CSC 311: Introduction to Machine Learning

Lecture 5 - Linear Models III, Neural Nets I

Roger Grosse Rahul G. Krishnan Guodong Zhang

University of Toronto, Fall 2021

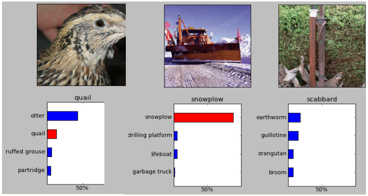
Multiclass Classification and Softmax Regression

Overview

- Classification: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete(>2)-valued target
- Examples of multi-class classification
  - predict the value of a handwritten digit
  - classify e-mails as spam, travel, work, personal

Multiclass Classification

- Classification tasks with more than two categories:



Multiclass Classification

- Targets form a discrete set $\{1, \ldots, K\}$.
- It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

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

Multiclass Linear Classification

- We can start with a linear function of the inputs.
- Now there are D input dimensions and K output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $\mathbf{W}$.
- Also, we have a K-dimensional vector $\mathbf{b}$ of biases.
- A linear function of the inputs:

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

- We can eliminate the bias $\mathbf{b}$ by taking $\mathbf{W} \in \mathbb{R}^{K \times (D+1)}$ and adding a dummy variable $x_0 = 1$. So, vectorized:

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

Multiclass Linear Classification

- How can we turn this linear prediction into a one-hot prediction?
- We can interpret the magnitude of z_k as an measure of how much the model prefers k as its prediction.
- If we do this, we should set

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

- **Exercise:** how does the case of $K = 2$ relate to the prediction rule in binary linear classifiers?

Softmax Regression

- We need to soften our predictions for the sake of optimization.
- We want soft predictions that are like probabilities, i.e., $0 \leq y_k \leq 1$ and $\sum_k y_k = 1$.
- A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

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

  - Outputs can be interpreted as probabilities (positive and sum to 1)
  - If z_k is much larger than the others, then $\text{softmax}(\mathbf{z})_k \approx 1$ and it behaves like argmax.
  - **Exercise:** how does the case of $K = 2$ relate to the logistic function?
- The inputs z_k are called the logits.

Softmax Regression

- If a model outputs a vector of class probabilities, we can 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 elementwise.

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

Softmax Regression

- Softmax regression (with dummy $x_0 = 1$):

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

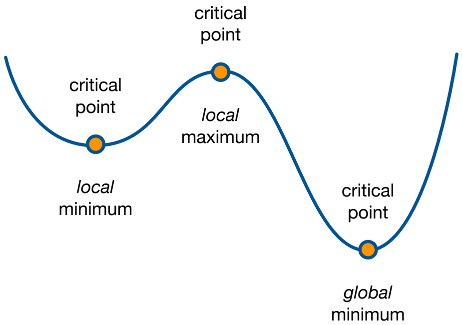
- Gradient descent updates can be derived for each row of $\mathbf{W}$:

$$\frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{\mathrm{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$
$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) \mathbf{x}^{(i)}$$

- Similar to linear/logistic reg (no coincidence) (verify the update)

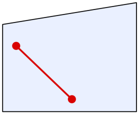
Convexity

When are critical points optimal?



- Gradient descent finds a critical point, but it may be a local optima.
- Convexity is a property that guarantees that all critical points are global minima.

Convex Sets



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

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

- A simple inductive argument shows that for $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{S}$, weighted averages, or convex combinations, lie within the set:

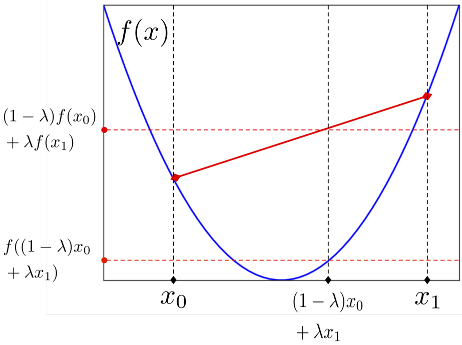
$$\lambda_1 \mathbf{x}_1 + \cdots + \lambda_N \mathbf{x}_N \in \mathcal{S} \quad \text{for } \lambda_i > 0, \; \lambda_1 + \cdots \lambda_N = 1.$$

Convex Functions

- A function f is convex if for any $\mathbf{x}_0, \mathbf{x}_1$ in the domain of f,

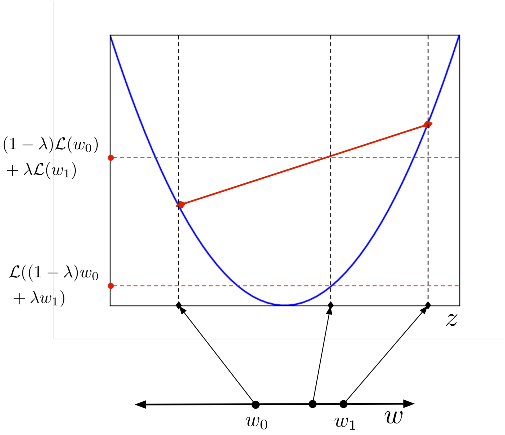
$$f((1-\lambda)\mathbf{x}_0 + \lambda\mathbf{x}_1) \leq (1-\lambda)f(\mathbf{x}_0) + \lambda f(\mathbf{x}_1)$$

- Equivalently, the set of points lying above the graph of f is convex.
- Intuitively: the function is bowl-shaped.



Convex Functions

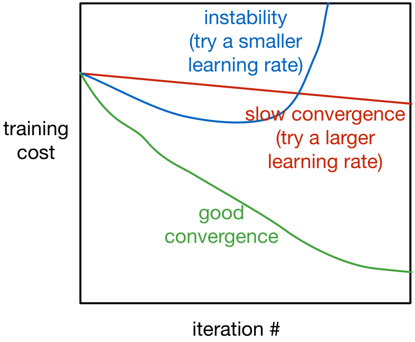
- We just saw that the least-squares loss $\frac{1}{2}(y - t)^2$ is convex as a function of y
- For a linear model, $z = \mathbf{w}^\top \mathbf{x} + b$ is a linear function of $\mathbf{w}$ and b. If the loss function is convex as a function of z, then it is convex as a function of $\mathbf{w}$ and b.



Tracking model performance

Progress during learning

- Recall we introduced training curves as a way to track progress during learning.



- The training criterion (e.g. squared error, cross-entropy) is chosen partly to be easy to optimize.
- We may which to track other metrics which better match what we're interested in, even if we can't directly optimize them.

Metrics for Binary classification

- Recall that the average of 0–1 loss is the error rate, or fraction incorrectly classified.
  - We noted we couldn't optimize it, but it's still a useful metric to track.
  - Equivalently, we can track the accuracy, or fraction correct.
  - Typically, the error rate behaves similarly to the cross-entropy loss, but this isn't always the case.
- Another way to break down the accuracy:
  - 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

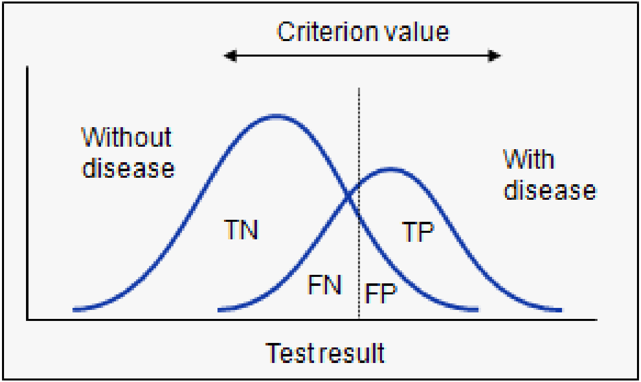
$$Acc = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Discuss:** When might accuracy present a misleading picture of performance?

The limitations of accuracy

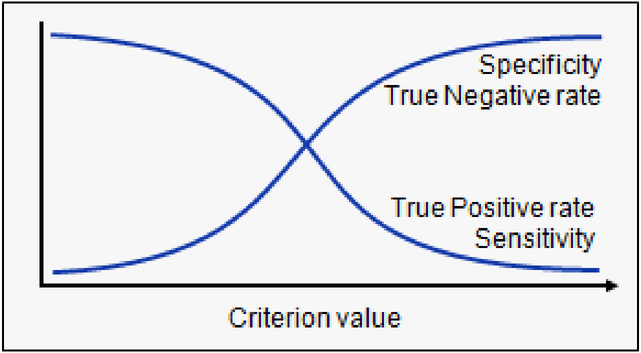
- Accuracy is highly sensitive to class imbalance.
  - Suppose you're trying to screen patients for a particular disease, and under the data generating distribution, 1% of patients have that disease.
  - How can you achieve 99% accuracy?
  - You are able to observe a feature which is 10x more likely in a patient who has cancer. Does this improve your accuracy?
- Sensitivity and specificity are useful metrics even under class imbalance.
  - Sensitivity $= \frac{TP}{TP+FN}$ [True positive rate]
  - Specificity $= \frac{TN}{TN+FP}$ [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 prediction model to indicate whether someone has a specific disease
- What happens to sensitivity and specificity as you slide the threshold from left to right?

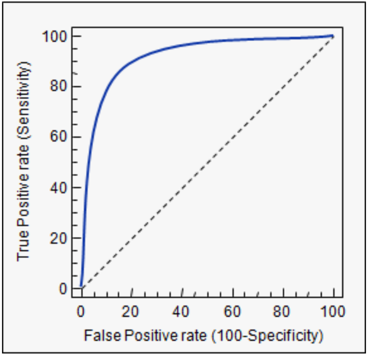
Sensitivity and specificity



- Tradeoff between sensitivity and specificity

Receiver Operating Characteristic (ROC) curve

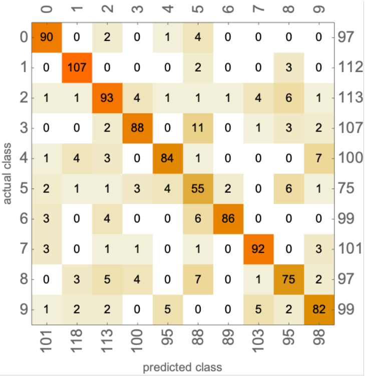
Receiver Operating Characteristic (ROC) curve



- y axis: sensitivity
- x axis: 100-specificity
- Area under the ROC curve (AUC) is a useful metric to track if a binary classifier achieves a good tradeoff between sensitivity and specificity.

Metrics for Multi-Class classification

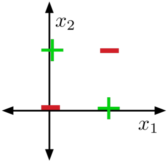
- You might also be interested in how frequently certain classes are confused.
- Confusion matrix: $K \times K$ matrix; rows are true labels, columns are predicted labels, entries are frequencies
- Question: what does the confusion matrix look like if the classifier is perfect?



Limits of Linear Classification

Limits of Linear Classification

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

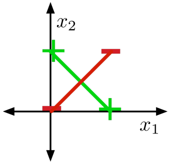


Visually obvious, but how to show this?

Limits of Linear Classification

Showing that XOR is not linearly separable (proof by contradiction)

- If two points lie in a half-space, line segment connecting them also lie in the same halfspace.
- 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!

Limits of Linear Classification

- Sometimes we can overcome this limitation using feature maps, just like for linear regression. E.g., for **XOR**:

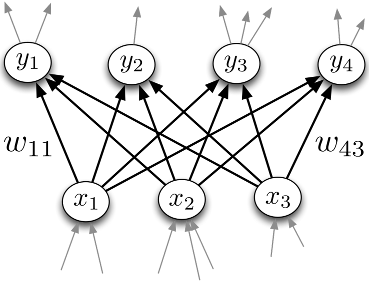
$$\boldsymbol{\psi}(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

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?

Neural Networks

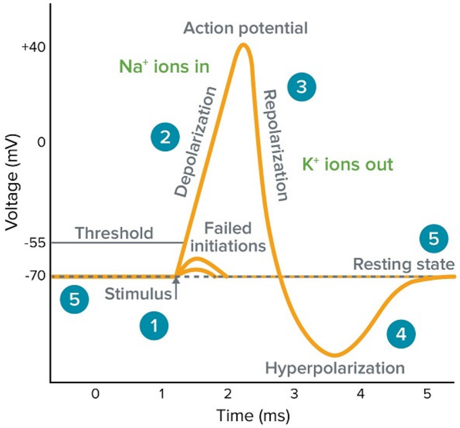
...



...

Inspiration: The Brain

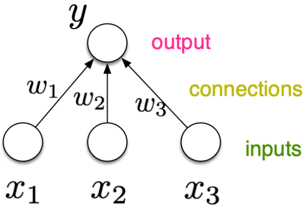
- Neurons receive input signals and accumulate voltage. After some threshold they will fire spiking responses.

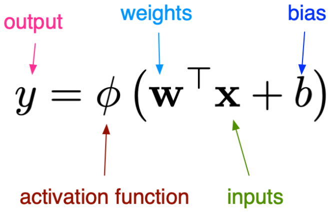


[Pic credit: www.moleculardevices.com]

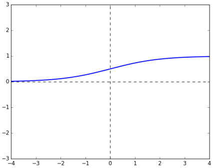
Inspiration: The Brain

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





- Compare with logistic regression: $y = \sigma(\mathbf{w}^\top \mathbf{x} + b)$

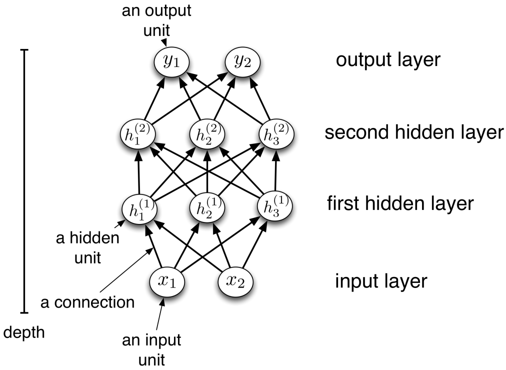


- By throwing together lots of these incredibly simplistic neuron-like processing units, we can do some powerful computations!

Multilayer Perceptrons

Multilayer Perceptrons

- We can connect lots of units together into a **directed acyclic graph**.
- Typically, units are grouped into **layers**.
- This gives a **feed-forward neural network**.



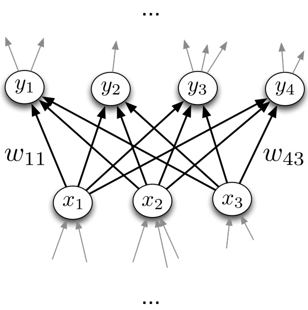
Multilayer Perceptrons

- Each hidden layer i connects N_{i-1} input units to N_i output units.
- In a fully connected layer, all input units are connected to all output units.
- Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network.

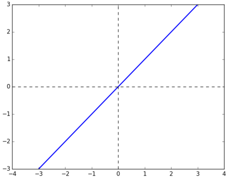
- If we need to compute M outputs from N inputs, we can do so using matrix multiplication. This means we'll be using a $M \times N$ matrix
- The outputs are a function of the input units:

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

  ϕ is typically applied component-wise.
- A multilayer network consisting of fully connected layers is called a multilayer perceptron.

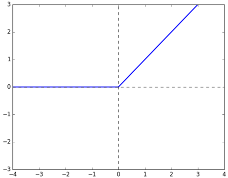


Some activation functions:



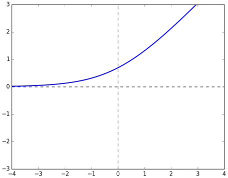
Identity

$$y = z$$



Rectified Linear Unit (ReLU)

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

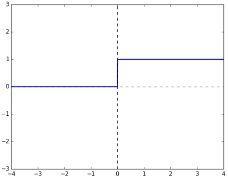


Soft ReLU

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

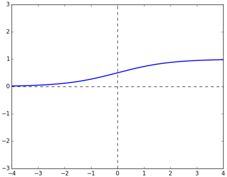
Multilayer Perceptrons

Some activation functions:



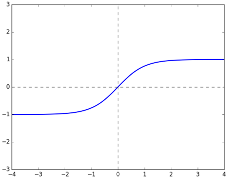
Hard Threshold

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



Logistic

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



Hyperbolic Tangent (tanh)

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

Multilayer Perceptrons

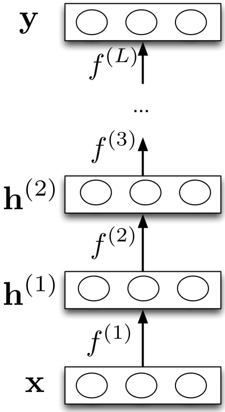
- Each layer computes a function, so the network computes a composition of functions:

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

- 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.

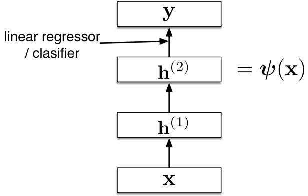


Feature Learning

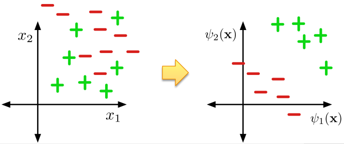
Last layer:

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

So neural nets can be viewed as a way of learning features:

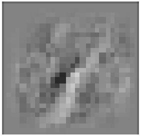


- The goal:



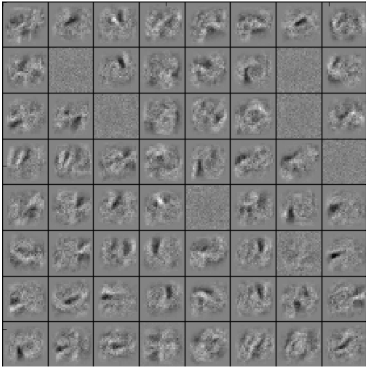
Feature Learning

- 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 first-layer hidden unit computes $\phi(\mathbf{w}_i^\top \mathbf{x})$. It acts as a **feature detector**.
- We can visualize $\mathbf{w}$ by reshaping it into an image. Here's an example that responds to a diagonal stroke.



Feature Learning

Here are some of the features learned by the first hidden layer of a handwritten digit classifier:

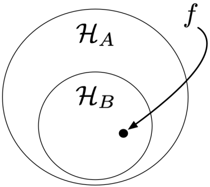


- Unlike hard-coded feature maps (e.g., in polynomial regression), features learned by neural networks adapt to patterns in the data.

Expressivity

- In Lecture 4, we introduced the idea of a hypothesis space $\mathcal{H}$, which is the set of input-output mappings that can be represented by some model. Suppose we are deciding between two models A, 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?

Expressivity—Linear Networks

- Suppose a layer's activation function was the identity, so the layer just computes a affine transformation of the input
  - We call this a linear layer
- Any sequence of *linear* layers can be equivalently represented with a single linear layer.

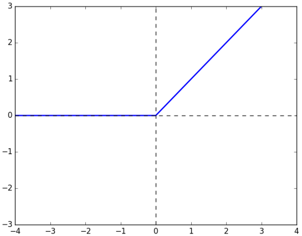
$$\mathbf{y} = \underbrace{\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$$

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

Expressive Power—Non-linear Networks

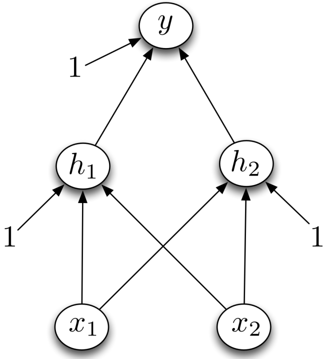
- Multilayer feed-forward neural nets with *nonlinear* activation functions are **universal function approximators**: they can approximate any function arbitrarily well, i.e., for any $f : \mathcal{X} \to \mathcal{T}$ there is a sequence $f_i \in \mathcal{H}$ with $f_i \to f$.

- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though ReLU is "almost" linear, it's nonlinear enough.

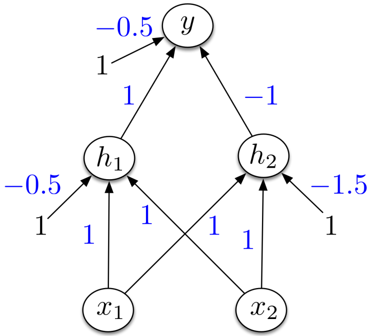


Designing a network to classify XOR:

Assume hard threshold activation function



Multilayer Perceptrons



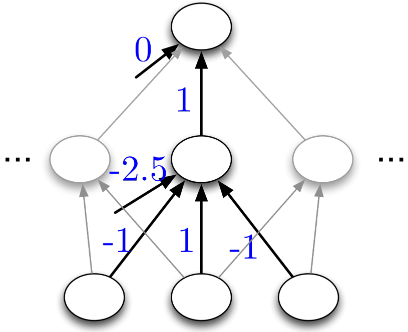
- h_1 computes $\mathbb{I}[x_1 + x_2 - 0.5 > 0]$
  - i.e. x_1 OR x_2
- h_2 computes $\mathbb{I}[x_1 + x_2 - 1.5 > 0]$
  - i.e. x_1 AND x_2
- y computes $\mathbb{I}[h_1 - h_2 - 0.5 > 0] \equiv \mathbb{I}[h_1 + (1 - h_2) - 1.5 > 0]$
  - i.e. h_1 AND (NOT h_2) $= x_1$ XOR x_2

Expressivity

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

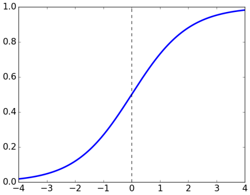
x_1	x_2	x_3	t
	$\vdots$		$\vdots$
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	$\vdots$		$\vdots$



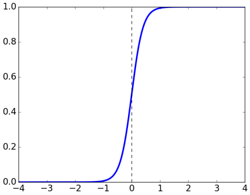
- Only requires one hidden layer, though it needs to be extremely wide.

Expressivity

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



$$y = \sigma(x)$$

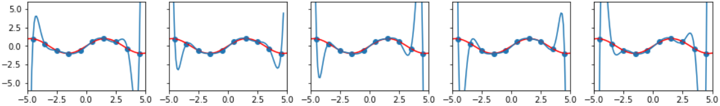


$$y = \sigma(5x)$$

- This is good: logistic units are differentiable, so we can train them with gradient descent.

Expressivity—What is it good for?

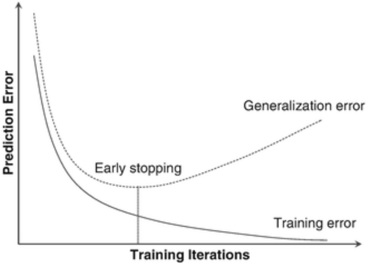
- Universality is not necessarily a golden ticket.
  - You may need a very large network to represent a given function.
  - How can you find the weights that represent a given function?
- Expressivity can be bad: if you can learn any function, overfitting is potentially a serious concern!
  - Recall 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