PERFECT EDGE® MICRO PERFORATED FOR CLEAN TEAR-OUTS

NORCOM INC. GRIFFIN, GA 30224
Item #21676
www.norcominc.com
MADE IN MEXICO

Machine Learning (IBM)

1 SUBJECT

COLLEGE RULED

70 SHEETS

Dataset Goal

Supervised - has target Predict
Unsupervised - does not have target Pattern

ML Workflow
1. Problem Statement
2. Data Collection
3. Data Exploration & preprocessing
4. Modeling
5. Validation
6. Decision Making & Deployment

Features | Target

sepal len	sepal width	species
6.7	3.0	virginica
4.6	3.4	setosa

example observation

label - value for target variable

Retrieving Data

pd.read_csv (Useful Arguments)
- sep - what type of separator String
- delim_whitespace - uses whitespace as delimiter bool
- header - specify which row is headers int
- names - list of column names list
- na_values - list of values that are NA list

pd.read_json & pd.to_json

SQL Databases (can use sqlite3)
- create connection to database
- write query
- pd.read_sql(query, connection)

NoSQL Databases
- most store in JSON format

APIs and Cloud Data Access
- use url instead of filepath

Data Cleaning

Main Data Problems:

- Lack of data
- Too much data
- Bad data

Messy Data:

- Duplicate/unnecessary
- Inconsistent text & typos
- missing data
- outliers
- data sourcing

Missing Data:

- remove observation
  - ✓ quick clean
  - ✗ losing data
- impute (replace w/ estimation)
  - ✓ no losing data
  - ✗ more uncertainty
- mask data (all missing values are own category)
  - ✓ maybe find pattern
  - ✗ more uncertainty

Outliers:

- some provide important info
- hist / boxplot to identify (or stats IQR)
  sns.distplot sns.boxplot
- Residuals = actual - pred
  - standardized residual - residual / std error
  - deleted - residual from model on data - current observation
  - studentized - deleted residuals / std error
- Dealing w/ outliers
  - remove them
  - assign diff value / estimate (regression)
  - transform variable
  - keep, but use outlier-resistant models

Exploratory Data Analysis & Feature Engineering

- Summary Statistics - avg, min, max, correlation, etc.
- Visualizations - hist, scatter, box
- Data Wrangling - Pandas, Data Visualization - Matplotlib, sns
- sampling - df.sample(n=n)
- sns.pairplot for correlation/relation between features
- sns.jointplot for density

Feature Engineering & Variable Transformation

- Data Transformations can fix skew
  - log transformation - $\log(x)$
  - polynomial features - x^n
- Variable Selection - choosing features to include
  - encoding (categorical → numeric)
  - scaling
  - Encoding
    - nominal (no order)
    - ordinal (ordered, eg. high, low)

    ] types of categorical data
    - Binary encoding (converts to 0/1)
    - One-hot encoding
    - Ordinal encoding (1 int per category)
  - Feature Scaling
    - adjusting scale so diff variables can be compared
    - standard scaling (z-score)
    - min-max scaling $\left(\frac{x-min}{max-min}\right)$
    - robust scaling - maps IQR to be 0-1

Common Variable Transformations

Nominal	Binary, one-hot
Ordinal	Ordinal encoding

Estimation, Inference, & Hypothesis Testing

- Estimation - application of algorithm
- Inference - involves putting an accuracy on the estimate
- Parametric model - set of distributions or regressions w/ finite # of parameters
- Non-parametric - fewer assumptions
  - ↳ assume data don't belong to particular distribution
- most common way of estimating parameters in parametric model is through maximum likelihood estimation (MLE)
  - likelihood function is related to probability and is a function of the parameters of the model
  - maximize ML function
- Common Distributions
  - Uniform
  - Normal
  - Log Normal
  - Exponential
  - Poisson

Frequentist vs. Bayesian Statistics

- frequentist is concerned w/ repeated observations in the limit
  - ↳ many repeats
- bayesian describes parameters by probability distributions
  - ↳ prior dist → posterior dist

Hypothesis Testing

- hypothesis - statement about population parameter
  - ↳ null hypothesis - H_0
  - ↳ alternative hypothesis - H_1 or H_A
- likelihood ratio - ratio of which (H_1 or H_0) is more likely

Type I vs Type II Errors

- Neyman-Pearson Paradigm is non-Bayesian

	Decision: Accept H_0	Decision: Accept H_1
Truth H_0	correct	Type I error
Truth H_1	Type II error	correct

- power of test = 1 - P(Type II Error)

Terminology

- test statistic - used to decide whether to accept/reject H_0
- rejection region - set of values for test statistic → reject H_0
- acceptance region
- null dist. - test statistic's dist. when H_0 true

Significance Levels & p-Values

- significance level (α) is threshold (probability) below which null hypothesis is rejected
  - ↳ must be chosen before computing test statistic
- p-value - smallest significance level at which null hypothesis is rejected
- confidence interval - values for which we accept the null

F-statistic

- H_0 - data modeled by setting $\beta_{n \cdot nEN} = 0$

Power & Sample Size

- more samples → ↑ P(Type I error)
- Bonferroni Correction - choose $p_{threshold}$ so P(Type I error) = .05
  - ↳ costs power (correctly rejecting null)

Correlation vs. Causation

- if X and Y are correlated, X is helpful in predicting Y
  - ↳ X causes Y
  - ↳ Y causes X
  - ↳ X & Y both caused by something else (confounding)
  - ↳ X & Y not related, got lucky (spurious)

Supervised ML

ML Framework:

$y_p = f(\Omega, x)$

- Ω (fit parameters) involves aspects of the model we fit using the data
- hyperparameters are decisions regarding how to produce Ω

2 categories
- Regression
  - ↳ predict numeric value
- Classification
  - ↳ predict categorical var

$J(y, y_p)$ (loss) drives updating of parameters
- ↳ using x & y, choose parameters Ω to min(J)

tradeoff:
- Interpretation - understanding mechanism
- Prediction - predicting outcome (focus on performance metrics)

Workflows:

Interpretation:
- gather x, y
- train for best y_p
- focus on Ω to gain insights

Linear Regression

given $y_\beta(x) = \beta_0 + \beta_1 x$,

residual ~~(Error)~~ $= y_\beta(x^{(i)}_{obs}) - y^{(i)}_{obs}$

$= (\beta_0 + \beta_1 x^{(i)}_{obs}) - y^{(i)}_{obs}$

error = L1 norm (abs val)

OR (more commonly) Euclidian distance

$J(\beta_0, \beta_1) = \frac{1}{2}\left(\frac{1}{m}\sum_{i=1}^{m}\left((\beta_0 + \beta_1 x^{(i)}_{obs}) - y^{(i)}_{obs}\right)^2\right)$ (L2 norm) (mean sq. error)

Other measures of Error

Sum of Squared Error (SSE): $\sum_{i=1}^{m}\left(y_\beta(x^{(i)}) - y^{(i)}_{obs}\right)^2$

Total Sum of Squares (TSS): $\sum_{i=1}^{m}\left(\overline{y_{obs}} - y^{(i)}_{obs}\right)^2$ (total variation)

Coefficient of Determination (R^2): $1 - \frac{SSE}{TSS}$ (total explained variation by our model)

↳ will almost always increase w/ more features

Syntax

import from sklearn, instantiate class, ~~train~~ fit

Train Test Splits

- data leakage - test data "leaks" into training data

train_test_split fn pass in data, % of data for test set OR pass in x, y

ShuffleSplit, StratifiedShuffleSplit

Polynomial Regression

- Addition of Polynomial Features - capture higher order features of data by adding polynomial features
  - still linear regression (linear combination of features)
  - can also include variable interactions ($\beta_3 x_1 x_2$)

Choosing correct functional form

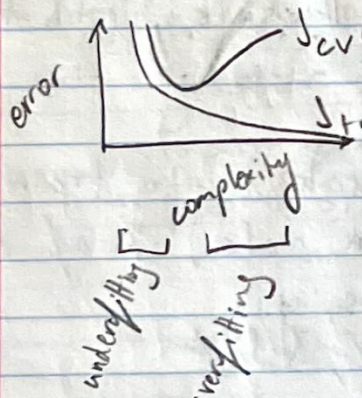
- check relationship of each variable w/ outcome

Syntax

```
polyFeat = PolynomialFeatures(degree=2)
polyFeat = polyFeat.fit(X_data)
X_poly = polyFeat.transform(X_data)
```

Cross Validation

- multiple train-test splits
  - avg. performance - more statistically significant
- called validation
  - test has never been seen
- diff model for each set



Most Common CV Approaches

- k-fold
- Leave one out
- Stratified

Bias - Variance Tradeoff

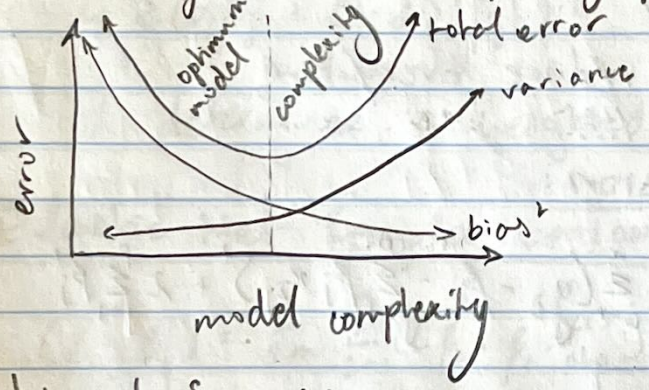
bias - tendency to miss
variance - tendency to be inconsistent

3 Sources of Error
model

being wrong (bias)
being unstable (variance)
unavoidable error (irreducable)

Bias:
- worsened by missing info, overly simplistic assumptions
- miss real patterns (underfit)

Variance:
- sensitivity of output to small changes in input
- overly complex or poorly-fit models (overfit)



- model adjustments usually yeild bias $\propto \frac{1}{\text{variance}}$
- bias-variance tradeoff is analogous to complexity tradeoff

Visual Signs:
- bias - too rigid to capture data
- variance - predictions fluctuate wildly

Regularization & Model Selection

Adjusted Cost Function:

$$M(w) + \lambda R(w)$$

- $M(w)$ - model error
- $R(w)$ - function of estimated parameters
- λ - regularization strength parameter

penalize model if it's too complex

$\lambda \propto$ penalizing larger parameters

λ helps manage complexity tradeoff:

- more regularization means simpler model
- less regularization means ~~not~~ complex model

- performs feature selection by shrinking contribution of features

↓ features can reduce overfitting

Ridge Regression:

Cost Function:

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Scale now matters

λ is applied proportionally to squared coeff. values

select best λ by cross validation

best practice to scale features

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left((\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)}\right)^2 + \lambda \sum_{j=1}^{k} \beta_j^2$$

$\lambda \propto \frac{1}{\text{standardized coeffs.}}$

↳ Least Absolute shrinkage and selection Operator

LASSO Regression

penalizes coeffs. differently

LASSO - L1	Ridge - L2
$\|\beta\|_1 = \sum \|\beta_j\|$	$\|\beta\|_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$

λ is proportional to abs(coeffs.)
LASSO is more likely than Ridge to perform feature selection

Slower to converge than Ridge

selectively shrinks some coeffs, opposed to Ridge, shrinks all at once

Elastic Net (hybrid)

$$\lambda \sum_{j=1}^{p} (\alpha \beta_j^2 + (1-\alpha)|\beta_j|)$$

α determines a weighted average of L1 and L2 penalties

Elastic Net Regularization

$$J(\beta_0, \beta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\beta_0 + \beta_1 x_{obs}^{(i)}) - y_{obs}^{(i)})^2 + \lambda \sum_{j=1}^{k} |\beta_j| + \lambda \sum_{j=1}^{k} \beta_j^2$$

Recursive Feature Elimination (RFE)

- define model & desired # of features
  - ↳ RFE repeatedly applies model, measures feature importance, recursively removes less important features

Details of Regularization

Geometric View:

Ridge $\quad \underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\}$ subject to $\sum_{j=1}^{p} \beta_j^2 \leq s$

LASSO $\quad$ same but subject to $\sum_{j=1}^{p} |\beta_j| \leq s$

Probabalistic View:

$$p(\beta | X, Y) \propto f(Y | X, \beta) p(\beta, X) = f(Y | X, \beta) p(\beta)$$

$$p(\beta) = \prod_{j=1}^{p} g(\beta_j)$$

letting f = probability of target given parameter vector β and $p(\beta)$ the prior distribution of β, we can calculate the posterior of β

$p(\beta)$ is derived from independent draws of a prior coefficient density function g chosen when regularizing

L2 - Gaussian prior, L1 - Laplacian prior

Logistic Regression

sigmoid function = $y = \frac{1}{1+e^{-x}}$

- removes skew of large samples
  - ↳ when x = linear regression model, becomes logistic regression

$$y_\beta(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x+\varepsilon)}}$$

$$P(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x+\varepsilon)}} = \frac{e^{(\beta_0+\beta_1 x)}}{1+e^{(\beta_0+\beta_1 x)}}$$

interpretative purposes

$$\Rightarrow \text{odds ratio} = \frac{P(x)}{1-P(x)} = e^{(\beta_0+\beta_1 x)}$$

$$\log\left(\frac{P(x)}{1-P(x)}\right) = \beta_0+\beta_1 x$$

Multiple Classes: One vs. All

set one class and combine others

- ↳ same for each class
- ↳ n logistic regression models (n classes)

estimated class is class w greatest probability

Metrics (Classification)

Confusion Matrix

	Pred. Pos	Pred. Neg
Actual Pos	TP	FN ← Type II err.
Actual Neg	FP ↑ Type I err.	TN

$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

↳ not good if skewed data

trade-off

$$\text{Recall (sensitivity)} = \frac{TP}{TP+FN}$$

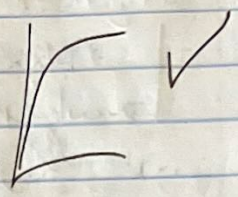
↳ $\text{Precision} = \frac{TP}{TP+FP}$

$$\text{Specificity} = \frac{TN}{FP+TN}$$

$$F1 = 2\,\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Reciever operating Characteristic (ROC) curve

y-axis - TP rate (sensitivity)
x-axis - FP rate (1-specificity)



~~close to~~ ~~AUC 1~~ top left is best
ROC AUC is $\int ROC$

precision-recall curve
y-axis - precision
x-axis - recall

ROC - better for balanced classes
P-R - better for unbalanced classes

multiclass is one v. all approach

---

k Nearest Neighbors

looks at k nearest neighbors to make pred.
- use odd k (no ties) $k = n\text{-classes}(m) + 1$
m is any $\mathbb{Z}$

decision boundary depends on k
find right k w/ trial & error

Distance Measurement:
L1 - Euclidian distance - literal distance
L2 - Manhattan distance - sum of |parameter| for all parameters

SCALE IS IMPORTANT

- regression by mean val. of neighbors

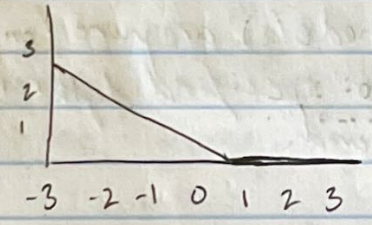
KNN Pros:
- simple
- adapts well to new data
- easy to interpret

KNN Cons:
- slow
- no insight to data generating process
- lots of memory
- ~~too~~ breaks down w/ many dims. (curse of dimensionality)

Support Vector Machines

SVM depends on hinge loss - heavily penalizes values inside margin

Cost Function



SENSITIVE TO OUTLIERS

↳ regularization

$J(\beta_i) = \text{SVM Cost}(\beta_i) + \frac{1}{c}\sum_i \beta_i$

$\frac{1}{c}\sum_i \beta_i$ penalizes overfitting

Kernels: transform data so it is linearly seperable

Approach 1: higher order features to transform data

★ Approach 2: (create) transform to diff coordinate space

↳ eg similarity to specific data points (ideally every point)

↳ similarity based on gaussian curve

$[x_1, x_2] \rightarrow [a_1, a_2, a_3]$

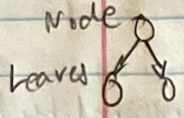
SVMs w/ RBF kernels are slow to train

↳ construct & approx. kernel map with SGD using Nystroem or RBF sampler

Workflow:

Features	Data	Model Choice
many (10k)	small (1k)	Linear SVC simple, logistic, or SVC
few (<100)	Medium (10k)	SVC w/ RBF
few (<100)	many (>100k)	add features, logistic, Linear SVC, kernel approx

---



Decision Trees

- Divides dataset in 2 at every opportunity
  - ↳ iterates
    - ↳ uses majority class at last leaves to predict

also regression
- ↳ avg. of final leaf
- ↳ more depth → better fit (overfit possible)

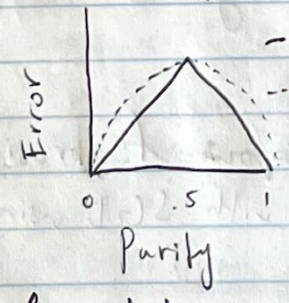
How long to keep splitting?

overfit → - keep splitting until leaf node(s) are pure (only one class
- max depth - prune to get to specified depth remains)
- performance metric is achieved

How to find best split?
- wave representing all splits, greedy search finds best
  - ↳ max info gain

classification error $E(t) = 1 - \max_i [p(i|t)]$

Entropy $\Rightarrow H(t) = -\sum_{i=1}^{n} p(i|t) \log_2 [p(i|t)]$
- ↳ reduce entropy
- ↳ reach homogeneous models



Gini Index → $G(t) = 1 - \sum_{i=1}^{n} p(i|t)^2$

- tend to overfit
  - ↳ max depth / prune trees
    - ↳ classification error threshold
    - ↳ info. gain threshold

Pros:
- easy to interpret
- handle any data category
- no preprocessing or scaling

Ensemble Based Methods & Bagging

(combining models)

bootstrap aggregation

- Combine preds. from many trees to reduce overfitting
  - ↳ each tree gets vote for final decision

How many trees?
- ↑ trees = ↓ overfitting - diminishing return after 50 (usually)

Specific to bagging:
- less variance
- can grow trees in parallel

- for n independent trees, each w/ variance σ^2, bagged variance = $\frac{\sigma^2}{n}$
- However, bootstrap samples are correlated (ρ):

$$\rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

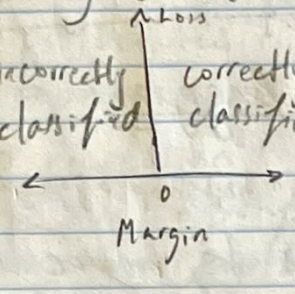
- further de-correlate trees
  - ↳ use random subset of features for each tree
    - classification: $\sqrt{m}$
    - regression: $m/3$
  - ↳ even more randomness: no greedy splits
    - ↳ extra random trees

Random forest

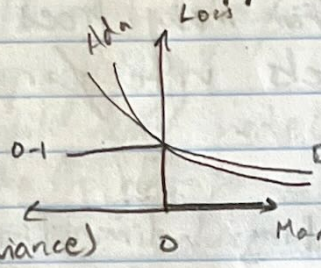
Boosting

- create initial decision stump
- weight wrong data points
- new decision stump
  ↳ repeat
- combine to form single classifier
- result is weighted sum of all classifiers
- using learning rate $\lambda \leq 1$ helps prevent overfitting

- boosting utilizes diff loss functions



0-1 Loss:
misclassified points × 1
correctly classified points × 0
difficult to optimize



Adaboost
$e^{-margin}$
- sensitive to outliers

Gradient Boosting (Deviance)
$\log(1 + e^{-margin})$

← use CV to set # trees

Bagging v.	Boosting
- bootstrapped samples	- fit entire dataset
- base trees indep	- base trees, successive
- only data points considered	- uses residuals from prev. models
- no weighting used	- weighting used
- excess trees will not overfit	- beware overfitting

Stacking

- train diff base learners on data ← any type of model
- combine outputs to create meta features
- train meta classifier on meta features
  - ↳ majority vote

- beware of increasing complexity

Model Interpretability

- self interpretable - minimum effort to understand
  - ↳ lin. reg.
  - ↳ tree models
  - ↳ KNN
- Non self-interpretable → post-hoc
  - ↳ ensemble models
- intrinsic

Model-agnostic Explanations

↳ produce explanations of models

- most important features
  - ↳ perm. feat. importance.
    - ↳ shuffle feat. values
      - ↳ most error is most important
  - ↳ relation to outcome
    - ↳ partial dependency plot

Surrogate Models

- explain complex black-box models by training self-interpretable surrogate models on input & output

Modeling Unbalanced Classes

↑ recall, ↓ precision

- downsampling - less samples of majority class
- upsampling - duplicate minority class
  - ↑ recall > precision, but diff is smaller

Steps:
- stratified train-test split
- up or down sample
- build models
  (use CV to find # samples)

random oversampling - resample w/ replacement
  ↳ good for categorical data

synthetic oversampling
- start w/ 1 pt
- choose k nearest neighbor
- add new pt. between them
- 2 main approaches
  - ↳ SMOTE
    - ↳ regular (kNN. can be any class)
    - ↳ borderline (outlier, safe, or in-danger)
    - ↳ SVM (uses SVM to generate new points)
  - ↳ ADASYN
    - ↳ # new samples ∝ competing classes

Near-Miss-1 - keep maj samples closest to minority pts.
  ↳ easily thrown off by outliers

Near-Miss-2 - keep maj samples closest to farthest min. pts.

Unsupervised Learning

- data points don't have unknown outcome

↳ Clustering - identify unknown structure in data
↳ Dimensionality Reduction - use structural characteristics to simplify data

K-Means Clustering

1. randomly initialize locations of k centroids
2. find clusters (based on distance to centroid)
3. adjust centroid location to mean of cluster
4. repeat 2 & 3 until centroids do not move

K-means is sensitive to initial locations of centroids

Smarter initialization:
pick next point w/ probability $\text{distance}(x_i)^2 / \sum_{i=1}^{n} \text{distance}(x_i)^2$
(k-means *++)

Selecting K:
→ Inertia: $\sum_{i=1}^{n} (x_i - c_k)^2$ sq dist of point from centroid of cluster
↳ sensitive to n
→ Distortion: $\frac{1}{n}$ Inertia
↳ less sensitive to n

use if want some # of points
use for ~~more~~ clusters

Elbow Method:
inflection point is right k

Gaussian Mixture Models

soft clustering

- probability of labels
- fits k Gaussian models to data
- each Gaussian defined by μ_i (mean vector), Σ_i (covariance matrix)

$p(x) = \sum_{i=1}^{k} \pi_i \mathcal{N}(x|\mu_i, \Sigma_i)$, $\sum_{i=1}^{k} \pi_i = 1$ (π_i (component weight)

Distance Metrics

Euclidean (L2) = n-dim Pythag.
↳ more sensitive to curse of dimensionality

Manhattan (L1) = $\sum_i \Delta x_i$ (sum of Δ components)
↳ better than L2 for higher dim data

Cosine Distance = $\frac{A \cdot B}{|A||B|}$ (insensitive to scaling wrt origin)
↳ better for text data

Jaccard Distance - given 2 sets $1 - \frac{A \cap B}{A \cup B} = 1 - \frac{\text{len(shared)}}{\text{len(unique)}}$
↳ applies to sets (word occurrence)

Hierarchal Agglomerative Clustering

- Find closest pair, merge into cluster
  - repeat
    - ↳ can use clusters instead of points

Can stop when avg. cluster distance passes certain threshold

Single Linkage - minimum pairwise distance between clusters
↳ bad if noise

Complete Linkage - maximum pairwise distance
↳ ok w/ noise
↳ break apart larger, existing clusters

Avg Linkage - ~~avg~~ pairwise distance between centroids

Ward Linkage - merge based on best inertia

DBSCAN

- can have points that don't belong to a cluster
- clustered using density of local neighborhood
  - finds core points @ high density regions & expands away from them

Inputs: Metric (distance), ε (radius of local neighborhood), n_clu (determines density threshold for ε)

3 possible labels for point:
- Core (has more than n_clu points in ε neighborhood)
- Density-reachable (an ε-neighbor of a core point that has < n_clu neighbors itself)
- Noise (has no core points in ε-neighborhood)

Mean-Shift

- assigns points to nearest cluster centroid

1. Choose point & window
2. calc weighted mean in window (closer => more weight)
3. shift centroid of window to new mean
4. repeat 2 & 3 until convergence
5. repeat 1-4 for all data points
6. data points leading to same mode are same cluster

Weighted mean:

$$m(x) = \frac{\sum_{x_i \in W} x_i K(x_i - x)}{\sum_{x_i \in W} K(x_i - x)}$$

0001 True - latent
False - inputs

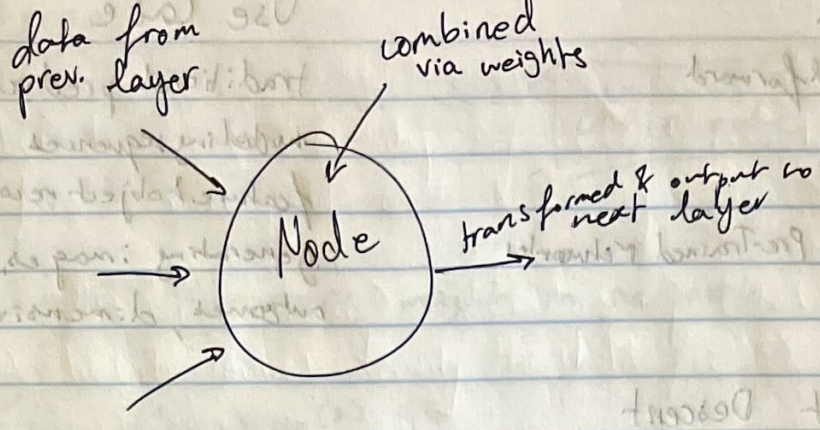
Method	K-means	Mean-Shift	Heirarchal	DBSCAN
Parameters	# clusters	bandwidth	# clusters	ε
Scalability	v. large n-samples, medium n_clusters	not scaleable w/ n_samples	large n-samples & n_clusters	same as k-means
General Use Case	general purpose even clusters, flat	many clusters uneven non-flat	many clusters	non-flat uneven many clusters

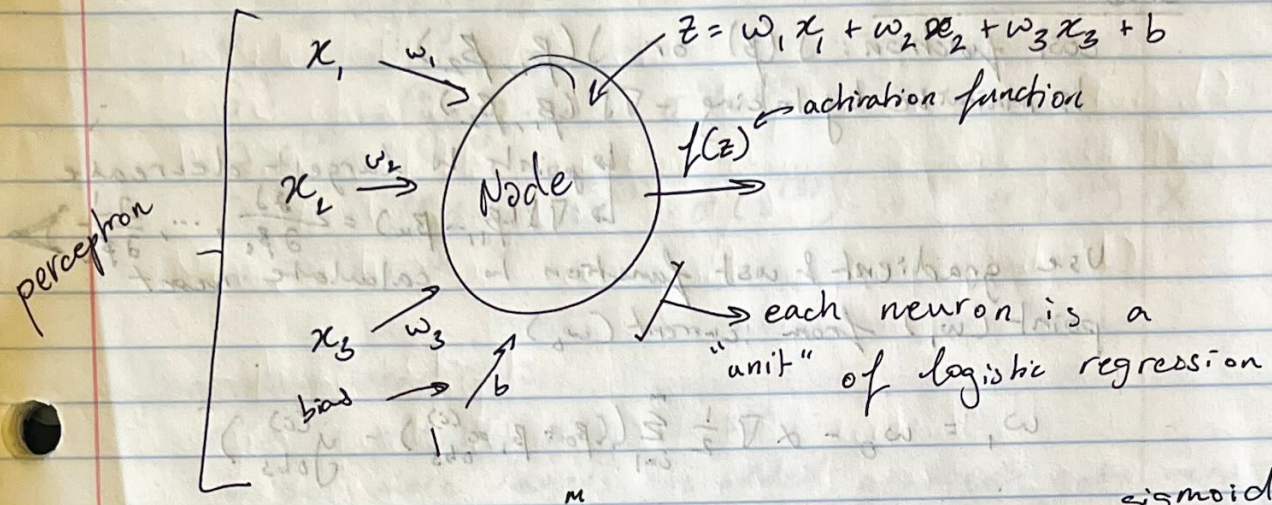
Dimensionality Reduction

Principal Component Analysis - replace colums w/ linear comb. of cols.
↳ Single Value Decomposition

Kernels for non-linear PCA

Neural Networks





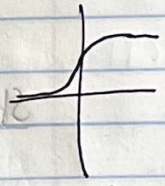
z - net input $= b + \sum_{i=1}^{m} x_i w_i$

b - bias term

f - activation function

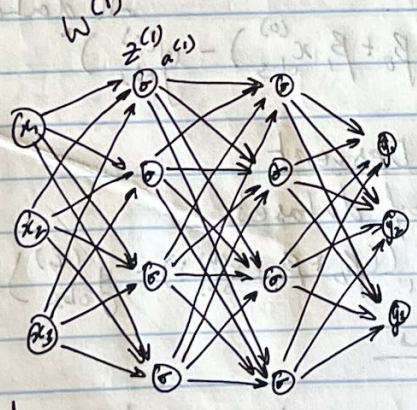
a - output to next layer

sigmoid



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$



$W^{(1)}$ is a 3×4 matrix

$z^{(1)}$ is a 4-vector

$a^{(1)}$ is a 4-vector

$x = a^{(0)}$

$z^{(1)} = x \cdot W^{(1)}$

$a^{(1)} = \sigma(z^{(1)})$

$z^{(2)} = a^{(1)} \cdot W^{(2)}$

$a^{(2)} = \sigma(z^{(2)})$

$z^{(3)} = a^{(2)} W^{(2)}$

$\hat{y} = a^{(3)} = \text{softmax}(z^{(3)})$

Real World:

input = $n \times 3$ matrix ← batches

output = $n \times 3$ matrix

Method	Use Case
MLP, feed forward	traditional predictive problems
RNN, LSTM	modeling sequences
CNN	feature & object recognition in visual data
Unsupervised Pre-Trained Networks	generating images, labeling outcomes, dimensionality reduction

Gradient Descent

cost function: $J(\beta)$ or $J(\beta_1, \beta_2)$

↳ minimize by taking $-\nabla J(\beta_1, \beta_2)$

↳ points to largest decrease

↳ $\nabla J(\beta_1, \ldots \beta_n) = \left\langle \frac{\partial J}{\partial \beta_0}, \ldots, \frac{\partial J}{\partial \beta_n} \right\rangle$

Use gradient & cost function to calculate next point (w_1) from current (w_0)

$$w_1 = w_0 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{m} \left(\left(\beta_0 + \beta_1 x_{obs}^{(i)} \right) - y_{obs}^{(i)} \right)^2$$

$\alpha \Rightarrow$ learning rate, determines step size

Stochastic Gradient Descent

- use a single data point to determine gradient & cost function instead of all data

$$w_1 = w_0 - \alpha \nabla \left(\frac{1}{2} \right) \left(\left(\beta_0 + \beta_1 x_{obs}^{(o)} \right) - y_{obs}^{(o)} \right)^2$$

Mini Batch Gradient Descent

- let $1 < n <$ size of dataset

$$w_1 = w_0 - \alpha \nabla \frac{1}{2} \sum_{i=1}^{n} \left(\left(\beta_0 + \beta_1 \left(x_{obs}^{(i)} \right) \right) - y_{obs}^{(i)} \right)^2$$

Activation ~~Cost~~ Functions

Sigmoid	0-1, vanishing gradient issues
tanh	-1-1, vanishing gradient issues
ReLU	capture large effects, no vanishing gradient
Leaky ReLU	ReLU, but allows negative outcomes

Backprop.

Network: $F: X \to Y$

Loss: $J(y, F(x))$

chain rule

Get $\frac{\partial J}{\partial w_k}$ for every weight in network

↳ $\frac{\partial J}{\partial w^{(3)}} = (\hat{y} - y) \cdot a^{(2)}$

$\frac{\partial J}{\partial w^{(2)}} = (\hat{y} - y) \cdot w^{(3)} \cdot \sigma'(z^{(3)}) \cdot a^{(1)}$

$\frac{\partial J}{\partial (w^{(1)})} = (\hat{y} - y) \cdot w^{(3)} \cdot \sigma'(z^{(3)}) \cdot w^{(2)} \cdot \sigma'(z^{(2)}) \cdot X$

Vanishing Gradients

$\sigma'(z) = \sigma^2(z)(1 - \sigma(z)) \leq .25$

↳ as more layers are added, gradient gets very small @ earlier layers

Activation Functions

$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{2x} - 1}{e^{2x} + 1}$

$\tanh(0) = 0$

$\tanh(\infty) = 1$

$\tanh(-\infty) = -1$

$ReLU(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases} = \max(0, z)$

$LReLU(z) = \begin{cases} \alpha z, & z < 0 \\ z, & z \geq 0 \end{cases}$ for $\alpha < 1 = \max(\alpha z, z)$

NN Optimizers

- variants to updating weights
  ↳ optimizers
- momentum - smooth out path, less fluctuation
- running avg. of step directions
  ↳ $v_t := \eta \cdot v_{t-1} - \alpha \cdot \nabla J$
  $W := W - v_t$

η is momentum
↳ how much past values influence next value

Nomenclature:
$\eta = \beta$
$\alpha = 1 - \beta$

Nesterov Momentum
- control overshooting by looking ahead
- apply gradient only to the non-momentum component
  ↳ $v_t = \eta \cdot v_{t-1} - \alpha \cdot \nabla (J - \eta \cdot v_{t-1})$
  $W := W - v_t$
- move even more smoothly to optimal value

AdaGrad - Adaptive Gradient Algorithm
- scale update for each weight separately
  ↳ update frequently updated weights less
  ↳ keep running sum of previous updates
  ↳ divide new updates by factor of prev. sum

with starting point $G_i(0) = 0$:

$$G_i(t) = G_i(t-1) + \left(\frac{\partial L}{\partial W_i}(t)\right)^2$$ → G will continue to increase

$$W := W - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla J$$ → smaller updates each iteration

RMSProp - Root Mean Square Propagation
- similar to AdaGrad
- rather than using sum of prev. gradients, decay older gradients more than more recent ones
- more adaptive to recent updates

Adam
- use both first & second order change info and decay both over time

by default: $\beta_1 = .9 \quad \beta_2 = .999$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla J \qquad v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla J$$

bias correction:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$W := W - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \odot \hat{m}_t$$

Batching Terminology
- Full-Batch - use entire dataset
- Mini-Batch - use smaller portion of data (usually 16, 32)
- Stochastic - use a single example
- Epoch - single pass through all training data

Data Shuffling
- shuffle data after every epoch to avoid cyclic movement

SCALE INPUTS TO AID CONVERGENCE

Linear Scaling:

$[0, 1] \quad x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$

$[-1, 1] \quad x_i = 2\left(\frac{x_i - x_{min}}{x_{max} - x_{min}}\right) - 1$

Convolutional Neural Networks

Multiclass Classification

- sigmoid → softmax
- $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$
- loss: categorical cross-entropy

$$C.E. = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

$$\frac{\partial CE}{\partial \text{softmax}} \cdot \frac{\partial \text{softmax}}{\partial z_i}$$

CNNs

- traditional NNs treat all inputs interchangeably
- want to incorporate domain knowledge
- Important structures in image data:
  - "Topology" of pixels
  - Translation invariance
  - Lighting & contrast
  - knowledge of human visual data
  - pixels tend to have similar values
  - edges & shapes
  - scale invariance

Kernels

- grid of weights overlaid on image, centered on 1 pixel
  - ↳ each weight multiplied by pixel
    - ↳ $\sum_{p=1}^{P} w_p \cdot \text{pixel}_p$ ← convolution function
      - ↑ output over centered pixel
- "local feature detectors"
- NN learns which kernels are most useful

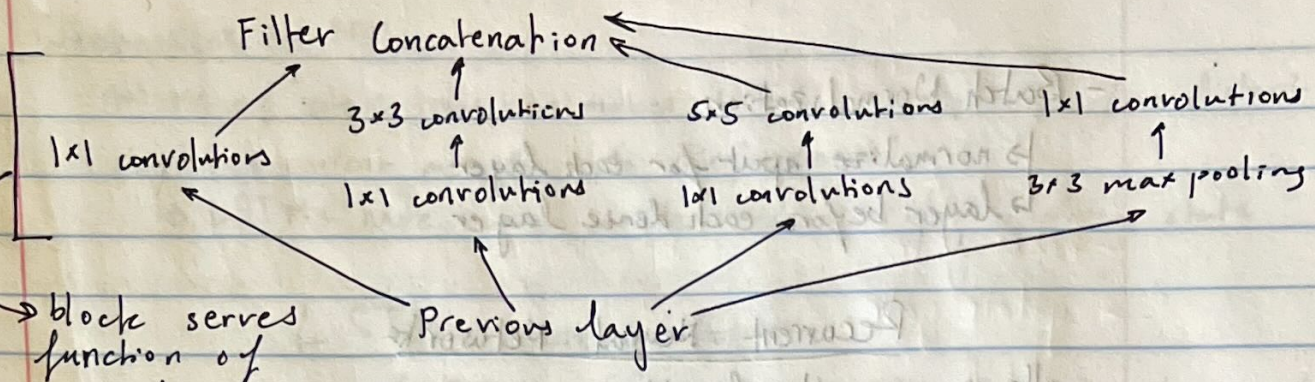
- Grid size - # of pixels kernel "sees"
  - ↳ typically use odd numbers
- padding
  - ↳ pixels near edge not used as center pixels
  - ↳ adds extra pixels around edge to fix edge issue
  - ↳ zero-padding
- stride
  - ↳ "step size" as kernel moves across image
  - ↳ when > 1, scales down output dim.
- Depth / Channels - multiple #s associated w/ each pixel location
  - kernel depth = image depth
  - output depth
  - # kernels in layer = output depth (has)
- Pooling - reduce image size by mapping patch of pixels to single value
  - ↳ Max-Pooling - take max val. in patch
  - ↳ Average-Pooling - take avg. val of patch ***

Transfer Learning

- earlier layers are slowest to train
- in CNNs, early layers represent primitive features (general)
- keep early layers of pre-trained network, re-train later layers for specific application
  - ↳ Fine-Tuning
- Guiding Principles for Fine-Tuning
  - more similar task ⇒ less fine tuning
  - more data available ⇒ more fine tuning
  - if data is very different, very little value in transfer learning

CNN Architectures

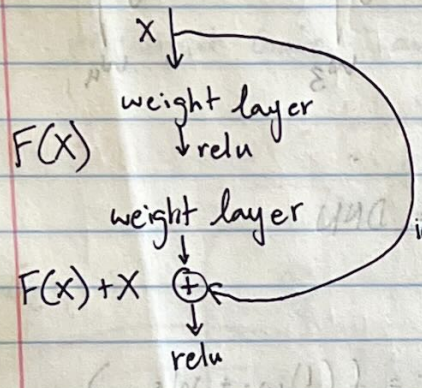
- LeNet
  - built for MNIST (32×32)
  1. Kernel 6@28×28
  2. Pooling 6@14×14
  3. Kernel 16@10×10
  4. Pooling 16@5×5
  5. Flatten
  6. MLP - output 10
- AlexNet
  - "flash point" for modern deep learning
  - data augmentation for training - cropping, flipping
  - convolutions w/ ReLUs
  - sometimes add maxpool after convolution
  - fully connected layers before softmax
- Inception
  - turn each layer into branches of convolutions
  - each branch handles smaller portion of workload
  - concatenate branches @ end



block serves function of prev. layer

convolutional

- Res Net
  - deeper networks performing worse
    - ↳ early layers of networks are slow to adjust
    - ↳ in theory, should be able to have "identity" transformation



X → input to series of layers

F(X) → function represented by several layers (such as convolutions)

← enforce "best transformation" by adding "shortcut connections"

- avoids vanishing gradient issue
- if weights of later layers = 0 ⇒ identity transformation

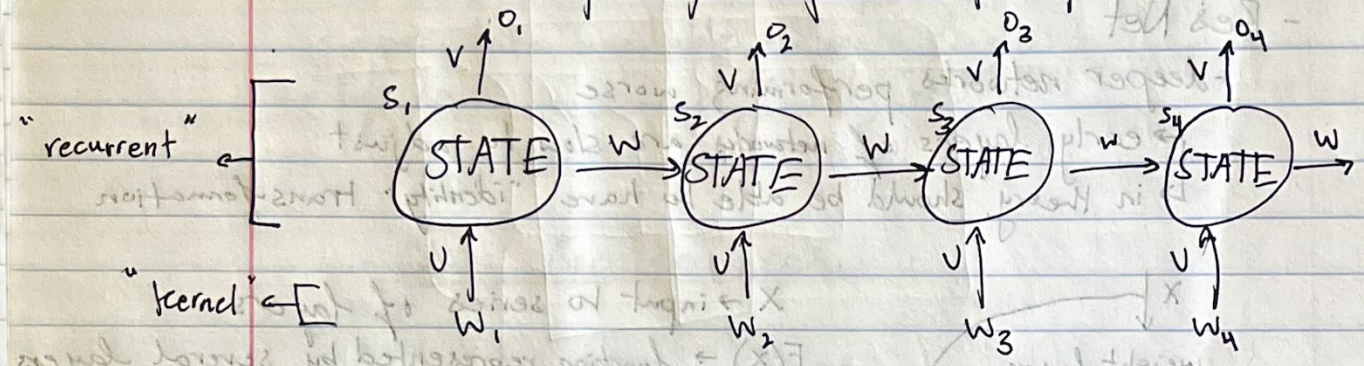
Regularization Techniques

- Penalized loss function (penalty for higher weights)
  - ↳ $J = \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{m} w_i^2$
- Dropout - at each batch, randomly remove a subset of neurons
  - ↳ prevents over-reliance on individual pathways
  - ↳ "rescale" weight of neuron to reflect % of time active
    - ↳ p = probability of neuron being present ⇒ $w \to pw$
- Early Stopping
  - ↳ choose some rules after which to stop training

- Batch Normalization
  - ↳ normalize input for each layer
  - ↳ layer before each dense layer

Recurrent Neural Networks

- allows variable length input
- each word is input separately & NN outputs prediction & state



- often only use last output
  - ↳ often passed into ~~dense~~ DNN

Mathematical Details:

w_i is word @ position i
s_i is state @ position i
o_i is output @ position i
r = dimension of input vector
s = dimension of hidden state
t = dimension of output vector (after dense layer)

$s_i = f(Uw_i + Ws_{i-1})$ (core RNN)
$o_i = \text{softmax}(Vs_i)$ (subsequent dense layer)

- U is an $s \times r$ matrix
- W is an $s \times s$ matrix
- V is a $t \times s$ matrix
- ↳ same across all positions

Practical Details:

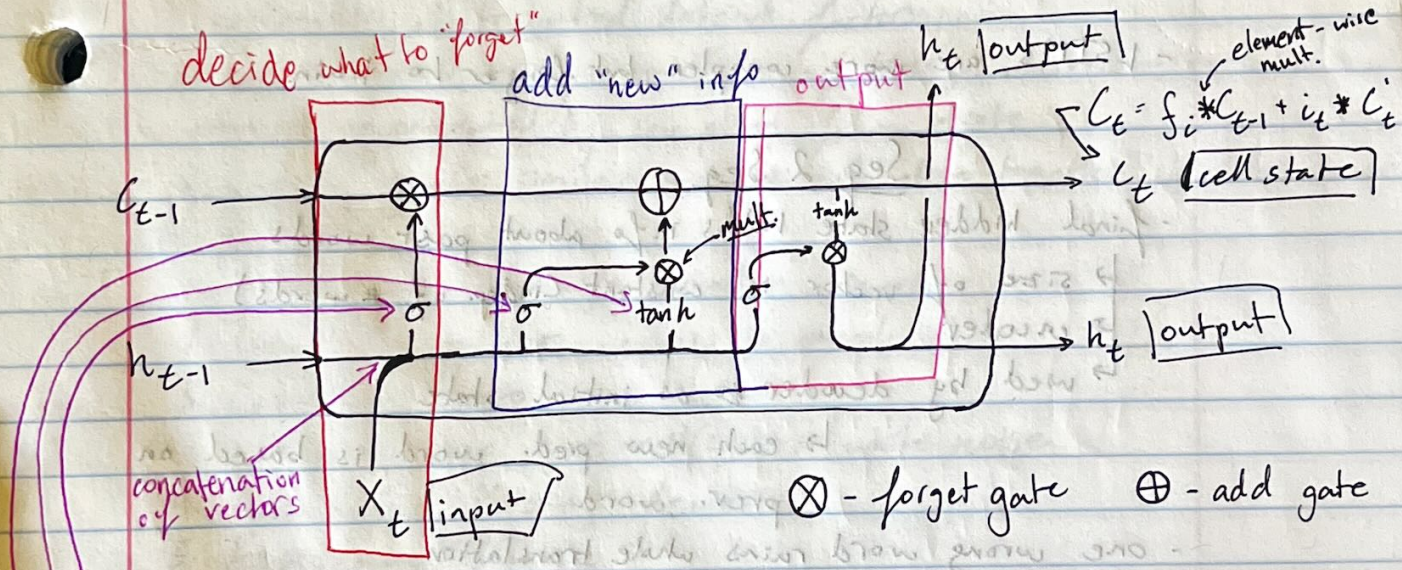
- train only on final output
  - ↳ use backprop through time (BPTT)
  - ↳ sensitive to length of sequence
    - ↳ padding / truncation

Weaknesses of RNNs:

- nature of state transition makes it hard to keep info from distant past in current memory w/o reinforcement
  - ↳ LSTMs have more complex mechanism for updating state

LSTMs ~~and GRUs~~

- improve memory of RNN by using more complex update mechanism for state
  - ↳ items are overwritten as an active choice
  - ↳ adds an explicit "memory" unit
- augment RNNs w/ Gate Units
  - ↳ control how long/if event stays in memory
  - ↳ Input Gate - causes items to be stored in memory
  - ↳ Forget Gate - causes items to be removed from memory
  - ↳ Output Gate - causes hidden unit to feed forward (output) in network



- cell state is updated in 2 stages

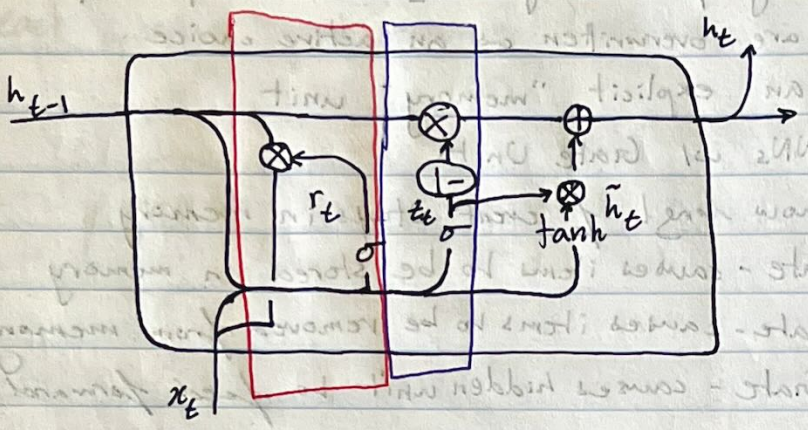
$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \rightarrow \text{what portion of new info to add}$$

$$C'_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \rightarrow \text{new info}$$

GRUs

- removed cell state
- past info is now used to transfer past info
- reset gate - decides how much past info to forget
- update gate - decides what info to discard & what new info to keep



- LSTMs are more complex, but longer to train

Seq 2 Seq

- final hidden state holds info about past words
  - ↳ size of vector is constant (indep. of # words)
  - ↳ encoder
  - ↳ used by decoder as initial state
    - ↳ each new pred. word is based on prev. word
- one wrong word ruins whole translation
  - ↳ beam search - generate many sequences, then determine most probable
- attention
  - ↳ focus on relevant parts of encoder state depending on decoder timestep
  - ↳ $s(i,j)$ is similarity measure between decoder state i & encoder state j
    - ↳ represent each word as vector
    - ↳ weight diff embedding layer hidden states

Autoencoders

- PCA problem: learned features are linear combination of original features

Steps:
1. Feed image through encoder network
2. Lower-dimensional embedding generated
3. Embedding fed through decoder network
4. Reconstructed version of original data generated
5. Compare result to original, compute loss & train network

Result: network will learn lower dimensional space representing original data

Variational Autoencoders

- data are assumed to be represented by a set of normally-distributed latent factors
- encoder generates parameters of these distributions (μ & σ)
- images generated by sampling from distrs.
- secondary goal: similar images are close together in latent space

Steps:
1. Feed image through encoder network
2. Generate μ & σ vectors to encode data
3. Combine into one vector
   - ↳ $\mu + \sigma * \epsilon$ $\quad \epsilon \sim N(0,1)$ ← white noise
4. Vector fed through decoder
5. Reproduce image
6. Train (Loss)

Loss Function:
2 components: penalty for not reconstructing properly, penalty if μ_i, σ_i are diff than 0,1
- ↳ use KL divergence
  - ↳ σ vector interpreted as $\log(\sigma)$
  - ↳ $\frac{1}{2} * (e^{\log(\sigma)} - (\log(\sigma) + 1) + (\mu^2))$

GANs

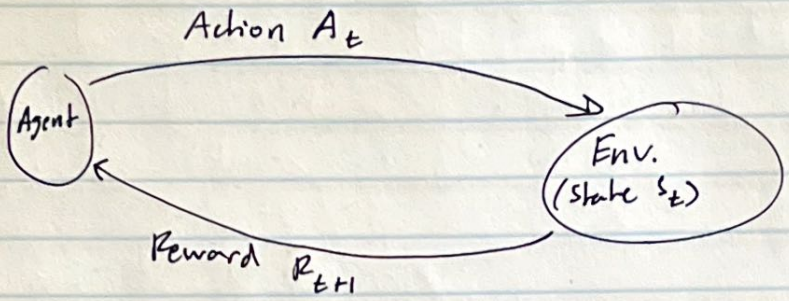
- Adversarial examples - generated by repeatedly adding a function of $\frac{\partial L}{\partial x}$ to the input image
- train two NN simultaneously
  - Generator - maps random noise to images indistinguishable from those in training set
  - Discriminator - determines real vs. fake
- Training Procedure:
  1. Randomly initialize weights of generator & discriminator
  2. Randomly initialize noise vector & generate image using generator
  3. Predict ~~probabi~~ probability that generated image is real using discriminator
  4. Compute losses both assuming
     i Image was fake, comparing $p_{real}^{x_G}$ to 0 $\rightarrow L_0^{x_G}$
     ii Image was real, comparing $p_{real}^{x_G}$ to 1 $\rightarrow L_1^{x_G}$
  5. Use $L_0^{x_G}$ to train discriminator
  6. Compute $\frac{\partial L_1^{x_G}}{\partial x_G}$ based on $L_1^{x_G}$ - do not use to train discriminator
  7. Use $\frac{\partial L_1^{x_G}}{\partial x_G}$ to train generator
  8. Use discriminator to calculate probability that real image x_R is real
  9. Compute $L_1^{x_R}$
  10. Use $L_1^{x_R}$ to train discriminator
- highly dependent on discriminator & generator learning @ same rate
  - affected by architectures, learning rates, loss functions, optimization techniques

LIMEs - Locally-Interpretable Model-Agnostic Explanations

- focus on sensitivity of outputs to changes in inputs

Reinforcement Learning

- agents interact w/ environment
  - ↳ choose from a set of available actions
    - ↳ impact environment, which impacts agents via rewards
      - ↳ rewards are generally unknown & must be estimated by agent
- process repeats dynamically, so agents learn how to estimate rewards over time



- Solution represents a policy by which agents choose actions in response to the state
  - ↳ maximize expected rewards over time