ML 2565 (Fall 2022): Scribe Notes

1 Introduction

(no scribing)

2 Supervised Learning

(no scribing)

3 A Bayesian Interlude

Start scribing here! Optimizing parameters of a linear regression is not the only form of learning. This lecture will give a brief overview on how some of the ideas of Bayesian probability theory can be applied to ML. Naive Bayes is not covered.

Basically, learning based on Bayes rule suggests that one should started with some overarching prior that we will gradually adjust based on incoming data. The posterior of 1st turn become the prior for the 2nd turn and so on.

$$Prior * Likelihood -> Posterior\ where$$

$$Prior = p(\theta), Likelihood = p(y|x,\theta), Posterior = p(\theta|y,x)$$

$$p(\theta, y|x) = p(\theta)p(y|x,\theta)$$

$$Normalizing joint probability : p(\theta|y,x) = p(\theta, y|x) / \int p(\theta, y|x) d\theta$$

Bayesian ML has 2 advantages: Prior could determine outcome with few data points. Posterior expectation minimizes loss.

For Bayesian Linear Regression, if our prior and likelihood are both normal, then posterior yields the outcome that is equivalent to normal equation outcome that minimize least square.

We use Hierarchical Linear Regression to tackle the case where data can be broadly divided into several groups, so we can apply neither a oversimplified uniform model nor too specific individual group models.

So we have a shared θ for all and θ_j for individual group, and we have θ_j shrunk toward θ as training progresses.

We use adjustable scalar τ_j to moderate each group's effect on the overall θ. For a very different from average group of θ_j, we want $1/\tau_j$ to be small to moderate its effect on the overall model. So we want τ_j to be naturally large.

The real-world use case of Bayesian ML involves posterior credible intervals, and Thompson Sampling. The bias of Bayesian view comes from the prior, and the bias of Frequentist view comes from finite sampling. The 95 percent confidence interval seems arbitrary. The drawback of Bayesian is computational difficulty and intensity as well as the need of a prior.

4 Regularization + Generalized Linear Models

In this section we try to deal with the following issues:

- What happens if there are too many features?
- What if the outcome variable is binary?

Finding importance of inputs via linear regression

- minimizes mean squared error
- sort θ vector to find greatest value
- standardize θ_i

What if the number of features is greater than the number of data points?
Example: Crohn's Disease

- Number of features in genomic data exceeds the number of possible data points in a sample.

Why we regularize in linear regression

- When the number of features is greater than the number of data points, there are a lot of solutions given by $\mathbf{X}\boldsymbol{\theta} = \mathbf{y}$. This is a subspace in which we can move around and still perfectly predict data.
- So we need to restrict model complexity in some way.
- Possible options: throw away variables (if some variable has low variance, then it's not explaining much about $\mathbf{y}$, so we can get rid of it); combine variables (if two variables are related, maybe we could take the average of them); sequentially add variables one at a time.
- Or, we could restrict the parameters by putting some limit on the values (regularization!).

α-norm of a vector:

$$\|\boldsymbol{\theta}\|_\alpha = \left(\sum_{i=1}^{p} |\theta_i|^\alpha\right)^{1/\alpha}$$

- $\alpha = 2$: euclidean norm
- $\alpha = 1$: Manhattan Distance
- $\alpha = 0$: Maximum Norm (Take the largest absolute value)

Why is this useful?:
α-norms give a type of distance. The equation above for the α-norm of a vector is just the generalized form of the length of a vector like this:

$$\boldsymbol{x} = (x_1, x_2, ..., x_n)$$

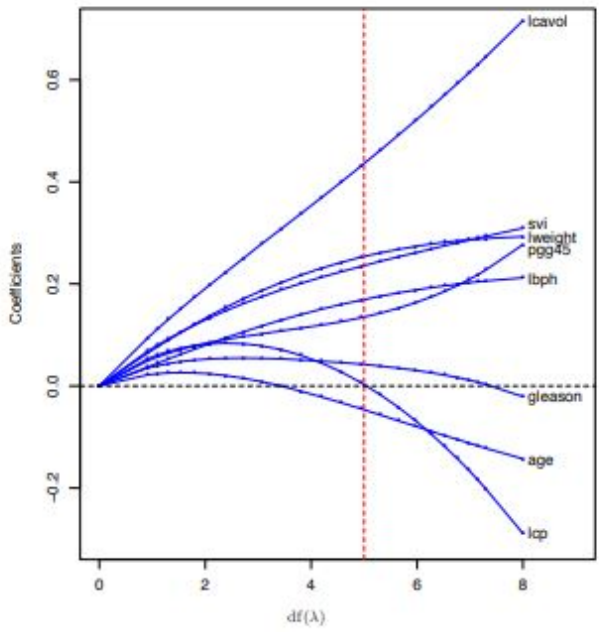
$$|x| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$$

- Wolfram MathWorld, https://mathworld.wolfram.com/Norm.html

4.1 L2 (or Ridge) regression

- Add a function to the minimization: $\mathscr{L}(\theta) = \sum_{i=1}^{n} (\theta^\top \mathbf{x}_i - y_i)^2 + \lambda \|\theta\|_2^2$
- Doing so penalizes the norm of the parameters: instead of just being close to the data (our original objective), now I *also* want the 2-norm squared of the parameters to be small.

  - If we set λ to 0, then there is no regularization. If we make λ large, then the impact of $\lambda\|\theta\|_2^2$ increases, driving up the cost, thereby giving greater incentive for your model to learn smaller weights.
- When we take the derivative and set it equal to zero, we get: $\theta = (\mathbf{X}^\top\mathbf{X} + \lambda I)^{-1}\mathbf{X}^\top y$
- Notice that when λ is large, then $(\mathbf{X}^\top\mathbf{X} + \lambda I)^{-1}$ will be small on the diagonal, so the solution is closer to 0; when λ is small, you get something that is almost the least square solution.
- Notice also that if we fix the value of λ and make n really large, then $\mathbf{X}^\top\mathbf{X} = \sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^\top$ dominates the λ term and we will get something that looks more like normal linear regression and the estimate will ignore the regularization.



[Elements of Statistical Learning]

Figure 1: L2 on different models

- With L2's use with different models, we see that sparsity of features immediately goes away as we move away from 0 for all models.

l2-regression

general regularize minimization problem:

$$\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) + \lambda g(\boldsymbol{\theta})$$

fix λ, optimal solution θ is θ^*
(We prove that this is indeed the optimal solution using proof by contradiction)

Proof: Suppose we have a $\hat{\theta} \neq \theta^*$ and:

$$f(\hat{\theta} < f(\theta^*)$$

This means that there exists a minimizer:

$$f(\hat{\theta}) + \lambda g(\hat{\theta}) < f(\theta^*) + \lambda g(\theta^*)$$

This is a contradiction, as θ^* is no longer the minimal. (If anyone has anything to add to this, feel free.)

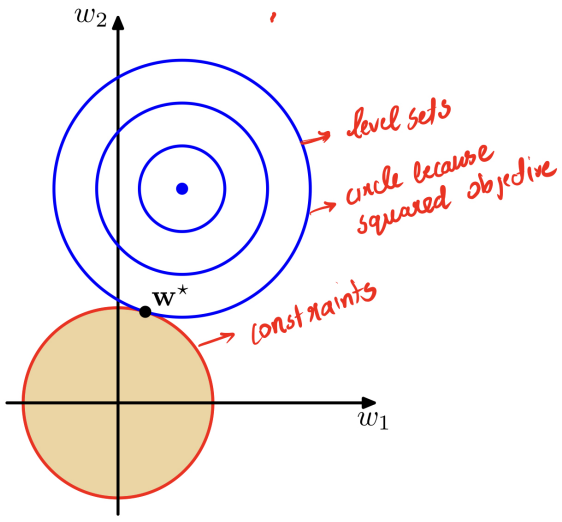


Figure 2: Understanding L2 Regularisation

Why not penalize with other norms?

Consider L1 and L2 norms: L1 leads to sparsity because of bigger penalties near zero

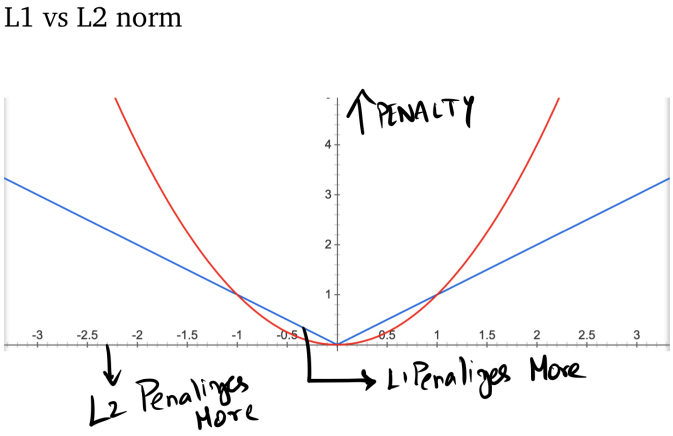


Figure 3: L1 VS L2

4.2 L1 (Lasso) regression

- $\mathscr{L}(\theta) = \sum_{i=1}^{n} (\theta^\top \mathbf{x}_i - y_i)^2 + \lambda \|\theta\|_1$
- L1 norm is bigger (and so leads to bigger penalties) than L2 closer to zero, so you would expect L1 to lead to sparer weighs (more zeroes in the vector).

- How to solve this loss function? Because the absolute value function is not differentiable at the part you care about (zero), you have to use quadratic programming, a custom solver, or advanced optimization.
- Because L1 regularization likes sparsity, it tries to pick individual features that are good predictors.
- If there are two features that are correlated and that both predict the label pretty well, then L1 regularization will just pick one of them. But that's not necessarily good, because a linear projection of the features might be more stable! For example, maybe there's some noise in both of the predictors, and if you average them, you might average out the noise, which would be good. L1 precludes that averaging, though.
- But maybe sparsity is good if you're a person looking at the parameter vector, since there will be less to look at.

L1: Lasso

More likely to happen along the axes

Understanding the L1 regularization:
- With correlated features, regularization parameter selects one

4.3 Elastic net regression

- Combine L1 and L2!

$$\mathscr{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} (\boldsymbol{\theta}^\top \mathbf{x}_i - y_i)^2 + \lambda(\alpha\|\boldsymbol{\theta}\|_1 + (1-\alpha)\|\boldsymbol{\theta}\|_2^2)$$

- Elastic net works well in practice, and so can be a good baseline.
- You still get peaks on the axes, but the rest is rounded, which is good.

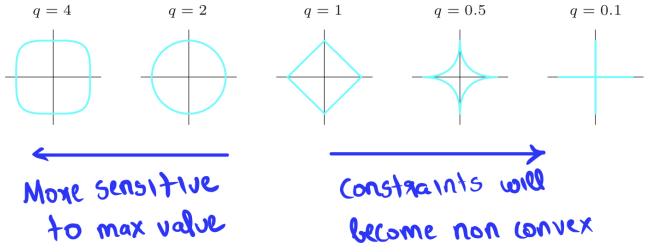


Figure 4: Regularizers' Graphs

4.4 Bias variance trade-off: training data as a random variable

- We said in Lecture 2 that when f is not linear, then the optimal f is the conditional expectation: $\min_f \mathbb{E}_{p(\mathbf{x},y)}[(y - f(\mathbf{x}))^2] = \min_f \mathbb{E}_{p(\mathbf{x})}[(\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x}))^2] + \mathbb{E}_{p(\mathbf{x})}[\mathbb{V}\text{ar}(y|\mathbf{x})]$.
- But we also just saw that things get complex when the number of features is bigger than the number of data points. So where does regularization fit in here? There's no idea of overfitting or memorizing your training data here.
- We need to consider where the training data came from, so we'll treat it as a random variable: $\mathscr{D} = (x_1, y_1)...(x_n, y_n) \sim p$.
- Example: Imagine wanting to predict heights from diet. I get $\mathscr{D}_1$ by doing a random survey of the heights and diets of 500 people. I get $\mathscr{D}_2$ by doing another random survey of 500 people. $\mathscr{D}_1$ and $\mathscr{D}_2$ are random because the full population was not surveyed, but $\mathscr{D}_1$ and $\mathscr{D}_2$ were both drawn from the same distribution.

- So now the optimal f is: $\min_f \mathbb{E}_{p(\mathbf{x},y),\mathscr{D}\sim p}[(y - f(\mathbf{x};\mathscr{D}))^2]$. We're asking for a predictor that is best on average, but that also respects the fact that the predictor is a random variable because it's trained on $\mathscr{D}$.
- Bias:
  - This term accounts for model mismatch of average predictor.
  - There's no way to calculate bias directly; it's just an intuition to have when you build models.
- Variance:
  - This term captures how much the predictor changes with respect to the average predictor.
  - This accounts for finite data. If you had infinite data, the random variable would be close to the expectation.
- Noise:
  - This term is the fundamental prediction error. We'll never be able to predict noise in the data from y given $\mathbf{x}$. Noise accounts for y not being a deterministic function of $\mathbf{x}$: y is not perfectly predictable from $\mathbf{x}$.
  - Your model can fail to generalize for three reasons: (1) if your model is incorrect (bias), (2) if you need a lot of data to train your model (variance), or (3) if your features don't predict your label (noise).
- Big vs. small models
  - We have two model classes here: $\mathscr{M}^{\text{big}}$ and $\mathscr{M}^{\text{small}}$. An example of $f^* \in \mathscr{M}^{\text{big}}$: using the L2 norm without any regularization. An example of $f^* \in \mathscr{M}^{\text{small}}$: constraining L2 norm to be small. The optimal predictor f^* lives inside $\mathscr{M}^{\text{big}}$ but outside $\mathscr{M}^{\text{small}}$.
  - Small models tend to...
    * Have more bias. Bias means: even if you give me a lot of data, the best I can do is still in the yellow circle; I can't get any closer to f^* than that. When you use two different datasets, the results have less space to move around in $\mathscr{M}^{\text{small}}$.
    * Underfit. They can't fit data well enough to get to the truth.
    * Small models are well regularized and constrained.
  - Big models tend to...
    * Have more variance. When you use two different datasets, you can follow the whims of the data: if the average of what I collect is large, then my result will be large (vice versa if small).
    * Overfit. They memorize the data, but may not generalize well because of variance.
    * Complex models have a lot of freedom, which means the model is weakly regularized, and the space we can live in is small.

Small models: more restriction, less variance, more *bias*, tend to *underfit*

Big models: more *variance*, *overfit*

4.5 Maximum Likelihood

- The likelihood function measures "probability" of data: $\ell(\mathbf{x}_i;\boldsymbol{\theta})$
- When data are independent, we get $\prod_i \ell(\mathbf{x}_i;\boldsymbol{\theta})$
- Generally work with logs for stability $\sum_i \log \ell(\mathbf{x}_i;\boldsymbol{\theta})$ and stochastic optimization for speed.
- Suppose we flipped a biased coin many times and got samples $x_i \in 0, 1$. We want to compute the maximum likelihood estimate for the probability of heads of the coin. How do we start?
- Recipe for calculating maximum likelihood:
  - (1) Need a distribution over each observed data point.
    * $x_i \sim \text{Bernoulli}(p) = p^{x_i}(1-p)^{1-x_i}$.
    * Bernoulli is the only choice here since the only degree of freedom is the probability (we can only specify the probability).

- (2) Write down the log-likelihood of the n observations.
  - $\mathcal{L}(p) = \sum_{i=1}^{n} \log(p^{x_i}(1-p)^{1-x_i})$
- (3) Maximize the log-likelihood by taking derivatives/gradients and setting to zero.
  - What happens when you only observe zeros? Then the maximum likelihood solution is going to give you zeros.

Maximum Likelihood Example Question:

- need distribution over coin flips

Example. MLE: Bernoulli

Example 4.4. (i) $X_1, ..., X_n$ i.i.d., $X_1 \sim Ber(p)$ $(\theta = p)$. We have

$$\begin{aligned} p_\theta^{(n)}(x_1, ..., x_n) &= \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i} = \theta^{\sum_{i=1}^n x_i}(1-\theta)^{n-\sum_{i=1}^n x_i} \\ \Rightarrow \frac{\partial}{\partial\theta} p_\theta^{(n)}(x_1, ..., x_n) &= \left(\sum_{i=1}^{n} x_i\right) \theta^{\sum_{i=1}^n x_i - 1}(1-\theta)^{n-\sum_{i=1}^n x_i} \\ &\quad - \theta^{\sum_{i=1}^n x_i}\left(n - \sum_{i=1}^{n} x_i\right)(1-\theta)^{n-\sum_{i=1}^n x_i - 1} \stackrel{!}{=} 0 \\ \Leftrightarrow \theta &= \frac{\sum_{i=1}^n x_i}{n}, \end{aligned} \tag{4.9}$$

so $\widehat{\theta}_n = \overline{X}_n$ is the MLE.

1. Need distirbution over each observed data point 2. write down log-likelhiood of observations 3. take derivate and set equal to zero

4.6 Logistic regression

- Logistic regression is: maximum likelihood for binary outcome linear regression
- (1) Need a distribution over each observed data point.
  - $p(y_i = 1|\mathbf{x}_i; \theta) = \frac{1}{1+\exp(-\theta^\top \mathbf{x}_i)} := \sigma(\theta^\top \mathbf{x}_i)$
  - σ is the sigmoid or logistic regression function, which maps from $\mathbb{R}$ to $(0, 1)$ (a probability)
- (2) Write down the log-likelihood of the n observations.
  - $\sum_{i=1}^{n} \log(p(y_i = 1|\mathbf{x}_i; \theta)) = \sum_{i=1}^{n} y_i \log\sigma(\theta^\top \mathbf{x}_i) + (1-y_i)\log(1-\sigma(\theta^\top \mathbf{x}_i))$
- (3) Maximize with respect to θ
  - Use $\sigma'(a) = \sigma(a)(1-\sigma(a))$
  - $\nabla_\theta \mathcal{L} = \sum_{i=1}^{n} \mathbf{x}_i(y_i - \sigma(\theta^\top \mathbf{x}_i))$
  - More intuitive form: $\nabla_\theta \mathcal{L} = \sum_{i=1}^{n} \mathbf{x}_i(y_i - \mathbb{E}[y_i|\mathbf{x}_i])$. $\sigma(\theta^\top \mathbf{x}_i)$ is the probability that $y_i = 1|x_i$, which is its expectation.
- Logistic regression assumptions
  - Logistic regression is just one way that we could have mapped from $\mathbb{R}$ to $(0, 1)$. For example, the CDF of Gaussian. So why use logistic regression?
  - The odds of an event A are: $\text{odds}(A) = \frac{P(A)}{P(A^c)}$. For example, if $P(A) = \frac{2}{3}$, we say the odds in favor of A are 2 to 1 (to $2:1$). Log odds: $\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$.

– Notice that $\text{logit}(p(y=1|\mathbf{x})) = \log\left(\frac{\frac{1}{1+\exp(-\theta^\top \mathbf{x}_i)}}{\frac{\exp(-\theta^\top \mathbf{x}_i)}{1+\exp(-\theta^\top \mathbf{x}_i)}}\right) = -\log(\exp(-\theta^\top \mathbf{x})) = \theta^\top \mathbf{x}$. Therefore, the features contribute linearly in the logit of probabilities.

What happens when we consider noise to be from the logistic distribution?

Because the logistic distribution has heavier tails than the normal distribution, the normal distribution will fit things that are extreme a little bit less!

4.7 Generalized linear models

- Our outputs can come in all different forms: $\mathbb{R}$, binary, counts (e.g. number of doctor's visits). Instead of having to derive a new model for each type of y, we can get a generic distribution $p(y|\mathbf{x})$ that works for all kinds of y's using exponential families.
- Exponential families are a distribution over a lot of types of distributions: Bernoulli (for binary values), Normal (for real values), Gamma (for positive values), Categorical (multiple types).
- Both linear and logistic regression are examples of GLMs.
- $p(\mathbf{x}) = h(\mathbf{x})\exp(\eta^\top t(\mathbf{x}) - a(\eta))$
  - h: the base measure (can be absorbed into underlying density)
  - η: parameters
  - $t(\mathbf{x})$: sufficient statistics
  - $a(\eta)$: log-normalizer (to ensure that this will integrate to 1)
    - You can solve for the log-normalizer by integrating (or summing) the unnormalized density of the distribution: $a(\eta) = \log \int h(\mathbf{x})\exp(\eta^\top t(\mathbf{x}))d\mathbf{x}$
- An important property of the exponential family: if you take the gradient of the log-normalizer, you get the expectation of $t(\mathbf{x})$: $\nabla a(\eta) = \int h(\mathbf{x})t(\mathbf{x})\exp(\eta^\top t(\mathbf{x}) - a(\eta))d\mathbf{x} = \mathbb{E}[t(\mathbf{x})]$

$$
\begin{aligned}
\nabla a(\boldsymbol{\eta}) &= \nabla \log \int h(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x} \\
&= \frac{\nabla_\eta \int h(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x}}{\int h(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x}} \\
&= \frac{\int h(\mathbf{x})\nabla_\eta \exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x}}{\int h(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x}} \\
&= \frac{\int h(\mathbf{x})t(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x}}{\exp(\log(\int h(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x}))} \\
&= \frac{\int h(\mathbf{x})t(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}))d\mathbf{x}}{\exp(a(\boldsymbol{\eta}))} \\
&= \int h(\mathbf{x})t(\mathbf{x})\exp(\boldsymbol{\eta}^\top t(\mathbf{x}) - a(\boldsymbol{\eta}))d\mathbf{x}
\end{aligned}
$$

Exchanging derivative & integral based on rule below ←

exp(a(η)) → constant w.r.t integral

Figure 5: Steps of obtaining above property

- To build a regression model with exponential families, follow the recipe of maximum likelihood:
  - Use exponential families for the conditional distribution: $p(y_i|\mathbf{x}_i) \sim$ Expfam
    - Side note: How do you design conditional distributions? $p(y|\mathbf{x}) \sim \text{Normal}(\mu, \sigma)$
    - If the parameters are fixed, that will be a bad model, because there's no dependence on $\mathbf{x}$. To have a good conditional distribution, you want the parameters to be some function of $\mathbf{x}$. So in this case you can make the mean and variance functions of the conditioning variable:
    - $\mu(\mathbf{x}_i) = f_\theta(\mathbf{x}_i)$
    - $\sigma(\mathbf{x}_i) = \log(1 + \exp(g_\theta(\mathbf{x}_i)))$
  - Maximum likelihood to estimate parameters

Can we always interchange derivative and integral? - One may check Leibniz integral rule.

Complex Models Regularized by parameter size:
- Complex models = lots of freedom, weakly regularized, λ is small

L1 Regularization finds sparse solutions:
Linear and logistic regression are examples of GLMs
GLM plus regulariation allow regression on all kinds of data

5 Neural Networks

- Neural networks are like sandwiches, made of many layers.
- We can think of Neural networks as the stack of linear and non-linear layers

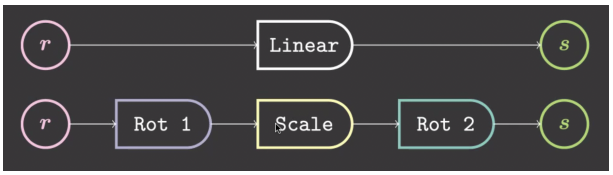


Figure 6: Breaking down linear transformation

As we can see from 6 Linear transformation can be split into (The singular value decomposition of the liner transformation):

1. Rotation 1 (V)
2. Scaling (Σ)
3. Rotation 2 (U)

Transformed output s can be written as ,

$$s = Wr \tag{1}$$

$$W = U\Sigma V^T \tag{2}$$

$$s = U\Sigma V^T r \tag{3}$$

What linear transformation does

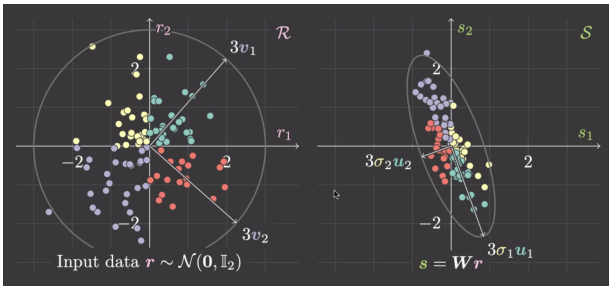


Figure 7: Transformation visualization

1. Squash and Rotate
2. Takes circumferences and makes it ellipses
3. Global transformation

This can be observed from the right figure in 7

Stack several linear layers on top of another makes a big linear layer- not interesting

Add something in the middle- nonlinear operations

5.1 Non-linear layers

1. ReLU
2. tanh
3. sigmoid

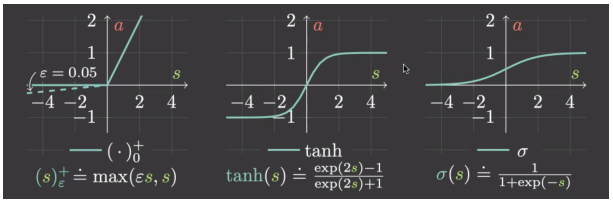


Figure 8: ReLU, tanh and sigmoid non linearities

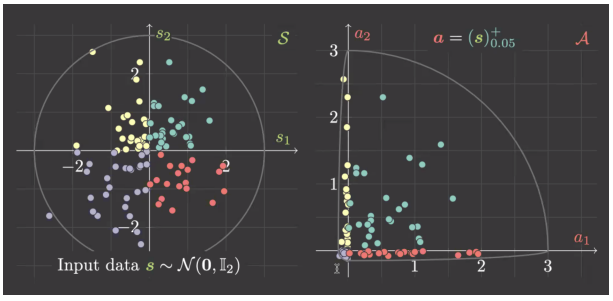


Figure 9: Left: Original Normal Distribution, Right: ReLU transformation

5.1.1 Non-linear operations-

1. Select a region- send it through or compress it
2. ReLU, tanh, signmoid, Leaky ReLU
3. Component wise functions- acts on element of a vector

Note

- linear layers affect all the inputs globally while the non-linear layers change the input based on their region in the space

ReLU

1. Quadrant 1 points stay the same because they are positive
2. Quadrant 2 points- y values stay same, but x values get squashed because they are negative

Tanh

1. Square of width 2 centered at origin
2. Creates kind of edges instead of original circle normal distribution

Sigmoid

1.
2.

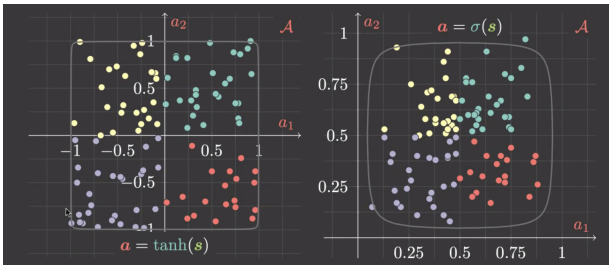


Figure 10: Left: tanh transformation, Right: Sigmoid transformation

5.2 Multi Layer perceptron

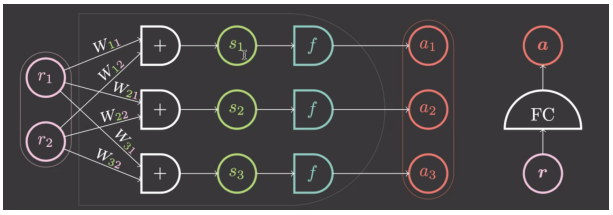


Figure 11: MLP on right, sh ort form representation of the same

The weight matrix will be 3X2 for the above MLP

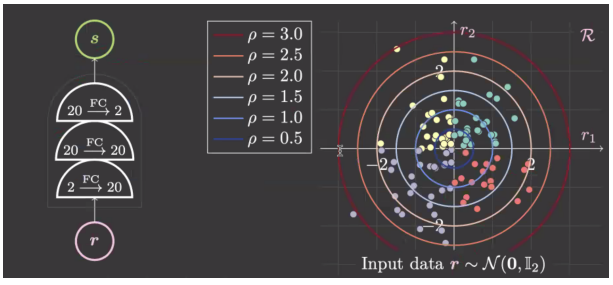


Figure 12: Multiple layer transformation

Q: Why are we sending multiple circle?
As the layers progress the transformation changes, which is shown using the multiple circles. ReLU is more crumbled as compared to tanh. Hence ReLU better able to transformations.

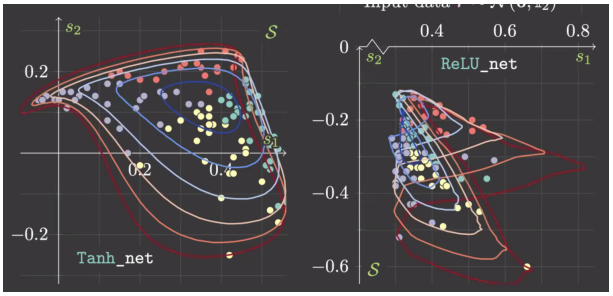


Figure 13: Multiple layer transformation

5.3 ANN

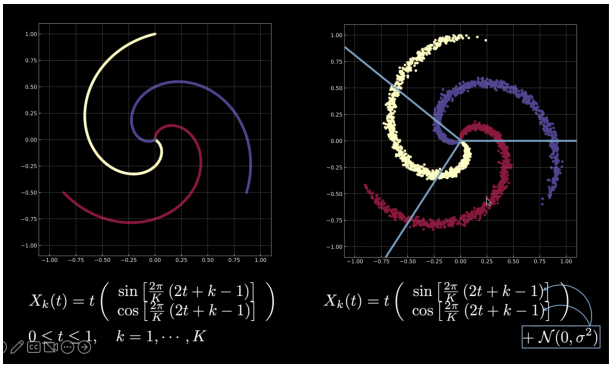


Figure 14: Problem statement

The problem statement is for model to be able to tell the color of the point based on the location.

input x - horizontal, vertical co-ordinates output y - colour

Simplest approach- Try to draw linear boundaries one for each colour

Problem with using linear regression - data is not linearly separable

Possible fixes- Try to transform the data to make it linearly separable

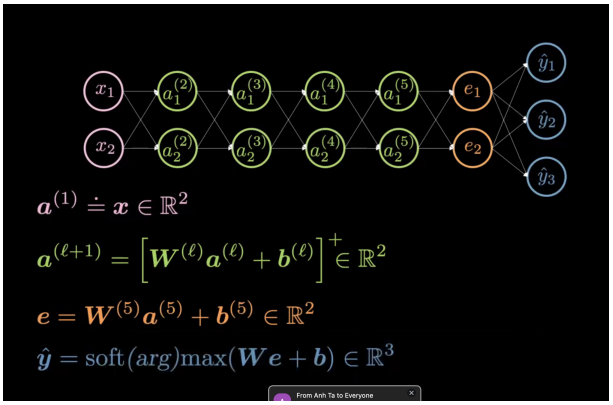


Figure 15: Network representation

More classes and with more width in the network leads to more smoother transformation in animation.

5.3.1 Neural Network Inference

Send x into a linear layer followed by activation function (goes from 2 to 100) then into a decoder which makes it from 100 to number of classes 5.

$$\hat{y} = softargmax_{\beta}(s) = \frac{\exp(\beta_s)}{\sum_{k=1}^{K} \exp(\beta_{s_k})} \in (0,1) \tag{4}$$

Take the output from decoder and the true y value to calculate Cost Function- price paid for varying from true y.

Cost- how far predicted is from true value (use distance metric)

Loss- how bad parameters are on training/validation/test set or for a specific sample. It is given by $\mathscr{L}(w,S) = 1/P \sum_{p=1}^{P} \mathscr{L}(w, x^{(p)}, y^{(p)})$, where S is the training set and P is the total number of training datapoints.

Use gradient descent or any optimizer to reach the minimum of the loss function. It uses chain rule of derivatives to decide in which direction to move to achieve the minima. The step size for this movement is decided by the learning rate.

5.4 Convolutional Neural Nets

Input layer

$$\chi = \{x^{(i)} \in \mathscr{R}^n | x^{(i)} \text{is a data sample}\}_{i=1}^{m} \tag{5}$$

Locality: means that neighboring pixels tend to be correlated, while faraway pixels are usually not correlated.

Stationary: mean that the values of the pixels do not change over time

5.4.1 Locality => sparsity

Yann was able to train image recognition via CNN which used locality property of the images.

Gives reduced amount of computation.

5.4.2 Stationarity => Parameter Sharing

Share paramenters- go from 15 to 3 weights. Advantages-

1. Faster convergence
2. Better generalization
3. Not constrained to input size
4. Kernel independence
   high parallelisation

5.5 Standard spatial CNN

Convert high width images to deeper thinner information layers

1. Multiple layers
   Convolution
   Non-linearity (ReLU and Leaky ReLU)
   Pooling
2. Residual Bypass Connection

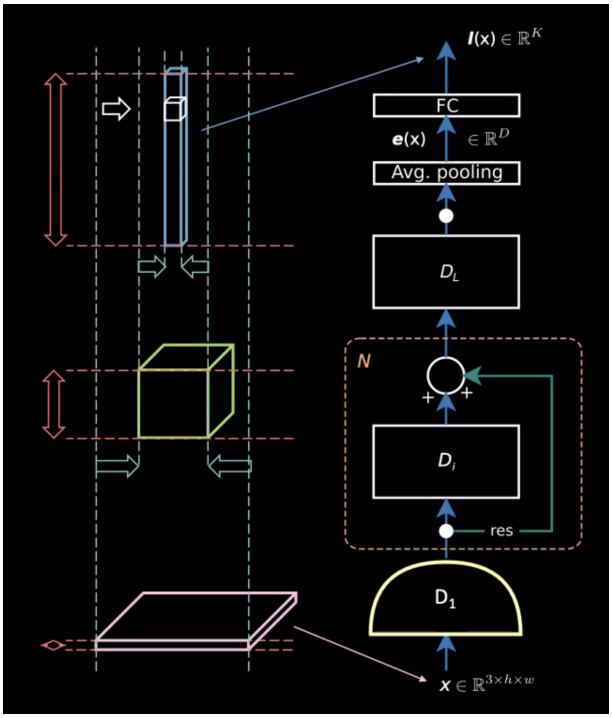


Figure 16: CNN representation

5.6 Generalisation Error Reduction

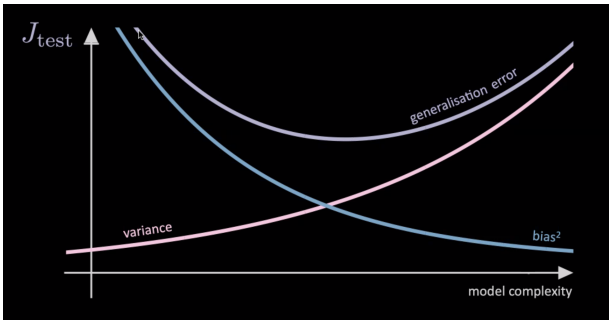


Figure 17: Generalisation error

- The figure 17 shows the trade-off between bias and variance. As the model becomes more complex the variance and the generalization error increases while the bias decreases.
- If the model is misspecified we can have CNN parameter space way off the actual prediction.

6 Trees and Forests

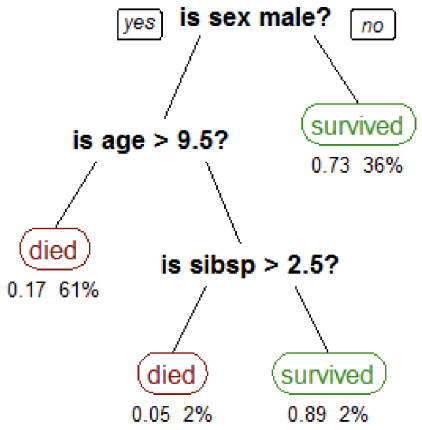
dsfont

6.1 Linear Model

1. Linear models can be non-linear
2. $y_i = \theta_1 x_i + \theta_0 + e_i$ (linear features)
3. $y_i = \theta_1 x_i + \theta_2 x_i^2 + \theta_0 + e_i$ (non linear features)

4. Still a linear problem in this new feature space, parameters can be learned with generalized linear models
5. Linear models can be become non-linear: (1)Adding higher order features to the feature matrix (2)Estimate the parameters of the augmented model
6. Linear model can include interaction between and still can be trained by generalized linear models

6.2 Decision Trees

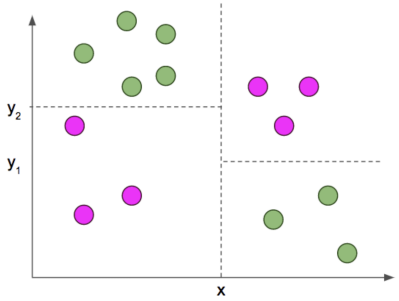


• Why we want trees?
Since, we want some nonlinear model and decision trees introduce non-linearity in the features.

• Decision tree basics
1. Work by dividing the feature space into different regions
2.Different Predictions in each region
3.Can be done for classification or regression: $pred(x^*) = \sum_{i=1}^{N} 1[x^* \in leaf_i] * E[y|x \in leaf_i]$, where N is the number of total leaves.

• Is there an optimal tree?
There are so many features can be chose to split on each level and so many levels, we can't find a optimal one, just use a greedy algorithm to find a tree

Greedy Tree Building: Basic Idea



1. Figure out a rule to split the input space
2. Split the space according to the split from (1)
3. Start over again in each new subspace

• Classification Tree Example
Assume all features x are continuous.

For each feature j
1. Find a split point cj that minimizes classification error in two subsets.
Prediction

$$prediction(A_j) = majority(x_j \epsilon A_j)$$

Error on split:

$$min_{cj} 1/n \sum_{i=1}^{n} 1[\text{prediction}(x_{i,j} \leq cj) \neq y_i] 1\ [x_{i,j} \leq c_j]$$

2. Choose the feature j^* with the lowest classification error and split the data at c_j^*

• How to learn the decision tree?
1. Naive approach: try find the optimal splitting condition at each step by minimize the error on split.
This approach is problematic since it will lead to overfit if we do not control the data points in each leaf. But even with some fix this method still has too high variance to be practical.
2. Random Forest: The basic idea of this method is that, we randomly build trees by using data subsamples and a subset of features for each tree, then we use the average(continuous output) or majority(discrete output) of these trees' predictions to serve as our final prediction.
This method learns fast, and does not cause overfit problems as we add more trees. But because of the randomness it may get stuck in some local minimum points.
3. Gradient Boost:
this method build new trees by project the functional derivative to a tree.In detaile just fit the functional derivative value and corresponding x value to form a new tree. And then update the predict function by moving toward the negative direction of the new tree result. This method move toward the derection we are interested in instead of randomly but will cause overfit as we train forever.

6.3 Random Forests

Build a random tree:

$$h(x, \theta_k), where \theta_k \sim q \tag{6}$$

Aside: Generalization Error:

$$E_{(x^*,y^*)\sim p}[Error(y^*, f(x^*))] \tag{7}$$

We can use some data to evaluate generalization error, but do not use training data since it biased and cannot test the overfitting problem

Suppose the random forest has K trees, define:

$$margin(x, y) = \frac{1}{K} \sum_{k=1}^{K} 1[h(x; \theta_k) = y] - \frac{1}{K} \sum k = 1^K \frac{1}{K} 1[h(x; \theta_k) = 1 - y] \tag{8}$$

and the generalization error is:

$$ge = E_{(x^*,y^*)\sim p}[margin(x^*, y^*) < 0] \tag{9}$$

For large number of trees, we have:

$$ge = E_{(x,y)\sim p}[E_{\theta\sim p}[1[h(x; \theta) = y] - 1[h(x; \theta) = 1 - y] < 0]] = P_{(x,y)}[margin(x, y) < 0] \tag{10}$$

Define the strength of the random forest:

$$s = E_{x,y}[margin(x, y)] \tag{11}$$

Assume S > 0, by Chebyshev:

$$ge = P_{(x,y)}[margin(x, y) < 0] \leq \frac{Var[margin]}{s^2} \leq \frac{\rho(1-s^2)}{s^2} \tag{12}$$

Good forest has:
• Low correlation between trees (ρ)
• High strength of trees (s)

Suppose:
• n features
• Half of the features are chosen for each split
• Combination of half of the features determine class
• Very rough estimate for building correct tree is exponential in n

Random forests rely on randomness to figure out what's important, and this randomness helps prevent overfitting with more compute, makes learning super fast, makes finding important corners hard

7 Latent Variable Models

7.1 Introduction

We want to model the inputs to fill in the x_{ij}, so we actually need :

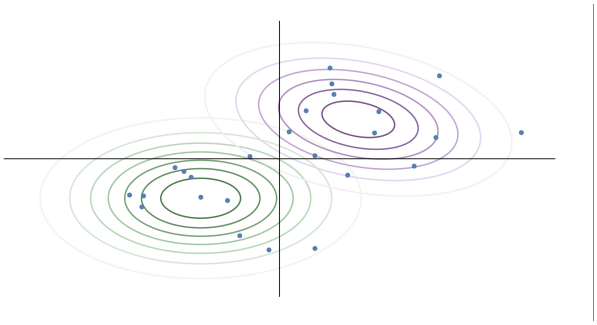
$Model = p(x, y)$

Given we have missing features, we need the expression:

$$p(x_{missing}|x_{observed})$$

If x and y are just random variables, finding p(y|x) is just like filling in y (which are the missing variables in the random variables we have).

7.2 Latent Variables

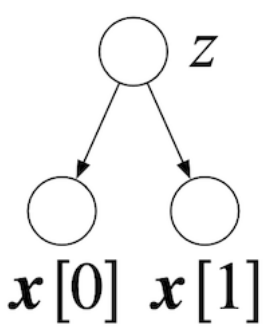


1. Latent variables represent the topical structure or summary of the data we observe
2. LVs represent by z, p(x,z) is the model we want.
3. We look at a sample case, where the data is coming from 2 Gaussians, and we want to identify which gaussian the data is coming from.
4. Define the latent variable to be either of class 1,2,3....K, with a categorical probability prior probability of $\frac{1}{K}$

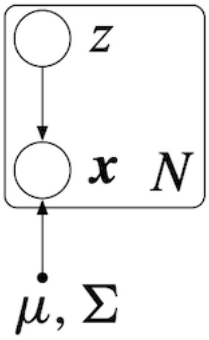
5. If we know the color, we know how the gaussian looks like formally $p(x|z=k) = N(\mu_k, \sigma_k)$
6. The μ_k, σ_k are the parameters, which we try and train by maximum likelihood, of $log(p(x_1, x_2....x_k; \mu_k, \sigma_k))$
7. One way of maximizing the likelihood is minimizing the KL divergence and minimizing KL divergence moves the model close to the data-generating distribution.
8. Even if we use a more powerful mixture model (i.e. non-gaussian) and minimize the KL divergence, the average of the mixture model converges close to the data-generating distribution.
9. Even in mixture models, $p(z_i = k|x)$ reduces to $\frac{1}{K}$.

7.3 Latent Variable Models

1. For a data generating distribution $F(x)$ and a latent variable model $p(x,z)$, there are more variables than there are constraints. $\Rightarrow$ multiple solutions are possible.
2. For independent distributions $F(x)$ and $p(z)$, the KL divergence will always be zero. So assumptions (constraints) are needed to solve.
3. Limitations are placed by making the latent variable responsible for the data-generating process. With limitations, we can only fit independent data.
4. A model can also be represented graphically. For example, graph (a) describes the conditional independence of $x[0]$ and $x[1]$ given z. And (b) describes a gaussian distribution.



(a)



(b)

5. A mixture of gaussians gives us information about the class of a data point. A mixture of arbitratry models doesn't place any limitations so the latent variables are unneeded to model the data.
6. A mixture of Gaussians is not a Gaussian and the mixture of arbitrary distributions is an arbitrary distribution.
7. To evaluate our model, we just calculate $log(p(x))$ on a held-out data sample.We know that some of these log probabilities on held-out set of data look like maximum likelihood and if we put this into average we can see that this quantity is actually an estimate of the negative KL divergence.

$$\sum_{i=1}^{N} \log p(\mathbf{x}_i), \quad \mathbf{x}_i \sim p$$

8. **Question**

   Question: What would be a good latent variable model where there's an underlying state that's responsible for dependence in a sequence of noisy measurements?

   Example- Imagine a bad sensor is measuring temperatures. Some random noise like gaussian noise are observed in the readings. The observed reading could be related to previous temperature, maybe yesterday's temperature or maybe the last year's temperature on the same day. It could be anything.

What we truly want to see is the true temperature. So, the hidden variable is true temperature. The underlying variable is actually tracking the true thing we would like to see. Whenever we take a reading, the noise keeps adding up. We end up getting the chain of related latent variables and have all the observations hanging off of them.

9. **Formal definition**
   For Probabilistic latent variable models, we have:
   - Data: x (example- noisy temperatures)
   - Hidden structure(latent variables): z (example- true temperatures)
   - Model: $p(x,z) = p(z)p(x|z)$
     where $p(z)$ is prior (what our belief is on those temperatures before observation) and $p(x|z)$ is the likelihood (how likely are those observed values given a particular state of that belief.
   - Posterior: $p(z|x)$- probability of the hidden state given your observation.

     Example Another example could be finding topics in documents.
     - The data is: N number of documents, V number of words and $W : N * V$ matrix of words.
     - The Hidden Structure that we would like to find is a group of objects that describe the document. Each document contains a distribution over topics.

     To describe topics, the topics have to be a distribution over the words called β_k.
     To describe the documents composition over topics: distribution over the topics called θ_i.

     So, now we have two distributions:
     - β_k distributions over words for each topic
     - θ_i distribution over topics for each document.

     The prior here is going to be a distribution over vectors that sum to one. For example- Dirichlet distribution because the conjugate prior for the multinomial distribution is the Dirichlet distribution.
     We are yet to find the likelihood of the data.

     For each topic: we will draw distribution over words from Dirichlet (α). For each document: we will draw distribution over topics from Dirichlet(κ)
     For word m in document l:
     First draw word's topic from $z_{m,l}$ $Categorical(\theta_i)$. Then draw word from topic: $w_{m,l}$ $Categorical(\beta_{z_{m,l}})$.

     At the end, compute the posterior:

$$p(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{z} \mid \mathbf{w}) = \frac{p(\boldsymbol{\beta}, \boldsymbol{\theta}, \mathbf{z}, \mathbf{w})}{p(\mathbf{w})}$$

     However, it is difficult to compute this posterior distribution in topic model.

8 Computing with Latent Variable Models

8.1 Introduction

Q: Why latent variable models? A: Latent variable models can be used to infer the class label of an input when there is no ground truth training label to build a supervised classifier from

Example: Classifying types of objects in space

We have prior knowledge about

$$p(x_i : \text{light measurement} | z_i = \text{galaxy}) = \text{Known from physics}$$

Posterior over mixture component "classifies":

$$p(z_i = \text{galaxy} | x_i) = \frac{p(z_i = \text{galaxy}) p(x_i | z_i = \text{galaxy})}{\sum_{\text{type} \in \{\text{galaxy, planet, ...}\}} p(x_i | z_i = \text{type}) p(z_i = \text{type})}$$

Uses of latent variable models:

- Encoding prior knowledge (e.g. physics in the astrophysics example above)
- Combining simple distributions to create a more complex one (e.g. modeling conditionally independent variables)
- Uncovering hidden structure (e.g. topic models)

8.2 Combining simple distributions

Take a categorical distribution

$$\text{Categorical}(1 \dots K)$$

Take a normal distribution

$$\mathcal{N}(\mu, \sigma)$$

Combine

$$z_i \sim \text{Categorical}(1 \dots K)$$
$$x_i \sim \mathcal{N}(\mu_{z_i}, \sigma_{z_i})$$

We get a mixture of Gaussians, which is far more flexible.

8.3 Q&A from last year

Do latent variables help predict x?

What does it mean to predict x?

In some sense, yes.

$$p(x) = p(x^{(1)}) p(x^{(2)} | x^{(1)}) p(x^{(3)} | x^{(1)}, x^{(2)}) ...$$

How do I know my latent variable is correct? Can I use cross-validation like when we predict Y?

This cannot be checked without assumptions, because:

- The data generating distribution $F(x)$ is all you get from the data
- $F(x)$ can be modeled without latent variables (though maye slow)
- For a fixed class, predictions and predictive checks help: models that assign a higher log probability would be a better fit given the same model class.

How can we create graphs?

- Based on prior knowledge (e.g. astrophysics example above)
- Based on computational considerations (e.g. if some things are hard to compute with you may not want to use them)
- Based on hidden structure useful for a problem (e.g. topics, inferring connections within the brain, genomics)

Where does one use latent variable? Can we have some more real-world examples?

- Feature engineering on light source classifications; we don't know the label of different light sources. Maximum vs. average(gives avg conditional expectation)
- Will talk about more in a second

Are latent variables about the noise and getting an accurate data generating distribution?

Yes and no.

- Noise variables could be latent variables
- Noise is part of the unpredictability of x. How can the first dimension of x be predicted below?
$$p(x) = p(x^{(1)})p(x^{(2)}|x^{(1)})p(x^{(3)}|x^{(1)}, x^{(2)})...$$
- Latent variables can exist in convert with noise variables (Gaussian noise)
- Latent variables could also be structure that is unobserved

8.4 Uncovering hidden structure: finding topics in documents

Data:

- M number of documents
- V number of words
- $W : M \times V$ matrix of word counts

Hidden Structure:

- A group of topics that describe the documents
- Each document contains a distribution over topics (for each document, what topics are in it?).

How do we describe the topics? As a distribution of the words called β_k.

How do we describe the document's topic composition? As a distribution over the topics called θ_i.

Have two distributions:

- β_k distributions over words for each topic
- θ_i distribution over topics for each document.

Note all distributions sum to 1. As they are random variables, they need priors.
How to build such distribution?

- Multivariate normal? Sampling will yield a vector of real numbers. To obtain a distribution, a deterministic transformation is needed. Softmax will do.
- Dirichlet distribution.

Still need a likelihood for data

- M number of documents
- V number of words
- $W : M \times V$ matrix of word counts

with hidden structure

- β_k distributions over words for each topic
- θ_i distribution over topics for each document.

Assume all documents have the same length for simplicity.

Topic model

For each topic: Draw distribution over words from Dirichlet(α)

For each document: Draw distribution over topics from Dirichlet(κ)

For word m in document l

1. Draw word's topic from $z_{m,l} \sim \text{Categorical}(\theta_i)$
2. Draw topic from $w_{m,l} \sim \text{Categorical}(\beta_{z_{m,l}})$

Compute posterior:

$$p(\beta, \theta, z|w) = \frac{p(\beta, \theta, z, w)}{p(w)}$$

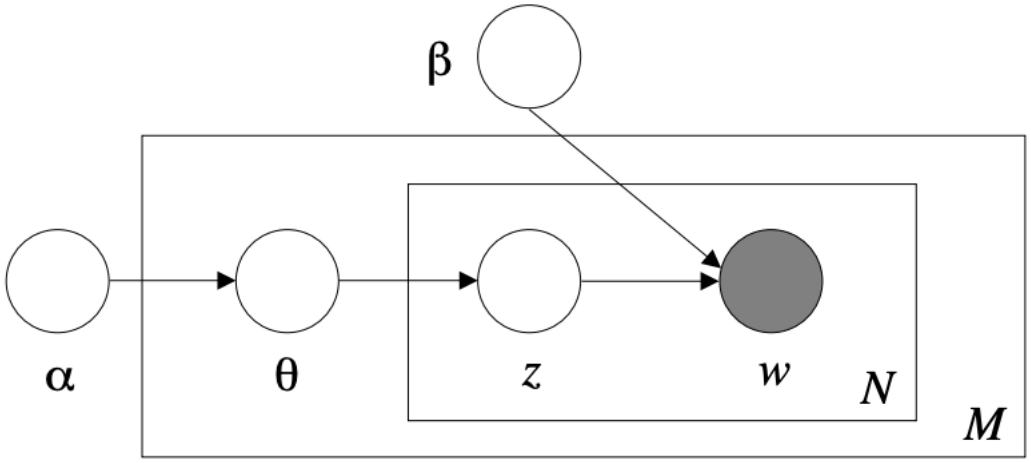


Figure 18: Graph: conditional dependencies

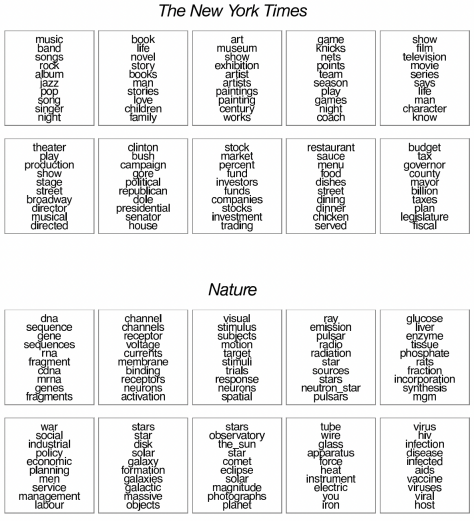


Figure 19: Result of applying the algorithm

Computing the posterior. Is it easy?

8.5 Computing the posterior is hard

The posterior distribution

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

The model $p(x,z)$ is given; the challenge lies in computing $p(x)$

$$p(x) \int p(x,z)dz$$

Bayesian Mixture of Gaussians:

$$\begin{aligned}\mu_k &\sim \mathcal{N}(0,1)\\ z_i &\sim \text{Categorical}(1\dots K)\\ x_i &\sim \mathcal{N}(\mu_{z_i},1)\end{aligned}$$

Marginal likelihood is

$$p(x) = \int \prod_{k=1}^{K} p(\mu_k) \prod_{i=1}^{N}\prod_{j=1}^{K} p(z_i=j)p(x_i|\mu_j)d\mu_1 ... d\mu_k$$

Swapping

$$p(\mathbf{x}) = \sum_z \prod_k \int p(\mu_k) \prod_{i:z[i]=k}^{N} p(z_i=k)p(x_i|\mu_k)d\mu_k$$

Show the equality

$$\begin{aligned}p(\mathbf{x}) &= \sum_z \prod_k \int p(\mu_k)\Pi_{i:z[i]=k}^{N} p(z_i=k)p(x_i|\mu_k)d\mu_k\\ &= \int \underbrace{\prod_k p(\mu_k)}_{\text{const.}} \sum_z \prod_i p(z_i=z[i])p(x_i|\mu_{z[i]})d\mu_1\dots d\mu_k\\ &= \int \sum_z \prod_k p(\mu_k) \prod_i p(z_i=z[i])p(x_i|\mu_{z[i]})d\mu_1\dots d\mu_k\\ &= \sum_z \int \prod_k p(\mu_k) \prod_i p(z_i=z[i])p(x_i|\mu_{z[i]})d\mu_1\dots d\mu_k\end{aligned}$$

(someone please double check)

Uses $\int_{a,b} f(a)g(b) = \int_a f(a) \int_b g(b)$
Integrate naively in $\mathbb{R}$? Trapezoidal, Riemann, etc.
How about in $\mathbb{R}^d$? Hard to evaluate.

8.6 Variational Inference

Instead of calculating the exact posterior, we would like to set this up as an optimization problem.

Posit a family of distributions $q(z;\lambda)$ over the hidden state indexed with parameter λ.

We want to find λ s.t. q is close to the original distribution $p(z|x)$

$$\begin{aligned}\text{KL}(q(z;\lambda)||p(z|x)) &= \mathbb{E}_q[\log q(z;\lambda) - \log p(z|x)]\\ &= \mathbb{E}_q[\log q(z;\lambda) - \log p(z,x) + \log p(x)]\\ &= \mathbb{E}_q[\log q(z;\lambda) - \log p(z,x)] + \log p(x)\end{aligned}$$

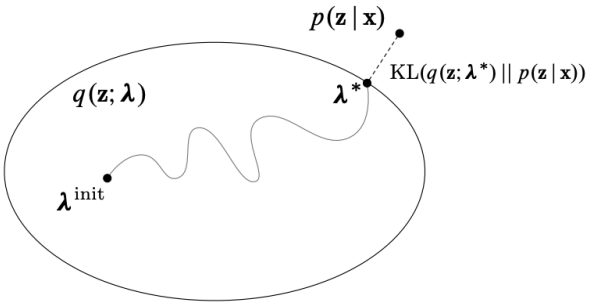


Figure 20: The family of distributions described above

Equivalently, we can think about this as a lower bound we're interested in:
Evidence lower bound (ELBO):

$$\begin{aligned}\log p(x) &= \mathbb{E}_q\{\log p(z,x) - \log q(z;\lambda)\} + KL\{q(z;\lambda)||p(z|x)\} \\ &\geq \mathbb{E}_q\{\log p(z,x) - \log q(z;\lambda)\}\end{aligned}$$

8.7 The evidence lower bound (ELBO)

$$\mathscr{L}(\lambda) = \mathbb{E}_q[\log p(x,z)] - \mathbb{E}_q[\log q(z;\lambda)]$$

- KL is intractable; variational inference optimizes the ELBO
  - It is a lower bound on $\log p(x)$
  - Maximizing the ELBO is equivalent to minimizing the KL
- The ELBO trades off two terms
  - The first term prefers $q(\cdot)$ to maximize the likelihood
  - The second term regularizes $q(\cdot)$ to the prior
- Approximation q is chosen to match types

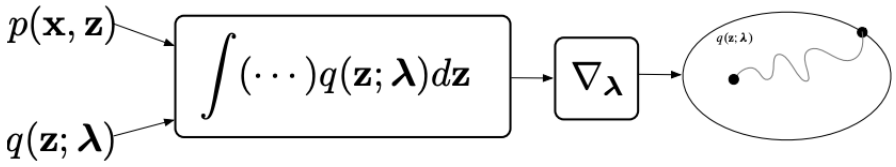


Figure 21: Variational Inference Recipe

8.8 Example: Bayesian logistics regression

- Data pairs y_i, x_i
- x_i are covariates
- y_i are labels
- z is the regression coefficient
- Generative process

$$p(z)\mathrm{m}\mathcal{N}(0,1)$$
$$p(y_i|x_i,z) \sim \text{Bernoulli}(\sigma(zx_i))$$

Assume:

- We have on data point (y,x)
- The approximating family q is the normal; $\lambda = (\mu, \sigma^2)$

The ELBO is

$$\mathcal{L}(\mu,\sigma^2) = \mathbb{E}_q\{\log p(z) + \log p(y|x,z) - \log q(z)\}$$

$$\begin{aligned}
&\mathcal{L}\left(\mu,\sigma^2\right) \\
&= \quad \mathbb{E}_q[\log p(z) - \log q(z) + \log p(y \mid x,z)] \\
&= -\frac{1}{2}\left(\mu^2+\sigma^2\right) + \frac{1}{2}\log\sigma^2 + \mathbb{E}_q[\log p(y \mid x,z)] + C \\
&= -\frac{1}{2}\left(\mu^2+\sigma^2\right) + \frac{1}{2}\log\sigma^2 + \mathbb{E}_q[yxz - \log(1+\exp(xz))] \\
&= \quad -\frac{1}{2}\left(\mu^2+\sigma^2\right) + \frac{1}{2}\log\sigma^2 + yx\mu - \mathbb{E}_q[\log(1+\exp(xz))]
\end{aligned}$$

We cannot analytically take that expectation; the expectation hides the objectives dependence on the variational parameters. This makes it hard to directly optimize.

8.9 Non-conjugate models

The models above are conjugate, and it might be possible to analytically derive solutions for some. But most models we're interested in are non-conjugate.

- Nonlinear Time series Models
- Deep Latent Gaussian Models
- Models with Attention (such as DRAW)
- Generalized Linear Models (Poisson Regression)
- Stochastic Volatility Models
- Discrete Choice Models
- Bayesian Neural Networks
- Deep Exponential Families (e.g. Sparse Gamma or Poisson)
- Correlated Topic Model (including nonparametric variants)
- Sigmoid Belief Network

Black Box Variational Inference(BBVI)

8.10 Computing Gradients of Expectations

Define

$$g(z,\lambda) = \log p(x,z) - \log q(z;\lambda)$$

$$\begin{aligned}
\nabla_\lambda \mathcal{L} &= \nabla_\lambda \int q(z;\lambda) g(z,\lambda) dz \\
&= \mathbb{E}_q\{\nabla_\lambda \log q(z;\lambda) g(z,\lambda) + \nabla_\lambda g(z,\lambda)\}
\end{aligned}$$

Using $\nabla_\lambda \log q = \dfrac{\nabla_\lambda q}{q}$.

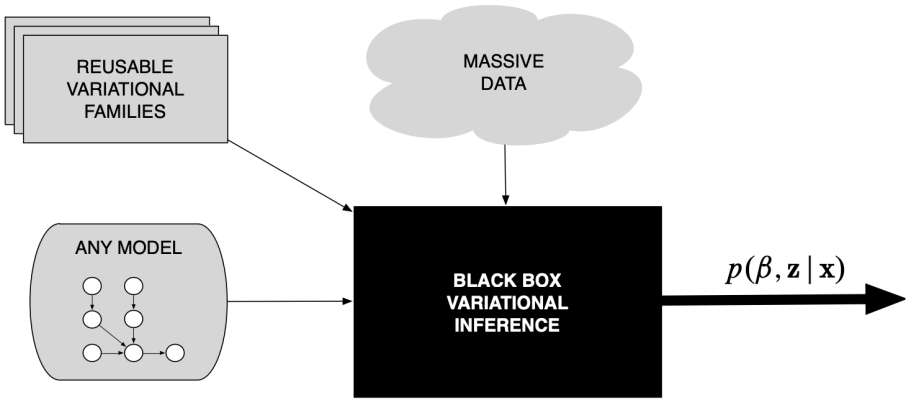


Figure 22: Black box variational inference

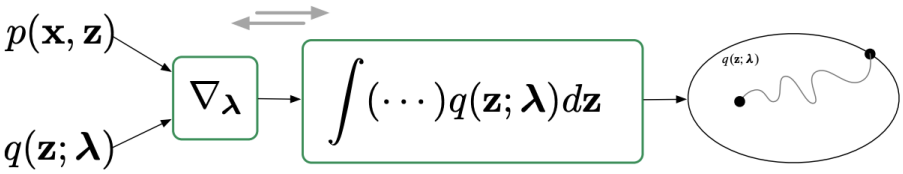


Figure 23: New variational inference recipe modified from Figure 21

8.11 Score function estimator (likelihood ratio, REINFORCE gradients)

$$\mathscr{L}(\lambda) = \mathbb{E}_{q(z;\lambda)}[\log p(x,z) - \log q(z;\lambda)]$$

Recall

$$\nabla_\lambda \mathscr{L} = \mathbb{E}_{q(z;\lambda)}[\nabla_\lambda \log q(z;\lambda) g(z,\lambda) + \nabla_\lambda g(z,\lambda)]$$

Why unbiased? Relevant: expected-grad-log-prob lemma from a PPO tutorial

Can have noisy unbiased gradient using Monte Carlo. To compute the noisy gradient of ELBO we need

- Sampling from $q(z)$
- Evaluating $\nabla_\lambda \log q(z;\lambda)$
- Evaluating $\log p(x,z)$ and $\log q(z)$

There is no model-specific work: black box criteria are satisfied.

$\boldsymbol{\lambda} = \{\boldsymbol{\theta}_k, \mu_k, \Sigma_k\}_{k=1}^K$

8.12 Problem: Basic BBVI doesn't work

Variance of the gradient could be a problem

$$Var_q = \mathbb{E}[(\nabla_\lambda \log q(z;v)(\log p(x,z) - \log q(z;v)) - nabla_\lambda \mathscr{L})^2]$$

Intuiton:
Sampling rare values can lead to large scores and thus high variance.

More Assumptions? Want assumptions that are not too restrictive.

8.13 Pathwise Gradients of the ELBO

Reparameterization Estimator

We assume

- $z = t(\epsilon, \lambda)$ for ϵ $s(\epsilon)$ implies z $q(z;\lambda)$. For example:
  - ϵ Normal$(0,1)$
  - $z = \epsilon\sigma + \mu \xrightarrow{z}$ Normal(μ, σ^2)
- $log(p(x,z))$ and $log(q(z))$ are differentiable with respect to z

Differentiate this we get

$$\nabla\mathscr{L}(\lambda) = \mathbb{E}_{s(\epsilon)}[\nabla_z[\log p(x,z) - \log g(z;\lambda)]\nabla_\lambda t(\epsilon,\lambda)]]$$

This is also known as the **pathwise gradient**.

What's an example problem that might have gradients where one is better than the other?

8.14 Score Function Estimator vs. Reparameterization Estimator

Score Function

- Differentiates the density $\nabla_\lambda q(z;\lambda)$
- Works for discrete and continuous models
- Works for a large class of variational approximations
- Variance can be a big problem

Pathwise

- Differentiates the function $\nabla_z[logp(x,z)logq(z;\lambda)]$
- Requires differentiable models
- Requires variational approximation to have form $z = t(\epsilon,\lambda)$
- Generally better behaved variance

How do we use both estimators at the same time?

Parameter Learning with Variational Inference

Train model $p_\theta(x,z)$ by MLE. Marginal likelihood is:

$$\log p_\theta(x) = \log \int p_\theta(x,z)dz$$

Hard to compute the integral. If posterior was known,

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

Posterior is hard because of integration (unknown $p(\mathbf{x})$).

Maximize the ELBO:

$$\log p_\theta(x) \geq \mathbb{E}_q[\log p(z,x) - \log q(z;\lambda)] =: \mathscr{L}(\lambda,\theta)$$

A lower bound on the evidence $\log p_\theta(x)$

Optimize $\mathscr{L}(\lambda,\theta)$

    - Use standard gradients for θ
- Use score/reparametrization gradients for λ

  Could instead maximize θ and choose optimal $q = p_\theta(z|x)$

    - Compute optimal $q_t = p_{\theta_{t-1}}(z|x)$
    - $\theta_t = \arg\max_\theta \mathcal{L}(q_t, \theta)$

9 Deep Generative Models

In this class, we discuss about sampling from $p(\mathbf{x})$ directly to build generative models.

9.1 Variational Autoencoders

Variational Autoencoder is a latent variable model. The model parameters are β.

If z is drawn from $Normal(0, 1)$, then $p(x|z) = Normal(\mu_\beta(z), \sigma^2_\beta(z))$.

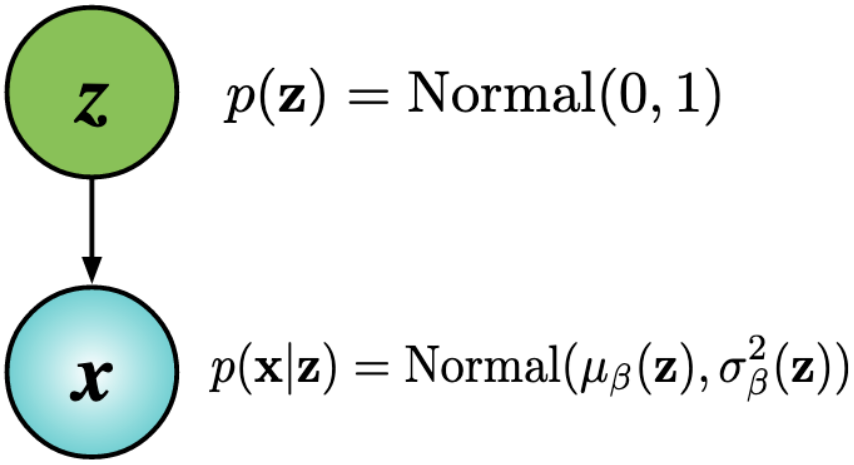


Figure 24: VAE

In order to estimate β, we cannot use Maximum Likelihood estimation since it is intractable. Hence, we use variational inference.

9.1.1 Evidence Lower Bound (ELBO)

For a single point, would like to

$$\max_\beta \log_\beta = \max_\beta \log \int p_\beta(x, z) \mathrm{d}z$$

As the integral is intractable we can approximate it my maximizing the variational objective which is given by:

$$L(q, \beta) = E_q[\log p - \log q] = \sum_{i=1}^{n} E_q[\log p(z_i, x_i) - \log q(z_i; m_i, s_i)$$

In order to optimize the variational objective, we can use reparameterization gradients or subsample the data.

We convert the local parameters into neural network parameters. This gives us the following:

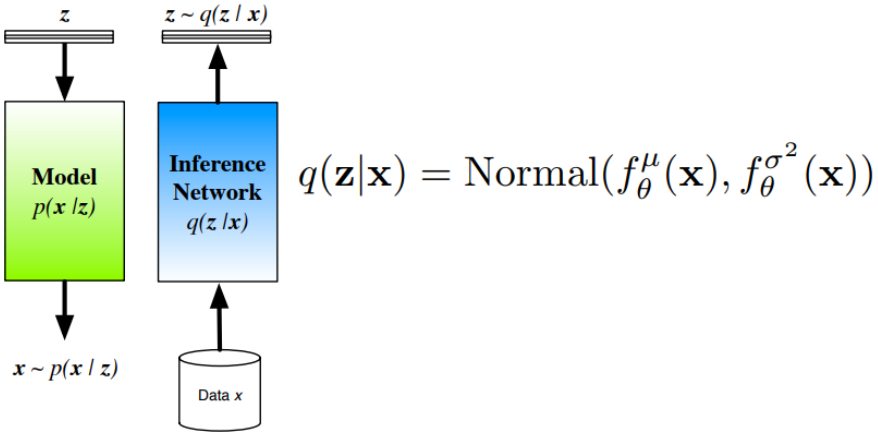


Figure 25: VAE: Inference Networks

$$L(q,\beta) = \sum_{i=1}^{n} E_q[\log p(z_i, x_i) - \log q(z_i; m_i = f_\theta^\mu(x_i), s_i = f_\theta^\sigma(x_i))$$

Through this we fixed the issue such that the size of the optimization problem is not dependent on the size of the dataset.

9.2 Direct Likelihood Models

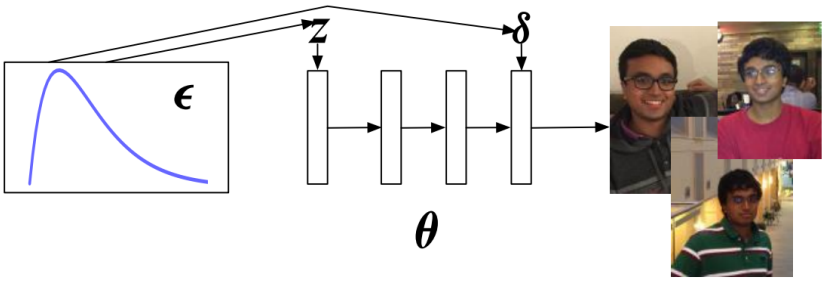


Figure 26: Deep generative models with latent variables

Instead of of using latent variables we can directly use the noise to model the density directly.

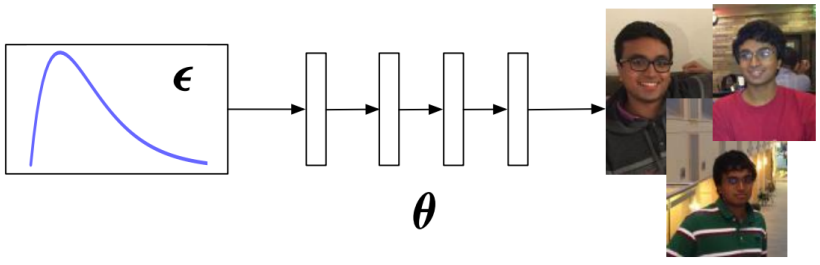


Figure 27: Deep generative models with Noise

$$x \sim p_\theta(x) \iff x = f(\theta, x)$$

In deep generative models, accurate inference of z given x is difficult and inference is needed because the amount of noise is larger than data.

Maximum Likelihood

1. Write down a model $p_\theta(x)$
2. Observed Likelihood: $\mathscr{L} = \log p_\theta(x_1, \cdots, x_n) = \sum_{i=1}^{N} \log p_\theta(x_i)$
3. Take gradient w.r.t θ: $\nabla_\theta \mathscr{L} = \sum_{i=1}^{N} \nabla_\theta p_\theta(x_i)$
4. Use stochastic gradients to maximize likelihood of the data.

Is the Maximum Likelihood good?

For an empirical distribution $\hat{F}(x) = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}(x)$, the maximum likelihood objective is

$$\mathscr{L} = -n \cdot KL(\hat{F}(x) || p_\theta(x)) + c$$

Maximum likelihood minimizes the KL divergence.

Limitations:

- Difficult to estimate for complex data like audio, text, images, etc.
- Difficult to work with multivariate observations.
- Works well for known distributions and they place strong assumptions on data.

Methods

- **Idea 1: Decompose** $p(x)$

  For k-dimensional vector, $p_\theta(x) = \prod_{i=1}^{k} p_\theta(x_i | x_{<i})$

  Advantage: We can use univariate distributions to build multivariate which are easier to work with.

  Challenges: As the dependency set grows, the conditional distribution parameter size grows exponentially. One way to solve this is through deep learning.

  Recurrent Neural Networks: Sequence models with hidden state $h_i = f_\theta(h_{i-1}, x_{i-1})$ can be used to build a probability model over x,

  $$p_\theta(x_i | x_{<i}) = p(x_i | h_i)$$

  Row LSTM: Long range dependencies are hard to capture in RNNs and Row LSTMs tackle that problem by operating on rows instead of pixels.

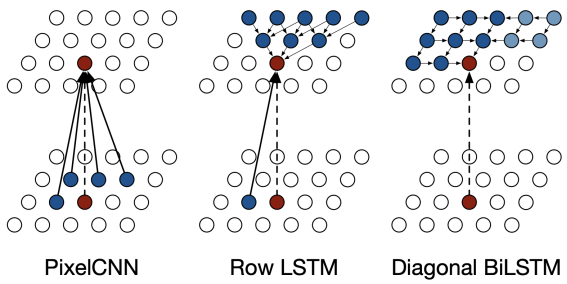


Figure 28: Different Architectures

- **Idea 2: Transform Noise Vectors**
  Get k dimensional densities by transforming noise and the densitry of x can be calculated with change of variables law:

$$\epsilon \sim s(\epsilon)$$
$$x = f_\theta(x)$$

  Then the density of x is, Assuming f is invertible

$$p(x) = s(f_\theta^{-1}(x)) \left| \frac{df_\theta^{-1}(x)}{dx} \right|$$

  Challenges:
  - Computation of the Jacobian is difficult.
  - Requires both f and f^{-1} to be easy for sampling.

  Solution: $O(n)$ **Jacobian**
  Using a lower triangular matrix, determinant can be computed in $O(n)$.
  Real NVP
  First d dimensions are unchanged and the other $k-d$ dimensions are obtained from transformations of previous d dimensions and the previous $k-d$ dimensions.

$$x_{1:d} = \epsilon_{1:d}$$
$$x_{d+1:k} = \epsilon_{d+1:k} \cdot \exp(s(\epsilon_{1:d})) + t(\epsilon_{1:d})$$

  The dimensions can be shuffled and this won't affect the model.

Connection between the two ideas
Autoregressive models use probabilities and for 3 dimensional vector,

$$p(x) = p_\theta(x_1)p_\theta(x_2|x_1)p_\theta(x_3|x_2, x_1)$$

This implies for some noise ϵ,

$$x_1 = f_\theta(\epsilon_1), x_2 = f_\theta(\epsilon_2, x_1), x_3 = f_\theta(\epsilon_3, x_1, x_2)$$
$$\implies x_1 = f_\theta(\epsilon_1)$$
$$\implies x_2 = f_\theta(\epsilon_2, f_\theta(\epsilon_1))$$
$$\implies x_3 = f_\theta(\epsilon_3, f_\theta(\epsilon_1)), f_\theta(\epsilon_2, f_\theta(\epsilon_1)))$$

This is a lower triangular Jacobian which shows that the two approaches are the same.

9.3 Generative Adversarial Networks

Adversarial Learning

How do we learn a model f_θ given samples $\mathbf{x}_1, ..., \mathbf{x}_n$.

1. Write down a model $f(\theta, \epsilon)$
2. Write down a discriminator D_ϕ between Real data $\mathbf{x}_i$ Model samples $\mathbf{x}_i^\star$
3. Classification objective for discriminator
$$\mathscr{L}(\phi, \theta) = \mathbb{E}_{\mathbf{x} \sim p_{data}}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{x}=f(\theta,\epsilon)}[log(1 - D(\mathbf{x}))]$$
4. Use stochastic gradients to maximize with respect to ϕ
5. Use stochastic gradients to minimize with respect to θ

Objective:

$$\mathbb{E}_{\mathbf{x} \sim p_{data}}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{x}=f(\theta,\epsilon)}[log(1 - D(\mathbf{x}))]$$

Goal:

$$\min_\theta \max_\phi \mathbb{E}_{\mathbf{x} \sim p_{data}}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{x}=f(\theta,\epsilon)}[log(1 - D(\mathbf{x}))]$$

Gradients:

$$\nabla_\phi \mathscr{L} = \mathbb{E}_{\mathbf{x} \sim p_{data}}[\nabla_\phi logD(\mathbf{x})] + \mathbb{E}_{\mathbf{x}=f(\theta,\epsilon)}[\nabla_\phi log(1 - D(\mathbf{x}))]$$

$$\nabla_\theta \mathscr{L} = \mathbb{E}_{\mathbf{x}=f(\theta,\epsilon)}[\nabla_x log(1 - D(\mathbf{x}))\nabla_\theta f(\theta, \epsilon)]$$

Optimal Discriminator

Loss for the discriminator

$$\max_D \mathbb{E}_{\mathbf{x}\sim p_{data}}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{x}=f(\theta,\epsilon)}[log(1-D(\mathbf{x}))]$$

p_{model} defined implicitly at preimage

$$p_{model}(\mathbf{x}) = \int_{\mathbf{x}=f(\theta\epsilon)} s(\epsilon)d\epsilon$$

Diifferentiate L functionally

$$\nabla_{\mathscr{L}} = \frac{p_{data(\mathbf{x})}}{D(\mathbf{x})} - \frac{p_{model}(\mathbf{x})}{1-D(\mathbf{x})}$$

Maximized at:

$$(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{model}(\mathbf{x}}$$

Recall

$$\mathscr{L}(M,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{x}=f(\theta,\epsilon)}[log(1-D(\mathbf{x}))]$$

Substitute optimal discriminator and finnally get the Jenson Shannon Divergence.

$$\begin{aligned}\mathscr{L}(M) &= -\log 4 + \mathbb{E}_{\mathbf{x}\sim p_{data}}\left[\log \frac{p_{data}(\mathbf{x})}{\frac{1}{2}p_{data}(\mathbf{x}) + \frac{1}{2}p_{model}(\mathbf{x})}\right] + \mathbb{E}_{\mathbf{x}=p_{model}}\left[\log \frac{p_{data}(\mathbf{x})}{\frac{1}{2}p_{data}(\mathbf{x}) + \frac{1}{2}p_{model}(\mathbf{x})}\right] \\ &= -\log 4 + KL\left(p_{data}||\frac{1}{2}p_{data} + \frac{1}{2}p_{model}\right) + KL\left(p_{model}||\frac{1}{2}p_{data} + \frac{1}{2}p_{model}\right) \\ &= -\log 4 + 2JSD(p_{model}||p_{data})\end{aligned}$$

Alternative View: Likelihood Ratio Estimation

Was there something special about the original objective?

- Consider pseudolabels y
- Assign pseudolabels $y = 1$ to $\mathbf{x} \sim p_{data}$
- Assign pseudolabels $y = 0$ to $\mathbf{x} \sim p_{model}$

$$\frac{p_{data}(\mathbf{x})}{p_{model}(\mathbf{x})} = \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=0)} = \frac{p(y=1|\mathbf{x})p(y=1)^{-1}}{p(y=0|\mathbf{x})p(y=0)^{-1}}$$

Assume equal samples from data and model

$$\frac{p_{data}(\mathbf{x})}{p_{model}(\mathbf{x})} = \frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} = \frac{p(y=1|\mathbf{x})}{1-p(y=1|\mathbf{x})}$$

Solve for $p(y=1|\mathbf{x})$

$$p(y=1|\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_{model}(\mathbf{x})}$$

- Conditional distribution has form of optimal discriminator
- Learn conditionals with classification
- Any classification method that is “proper” recovers this!

Problem: Consider two distribution whose supports don’t overlap.

- Discriminator is optimal
- Gradient of discriminator is zero at non-zero probability

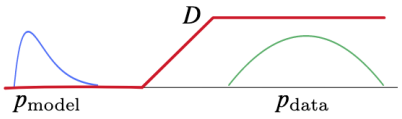


Figure 29: Problem illustration of GAN LRE

Comparison with Maximum Likelihood

The objective for nagative maximum likelyhood is

$$L_{neg-ml} = nKL(\hat{F}(x)||p_\theta(x)) + C$$

Whereas the objective for vanilla GAN is

$$L_{gan} = JSD(p_{model}||p_{data}) + C$$

Suppose $Supp(p_\theta(x)) \subset Supp(\hat{F}(x))$ Then.

- KL is infinite, JSD is not
- Difference lies how they measure distances.

Can go beyond KL with Wasserstein distances

10 Causal Inference

10.1 Causal Inference

The model $p(y|\mathbf{t})$ doesn't let us know whether the change in $\mathbf{t}$ renders changes in y.

Causal inference seeks to estimate the effect of an intervention.

Finding important features in regression is not obvious finding causes. We need a mathematical definition of causal inference.

Causal effects are functions of the potential outcomes, hence for a given scenario with A and B as solutions, we use the potential outcomes to determine which solution to choose.

Once you effect changes to an entity (i.e single person) with a cause, that entity changes its state; hence, we can't observe the same changes to the same entity from a different cause.

It's not easy to answer what the effect of all the causes may be on a single person, which is when we can move to answer the question on a population level and find the average over the distribution of the input.

Average Treatment Effect: $ATE = Ei[y_{1,i}] - Ei[y_{0,i}]$
The Average Treatment Effect is the average outcome for those who received treatment 1 minus the average outcome for those who received treatment 0.

Individualized Treatment Effect: $ITE(\mathbf{x}_i) = Ei[y_{1,i}|\mathbf{x}_i] - Ei[y_{0,i}|\mathbf{x}_i]$

These are equivalent by averaging over $p(\mathbf{x}_i)$: $ATE = E_{p(\mathbf{x}_i)}[ITE(\mathbf{x}_i)]$
Terminology:

1. Treatment t.

2. Controls: people who receive treatment 0, $t = 0$

3. Treated: people who receive treatment 1, $t = 1$

4. Counterfactuals: The unobserved outcome $y_{1-t,i}$

For a given problem, to produce a causal effect, we need to consider more than naively considering the average treatment effect as it is possible that all the people receiving treatment may be sicker, older, etc. In other words you do not know if the results are comparable.

Mathematically, we have:

$$E[y_1|T = 1] - E[y_0|T = 0] \neq E[y_1] - E[y_0]$$

One of the ways to actualize this difference is randomization where we assign treatments independent of the rest of the variables, which removes dependence. Since the treatment is random, the conditional expectation conditioned on treatment would be the same as the marginal expectation.

Mathematically, this is:

$$E[y_1|T = 1] - E[y_0|T = 0] = E[y_1] - E[y_0]$$

given the removal of dependence.

However, many randomized trials are difficult or unethical to run (i.e. the lung cancer and smoking trial). There are practical limitations to randomization too in terms of the number of combinations that is possible to undertake and test.

10.2 Observational Causal Inference

Observational causal inference aims to estimate the causal effects from non-randomized data.

The problem with Observational Causal Inference is that outcomes may depend on treatment.

Strong ignorability says that potential outcomes are independent of treatment given a covariate **x**.

Mathematically,

$$E[y_1] - E[y_0] = E_x[E[y_1|\mathbf{x}]] - E_x[E[y_0|\mathbf{x}]] = E_x[E[y_1|\mathbf{x}, t=1]] - E_x[E[y_0|\mathbf{x}, t=0]]$$

For fixed covariates **x**:
1. $E[y_1|\mathbf{x}, t=1]$ is the average treated outcome on the treated
2. $E[y_0|\mathbf{x}, t=0$ is the average untreated outcome on untreated

Strong ignorability assumes the world consisting of conditionally randomized experiments (given the value of **x**).

Strong ignorability is a statement about both potential outcomes, which is why treatment cannot be checked for a single individual (given a dataset of **x**'s, **t**'s and **y**'s). Hence, strong ignorability is an actual assumption that might not be checkable given actual data.

$Supp$ is defined as the support of the distribution.

To compute $Supp(p(\mathbf{x}|t=1)) \cap Supp(p(\mathbf{x}|t=0)) = \emptyset$,

we consider $ATE = E_x[E[y_1|\mathbf{x}, t=1]] - E_x[E[y_0|\mathbf{x}, t=0]]$

We require full support, which is called "positivity" in literature because we cannot distinguish between **x** and t since t determines part of **x**.

10.3 Computational Tool: Regression

A computational tool to estimate $E[y|\mathbf{x}, t]$ can be a regression model as direct averages can require too many samples. We use regression for the minimization problem of expected squared error. The resulting minimizer would be the required conditional average.

$$argmin_f E[(y - f(\mathbf{x}, t))^2]$$

$$= argmin_f \int p(\mathbf{x}, t) \int p(y|\mathbf{x}, t)(y - f(\mathbf{x}, t))^2 dy dt dx$$

$$= argmin_f \int p(\mathbf{x}, t) \int p(y|\mathbf{x}, t)(y^2 - 2yf(\mathbf{x}, t) + f(\mathbf{x}, t)^2) dy dt dx$$

$$= argmin_f \int p(\mathbf{x}, t)(E[y|\mathbf{x}, t]^2 + Var(y|\mathbf{x}, t) - 2E[y|\mathbf{x}, t]f(\mathbf{x}, t) + f(\mathbf{x}, t)^2) d$$

$$= argmin_f \int p(\mathbf{x}, t)((E(y|\mathbf{x}, t) - f(\mathbf{x}, t)^2 + Var(y|\mathbf{x}, t)) dt d\mathbf{x}$$

$$= E[y|\mathbf{x}, t]$$

$$\implies argmin_f E[(y - f(\mathbf{x}, t))^2] = E[y|\mathbf{x}, t]$$

We can then make f parametric f_θ to solve $argmin_\theta E[(y - f_\theta(\mathbf{x}, t))^2]$ which we can re-substitute in ATE such that,

$$ATE = E_x[E[y_1|\mathbf{x}, t=1]] - E_x[E[y_0|\mathbf{x}, t=0]]$$
$$= E_x[f_\theta(\mathbf{x}, 1)] - E_x[f_\theta(\mathbf{x}, 0)]$$

If treatment 0 and treatment 1's behaviours are more or less similar we can use single regression, but if they differ a lot, we should use 2 separate regressions for them.

$E[y|\mathbf{x}, t]$ is valid when $p(\mathbf{x}, t) > 0$.

A **confounder**, which is observed (different from a latent variable which is unobserved) is a variable that influences both the dependent variable and independent variable. Therefore, we need to separate the background variation from the treatment variation. However, the treatment needs to have some variation given the background.

10.4 Functional View

Under an observational distribution, the conditional expectation is not equal to an intervened distribution setting.

Under a randomized distribution, the conditional expectation is equal to an intervened distribution setting.

Causal inference helps us to answer questions about out-of-distribution data.

10.5 Computational Tool: Propensity Scores

We see one distribution (observational or randomized) and we compute the data for the intervention.

We know from strong ignorability, that $\mathbf{x}$ predicts the treatment probability, which has enough information to break dependence. We can use regression to model treatment chance.
Propensity score = $p_\phi(t = 1|\mathbf{x})$

We use importance sampling to compute expectations.

10.6 Computational Tool: Propensity Score and Regression

We are trying to compute the average potential outcome if treated (using the Law of Total Probability).

We can use Monte Carlo methods and take the average to estimate Propensity Scores (Both regression and propensity scores give Monte Carlo averages).

How to check how good the causal effect estimate is?
- Check the correctness of assumptions.
- We can check how sensitive the causal effects are to the assumptions.
- We can check the effect on an observer
- We can certainly hold out some data and calculate the propensity score.

listings

11 Reinforcement Learning

11.1 Introduction

Reinforcement Learning consists of the following essential components: goal, action, and state.

Take chess for example. The goal is to have the opponent's king be in check and cannot avoid capture on the next move. The action is to make an available move for one unit on the player's side. The state is a distribution of current units on the board.

11.2 Markov Decision Processes

Reinforcement Learning (RL) mainly aims to address a class of problems called the Markov Decision Process (MDP). An MDP can be typically defined by a tuple of $\{S, A, r, P\}$. S is the collection of states; A is a collection of all possible actions; r is the reward function that maps a state-action pair (s, a) to a real number (reward functions can also be random); and $P(s_{t+1}|s_t, a_t)$ is the transition probability from the current state s_t to the next state s_{t+1} by taking the action a_t, where t is the time step.

Potential questions with the above definitions:

- Things only depend on the previous time step (Markov assumptions). Is it Okay?
  This is reasonable because we can combine several dependent states into one big state.
- What about states that are hidden?
  For partially observed MDP, we can refer to the tools we learned before, like latent variable models.

The goal of an MDP is to find a policy π which maximizes a quantity called total reward, which is the accumulated (discounted) rewards obtained from each experienced state. In order to formulate the goal, we define the state-value function $v_\pi(s)$ and the action-value function $q_\pi(s, a)$ given the current policy π, state s and action a. The state-value (V) function represents the expected accumulated rewards of the episode starting at the state s and following the policy π, while the action-value (Q) function represents the expected accumulated rewards of the episode starting at the state s and choosing the immediate action a, and then following the policy π.

I suggest going through sections 3.1 & 3.5 of the Sutton & Barto RL book for detailed definitions of the basic concepts mentioned above, before going deeper.

11.3 Policy Gradients Method

Since we have an objective to maximize which is:

$$v_\pi(s_0) = \mathbb{E}_{\pi,P}[\sum_{t=0} \gamma_t r_t | s_0]$$

, by following the generic idea of machine learning optimization, we want to directly compute its gradient and use gradient methods. By some computation (refer to Section 13.2-13.3 of the Sutton & Barto RL book for derivations), the Monte Carlo unbiased gradient estimator is:

$$\hat{\nabla}_\theta \mathscr{L}(\theta) = \frac{1}{S} \sum_{s=1}^{S} \nabla_\theta \log \pi_\theta(\tau_s) r_{\tau_s}$$

, where the Monte Carlo sampling is used due to the intractable expectation operator.

However, the inefficiency of Monte Carlo sampling introduces a high variance of the gradient (though unbiased), which makes the algorithm collapse. In order to mitigate it, the baseline method is proposed (see Section 13.4 of the Sutton & Barto RL book). Basically, this method keeps the estimator unbiased but decreases the value of the gradient estimates by substructing the accumulated rewards with a constant.

11.4 Actor-Critic Method

Actor-critic methods are methods that have a separate memory structure to explicitly represent the policy independent of the value function. The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic, because it criticizes the actions made by the actor. Learning is always on-policy: the critic must learn about and critique whatever policy is currently being followed by the actor. In actor–critic methods, the state-value function is applied also to the second state of the transition. The estimated value of the second state, when discounted and added to the reward, constitutes the one-step return, $G_{t:t+1}$, which is a useful estimate of the actual return and thus is a way of assessing the action.

One-step actor–critic methods with the one-step return and use a learned state-value function as the baseline as follows:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &:= \boldsymbol{\theta}_t + \alpha(G_{t:t+1} - \hat{v}(S_t, w))\frac{\nabla(At|S_t, \boldsymbol{\theta}_t)}{(At|S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha(R_{t+1} - \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w))\frac{\nabla(At|S_t, \boldsymbol{\theta}_t)}{(At|S_t, \boldsymbol{\theta}_t)} \\ &= \boldsymbol{\theta}_t + \alpha\delta_t\frac{\nabla(At|S_t, \boldsymbol{\theta}_t)}{(At|S_t, \boldsymbol{\theta}_t)}\end{aligned}$$

We further define Q-function by

$$Q_t^\pi(\boldsymbol{a}_t, \boldsymbol{s}_t) = \mathbb{E}[(\sum_{t'=t+1}^{T} r_{t'})|(\boldsymbol{a}_t, \boldsymbol{s}_t)]$$

and the Value function

$$V_t^\pi(\boldsymbol{s}_t) = \mathbb{E}_{\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)}\mathbb{E}[(\sum_{t'=t+1}^{T} r_{t'})|(\boldsymbol{a}_t, \boldsymbol{s}_t)]$$

, and we will see that

$$\nabla_\theta \mathscr{L} = \sum_{t=1}^{T}\mathbb{E}[Q_t^\pi(\boldsymbol{a}_t, \boldsymbol{s}_t)\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t, \boldsymbol{s}_t)]$$

and

$$\sum_{t=1}^{T}\mathbb{E}[V_t^\pi(\boldsymbol{a}_t, \boldsymbol{s}_t)\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t, \boldsymbol{s}_t)] = 0$$

Then we define the Advantage function as

$$A_t^\pi(\boldsymbol{a}_t, \boldsymbol{s}_t) = Q_t^\pi(\boldsymbol{a}_t, \boldsymbol{s}_t) - V_t^\pi(\boldsymbol{s}_t) = \mathbb{E}[r_{t+1} + V_t^\pi(\boldsymbol{s}_{t+1})|\boldsymbol{a}_t, \boldsymbol{s}_t] - V_t^\pi(\boldsymbol{s}_t)$$

The stochastic gradients that we want to use to update the value function is

$$\nabla_\phi = \sum_{j=1}^{J}(f_{\phi^*}(s_{t+1}^j) + r_{t+1}^j - f_\phi(s_t^j))\nabla_\phi f_\phi(s_t^j)$$

Algorithm of One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

```
Input: a differentiable policy parameterization
Input: a differentiable state-value function
   parameterization v̂(s, w)
Parameters: step size α^θ > 0, α^w > 0
Initialize policy parameter θ ∈ ℜ^{d'} and state-value weights
   w ∈ ℜ^d (e.g., to 0)
Loop forever (for each episode):
Initialize S (first state of episode)
```

$I \leftarrow 1$
Loop while S is not terminal (for each time step):
$A \sim \pi(\cdot|S,\boldsymbol{\theta})$
Take action A, observe S', R
$\delta \leftarrow R + \gamma\hat{v}(S',\mathtt{w}) - \hat{v}(S,\mathtt{w})$ (if S' is terminal, then $\hat{v}(S',\mathtt{w}) = 0$)
$\mathtt{w} \leftarrow \mathtt{w} + \alpha^{\mathtt{w}}\delta\nabla\hat{v}(S,\mathtt{w})$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla \ln\pi(A|S,\boldsymbol{\theta})$
$I \leftarrow \gamma I$
$S \leftarrow S'$

The generalizations to the forward view of $n-$step methods and then to a $\lambda-$ -return algorithm are straightforward. The one-step return in above is merely replaced by $G_{t:t+n}$ or G_t^{λ} respectively. The backward view of the λ-return algorithm is also straightforward, using separate eligibility traces for the actor and critic, each after the pattern before.

Pseudocode for the complete algorithm is given in the box below.

Actor Critic with Eligibility Traces (episodic), for estimating $\pi_\theta \approx \pi_*$
Input: a differentiable policy parameterization
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathtt{w})$
Parameters: trace-decay rates $\lambda^{\boldsymbol{\theta}} \in [0,1], \lambda^{\mathtt{w}} \in [0,1]$; step sizes
Initialize policy parameter $\boldsymbol{\theta} \in \Re^{d'}$ and state-value weights $\mathtt{w} \in \Re^{d}$ (e.g., to 0)
Loop forever (for each episode):
Initialize S (first state of episode)
$\mathtt{z}^{\boldsymbol{\theta}} \leftarrow 0$ (d'-component eligibility trace vector)
$\mathtt{z}^{\mathtt{w}}$ (d-component eligibility trace vector)
$I \leftarrow 1$
Loop while S is not terminal (for each time step):
$A \sim \pi(\cdot|S,\boldsymbol{\theta})$
Take action A, observe S', R
$\delta \leftarrow R + \gamma\hat{v}(S',\mathtt{w}) - \hat{v}(S,\mathtt{w})$ (if S' is terminal, then $\hat{v}(S',\mathtt{w}) = 0$)
$\mathtt{z}^{\mathtt{w}} \leftarrow \gamma\lambda^{\mathtt{w}}\mathtt{z}^{\mathtt{w}} + \nabla\hat{v}(S,\mathtt{w})$
$\mathtt{z}^{\boldsymbol{\theta}} \leftarrow \gamma\lambda^{\boldsymbol{\theta}}\mathtt{z}^{\boldsymbol{\theta}} + I\nabla \ln\pi(A|S,\boldsymbol{\theta})$
$\mathtt{w} \leftarrow \mathtt{w} + \alpha^{\mathtt{w}}\delta\mathtt{z}^{\mathtt{w}}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\delta\mathtt{z}^{\boldsymbol{\theta}}$
$I \leftarrow \gamma I$
$S \leftarrow S'$

Advantage function Get rid of Q functions <- Bellman Equation Sampling method v.s. Parametrized Model Actor-Critic with Advantage function

11.5 Fitted Value Iteration

The Actor-Critic method in the previous section needs two parametrized models. The policy model is called 'actor', because it outputs the potential good actions to be executed. And the state-value model is called 'critic' because it tells how promising one state is.

Suppose we know how promising each state is, that is, we know the value functions (critic) of any state. For a given state, we thus want to set our policy to actions that always lead to more promising next states. In this sense, we wonder whether we can learn by only critics, and induce a policy by the learned value functions.

Indeed, the Value Iteration algorithm (section 4.4 of S & B Book) serves as a direct solution following this intuition. We recursively update the state-value estimation by the following equation for each

state:

$$\begin{aligned} v_{k+1}(s) &:= \max_a q_k(s,a) \\ &= \max_a \mathbb{E}[r_{t+1} + \gamma v_k(s_{t+1}) | s_t = s, a_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \end{aligned}$$

and after the value function converges, we set the policy to be:

$$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\infty(s')]$$

Note that the reason why we set $v_{k+1}(s) := \max_a q_k(s,a)$ actually comes from the Bellman Optimality Equation (section 3.6). (Also, please take a look at the Policy Improvement Theorem to better understand this algorithm; e.g. section 4.2 of S&B RL Book).

The above algorithm is essentially a Dynamic Programming algorithm. The major limitation is that to recursively compute the state-value estimation, we need to know the MDP's transition probability (dynamics). In order to get rid of this limitation, we again can use the Bellman equation (sections 3.5 & 3.6) to calculate: given a transition (s_i, a_i, s_i')

$$\begin{aligned} v(s_i') &:= \max_{a'} q(s_i', a\prime) \\ &= r(s_i, a_i) + \gamma \max_{a'} q(s_i', a') \end{aligned}$$

Then, let the target y_i be equal to $v(s_i')$, we then can optimize the action-value estimation by minimizing the square error between y_i and $q(s_i, a_i)$ over all collected transitions. If we update the Q estimation every time by only one transition data, we then have an algorithm called Online Q-learning. By parameterizing the Q function, we can then perform the gradient method to minimize the objective squared loss.

Note that in this algorithm, transition data are collected by actually interacting with the environments, that is, the agent needs to execute an action to 'move forward. Now, the question is that how we should choose actions to be executed given the currently estimated Q function. Apparently, if the agent always picks the currently best action $\arg\max_a q_\theta(s,a)$ in any state, it will downplay the potential action that has a poor Q estimation now but leads to a better future return in fact. Thus, the agent needs some 'exploration' of actions other than the best action. This trade-off is a fundamental problem in RL, called the exploitation-exploration dilemma.

11.6 DQN

If we think deeper into the online Q-learning algorithm, we will find two issues. First, the y_i we compute is not a real target, since it depends on the model parameters. Thus, although we use gradients to optimize the objective function, it is not a real gradient method, which often leads to bad performance. Second, the transition data we use are not i.i.d, since the policy is changing due to the update of the Q function. The policy we use to generate transition is $\arg\max_a q_{\theta_t}(s,a)$ plus the ϵ-greedy exploration. As a result, the policy changes as the Q networks are being updated. (Using the policy we are optimizing to generate the transition data is usually classified as the *On-policy* RL algorithm.)

The DQN algorithm then serves as a remedy to these two problems. To summarize, two important tricks are proposed: the Replay Buffer and the delayed update of target Q networks. The DQN maintains two Q networks, and the weights of the target Q network are copied from the other Q network every C steps, while the other Q network is updated for each step. The data used to update the Q network are sampled from the Replay Buffer which contains a large number of past transitions. By doing so, the correlation among data can be mitigated. Please refer to that paper to get a better understanding.

11.7 Future Discussion

Low sample efficiency, because agents do not have prior knowledge about the world. Then, it may be helpful to let agents learn some meta-knowledge and then transfer that knowledge to downstream

challenging tasks. In order to allow input of this prior knowledge, several deep learning methods could have been taken. For instance, the well-known chess player alphaGo employed a convolutional neural network to recognize the structure of each chess state which becomes the input to each conditional distribution to be trained.