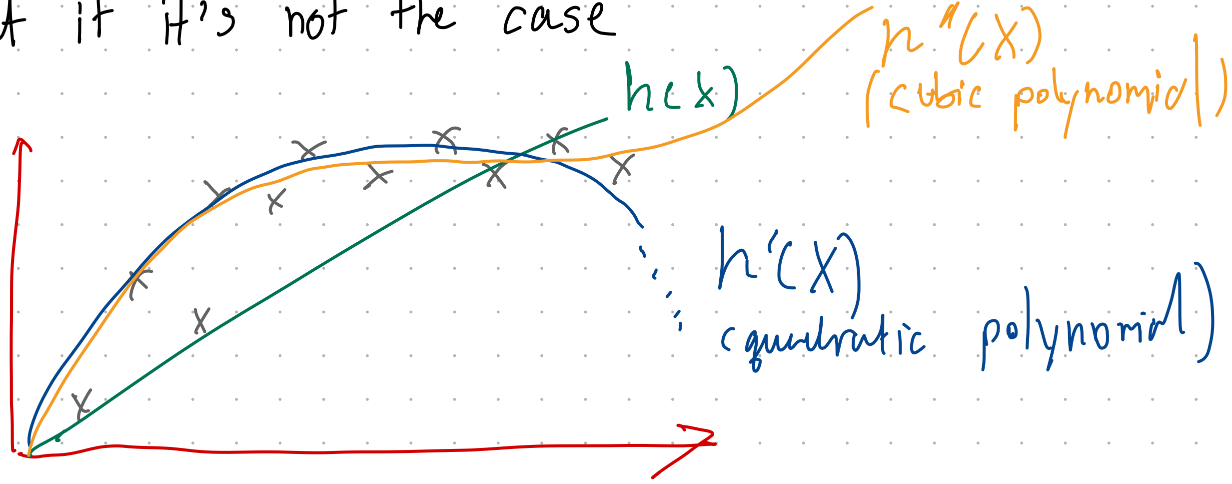
- So far, we have considered only linear decision boundary, what if it's not the case



- If we apply linear regression on the data set, we can learn on linear linear function h(x)
- To capture the non-linearity of functions, we can use feature expansion.

Feature expansion: Make linear classifiers non-linear

> IDEA: Transform $\vec{x} \longrightarrow \phi(\vec{x})$
> where $\phi(\vec{x}) \in \mathbb{R}^m$ with $m >> d$

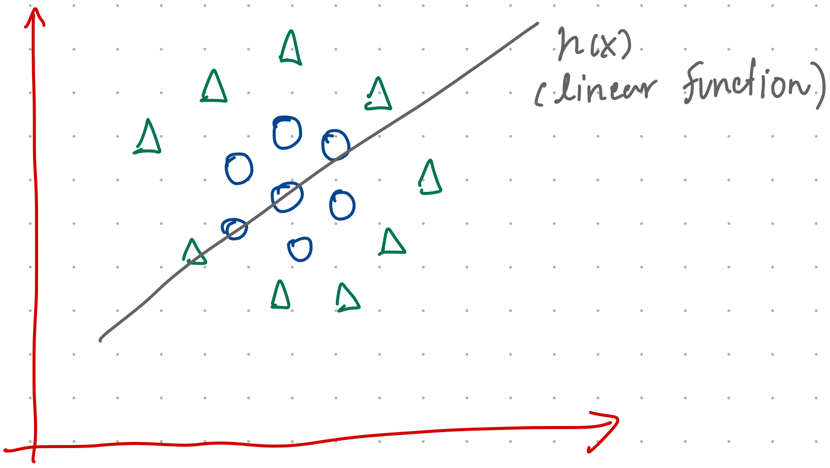
Ex 1:

$$\underbrace{\begin{pmatrix} x_1 \end{pmatrix}}_{\vec{x}} \longrightarrow \underbrace{\begin{pmatrix} 1 \\ x_1 \\ x_1^2 \end{pmatrix}}_{\phi(\vec{x})}$$

This means we learn a quadratic function

$$h''(x) = \underbrace{w_2 x_1^2 + w_1 x_1 + b}_{\text{parabola}}$$

Quiz: What is the feature expasion in this daset



Answer: Recall general form circle equation

$$x^2 + y^2 + Cx + Dy + E = 0$$

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \longrightarrow \phi(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{pmatrix}$$

So, we are learning function

$$w_4 (x_1^2 + x_2^2) + w_3 x_2 + w_4 x_1 + w_1$$

- Advantage: Simple, our problem stays convex and well behaved
- Disadvantage: $\phi(\vec{x})$ might be very high dimensional.

Extreme case:

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \longrightarrow \phi(\vec{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}$$

$\phi(\vec{x})$ causes 2^d features for its representation!!

Kernel Trick: Learning a function in the much higher dimensional space, without ever computing $\phi(x)$ or ever compute $\vec{w}$.

Sorry, I don't have enough time to teach this topic. :(

Matrix-Vector Multiplication (Linear Transformation)

$$U\vec{x} = \underbrace{\begin{bmatrix} \text{—} \; u_1 \; \text{—} \\ \text{—} \; u_2 \; \text{—} \\ \vdots \\ \text{—} \; u_m \; \text{—} \end{bmatrix}}_{m \times n} \underbrace{\begin{bmatrix} \; \\ \; \\ \; \end{bmatrix}}_{n \times 1} = \underbrace{\begin{bmatrix} \; \\ \; \\ \; \\ \; \end{bmatrix}}_{m \times 1}$$

($\sim$ Feature expansion)

Here comes the last topics

"Neural Networks / Deep Learning"

A bit of history of Neural Networks:

- Neural Networks were invented by Frank Rosenblatt in 1963.
- It was call "Multi-layer Perceptron" back then. A few years ago, people started to call it "Deep Learning".
- Perceptrons had a branding because of the AI winter.
- Caution: It is exaggerated to say Neural Networks work like our brain!!!
- Neural Networks were popular in 1980s (logic based ML vs. neural networks based ML)
- The Neural Networks based ML was not accepted, so people in Neural Networks started their own conference called Neuro Information processing Systems / NIPS
- NIPS is now #1 ML conference.
- Until 1990s, SVM came around and it got popular.
- In 2002-2003, there was no a singe Neural Networks paper anymore in NIPS.
- The rebranding of Neural Networks as "Deep Learning" happended in 2006 to avoid rejection of Neural Networks papers

History of DL: Geoff Hinton at U of Toronto bought a bunch of GPUs by the time they were expensive. He started to train neural networks and won image recognition competition.

Areas that DL works very well: Object recognition (since 2012) & speech recognition (since 2006)

Neuron Networks / Deep Learning

- From now on, let's make linear classifiers non-linear by mapping the data into a fixed high dimensional feature space.

$$\boxed{\begin{aligned} x &\longrightarrow \phi(\vec{x}) \\ h(\vec{x}) &= \phi(\vec{x})^T \vec{w} + b \end{aligned}}$$

- In Neural Networks, we try to learn $\underset{\sim}{\vec{w}}$ and $\underset{\sim}{\phi(\vec{x})}$.

KEY COMPONENTS OF NEURAL NETWORKS / DEEP LEARING

① Nonlinear transition function:

$$\boxed{\phi(\vec{x}) = \sigma(\overbrace{U\vec{x} + \vec{c}}^{\text{transformation of } \vec{x}})}$$

- $\sigma(\vec{z})$ is a non-linear transition function, which operates elementes element-wise on each dimension.

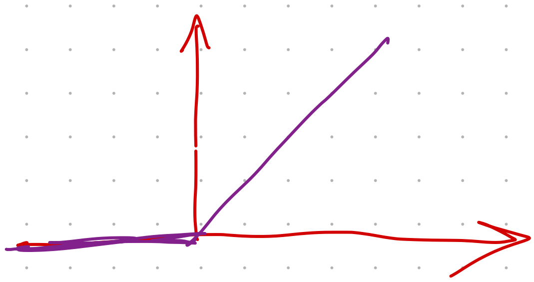
NOTE: Without transition, the classifier would just be linear

$$\vec{w}^T \phi(x) + b = w^T(U\vec{x} + \vec{c}) + b = \underline{w'^T \vec{x} + b'}, \text{ where } \begin{aligned} w' &= w^T U \\ b' &= w^T c + b \end{aligned}$$

Examples of transition functions

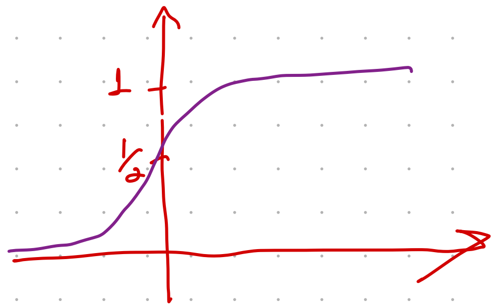
- ReLU (Rectified Linear Unit)

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



- Sigmoidal unit

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



② GPUs:

- Neural Networks have been around since 1963.
- It is quite expensive to train, and it was slow compared to SVM back then.
- The computation within Neural Networks are basically matrix multiplications.
- With GPUs, we can do matrix multiplications really well.

③ Stochastic Gradient Descent (SGD):

- Terrible algorithm, but it works !!

④ Rebranding: Neural Networks → Deep learning

Example (ReLU): Consider a regression problem

$$h(\vec{x}) = w^T \phi(x) + b \qquad \phi(x) = \sigma(U\vec{x} + \vec{c})$$

$$l = \sum_{i=1}^{n} l(h(\vec{x}_i), y_i) = \sum_{i=1}^{n} (h(\vec{x}_i) - y_i)^2$$

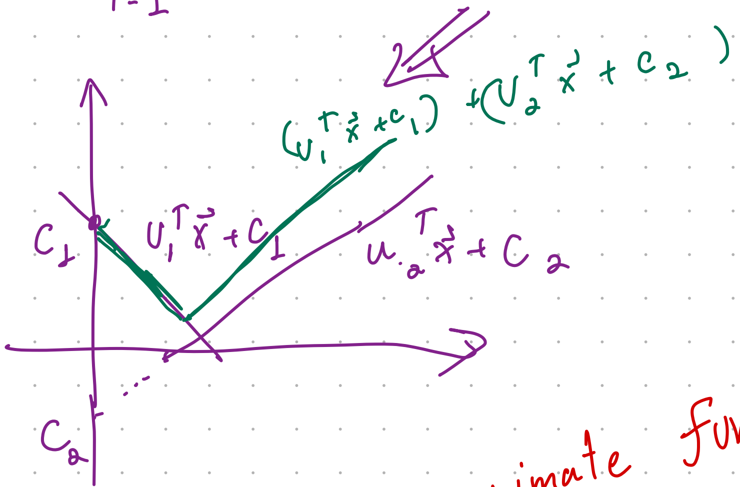
$$U = \begin{array}{c} d \\ \begin{bmatrix} \text{---} u_1 \text{---} \\ \text{---} u_2 \text{---} \\ \vdots \\ \text{---} u_m \text{---} \end{bmatrix} \end{array} \Big\} m \qquad \vec{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_d \end{pmatrix}$$

$$U\vec{x} = \begin{bmatrix} u_1^T \vec{x} \\ \vdots \\ u_n^T \vec{x} \end{bmatrix}$$

$$h(\vec{x}) = w^T \sigma(U\vec{x} + \vec{c}) + b$$

$$= w^T\left(\max(U\vec{x} + \vec{c}, \vec{0})\right) + b$$

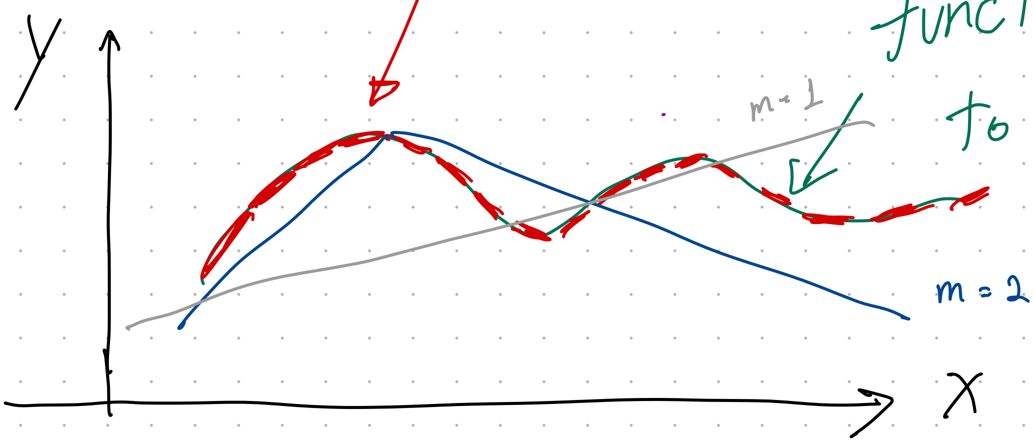
$$= \sum_{i=1} w_i \cdot \underbrace{\max(U_i^T \vec{x} + c_i, 0)} + b$$



Approximate function by piecewise linear combination

$m \geq 1$

function we try to learn



- Layers in Neural Networks:

$$h(x) = w^T \phi(\vec{x})$$

$$\cancel{\phi(\vec{x}) = \sigma(U\vec{x} + \vec{c})}$$ — one layer neural networks

$$\Downarrow$$

more than one layer ("Deep" Learning)

$$\begin{cases} \phi(\vec{x}) = \sigma(\underline{U\phi'(\vec{x})} + \underline{\vec{c}}) \\ \phi'(\vec{x}) = \sigma(\underline{U'\phi''(\vec{x})} + \vec{c}') \\ \phi''(\vec{x}) = \sigma(\underline{U'\phi'''(\vec{x})} + \vec{c}'') \\ \vdots \end{cases}$$

Matrix multiplication

Theory of Neural Networks: Any function can be learned with deep learning, can also be leared with one-layer neural networks.

**** For one layer, the matrix U has to be exponentially wide, With multiple layers, we have exponential effects from multiplying the number of possible lines.