CS 161 Intro. To Artificial Intelligence

Week 10, Discussion 1C

Li Cheng Lan

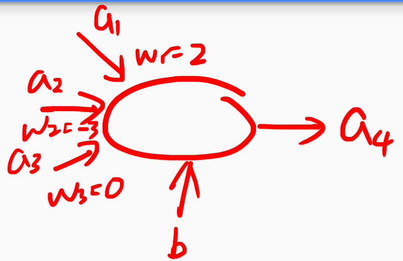
Today’s Topics

- **Neural Network (NN)**
  - Neurons
  - Feedforward NN
  - Neurons with Step Activation Functions
  - NN as a Function
  - Training NN
  - Convolutional Neural Network (CNN)
  - Strength & Limitation of NN
- Review of Final

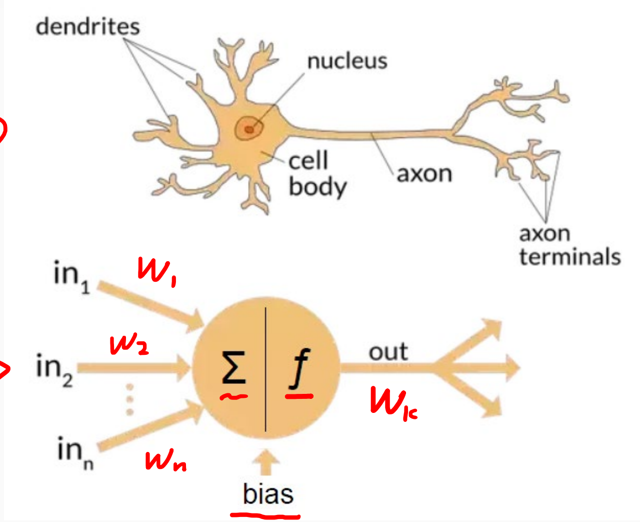
Neurons

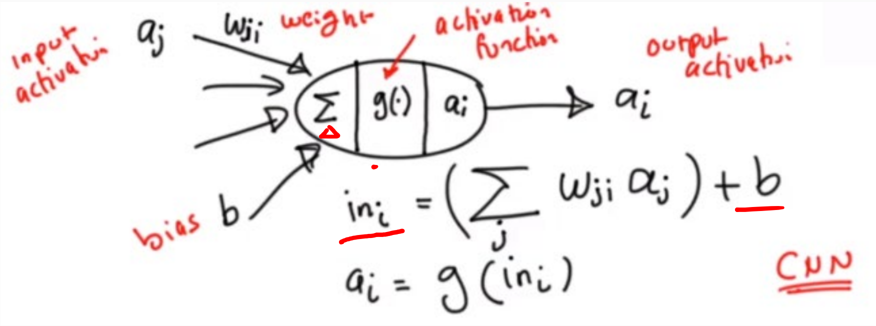
Important Terms:

- Activations
  - input
  - output
- Weights
- Bias
- Activation functions - $g(x)$
  - Binary step functions
    - E.g. Step, Sign
  - Linear functions
  - Non-linear functions
    - E.g. Sigmoid, ReLU



$$a_4 = g(\underline{2a_1 - 3a_2 + b})$$



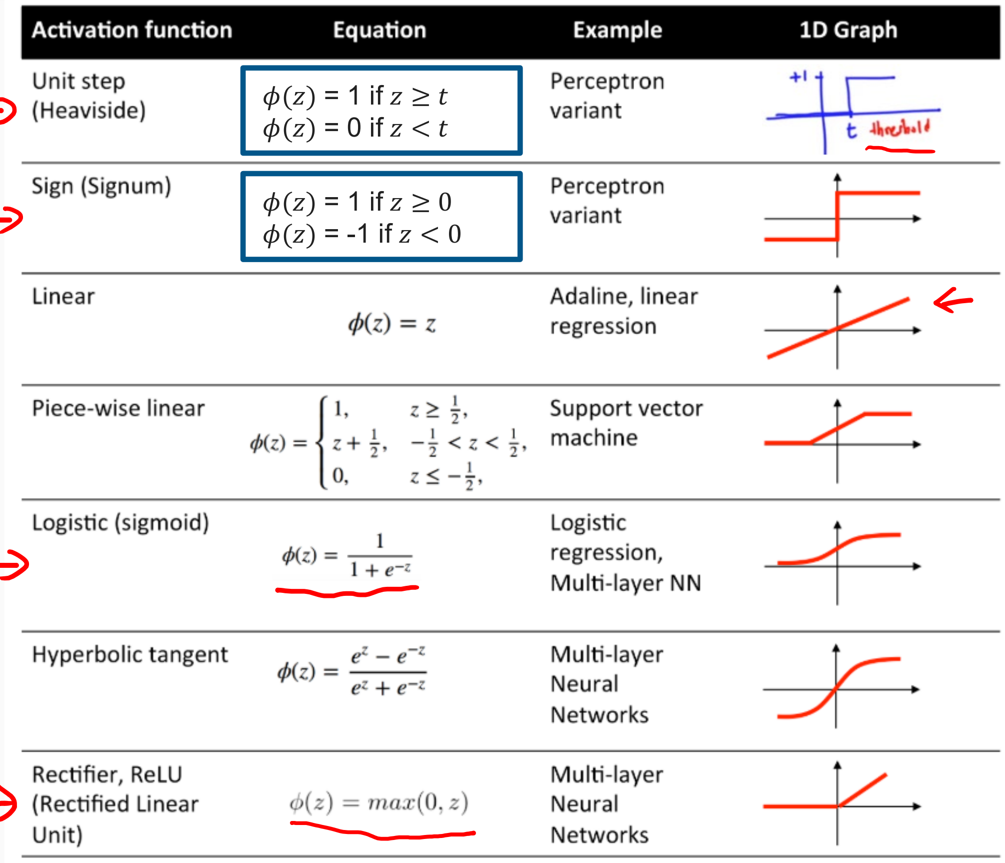


$$\underline{in_i} = \left(\sum_j W_{ji} a_j\right) + \underline{b}$$

$$a_i = g(in_i)$$

Activation Functions

- Activation functions
  - Binary step functions 1
    - E.g. Step, Sign
  - Linear functions
    - Same as linear regression
  - Non-linear functions 2
    - E.g. Sigmoid, ReLU



	Activation function	Equation	Example	1D Graph
1 →	Unit step (Heaviside)	$\phi(z) = 1$ if $z \geq t$ $\phi(z) = 0$ if $z < t$	Perceptron variant	+1 t threshold
1 →	Sign (Signum)	$\phi(z) = 1$ if $z \geq 0$ $\phi(z) = -1$ if $z < 0$	Perceptron variant	
	Linear	$\phi(z) = z$	Adaline, linear regression	←
	Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
2 →	Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
	Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
2 →	Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	

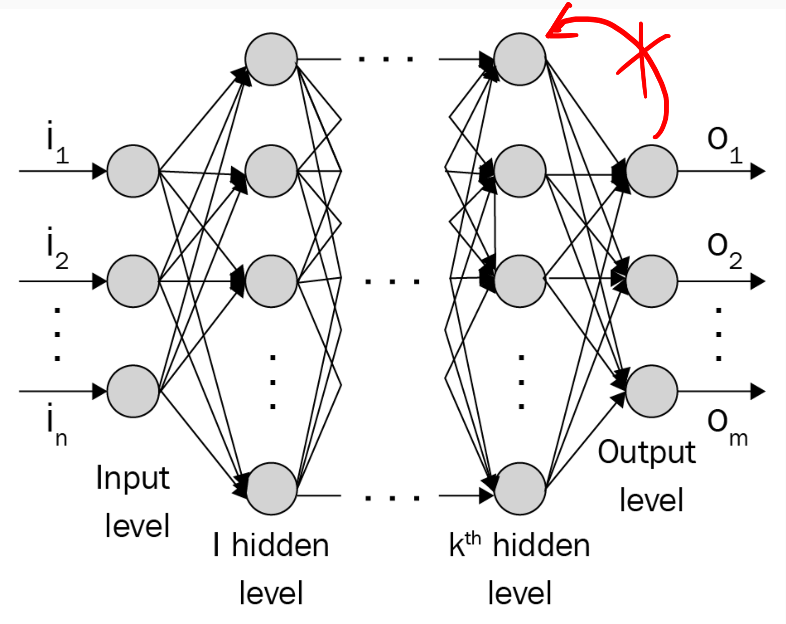
Source of image: http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/

Feedforward NN

A NN is also a universal function approximator

- A simple NN can represent a wide variety of functions when given appropriate parameters
- **Feedforward NN**: a NN that connections between the nodes do not form a cycle. Recurrent NNs are not feedforward

RNN



Source of image: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789346640/1/ch01lvl1sec12/architecture-of-a-simple-neural-network

Neurons with Step Activation Functions

Example Functions:

- And
- Or
- Not
- Limitation:
  - A single neuron can only represent linearly separable function
  - E.g. XOR won't work

And.

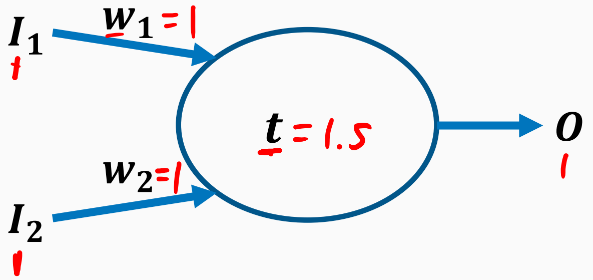
I_1	I_2	O
1	1	1
1	0	0
0	1	0
0	0	0

$$I \xrightarrow[w=-1]{} t=-0.5 \rightarrow O$$

Not:

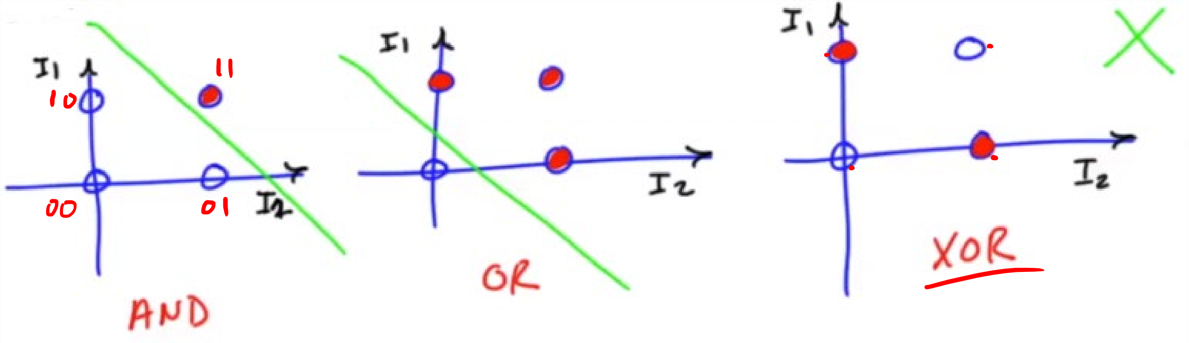
I	O
1	0
	1

$$I_1 W_1 + I_2 W_2 = \cancel{2}\ 1$$

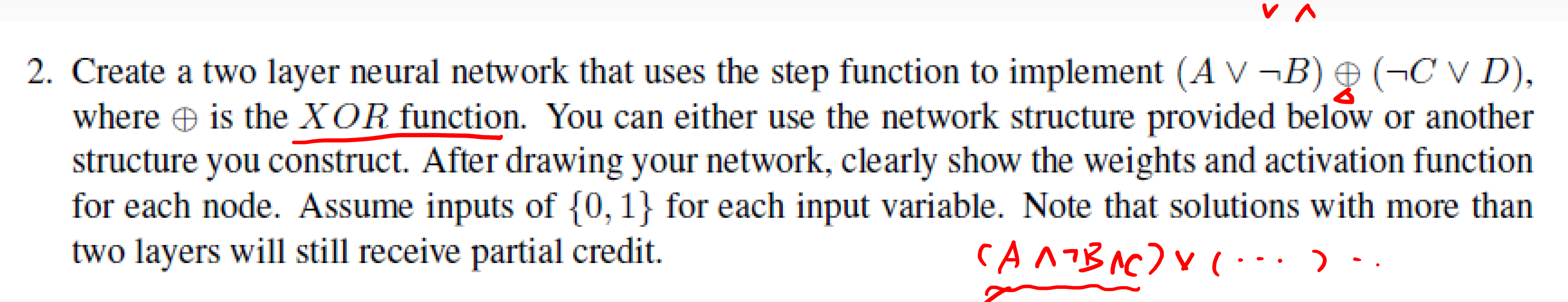


I_1	I_2	O
1	1	0
1	0	1
0	1	1
0	0	0

→



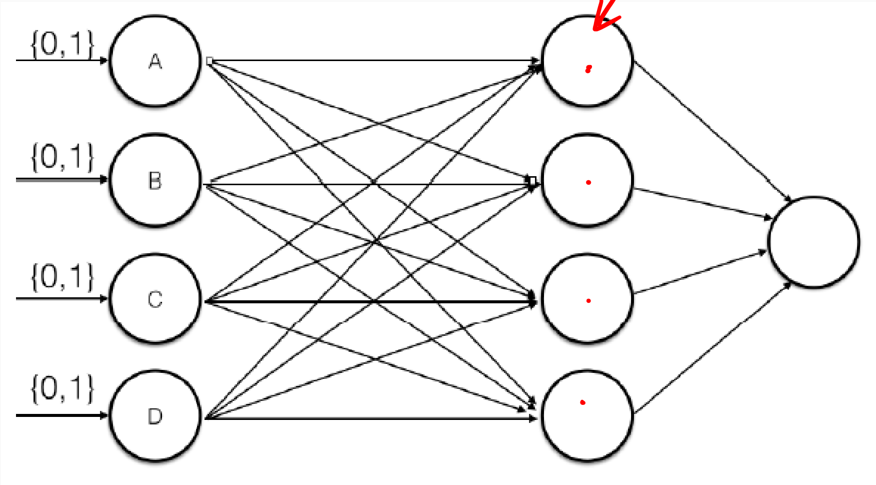
Hint for HW9 Q2



$\lor \quad \land$

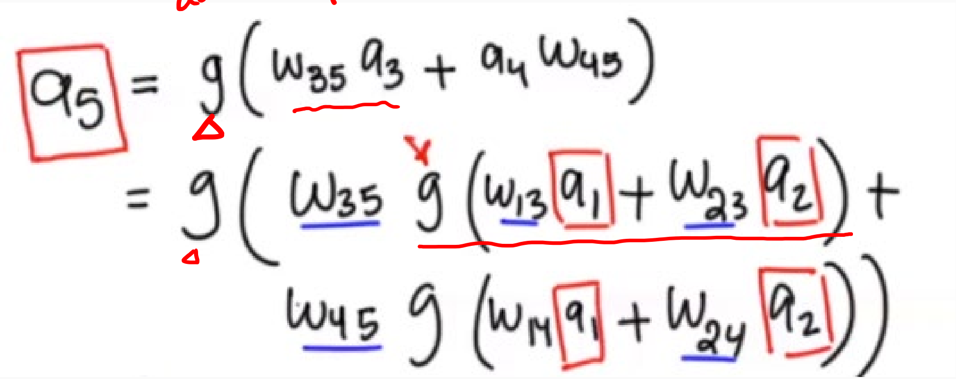
2. Create a two layer neural network that uses the step function to implement $(A \lor \neg B) \oplus (\neg C \lor D)$, where $\oplus$ is the XOR function. You can either use the network structure provided below or another structure you construct. After drawing your network, clearly show the weights and activation function for each node. Assume inputs of $\{0, 1\}$ for each input variable. Note that solutions with more than two layers will still receive partial credit.

$(A \land \neg B \land C) \lor (\cdots) \cdots$



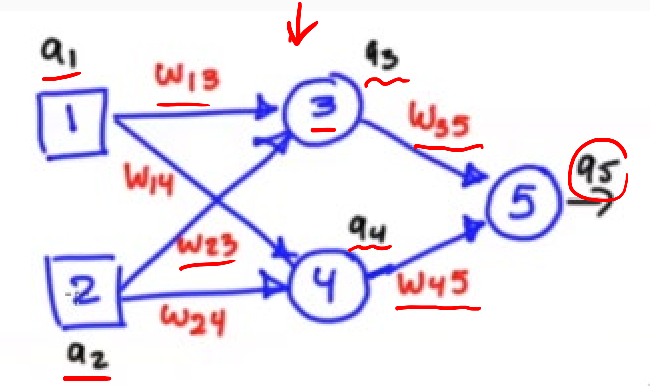
NN as a Function

activation function



$$\boxed{a_5} = g(\underline{w_{35} a_3} + a_4 w_{45})$$

$$= g(\underline{w_{35}}\, \underline{g(w_{13} \boxed{a_1} + w_{23} \boxed{a_2})} + \underline{w_{45}}\, g(w_{14} \boxed{a_1} + w_{24} \boxed{a_2}))$$

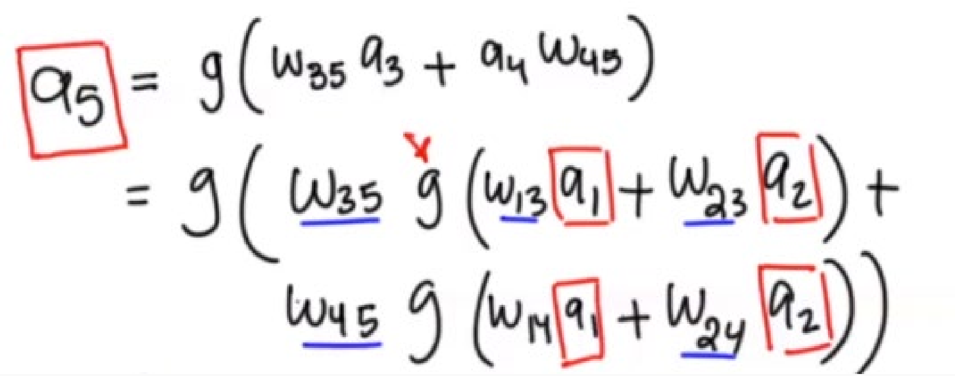


- $a_5 = f(a_1, a_2, w_{13}, w_{14}, \dots, w_{45})$
- If we are given the dataset, then given each input (data case), output activation is a function of weights
    - $\boldsymbol{a_5 = f(w_{13}, w_{14}, \dots, w_{45})}$

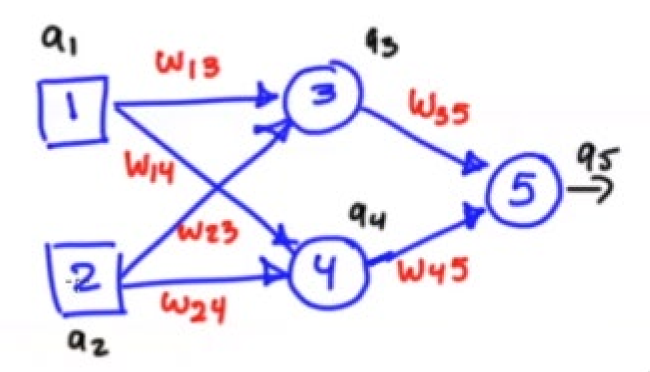
n

	a_1	a_2	a_5	
I_1	·	·	·	O_1
I_2	·	·	·	O_2
⋮	⋮	⋮	⋮	⋮
I_n	·	·	·	O_n

Training NNs



$$\boxed{a_5} = g\left(w_{35} a_3 + a_4 w_{45}\right)$$
$$= g\Big(\underline{w_{35}}\, g\left(\underline{w_{13}}\boxed{a_1} + \underline{w_{23}}\boxed{a_2}\right) + \underline{w_{45}}\, g\left(w_{14}\boxed{a_1} + \underline{w_{24}}\boxed{a_2}\right)\Big)$$



Loss function is used to find **optimal weights**:

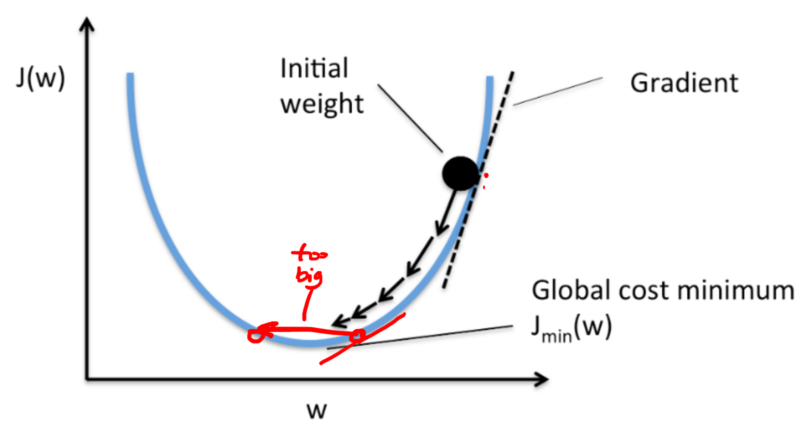
- Cross Entropy (CE)
- Mean Square Error (MSE):
  - MSE = $\frac{1}{n}\sum_{i=1}^{n}(NN(I_i) - O_i)^2$ ←
  - $NN(I_i)$ is $\mathbf{f_i}(\boldsymbol{w_1}, \boldsymbol{w_2}, \dots, \boldsymbol{w_k})$

	a_1	a_2	a_5
I_1	.	.	. O_1
I_2	.	.	. O_2
$\vdots$		$\vdots$	$\vdots$ $\vdots$
I_n	.	.	. O_n

Training NNs

Loss function is what we want to optimize:

- Mean Square Error (MSE):
  - MSE = $\frac{1}{n}\sum_{i=1}^{n}(NN(I_i) - O_i)^2$
  - $NN(I_i)$ is $\mathbf{f_i}(\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_k})$



Gradient Descent (GD) is often used to find weights that optimize loss functions

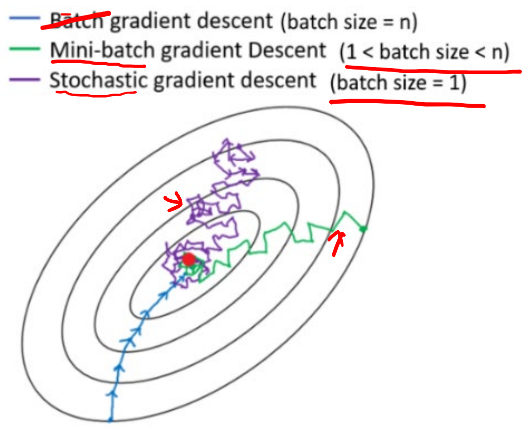
- Partial derivatives: $(\frac{\delta f}{\delta w_1}, \frac{\delta f}{\delta w_2}, \ldots, \frac{\delta f}{\delta w_k})$ → this vector is called **gradient**
- If step size is too big, we may miss the optimal value
- GD has many variations (e.g. Adam optimizer), can have different step sizes, etc.
- Calculation proceeds backwards through the network → **backpropagation**

Source of image: http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/

Training NNs

data < 80% train < 80% train, 20% validation
20% test

{ 10-fold c.v.
5-fold c.v.


~~Batch~~ gradient descent (batch size = n)
Mini-batch gradient Descent (1 < batch size < n)
Stochastic gradient descent (batch size = 1)

Other important concepts about training NNs:

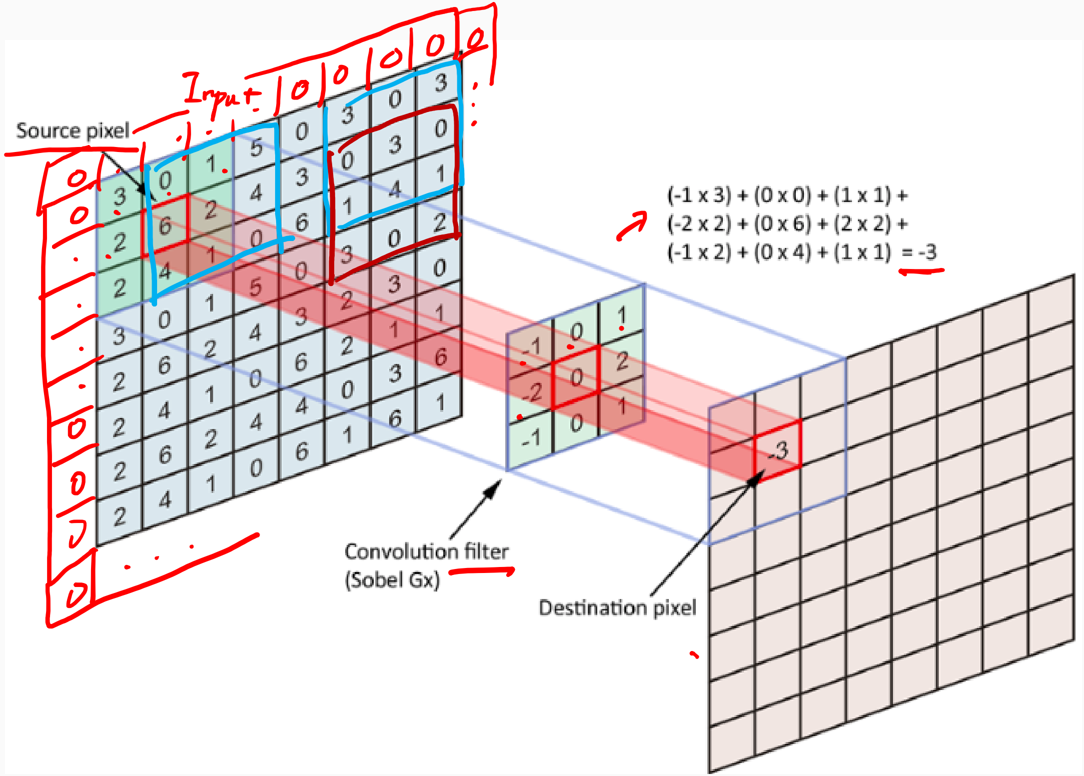
- Performance metric: the **accuracy** obtained from test or validation data
- Train / validation / test data:
  - Test data: should never be used or seen in the training step
  - Validation data: normally part of the training data, may be used for cross-validation
- Epoch: one iteration of gradient descent(s) GD: ↓[train loss, metric from validation set   SGD: ↓ E E E mean loss, metric
  - Classical GD goes through all training examples to compute MSE = $\frac{1}{n}\sum_{i=1}^{n}(NN(I_i) - O_i)^2$
  - **Stochastic GD (SGD)** goes through single examples or batches to compute MSE → more efficient
- Batch (mini-batch): a subset of the training data → no need to compute MSE on entire dataset
  - Often divided randomly → better generalize & performance
  - Batch size is a hyper-parameter (e.g. 32, 64, 128)
- Stopping criteria: can based on # of epochs, loss and performance metrics (using validation data) → overfitting

Source of image: https://datascience.stackexchange.com/questions/52884/possible-for-batch-size-of-neural-network-to-be-too-small

Convolutional Neural Network (CNN)

Convolution layer:

- Padding: add additional boundaries (of 0's) to source pixels to preserve dimensions
- Filter size: e.g. f=3 for 3 x 3 filter
- Stride: how big are the steps of the filter in each move
- Channels: multiple # for colored images
- Output of convolution will often be passed through a non-linear activation function like ReLU



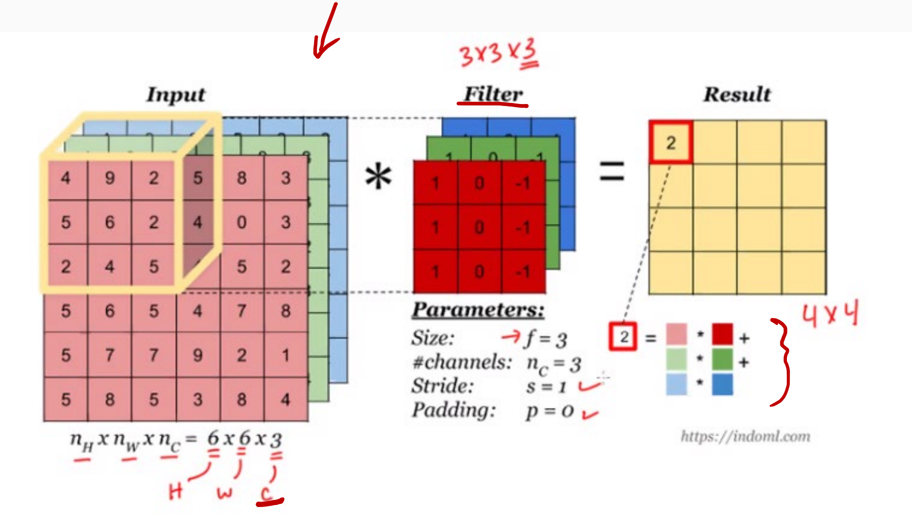
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

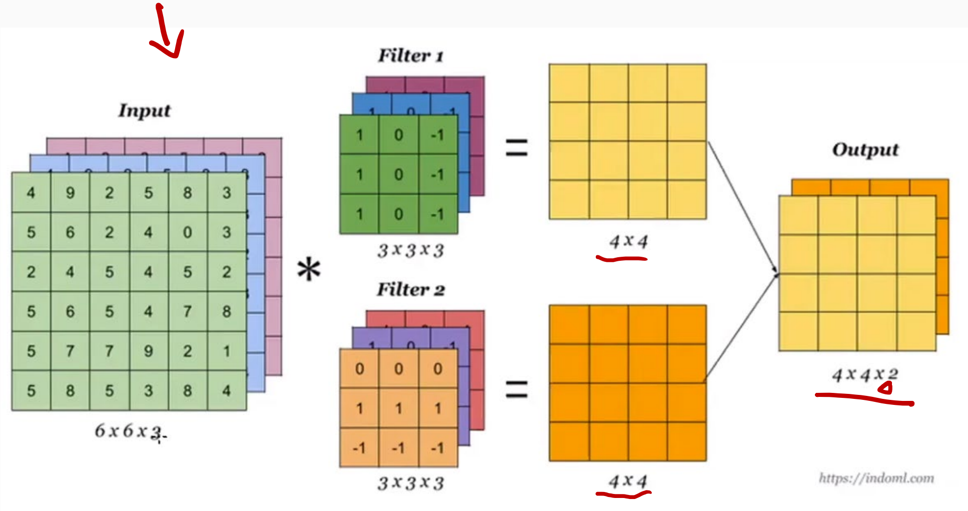
Source of image: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

CNN - Filter

Same filter can be stacked to apply on colored images (multiple channels)

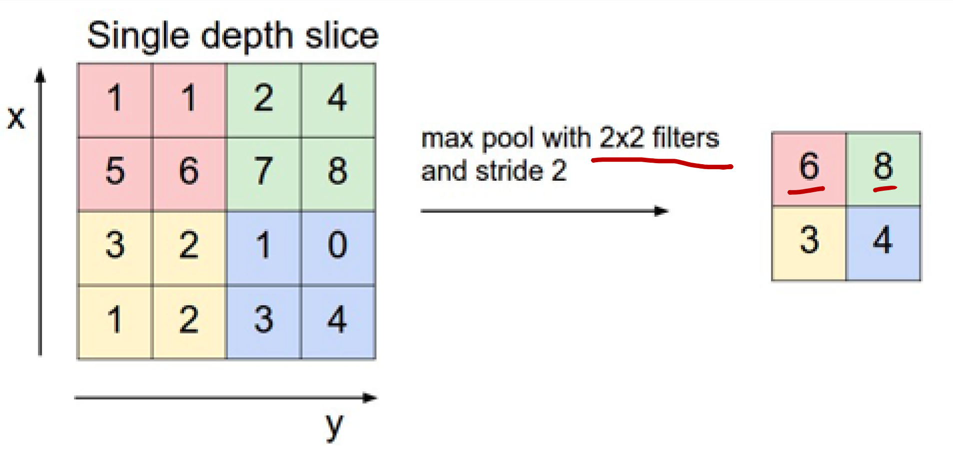
Multiple filters can be applied on one image to extract different features





CNN – Max Pooling

Max pooling layer:



Single depth slice

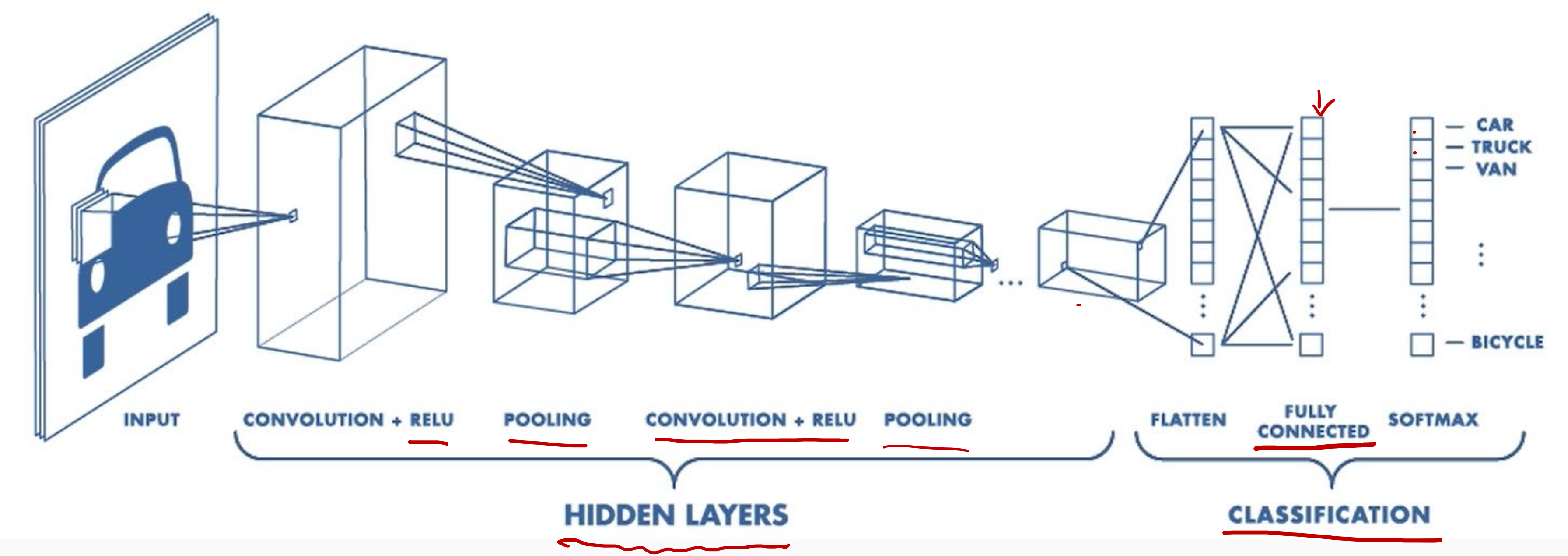
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

6	8
3	4

Source of image: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/

CNN Architecture



Source of image: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

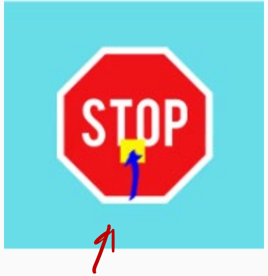
Strength & Limitation of NN

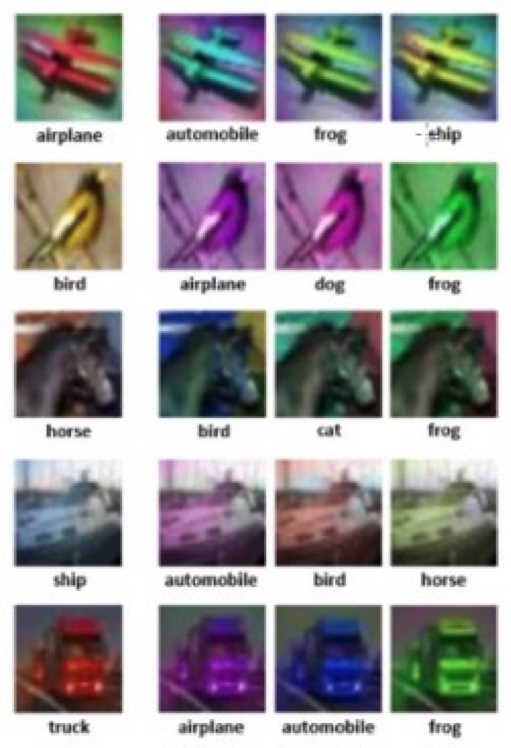
Strength:

- Universal function estimator – very expressive
- Conceptually easy to build
- Allow us to do things that couldn't do before
  - E.g. CNN for image analysis

Limitation:

- Requires huge amount of data
- Not robust
- Results are not easy to explain → needs explainable AI





Today’s Topics

- Neural Network (NN)
  - Neurons
  - Feedforward NN
  - NN as a Function
  - Training NN
  - Convolutional Neural Network (CNN)
  - Strength & Limitation of NN
- **Review of Final**

General Info

- 2 time slots on Monday, 6/8:
  - 11:30 am – 2:30 pm PDT
  - 8:00 pm – 11:00 pm PDT
- Format: on CCLE, contains only T/F and multiple-choice
  - 60 questions in total
  - Cumulative, but focus more on the content after the midterm
  - Some questions need computation but not complex
- Open book and notes, electronic ones are ok. No internet
- Go to the same Zoom link if not specifically noticed by us. More details will be announced soon

Final Review – Content before Midterm

LISP

- quote or ‘ : everything under it is kept symbolic
- nil or (): empty list
- car: first element of the list, cdr: the rest of the list (always a list)
- (**cons** arg1 arg2): reverse of car+cdr
- (**list** arg1 … argn): construct a list ‘(arg1 … argn)
- (**append** ‘(l11 l12 …) ‘(l21 l22 …) … ‘(ln1 ln2 …)): ‘(l11 l12 … l21 l22 … ln1 ln2 ...)
- predicates: atom, listp, null, equal
- (cond (cond1 value1) (cond2 value2) … (condn valuen))
- (let ((var1 value1) … (varn valuem)) (expression))
  - let: parallel assignment, let*: sequential assignment

Final Review – Content before Midterm

LISP

- (defun functionName (arg1 … argn) (expression))
- Calling a function: (functionName arg1 … argn)
- General form for a lisp recursion function

```
(defun functionName (arg1 … argn)
        (cond
                (baseCase someValue)
                (Case1 someValue/recursiveCall)
                …
                (t someValue/recursiveCall)
        )
)
```

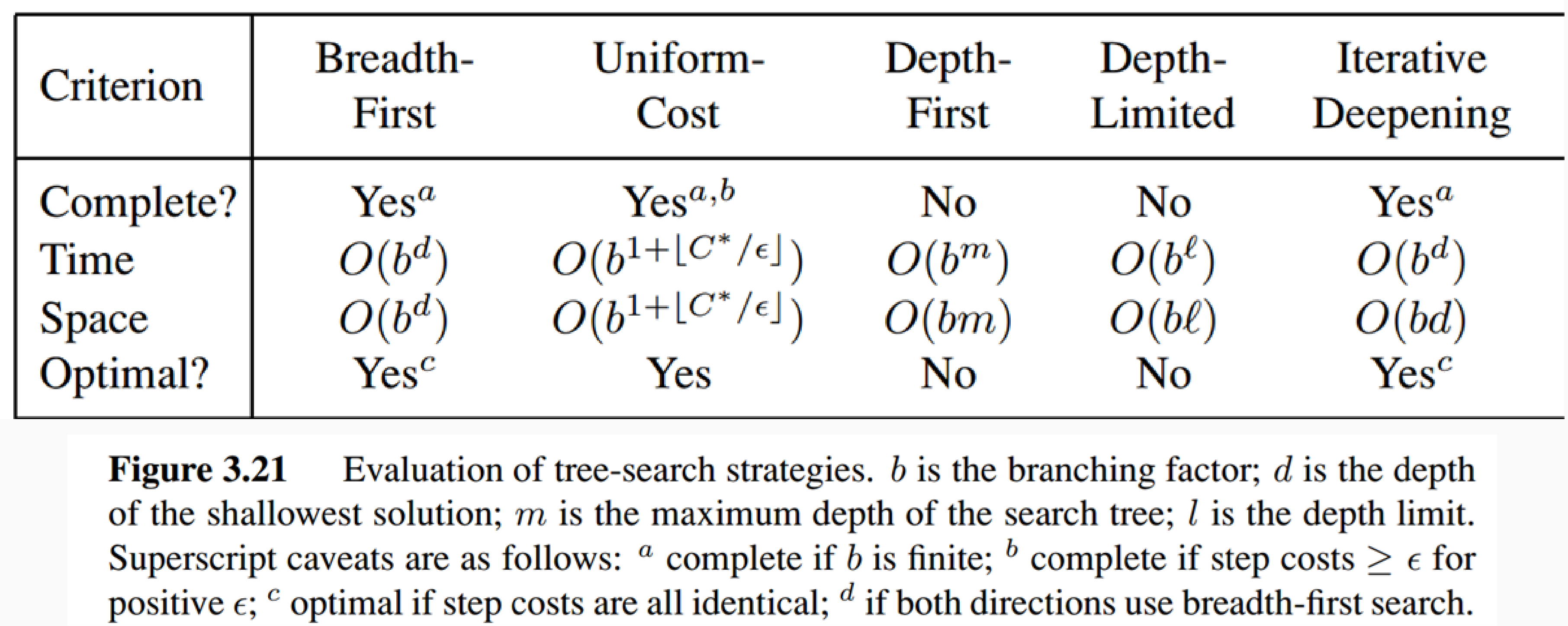
Final Review – Content before Midterm

SEARCH

- Search Problem Formulation
  - Initial state, State space, Actions, Transition model, Goal Test
  - 8 queens - complete formulation and incremental formulation
- State space and search tree
- Solution
  - A path from initial state to goal state
- Node **generation** and **expansion**
- Fringe
  - Nodes to expand. Keep in memory
- **Properties** of search strategies
  - Completeness, Optimality, Time complexity, Space complexity
- Tree search and graph search
  - Graph search maintains an “explored” set and does not re-expand states

Final Review – Content before Midterm

Uninformed Search: Properties



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes^a	Yes^{a,b}	No	No	Yes^a
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$
Optimal?	Yes^c	Yes	No	No	Yes^c

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

Final Review – Content before Midterm

Uninformed Search:

- **Breadth-first search**: expands the shallowest nodes first
  - Complete, optimal for unit step costs, exponential space complexity.
- **Uniform-cost search**: expands the node with lowest path cost
  - Complete, optimal
- **Depth-first search**: expands the deepest unexpanded node first.
  - Neither complete nor optimal, but has linear space complexity.
- **Depth-limited search:** adds a depth bound to DFS
- **Iterative deepening search**: calls depth-first search with increasing depth limits until a goal is found.
  - Complete, optimal for unit step costs, time complexity comparable to breadth-first search, linear space complexity.

Final Review – Content before Midterm

Informed Search:

- Informed search methods have access to heuristic function h(n)
    - Evaluate cost from node n to goal
    - **Admissible**, consistent
- **Greedy search** expands nodes with minimal h(n)
    - Not always optimal but efficient
- **A-star search** expands nodes with minimal g(n) + h(n)
    - Complete and optimal
    - Tree-search version when h is admissible
    - Graph-search version when h is consistent

Final Review – Content before Midterm

Constraint Satisfaction:

- Constraint satisfaction problem formulation
    - Variables, Domains, Constraints
- **Backtracking DFS**
    - Fail and backtrack when a consistent assignment is not possible
- **Heuristics**: increase the efficiency of backtracking DFS
    - Variable selection
        - Most constraint variable / Minimum Remaining Values heuristic
        - Most constraining variable / Degree heuristic
    - Value selection
        - Least constraining value

Final Review – Content before Midterm

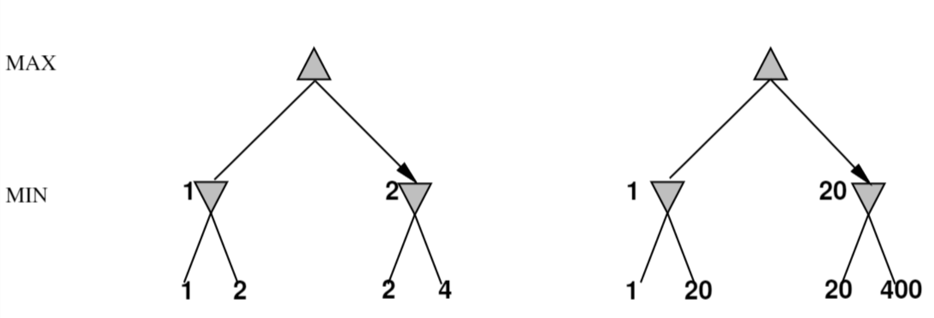
Constraint Satisfaction:

- **Constraint propagation**
  - Node consistency and arc consistency
  - AC-3 (push all the arcs)
  - Forward checking (variable-level arc consistency)
- **Problem Structure**
  - Tree-structured CSP

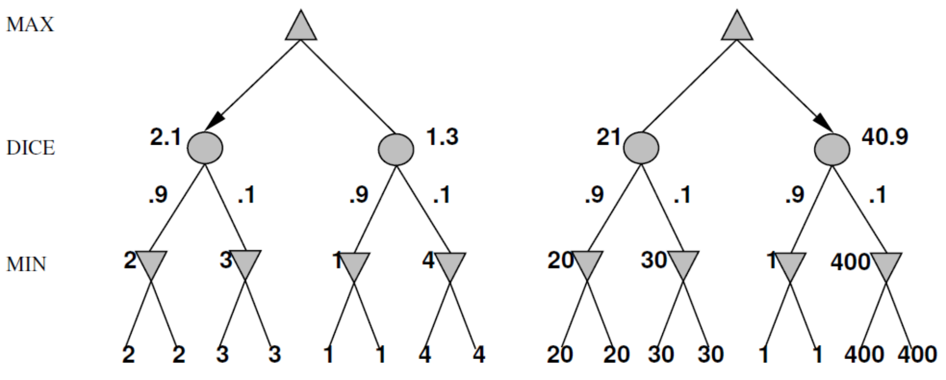
Final Review – Content before Midterm

Game Playing: Basics

- **Minimax**:
  - ○ a utility value for all goal states (leaves)
  - ○ max player: value is max of its children
  - ○ min player: value is min of its children
  - ○ value of the root: the value of the game outcome



- **Expected-minimax**:
  - ○ calculate the expectation over children



Final Review – Content before Midterm

Game Playing: Alpha-beta pruning

- Motivation: skip branches that won't matter to improve efficiency
- **A generic algorithm**:
  - ○ During the DFS search, each node carries an lower bound α and an upper bound β.
  - ○ Pushing bound upward: when a child returns, it pushes its value onto the parent (**always tighten the bound**). Min-child pushes onto max-parent's α. Max-child pushes onto min-parent's β. (How to remember: Max player will modify its lower bound, and min player will modify its upper bound.)
  - ○ Pushing bound downward: right before analyzing a child, parent pushes its bound (**Both α and β**) onto that child.
  - ○ Prune all unsearched children when parent has α >= β
- A website for alpha-beta pruning practice:

  http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

Final Review – Content before Midterm

Propositional Logic:

- Syntax and semantics
- Important terms: model, satisfiability, validity, entailment, etc.
- **Syntactic forms**: CNF, DNF, Horn clauses, NNF, DNNF
    - All but horn clauses are <u>universal</u>. DNF, horn and DNNF are <u>tractable</u>
- **Propositional Inference**:
    - Inference rules
    - Method 1: Proof by enumeration - model checking: E.g. Using a truth table
    - Method 2: Proof by refutation (resolution):
        - Step 1: Convert KB to CNF
        - Step 2: Keep applying inference rules until an empty clause appear

          → Showing $\Delta \wedge \neg\alpha$ is unsatisfiable!
    - Method 3 (SAT solver) and 4 (NNF circuits) won't be covered in the midterm.

List of Topics

Lisp and search strategies

1. Evaluate a simple LISP expression or function, or choose a sentence to complete it.
2. Understand differences among search algorithms and determine completeness, optimality, time, and space complexity for any of them.
3. Understand backtracking DFS and heuristics (variable order, value order, etc.) in constraint satisfaction problems.
4. MINIMAX and α-β pruning.

List of Topics

Propositional logic (PL) and first-order logic (FOL):

5. The concepts in PL and FOL, e.g. satisfiability, validity, entailment, consistency.
6. Translate English to FOL sentences, or the other way around.
7. Convert a propositional or first-order logic sentence to CNF. Perform Skolemization.
8. Apply resolution or other inference rules to PL/FOL sentences. Completeness and soundness of inference rules.
9. Find unifiers for two FOL sentences.
10. Decide whether a propositional or first-order sentence entails another sentence.

Reasoning over uncertainty:

11. Independence, conditional independence. Bayes rule.
12. Given background information, compute probabilities for events. → use Bayesian rule
13. Compute probability for PL sentences given possible worlds.

List of Topics

Bayesian Network:

14. Model a problem as a Bayesian network.

15. Identify Markovian assumptions encoded by a Bayesian network (its semantics). Give joint probability using the chain rule.

16. Utilize d-separation to identify independence.

→ 3 types of valves, relationship between d-separation and independence

Machine Learning and Neural Network:

17. Concepts about Machine learning. Supervised learning and unsupervised learning.

→ with complete or incomplete dataset, learning BN parameters and BN structures

18. Definition for Entropy. Choose splitting attributes for a decision tree.

19. Concepts about Neural Network. Given input and NN structure, predict output.