Quiz 1

- Due No due date
- Points 10
- Questions 10
- Available Jan 16 at 7:10pm - Jan 18 at 11:59pm
- Time Limit None
- Allowed Attempts 3

Instructions

Intro and Universal Approximators

This quiz covers lectures 1 and 2. Several of the questions invoke concepts from the hidden slides in the slide deck, which were not covered in class. So please go over the slides before answering the questions.

You will have three attempts for the quiz. Questions will be shuffled and you will not be informed of the correct answers until after the deadline. While you may discuss the concepts underlying the questions with others, you must solve all questions on your own - see course policy.

Attempt History

	Attempt	Time	Score
KEPT	Attempt 3	10 minutes	8.67 out of 10
LATEST	Attempt 3	10 minutes	8.67 out of 10
	Attempt 2	49 minutes	7 out of 10
	Attempt 1	99 minutes	7.92 out of 10

ⓘ Correct answers are hidden.

Score for this attempt: 8.67 out of 10
Submitted Jan 18 at 10:33pm
This attempt took 10 minutes.

Question 1
1 / 1 pts

Which of your quiz scores will be dropped?

Hint: watch Lecture 0

- No scores will be dropped
- Lowest 2 quiz scores
- Lowest 1 quiz scores
- Lowest 3 quiz scores

Question 2
1 / 1 pts

We sometimes say that neural networks are connectionist machines as opposed to von Neumann machines. Which of the following describe why we make this distinction? (select all that apply)

Slide: lec 1, “Connectionist Machines”. slide 47-48

- [x] A von Neumann machine has a general purpose architecture with a processing unit that is distinct from the memory that holds the programs and data. A connectionist machine makes no distinction between processing unit and the program.
- [ ] It is possible to create hardware implementations of von Neumann machines (e.g. CPU's) as well as software implementations (e.g. virtual machines). However, connectionist machines can only be implemented in software (e.g. neural networks in Python).
- [x] A von Neumann machine can be used for general-purpose computing by simply providing a different program, without changing the machine itself. A connectionist machine implements a specific program, and changing the program requires changing the machine.
- [ ] Because of its flexibility, a von Neumann machine is capable of computing any Boolean function of a given number of Boolean inputs, whereas connectionist machines, no matter how complex, are fundamentally unable to model certain types of Boolean functions.

See lec 1 recording, slide “Connectionist Machines”. The main idea is that von Neumann machines have a separate processor and memory. Programs reside in memory and are called into the processor. They can swap out programs without changing architecture, simply by changing the program in memory.

Connectionist machines, on the other hand, encode the program in the connections between their elements. To change the program, these connections, and hence the very architecture of the machine must be changed – i.e. the machine must be rebuilt.

Both von Neumann and connectionist machines have hardware implementations. (example of connectionist machines: your brain lol)

Connectionist machines with multiple layers can compute any Boolean function (lec 1, slide “A more generic model”).

Partial

Question 3
0.67 / 1 pts

What computational systems can compose arbitrary Boolean functions? (select all that apply)

Hint: Lecture 1, slides 52, 58–63 and 76–81. Also, “compose” is read as arbitrary Boolean functions, not some Boolean functions.

- [ ] Digital circuits containing only OR and NOT logic gates.
- [x] McCulloch and Pitts's connectionist networks.
- [ ] One of Rosenblatt's perceptrons.
- [x] Turing’s B-type machines.

Arbitrary Boolean functions can be composed by systems that are functionally complete or that allow compositional networks. Digital circuits using OR and NOT gates are functionally complete, since they can be combined to implement all Boolean functions. McCulloch and Pitts showed that networks of Boolean threshold units can compute arbitrary Boolean propositions. Turing’s B-type machines extend A-type machines by adding trainable connections, enabling them to learn arbitrary Boolean functions. In contrast, a single Rosenblatt perceptron cannot represent all Boolean functions (e.g., XOR), so it does not qualify.
Incorrect

Question 4
0 / 1 pts

Is the following statement true or false: for any Boolean function on the real plane which outputs a 1 within a single connected but NON-CONVEX region (*i.e.* not a convex polygon), you need at least three layers (including the output layer) to model the function EXACTLY.

Hint: Lecture 2 slides "Complex decision boundaries"

- (•) True
- ( ) False

A 5-point star can be exactly modelled with two layers (1 hidden + 1 output). We would get this by using a threshold of 4 at the output neuron. A 5-point star is not convex.

Question 5
1 / 1 pts

How does the number of weights (note: not neurons) in an XOR network with *threshold logic* perceptrons with *1 hidden layer* grow with the number of inputs to the network?

See lec 2: Slides on "Optimal depth" and "Network size" 113-123

- Exponential or faster
- Linear
- Polynomial but faster than linear
- Between polynomial and exponential

In the XOR circuit shown in the slides, the number of perceptrons grows linearly with input, and each perceptron has only two inputs and, hence, a fixed number of weights (2). Thus, the number of weights too grows linearly with input

Question 6
1 / 1 pts

In general, as the depth of a NN increases, at what rate does the number of neurons/params required to represent a function change?

(Note: for the definition of network depth, see the lecture 2 recording)

Hint: Review Lec 2, slides on "The challenge of depth" (Slides 67-68)

- Decreases linearly
- Increases linearly
- Increases quadratically
- Decreases exponentially

The worst case problem of XORs is what we'll use for illustration. With one hidden layer you need O (exp(N)) neurons. With 3 hidden layers, you need O(exp(N/2)) neurons (the first 2 layers to compute XORs, which results in N/2 variables, and then one hidden layer to compute the XOR of N/2 variables. With 5 hidden layers the XOR net after K layers, the number of neurons required is O (exp (N/4)) etc.

So, the reduction is exponential with increasing depth.

In general, for a given function, deeper networks will require exponentially fewer neurons (and hence parameters) than shallower ones to model the function accurately (exactly, or with arbitrary precision).

Question 7
1 / 1 pts

A majority function is a Boolean function of N variables that produces a 1 if at least N/2 of the inputs are 1. Which of the following are true? (select all that apply)

Hint: Relevant Lecture 2 Slides: slides 29-30, 70, 75

- [ ] The number of gates in the smallest Boolean circuit of AND, OR and NOT gates that computes the majority function is polynomial in N.
- [x] A single perceptron can compute a majority function.
- [x] A fixed-depth Boolean circuit, comprising only AND, OR and NOT gates, will require $\Omega(\exp(N^{\alpha}))$ gates to compute the majority function ($\alpha > 0$)
- [ ] We will require a multilayer perceptron with $\Omega(\exp(N))$ perceptrons to compute a majority function

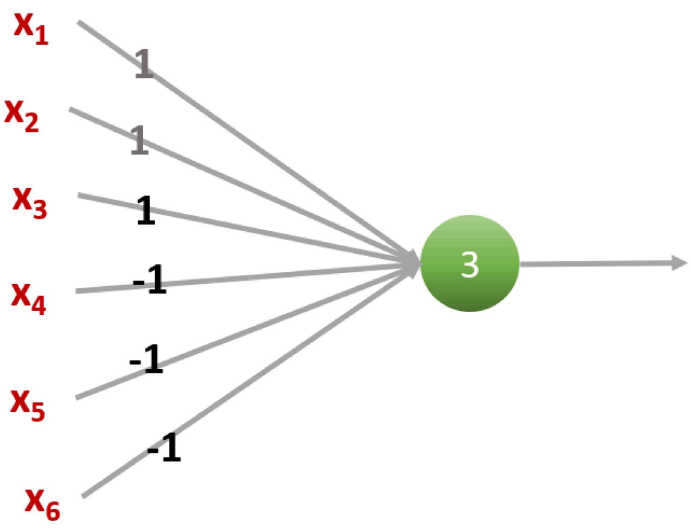
A single perceptron can receive all inputs, compute the sum of their values, and use a simple threshold activation of >= N/2. Thus, an MLP is not required.

For Boolean circuits, though, the specific lower bound is $\Omega(\exp(N^{1/(d-1)}))$, where d is the depth of the circuit (Smolensky 1993).

Decent explanation at https://eccc.weizmann.ac.il/report/2019/133/download/ (https://eccc.weizmann.ac.il/report/2019/133/download/)

If depth is allowed to grow, polynomial-size Boolean circuits are possible.

Question 8
1 / 1 pts



Under which condition(s) is the perceptron graph above guaranteed to fire? Note that ~ is NOT. (select all that apply)

Slide: lec 2, "Perceptron as a Boolean gate" slides 26-30

- [ ] ~x1 & ~x2 & ~x3 & x4 & x5 & x6
- [ ] Never fires
- [ ] x1 &~x2 & x3 & ~x4 & x5 & ~x6
- [x] x1 & x2 & x3 & ~x4 & ~x5 &~x6

x1 & x2 & x3 & ~x4 & ~x5 &~x6 = 1(1) + 1(1) +1(1) + 0(-1)+0(-1) +0(-1) =3

x1 &~x2 & x3 & ~x4 & x5 & ~x6= 1(1) + 0(1) +1(1) + 0(-1)+1(-1) +0(-1) =1

~x1 & ~x2 & ~x3 & x4 & x5 & x6 = 0(1) + 0(1) +0(1) + 1(-1)+1(-1) +1(-1) =-3

For this perceptron to fire, you need the total to be >= 3. Clearly you'll need all possible positive contributions, with all negative inputs turned off

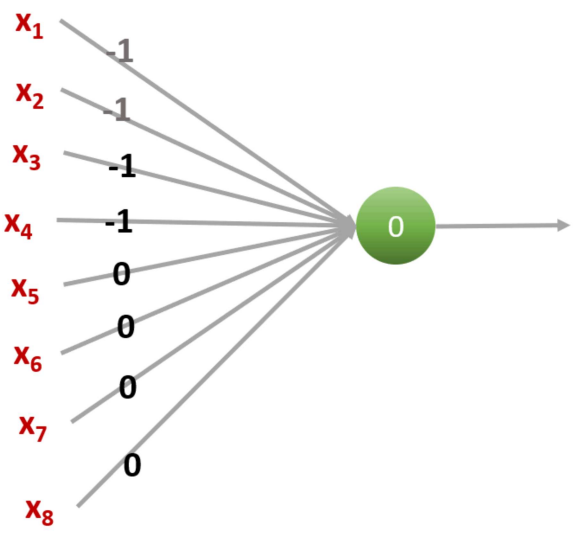
x1 & x2 & x3 & ~x4 & ~x5 &~x6 = 1(1) + 1(1) +1(1) + 0(-1)+0(-1) +0(-1) =3

x1 &~x2 & x3 & ~x4 & x5 & ~x6= 1(1) + 0(1) +1(1) + 0(-1)+1(-1) +0(-1) =1

~x1 & ~x2 & ~x3 & x4 & x5 & x6 = 0(1) + 0(1) +0(1) + 1(-1)+1(-1) +1(-1) =-3

Question 9
1 / 1 pts

Under which conditions will the perceptron graph below fire? Note that ~ is NOT. (select all that apply)



Slide: lec 2, "Perceptron as a Boolean gate", slides 26-30

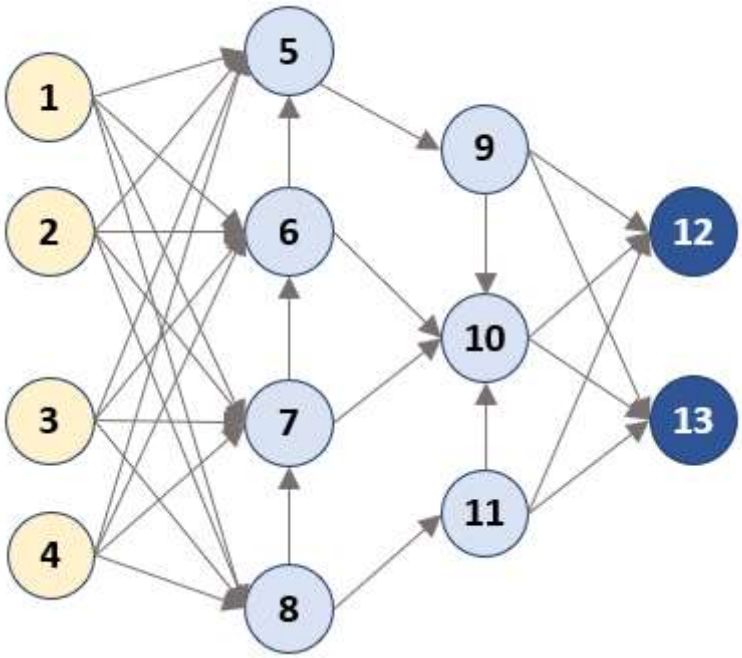
- [ ] x1 & x2 & x3 &x4
- [ ] Never fires
- [x] fires only if x1, x2, x3, x4 are all 0, regardless of x5 .. x8
- [x] ~x1 & ~x2 & ~x3 & ~x4

The number above each connection is the connection's weight w_i, which is multiplied to its corresponding X_i input (either 0 or 1). For the perceptron to fire, the sum of all w_i * X_i must be >= to the number in the circle.

As w_5 ~ w_8 are 0, any activations here won't influence firing. But if at least one of [X_1, X_2, X_3, X_4] are 1, the perceptron will not fire, as the total will never be >= the threshold 0.

(~x1 & ~x2…) is only true when x1…x4 are all zero.

Question 10
1 / 1 pts



If the yellow nodes are inputs (not neurons) and the dark blue nodes are outputs, what is the depth of this NN?

(Note: for the definition of network depth and layer number, see the lecture 2 recording)

Hint: lec 2, "Deep Structures", *Slides:* 17-18

7

Notice that the definition of depth is the longest path from the input to the output.

Quiz Score: 8.67 out of 10