CSC 311: Introduction to Machine Learning

Lecture 5 - Linear Models III

Rahul G. Krishnan

University of Toronto, Fall 2023

Outline

Announcements

- HW2 out
- TA OH schedule posted!

IMPORTANT:

Ethics module pre survey out.

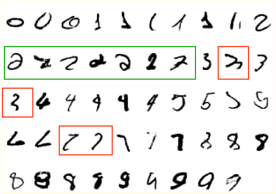
Due October 16 2023

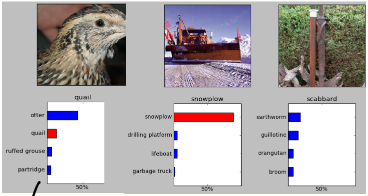
1. Softmax Regression
2. Tracking Model Performance
3. Limits of Linear Classification
4. Introducing Neural Networks
5. Expressivity of a Neural Network

Multi-class Classification

Last week: Binary classification
- Binary linear classification
- Logistic Regression

Task is to predict a discrete(> 2)-valued target.





MNIST Dataset 10 classes

Object classification.

visualizing probability of top 4 classes.

Two representations are equivalent.

- Targets form a discrete set $\{1, \ldots, K\}$. $t = k$ OR,
- Represent targets as one-hot vectors or one-of-K encoding:

$$\mathbf{t} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{\text{entry } k \text{ is } 1} \in \mathbb{R}^K$$

e.g if we have 4 classes

1- indexing notation.

$t = 1$

1	0	0	0

$t = 2$

0	1	0	0

Note: Be careful to map this correctly when zero indexing.

Linear Function of Inputs

Vectorized form:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \text{ or}$$
$$\mathbf{z} = \mathbf{W}\mathbf{x} \text{ with dummy } x_0 = 1$$

Non-vectorized form:

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k \quad \text{for} \quad k = 1, 2, ..., K$$

- $\mathbf{W}$: K x D matrix.
- $\mathbf{x}$: D x 1 vector.
- $\mathbf{b}$: K x 1 vector.
- $\mathbf{z}$: K x 1 vector.

(A) Setup

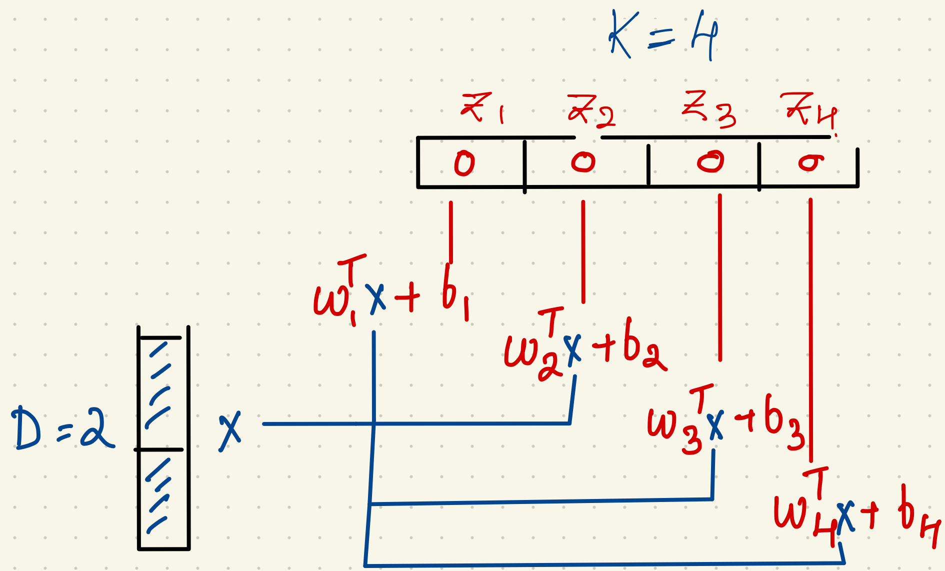
- 2 dim. input
- 4 class problem

(B) Model parameters

$$\overset{D=2}{} \qquad K=4 \qquad W = \begin{bmatrix} \text{---} & w_1 & \text{---} \\ \text{---} & w_2 & \text{---} \\ \text{---} & w_3 & \text{---} \\ \text{---} & w_4 & \text{---} \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

(C) Computation

$$z = Wx + b$$



Generating a Prediction

Interpret z_k as how much the model prefers the k-th prediction.

$$y_i = \begin{cases} 1, & \text{if } i = \arg\max_k z_k \\ 0, & \text{otherwise} \end{cases}$$

$\arg\max_k z_k$

$i=4$

z $\begin{bmatrix} -3 & -2 & 5 & 12 \end{bmatrix}$

y $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$

How does the $K = 2$ case relate to the binary linear classifiers?

Softmax Regression

Recall that we worked to interpret the output of linear classifiers as a probability.

- Soften the predictions for optimization.
- A natural activation function is the softmax function, a generalization of the logistic function:

make numbers positive & normalize

$$y_k = \text{softmax}(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

Note: $\sum_{k=1}^{K} y_k = 1$

- Inputs z_k are called the logits.
- Interpret outputs as probabilities.
- If z_k is much larger than the others, then $\text{softmax}(\mathbf{z})_k \approx 1$ and it behaves like argmax.

What does the $K = 2$ case look like?

$$\vec{z} \quad \begin{array}{|c|c|c|} \hline -2 & 0 & 2 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline e^{-2} & e^{0} & e^{2} \\ \hline \end{array}$$

$$Z = \sum_{k'} e^{z_{k'}} = e^{-2} + e^{0} + e^{2}.$$

$$\vec{y} \quad \begin{array}{|c|c|c|} \hline \frac{e^{-2}}{Z} & \frac{e^{0}}{Z} & \frac{e^{2}}{Z} \\ \hline \end{array}$$

Consider the case where $k=2$

Ⓐ Multiclass classification $k=2$

$$Y_{mc} \quad \frac{e^{z_1}}{e^{z_1}+e^{z_2}} \;\Big|\; \frac{e^{z_2}}{e^{z_1}+e^{z_2}}$$

$\uparrow$ softmax

$$z_1 \qquad z_2$$

$$w_1^T x + b_1 \;\Big|\; w_2^T x + b_2$$

$\uparrow$ linear output

x

Decision Rule: $\max\left(\underbrace{\frac{e^{z_1}}{e^{z_1}+e^{z_2}}}_{}\,,\ \frac{e^{z_2}}{e^{z_1}+e^{z_2}} \right)$ with Y^1_{mc} above the first term and Y^2_{mc} above the second

Ⓑ Binary classification

Ⓒ Mapping from Ⓐ $\rightarrow$ Ⓑ

$$\frac{Y^1_{mc}}{Y^2_{mc}} = e^{z_1 - z_2}$$

$$= e^{w_1^T x + b_1 - w_2^T x - b_2}$$

$$= e^{\underbrace{(w_1 - w_2)^T}_{w_{bc}} x + \underbrace{(b_1 - b_2)}_{b_{bc}}}$$

$$\underset{1}{\geq} \;\Rightarrow\; w_{bc}^T x + b_{bc}$$

Cross-Entropy as Loss Function

$\vec{Y}$

0.01	0.9	0.05	0.04

e.g $\vec{t}=$

0	1	0	0

Only one of the outputs contributes to the loss

Use cross-entropy as the loss function.

$$\mathcal{L}_{\text{CE}}(\mathbf{y}, \mathbf{t}) = -\sum_{k=1}^{K} t_k \log y_k = -\mathbf{t}^\top (\log \mathbf{y}),$$

where the log is applied element-wise.

Often use a combined softmax-cross-entropy function.

Gradient Descent Updates for Softmax Regression

Softmax Regression:

$$\begin{aligned} \mathbf{z} &= \mathbf{W}\mathbf{x} \\ \mathbf{y} &= \mathrm{softmax}(\mathbf{z}) \\ \mathcal{L}_{\mathrm{CE}} &= -\mathbf{t}^\top(\log \mathbf{y}) \end{aligned}$$

Gradient Descent Updates:

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) \mathbf{x}^{(i)}$$

Progress During Learning

- Track progress during learning by plotting training curves.
- Chose the training criterion (e.g. squared error, cross-entropy) partly to be easy to optimize.
- May wish to track other metrics to measure performance (even if we can't directly optimize them).

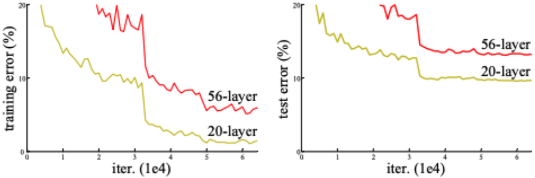


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Tracking Accuracy for Binary Classification

We can track accuracy, or fraction correctly classified.

- Equivalent to the average 0–1 loss, the error rate, or fraction incorrectly classified.
- Useful metric to track even if we couldn't optimize it.

Another way to break down the accuracy:

$$Acc = \frac{TP + TN}{P + N} = \frac{TP + TN}{(TP + FN) + (TN + FP)}$$

- P=num positive; N=num negative;
- TP=true positives; TN=true negatives
- FP=false positive or a type I error
- FN=false negative or a type II error

true class

pred. class	0	1
0	TN	FN
1	FP	TP

Accuracy is Highly Sensitive to Class Imbalance

Suppose you are screening patients for a particular disease.
It's known that 1% of patients have that disease.

- What is the simplest model that can achieve 99% accuracy?

Sensitivity and Specificity

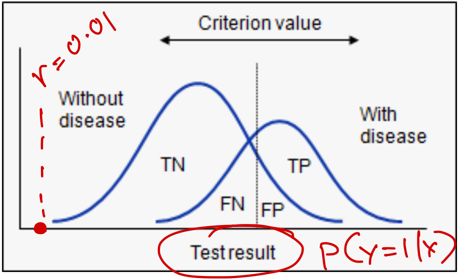
Useful metrics even under class imbalance!

$$\text{Sensitivity} = \frac{TP}{TP+FN} \text{ [True positive rate]}$$

$$\text{Specificity} = \frac{TN}{TN+FP} \text{ [True negative rate]}$$

What happens if our problem is not linearly separable?
How do we pick a threshold for $y = \sigma(x)$?

- A binary model to predict whether someone has a disease.
- What happens to sensitivity and specificity as you slide the threshold from left to right?



Logistic

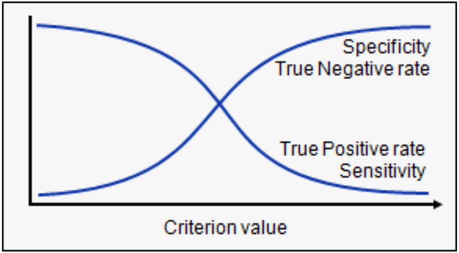
$p(y=1|x) \geq r$

$\Rightarrow 1$

0 o/w

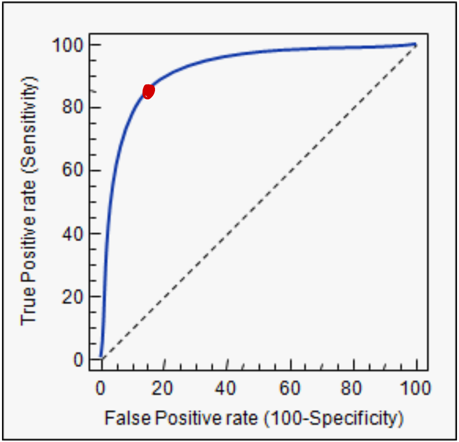
Tradeoff between Sensitivity and Specificity

As we increase the criterion value (i.e. move from left to right), how do the sensitivity and specificity change?



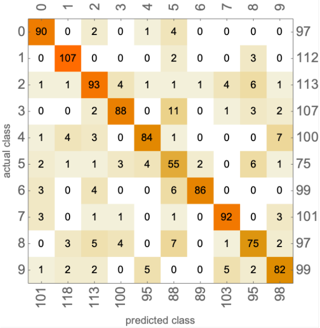
Receiver Operating Characteristic (ROC) Curve

Area under the ROC curve (AUC) can quantify if a binary classifier achieves a good tradeoff between sensitivity and specificity.



Confusion Matrix for Multi-Class classification

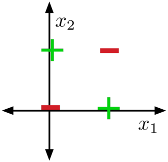
- Visualizes how frequently certain classes are confused.
- $K \times K$ matrix; rows are true labels, columns are predicted labels, entries are frequencies
- What does the confusion matrix for a perfect classifier look like?



XOR is Not Linearly Separable

NOT, AND are linearly separable.

Some datasets are not linearly separable, e.g. **XOR**.

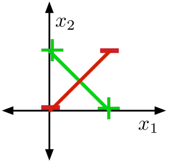


Visually obvious, but how can we prove this formally?

Proof That XOR is Not Linearly Separable

Proof by Contradiction:

- Half-spaces are convex. That is, if two points lie in a half-space, the line segment connecting them also lie in the same half-space.
- Suppose that the problem is feasible.
- If the positive examples are in the positive half-space, then the green line segment must be as well.
- Similarly, the red line segment must lie in the negative half-space.
- But, the intersection can't lie in both half-spaces. Contradiction!



Classifying XOR Using Feature Maps

Sometimes, we can overcome this limitation using feature maps, e.g., for **XOR**.

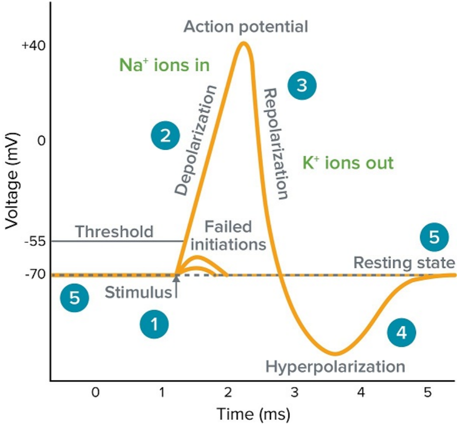
$$\boldsymbol{\psi}(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

x_1	x_2	$\psi_1(\mathbf{x})$	$\psi_2(\mathbf{x})$	$\psi_3(\mathbf{x})$	t
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	1	1	1	0

- This is linearly separable. (Try it!)
- Designing feature maps can be hard. Can we learn them?

Neurons in the Brain

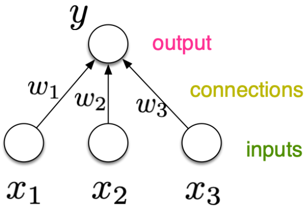
Neurons receive input signals and accumulate voltage. After some threshold, they will fire spiking responses.



[Pic credit: www.moleculardevices.com]

A Simpler Neuron

For neural nets, we use a much simpler model for neuron, or **unit**:

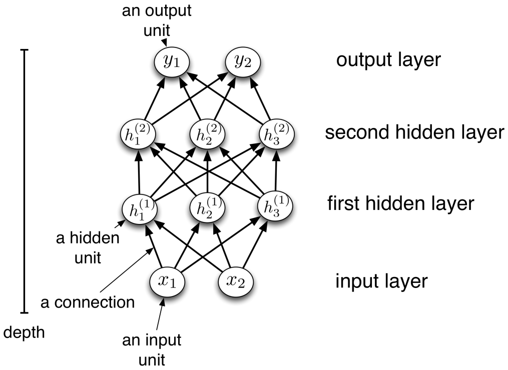


$$\overset{\text{output}}{y} = \underset{\text{activation function}}{\phi}\left(\overset{\text{weights}}{\mathbf{w}}^\top \underset{\text{inputs}}{\mathbf{x}} + \overset{\text{bias}}{b}\right)$$

- Similar to logistic regression: $y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$
- By throwing together lots of these simple neuron-like processing units, we can do some powerful computations!

A Feed-Forward Neural Network

- A directed acyclic graph (DAG)
- Units are grouped into layers

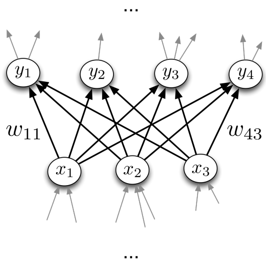


Multilayer Perceptrons

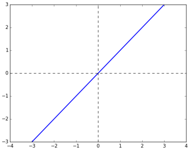
- A multi-layer network consists of fully connected layers.
- In a fully connected layer, all input units are connected to all output units.
- Each hidden layer i connects N_{i-1} input units to N_i output units. Weight matrix is N_i x N_{i-1}.
- The outputs are a function of the input units:

$$\mathbf{y} = f(\mathbf{x}) = \phi\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

ϕ is applied component-wise.

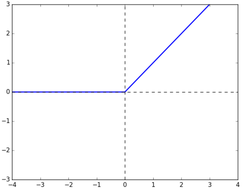


Some Activation Functions



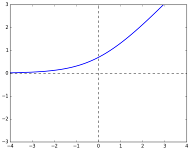
Identity

$$y = z$$



Rectified Linear Unit (ReLU)

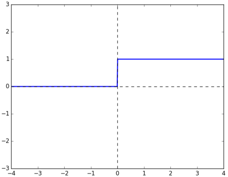
$$y = \max(0, z)$$



Soft ReLU

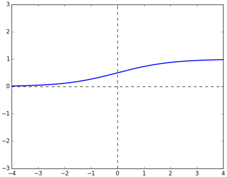
$$y = \log 1 + e^z$$

More Activation Functions



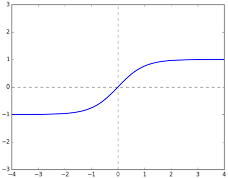
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Computation in Each Layer

Each layer computes a function.

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}) = \phi(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)}) = \phi(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

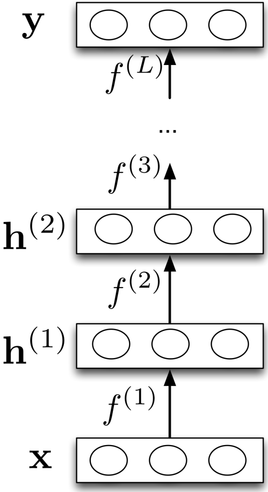
$$\vdots$$

$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$

Final layer

If task is regression: choose
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = (\mathbf{w}^{(L)})^\top \mathbf{h}^{(L-1)} + b^{(L)}$$

If task is binary classification: choose
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = \sigma((\mathbf{w}^{(L)})^\top \mathbf{h}^{(L-1)} + b^{(L)})$$

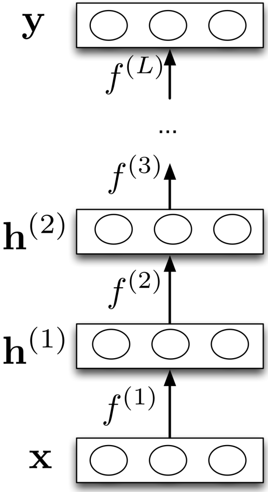


A Composition of Functions

The network computes
a composition of functions.

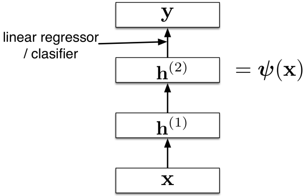
$$\mathbf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).$$

Modularity: We can implement each layer's computations as a black box.

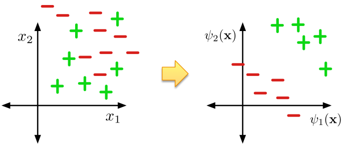


Feature Learning

Neural nets can be viewed as a way of learning features:

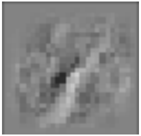


The goal:



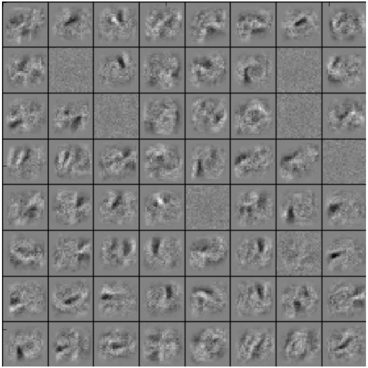
Feature Learning

- Suppose we're trying to classify images of handwritten digits.
- Each image is represented as a vector of $28 \times 28 = 784$ pixel values.
- Each hidden unit in the first layer acts as a **feature detector**.
- We can visualize $\mathbf{w}$ by reshaping it into an image.
  Below is an example that responds to a diagonal stroke.



Features for Classifying Handwritten Digits

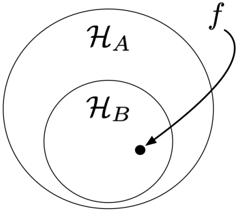
Features learned by the first hidden layer of a handwritten digit classifier:



Unlike hard-coded feature maps (e.g., in polynomial regression), features learned by neural networks adapt to patterns in the data.

Expressivity

- A hypothesis space $\mathcal{H}$ is the set of functions that can be represented by some model.
- Consider two models A and B with hypothesis spaces $\mathcal{H}_A, \mathcal{H}_B$.
- If $\mathcal{H}_B \subseteq \mathcal{H}_A$, then A is more expressive than B.
  A can represent any function f in $\mathcal{H}_B$.



- Some functions (XOR) can't be represented by linear classifiers.
  Are deep networks more expressive?

Expressive Power of Linear Networks

- Consider a linear layer: the activation function was the identity. The layer just computes an affine transformation of the input.
- Any sequence of linear layers is equivalent to a single linear layer.

$$\mathbf{y} = \underbrace{\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$$

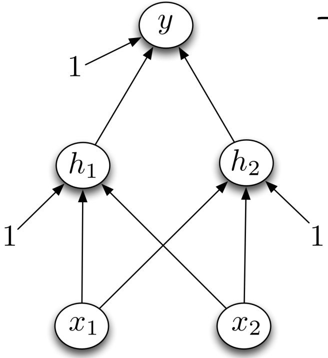
- Deep linear networks can only represent linear functions — no more expressive than linear regression.

Expressive Power of Non-linear Networks

- Multi-layer feed-forward neural networks with non-linear activation functions
- **Universal Function Approximators**: They can approximate any function arbitrarily well, i.e., for any $f : \mathcal{X} \rightarrow \mathcal{T}$ there is a sequence $f_i \in \mathcal{H}$ with $f_i \rightarrow f$.
- True for various activation functions (e.g. thresholds, logistic, ReLU, etc.)

Designing a Network to Classify XOR

Assume a hard threshold activation function.



x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Designing a Network to Classify XOR

h_1 computes $x_1 \vee x_2$

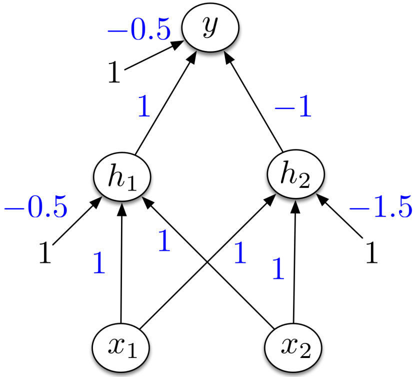
$$\mathbb{I}[x_1 + x_2 - 0.5 > 0]$$

h_2 computes $x_1 \wedge x_2$

$$\mathbb{I}[x_1 + x_2 - 1.5 > 0]$$

y computes $h_1 \wedge (\neg h_2) = x_1 \oplus x_2$

$$\begin{aligned} &\mathbb{I}[h_1 - h_2 - 0.5 > 0] \\ &\equiv \mathbb{I}[h_1 + (1 - h_2) - 1.5 > 0] \end{aligned}$$

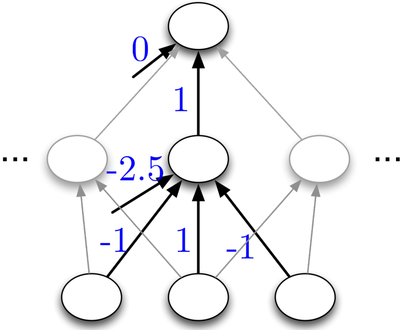


x_1	x_2	x_1 OR x_2	x_1 AND x_2	$\neg(x_1$ AND $x_2)$	$(x_1$ OR $x_2)$ AND $\neg(x_1$ AND $x_2)$
0	0	0	0	1	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	1	1	0	0

Universality for Binary Inputs and Targets

- Hard threshold hidden units, linear output
- Strategy: 2^D hidden units, each of which responds to one particular input configuration

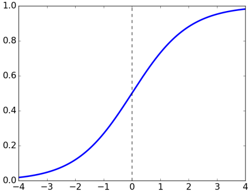
x_1	x_2	x_3	t
	$\vdots$		$\vdots$
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	$\vdots$		$\vdots$



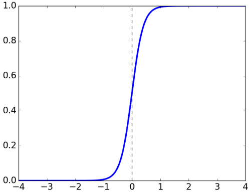
- Only requires one hidden layer, though it is extremely wide.

Expressivity of the Logistic Activation Function

- What about the logistic activation function?
- Approximate a hard threshold by scaling up w and b.



$$y = \sigma(x)$$



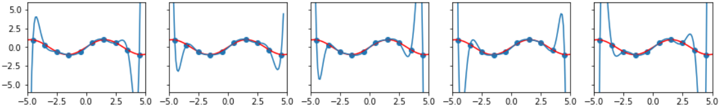
$$y = \sigma(5x)$$

- Logistic units are differentiable, so we can learn weights with gradient descent.

What is Expressivity Good For?

- May need a very large network to represent a function.
- Non-trivial to learn the weights that represent a function.
- If you can learn any function, over-fitting is potentially a serious concern!

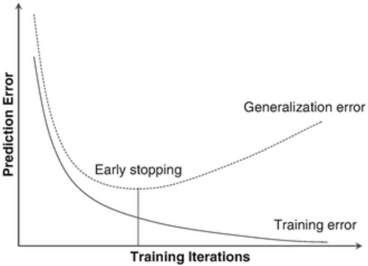
  For the polynomial feature mappings, expressivity increases with the degree M, eventually allowing multiple perfect fits to the training data. This motivated L^2 regularization.



- Do neural networks over-fit and how can we regularize them?

Regularization and Over-fitting for Neural Networks

- The topic of over-fitting (when & how it happens, how to regularize, etc.) for neural networks is not well-understood, even by researchers!
  - In principle, you can always apply L^2 regularization.
  - You will learn more in CSC413.
- A common approach is early stopping, or stopping training early, because over-fitting typically increases as training progresses.



- Don't add an explicit $\mathcal{R}(\boldsymbol{\theta})$ term to our cost.

Conclusion

- Multi-class classification
- Selecting good metrics to track performance in models
- From linear to non-linear models