UCLA Samueli Computer Science

CS M146 Discussion: Week 5
Overfitting and Regularization, Neural Nets

Junheng Hao

Friday, 02/05/2021

Roadmap

- Announcement
- Overfitting and Regularization
- Neural Nets

Announcements

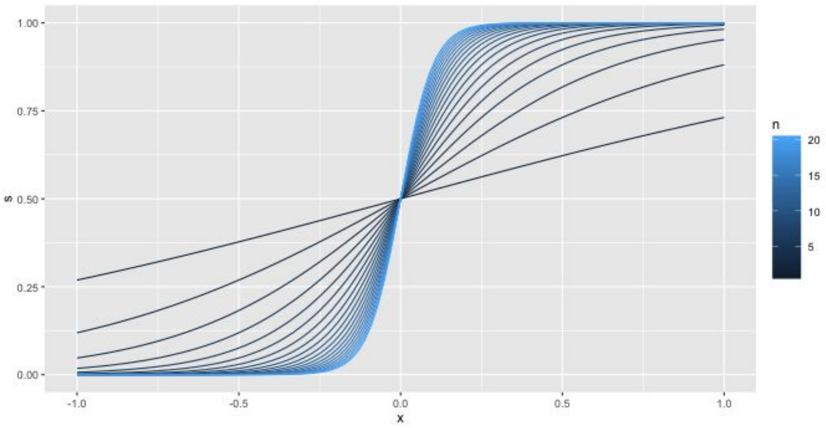
- **5:00 pm PST, Feb 5 (Friday):** Weekly Quiz 5 released on Gradescope.
- **11:59 pm PST, Feb 7 (Sunday):** Weekly quiz 5 closed on Gradescope!
  - Start the quiz before **11:00 pm PST, Feb 7** to have the full 60-minute time
- **Problem set 2** released on CCLE, submission on Gradescope.
  - Please assign pages of your submission with corresponding problem set outline items on GradeScope.
  - You do not need to submit code, only the results required by the problem set
  - Due on **11:59pm PST, Feb 12 (Friday)**

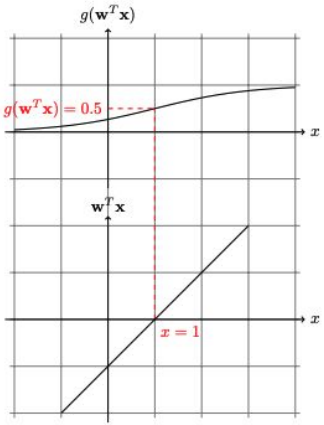
About Quiz 5

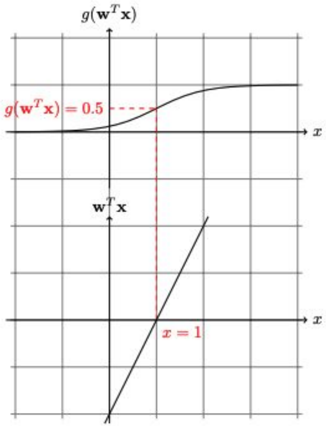
- Quiz release date and time: **Feb 5, 2021 (Friday) 05:00 PM PST**
- Quiz due/close date and time: **Feb 7, 2021 (Sunday) 11:59 PM PST**
- You will have up to **60 minutes** to take this exam. → Start before **11:00 PM** Sunday
- You can find the exam entry named "Week 4 Quiz" on GradeScope.
- Topics: **Overfitting, Regularization, Neural Nets (without Backprop)**
- Question Types
    - True/false, multiple choices
    - Some questions may include several subquestions.
- Some light calculations are expected. Some scratch paper and one scientific calculator (physical or online) are recommended for preparation.

Clarification: Logistic Regression Convergence

Logistic regression (without regularization) **cannot** converge on a linearly separable dataset.







Campuswire Post: https://campuswire.com/c/GB5E561C3/feed/230
Reference: https://www.cscu.cornell.edu/news/statnews/82_lgsbias.pdf

Clarification: Logistic Regression Convergence

```
X, y = np.array([[-1],[1]]), np.array([0,1]) # train data
```

In sklearn, you have solver options as `newton-cg, lbfgs, liblinear, sag, saga`.

Then train a logistic regression model **without** penalty

```
clf = LogisticRegression(random_state=0, penalty='none', solver='sag', max_iter=1000).fit(X, y) # or other solver except 'liblinear'
```

You may notice different solvers may result in different w ranging from 5 to 10 (printed by `clf.coef_`). Sometimes you might have a convergence error as follows:

```
clf=None
# solver = 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'
clf = LogisticRegression(random_state=12, penalty='none', solver='sag', max_iter=1000, verbose=10).fit(X, y)
clf.coef_, clf.intercept_, clf.n_iter_
```

```
max_iter reached after 0 seconds
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/Users/junhenghao/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/_sag.py:330: ConvergenceWarning: The
max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
```

```
(array([[8.29936548]]), array([0.00109003]), array([1000], dtype=int32))
```

And again train a logistic regression model **with** L2 penalty

```
clf = LogisticRegression(random_state=0, penalty='l2', solver='lbfgs').fit(X, y) # L2 used
```

This time, you will find different solvers converges to $w = 0.675$.

Colab Link: https://colab.research.google.com/drive/1HrmthtXmg2PQ_9BHry1zrePWnSs2iQLn?usp=sharing

Overfitting

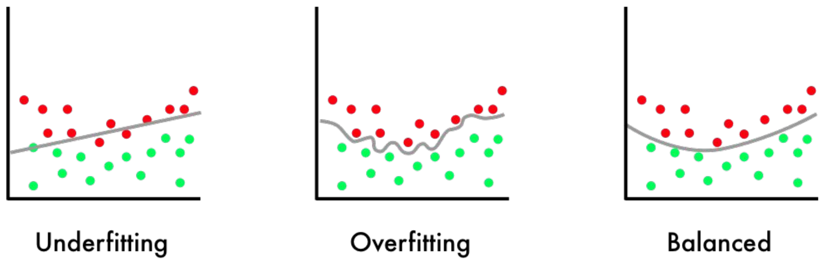
Key Questions:

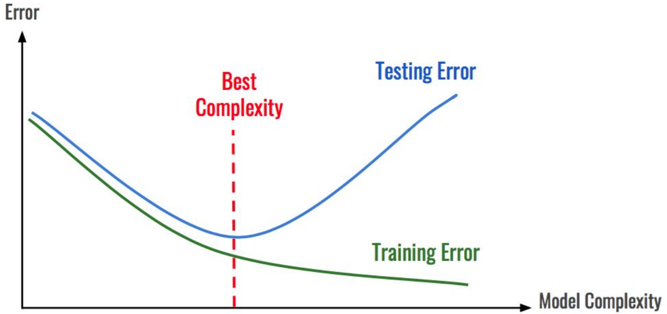
- How to identify overfitting?
- How to avoid overfitting?

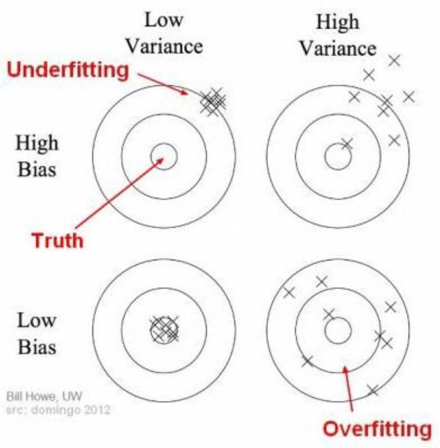
Credit: https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42

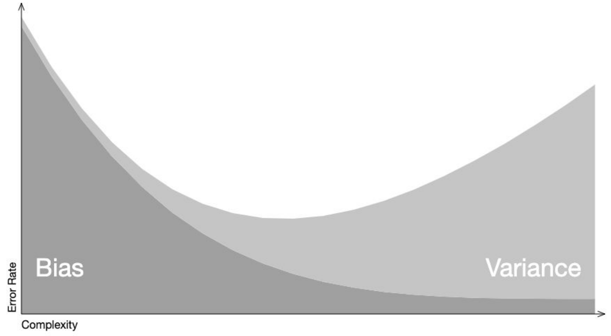
	Low *Training* Error	High *Training* Error
Low *Testing* Error	The model is learning!	Probably some error in your code. Or you've created a *psychic* AI.
High *Testing* Error	OVERFITTING	The model is not learning.

Overfitting: Polynomial Regression









Overfitting

Overfitting

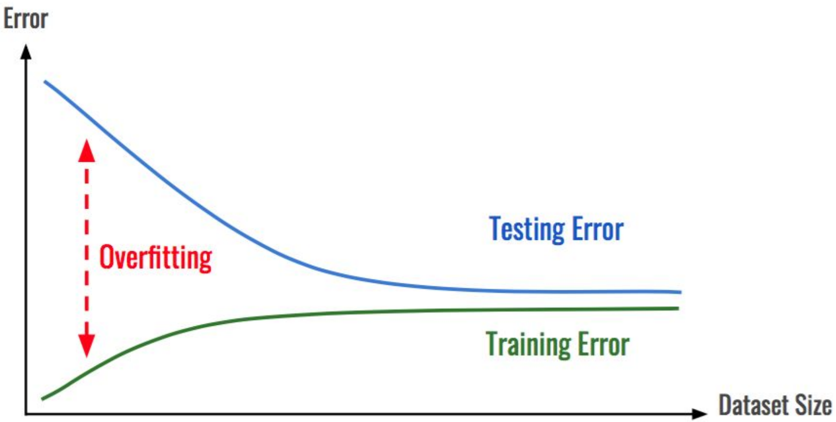
Data Size

Model Setting

Regularization

Overfitting: Data solution

- Collecting more data
- Data augmentation



Overfitting: Model Solution

- Avoid overfitting by changing model hyperparameter selection, from the mechanism and inductive bias of the model.

Overfitting Prescriptions

Our current model library

Decision Tree → ?

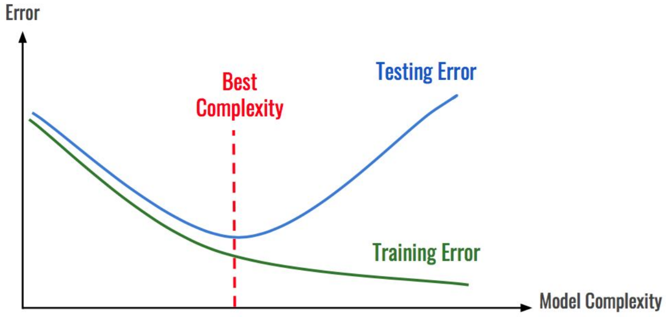
k-NN → ?

Logistic Regression → ?

Linear Regression → ?

Neural Nets → ?

...



Regularization

Linear Regression

- Model

$$\hat{y} = \boldsymbol{x}^T \boldsymbol{\beta} = x_1\beta_1 + x_2\beta_2 + \cdots + x_p\beta_p$$

- Original Objective

$$\min_{\boldsymbol{\beta}} \ J(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{x}^T\boldsymbol{\beta} - y)^2$$

- L2-Regularized Objective

$$\min_{\boldsymbol{\beta}} \ J(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{x}^T\boldsymbol{\beta} - y)^2 + \boxed{\frac{\lambda}{2}||\boldsymbol{\beta}||^2}$$

Logistic Regression

- Model

$$y = \sigma(X) = \frac{1}{1 + e^{-X^T\beta}}$$

- Original Objective

$$J(\beta) = -\frac{1}{n}\sum_{i}\left(y_i x_i^T\beta - \log\left(1 + \exp\{x_i^T\beta\}\right)\right)$$

- L2-Regularized Objective

$$J(\beta) = -\frac{1}{n}\sum_{i}\left(y_i x_i^T\beta - \log\left(1 + \exp\{x_i^T\beta\}\right)\right) + \boxed{\lambda\sum_{j}\beta_j^2}$$

Linear Regression + Regularization

$$\boxed{\min_{\boldsymbol{\beta}} \; J(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{x}^T\boldsymbol{\beta} - y)^2 + \frac{\lambda}{2}||\boldsymbol{\beta}||^2}$$

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{N}\boldsymbol{x}(\boldsymbol{x}^T\boldsymbol{\beta} - y) + \frac{\partial \lambda \; ||\boldsymbol{\beta}||^2}{\partial \boldsymbol{\beta}}$$

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{N}\boldsymbol{x}(\boldsymbol{x}^T\boldsymbol{\beta} - y) + \lambda\boldsymbol{\beta}$$

$$\boxed{\nabla_{\boldsymbol{\beta}} J(\boldsymbol{\beta}) = 0}$$

$$\nabla_{\boldsymbol{\beta}}\left(\frac{1}{2}||\boldsymbol{X\beta} - y||^2 + \frac{\lambda}{2}||\boldsymbol{\beta}||^2\right) = 0$$

$$\nabla_{\boldsymbol{\beta}}\left(\frac{1}{2}(\boldsymbol{X\beta} - y)^T(\boldsymbol{X\beta} - y) + \frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}\right) = 0$$

$$\boldsymbol{X}^T\boldsymbol{X\beta} - \boldsymbol{X}^T\boldsymbol{y} + \lambda\boldsymbol{I\beta} = 0$$

$$\boxed{\boldsymbol{\beta} = \left(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}}$$

About Norms (Vectors)

$$||\mathbf{w}||_1 = |w_1| + |w_2| + ... + |w_N|$$

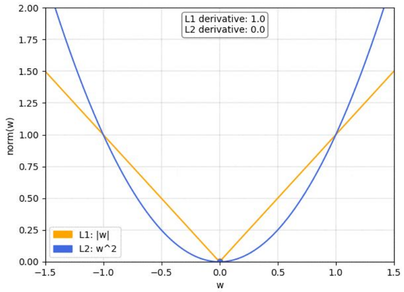
1-norm (also known as L1 norm)

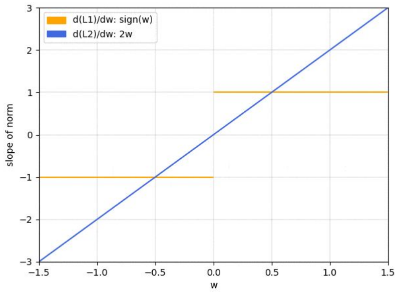
$$\|\mathbf{w}\|_2 = \left(|w_1|^2 + |w_2|^2 + \ldots + |w_N|^2\right)^{\frac{1}{2}}$$

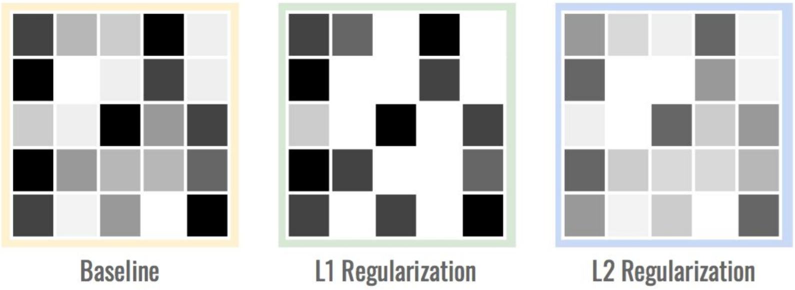
2-norm (also known as L2 norm or Euclidean norm)

$$\|\mathbf{w}\|_p = \left(|w_1|^p + |w_2|^p + \ldots + |w_N|^p\right)^{\frac{1}{p}}$$

p-norm







UCLA
Engineer Change.

About Norms (Vectors)

$$||\mathbf{w}||_1 = |w_1| + |w_2| + \ldots + |w_N|$$

1-norm (also known as L1 norm)

$$\|\mathbf{w}\|_2 = \left(|w_1|^2 + |w_2|^2 + \ldots + |w_N|^2\right)^{\frac{1}{2}}$$

2-norm (also known as L2 norm or Euclidean norm)

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

Loss function with L1 regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Loss function with L2 regularisation

How does L1/L2 regularization change the gradient descent step?

About "Entrywise" Norms (Matrix)

- L(2,1) Norm and L(p,q) Norm

$$\|A\|_{2,1} = \sum_{j=1}^{n} \|a_j\|_2 = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} |a_{ij}|^2 \right)^{\frac{1}{2}} \Rightarrow \|A\|_{p,q} = \left(\sum_{j=1}^{n} \left(\sum_{i=1}^{m} |a_{ij}|^p \right)^{\frac{q}{p}} \right)^{\frac{1}{q}}$$

- Frobenius norm (Hilbert–Schmidt norm)

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

- Max Norm

$$\|A\|_{\max} = \max_{ij} |a_{ij}|.$$

Credit: https://en.wikipedia.org/wiki/Matrix_norm

Effect on Regularization

Without L2 Regularization

```
clf = LogisticRegression(random_state=0, penalty='none', solver='lbfgs', max_iter=100).fit(X, y) # or other solver except 'liblinear'
print(clf.coef_, clf.intercept_, clf.n_iter_)
```

```
[[9.91926856]] [0.] [13]
```

```
clf = LogisticRegression(random_state=0, penalty='none', solver='newton-cg', max_iter=1000).fit(X, y) # or other solver except 'liblinear'
print(clf.coef_, clf.intercept_, clf.n_iter_)
```

```
[[10.20283614]] [0.] [9]
```

With L2 Regularization

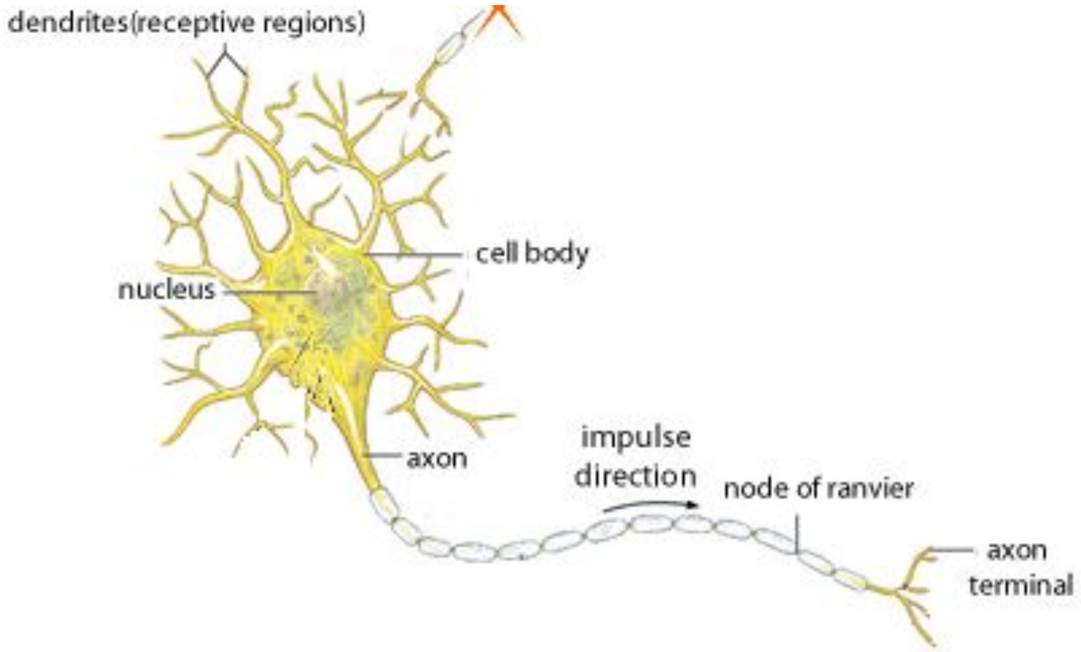
```
clf = LogisticRegression(random_state=0, penalty='l2', solver='lbfgs').fit(X, y)
print(clf.coef_, clf.intercept_, clf.n_iter_)
```

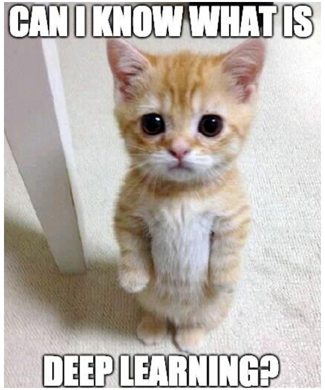
```
[[0.67483169]] [0.] [4]
```

```
clf = LogisticRegression(random_state=0, penalty='l2', solver='newton-cg').fit(X, y)
print(clf.coef_, clf.intercept_, clf.n_iter_)
```

```
[[0.67482829]] [0.] [2]
```

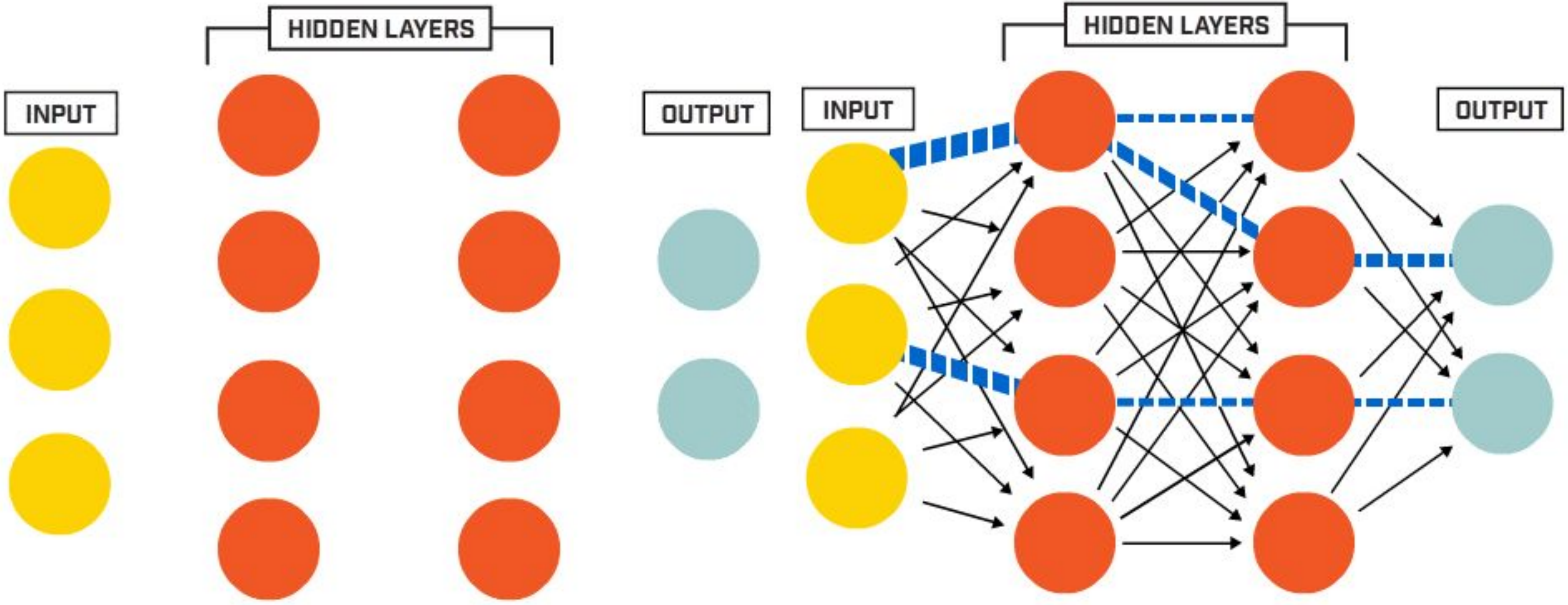
Neural Networks: Neuron/Perceptron





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

NN Example: XOR

- Which NN architecture corresponds to which function?

Y	1	**0**	**1**
	0	**0**	**0**
		0	1
		X	

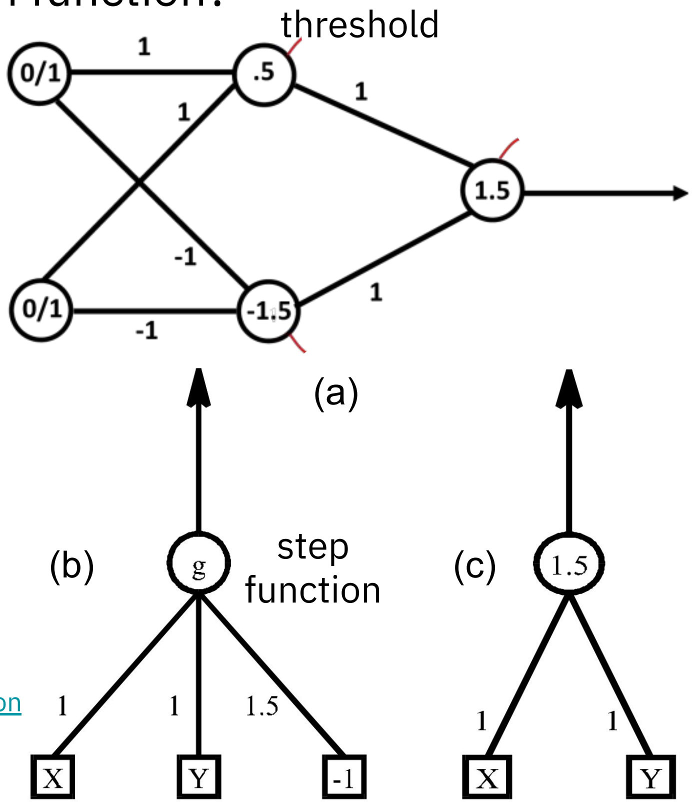
Table 1: Truth table for AND

Y	1	**1**	**1**
	0	**0**	**1**
		0	1
		X	

Table 2: Truth table for OR

Y	1	**1**	**0**
	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

UCLA Engineer Change.

NN Example: XOR

Y \ X	0	1
1	0	1
0	0	0

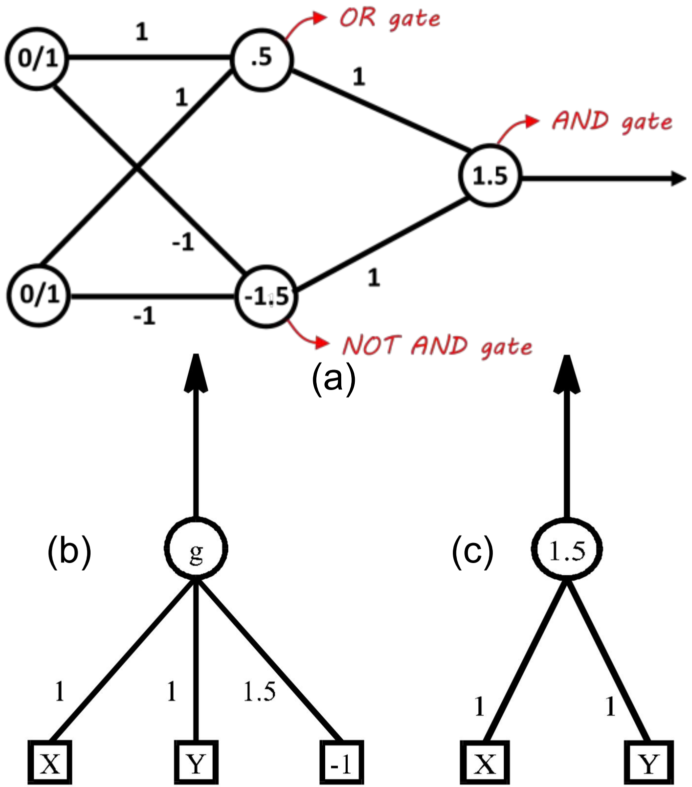
Table 1: Truth table for AND

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

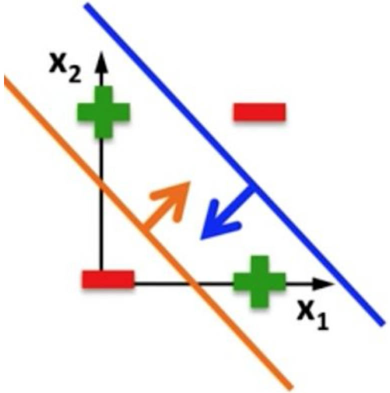
Table 3: Truth Table for XOR

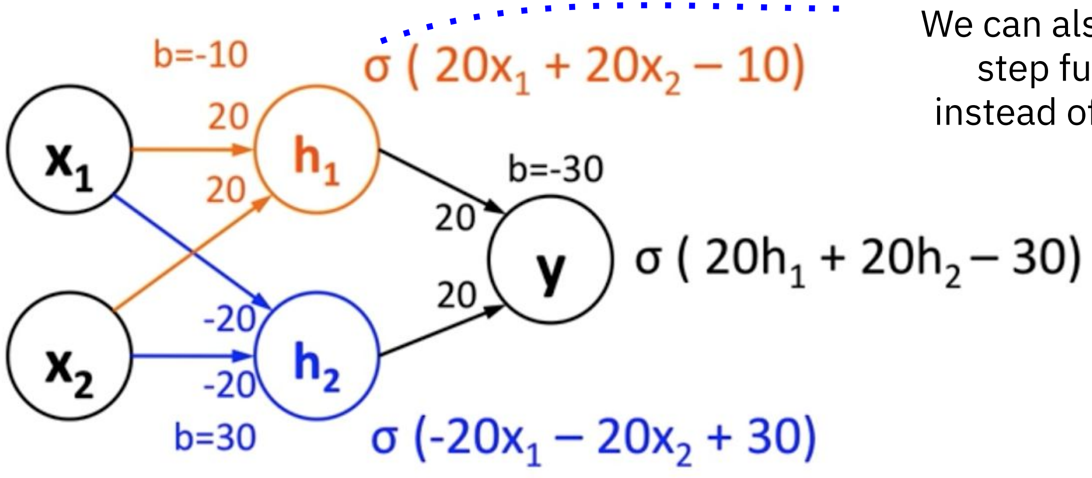


https://datascience.stackexchange.com/questions/11589/creating-neural-net-for-xor-function
http://yen.cs.stir.ac.uk/~kjt/techreps/pdf/TR148.pdf
https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b

NN Example: XOR

Linear classifiers cannot solve this





We can also use the 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

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

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

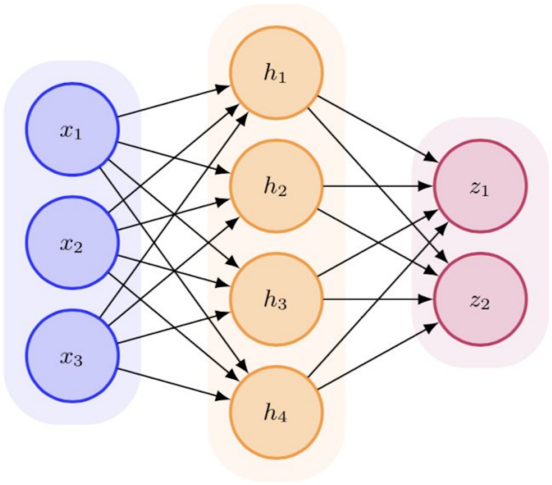
We haven't yet discussed how to optimize these parameters, but the point here is to show that by introducing a simple nonlinearity like $f(x) = \max(0, x)$, we can now solve the $\mathrm{xor}(\cdot)$ problem. Consider the solution:

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

2-Layer NN Example

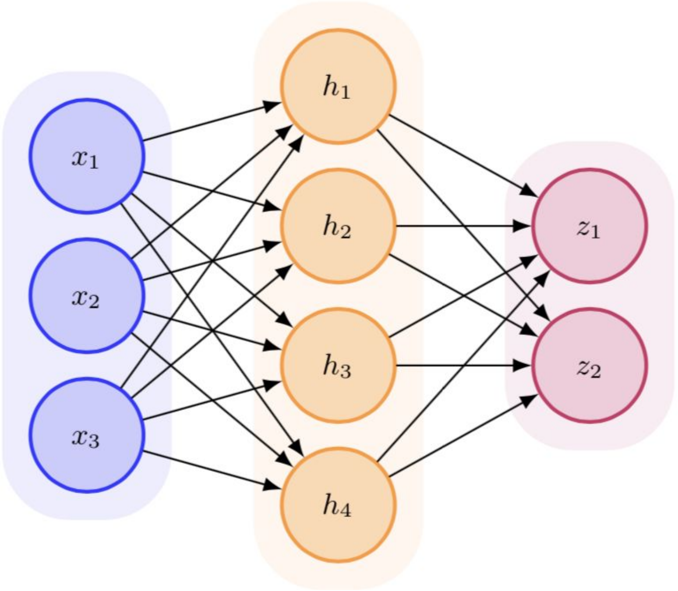
Neural network architecture

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



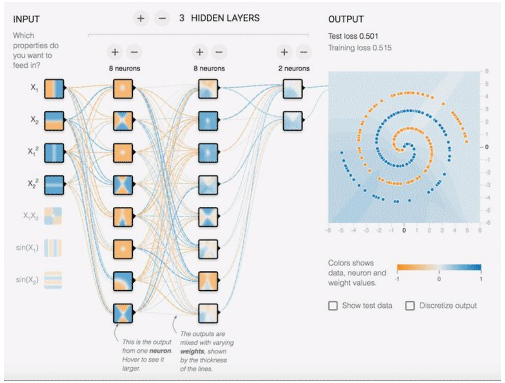
- Layer 1: $\mathbf{h}_1 = f(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
- Layer 2: $\mathbf{h}_2 = f(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$
- $\vdots$
- Layer N: $\mathbf{z} = \mathbf{W}_N\mathbf{h}_{N-1} + \mathbf{b}_N$

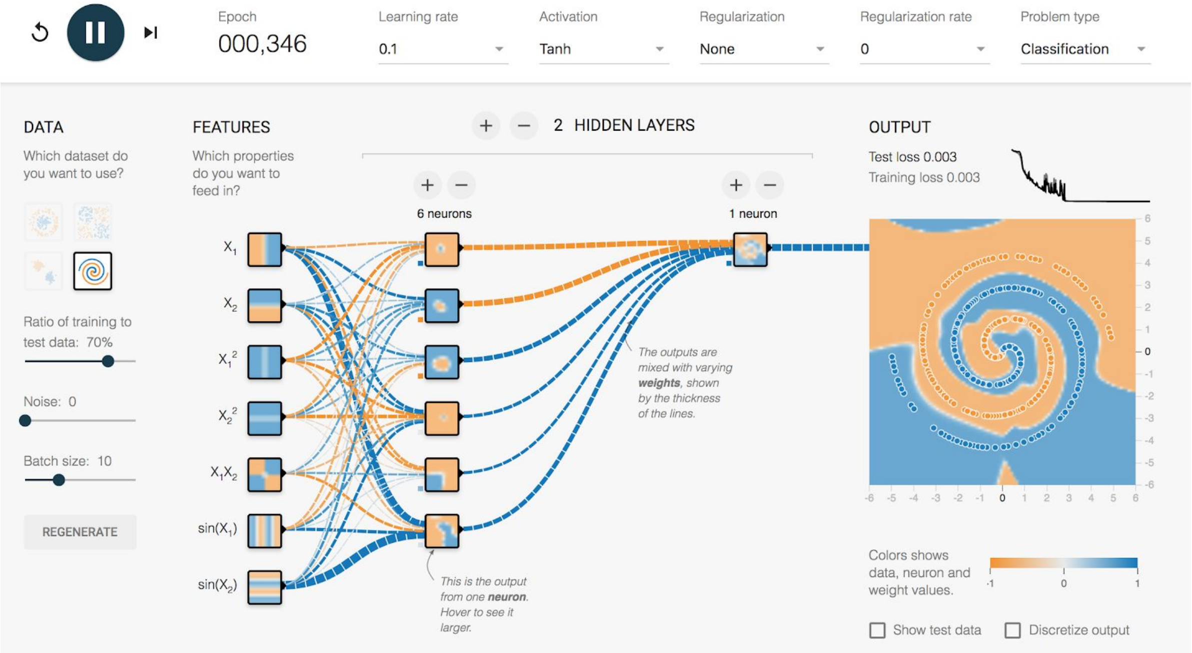
Questions:

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

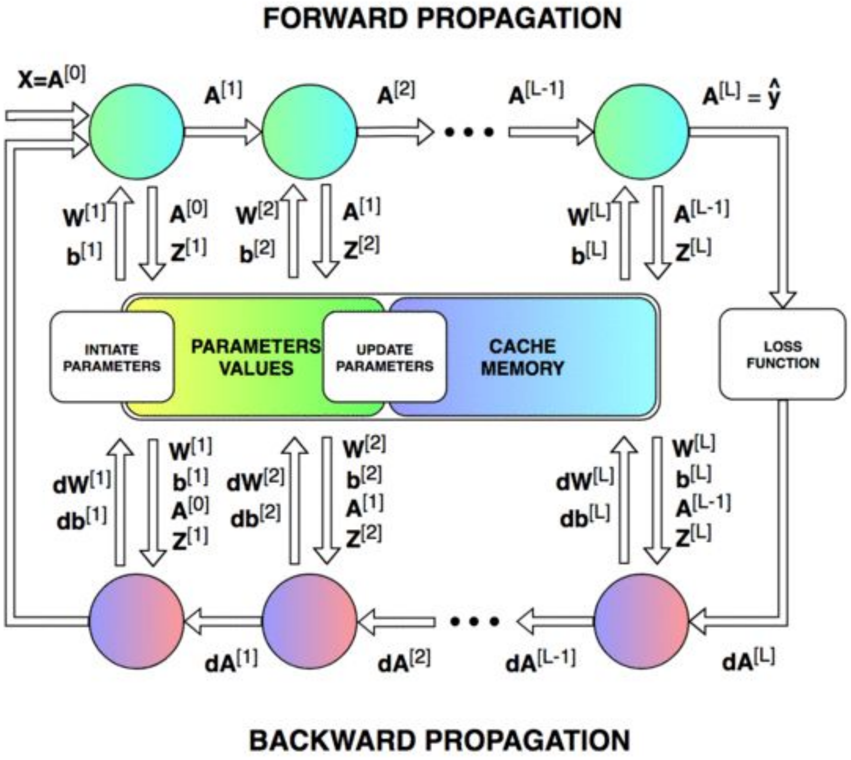
Neural Networks: Demo

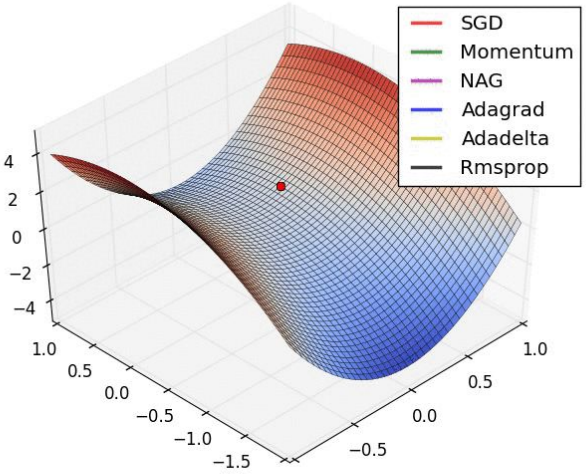
- Let’s play with it: https://playground.tensorflow.org/





Neural Networks: Backpropagation



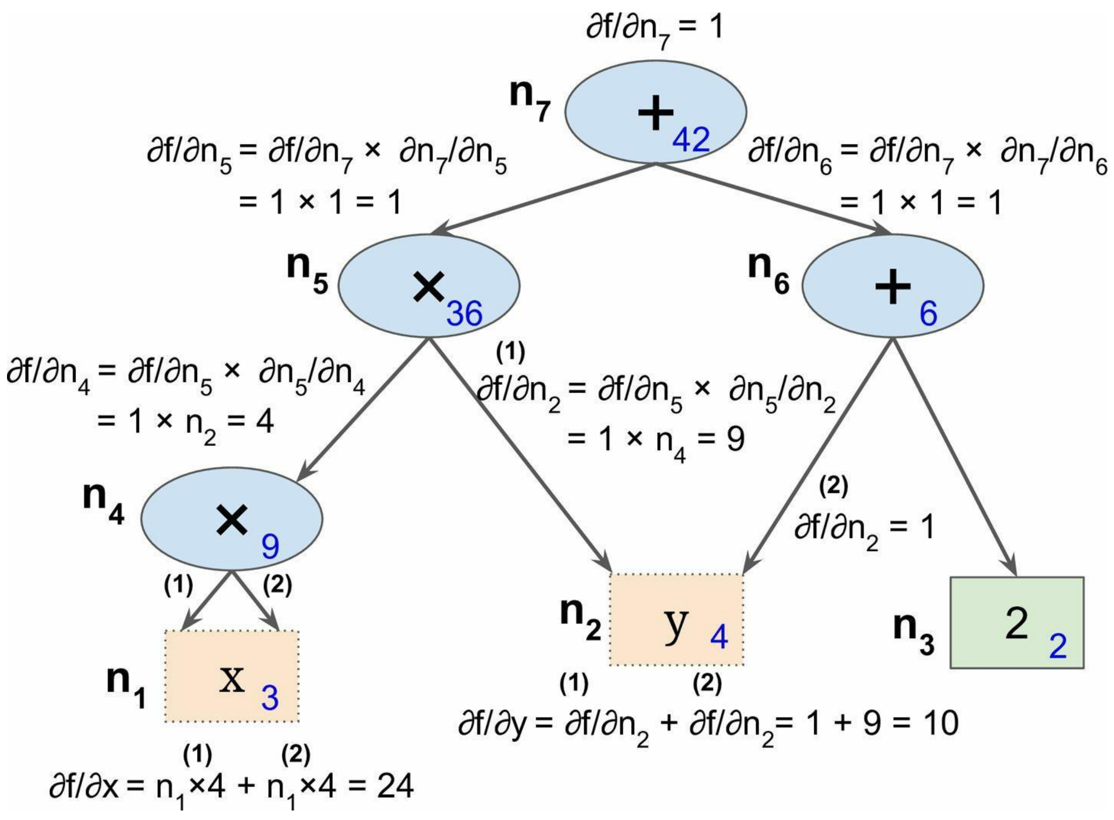


https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

Neural Networks: Backpropagation

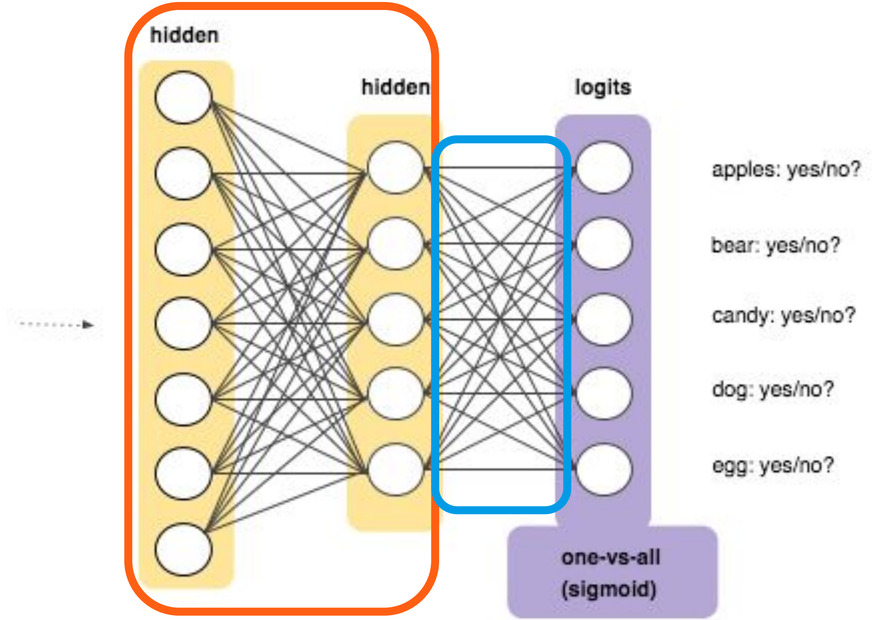
- A simple example to understand the intuition
- **f(x,y) = x^2y + y + 2**
- Forward pass:
  - x=3, y=4 → f(3,4)=42
- Backward pass:
  - Chain rule:

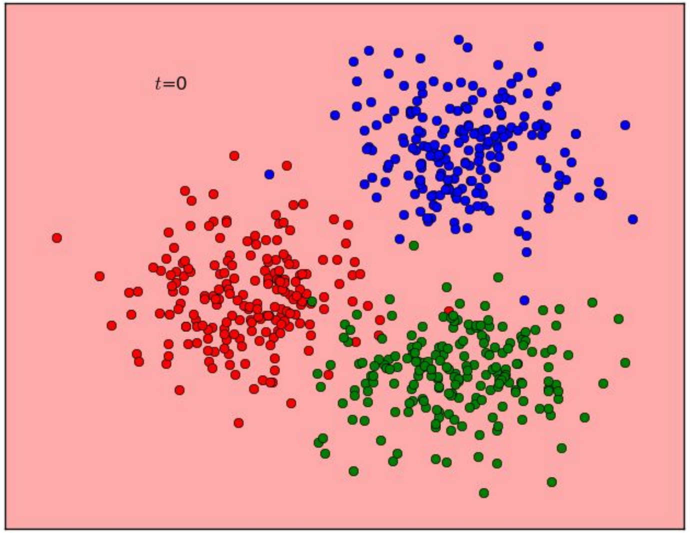
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n_i} \times \frac{\partial n_i}{\partial x}$$



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

Multiclass Classification





5 separate **binary classifiers**

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

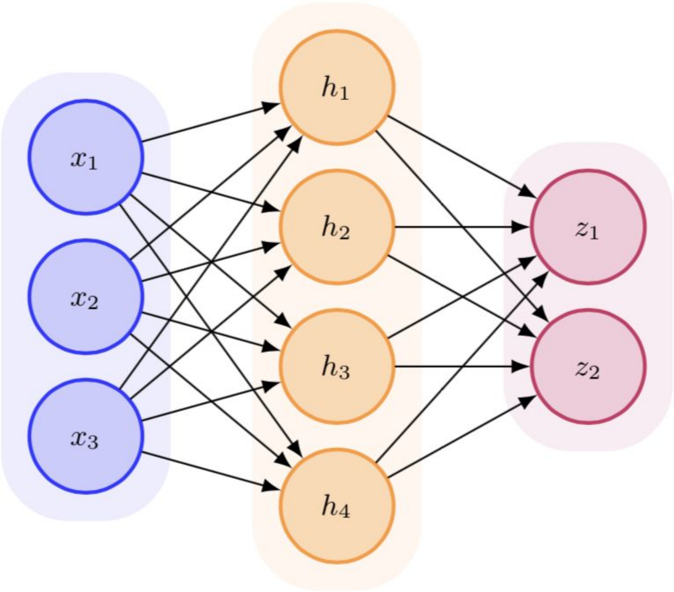
Question: Pros and cons?

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

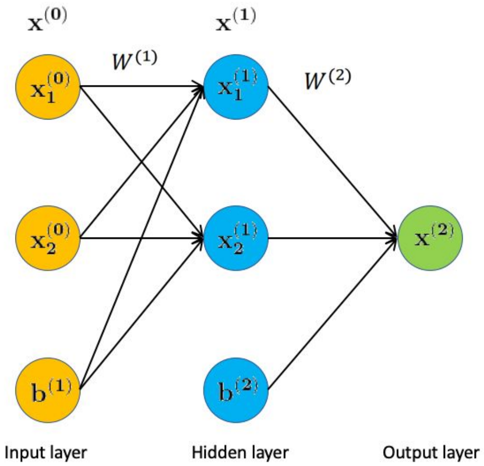
http://www.briandolhansky.com/blog/2013/9/23/artificial-neural-nets-linear-multiclass-part-3

2-Layer NN Example

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



Backprop: Exercise (next week)



In this question, let's consider a simple two-layer neural network and manually do the forward and backward pass. For simplicity, we assume our input data is two dimension. Then the model architecture looks like the following. Notice that in the example we saw in class, the bias term `b` was not explicit listed in the architecture diagram. Here we include the term `b` explicitly for each layer in the diagram. Recall the formula for computing $\mathbf{x}^{(l)}$ in the l-th layer from $\mathbf{x}^{(\mathbf{l-1})}$ in the $(l-1)$-th layer is $\mathbf{x}^{(l)} = \mathbf{f}^{(l)}(\mathbf{W}^{(l)}\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)})$. The activation function $\mathbf{f}^{(l)}$ we choose is the `sigmoid` function for all layers, i.e. $\mathbf{f}^{(l)}(z) = \frac{1}{1+\exp(-z)}$. The final loss function is $\frac{1}{2}$ of the `mean squared error` loss, i.e. $l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2}||\mathbf{y} - \hat{\mathbf{y}}||^2$.

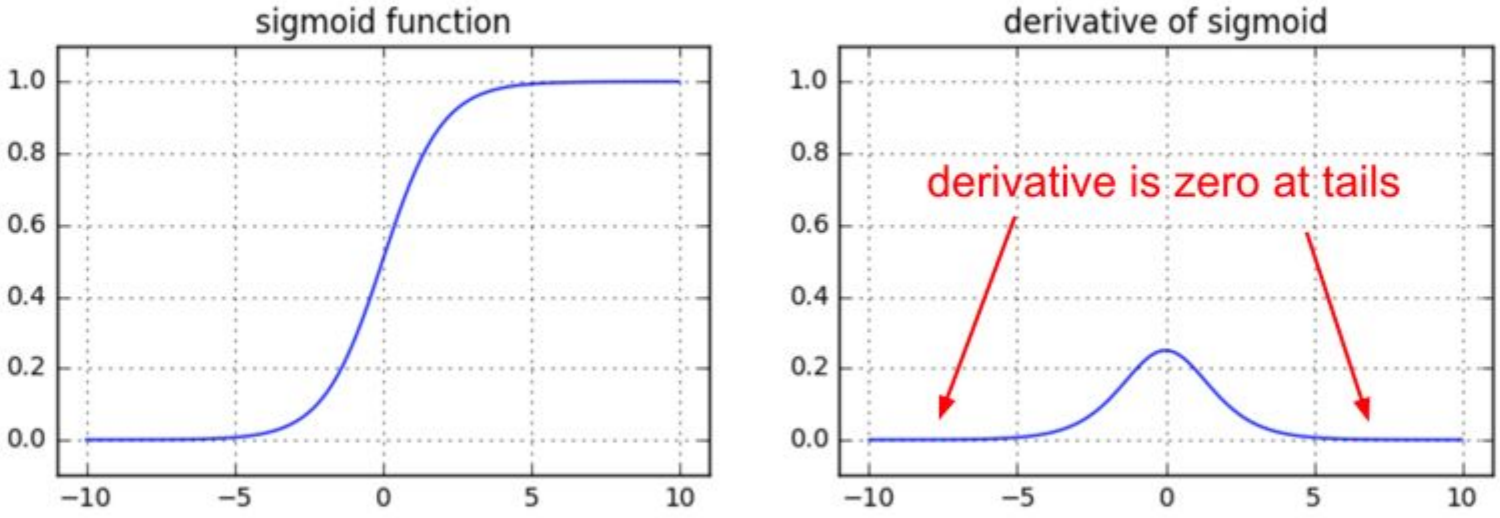
We initialize our weights as

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0.15 & 0.2 \\ 0.25 & 0.3 \end{bmatrix}, \quad \mathbf{W}^{(2)} = [0.4, 0.45], \quad \mathbf{b}^{(1)} = [0.35, 0.35], \quad \mathbf{b}^{(2)} = 0.6$$

1. When the input $\mathbf{x}^{(0)} = [0.05, 0.1]$, what will be the value of $\mathbf{x}^{(1)}$ in the hidden layer? (Show your work).
2. Based on the value $\mathbf{x}^{(1)}$ you computed, what will be the value of $\mathbf{x}^{(2)}$ in the output layer? (Show your work).
3. When the target value of this input is $y = 0.01$, based on the value $\mathbf{x}^{(2)}$ you computed, what will be the loss? (Show your work).

Why understanding backpropagation?

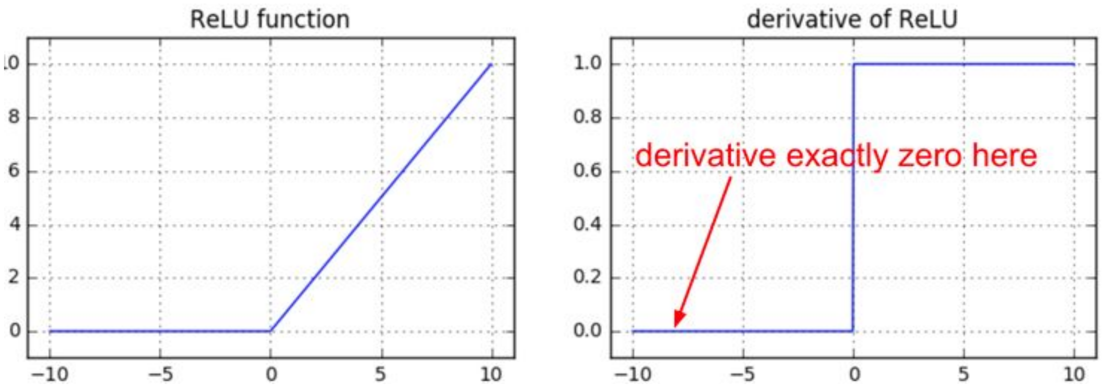
- "Why do we have to write the backward pass when frameworks in the real world, such as TensorFlow/PyTorch, compute them for you automatically?"
- Vanishing gradients on Sigmoids



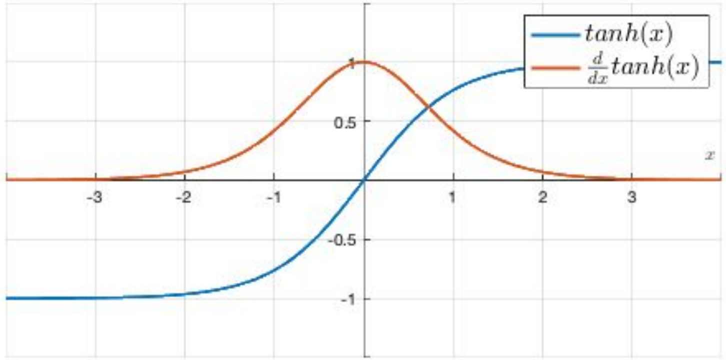
https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

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

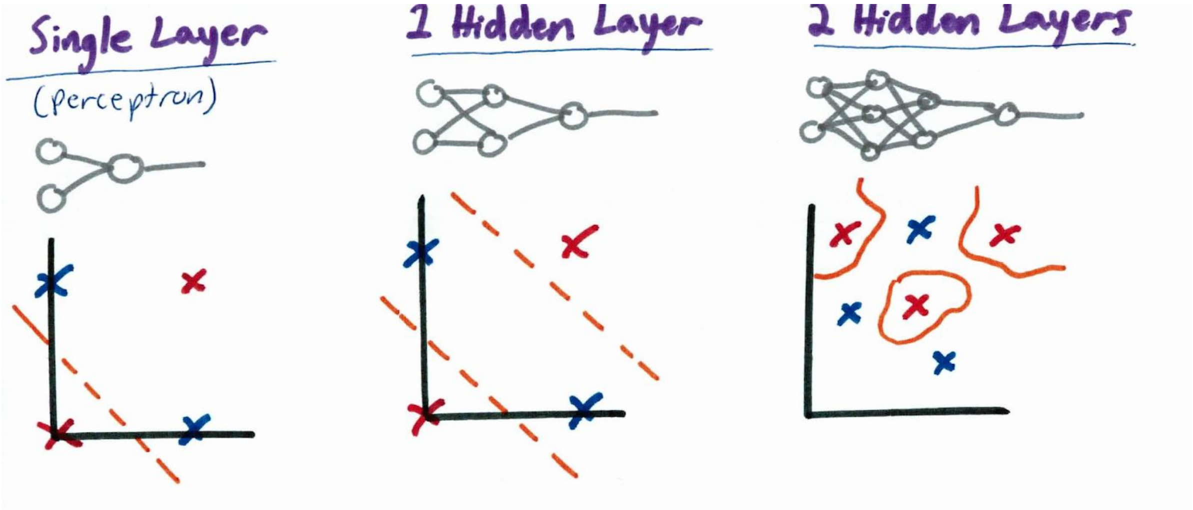
- Backpropagation (CS 231N at Stanford)
    - https://cs231n.github.io/optimization-2/
    - https://www.youtube.com/watch?v=i94OvYb6noo
- (Optional) Matrix-Level Operation:
    - https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c

NN: Number of iterations to converge

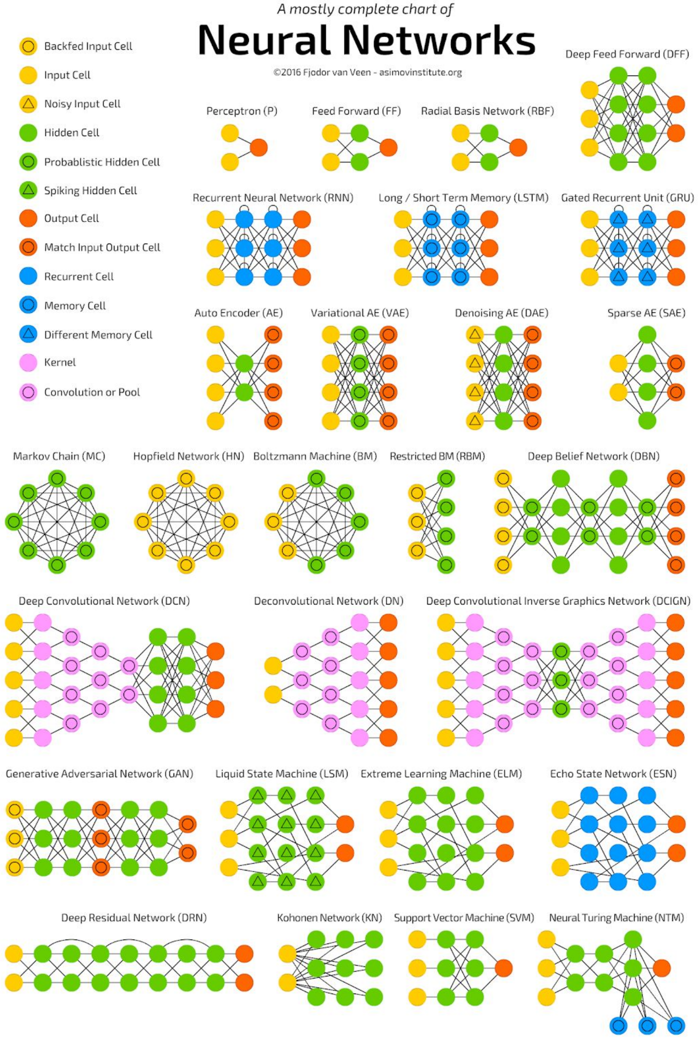
- Architecture/Meta-parameters of the network, e.g. # layers, activation
- Quality of training data (input-output correlation, normalization, noise cleansing, class distribution/imbalance)
- Random initialization of the parameters/weights
- Optimization algorithm, e.g. SGD, Adam, etc.
- Learning rate
- 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

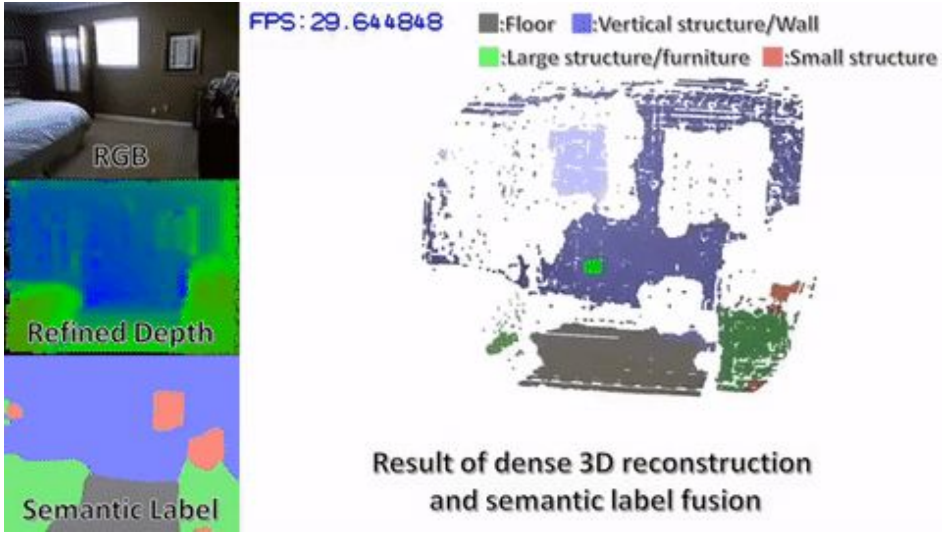


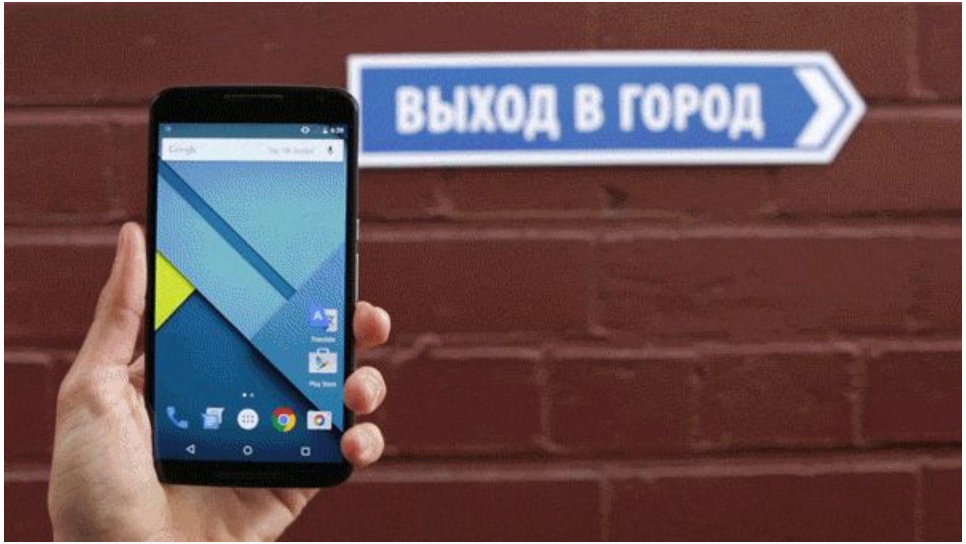
Flexibility



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

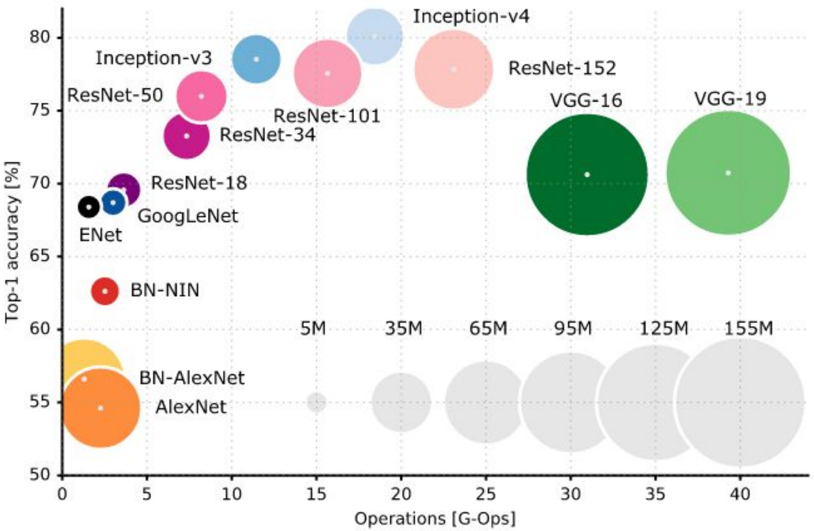




Efficiency (In many cases, prediction/inference/testing is fast)

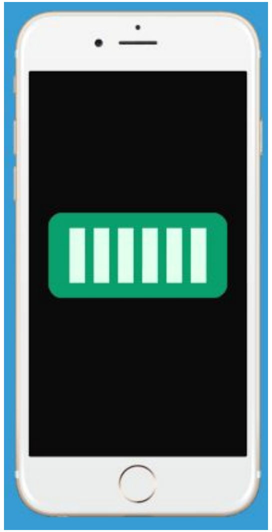
https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781788397872/1/ch01lvl1sec27/pros-and-cons-of-neural-networks
http://www.luigifreda.com/2017/04/08/cnn-slam-real-time-dense-monocular-slam-learned-depth-prediction/
http://www.missqt.com/google-translate-app-now-supports-instant-voice-and-visual-translations/

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-



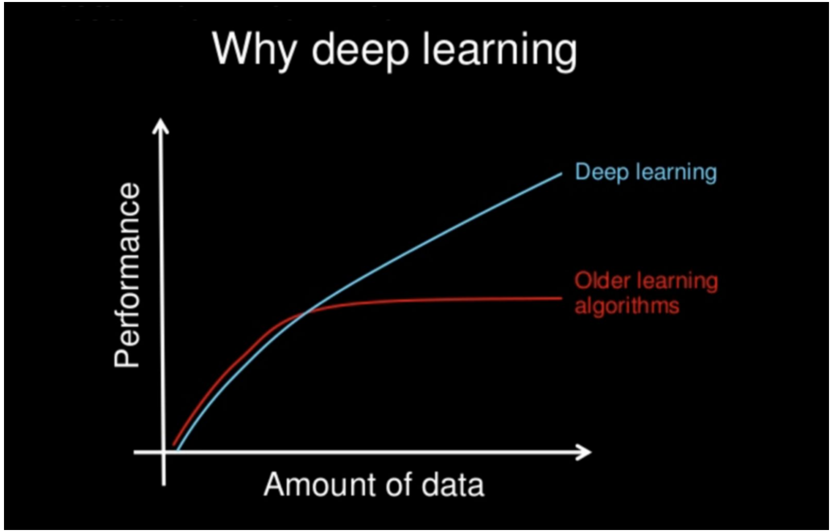
Efficiency (Big model → slow training, huge energy consumption (e.g. for cell phone))

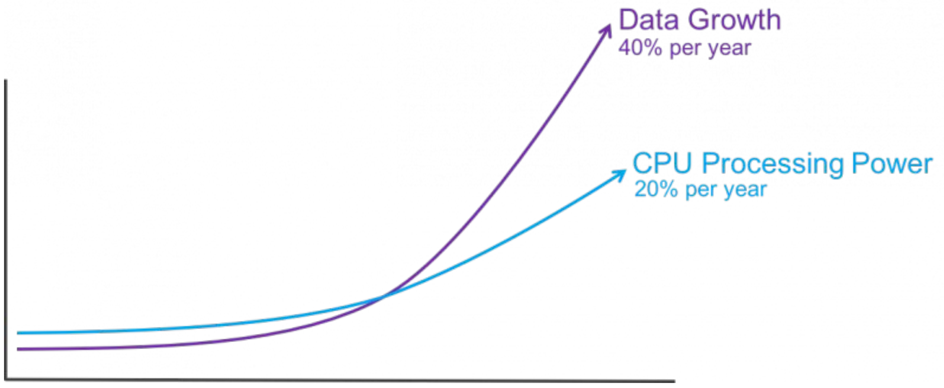
https://www.kdnuggets.com/2017/08/first-steps-learning-deep-learning-image-classification-keras.html/2

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

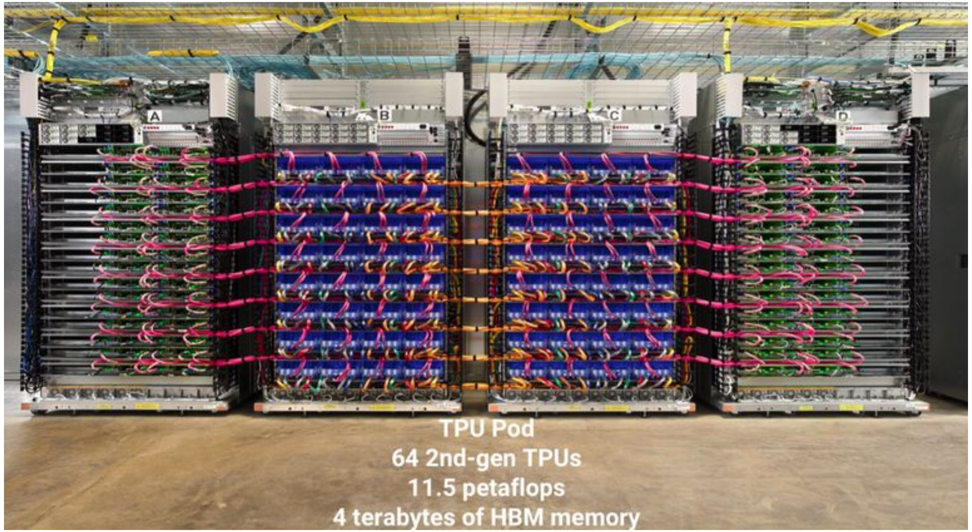




Data (Both a pro and a con)

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

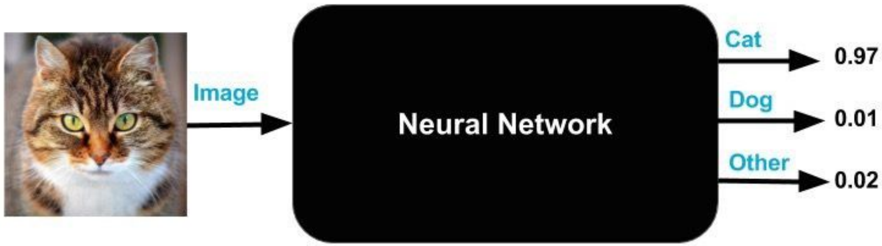
NN Summary: Pros and Cons

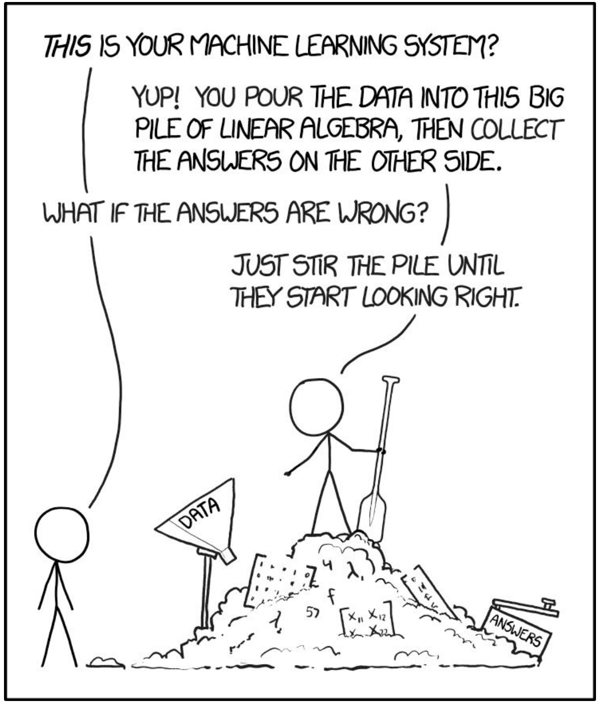


Computational Power (Both a pro and a con)

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/

What’s next?

- In next week’s discussion, we will continue to discuss:
    - Backpropagation in neural nets
- Programming Guide
    - PyTorch (for PS3)

UCLA Samueli Computer Science

Thank you!

Q & A