CSC 311: Introduction to Machine Learning

Lecture 5 - Linear Models III, Neural Nets I

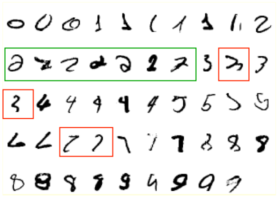
Sayyed Nezhadi

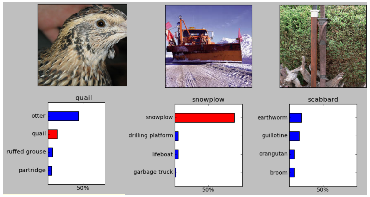
University of Toronto, Summer 2023

Outline

Multi-class Classification

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





Targets in Multi-class Classification

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

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

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.

Generating a Prediction

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

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

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

Softmax Regression

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

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

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

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

Cross-Entropy as Loss Function

Use cross-entropy as the loss function.

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

where the log is applied element-wise.

Often use a combined softmax-cross-entropy function.

Gradient Descent Updates for Softmax Regression

Softmax Regression:

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

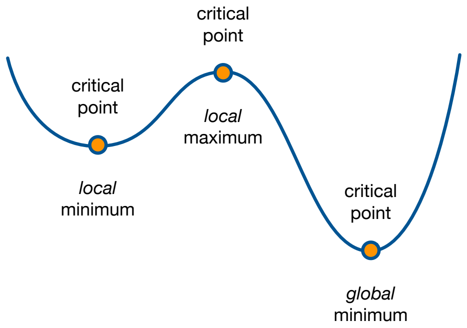
Gradient Descent Updates:

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

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

When are Critical Points Optimal?

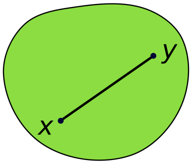
- Gradient descent finds a critical point, but is it a global optimum?
- In general, a critical point may be a local optimum only.
- If a function is convex, then every critical point is a global optimum.

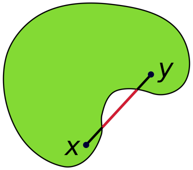


Convex Sets

A set $\mathcal{S}$ is convex if
any line segment connecting two points in $\mathcal{S}$ lies entirely within $\mathcal{S}$.

$$\begin{aligned} &\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{S} \\ &\Rightarrow \quad \lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in \mathcal{S} \quad \text{for } 0 \leq \lambda \leq 1. \end{aligned}$$





Weighted averages or convex combinations of points in S lie within $\mathcal{S}$.

$$\begin{aligned} &\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{S} \\ &\Rightarrow \lambda_1 \mathbf{x}_1 + \cdots + \lambda_N \mathbf{x}_N \in \mathcal{S} \quad \text{for } \lambda_i > 0,\ \lambda_1 + \cdots \lambda_N = 1. \end{aligned}$$

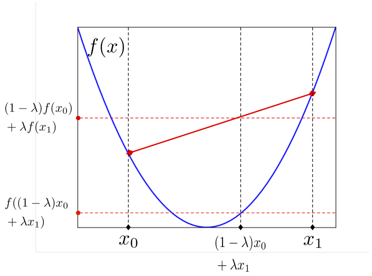
Convex Functions

A function f is convex if

- the line segment between any two points on f's graph lies above f's graph between the two points.
- the set of points lying above the graph of f is convex.
- for any $\mathbf{x}_0, \mathbf{x}_1$ in the domain of f,

$$f((1-\lambda)\mathbf{x}_0 + \lambda\mathbf{x}_1) \leq (1-\lambda)f(\mathbf{x}_0) + \lambda f(\mathbf{x}_1)$$

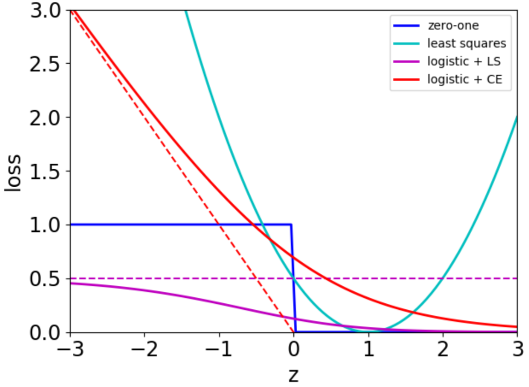
- f is bowl-shaped.



Convex Loss Functions

For linear models, $z = \mathbf{w}^\top \mathbf{x} + b$ is a linear function of $\mathbf{w}$ and b. If the loss function is a convex function of z, then it is also a convex function of $\mathbf{w}$ and b.

Which loss functions are convex?



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).

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:

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

- 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

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?
- You are able to observe a feature which is 10 times more likely in a patient who has cancer. Does this improve your accuracy?

Sensitivity and Specificity

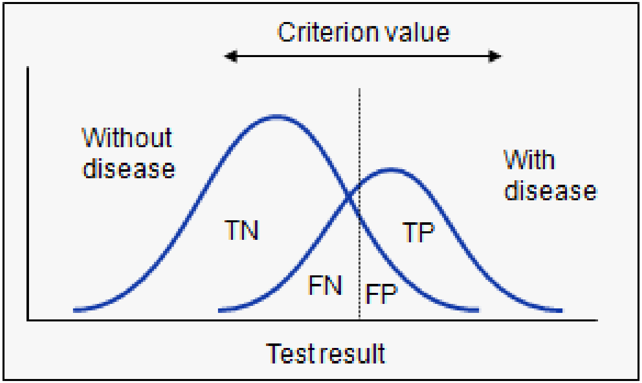
Useful metrics even under class imbalance!

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

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

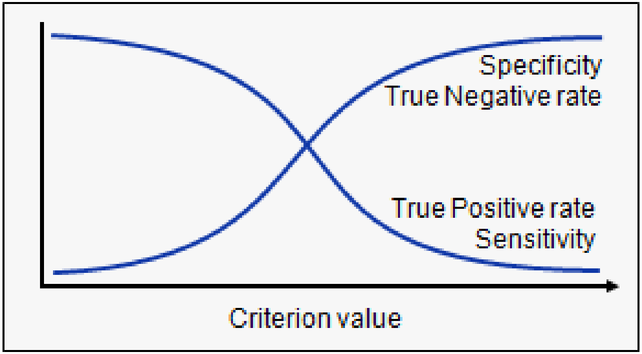
What happens if our classification problem is not truly (log-)linearly seperable? How do we pick a threshold for $y = \sigma(x)$?

Designing Diagnostic Tests

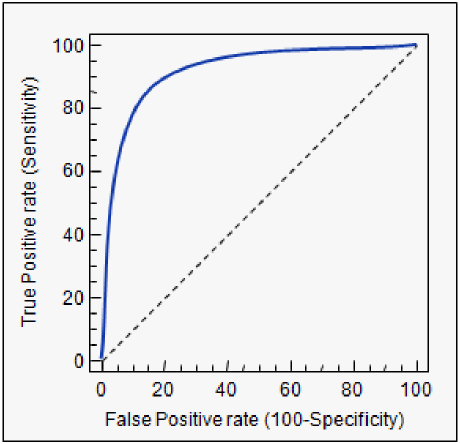


- You've developed a binary model to predict whether someone has a specific disease.
- What happens to sensitivity and specificity as you slide the threshold from left to right?

Tradeoff between Sensitivity and specificity



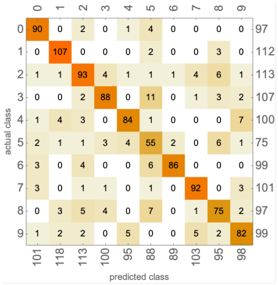
Receiver Operating Characteristic (ROC) curve



Area under the ROC curve (AUC) is a useful metric to track if a binary classifier achieves a good tradeoff between sensitivity and specificity.

Confusion Matrix for Multi-Class classification

- You might also be interested in how frequently certain classes are confused.
- Confusion matrix: $K \times K$ matrix; rows are true labels, columns are predicted labels, entries are frequencies
- What does the confusion matrix look like for a perfect classifier?

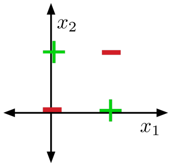


	0	1	2	3	4	5	6	7	8	9	
0	90	0	2	0	1	4	0	0	0	0	97
1	0	107	0	0	0	2	0	0	3	0	112
2	1	1	93	4	1	1	1	4	6	1	113
3	0	0	2	88	0	11	0	1	3	2	107
4	1	4	3	0	84	1	0	0	0	7	100
5	2	1	1	3	4	55	2	0	6	1	75
6	3	0	4	0	0	6	86	0	0	0	99
7	3	0	1	1	0	1	0	92	0	3	101
8	0	3	5	4	0	7	0	1	75	2	97
9	1	2	2	0	5	0	0	5	2	82	99
	101	118	113	100	95	88	89	103	95	98	

actual class
predicted class

XOR is Not Linearly Separable

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

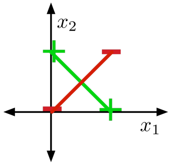


Visually obvious, but how can we prove this formally?

Proof That XOR is Not Linearly Separable

Proof by Contradiction:

- Half-spaces are convex: if two points lie in a half-space, line segment connecting them also lie in the same half-space.
- Suppose there were some feasible weights (hypothesis). 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 line within 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, just like for linear regression. E.g., for **XOR**:

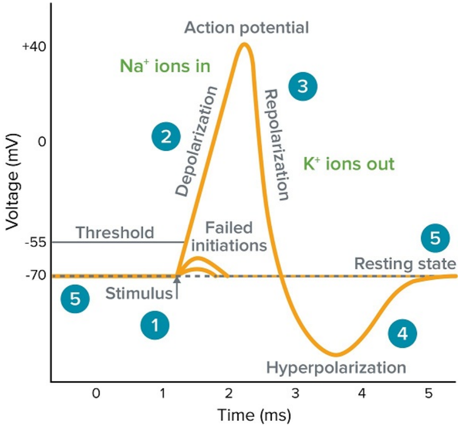
$$\boldsymbol{\psi}(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

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?

A Neuron in the Brain

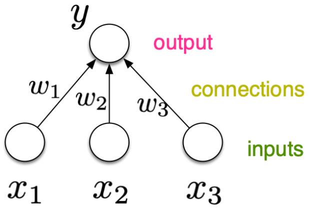
Neurons receive input signals and accumulate voltage. After some threshold they will fire spiking responses.

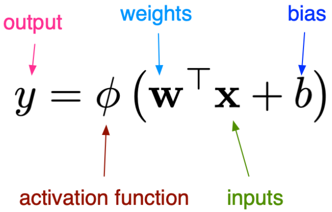


[Pic credit: www.moleculardevices.com]

A Simpler Neuron

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

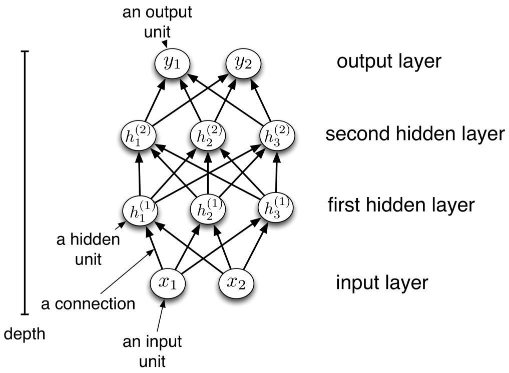




- Compare with logistic regression: $y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$
- By throwing together lots of these incredibly simplistic neuron-like processing units, we can do some powerful computations!

A Feed-Forward Neural Network

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

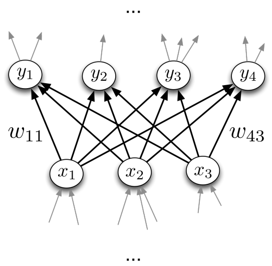


Multilayer Perceptrons

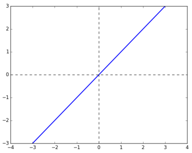
- A multi-layer network consists of fully connected layers.
- In a fully connected layer, all input units are connected to all output units.
- Each hidden layer i connects N_{i-1} input units to N_i output units. Weight matrix is N_i x N_{i-1}.
- The outputs are a function of the input units:

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

ϕ is applied component-wise.

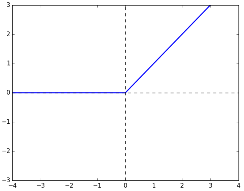


Some Activation Functions



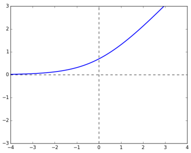
Identity

$y = z$



Rectified Linear Unit (ReLU)

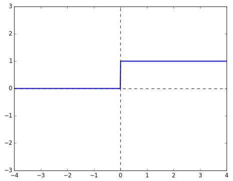
$y = \max(0, z)$



Soft ReLU

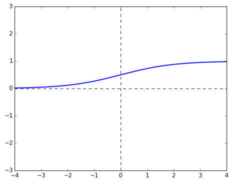
$y = \log 1 + e^z$

More Activation Functions



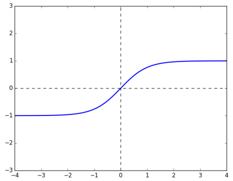
Hard Threshold

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



Logistic

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



Hyperbolic Tangent (tanh)

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

A Composition of Functions

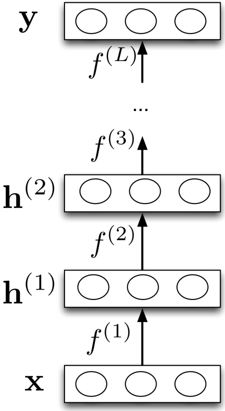
- Each layer computes a function, so the network computes a composition of functions:

$$\begin{aligned}
\mathbf{h}^{(1)} &= f^{(1)}(\mathbf{x}) = \phi(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \\
\mathbf{h}^{(2)} &= f^{(2)}(\mathbf{h}^{(1)}) = \phi(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\
&\vdots \\
\mathbf{y} &= f^{(L)}(\mathbf{h}^{(L-1)})
\end{aligned}$$

- Or more simply:

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

- Neural nets provide modularity: we can implement each layer's computations as a black box.

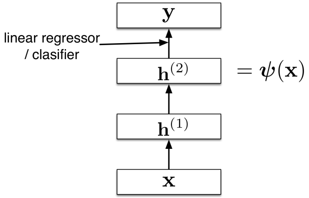


Last Layer

- 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 $\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = \sigma((\mathbf{w}^{(L)})^\top \mathbf{h}^{(L-1)} + b^{(L)})$

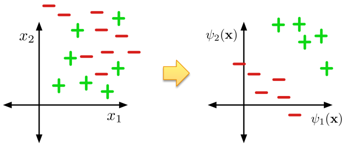
Feature Learning

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



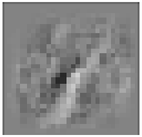
$\mathbf{y}$
linear regressor / clasifier
$\mathbf{h}^{(2)}$
$= \boldsymbol{\psi}(\mathbf{x})$
$\mathbf{h}^{(1)}$
$\mathbf{x}$

The goal:



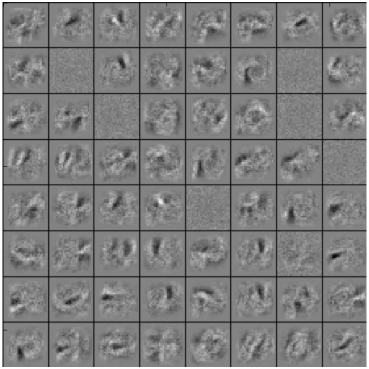
Feature Learning

- 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

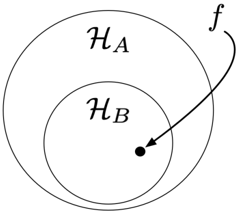
Some features learned by the first hidden layer of a handwritten digit classifier:



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.

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

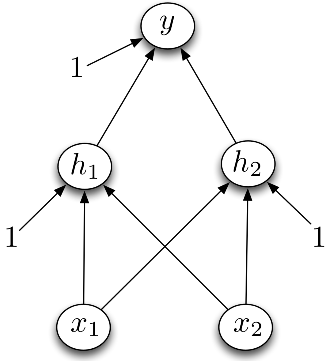
- Deep linear networks can only represent linear functions — no more expressive than linear regression.

Expressive Power of Non-linear Networks

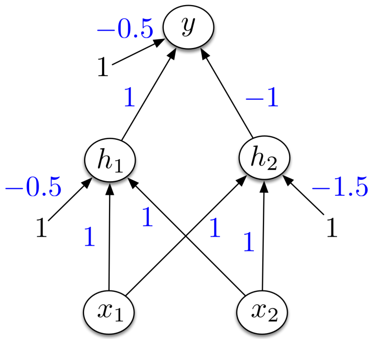
- Multilayer feed-forward neural nets with *nonlinear* activation functions
- **Universal Function Approximators**:
  They can approximate any function arbitrarily well,
  i.e., for any $f : \mathcal{X} \to \mathcal{T}$ there is a sequence $f_i \in \mathcal{H}$ with $f_i \to f$.
- True for various activation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

Designing a Network to Classify XOR

Assume hard threshold activation function



Designing a Network to Classify XOR

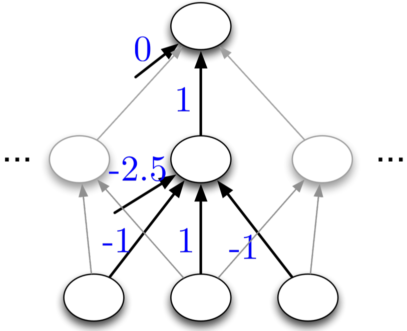


- h_1 computes $\mathbb{I}[x_1 + x_2 - 0.5 > 0]$
  - i.e. x_1 OR x_2
- h_2 computes $\mathbb{I}[x_1 + x_2 - 1.5 > 0]$
  - i.e. x_1 AND x_2
- y computes $\mathbb{I}[h_1 - h_2 - 0.5 > 0] \equiv \mathbb{I}[h_1 + (1 - h_2) - 1.5 > 0]$
  - i.e. h_1 AND (NOT h_2) $= x_1$ XOR x_2

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

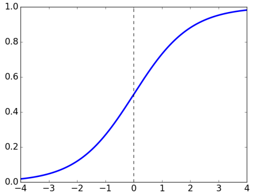
x_1	x_2	x_3	t
	⋮		⋮
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	⋮		⋮



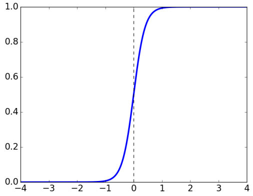
- Only requires one hidden layer, though it is extremely wide.

Expressivity

- What about the logistic activation function?
- Approximate a hard threshold by scaling up the weights and biases:



$y = \sigma(x)$

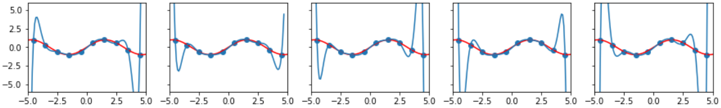


$y = \sigma(5x)$

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

Expressivity—What is it good for?

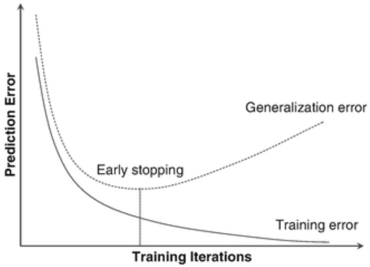
- Universality is not necessarily a golden ticket.
  - You may need a very large network to represent a given function.
  - How can you find the weights that represent a given function?
- Expressivity can be bad: if you can learn any function, overfitting is potentially a serious concern!
  - Recall the polynomial feature mappings from Lecture 2. Expressivity increases with the degree M, eventually allowing multiple perfect fits to the training data.



    This motivated L^2 regularization.
- Do neural networks overfit and how can we regularize them?

Regularization and Overfitting for Neural Networks

- The topic of overfitting (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 overfitting typically increases as training progresses.



- Unlike L^2 regularization, we don't add an explicit $\mathcal{R}(\boldsymbol{\theta})$ term to our cost.

Conclusion

- Multi-class classification
- Convexity of loss functions
- Selecting good metrics to track performance in models
- From linear to non-linear models