10707
Deep Learning: Spring 2020

Andrej Risteski

Machine Learning Department

Lecture 1:
Introduction to course, recap of basics

Course logistics

Time and location: MW 1:30-2:50 pm, GHC 4401 Rashid Auditorium

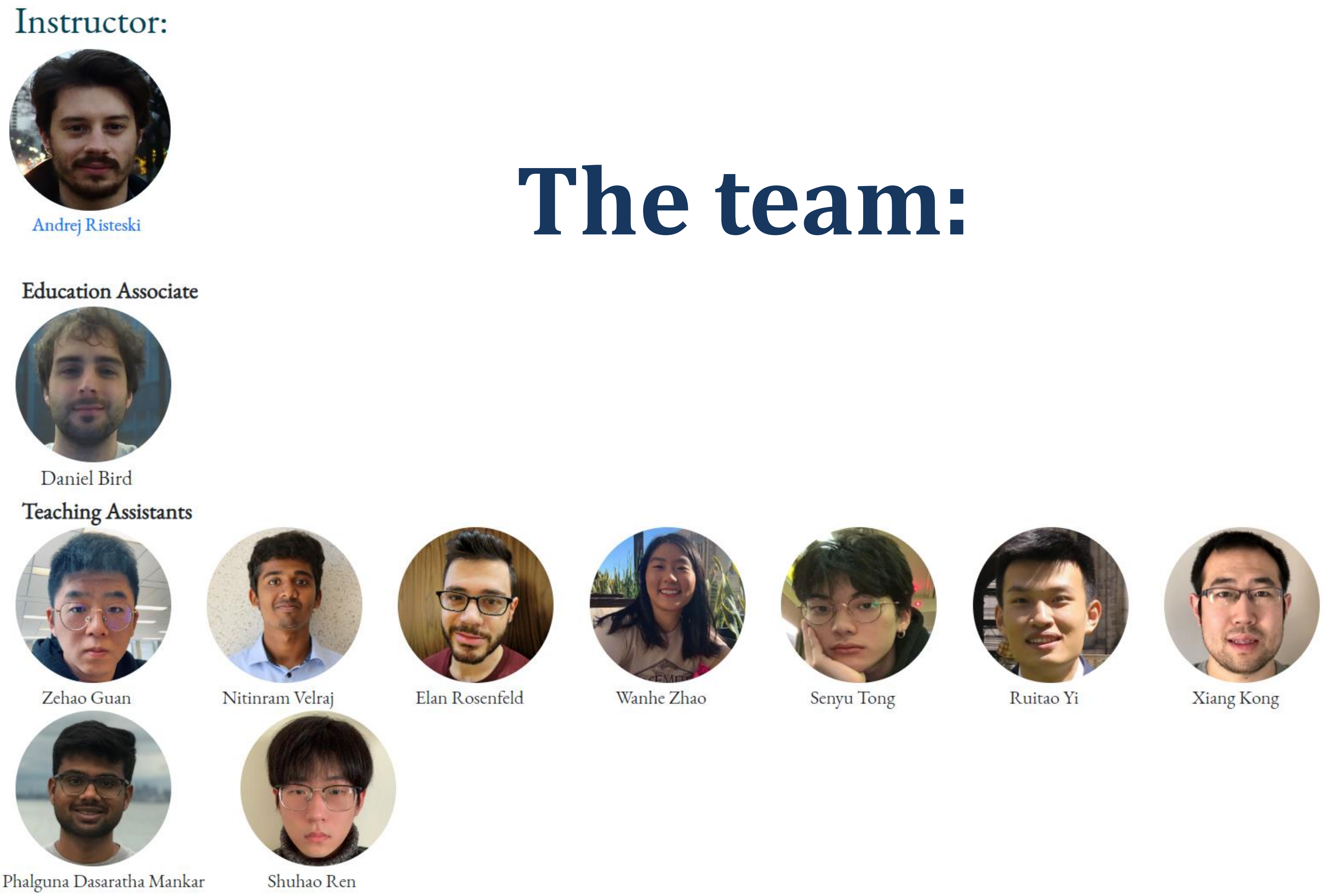
Course website: https://andrejristeski.github.io/10707-S20

Piazza: http://piazza.com/cmu/spring2020/10707/home

Contact Information: to get a quick response to questions from the teaching staff we strongly encourage you to post to **Piazza**. For private matters, make a private note visible only to course instructors or contact course Education Associate **Daniel Bird (dpbird@andrew.cmu.edu)**.

For longer discussions with TAs and to get help in person, we strongly encourage you to come to office hours. (Posted on course website.)

The team:



Instructor:

Andrej Risteski

Education Associate

Daniel Bird

Teaching Assistants

Zehao Guan

Nitinram Velraj

Elan Rosenfeld

Wanhe Zhao

Senyu Tong

Ruitao Yi

Xiang Kong

Phalguna Dasaratha Mankar

Shuhao Ren

Prerequisites

MOST IMPORTANT !!: Mathematical maturity

Mathematical prerequisites: strong background in linear algebra, machine learning, statistics and probability theory.

Course prerequisites: 10315 or 10401 or 10715 or 10701 or 10601

Coding prerequisites: a basic understanding of coding (Python preferred) there will be a coding component.

Course materials

Readings: (self-contained) slides will be posted periodically on the course website. The instructor will try to upload slides before class.

We will not follow a textbook, and additional readings will be posted whenever relevant.

Homeworks: Homework assignments will be announced on Piazza when released.

Tentative release/due dates are on the website calendar.

Assignments and grading

Assignments: there will be two types of assignments - **problem sets** (written, mathematical in nature) and **coding assignments**.

The tentative schedule of release and due dates can be seen in the Syllabus. Latex templates will be released with the homework for students to complete with their solutions.

Exams: the course will have **two midterm exams** which are scheduled to be hosted on **Febuary 12th** and **April 1st**.
There will be a **final exam** scheduled during finals week, the official date of which will be announced as soon as it is confirmed.

Grading: assignments contribute **45%** in total, 2 midterms (**15% each**), final exam contributes (**25%**).

Assignment policies

Collaboration: you may discuss the general idea of the questions with anyone you like, but your discussion may not include the specific answers to any of the problems and when writing your solutions you must close all notes and write the answer entirely yourself.

Submitting: assignments will be submitted through **Gradescope**. Additionally, you should upload your code to **Autolab**. Writeups should be typeset in Latex and should be submitted in PDF form.

Code should be submitted w/ a README file w/ instructions on how to execute code. You will receive an invite to Gradescope for 10707 Deep Learning Spring 2019 by 01/08/2019. If you have not received an invite, email Daniel Bird (dpbird@andrew.cmu.edu) w/ details of your Andrew email address and your full name.

Assignment policies

Regrades: submit a regrade request on Gradescope.

Late policy: each student will have a total of **5 grace days** that a student may choose to apply to the homework assignments. No more than **3 grace days** can be used on any single assignment.

Late homeworks when the student has no Grace days remaining or 3 days past the deadline will be given a score of 0.

Extensions: in general, we do not grant extensions on assignments. There are several exceptions: medical emergencies, family/personal emergencies, university-approved absences. (See website).

Academic integrity policies: please check website!

Goals of this course

Convey levels of understanding of phenomena in deep learning

Fully **mathematical** understanding: satisfying mathematical theory. (*Rare*.)

“**Physics**” understanding: mathematical theory under strong structural assumptions, e.g. random inputs, independence. (*Semi-common*.)

“**Practical**” understanding: imprecise intuitions/conjectures from trial-and-error. (*Common*.)

Survey classical and recent topics in deep learning

Classical topics provide historical context and intuition for modern ones.

Mix of hands-on experience and rigorous thinking

Problem sets will force you to think mathematically about topics we cover.

Coding assignments will involve implementing algorithms taught in class.

Differences with prior offering of 10-707

Topic selection: while there is overlap, the topic selection is somewhat different from prior course. Check Schedule to for details.

More mathematical content: the course will be mathematically more demanding, and we will cover substantively more theory.

Assignments: problem sets, as well as coding assignments. Final exam instead of a project.

Course organization

Supervised learning: learning in the presence of labels

Unsupervised learning: learning in the absence of labels

Sequential data: models for text / language / translation

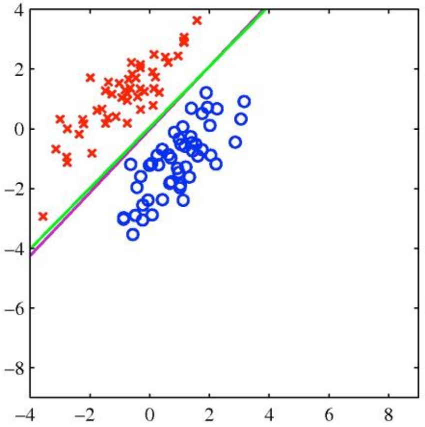
Other topics: robustness, causality, etc.

Part I: Supervised learning

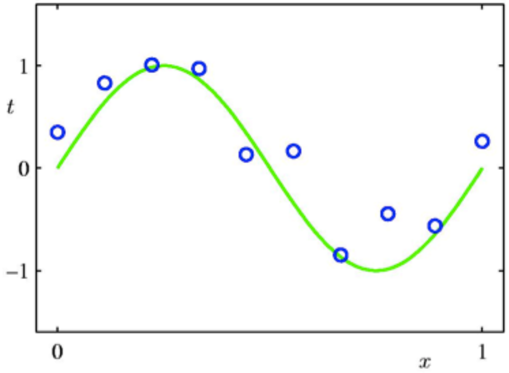
Receive train samples (x, y), x is data (e.g. image), y is label (e.g. "dog").

Goal: learn to label new (unseen) x's.

Classification: labels y are discrete class labels. Goal is to correctly classify new inputs.

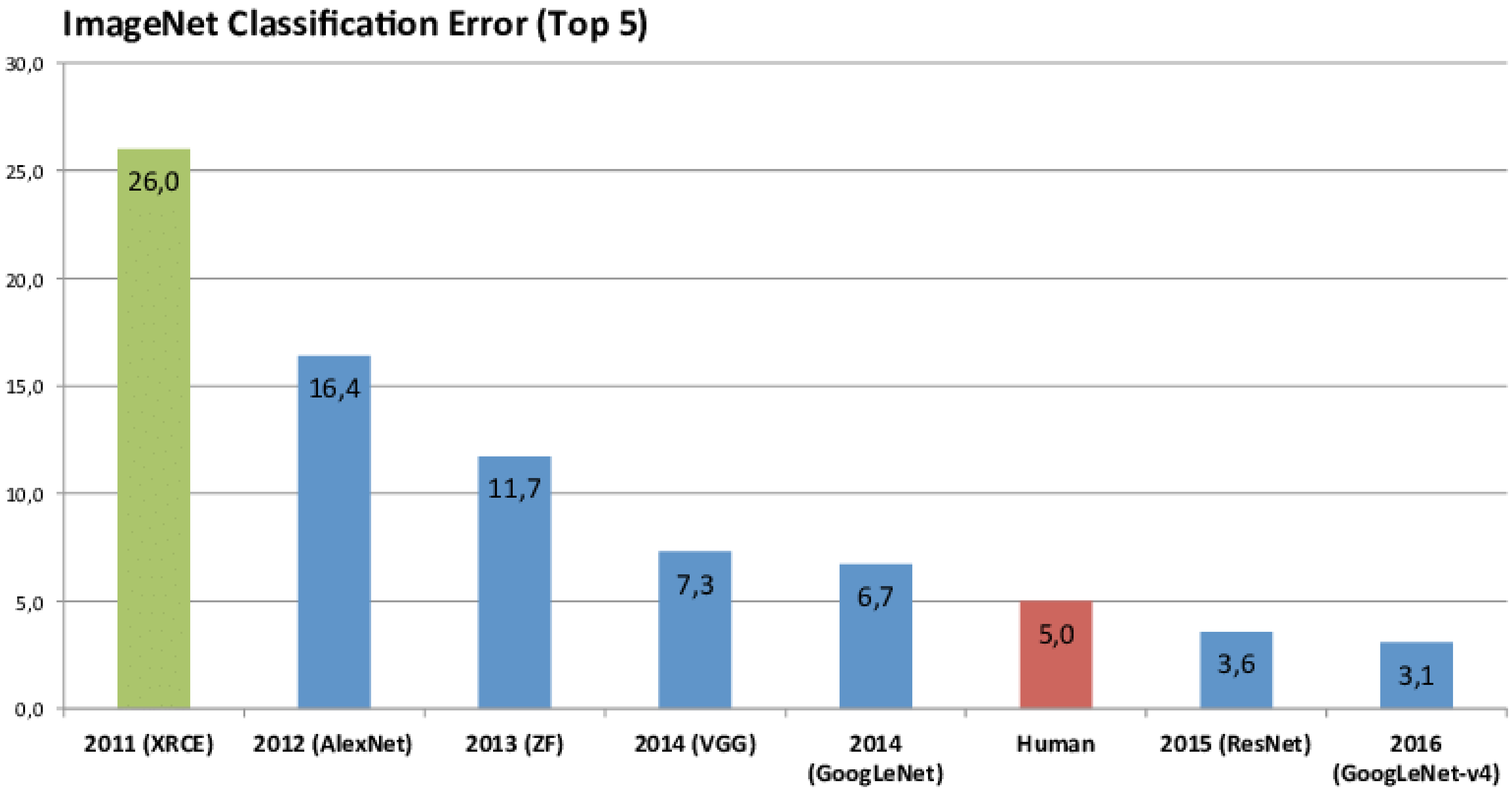


Regression: outputs y are continuous. Goal is to predict the output given new inputs.



Part I: Supervised learning

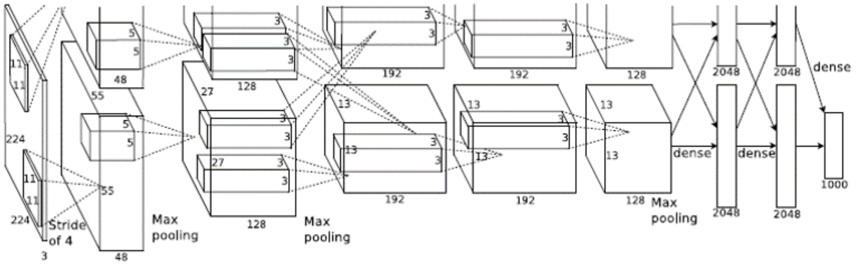
The start of the deep learning renaissance:



Part I: Supervised learning

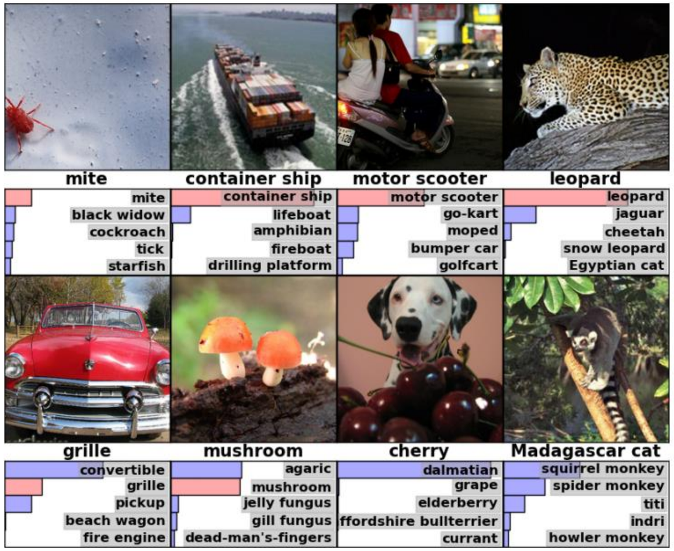
Deep Convolutional Nets for Vision (Supervised)

Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS, 2012.



IMAGENET

1.2 million training images
1000 classes



Deep Nets for Speech (Supervised)

Hinton et. al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, IEEE Signal Processing Magazine. 2012.

Part I: Supervised learning

Empirical risk minimization approach:
minimize a **training** loss l over a class of **predictors** $\mathcal{F}$:

$$\hat{f} = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Three pillars:

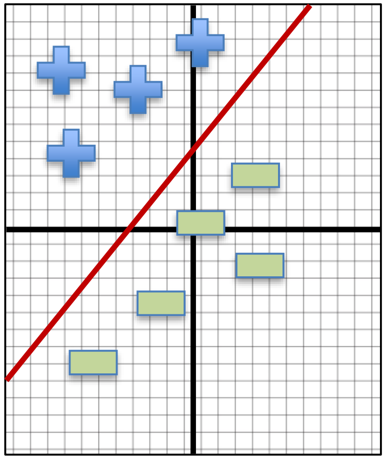
(1) How expressive is the class $\mathcal{F}$? (**Representational power**)

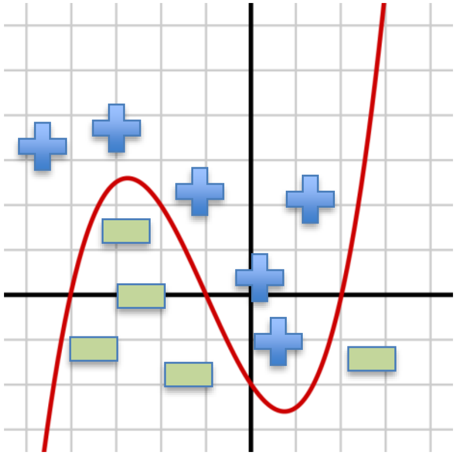
(2) How do we minimize the training loss efficiently? (**Optimization**)

(3) How does $\hat{f}$ perform on unseen samples? (**Generalization**)

Expressivity

What do we mean by expressivity?





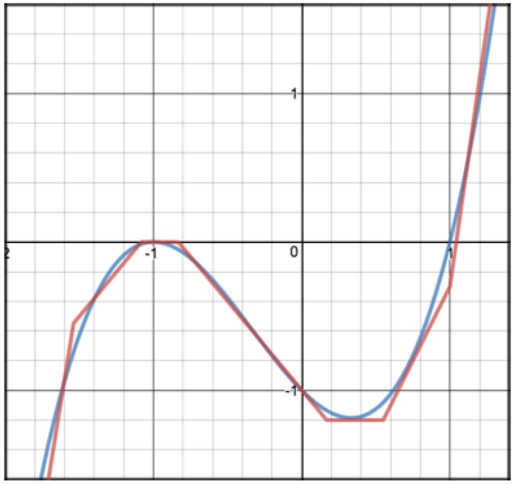
Expressive = functions in class can represent “complicated” functions

“Universal” expressivity of neural networks

(1): Neural networks are **universal approximators**: given any (reasonably nice) function f: $\mathbb{R}^d \to \mathbb{R}$, a **shallow** (3-layer) neural network with $\sim \left(\frac{1}{\epsilon}\right)^{\mathrm{d}}$ neurons can approximate it to within ϵ error.

“curse of dimensionality”

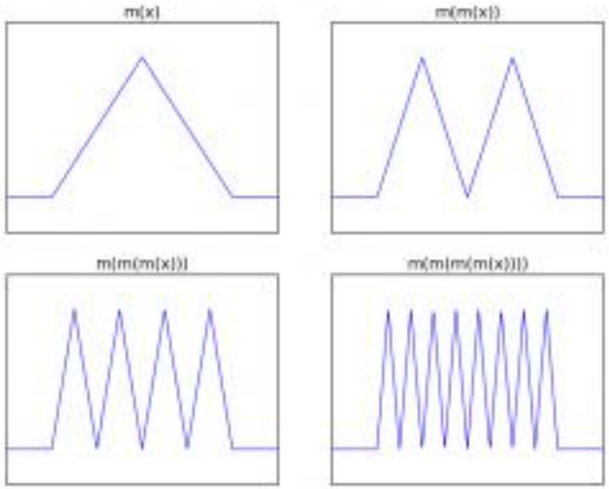
(2): Neural networks can **circumvent** the curse of dimensionality for functions w/ decaying Fourier coefficients: **shallow** neural networks with $\sim \left(\frac{1}{\epsilon}\right)$ neurons can approximate them to within ϵ error.



Does depth help?

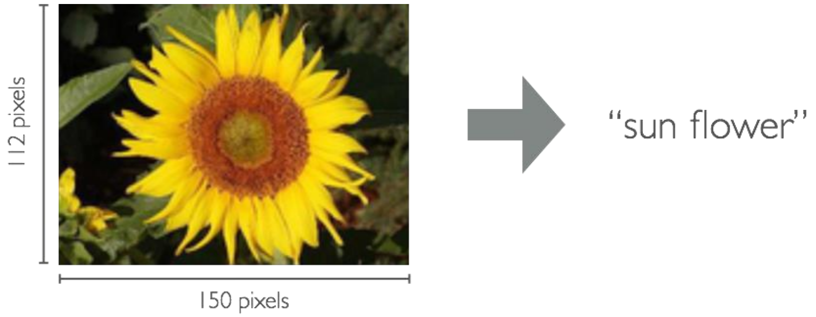
Universal approximation constructions yield good approximators that are **shallow**. Is there a benefit to using deeper neural networks?

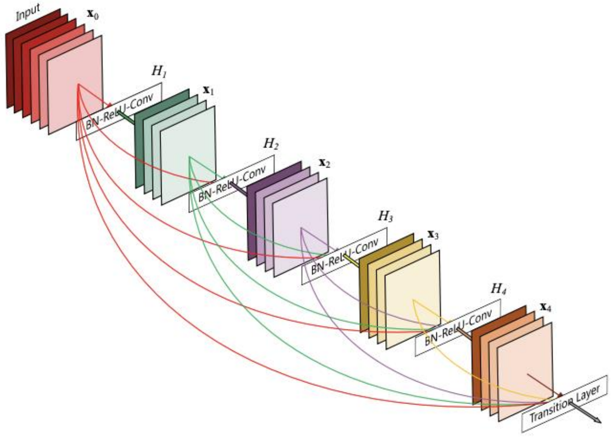
Depth separation theorems: for every $k \in \mathbb{N}$, there are functions representable by k^3 -layer neural networks of size $O(k^3)$, s.t. every $O(k)$ -depth network approximating them has size $\Omega(2^k)$.



Modern architectures

Applications in vision: architecture should make use of structure in visual data (pixel locality, texture, hierarchy, etc.).
Solution: convolutional networks





Modern architectures: large, deep networks have various training problems (vanishing gradients, poor generalization).
Solution(s): Residual networks (ResNets), dense convolutional networks (DenseNets).

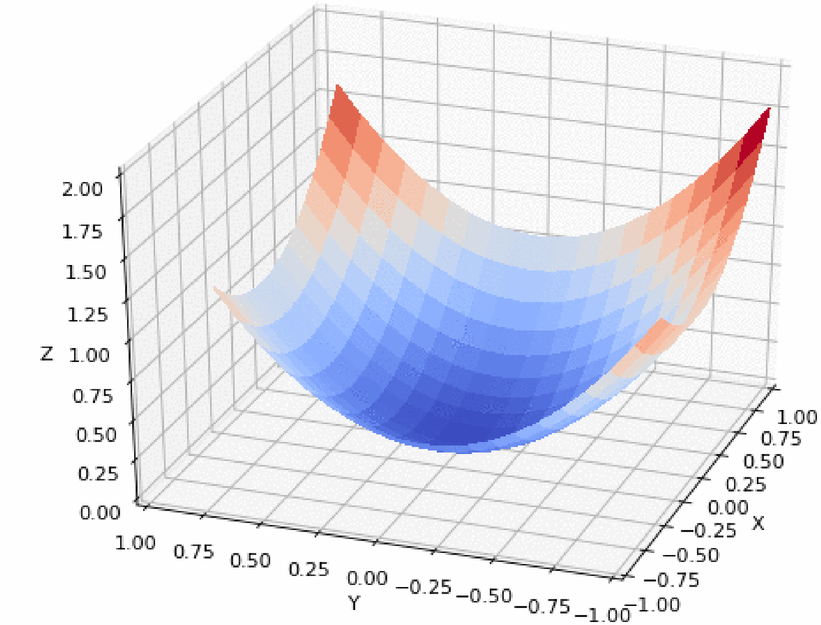
Optimization

How difficult is it to minimize the training loss?

The algorithm: gradient descent (& friends: stochastic, momentum, etc.)

Follow the negative gradient (greedy direction of most improvement)

Remember/reuse previous gradients



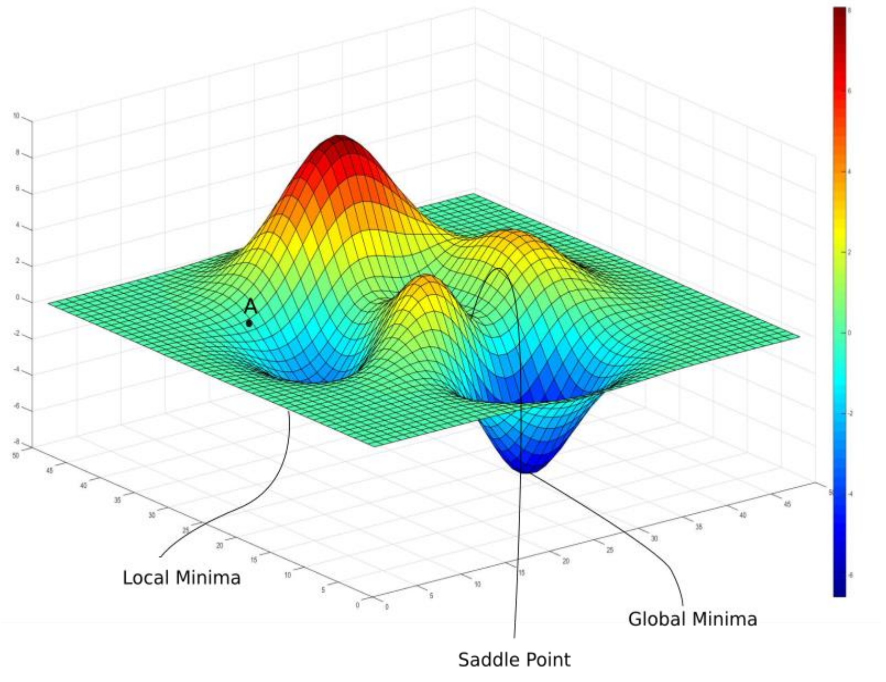
Optimization

How difficult is it to minimize the training loss?

The villains: saddle points and poor local minima

Points with gradient = 0, but not local minima

Local minima with value much worse than global minimum



Generalization

$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Basic question: if (x, y) are sampled i.i.d from distribution $\mathcal{D}$,

is $\mathbb{E}_{(x,y)\sim\mathcal{D}} l\left(\hat{f}(x), y\right)$ comparable to training loss?

Traditional view: "**gap**" between training loss and expected loss on $\mathcal{D}$ depends on "**complexity**" of class $\mathcal{F}$ (the more complex, the more training data is necessary). Basic VC dimension bounds on neural networks.

Generalization

$$\hat{f} = \underset{f \in \mathcal{F}}{\text{argmin}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Basic question: if (x, y) are sampled i.i.d from distribution $\mathcal{D}$, is $\mathbb{E}_{(x,y)\sim\mathcal{D}} l\left(\hat{f}(x), y\right)$ comparable to training loss?

Modern view:

Modern networks don't seem to fit this paradigm (size is large enough to fit random labels).

There are instances where larger models don't overfit, but have *better* generalization (**double-descent** phenomenon).

Conjectures and speculations: **shallow vs flat** minima.

Part II: Unsupervised learning

Learning from data **without** labels.

What can we hope to do:

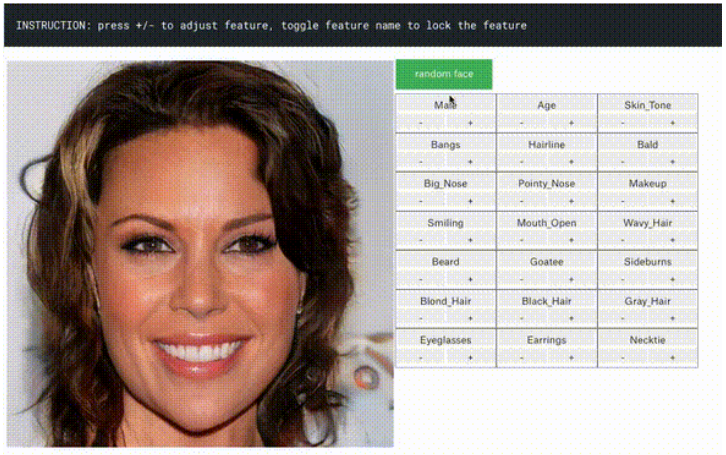
Fit a parametrized **structure** (e.g. clustering, low-dimensional subspace) to data to reveal something meaningful about data. (**Structure learning**)

Learn a (parametrized) **distribution** *close* to data generating distribution. (**Distribution learning**)

Learn a (parametrized) distribution that implicitly reveals an **"embedding"/"representation"** of data for downstream tasks. (**Representation learning**)

Part II: Unsupervised learning

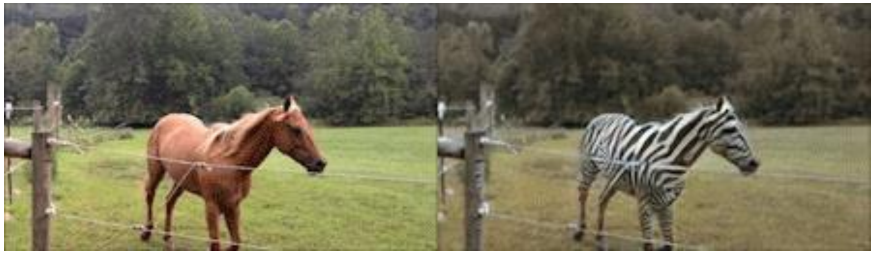
Photorealistic image/video generation, learning to generate new samples, complex conditioning



```
INSTRUCTION: press +/- to adjust feature, toggle feature name to lock the feature
```

(Video from Guan et. al '18)

Extracting complex features for downstream applications, domain adaptation

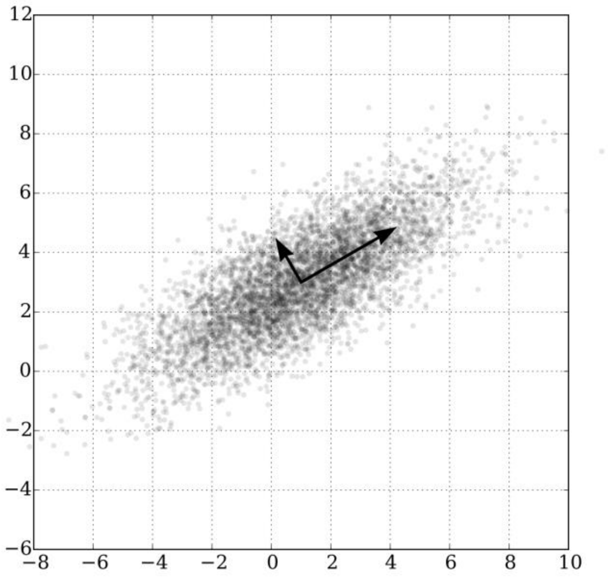


(Video from Zhu et. al '17)

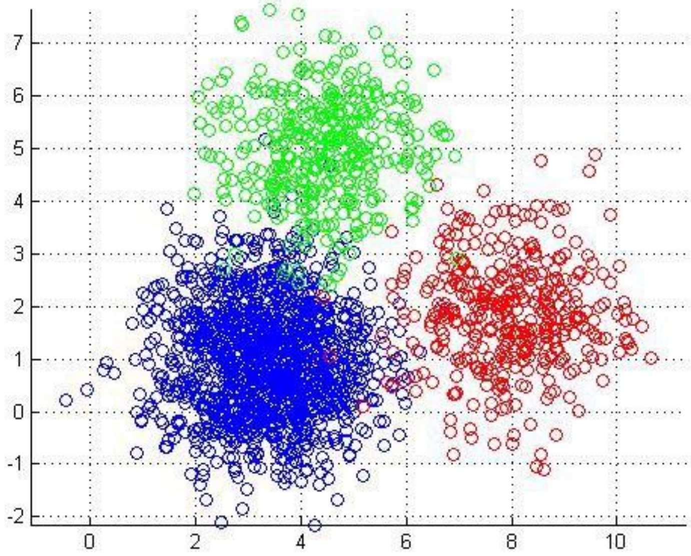
Structure learning

Fit a parametrized **structure** (e.g. clustering, low-dimensional subspace) to data to reveal something meaningful about data.

PCA(principal component analysis), direction of highest variance



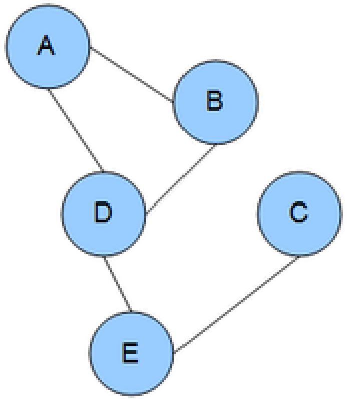
Clustering



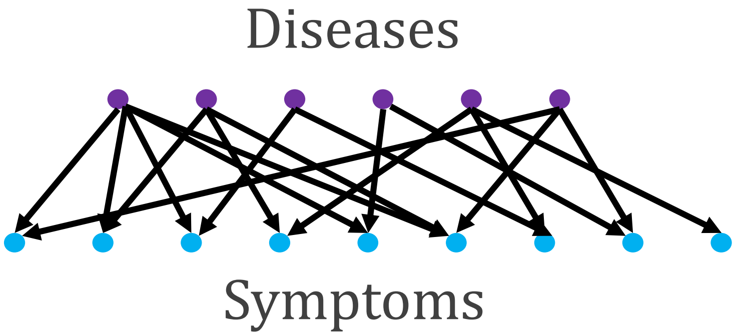
Distribution learning

Some typical choices of parametrized distributions:

Classical choices: fully-observed graphical models (undirected and directed), latent-variable graphical models (mixture models, sparse coding, topic models).



Markov Random Fields: sparse independence structure: "A is independent of other vars, given B, D"

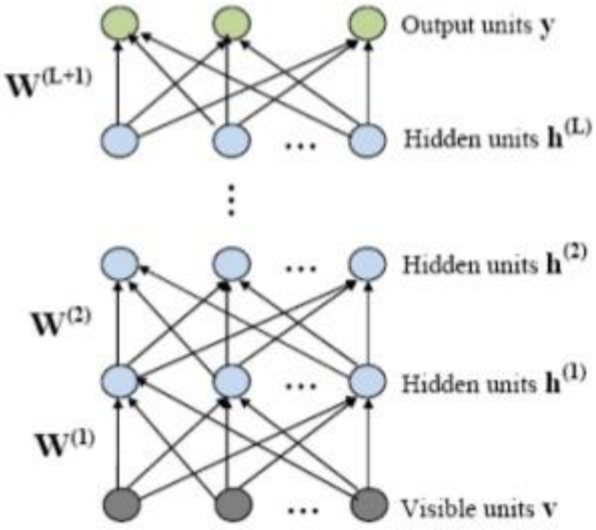


Latent variable models: data is "simple" conditioned on some unobserved (latent) variables

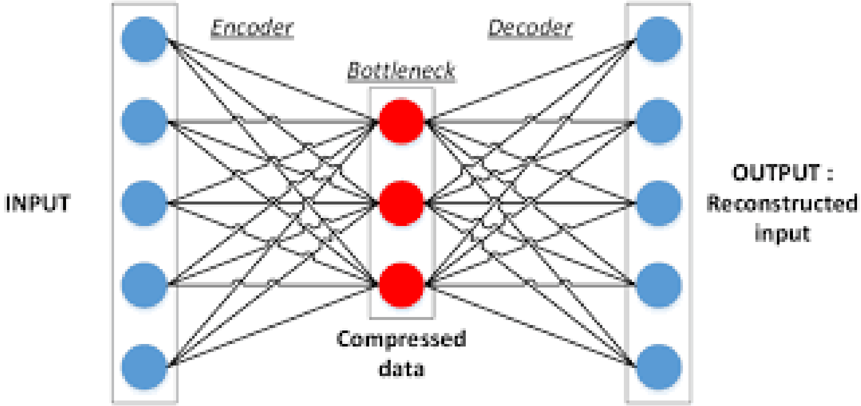
Distribution learning

Some typical choices of parametrized distributions:

Semi-modern choices: deep Boltzmann machines, deep belief networks, (variational) auto-encoders, energy models.



Deep Boltzmann machines, belief networks: graphical model analogues of deep neural networks.

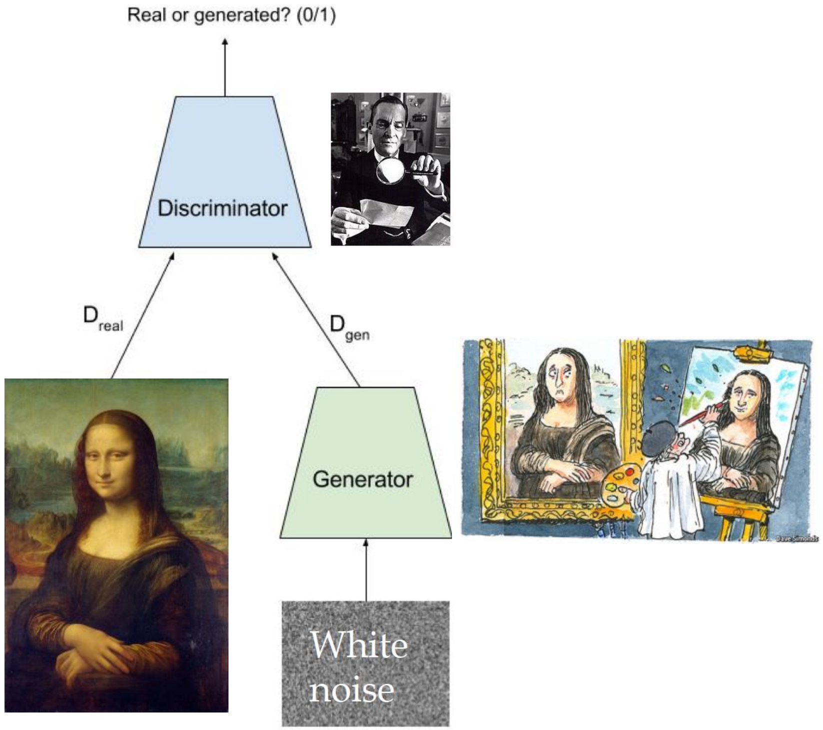


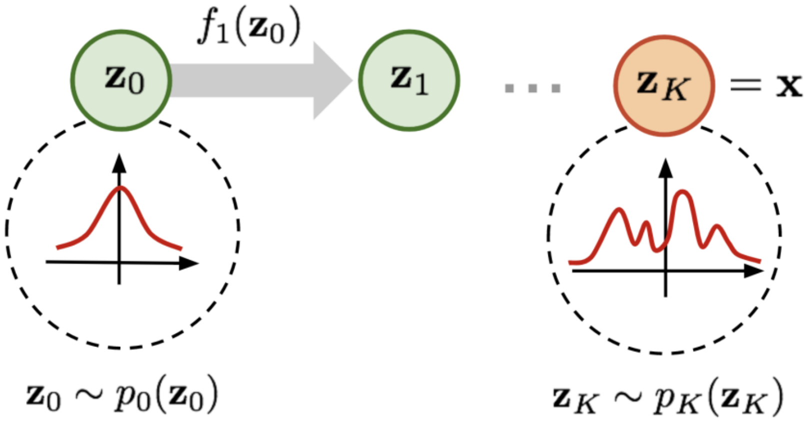
(Variational) autoencoders: model data by enforcing a latent space “bottleneck:

Distribution learning

Some typical choices of parametrized distributions:

Modern choices: generative adversarial networks, autoregressive models, flow models, etc.




$f_1(\mathbf{z}_0)$
$\mathbf{z}_0$
$\mathbf{z}_1$
$\mathbf{z}_K = \mathbf{x}$
$\mathbf{z}_0 \sim p_0(\mathbf{z}_0)$
$\mathbf{z}_K \sim p_K(\mathbf{z}_K)$

Training techniques: Likelihood-based vs likelihood-free

Two typical starting points for training:

Likelihood-based: maximize the likelihood of the data under the model (possibly with some approximations)

Typical approximations used: variational inference (optimize tractable deterministic approximation of likelihood), MCMC methods (idea: approximate difficult quantities like partition functions with sampling)

Likelihood-free: use a surrogate loss – e.g. in GANs, train a discriminator to tell real and generated samples apart; noise-contrastive training: encourage model to put probability mass away from “fake” samples.

Representation learning and self-supervised learning

Given unlabeled data, design supervised tasks that induce a good representation for downstream tasks.

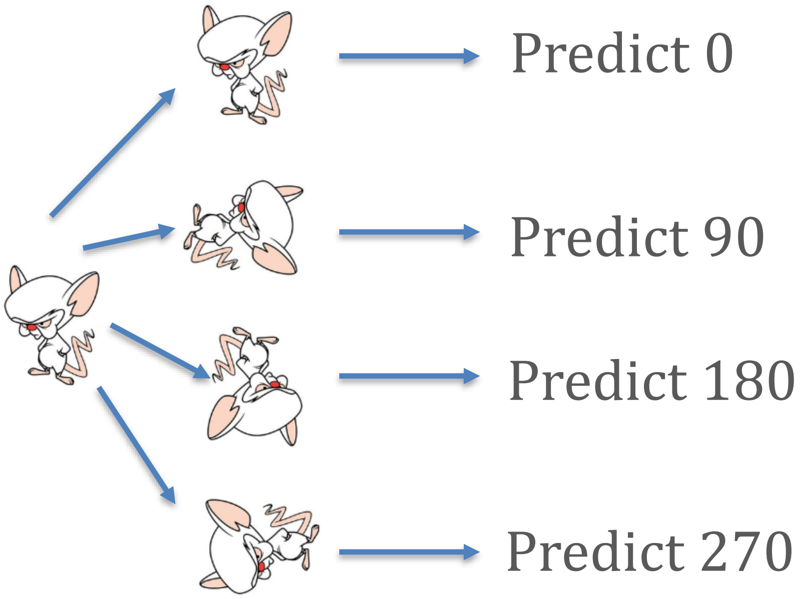
No good mathematical formalization, but the intuition is to “force” the predictor used in the task to learn something “semantically meaningful” about the data.

Examples in NLP: predict next word, given previous 5 words; predict middle word, given surrounding 5 words; etc.

Representation learning and self-supervised learning

Examples in vision: a lot, and quite different in nature.

Rotation prediction


Predict 0

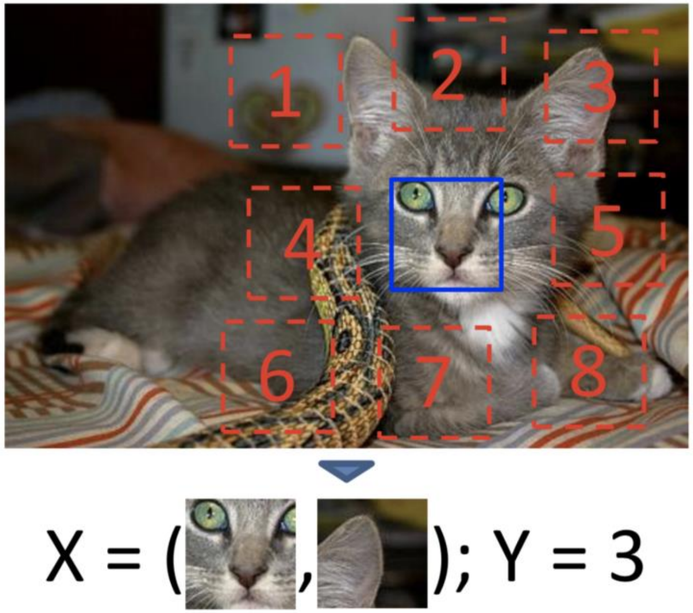
Predict 90

Predict 180

Predict 270

Predict one of four angles an image is rotated by

Jigsaw puzzles



Predict position of second piece wrt to first

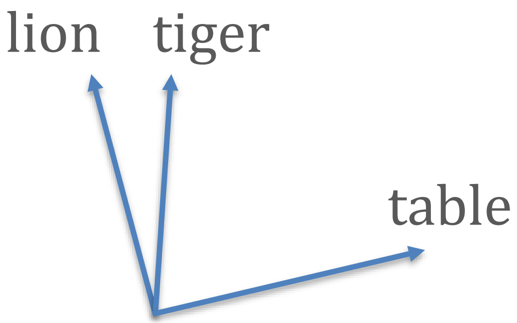
Sequence learning and language models

Modeling language: classical and modern methods.

n-gram models: Markov models, where current word depends on previous n-1 words.

$$P(x_i|x_{i-1,}\, x_{i-2,}\, \dots, x_1) = P(x_i|x_{i-1,}\, x_{i-2,}\, \dots, x_{i-n+1})$$

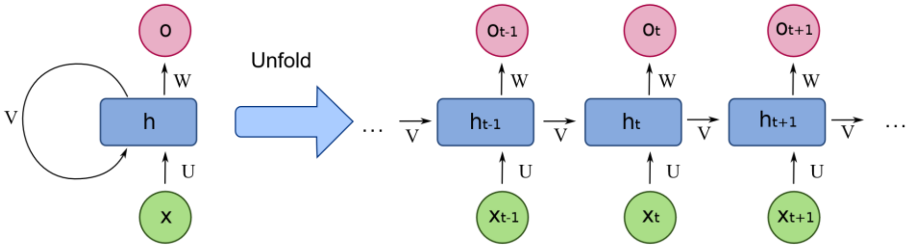
Co-occurrence based models: word2vec, GloVe. Find vector embeddings for words, s.t. inner products approximate co-occurrence counts.



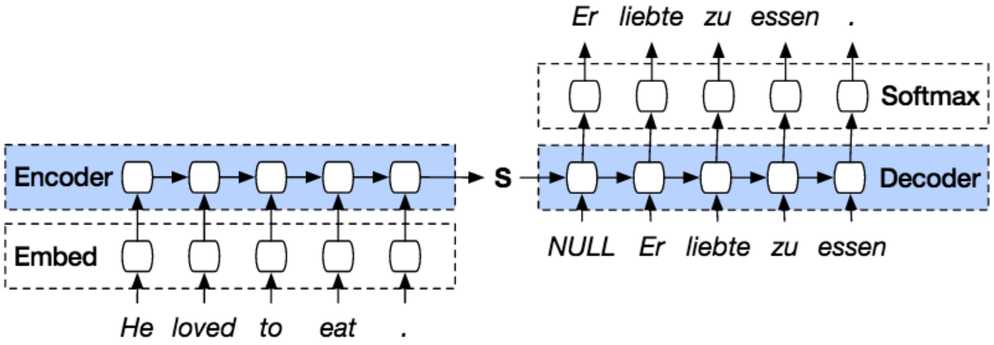
Sequence learning and language models

Modeling language: classical and modern methods.

Sequential neural net architectures: Recurrent Neural Nets (RNNs), Long Short-Term Memory (LSTM), BERT, etc.



Models for translation: seq2seq, transformers, universal machine translation.



Neural network basics: the artificial neuron

Neuron **pre-activation:**

$$a(\boldsymbol{x}) = b + \sum_i w_i x_i = b + \boldsymbol{w}^T \boldsymbol{x}$$

Neuron **post-activation:**

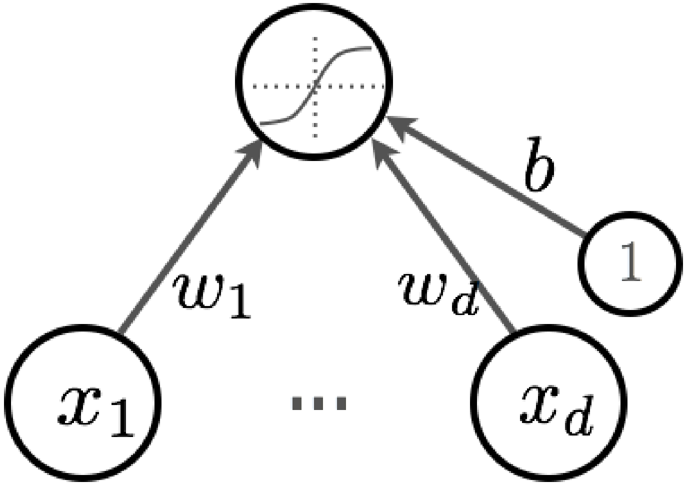
$$h(\boldsymbol{x}) = \sigma(b + \boldsymbol{w}^T \boldsymbol{x})$$

Where:

$\boldsymbol{w}$ are the **weights** (parameters)

b is the **bias** term

$\sigma(\cdot)$ is called the **activation function**

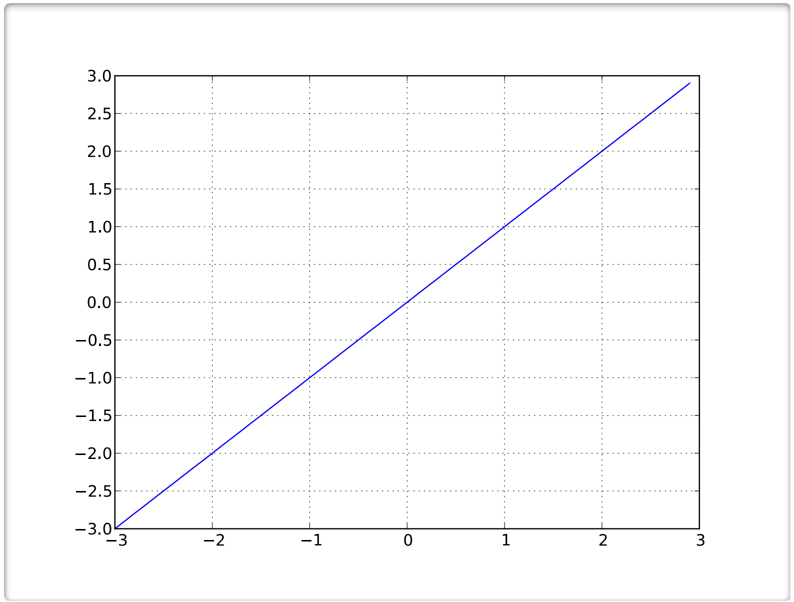


Popular activations

Linear activation function:

$$\sigma(a) = a$$

- No nonlinear transformation
- No output squashing
- Poor representational power (stay tuned)

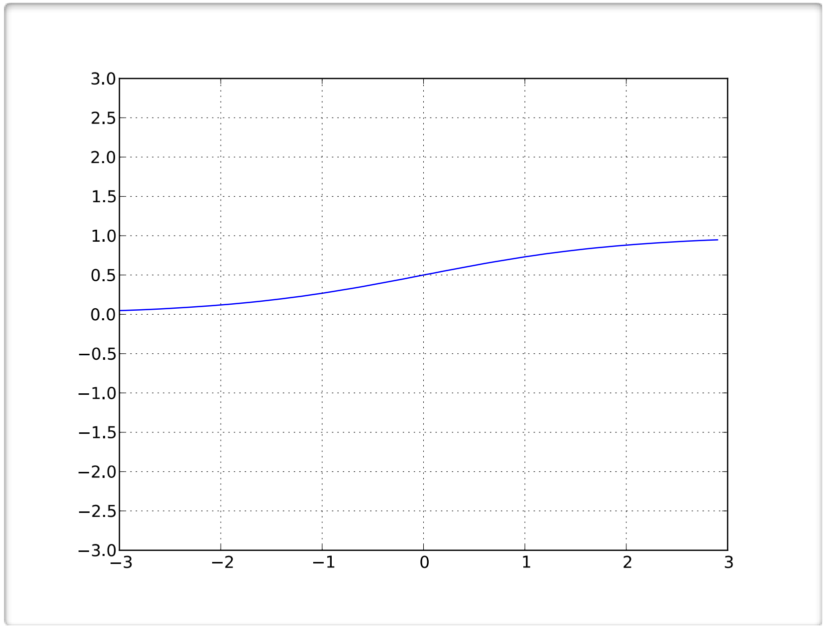


Popular activations

Sigmoid activation function:

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

- Squashes the neuron's output between 0 and 1: can be interpreted as $\mathrm{P}(\mathrm{ouput} \ = 1|a)$ (i.e. **logistic classifier**)
- Always positive
- Bounded
- Strictly Increasing

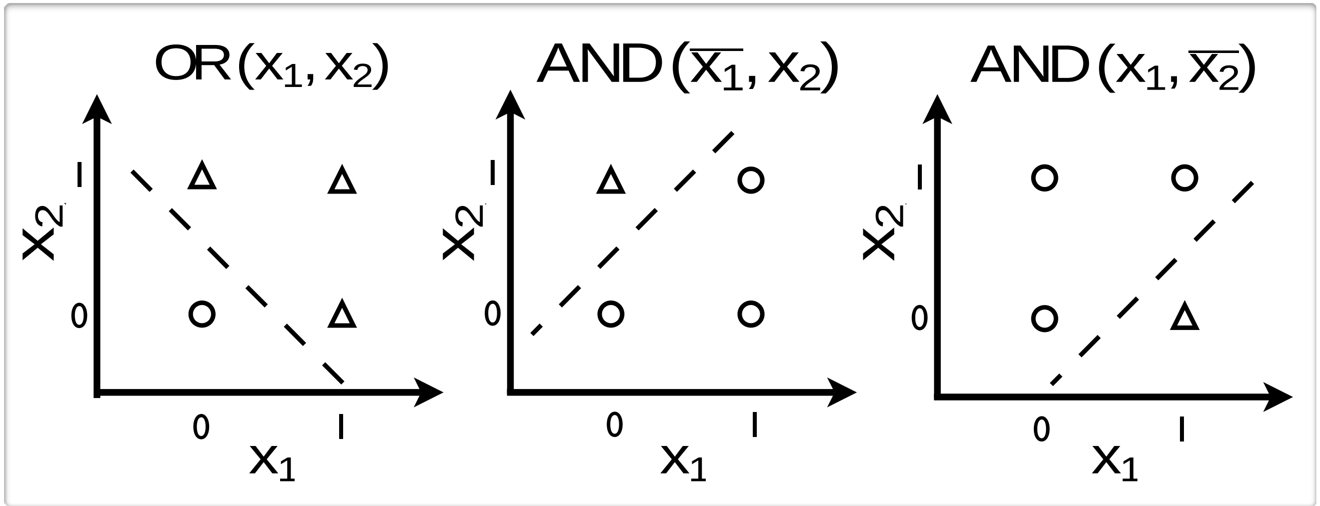


Aside: classification power of a single neuron

Sigmoid activation function: $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

- Squashes the neuron's output between 0 and 1: can be interpreted as $P(\text{ouput} \;= 1|a)$ (i.e. **logistic classifier**)

If activation is greater than 0.5, predict 1. Otherwise predict 0


$\mathrm{OR}(x_1, x_2)$
$\mathrm{AND}(\overline{x_1}, x_2)$
$\mathrm{AND}(x_1, \overline{x_2})$
x_2
x_1
0
1

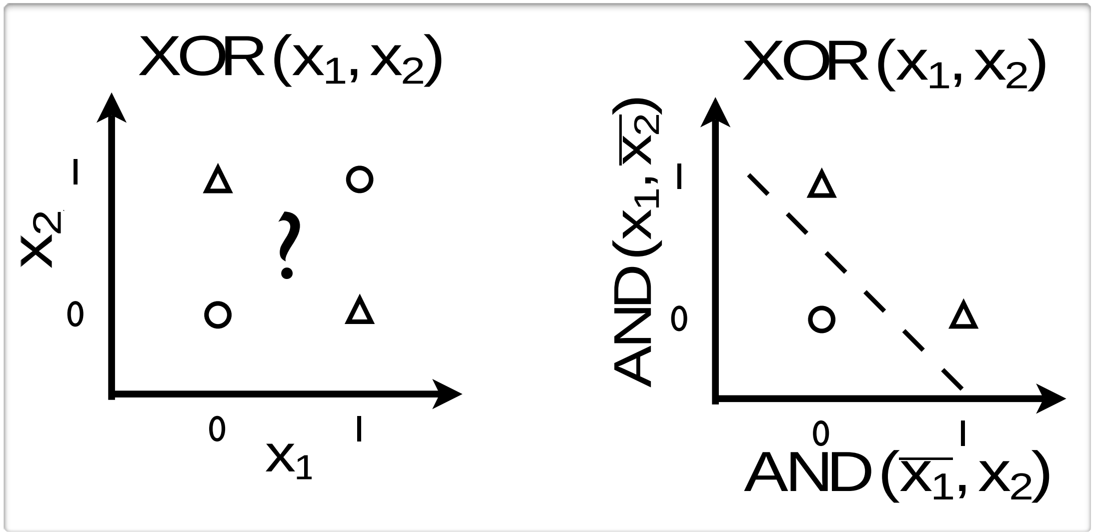
Can perfectly classify linearly separable datasets, e.g. OR, AND,

Aside: classification power of a single neuron

Sigmoid activation function: $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

- Squashes the neuron's output between 0 and 1: can be interpreted as P(ouput $= 1|a$) (i.e. **logistic classifier**)

→ If activation is greater than 0.5, predict 1. Otherwise predict 0



Cannot perfectly classify linearly non-separable datasets, e.g. XOR.

('69, Minsky and Papert, *Perceptrons)*

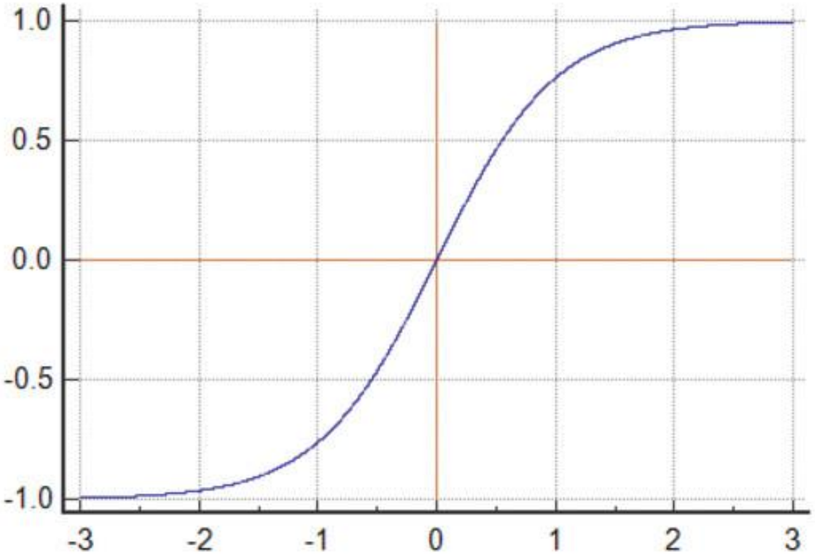
Popular activations

Hyperbolic tangent (“tanh”) activation function:

$$\sigma(a) = \tanh(a)$$

$$= \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

- Squashes neuron’s output between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

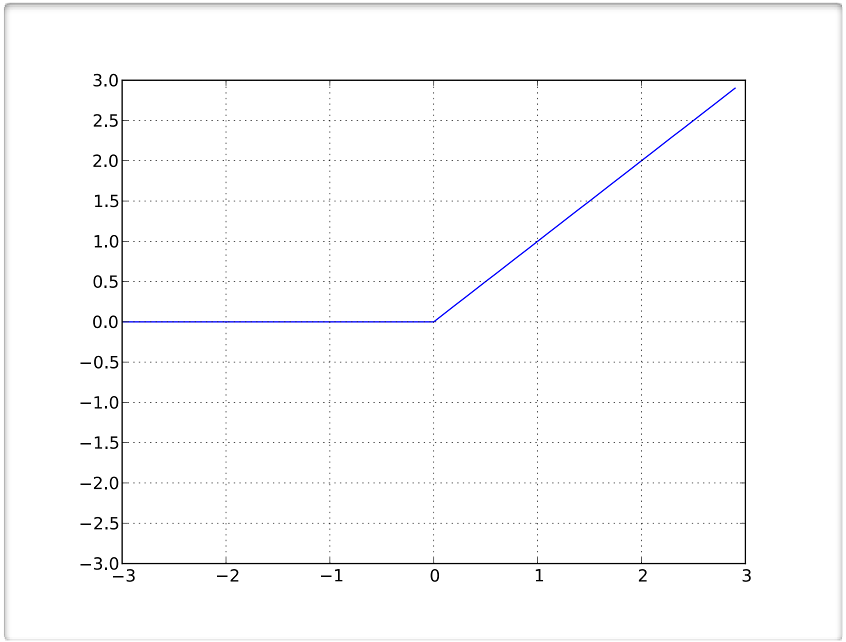


Popular activations

Rectified linear ("ReLU") activation function:

$$\sigma(a) = \max(a, 0)$$

- Bounded below by 0 (always non-negative)
- Tends to produce units with sparse activities
- Not upper bounded
- Strictly increasing



Single Hidden Layer Neural Net

Hidden layer **pre-activation**:

$$\boldsymbol{a}(\boldsymbol{x}) = \boldsymbol{b}^{(1)} + \boldsymbol{W}^{(1)}\boldsymbol{x}$$

$$\left(\boldsymbol{a}(\boldsymbol{x})_i = \boldsymbol{b}_i^{(1)} + \sum_j \boldsymbol{W}_{i,j}^{(1)}\boldsymbol{x}_j\right)$$

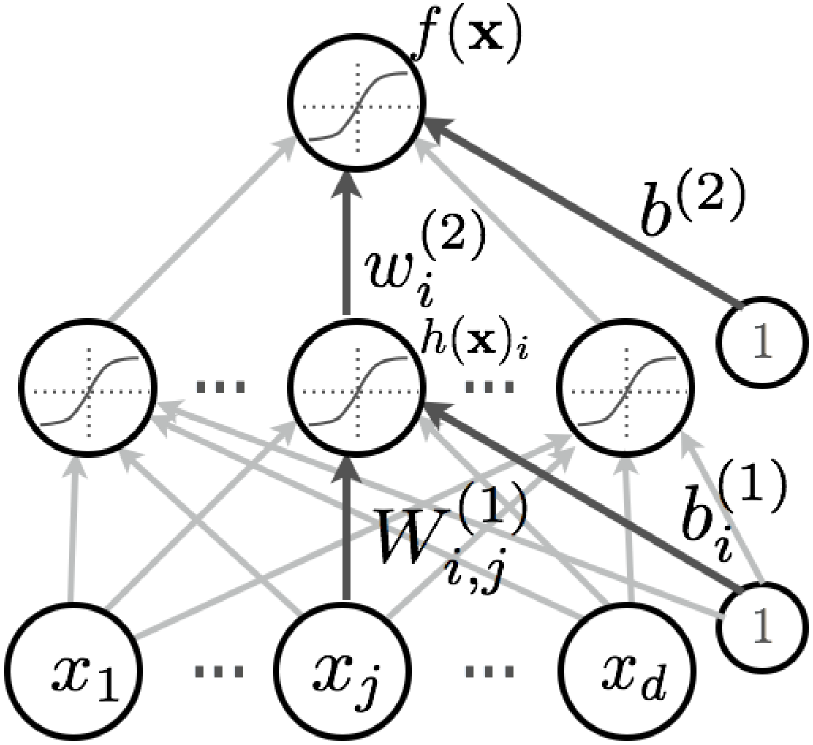
Hidden layer **post-activation**:

$$\boldsymbol{h}(\boldsymbol{x}) = \sigma(\boldsymbol{a}(\boldsymbol{x}))$$

Output layer activation:

$$\boldsymbol{f}(\boldsymbol{x}) = o(b^{(2)} + \boldsymbol{w}^{(2)^T}\boldsymbol{h}^{(1)}(\boldsymbol{x}))$$

Output activation function



Softmax output activation

In **multi-way classification**, we need multiple outputs (1 per class)

Natural: model calculates conditional probabilities P(ouput $= c|\boldsymbol{x}$)

Softmax activation function at the output

$$\mathbf{o}(\mathbf{a}) = \mathrm{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \ldots \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]^\top$$

- strictly positive
- sums to one

Predict class with the highest estimated class conditional probability.

Multilayer Neural Net

Consider a network with L hidden layers.

Layer **pre-activations**:

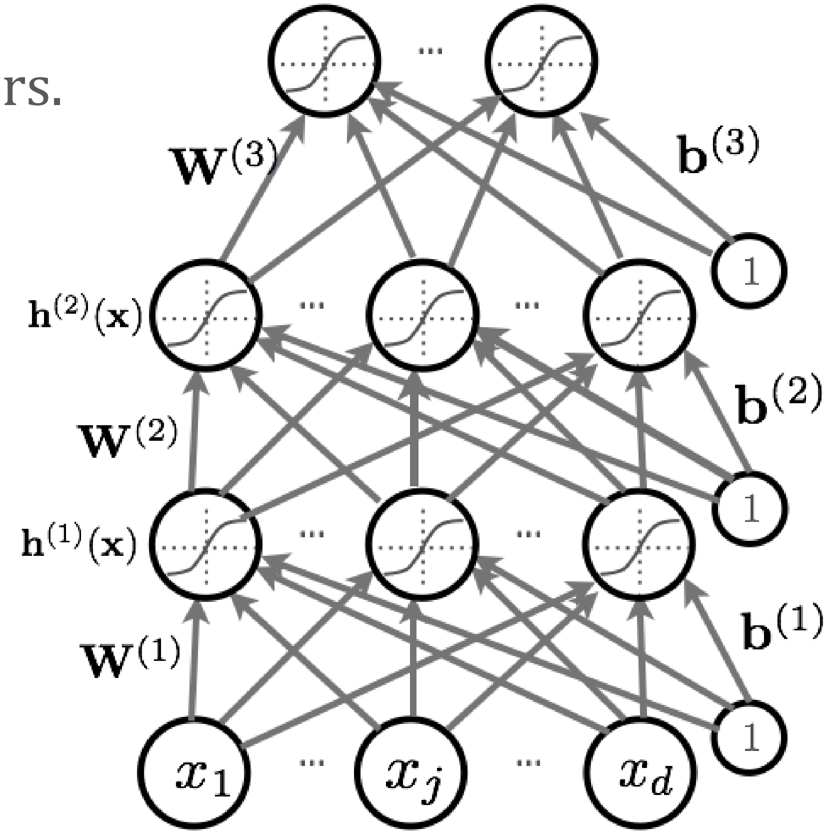
$$\boldsymbol{a}^{(k)}(\boldsymbol{x}) = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}(\boldsymbol{x})$$

Hidden layer post-activations:

$$\boldsymbol{h}^{(k)}(\boldsymbol{x}) = \sigma(\boldsymbol{a}^{(k)}(\boldsymbol{x}))$$

Output layer activation:

$$\boldsymbol{h}^{(L+1)}(\boldsymbol{x}) = o\left(\boldsymbol{a}^{(L+1)}(\boldsymbol{x})\right) = \boldsymbol{f}(\boldsymbol{x})$$



$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

Loss functions

Recall: typical approach is to minimize a training loss l over predictors $\mathcal{F}$:

$$\hat{f} = \operatorname*{argmin}_{f\in\mathcal{F}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Common losses:

l_2: $l(f(x), y) = ||f(x) - y||^2$, more common for **regression**, y can be vector or scalar

$0-1$: $l(f(x), y) = 1_{f(x)\neq y}$, ideal loss for **classification**, but poorly behaved for optimization

Log-loss/ cross entropy: $l(f(x), y) = -\log f(x)_y$, for f using a **softmax** output layer, well-behaved gradients

For softmax, $f(x)_c = P(\text{output } = c|\text{x})$, so we maximize the log-probability of correct label. Generalizes naturally when y **not** deterministic fn of x in $\mathcal{D}$:

$$-\log f(x)_y = -\textstyle\sum_c 1_{y=c} \log f(x)_c = -\sum_c 1_{y=c} \log P(\text{output } = c|\text{x})$$

Taking expectation of y: $\mathbb{E}_{y|x} l(f(x), y) = -\mathbb{E}_{y|x} \log P(\text{output} = y\,|x)$

Basic optimization algorithm: stochastic gradient descent

Recall: typical approach is to minimize a training loss l over predictors $\mathcal{F}$:

$$\hat{f} = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

Basic algorithm (Stochastic Gradient Descent)

Glossing over many details. Stay tuned.

```
- Initialize: $\theta_0 := \{W^{(1)}, b^{(1)}, \ldots, W^{(L+1)}, b^{(L+1)}\}$
- For t=1 to T
    - Pick a uniformly random training example $(x, y)$:
        - Set $\theta_{t+1} = \theta_t - \eta \underbrace{\nabla_\theta l(f_\theta(x, y))}$
```

Step size

***"Steepest" descent**: direction of most (local) improvement*

*Neural nets: gradients can be efficiently calculated, using **backpropagation***

Next time: universal approximation using neural networks