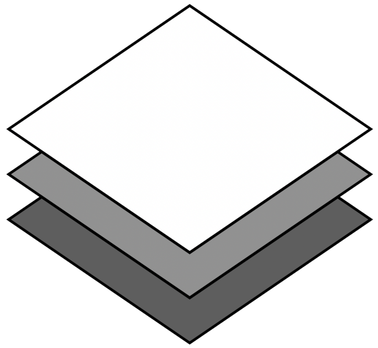


Deep Learning Fundamentals

There is a lot here

- We assume this is mostly review for most of you.
- If not, try to go through http://neuralnetworksanddeeplearning.com

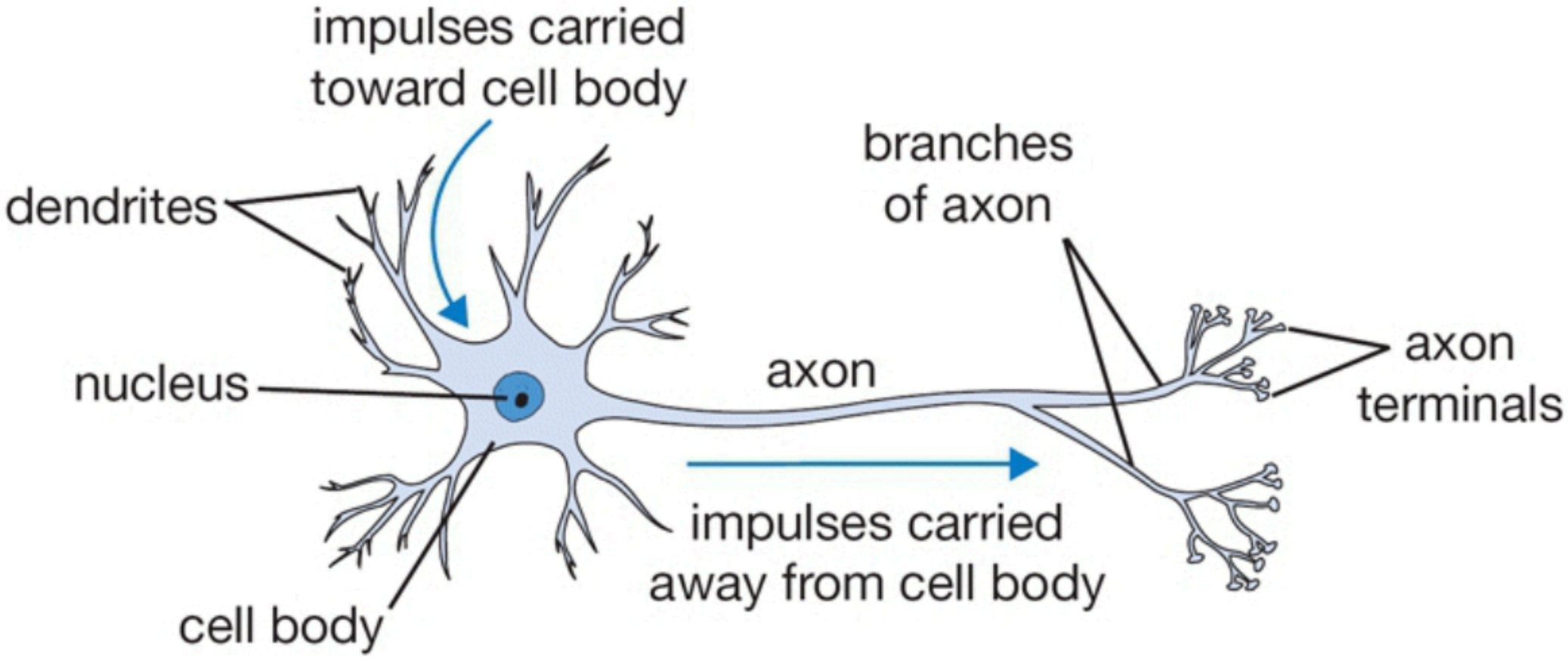
Outline

- Neural Networks
- Universality
- Learning Problems
- Empirical Risk Minimization / Loss Functions
- Gradient Descent
- Backpropagation / Automatic Differentiation
- Architectural Considerations (deep / conv / rnn)
- CUDA / Cores of Compute

Outline

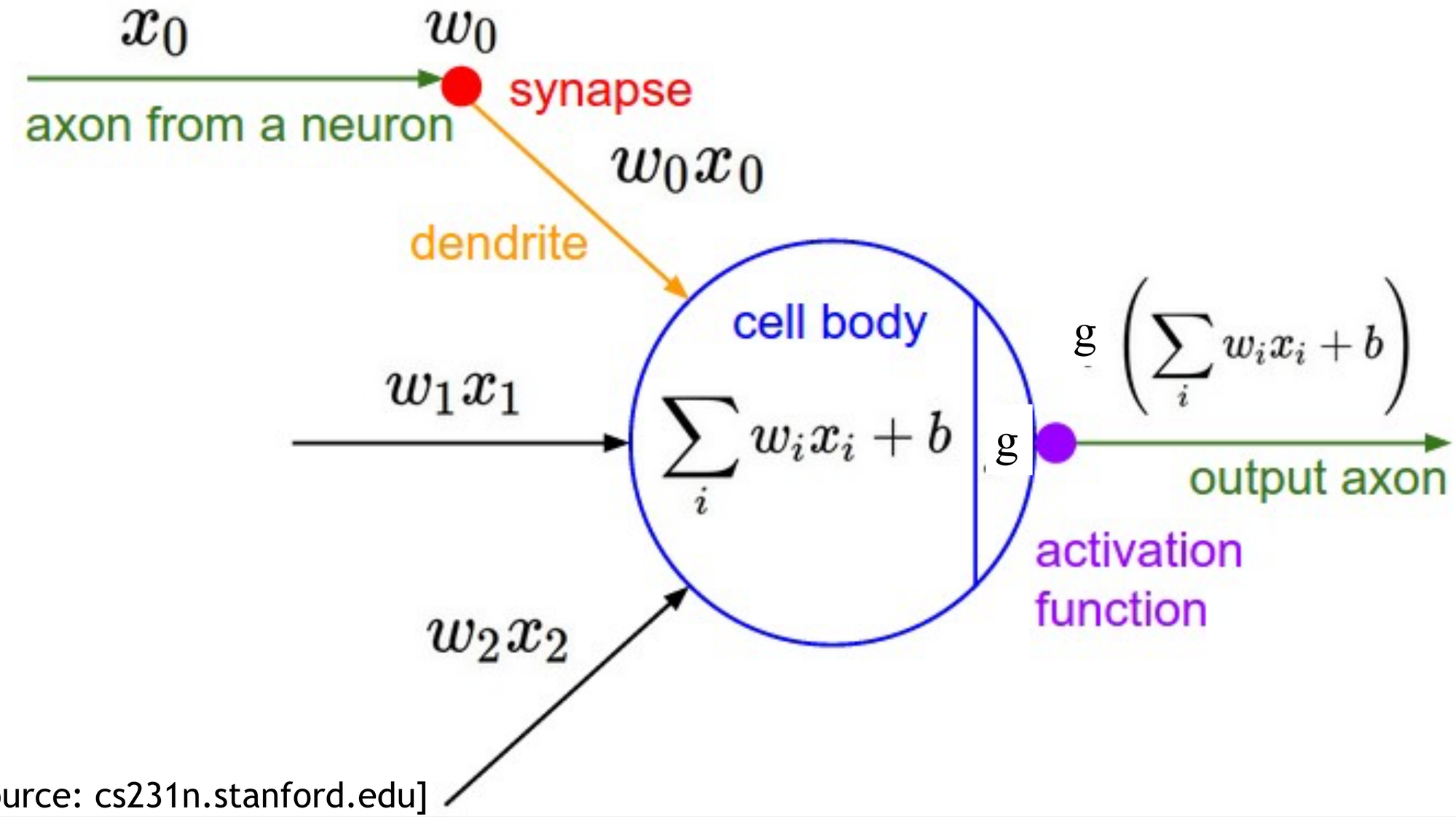
- ***Neural Networks***
- Universality
- Learning Problems
- Empirical Risk Minimization / Loss Functions
- Gradient Descent
- Backpropagation / Automatic Differentiation
- Architectural Considerations (deep / conv / rnn)
- CUDA / Cores of Compute

Single (Biological) Neuron



[image source: cs231n.stanford.edu]

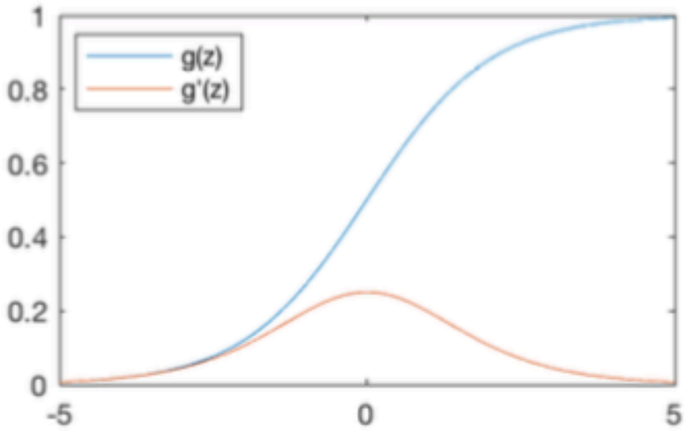
Single (Artificial) Neuron



[image source: cs231n.stanford.edu]

Common Activation Functions

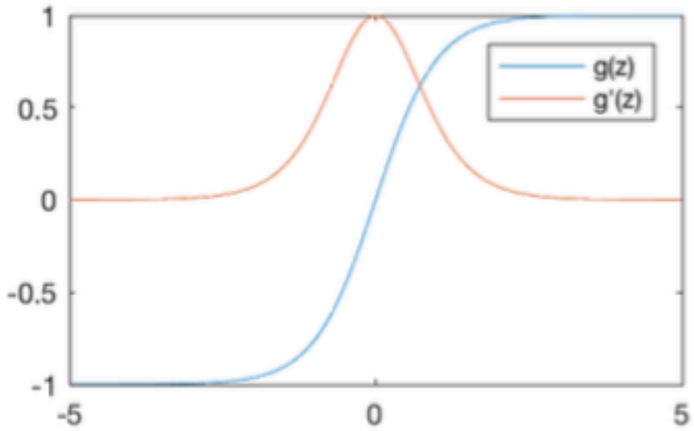
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

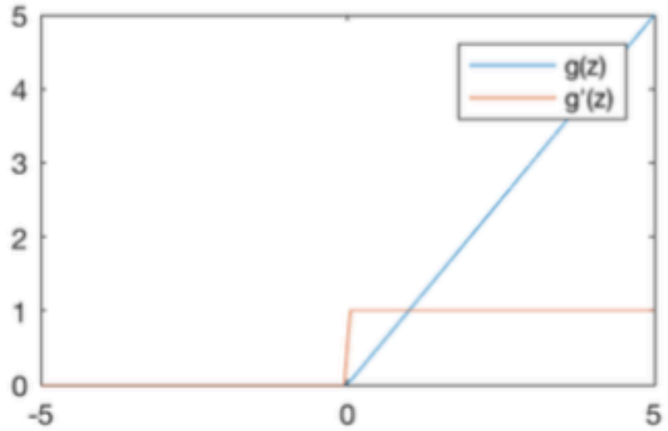
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

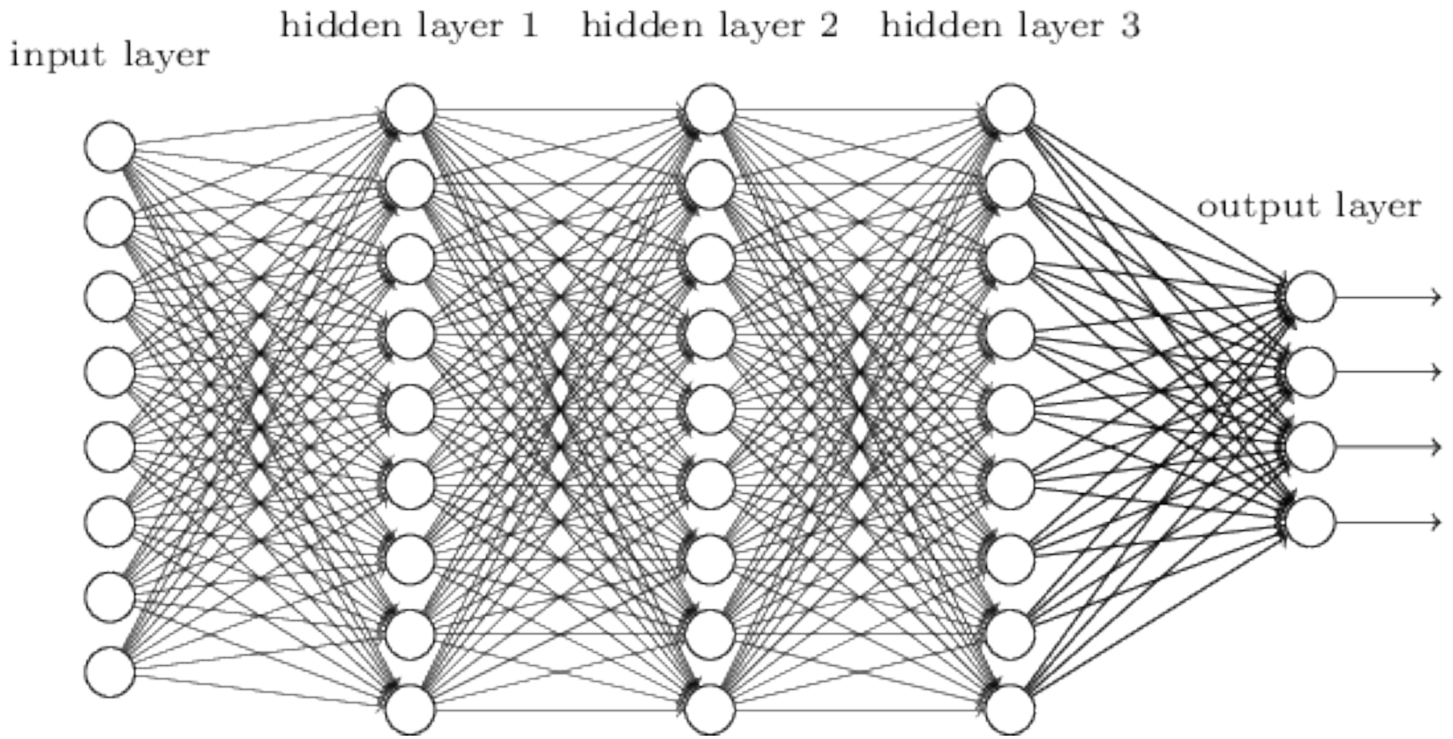


$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Neural Network



Notation: x $z^{(1)}$ $z^{(2)}$ $z^{(3)}$ $y = f(x, w)$

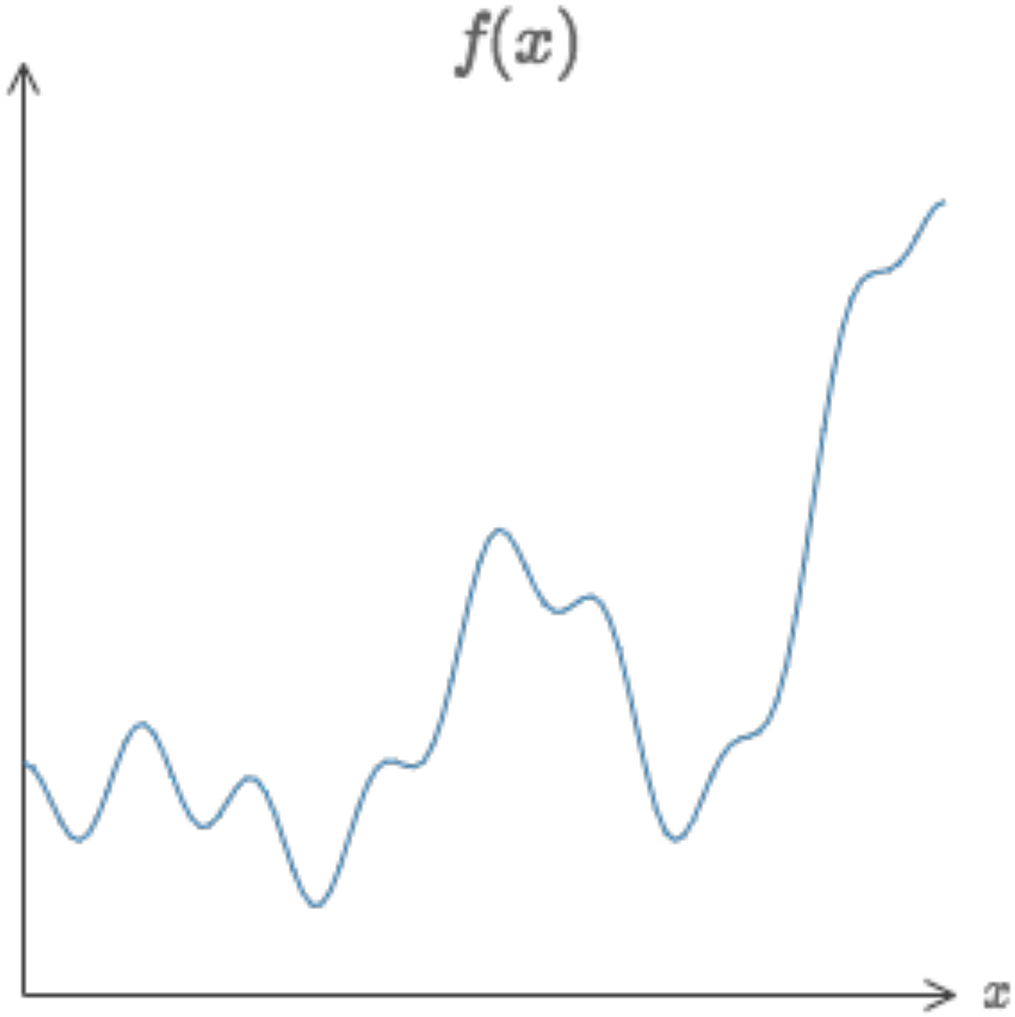
Choice of w determines the function from x --> y

Questions?

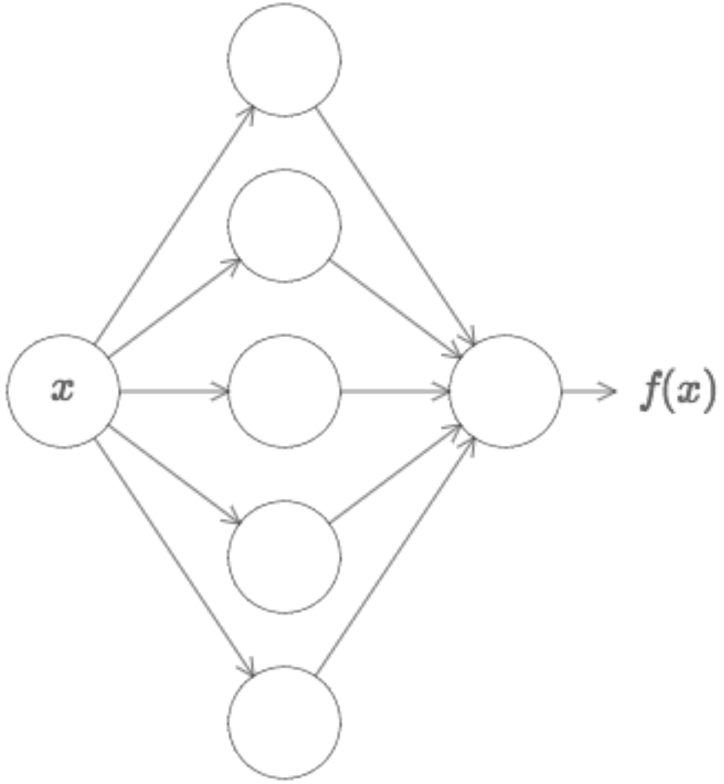
Outline

- Neural Networks
- ***Universality***
- Learning Problems
- Empirical Risk Minimization / Loss Functions
- Gradient Descent
- Backpropagation / Automatic Differentiation
- Architectural Considerations (deep / conv / rnn)
- CUDA / Cores of Compute

What Functions Can a Neural Net Represent?



Does there exist a choice for w to make this work?



[images source: neuralnetworksanddeeplearning.com]

Universal Function Approximation Theorem

Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure μ, standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

Hornik theorem 2: Whenever the activation function is *continuous, bounded and nonconstant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

- In words: Given any continuous function f(x), if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate f(x).

Cybenko (1989) “Approximations by superpositions of sigmoidal functions”
Hornik (1991) “Approximation Capabilities of Multilayer Feedforward Networks”
Leshno and Schocken (1991) ”Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function”

Universal Approximation Function

Math. Control Signals Systems (1989) 2: 303–314

Mathematics of Control, Signals, and Systems

Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.

1. Introduction

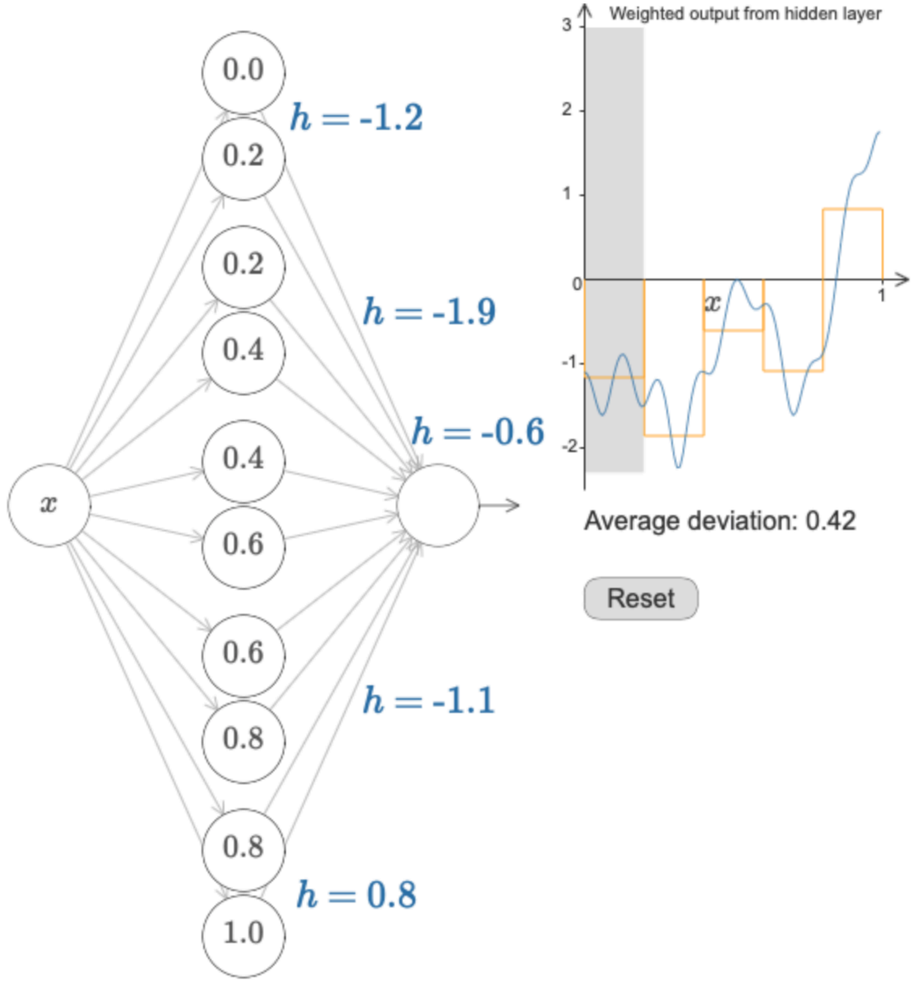
A number of diverse application areas are concerned with the representation of general functions of an n-dimensional real variable, $x \in \mathbb{R}^n$, by finite linear combinations of the form

$$\sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j) \qquad (1)$$

where $y_j \in \mathbb{R}^n$ and $\alpha_j, \theta \in \mathbb{R}$ are fixed. (y^T is the transpose of y so that $y^T x$ is the inner product of y and x.) Here the univariate function σ depends heavily on the context of the application. Our major concern is with so-called sigmoidal σ's:

$$\sigma(t) \to \begin{cases} 1 & \text{as} \quad t \to +\infty, \\ 0 & \text{as} \quad t \to -\infty. \end{cases}$$

Such functions arise naturally in neural network theory as the activation function of a neural node (or *unit* as is becoming the preferred term) [L1], [RHM]. The main result of this paper is a demonstration of the fact that sums of the form (1) are dense



Explore the idea interactively at
http://neuralnetworksanddeeplearning.com/chap4.html

Questions?

Outline

- Neural Networks
- Universality
- ***Learning Problems***
- Empirical Risk Minimization / Loss Functions
- Gradient Descent
- Backpropagation / Automatic Differentiation
- Architectural Considerations (deep / conv / rnn)
- CUDA / Cores of Compute

Types of Learning Problems

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

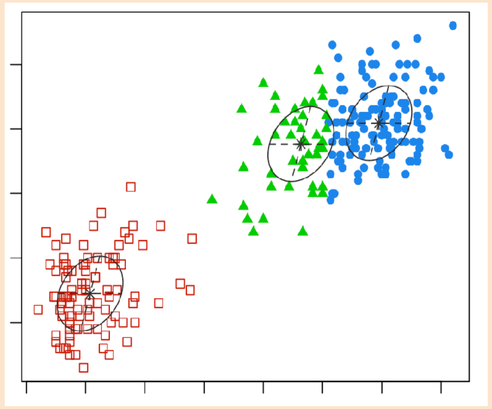
- [also transfer learning, imitation learning, meta-learning, …]

Types of Learning Problems

Unsupervised Learning

- ➔ Unlabeled data X
- ➔ Learn X
- ➔ Generate fakes, insights

"This product does what it is supposed to. I always keep three of these in my kitchen just in case ever I need a replacement cord."



Supervised Learning

- ➔ Labeled data X and Y
- ➔ Learn X -> Y
- ➔ Make Predictions

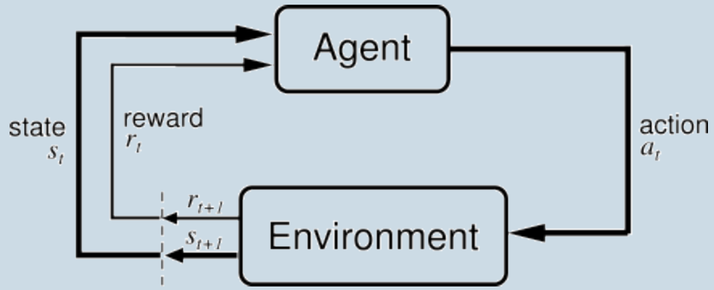
→ *cat*

→ *"Hey Siri"*

Commercially Viable Today

Reinforcement Learning

- ➔ Learn how to take Actions in an Environment



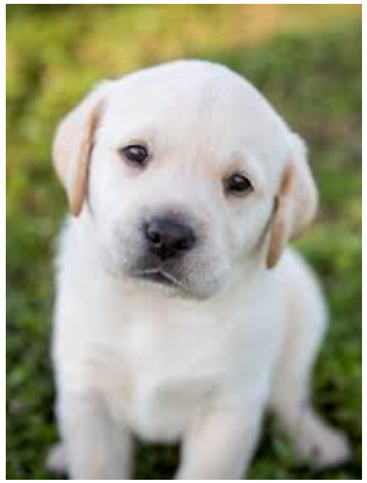
Up Next

Supervised Learning: X → Y

- Ex 1: Image Recognition
  - X = pixel values
  - Y = one hot vector encoding category

$$\rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

x y



$$\rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

x y

[figure sources: https://en.wiktionary.org/wiki/cat; https://www.guidedogs.org/]

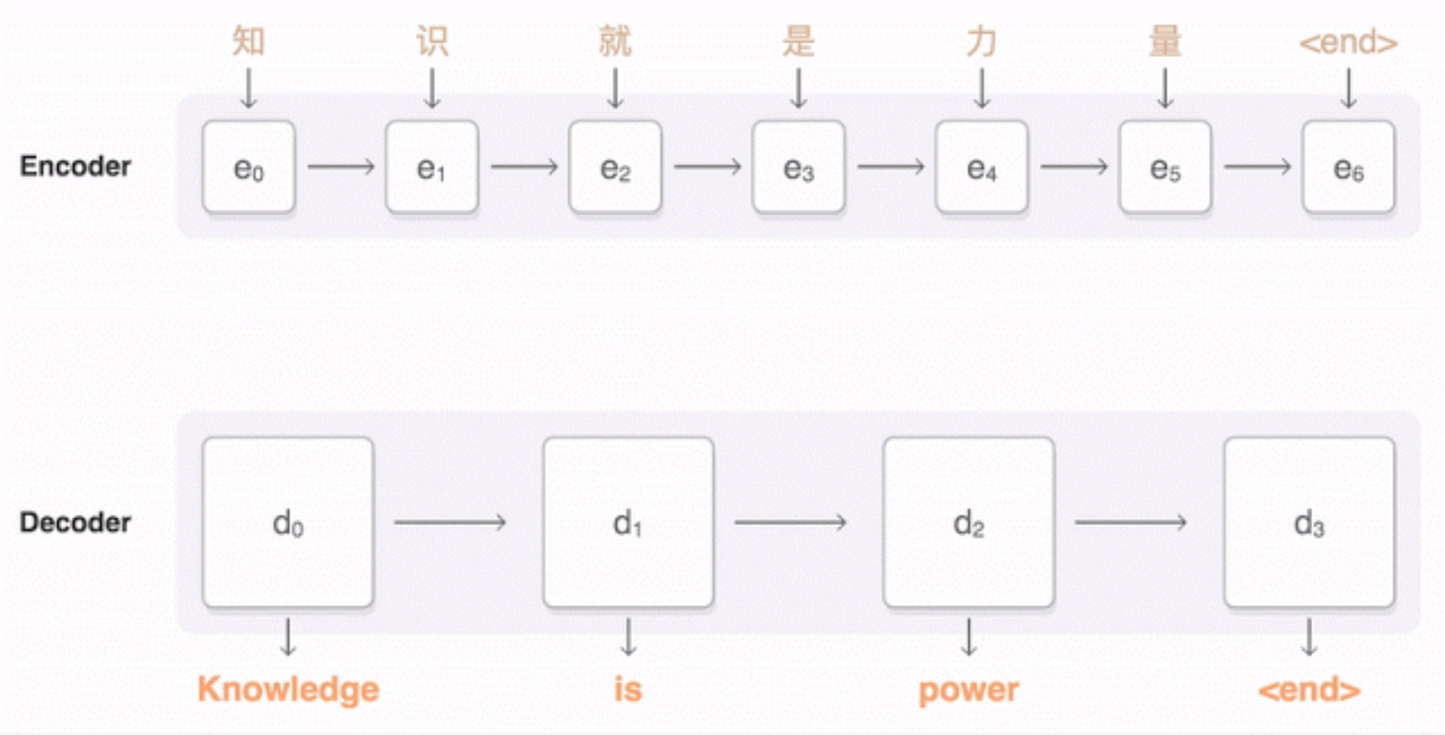
Supervised Learning: X → Y

- Ex 2: Speech Recognition
  - X = sequence of pressure readings
  - Y = sequence of one-hot encodings of words

[source: vitecinc.com]

Supervised Learning: X → Y

- Ex 3: Machine Translation
  - X = sequence of one-hot encodings of words in first language
  - Y = sequence of one-hot encodings of words in second language

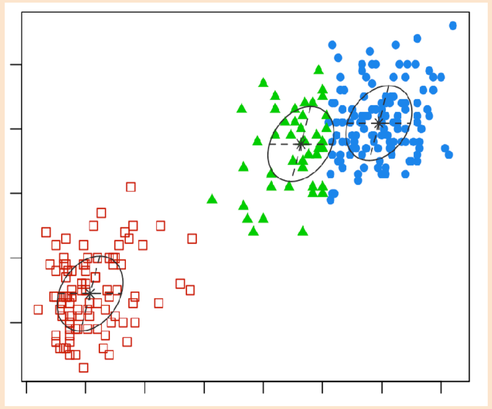


Types of Learning Problems

Unsupervised Learning

- ➔ Unlabeled data X
- ➔ Learn X
- ➔ Generate fakes, insights

"This product does what it is supposed to. I always keep three of these in my kitchen just in case ever I need a replacement cord."



Supervised Learning

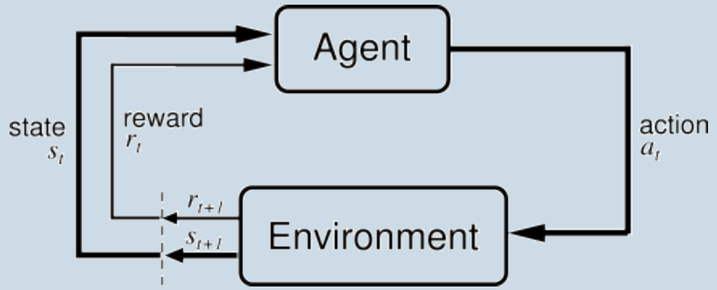
- ➔ Labeled data X and Y
- ➔ Learn X -> Y
- ➔ Make Predictions

→ *cat*

→ *"Hey Siri"*

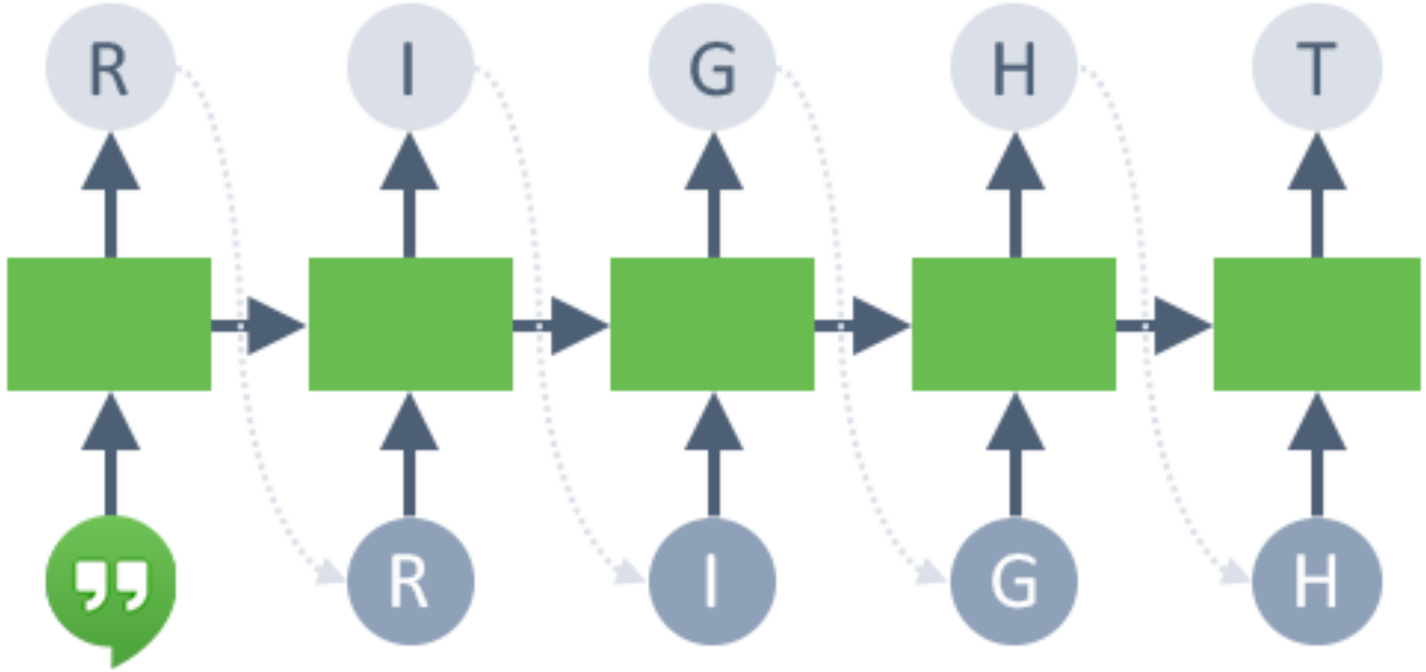
Reinforcement Learning

- ➔ Learn how to take Actions in an Environment



Unsupervised Learning: X

- Ex. 1: predict next character (charRNN)



[source: Tommy Mullaney]

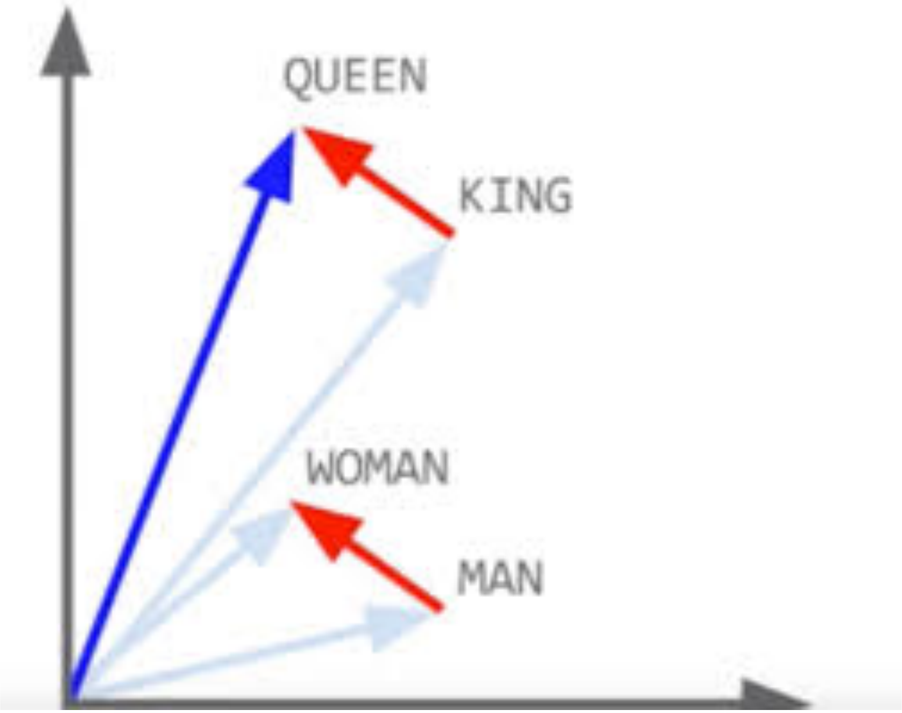
Example of next character prediction

This product does what it is supposed to. I always keep three of these in my kitchen just in case ever I need a replacement cord.	Great little item. Hard to put on the crib without some kind of embellishment. My guess is just like the screw kind of attachment I had.

[Radford et al, 2017]

Unsupervised Learning: X

- Ex. 2: predict nearby words (word2vec)



[Mikolov et al, 2013]

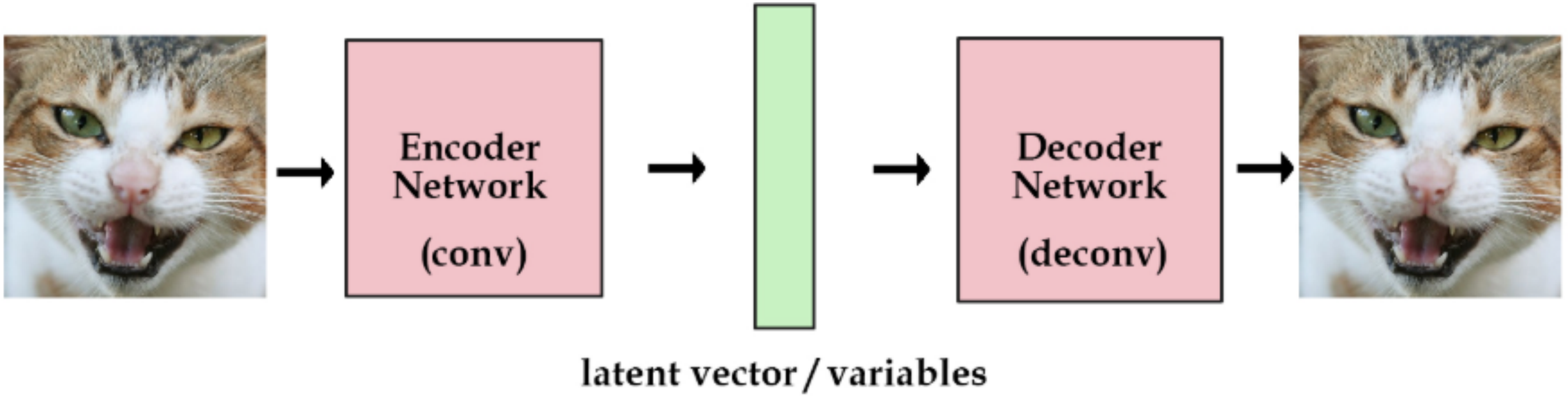
Unsupervised Learning: X

- Ex. 3: predict next pixel (pixelCNN)

[van den Oord et al, 2016]

Unsupervised Learning: X

- Ex. 4: Variational Autoencoder (VAE): X -> Z -> Xhat

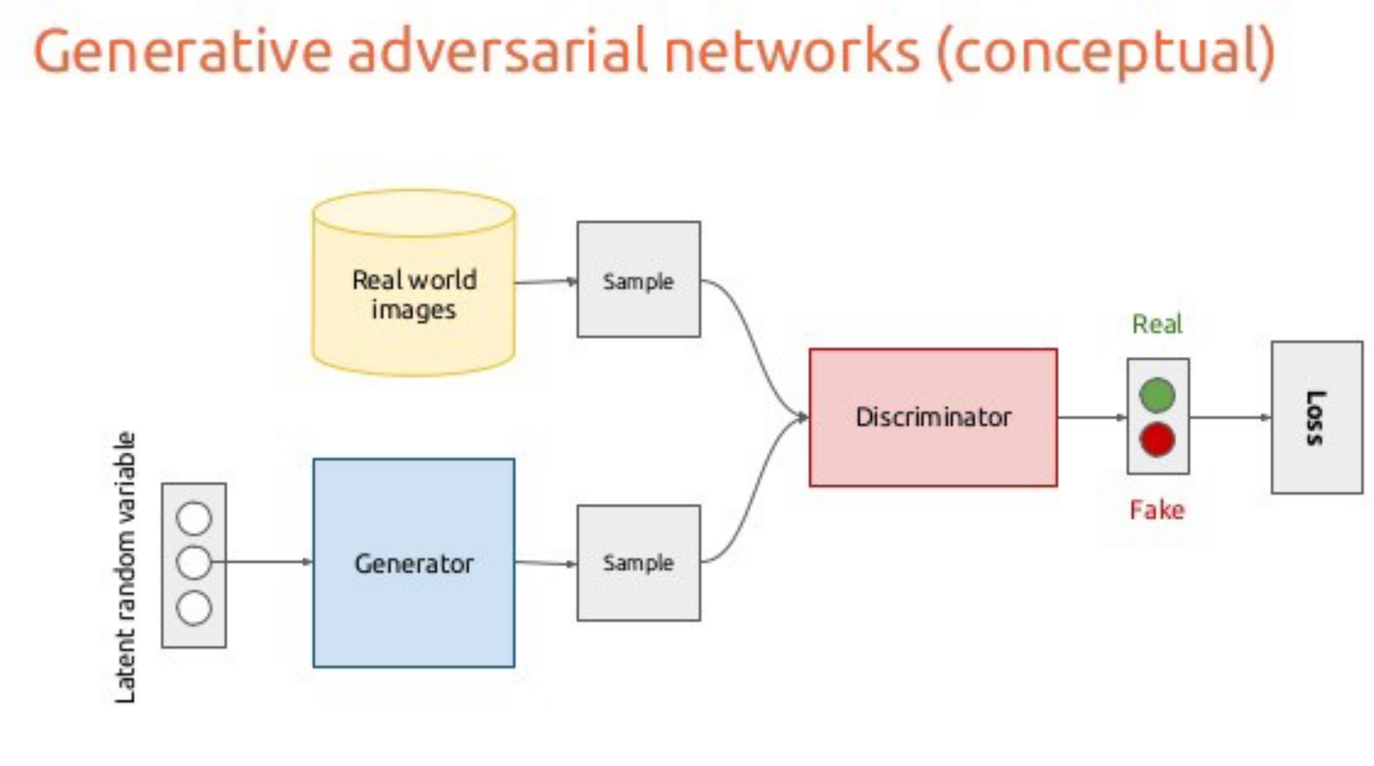


^introduce noise here

[Kingma and Welling, 2014] [Figure: Kevin Frans]

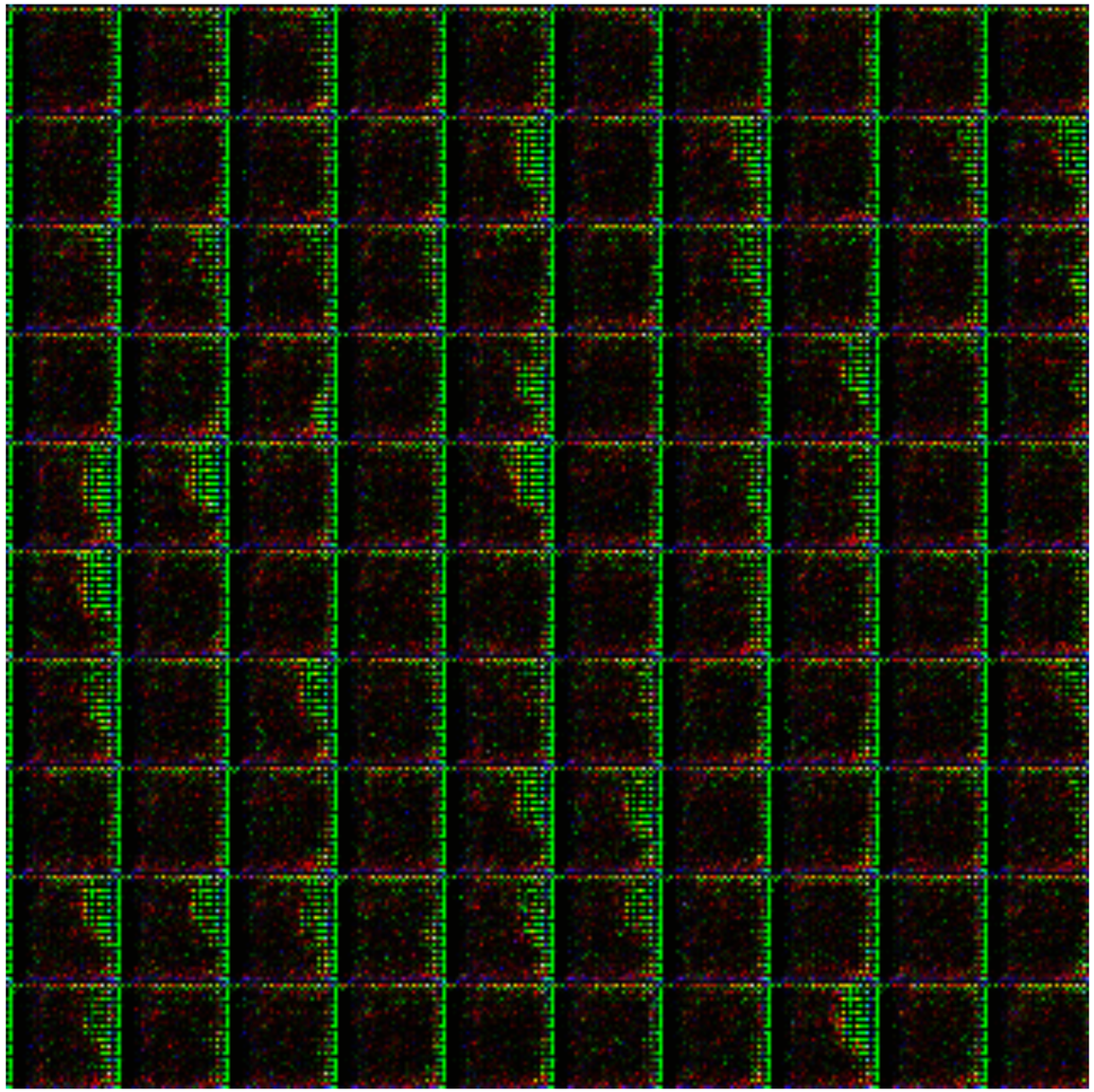
Unsupervised Learning: X

- Ex. 5: predict X that is indistinguishable from real X (GAN)



[Goodfellow et al, 2015]. [Figure source: https://www.slideshare.net/xavigiro

Example of Unsupervised Learning in Action



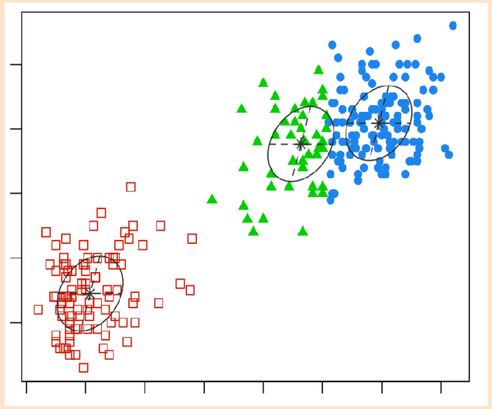
[Salimans, Goodfellow, Zaremba, Cheung, Radford & Chen, NIPS 2016]

Types of Learning Problems

Unsupervised Learning

- ➔ Unlabeled data X
- ➔ Learn X
- ➔ Generate fakes, insights

"This product does what it is supposed to. I always keep three of these in my kitchen just in case ever I need a replacement cord."



Supervised Learning

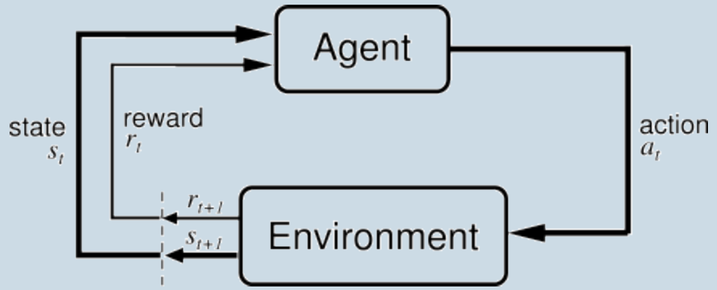
- ➔ Labeled data X and Y
- ➔ Learn X -> Y
- ➔ Make Predictions

→ *cat*

→ *"Hey Siri"*

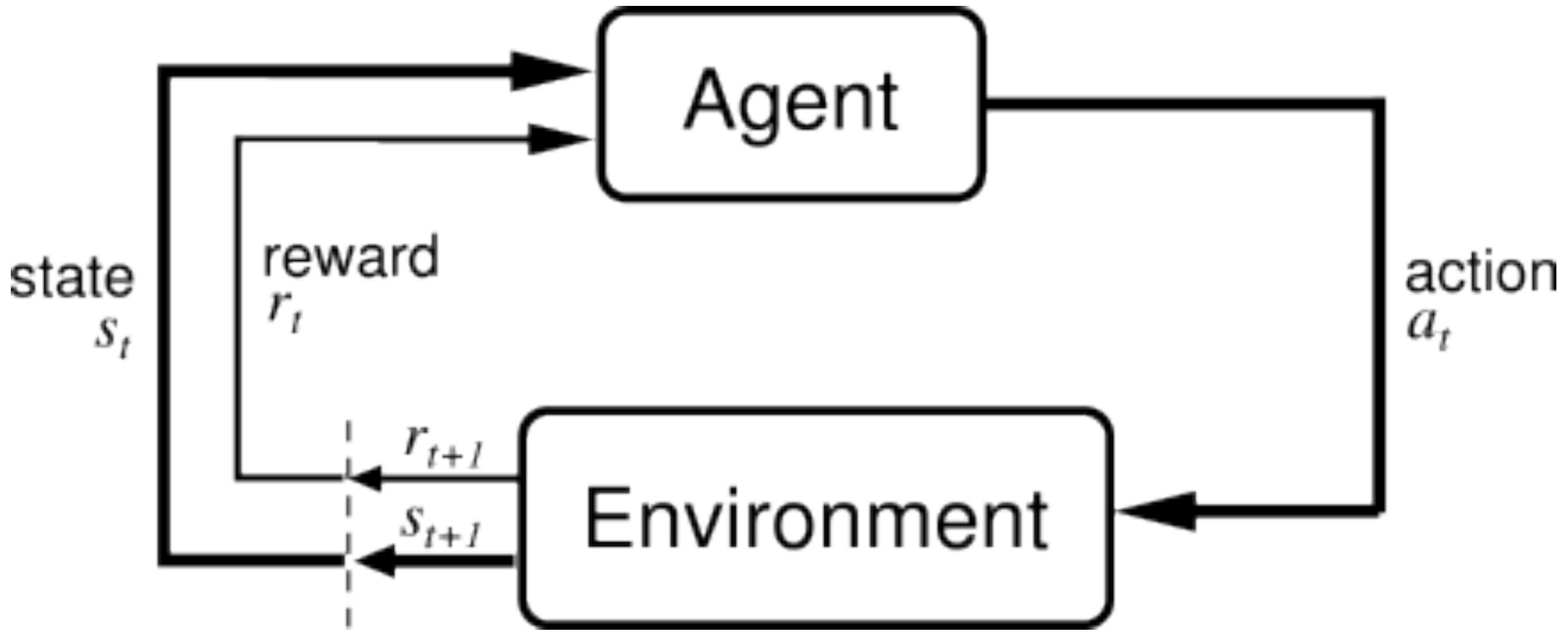
Reinforcement Learning

- ➔ Learn how to take Actions in an Environment



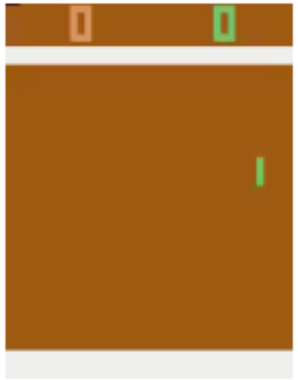
Reinforcement Learning

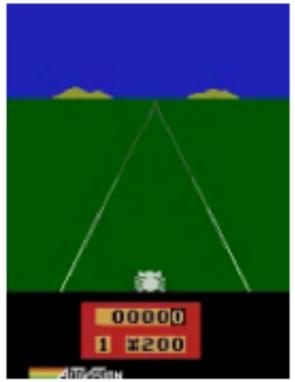
- Learn to act to maximize reward

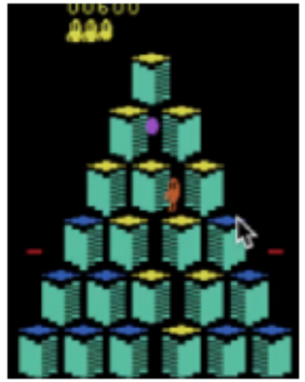


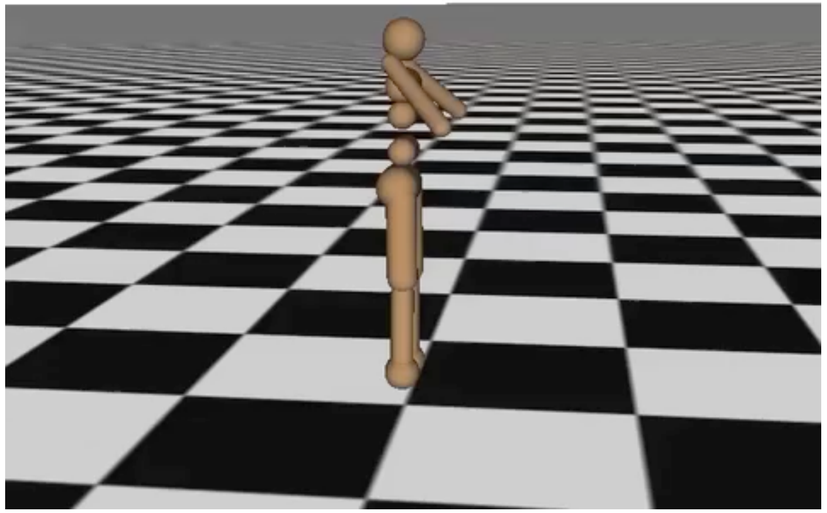
$$\max_{\theta} \quad \mathrm{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$$

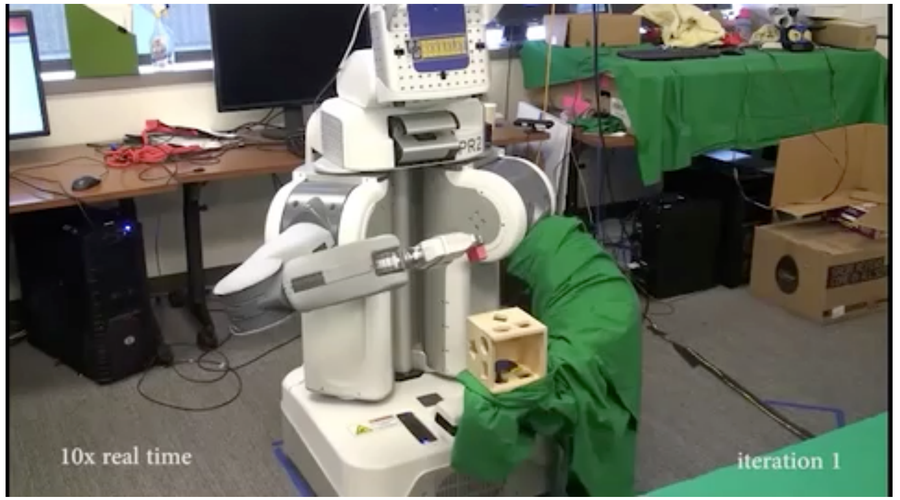
Reinforcement Learning Examples











Types of Learning Problems

- Supervised Learning: learn X --> Y
- Unsupervised Learning: learn X
- Reinforcement Learning: learn to interact with environment
  x_t -> a_t, x_{t+1} -> a_{t+1}, …
- [also transfer learning, imitation learning, meta-learning, …]

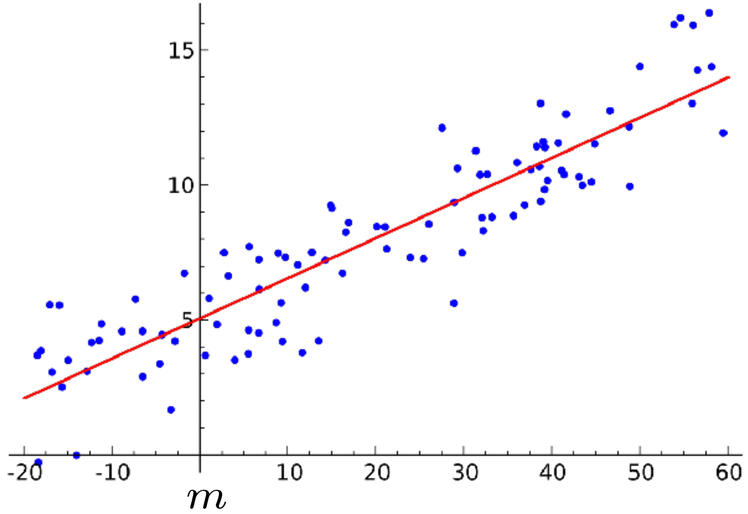
Questions?

Outline

- Neural Networks
- Universality
- Learning Problems
- ***Empirical Risk Minimization / Loss Functions***
- Gradient Descent
- Backpropagation / Automatic Differentiation
- Architectural Considerations (deep / conv / rnn)
- CUDA / Cores of Compute

Linear Regression

- Why this line?
  - "Best fit through data"
  - Formally, minimize squared error:



$$\min_{w,b} \sum_{i=1}^{m} (w \cdot x^{(i)} + b - y^{(i)})^2$$

  - More generally, minimize loss L:

$$\min_{w,b} \sum_{i=1}^{m} L(f_{w,b}(x^{(i)}), y^{(i)})$$

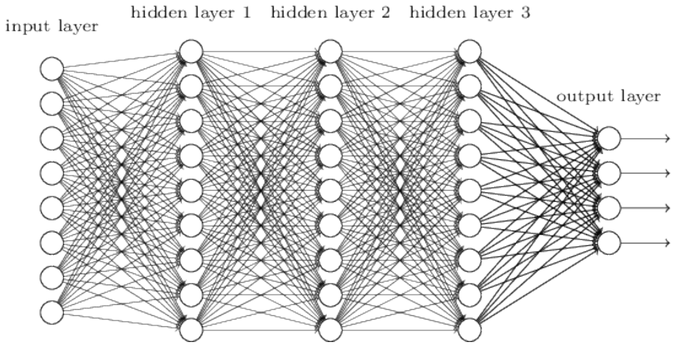
[Figure source: Wikipedia]

Empirical Risk Minimization = minimize loss L as measured (empirically) on the data

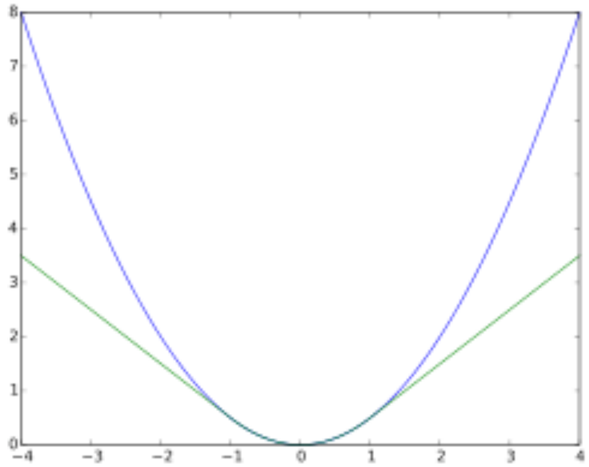
Neural Net Regression

- Find w, b parameters that optimize loss:

$$\min_{w,b} \sum_{i=1}^{m} L(f_{w,b}(x^{(i)}), y^{(i)})$$



- Typical losses:
  - Mean Squared Error (MSE)
  - Huber loss (= more robust to outliers)

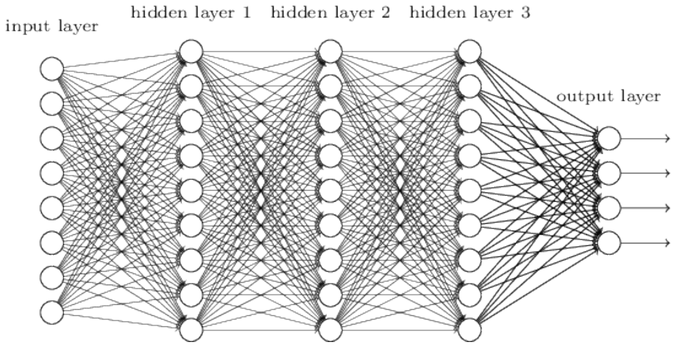


[Figure source: Wikipedia]

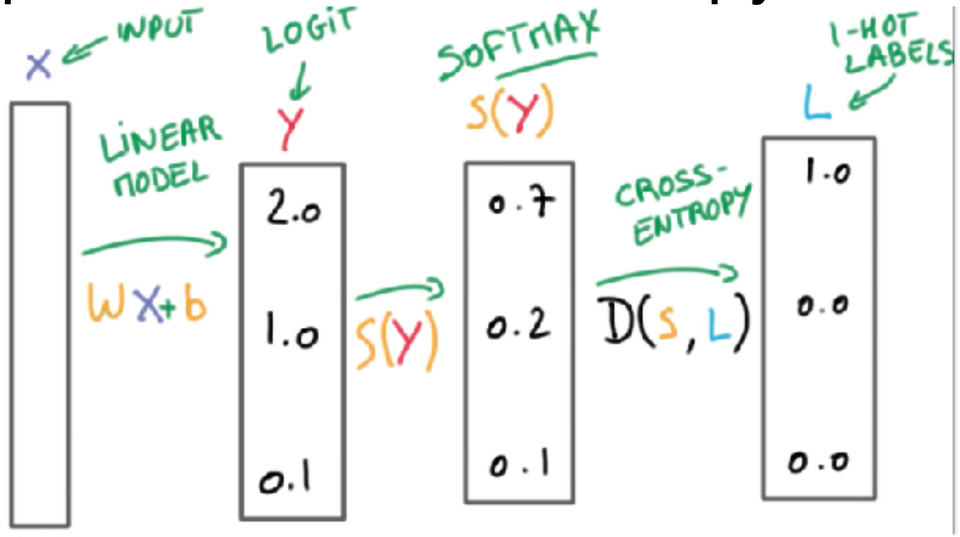
Neural Net Classification

- Find w, b parameters that optimize loss:

$$\min_{w,b} \sum_{i=1}^{m} L(f_{w,b}(x^{(i)}), y^{(i)})$$



- Typical loss: cross-entropy



[figure source: Ritchie Ng]

$$(s(y))_j = \frac{e^{y_j}}{\sum_i e^{y_i}}$$

$$D(s, l) = -\sum_i l_i \log s_i$$

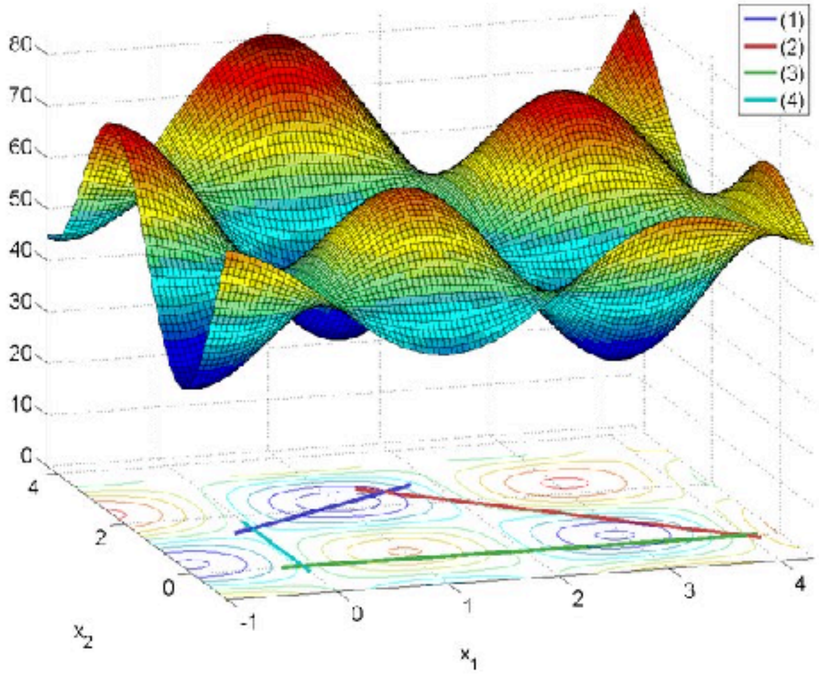
Outline

- Neural Networks
- Universality
- Learning Problems
- Empirical Risk Minimization / Loss Functions
- ***Gradient Descent***
- Backpropagation / Automatic Differentiation
- Architectural Considerations (deep / conv / rnn)
- CUDA / Cores of Compute

Optimizing the Loss

- Our goal: find w,b that optimize

$$\min_{w,b} \mathcal{L}(w,b) = \sum_{i=1}^{m} L(f_{w,b}(x^{(i)}), y^{(i)})$$



How to Improve One Parameter?

- Update w_i

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \mathcal{L}(w, b)$$

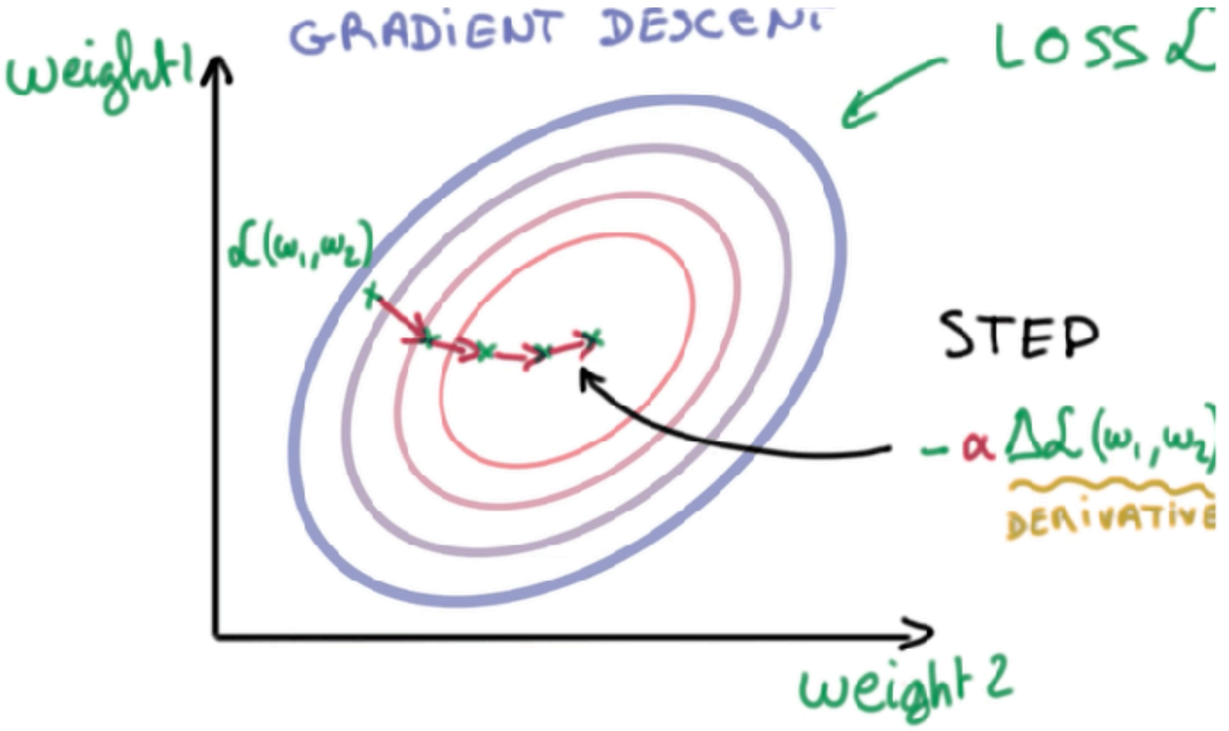
$$\frac{\partial}{\partial w_i} \mathcal{L}(w, b) = \lim_{\varepsilon \to 0} \frac{\mathcal{L}(w + \varepsilon e_i, b) - \mathcal{L}(w - \varepsilon e_i, b)}{2\varepsilon}$$

How to Improve All Parameters?

- Gradient Descent (aka Steepest Descent):

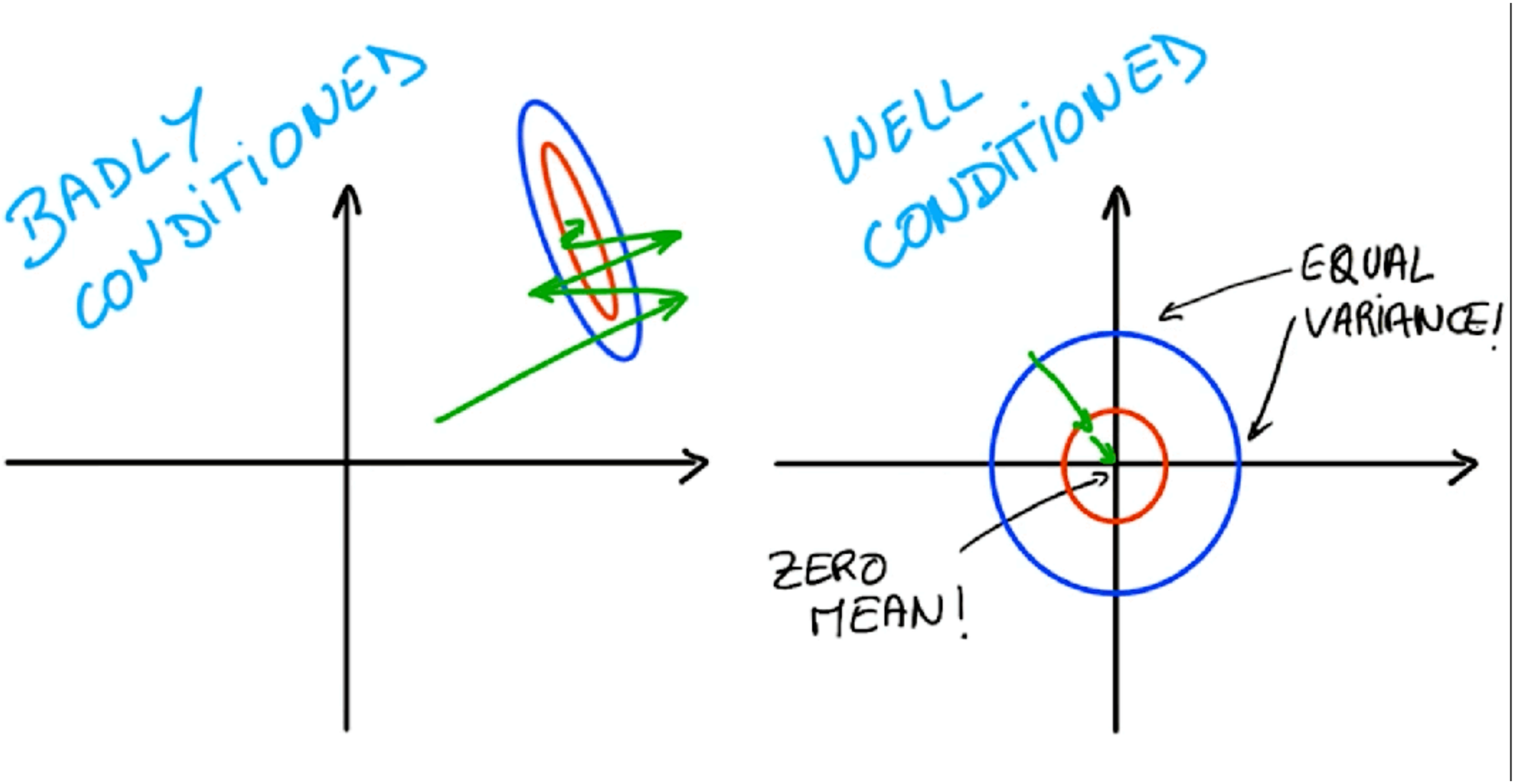
$$w \leftarrow w - \alpha \nabla_w \mathcal{L}(w, b)$$

$$(\nabla_w \mathcal{L}(w, b))_i = \frac{\partial}{\partial w_i} \mathcal{L}(w, b)$$



[figure source: neuralnetworksanddeeplearning.com (Nielsen)]

Conditioning



[figure source: neuralnetworksanddeeplearning.com (Nielsen)]

Conditioning

- Initialization (more later)
- Normalization
  - Batch norm, weight norm, layer norm, … (more later)
- Second order methods:
  - Exact:
    - Newton's method
    - Natural gradient
  - Approximate second order methods:
    - Adagrad, Adam, Momentum

Sampling Schemes for Gradient Descent

- Gradient Descent

$$w \leftarrow w - \alpha \nabla_w \sum_{i=1}^{m} L(w, b, x^{(i)}, y^{(i)})$$

- Stochastic Gradient Descent
  - Compute each gradient step on just a subset (“batch”) of data
  - → less compute per step
  - → more noisy per step
  - Overall: faster progress per amount of compute

Questions?

Outline

- Neural Networks
- Universality
- Learning Problems
- Empirical Risk Minimization / Loss Functions
- Gradient Descent
- ***Backpropagation / Automatic Differentiation***
- Architectural Considerations (deep / conv / rnn)
- CUDA / Cores of Compute

Our Status

- We have reduced learning to optimizing a loss function:

$$\min_{w,b} \mathcal{L}(w,b) = \sum_{i=1}^{m} L(f_{w,b}(x^{(i)}), y^{(i)})$$

- We can do this by (stochastic) gradient descent, which iterates:

$$w \leftarrow w - \alpha \nabla_w \sum_{i \in \text{minibatch}} L(w, b, x^{(i)}, y^{(i)})$$

- How to efficiently compute the gradients?

Gradients are just derivatives

- Derivatives tables:

$$\frac{d}{dx}(a) = 0$$
$$\frac{d}{dx}(x) = 1$$
$$\frac{d}{dx}(au) = a\frac{du}{dx}$$
$$\frac{d}{dx}(u+v-w) = \frac{du}{dx}+\frac{dv}{dx}-\frac{dw}{dx}$$
$$\frac{d}{dx}(uv) = u\frac{dv}{dx}+v\frac{du}{dx}$$
$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{1}{v}\frac{du}{dx}-\frac{u}{v^2}\frac{dv}{dx}$$
$$\frac{d}{dx}(u^n) = nu^{n-1}\frac{du}{dx}$$
$$\frac{d}{dx}(\sqrt{u}) = \frac{1}{2\sqrt{u}}\frac{du}{dx}$$
$$\frac{d}{dx}\left(\frac{1}{u}\right) = -\frac{1}{u^2}\frac{du}{dx}$$
$$\frac{d}{dx}\left(\frac{1}{u^n}\right) = -\frac{n}{u^{n+1}}\frac{du}{dx}$$
$$\frac{d}{dx}[f(u)] = \frac{d}{du}[f(u)]\frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{d}{dx}[\log_e u] = \frac{1}{u}\frac{du}{dx}$$
$$\frac{d}{dx}[\log_a u] = \log_a e\frac{1}{u}\frac{du}{dx}$$
$$\frac{d}{dx}e^u = e^u\frac{du}{dx}$$
$$\frac{d}{dx}a^u = a^u \ln a\frac{du}{dx}$$
$$\frac{d}{dx}(u^v) = vu^{v-1}\frac{du}{dx}+\ln u\ u^v\frac{dv}{dx}$$
$$\frac{d}{dx}\sin u = \cos u\frac{du}{dx}$$
$$\frac{d}{dx}\cos u = -\sin u\frac{du}{dx}$$
$$\frac{d}{dx}\tan u = \sec^2 u\frac{du}{dx}$$
$$\frac{d}{dx}\cot u = -\csc^2 u\frac{du}{dx}$$
$$\frac{d}{dx}\sec u = \sec u\tan u\frac{du}{dx}$$
$$\frac{d}{dx}\csc u = -\csc u\cot u\frac{du}{dx}$$

- But neural net f is never one of those?
  - No problem: CHAIN RULE:

If $f(x) = g(h(x))$

Then $f'(x) = g'(h(x))h'(x)$

[source: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/derfunc.html

Automatic Differentation

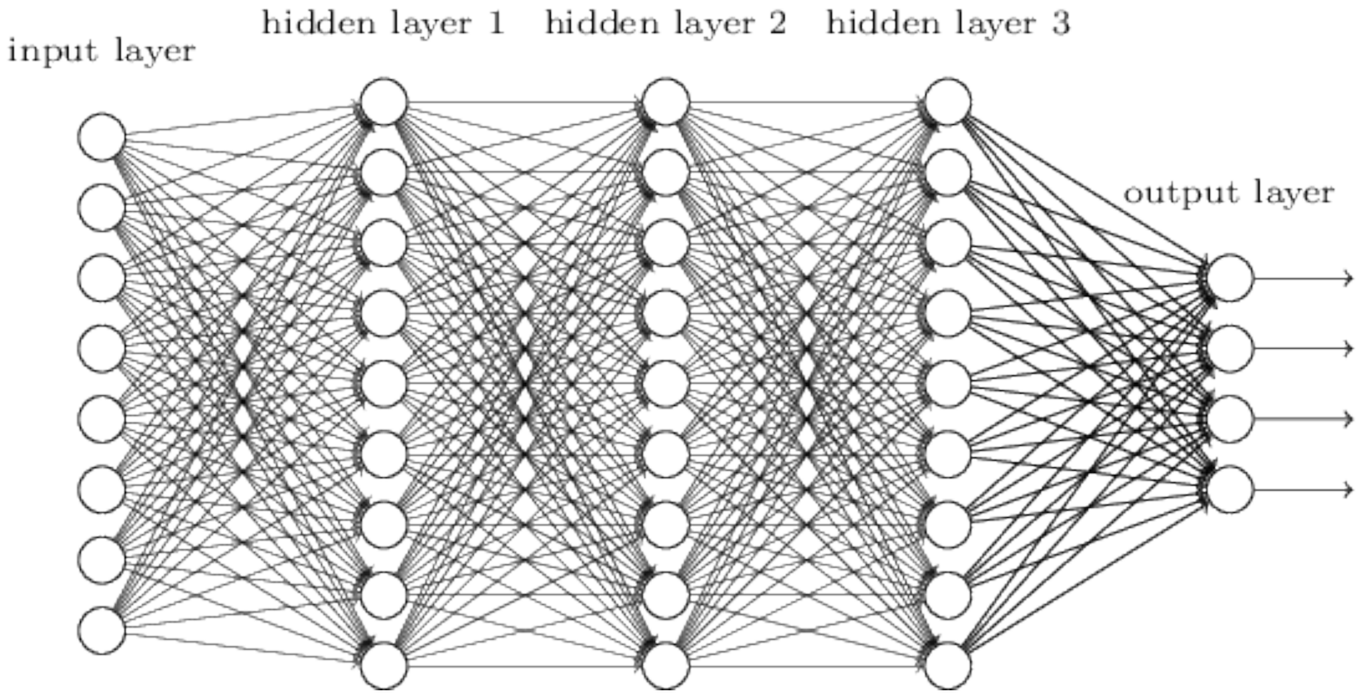
- Automatic differentiation software
  - e.g. PyTorch, TensorFlow, Theano, Chainer, etc.
  - Only need to program the function f(x,w).
  - Software automatically computes all derivatives
  - This is typically done by caching info during forward computation pass of f, and then doing a backward pass = “backpropagation”

Outline

- Neural Networks
- Universality
- Learning Problems
- Empirical Risk Minimization / Loss Functions
- Gradient Descent
- Backpropagation / Automatic Differentiation
- ***Architectural Considerations (deep / conv / rnn)***
- CUDA / Cores of Compute

Neural Net Architectures

- “simplest”: sequence of fully connected layers:



NN Architecture Considerations

- Data efficiency:
  - Extremely large networks can represent anything (see “universal function approximation theorem”) but might also need extremely large amount of data to latch onto the right thing
  - → Encode prior knowledge into the architecture, e.g.:
    - Computer vision: Convolutional Networks = spatial translation invariance
    - Sequence processing (e.g. NLP): Recurrent Networks = temporal invariance
- Optimization landscape / conditioning:
  - Depth over Width, Skip connections, Batch / Weight / Layer Normalization
- Computational / Parameter efficiency
  - Factorized convolutions
  - Strided convolutions

Outline

- Neural Networks
- Universality
- Learning Problems
- Empirical Risk Minimization / Loss Functions
- Gradient Descent
- Backpropagation / Automatic Differentiation
- Architectural Considerations (deep / conv / rnn)
- ***CUDA / Cores of Compute***

CUDA

- Lastly, why did deep learning explosion really kick off around 2013?
  - Bigger datasets are one part of the story
  - Good libraries for matrix computations on GPUs
- Why are GPUs crucial?
  - All NN computations are just matrix multiplications, which are well parallelized onto the many cores of a GPU.

Questions?