CS145 Discussion: Week 4
SVM, Neural Networks, KNN, Time Series Prediction

Junheng Hao

Friday, 10/30/2020

Roadmap

- Announcements
- SVM (Cont’d)
- Neural Networks
- KNN
- Project tips: Time Series Prediction

UCLA
Engineer Change.

Announcement

- Homework 2 due **today, Oct 30 (Friday) 11:59 PT**
  - Submit through **GradeScope** of 1 PDF (2 python file and 1 jupyter notebook into 1 PDF file)
  - Assign pages to the questions on GradeScope
- Homework 3 will be released later today, due **Nov. 9 (Monday, Week 6) 11:59 PT**
- Midterm project due on **Nov. 11 (Wednesday, Week 6)**
  - 3-page midterm project report
  - At least one submission to Kaggle

- **Approximately 3 pages**
- Current progress about project, including
  - Data processing and transformation
  - Designed & tested models / methods
- Discussion and future project plan
  - Some conclusions and findings
  - Analysis of current models and techniques
  - Timeline of future project plan (around the next 4 weeks)

SVM: Kernel

- The linear SVM relies on an inner product between data vectors,

$$K(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i}^{\mathbf{T}} \mathbf{x_j}$$

- If every data point is mapped into high-dimensional space via transformation, the inner product becomes,

$$K(\mathbf{x_i}, \mathbf{x_j}) = \phi^T(\mathbf{x_i}) \cdot \phi(\mathbf{x_j})$$

- Do we need to compute ***ϕ(x)*** explicitly for each data sample? → **Directly compute kernel function *K(xi, xj)***

SVM: Kernel Function Example

$$\textbf{Polynomial kernel of degree } h: \quad K(X_i, X_j) = (X_i \cdot X_j + 1)^h$$

$$\textbf{Gaussian radial basis function kernel}: \quad K(X_i, X_j) = e^{-\|X_i - X_j\|^2 / 2\sigma^2}$$

$$\textbf{Sigmoid kernel}: \quad K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$$

- Kernel matrix is symmetric positive semi-definite.
- Given the same data samples, what is the difference between linear kernel and non-linear kernel? Is the decision boundary linear (in original feature space)?

Non-linear SVM Solution

- Decision Boundary

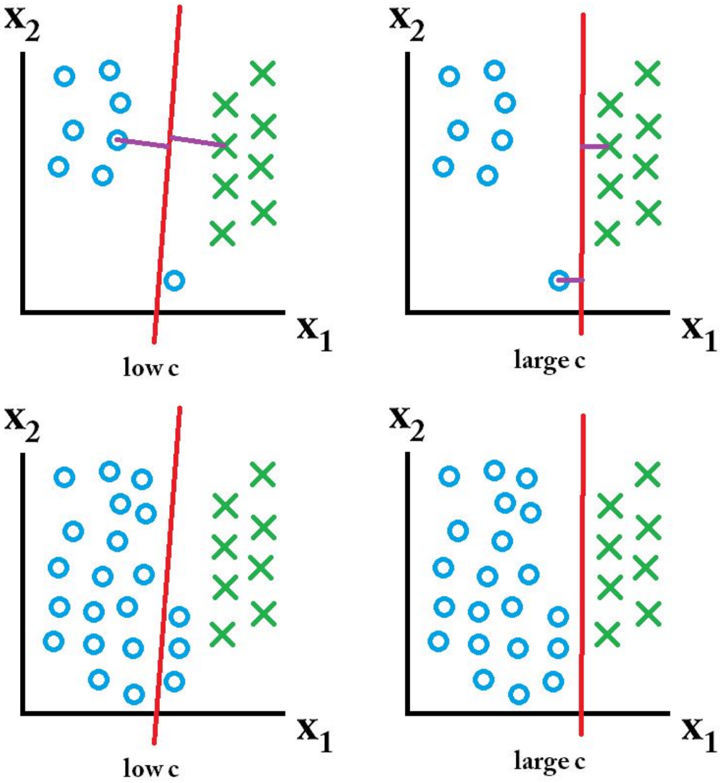
$$y \leftarrow \operatorname{sign}\left[\sum_i \alpha_i y_i K(x_i, x) + b\right]$$

SVM: Overfitting

- Huge feature space with kernels: should we worry about overfitting?
  - SVM objective seeks a solution with large margin.
  - Theory says that large margin leads to good generalization.
  - But everything overfits sometimes.
  - Can control by:
    - Setting C
    - Choosing a better Kernel
    - Varying parameters of the Kernel (width of Gaussian, etc.)

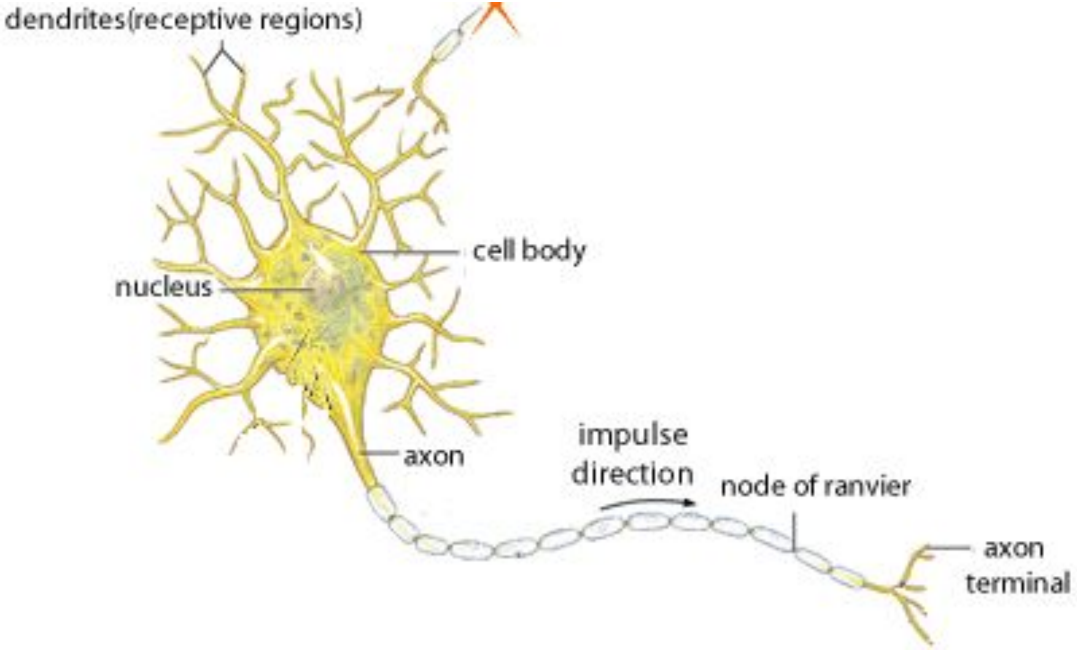
SVM: Understanding C

- The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example.
- For large values of C, the optimization will choose **a smaller-margin hyperplane** if that hyperplane does a better job of getting all the training points classified correctly.
- Conversely, a very small value of C will cause the optimizer to look for **a larger-margin separating hyperplane**, even if that hyperplane misclassified more points.



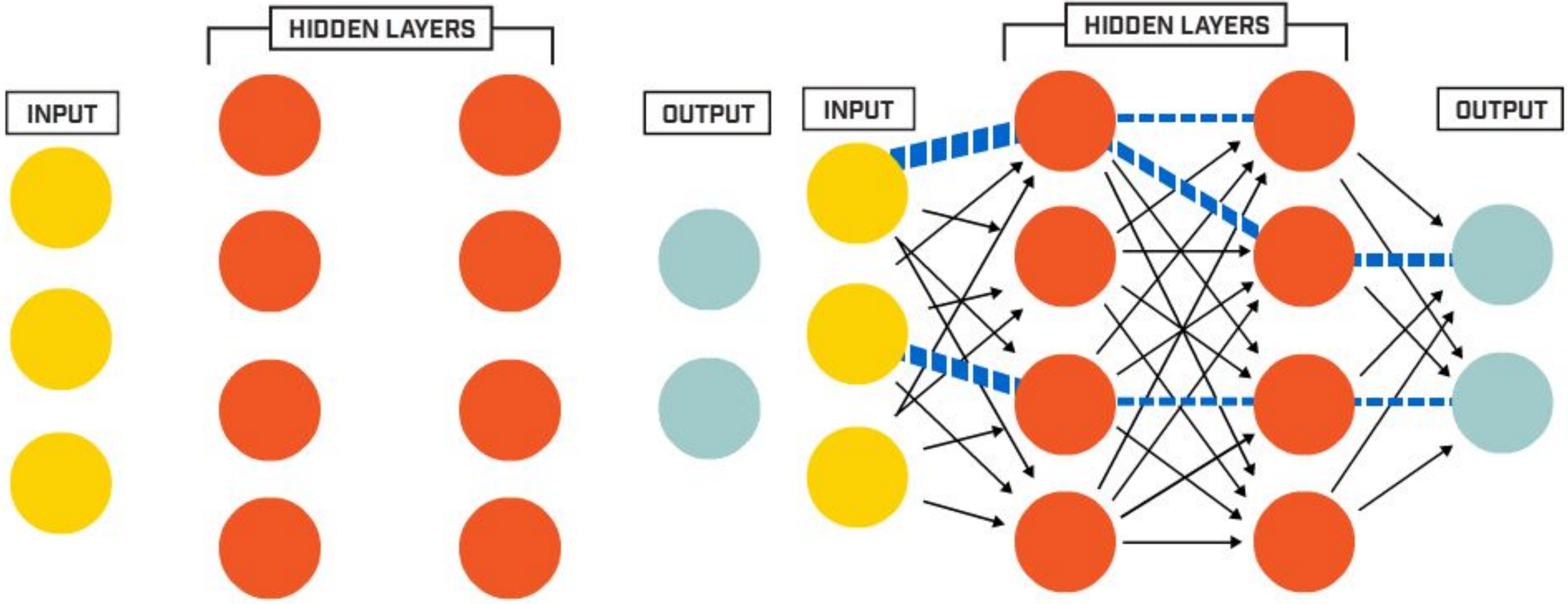
https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

Neural Networks: Neuron/Perceptron



https://medium.com/typeme/lets-code-a-neural-network-from-scratch-part-1-24f0a30d7d62
https://becominghuman.ai/what-is-an-artificial-neuron-8b2e421ce42e

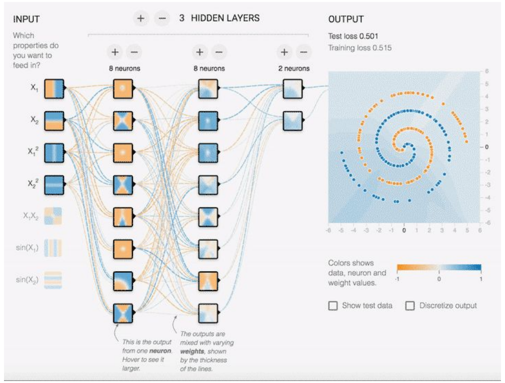
Neural Networks: A Simple Architecture

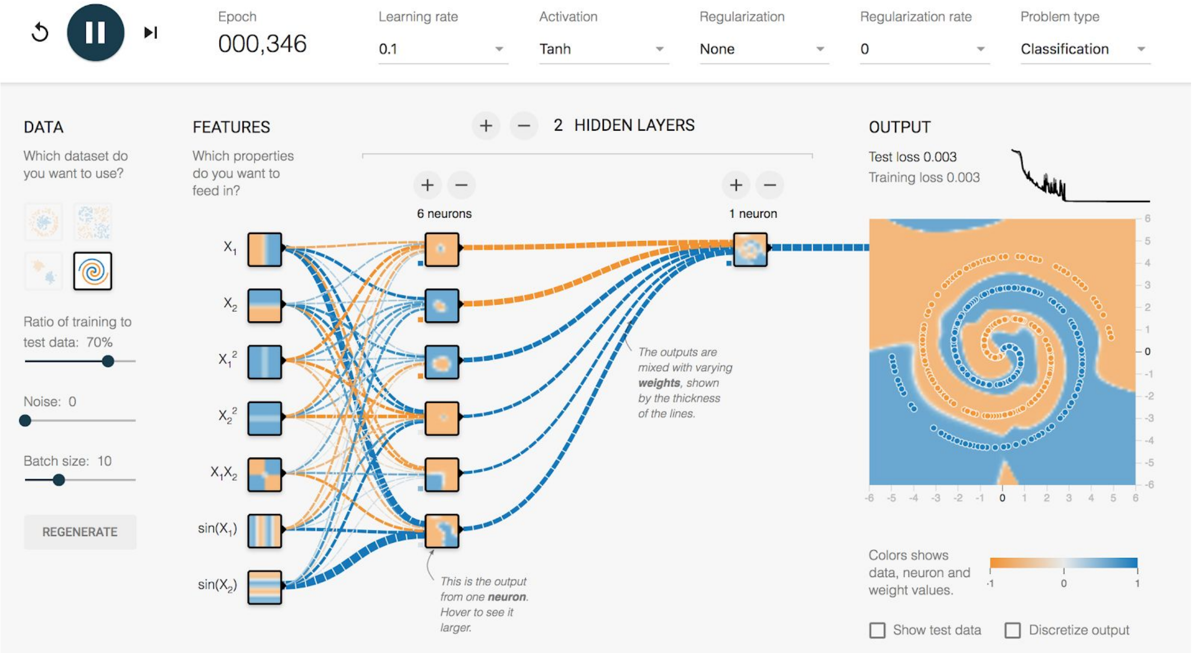


https://www.ptgrey.com/deep-learning

Neural Networks: Demo

- Let's play with it: https://playground.tensorflow.org/





NN Example: XOR

- Which NN architecture corresponds to which function?

Y \ X	0	1
1	0	1
0	0	0

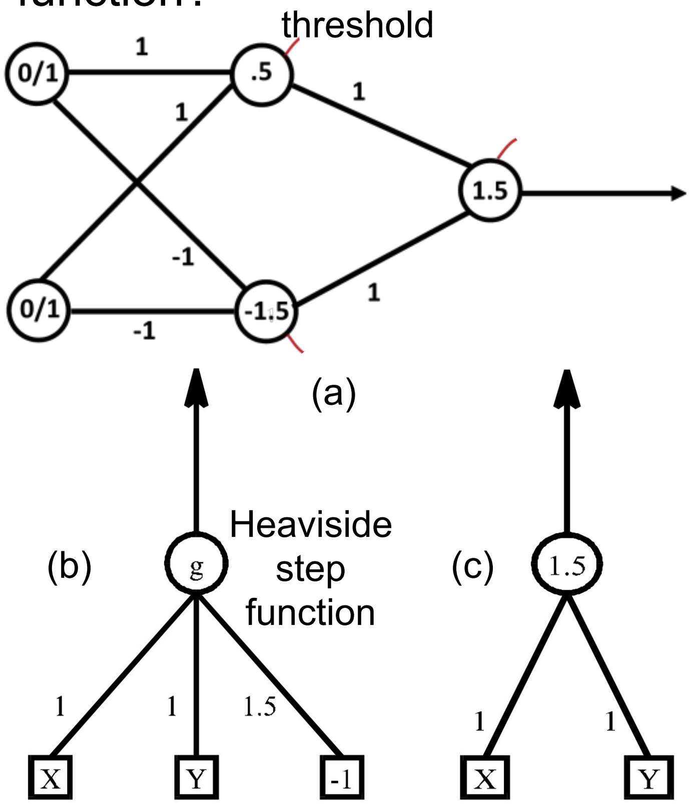
Table 1: Truth table for AND

Y \ X	0	1
1	1	1
0	0	1

Table 2: Truth table for OR

Y \ X	0	1
1	1	0
0	0	1

Table 3: Truth Table for XOR



https://datascience.stackexchange.com/questions/11589/creating-neural-net-for-xor-function
http://yen.cs.stir.ac.uk/~kjt/techreps/pdf/TR148.pdf
https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b

NN Example: XOR

	1	0	1
Y	0	0	0
		0	1
		X	

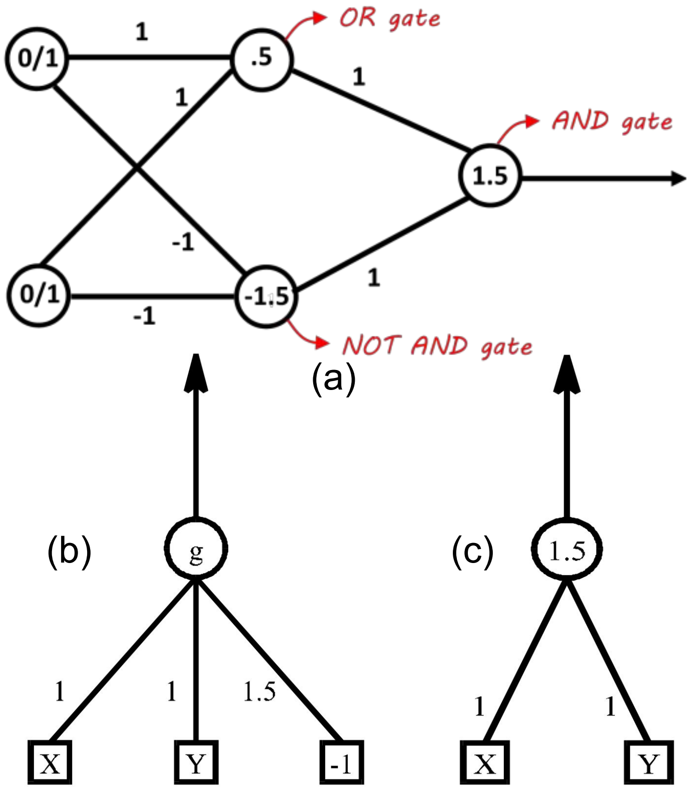
Table 1: Truth table for AND

	1	1	1
Y	0	0	1
		0	1
		X	

Table 2: Truth table for OR

	1	1	0
Y	0	0	1
		0	1
		X	

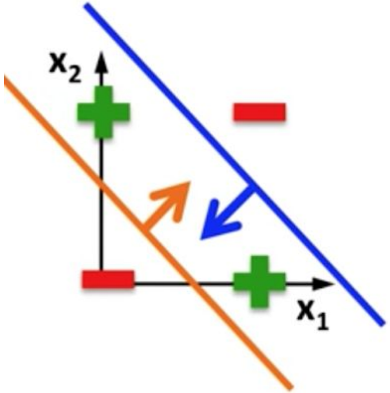
Table 3: Truth Table for XOR

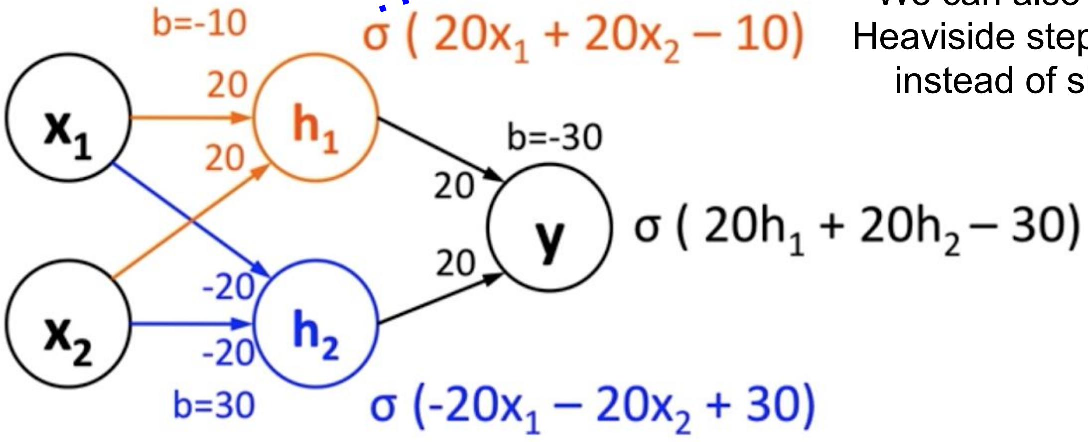


https://datascience.stackexchange.com/questions/11589/creating-neural-net-for-xor-function
http://yen.cs.stir.ac.uk/~kjt/techreps/pdf/TR148.pdf
https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b

NN Example: XOR

Linear classifiers cannot solve this





We can also use the Heaviside step function instead of sigmoid

x_1 x_2	x_1 x_2	
$\sigma(20*0 + 20*0 - 10) \approx 0$	$\sigma(-20*0 - 20*0 + 30) \approx 1$	$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$	$\sigma(-20*1 - 20*1 + 30) \approx 0$	$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*0 + 20*1 - 10) \approx 1$	$\sigma(-20*0 - 20*1 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$	$\sigma(-20*1 - 20*0 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$

https://www.youtube.com/watch?v=kNPGXgzxoHw

NN Example: XOR (Cont'd)

Example: XOR

Consider a system that produces training data that follows the $\mathrm{xor}(\cdot)$ function. The xor function accepts a 2-dimensional vector $\mathbf{x}$ with components x_1 and x_2 and returns 1 if $x_1 \neq x_2$. Concretely,

x_1	x_2	$\mathrm{xor}(\mathbf{x})$
0	0	0
0	1	1
1	0	1
1	1	0

$$J(\theta) = \frac{1}{2} \sum_{\mathbf{x}} (g(\mathbf{x}) - y(\mathbf{x}))^2$$

(Note, we wouldn't know $\mathrm{xor}(\mathbf{x})$, but we would have samples of corresponding inputs and outputs from training data. Hence, it may be better to simply replace $\mathrm{xor}(\mathbf{x})$ with $y(\mathbf{x})$ representing training examples.)

NN Example: XOR (Cont'd)

Example: XOR

Consider first a linear approximation of xor, via $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$. Then,

$$\begin{aligned} \frac{\partial J(\mathbf{w}, b)}{\partial \mathbf{w}} &= \sum_{\mathbf{x}} (\mathbf{w}^T\mathbf{x} + b - y(\mathbf{x}))\mathbf{x} \\ \frac{\partial J(\mathbf{w}, b)}{\partial b} &= \sum_{\mathbf{x}} (\mathbf{w}^T\mathbf{x} + b - y(\mathbf{x})) \end{aligned}$$

Equating these to 0, we arrive at:

$$(w_1 + b - 1)\begin{bmatrix} 1 \\ 0 \end{bmatrix} + (w_2 + b - 1)\begin{bmatrix} 0 \\ 1 \end{bmatrix} + (w_1 + w_2 + b)\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

These two equations can be simplified as:

1

$$\begin{aligned} (w_1 + b - 1) + (w_1 + w_2 + b) &= 0 \\ (w_2 + b - 1) + (w_1 + w_2 + b) &= 0 \end{aligned}$$

These equations are symmetric, implying $w_1 = w_2 = w$. This means:

$$3w + 2b - 1 = 0 \implies b = \frac{1 - 3w}{2}$$

NN Example: XOR (Cont'd)

Example: XOR

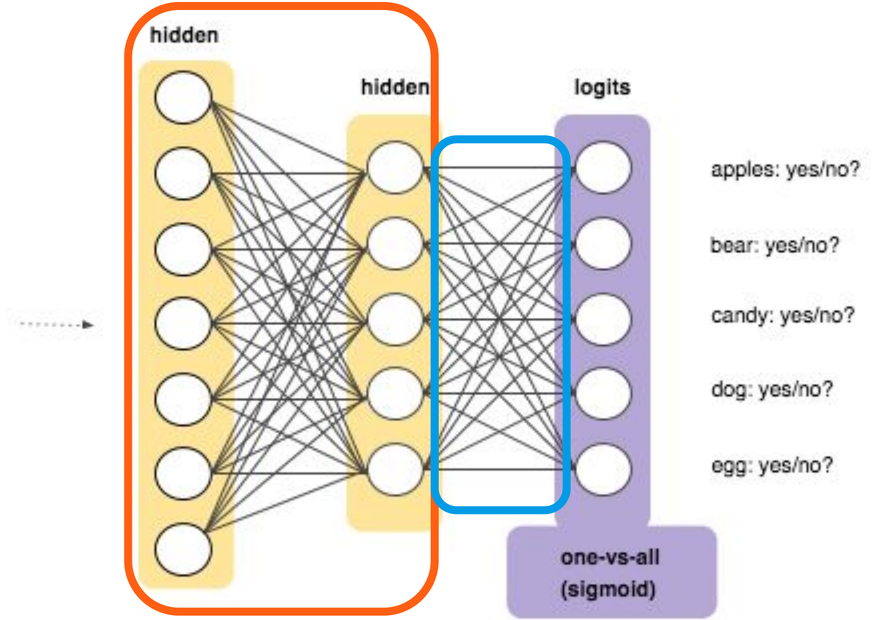
Now let's consider using a two-layer neural network, with the following equation:

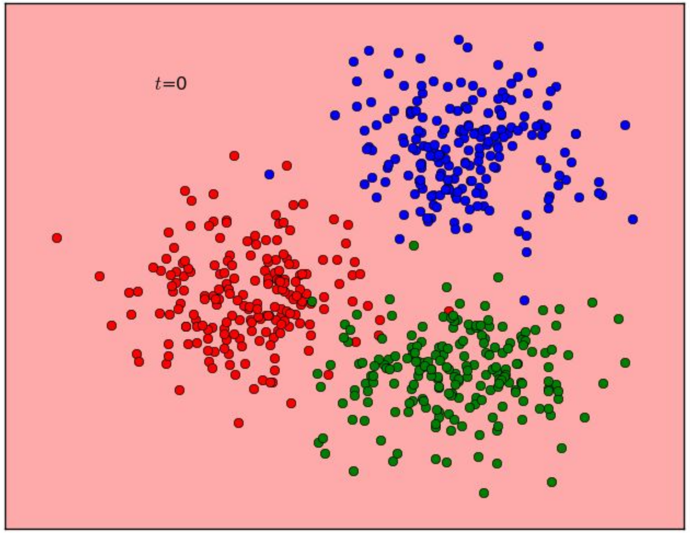
$$g(\mathbf{x}) = \mathbf{w}^T \max(0, \mathbf{W}^T\mathbf{x} + \mathbf{c}) + b$$

We haven't yet discussed how to optimize these parameters, but the point here is to show that by introducing a simple nonlinearity like $f(x) = \max(0, x)$, we can now solve the $\operatorname{xor}(\cdot)$ problem. Consider the solution:

$$\begin{aligned} \mathbf{W} &= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \\ \mathbf{c} &= [0, -1]^T \\ \mathbf{w} &= [1, -2]^T \end{aligned}$$

Multiclass Classification





5 separate **binary classifiers**

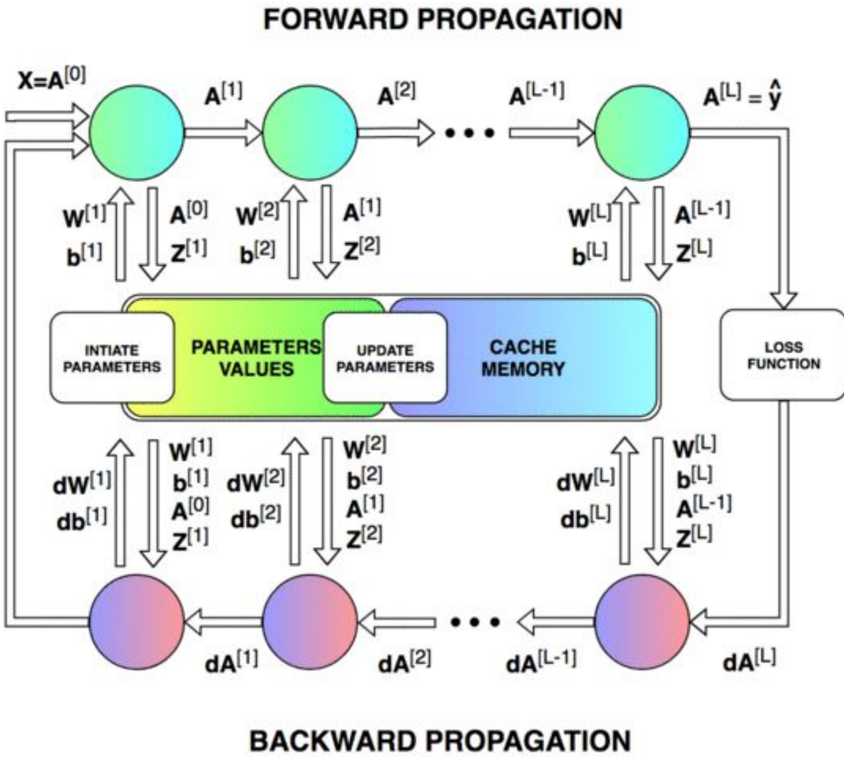
Key: **sharing the same hidden layers** with **different weights at the end**

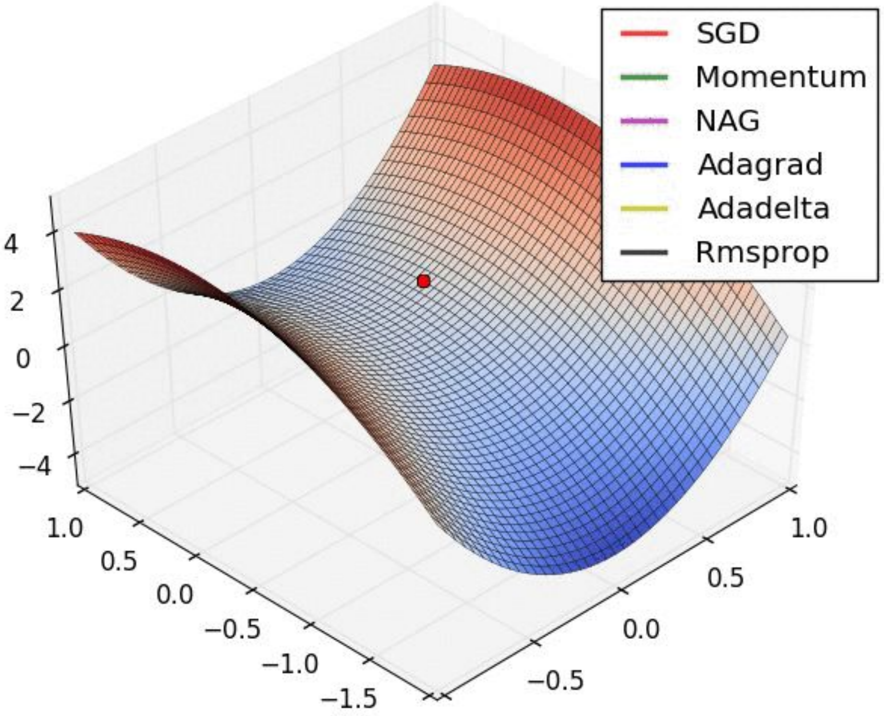
Question: Pros and cons?

https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/one-vs-all

http://www.briandolhansky.com/blog/2013/9/23/artificial-neural-nets-linear-multiclass-part-3

Neural Networks: Backpropagation



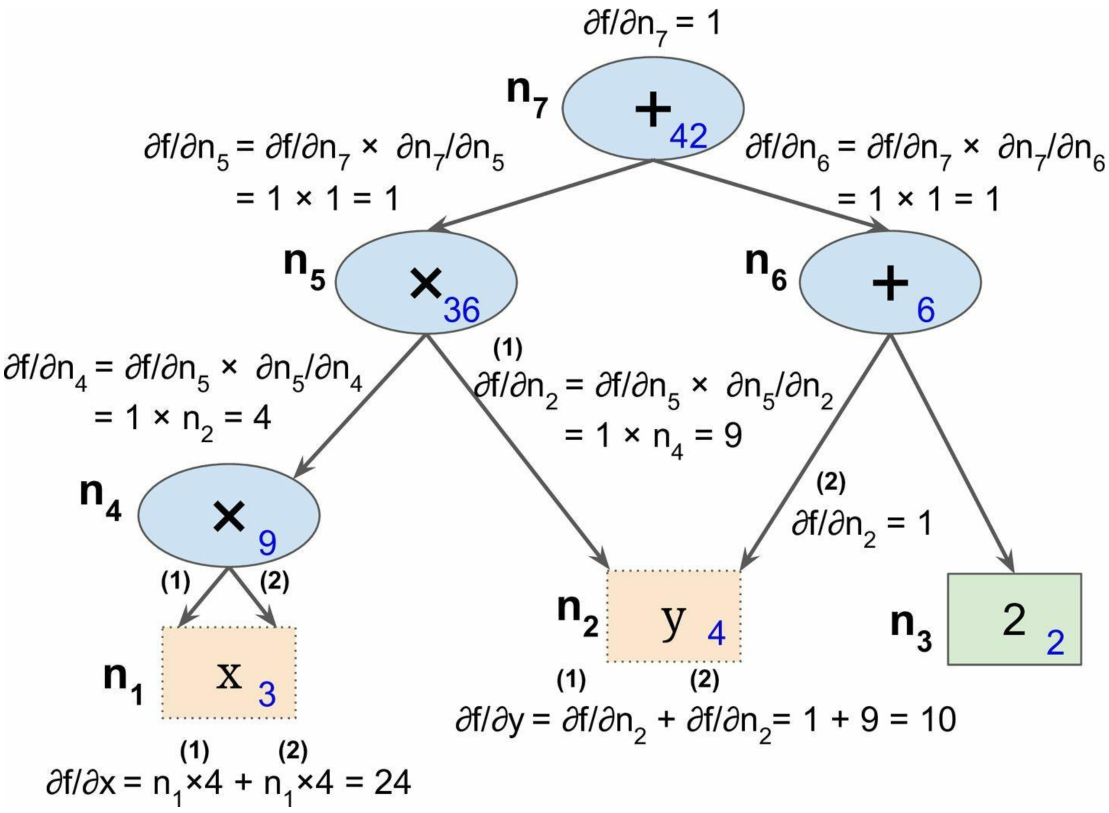


https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

Neural Networks: Backpropagation

- A simple example to understand the intuition
- `f(x,y) = x²y + y + 2`
- Forward pass:
  - `x=3, y=4 → f(3,4)=42`
- Backward pass:
  - Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n_i} \times \frac{\partial n_i}{\partial x}$$

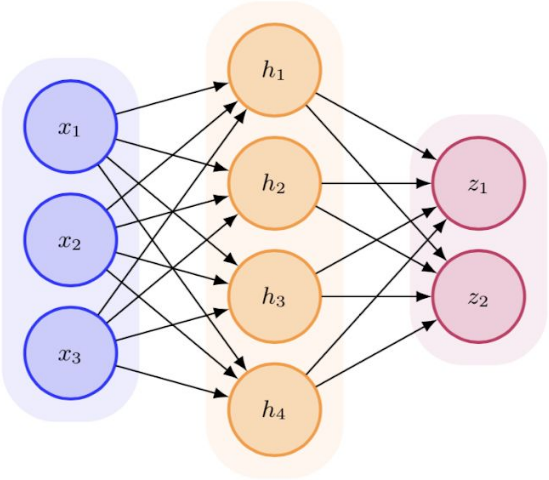


Another better demo: http://colah.github.io/posts/2015-08-Backprop/

2-Layer NN Example

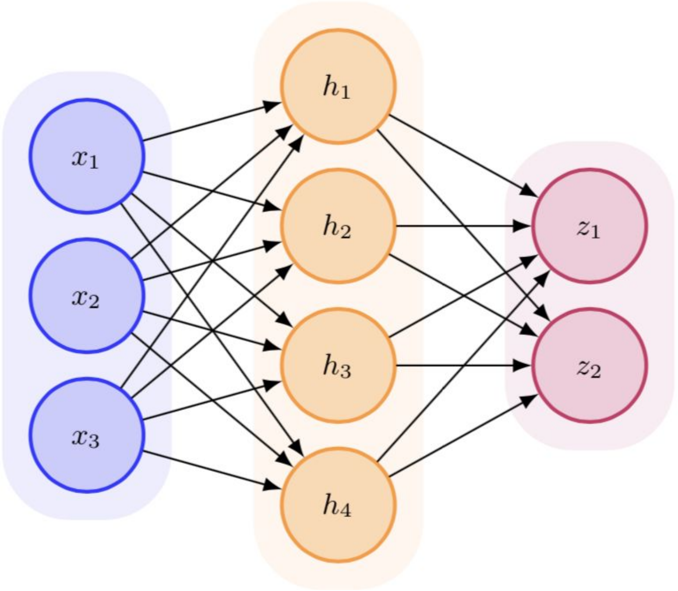
Neural network architecture

An example 2-layer network is shown below.



Here, the three dimensional inputs ($\mathbf{x} \in \mathbb{R}^3$) are processed into a four dimensional intermediate representation ($\mathbf{h} \in \mathbb{R}^4$), which are then transormed into the two dimensional outputs ($\mathbf{z} \in \mathbb{R}^2$).

2-Layer NN Example



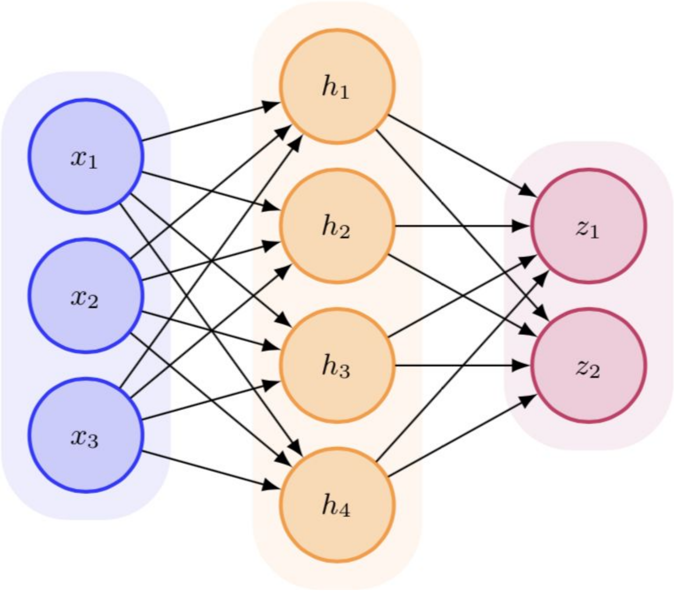
- Layer 1: $\mathbf{h}_1 = f(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
- Layer 2: $\mathbf{h}_2 = f(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$
- $\vdots$
- Layer N: $\mathbf{z} = \mathbf{W}_N\mathbf{h}_{N-1} + \mathbf{b}_N$

Questions:

1. Neural network model (in equations)
2. Number of neurons?
3. Number of weight parameters / bias parameters / total learnable parameters?

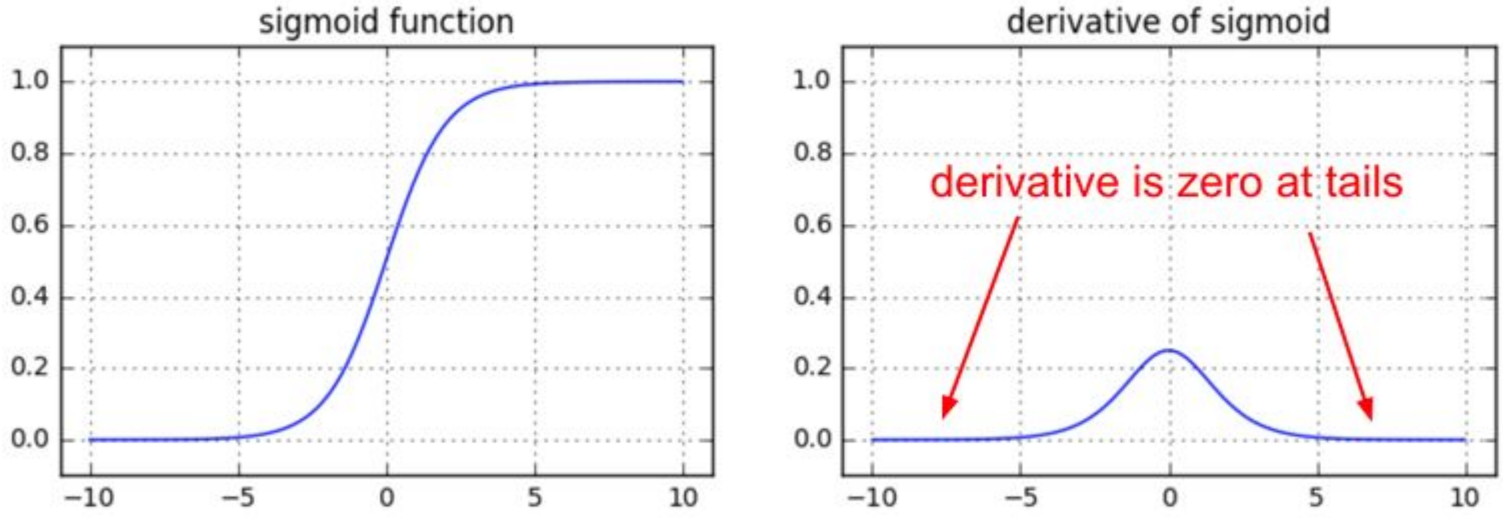
2-Layer NN Example

Demo in class : Back propagation for a 2-layer network



Why understanding backpropagation?

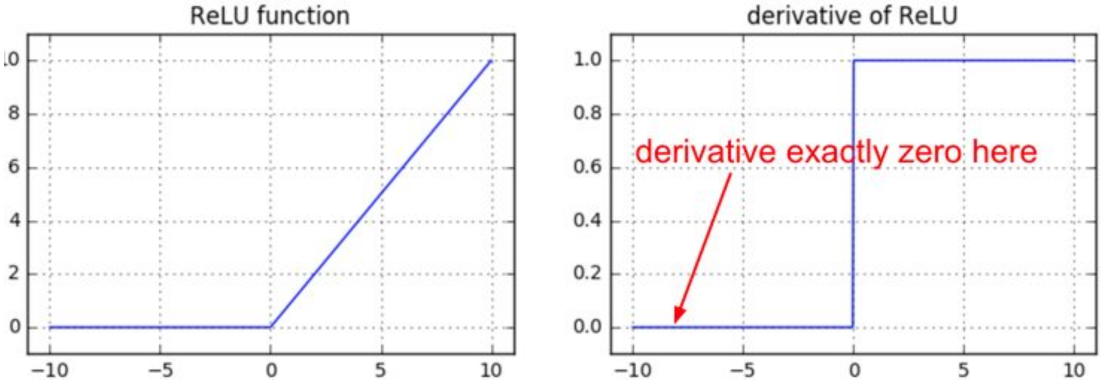
- “Why do we have to write the backward pass when frameworks in the real world, such as `TensorFlow/PyTorch`, compute them for you automatically?”
- Vanishing gradients on Sigmoids



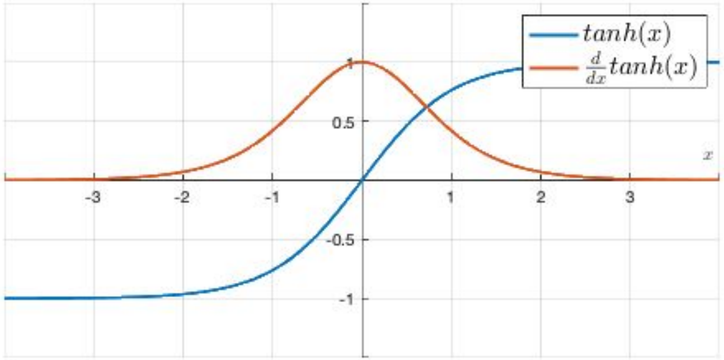
https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

Why understanding backpropagation?

- ReLUs



- tanh



Why understanding backpropagation?

- Examples of activation function: Sigmoid, ReLU, leaky ReLU, **tanh**, etc
- Properties we focus:
  - Differentiable
  - Range: Whether saturated or not? (
  - Whether zero-centered or not?
- Activation function family
  - Wiki: https://en.wikipedia.org/wiki/Activation_function

NN: Backpropagation Reading List

- Backpropagation (CS 231N at Stanford)
  - https://cs231n.github.io/optimization-2/
  - https://www.youtube.com/watch?v=i94OvYb6noo
- (Optional) Matrix-Level Operation:
  - https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c

NN: Number of iterations to converge

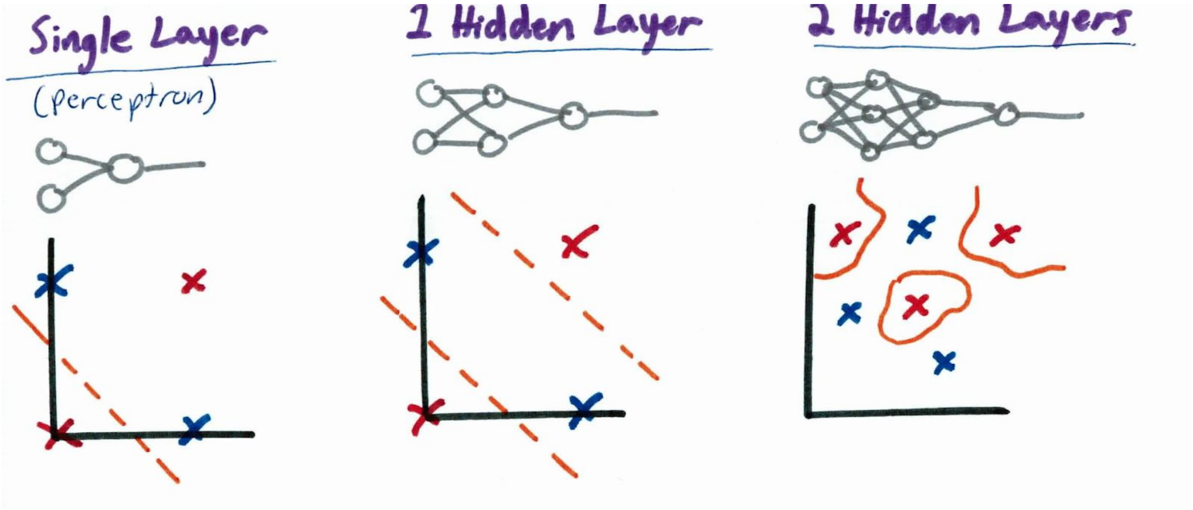
- Architecture/Meta-parameters of the network, e.g. # layers, activation
- Quality of training data (input-output correlation, normalization, noise cleansing, class distribution/imbalance)
- Random initialization of the parameters/weights
- Optimization algorithm, e.g. SGD, Adam, etc.
- Learning rate
- Batch size
- (In practice) Implementation quality (Bug-free? Optimized?)

https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e
https://www.quora.com/Machine-Learning-What-are-some-tips-and-tricks-for-training-deep-neural-networks

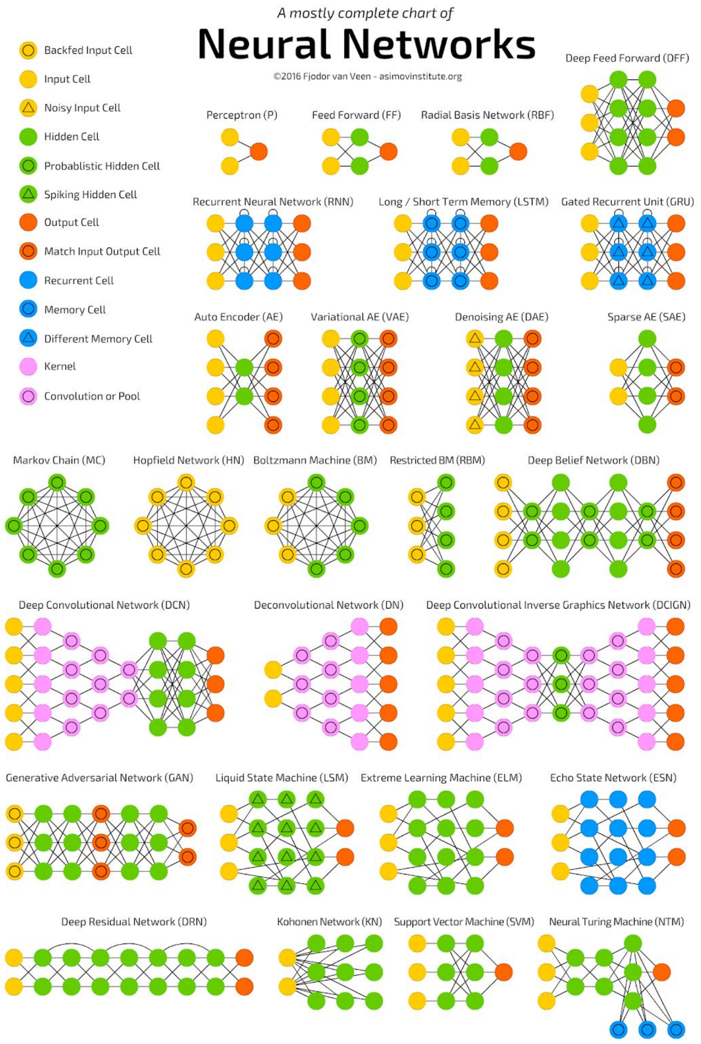
NN Summary: Pros and Cons

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks
  - Deep neural network is powerful

NN Summary

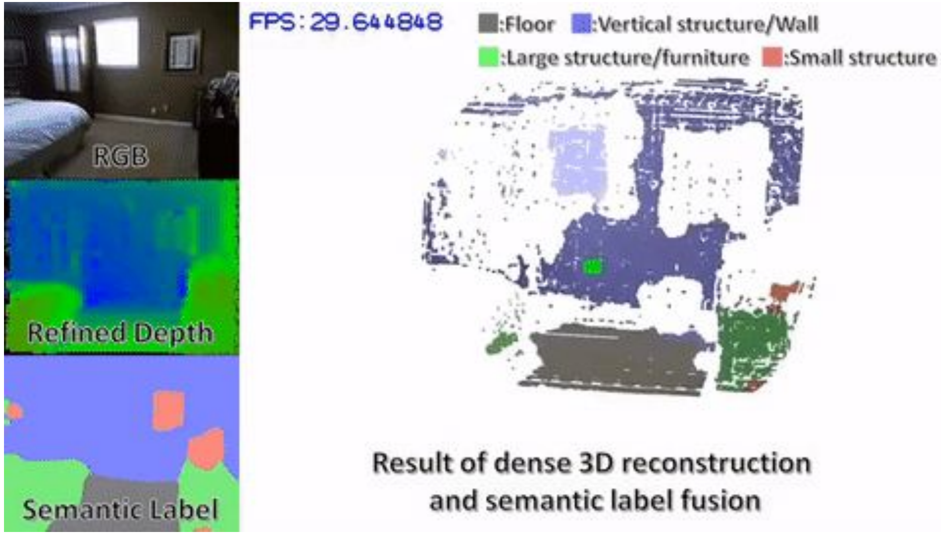


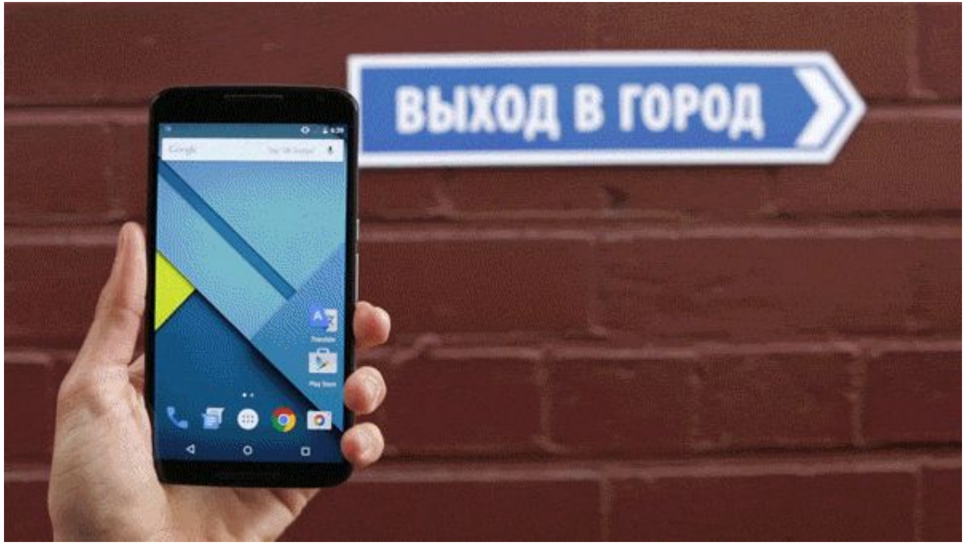
Flexibility



https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec27/pros-and-cons-of-neural-networks
http://kseow.com/nn/
https://towardsdatascience.com/hype-disadvantages-of-neural-networks-6af04904ba5b

NN Summary: Pros and Cons

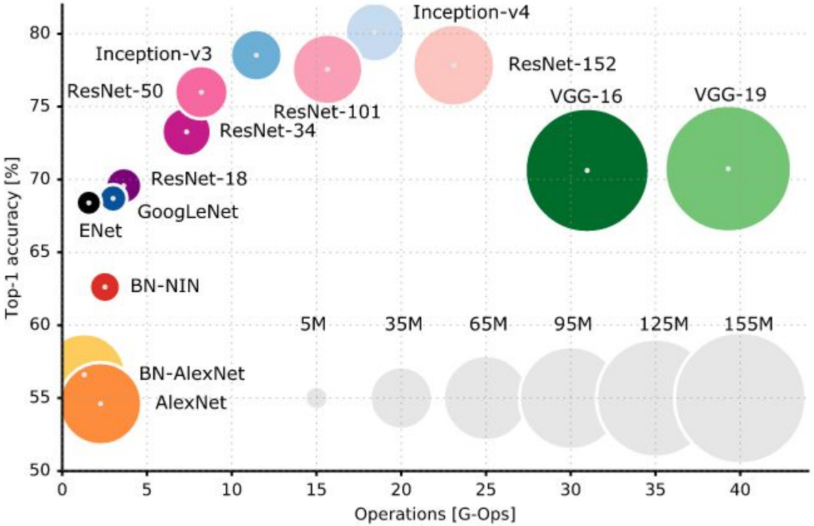




Efficiency (In many cases, prediction/inference/testing is fast)

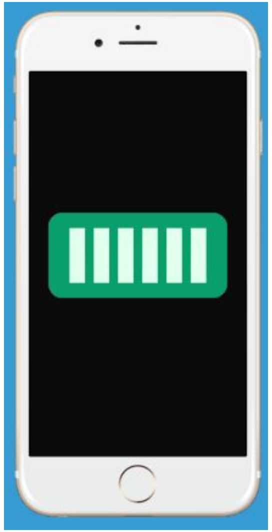
https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec27/pros-and-cons-of-neural-networks
http://www.luigifreda.com/2017/04/08/cnn-slam-real-time-dense-monocular-slam-learned-depth-prediction/
http://www.missqt.com/google-translate-app-now-supports-instant-voice-and-visual-translations/

NN Summary: Pros and Cons



We trained both our baseline models for about 600,000 iterations (33 epochs) – this is similar to the 35 epochs required by Nallapati et al.'s (2016) best model. Training took 4 days and 14 hours for the 50k vocabulary model, and 8 days 21 hours for the 150k vocabulary model. We found the pointer-generator model quicker to train, requiring less than 230,000 training iterations (12.8 epochs); a total of 3 days and 4 hours. In particular, the pointer-generator model makes much quicker progress in the early phases of training.

ments. This work was begun while the first author was an intern at Google Brain and continued at Stanford. Stanford University gratefully acknowl-



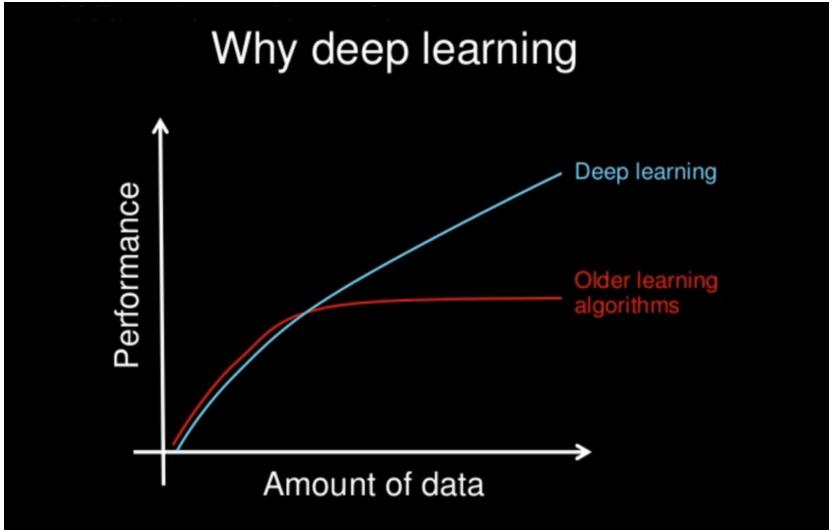
Efficiency (Big model → slow training, huge energy consumption (e.g. for cell phone))

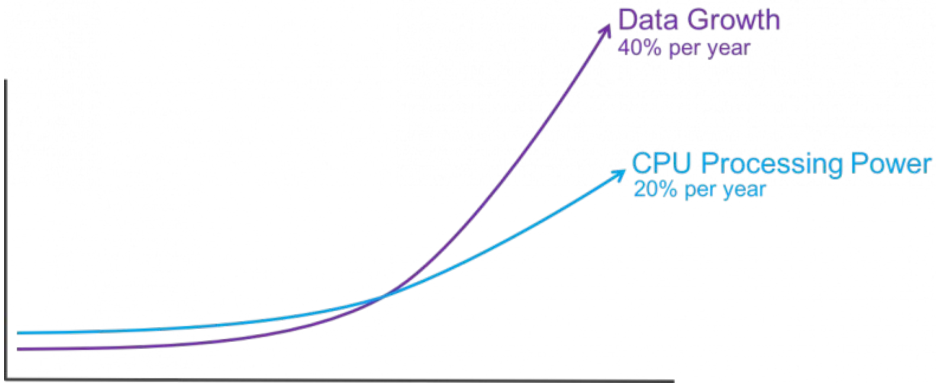
https://www.kdnuggets.com/2017/08/first-steps-learning-deep-learning-image-classification-keras.html/2

See, Abigail, Peter J. Liu, and Christopher D. Manning. "Get to the point: Summarization with pointer-generator networks." *arXiv preprint arXiv:1704.04368* (2017).

https://www.lifewire.com/my-iphone-wont-charge-what-do-i-do-2000147

NN Summary: Pros and Cons

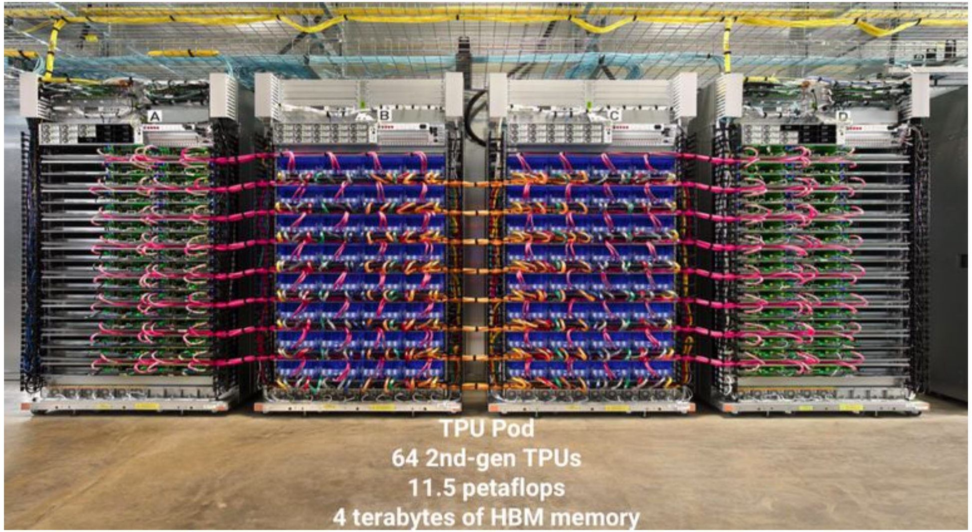




Data (Both a pro and a con)

https://towardsdatascience.com/hype-disadvantages-of-neural-networks-6af04904ba5b

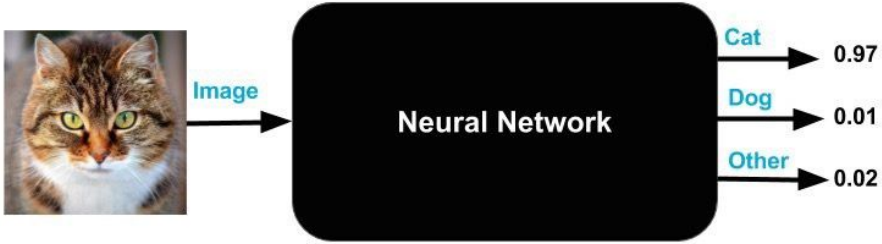
NN Summary: Pros and Cons



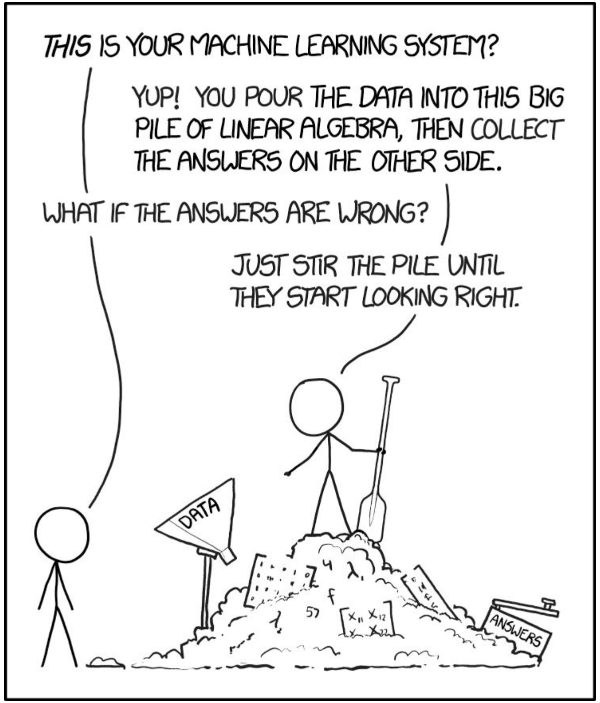
Computational Power (Both a pro and a con)

https://www.anandtech.com/show/10864/discrete-desktop-gpu-market-trends-q3-2016
https://www.zdnet.com/article/gpu-killer-google-reveals-just-how-powerful-its-tpu2-chip-really-is/

NN Summary: Pros and Cons



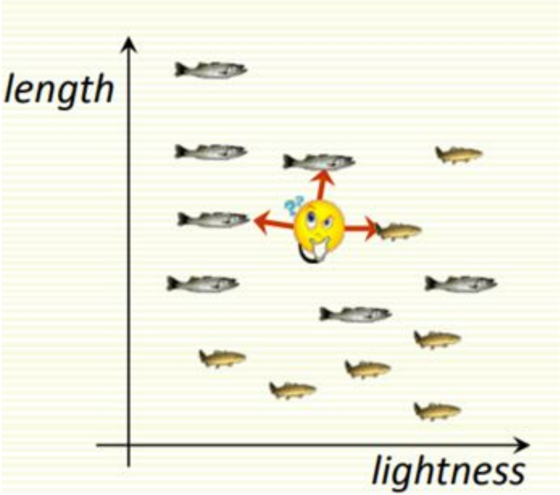
Black Box
Interpretability



https://towardsdatascience.com/hype-disadvantages-of-neural-networks-6af04904ba5b
https://xkcd.com/1838/

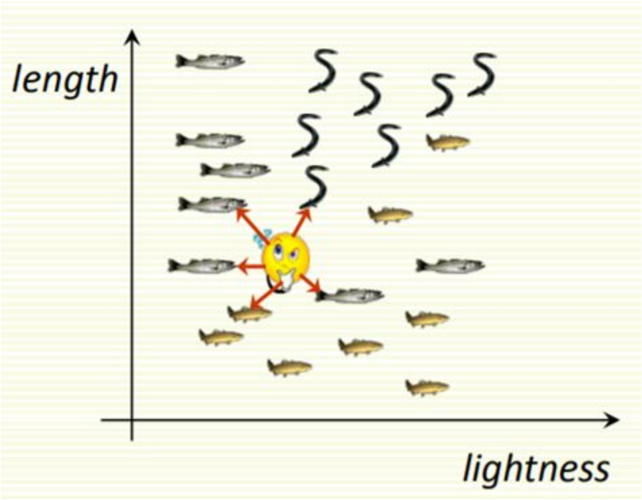
KNN

- Classify an unknown example with the most common class among K nearest examples
    - “Tell me who your neighbors are, and I’ll tell you who you are”
- Example
    - K = 3
    - 2 sea bass, 1 salmon
    - Classify as sea bass



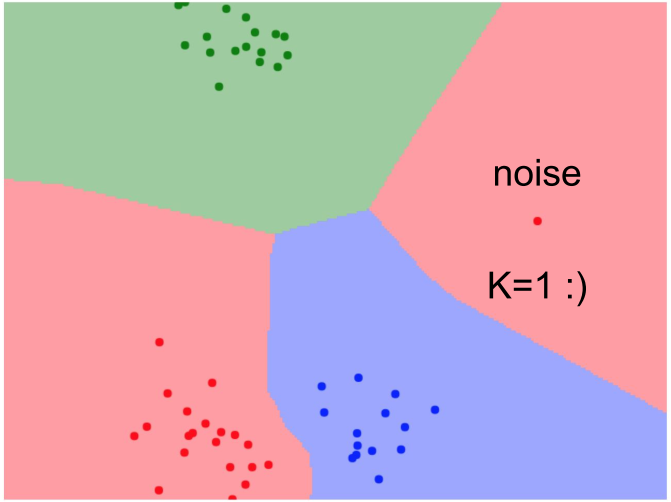
KNN: Multiple Classes

- Easy to implement for multiple classes
- Example for K = 5
  - 3 fish species: salmon, sea bass, eel
  - 3 sea bass, 1 eel, 1 salmon → classify as sea bass

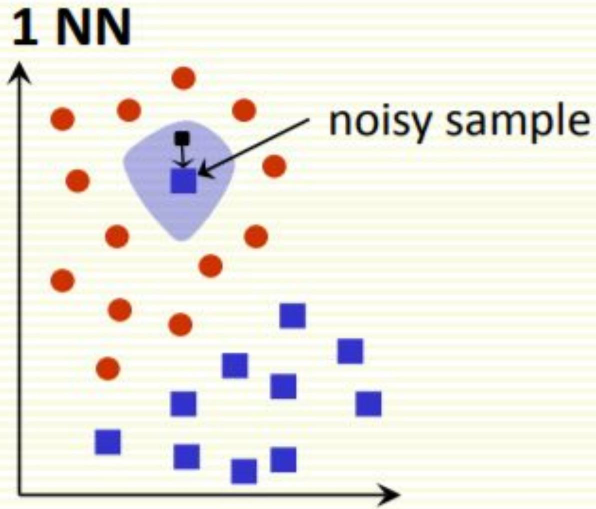


KNN: How to Choose K?

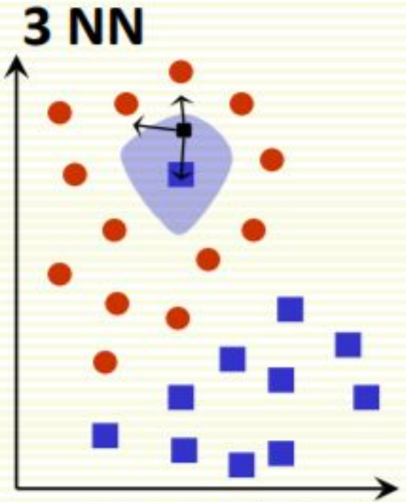
- In theory, if infinite number of samples available, the larger K, the better classification result you'll get.
- Caveat: all K neighbors have to be close
  - Possible when infinite # samples available
  - Impossible in practice since # samples if finite
- Should we "tune" K on training data?
  - Underfitting → Overfitting
- K = 1 → sensitive to "noise" (e.g. see right)



KNN: How to Choose K?



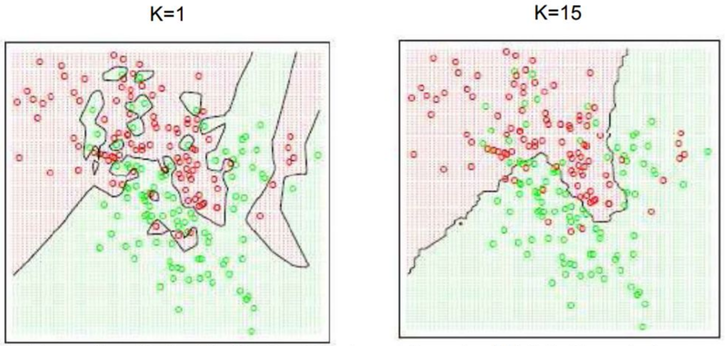
every example in the blue shaded area will be misclassified as the blue class



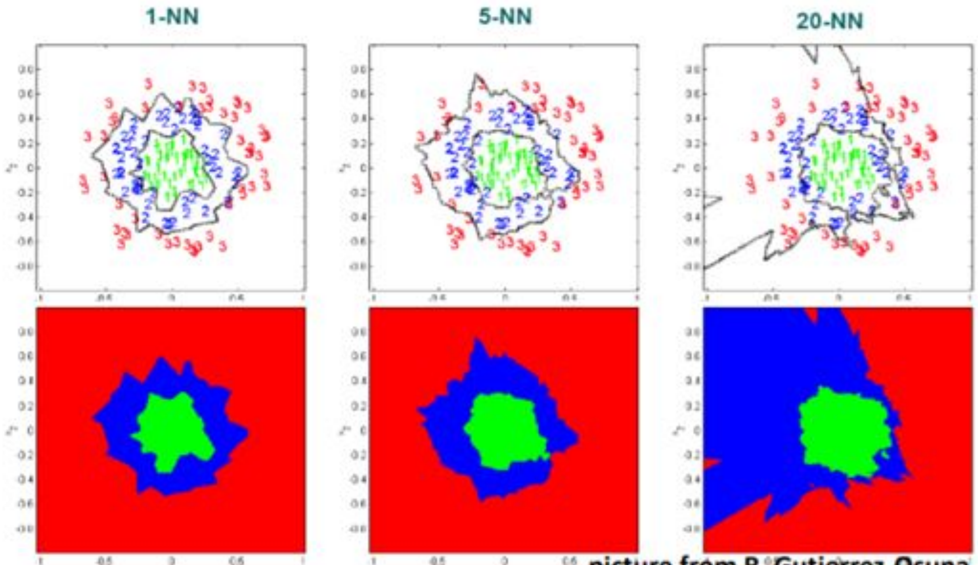
every example in the blue shaded area will be classified correctly as the red class

KNN: How to Choose K?

- Larger K gives smoother boundaries, better for generalization
  - Only if locality is preserved
  - K too large → looking at samples too far away that are not from the same class
- Can choose K through cross-validation



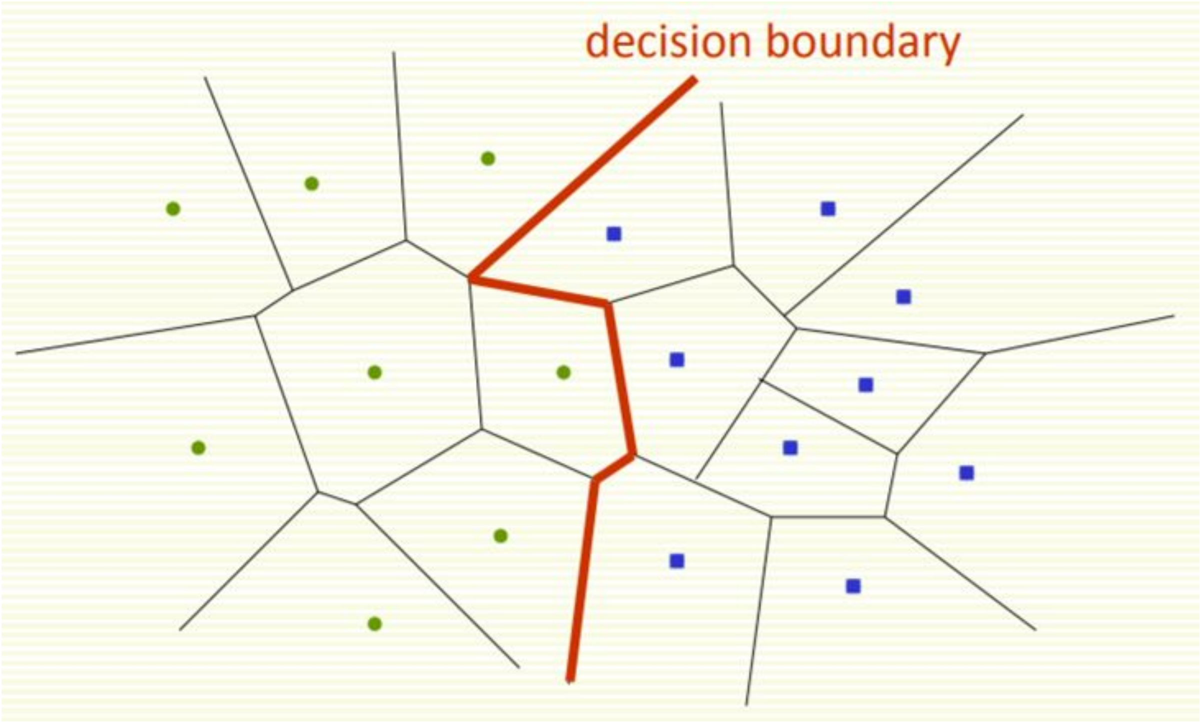
Figures from Hastie, Tibshirani and Friedman (Elements of Statistical Learning)



picture from R. Gutierrez-Osuna

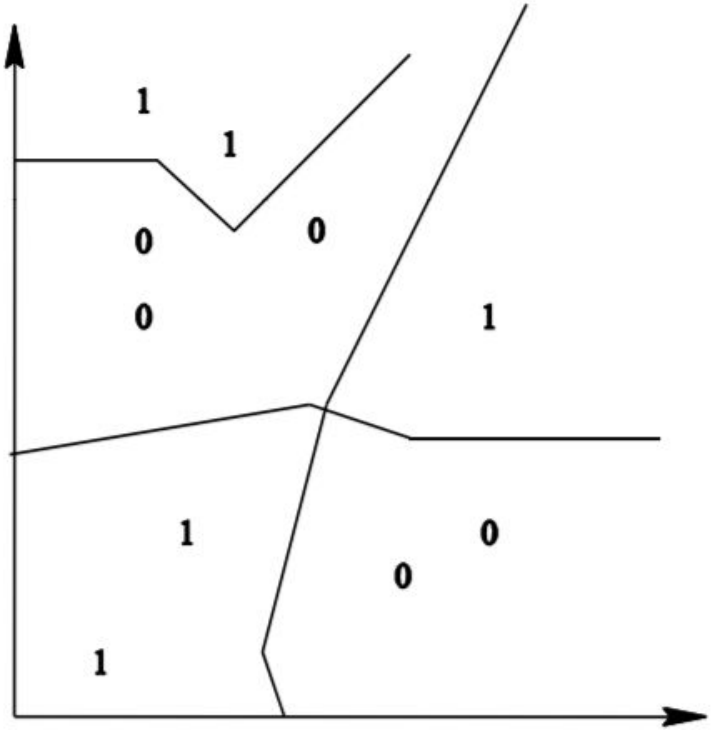
KNN: Decision Boundary

- Voronoi diagram



KNN: Decision Boundary

- Decision boundaries are formed by a subset of the Voronoi Diagram of the training data
- Each line segment is equidistant between two points of opposite class
- The more examples that are stored, the more fragmented and complex the decision boundaries can be.



KNN: Distance

- If we use Euclidean Distance to find the nearest neighbor:

$$D(a,b) = \sqrt{\sum_k (a_k - b_k)^2}$$

- Euclidean distance treats each feature as equally important
- Sometimes, some features (or dimensions) may be much more discriminative than other features

KNN: Distance

- Feature 1 gives the correct class: 1 or 2
- Feature 2 gives irrelevant number from 100 to 200
- Dataset: [1, 150], [2, 110]
- Classify [1, 100]

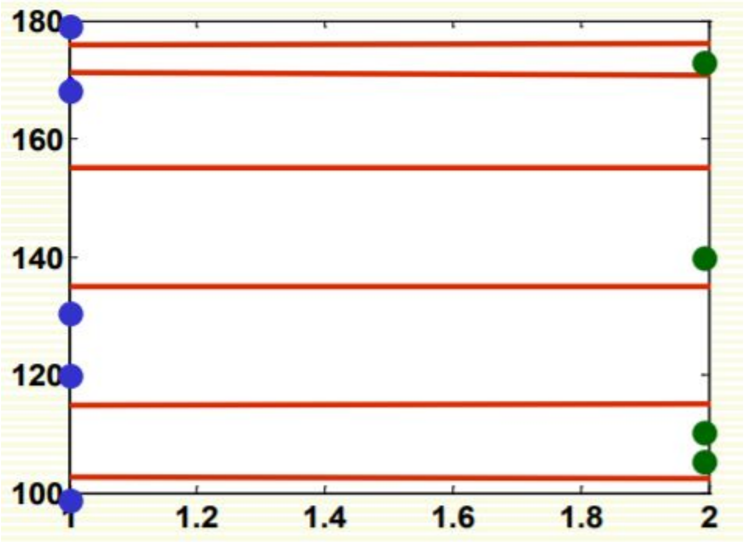
$$D\left(\begin{bmatrix}1\\100\end{bmatrix}, \begin{bmatrix}1\\150\end{bmatrix}\right) = \sqrt{(1-1)^2 + (100-150)^2} = 50$$

$$D\left(\begin{bmatrix}1\\100\end{bmatrix}, \begin{bmatrix}2\\110\end{bmatrix}\right) = \sqrt{(1-2)^2 + (100-110)^2} = 10.5$$

- Use Euclidean distance can result in wrong classification
- Dense Example can help solve this problem

KNN: Distance

- Decision boundary is in red, and is really wrong because:
  - Feature 1 is discriminative, but its scale is small
  - Feature gives no class information but its scale is large, which dominates distance calculation



KNN: Feature Normalization

- Normalize features that makes them be in the same scale
- Different normalization approaches may reflect the result
- Linear scale the feature in range [0,1]:

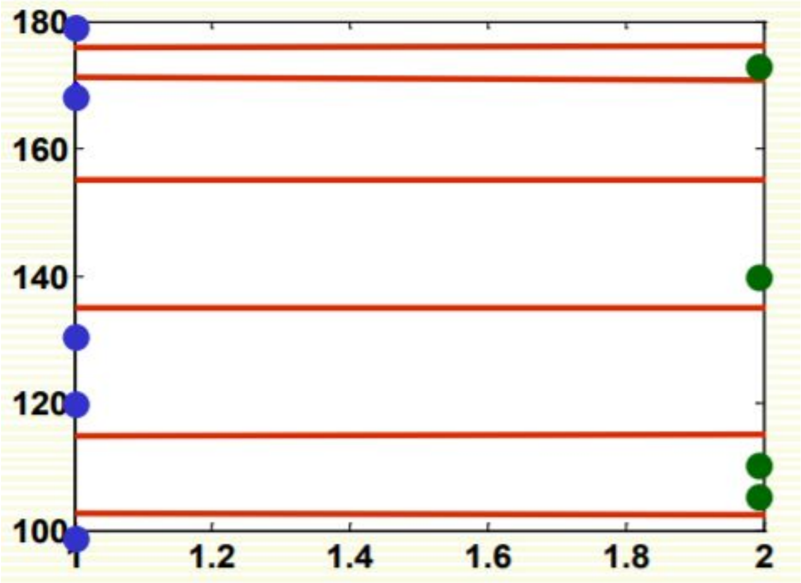
$$f_{new} = \frac{f_{\text{old}} - f_{\text{old}}^{\text{min}}}{f_{\text{old}}^{max} - f_{\text{old}}^{\text{min}}}$$

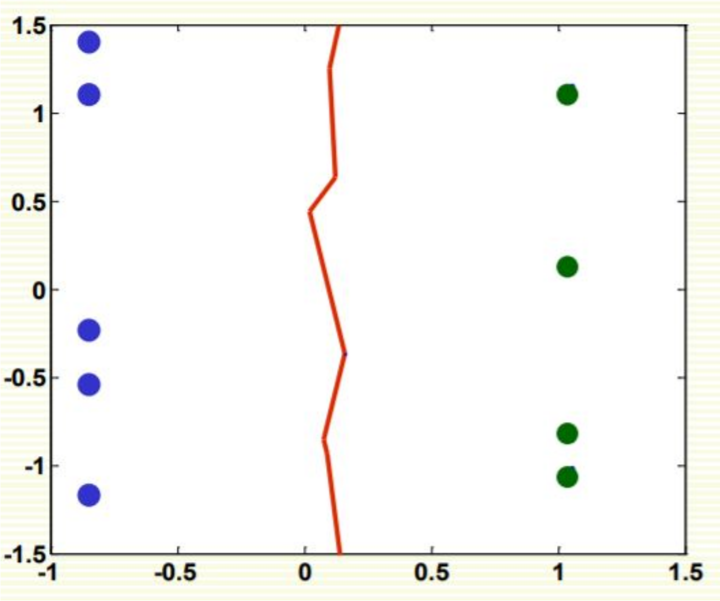
- Linear scale to 0 mean standard deviation 1(Z-score):

$$f_{new} = \frac{f_{\text{old}} - \mu}{\sigma}$$

KNN: Feature Normalization

- Result comparison non-normalized vs normalized





KNN: Feature Weighting

- Scale each feature by its importance for classification

$$D(a,b) = \sqrt{\sum_k w_k (a_k - b_k)^2}$$

- Use prior/domain knowledge to set the weight w
- Use cross-validation to learn the weight w

KNN: Computational Complexity

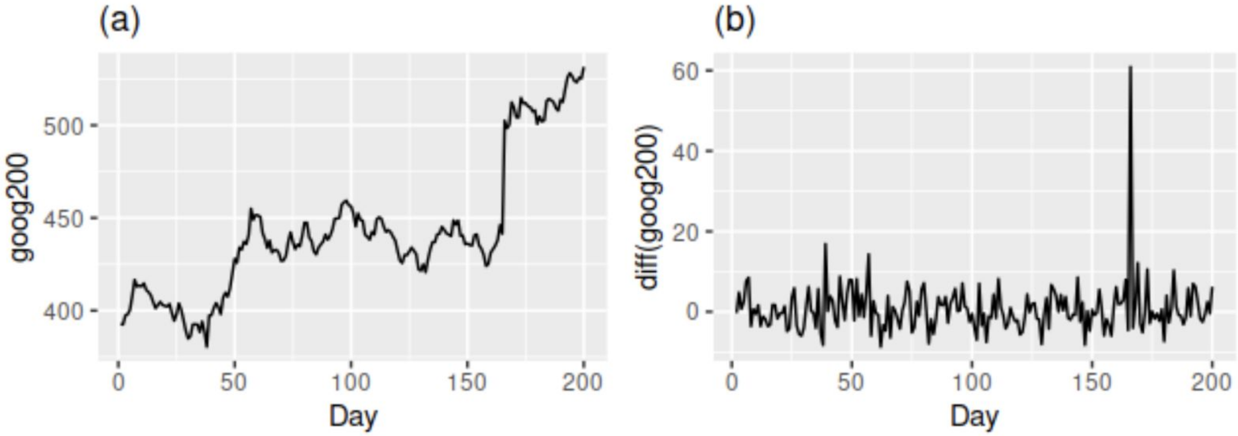
- Suppose ***n*** examples with dimension ***d***
- Complexity for kNN training?
- Complexity for kNN training?
  - For each point to be classified:
  - Complexity for computing distance to one example
  - Complexity for computing distances to all examples
  - Find ***k*** closest examples
- Is it expensive for a large number of queries, compared to logistic regression, SVM or neural network?

KNN: Summary

- Advantages:
  - Can be applied to the data from any distribution
  - The decision boundary is not necessarily to be linear
  - Simple and Intuitive
  - Good Classification with large number of samples
- Disadvantages:
  - Choosing k may be tricky
  - Test stage is computationally expensive
    - No training stage, time-consuming test stage
    - Usually we can afford long training step but fast testing speed
  - Need large number of examples for accuracy

Time Series Prediction Models

- Most models assume the timeseries to be **stationary**, i.e. it tends to wonder more or less uniformly about some fixed level. In practice, **differencing** timeseries to achieve stationary (i.e. instead of predicting cummulative value x_t, predict $\Delta x_t = x_t - x_{t-1}$.



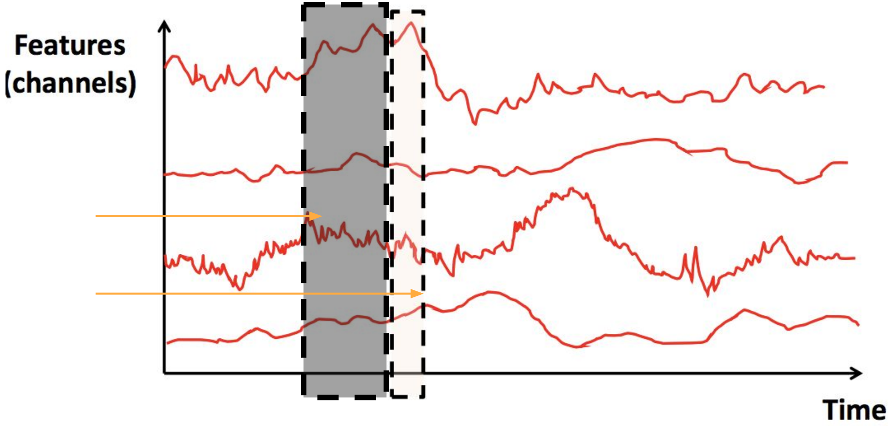
Time Series Prediction Models

- AR (autoregressive) model. An AR model of order p can be written as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

- This is similar to linear regression model when we view historical data as feature input.

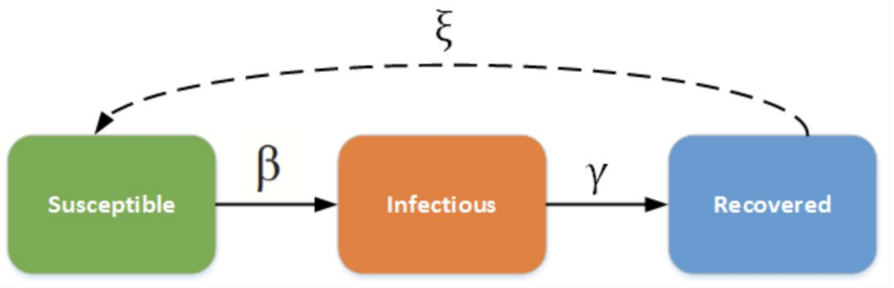
Historical data to be conditioned on data to be predicted.

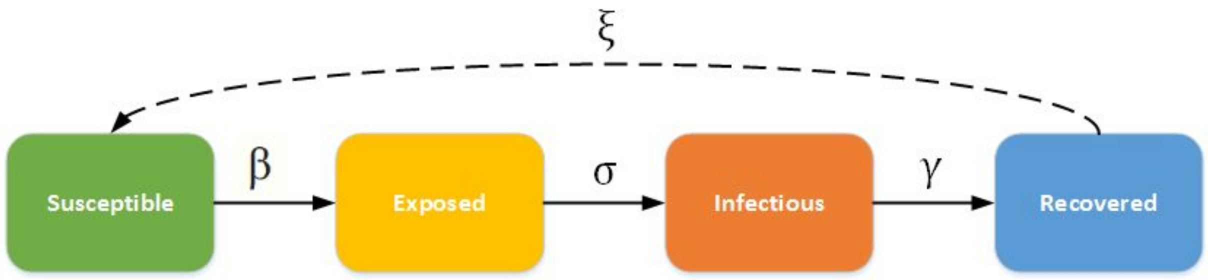


Choose a proper window size!

Time Series Prediction Models

- Diffusion model (SIR, SEIR , etc). Model continuous dynamics using ODE [1].





https://docs.idmod.org/projects/emod-generic/en/latest/model-sir.html

Time Series Prediction Models

- Deep Learning Based [1] models:

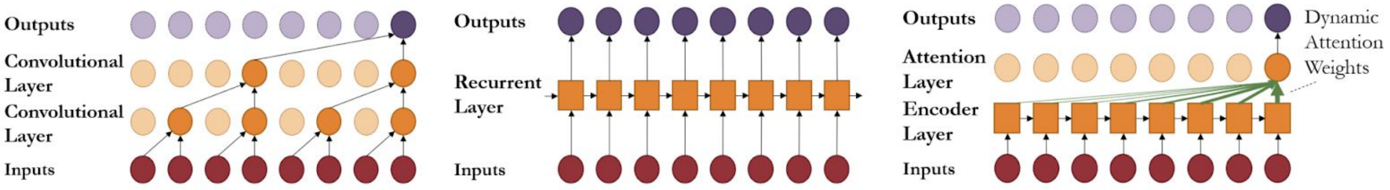
$$\hat{y}_{i,t+1} = f(y_{i,t-k:t}, \boldsymbol{x}_{i,t-k:t}, \boldsymbol{s}_i)$$

Label

Dynamic Features

Static Features

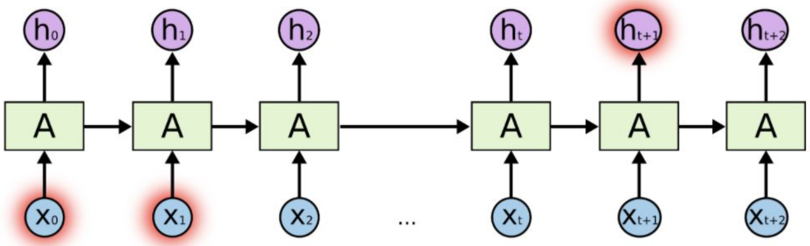
- Examples

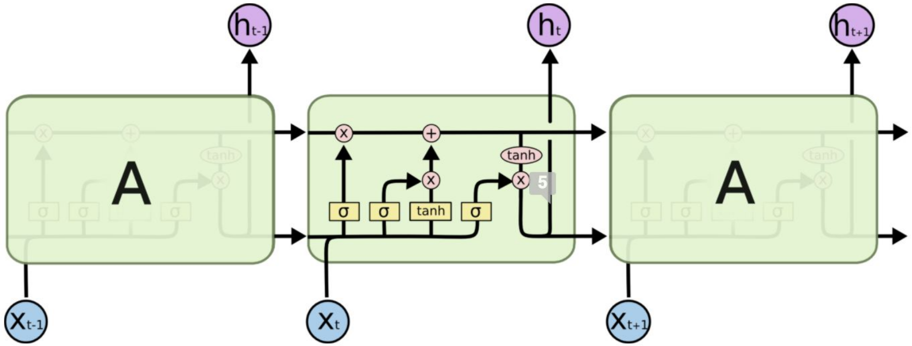


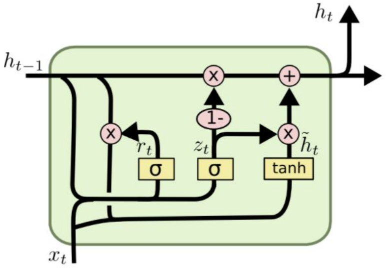
(a) CNN Model. (b) RNN Model. (c) Attention-based Model.

https://arxiv.org/pdf/2004.13408.pdf

Time Series Prediction Models: RNN







$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Thank you!

Q & A