

# Feature Learning

He He

CDS, NYU

April 20, 2021

# Today's lecture

- Neural networks: huge empirical success but poor theoretical understanding
- Key idea: representation learning
- Optimization: backpropagation + SGD

# Overview

- Learning non-linear models in a linear form:

$$f(x) = w^T \phi(x). \quad (1)$$

- What are possible  $\phi$ 's we have seen?
  - Feature maps that define a kernel, e.g., polynomials of  $x$
  - Feature templates, e.g.,  $x_i$  AND  $x_{i-1}$
  - Basis functions, e.g., (shallow) decision trees

# Decompose the problem

- Example:

Task Predict popularity of restaurants.

Raw features #dishes, price, wine option, zip code, #seats, size

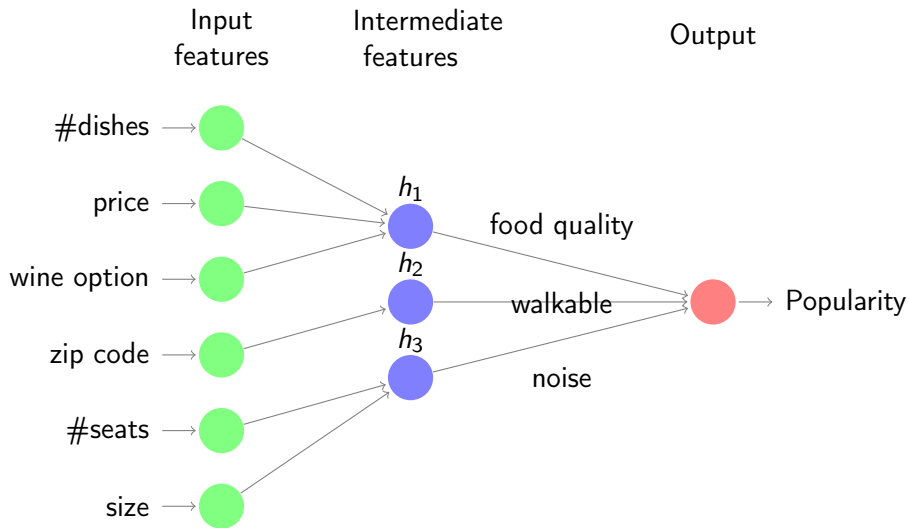
- Decompose into subproblems:

- $h_1$ ([#dishes, price, wine option]) = food quality
- $h_2$ ([zip code]) = walkable
- $h_3$ ([#seats, size]) = nosie

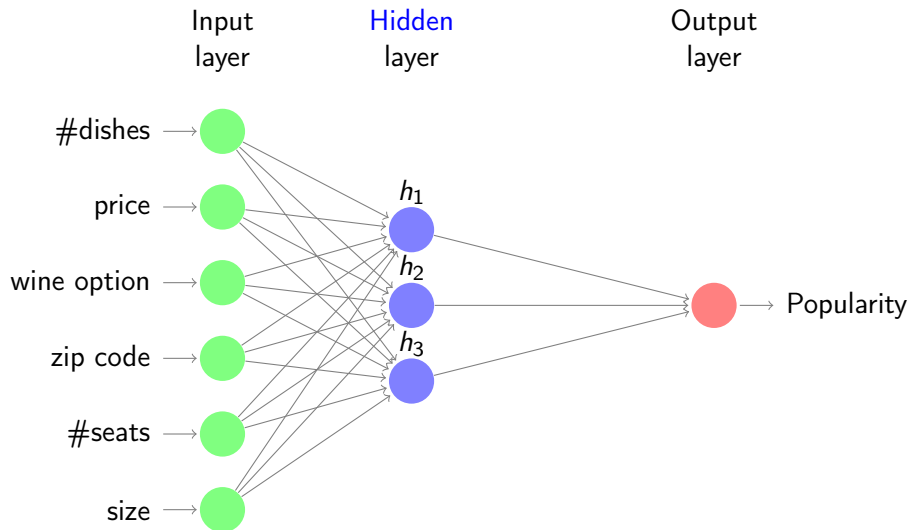
- Final *linear* predictor uses **intermediate features** computed by  $h_i$ 's:

$$w_1 \cdot \text{food quality} + w_2 \cdot \text{walkable} + w_3 \cdot \text{nosie}$$

# Predefined subproblems



## Learned intermediate features



**Key idea:** automatically learn the intermediate features.

**Feature engineering** Manually specify  $\phi(x)$  based on domain knowledge and learn the weights:

$$f(x) = \mathbf{w}^T \phi(x). \quad (2)$$

**Feature learning** Automatically learn both the features ( $K$  hidden units) and the weights:

$$h(x) = [\mathbf{h}_1(x), \dots, \mathbf{h}_K(x)], \quad (3)$$

$$f(x) = \mathbf{w}^T h(x) \quad (4)$$



# Activation function

- How should we parametrize  $h_i$ 's? Can it be linear?

$$h_i(x) = \sigma(v_i^T x). \quad (5)$$

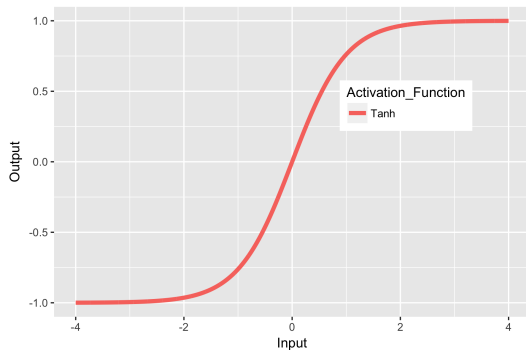
- $\sigma$  is the **nonlinear** activation function.
- What might be some activation functions we want to use?
  - sign function? **Non-differentiable**.
  - *Differentiable* approximations: sigmoid functions. *s-shaped*
    - E.g., logistic function, hyperbolic tangent function.
- Two-layer neural network (one **hidden layer** and one **output layer**) with  $K$  hidden units:

$$f(x) = \sum_{k=1}^K w_k h_k(x) = \sum_{k=1}^K w_k \sigma(v_k^T x) \quad (6)$$

# Activation Functions

- The **hyperbolic tangent** is a common activation function:

$$\sigma(x) = \tanh(x).$$

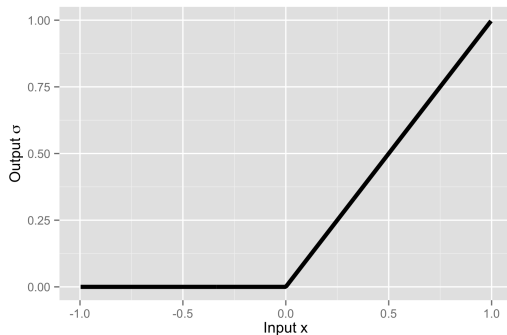


# Activation Functions

- More recently, the **rectified linear (ReLU)** function has been very popular:

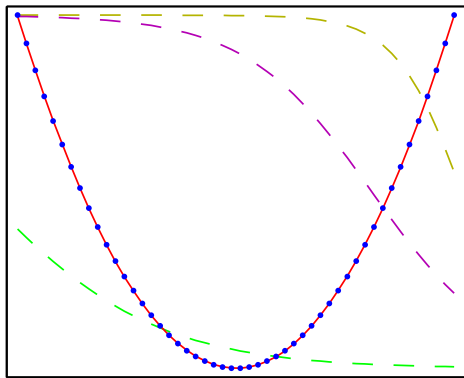
$$\sigma(x) = \max(0, x).$$

- Much **faster** to calculate, and to calculate its derivatives.
- Also often seems to work better.



## Approximation Ability: $f(x) = x^2$

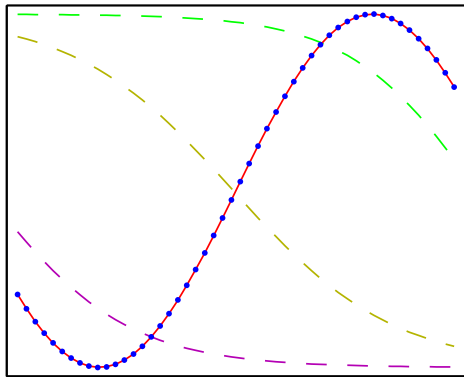
- 3 hidden units; tanh activation functions
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.



From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3

## Approximation Ability: $f(x) = \sin(x)$

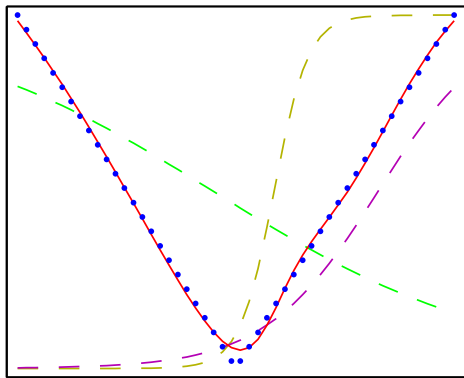
- 3 hidden units; logistic activation function
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.



From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3

## Approximation Ability: $f(x) = |x|$

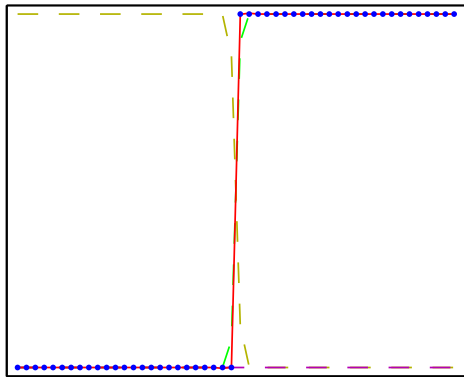
- 3 hidden units; logistic activation functions
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.



From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3

## Approximation Ability: $f(x) = 1(x > 0)$

- 3 hidden units; logistic activation function
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.



From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3

# Universal approximation theorems

How much expressive power do we gain from the nonlinearity?

## Theorem (Universal approximation theorem)

A neural network with one *possibly huge hidden layer*  $\hat{F}(x)$  can approximate any continuous function  $F(x)$  on a closed and bounded subset of  $\mathbb{R}^d$  under mild assumptions on the activation function, i.e.  $\forall \epsilon > 0$ , there exists an integer  $N$  s.t.

$$\hat{F}(x) = \sum_{i=1}^N w_i \sigma(v_i^T x + b_i) \quad (7)$$

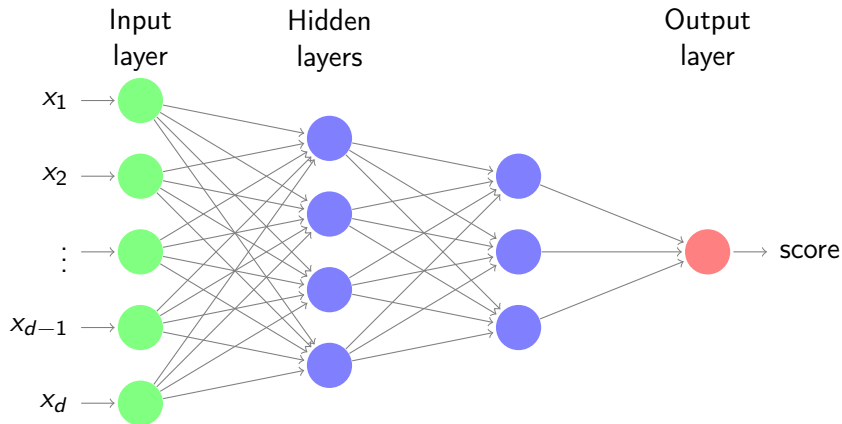
satisfies  $|\hat{F}(x) - F(x)| < \epsilon$ .

- Number of hidden units needs to be exponential in  $d$ .
- Doesn't say how to learn these parameters.



# Multilayer perceptron / Feed-forward neural networks

- Wider: more hidden units.
- Deeper: more hidden layers.



# Multilayer Perceptron: Standard Recipe

- **Input space:**  $\mathcal{X} = \mathbb{R}^d$       **Action space**  $\mathcal{A} = \mathbb{R}^k$  (for  $k$ -class classification).
- Let  $\sigma: \mathbb{R} \rightarrow \mathbb{R}$  be an activation function (e.g. tanh or ReLU).
- Let's consider an MLP of  $L$  hidden layers, each having  $m$  hidden units.
- First hidden layer is given by

$$h^{(1)}(x) = \sigma \left( W^{(1)}x + b^{(1)} \right),$$

for parameters  $W^{(1)} \in \mathbb{R}^{m \times d}$  and  $b \in \mathbb{R}^m$ , and where  $\sigma(\cdot)$  is applied to each entry of its argument.

# Multilayer Perceptron: Standard Recipe

- Each subsequent hidden layer takes the *output*  $o \in \mathbb{R}^m$  of *previous layer* and produces

$$h^{(j)}(o^{(j-1)}) = \sigma\left(W^{(j)} o^{(j-1)} + b^{(j)}\right), \text{ for } j = 2, \dots, L$$

where  $W^{(j)} \in \mathbb{R}^{m \times m}$ ,  $b^{(j)} \in \mathbb{R}^m$ .

- Last layer is an *affine* mapping (no activation function):

$$a(o^{(L)}) = W^{(L+1)} o^{(L)} + b^{(L+1)},$$

where  $W^{(L+1)} \in \mathbb{R}^{k \times m}$  and  $b^{(L+1)} \in \mathbb{R}^k$ .

- The full neural network function is given by the *composition* of layers:

$$f(x) = \left(a \circ h^{(L)} \circ \dots \circ h^{(1)}\right)(x) \tag{8}$$

- Last layer typically gives us a score. How to do classification?

# Multinomial Logistic Regression

- From each  $x$ , we compute a linear score function for each class:

$$x \mapsto (\langle w_1, x \rangle, \dots, \langle w_k, x \rangle) \in \mathbb{R}^k$$

- We need to map this  $\mathbb{R}^k$  vector into a probability vector  $\theta$ .
- The **softmax function** maps scores  $s = (s_1, \dots, s_k) \in \mathbb{R}^k$  to a categorical distribution:

$$(s_1, \dots, s_k) \mapsto \theta = \mathbf{Softmax}(s_1, \dots, s_k) = \left( \frac{\exp(s_1)}{\sum_{i=1}^k \exp(s_i)}, \dots, \frac{\exp(s_k)}{\sum_{i=1}^k \exp(s_i)} \right)$$

$P(y=1|x) \quad \dots \quad P(y=k|x)$

# Nonlinear Generalization of Multinomial Logistic Regression

- From each  $x$ , we compute a non-linear score function for each class:

$$x \mapsto (f_1(x), \dots, f_k(x)) \in \mathbb{R}^k$$

where  $f_i$ 's are outputs of the last hidden layer of a neural network.

- Learning: Maximize the log-likelihood of training data

$$\arg \max_{f_1, \dots, f_k} \sum_{i=1}^n \log \left[ \text{Softmax}(f_1(x), \dots, f_k(x))_{y_i} \right].$$

# Neural network as a feature extractor

- OverFeat is a neural network for object classification, localization, and detection.
  - Trained on the huge ImageNet dataset
  - Lots of computing resources used for training the network.
- All those hidden layers of the network are very valuable *features*.
  - Paper: “CNN Features off-the-shelf: an Astounding Baseline for Recognition”
  - Showed that using features from OverFeat makes it easy to achieve state-of-the-art performance on new vision tasks.

We've seen

- Key idea: automatically discover useful features from raw data—*feature/representation learning*.
- Building blocks:
  - Input layer no learnable parameters
  - Hidden layer(s) perceptron + *nonlinear* activation function
  - Output layer affine (+ transformation)
- A single hidden layer is sufficient to approximate any function.
- In practice, often have multiple hidden layers.

Next, how to learn the parameters.