Supervised Learning: Regression and Classification

Presentation · July 2019

CITATIONS
0

READS
3,871

1 author:

Logan Ward
Argonne National Laboratory
117 PUBLICATIONS **4,015** CITATIONS

SEE PROFILE

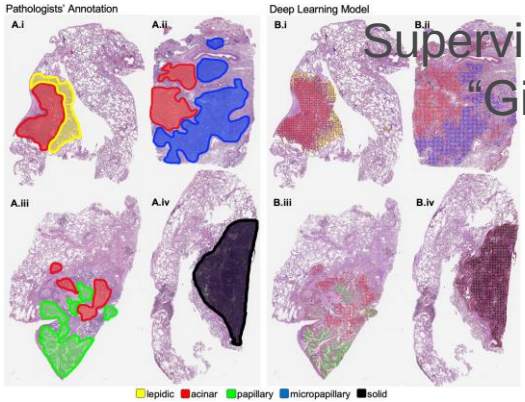
SUPERVISED LEARNING: REGRESSION AND CLASSIFICATION

LOGAN WARD
Asst. Computational Scientists
Data Science and Learning Division

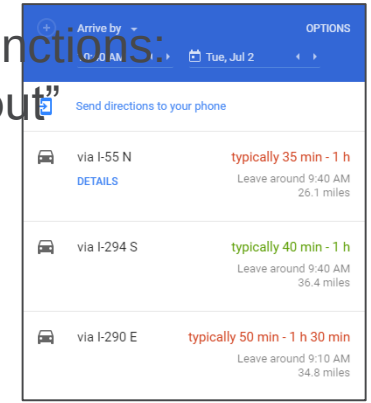
SUPERVISED LEARNING

The ML you have likely both heard of, used, and done before

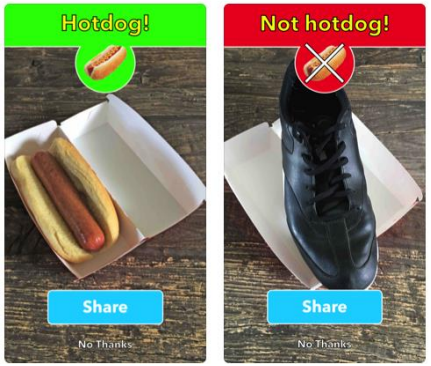
Supervised Learning models functions: "Given inputs, predict output"



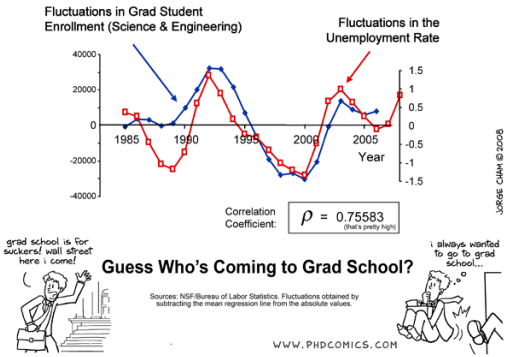
Wei *et al.* Sci Rep. (2019), 3358



Google Maps



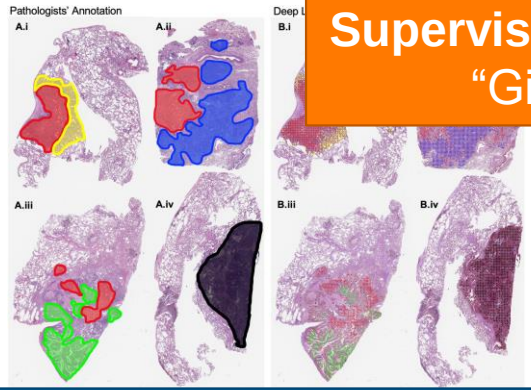
https://apps.apple.com/us/app/not-hotdog/id1212457521



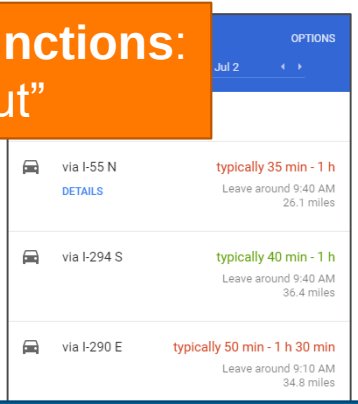
http://phdcomics.com/comics/archive.php?comicid=1078

SUPERVISED LEARNING

The ML you have likely both heard of, used, and done before



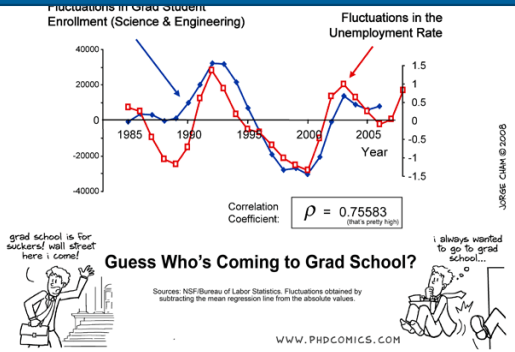
Supervised Learning models functions:
"Given inputs, predict output"



Classification: Outputs are *discrete*

https://apps.apple.com/us/app/not-hotdog/id1212457521

Regression: Outputs are *continuous*



http://phdcomics.com/comics/archive.php?comicid=1078

GOALS FOR TODAY

Get familiar enough to start using ML

Goal 1: What do I need to know to train a useful ML model?

Goal 2: How do I do it in Keras and Scikit-Learn?

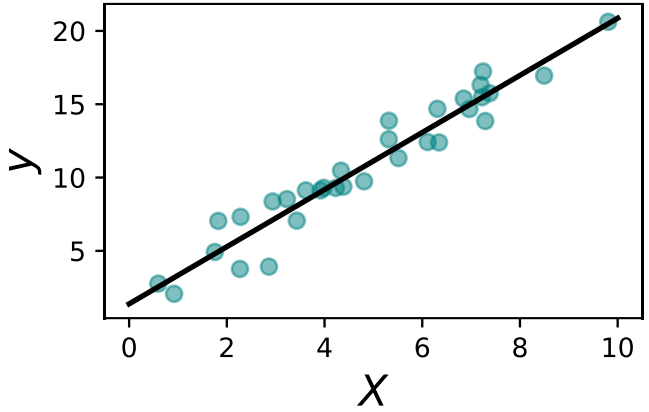
LINEAR REGRESSION IS “MACHINE LEARNING,” …BUT LIKELY INSUFFICIENT

AN OLD FRIEND: SIMPLE LINEAR REGRESSION

… but let's make it sound modern

Model Architecture

$$f(x; m, b) = mx + b$$



Training Data: Inputs (x_i) and outputs (y_i)

Goal: Determine m and b that minimize

Loss Function

$$\sum_i (f(x_i; m, b) - y_i)^2$$

by computing

Optimizer

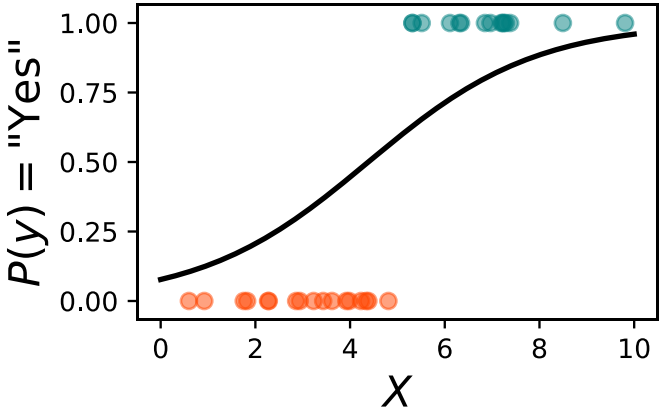
$$m = \mathrm{Cov}[x, y]/Var[x]$$
$$b = \bar{y} - m\bar{x}$$

SIMPLE LOGISTIC REGRESSION

A version of Linear Regression suitable for classification

Model Architecture

$$f(x; m, b) = \frac{1}{1+e^{-(mx+b)}}$$



Training Data: Inputs (x_i) and outputs (y_i)

Goal: Determine m and b that minimize

Loss Function

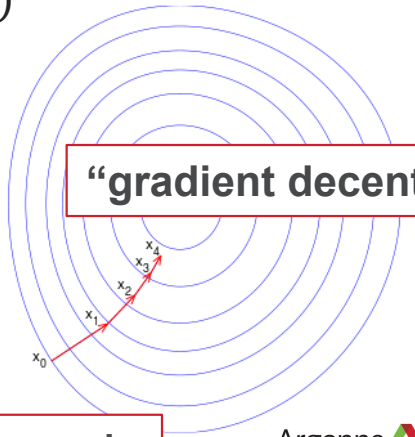
$$L(m, b) \sum_i y_i \ln\big(f(x_i)\big) + (1 - y_i)\ln(1 - f(x_i)$$

"log loss"*

by computing

Optimizer

$$x_0 = (1, 0)$$
$$x_{n+1} = x_n + \gamma \nabla L(m, b)$$



"gradient decent"*

Architecture + Loss + Optimizer = ML Algorithm
For Regression *and* Classification

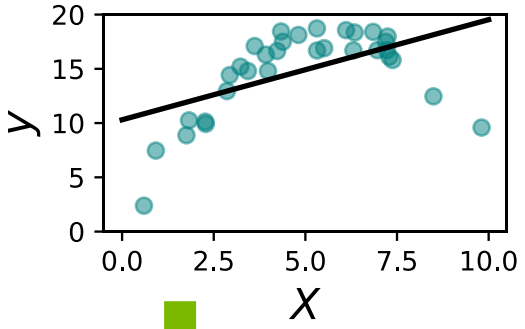
*You will be seeing these again

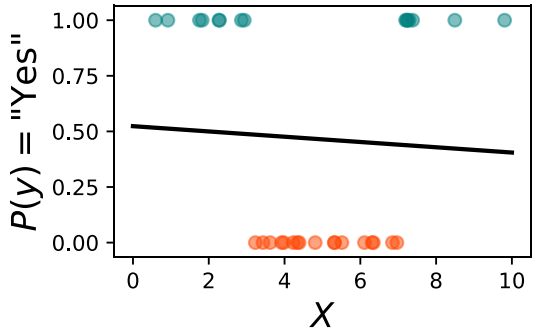
LINEAR MODELS ARE NOT SUFFICIENT

Otherwise, this would be a very short lecture

Why Not? Model complexity is limited

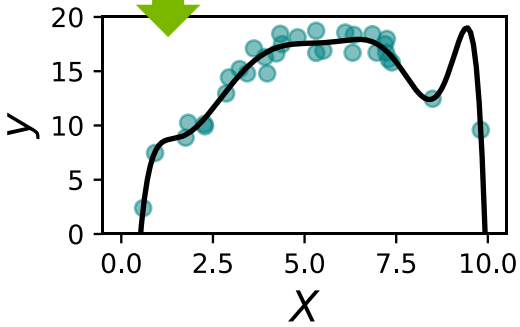
"underfit"*





… and adding complexity comes with risks

"overfit"*



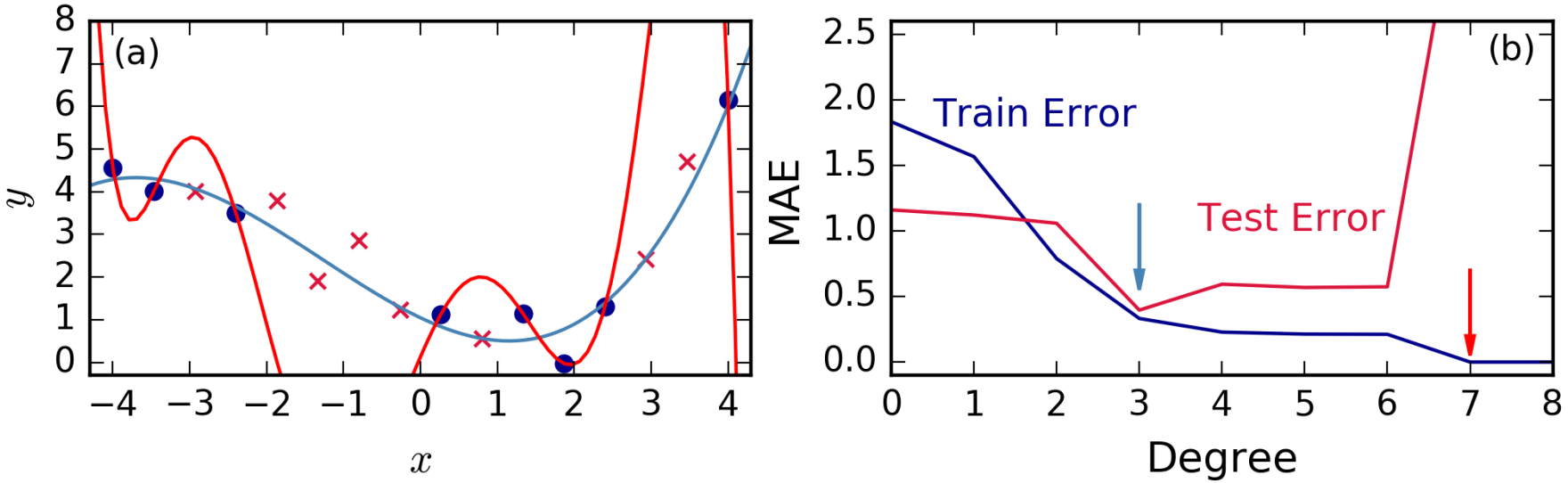
Key Questions for Supervised ML:

1. How to add more complexity?
2. How to know when "overfit"?

MODEL SELECTION:
THE KEY TASK IN SUPERVISED ML

A REPRISE: OVERFITTING AND COMPLEXITY

Training accuracy vs “generalizability”



TWO *MAJOR* CONCEPTS

Don't leave this room without them!

CROSS VALIDATION

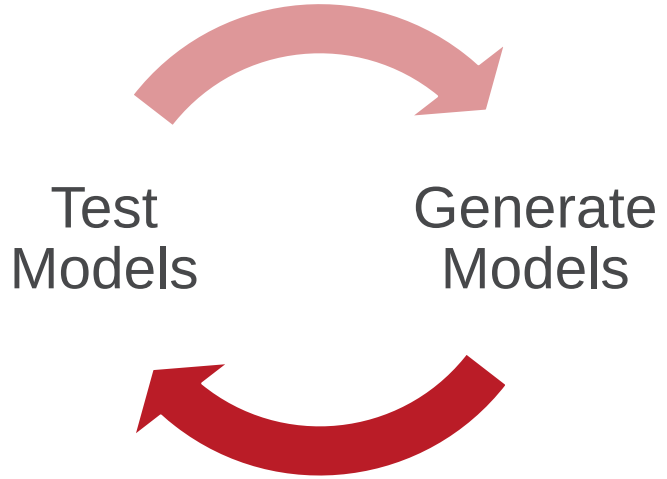
Given available data, test whether model is predictive

Basic Techniques:

- *Shuffle Split*
  1. Pick 10% as test set
  2. Train on remaining 90%
  3. Test model on test set
- *Leave-one-out*
  1. Pick one entry
  2. Train on remaining entries
  3. Test model on held-out entry
  4. Repeat using each entry

HYPERPARAMETER OPTIMIZATION

Adjust model settings to maximize CV performance



HOW DO WE ADJUST COMPLEXITY?

We'll just talk linear models for now, but these are general ideas

LIMIT COMPLEXITY

Reduce degrees of freedom

$$f(x) = a_0 + a_1 x + a_2 x^2 + [\dots]$$

PENALIZE COMPLEXITY

Add "complexity" to loss function

$$\sum_i (f(x_i) - y_i)^2 + \lambda a a^T$$

Larger coefficients = bigger penalty

EXERCISE 1: USING SCIKIT-LEARN

- Open up “sess1_supervised/exercise-1_cv-and-hpo.ipynb”

NONLINEAR REGRESSION/CLASSIFICATION …BUT REALLY ONLY NEURAL NETWORKS

THERE IS A ZOO OF ML ALGORITHMS

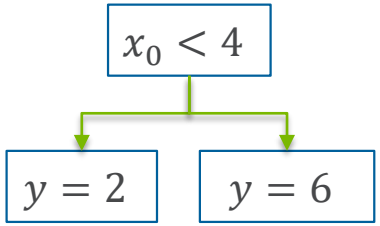
Here are just a few

Variations of Bayes' Theorem

Linear regression, etc.

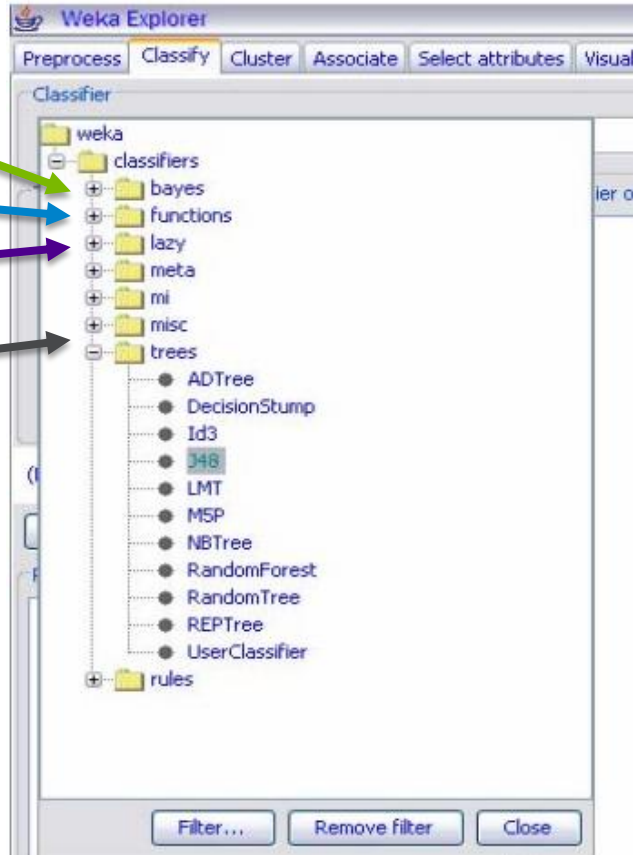
k-Nearest neighbors

Many kinds of decision trees



Trees vary in…

- whether to process inputs
- ways picking splits
- what are on the "leaves"
- how to prune after training

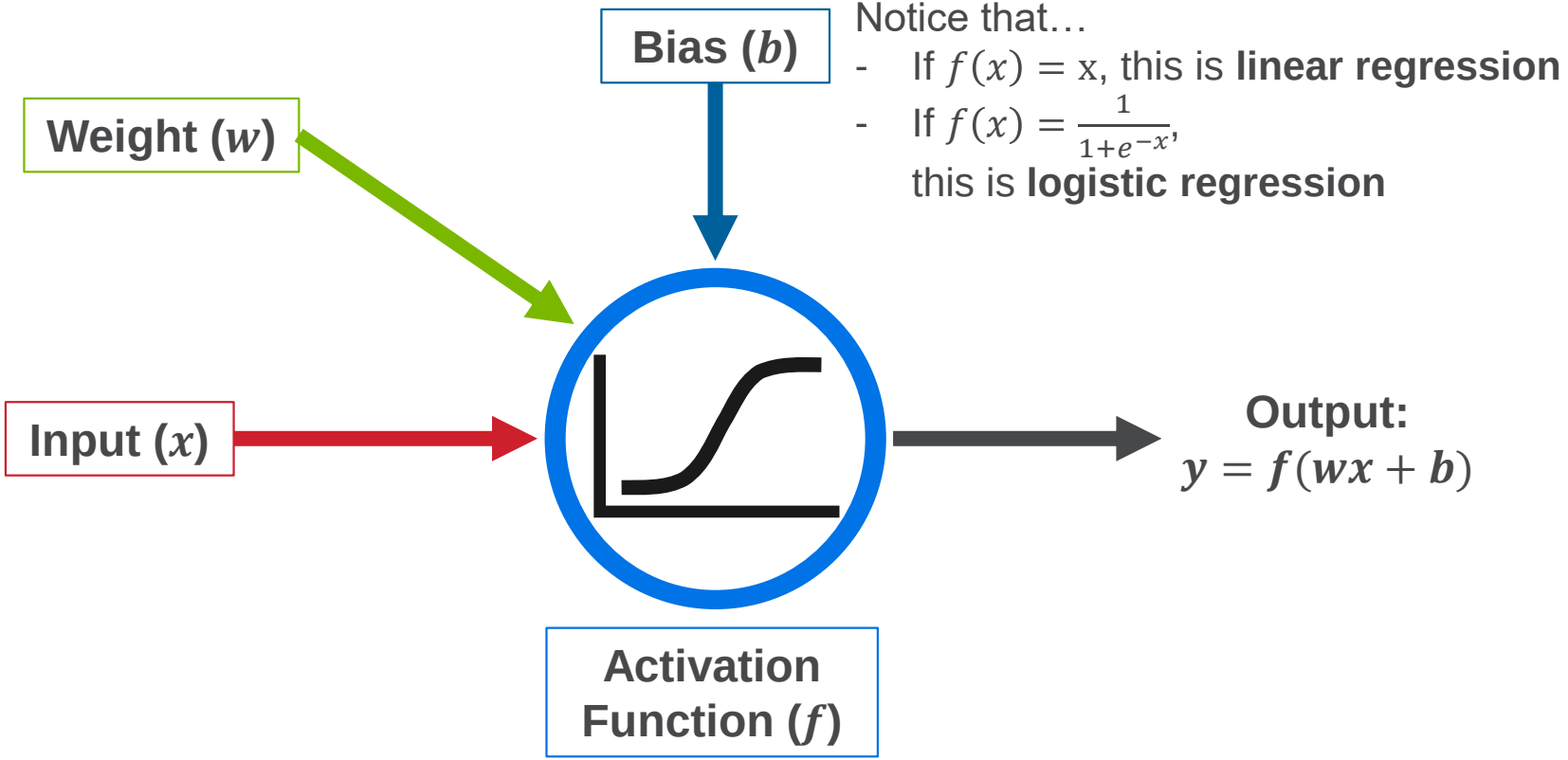


There is no "algorithm to rule them all"

http://www.site.uottawa.ca/~stan/csi7163/weka_workshop_slides.pdf

TODAY'S FOCUS: NEURAL NETWORKS

Build on a small building block: "The Perceptron"



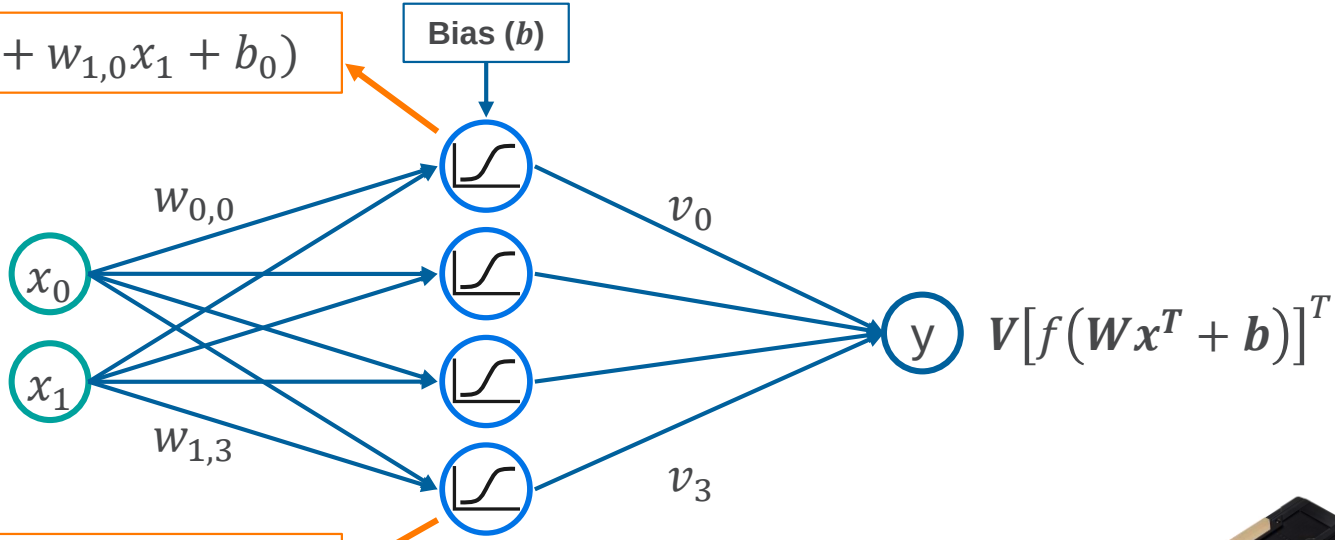
Notice that…

- If $f(x) = x$, this is **linear regression**
- If $f(x) = \frac{1}{1+e^{-x}}$, this is **logistic regression**

MANY PERCEPTRONS = NEURAL NETWORK

Stack enough, and you get *very* complex functions

$$h_0 = f(w_{0,0}x_0 + w_{1,0}x_1 + b_0)$$



$$h_3 = f(w_{0,3}x_0 + w_{1,3}x_1 + b_3)$$

Why popular now?

Many small operations
\+ performed on many data
Massive Parallelism

HOW DO I TRAIN A NEURAL NETWORK?

Remember when I said “gradient decent”

Key Terminology:

Architecture: How inputs/outputs are linked, adjustable weights

Loss Function: Generates error between “current” and “desired” outputs

Optimizer: Algorithm for finding parameters that minimize a function

These are the three ingredients forming all* neural networks

*Probably an overgeneralization

HOW DO I TRAIN A NETWORK?

Short Answer: Gradually make the weights better

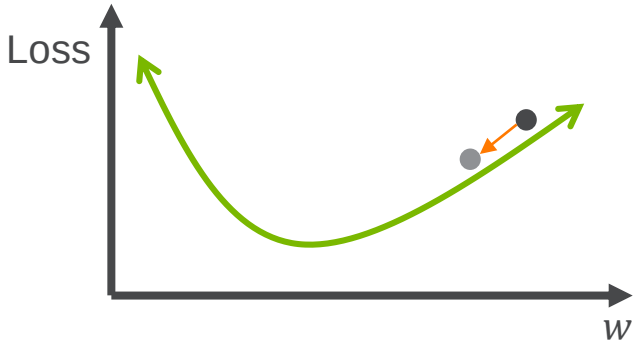
Simple procedure:

1. Compute output
2. Compute “loss”
3. Compute how each weight affects loss
   (This is called “back propogation”)
4. Adjust weights to lower loss
   (More complicated than you might think)
5. Repeat with new weights

$$\hat{y} = f(X; w)$$

$$L = (y - \hat{y})^2$$

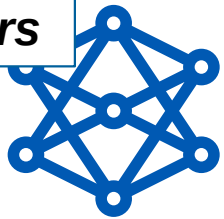
$$w' = w + \gamma \frac{\delta L}{\delta w}$$



THERE IS A RICH VARIETY IN NEURAL NETWORKS

Optimizers, layers, and loss functions

Layers



Activation: Applies function to an input

Batch Normalization: Make batch mean 0, std. 1

Convolution: Apply spatial/temporal filters

… Dense, Dropout, Embedding, ….

Loss Functions

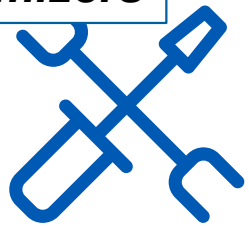
Log-loss: Classification, same loss function as logistic regression

Mean Absolute Error: Regression, small penalty for outliers

Mean Squared Error: Regression, large penalty for outliers

… KL divergence, accuracy …

Optimizers



Many difference techniques:

Momentum: Keep moving in direction of last step

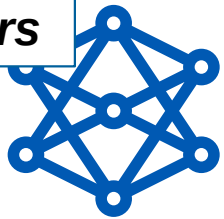
Decay: Gradually lower step size

Clipping: Prevent too large of gradient changes

THERE IS A RICH VARIETY IN NEURAL NETWORKS

Optimizers, layers, and loss functions

Layers



Activation: Applies function to an input

Batch Normalization: Make batch mean 0, std. 1

Convolution: Apply spatial/temporal filters

… Dense, Dropout, Embedding, ….

Loss Functions

Log-loss: Classification [illegible] regression

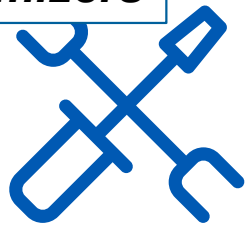
Mean [illegible], small penalty for outliers

[illegible]ared Error: Regression, large penalty for outliers

… KL divergence, accuracy …

Learning when to apply what technique takes practice!

Optimizers



Many difference techniques:

Momentum: Keep moving in direction of last step

Decay: Gradually lower step size

Clipping: Prevent too large of gradient changes

DNN EXERCISE: KEY SKILLS

Learning how to make and train a model effectively with Keras

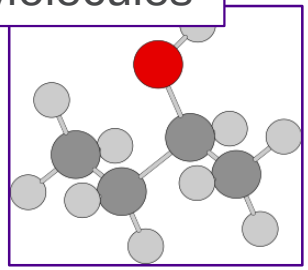
- Open the second exercise!

WHAT IF MY DATA ISN'T A 1D VECTOR!?

NOT ALL DATA ARE VECTORS

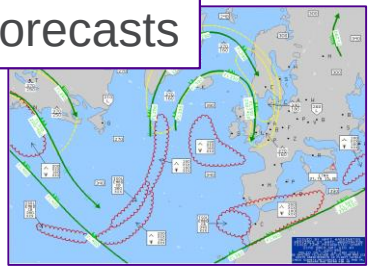
And that's OK!

Molecules

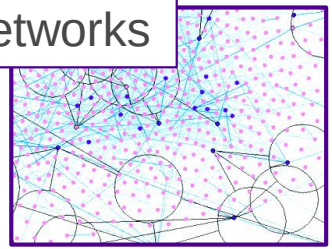


Images

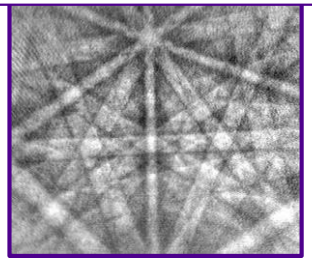
Weather Forecasts



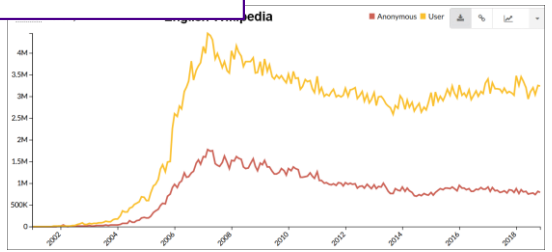
Social Networks



EBSD Patterns



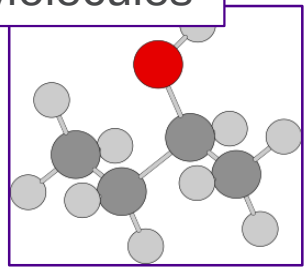
Timeseries



NOT ALL DATA ARE VECTORS

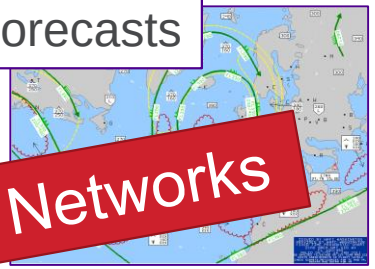
And that's OK!

Molecules



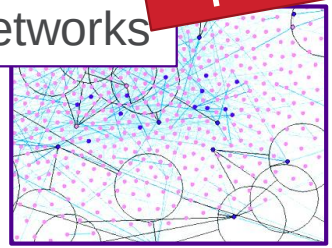
Images

Weather Forecasts

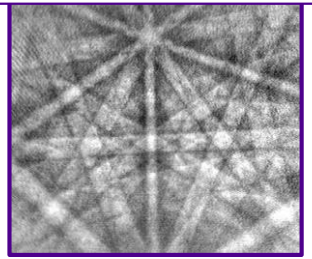


Special Data Can Need Specialized Networks

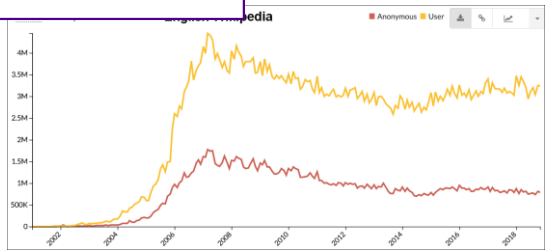
Social Networks



EBSD Patterns



Timeseries

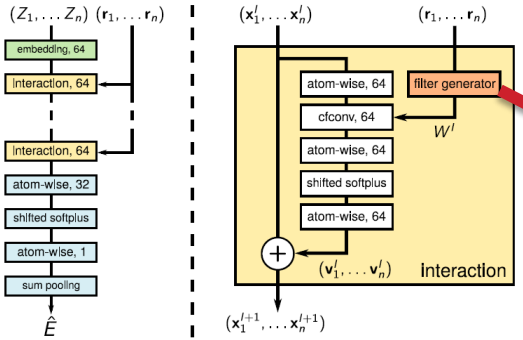


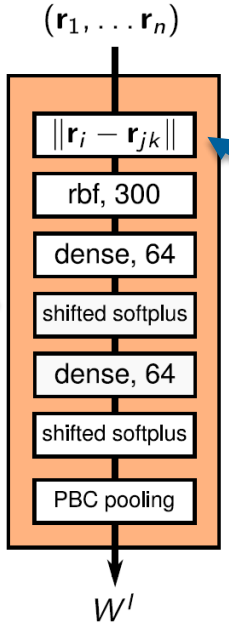
GENERAL IDEA: EXPLOIT SYMMETRIES IN DATA

Special data gets special networks

Example: Molecules have rotational and translational symmetry

Z: Atom type
r: Atom position





pairwise distances,
not atomic coordinates
=
Desired invariances

We are going to do simpler examples

"SchNet": Schütt et al. JCP. (2018), 241722

WORD EMBEDDINGS

Many relations between meanings of words ("analogies")

Challenge: Represent words as inputs

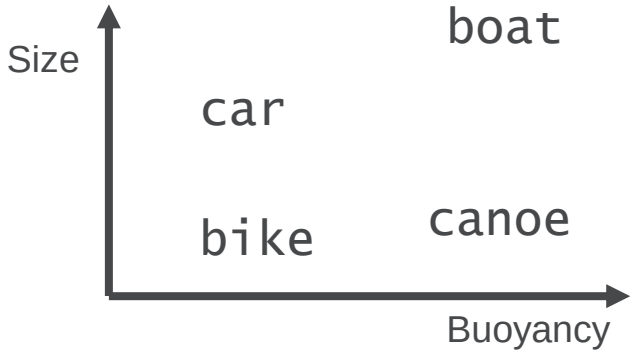
Idea 1: One input per word

```
canoe   =   [1, 0, 0, 0]
ship    =   [0, 1, 0, 0]
bike    =   [0, 0, 1, 0]
car     =   [0, 0, 0, 1]
```

Problem: Information lean!

- >10^5 features for some languages
- Mutually orthogonal for each word

Idea 2: *Embed* words with meaning



Problems are fixed!

- Arbitrary number of features
- Feature vectors encode meaning

WORD EMBEDDINGS

Many relations between meanings of words (“analogies”)

Challenge: Represent words as inputs

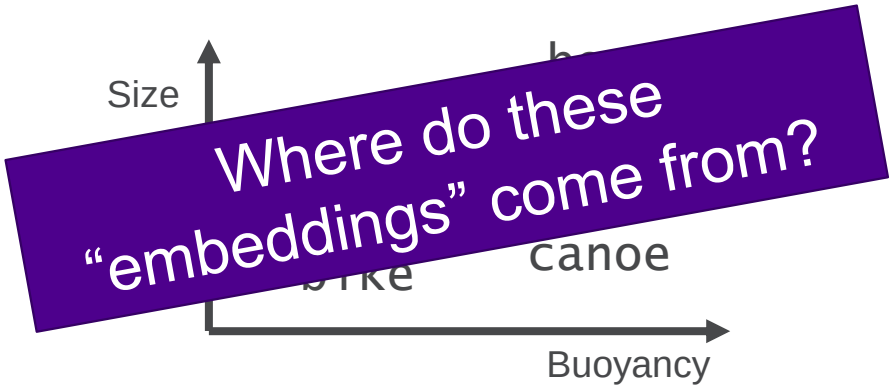
Idea 1: One input per word

```
canoe   =   [1, 0, 0, 0]
ship    =   [0, 1, 0, 0]
bike    =   [0, 0, 1, 0]
car     =   [0, 0, 0, 1]
```

Problem: Information lean!

- >10^5 features for some languages
- Mutually orthogonal for each word

Idea 2: *Embed* words with meaning



Problems are fixed!

- Arbitrary number of features
- Feature vectors encode meaning

ORIGIN OF THE EMBEDDINGS

Wikipedia, of course!

Word vectors for 157 languages

We distribute pre-trained word vectors for 157 languages, trained on *Common Crawl* and *Wikipedia* using fastText. These models were trained using CBOW with position-weights, in dimension 300, with character n-grams of length 5, a window of size 5 and 10 negatives. We also distribute three new word analogy datasets, for French, Hindi and Polish.

Similar embeddings mean words used in similar contexts

Format

The word vectors are available in both binary and text formats.

Using the binary models, vectors for out-of-vocabulary words can be obtained with

```
$ ./fasttext print-word-vectors wiki.it.300.bin < oov_words.txt
```

where the file oov_words.txt contains out-of-vocabulary words.

In the text format, each line contain a word followed by its vector. Each value is space separated, and words are sorted by frequency in descending order. These text models can easily be loaded in Python using the following code:

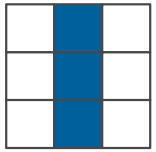
Example from http://fasttext.cc

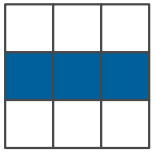
```
import io

def load_vectors(fname):
```

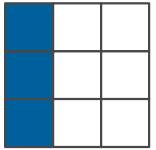
IMAGE CLASSIFICATION AND CONVOLUTIONS

Better classification by translation symmetry

Example: Classify Horizontal vs Vertical lines





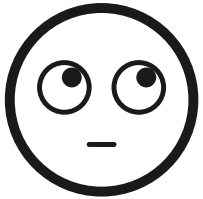


Initial Approach: Just flatten the images. They are now vectors.

How do we know which are which? Adjacent blue blocks

Problem! Fully connected NNs don't care about order

Solution: Make new features that deal with order



CONVOLUTIONS, PADDING, AND POOLING

Borrow from computer vision, graphics

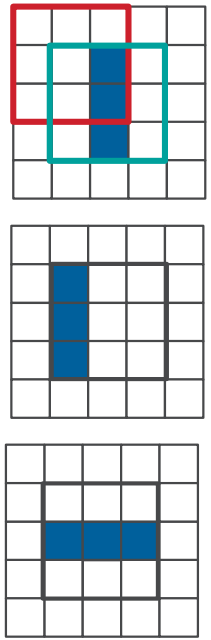
1. Pad Image

2. Convolve Filter

3. Maximum of Image (“Pooling”)

Vertical Edge Filter:

0	1	0
0	0	0
0	1	0



Classification is **easy** with filters!

CONVOLUTIONS, PADDING, AND POOLING

Borrow from computer vision, graphics

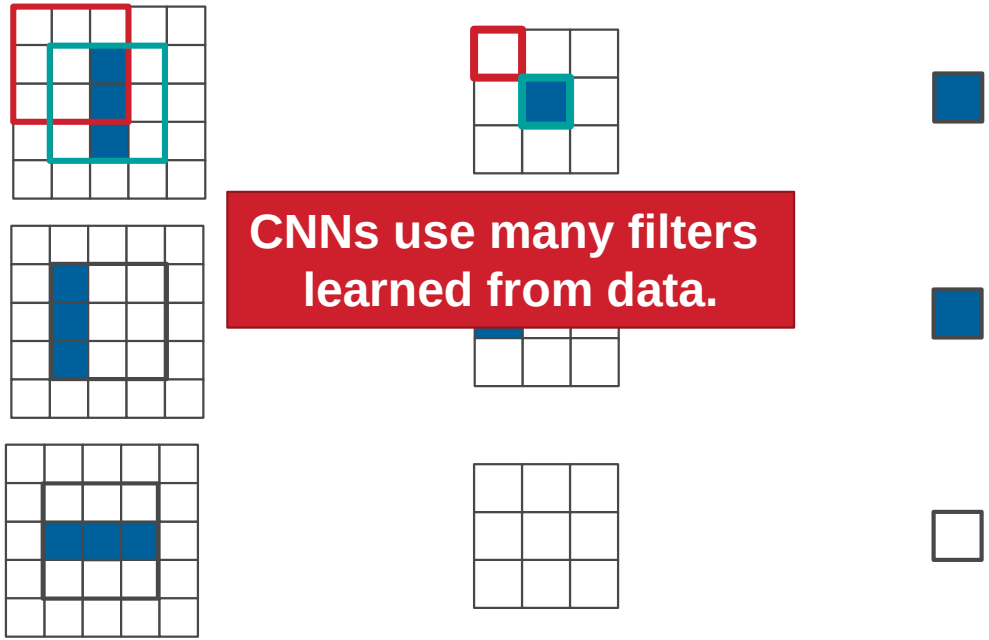
1. Pad Image

2. Convolve Filter

3. Maximum of Image (“Pooling”)

Vertical Edge Filter:

0	1	0
0	0	0
0	1	0



CNNs use many filters learned from data.

Classification is **easy** with filters!

EXERCISE: DIGIT CLASSIFICATION

It’s a great tutorial example!

- Let’s do example #3

TAKE-HOME MESSAGES

1. **Use cross-validation to detect overfitting**
   - Restrict or penalize model complexity

2. **Hyperparameter optimization to maximize generalizability**
   - GridSearchCV for scikit-learn, consider HyperOpt for Neural Networks

3. **Neural Networks have three main components**
   1. *Architecture:* How the “perceptrons” are arranged
   2. *Loss Function:* Measures difference between “actual” and “expected”
   3. *Optimizer*: How network weights are adjusted to lower loss

4. **Special data requires special networks**
   - General concept: Exploit symmetries / domain knowledge
   - Special Example: Convolutions exploit translation symmetry and that “nearby” pixels/inputs are related

EMAIL ME AT LWARD@ANL.GOV
IF YOU HAVE QUESTIONS!