^i)$
        * Return $\hat{\theta} \approx \theta^i$
        * To ensure a unique global minimum, gradient descent should be used on strictly convex function
    - Approximate iterative solution: SGD
        * Outer loop: each epoch, sweep through all training data
        * Within each epoch, randomly shuffle training data, then do gradient steps only on batches of data, with batch size of 1 or a few.
        * Faster than GD, but does not guarantee convergence.

  - Approximate iterative solution: Newton-Raphson
    * Intuitively, find the approximate value of $L'(\theta) = 0$
    * $\theta_{t+1} = \theta_t - L'(\theta_t)/L''(\theta_t)$
    * Converges faster, fitting a quadratic with curvature information. However, it's computationally expensive due to second derivatives (unless approximating Hessian)

Logistic Regression

- Logistic regression assumes a Bernoulli distribution: $Bernoulli(logistic(\boldsymbol{x'w}))$
- $P(Y=1|x) = \frac{1}{1+exp(-\boldsymbol{x'w})}$
  - Iterative Optimisation Gradient descent on Logistic:
  - $\mu(z) = logistic(z)$, then $\frac{d\mu}{dz} = \mu(z)(1-\mu(z))$ Z is linear response, μ is true response, $\hat{\mu}$ is logistic response
  - $\nabla L(\boldsymbol{w}) = \boldsymbol{X'}(\boldsymbol{y} - \boldsymbol{\mu})$
  - Iteratively-Reweighted Least Squares
  - $\nabla_2 L(\boldsymbol{w}) = -\sum_i \mu(x_i)(1-\mu(x_i))x_n x_n' = -\boldsymbol{X'MX}$
  - Newton-Raphson:
    $\boldsymbol{w}_{t+1}$
    $= \boldsymbol{w}_t + (\boldsymbol{X'M}_t\boldsymbol{X})^{-1}\boldsymbol{X'}(\boldsymbol{y} - \boldsymbol{\mu}_t)$
    $= \boldsymbol{X'M}_t\boldsymbol{X})^{-1}[\boldsymbol{X'M}_t\boldsymbol{X}\boldsymbol{w}_t) + \boldsymbol{X'}(\boldsymbol{y} - \boldsymbol{\mu}_t)]$
    $= (\boldsymbol{X'M}_t\boldsymbol{X})^{-1})\boldsymbol{X'M}_t\boldsymbol{b}_t$, where $\boldsymbol{b}_t = \boldsymbol{X}\boldsymbol{w}_t + \boldsymbol{M}_t^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_t)$

Week 3 Regularisation & PAC Learning Theory

Regularisation

- Regularisation is a process of **introducing additional information** in order to **solve an ill-posed problem** or to **prevent overfitting**.
- Achieved by augmenting the objective function
- Problem of irrelevant features:
  - Non-unique solutions
  - Lack of interpretability
  - Optimising to learned parameters is ill-posed problem
  - Feature $X_{\cdot j}$ is irrelevant if it's a linear combination of other columns, i.e. some eigenvalues of $X'X$ is zero. Also, near-irrelevance or co-linearity can be problematic, i.e. small eigenvalues of $X'X$.
  - Lack of data leads to underdetermined system, i.e. p>n.
  - Above leads to ill-posed problem, meaning that the problem solution is not defined.
- We can solve problems mentioned above by re-conditioning the problem, i.e. introduce an **additional condition** into the system:
  - Change the original problem from minimising $||\boldsymbol{y} - \boldsymbol{Xw}||_2^2$ to $||\boldsymbol{y} - \boldsymbol{Xw}||_2^2 + \lambda||w||_2^2$
    The solution becomes (**Ridge Regression**): $\hat{w} = (\boldsymbol{X'X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X'y}$
  - Regulariser as a prior: $\boldsymbol{W} \sim \mathcal{N}(0, \frac{1}{\lambda})$
  - Regulariser as a constraint: minimise $\boldsymbol{y} - \boldsymbol{Xw}||_2^2$ subject to $||\boldsymbol{w}||_2^2$ for $\lambda > 0$

Algorithm	Minimises	Solution
Linear regression	$\boldsymbol{y}-\boldsymbol{Xw}\|\|_2^2$	$\boldsymbol{(X'X)^{-1}X'y}$
Ridge regression	$\|\|\boldsymbol{y}-\boldsymbol{Xw}\|\|_2^2+\lambda\|\|w\|\|_2^2$	$\boldsymbol{X'X}+\lambda\boldsymbol{I})^{-1}\boldsymbol{X'y}$
Lasso	$\|\|\boldsymbol{y}-\boldsymbol{Xw}\|\|_2^2+\lambda\|\|w\|\|_1$	No closed-form, solutions are sparse and suitable for high-dim data

- Assessing generalisation:
  Train: ERM/minimisation of training error
  Generalisation: Captured by risk/test error. Model complexity is a major factor that influences the ability of the model to generalisation.

- Bias-variance decomposition:
  $E[l(Y,\hat{f}(X_0))]=(E[Y]-E[\hat{f}])^2+Var[\hat{f}]+Var[Y]$

- Simple model: High bias, low variance
  Complex model: Low bias, high variance

PAC Learning Theory

- $R[f_m]-R^*=(R[f_m]-R[f^*])+(R[f^*]-R^*)$
  $R[f_m]-R^*$: Excess risk
  $R[f_m]-R[f^*]$: Estimation error. Depend on algorithms and sample.
  $R[f^*]-R^*$: Approximation error. Depend on model family and all.
  $R[f_m]$: Error of the chosen classifier on random m samples
  $R[f^*]$: Best error of any classifier in this family
  R^*: Bayes risk possible. Depend on distribution and loss function.

- Simple family may underfit due to approximation error
  Complex family may overfit due to estimation error

- Since we do not know the data distribution, we need to bound generalisation to be small:
  With probability $\geq 1-\delta : R[f_m]\leq \hat{R}[f_m]+\epsilon(m,\mathcal{F})$

- **Hoeffding's Inequality**:
  The probability that the empirical average is far from the expectation is small:
  $Pr(|\mathbb{E}[h(Z)]-\frac{1}{m}\sum_{i=1}^{m}h(Z_i)|>\epsilon)\leq 2exp(-\frac{2m\epsilon^2}{(b-a)^2})$
  $Pr(\mathbb{E}[h(Z)]-\frac{1}{m}\sum_{i=1}^{m}h(Z_i)>\epsilon)\leq exp(-\frac{2m\epsilon^2}{(b-a)^2})$

- Proof: With high probability $\geq 1-\delta : R[f]\leq \hat{R}[f]+\sqrt{\frac{log(1/\delta)}{2m}}$
  $Pr(R-\hat{R}\leq\epsilon)\geq 1-exp(-2m\epsilon^2)$
  $\epsilon=\sqrt{\frac{log(1/\delta)}{2m}}$
  Now, we have bounded true risk of a classifier based on its empirical risk + some error.

- Uniform Deviation Bounds: We need the bounds to simultaneously or uniformly hold over a family of functions.

- ERM's estimation error is at most twice the uniform divergence:
  $R[f_m]\leq 2sup_{f\in\mathcal{F}}|R[f]-\hat{R}[f]|+R[f^*]$

- The Union Bound: $Pr(\bigcup_i A_i)\leq\sum_i Pr(A_i)$

- Bound for finite classes $\mathcal{F}$:
  Consider any $\delta>0$ and finite class $\mathcal{F}$. Then, with high probability at least $1-\delta$: $\forall f\in\mathcal{F}$, $R[f]\leq \hat{R}[f]+\sqrt{\frac{log|\mathcal{F}|+log(1/\delta)}{2m}}$

Week 4 VC Dimension & SVM

Vapnik-Chervonenkis Theory

Previously, PAC bound approach organises bad events by model and applies uniform bound is only tight if bad events are disjoint. However, in reality, some models generate overlapping bad events. To have a tighter bound, we better organise families bt possible patterns of labels on a dataset: the dichotomies of the family. Counting possible dichotomies gives a growth function.

- VC theory focuses on the pattern of labels that any $f \in \mathcal{F}$ could make.
- **Dichotomies and Growth Function**:
  Dichotomy: Given sample $x_1, .., x_m$ and family $\mathcal{F}$, a dichotomy is a $(f(x_1), ..., f(x_m)) \in \{-1, +1\}^m$ for some $f \in \mathcal{F}$.
  Even with infinite $\mathcal{F}$, $|\mathcal{F}(\boldsymbol{x})| \leq 2^m \leq |\mathcal{F}|$
  Growth Function:$S_{\mathcal{F}}(m) = sup_{x \in D^m} |\mathcal{F}(x)|$ is the max number of label patterns achievable by $\mathcal{F}$ for any m samples.
- PAC Bound with Growth Function
  Consider any $\delta > 0$ and any class $\mathcal{F}$. Then with high probability at least $1-\delta$: For all $f \in \mathcal{F}$: $R[f] \leq \hat{R}[f] + 2\sqrt{2\frac{logS_{\mathcal{F}(2m)} + log(4/\delta)}{m}}$

The VC Dimension

- Definition: The VC dimension VC($\mathcal{F}$) of a family $\mathcal{F}$ is the largest m such that $S_{\mathcal{F}}(m) = 2^m$
- Sauer-Shelah Lemma: Consider any $\mathcal{F}$ with finite VC($\mathcal{F}$) $= k$, any sample size m. Then $S_{\mathcal{F}(m)} \leq \sum_{i=0}^{k} \binom{m}{i}$
  Therefore we have, $R[f] \leq \hat{R}[f] + 2\sqrt{2\frac{klog\frac{2em}{k} + log(4/\delta)}{m}}$

Support Vector Machine

- $f(x) = w^T x + b$, where $w = \frac{\hat{w}}{b_0}$, $b = \frac{\hat{b}}{b_0}$
  Decision boundary: $f(x) = 0$
  Class1: $f(x) = -1$
  Class2: $f(x) = +1$
- Margin of classes: $||w|| \cdot Margin \cdot cos\theta = 2$
  $Margin = \frac{2}{||w||}$
  We want to maximise margin: $max\frac{2}{||w||} = min\frac{||w||^2}{2}$ s.t. $1 - y_i(w^T x_i + b) \leq 0$. That is, the distance of each data point to the decision boundary is greater than 1.
- Lagrange Duality:
  Lagrangian function: $L(x, \lambda) = f(x) + \lambda g(x)$
  Lagrangian multiplier: $\lambda \geq 0$
  Primal problem: $min_x f(x) = min_x max_\lambda L(x, \lambda)$
  Dual problem: $max_\lambda min_x L(x, \lambda) = max_\lambda \theta_d(\lambda)$, where $\theta_d(\lambda) = min_x L(x, \lambda)$
- KKT conditions:
  - Primal feasibility: $g(x) \leq 0$. Primal chooses x that minimises L.
  - Dual feasibility: $\lambda \geq 0$. Dual chooses λ that maximises L.
  - Complementary slackness: $\lambda g(x) = 0$.
  - Stationarity $\frac{\partial L}{\partial x} = 0$

Soft-margin SVM

- Motivation: Hard-margin loss is stringent and real data is unlikely to be linearly separable.
- Address non-seperable problem:
  - Relax the constraints (soft-margin)
  - Transform data but still using har-margin
  - Combine above
- Objective: $min_w(\frac{||w||^2}{2} + C\sum_{i=1}^{n}\xi_i)$ s.t. $y_i(w^T x_i + b) \geq 1 - \xi_i$, $\xi_i = max(0, 1 - y_i(w^T x_i + b))$ (Hinge loss) C is slack penalty.
  C = 0: mis-classified data is ignored, underfitting
  C = ∞: data has to be all correctly classified, overfitting

Summary

- Kernel is a function that can be expressed as a dot product in some feature space $K(\boldsymbol{u}, \boldsymbol{v})$
- RBF Kernel: $K(\boldsymbol{u}, \boldsymbol{v}) = exp(-\gamma||\boldsymbol{u} - \boldsymbol{v}||^2)$
  γ small: underfitting
  γ large: overfitting

Week 5 Recap of NN & Convolutional Neural Networks

Neural Network

- Limitations of perceptron learning: If the data is linearly separable, the algorithm will converge to a correct solution. Otherwise, it will fail to complete rather than give some approximate solution.
- Multi-layer perceptron consisted of only fully connected layers but no spatial invariant, i.e. not translation invariant.
  More parameters with more hidden layers.
  Do not scale well for images.

Convolutional Layer

- An image can be decomposed into local patches
  Different local patches could have different patterns
  To do classification, we can first extract local features and combine the local features for classification.
- Advantage:
  Learn translation-invariant pattern
  Weight sharing and sparse connection. i.e. faster computation and learn same features from different parts of a graph
  Learn hierarchical pattern
- Padding: Adding an appropriate number of rows and columns on each side of the input feature map
- Stride: The distance between two successive windows
- Output size calculation:
  No padding: $Size_{out} = ceil[(Size_{input} - Size_{kernel} + 1)/stride]$
  Padding: $Size_{out} = ceil(Size_{input}/stride)$

Max-pooling Layer

In all cases, pooling helps to make the representation become approximately invariant to small translations of the input. Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.

- Maximum pooling is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map.
- The results are down sampled or pooled feature maps that highlight the most present feature in the patch, not the average presence of the feature in the case of average pooling. The reason is that features tend to encode the spatial presence of some pattern or concept over the different tiles of the feature map (hence, the term feature map), and it's more informative to look at the maximal presence of different features than at their average presence.
- Advantage:
  Downsample feature map, reduce computational burden
  Increase size of receptive field

- Average pooling: dilute feature-presence information.

SGD

- Batch size = 1:
  Quick update in each step
  High variance in gradient
  Update model too often
- Batch size = N:
  Stable update
  Not computational feasible for large dataset
  Takes a long time to move each step
- Mini-batch SGD:
  More stable update compared with size = 1
  More computationally feasible compared with size = N
- Problem: Difficult to find a single global learning rate

Momentum

Instead of using only the gradient of the current step to guide the search, momentum also accumulates the gradient of the past steps to determine the direction to go.

- $v_t = \alpha \cdot v_{t-1} + (-\eta \Delta L(w_t))$
  $w_{t+1} = w_t - \eta \Delta L(w_t) + v_t$
  α: momentum, decay the previous velocity
- Advantages:
  Fast convergence speed
  Avoid local minimum

Adaptive Gradient (AdaGrad)

Above methods adapt updates to the slope of our error function and speed up SGD in turn. Adagrad adapts updates to each individual parameter to perform larger or smaller updates depending on their importance. It keeps an accumulative sum of the squared gradient.

Root Mean Square Propagation (Rmsprop)

RMSProp also tries to dampen the oscillations, but in a different way than momentum. RMS prop also takes away the need to adjust learning rate, and does it automatically. More so, RMSProp chooses a different learning rate for each parameter. Keep a moving average of the squared gradient.

Adaptive moment estimation (Adam)

So far, we've seen RMSProp and Momentum take contrasting approaches. While momentum accelerates our search in direction of minima, RMSProp impedes our search in direction of oscillations. Adaptive Moment Optimization algorithms combines the heuristics of both Momentum and RMSProp.

Week 6 CNN Architecture and Generative Models

LeNet5

- Layer 1: Convolutional layer
- Layer 2: Subsampling layer
- Layer 3: Convolutional layer. Non-complete connection scheme.
- Layer 4: Subsampling layer
- Layer 5: Convolutional layer
- Followed by fully connected layers.

AlexNet

- Trained on 2 GPU, each store half of the feature map.
- Layer 1: Convolutional layer
  Calculate total parameters: $n_{channels} * n_{filters} * Size_{filter}$ because each filter has the same channel number as the input
- Layer 2: Max-pooling layer
  Parameters = 0, because we simply output the max value of each local patch. No parameters used to apply operation.
- Layer 3: Convolutional layer. Non-complete connection scheme.
- Layer 4: Max-pooling layer
- Layer 5: Convolutional layer
- Followed by convolutional layers
- Followed by max-pooling to downsample the feature map, and flatten them out.
- Followed by fully connected layers

VGG 16/19

Total 16 layers that contain parameters: convolutional layers and fully connected layers. Vice versa for VGG 19. In VGG, all convolutional layers have a small kernel size in convolutional filters, different fro AlexNet. In VGG, we increase the size of receptive field by stacking more layers.
Using staking for smaller filter size takes the advantage of having less parameters.
Since each convolutional layer uses ReLU as the activation function, more layers means more non-linear rectification layers, thus more powerful network.

GoogleNet

- Motivation: Images/data have different scales which require different convolutional filter sizes.
- **Inception module Naive version**:
  Input → Processed by different convolutional layers with padding and max-pooling layers → Filter concatenation
- **Inception module with dimensionality reduction**:
  Use 1x1 convolution to reduce channels. (Applied before larger size convolutional layers)

ResNet (Residual Network)

- Motivation: For previous method, when the layers increased to above 20, the error rate will also increase. ResNet can still have better performance after 100 layers.
  In previous architectures, the network directly learns the original mapping between input and output. In ResNet, the network contains multiple residual blocks, which learn the residual mapping (difference original mapping and input), given that the residual mapping is easy to learn.

Generative Models

Autoencoder (AE)

- Meaningful representations can reconstruct the input data.
  Data → Encoder → Latent Representation →Decoder → Data
- We can use MLP for encoder or decoder. Or we can use CNN as encoder and decoder.
  If using CNN, we use max-pooling in encoder to downsample the feature map, while using up-sampling layer to increase the feature map so that the output size equals the input size.
- Reconstruction loss: output and input
- AE can be used for:
  Dimension reduction
  Image denoisying
  However, we can only learn fixed representation from the input data and reconstruct it, so we cannot generate new data. Hence, it does not count as a generative model.

Variational Autoencoder (VAE)

- Generative model. Can create new data.
- Determine a distribution of the latent variables, then we can randomly sample the latent variables from the distribution, thus generating data.
  $p(x) = \int p(x|z)p(z)dz$
  $p(x)$: probability of data x
  $p(x|z)$: probability of x given z
  $p(z)$: probability of latent variable z
  However, we do not know if the distribution of the output is the same as the distribution of the training set. And the latent variables we've sampled may not be a good representation.
- Solution to the above problem is to use posterior distribution to generate z:
  $p(z|x)$: $q(z|x) \sim \mathcal{N}(\mu, \sigma^2)$
- Network optimisation: log ML and Gradient ascent because we want to maximise the function (ML).
  $logp(x_i) = E_{z\sim q(z|x)}[log\frac{p(x_i|z)q(z|x_i)}{p(z|x_i)q(z|x_i)}]$
  $= E_{z\sim q(z|x)}[logp(x_i|z)] - D_{KL}(q(z|x_i)||p(z)) + D_{KL}(q(z|x_i)||p(z|x_i))$
  $\geq E_{z\sim q(z|x)}[logp(x_i|z)] - D_{KL}(q(z|x_i)||p(z))$ because the true distribution $p(z|x_i)$ is unknown.
  Note: $KL_{(p_1||p_2)} = \int p_1(x)log\frac{p_1(x)}{p_2(x)}dx$

- Two losses:
  - Reconstruction loss
  - KL divergence loss: between prior distribution and output encoder.
- During training , we minimise the KL between prior distribution and true distribution $\mathcal{N}(\mu, \sigma^2)$
  During testing, we do not have input data, so we sample from prior distribution as an approximation for the true distribution.

Generative Adversarial Model (GAN)

- Generate image that is similar to a real image.
  Consisted of generator ($z \sim \mathcal{N}$) and discriminator (binary classifier).
- View GAN as a two-player game:
  $min_G max_D V(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[log D(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$
  For D: If input is from data, then maximise first term. Otherwise, minimise $D(G(z))$, i.e. maximise second term.
  For G: Maxmise $D(G(z))$, i.e. minimise second term.
- Train Discriminator with binary cross-entropy. Architecture can be MLP or CNN. Input will be real data or generated image.
- Generator tries to fool discriminator by adversarial loss, also use binary cross-entropy.
  Here, we want Discriminator to predict $y = 1$, because we want it to be fooled. Therefore, use Discriminator to score the image, then, we measure the difference between score and 1 by using binary cross-entropy. During training, generator will update weights to minimise the loss. If loss decrease, then the score is higher, thus meaning our generated image is closer to the real image.
  Then, we use binary cross-entropy to calculate the difference between 1 and score.
- During training of generator, we should set the training of discriminator to false, i.e. discriminator should only be used for prediction. Otherwise, we reduce the power of discriminator.
- **Context Encoder**:
  Discriminator is the same. But the input of the generator is an image with a missing region, and the output is also the missing region.
  Two losses:
  The reconstruction loss between the real image and generated image in the missing region.
  Adversarial loss.

Week 7 TCN, RNN, Graph CNN

Temporal Convolutional Network

- One-hot Encoding
  - Create a feature vector
  - Dimension of the vector = size of the vocabulary
  - If the word is the i^{th} word in the vocabulary, then the i^{th} element of the vector is 1, all the others are 0.
  - Sparse, hardcoded and high-dimensional for very large vocabulary dictionary, therefore not practical
- Word Embedding
  - Create a weight matrix, where each column is a feature vector of the word. Rows are the number of features for each word.
  - Return the i^{th} column as the feature vector for the i^{th} word

  - Learn word embeddings jointly with the main task or load pre-trained word embeddings
  - Dense, lower-dimensional and learned from the data
- TCN for Sentence Classification
  - n x k representation of sentence with static and non-static channels.
  - Convolutional layer with multiple filter widths and feature maps.
  - Max-over-time pooling
  - Fully connected layer with dropout and softmax output
- Dilated Convolution: increase receptive field
- Dilated casual convolution: The output at time step t does not depend on the input information after time step t.
  This is achieved by padding. For example, perform padding before the current time step to include in the past information.

Recurrent Neural Network

- Process sequence step by step: At each step t: combine current time step x_t and historical information s_t to generate o_t. (zero-vector if current $t = 0$)
- Output generated by fully connected layer
- $o_t = activation(W \cdot x_t + U \cdot s_t + b)$, where W, U, b are shared over time.
- RNN is evaluated by back propagation through time
- Long short-term memory (LSTM)

Graph Convolution Networks

- CNNs are:
  - Flexible
  - Computationally efficient
  - Expressive
  - Gives great results
- Assumptions of CNNs data:
  - Regular, i.e. there is a consistent structure within the data
  - Dense (or non-sparse)
  - Have some local relational properties.
  - Consistent
  - Suitable for taking layers of filtering
  - Real world data doesn't ofter exist on nice grids. If assumptions do not hold, we can map data to a more regular structure or downsample the data. We can also perform imputation by filling in missing data to make a regular structure.
- **Graph Neural Networks**: With a graph neural network, we want to learn how to aggregate and propagate information across the graph, in a way that helps us extra local (node specific) or global (graph specific) features.
  We can parallelise the training by taking parts of the graph each time. Changing the subsets over

training allows global behaviours to be learned.
With careful design, even with the additional overhead of managing the graph, they also hold the potential to be more scalable to large data sets than other NN approaches.

Week 8 Multi-armed Bandits

Infallible Expert

- Infallible expert assumption: 1 or more experts makes no mistakes
- Algorithm: Majority Vote
  - Initialise set of experts who haven't made mistakes $E = \{1, ..., n\}$
  - Repeat per round:
    Observe predictions $E_i, \forall i \in \{1, ...n\}$
    Make majority prediction $argmax_{y \in \{-1,1\}} \sum_{i \in E} \mathbb{I}[E_i = y]$
    Observe correct outcome
    Remove mistaken experts from E
  - Mistake Bound for Majority Vote: Under infallible expert assumption, majority vote makes total mistakes $M \leq log_2 n$

Imperfect Experts and The Halving Algorithm

- No guarantee of an infallible expert
- Algorithm: Halving Algorithm
  - Initialise $w_i = 1$ weight of expert E_i
  - Repeat per round:
    Observe predictions $E_i, \forall i \in \{1, ...n\}$
    Make weighted majority prediction $argmax_{y \in \{-1,1\}} \sum_{i \in E} w_i \mathbb{I}[E_i = y]$
    Observe correct outcome
    Downweight each mistaken expert $E_i : w_i \leftarrow w_i/2$
  - Mistake Bound for Halving: If the best expert makes m mistakes, then weighted majority vote makes $M \leq 2.4(m + log_2 n)$ mistakes

Geralising weighted majority

- Initialise $w_i = 1$ weight of expert E_i
- Repeat per round:
  Observe predictions $E_i, \forall i \in \{1, ...n\}$
  Make weighted majority prediction $argmax_{y \in \{-1,1\}} \sum_{i \in E} w_i \mathbb{I}[E_i = y]$
  Observe correct outcome
  Downweight each mistaken expert $E_i : w_i \leftarrow (1 - \epsilon) w_i$
- Mistake Bound for Generalised Weighted Majority Vote: If the best expert makes m mistakes, then generalised weighted majority vote makes $M \leq 2(1 + \epsilon)m + (2ln(n))/\epsilon$ mistakes

Probabilistic Experts Algorithm

- Change from mistakes: Loss $l_i^t \in [0, 1]$ of E_i at round t.
- Randomised algorithm means, bounding expected losses
- Algorithm: Probabilistic Experts Algorithm

  - Initialise $w_i = 1$ weight of expert E_i
  - Repeat per round:
    Observe predictions $E_i, \forall i \in \{1, ...n\}$
    Predict E_i of expert i with probability $\frac{w_i}{W}$ where $W = \sum w_i$
    Observe losses
    Update $w_i \leftarrow (1-\epsilon)^{l_i^t} w_i$
  - Expected loss bound: Expected loss of the probabilistic experts algorithm is $L \leq \frac{ln(n)}{\epsilon} + (1+\epsilon)L^*$ where L^* is the minimum loss over experts.

Multi-armed Bandits

- MAB is the simplest setting for balancing exploration-exploitation. It's the same family of ML tasks as reinforcement learning.
- **Stochastic MAB setting**
  - Possible actions $1, ..., k$ are called arms, where arm i has distribution P_i on bounded rewards with mean μ_i
  - In round $t = 1...T$:
    Play action $i_t \in 1, ...k$
    Receive reward $R_{i_t} \sim P_{i_t}$
  - Goal: minimise cumulative regret: $\mu^* T - \sum_{t=1}^{T} E[R_{i_t}(t)]$
    $\mu^* = max_i \mu_i$
    $\mu^* T$: Best expected cumulative reward with hindsight
    $\sum_{t=1}^{T} E[R_{i_t}(t)]$: Expected cumulative reward of bandit

Greedy MAB

- At round t: Estimate value of each arm i as average reward observed:

$$Q_{t-1}(i) = \begin{cases} \frac{\sum_{s=1}^{t-1} R_i(s)\mathbb{I}[i_s=i]}{\sum_{s=1}^{t-1} \mathbb{I}[i_s=i]}, \text{if} \sum_{s=1}^{t-1} \mathbb{I}[i_s = i] > 0 \\ Q_0, \text{otherwise} \end{cases}$$

- $i_t \in argmax_{1 \leq i \leq k} Q_{t-1}(i)$
- Tie break randomly
- Initialisation of Greedy MAB:
  Pessimism: Init Q_0 below observable rewards: Only try one arm.
  Optimism: Init Q_0 above observable rewards: Explore arms once.
  Optimism improves Greedy

Epsilon-Greedy MAB

- At round t: Estimate value of each arm i as average reward observed:

$$Q_{t-1}(i) = \begin{cases} \frac{\sum_{s=1}^{t-1} R_i(s)\mathbb{I}[i_s=i]}{\sum_{s=1}^{t-1} \mathbb{I}[i_s=i]}, \text{if} \sum_{s=1}^{t-1} \mathbb{I}[i_s = i] > 0 \\ Q_0, \text{otherwise} \end{cases}$$

- $$i_t = \begin{cases} argmax_{1 \leq i \leq k} Q_{t-1}(i), w.p.(1-\epsilon) \\ Unif(\{1, ...k\}), w.p.\epsilon \end{cases}$$

- Tie break randomly
- ϵ-greedy performs better long-term by exploring then greedy.
- Limitations:
  - Improve with optimistic initialisation and decreasing ϵ
  - Exploration actions are blind to promising arms, i.e. we view all arms as equal
  - Exploitation is blind to confidence of estimates

Upper-Confidence Bound (UCB)

- Hoeffding's inequality
  Let $R_1, ...R_n$ be i.i.d. $r.v.$ in [0, 1] mean μ, denoted by $\bar{R}_n$ their sample mean.
  For any $\epsilon \in (0, 1)$ with probability at least $1 - \epsilon$: $\mu \leq \bar{R}_n + \sqrt{\frac{log(1/\epsilon)}{2n}}$, where:
  - $n = N_{t-1}(i) = \sum_{s=1}^{t-1} \mathbb{I}[i_s = i]$
  - $\bar{R}_n = Q_{t-1}(i)$
  - $\epsilon = 1/t^4$, goes down with the number of rounds, i.e. the more rounds the less worry about true rewards.
- At round t: Estimate value of each arm i as average reward observed:

$$Q_{t-1}(i) = \begin{cases} \hat{\mu}_{t-1}(i) + \sqrt{\frac{\rho \cdot log(t)}{N_{t-1}(i)}}, \text{if} \sum_{s=1}^{t-1} \mathbb{I}[i_s = i] > 0 \\ Q_0, \text{otherwise} \end{cases}$$

- $N_{t-1}(i) = \sum_{s=1}^{t-1} \mathbb{I}[i_s = i]$
  $\hat{\mu}_{t-1}(i) = \frac{\sum_{s=1}^{t-1} R_i(s)\mathbb{I}[i_s=i]}{\sum_{s=1}^{t-1} \mathbb{I}[i_s=i]}$
  $\sqrt{\frac{\rho \cdot log(t)}{N_{t-1}(i)}}$: Exploration boost. If we have not tries an arm many times, $N_{t-1}(i)$ will be very small, so the term will be big and gives an exploration boost. Here, we are being optimistic about the under-explored arms.
  For arms that do not have a good sample mean and we have tries the arm many times, the Q-value will be small, so we are unlikely to select that arm in the future.
- Regret bound: $O(log(t))$

Contextual bandits

- Arm's reward depend on state: $E[R_i(t)|X_i(t)]$

Week 9 Bayesian Regression and Classification

- Bayesian theorem views parameters as *r.v.* instead of a fixed value (as in the case of frequentest), which are represented as posterior probability.
  This provides a more robust prediction, and is less sensitive to overfitting, particularly with small training sets. It can also give rise to more expressive model classes.
- Bayesian maintains uncertinty, marginalises out unknowns during inference. It is, inmost cases, comparatively more complex in algorithms and more computationally expensive.

Bayesian Regression

- $y = \boldsymbol{X\beta} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 I_n)$ or in univariate case $\epsilon \sim \mathcal{N}(0, \sigma^2)$
  Likelihood: $y \sim \mathcal{N}(\boldsymbol{X\beta}, \sigma^2 I_n)$
  Here we assume prior $\beta \sim \mathcal{N}(\mathbf{0}, \gamma^2 I_D)$. However, we usually assume a flat prior for β to match MLE = MAP. Also we've assumed a known variance. Otherwise, $\sigma^2 = \tau^{-1} = p(\tau)$
- normal likelihood x normal prior = normal posterior
  Normal prior is a **conjugate prior**: when product of likelihood x prior results in the same distribution as the prior.
- The advantage of prior is that every time a newly labelled datapoint is observed, we can use that information as a new prior to compute the posterior.
  $p(\theta|y) = \frac{p(y_2|\theta)p(y_1|\theta)p(\theta)}{p(y_1,y_2)}$
  $\frac{p(y_2|\theta)p(y_1,\theta)}{p(y_1,y_2)}$
  $\frac{p(y_2|\theta)p(\theta|y_1)}{p(y_1,y_2)\cdot\frac{1}{p(y_1)}}$
  $\frac{p(y_2|\theta)p(\theta|y_1)}{p(y_2|y_1)}$
  $p(\theta|y_1)$ can be seen as a prior for the new data.
- Stages of Training
  - Decide on model formulation and prior
  - Compute posterior over parameters $p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y})$
  - MAP: Find mode for $\boldsymbol{w}$
    Approximate Bayes: Sample many $\boldsymbol{w}$ using MCMC.
    Exact Bayes: Use all $\boldsymbol{w}$ to make expected prediction on test.
- Prediction with uncertain $\boldsymbol{w}$ using Monte Carlo Integration:
  - sample multiple regression curves:
    sample S parameters, $\boldsymbol{w}^s$
  - for each sample compute prediction y_*^s at test point x_*
  - compute the mean and variance over the predictions
- Prediction using Bayesian regression:
  $p(\hat{y}_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x_*}, \sigma^2) = \int p(\boldsymbol{w}|\boldsymbol{X}, \boldsymbol{y}, \sigma^2) p(y_*|\boldsymbol{w}, \boldsymbol{x_*}\sigma^2) d\boldsymbol{w}$
- Generative binary models:
  Beta-binomial and Dirichlet-Multinomial
- Discriminative models:
  Logistic sigmoid
  Inference approach: Monte Carlo sampling

Week 10 Probabilistic Graphical Models (PGM)

- Marriage of graph theory and probability theory. Tool of choice for Bayesian statistical learning.
- PGM, aka, Bayes Nets
  Efficient joint representation
  Independence made explicit
  Trade-off between expressiveness and need for data, easy to make
  Easy for practitioners to model
  Algorithms to fit parameters, to compute marginals and posterior.
- Factoring Joint Distributions
  $Pr(X_1, ..., X_k) = \prod_{i=1}^{k} Pr(X_i | X_{i+1}, ..., X_k)$
  We can simplify by:
  - Drop unconditional independence: $Pr(X|Y) = Pr(X)$
  - Drop conditional independence: $Pr(X|Y, Z) = Pr(X|Y)$
  - Simpler factors allows us to speed up inference and avoid overfitting.

Directed PGM

- Joint factorisation:
  $Pr(X_1, ..., X_k) = \prod_{i=1}^{k} Pr(X_i | X_j \in parents(X_i))$
- PGMs represent joints, which are central to Bayes. Bayesians add node per parameters, with table being the parameter's prior.

Undirected PGM

- Indirected variant of PGM, parameterised by arbitrary positive valued functions of the variables, and global normalisation. A.k.a. Markov Random Field.

Undirected	Directed
Each node is a $r.v.$	Each node is a $r.v.$
Each clique C has "factor" $\Psi_c(X_j : j \in C) \geq 0$	Each node has conditional $p(X_i \vert X_j \in parents(X_i))$
Joint $\propto$ product of factors	Joint = product of conditionals

  Key Difference = normalisation

- Formulation:
  Clique: A set of fully connected nodes
  Maximal clique: Largest cliques in graph
  $p = \frac{1}{Z}\psi_i(MC_{nodes})$, where Z is a normalising constant and ψ is a positive function
- Pros
  - Generalisation of D-PGM
  - Simpler means of modelling without the need for per-factor normalisation
  - General inference algorithms use U-PGM representation (supporting both types of PGM)
- Cons
  - Weaker independence
  - Calculating global normalisation term Z intractable in general (tractable for chain or trees)

Examples of PGMs

- HMM
  - HMMs are generative. Sequential observed outputs from hidden state
  - $A = \{a_{ij}\}$, transition probability matrix, $\forall i : \sum_j a_{ij} = 1$
  - $B = \{b_i(o_k)\}$, output probability matrix, $\forall i : \sum_k b_i(o_k) = 1$
  - $\Pi = \{\pi_i\}$, initial state distribution, $\sum_i \pi_i = 1$
  - Applied to NLP, speech recognition, CV, alignment
  - Fundamental HMM tasks:
    - Evaluation: Given an HMM μ and observation sequence O, determine likelihood $Pr(O|\mu)$. (Probabilistic inference in PGM task)
    - Decoding: Given an HMM μ and observation sequence O Determine most probable hidden state sequence Q. (MAP point estimates in PGM)
    - Learning: Given an observation sequence O, and set of states, learn parameters A, b, Π. (Statistical inference in PGM task)
- Kalman Filter is the same but with continuous Gaussian $r.v.$
- Conditional Random Field (CRF) is the undirected analogue. CRFs are discriminative.

Inference on PGMs

- Probabilistic inference on PGMs: Computing marginal and conditional distributions from the joint of a PGM using Bayes rule and marginalisation.

Probabilistic Inference by Simulation

- Motivation: Exact probabilistic inference can be expensive/impossible
- Solution: Approximate distribution by histogram of a sample.
- Gibbs Sampler:
  It is assumed that is possible to directly sample from the set of conditional posterior $p(\theta_i|\theta_{-i}, y)$
  First, define starting point $\theta^0 = (\theta_1^0, ..., \theta_K^0)$
  For $i = 1, ...K$, draw θ_i^t from conditional posterior $p(\theta_i^t|\theta_{-i}^*, y)$, where $\theta_{-i}^* = (\theta_1^t, ..., \theta_{i-1}^t, \theta_{i+1}^{t-1}, ..., \theta_K^{t-1})$
- Markov Blanket: All the nodes that a node directly depend on. Not just the parents/children.
  In D-PGM Markov blanket is: Parent of i, children of i, parents of the children of i.
  $p(X_i|MB(X_i)) \propto p(X_i|X_{\pi_i}) \prod_{k:i\in\pi_k} p(X_k|X_{\pi_k})$

Week 11 Hidden Markov Model and Gaussian Mixture Model

Hidden Markov Models

- Model of choice for sequential data. Aform of clustering or dimensionality reduction for discrete time series.
- Two assumptions:
  - Markov chain: $P(q_{t+1}|q_1, o_1..., q_t, o_t) = P(q_{i+1}|q_i)$
  - Independent assumption: $P(o_t|q_1, o_1..., q_t, o_t) = P(o_t|q_t)$
- Joint: $P(\boldsymbol{o}, \boldsymbol{q}) = P(q_1)P(o_1|q_1) \prod_{i=2}^T P(q_i|q_{i-1})P(o_i|q_i)$
- Evaluation (Marginalisation): Given $\mu = \{A, B, \Pi\}$, find $P(o|\mu)$
  $P(o|\mu) = \sum_{q_1} P(q_1)P(o_1|q_1) \sum_{q_2} P(q_2|q_1)P(o_1|q_1)... \sum_{q_T} P(q_T|q_{T-1})P(o_T|q_T)$

Forward Algorithm

- Let $m_t(q) = p(o_1, q_1..., o_t, q_t|\mu)$
  $P(\boldsymbol{o}|\mu) = \sum_{t=1}^T P(\boldsymbol{o}, q_t|\mu) = \sum_{t=1}^T m_t(q)$
  Now, we want to use recursion to represent m_{t+1} by m_t
  $m_{t+1} = p(o_1, q_1..., o_t, o_{t+1}, q_{t+1}|\mu)$
  $= p(o_{t+1}|o_1, q_1..., q_t, o_t, q_{t+1}, \mu) \cdot p(o_1, ..., o_t, q_{t+1}|\mu)$
  By Markov, $= p(o_{t+1}|q_{t+1}) \cdot p(o_1, q_1..., o_t, q_t, q_{t+1}|\mu)$
  $= p(o_{t+1}|q_{t+1}) \cdot p(q_{t+1}|o_1, q_1..., o_t, q_t, \mu) \cdot p(o_1, q_1..., o_t, q_t|\mu)$
  By *i.i.d.* assumption, $= p(o_{t+1}|q_{t+1}) \cdot p(q_{t+1}|q_t) \cdot m_t$
  $= b_{t+1} \cdot a_{q_t q_{t+1}} \cdot m_t$

Backward Algorithm

- Let $m_t(q) = p(o_{t+1}, o_t|q_t, \mu)$
  $m_1(q) = p(o_2, ..., o_t|q_1, \mu)$
  $P(\boldsymbol{o}|\mu) = \sum_{t=1}^T P(\boldsymbol{o}|\mu)$

  Now, we want to use recursion to represent m_t by m_1. we introduce q_1 to the equation
  $P(\boldsymbol{o}|\mu) = \sum_{q_1} P(o_1, ..., o_t, q_1|\mu)$
  $= \sum_{q_1} p(o_1, ..., o_t, |q_1, \mu) \cdot p(q_1|\mu)$
  $= \sum_{q_1} p(o_1, ..., o_t, |q_1, \mu) \cdot p(q_1|\mu)$
  $= \sum_{q_1} p(o_1|o_2, ..., o_t, q_1, \mu) \cdot p(q_1|\mu) \cdot p(o_2, ..., o_t|q_1, \mu)$
  By *i.i.d.* assumption, $= \sum_{q_1} p(o_1|q_1, \mu) \cdot p(q_1|\mu) \cdot p(o_2, ..., o_t|q_1, \mu)$

$= \sum_{q_1} p(o_1|q_1, \mu) \cdot p(q_1|\mu) \cdot m_1$
$= \sum_{q_1} b_1(o_1) p(q_1|\mu) \cdot m_1$

Baum Welch using EM

- Learning: $\mu_{MLE} = argmax_{\mu} P(o|\mu)$
- initailise μ^1
- Compute marginals of $P(q_t|o, \mu^j)$, $P(q_{t-1}, q_t|o, \mu^i)$
- fit model μ^{j+1} based on expectations
- repeat
- Computation based on forward-backward.

Forward-Backward Algorithm

- Marginalisation: $P(q_{i-1}, q_i, \boldsymbol{o}) = \sum_{q_1,...,q_{i-2},q_{i-1},q_T} P(\boldsymbol{o}, \boldsymbol{q})$
  $= \sum_{q_1,...,q_{i-2}} P(o_1, ..., o_{i-2}, q_1, ..., q_{i-1}) \cdot P(o_{i-1}|q_{i-1} P(q_i|q_{i-1}) P(o_i|q_i) \cdot \sum_{q_{i+1},...,q_T} P(o_{i+1}, ..., o_T, q_{i+1}, ..., q_T|q_i)$

Inference as Message Passing

- Each m can be considered as a message which summarises the effect of the rest of the graph on the current node marginal. Inference = passing messages between all nodes. Messages are recursively decided.
- Viterbi algorithm is the max-product variant of forward algorithm, solves the $argmax_q P(\boldsymbol{q}|\boldsymbol{o})$

Unsupererversed Learning

- Aim: Explore the structure/patterns/regularities of data.
- Catagory:
  Clustering
  Dimensionality reduction (autoencoders)
  Learning parameters of probablistic models
- K-means Clustering
  Initialisation: Choose k cluster centroids randomly
  Update: Assign points to the nearest centroid and compute the new centroids under the current assignment
  Termination: If no change then stop
  Go update
- A probabilistic mixture model, like k-means, requires us to choose the number of clusters in advance. Contrary, the probabilistic model gives a power to express uncertainty about the origin of each point.

Gaussian Mixture Models

- A probabilistic view of clustering. Simple example of a latent variable model.
- Multi-dimensional Gaussian is:
  $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \sim (2\pi)^{-\frac{d}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^{\boldsymbol{T}} \boldsymbol{\Sigma}^{\boldsymbol{-1}} (\boldsymbol{x} - \boldsymbol{\mu})$
- Cluster assignment of points: Multinomial distribution on k outcomes
- Location of point: Each cluster has its own Gaussian distribution. Location of point is governed by its cluster assignment.

- Class conditional density: $P(X|Z=j) \sim \mathcal{N}(\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})$
- When fitting the model to observations, we'll be maximising likelihood of observed portions of the data not the latent parts. Marginalising out Z (latent variables), we can derive the mixture distribution of data.
- $P(\boldsymbol{x}) = \sum_{j=1}^{k} w_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})$
  $= \sum_{j=1}^{k} P(C_j)P(\boldsymbol{x}|\boldsymbol{C_j})$
  Here w_j is equivalent to $P(C_j)$ which can be viewed as the weight of class j.
- Clustering now amounts to finding parameters of the GMM that best explains observed data.
- GMM can be viewed as another Direct PGM
  Some variables are observed and some are latent]newline Convenient to model location as generated by cluster assignment
  Shared cluster arise from independence between points
  Mixture distribution arises algebraically from marginalisation

Expectation-Maximisation

- Motivation: We want to implement MLE but we have unobserved *r.v.* that prevent clean decomposition as happens in fully observed settings.
- Objective: Modelling the data points as independent, find:
  $P(\boldsymbol{C_j})$, $\boldsymbol{\mu_j}$, $\boldsymbol{\Sigma_j}$, for $j = 1, ...k$, that maximise $\sum_{j=1}^{k} P(C_j)P(\boldsymbol{x}|\boldsymbol{C_j})$, where $P(\boldsymbol{x}|\boldsymbol{C_j}) \sim \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})$
  Taking log: $logP(x_1, ..., x_n) = \sum^n log(\sum^k)P(C_j)P(x_i|C_j)$
- Issues:
  Sometimes we don't observe some of the variables needed to compute the log likelihood. (e.g. GMM cluster membership is not known in advance)
  Sometimes the form of the log likelihood is inconvenient to work with. (Taking a derivative of GMM log likelihood results in a cumbersome equation).
- Algorithm:
  Initialse K clusters: $C_1, ..., C_k$, (μ_j, Σ_j), $P(C_j)$ for each cluster j
  Iteration:
  Expectation: Estimate the cluster of each X: $P(C_j|x_i)$
  Maximisation: Re-estimate the cluster parameters. (μ_j, Σ_j), $P(C_j)$ for each cluster j

Week 12 Expectation-Maximisation and Bayesian Record Linkage

- MLE v.s. EM:
  MLE is a frequentist principle that suggests that given a dataset, the "best" parameters to use are the ones that maximise the probability of the data
  EM is an algorithm, a way to solve the problem posed by MLE. Especially convenient under unobserved latent variables.
  EM uses **coordinate ascent** on a lower bound on the log-likelihood:
  M-step: Ascent in modeled parameters $\boldsymbol{\theta}$
  E-step: Ascent in the marginal likelihood P(Z)
  Each step moves towards a local m=optimum and can get stuck. Might need random restarts.
  We fix θ and optimse the lower bound for $p(Z)$
  Then fix $p(Z)$ and optimise for θ
- Variables and parameters in GMMs:
  Variables: Point locations $\boldsymbol{X}$ and cluster assignments $\boldsymbol{Z}$.
  Parameters: $\boldsymbol{\theta}$ are cluster locations and scales.

- **Jensen's Inequality**: $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$
  maximise $log(p(\boldsymbol{X}|\boldsymbol{\theta}))$
  $log(p(\boldsymbol{X}|\boldsymbol{\theta})) = log\sum_Z(p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta}))$
  $= log\sum_Z(p(\boldsymbol{Z})\frac{p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})}{p(\boldsymbol{Z})})$
  $= log\mathbb{E}_Z(\frac{p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})}{p(\boldsymbol{Z})})$
  $\geq \mathbb{E}_Z(log\frac{p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})}{p(\boldsymbol{Z})})$
  $= \mathbb{E}_Z[logp(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})] - \mathbb{E}_Z[logp(\boldsymbol{Z})]$
- Hence we have: $log(p(\boldsymbol{X}|\boldsymbol{\theta})) \geq \mathbb{E}_Z[logp(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})] - \mathbb{E}_Z[logp(\boldsymbol{Z})]$ and this is a lower bound on the original log likelihood.
- To maximise the LHS term, we want to push the lower bound up. This lower bound is a function of two "variables" θ and $p(\boldsymbol{Z})$ We want to maximise the RHS as a function of these two "variables". It is hard to optimise with respect to both at the same time, so EM resorts to an iterative procedure.
- For any new $theta^*$, we make the lower bound tighter.
  This is because, $logp(\boldsymbol{X}|\boldsymbol{\theta}) \geq \mathbb{E}_Z + logp(\boldsymbol{X}|\boldsymbol{\theta})$, the first term is always less than or equal to 0.
- When $p(Z) = p(Z|X,\theta^*)$, the first term can usually be maximised as a function of $theta$ in a closed form. If not, probably do not use EM.

EM as Iterative Coordinate Ascent

- Algorithm:
  Initialse values of $\boldsymbol{\theta}^1$
  Update:
  Expectation: Compute $Q(\boldsymbol{\theta},\boldsymbol{\theta}^t) = \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta}^t}[logp(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})]$
  Maximisation: $\boldsymbol{\theta}^{T+1} = argmax_\theta Q(\boldsymbol{\theta},\boldsymbol{\theta}^t)$
  Terminate if no change
  Repeat
- This algorithm will eventually stop (converge), but the resulting estimate can be only a local maximum

Latent Variables of GMMs

- Let $z_1,...z_n$ denote true origins of the corresponding points $x_1,...x_n$ Each z_i is a discrete variable that takes values in $1,...,k$ where k is the number of clusters.
- Original likelihood:
  $logP(x_1,...,x_n) = \sum_{i=1}^n log(\sum_{c=1}^k w_c\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_c},\boldsymbol{\Sigma_c}))$
- Complete log likelihood with known Z:
  $logP(x_1,...,x_n,\boldsymbol{z}) = \sum_{i=1}^n log(w_{z_i}\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_{z_i}},\boldsymbol{\Sigma_{z_i}}))$
- EM handles uncertainty by replacing $logp(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})$ with expectation $\mathbb{E}_Z[logp(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})]$

Cluster Responsibilities

- In E-step, we set latent Z as originating cluster:
  $P(z_i = c|x_i,\theta^t) = \frac{w_c\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_c},\boldsymbol{\Sigma_c})}{\sum_{j=1}^k log(w_j\mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_j},\boldsymbol{\Sigma_j}))}$
  This is called **responsibility** that cluster c takes for data point i:
  $r_{ic} = P(z_i = c|x_i,\theta^t)$

K-means as a EM for a restricted GMM

- k-means algorithm is a EM algorithm for the restricted GMM model

- GMM model with $w_c = \frac{1}{k}$, and each has the same fixed variance-covariance matrix $\Sigma_c = \sigma^2 I$, thus only μ_c requires estimation.
  Next, approximate a probabilistic cluster responsibility with a deterministic assignment $r_{ic} = 1$ if centroid μ_c^t is closest to point x_i, 0 otherwise. (E)
  New μ_c should be set as a centroid of points assigned to cluster c.(M)

Record Linkage

- Identifying records that refer to the same entity. A.k.a entity resolution, data matching, deduplication, merge/purge. Can formulate as a classification problem on pairs of records, although may get conflicting predictions.
  Practical issue: ground truth labels are often unavailable

Bayesian Record Linkage

- Motivation:
  Data-efficient
  Model encodes constraints and prior beliefs about the generative process
  Apply Bayes' rule to update beliefs about unknown parameters (i.e. coreference relation), conditional on observed data
  Distributions represent uncertainty in beliefs—can propagate to analyses on linked data

- Research Directions
  - Improving MCMC efficiency: Gibbs sampling: Markov chain converges slowly and exhibits high autocorrelation
  - Marginalising out latent variables can help
  - Designing proposals that make more "global" updates under a Metropolis-Hastings framework