# 10707
# Deep Learning: Spring 2021

Andrej Risteski

Machine Learning Department

## **Lecture 2:**
## Representational power of neural networks

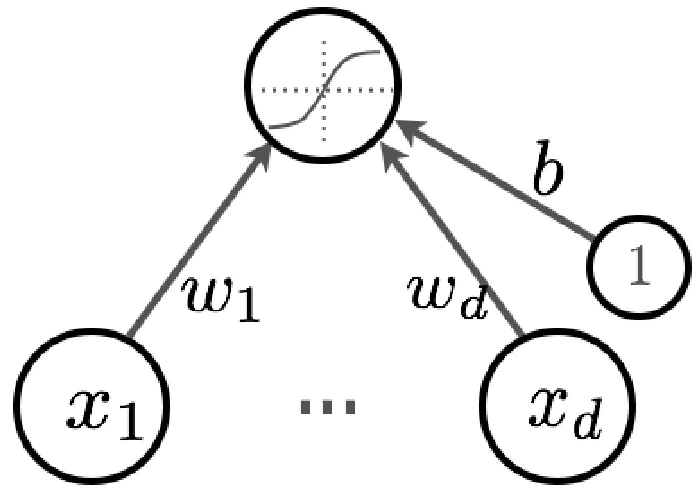
# Neural network basics: the artificial neuron

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

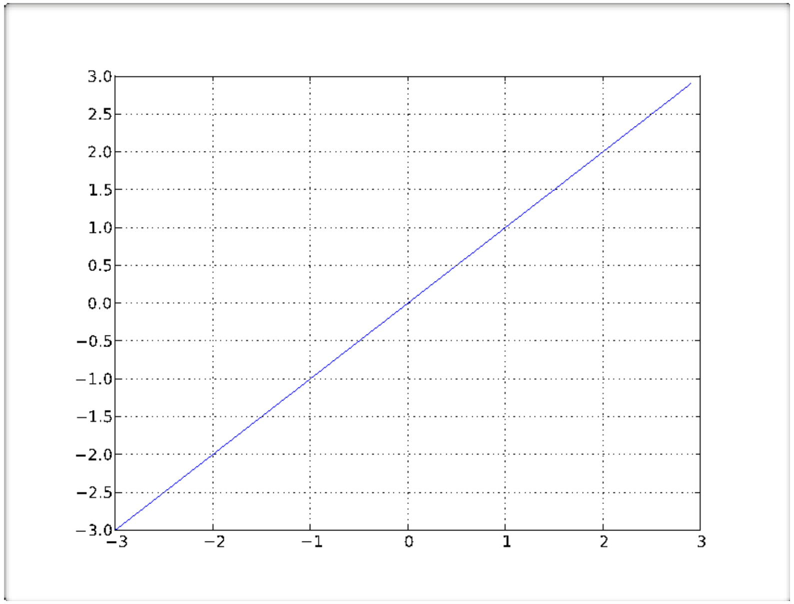
$\sigma(\cdot)$ is called the **activation function**

# Popular activations

Linear activation function:

$$\sigma(a) = a$$

- No nonlinear transformation
- No output squashing
- Poor representational power (linear composed w/ linear = linear)

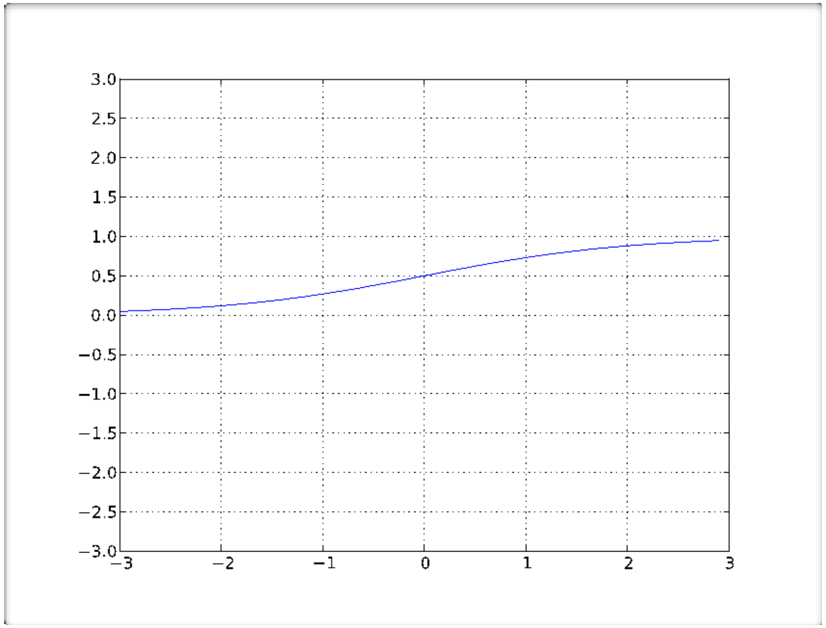


# Popular activations

Sigmoid activation function:

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

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

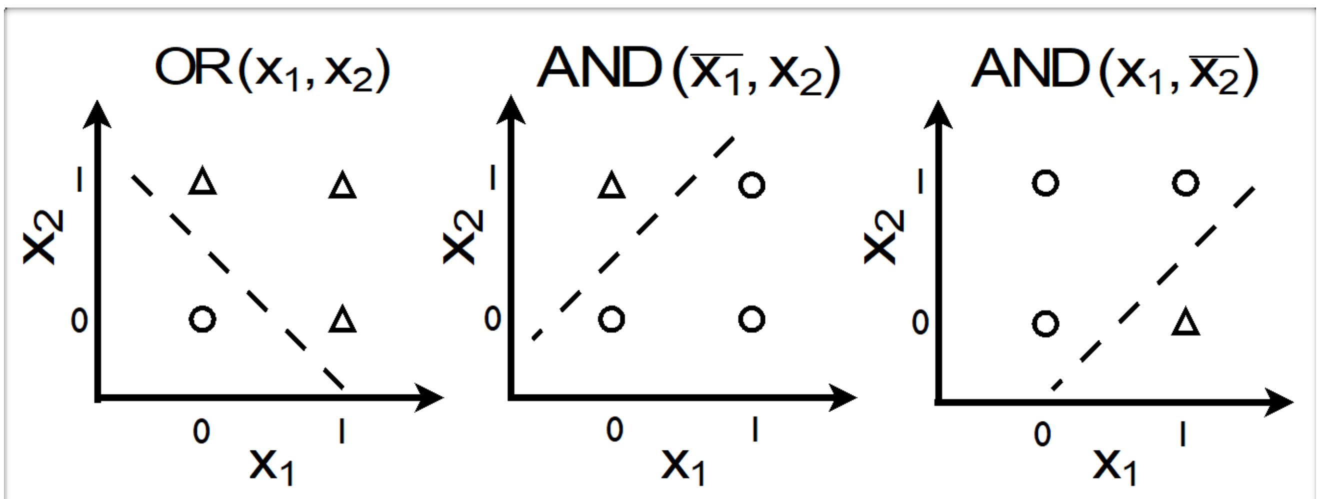


# Aside: classification power of a single neuron

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

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

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

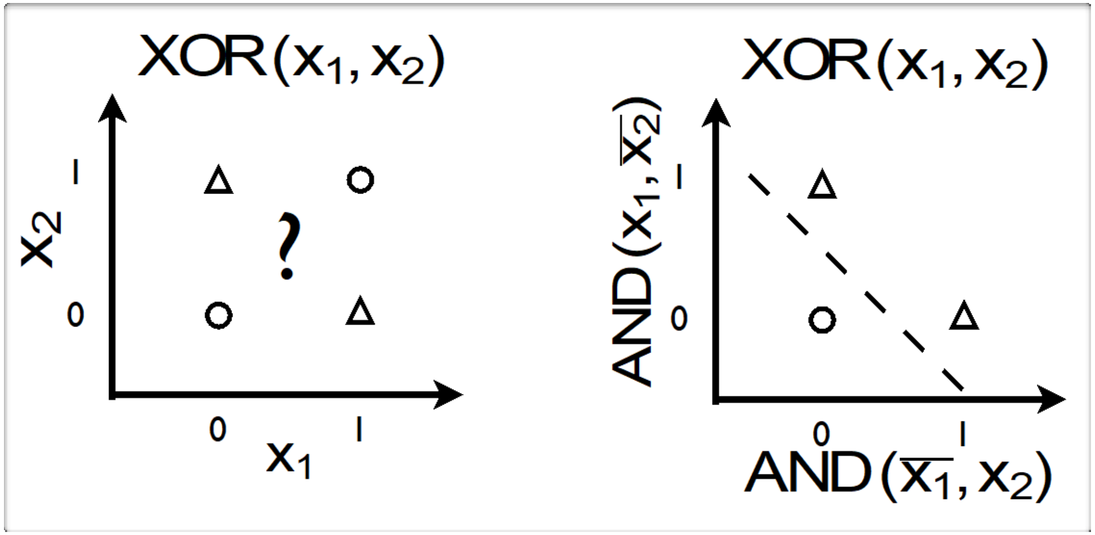



Can perfectly classify linearly separable datasets, e.g. OR, AND,

# 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*)

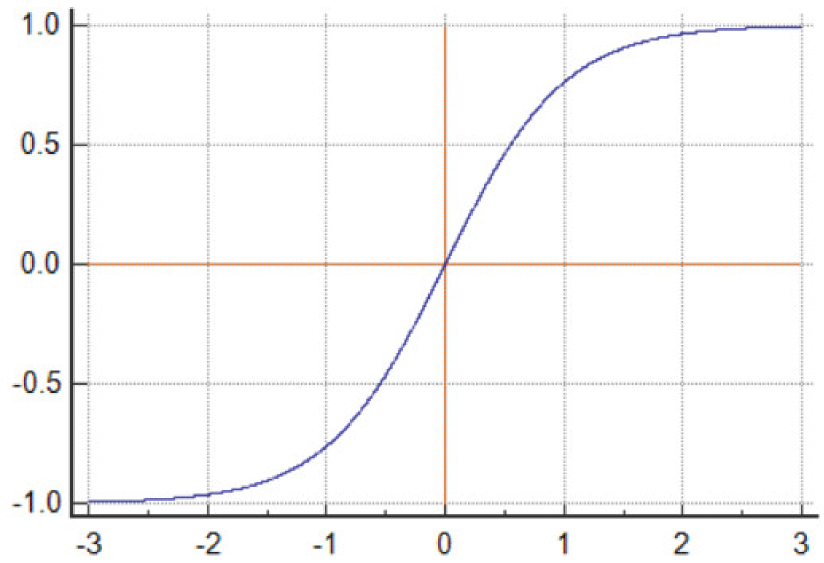
# Popular activations

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

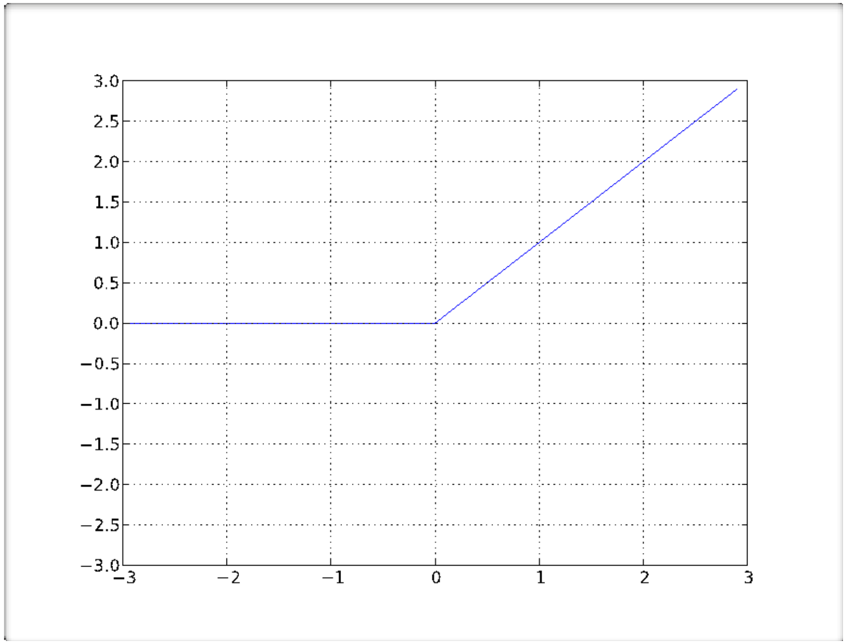


# Popular activations

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**:

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

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

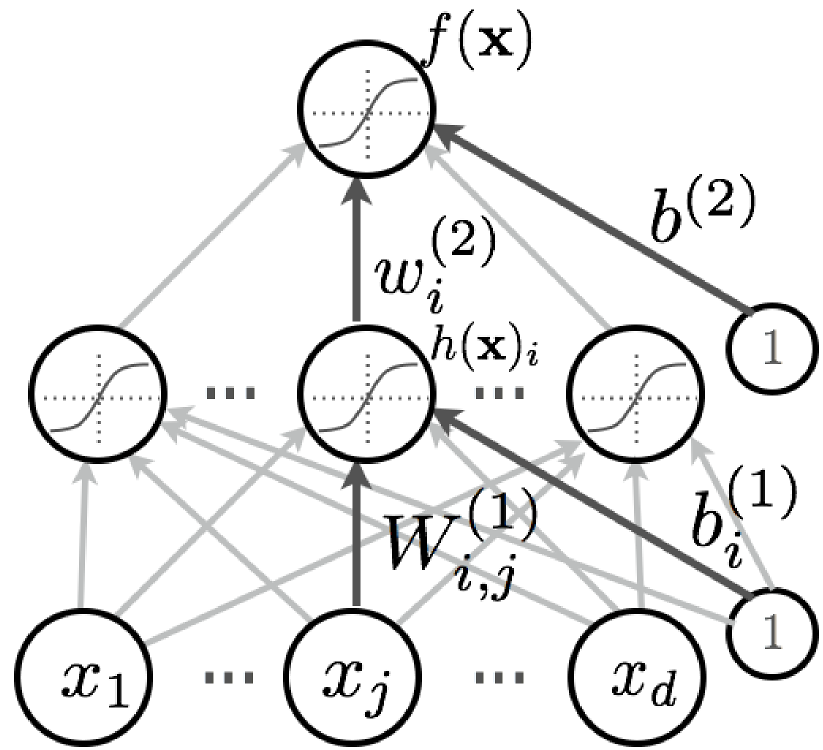
Hidden layer **post-activation**:

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

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

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

- 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**:

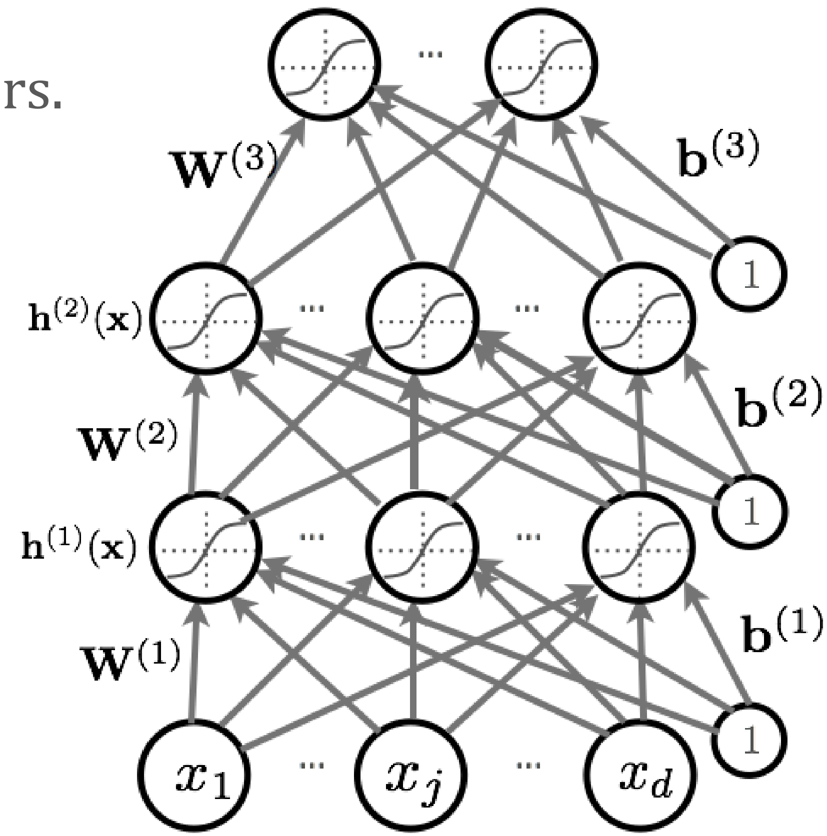
$$\boldsymbol{a}^{(k)}(\boldsymbol{x}) = \boldsymbol{b}^{(k)} + \boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)}(\boldsymbol{x})$$

Hidden layer post-activations:

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

Output layer activation:

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




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

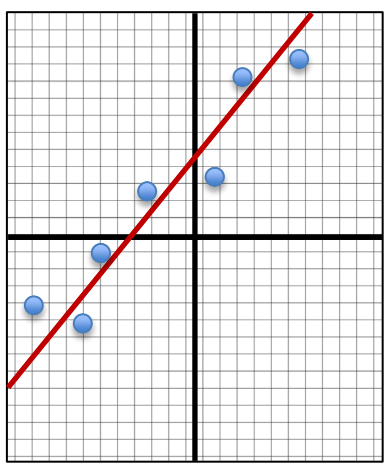
# Loss functions

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

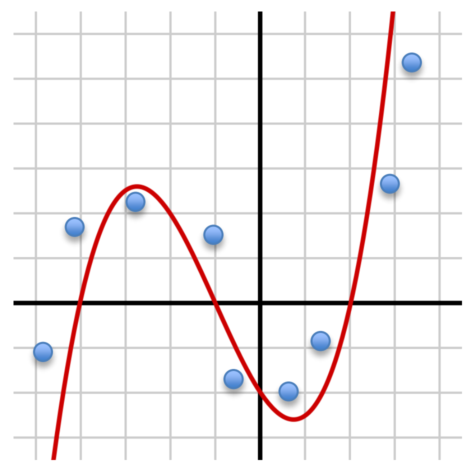
$$\hat{f} = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{(x,y):\text{training samples}} l(f(x), y)$$

**Common losses**:

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



$\boldsymbol{f(x) = \langle w, x \rangle}$



$\boldsymbol{f(x) = \sum_i a_i x^i}$

# Loss functions

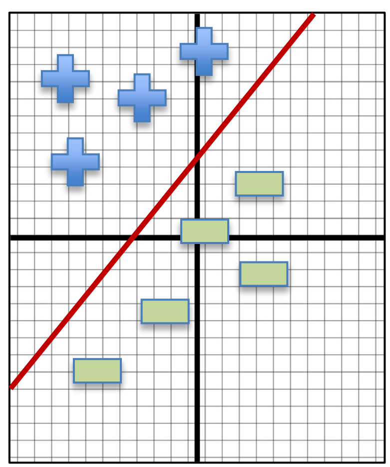
Recall: typical approach is to minimize a training loss $l$ over predictors $\mathcal{F}$:

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

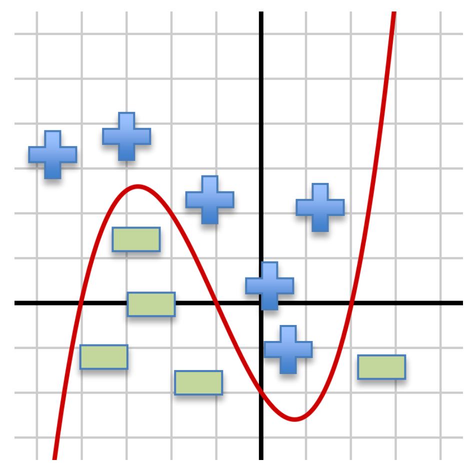
**Common losses**:

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

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



$\boldsymbol{f(x) = sgn(\langle w, x \rangle)}$



$\boldsymbol{f(x) = sgn(\sum_i a_i x^i)}$

# 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 $\mathcal{D}$:

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

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***

# Supervised learning

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

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

**Three pillars:**

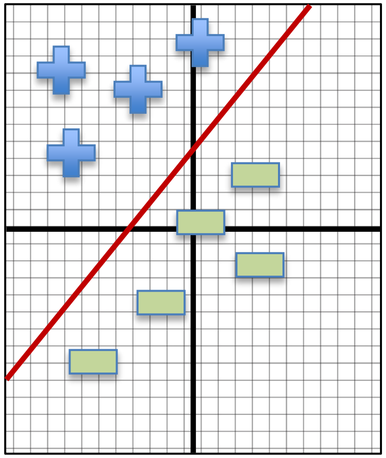
(1) How expressive is the class $\mathcal{F}$? (**Representational power**)

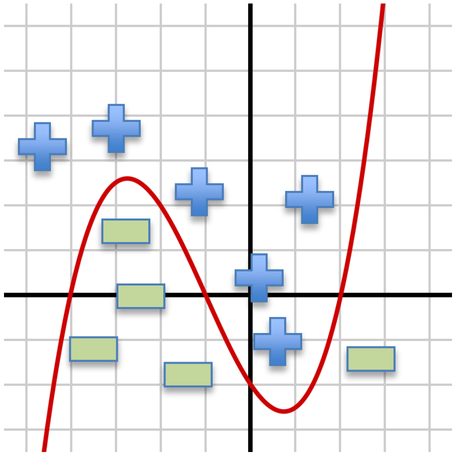
(2) How do we minimize the training loss efficiently? (**Optimization**)

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

# Expressivity

What do we mean by expressivity?

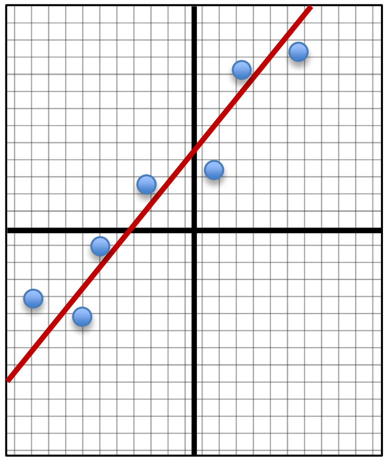




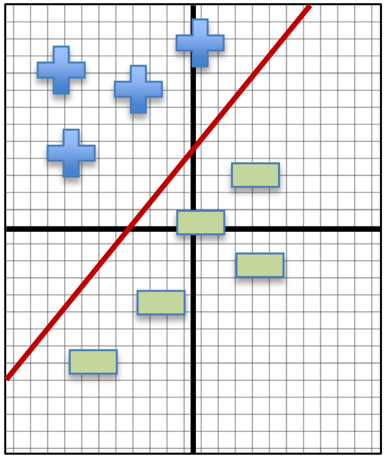
Expressive = functions in class can represent "complicated" functions

# Linear classification

The arguably simplest class of  classifiers is **linear:**



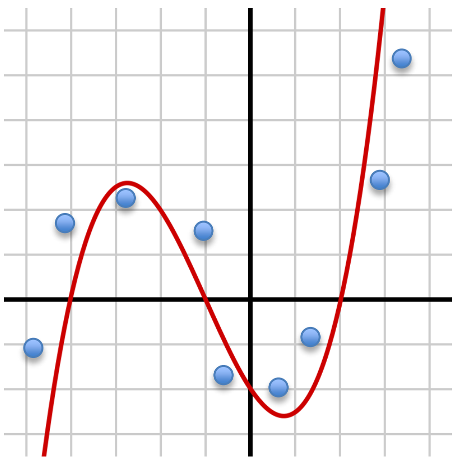
$f(x) = \langle w, x \rangle$



$f(x) = sgn(\langle w, x \rangle)$

# How do we make classifiers "more expressive"?

One pervasive idea in machine learning (from kernels onward): train a linear classifier on a **feature embedding** of data.



$$f(x) = \sum_{i=0}^{k} a_i x^i$$

For instance, we can write

$\boldsymbol{f(x) = \langle a, \phi(x) \rangle}$, where

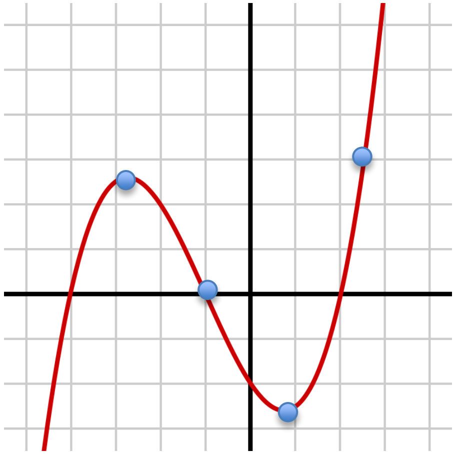
$$\boldsymbol{a = (a_1, a_2, \dots, a_k)^T, \phi(x) = (1, x, x^2, \dots, x^k)}$$

Hence, we first embed $x$ via $\boldsymbol{\phi}$ from $\mathbb{R}$ into $\mathbb{R}^k$, and train a linear classifier on these new features.

# How do we make classifiers "more expressive"?

By increasing degree we can increase expressiveness *a lot*:

For finite set of points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$ by **Lagrange's interpolation theorem**, we can find a polynomial $p$ of degree $d-1$, s.t. $\forall i, y_i = f(x_i)$



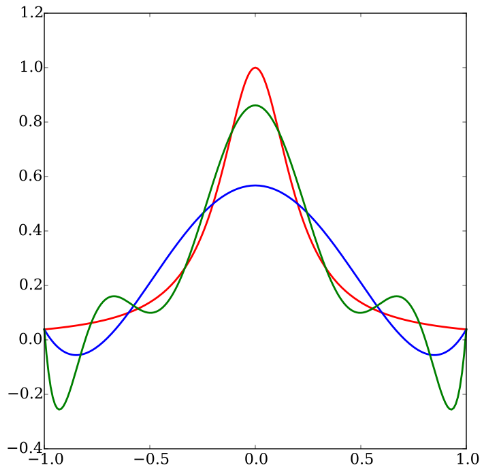
# How do we make classifiers "more expressive"?

By increasing degree we can increase expressiveness *a lot*:

For any function f, we can **approximate it** on any compact set Ω by a sufficiently high degree polynomial: for every $\epsilon > 0, \exists p$ of sufficiently high degree, s.t.

$$\max_{x \in \Omega} |f(x) - p(x)| \leq \epsilon$$

**(Stone-Weierstrass)**




Vague intuition: think of Taylor series; near point $x_0$, we have

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + \cdots$$

$$\boldsymbol{f(x) = \langle w, \phi(x) \rangle, \qquad \phi(x) = \left(1, x, x^2, \ldots, \right), w = \left(f(x_0), f'(x_0), \ldots.\right)}$$

# Lots of choices!

The name of the game in **kernel methods** was choose a good embedding $\boldsymbol{\phi}$ we explored. Lots of latitude here:

*Polynomial kernel* (in d dim.):

$$\boldsymbol{\phi}(\boldsymbol{x}) = (\mathbf{1}, \boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_d, \boldsymbol{x}_1^2, \boldsymbol{x}_1 \boldsymbol{x}_2, \dots, \boldsymbol{x}_d^2, \dots, \boldsymbol{x}_d^k)$$

*Gaussian kernel* (in 1d.):

$$\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{e}^{-\frac{x^2}{2\sigma^2}} \left( \mathbf{1}, \sqrt{\frac{\mathbf{1}}{\mathbf{1}!}} \frac{\boldsymbol{x}}{\boldsymbol{\sigma}}, \sqrt{\frac{\mathbf{1}}{\mathbf{2}!}} \sqrt{\frac{\mathbf{1}}{\mathbf{1}!}} \left( \frac{\boldsymbol{x}}{\boldsymbol{\sigma}} \right)^{\mathbf{2}}, \dots \right)$$

Choices of these kernels is closely related to something called the “kernel trick”, which allows for cheap computation of the **kernel** $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y}) \rangle$. Beyond the scope of this course!

# Lots of choices!

The name of the game in **kernel methods** was choose a good embedding $\boldsymbol{\phi}$ we explored. Lots of latitude here:

*Polynomial kernel* (in d dim.):

$$\boldsymbol{\phi}(\boldsymbol{x}) = (\mathbf{1}, \boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_d, \boldsymbol{x}_1^2, \boldsymbol{x}_1\boldsymbol{x}_2, \dots, \boldsymbol{x}_d^2, \dots, \boldsymbol{x}_d^k)$$

*Gaussian kernel* (in 1d.):

$$\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{e}^{-\frac{\boldsymbol{x}^2}{2\boldsymbol{\sigma}^2}} \left( \mathbf{1}, \sqrt{\frac{\mathbf{1}}{\mathbf{1}!}} \frac{\boldsymbol{x}}{\boldsymbol{\sigma}}, \sqrt{\frac{\mathbf{1}}{\mathbf{2}!}} \sqrt{\frac{\mathbf{1}}{\mathbf{1}!}} \left(\frac{\boldsymbol{x}}{\boldsymbol{\sigma}}\right)^2, \dots \right)$$

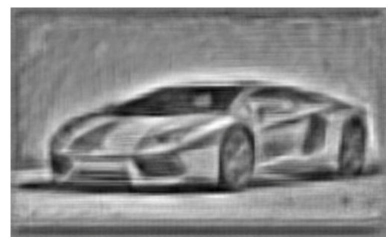
Choices of these kernels is closely related to something called the “kernel trick”, which allows for cheap computation of the **kernel** $\langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y}) \rangle$. Beyond the scope of this course!

# Part of the deep learning story

Data → Feature extraction → Learning algorithm

Object detection

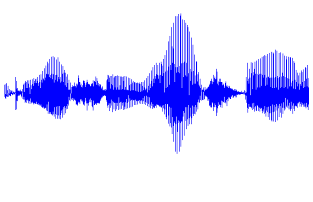
Image



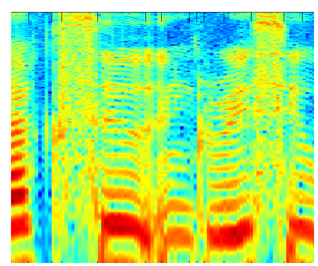
vision features

Recognition

Audio classification



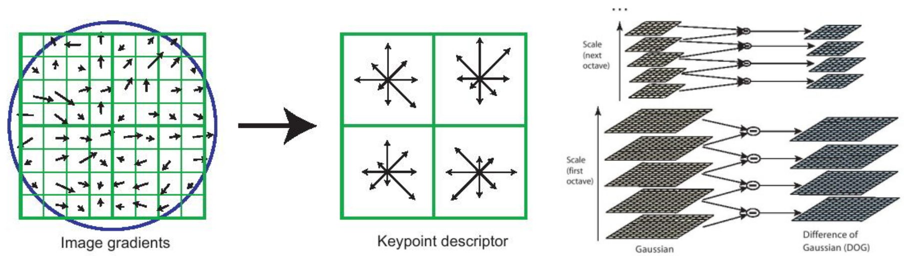
Audio



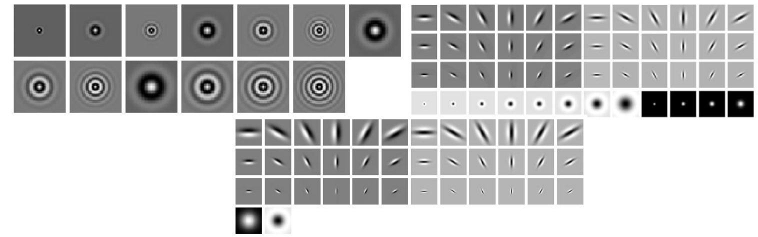
audio features

Speaker identification

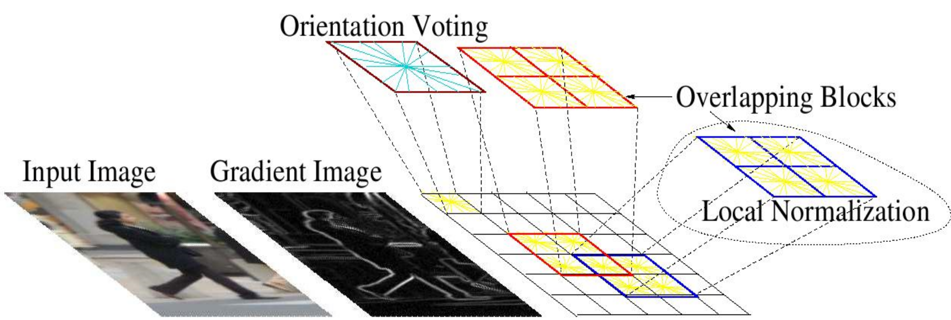
# Old school: hand-craft features



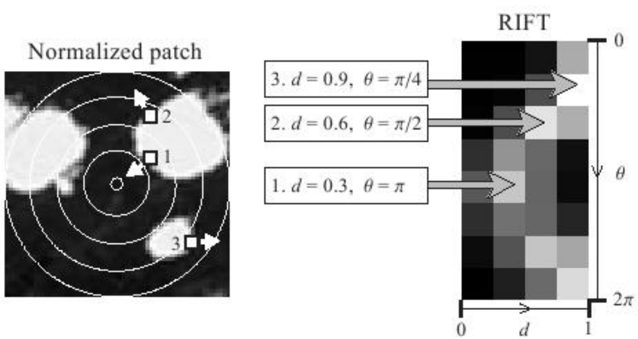

SIFT



Textons

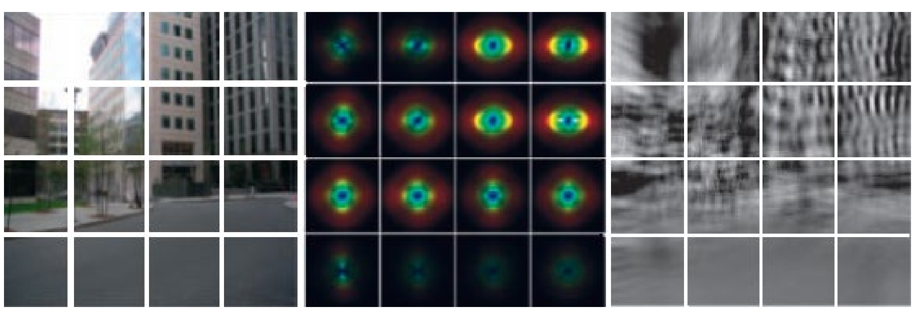



HoG

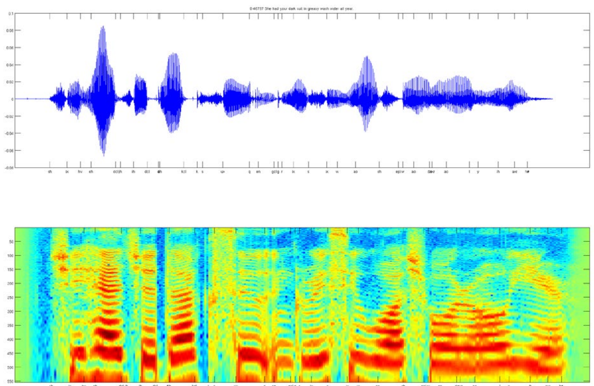



RIFT

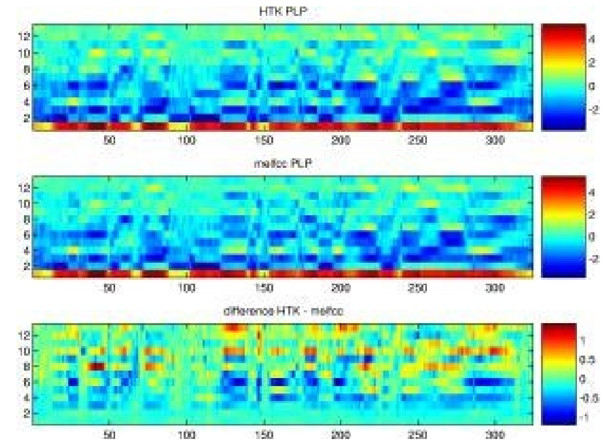
GIST



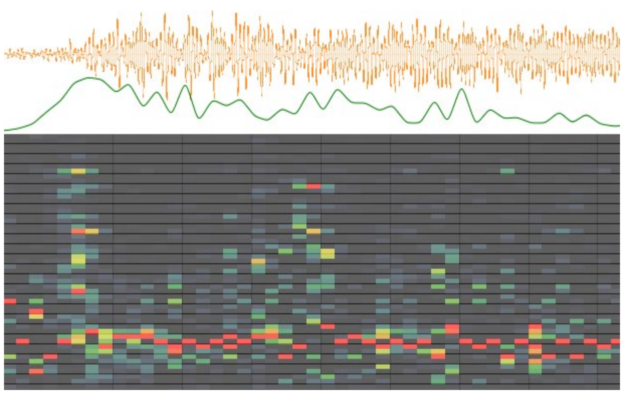
# Old school: hand-craft features



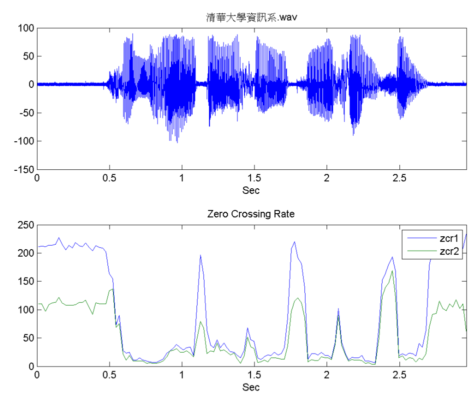
Spectrogram



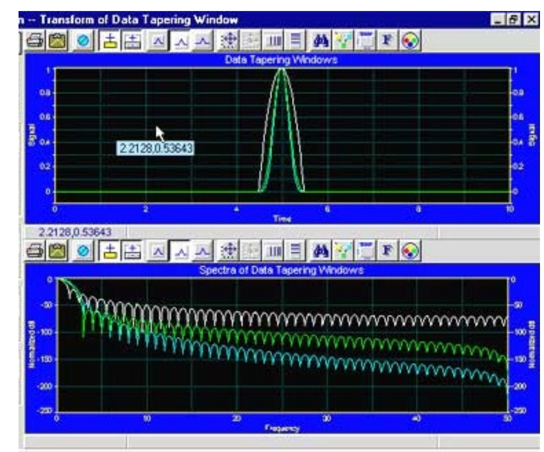

MFCC



Flux

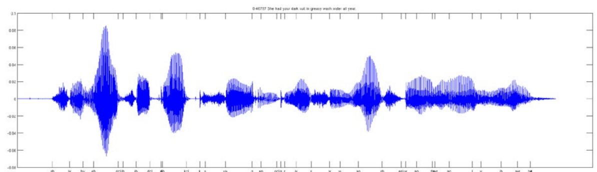


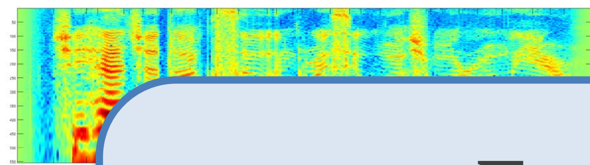

ZCR

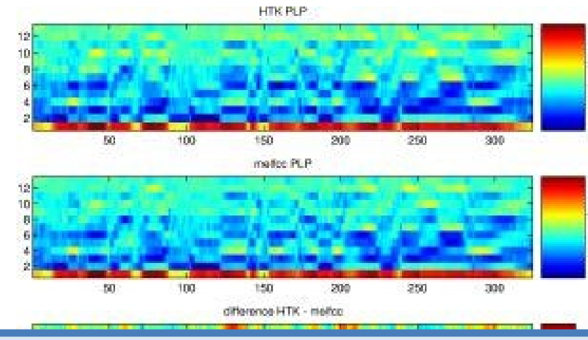



Rolloff

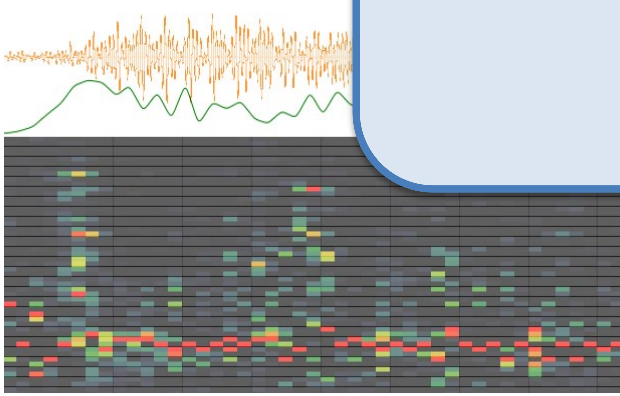
# Old school: hand-craft features



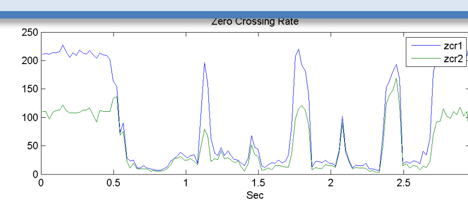




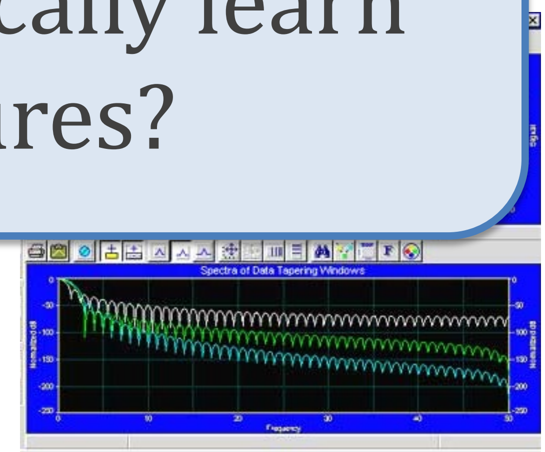

**Feature learning**:
Can we automatically learn useful features?



Flux

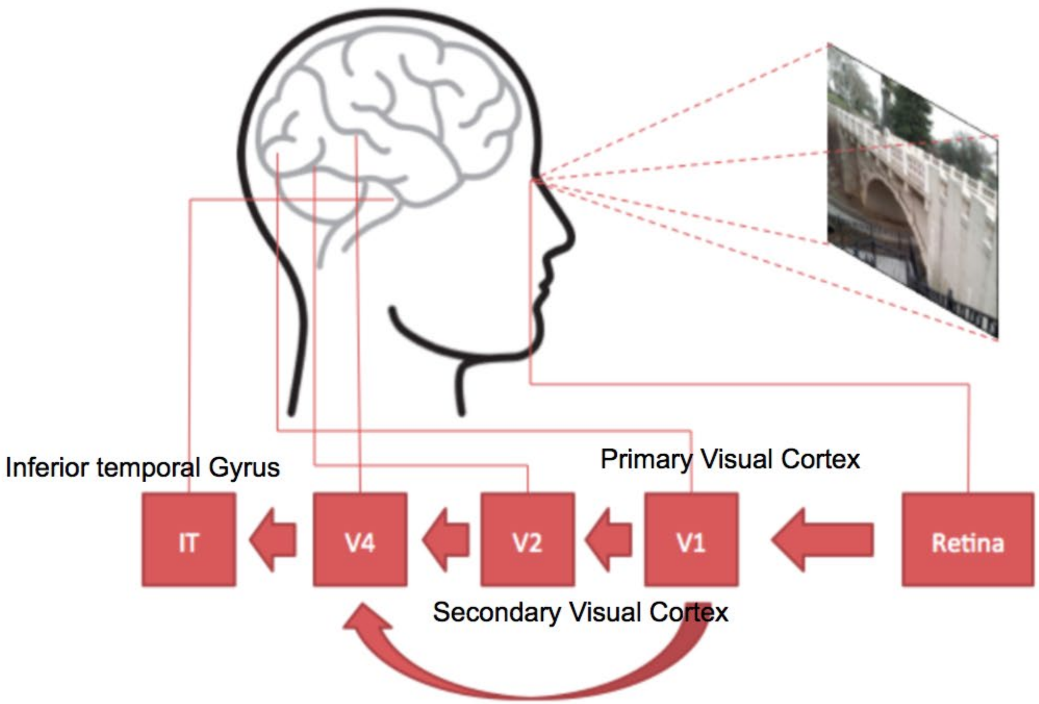



ZCR




Rolloff

# Early inspirations from visual cortex




V1: Edge detection, etc.

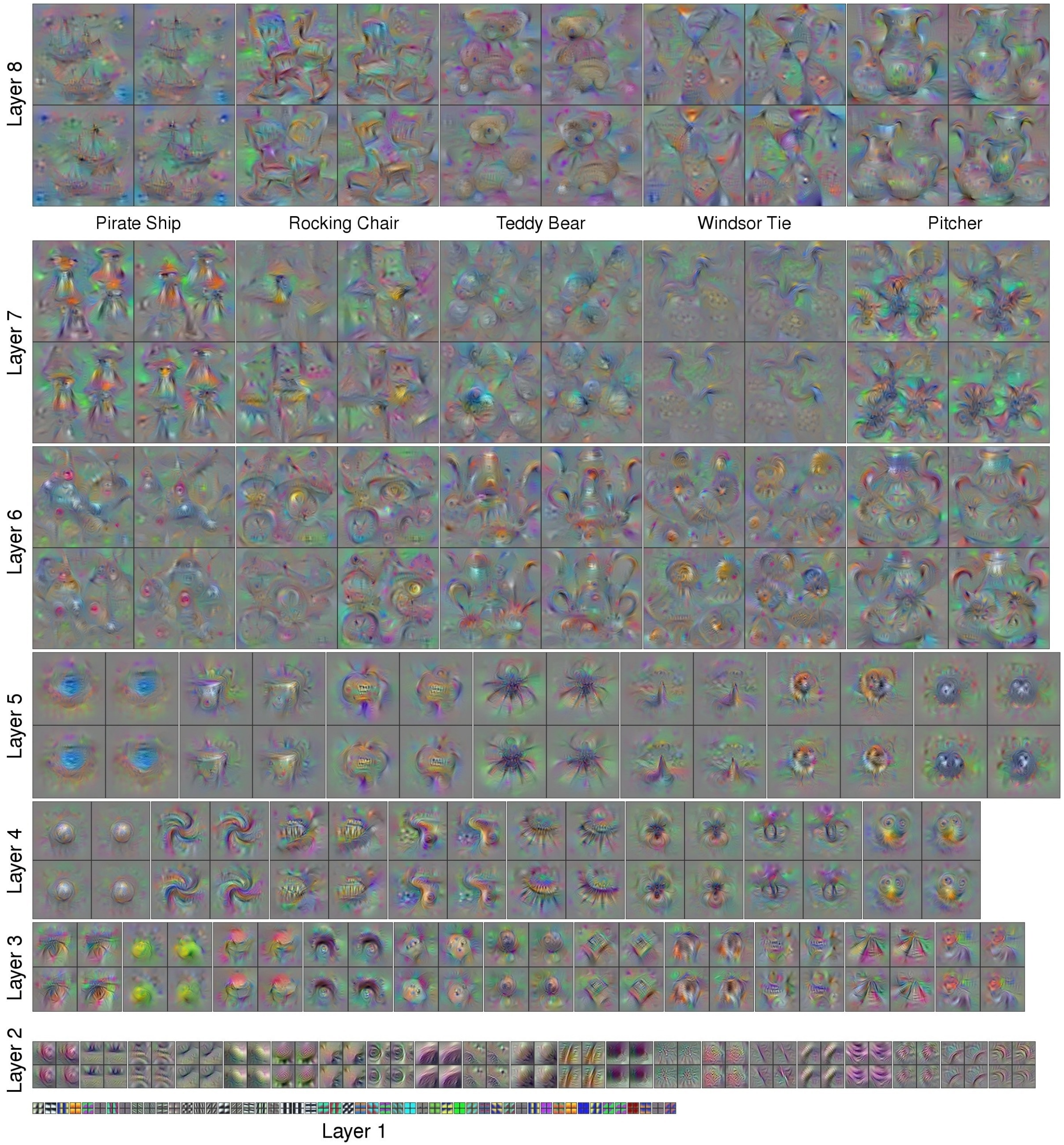
V2: Extract simple visual properties (orientation, spatial frequency, color, etc)

V4: Detect object features of intermediate complexity

TI: Object recognition.

*Image: Wang, Raj [“On the Origin of Deep Learning.”]*

# What do deep networks learn?



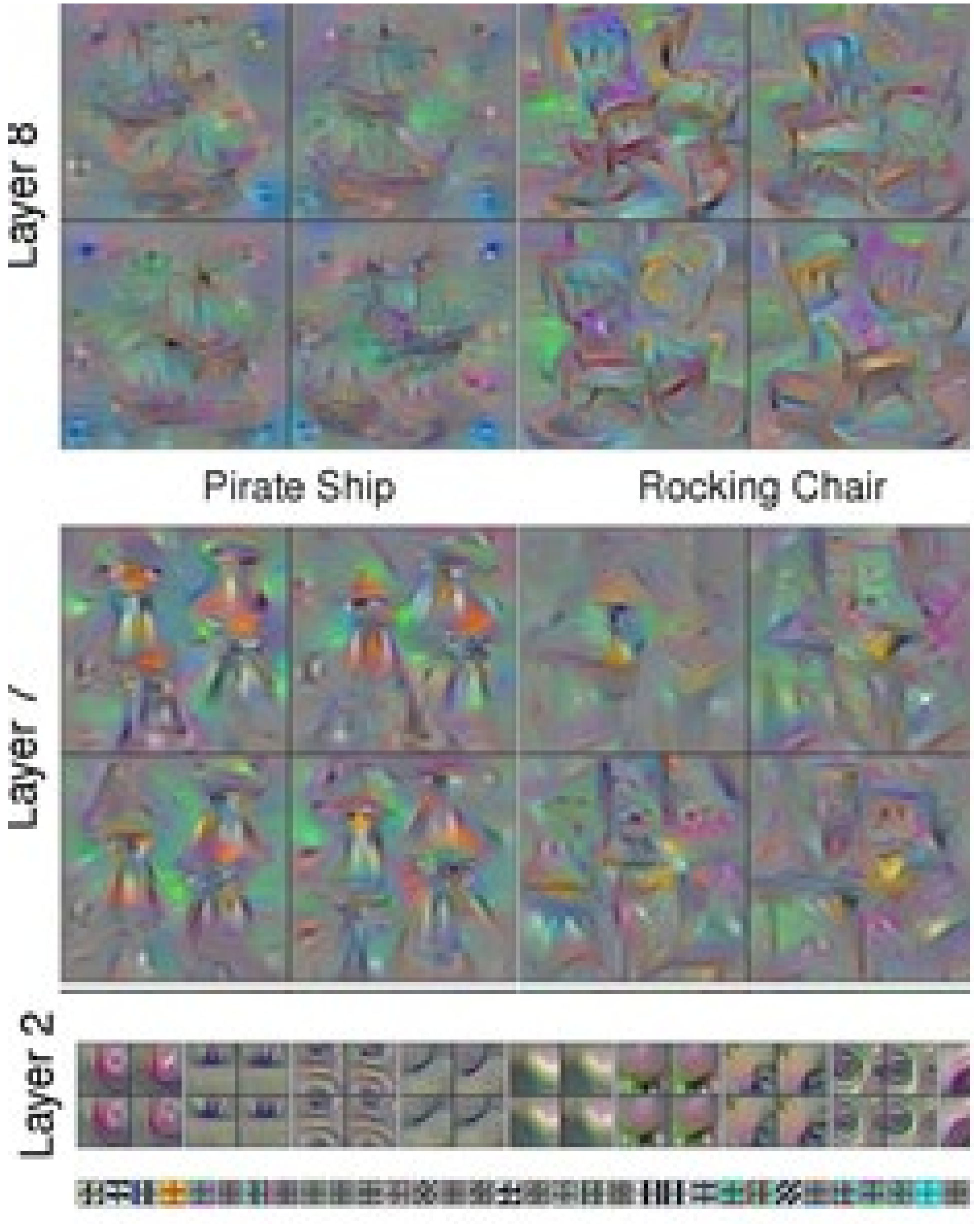

Yosinski et al ‘15:
http://yosinski.com/deepvis

**Q**: What “patterns” do neurons respond to?

**A**: From random start, do gradient descent to find an input for which neuron activation* is high.

*: This produces completely unrecognizable images – they are regularized w/ an image prior.

# What do deep networks learn?




Yosinski et al ‘15: [http://yosinski.com/deepvis](http://yosinski.com/deepvis)

**Q**: What “patterns” do neurons respond to?

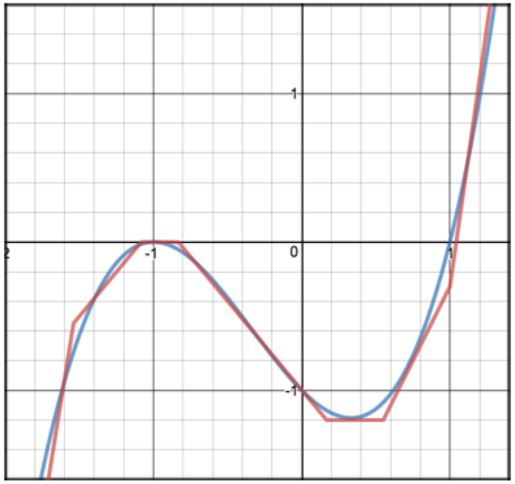
**A**: From random start, do gradient descent to find an input for which neuron activation* is high.

*: This produces completely unrecognizable images – they are regularized w/ an image prior.

# "Universal" expressivity of neural networks

(1): Neural networks are **universal approximators**: given any Lipschitz 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 $\epsilon$ error.

*"curse of dimensionality"*



# Universal approximation I: Lipschitz function are approximable

Recall, a function f: $[0,1]^d \to \mathbb{R}$ is **L-Lipschitz** (*in an $l_\infty$ sense*) if:
$\forall x, y \in [0,1]^d,\ |f(x) - f(y)| \leq L \max_{i \in [d]} |x_i - y_i|$

First, we show neural networks are **universal approximators**: given any Lipschitz function f: $[0,1]^d \to \mathbb{R}$ , a **shallow** (3-layer) neural network with ~ $\left(\frac{1}{\epsilon}\right)^{\mathrm{d}}$ neurons can approximate it to within $\epsilon$ error.

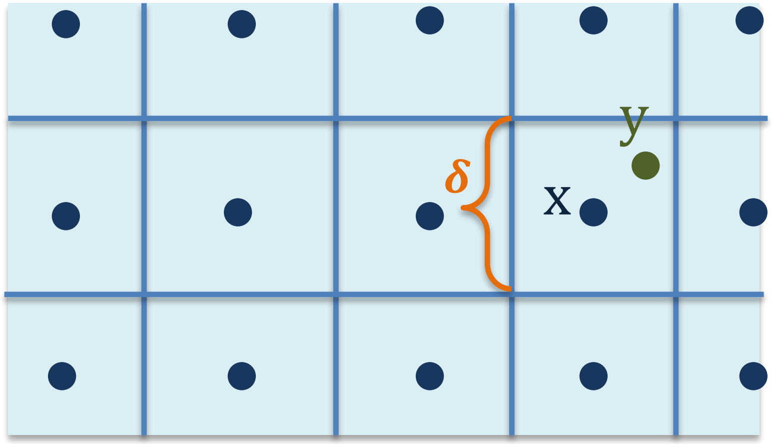
**Theorem**: For any L-Lipschitz function f: $[0,1]^d \to \mathbb{R}$, there is a 3-layer neural network $\hat{\mathrm{f}}$ with $\mathrm{O}\left(\mathrm{d}\left(\frac{\mathrm{L}}{\epsilon}\right)^{d}\right)$ ReLU neurons, s.t.

$$\int_{[0,1]^d} |f(x) - \hat{f}(x)|\, dx \leq \epsilon$$

$l_1$ *error*

# Universal approximation I: Proof intuition

**Part 1**: using Lipschitzness, we can "query" the values of function f approximately by querying its values on a fine grid.




$$|\boldsymbol{f}(\boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{x})| \leq L\delta$$

**Part 2**: we can approximate f as linear combination of "queries".

$$f(x) \approx \sum_{\text{cells } C_i} 1_{x \in C_i} f(x_i)$$

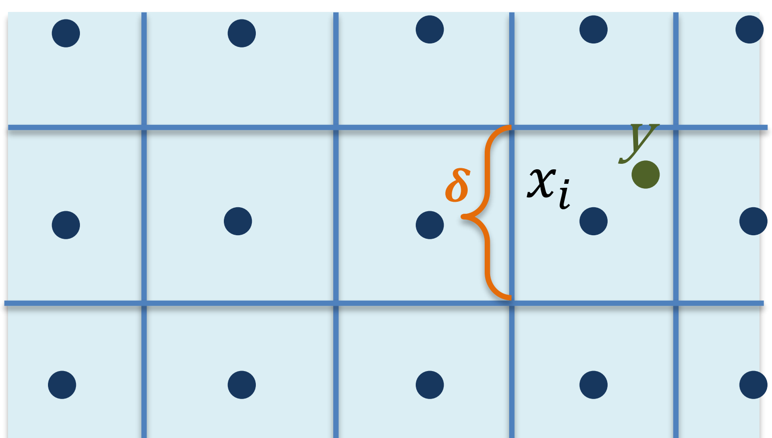
$$f(x) \approx \langle w, \phi(x) \rangle, \quad \begin{array}{c} \phi(x) = (1(x \in C_i))_i \\ w = \left(f(x_i)\right)_i \end{array}$$

**Part 3:** Approximate the indicators using ReLUs

# Universal approximation I: Part 1, formally

**Lemma**: Let f: $[0,1]^d \to \mathbb{R}$ be L-Lipschitz and $P = (C_1, C_2, \dots, C_N)$ a partition of $[0,1]^d$ into cells of side lengths at most $\delta$. Consider any set $(x_1, x_2, \dots, x_N), x_i \in C_i$ . Then:

$$\sup_{i \in N} \sup_{y \in C_i} |f(y) - f(x_i)| \leq L\delta$$




**Proof**: By Lipschitzness, we have

$$\forall i, y \in C_i{:}\ |\boldsymbol{f}(\boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{x_i})| \leq L \max_{i \in [d]} |\boldsymbol{y} - \boldsymbol{x_i}|$$
$$\leq L\delta \qquad \square$$

# Universal approximation I: Part 2, formally

**Lemma**: Let f: $[0,1]^d \to \mathbb{R}$ be 1-Lipschitz, $P = (C_1, C_2, \dots, C_N)$ a partition of $[0,1]^d$ into rectangles of side lengths at most $\delta$, and a set $(x_1, x_2, \dots, x_N), x_i \in C_i$ . Then,

$$\mathrm{g(x)} = \sum_{i=1}^{N} 1_{x \in C_i} f(x_i) \text{ satisfies } \sup_{x \in [0,1]^d} |\mathrm{f(x)} - \mathrm{g(x)}| \leq L\,\delta$$

**Proof**: Let x $\in C_i$. Then, $1_{x \in C_i} = 1$, and $1_{x \in C_j} = 0$ for $j \neq i$.

So, g(x) = $f(x_i)$.

By Lemma 1, $|\mathrm{f(x)} - \mathrm{g(x)}| = |\mathrm{f(x)} - g(\mathrm{x}_i)| \leq L\,\delta$ $\square$

# Universal approximation I: approximating indicators of cells

**Lemma**: Let $C \subseteq \mathbb{R}^d$ be a cell, namely $C = \{x\colon\ x \in [l_i, r_i], i \in d\}$. Then, there exists a 2-layer network $\tilde{\tilde{h}}(x)$ of size O(d) and ReLU activation, s.t. $\int_{x\in[0,1]^d} \left|\tilde{\tilde{h}}(x) - 1(x \in [l_i, r_i], i \in d)\right| dx \to 0$

**Proof**: First, write indicator for cell as:

*For any $\gamma > 0$, we'll take $\gamma$ small*

$$1(x \in [l_i, r_i], i \in d) = 1\left(\sum_{i=1}^{d}\big(1(x_i \geq l_i) + 1(x_i \leq r_i)\big) \geq 2d - 1 + \gamma\right)$$

**Why**? x is in cell iff all the indicators $1(x_i \geq l_i) + 1(x_i \leq r_i)$ are on.
All these indicators are on iff they sum to $2d$.
(If at least one is off, they sum to $2d$-1)

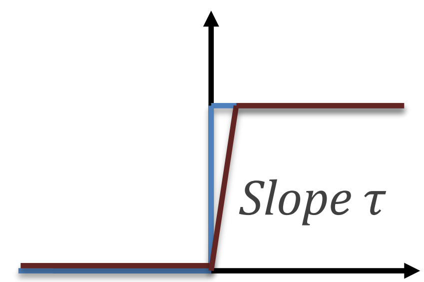
If we can approximate indicators, we're all good!

# Universal approximation I: approximating indicators of cells

**Claim**: For $\tau \geq 0,\ x \in \mathbb{R}$:

$$\left|1(x \geq 0) - \left(\sigma(\tau x) - \sigma(\tau x - 1)\right)\right| = \begin{cases} \leq 1, & \text{if } 0 \leq x \leq 1/\tau \\ 0, & \text{otherwise} \end{cases}$$

**Proof**: Consider several cases:




**Case 1**, $\boldsymbol{x \leq 0}$: $1(x \geq 0) = 0$ and $\sigma(\tau x) - \sigma(\tau x - 1) = 0$, so

$$1(x \geq 0) - \left(\sigma(\tau x) - \sigma(\tau x - 1)\right) = 0$$

**Case 2**, $\boldsymbol{x \geq 1/\tau}$: $1(x \geq 0) = 1$ and $\sigma(\tau x) - \sigma(\tau x - 1) = 1$, so

$$1(x \geq 0) - \left(\sigma(\tau x) - \sigma(\tau x - 1)\right) = 0$$

**Case 3**, $\boldsymbol{0 \leq x \leq 1/\tau}$: $1(x \geq 0) = 1$ and $\sigma(\tau x) - \sigma(\tau x - 1) = \tau x$, so

$$\left|1(x \geq 0) - \left(\sigma(\tau x) - \sigma(\tau x - 1)\right)\right| = \tau x \leq 1$$

# Universal approximation I: approximating indicators of cells

$$h(x) := 1\left(\sum_{i=1}^{d}\left(1(x_i \geq l_i) + 1(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$

Replace all indicators by difference of ReLUs. *What is the error?*

For brevity, let $\tilde{1}(x \geq 0) = \sigma(\tau x) - \sigma(\tau x - 1)$, for some $\tau$ we will choose.

Let $\tilde{h}(x) \coloneqq 1\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \geq l_i) + \tilde{1}(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$

$\tilde{\tilde{h}}(x) \coloneqq \tilde{1}\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \geq l_i) + \tilde{1}(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$

(Change the approximations "iteratively".)

# Universal approximation I: approximating indicators of cells

$$h(x) := 1\left(\sum_{i=1}^{d}\left(1(x_i \geq l_i) + 1(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$
$$\tilde{h}(x) := 1\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \geq l_i) + \tilde{1}(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$
$$\tilde{\tilde{h}}(x) := \tilde{1}\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \geq l_i) + \tilde{1}(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$

We have:

$$\int_{x\in[0,1]} \left|\tilde{\tilde{h}}(x) - h(x)\right| dx = \int_{x\in[0,1]^d} \left|\tilde{\tilde{h}}(x) - \tilde{h}(x) + \tilde{h}(x) - h(x)\right| dx$$

$$\leq \int_{x\in[0,1]^d} \left(\left|\tilde{\tilde{h}}(x) - \tilde{h}(x)\right| + \left|\tilde{h}(x) - h(x)\right|\right) dx$$

*Triangle inequality*

Let's handle two terms one by one.

# Universal approximation I: approximating indicators of cells

$$\tilde{h}(x) := 1\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \ge l_i) + \tilde{1}(x_i \le r_i)\right) \ge 2d - 1 + \gamma\right)$$
$$\tilde{\tilde{h}}(x) := \tilde{1}\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \ge l_i) + \tilde{1}(x_i \le r_i)\right) \ge 2d - 1 + \gamma\right)$$

**Claim**: $\left|1(x \ge 0) - \left(\sigma(\tau x) - \sigma(\tau x - 1)\right)\right| = \begin{cases} \le 1, & \text{if } 0 \le x \le 1/\tau \\ 0, & \text{otherwise} \end{cases}$

**First:** $\tilde{\tilde{h}}(x) - \tilde{h}(x) \ne 0$ only if $\exists i :\ x_i \in \left(l_i, l_i + \frac{1}{\tau}\right)$ or $x_i \in \left(r_i, r_i - \frac{1}{\tau}\right)$

(If $\tilde{\tilde{h}}(x) - \tilde{h}(x) \ne 0, \sum_i \tilde{1}(x_i \ge l_i) + \tilde{1}(x_i \le r_i) - (2d - 1) \in [\gamma, \gamma + \frac{1}{\tau})$, and if condition above isn't satisfied, $\tilde{1}$ = 1, so $\sum_i \tilde{1}(x_i \ge l_i) + \tilde{1}(x_i \le r_i) - (2d - 1)$ is integer, so cannot belong to interval for small enough $\gamma > 0$.

Measure of such x's is bdd by $\sum_i\left(\int_{\substack{x \in [0,1]^d \\ x_i \in \left(l_i, l_i + \frac{1}{\tau}\right)}} 1\, dx + \int_{\substack{x \in [0,1]^d \\ x_i \in \left(r_i, r_i - \frac{1}{\tau}\right)}} 1\, dx\right) \le \frac{2\mathrm{d}}{\tau}$, so

$\int_{x \in [0,1]^d} |\tilde{\tilde{h}}(x) - \tilde{h}(x)| \le \frac{2d}{\tau}$

# Universal approximation I: approximating indicators of cells

$$h(x) := 1\left(\sum_{i=1}^d \left(1(x_i \geq l_i) + 1(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$
$$\tilde{h}(x) := 1\left(\sum_{i=1}^d \left(\tilde{1}(x_i \geq l_i) + \tilde{1}(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$

**Claim**: $\left|1(x \geq 0) - \left(\sigma(\tau x) - \sigma(\tau x - 1)\right)\right| = \begin{cases} \leq 1, & \text{if } 0 \leq x \leq 1/\tau \\ 0, & \text{otherwise} \end{cases}$

**Second**: $\displaystyle\int_{x\in[0,1]} \left|\tilde{h}(x) - h(x)\right| dx$

*Indicators are equal if inputs are equal*

$$\leq \int_{x\in[0,1]^d} \left|1\left(\sum_{i=1}^d \left(1(x \geq l_i) + 1(x \leq r_i)\right) \neq \sum_{i=1}^d \left(\tilde{1}(x \geq l_i) + \tilde{1}(x \leq r_i)\right)\right)\right| dx$$

$$\leq \int_{x\in[0,1]^d} \sum_{i=1}^d 1\left(1(x \geq l_i) \neq \tilde{1}(x \geq l_i)\right) + \sum_{i=1}^d 1\left(1(x \leq r_i) \neq \tilde{1}(x \leq r_i)\right) dx$$

*By Claim* $\leq 2d/\tau$

# Universal approximation I: approximating indicators of cells

$$h(x) := 1\left(\sum_{i=1}^{d}\left(1(x_i \geq l_i) + 1(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$
$$\tilde{h}(x) := 1\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \geq l_i) + \tilde{1}(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$
$$\tilde{\tilde{h}}(x) := \tilde{1}\left(\sum_{i=1}^{d}\left(\tilde{1}(x_i \geq l_i) + \tilde{1}(x_i \leq r_i)\right) \geq 2d - 1 + \gamma\right)$$

Putting together, we have:

$$\int_{x\in[0,1]^d} \left|\tilde{\tilde{h}}(x) - h(x)\right| dx = \int_{x\in[0,1]^d} \left|\tilde{\tilde{h}}(x) - \tilde{h}(x) + \tilde{h}(x) - h(x)\right| dx$$
$$\leq 4d/\tau$$

Also, $\tilde{\tilde{h}}(x)$ is a 2-layer net with ReLU activations and $O(d)$ nodes!

# Universal approximation I: Putting everything together

By Part 1+2, $\sup_{x\in[0,1]^d} \left|\mathrm{f(x)} - \sum_{i=1}^{N} 1_{x\in C_i}\, f(x_i)\right| \leq L\,\delta$

Moreover, the number of cells N can be bounded by $\left(\frac{1}{\delta}\right)^d$

By indicator approximation: can approximate arbitrarily well by taking $\tau \to \infty$ with a 2-layer ReLU net.

Combining the above two points, we get a $\left(\frac{1}{\delta}\right)^d$ $-$sized 3-layer net s.t.

$$\int_{[0,1]^d} |\,f(x) - \hat{f}(x)|\,dx \leq L\,\delta$$

Taking $\delta = \frac{\epsilon}{L}$, the theorem follows.

# Parting thoughts

All results we proved are **existential**: they prove that a good approximator exists. Finding one efficiently (much less so using gradient descent) is a different matter.

The choices of non-linearities are usually very **flexible**: most results of the type we saw can be re-proven using different non-linearities. (Examples in homework.)

Many other results of similar flavor. For instance, there are also results that deep, but narrow networks are universal approximators.