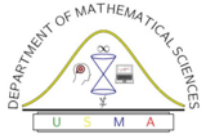
Aerial Recon of ~~Introduction to~~ Machine Learning

USMA D/MATH
CPT Steve Morse

December 6, 2018



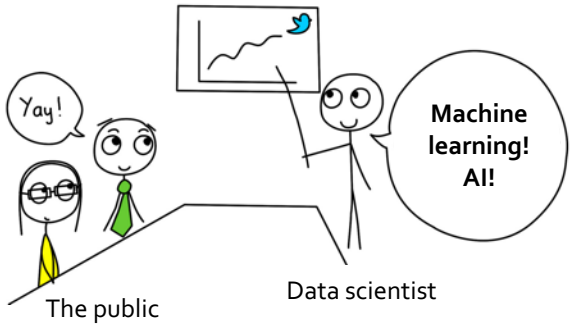
Agenda

- Background — *Why machine learning? Why Python? And um … isn't this just statistics?*
- The 10,000 ft view — *Building the black box, some terminology, a typical pipeline*
- Fly-over of some models:
  - Supervised — *Least squares is cool again, logistic regression, boosting/bagging, support vector machines*
  - Deep Learning — *(Deep) neural networks of the plain, convolutional, and recurrent variety*
  - Unsupervised — *Dimension reduction, clustering*
- Other topics — *First order methods, RKHS's, oh hello Bayes I didn't see you there*
- Closing thoughts — *What do the haters say, what's next, and how do I start?*

Why machine learning?

- It's famous (nb: mostly versions of deep neural nets)
  - **World champs** at chess, Go, Doom, … still working on Warcraft …
  - **Computer vision:** StreetView, self-driving cars (Tesla, Uber)
  - Voice, image recognition (FB's DeepFace 97.4% accurate)
  - **Translation** (Microsoft, Google Translate approach human acc.)
  - **Expert systems:** healthcare, finance

- Operations Research wants to keep a foothold in "Analytics"



- Most importantly, it's fun / exciting / cool / mathy / code-y

Google Intern :

```python
from keras.layers import *
from keras.models import *
from .data import load_data

x, y, x_test, y_test = load_data()

def get_model(num_layers):
    model = Sequential()
    for _ in range(num_layers):
        model.add(Dense(100, activation='sigmoid'))
    model.compile(loss='mse', optimizer='sgd')
    return model

best_model = None
best_loss = None

for i in range(1, 10):
    model = get_model(i)
    model.fit(x, y)
    loss = model.evaluate(x_test, y_test)
    if best_loss is None or loss < best_loss:
        best_loss = loss
        best_model = model
```

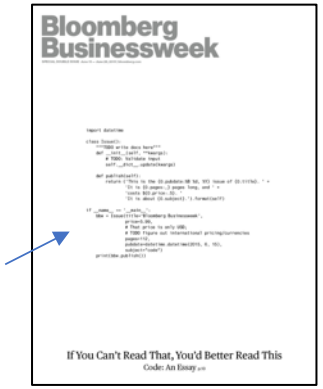
Media :

Why Python?

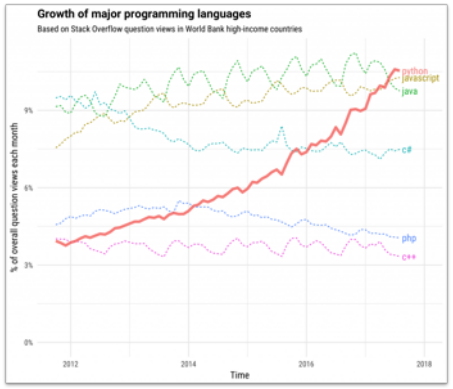
Alternate title: "I just learned R. Now why should I learn Python?"

- **Reason #1:** Python is the most popular programming language in the world

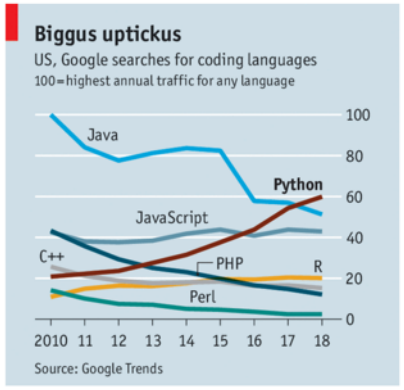
Python code!



"What is Code" Bloomberg cover, June 2015



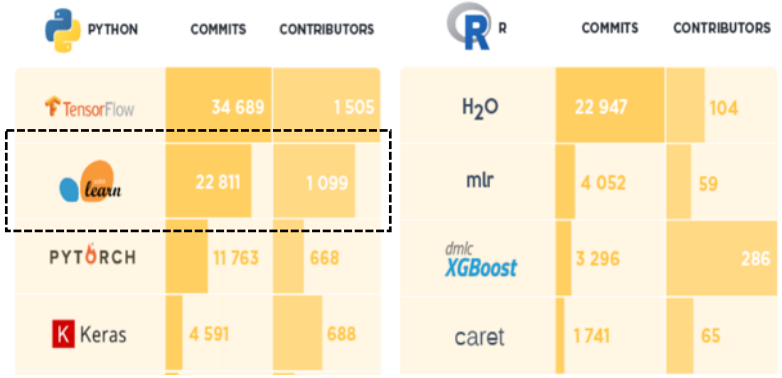
Python top viewed on Stack Overflow, Jun 2018



"Python has brought computer programming to a vast new audience" Economist, Jul 2018

- Yeah, but … `ggplot2` is still the king of visualization ¯_(ツ)_/¯
  - Python contenders: `matplotlib`, `Seaborn`, `pandas`, `ggpy`, `plotnine`, `Altair`
  - "All happy Python programmers are the same. Each unhappy Python programmer has his/her own plotting library." – *Anna Karenina*, Leo Tolstoy

- **Reason #2:** `sklearn` is biggest all-in-one open-source ML library



As of June 2018, source: activewizards.com

sklearn
- caret
- rpart
- randomForest
- e1071
- mclust
- …

- Rays of hope: (1) R has very good ML libraries if you prefer it, (2) `sklearn` is beautifully easy even for non-native speakers, and (3) using `pandas` feels like R

Statistics, machine learning … what's the diff?

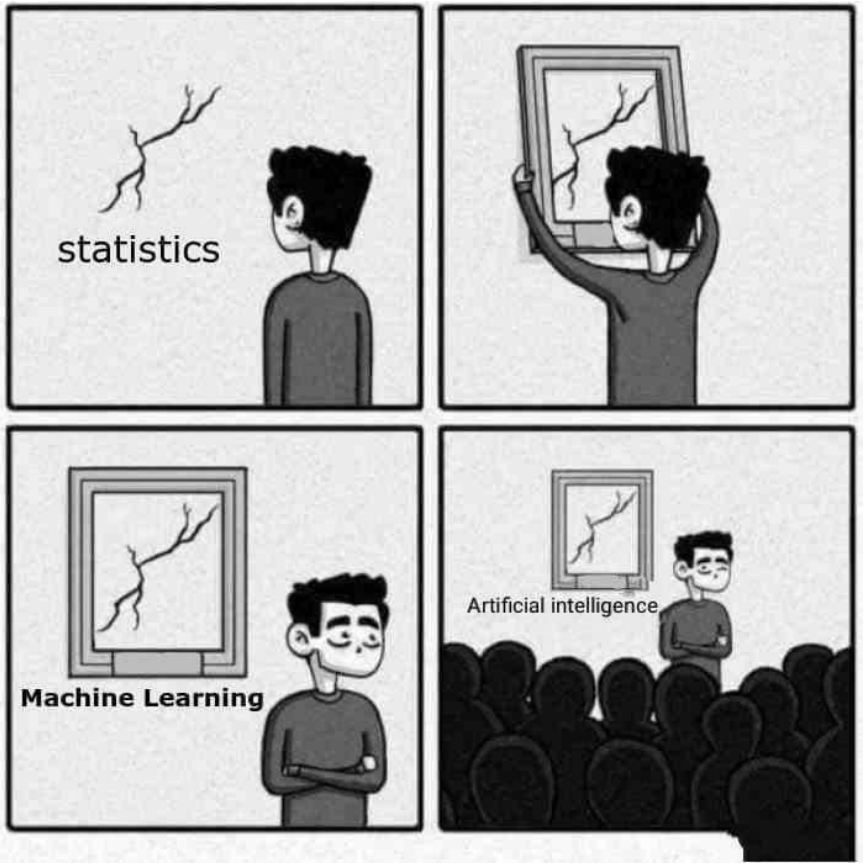
- Maybe it's … inference vs. prediction
  - "Statistics cares about $\hat{\beta}$ … machine learning cares about $\hat{y}$"
    – *Machine Learning: An Applied Econometric Approach*, J. Econ Persp., 2017
  - "Machine learning is statistics minus any checking of models or assumptions"
    – stack overflow commenter, 2017

- Maybe it's … institutional differences
  - Stats : Math :: ML : CS
  - Stats -> journals. ML -> conferences!

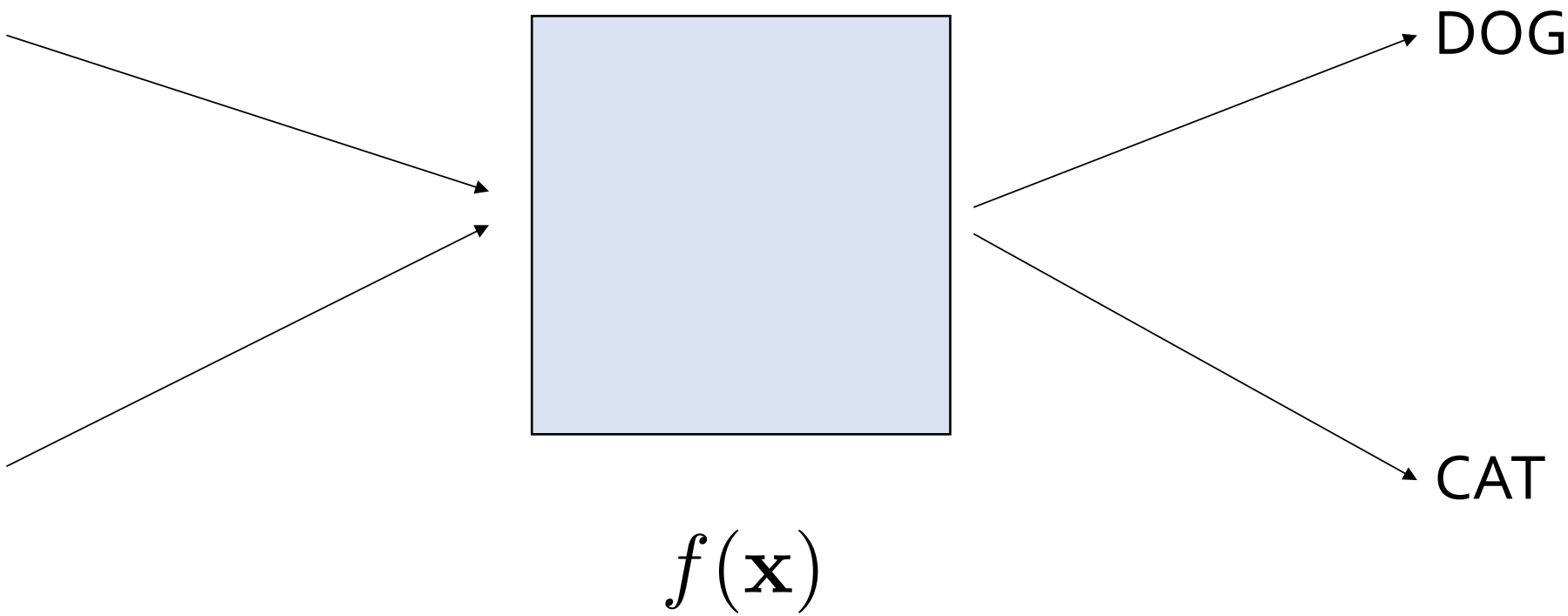
- Maybe it's … not that much really!

Statistics	Machine Learning!
Fitting	Learning
Test set performance	Generalization
Regression/classification	Supervised learning
Large grant = $50,000	Large grant = $1,000,000
Nice place for a meeting: Las Vegas in August	Nice place for a meeting: Snowbird, Utah, French Alps

Works cited: Robert Tibshirani



The 10,000 ft view



$$f(\mathbf{x})$$

Wouldn't it be nice to know this function?

The 10,000 ft view

Parameters

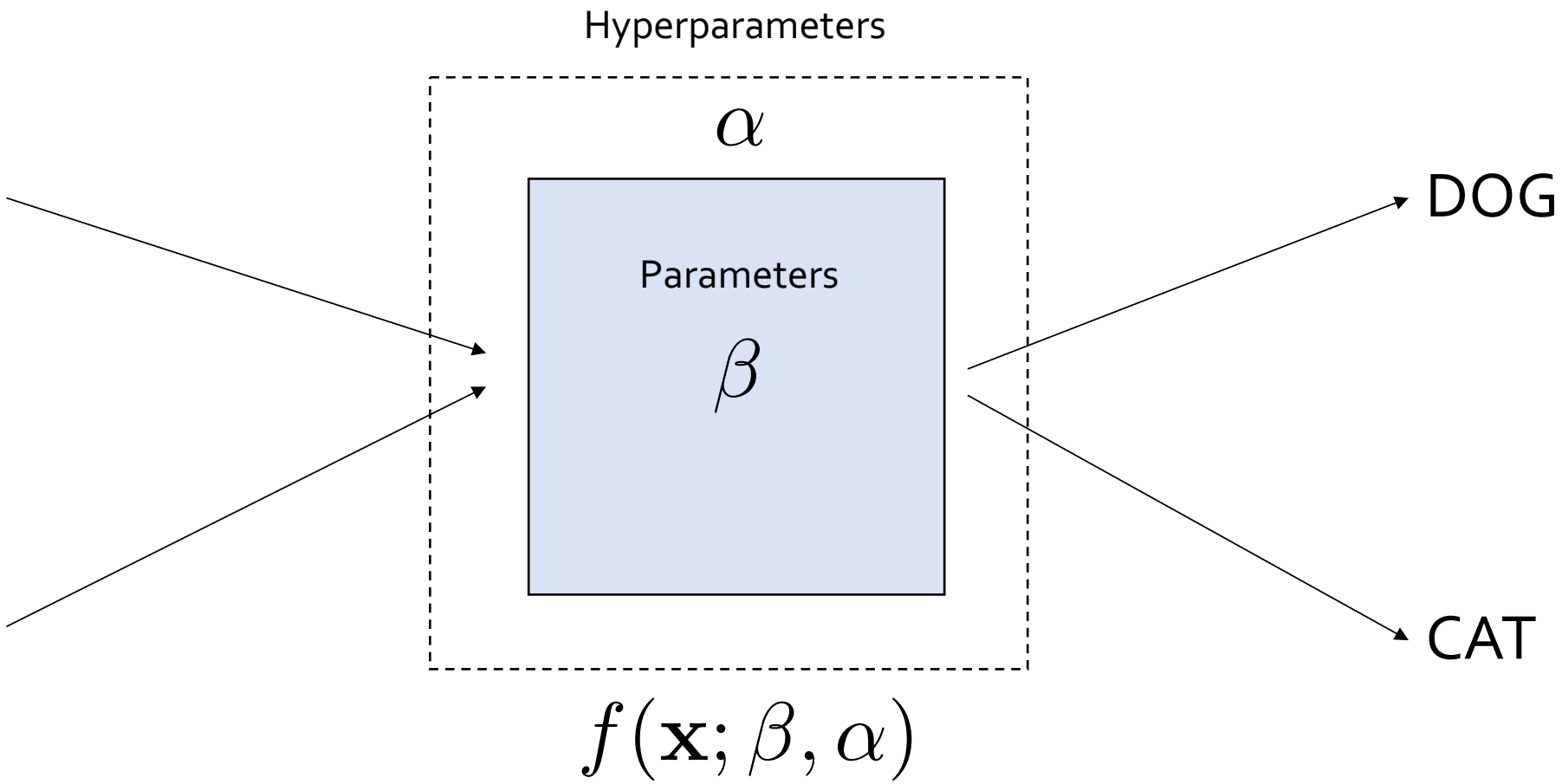
β

DOG

CAT

$f(\mathbf{x};\beta)$

The 10,000 ft view



The 10,000 ft view

Feature engineering

Hyperparameters

α

$\phi(\mathbf{x})$

Parameters

β

$f(\phi(\mathbf{x}); \beta, \alpha)$

DOG

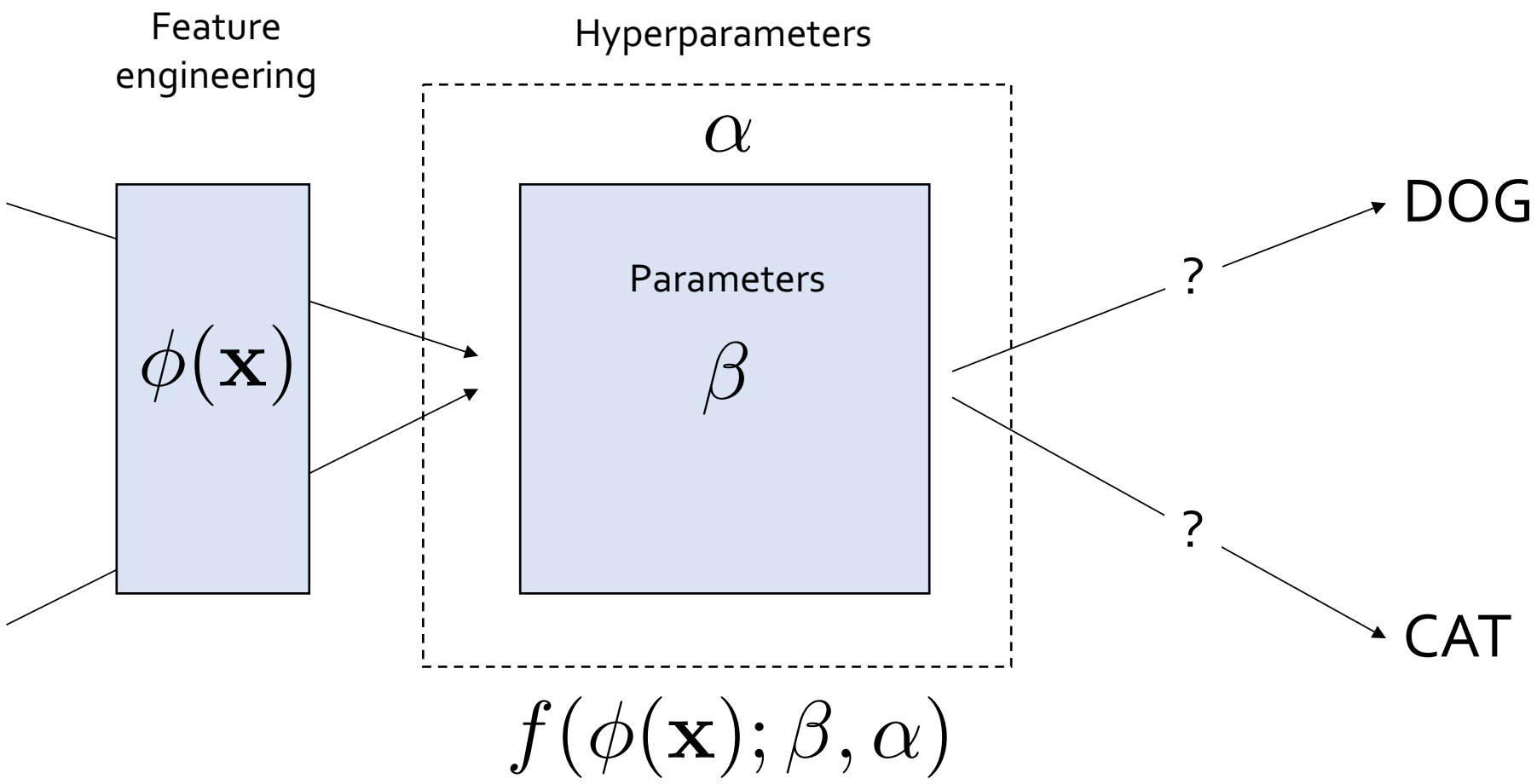
CAT

$$\min \sum_i L\left(y^{(i)}, f(\phi(\mathbf{x}^{(i)}); \beta, \alpha)\right) + \lambda||\beta||_p$$

Loss /
Empirical risk /
Negative log-likelihood

Penalty on complicated model
("regularization")

The 10,000 ft view



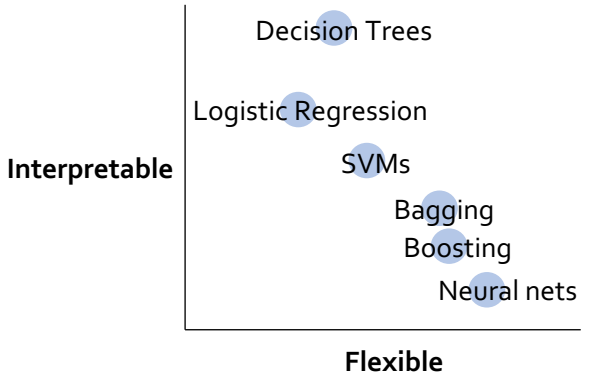
An assortment of terminology and themes

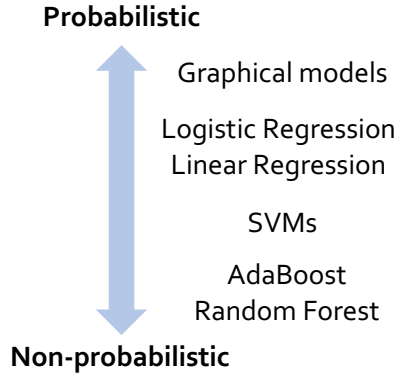
- Supervised learning $\mathcal{D} = \{(x_i, y_i)\}$
  - Labeled data
  - Regression ($y_i \in \mathbb{R}$) vs. classification ($y_i \in S$)
  - Examples: Generalized linear models, support vector machines (SVMs), boosted methods, bagging (Random Forests), neural networks, …

- Unsupervised learning $\mathcal{D} = \{x_i\}$
  - Unlabeled data
  - Clustering: k-means, hierarchical, spectral, …
  - Density estimation: GPs, DPs, …
  - Dimensionality reduction: PCA, SOM, topic models, …

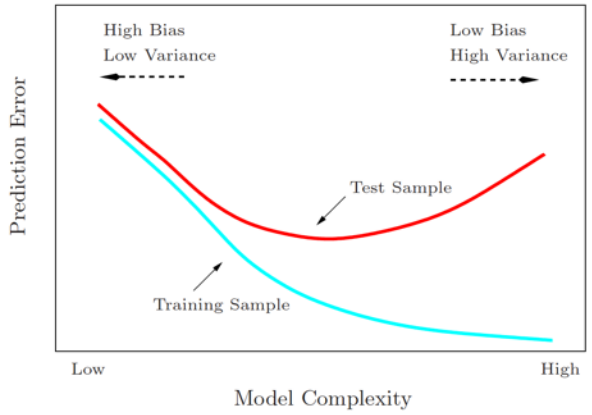
- Prediction rules vs. probabilistic (MLE / Bayesian)

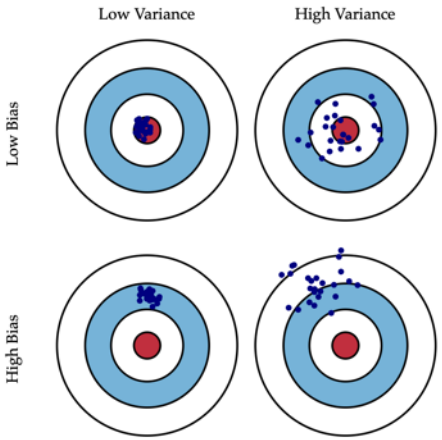
- Model selection
  - Feature engineering
  - Cross-validation
  - Bias-variance trade-off





Bias / variance tradeoff





$$\text{err}(x) = \mathbb{E}\left[(Y - \hat{f}(x))^2\right] = \underbrace{\left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2}_{\text{Bias}} + \underbrace{\mathbb{E}\left[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2\right]}_{\text{Variance}} + \sigma^2$$

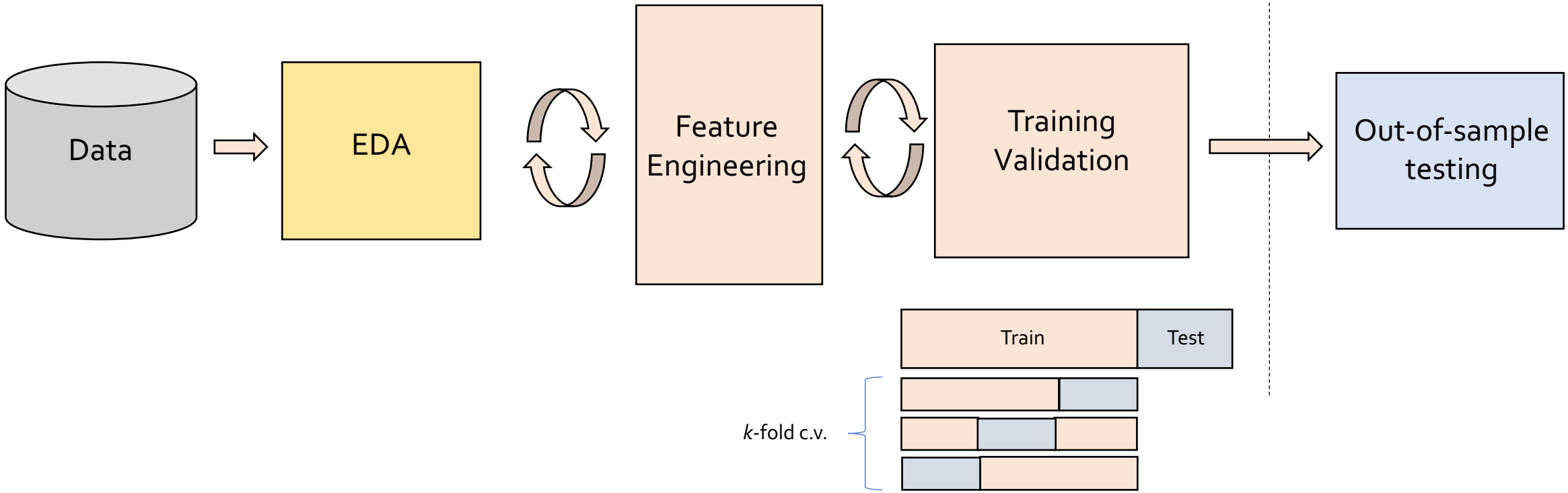
Works cited: ESL, Bishop

A machine learning pipeline

Exploratory Data Analysis
- Histograms
- Scatterplots
- Pattern finding

Feature Engineering
- By-hand or automated
- Transformations / kernels
- Typically requires domain knowledge
- Related terms: feature selection, feature learning

Model selection
- Training data
- Cross-validation selects model which does well out-of-sample in expectation



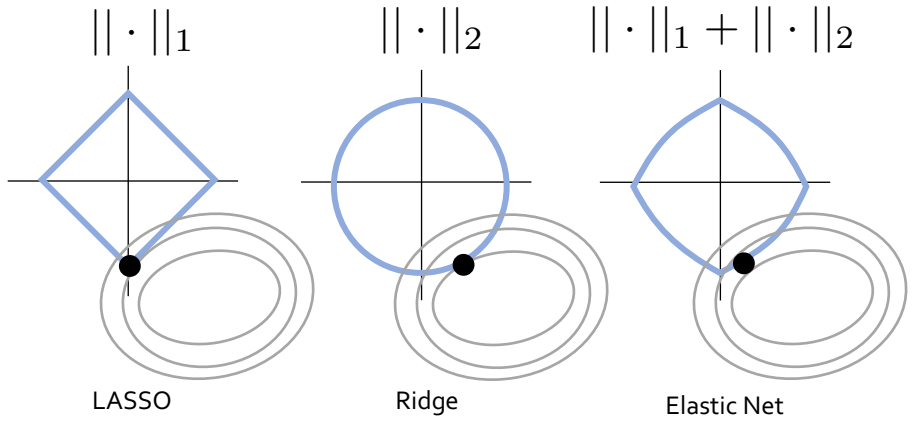
Case Study: Making least squares cool again

- Old school + slap some regularization on there!

$$\arg\min_{w_0,\mathbf{w}} \sum_{i=1}^{N}(y_i - w_0 - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda||\mathbf{w}||_q$$

SSE

Penalty on big **w** ("regularization")



- Maximum likelihood estimation (MLE)

Assume a linear model with Gaussian error

$$y = \mathbf{w}^T\mathbf{x} + \varepsilon$$

Implies *y* is normally dist., given an *x*

$$Y|X \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$$

Likelihood ("how likely is it these parameters created this data?")

$$\mathcal{L}(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^{N} \Pr(y_i|\mathbf{x}_i, \mathbf{w})$$

$$\arg\min_{\mathbf{w}} -\log \mathcal{L}(\mathbf{w})$$

math!

Identical to least squares!*

Themes:
- Minimizing error/NLL + penalty
- Regularization
- Re-framing prediction-rule paradigms (e.g. squared error) in probabilistic terms (e.g. MLE)

- Let's get (a little bit) Bayesian

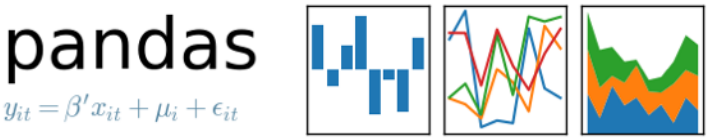
$$\mathbf{w}_{\text{MAP}} \stackrel{\text{def}}{=} \arg\max \Pr(\mathbf{w}|\mathcal{D}) = \arg\max \frac{\Pr(\mathcal{D}|\mathbf{w})\Pr(\mathbf{w})}{\Pr(\mathcal{D})}$$
$$= \arg\max \Pr(\mathcal{D}|\mathbf{w})\Pr(\mathbf{w})$$
$$= \arg\max \log \mathcal{L}(\mathbf{w}) + \log \Pr(\mathbf{w})$$

- Uniform prior = OLS
- Gaussian prior = Ridge

Least squares again … with regularization!

* Gauss invented Least Squares. Gauss invented the normal distribution. Coincidence???

Code break!



Logistic Regression and deeper waters

- Idea: assume a "sigmoid" form for the probability of output given a linear combination of the input.

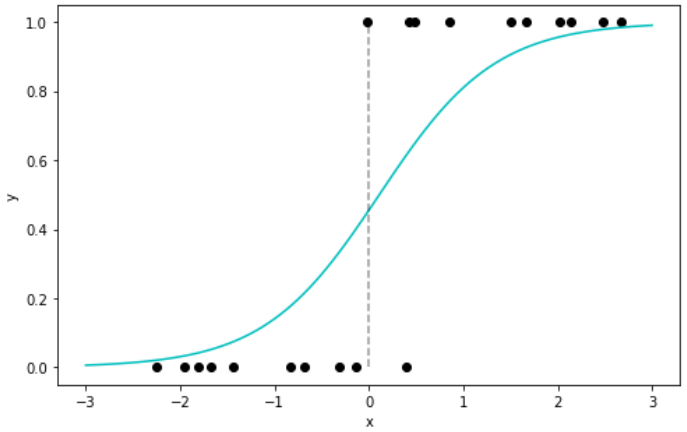
- Math (one of many versions) (binary classification case):

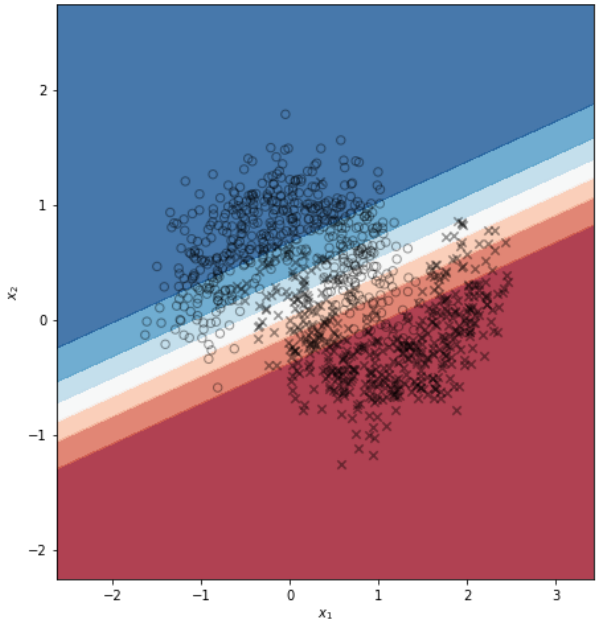
logistic function ("sigmoid") $$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

conditional probability $$P(Y = 1|X = \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$$

likelihood of data $$\mathcal{L}(\mathcal{D}|\mathbf{w}) = \prod_i \sigma(\mathbf{w}^T\mathbf{x}_i)^{y_i}(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))^{(1-y_i)}$$

MLE (look familiar?) $$\text{argmin } \text{NLL}(\mathbf{w}) + \lambda||\mathbf{w}||_q$$





- Pros/Cons: powerful, classical, simple model, will often outperform more exotic models when data has good features and small size

- Slightly deeper statistical waters:
  - Note that linear regression and logistic regression assumed specific forms for the conditional distribution Y | X. (Gaussian for OLS, Bernoulli for logistic.)
  - Generalizing this idea to other forms (from the exponential family) gives rise to *generalized linear models*.

Support Vector Machines (SVM)

- Idea: place a separating hyperplane that maximizes separation between different classes

- History: origins in 1963, but the "kernel trick" and "soft-margin" versions launched it into popularity in mid-90's into 2000's.

- Math:

$$\min \quad ||\mathbf{w}||^2 + C\sum_i \zeta_i$$

$$\text{s.t.} \quad y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \zeta_i$$

$$\zeta_i \geq 0$$

$$\min \quad ||\mathbf{w}||^2 + C\sum_i \max\left\{0, 1 - y^{(i)} f(\mathbf{x}^{(i)})\right\}$$

Penalty on big **w** ("regularization")

Hinge loss

- The "kernel trick"

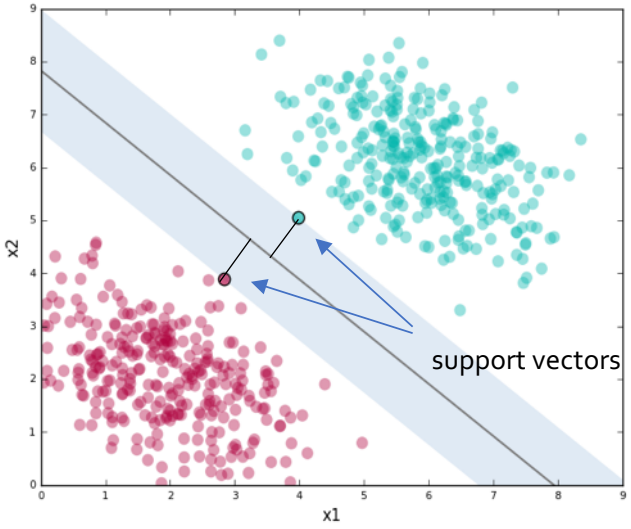
Computing the classifier (need phi):

$$\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)$$

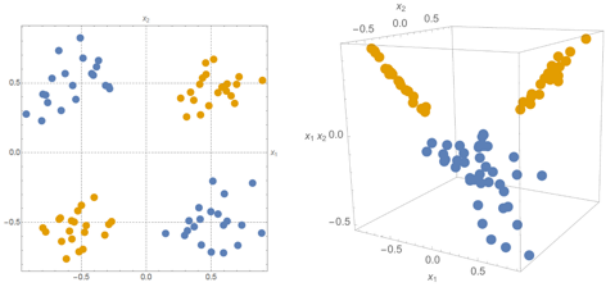
Computing the classification (just need kernel):

$$\text{sgn}(\mathbf{w}^T\phi(\mathbf{x}) + b) = \text{sgn}(\sum_i \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b)$$

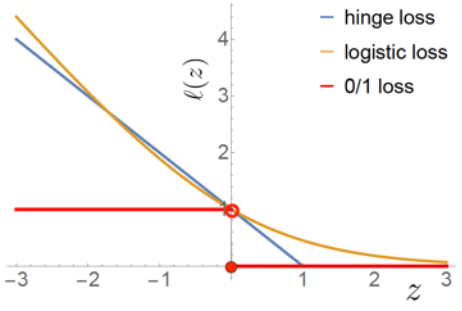
- Pros/Cons: powerful, flexible classifier that works even with small/medium amounts of data, but requires more feature engineering (preprocessing, selection of kernel) than methods like boosting or NNs



A kernel transformation on XOR data:



Logistic regression is like a smoothed version of SVM:



Boosting, Bagging, and Beating Navy

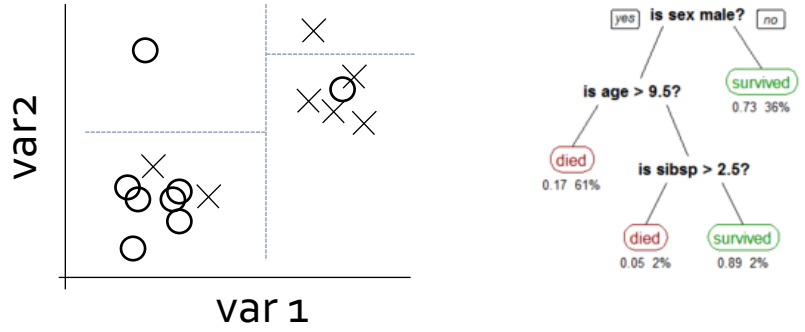
- Idea: lots of "weak learners" make a strong learner

- Decision Trees / CART
  - Greedily select partitions of the dataspace
  - Grow a large tree, then prune
  - *Pros:* interpretable, fast, handles nonlinearities
  - *Cons:* tends to overfit, not robust, usually a weak predictor

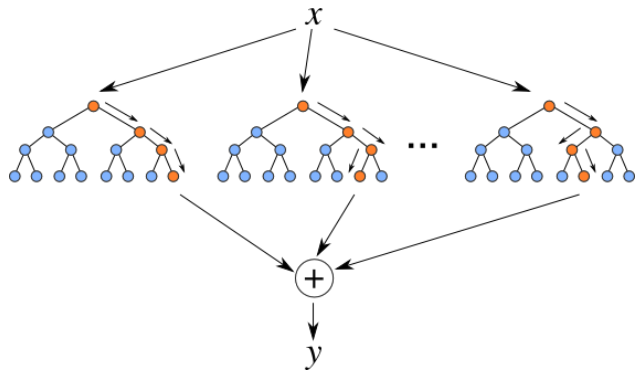
- (Gradient) Boosting – example: AdaBoost
  - Fit a series of weak learners (possibly 1-split DTs – "decision stumps"!)
  - Reweight areas that are harder to classify
  - *Pros:* great out-of-the-box model, fast
  - *Cons:* sensitive to outliers / mislabeled points

- Bootstrap aggregation ("bagging") – example: Random Forests
  - Randomly select different parts of the data (bootstrap)
  - Grow a decision tree on each one
  - Take a weighted average of the "vote" of each DT (aggregate)
  - *Pros:* powerful predictor, esp. with nonlinearities
  - *Cons:* not interpretable, slow

Decision Tree



Random Forest



Neural nets

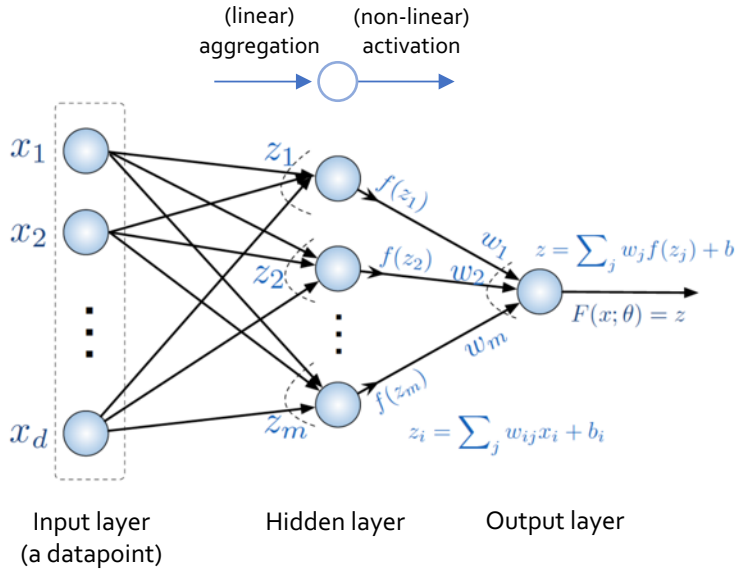
- Idea: choosing the right parametric form is hard, so let's have a layered, non-linear model and we can capture anything!

- History:
  - Pre-1990: "Perceptron" (1958), back-prop (1970-80), power of multilayer (1989)
  - 1990's: Variants (CNN, RNN, RL) and fixes (LSTM, dropout)
  - Now: (2004-present) big data + faster tech (GPUs) + various hacks = **deep learning**

- Pros/Cons: powerful, flexible, requires little-to-no FE, but not well-suited for small datasets, not interpretable

- Why do NNs work so well?
  - Maybe: SGD tends to find "flat" optimums, which generalize better (https://www.inference.vc/everything-that-works-works-because-its-bayesian-2/)

Feed-forward neural network architecture:



Back-propagation:

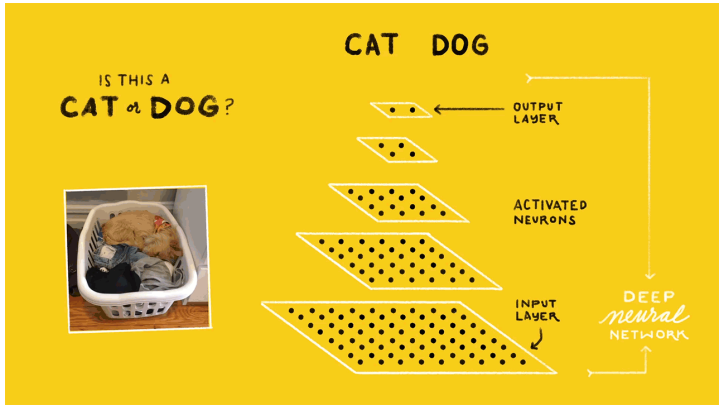
$$\frac{\partial \ell(y,z)}{\partial w_{ij}} = \left[\frac{\partial z_j}{\partial w_{ij}}\right]\left[\frac{\partial f(z_j)}{\partial z_j}\right]\left[\frac{\partial z}{\partial f(z_j)}\right]\left[\frac{\partial \ell}{\partial z}\right]$$

Code break!

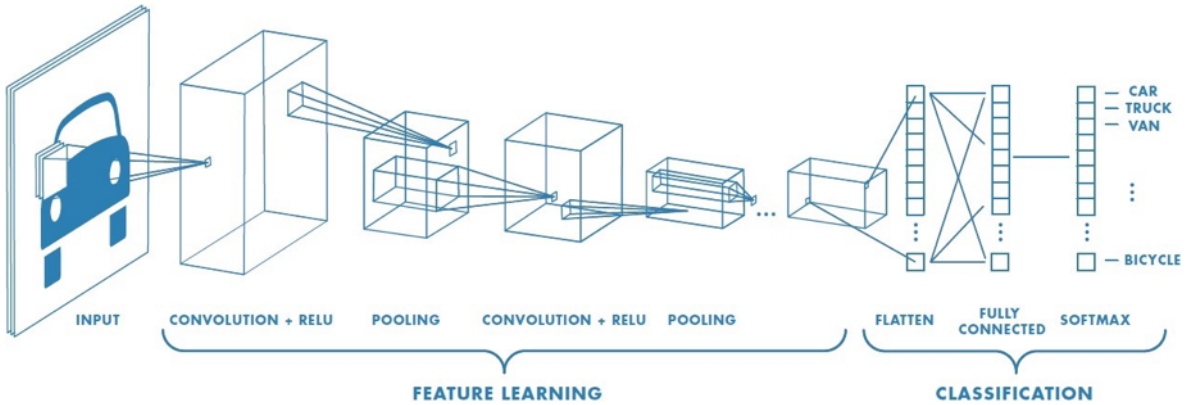
Convolutional Neural Nets (CNNs)

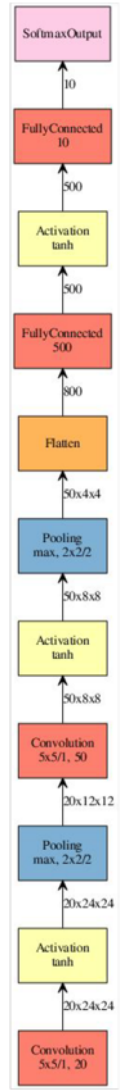
- Idea: there is local structure within each input (for example: an image)

- Math:
  - Convolutional layers
    - Weight sharing = structure
    - Fast matrix multiplication
  - Pooling layers
    - Max / averaging



- Typical application is "computer vision" (image recognition)





LeNet
1998

AlexNet
2012

GoogLeNet
2014

Inception
2015

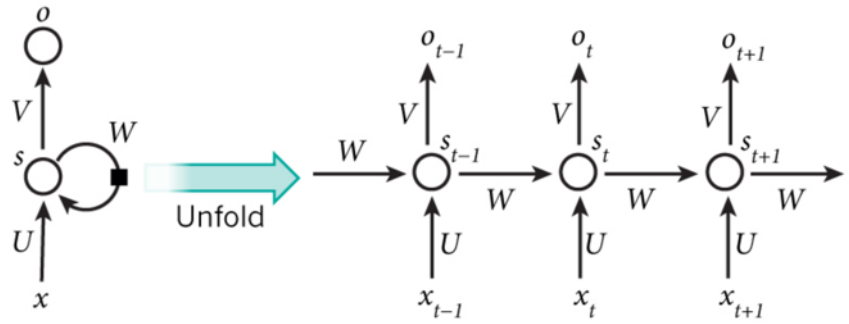
Works cited: medium.com/@RaghavPrabhu, josephpcohen.com

Recurrent Neural Nets (RNNs)

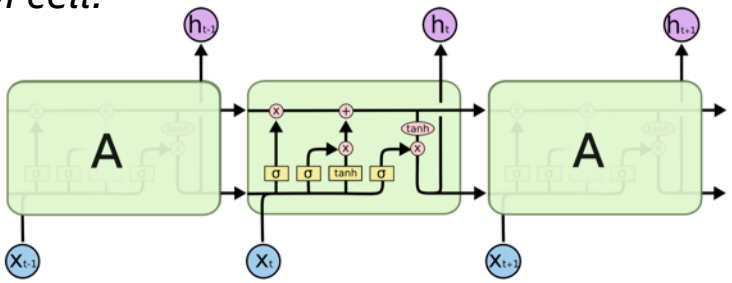
- Idea: nearby input vectors are correlated (think time series, or language)

- Math:
  - Hidden state (like a memory) computes activation using previous hidden state and current input
  - Back-prop through time (BPTT) used to train (but vanishing gradients lead to short-term memory)
  - Long-short-term-memory (LSTM) cells expand the long-term memory of the hidden states
    - Ex. "I grew up in France. I speak fluent _____."

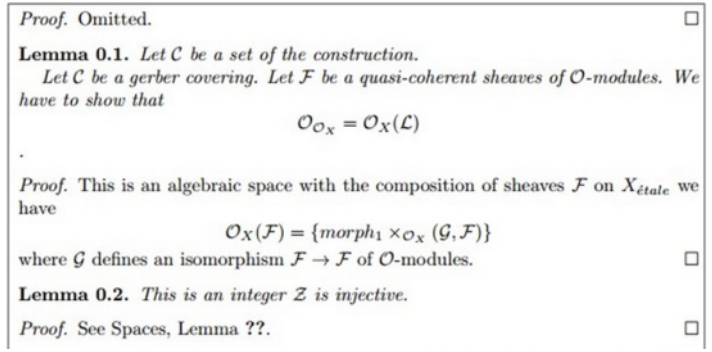
- Typical applications: time series forecasting, sentiment analysis, machine translation, image captioning
  - Check out: karpathy.github.io/2015/05/21/rnn-effectiveness/



LSTM cell:



Example:
RNN given an algebraic geometry text, asked to generate new samples. This requires learning the structure of LaTeX and of things like Lemma-Proof.



Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*
Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{étale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Generative Adversarial Networks (GANs)

- Idea: make a neural network generate more realistic samples by having a different neural network try to classify whether the sample was fake or not

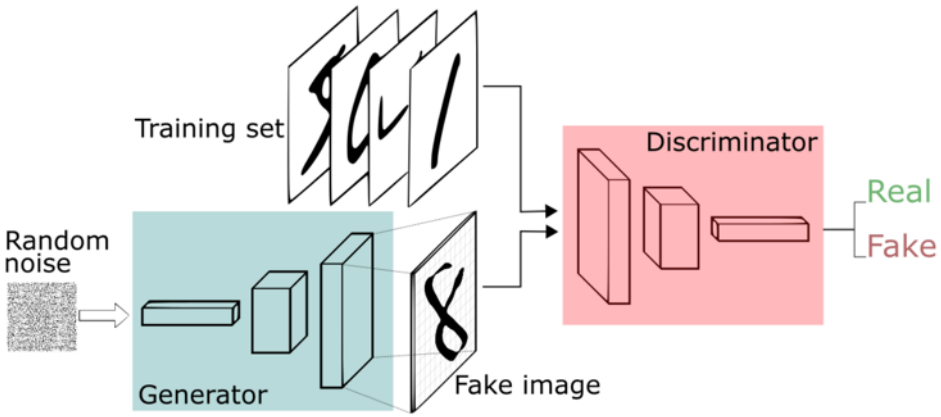
- Applications:

  - Nice: Image enhancement, data generation, generating photorealistic images for designers, games

  - Naughty: Generating counterfeits: ("deep fake")

    - Fake video: Pres. Trump chastising Belgian citizens https://www.theguardian.com/technology/2018/nov/12/deep-fakes-fake-news-truth

    - Fake images: AI generated painting sells for $432k https://www.nytimes.com/2018/10/25/arts/design/ai-art-sold-christies.html



Reinforcement Learning (RL)

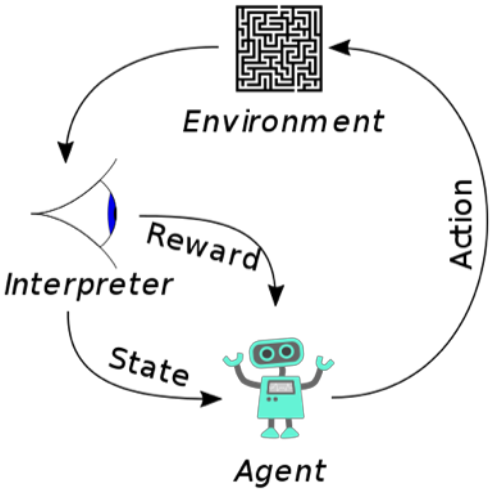
- Idea: create a flexible model, reward it for desired behavior, and let it evolve over time

- Math:
  - Agent observes state x_0 and selects an action y_0
  - It receives a "reward" r_0 which depends on x_0 and y_0
  - Environment transitions probabilistically to a new state x_1 with distribution depending only on x_0, y_0

- Goal: find a policy mapping states to actions that maximizes rewards

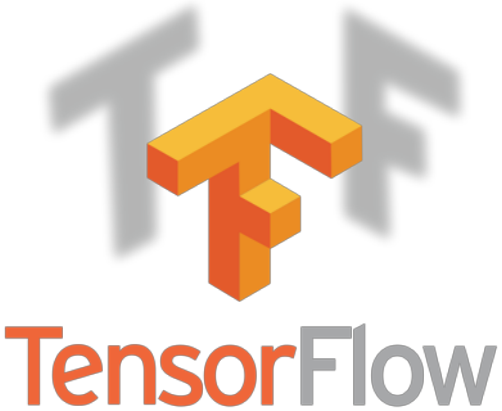
- A big limitation: requires an enormous amount of data, so typically limited to areas where data can be generated

- Applications: robot control, gaming, network control, automated bidding, …



Google DeepMind's AlphaGo algorithm defeats World Champion Lee Sedol in March 2016

Code break!



Dimensionality reduction

- Idea: many high-dimensional datasets can be well-represented on a low-dimensional manifold

- Example: Principal Component Analysis (PCA)
  - "Principal components" are coordinates of data projected onto hyperplane which maximizes variance
  - Reduce (project) $n \times p$ to $n \times k$, $k < p$

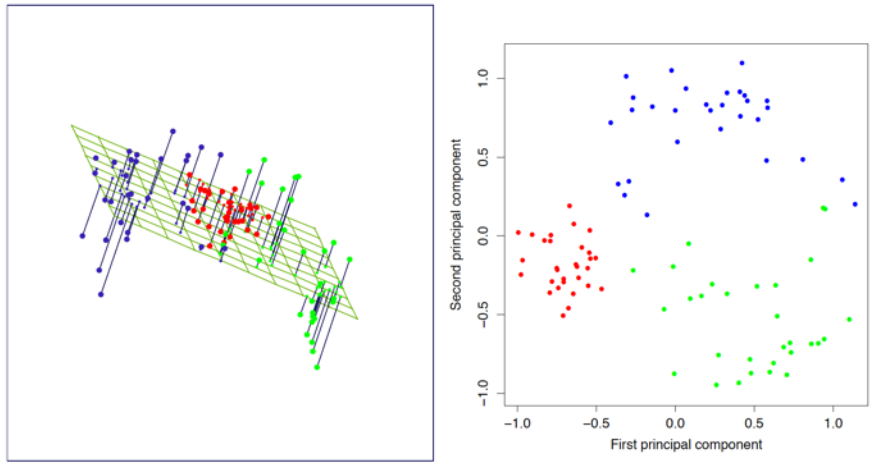
$$\underbrace{\tilde{\mathbf{X}} = \mathbf{U}_k \mathbf{S}_k \mathbf{V}_k^T}_{\text{Truncated SVD}} \qquad \mathbf{T} = \mathbf{X}\mathbf{V}_k$$

Principal components — Principal directions

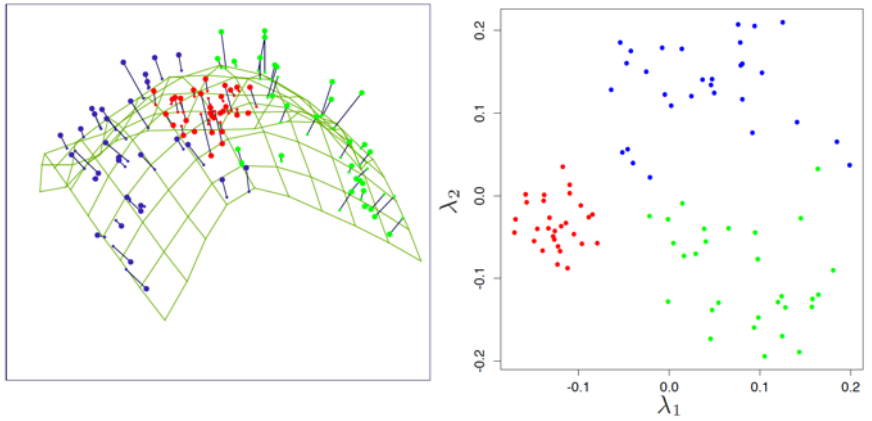
- Applications:
  - Factor analysis
  - Topic modeling
  - Feature engineering
  - Visualization

$$\mathbb{R}^3 \longrightarrow \mathbb{R}^2$$

PCA:



Kernel PCA:



Works cited: ESL

Clustering

- Idea: find groupings of data so data in the same group is similar, and data in different groups is dissimilar

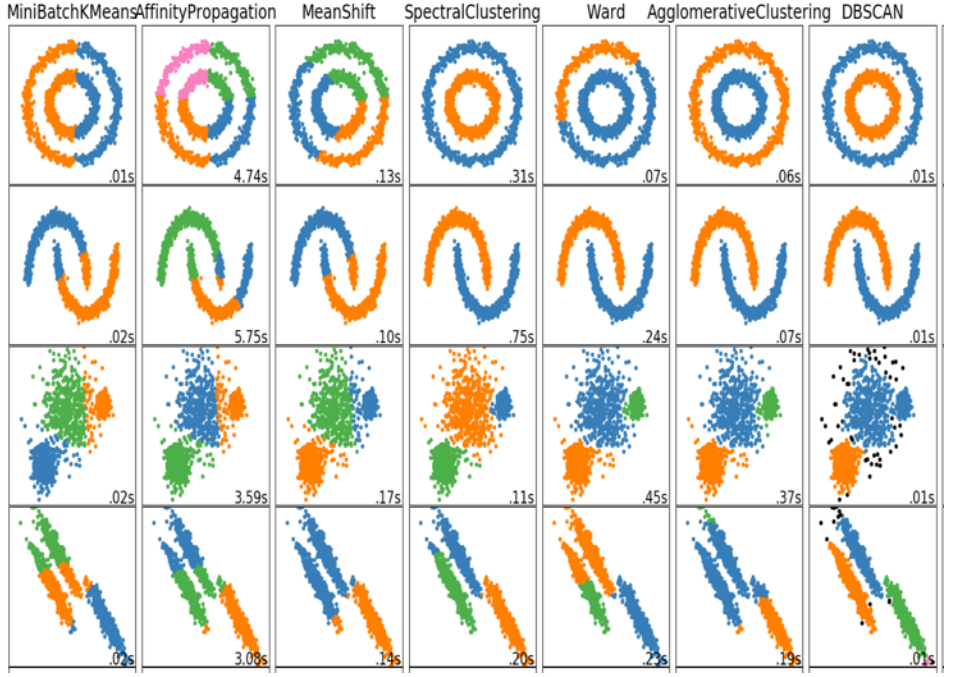
- *k*-means clustering
  - Workhorse, fast, assumes spherical clusters

- Hierarchical / agglomerative clustering
  - Handles any distance metric, fast but greedy, handles non-convexity, sensitive to reordering of the data

- Spectral clustering
  - Clusters in eigenspace of the distance matrix, slow, handles non-convexity

- DBSCAN
  - Fast, handles non-convexity, auto-selects no. of clusters, identifies outliers, tricky to tune parameters

- Affinity propagation
  - Assigns data to same cluster if they share a basin in similarity-space, does not scale well to large datasets



Works cited: sklearn docs

- Mean shift
- Gaussian Mixture Models
- Latent Dirichlet Allocation (LDA)
- Birch
- …

We are ~~Borg~~ Bayesian

- Math (one view):

$$p(y|X) = \int p(y|\Theta, X)p(\Theta|X)\ d\Theta$$
$$= \int p(y|\Theta)p(\Theta|X)\ d\Theta$$

Predictive Posterior

$$\hat{\Theta}_{\text{ML}} = \underset{\Theta}{\text{argmax}}\ p(X|\Theta)$$

$$\hat{\Theta}_{\text{MAP}} = \underset{\Theta}{\text{argmax}}\ \frac{p(X|\Theta)p(\Theta)}{p(X)}$$
$$= \underset{\Theta}{\text{argmax}}\ p(X|\Theta)p(\Theta)$$

don't approximate!

Approximation

$$p(y|X) = \int p(y|\Theta)p(\Theta|X)d\Theta$$
$$\approx \int p(y|\hat{\Theta})p(\Theta|X)d\Theta = p(y|\hat{\Theta})$$

$$p(y|X) = \int p(y|\Theta)p(\Theta|X)\ d\Theta$$
$$= \int p(y|\Theta)\ \frac{p(X|\Theta)p(\Theta)}{p(X)}\ d\Theta$$
$$= \int p(y|\Theta)\ \frac{p(X|\Theta)p(\Theta)}{\int p(X|\Theta)p(\Theta)\ d\Theta}\ d\Theta$$

- Nearly every method mentioned thus far has a Bayesian variant (counterpoint: every method thus far is just a degenerate case of a Bayesian approach – example: k-means vs. GMM using EM)

- Approaches to attack the nasty integrals:
  - Variational methods: compute an **approximate** posterior **exactly**
  - Monte Carlo methods: compute an **exact** posterior **approximately**

- Interested?? Check out "Fridays with Reverend Bayes"

I'm feeling a little bit ~~saucy~~ mathy

- Optimization (old-school OR!)
  - First-order methods: long ignored for being perceived as overly simple, now the workhorse of modern ML because they're simple and fast
  - Gradient descent
  - Variants:
    - Stochastic Gradient Descent
    - Accelerated methods ("momentum")
    - Expectation-Maximization / Projected GD

$$\min \quad f(\mathbf{w}) := \sum_i f_i(\mathbf{w})$$

GD: $\mathbf{w}_{k+1} = \mathbf{w}_k - h\nabla f(\mathbf{w}_k)$

SGD: $\mathbf{w}_{k+1} = \mathbf{w}_k - h\nabla f_i(\mathbf{w}_k)$

AGD: $\mathbf{w}_{k+1} = \mathbf{w}_k - h\nabla f(\mathbf{w}_k) + \beta(\mathbf{w}_k - \mathbf{w}_{k-1})$ *momentum*

EM: (PGD)
$Q(\mathbf{w}|\mathbf{w}_k) = \mathbb{E}[\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{Z})]$ *projection*
$\mathbf{w}_{k+1} = \operatorname{argmax} Q(\mathbf{w}|\mathbf{w}_k)$ *maximization*

- Statistical Learning Theory
  - Reproducing Kernel Hilbert Spaces (RKHS)
  - **Representer theorem**: "best possible" estimator has finite representation
    - Implication: SVM+RBF kernel, NN are *universal estimators*

$$f \in \mathcal{H}$$

$$f^* = \operatorname{argmin} L(\mathbf{X}, f) + g(f)$$

$$f^* = \sum_i \alpha_i k(\cdot, x_i)$$

Parting thoughts

- What do the haters say
  - Neural nets are "still just curve-fitting"
  - AI is still no closer to understanding cause & effect, agency, free will
  - (They're probably right…)

- How do I get started?
  - Hobbyist tinkering is the best way to learn:
    Sports analytics, market data, Kaggle competitions, your own data, …
  - Build understanding of theory as you tinker

- Some good books:

Start with:

Intro to Statistical Learning, by Hastie/Tibshirani (it's $free$, uses R)

Advance to:

Elements of Statistical Learning, by Hastie/Tibshirani ($free$)
Pattern Recognition and Machine Learning, by Bishop (also $free$ as of November 2018!)
Machine Learning: a Probabilistic Perspective, by Murphy

Judea Pearl
Inventor of Bayesian networks
Turing Prize winner (2011)
Hater

Thanks!

Questions?