Intermediate Machine Learning

Lecture 1: Sparse Regression

Regression typically involves n pairs $(X_1, Y_1), \dots (X_n, Y_n)$

$Y_i \in \mathbb{R}$ is the response

$X_i \in \mathbb{R}^p$ is the covariate or feature vector

The goal is to predict new Y from a given X

Linear regression

$$m(x) = \mathbb{E}(Y|X)$$

Regression function

$$m(x) = \mathbb{E}(Y|X=x) = \int y\, p(y|x)\, dx$$

Prediction Error

$R = \mathbb{E}\left(Y - \hat{Y}\right)^2$ where $\hat{y}$ is the prediction based on training data

Bias-Variance Decomposition

$$R(\hat{m}) = \mathbb{E}\left(Y - \hat{m}(x)\right)^2 = \int \text{bias}^2(x)\, p(x)\, dx + \int \text{Var}(x)\, p(x) + \sigma^2$$

$$\text{bias} = \mathbb{E}\left(\hat{m}(x)\right) - m(x)$$

$$\text{Var} = \text{Variance}\left(\hat{m}(x)\right)$$

$$\sigma^2 = \mathbb{E}\left(Y - m(x)\right)^2 \leftarrow \text{Noise is intrinsic}$$

good prediction require a balance between bias and variance

Error can be decomposed into approximation error and estimation error

$$R(\hat{f}) - R^* = \underbrace{R(\hat{f}) - \inf_{f \in \mathcal{F}} R(f)}_{\text{estimation error} \approx \text{variance}} + \underbrace{\inf_{f \in \mathcal{F}} R(f) - R^*}_{\text{approximation error} \approx \text{bias}^2}$$

Goal is to fit the model

$$m(x) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p = \beta^T x$$

Don't assume true regression function is linear

minimize training error

$$\frac{1}{n} \sum_{i=1}^{n} \left(Y - \beta^T X\right)$$

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

$$\underset{n \times p}{X} = \begin{pmatrix} X_{11} & \cdots & X_{1p} \\ \vdots & & \vdots \\ X_{n1} & \cdots & X_{np} \end{pmatrix}$$

$$\hat{Y} = \hat{m}(x) = \hat{\beta}^T X$$

Improve the model to deal with high dimensions and more flexible predictions

Lecture 2: Sparse Regression continued

Ridge Regression

Penalizes large coefficient

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \beta^T X_i\right)^2 + \frac{\lambda}{n} \|\beta\|_2^2$$

$$\|\beta\|_2 = \sqrt{\sum_j \beta_j^2}$$

$$\hat{\beta} = \left(X^T X + \lambda I\right)^{-1} X^T Y$$

$\lambda = 0 \Rightarrow$ least squares

$\lambda = \infty \Rightarrow \hat{\beta} = 0$ (variance approaches 0)

Use leave-one-out cross-validation

1. leave out (X_i, Y_i)
2. Find $\hat{\beta}$
3. predict $Y_i = \hat{Y}_{(-i)} = \beta^T X_i$
4. Repeat for each i

Computational shortcut for bias correction

$$\hat{R}(\lambda) \approx R_{training} + \frac{2 P_\lambda \hat{\sigma}^2}{n}$$

$$H = X\left(X^T X + \lambda I\right)^{-1} X^T$$

$$P_\lambda = \text{trace}(H)$$

Ridge regression doesn't zero out any coefficients

We can use sparsity and convex relaxation to simplify the model selection process

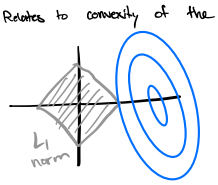
Sparsity: only a few predictors are relevant

Convex Relaxation: Replace model search with easier problem

$$\|\beta\|_q = \left(|\beta_1|^q + \ldots + |\beta_p|^q\right)^{1/q}$$

L-2 norm doesn't measure sparsity but L-1 and L-0 do

Relates to convexity of the norms (L-1 and L-0 are convex)



overlap tends to occur at corners which zero out certain coefficients

Lasso Regression

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{2n} \sum_{i=1}^{n} (Y_i - \beta^T X_i)^2 + \lambda \|\beta\|_1$$

← No closed form

$$\|\beta\|_1 = \sum_j |\beta_j|$$

Penalized regression requires standardized coefficients

Risk estimation in the lasso

1. Find $\hat{\beta}(\lambda)$ and $\hat{S}(\lambda)$ for each λ
2. Choose λ to minimize estimated risk
3. Let $\hat{S}$ be the selected variables
4. Let $\hat{\beta}$ be the least squares estimator using only $\hat{S}$
5. Prediction $\hat{Y} = X^T \hat{\beta}$

Lecture 3: Lasso, Smoothing and Kernels

Selecting λ

Re-fit the model with non-zero coefficients and apply leave-one-out cross validation

$$\hat{R}(\lambda) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_{(i)})^2 = \frac{1}{n}\sum_{i=1}^{n}\frac{(Y_i - \hat{Y}_i)^2}{(1-H_{ii})^2} \approx \frac{1}{n}\frac{RSS}{(1-S/n)^2}$$

H is hat matrix $\quad S = \#\hat{\beta}_j \neq 0$

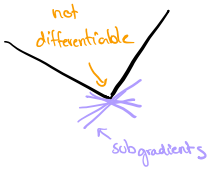
Choose λ to minimize risk

Algorithm for calculating Lasso

Consider $L = \frac{1}{2}(Y-\beta)^2 + \lambda|\beta|$

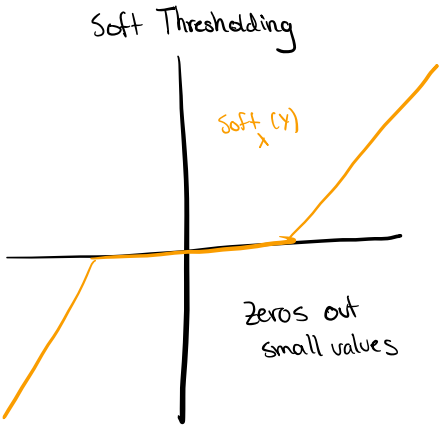
$$\frac{\partial L}{\partial \beta} = \beta - Y + \lambda r = 0$$

r: subgradient vector



$$\beta = Y - \lambda r = \begin{cases} Y-\lambda & \beta > 0 \\ Y+\lambda & \beta < 0 \\ Y - \lambda\frac{Y}{\lambda} & \beta = 0 \end{cases}$$

$$\Leftrightarrow \quad \hat{\beta} = \begin{cases} Y-\lambda & Y > \lambda \\ Y+\lambda & Y < \lambda \\ 0 & |Y| \leq \lambda \end{cases}$$



Compute least squares estimator and then run it through the soft thresholding function

Lasso by coordinate descent

1. Set $\hat{\beta} = (0, \ldots 0)$ then iterate until convergence
2. For $j = 1, \ldots, p$

   set $r_i = Y_i - \sum_{s \neq j}\hat{\beta}_s x_{s_i}$

   Set $\hat{\beta}_j$ to be least squares fit of r_i's on x_j

   $\hat{\beta}_j \leftarrow \text{Soft}_{\lambda_j}(\hat{\beta}_j)$ where $\lambda_j = \frac{\lambda}{\frac{1}{n}\sum_{x_i} x_{ij}^2}$
3. Use least squares $\hat{\beta}$ on selected subset $\hat{S}(\lambda)$

Nonparametric Regression

Two types of kernels

1. Smoothing kernels: local averaging
2. Mercer kernels: Regularization

Smoothing kernel Estimator

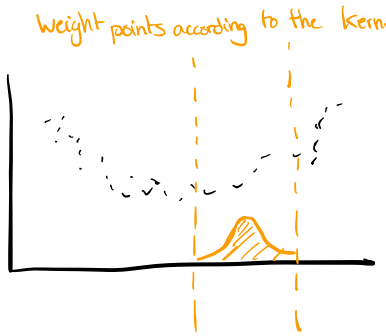
$$\hat{m}_h(x) = \frac{\sum_{i=1}^{n} Y_i K_h(X_i, x)}{\sum_{i=1}^{n} K_h(X_i, x)}$$

$\leftarrow$ weights; $\leftarrow$ normalization constant

$$K_h(x,z) = \exp\left(-\frac{\|x-z\|^2}{2h^2}\right)$$

$\leftarrow$ gaussian kernel

h is the bandwidth



Small h -> large variance, large h -> large bias

$$\text{Risk} = \mathbb{E}\left(Y - \hat{m}_h(X)\right)^2 = \text{bias}^2 + \text{variance} + \sigma^2$$

Under mild assumptions

$$\text{bias}^2 \approx h^4$$

$$\text{variance} \approx \frac{1}{nh^p}$$

The curse of dimensionality

risk goes down no faster than $n^{-4/(4+p)}$

To make risk of size ϵ we need

$$n \geq \left(\frac{1}{\epsilon}\right)^{1+p/4}$$ ← sample size grows exponentially

We choose h by estimating risk $R(h)$

1. Omit (X_i, Y_i) to get $\hat{m}_{h(i)}$ then predict $\hat{Y}_{(i)} = \hat{m}_{h(i)}(X_i)$
2. Repeat for all observations
3. Cross validation risk estimate

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_{(i)}\right)^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{Y}_i}{1 - L_{ii}}\right)^2$$

$$L_{ii} = \frac{K_h(X_i, X_i)}{\sum_{i'=1}^{n} K_h(X_i, X_{i'})}$$

Additive Models

compromise between linear models and nonparametric models

$$\hat{m}(x) = \hat{m}_1(x_1) + \cdots \hat{m}_p(x)$$

Each $\hat{m}_j(x_j)$ is estimated by smoothing while holding other functions fixed

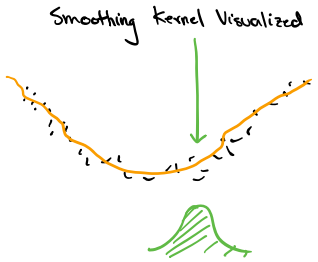
Lecture 4: Smoothing and Density Estimation

Smoothing Kernel

$$\hat{m}_h(x) = \frac{\sum_{i=1}^n Y_i K_h(X_i, x)}{\sum_{i=1}^n K_h(X_i, x)}$$ ← local weighted average of Y_i's near x

$$K_h(x, z) = \exp\left(-\frac{\|x-z\|^2}{2h^2}\right)$$

$h > 0$ is the bandwidth



h controls bias variance tradeoff

Small h → high variance

large h → high bias

Estimating Risk via Cross-Validation

① Omit (X_i, Y_i) to get $\hat{m}_{h(i)}$ and predict $\hat{Y}_{(i)} = \hat{m}_{h(i)}(X_i)$

② Repeat for all observations

③ Cross-validated estimate of risk

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^n (Y_i - \hat{Y}_{(i)})^2$$

Shortcut formula

$$\hat{R}(h) = \frac{1}{n}\sum_{i=1}^n \left(\frac{Y_i - \hat{Y}_i}{1 - L_{ii}}\right)^2$$

Proof →

$$\hat{Y}_{(i)} = \frac{\sum_{j \neq i} K_{ij} Y_i}{\sum_{j \neq i} K_{ij}} = \frac{\sum_j K_{ij} Y_j - K_{ii} Y_i}{\sum_j K_{ij} - K_{ii}} = \frac{\sum_j L_{ij} Y_j - L_{ii} Y_i}{1 - L_{ii}} = \frac{\hat{Y}_i - L_{ii} Y_i}{1 - L_{ii}}$$

$$\Rightarrow \left(Y_i - \hat{Y}_{(i)}\right)^2 = \left(Y_i - \frac{\hat{Y}_i - L_{ii} Y_i}{1 - L_{ii}}\right)^2 = \left(\frac{Y_i - \hat{Y}_i}{1 - L_{ii}}\right)^2$$

$$L_{ii} = \frac{K_h(X_i, X_i)}{\sum_\ell K_h(X_i, X_\ell)}$$

where $K_{ij} = K_h(X_i, X_j)$

General procedure for choosing bandwidth

① Compute $\hat{m}_h$ for each h

② Estimate risk $\hat{R}(h)$ using LOOCV

③ Choosing $\hat{h}$ to minimize $\hat{R}(h)$

④ Let $\hat{m}(x) = \hat{m}_{\hat{h}}(x)$

Kernel Density Estimation

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^n K_h(X_i, x) = \frac{1}{n} \cdot \frac{1}{h^p} \sum_{i=1}^n K\left(\frac{X_i - x}{h}\right)$$

↑ why

We require $\int K(u)\,du = 1$ and $K \geq 0$ is symmetric around 0

Bias-variance tradeoff

$\text{bias}^2(x) \asymp h^4$

$\text{var}(x) \asymp \frac{1}{nh^p}$

Derivation

$$\hat{f}(x) = \frac{1}{nh^p}\sum_{i=1}^n K\left(\frac{x - X_i}{h}\right)$$

Bias: $\mathbb{E}\,\hat{f}(x) - f(x)$

$$\mathbb{E}\,\hat{f}(x) = \frac{1}{nh^p}\sum_{i=1}^n \mathbb{E}\left(K\left(\frac{x - X_i}{h}\right)\right) = \frac{1}{h^p}\mathbb{E}\left(K\left(\frac{x - X_i}{h}\right)\right) = \frac{1}{h^p}\int K\left(\frac{u-x}{h}\right) f(u)\,du = \int K(r)\, f(x + hr)\,dr$$

($f(u)$: true density)

change of variable: $\frac{u-x}{h} = r$, $u = x + hr$, $du = h^p dr$

$$= \int K(r)\left\{f(x) + hr^T \nabla f(x) + \frac{h^2}{2} r^T \nabla^2 f(x) r + O(h^2)\right\} dr$$

taylor expansion around x as $h \to 0$

$$= f(x)\underbrace{\int K(r)\,dr}_{1} + \nabla f(x)^T h \underbrace{\int r\,K(r)\,dr}_{0} + \frac{h^2}{2}\int K(r)\, r^T \nabla^2 f(x) r\,dr + O(h^2) \leq ch^2$$

0: expected value of even function

$\therefore$ bias $\leq ch^2$

Variance : $\mathbb{E}[\hat{f}(x)^2] - \mathbb{E}[\hat{f}(x)]^2$

$$\mathbb{E}[\hat{f}(x)] = C_2 \frac{n}{n^2 h^{2p}} \int K\left(\frac{u-x}{h}\right)^2 f(u)\,du = \frac{C_2}{nh^p} \int K(r)^2 f(x+hr)\,dr = \frac{C_2}{nh^p} f(x) \int K(r)^2\,dr + O\left(\frac{1}{nh^p}\right)$$

same change of variables

taylor approx

$$\therefore \text{ var} \leq \frac{1}{nh^p}$$

$$\boxed{\text{Risk} \leq C_1 h^4 + C_2 \frac{1}{nh^p}}$$

Heuristically, we can set $h^4 = \frac{1}{nh^p}$

$$\text{Risk} \approx \left(\frac{1}{n}\right)^{4/4+p}$$

$$h = \frac{1}{n^{1/4+p}}$$

Sample size grows exponentially as we increase dimensions to maintain the same risk levels

Kernel smoothing estimator is a plug-in estimator of the kernel density estimator

subject to the same curse of dimensionality

Lecture 5: Mercer kernels

At a high level mercer kernels are penalties to smoothing

A kernel is a bivariate function $K(x, x')$ that can be used as a measure of similarity between x and x'

A mercer kernel has a special property that for n points the $n \times n$ matrix where

$\mathbb{K}_{ij} = [K(x_i, x_j)]$ is positive semi-definite ← necessary and sufficient condition

↑ no negative eigenvalues

Gaussian and triangular kernels are mercer kernels

Mercer kernel $K(x,x')$ is symmetric and positive semi-definite bivariate function

$$\iint f(x) f(x') K(x, x')\,dx\,dx' \geq 0$$

for all univariate functions f

We can create a set of basis functions based on K by fixing z

$$K(z, x) = K_z(x)$$

Defining a norm from the kernel

Suppose $f(x) = \sum_r \alpha_r K_{z_r}(x)$ and $g(x) = \sum_s \beta_s K_{y_s}(x)$

$$\langle f, g \rangle_K = \sum_r \sum_s \alpha_r \beta_s K(z_r, y_s)$$

inner product

$$= \alpha^T \mathbb{K} \beta$$

$$\mathbb{K} = [K(z_r, y_s)]$$

Positive semidefinite condition ensures that the inner product is positive

$$\|f\|_K^2 = \langle f, f \rangle_K = \sum_r \sum_s \alpha_r \alpha_s K(z_r, z_s)$$

$$= \alpha^T \mathbb{K} \alpha \geq 0$$

Reproducing Kernel Hilbert Space

$$\langle f, K_x(\cdot)\rangle_K = f(x)$$

reproduces the function through the inner product

Norm allows us to penalize overly complex functions

Minimize $\sum_{i=1}^{n} (Y_i - m(X_i))^2 + \lambda \|m\|_K^2$ ← ridge regression

over RKHS

Representer Theorem

Let $\hat{m}$ minimize

$$J(m) = \sum_{i=1}^{n} (Y_i - m(X_i))^2 + \lambda \|m\|_K^2$$

Then

$$\hat{m}(x) = \sum_{i=1}^{n} \alpha_i K(X_i, x)$$

for some $\alpha_1, \ldots, \alpha_n$

Plugging $\hat{m}$ into J we get

$$J(\alpha) = \|Y - K\alpha\|^2 + \lambda \alpha^T K \alpha$$ ← Kernel ridge regression

to minimize J

$$\hat{\alpha} = (K + \lambda I)^{-1} Y$$

$$\hat{m}(x) = \sum_i \hat{\alpha}_i K(X_i, x)$$

Estimator depends on regularization parameter λ

We choose via cross validation

Smoothing Kernels vs. Mercer Kernels

Smoothing kernels: bandwidth h controls the amount of smoothing

Mercer Kernels: norm $\|f\|_K$ controls the amount of smoothing

Both subject to the curse of dimensionality

Kernels from features

If $X \to \varphi(x) \in \mathbb{R}^d$, we can define a Mercer kernel

$$K(x, x') = \varphi(x)^T \varphi(x')$$

Conversely we can use the spectral theorem to get a feature map from a Mercer kernel

Lecture 6: Mercer Kernels and Neural Networks

Basics of Neural Nets

In linear regression our loss function is given by

$$L(x,y) = \frac{1}{2}(y - \beta^T x - \beta_0)^2$$

$$= \frac{1}{2}(y - f(x))^2$$

$f(x) = \beta^T x + \beta_0$

If we add a layer

$$L = \frac{1}{2}(y - f(x))^2$$

$f(x) = \beta^T h(x) + \beta_0$ where $h(x) = Wx + b$

However this model is still linear, just reparameterized

Adding non-linearities

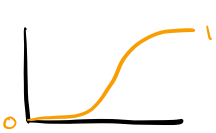
$h(x) = \varphi(Wx + b)$ ← Activation threshold

Common Non-linearities

$\tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$

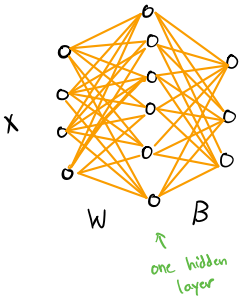
$\text{sigmoid}(u) = \frac{e^u}{1 + e^u}$

$\text{relu}(u) = \max(u, 0)$



← piecewise linear boundaries

Multi-layer Perceptron



Neural Networks are just parametric regression models with a restricted type of non-linearity

Backpropagation

Calculate derivatives via chain rule starting from the final layer

Classification

To calculate final probabilities we use

$$(P_1, P_2, P_3) = \frac{1}{e^{f_1} + e^{f_2} + e^{f_3}} \left(e^{f_1}, e^{f_2}, e^{f_3}\right)$$

Lecture 7: Neural Networks and Convolutional Neural Networks

Neural Nets without non-linear components are simply least squares or logistic regression

Stochastic Gradient Descent

$$\theta \leftarrow \theta - \eta \nabla_\theta \text{Loss}(\theta)$$

η: step size

Compute gradients via chain rule

Work your way backwards from the final node

Neural Nets are robust to overfitting

Double Descent

$$f(x) = \beta^T \varphi(Wx)$$

x: inputs; W: first layer weights; φ: activation function

$$x \in \mathbb{R}^d$$
$$W \in \mathbb{R}^{p \times d}$$

If we fix W to be gaussian weights

$$\varphi(Wx) = h(x)$$

$h(x)$: random features

Linear model in the features

(x_i, y_i) for $i = 1, \dots, n$

$$X_i = h(x_i) \in \mathbb{R}^p$$

Consider (X_i, y_i)

$$\mathbb{X} \in \mathbb{R}^{n \times p}$$
$$\mathbb{X}_i = X^T$$

Classical Regime $(n > p)$

$$\text{OLS}: \quad \hat{\beta} = (\mathbb{X}^T \mathbb{X})^{-1} \mathbb{X}^T Y$$
$$\hat{y} = \mathbb{X} \hat{\beta}$$

Over-parameterized Regime $(n < p)$

OLS doesn't exist

Minimum norm solution

$$\hat{\beta}_{mn} = \mathbb{X}^T (\mathbb{X} \mathbb{X}^T)^{-1} Y \in \mathbb{R}^p$$

$$\hat{y} = \mathbb{X} \hat{\beta}_{mn} = \mathbb{X} \mathbb{X}^T (\mathbb{X} \mathbb{X}^T)^{-1} Y = Y$$

Functions to interpolate

$\nexists \hat{y}_i = m(x_i)$?

$$(\beta - \hat{\beta}_{mn})^T \hat{\beta}_{mn} = (\beta - \hat{\beta}_{mn})^T X^T (XX^T)^{-1} Y$$

$$= \left(X(\beta - \hat{\beta}_{mn})\right)^T (XX^T)^{-1} Y = 0$$

$$\|\beta\|^2 = \|\beta - \hat{\beta}_{mn} + \hat{\beta}_{mn}\|^2$$

$$= \|\hat{\beta}_{mn}\|^2 + \|\beta - \hat{\beta}_{mn}\|^2 \geq \|\hat{\beta}_{mn}\|^2$$

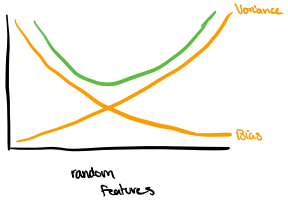
Ridge Regression

$$\hat{\beta}_\lambda = \left(X^T X + \lambda I_p\right)^{-1} X^T Y \xrightarrow{\lambda \to 0}$$

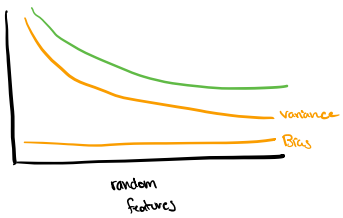
- OLS — Classical regime
- minimum norm — Over parameterized regime

Bias-Variance Trade off

Classical Regime



Over parameterized Regime



Kernel Connection

Consider the neural net as a function

$$f(\theta(x))$$

Neural Tangent Kernel -> Associated kernel for neural net

$$K_\theta(x, x') = \nabla_\theta f(x)^T \nabla_\theta f(x')$$

derivatives of the function

In the random features model

$$\nabla_\theta f_\theta(x) = \nabla_B B^T h(x) = h(x) \in \mathbb{R}^P$$

$$K = HH^T$$ ← mercer kernel

Lecture 8: Convolutional Neural Networks

Convolution sweeps a function $k(u)$ over the values of another function

$$K * f(x) = \int K(x-u) f(u) du$$

Same operation as smoothing kernels

Convolution can be considered a linear operation

Filters can be used as an edge detector

Principles of CNN

- Parameter Tying: let neurons share the same weights
- Kernel Learning: kernels should not be fixed, instead they should be learned
- Invariance: Doesn't matter where features are selected

CNN is most commonly used with images or spacial data

Tensor is an n-dimensional array

Convolutional Layer

- Set of kernels are swept over the input
- Kernels are tensors with trained weights
- Resulting convolution is the feature map
- Equivalent to a traditional layer but weights are constrained to be sparse and tied

Adding nonlinearity

hit the featuremap with an intercept and an activation function

Convolutional layers are intended to learn "features" of the input

Pooling the feature map reduces dimension

Max pooling: max value in a block

Average pooling: Average value in a block

Compounded convolutional layers can learn features of features

First layer tends to be edge detection

Second layer builds shapes from edges

Backpropagation

Forward pass builds predictions

Backward pass propagates predictive error to update weights

Typically each kernel has the same number of channels as the input tensor

Each kernel outputs a feature map

Recombine feature maps to channels

channels in output is # kernels in the layer

First dimension of input output tensor is always n

Dropout

Randomly zero out entries in a tensor

prevents overfitting

Flattening

Add dense layers after one or more convolutional layers

Standard layers used to make the final prediction

Lecture 9: Nonparametric Bayes: Gaussian Processes

Nonparametric Bayes

Extension of bayes inference to infinite dimensions

Typically neither prior nor posterior have a defined density

Well defined posterior

Statistical Problem	Frequentist Approach	Bayesian Approach
Regression	Kernel Smoother	Gaussian Process
CDF Estimation	Empirical CDF	Dirichlet Process
Density Estimation	Kernel Density estimation	Dirichlet Process mixture

Stochastic process is a collection of random variables with a shared text

Gaussian Process

$Y_i = m(X_i) + \epsilon_i \quad i = 1, \dots, n$

Frequentist Kernel Estimator

$$\hat{m}(x) = \frac{\sum_{i=1}^{n} Y_i K\left(\frac{x - X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)}$$

K: kernel, h: width

Bayesian method requires a gaussian prior

Exploits gaussian conditional/margins

Slices of multivariate gaussians are gaussian with constant variance

A stochastic process is a gaussian process if for each set of points $X_1, \dots X_n$ the vector $m(X_1), \dots m(X_n)$ is normally distributed

$$(m(X_1), \dots m(X_n))^T \sim N(\mu(x), K(x))$$

K: mercer kernel

Gaussian Process Prior

$$\pi(m) = (2\pi)^{-n/2} |K|^{-1/2} \exp\left(-\frac{1}{2} m^T K^{-1} m\right)$$

Change of variable

$$\pi(\alpha) = (2\pi)^{-n/2} |K|^{-1/2} \exp\left(-\frac{1}{2} \alpha^T K \alpha\right)$$

Gaussian prior favors functions with weight on eigenvectors with large eigenvalues

Smoother functions

Lecture 10: Nonparametric Bayes: Gaussian Processes + Dirichlet Processes

Maximum a posteriori estimate for gaussian process is equivalent to mercer kernel regression

Prior prefers smooth functions but the definition of smoothness is dictated by the mercer kernel (bandwidth)

If X_1 and X_2 are jointly gaussian, then the conditional distributions are also gaussian

Covariance of both distributions is independent of X

Predicting a New Point

Let $K = \left(K(X_1, X_{n+1}), \ldots, K(X_n, X_{n+1})\right)$

Then $Y_1 \ldots Y_{n+1}$ are jointly gaussian with covariance

$$\begin{pmatrix} K + \sigma^2 I & K \\ K^T & K(X_{n+1}, X_{n+1}) + \sigma^2 \end{pmatrix}$$

The posterior mean and variance are as follows

$$\mathbb{E}(Y_{n+1} | X_{1:n}, Y_{1:n}) = K^T (K + \sigma^2 I)^{-1} Y$$

$$\mathrm{Var}(Y_{n+1} | X_{1:n}, Y_{1:n}) = K(X_{n+1}, X_{n+1}) + \sigma^2 - K^T (K + \sigma^2 I) K$$

↑ Select via domain knowledge

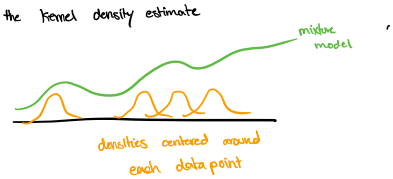
The Dirichlet Process is analogous to the gaussian process

Method for density estimation instead of regression

Bayesian approach to density estimation

Every partition of sample space has a dirichlet distribution

Recall the Kernel density estimate



Building the Bayesian Model

Construct prior via imaginary data ← guess a beta distribution

Each sample from a Dirichlet prior has a random collection of weights assigned to a random selection of data

Each sample from a Dirichlet process mixture has a random collection of weights assigned to a random selection of model parameters

Stick Breaking Process for Dirichlet Process mixture

1. Draw $\theta_1, \theta_2, \ldots$ independently from F_0
2. Draw $V_1, V_2, \ldots \sim \mathrm{Beta}(1, \alpha)$

   Set $w_j = v_j \prod_{i=1}^{j-1} (1 - v_i)$
3. Fixture model

$$f(x) = \sum_{j=1}^{\infty} w_j f(x | \theta_j)$$

A dirichlet process is a stochastic process where every partition of the sample space has a dirichlet distribution

We use an algorithm to sample from the posterior

No closed form

Lecture 11: Approximate Inference: Simulation and Variational Methods

Simulation and variational methods are ways to work with Bayesian posteriors

Generative Model

1. Choose Z ← training point
2. Given Z, generate X ← gaussian sample

Inverting generative Model

1. Given X
2. What Z generated it?

Ising Model

$z_1 \ldots z_n$ binary variables

$$P_\beta(z_1, \ldots z_n) \propto \exp\left(\sum_{s \in V} B_s z_s + \sum_{(s,t) \in E} B_{st} z_s z_t \right)$$

Gibbs Sampler

1. Choose vertex $s \in V$ at random
2. Sample $u \sim \text{Uniform}(0,1)$

$$z_s = \begin{cases} 1 & u \le \left(1 + \exp\left(-B_s - \sum_{t \in N(s)} B_{st} z_t\right)\right)^{-1} \\ 0 & \text{otherwise} \end{cases}$$

← akin to logistic

↑ conditional probability $z_s = 1$

3. Iterate

Encourages neighboring pixels to be similar

Gibbs sampling is a stochastic approximation

Deterministic Approximation

Mean Field Approximation

1. Choose vertex $s \in V$ at random
2. Update $\mu_s = \left(1 + \exp\left(-B_s - \sum_{t \in N(s)} B_{st} \mu_t\right)\right)^{-1}$
3. Iterate

Mixture Model

Fix two distribution F_0 and F_1 with densities $f_0(x)$ and $f_1(x)$

$\theta \sim \text{Beta}(\alpha, \beta)$ ← mixing coefficient

$X|\theta \sim \theta F_1 + (1-\theta) F_0$

Likelihood

$$p(x_{1:n}) = \int_0^1 \text{Beta}(\theta|\alpha,\beta) \prod_{i=1}^n (\theta f_1(x_i) + (1-\theta) f_0(x_i)) d\theta$$

Gibbs Sampler

1. Sample $z_i | \theta, x_{1:n}$
2. Sample $\theta | z_{1:n}, x_{1:n}$

For step 1 sample $u_i \sim \text{Uniform}(0,1)$

$$z_i = \begin{cases} 1 & u_i \le \frac{\theta f_1(x_i)}{\theta f_1(x_i) + (1-\theta) f_0(x_i)} \\ 0 & \text{otherwise} \end{cases}$$

For step 2

$$\theta \sim \text{Beta}\left(\sum_{i=1}^n z_i + \alpha, n - \sum_{i=1}^n z_i + \beta\right)$$

Lecture 12: Variational Inference and VAEs

Inverting Models

Bayesian set up:

1. Choose θ
2. Given θ, generate X sample

Posterior Inference:

1. Given X
2. What θ generated x?

Approximate these answers with stochastic or deterministic methods

Gibbs Sampling → Stochastic

Variational Methods → Deterministic

Stochastic Approximation (Gibbs Sampler)

Iterate until convergence

1. Choose vertex $s \in V$ at random
2. Sample z_s while holding others fixed

$$\theta_s = \text{sigmoid}\left(B_s + \sum_{t \in N(s)} B_{st} z_t\right)$$

$$z_s \mid \theta_s \sim \text{Bernoulli}(\theta_s)$$

In Bayesian notation

1. Select a component s
2. Calculate conditional distribution $p(\theta_s \mid \theta_t, t \neq s, x)$
3. Sample

} Repeat

Low dimensional so it is tractable calculation

Deterministic Approximation (Mean field variational)

Iterate until convergence

1. Choose vertex $s \in V$ at random
2. Update mean μ_s holding others fixed

$$\mu_s = \text{sigmoid}\left(B_s + \sum_{t \in N(s)} B_{st} \mu_t\right)$$

sigmoid: $\frac{1}{1-e^{-x}}$

Comparison of Deterministic and Stochastic Approximation

- All z_s are random
- The μ_s are deterministic
- Gibbs converges in distribution but mean field is numerical
- Gibbs approximates full distribution but mean field only approximate mean of each node

A finite Mixture Model

Consider two distributions F_0 and F_1 with densities $f_0(x)$ and $f_1(x)$ with the following mixtue model

mixing parameter

$$\theta \sim \text{Beta}(\alpha, \beta)$$

$$X \mid \theta \sim \theta F_1 + (1-\theta) F_0$$

Likelihood for given data

$$p(X_{1:n}) = \int_0^1 \text{Beta}(\theta \mid \alpha, \beta) \prod_{i=1}^{n} (\theta f_1(x_i) + (1-\theta) f_0(x_i))\, d\theta$$

Gibbs Sampling Approach

1. Sample $z_i \mid \theta, X_{1:n}$ for $i = 1, \ldots n$

repeat

$$z_i = \begin{cases} 1 & \propto \theta f_1(x_i) \\ 0 & \propto (1-\theta) f_0(x_i) \end{cases} \quad \leftarrow \quad p(z_i = 1) = \frac{\theta f_1(x_i)}{\theta f_1(x_i) + (1-\theta) f_0(x_i)}$$

2. Sample $\theta \mid z_{1:n}, x_{1:n}$

$$\theta \sim \text{Beta}\left(\sum_{i=1}^{n} z_i + \alpha,\ n - \sum_{i=1}^{n} z_i + \beta\right)$$

Sampled from a mixture of beta distributions over multiple iterations

Collapsed Gibbs Sampler

better algo

Integrate out θ and sample $z_1 \ldots z_n$

Variational Inference (Deterministic Approximation)

Goal: compute $p(\theta, z \mid x)$

Since the goal is too complicated, we approximate via $q(\theta, z)$ that has a nice form (fixed x)

q is a function of variational parameters optimized for each x

Maximize lower bound on $p(x)$ (evidence)

Bound $p(x)$ or alternatively $\log p(x)$

$$\log p(x) = \log p(x) \int q(\theta, z)\, d\mu(\theta, z)$$

$$= \int q(\theta,z) \log p(x)\, d\mu(\theta,z) = \int q(\theta,z) \log \left(\frac{p(x)\, p(x,\theta,z)}{p(x,\theta,z)} \cdot \frac{q(\theta,z)}{q(\theta,z)} \right) d\mu(\theta,z)$$

Rearrange

$$\log p(x) = \int q(\theta,z) \log \left(\frac{p(\theta,x,z)}{p(\theta,z|x)} \cdot \frac{q(\theta,z)}{q(\theta,z)} \right) d\mu(x,z)$$

$$= \underbrace{\int q(\theta,z) \log \left(\frac{q(\theta,z)}{p(\theta,z|x)} \right) d\mu(\theta,z)}_{\text{drop non-negative term}} + \int q(\theta,z) \log \left(\frac{p(\theta,x,z)}{q(\theta,z)} \right) d\mu(\theta,z)$$

KL divergence

$$\log p(x) \geq \int q(\theta,z) \log \frac{p(\theta,x,z)}{p(\theta,z)}\, d\mu(\theta,z)$$

$$= \mathbb{E}_q \log p(\theta,x,z) + H(q)$$

$\mathbb{E}_q \log p(\theta,x,z)$: cross entropy — favors fit to the data

$H(q)$: entropy — favors spreading out variational distribution

ELBO: Evidence Lower Bound

Numerically optimize w.r.t terms (variational parameters)

Variational Algorithm for Mixture

Iterate for $q_{1:n}$ and γ_1, γ_2

1. Holding q_i fixed set

$$\gamma_1 = \alpha + \sum_{i=1}^{n} q_i \qquad \gamma_2 = \beta + n - \sum_{i=1}^{n} q_i$$

2. Holding γ_1 and γ_2 fixed set

$$q_i = \frac{f_1(x_i) \exp \psi(\gamma_1)}{f_1(x_i) \exp \psi(\gamma_1) + f_0(x_i) \exp \psi(\gamma_2)}$$

After convergence

$$\hat{p}(\theta | x_{1:n}) = \text{Beta}(\theta | \gamma_1, \gamma_2)$$

Easier to use than gibbs sampling

Posterior is estimated as a single Beta distribution

Lecture 12: Variational Autoencoders

Variational Inference: Strategy

goal is to compute $p(\theta, z|x)$ but that's too complicated

Instead, we approximate with $q(\theta, z)$ that has a nice form ← typically gaussian

q is a function of variational parameters that is optimized for each x

Maximize a lower bound on $p(x)$

ELBO is the lower bound on $\log p(x)$

$$\log p(x) = H(q) + \mathbb{E}_q(\log p(x,z,\theta))$$

$H(q)$: entropy

maximize over the parameters of q

Entropy is maximized when the variational distribution is spread out

$$\text{ELBO} = H(q) + \mathbb{E}_q \log p(x,\theta,z)$$

$$H(q) + \mathbb{E}_q \log p(x|z,\theta) - \mathbb{E}_z \log p(z,\theta) \quad \leftarrow \text{prior}$$

$$-\int (q \log q + q \log p) = -D(q\|p) = -\int q(z,\theta) - \log \frac{q(z,\theta)}{p(z,\theta)} \, d\mu(z,\theta)$$

$$\text{ELBO} = \mathbb{E}_q \log p(x|z,\theta) - D(q\|p)$$

Equivalent to minimizing

$$-\text{ELBO} = -\mathbb{E}_q \log p(x|z,\theta) + D(q\|p)$$

Data term: log prob of data

↖ regularizer making variational distribution close to prior distribution

Variational Autoencoders

Generative models that are trained via variational inference

"Decoder" is a neural net that generates from a latent variable

"Encoder" approximates posterior distribution with another neural network trained using variational inference

Example: Generative Model

$$z \sim N(0, I_n)$$

$$x|z \sim N(G(z), I_d)$$

generator network or decoder

$$G(z) = A_2 \varphi(A_1 z + b_1) + b_2$$

We want to estimate $p(z|x)$ but since G is non-linear, we estimate via variational inference

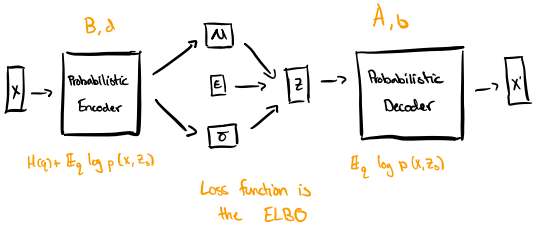
$$q(z|x) = N(\mu_x, \sigma_x^2 I_n)$$

variational parameters (shared across images)

We make an encoder neural network to output mean and variance

$$\mu(x) = B_2 \varphi(B_1 x + d_1) + d_2$$

Similar for $\log \sigma^2(x)$



We approximate $\mathbb{E}_q(\log p(x,z))$ via sampling

$$\mathbb{E}_q(\log p(x,z)) \approx \frac{1}{N} \sum_{s=1}^{N} \log p(x, z_s)$$

parameters of network have disappeared!

Reparameterize $z_i = \mu(x) + \sigma(x)\epsilon_i$, $\epsilon_i \sim N(0, I_n)$

Example: Suppose $x|z \sim N(G(z), I)$ where

$$G(z) = \varphi(Az + b)$$

Then $-\log p(x|z)$

$$\frac{1}{2}\|x - \varphi(Az+b)\|^2 + \text{constant}$$

$-\log(z)$

$$\frac{1}{2}\|z\|^2 + \text{constant}$$

Further suppose $q(x) = N(\mu(x), \sigma^2(x) I)$

FFN outputs

$$\mu(x) = \varphi(Bx + d)$$

$$-\mathbb{E}_p \log p(x|z) \simeq \frac{1}{N}\sum_{s=1}^{N} \frac{1}{2}\|x-\psi(Az_s+b)\|^2$$

$$= \frac{1}{N}\sum_{s=1}^{N} \frac{1}{2}\|x-\psi(A(\mu(x)+\epsilon_s\sigma^2(x))+b)\|^2$$

$$= \frac{1}{N}\sum_{s=1}^{N} \frac{1}{2}\|x-\psi(A(\psi(Bx+d)+\epsilon_s\sigma^2(x))+b)\|^2$$

Regularizer

$$-\mathbb{E}_q \log p(z) = -\frac{1}{2}\mathbb{E}_q\|z\|^2 = \frac{1}{2}\|\mu(x)\|^2 + \frac{K}{2}\sigma^2(x)$$

$$-\mathbb{E}_q \log q(z) = \frac{K}{2}\log\sigma^2(x)$$

(K ← dimension of z)

↑ entropy

Tensor flow then calculates the gradients for you

Lecture 13: Sparsity and Graphs

graphs are a natural language for relationships

Undirected Graph

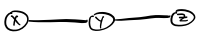
$G=(V,E)$ ← V vertices and E edges

If $X \in \mathbb{R}^p$, we look at graphs with p vertices

Inverse of covariance matrix is the precession matrix

shows conditional independence relationships

Example:



$X \perp\!\!\!\perp Z \mid Y$

X and Y are independent conditioned on Y

Markov Property

Probability distribution P satisfies global Markov property with respect to a graph G if

For any disjoint vertex subsets A,B,C s.t. C separates A and B

$$X_A \perp\!\!\!\perp X_B \mid X_C$$

- X_A are random variables X_j with $j \in A$
- C separates A, B means every path between A and B passes through C

Graph Estimation

Given n samples, estimate the graph G

Gaussian Case

$\Omega = \Sigma^{-1}$ ← precision matrix

0s in the precision matrix implies conditional independence

Parallel Lasso:

1. For each $j=1 \cdots p$, regress X_j on all other variables using lasso
2. Put an edge between X_i and X_j if each appears in the regression of the other

intersection and union rule have the same asymptotic result

Graphical Lasso:

- Assume multivariate gaussian model
- Subtract out sample mean ← center the data
- Minimize negative log-likelihood of the data s.t. constraint on sum of absolute values of inverse covariance

↖ encourages sparsity

Assuming a mean of 0

probability density

$$p(x) = \frac{1}{\sqrt{(2\pi)^p |\Sigma|}} \exp\left(-\frac{1}{2} \operatorname{tr}(\Omega x x^T)\right)$$

$$-\log p(x) = \frac{1}{2} \log |\Sigma| + \frac{1}{2} \operatorname{tr}(\Omega x x^T) = -\frac{1}{2} \log |\Omega| + \frac{1}{2} \operatorname{tr}(\Omega x x^T)$$

using $\log |A| = -\log |A^{-1}|$

Sum over all datapoints

$$-\sum_{i=1}^{n} \log p(x_i) = \frac{1}{2} \sum_{i=1}^{n} \operatorname{tr}(\Omega x_i x_i^T) - \frac{n}{2} \log |\Omega|$$

$$= \frac{n}{2} \operatorname{tr}(\Omega S_n) - \frac{n}{2} \log |\Omega|$$

S_n is sample Covariance

$$S_n = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T$$

↖ Convex function of Ω

Add a ℓ_1 penalty for sparsity

$$O(\Omega) = \operatorname{tr}(\Omega S_n) - \log |\Omega| + \lambda \sum_{k \neq j} |\Omega_{jk}|$$

Objective function

rescaled by $\frac{2}{n}$

Parallel and graphical lasso produce different graphs

Choosing λ

① Cross-Validation

tends to over select

② BIC = log-likelihood $- (p/2) \log n$

③ AIC = log-likelihood $- p$

Lecture 14: Discrete Data Graphs and Graph Neural Networks

Hammersley, Clifford, Besag Theorem

A positive distribution over random variables $z_1, \dots z_p$ satisfies the Markov properties of graph G iff it can be represented as

$$p(z) \propto \prod_{c \in C} \psi_c(z_c)$$

C is the set of cliques in the graph G

↑ maximally connected subgraph

Discrete graphical models don't have a closed form ← no normalizing constant

Need to use gibbs sampling and variational inference

Via Hammersley, Clifford and Besag Theorem

$$p(z; B) \propto \exp\left(\sum_{c \in C} B_c \phi_c(z_c)\right)$$

Ising Model

$$p(z; B) = \exp\left(\sum_{i \in V} B_i z_i + \sum_{(i,j) \in E} B_{ij} z_i z_j\right)$$

Stochastic Approximation : Gibbs Sampling

Deterministic Approach: Mean field variational algorithm

No analog for the graphical lasso

Parallel Lasso: penalized logistic regression on each node z_i to estimate neighbors

- Create edge (i,j) if $j \in \hat{N}(i)$ and $i \in \hat{N}(j)$ } Asymptotically equal

- Create edge (i,j) if $j \in \hat{N}(i)$ or $i \in \hat{N}(j)$

Scaling Behavior

Ising Model: $n \geq d^3 \log p$

Graphical Lasso: $n \geq d^2 \log p$

Parallel Lasso: $n \geq d \log p$

Lower bound: $n \geq d \log p$

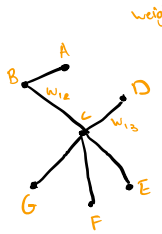
} Varying incoherence assumptions

Graph Neural Networks

Graph Laplacian

$L = D - W$

degree matrix, weight matrix



weights $W \in \mathbb{R}^{6\times 6}$

	A	B	C	D	E	F	G
A	1	-1					
B	-1	2	-1				
C		-1	5	-1	-1	-1	-1
D			-1	1			
E			-1		1		
F			-1			1	
G			-1				1

For p nodes $L \in \mathbb{R}^{p\times p}$

Polynomials of Laplacian are analagous to CNN filter

$$P_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \cdots w_d L^d$$

weights w_i are analagous to filter coefficients

degree d of the polynomial is like the size of the kernel

Laplacian is a Mercer kernel

$$f^T L f = \sum_{(u,v)\in E} (f_u - f_v)^2 \geq 0$$

Laplacian Operator

$\frac{\partial^2}{\partial x^2} + \cdots \frac{\partial^2}{\partial x_p^2}$ → elegant connection

Equivariance

A transformation is equivariant if

$$f(Px) = Pf(x)$$

where P is a permutation matrix $PP^T = I$

Building Layers

$$h^{(k+1)} = \varphi\left(P_w(L) h^{(k)}\right)$$

Lecture 15: Reinforcement Learning

Agent interacts with environment

Actions the agent takes change the state of the environment

Agent recieves a reward for each action and seeks to maximize cumulative reward

Framework for sequential decision making to achieve a long term goal

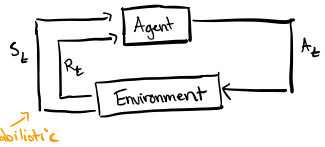
Formalization of Reinforcement Learning

- Environment is in state s at a given time
- Agent takes action a
- Environment transitions to state $s' = \text{next}(s,a)$
- Agent recieves reward $r = \text{reward}(s,a)$

} Markov Decision Process

goal is to make optimal decisions from experience

In between supervised and unsupervised learning



probabilistic

Policy: mapping from states to actions

Reward Signal: Sequence of rewards recieved at each time step

Value function: A mapping from states to total reward

Values are predictions of future rewards

Q-Learning

Maintains quality variable $Q(s,a)$ for taking action a in states

Measure of cumulative rewards obtained by algorithm when it takes action a in states

Algorithm

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(\text{reward}(s,a) + \gamma \max_{a'} Q(\text{next}(s,a),a') - Q(s,a)\right)$$

"Gradient Ascent"

discount factor

"weighting of short term vs. long term rewards"

Bellman Equation: Deterministic Case

Value:

$$V_*(s) = \max_a \left\{ \text{reward}(s,a) + \gamma V_*(\text{next}(s,a)) \right\}$$

Fixed point equation

Q-function:

$$Q_*(s,a) = \text{reward}(s,a) + \gamma \max_{a'} Q_*(\text{next}(s,a),a')$$

Q-Learning algorithm satisfies value function bellman equation

Lecture 16: Deep Reinforcement Learning

Q-Learning Algorithm is intended to estimate Q^*

Parameters: step size α, exploration probability ε, discount factor γ

Initialize $Q(s,a)$ arbitrarily

Loop for each episode

Initialize S

Loop for each step

Choose action A using Q with ε-greedy policy

Take action A, observe reward R and new state S'

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma \max_a Q(S',a) - Q(S,A))$$

best action for next state

$S \leftarrow S'$

Until S is terminal

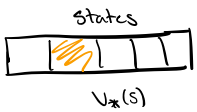
Q-learning algorithm is estimating bellman equation from data

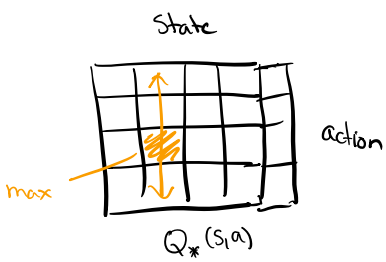
If we know Q_*,

$$V_*(s) = \max_a Q(s,a)$$

$$= \max_a \left\{ \text{reward}(s,a) + \gamma \max_{a'} Q_*(\text{next}(s,a),a') \right\}$$

$$= \max_a \left\{ \text{reward}(s,a) + \gamma V_*(\text{next}(s,a)) \right\}$$





Given Q_*, the system evolves deterministically

$$R_{total} = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$$

Bellman Equation : Random Environments

$$V_*(s) = \max_a \sum_{s',r} p(s',r|s,a)\{r + \gamma V_*(s')\}$$

$$= \max_a \mathbb{E}\left[R_{t+1} + \gamma V_*(S_{t+1}) \,\middle|\, S_t = s, A_t = a\right]$$

Direct Application of Q-learning is only possible for small state and action spaces

Iterating over all possibilities

For large state space we map states to "features"

Deep Reinforcement learning uses a multilayer neural network to learn features and the Q-function

Goal: Represent Q_* as a neural network

$$Q(s,a;\Theta_*) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \Theta_*) \,\middle|\, S_t = s, A_t = a\right]$$

Θ_* are weights in the neural network

Value iteration

i) $Q \leftarrow \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q_{current}(S_{t+1}, a') \,\middle|\, S_t = s, A_t = a\right]$

ii) Set $Q_{current}$ to Q as updated in i)

Can be seen as stochastic gradient descent w.r.t least-squared loss

$$(Y(s,a;Q_{current}) - Q(s,a))^2$$

Strategy

target network : $Q(\dots, \Theta_{current})$ ← used to evaluate bellman target

Policy network : $Q(\dots, \Theta)$

- $Y_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \Theta_{current})$
- Adjust parameters Θ to minimize squared error

$$(Y_t - Q(s,a,\Theta))^2$$

Use SGD / backpropagation

- Update $\Theta_{current}$ to Θ and repeat procedure

Learning takes place when $\mathbb{E}[Y] \neq Q$

Lecture 17: Policy Gradient Method

A policy is a mapping from states to action

A value function is a mapping from states to total reward

Rewards are short term and values are predictions of future rewards

Q-Learning

Parameters: step size α, exploration probability ϵ, discount factor γ, initialize $Q(s,a)$ arbitrarily except terminal

Loop for each episode:

Initialize state s

Loop for each step of episode:

Choose action a using Q with ϵ-greedy policy

Take action a; observe reward r and new state s'

$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$

$s \leftarrow s'$

Until s is terminal

Q-Learning is an example of temporal-difference learning

"Off-policy" approach that is only practical if the space of states and actions are small

Deep Reinforcement learning uses a multilayer neural network to build a map from states to features for large state spaces

General Strategy

$$Q(s,a;\theta) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q(s_{t+1};a,\theta) \mid S_t = s, A_t = a\right]$$

Sample Y_t from the conditional distribution

$$Y_t = R_{t+1} + \gamma \max_{a'} Q(s_{t+1},a';\theta_{old})$$

Adjust parameters θ to make squared error small

$$(Y_t - Q(s,a;\theta))^2$$

↑ via stochastic gradient descent

Learning happens when expectations are violated

Automatic Differentiation allows us to handle loss functions that are constructed dynamically

Calculate gradients on a "tape"

Use backpropagation on the "tape" to produce numeric gradients

Policy Iteration

0. Initialize Policy arbitrarily
1. Compute values for current policy (Policy Evaluation)
   Loop over all states and calculate expected reward
2. Update policy to match values (Policy improvement)
   Verify if policy agrees with Bellman equation
3. Return to step 1

Policy Gradient Calculation

Episode:

$$\underbrace{(s_0,r_0) \rightarrow (s_1,r_1,a_1) \rightarrow (s_2,r_2,a_2) \rightarrow \dots \rightarrow (s_t,r_t,a_t)}_{\tau} \rightarrow s_{t+1}$$

(s_{t+1}: terminal)

$$P(\tau|\theta) = \prod_{t=1}^{T} \underbrace{\pi_\theta(a_t|s_{t+1})}_{\text{policy}} \underbrace{P(s_{t+1},r_{t+1}|s_t,a_t)}_{\text{environment}}$$

$$R(\tau) = \sum_{t=1}^{T} r_t$$

($R(\tau)$: reward)

Objective-function:

$$J(\theta) = \mathbb{E}_\theta[R(\tau)]$$

Use gradient descent to optimize objective function

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

$$J(\theta) = \mathbb{E}[R(\tau)] = \int R(\tau)\, p(\tau|\theta)\, d\theta(\tau)$$

$$\nabla_\theta J(\theta) = \int R(\tau)\, \nabla_\theta p(\tau|\theta)\, d\theta(\tau)$$

$$= \int R(\tau)\, \frac{\nabla_\theta p(\tau|\theta)}{p(\tau|\theta)}\, p(\tau|\theta)\, d\theta(\tau)$$

$$= \mathbb{E}\left[R(\tau)\, \nabla_\theta \log p(\tau|\theta)\right]$$ ← can now sample trajectories

$$\approx \frac{1}{N} \sum_{s=1}^{N} R(\tau^{(s)}) \nabla_\theta \log p(\tau^{(s)}|\theta)$$

$$= \frac{1}{N} \sum_{s} \sum_{t}^{T} R(\tau^{(s)}) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Lecture 19: Reinforcement Learning: Actor-Critic Methods

Multi-armed Bandits

- Rewards are independent and noisy
- Arm k has payoff μ_k with variance σ_k^2 on each pull
- Each step pull an arm and observe the reward
- Can estimate mean reward of each arm

Exploration-exploitation trade-off

Policy Gradient Method: Loss Function

Policy: $\pi_\theta(a_t|s_t)$

Environment: $p(s_{t+1}, r_{t+1}|s_t, a_t)$

Episode:

$(s_0, r_0) \longrightarrow (s_1, r_1) \longrightarrow \dots \longrightarrow (s_T, r_T, a_T) \longrightarrow s_{T+1}$

terminal state

τ

policy parameters

$$p(\tau|\theta) : \prod_{t=1}^{T} \pi_\theta(a_t|s_t)\, p(s_{t+1}, r_{t+1}|s_t, a_t)$$

Gradient Descent Over policy parameters

$$\nabla_\theta \log p(\tau|\theta) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Total Reward

$$R(\tau) = \sum_{t=1}^{T} r_t \quad \text{(no discounting)}$$

Loss Function

$$J(\theta) = \mathbb{E}_\theta(R(\tau)) = \int R(\theta)\, p(\tau|\theta)\, d\theta$$

$$\nabla_\theta J_\theta = \int R(\theta) \nabla_\theta p(\tau|\theta)\, d\theta$$

$$= \int R(\tau) \frac{\nabla_\theta p(\tau|\theta)}{p(\tau|\theta)} p(\tau|\theta)\, d\theta$$

$$= \int R(\tau) \nabla_\theta \log p(\tau|\theta)\, p(\tau|\theta)\, d\theta$$

$$= \mathbb{E}_\theta\left(R(\tau) \nabla \log p(\tau|\theta)\right)$$

$$= \mathbb{E}_\theta\left[R(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

Sample

$$\approx \frac{1}{N} \sum_{s=1}^{N} R(\tau^{(s)}) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(a_t^{(s)}|s_t^{(s)}\right)$$

Actor-Critic Approach

- Estimate policy and value function together

  Actor is the policy used to select actions

  Critic is the value function used to evaluate the actor

After a selected action, critic evaluates the state and updates actor and value function accordingly

Error Signal

$$d_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

> 0 better than expected

< 0 less than expected

Value Function update

$V(s_t) = V(s_t) + \alpha \delta_t$

Policy Parameter update

use gradient and δ_t

Lecture 19: Sequence Models and Recurrent Neural Networks

Language models are ways of assigning probability to any sequence of words

$$p(w_1, \dots w_n) = p(w_1)p(w_2|w_1) \dots p(w_n|w_1, \dots w_{n-1})$$

↖ conditional probability decomposition

Generate text by sampling conditional distribution

Perplexity is a rescaling of log-likelihood

geometric mean is no greater than the arithmetic mean

$$\text{Perplexity}(\theta) = \left(\prod_{i=1}^{N} p_\theta(w_i|w_{1:i-1}) \right)^{-1/N}$$

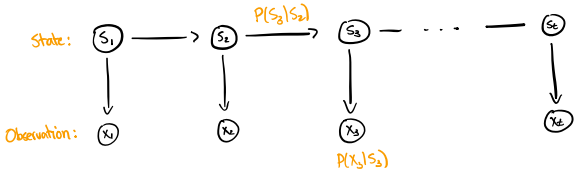
Inverse of the geometric mean of conditional probabilities evaluated on test data

Interpretation: if perplexity is 100, the model effectively predicts the next word with probability 1/100

geometric average "branching behavior" of model

Hidden Markov Models

- State is chosen stochastically
- Non-markov on observations



probability of a word sequence

$$p(x_1, \dots x_n) = \sum_{s_1 .. s_n} \prod_{t=1}^{n} p(s_t|s_{t-1})\, p(x_t|s_t)$$

computed via forward-backward algorithm

Recurrent neural networks are similar to HMM but they are not stochastic

Lecture 20: RNNs, LSTMs, GRUs, and All That

Recurrent neural networks evolve states deterministically

Not mixture models like HMMs

State is distributed, not categorical like HMMs

$$\text{State: } h_t = \tanh\left(\underset{3\times3}{W_{hh}} \underset{3\times1}{h_{t-1}} + \underset{3\times4}{W_{xh}} \underset{4\times1}{X_t} + \underset{3\times1}{b_h} \right)$$

3×3 matrix

↑ previous state; ↑ current character; ↖ bias

$$\underset{4\times3}{\quad} \quad y_t = \underset{3\times1}{W_{hy} h_t}$$

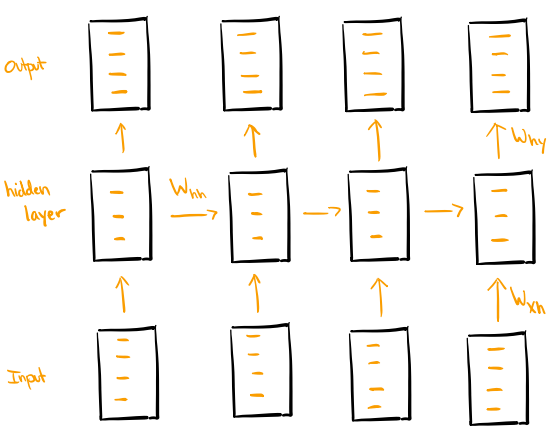
↖ vocabulary vector

$$X_{t+1}|y_t \sim \text{Multinomial}\left(\pi(y_t)\right)$$

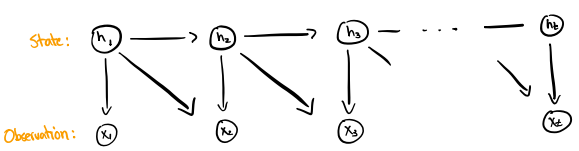
↑ soft-max

Calculate derivatives w.r.t. probabilities and use for stochastic gradient descent

Example RNN Structure



RNN Graphical Structure



LSTM: Long Short-Term Memory

gradients in traditional RNNs get really small or really big as the model learns

Adds a layer that "forgets" memory

Gated Recurring Units (GRU)

High level idea:

- Learn when to update hidden state to "remember" important information
- Keep information until they are used
- Reset or forget info when it isn't useful

GRUs rely on gates

$\Gamma = 1$: gate is open, information flows

$\Gamma = 0$: gate is closed, information is blocked

Two-types of gates

Γ^u : Long-term memory propogation when open, otherwise uses local state information

Γ^r : When closed, local gate is reset. When open, the state is updated as a vanilla RNN

State evolution

$$c_t = \tanh\left(W_{hx} x_t + W_{hh}\left(\Gamma^r \odot h_{t-1}\right) + b_h\right)$$

candidate state

ignore previous state when closed

$$h_t = \left(1 - \Gamma_t^u\right) \odot c_t + \Gamma_t^u \odot h_{t-1}$$

determines whether we use long term propogation

Hadamard Product

$$\begin{pmatrix}1\\2\\3\end{pmatrix} \odot \begin{pmatrix}4\\5\\6\end{pmatrix} = \begin{pmatrix}4\\10\\18\end{pmatrix}$$

Everything needs to be differentiable so each gate is calculated via a sigmoid function

$$\Gamma_t^u = \sigma\left(W_{ux} x_t + W_{uh} h_{t-1} + b_u\right)$$

$$\Gamma_t^r = \sigma\left(W_{rx} x_t + W_{rh} h_{t-1} + b_r\right)$$

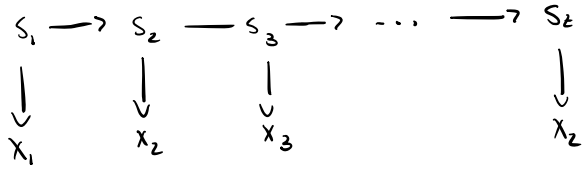
sigmoid function

Lecture 21: Sequence-to-Sequence models

Sequence models develop a conditional probability for the next state based on previous ones

Kalman Filters

gaussian version of HMM (generative model)



$$S_t \mid S_{t-1} \sim N(AS_{t-1}, \Gamma)$$

$$X_t \mid S_t \sim N(BS_t, \Sigma)$$

State is a real vector and evolves stochastically

Everything is linear

Recurrent Neural Networks

- State is deterministic
- Has an additional non-linearity

} differentiates from Kalmen Filter

Gated Recurrent Units

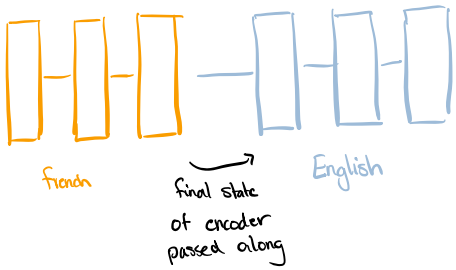
- Learn when to update hidden state to "remember" important information
- Keep information in memory until it is used
- Reset / forget information once it is no longer useful

$T = 1$: gate open ; information flows

$T = 0$: gate is closed ; information blocked

Sequence-to-Sequence Models

Uses two RNNs : Encoder and Decoder



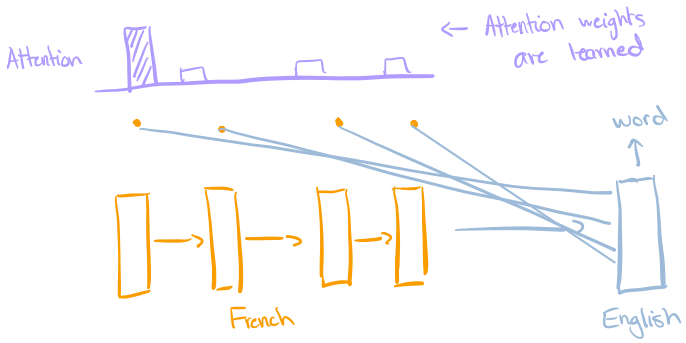
Decoder computes $\prod_{t=1}^{T} p(y_t | h, y_1, \ldots y_{t-1})$

↑ all information from French held in this state

Attention/Alignment

Each step of decoding directly connects to encoder

focuses attention on a certain segment



focuses more on the first word of the french sentence

Attention Mechanism

- m feature vectors or values $V \in \mathbb{R}^{m \times v}$
- Model dynamically chooses which to use base on similarity between query $q \in R^q$ to a set of m keys $K \in \mathbb{R}^{m \times k}$
- If q is most similar to i, then we use v_i

Kernel Regression as Attention

$$\hat{y}(x) = \frac{\sum_{i=1}^{n} K_h(x_i, x)\, y_i}{\sum_{i=1}^{n} K_h(x_i, x)} = \sum_{i=1}^{n} \alpha(x_i, x)\, y_i$$

attention weights; keys; query; values

In a gaussian Kernel

$$\alpha(x_i, x) = \frac{e^{-\frac{1}{2h^2}\|x_i - x\|^2}}{\sum_j e^{-\frac{1}{2h^2}\|x_i - x\|^2}}$$

softmax

Lecture 22: Attention and Transformers

Decoder state evolution

$$h_{t+1} = \tanh\left(W_{hh} h_t + W_{xh} x_t + W_{ah} a_t + b_h\right)$$

decoder; output; attention; bias

Attention mechanisms give a module for building new features

Algebraic representation of Attention

$$\text{Attn}\left(q, \{(k_1, v_1), \ldots (k_m, v_m)\}\right) = \sum_{i=1}^{m} w(q, k_{1:m})\, v_i$$

$$w_i(q, k_{1:m}) = \frac{\exp(a(q, k_i))}{\sum_{j=1}^{m} \exp(a(q, k_j))}$$

Kernel Regression as attention

$$\hat{m}(x) = \sum_{i=1}^{n} w(x, x_{1:n})\, y_i$$

query: test point x

keys: data $x_1 \ldots x_n$

values: responses $y_1 \ldots y_n$

If query and keys have the same dimension d

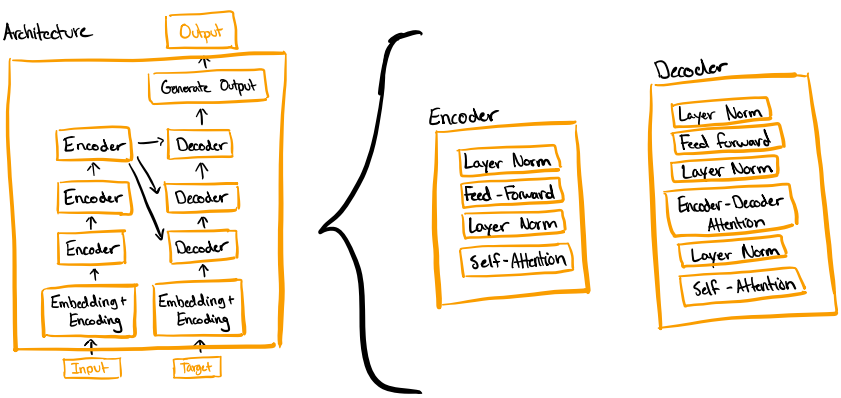
$$a(q, k) = q^T \frac{k}{\sqrt{d}} \in \mathbb{R}$$

Matrix notation

$$\text{Attn}(Q, K, V) = \underbrace{\text{Softmax}\left(QK^T/\sqrt{d}\right)}_{\text{weights}} V$$

Transformers

Transformers are RNN with no markov structure



Transformers use self-attention instead of RNN

multi-headed attention

Embedding Layers

Embed words via vocabulary indicies

Positional embedding is done via sin/cos

Neural networks don't do well with integers

For the t^{th} word in the sequence

$$PE_{(t,2i)} = \sin\left(\frac{t}{C^{2i/d}}\right)$$

$$PE_{(t,2i+1)} = \cos\left(\frac{t}{C^{2i/d}}\right)$$

d is the embedding dimension

C is a maximum sequence length

position is coded in the phase of the trig function

Concatenate embedding and positional encoding

Multi-head Attention

Residual connections: add previous input to output of a step

Encodes multiple parsings of a sentence

Encoder-Decoder Attention

Lecture 23: Transformers and AI/ML Ethics

Attention Mechanisms give a flexible method for building new features

- m feature vectors or values $V \in \mathbb{R}^{m \times v}$
- Model dynamically chooses which to use
- Decide feature vector based on similarity between query vector $q \in \mathbb{R}^q$ and set of m keys $K \in \mathbb{R}^{m \times k}$
- If q is most similar to key i, then we use value/feature V_i

Positional encoding via sin/cos functions

$$PE_{(t,2i)} = \sin\left(\frac{t}{C^{2i/d}}\right)$$

$$PE_{(t,2i+1)} = \cos\left(\frac{t}{C^{2i/d}}\right)$$

d is embedding dimension

C is maximum sequence length

transformer predicts sequentially

Residual connection brings back the input

Decoder masks future words

Attention all over the place in the input

Image vs. Caption Example

Self-attention: Image to Image Encoder

Words vs. Words Decoder

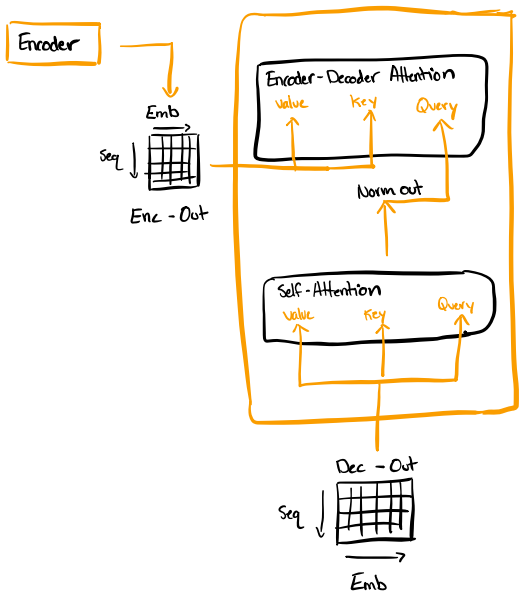
Encoder-Decoder Attention: Image to Words

Differences between encoder and decoder

Decoder has temporal masking

Decoder conditions on encodes state

Encoder-Decoder Attention



Decoder Out -> Linear —> Softmax —> Out

AI vs. ML

AI system exhibits a behavior resulting from decisions that are made

Embeddings encode societal biases

Lecture 24: Course Wrap Up