L2 - Supervised ML

October 20, 2024 1:05 AM

- mapping a given input to a target
- i.e. if given a dataset where each data instance is x_i, and the target is y, then the goal of supervised learning is to find f s.t.
  - f(x1, x2, …, xp) -> y

f	model
x1, … xp	dataset
y	target

- We build the predictive model using a dataset and an algorithm

- Learner: the statistical model that we are learning for the task
- Observation: the single data instance of the model
- Training set: the set of estimation samples used for computing the model
- Test set: the out-of-sample observation that the learner has not seen before
- Algorithm: the estimation method used to compute the model
- Features: the vector representation of a single data instance used in the algorithm

- Overfitting: models following errors (noise) too closely
  - happens when model is too flexible/complex

Prediction and inference

- two reasons to estimate a model (f)
- Prediction
  - when a set of X is available but the output Y cannot be easily obtained
  - the error term averages to zero, therefore we can predict Y with Y = g(X), where g is an estimation of the actual function f
  - reducible error
    - can be corrected by choosing the most appropriate technique to estimate f
  - irreducible error
    - variability associated with the error term in X (epsilon)
- Inference
  - understand the association between X and Y
  - goal is to estimate f but not necessarily predict y
  - f is not a black box, we want to know its exact form
  - Which predictors are associated with the response?
  - What is the relationship between the response and each predictor?
  - Can the relationship between Y and each predictor be summarized using a linear equation, or is it more complicated?
  - i.e. what tv do people purchase based on price, store location, discount, competition price

Parametric models

- a learning model that summarizes data with a set of parameters of fixed size
- line of best fit
- ex:
  - linear discriminant analysis
  - perceptron
  - naïve bayes
  - simple neural networks

- of the form f(X) = B0 + B1X1 + B2X2 … BpXp assuming f is linear in X
- makes it so that we only need to estimate the coefficients, instead of a p-dimensional function f(X)
    - can estimate the B's using OLS
- Pros:
    - speed: very fast to learn
    - less data needed: no large datasets necessary and can work well with imperfect data
- cons:
    - constrained to their very specific form
        - used only for simpler problems
    - not very accurate
        - unlikely to match the underlying mapping function
        - usually doesn't match the true form of f

Non-parametric models

- do not make a lot of assumptions about the data
- examples:
    - k-nearest neighbors
    - decision trees
    - SVM
- pros:
    - flexibility: fit many functional forms
    - power: no assumptions/weak assumptions about the underlying function
    - performance
- cons:
    - more data needed to estimate mapping function
    - slower to train, far more parameters
    - overfitting: high risk of overfitting
    - explainability: harder to explain why specific predictions are made

Accuracy vs Interpretability

- linear regression is inflexible because it can only generate linear functions
- Why use a more restrictive approach vs a flexible one?
- For inference, restrictive models are much more easily interpretable
    - easy to show relationships between X_i and Y
- Very flexible methods can lead to very complicated estimates of f
- For prediction not inference, we don't care about interpretability
    - however, sometimes we get better results using inflexible methods
        - because of overfitting

Linear regression

$$y' = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \cdots + \beta_p * x_p$$

-
$$y_1' = \beta_0 + \beta_1 * x_{11} + \beta_2 * x_{12} + \cdots + \beta_n * x_{1p}$$
$$y_2' = \beta_0 + \beta_1 * x_{21} + \beta_2 * x_{22} + \cdots + \beta_n * x_{2p}$$
$$y_n' = \beta_0 + \beta_1 * x_{n1} + \beta_2 * x_{n2} + \cdots + \beta_n * x_{np}$$

-

Bi	feature weights
xi	feature value
n	number of features

- $$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

  p =number of features
  n = number of observations

- $$Y' = h_\beta(x) = \beta . X$$

  h_β is the hypothesis function

  β is model parameters

Ordinary least squares

- find B such that it minimizes sum(d^2) where d is the distance from each datapoint to the line of best fit
- Input data $X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$ Target $y = \begin{matrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{matrix}$ Objective $MSE = \frac{1}{n}\sum_{n=1}^{n}(y_i - y_i')^2$

  Loss function (K) = Mean Squared Error
- closed form of loss function
  - Closed Form Equation

    $$\beta' = (X^T X)^{-1} X^T y$$

Other loss functions

- RMSE
  - $$\sqrt{\frac{1}{n}\sum_{n=1}^{n}(y'_i - y_i)^2}$$
- MAE
  - $$\frac{1}{n}\sum_{n=1}^{n}|y'_i - y_i|$$

L3 - gradient descent

October 20, 2024 5:43 PM

OLS

- $\text{Loss Function (K)} = \frac{1}{n}\sum_{n=1}^{n}(y_i - y_i')^2$
- Find beta such that it minimizes the loss
  1. Find the derivative of K wrt B
  2. Make it equal 0
     - $\nabla_\beta \mathrm{K}(\beta) = 0 \Rightarrow \cdots \Rightarrow \nabla_\beta \frac{1}{2}(X\beta - \mathrm{y})^T(X\beta - \mathrm{y}) = 0 \Rightarrow \cdots \Rightarrow \beta = (\mathrm{X}^T X)^{-1}\mathrm{X}^T y$
  3. Predicted value B' = $(X^TX)^{-1}X^Ty$
- What do we do if there are too many variables?

Gradient descent

- continuously check for local minimum and head in the direction of the negative gradient
- To find the minimum value of the loss funciton K
  1. obtain the gradient of K
  2. follow the gradient downhill
     i. Start with any arbitrary value of Beta
     ii. find the steepest descent = Grad(B)*K
     iii. compute the new value B_new = B - alpha*grad(B)*K
     iv. go to 2(ii) and repear
     - steepest descent converges when every element of the gradient is zero: $\nabla_B * K = 0$
     - grad(B)*K = vector of partial derivatives w.r.t. each beta

       =

$$\nabla_\beta \mathrm{f(x)} = \begin{vmatrix} \frac{\partial \mathrm{f(x)}}{\partial \beta_1} \\ \frac{\partial \mathrm{f(x)}}{\partial \beta_2} \\ \vdots \\ \frac{\partial \mathrm{f(x)}}{\partial \beta_p} \end{vmatrix}$$

- for linear regression….
  - K = $\frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - B * x_i)^2$
  - $\nabla_B * K = \nabla_B * \left(\sum_{i=1}^{n}(y_i - B * x_i)^2\right) = \frac{2}{n}(B * x - y) * x^T$

L3-II - polynomial regression

October 20, 2024 6:20 PM

Polynomial regression

- assume non-linear relationship between variables
- Our linear function: y' = B_0 + B_1*x_1 + erorr
- our non-lnear (n-degree) polynomial function

$$y' = \beta_0 + \beta_1 * x_1 + \beta_2 * {x_1}^2 + \beta_3 * {x_1}^3 + \cdots + \beta_n * {x_1}^n + \epsilon$$

- different from multiple linear
  - y' = B0+B1*x1+B2*x2+…+Bp*xp
  - powers of x1*each beta in n-degree
  - x1*one beta, x2*another, … for multiple linear

- How to learn parameteres for poly regreesion
  1. hypothesize a non linear model

$$y' = \beta_0 + \beta_1 * x_1 + \beta_2 * {x_1}^2 + \beta_3 * {x_1}^3 + \cdots + \beta_n * {x_1}^n + \epsilon$$

  2. select a loss function i.e. MSE = (1/n)*sum((yi-yi')^2)
  3. Find beta s.t. it minimizes K

$$\beta' = (X^T X)^{-1} X^T y$$

- beta can be easily estimated using OLS because hypothesis is just a standard linear model with predictors xp, xp^2, xp^3 … xp^n
  - $\beta' = (X^T X)^{-1} X^T y$ $\quad X = \begin{vmatrix} x_1 & {x_1}^2 \ldots & {x_1}^n \\ x_2 & {x_2}^2 \ldots & {x_2}^n \\ \vdots & \vdots \ \ldots & \vdots \\ x_p & {x_p}^2 \ldots & {x_p}^n \end{vmatrix}$ $\quad y = \begin{vmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{vmatrix}$

p = number of observations
n = degree of polynomial

- in polynomial functions, flexibility increases as n increases

L4 - evaluation

October 20, 2024 6:29 PM

- to evaluate a model we need to quantify how close the predicted response value is to the true response value
- for regression, this difference is called error
  - should be very small for each observation
- we care about the accuracy on the unseen instances (test set)
- If we had a large number of test observations, we would compute the test MSE

$$\text{Test } MSE = \frac{1}{n}\sum_{n=1}^{n}(y_i - y_i')^2$$

- When only training data is available: use training MSE
- When training and test data are available: use test MSE

- Overfitting: when small training MSE yields large test MSE
  - we always expect training MSE to be smaller than the test MSE
  - overfitting is when a less flexible model would have yielded a smaller test MSE
- Underfitting: when a high training MSE yields a high test MSE
  - underfitting is when a model is too simple and fails to capture relationships between the i/o
  - result of a model needing more training time, more input features, less regularization

Bias variance tradeoff

- every model has an error term
  - $y' = \beta_0 + \beta_1 * x_1 + \beta_2 * {x_1}^2 + \beta_3 * {x_1}^3 + \cdots + \beta_n * {x_1}^n + \epsilon$
- For a given observation x0,
  expected test MSE = Variance of y' + (Bias of y')^2 + Variance of ε

- variance is the amount by which y' would change if we trained it using a different data set
- bias is the amount of error in y' when applied to real-world problems
  - more flexible methods = less bias
  - i.e. training using linear regression when the realtionship between x and y is highly nonlinear = high bias
    - no matter how many training observsations, it will not be possible to produce an accurate estimate using linear regression
- variance of ε is irreducible

- **the bias-variance tradeoff**: good test set performance requires low variance as well as low squared bias
  - underfitting: simple models trained on different samples of the data do not differ much from each other
  - high variance, overfitting: complex models trained on different samples of the data are very different from eachother

Cross validation

- can be used to estimate the test error associated with a learning method to evaluate its performance
- a good model is one whose test error rate is small
- Validation sets
  - divide the data into training and validation set
  - train on the training set, compute performance on the validation set
  - cons:

        - inconsistent test MSE performance - depends on which observations are in the training set
            - bias
        - number of training set observations reduces significantly
- Leave out one (LOOCV)
    - divide into training set and one validation observation
    - train on training set, compute performance on the single observation
    - repeat with every observation
    - pros:
        - less bias
        - no randomness in split
    - cons
        - computation heavy
        - highly variable
- K folds
    - randomly divide the set of observations into k groups (folds) of approximately equal size
    - first fold is trwated as a validation set, remaining k-1 folds are used for training
    - tradeoff between loocv and splitting into one training and one validation set
        - loocv has slightly higher variance than k fold, but lower bias
    - typically we choose k = 5, 10 because they have beenb shown to suffer from neither excessively high bias nor from high variance

Error

- test error - average error that results from using a learning method to predict the response on a new observation
    - the use of a particular method is warranted if it results in low test error
- training error - average error resulting from using a learning model on the test set
-

L5 - classification and logistic regression

October 20, 2024 7:27 PM

- predicting where y is a categorical variable

Binary classification

- yes or no problems
- linear model eqn: y = B0 + sum(Bi*xi) + ε
  - right side is continuous unbounded, left side is binary
    - y is 0 or 1
  - predict the probability that a given instance belongs to 1 class of y
    - $P(y = 1|x) \propto \beta_0 + \sum_{i=1}^{p} \beta_i * x_i + \varepsilon$
      - compares 2 inputs and identifies which one leads to a higher probability of y belonging to a given class
        - we can rank these probabilities
  - β is a parameter matrix and x represents features, so we can rewrite as
    $P(y = 1|x) = b + W * X$
    - sum of the weighted features (dot product), adding them to a bias term
    - our algorithm needs to learn W and b
    - does not provide probability, still looks like a linear model
  - squash outputs to (0, 1) using the logistic function (sigmoid function)
    - $\sigma(z) = \frac{1}{1 + e^{-z}}$
    - has the property $1 - \sigma(z) = \sigma(-z)$
  - Apply logistic function to our linear model
    $P(y = 1|x) = \sigma(b + W * X)$
  - simplifies to

$$log\ (\frac{p(x)}{1 - p(x)}) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i$$

where p(x) = b + W*X
logit function = logistic regression model

- can predict y=1 if probability is high and y=0 else
- set threshold, usually 0.5

Organizing data

$$log\ (\frac{p(x)}{1 - p(x)}) = \beta_0 + \sum_{i=1}^{p} \beta_i x_i \quad \Rightarrow y = b + \boldsymbol{W}.\boldsymbol{X}$$

p = number of observations
n = number of features

-

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix} \quad X = \begin{bmatrix} x_{11} & x_{12} \dots & x_{1n} \\ x_{21} & x_{22} \dots & x_{2n} \\ \vdots & \vdots \dots & \vdots \\ x_{p1} & x_{p2} \dots & x_{pn} \end{bmatrix} \quad W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} \quad b = \begin{bmatrix} b \\ b_2 \\ \vdots \\ b_p \end{bmatrix}$$

$p * 1$ $\quad p * n$ $\quad p * 1$ $\quad p * 1$

$y = X.dot(W) + b$ To align our vectors for multiplication

Logistic regression

- how to train?
- select a cost function: the likelihood function
  - probability of y = 1 and y = 1 given p(x)
    - $P(x_i, y_i \mid \beta) = \begin{cases} p(x_i) & ,if\ y_i = 1 \\ 1 - p(x_i) & ,if\ y_i = 0 \end{cases}$
- generalize and apply log to both sides gives log-likelihood function for entire dataset

$$\ell(\beta) = \sum_{i=1}^{N} log\big(P(`x_i`,\ y_i\ \beta)\big) = \sum_{i=1}^{N} \{y_i log\big(p(x_i)\big) + (1 - y_i)\ log(\ 1 - p(x_i)\ )\}$$

- goal is to learn W and b to maximizw the log probability of correct label p(y|x) in the training data

$$log(p(y_i|x_i)) = -(y_i log\big(p(x_i)\big) + (1 - y_i)\ log(\ 1 - p(x_i)\ ))$$

How to train logistic regression?

1. Cross entropy loss
   - **ensures the probability of the correct answer is maximized and probability of incorrect answer is minimized**
   - the log likelihood loss function

$$log(p(y_i|x_i)) = y_i log(p(x_i)) + (1 - y_i) log(1 - p(x_i))$$

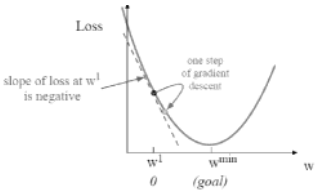
- the loss function needs to me minimized
  - use the cross-entropy loss (also called negative log likelihood)
  - sub $p(x) = \sigma(b + W * X)$

$$log(p(y_i|x_i)) = -(y_i log(\sigma(b + \mathbf{w}.\mathbf{x})) + (1 - y_i) log(1 - \sigma(b + \mathbf{w}.\mathbf{x})))$$

- i.e. if b = 10.875 and w = -0.171
  if the correct label for x = 31 is y = 1, then CE loss = $\log(\sigma(b + W * x)) = \log(\sigma(10.875 - 0.171 * 31))$
  if the correct label for x = 31 is y = 0, then CE loss = $-\log(1 - \sigma(b + W * x)) = -\log(1 - \sigma(10.875 - 0.171 * 31))$

2. We need an optimization algorithm
   - we know the sigmoid function is differentiable
   - we know logistic regression is convex
     - it has at most one minimum
     - there is no local minimum, only one global min
       - cant get stuck

ideal for Gradient Descent



   - cross entropy loss (L)

$$L = -(\ y_i * \log(\sigma(b + w * x)) + (1 - y_i) * \log(1 - \sigma(b + w * x))\ )$$

for a single observation:
gradient of CE loss

$$\frac{\partial L}{\partial w_i} = [\ \sigma(b + w * x) - y\ ] * x_i = (y' - y) * x_i$$

$$\frac{\partial L}{\partial b} = [\ \sigma(b + w * x) - y\ ] = (y' - y)$$

   - for many observations

$$L = -\frac{1}{n} \sum_{i=1}^{n} y_i * \log(\sigma(b + w * x)) + (1 - y_i) * \log(1 - \sigma(b + w * x))$$

gradient of CE loss

$$\frac{\partial L}{\partial w_i} = \frac{1}{n}[\ \underbrace{\sigma(b + w * X)}_{\text{Predicted value}} - \underbrace{y}_{\text{Actual value}}\ ] * X^T$$

$$\frac{\partial L}{\partial b} = \frac{1}{n}[\ \sigma(b + w * X) - y\ ]$$

- Stochastic gradient descent
  - online algorithm that minimizes the loss function by computing the gradient one sample at a time

1. Init b and w
2. compute esitmated value $y' = \sigma(b + w * x)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$
3. compute the CE loss $L = -\frac{1}{n} \sum_{i=1}^{n} y_i * \log(\sigma(b + w * x)) + (1 - y_i) * \log(1 - \sigma(b + w * x))$
4. Compute the gradients

$$\frac{\partial L}{\partial w_i} = \frac{1}{n}[\ \sigma(b + w * X) - y\ ] * X^T$$

$$\frac{\partial L}{\partial b} = \frac{1}{n}[\ \sigma(b + w * X) - y\ ]$$

5. update the parameters

$$w' = w - \alpha \frac{\partial L}{\partial w_i}$$

$$b' = b - \alpha \frac{\partial L}{\partial b}$$

6. predict now that we learned the weights

$$P(y = 1|x) \propto \beta_0 + \sum_{i=1}^{p} \beta_i * x_i + \varepsilon$$

$$P(y = 1|x) = \sigma(b + W * X)$$

* remember to account for error ε (slide 37)

- **Batch training**: instead of computing one instance at a time, we can compute on the entire dataset
- Mini batch training: train on a subset of the dataset
  - introduce noise to help escape local minima
  - mini batches are easily vectorized
  - very good for parallelization

L7 - Data preprocessing

October 20, 2024 9:53 PM

- Data model
  - asks: what is the data? Operations? Statistical revelations?
    - **what is the relationship?**
  - a data model can capture the complex relationships between these numbers in a very statistical sens
    - i.e. high f1 leads to higher f2
- Conceptual model
  - asks: what does the data mean?
    - **i.e. why is the data increasing/decreasing?**
  - A conceptual model catpures the constructs in data and adds meaning to attributes
    - i.e. if f1, f2, anf f3 are construct like calories, protein, fats

Types of data

- nominal (labels, categories, groups)
  - qualitative data that represents categories or labels
  - cannot be ranked
  - mutually exclusive
  - i.e. dog breeds, plant families, car manufacturers, car body styles, gender, nationality
- Ordered (ordinal, rankings)
  - qualitative or quantitative datas that represents categories or values
  - have inherent order, but there is no defined informative difference between them
  - i.e. class grade (A, B, C+, etc), tier list, likert scale (like neutral dislike)
- Interval (no true zero)
  - quantitiative data that represents measurements on a scale
  - intervals between the values are equal, but there is no absolute zero
  - i.e. time, date
- Ratio (has true zero)
  - quantitiative data that represents measurements on a scale
  - there is inherent order, equal intervals, differences are meaningful
  - i.e. distance, height, weight, income

i.e. data = 44, 54, 78, 42, 102
concept = temperature in celsius

- Nominal: water freezes, doesn't freeze
- ordinal: warm cold freezing
- interval: celsius
- ratio: kelvin (true zero)

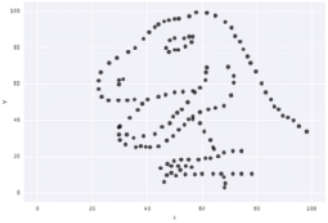
Empirical operations

- Nominal (labels, categories, groups)
  - determination of equality only
  - two nominal values can be equal, not equal, in, not in
  - =, !=
  - i.e. cant say dogs > cats
- Ordered (ordinal, rankings)
  - determination of greater or less (+ equality)
  - one ordinal value can be greater or less than another
    - two ordinal values can be equal, not equal, in, not in
  - =, !=, >, <
  - i.e. cant add cold to freezing
- Interval

    - determination of equality of intervals or differences (+ greater or less, equality)
    - two intervals can be added, subtracted
        - can be gt or lt, can be equal, not equal, in, not in
    - =, !=, >, <, +, -
    - i.e. cant multiply 1:00 by 2:00
- Ratio
    - determination of equality of ratios (+ all others)
    - can multiply two ratios, or divide one ratio by another
        - gt, lt, =, !=, +, -, x, /, %
    - i.e. can multiply $10 by $20

Conventions
- Dimensions
    - qualitative values
    - categorize, segment
    - i.e. dates, names
- Measures
    - mathematical data
    - numeric, can be aggregated
    - height, temperature, gpa, petal width

Data model vs concept model
- concept model observes how features are correlated with each other and accounts for that
- i.e. when evaluating different houses, we want to know how many rooms are in each house
    - but it would be VERY useful to also know how many of those rooms are bedroms, how many people are in each house
- in data model, we do not account for this
    - different distributions heavily affect our correlations
    - same statistics fit different data
    - i.e.



```
X Mean: 54.26
Y Mean: 47.83
X SD  : 16.76
Y SD  : 26.93
Corr. : -0.06
```

corr = 0 so they must be correlated right? No, there's literally no correlation

Preparing Data for ML
- data cleaning
    - remove missing data
        - delete entire row, or entire column
        - replace missing values with some value (the mean, zero, etc)
- categorical variables
    - i.e. converting text to numbers (assigning numerical values to nominal data)
    - i.e. converting numbers to arrays
- feature scaling
    - normalization: min-max scaling and standard scaling
- creating test sets
    - splitting the dataset
    - k fold, 80:20, stratified split

L8 - Multiclass classification

October 20, 2024 10:24 PM

Binary classification - 2 classes
- i.e.
  - rain/no rain
  - buy/sell

Multiclass classification - more than 3 classes
- i.e. animals - dog cat bird tiger
- i.e. tv program genre - politics, sports, cartoons
- i.e. positive negative neutral

Binary logistic regression
- given a set of features (x) of a data instance, compute the probability of the instance belonging to class 1
  - p(Y|x) = p(x) if Y = 1, 1-p(x) if Y = 0
- eqn form: P(Y = 1|x) = σ(b + W*x)
- training objective: learn W and b to maximize the log probability of the correct label using training datsaset
- logit/log ratio = $\log\left(\frac{p(x)}{1-p(x)}\right)$
- sigmoid function
  - if z is large sigmoid -> 1
  - if z is small sigmoid -> 0

- Binary logistic regression with multiple features
  ** one feature p(Y = 1| x) = σ(b + W*x)

  p(Y = 1|X) = σ(b + W*X) = σ(b + w_1x_{i1} + w_2x_{i2} w_3x_{i3} + ... + w_mx_{im})
  m = number of observations
  n = number of features

  - Loss
    $L = -\big(\ y_i * \log\big(\sigma(b + w * x)\big) + (1 - y_i) * \log\big(1 - \sigma(b + w * x)\big)\ \big)$

    where training goal is to minimze average loss
    $L = \frac{1}{n}\sum_{i=1}^{n} L_i$
    - why entropy loss? CE loss measures the number of bits we send

Multinomial logistic regression
Approach #1
- for each class, we predict the probability of the observation belonging to the class
  - p(Y = 0|X), p(Y=1|X), p(Y=2|X)
- finally, we assign the class as the one with the highest probability
  - class of X is 1 if p(Y=1|X) > p(Y=0|X) and p(Y=2|X)
- limitations
  - computation can be intense for large classes
  - we might not need all possible computaitons

Approach #2
- use softmax regression to compute the probability that a data point belongs to each class by using softmax function
- treats every class symetrically instead of using one class as a baseline

- estimat ecoefficients for all K classes, rather than estimaitng coeffs for K-1 classes
  - $softmax(e^x) = \frac{e^{z_i}}{\sum_{i=1}^{K} e^{z_i}}, 1 \leq i \leq K$

- the softmax function of an input vector is given by
  - $softmax(z) = \frac{e^{z_1}}{\sum_{i=1}^{K} e^{z_i}} + \frac{e^{z_2}}{\sum_{i=1}^{K} e^{z_i}} + \cdots + \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}}$
    where z is the vector of score, called the logit

- the denominator sum normalizes all values into probabilities
- like the sigmoid, the softmax squashes values between 0 and 1

- In logistic regression
  - $P(y_k = 1|x) = \frac{e^{w_k * x + b_k}}{\sum_{i=1}^{K} e^{w_k * x + b_i}}$, where k is the class

-

Example: 5 classes, 3 data instances	Positive	Negative	Neutral	Too Positive	Too Negative
Data instance 1: Professor is awesome	1	0	0	0	0
Data instance 2: Professor is horrible	0	0	0	0	1
Data instance 3: Professor is meh	0	0	1	0	0

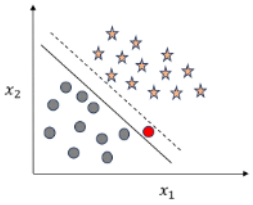
- When else is softmax used?
  - neural networks
    - final layer of NN
    - log loss/cross entropy loss
  - reinforcement learning
    - can convert action value corresponding to expected reward into probabilities

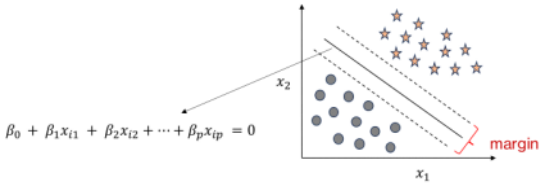
Evaluating classifiers
- True positive, true negative, false positive, false negative
- Accuracy = (TP + TN)/ (P+N)
- Sensitivity = TP/P
- Specificity = TN/N
- Precision = TP/ (TP+FP)
- Recall = TP/ (TP+FN)

L9 - SVM classification

October 21, 2024 2:06 PM

Hyperplane

- in p-space, a hyperplane is a flat affine subspace of p-1
  - in 2d, a hyperplane is a line
  - in 3d, a hyperplane is a plane
- when the hyperplane is too close to one of the classes, it fails to accurately classify any new instances
  - 

- The goal of SVM is to find the separating hyperplane that is furthest away from the training set
  - defined by the optimal **margin**
  - hyperplane -> B0 + B1xi1 + … + Bpxip = 0
- SVM is the generalization of the maximum margin classifier
  - margin is the distance between the observations and the hyperplane
  - 

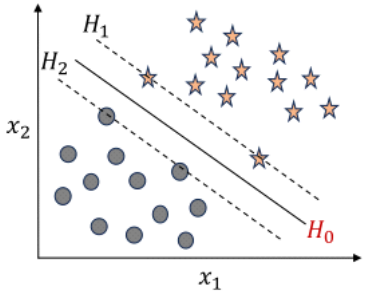
- A test observation is defined based on which side of the hyperplane it is on

  $B_0 + B_1 x_{i1} + \cdots + B_p x_{ip} > 0 \;\; if \; y_i = 1$ --> class star
  - Hyperplane here --> H1 = $B_0 + B_1 x_{i1} + \cdots + B_p x_{ip} = 1$

  $B_0 + B_1 x_{i1} + \cdots + B_p x_{ip} > 0 \;\; if \; y_i = -1$ --> class circle
  - Hyperplane here --> H2 = $B_0 + B_1 x_{i1} + \cdots + B_p x_{ip} = -1$

  Given these, we can say

  H0 = $B_0 + B_1 x_{i1} + \cdots + B_p x_{ip} = 0$



Training

- The training objective is the widest gap possible between H2 and H1
  - Widest Margin
  - margin = $d^- + d^+$
    - d^- is the shortest distance to the closest negative point
    - d^+ is the shortest distance to the closest positive point
      - these closest points are called the support vectors
- Support vectors define the margin
  - the support vectors moving have an effect on the margin

    - moving the non-support vectors has no effect
- Our optimization algorithm must learn the weights in such a way that the support vectors determine the weights and thus the boundary

Optimization

- The optimization goal of SVM is to maximize the margin using the betas

$$\underset{\beta_0, \beta_1, \ldots, \beta_p, \epsilon_1, \ldots \epsilon_n, M}{\text{maximize}} \; M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

-

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

    - B1, B2, … Bp --> the weight vector W
    - B0 --> the bias vector b

- if the weight vector is w, then $d^+ = \frac{1}{||w||}$ and $M = \frac{2}{||w||}$
- maximize M --> maximize $\frac{2}{||w||}$ using b, w
    - maximize M --> minimize w
- alternatively

$$\underset{b,w}{\text{minimize}} \left\{ C \sum_{i=1}^{n} \max[0, 1 - y_i(b + w.x_i)] + \frac{1}{2}||w||^2 \right\}$$

Loss Regularization / Penalty

- we can combine w and b into one weight matrix $W = [w_1, w_2, \ldots, w_p, b]$
    -

Whats the difference?

- Maximal margin classifier
    - want to maxmize the margin size to classify points
    - want every point to be on the right side of the margin and hyperplane
- SVC/soft margin classifier
    - allow points to be on the wrong side of the margin and sometimes hyperplane
        - in the interest of robustness and better classification of MOST training observations

SVC

$$\underset{\beta_0, \beta_1, \ldots, \beta_p, \epsilon_1, \ldots \epsilon_n, M}{\text{maximize}} \; M$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C,$$

- C is the tuning parameter (C>0)
    - bounds the sums of the ϵ_i's
    - determines the number and severity of the violations
    - "budget"
    - C = 0 ---> turns SVC into maximal margin hyperplane
    - chosen via CV
    - controls the bias-variance tradeoff
        - big C = low variance but potentially high bias
        - little C = low bias but high variance

- M is the margin width
- $\varepsilon_1 \dots \varepsilon_n$ are slack vartiables that allow individual observations to be on the wrong side of the margin or the hyperplane
    - tells us where the ith observation is located
        - $\varepsilon_i = 0$ --> right side of the margin
        - $1 > \varepsilon_i > 0$ --> wrong side of the margin but right side of the hyperplane
        - $\varepsilon_i > 1$ --> wrong side of the hyperplane

The non-linear case

- when there is a non linear relationship between the predictors and the oucome
    - rather than fitting an svc using p features, we can use 2p features and enlarge the feature space to be quadratic
        - X1, X2, … Xp --> $X_1, X_1^2, X_2, X_2^2, \dots, X_p, X_p^2$
- Our new optimization goal

$$\underset{\beta_0,\beta_{11},\beta_{12},\dots,\beta_{p1},\beta_{p2},\epsilon_1,\dots,\epsilon_n,\,M}{\text{maximize}} \quad M$$

$$\text{subject to } y_i \left(\beta_0 + \sum_{j=1}^{p} \beta_{j1} x_{ij} + \sum_{j=1}^{p} \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i),$$

$$\sum_{i=1}^{n} \epsilon_i \leq C, \;\; \epsilon_i \geq 0, \;\; \sum_{i=1}^{p} \sum_{k=1}^{2} \beta_{jk}^2 = 1.$$

- in the enlarged feature space, the resulting decision boundary is linear, but in the original feature space, it is quadratic

SVM

- an extension of the SVC
- enlarging the feature space in a specific way using **kernels**
- kernels quanitify the similarity of two obsrevations
- The linear support vector can be classified as

  $f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i)$

  where K is the kernel funciton
    - can be inner product, sum, difference, … anything

- The radial kernel
    - $K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2).$
    - if given a test observation x* = $(x_1^*, \dots, x_p^*)^T$ is far from the training observatrion x_i (in terms of euclidian distance) then sum in the eqn will be large, so K will be very small
    - so, x_i will play virtually no role in f(x*)
    - **training observations that are far from x* will play essentially no role in its predicted class label**
        - localization of behaviour

L10 - SVM II

October 21, 2024 7:49 PM

The loss function

- with SVM, our optimization goal is

$$\underset{w}{\text{minimize}} \left\{ C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w} * x_i)] + \frac{1}{2}||\boldsymbol{w}||^2 \right\}$$

Loss

Regularization/penalty

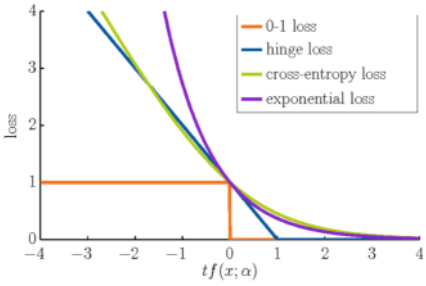
$$\ell(y) = \max(0, 1 - z * y)$$

Hinge loss function

- $\ell(y) = \max(0, 1 - z * y)$
- this is hinge loss defined for an intended output z = +- 1 and a predicted classifer score y

Gradient of hinge loss

- hinge loss is a convex function
- hinge loss is not differentiable
-



- gradient of hinge loss wrt w is given by

$$\frac{\partial \ell}{\partial w} = \begin{cases} 0 & y_i(w * x_i) \geq 1 \\ -y_i x_i & y_i(w * x_i) < 1 \end{cases}$$

- gradient of total loss $L = C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w} * x_i)] + \frac{1}{2}||\boldsymbol{w}||^2$

$$\frac{\partial L}{\partial w} = \begin{cases} w + 0 & 1 - y_i(w * x_i) \geq 0 \\ w - C y_i x_i & 1 - y_i(w * x_i) < 0 \end{cases}$$

Training SVMs

- to train SVMs we can use stochastic gradienr descent
  - computes the gradient one sample at a time
  - minimizes the loss function

1. init model parameters
   - weights (w), learning rate (η), epoch (E)
2. compute loss gradients
   - $\frac{\partial L}{\partial w} = \begin{cases} w + 0 & 1 - y_i(w * x_i) \geq 0 \\ w - C y_i x_i & 1 - y_i(w * x_i) < 0 \end{cases}$
3. compute loss
   - $L = C \sum_{i=1}^{n} \max[0, 1 - y_i(\boldsymbol{w} * x_i)] + \frac{1}{2}||\boldsymbol{w}||^2$
4. update weights

    - $w' = w - \eta \frac{\partial L}{\partial w}$
5. repeat 2-4 for E epochs

Kernel trick

- what happens if the data we are working with is not linearly separable
- we use polynomial predictor variables
    - creates large feature space
    - SVM is not ideal for this
- controlling for large feature spaces: kernel is a function that quantifies the similarity of two observations

Design considerations

- If the model is overfitting, reduce the polynomial degree
- underfitting --> increase degree
- SVM's are very sensitive to feature scale
- SVM classifiers do not output probabilities for each class
- at a low polynomial degree, SVMs cannot deal with with very complex datasets
- at a high polynomial degree, it creates a huge number of features, making the model too slow
- when C is small
    - more violations to the margin are tolerated
    - results in low variance but high bias classifier

L11 - crowdsourcing

October 21, 2024 8:29 PM

Human computation

- computation: mapping an input representation using an explicit finite set of instructions (an algorithm)
- Human computation
    - when humans actively decide when and where to collect the data
    - intelligent systems that organize humans to carry out the process of computation for performing…
        - the basic operations
        - taking charge of the control process
            - i.e. decide what operations to execute next or when to stop
        - synthesizing the program itself
            - i.e. creating new operations and specifying the order
- Non-human computation
    - when humans have no consious role in determining the outcome
    - i.e. logging location data from phone

- aspects of human computational systems
    - WHAT - decide what operations need to be performed in what order
    - HOW - decide how each operation is to be performed
    - WHO - decide to whom each operation should be assigned

Applications of human computation

- Crowdsourcing
    - outsourcing tasks to an undefined large group of people through an open call
- Collective intelligence
    - a shared group intelligence that emerges from the collaboration and competition of many individuals
- Social computing
    - technology for supporting social behaviour in or through computational systems i.e. social media
    - technology for supporting communications that are carried out out by groups of people i.e. online auctions

Wisdom of the crowds

- the collective opinion of a diverse and independent group of individuals rather than that of a single expert
- Beneficial when…
    - combinatorial problems i.e. minimum spanning trees
    - ordering problems i.e. order of the US presidents or world cities by population
    - Multi-armed bandit problem - participants choose from a set of alternatives with fixed but unknown reward rates

Human computation in machine learning applications

- generating binary or categorical labels
    - i.e. profanity detector
    - popular model for human computation is Dawid-Skene
        - assumes instances are homogenous
    - a worker's probability of labelling any given instance correctly is controlled by one or

more worker-specific quality parameters

- Generating transcriptions, translations, and image annotations
  - i.e. recaptcha
- N = number of computational tasks
- Y_n - true output each task is unknown

- goal is to estimate Y_n given an output matrix O, containing the responses from M workers

$$\begin{array}{cc} & \text{computational task} \\ & \begin{array}{cccc} 1 & 2 & \cdots & N \end{array} \\ \text{worker} \begin{array}{c} 1 \\ 2 \\ \vdots \\ M \end{array} & \begin{bmatrix} O_{11} & O_{12} & \cdots & O_{1N} \\ O_{21} & O_{22} & \cdots & O_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ O_{M1} & O_{M2} & \cdots & O_{MN} \end{bmatrix} \end{array}$$

- Evaluating and debugging models
  - the crowdsourced evaluation is especially common for unsupervised models
    - generally cannot be evaluated in terms of simple metrics like accuracy or precision
    - no objective ground truth

- What makes a good model? asking the right question!
  - influence of presented information on task performance
  - Task granularity
    - Well defined
    - Not cognitively overwhelming
  - Task should be done independently
  - Incentive should be adequate
  - Outcome must have a quality control

L12 - Uncertainty and active learning

October 21, 2024 9:02 PM

What is uncertainty
- what do probability distributions tell us?
  - i.e. for tumor assesment, probability of malignant vs probability of benign
- uncertainty: the extent to which the machine is confident in its predictions
- it is important to compute uncertainty to support reasoning stages

The concept of uncertainty
- all data instances are not equally informative
- instances closer to the decision boundary can significantly change the shape of the boundary
  - (SVM)
- all data instances can not ncessarily improve model uncertainty

Source of uncertainty
- two sources
  - consider CE loss: $L = -\big(\ y_i * \log\big(\sigma(b + w * x)\big) + (1 - y_i) * \log\big(1 - \sigma(b + w * x)\big)\ \big)$
- **Epistemic** uncertainty: when model parameters (Beta) lack sufficient knowledge
  - captured by individual predictions
- **Aleatoric** uncertainty: when observations are noise and data has sufficient knowledge
  - captured by multiple predictions

Reducing uncertainty
- **epistemic**
  - **can be reduced by**
    - **data augmentation**
    - **ensemble methods**
  - augmented by adversarial training
    - introduce minor change in input
  - encourage p(y_i|x_i; B) ot be similar to p(y_i|x_i+deltax_i; B)
  - Train N models as an ensemble with random initalization
  - combine predictive responses by averaging p(y|x) = (1/N)sum(p(y|x, B_i))
- **aleatoric**
  - **can be reduced by**
    - **reducing our assumptions about the correct model that generated the data**

Active learning
- special case of machine learning
- algorithm can interactively query an oracle (machine or human) to provide annotation on unlabelled or noisy labelled data instances
- the label is fed back into the model, and a new decision boundary is learned
- a reasonable strat is to request feedback from the oracle on an instance which might be most imformative/important
- i.e.
  a. apply the current classifier to each unlabelled instance
  b. find the b instsnces for which the classifier is least certain oif class membership
  c. have the expert label the subsample of b instances
  d. train a new classifier on all labelled isntances

Measuring uncertainty
- **Least Confidence Sampling:** Difference between 100% confidence and most confidently predicted labels
  - captures how confident the most confident prediction is

- Least Confidence Sampling = $\left(1 - P_{cyclist}\right) * \frac{k}{k-1}$
  k = number of classes

- uncertainty score for this instance = (1-0.46)*4/3 = 0.72
- uncertainty score for next instance = (1-0.68)*4/3 = 0.43

- **Margin of confidence sampling**: difference between the two most confident predicted variables
  - = (1 - Pcyclist - Ppedestrian)

  - Uncertainty score for this instance = (1 - 0.46 - 0.32) = 0.22
  - Uncertainty score for next instance = (1 - 0.68 - 0.19) = 0.13
- **Ratio of confidence sampling:** difference between the two most c
  - = ($P_{cyclist}/P_{pedestrian}$)

  - Uncertainty score for this instance = (0.46/0.32) = 1.44
  - Uncertainty score for next instance = (0.68/0.19) = 3.58

This instance

$P_{cyclist}$ = 0.46
$P_{pedestrian}$ = 0.32
P_{sign} = 0.19
P_{animal} = 0.02

Next instance

$P_{cyclist}$ = 0.68
$P_{pedestrian}$ = 0.11
P_{sign} = 0.19
P_{animal} = 0.02

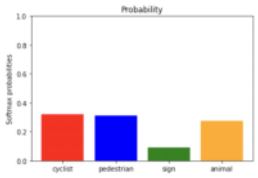
- **Entropy based sampling**: The entropy gives a measure of uncertainty about the actual structure of a system

  Entropy based Sampling $= \frac{-\sum_{i=0}^{k} P_i \log(P_i)}{\log(k)}$

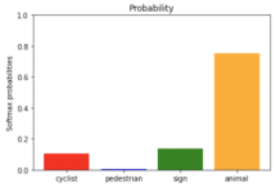
  Uncertainty score for this instance = -0.80

  - Uncertainty score for next instance = -0.64

  - High entropy = Less variability, all equally likely events
  - Low entropy = more variability, 3 unlikely events
  - where entropy is basically the difference between all the probabilities
    - i.e. if all the probabilities are 0.4, entropy is high
    - if one is 0.1 and another is 0.9 and another is 0.2, entropy is low



High Entropy



Low Entropy

M

Sampling strategies in active learning

- Uncertainty based sampling
  - least confidence sampling: captures how confident the most confident prediction is
  - Margin of confidence sampling: captures difference between two most confident predictions
  - Ratio of confidence sampling: captures proportion of 2 most confident predictions
  - Entropy based sampling: how much every confidence differs
- Diversity based sampling
  - sampling training samples that are maximally different (outliers) from the training set
  - **Model-based outlier sampling**
    - determine which instances are unknown to the model in its current state
  - **Cluster based sampling**
    - identify a diverse selection of instances using clustering algorithms
  - **Representative sampling**
    - identify samples that are like the target domain
  - **Real-world diversity**
    - identify samples that may have been missed for certain demographics

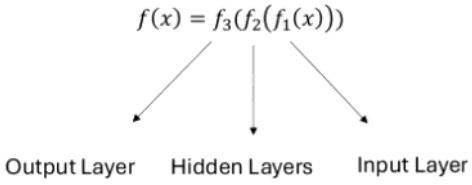
L13 - Neural networks

October 22, 2024 1:09 AM

Fundamentals of Neural networks

- NN are inspired by the brain structure seen in many organisms i.e. humans
- they form the fundamental building block of deep learning
- also called feedforward netwoeks, or multilayer perceptron (MLP)
    - neural networks with feedback = recurrent neural networks

- they take an input vector of p variables and can transform it to a target variable using a non-linear function
    - improvement on linear models

- input layer: containts the input weights
- hidden layer(s): the behaviour of these layers id determined by the learning algorithm. To determine the close approximation of f(x)
- Output layer: where the final output will be available for use
- 

- f2 is the activation function that computes the output of the hidden layer
- • Let us consider a neural network model written in linear form as:

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

K = number of activations

$h_k(X)$ instead of X means some transformation of X as opposed to X

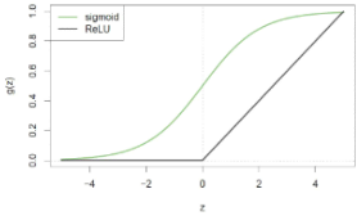
- • Then the functions of the hidden layer can be written as:

$$h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$$

where $g(z)$ = non-linear activation function

    - derives 5 new features by computing 5 different linear combinations of x, then squashes their inputs using g(z) to transform it

$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

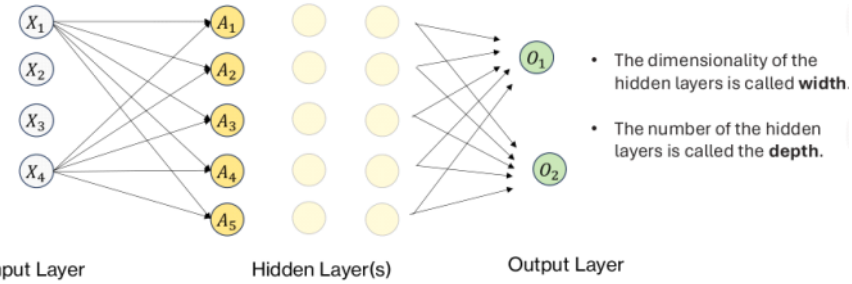
- 

- The final model is linear in these derived vartiables

- Most popular activation function is ReLu (rectified linear units)
    - squash output to be >= 0
    - 0 if z<0, no transformation if z>=0
    - used to be sigmoid

Architecture of NN

-



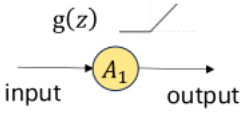
- each of these activations are units
- hidden layers are vector values
  - essentially a transformation function applied to input
- when trhere are multiple hidden layers, the first hidden layer:

$$A_k^{(1)} = h_k^{(1)}(X) = g(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j)$$

- the second hidden layer

$$A_l^{(2)} = h_l^{(2)}(X) = g(w_{l0}^{(2)} + \sum_{k=1}^{K_1} w_{lk}^{(2)} A_k^{(1)})$$

- where g(z) is the non linear actuvation function

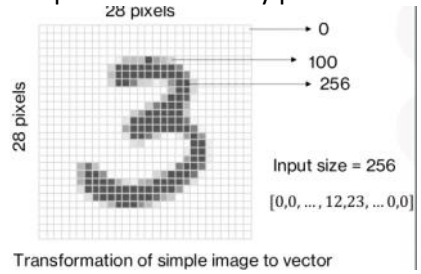


$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

- NN comes from thinking of the model as a brain
  - values of activations $A_K = h_k(X)$ that are close to one are "firing", and activations close to zero are "silent"
    - using the sigmoid activation function
- The nonlinearity of g(z) is essential
  - without it, f(X) would collapse into a linear model in $X_1 \ldots X_p$
  - allows model to capture complex nonlinearities

- wwe can envision these layers as consisting of many vector to scalar operations
- envision each layer performing a single vector to vector operation

Example: NN to identify pictures of numbers



- To design architecture
  - we need an input layer
    - size?
      - depends on the type of neural network
      - =256 units
  - we need output layers
    - size? = 10 (0-9) units
      - the choice of the cost function depends upon said cost function
        - i.e. If we use CE loss, output must be qualitative
      - Linear unit: with no nonlinearity
      - sigmoid: for predicting the value of a binary variable

        - softmax: for representing the probability distribution ove a discrete variable (n > 2 classes)
        - squared error loss
    - train (find coefficient estimates) to minimize the cost function we chose
- We need hidden layers
    - size = ? L1 = 256, L2 = 12
    - activation function = ReLu

$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

General Relu can be written as:

$$h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

        - a_i
            - when a_i = -1: absolute value rectification
            - when a_i = 0.01: leaky relu
            - when a_i is learned: parametric relu
    - the design of hidden units does not have well defined guiding theoretical principles
    - too many hidden layers for a simple problem is a bad design choice
    - experimenting with hidden layers is very common
    - using variaction of relu reuires visualization of outputs

ONEHOT ENCODING

- ONE in position corresponding to label, zeroes everywhere else
- i.e. Y = [0, 1, 0, 0, 0]

L14 - backpropagation

October 22, 2024 11:34 AM

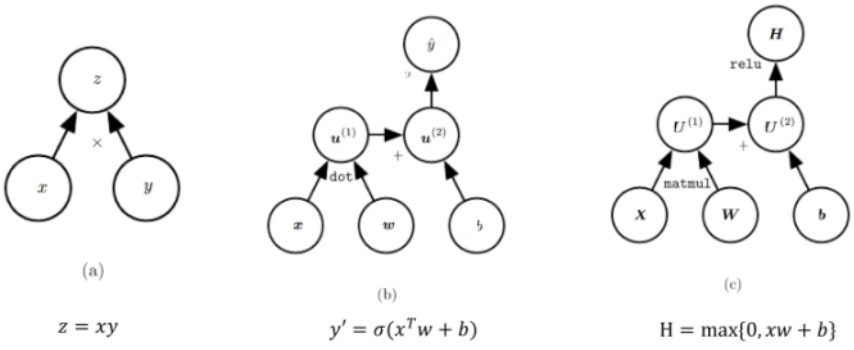
Forward propagation in NN

- pushing the inputs X through the hidden layers and to the output layer
  - calculating cost function based on this to estimate coefficients
  - $X_{hidden} = W_{XA} * X$, $X_{output} = W_{AO} * X_{hidden}$

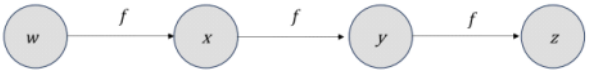
Graphing neural networks

- each node (or unit or neuron) refers to a variable
  - variables can be scalar, matrix, vector, tensor, or something else
    - tensor: array with more than two axes
      - an array of numbers arranged on a regular grid with a variable number of axes
- An operation
  - function of one or more variables
  - operation returns a single output variable
- If an operation applied to x gives y, then x and y have an **edge**

Neural Network as Computational Graph



Chain rule

- y = g(x) and z = f(g(x)) = f(y), then dz/dx = (dz/dy)*(dy/dx)
- if we want to calculate the gradient of x, we can rewrite the chain rule into a vector notation: the jacobian gradient product
  - $\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z,$
- 

$x = f(w)$
$y = f(x)$
$z = f(y)$

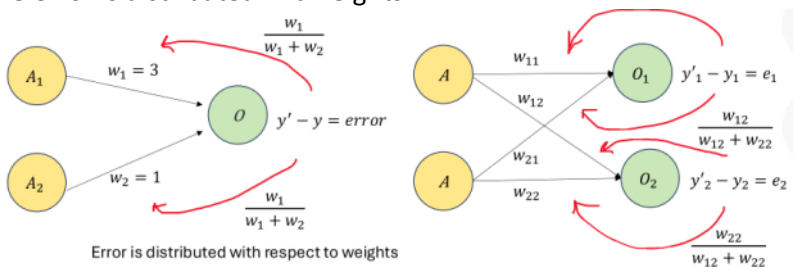
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} = f'(y)f'(x)\,f'(w) = f'\Big(f\big(f(w)\big)\Big)\,f'(f(w)))\,f'(w)$$

Backpropagation

- to compute the gradient of some scalar z wrt one of its ancestors X in the graph
  - compute the gradient wrt each parent of z
    - multiply the current gradient by the jacobian of the operation that produced z
  - continue multiplying by jacobians travelling backwards through the graph until we reach x
  - for any node that may be reached by going backwards from z through two or more

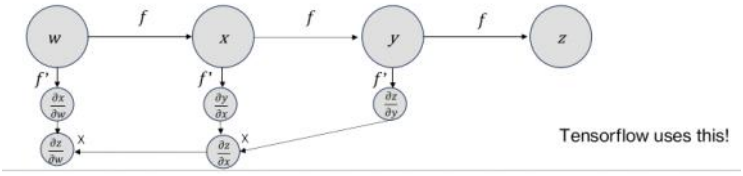
paths, we sum the gradients arriving from different paths at that node

- algorithm basis
  - set table to store and retrieve the gradients we need
    - the gradients of the targets, their descendants, as well as the gradient s of the ancestors of z
  - for each variable V whose gradient we have, scan its children C, get the operation that computes V, multiply the jacobian by the parents of V
- backpropagation takes the signal (in this case, the error) and moves it from output to input
  - the error is distributed wrt weights



- When error is nonzero, we update the weights using gradient descent
  - for any node w_{jk}, the new weights associated with it are

$$w'_{jk} = w_{jk} - \alpha \frac{\partial e}{\partial w_{jk}}$$

    - backprop computes the gradients of units (nodes) wrt to variables
    - starts at the end and rescursively applies the chain rule to compute the gradients all the way from the outputs to the inputs of the network
      - gradients flow backwards through the networks
- 

Training neural networks

- we need a learning algorithm
  - backprop
  - gradient descent
- we need a cost function
  - CE loss (qualitative)
  - mean abs error
  - mean squared error

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 .$$

- we need a training objective
  - minimize the negative multinomial log-likelihood
  - minimize error
    - error is usually nonconvex in the parameters = multiple solutions = use gradient descent to find global minimum

Gradient descent

- suppose we represent all the parameters in one long vector θ

$$R(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f_\theta(x_i)\right)^2$$

1. start with a guess θ^0 for all the parameters in θ, and set t = 0
2. Iterate until R(θ) faiuls to decrease
   i. find a vector δ that reflects a small change in θ st $R(\theta^{t+1}) < R(\theta^t)$ where $\theta^{t+1} = \theta^t + \delta$

ii. increment t

Backpropagation and GD

- how do we find he directions to move θ st we decrease R(θ)?
- the gradient of R evaluated at some value θ=θ^m

$$\nabla R(\theta^m) = \frac{\partial R(\theta)}{\partial \theta}\Big|_{\theta=\theta^m}.$$

  - evaluating at θ^m gives direction in θ-space which R increases most rapidly
    - we want to move in the opposite direction
  - $\theta^{m+1} = \theta^m - \rho \nabla R(\theta^m)$
    where p is the learning rate
- for a small enough learning rate, this step will decrease R
- if the gradient vectoe is zero, we may have arrive at a minimum of the objective
- Chain rule allows this computation to be simple
- **Backpropagation: The act of differentiation (with chain rule) assigns a fraction of the residual ($y_i - f_\theta(x_i)$) to each of the parameters**

L15 - Evaluation

October 22, 2024 12:27 PM

- review
  - underfittig: when a model is too simple
    - needs more training time, more features, less regularization
  - we always expect the training MSE to be smaller than the test MSE
  - overfitting: when less flexible models would have yielded a smaller test MSE

Evaluation goals

- find the model that performs best on the test set
  - should have high variance
  - should have low bias
  - should have high predictive accuracy
  - should be interpretable, only relevant features should matter
- Since complex models may overfit, we can use several strategies to evaluate them
  - train all possible models then select the best one
    - best = smallest RSS/largest R^2
      - select the best of the best from each strategy using CV
  - **subset selection**
    - identifying a subset of features that contribute directly to predictions
    - when p predictors, train 2^p modesl
      - that's a lot of models
      - there are more efficient alternatives…
        - @p = 20, need to fir 1048576 models
  - **forward stepwise selection**
    - begin with a model containing no predictors
    - add predictors to the model one at a time until all predictors are in the model
    - at each step, the variable that gives the greatest additional improvement to the fir is added to the model
    - when p predictors, train $1 + \frac{p(p+1)}{2}$ models
      - @p = 20, need to fir 211 models
  - **backward stepwise selection**
    - begin with full model containing all predictors, iteratively remove the least useful ones
    - when p predictors, train $1 + \frac{p(p+1)}{2}$ models
    - cannot be used for very large p (p > n)
- All these approaches yield multiple models. To select the best model…
  - indirectly estimate the test errror by making adjust ments to the training error
  - directly estimate the test error by using a validation set or CV approach

Regularization

- goal in ML: minimize test error, at the expense of increased training error
- regularization: any modification we make to the learning algo that is intended to reduce generalization error but not training error
  - i.e. adding restrictions on parameter values

$$J(\beta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_\beta(x_i)) + \lambda R(\beta)$$

- L = Loss Function
- R = Regularization Function or Parameter norm penalty
- λ = Strength of R

- regularization of an estimator works by trading increased bias for reduced variance
- best fitting model is a large model that has been regularized appropritately

- reg. has effect on all aspects of training

L2 regularization
- used for linear models and NN
- $J(\beta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_\beta(x_i)) + \frac{\lambda}{2}.||\beta||_2^2$ where, $||\beta||_2^2 = \sum_{j=1}^{d} \beta_j^2$

- for NN, we only impose penalty on the weights, not the biases
- the regularizer penalizes large parameters
    - prevents model from overrelying on a single features
- penalizes widely irregular solutions

- How do we choose λ?
    - choose the value thart results in the best performance on a held-out validation set

L1 regularization
- Used for NN and linear models
- very similar to L2 except last term

$$J(\beta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_\beta(x_i)) + \frac{\lambda}{2}||\beta||_1 \quad \text{where,} \quad ||\beta||_1 = \sum_{j=1}^{d} |\beta_j|$$

- Also penalizes large weights
- forces more weights to decay to zero

L1 vs L2
- sparsity of data
- L2 loss penalizes larger errors more heavily than L1 loss.
- L1 Loss minimizes the Absolute Error, while L2 Loss minimizes Squared Error.

L16 - Fairness

October 22, 2024 12:55 PM

Bias
- disproportionate weight in favour or against an idea or thing
  - usually in a way that is inaccurate or unfair
- bias is a systematic error
  - the difference between a measured value and its unknown true value
- statistical bias
  - systamatic tendency in which methods used to gather data or generate statistics present an inaccurate, skewed, or biased depiction of reality

Types of biases
- cognitive bias
  - repeating or basic misstep in thinking, assessing, recollecting, or another cognitive process
- confirmation bias
  - tendency to search for, interpret, favour, recall info in a way that confirms one's existing beliefs or hypothesis
  - gives less attention to info that contradicts it
- automation bias
  - human tendency to favour suggestions from automated decision making systems
  - ignore contradictory informaiton made without automation, even if it is correct

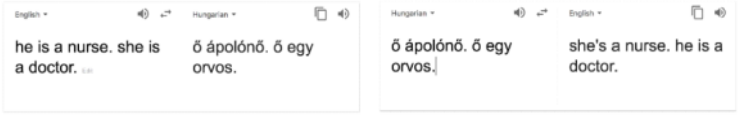
algorithmic bias
- human bias is applied far less systematically than machine bias
- bias can cascade with ML
- hman decisions, even when biased, are often tempered within reasonable bounds
  - machines can sprial out of control
- explaining bias in black box algos is feasible
  - vs asking a person to explain how they arrived at a biased decision is much harder

causes of algo bias
1. poorly designed and deployed data annotation
   - inconsistent, human generated, labels
   - i.e. decisions driven by stereotyping eventually accumulate in datasets
   - Toxic comments are disrespectful, abusive, or unreasonable online comments that usually make other users leave a discussion.

Sentence	Toxicity	Sentiment
I hate Justin Timberlake.	0.90	-0.30
I hate Katy Perry.	0.80	-0.10
I hate Taylor Swift.	0.74	-0.40
I hate Rihanna.	0.69	-0.60

Table 1: Sensitivity of NLP models to named entities in text. Toxicity score range: 0 to 1; Sentiment score range: -1 to +1.

     - bias: the same comments against the women get a lower toxicity score

2. training objective function
   - if the loss function is overly focused on outliers, which tend to be more from one group than another
     - results in models that treat each group differently
   - 

3. feature bias or curation bias
    - occurs when the modeler is biased towards choosing some features over others
    - leads to a disfavouring

4. Machines are used to make predictions
    - if these are biased, then they feedback into the training of future models
    - perpetuating the same kind of biased labels

5. random bias may occur if an algo does not have guard rails or constraints

Measuring algo biases

- **Accuracy Comparison**: Check if the model performs equally well for different groups (e.g., similar accuracy rates).
- **False Positives/Negatives**: Look at how often the model makes incorrect predictions for each group (e.g., higher false positive rate for one group may indicate bias).
- **Demographic Parity**: Measure whether each group has a similar chance of being predicted positively (e.g., equal job selection rates across genders).
- **Equalized Odds**: Ensure that the model's accuracy is similar for each group, both for true positive and true negative rates.

L17 - measuring bias

October 22, 2024 1:09 PM

How do we improve fairness

- Let us consider a model that predicts whether a loan gets approved or not.
- Bias arises when one group is favored over another.

$Y' = 1$ if the loan is approved positive outcome $P(Y' = 1|Y = 1) > \tau$

negative outcome $P(Y' = 0|Y = 0) > \tau$ where τ probability threshold

-

Recall = 0.60 Precision = 0.92

Groups B only		True Labels	
		0	1
Predicted Labels	0	70	40
	1	5	60

Recall = 0.88 Precision = 0.80

Groups A only		True Labels	
		0	1
Predicted Labels	0	90	10
	1	20	80

- observe confusion matrix to make determinations
- recall (TP/TP+FN)
  - how good an algo is at idenitfying true positives
  - if there's not many true positives and a lot of false negatives, recall will be small, therefore bad at identifying true positives
    - bias against
- precision
- accuracy (TP/TP+FP)
  - high accuracy means potentially biased towards this group

Dataset fairness

- i.e. number of members in group A significantly exceeds group B
  - group imbalance (GI)
- Class imbalance (CI): if the ratio of approved to declined applications differes significantly across groups
  - leads to biased results
  - $CI = \frac{\#(Y=1|A)}{\#(Y=0|A)} - \frac{\#(Y=1|B)}{\#(Y=0|B)} \leq \beta$
- Distributional imbalance (DI): generalization of class imbalance to multiple classes
  - $DI = \sum_y f_A(y) \log_2\left(\frac{f_A(y)}{f_B(y)}\right) \leq \beta$ where, $f_A(y) = \frac{\#(Y=y|g)}{n_g}$

Classifer fairness

- Disparate impact (Dimp): when decisions made about the positive or negative outcomes lead to unintentional discrimination
  - $DI = \frac{Pr(Y'=1|B)}{Pr('Y=1|A)} \geq 1 - \beta$
  - presented as a ratio
  - Demographic parity (DP): Disparate impact presented as a difference instead of a ratio
- Equal opportunity (EOpp): metrix to assess whether a model is predicting outcomesequally well for all groups wrt both the positive and negative class
  - TPR difference : $| Pr(Y'=1|B,Y=1) - Pr(Y'=1|A,Y=1) | \leq \beta$
    - True positive rate
- Equalized odds (EOdds): a classifer statisfies this definition if the subjects in the protected and unprotected groups have equal TPR and equal FPR
  - $FPR\ difference = | Pr(Y'=1|B,Y=0) - Pr(Y'=1|A,Y=0) | \leq \beta$

- Accuracy difference (AD): difference in accuracies of individual groups
    - TP/(TP+FP)
- Treatment equality (TE): assesses whether kinds of errors made by the algo are similar across groups
    - are the ratios of fn to fp the same
        - $\frac{\#(Y'=0|A,Y=1)}{\#(Y'=1|A,Y=0)} - \frac{\#(Y'=0|B,Y=1)}{\#(Y'=1|B,Y=0)} \leq \beta$
        - similar to CI
- Predictive parity (PPP, NPP): the diff in the ratios of fn to fp
- Individual fairness: similar individuals should receive similar outcomes
- counterfactual fairnes: two examples that are identical in all respects, expects fiven a sensitive attr, should result in the same model prediction

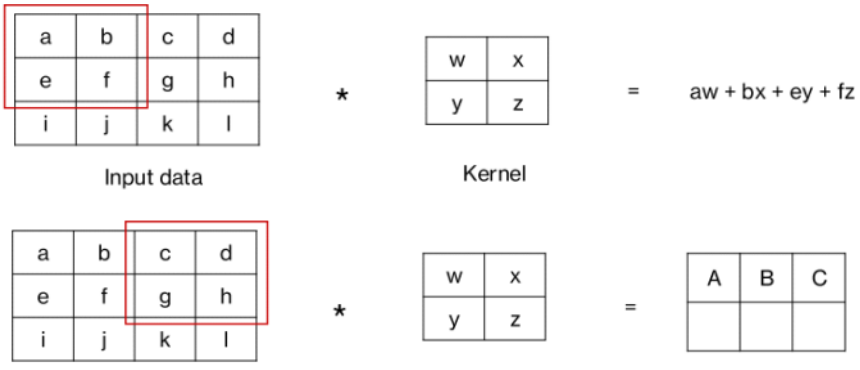
L18 - Convolutional NN

November 14, 2024 2:53 PM

- NN that use convolutional (folded) operations in place of general matrix multiplication in at least one of the layers
- Convolution is an operation on two functions of a real valued argument
  - s(t) = (x * w)(t)
  - where s(t) is the feature map, x is an input function, w is a kernel or filter
    - i.e. for images, the rgb value of each pixel is the feature map (3d array)
      - first 2 axes are spacial (32x32 pixels = 32x32 matrix), 3rd axis is the channel axis representing the 3 colours
- Convolutional NN
  - special family of CNNs has evolved for classifying images
  - CNNs mimic how humans classify images
    - recognize specific features and patterns in the image that distinguish classes

- Discrete convolution
  s(t) = $\sum_{a=-\infty}^{\infty} x(a) * w(t-a)$

Convolutional matrix multiplication:



$$A = aw + bx + ey + fz$$

$$B = bw + cx + fy + gz$$

$$C = cw + dx + gy + hz$$

Advantages of convolutional operation
- uses matrix multiplication

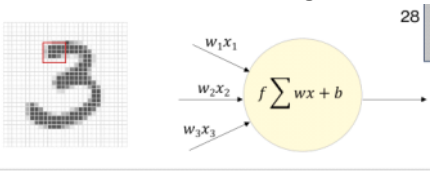
Architecture of NN
- different from normal NN
- the dimensionality of hidden layers is called width
- the number of the hidden layers is called depth
- depth != depth of the network

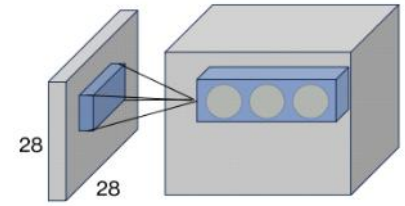
Layering - how CNNs work
1. first identifies low level features in the input image, such as small edges, patches of color, etc
2. LL features are then combined to form high level features, such as ears, eyes, etc
3. Eventually, the presence or absence of of these HL features contribute to the probability of the output class

How does it do that?

- Layers
- Uses two specialized types of hidden layers
    - convolution layer
        - searches for instances of small patterns in the image
    - pooling layers
        - downsample these to select a prominent subset (and forma a HL feature)
- CNNs have many convolution and pooling layers

Types of layers

- Convolutional Layer
    - filters do not have the same weights
    - filters look at the same region
    - 

    - 

    - $wx + b$ For 3 filters the size of convolutional layer = 28x28x3
- Pooling layer
    - down samples the input along the height and width
    - also called the detector
    - A pooling unit
        - pools over multiple features that are learned with separate parameters can learn to be invariant to transformations
- Fully connected layer
    - Computes class scores and is like just another neural network
    - Output of fully connected layer is 1x1xN

CONVOLUTIONAL LAYERS

- made up of a large number of convolution filters
    - each determines whether a particular local feature is present in an image
    - operate onl localizes patches in the input image
    - relies on a very simple operation called a convolution
        - repeatedly multiplying matrix elements then adding results
- $$\text{Original Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix}.$$

  Now consider a 2×2 filter of the form

  $$\text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

  When we *convolve* the image with the filter, we get the result[7]

  $$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}.$$

- if a submatrix of the original image resembles the convolution filter, then it will have a large value in the convolved image
- the convolved image highlights regions of the original image that hresemble the convolution filter
- In a convolution layer, we use a bank of filters to pick out a variety of edges and shapes in the image
- With CNNs, the filters are LEARNED for the specific classification task
    - think of filter weights as parameters going fron an input layer to a hidden layer, with one hidden unit for each picel in the convolved image
- A colour image will have nxnx3, filter will me mxmx3
    - results of the 3 convolutions are summed to form a 2d feature map (black and white)
- If we use K convolution filters in the first hidden layers, we get K 2d output feature maps, which we treat as a single 3d feature map
    - view each of the K output feature maps as a separate channel of informaiton
    - just like the activations in a hidden layer of a NN
- We apply ReLU activation funciton to the convolved image
    - sometimes viewed as a separate layer (detector layer)

Pooling Layers
- condenses a large image into a smaller summary image
- max pooling summarizes each non overlapping 2x2 block of pixels in an image using the maximum value in the block
- provides location invariance: as long as there is a large value in 1/4 pixels, whole block has a large value

Architecture of CNN
- number of convolution filters in a layer is akin to number of units at a particular hidden layer
    - also defines number of channels in 3d feature map
- at the input layer, we see 3d feature map of a color image where the channel axis represents each color by a 32x32 matrix
- each convolution filter produces a new channel at the first hidden layer
- after rd 1 of convolutions, we now have a new image
    - feature map with considerably more channels than 3 color channels
- Now, we do a max-pool layer, which reduces the size of the feature map in each channel by a factor of 4
- Repeart convolve-pool sequence for next two layers
    - Each subsequent convolve layer is similar to the first. it takes a 3d feature map from prev layer and treats it like a single multichannel image
        - each conv filter learned has as many channels as the feature map
    - Since the channel feature maps are reduced in size after each pool layer, we usually increase the number of filters in the next convolve lyer to compensaet
    - Sometimes we convolve several times before pooling
        - increases the dimension of the filter
- Repeat until pooling reduces each channel to ust a few pixels in each dimension
- Then, flatten the feature maps
    - feed into one or more fully connected layer before eaching output layer
        - output layer is softmax activation for the classes

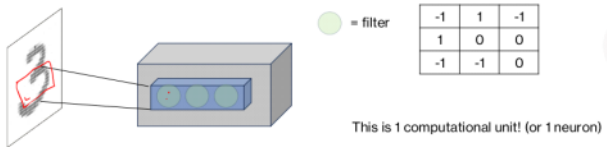
Data augmentation
- training image is replicated many times, with each replicate randomly distorted in a natural way
    - human recognition is unaffected
- 

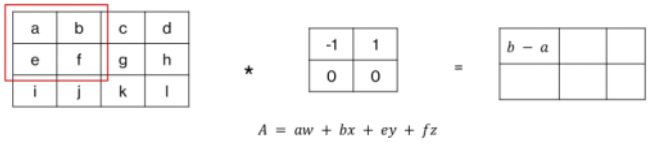
- Zoom, horiz/vert shift, shear, small rotation, flips
- increases training set considerably, protects from overfitting
- is a form of regularization
    - built a cloud of images around the original, all with the same label

L19 - CNN ++

November 16, 2024 3:40 PM

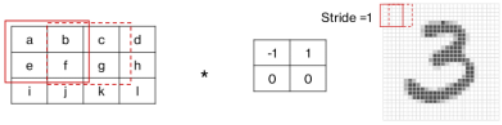
TLDR

- Typically, NN have 4 main types of layers
  - conv layer
  - pooling layer
  - fully connected layer
  - relu layer (element wise)
- Convolutional layer architecture
  - filters of smaller width and height than the full image
  - number of filters extends along depth
  - Filters are stacked behind eachother, they look for different features of the same region
  - 

- For forward pass, convolve the filter with the input volume (compute the dot product)
  - 

$$A = aw + bx + ey + fz$$

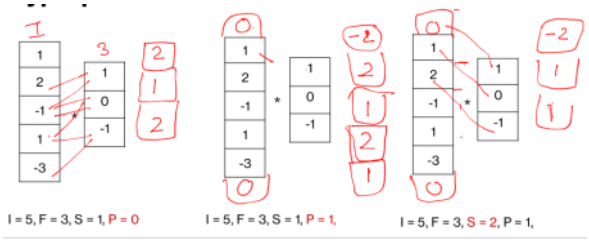
Hyperparameters

- depth (D): number of filters used in a convolutional layer
- stride (S): the number of pixels by which we slide the filter for convolution
  - 

- Zero Padding (P): size of 0 padding added on all 4 sides
  - used to keep the input and output width and height the same
- Filter size (F): spatial extent of each filter looking at a specific region
- Input volume (I)

- The number of computational units that fit into a layer

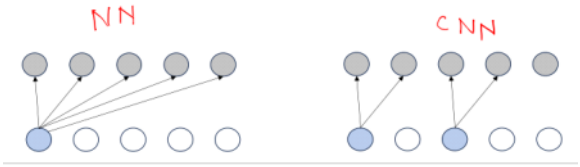
$$= \frac{I - F + 2P}{S} + 1$$

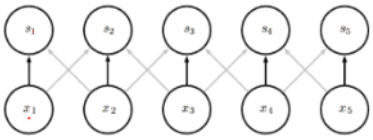
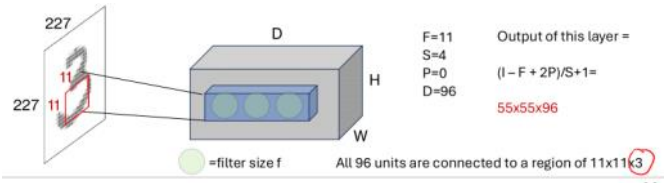
- 

- Some configurations are impossible
- i.e. I = 10, P = 0, F = 3, number of computational units is not feasible

Properties of Convolutional neural networks

- sparse interactions
  - every output unit does not interact with every input unit
  - filters focus on small, meaninful features (i.e. edges) rather than full image

- 

  - reduce memory requirements
  - requires fewer computations
- Parameter sharing
  - in traditional NN, each element of the weight matrix is used exactl once
    - multiplied by one element and never visited
  - CNNS use the same parameter for more than one function in a model
    - tied weights
    - 
    - 

      - each computaitonal unit has 11x11x2 weights, and 1 bias
      - without parameter sharing:
        output = 55x55x96x(11x11x3 + 1)
      - with parameter sharing:
        output = 96x(11x11x3 + 1)
    - If one feature is useful to compute at some position, then it should also be useful to compute at a different but related position
    - we constrain the computational units in each depth slice to use the same weights and bias
  - In practice, during backpropagation, every unit computes the gradient for its weights
  - These gradients will be added up across each depth slice
  - the gradients only update a single set of weights per slice
  - only because of this technique, we can apply convolution during fwd pass

- Equivarient representations
  - if the input changes, the output changes in the same way
  - parameter sharing causes equivarience to translation
  - if *g* translates/shifts the input, then the convolution function is equivarient to *g*
    - f(g(x)) = g(f(x))
  - Assume an image I(x, y)
    - Transformation to I + convolution = convolution to I' + transformation

L20 - Generative Models

November 16, 2024 4:40 PM

So far, we have modeled our problems as a probablilty of belonging to 2 classes, such as

$\mathrm{P}(Y = k|\, X = x)$

An alternative approach: model each predictor X separately and use Bayes Theorem to flip them

$f_k(X) \equiv \mathrm{P}(X|Y = k)$ (probablility of X s.t. Y = k)

- f_k is the density funciton of X for an observation that comes from the kth class
- Reverse engineering

- Bayes theorem: $P(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$
  - where π_k is the prior probability that a randomly chosen observation comes from the kth class
  - *** π_k and f_k are estimated

Generative model vs discriminative model

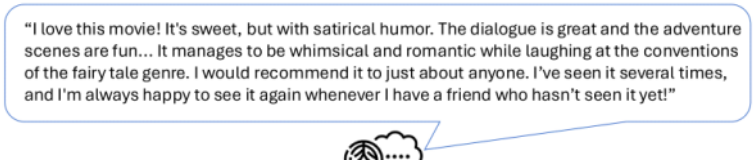
- Generative modesl model the problem as $P(x, y) = X, Y \rightarrow [0, 1]$
- Discriminative models model the problem as $P(y|x) = X, Y \rightarrow [0, 1]$

Generative models are most useful when

- differences between classes are too huge to quantify
- if the distribution of the predictors is approx normal in each of the classes
- sample size is small

Converting text to vectors

I.e. Text classification



- Technique: bag of words

word	frequency
It	6
I	5
the	4
satirical	1
whimsical	1
would	1
adventure	1
and	3

- Technique: TF - IDF

word	position
It	6
I	1
the	4
satirical	9

whimsical	1
would	1
adventure	1
and	3

Naïve bayes

- bayes is a probabilistic classifier
- for an input document d, out of all classes, the classifier returns the class c' which has the maximum posterior probability given d
  c' = argmax P(c|d)

- Use bayesian classification to transform into other probabilities
- $$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$
- $$c' = \underset{c \in C}{argmax}\, P(c|d) = \underset{c \in C}{argmax}\, \frac{P(d|c)\,P(c)}{P(d)} = \underset{c \in C}{argmax}\, \overbrace{P(d|c)}^{\text{Likelihood}} \overbrace{P(c)}^{\text{Prior}}$$
- $$c' = \underset{c \in C}{argmax}\, P(c|d) = \underset{c \in C}{argmax}\, \overbrace{P(d|c)}^{\text{Likelihood}} \overbrace{P(c)}^{\text{Prior}}$$
- $$= \underset{c \in C}{argmax}\, P(f_1, f_2, f_3 \ldots . f_n|c)P(c)$$
  $$C = argmax \prod_{f \in F} P(c)\,P(f|c)$$
- $$C = argmax \log(P(c) + \sum_{i \in word\ positions} P(f_i|c)$$

Training on Naïve Bayes

- the goal is to learn probabilities
- $$C = argmax \log(P(c) + \sum_{i \in word\ positions} P(f_i|c)$$

- P(c) - what percentage of the documents in our training set are in each class C
  - number of d in class c/number of documents (d)
- $P(f_i|c)$ - what fraction of times the word f_i appears among all words in all documents of class c
  - (count (f_i, c) + 1) / sum(count(f, c)) + |V| for the entire vocabulary (all words)

Document classification

November 16, 2024 5:09 PM

Example IMDB ratings: response is the sentiment of the review (positive or negative)

- Find a way to featurize each document (movie review)
    - define a set of predictors

Bag of words model

- Summarizes a document by the words present and ignores their context
- score each document for presence/absence of each of the words in a dictionary
- if the dict contains M words, then for each document we create a binary feature vector of length M and score a 1 for every word present
- ** limit dict i.e. to 10000 most frequently used words in the training set
    - omit unknown words

- Build training set and test set balanced with regard to sentiment
    - resulting training feature matrix X has dimensions # of reviews in the set x # of words in the dict
    - only 1.3% of entries are non-zero -- sparse X

How to account for context

- bag of n grams model
    - i.e. a bag of 2 grams records the consceutive co-occurrence of every distinct pair of words
        - blissfully long can be seen as a positive review
        - blissfully short can be seen as negative
    - treat the document as a sequence, taking account all the words in the context of those that preceded and those that follow

Naïve bayes

November 17, 2024 3:18 PM

Assume f_k is the density function for a multivariate normal random variable with class specific mean μ_k, and shared covariance matrix Σ

- These very strong assumptions alow us to simplify the problem of estimating K p-dimensional density functions (f_k's) into estimating K p-dimensional mean vectors and ONE pxp-dimensional covariance matrix

The Naïve bayes assumption

The Naïve bayes classifier takes a different tack for estimating $f_1(x) \dots f_k(x)$

- Instead of assuming these functions belong to a particular family of distributions (i.e. multivariate normal)
- Make one assumption: Within the k^{th} class, the p predictors are independent
  - for k = 1… K
    $f_k(x) = f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times f_{kp}(x_p)$
    where f_{kj} is the density function of the j^{th} predictor among observations in the k^{th} class

- By assuming that the p covariates are independent within each lass, we odnt have to worry about the association between the p predictors
  - we assume there is no association between them
- Useful in settings where n is not large enough relative to p for us to effectively estimate the joint distribution of predictors within each class
- Introduces some bias, reduces variance

Posterior probability

$$\Pr(Y = k|X = x) = \frac{\pi_k \times f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times f_{kp}(x_p)}{\sum_{l=1}^{K} \pi_l \times f_{l1}(x_1) \times f_{l2}(x_2) \times \cdots \times f_{lp}(x_p)}$$

for $k = 1, \dots, K$.

- to esitmate the 1d density funciotn f_kj using training data x_1j, … , x_nj, we can
  - If X_j is quantitative, assume $X_j|Y=k$ ~
    $N(\mu_{jk}, \sigma_{jk}^2)$
    - Assume that within each class, each predictor is drawn from a normal dist
    - assume predictors are independent
  - If X_j is quantitative
    - non-parametric estimate for f_{kj}
    - estimate $f_{kj}(x_j)$ as the fraction of the training observations in the k^{th} class that belong to the same histogram bin as x_j
      - kernel density estimator
  - If X_j is qualitative
    - count the proportion of training observations for the jth predictor corresponding to each class
    - i.e. if we have 100 observations of the kth class, and the jth predictor takes on values of 1, 2, and 3 in 32, 55, and 13 of those observations, then we can estimate

$$f_{kj}(x_j) = \begin{cases} 0.32 & if\ x_j = 1 \\ 0.55 & if\ x_j = 2 \\ 0.13 & if\ x_j = 3 \end{cases}$$

- Any classifier with a linear decision boundary is a special case of naïve bayes
  - LDA is a special case of naïve bayes
  - LSA assumes features are normally distributed with a common within-class convariance matrix
  - naïve bayes assumes independence of features
- KNN is better when the decision boundary is highly non linear
  - need a lot of observations relative to number of predictiors
    - n >> p
  - when decision bounary is non linear but n is modest or p is not very cmall, QDA may be better than KNN
  - KNN does not tell us which predictors are important

L21 - Clustering

November 17, 2024 2:50 PM

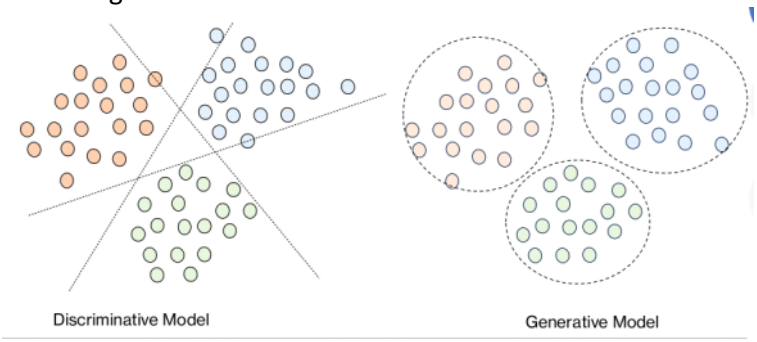
Text to vectors: Term Frequency - Inverse Document Frequency (TF - IDF)

- evaluate words wrt document (not position within a sentence)
- relative frequency of occurrence in a document
- can also account for number of documents in which each term can be found

Generative vs discriminative models

- Generative
  - Can generate new data instances
  - $P(x, y) = X, Y \rightarrow [0, 1]$
  - Capture the joing probability P(x, y)
  - Can work without labels, so can compute P(x)
- Discriminative
  - Discriminate between different kinds of data instances
  - $P(y|x) = X, Y \rightarrow [0, 1]$
  - Capture conditional probability P(y given x)
  - Needs labels to work

Clustering



- We may cluster the data into subgroups and partition the entire data distribution based on similarity
- The goal of clustering is to find homogenous subgroups among observations
- When a new data instance arrives, we find the subgroup it may belong to or is closest to
- Relies on data distribution to build a classification model
- They are challenging, tackle difficult tasks

Types of clustering

- K-means
  - Let C1, C2, … Ck be sets containing observations in each cluster
  - These sets must satisfy…
    - Each observaiton must belong to at least one of the k clusters
    - The clusters are non overlapping, and no observation belongs to more than one cluster
  - The variation of a cluster is $W(C_k)$
  - Good clustering is one for which the within-cluster variation is minimal
    - Optimize:

      $minimze\ \{ \sum_{k=1}^{K} W(C_k) \}$
    - $W(C_k) = \frac{1}{|C_k|} \sum_{i,i` \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i`j})^2$
      - Where $|C_k|$ is the number of observations in the k^{th} cluster
  - Defining similarity

- Feature similarity using Euclidian distance
  - distance between 2 pts d(p, q) = |p, q|
  - In high dimensionality $d(p, q) = \sqrt{(p - q)^2}$

- Training k means
  1. Randomly assign a number, from 1 to K, to each observation. Serve as initial cluster assingments
  2. Iterate until the cluster assignments stop changing
     a) for each of the K clusters, find the cluster cnetroid
        - the k^{th} clister centroid is the vector of the p feature means for the observations in the k^{th} cluster

          $\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$
     b) Assign each observation to the cluster whose centroid is the closest
        - (Euclidian distance is smallest)

K means clustering

November 17, 2024 3:56 PM

- For partitioning the dataset into K distinct, non-overlapping clusters
- let c1 … ck denote sets containing the indices of the observations in each cluster
  1. C1 U C2 U … U Ck = {1, … n}
     Each observation belongs to at least one of the K clusters

  2. Ck ∩ Ck' = 0 for all k != k'
     The clusters are non-overlapping: no observation belongs to more than one cluster

- A good cluster = minimial within-cluster variation
  - amount by which observations within a cluster differ from eachother
  - Partition observations into K clusters s.t. sum of within-cluster variations across K clusters is minimal
    - calculate variation using euclidian distance
  - $$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}$$

- K means clustering algorithm
  1. Randomly assign a number from 1 to K, to each observation
     - Initial cluster assignments
  2. Iterate until the cluster assignments stop changing
     a. For each cluster, compute the centroid
        - centroid = vector of the p feature means for the observations in the kth cluster
     b. Assign each observation to the cluster whose centroid is closest
  - $$\frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 = 2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2,$$

    - $$\text{where } \bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$$

  - As the algorithm is run, reallocating the observations can only improve the clustering
    - the clustering obtained will be continuously improved until the result no longer changes
    - finds a local min and stays there
      - because it does not necessarily find a global min, depends on the initial cluster assignment
    - run the algo multiple times with different init configs to select the best solution

L22 - Clustering and dimensionality reduction

November 17, 2024 4:38 PM

Types of clustering
- K-means
- Hierarchical
  - Uses tree based representation of observations called Dendrograms
  - The lower in the tree, fusions occur, the more similar the groups of observations are to eachother
  - Most common type is bottom-up (aggloerative)
  - For n observations, 2^{n-1} reordering of dendrograms

Hierarchical clustering algorithm
1. Begin with n observations and a measure (i.e. Euclidian distance) of all the $\binom{n}{2} = \frac{n(n-1)}{2}$ pairwise dissimilarities
   a. treat each observation as its own cluster
2. For i = n, n-1, …. 2:
   a. Examine all pairwise intercluster dissimilarities among the i clusters
      i. identify the pair that are the most similar (least dissimilar)
      ii. fuse the two similar clusters
      iii. the dissimilarity between those two clusters indicates the height in the dendrogram at which the fusion should be placed
   b. compute the new pairwise intercluster dissimilarities along the i-1 remaining clusters

Design considerations
- Linkage techniques
  - Complete: maximal intercluster dissimilarity
  - Single: minimal intercluster dissimilarity
  - Average: mean intercluster dissimilarity
  - Centroid: dissimilarity between the centroid
- Measures
  - Euclidian distance
  - correlation based measures

Principal component analysis
- Suppose we wish to visualize n observations with measurements on a set of p features
- not all of p dimensions are equally interesting
- PCA find a low dimensional representation of the data set that constains as much as possible of the variation
  - seeks a small number of dimensions that are as interesting as possible

- First principal component
  - Normalized linear combination of the features that has the largest variance

    $Z = \emptyset_{11}X_1 + \emptyset_{21}X_2 + \ldots + \emptyset_{p1}X_p$

    $\emptyset_{11}$ = loadings of the PCA
    - define a direction in feature space along which the data varies the most
  - Normalized linar combination of the features

    $\sum_{j=1}^{p} \phi_{j1}^2 = 1$

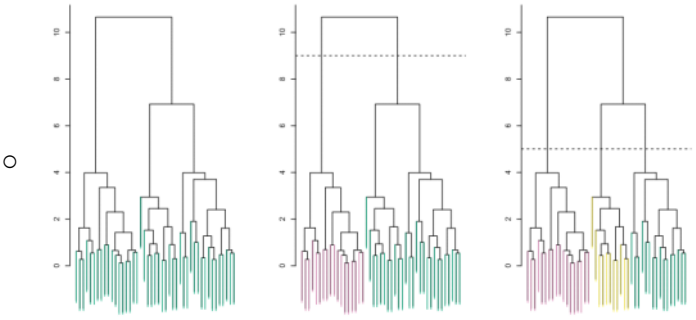
Hierarchical clustering

November 17, 2024 4:50 PM

K-clustering requires us to pre-specify the number of clusters K

Hierarchical clustering
- an alternative approach which deos not require us to commit to a particular choice K
- Results in a tree-based representation called a dendrogram
- Bottom-up clustering
    - most common type
    - dendrogram is built from the leaves up, combining clusters up to the trunk

Interpreting the dendrogram
- each leaf represents an observation
- as we move up in the tree, some leaves *fuse* into branches
    - correspond to similar observations
- as we move more up the tree, some branches fuse
- these fused branches fuse and so on and so on
- the earlier the fusions occur, the more similar the groups of observations are
- observations that fuse later can be v differrent
- There are 2^{n-1} possible reorderings of the dendrogram, where n is the number of leaves
    - at the n-1 pointw where fusion occur, the positions of two fused branches can be swapped without changing the meaning of the dendrogram
- do not draw conclusions about similarity of two observations based on their proximity along the horizontal axis
    - look at vertical axis position where branches are fused

Clustering with a dendrogram
- Cutting the dendrogram at various heights results in clusters
    - 
- number of observations beneath the cut = cluster
- can cut the dendrogram to get any number of clusters between 1 and n
- Sometimes, data cannot be well represented by hierarchical clustering
    - i.e. the best division into 3 groups is not the result from taking the best division into 2 groups and splitting up 1 of them into 2
        - i.e. suppose observations correspond to group of men and women evenly split among americans, japanese, and french
            - best division into 2 is by gender
            - best division into 3 is by nationality

The hierarchical clustering algorithm
- Simple algo
- measure dissimilarity (using euclidian distance)

1. each observation is treated as its own cluster
2. the two most similar clusters are fused so that there are now n-1 clusters
3. The next two clusters are fused again, n-2 clusters

4. Repeat until all of the observations belong to one single cluster

- How do we define dissimilarity between two clusters if one or both contains multiple observations
    - linkage: defines dissimilarity between two groups of observations
        - Complete, average, single, and centroid
        - Average and complete are preferred

Measuring dissimilarity

- alternatives to euclidian distance
- correlation-based distance
    - considers two observations to be similar if their features are highly correlated
    - may be far apart wrt euclidian distance
    - focuses on the shape of observations
- choose the measurement type which fits the problem well
- i.e. shopping preferences - assume data is a matrix where rows are shopers and columns are items available for purchase, entries is how many times they bought each item
    - if euclidian distance is used, shoppers who don't buy many things will be clustered together
    - if correlation-based distance is used, shoppers with similar preferences will be clustered

Issues with clustering

- Small decisions with big consequences
    - Should the observations or features first be standardized in some way?
        - i.e. scaling
    - What dissimilarity measure should we use?
    - What type of linkage should be use?
    - Where do we cut the dendrogram?
    - In k-means clusteringl, how many clusters should we look for in the data?
    - In practice, try several different choices and observe results - there is no single right answer
- Validating the obtained clusters
    - do the clusters represent the true subgroups of the data
        - could be a result of clustering the noise
    - no single best approach on validaitng this
- Other considerations
    - both k means and hierarchical will assign each observation to a cluster
        - sometimes this may not be appropriate
        - i.e. most of the observations truly belong to a small number of unknown subgroups
        - a small subset of the observations are very different from eachother and from all other observations
        - since every observation is forced into a cluster, the clusters found might be distorted due to the presence of outliers that don't belong to any cluster
        - mixture approaches address this, i.e. soft k means clustering
    - Clustering methods are not robust
        - cluster n observations, remove a subset, cluster again: you'll likely end up with different clusters

L23 - Unsupervised Learning: PCA

November 17, 2024 5:43 PM

Principal component analysis

- reduction of dimensionality in visualization of features
- Suppose we wish to visualize n observations with measurements on a set of p features
- not all of p dimensions are equally interesting
- PCA find a low dimensional representation of the data set that constains as much as possible of the variation
    - seeks a small number of dimensions that are as interesting as possible

- First principal component
    - Normalized linear combination of the features that has the largest variance

      $Z = \emptyset_{11}X_1 + \emptyset_{21}X_2 + \ldots + \emptyset_{p1}X_p$

      $\emptyset_{11}$ = loadings of the PCA
        - define a direction in feature space along which the data varies the most
    - Normalized linear combination of the features

      $\sum_{j=1}^{p} \phi_{j1}^2 = 1$

- $Z_m = \sum_{j=1}^{p} \emptyset_{jm}X_j \;\rightarrow\; y_i = \theta + \sum_{m=1}^{M} \theta_m z_m + \epsilon_i$

  can solve this with OLS

    - Solve the optimization problem:

$$\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n}\sum_{i=1}^{n} \underbrace{\left(\sum_{j=1}^{p} \phi_{j1}x_{ij} \right)^2}_{z_{i1}^2} \right\} \text{ subject to } \sum_{j=1}^{p} \phi_{j1}^2 = 1$$

- Second principal component
    - normalized linear combination of the features that has the largest variance out of all linear combinations that are uncorrelated with Z_1
    - $Z = \emptyset_{12}X_1 + \emptyset_{22}X_2 + \ldots + \emptyset_{p2}X_p$

      $\emptyset_{i2}$ = loadings of the PCA
    - Constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction of $\emptyset_1$ to be orthogonal to the direction of $\emptyset_2$
    - $\underset{\phi_{11},\ldots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n}\sum_{i=1}^{n} \left(\sum_{j=1}^{p} \emptyset_{j2}x_{ij} \right)^2 \right\}$ subject to $\sum_{j=1}^{p} \phi_{j1}^2 = 1$

PCA

- For multidimensional data, there are multiple principal components
- Principal components can be used to produce low-dimensional views
    - It is not feature selection, we consider linear combination of all features
- Eigen decomposition method is used to solve PCA
    - PC direction are computed using the order sequence of eigenvectors of the matrix X^TX
- For a set of p features and n observations, there are at most min(n-1, p) principal components
    - we use the smallest subset, determined by proportion of variance

- Before PCA is performed, the variables should be centered to have mean=0
    - PCA results are heavily dependent on scaling

Dimensionality reduciton

November 17, 2024 6:10 PM

Instead of taking a subset or shrinking coefficients, we can transform the predictors and fit a OLS model to reduce dimensions

Let $Z_1, Z_2, \ldots, Z_M$ represent $M < p$ *linear combinations* of our original p predictors. That is,

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j \tag{6.16}$$

for some constants $\phi_{1m}, \phi_{2m} \ldots, \phi_{pm}$, $m = 1, \ldots, M$. We can then fit the linear regression model

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i, \quad i = 1, \ldots, n, \tag{6.17}$$

- the thetas are the regression coefficients

Dimension reduction

- reduces the problem of estimating the p+1 coefficients (betas) to the simpler problem of estimating the M+1 coefficients (thetas), where M < p
  - the dimensions of the problem have been reduced from p+1 to M+1
- Dimension reduction serves to constrain the estimated β_j coefficients since now they must take ther form:

$$\beta_j = \sum_{m=1}^{M} \theta_m \phi_{jm}.$$

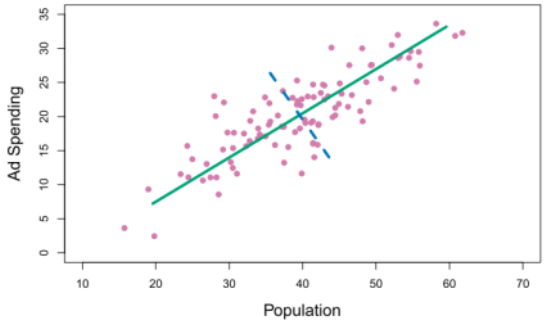
  - potentially increased Bias but significantly reduced variance
- If M = p and all Z_m are linearly independent, then this poses no constraints and no dimension reduction occurs

How does dimension reduction work

1. The transformed predictors Z_1, Z_2, … , Z_M are obtained
2. the model is fit using these M predictors

- the choice of Z1, Z2, … Zm (or the phi's) can be achieved in different ways
  - principal components and partial least squares

Principal components regression/PCA

- reducing the dimensions of an nxp matrix X
- First principal component direction of the data
  - the direction along which the observations vary the most
  - the line s.t. if we project the points onto the line, the points will have the largest possible variance
    - finding the location on the line which is closest to the point
    - 

    - fpc = green line
  - First principal component eqn

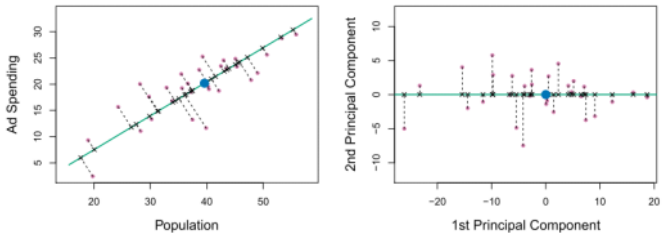
$$Z_1 = \varphi_{11} * (pop - mean_{pop}) + \varphi_{21} * (ad - mean_{ad})$$
where $\varphi_{11} = 0.839$ and $\varphi_{21} = 0.544$

- The idea: out of every possible linear combination of pop and ad s.t. $\varphi_{11}^2 + \varphi_{21}^2 = 1$, this linar comb. yields tha highest variance
  - maximized

$$\text{Var}(\phi_{11} \times (\text{pop} - \overline{\text{pop}}) + \phi_{21} \times (\text{ad} - \overline{\text{ad}}))$$

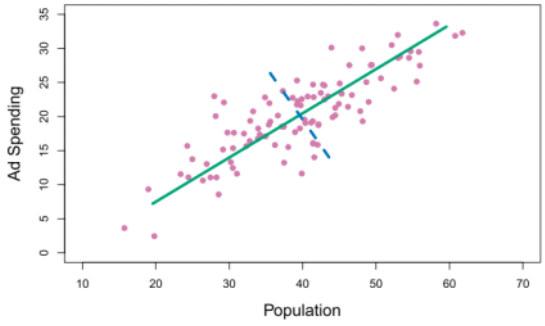
- Principal component scores
- $z_{i1} = \varphi_{11} * (pop_i - mean_{pop}) + \varphi_{21} * (ad_i - mean_{ad})$

Alternate interpretation

- the first principal component vector is the line that is as close as possible to the data
  - minimizes the sum of the squared perpendicular distances between each pt and the line
- 

- the first principal component appears to capture most of the info contained in the pop and ad predictors
  - the plots show a strong relationship between the first principal component and the two features
  - pop and ad have an approximately linear relationship

The second principal components

- we can have up to p distinct principal components
- Z_2 - the 2nd principal component: a linear combination of the variables that is uncorrelated with Z_1 and has the largest variance subject to this constraint
- 

- spc = blue line
- must be perpendicular to the fpc direction
- Second principal component formula
  $Z_2 = 0.544 * (pop - mean_{pop}) - 0.839 * (ad - mean_{ad})$
- since the data has 2 predictors, fpc and spc contain all the information that is in pop and ad
  - the fpc will contain the most information

Principal components regression

- construct the first M principal components Z1 … Zm and use these components as predictors in a linear regression model that is fit using least squares
- often a small number of principal components suffice to explain most of the variability in the data
- We assume that the directions in which X1, … Xp show the most variation are the directions that are associated with Y
  - approximation
- Fitting the least squares model to Z1 … Zm will lead to better results than fitting to X1 … Xp, since most or all of the information in the data pertaining to Y is contained in the Z's

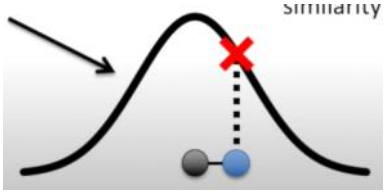
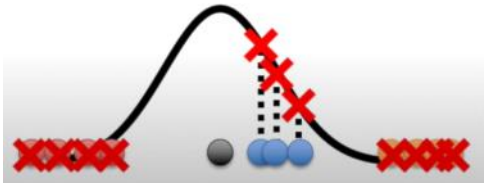
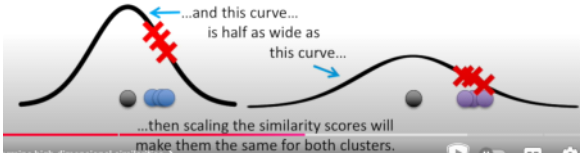
    - by estimating fewer coefficients, we can mitigate overfitting (M << p)
- More principal components = less bias but more variance
    - MSE still a U shape
- PCR does best when the first few principal components are sufficient to capture most of the variation in the predictors as well as the relationship with the response
- NOT a feature selection method

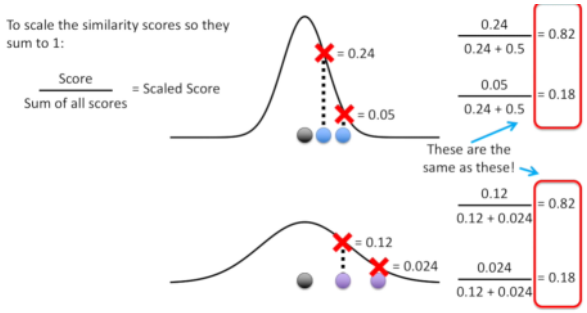
L24 - Visualization

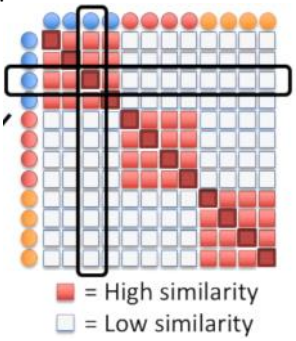
November 20, 2024 3:34 PM

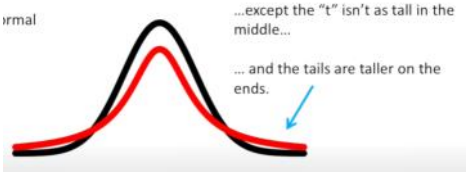
- Need visualization for
  - understanding training data
  - inspecting model
  - communicating model results
  - dealing with high dimensional data

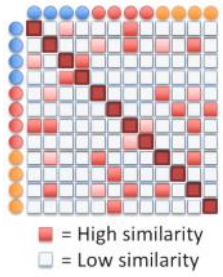
High-dimensionality: T-SNE

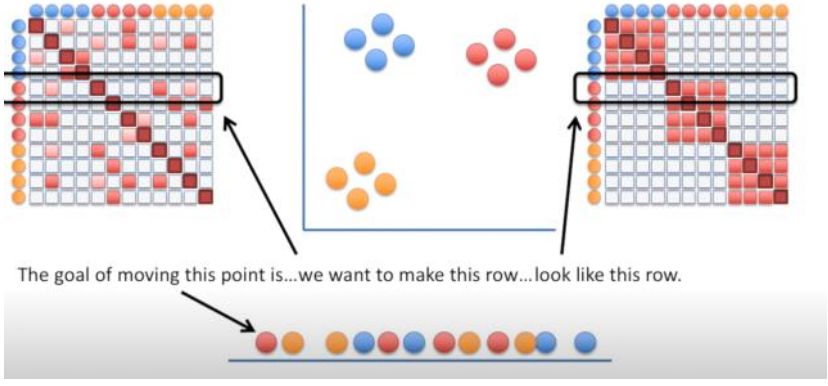
- Gaussian dist
  - a normal (gaussian) distribution is a type of continuous probability distribution for a real-valued random variable
- Kullback - Leibler divergence
  - a measure of how one reference probability dist P is different from a second probability dist Q
  - Also called relative entropy and I-divergence
  - $$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$
- T-sne
  - finds a way to project the data into a low dimensional space while preserving clustering
    - i.e. 2d graph onto 1d line
      - start by randomly placing all points onto the line
      - tsne will move these points a little at a time until they have been clustered
        - points are attracted to other points of the same cluster
        - pointd are repelled by points of distant/different clusters
        - think one point at a time
        - attraction is stronger than repelling
    - determining the similarity between points
      - measure the distance between 2 points
      - plot that distance on a normal/gaussian curve, with the point of interest centered
        - 
      - x is the unscaled similarity between the point of interest (black) and the other point (blue)
      - calculate unscaled similarity for black point and all other points one by one
      - 
    - rescale all unscaled similarities so they add to 1
      - width of normal curve depends on density of the cluster
      - scaling will equalize distances for dense and sparse clusters
      - 

-
  - 

  - perplexity: expected density
  - because normal dist is based on density of nearby points (cluster), similarity of point A1 to B1 might be different than similarity of point B1 to A1 because cluster A and cluster B may have different densities
    - average similarities from both directions to fix this
- end up with a matrix of similarity scores
  - 

- Now, go back to the line example
  - For all the points on the line, measure the distance as described above using a t-distribution instead of a normal dist
  - 

  - 

- 

  - move the points a little at a time, and at each step it chooses a direciton that makes the t-dist matrix look more like the n-dist matrix

- for n high dimensionality points, stoachastic neighbour embeddeing (sne) starts converting the high dimensional euclidian distances between datapoints into conditional probabilities that represent similarities
- the similarituy of xj to xi is p_j|i
  - is the probability that xi would pick cj as its neighbour if neighbours were picked in proportion to their probability density undera gaussian centered at xi
- For nearby points, p_j|i is relatively high
- for widely separated points, p_j|i is extremely small
- $$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

- Process
  - since we are only interested in modeling pairwise similarities, p_i|i is set to 0
  - consider the low dimensionality counterparts (yi, yj) of the high dimensionality points (xi, xj)
  - The similar the conditional probability q_j|i can be computed
  - Gaussian variance σ is set to sqrt(1/2)
  - $$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)} \qquad q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)}$$

- How it works
  - SNE minimizes the sum of divergences over all datapoints using a gradient descent method where…
    - P_i represents cond prob dist over all other datapoints given data point x
    - Q_i represents cond prob dist over all other map points given map point yi
  - $$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \qquad \frac{\delta C}{\delta y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

    Gradient updates $$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$$ Without symmetric adjustments
  - $$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \qquad \frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

    Gradient updates $$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$$ With symmetric adjustments
  - Parameters to be selected
    - variance of the gaussian that is centered over each high-dim datapoint
  - SNE performs binary search for the value σ_i that produces a P_i with a fixed perplexity that is specified by the user
    - $Perp(P_i) = 2^{H(P_i)}$
  - Where shannon entropy of Pi measured in bits
    - $$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$
  - Step size for gradient descent

!L25 - ML OPs

November 20, 2024 3:35 PM

L26 - Intro to reinforcement learning

November 23, 2024 1:28 PM

- A software agent
  - makes observations
  - takes actions within an environment
  - recieves a reward
    - the reward may be positive - award for taking the right action
    - the rweard may be negative - punishment for taking the wrong action
- Agent must maximize the reward and minimize punishment
- entirely trial and error

- Pac man
  - environment is the simulation of the game
  - agent controls pac man's movements
  - actions are possible directions in which the movements can occur
    - up left right down
  - reward is points earned/game score
  - observation is screenshots of the game stages

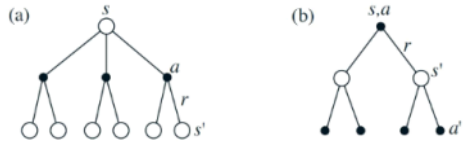
Policy
- RL needs policy function to determine behaviour of the agent
- policy defines agent's way of behaving at a given time
- mapping from percieved states (environment) to actions the agent must take when that state is encountered
- can be a simple hash table or a complex search algorithm

Reward
- at each time step, agent receives a numeric reward and wants to maximize it
- reward depends on current action and current observation
- agent may learn a policy based on reward
  - not learn reward based on policy
- if in the current state, the policy selected gives low reward, the policy is changed for the future state
- reward signal indicates what is good in current state

Value
- indicates what is advantageous in the long run
- long-term-desirability of a given state in the env
- a state with low immediate reward does not necessarily yield a bad score, maybe it is helpful in the long run
- value function is estimated through observations and reward
- goal is to seek state of highest value, not highest reward

Model
- simulation of the environment and mimic its behaviour in real time
- for given state and action, model might predict the next state and reward
- models are useful for planning a set of actions to anticipate states an agent might experience even before they are encountered
- model based methods use a model and planning
  - model free methods do not

Learning
- the idea is to play many games

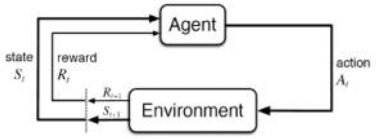
- we observe the result from each move many times, select states that lead to the greatest value
- as we progress, we change the values of our current state
- to make more accurate estimates of the probability of success, we back up the value of state after each greedy move
  - 

    Backup diagrams for (a) value function and (b) back up policy

  - s = state before greedy move,
    s' = state after the move,
    V = estimated value of s $V(s) \leftarrow V(s) + \alpha\Big[V(s') - V(s)\Big]$

- if the step size parameter is reduced properly over time, this method converges, for any fixed opponent
- if the step size parameter is not reduced all the way to zero over time
  - player also plays well against opponents that slowly change their way of playing

Policy evaluation

- how to we compute state-value of an aribitrary policy
  - $$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \end{aligned}$$
  - 

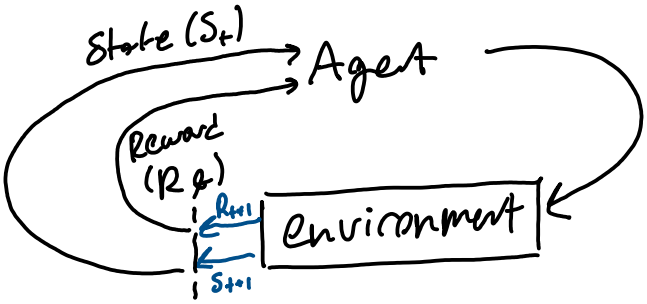
- if the env's dynamics are completely known, then the above is a system of |S| simultaneous linear equations
- Use the bellman equation
  - averages over all the possibilities, weighting each by its probability of occurring
  - $= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big]$
  - $$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big] \end{aligned}$$

L28 - Reinforcement

November 20, 2024 3:36 PM

Markov decision process

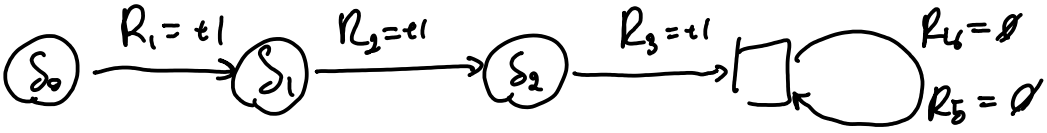
- Reward received now versus in the future does not carry the same weight



$0 \leq \gamma < 1$ is the discount rate

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{k-1} R_{T+k+1}$$

- a state signal succeeds in retaining all relevant info is said to be Markov, or to have the Markov property
  - Markov signal: retains all relevant info



- i.e. a game of checkers, screenshot of pac man, can see al important information at once

- When is it not a markov decision process?
  - The marble draw: when there is a connection between the previous marble draw and the current marble draw, this is not a markov process
- The environment's response at $t + 1$ depends only on the state and action representations at t

- The markov property is important because decisions and values are assumed to be a function of ONLY THE CURRENT STATE

Markov decision process

- a reinforcement learning task that satisfies the markov property (MDP)

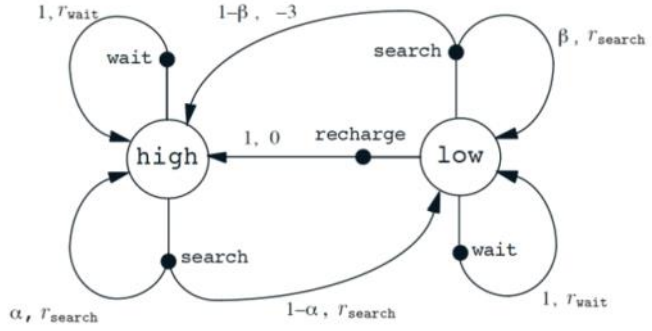
- i.e. a robot throwing out trash
  - Energy level = {high, low} (battery)
  - A {high} = {search, wait} where search and wait are actions
  - A {low} = {search, wait, recharge}

  - Can use this to make a hash table

$$p(s', r|s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t = s, A_t = a\}$$

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

- Can use to make a flow diagram



r is *reward*

Value functions and Dynamic programming

- value functions are estimated for states (or state-action pairs) to estimate how good it is for an agent to be in each state
  - or how good it is to perform an action while in a state
- State value function and action value function is optimizes based on a certain policy and is estimate from experience

- state-value function: $V_\pi(s)$
- action-value function: $q_\pi(s)$
- policy: π

- Bellman equation: $V_\pi(s) = \sum_a \pi(s|a) \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')]$

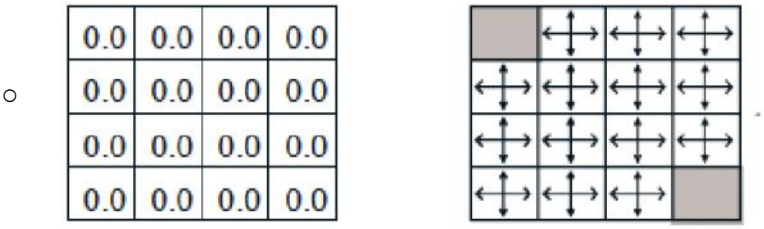
Policy iteration

- nonterminal states {1, 2, … 14} R = -1 on all transistions until the terminal state is reached
- actions: move up down left right
- we need a policy and a value funtion

$p(6|5, \text{right}) = 1$
$p(10|5, \text{right}) = 0$
$p(7|7, \text{right}) = 1$
$r(s, a, s') = 1$

- Start with random policy, then create greedy policy wrt random policy

V_k for the Random Policy | Greedy Policy w.r.t. V_k



- Iteration
  input pi, the policy to be evaluated
  init an array V(s) = 0, for all seS
  Repeat
  - $\Delta \leftarrow 0$
  - For each s $\in S$:
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow \sum_a \pi(s|a) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$
    - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

  Until **Δ <** theta (a small positive number)
  Ouput $V \approx v_\pi$

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Limitations of policy iteration
- each iteration involved policy evaluation
  - heavy computation

Value iteration
- Initialize an array $V(s) = 0$, for all $s \in S$
- Repeat
  - $\Delta \leftarrow 0$
  - For each s $\in S$:
    - $v \leftarrow V(s)$
    - $V(s) \leftarrow max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$
    - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
- Until $\Delta < \theta$ (a small positive number)
- Output a deterministic policy, π, such that

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

L29 - RNN

November 23, 2024 2:58 PM

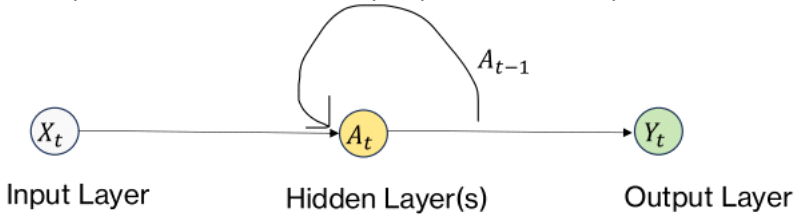
- when sequence is important in datasets
- when applying ML to sequences
  - imput variable is a sequence
  - target var is also a sequence
- Rarely do we have fixed target sequences
  - in this case, we predict the next item in the sequence at next time step t+1 based on input data until time step 1

Feedforward NN

- fixed size input vectors with associated weights to capture all relevant aspects of an example at once

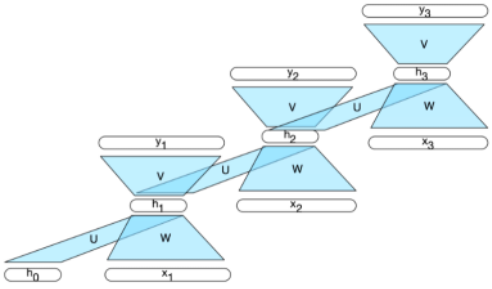
RNN

- recurrent neural network contains a feedback connection within its network connections
- the outputs of one unit is directly dependent on the previous unit
-



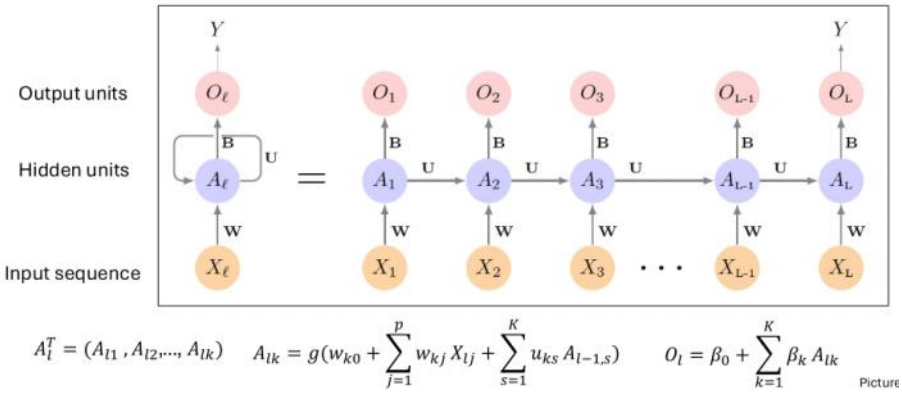
Inference

- the mapping between inptus and outputs is almost like traditional NNs
- $h_t = g(Uh_{t-1} + Wx_t)$
- $y_t = softmax(Vh_t)$
- Given a hidden state in our model, allows storage of info
- goal is to get a good estimate of prob dist over the space of hidden state vectors to compute the next output
-



- Giving a hidden state to our model allows for storage of info
- get a good enough estimate of prob dist over the space of hidden state vectors, to compute the next output
-
  - $h_0 \leftarrow 0$
    - for $i \leftarrow 1$ to length (x) do
      - $h_i \leftarrow g(Uh_{i-1} + Wx_i)$
      - $y_i \leftarrow f(Vh_i)$
    - return y

-



Texr classification

- use embedding spaces instead of onehot
- embedding spce can be learned or weights can be static
    - popular frozen weights: word2vec, gloVe
        - Glove: unsupervised learning algo for obtaining vector represenations for words

-

```
word_to_ix = {"how": 0, "are": 1, "you":2}
embeds = nn.Embedding(3, 5)  # 3 words in vocab, 5 dimensional embeddings
lookup_tensor = torch.tensor([word_to_ix["how"]], dtype=torch.long)
how_embed = embeds(lookup_tensor)
print(how_embed)
```

Time-series forecasting

- predicting trading trend/stocks to buy/stock price is very hard
- predicting trade volume is more reasonable

Read chapter 10 of textbook