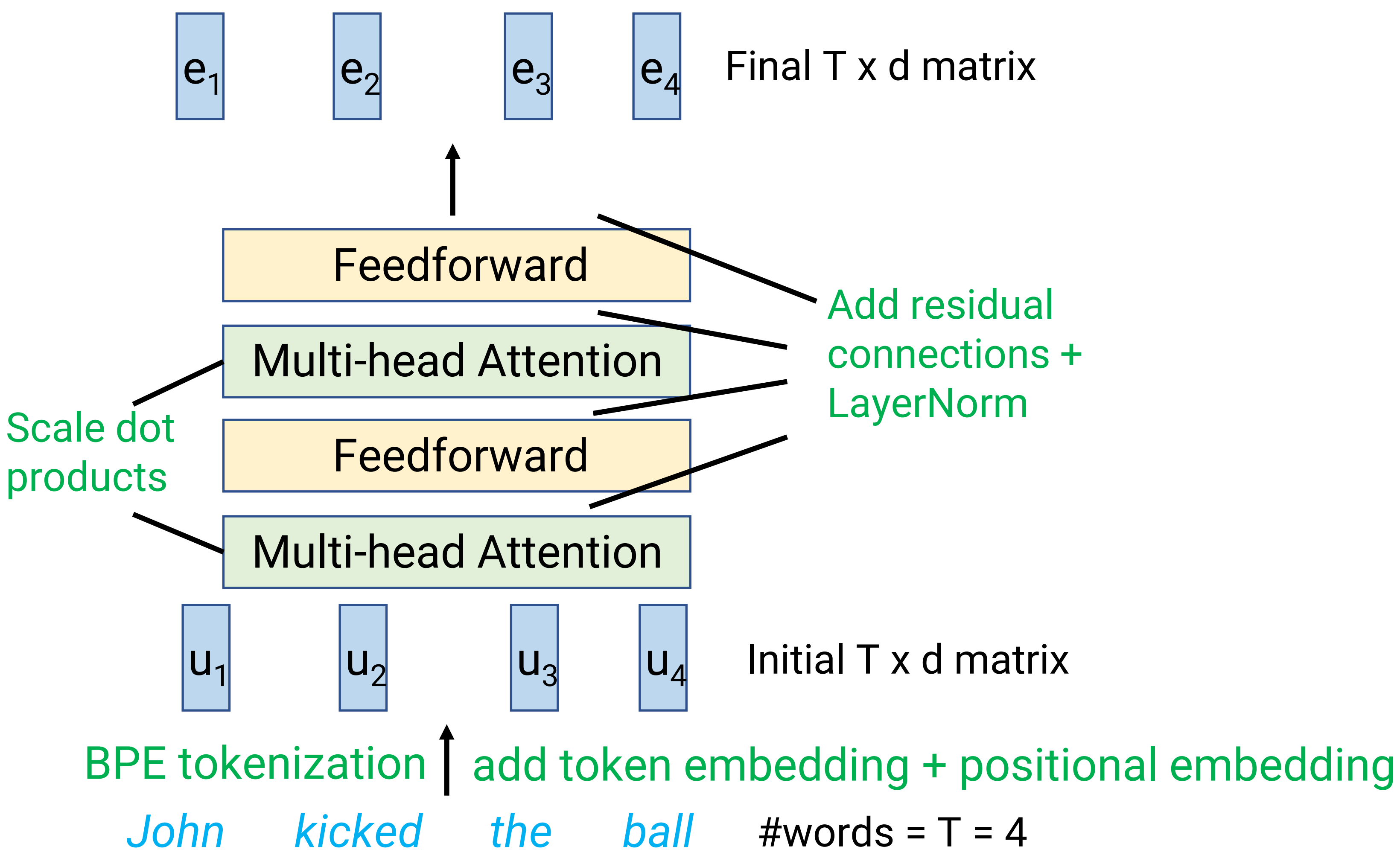
Pretraining Neural Networks; Decision Trees, Ensembles

Robin Jia
USC CSCI 467, Fall 2023
October 24, 2023

Review: The Full Transformer



- Main Components
  - Multi-head Attention
  - Feedforward layers
- Also includes
  - Positional embeddings
  - Byte pair encoding
  - Scaled dot product attention
  - Residual connections between layers
  - LayerNorm

Review: Transformer Decoder

- How to do autoregressive language modeling?
- Modify multi-headed attention so that each token can only attend to previous/current token(s)
  - Test time: Model generates tokens one at a time
  - Training time: Can compute predictions for every token in sequence in parallel (fast!)

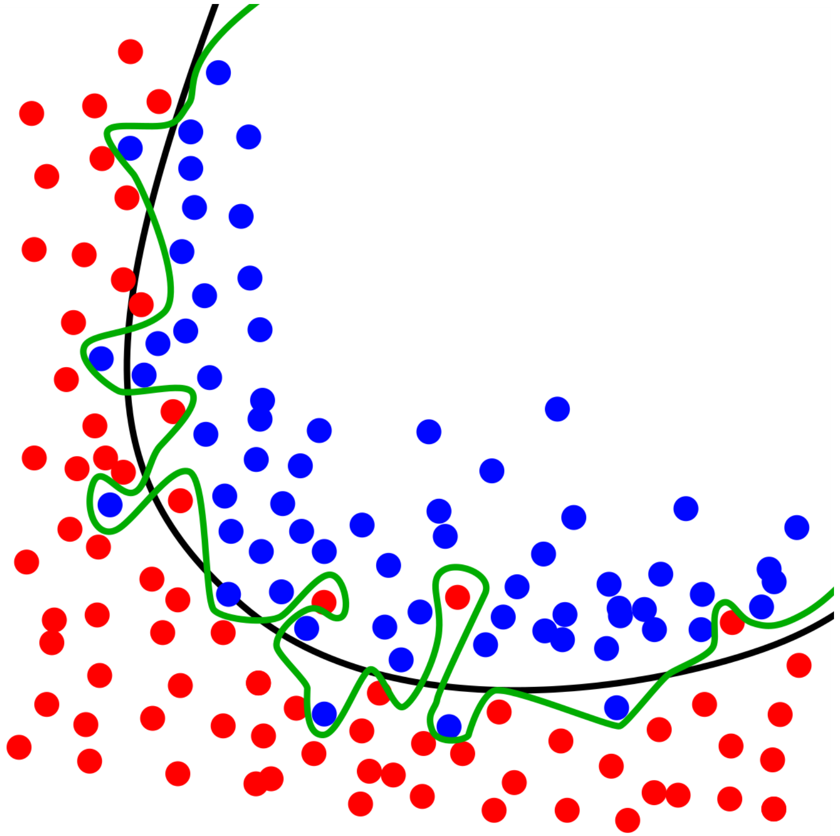
Queries

	[BEGIN]	John	kicked	the
[BEGIN]	10	$-\infty$	$-\infty$	$-\infty$
John	0	7	$-\infty$	$-\infty$
kicked	-3	4	5	$-\infty$
the	2	1	7	6

Keys

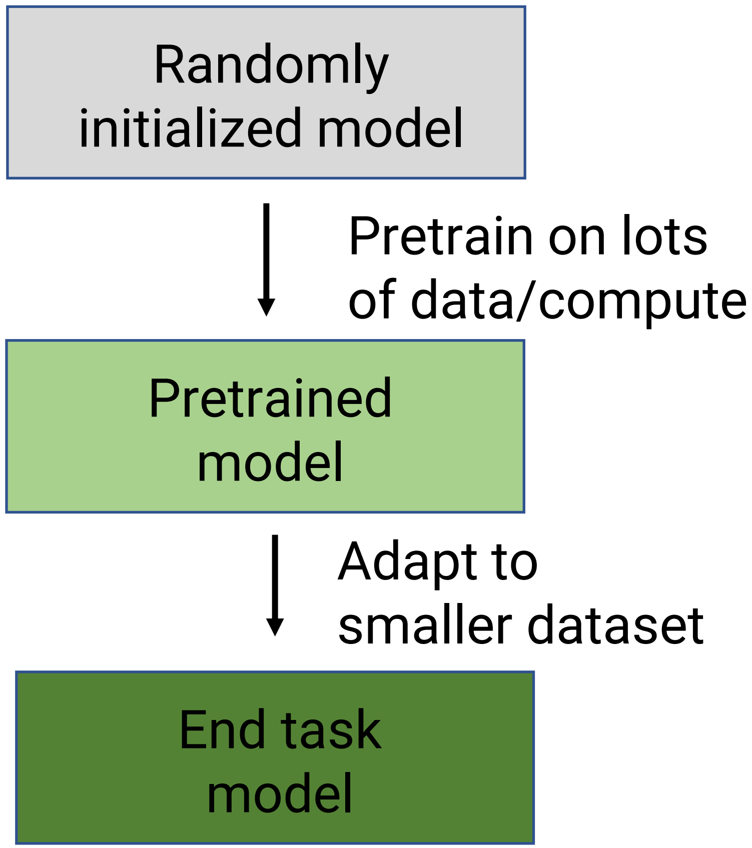
Neural Networks and Scale

- Deep learning models (e.g. Transformers) are nothing without training data!
- Neural networks are very expressive, but have tons of parameters
  - Very easy to overfit a small training dataset
- Traditional view: Neural Networks are flexible but very **"sample-inefficient"**: they need many training examples to be good
  - Reason: Low bias but high variance because many ways to perfectly fit the training data



Pretraining

- Neural networks learn to extract features useful for some training task
  - The more data you have, the more successful this will be
- If your training task is very general, these features may also be useful for other tasks!
- Hence: **Pretraining**
  - First pre-train your model on one task with a lot of data
  - Then use model's features for a task with less data
  - Upends the conventional wisdom: You can use neural networks with small datasets now, if they were pretrained appropriately!



ImageNet Features

Faces

Dogs (eyes?)

Red ornaments/ flowers

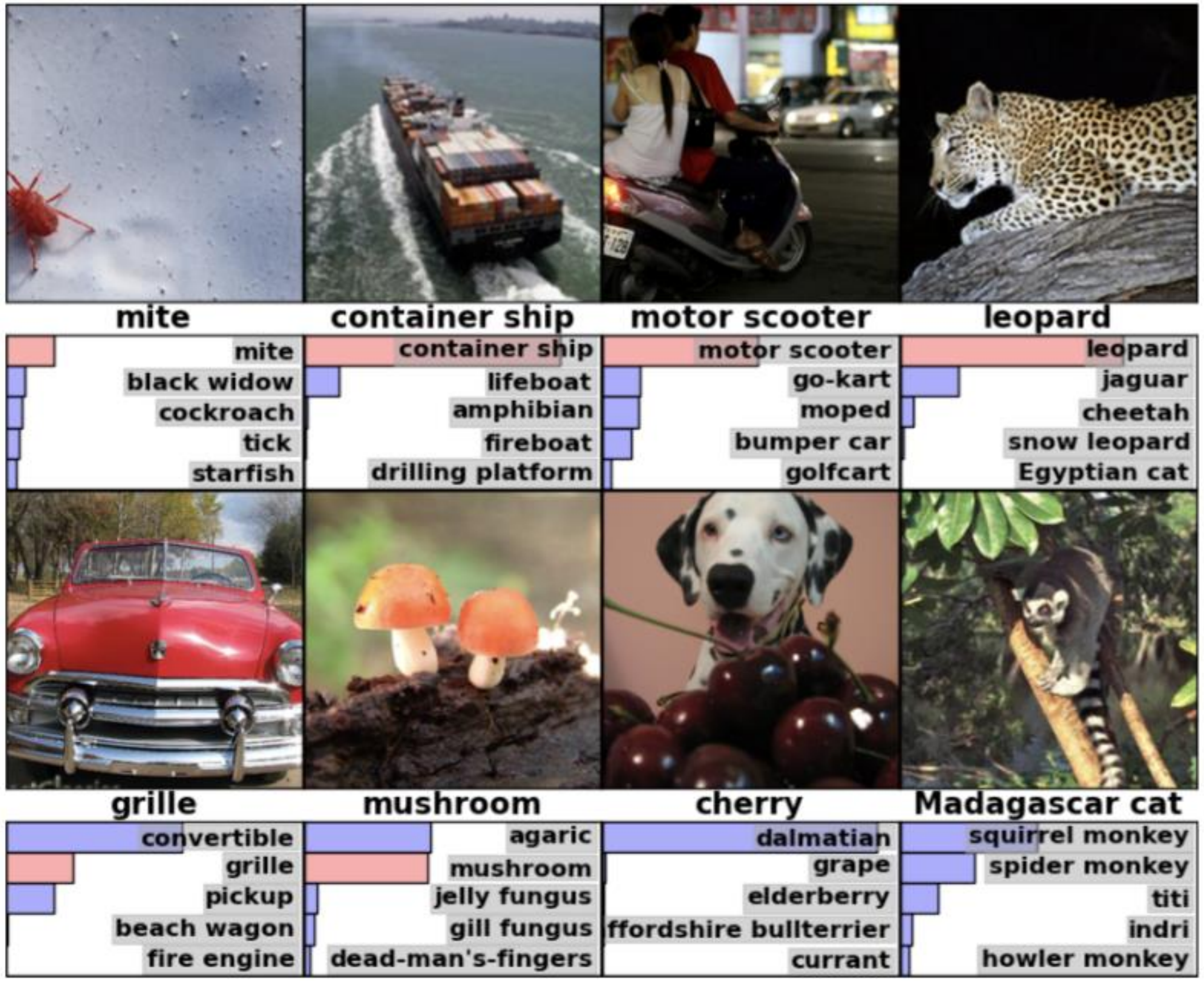
Text (years?)

Houses

Lens flare?

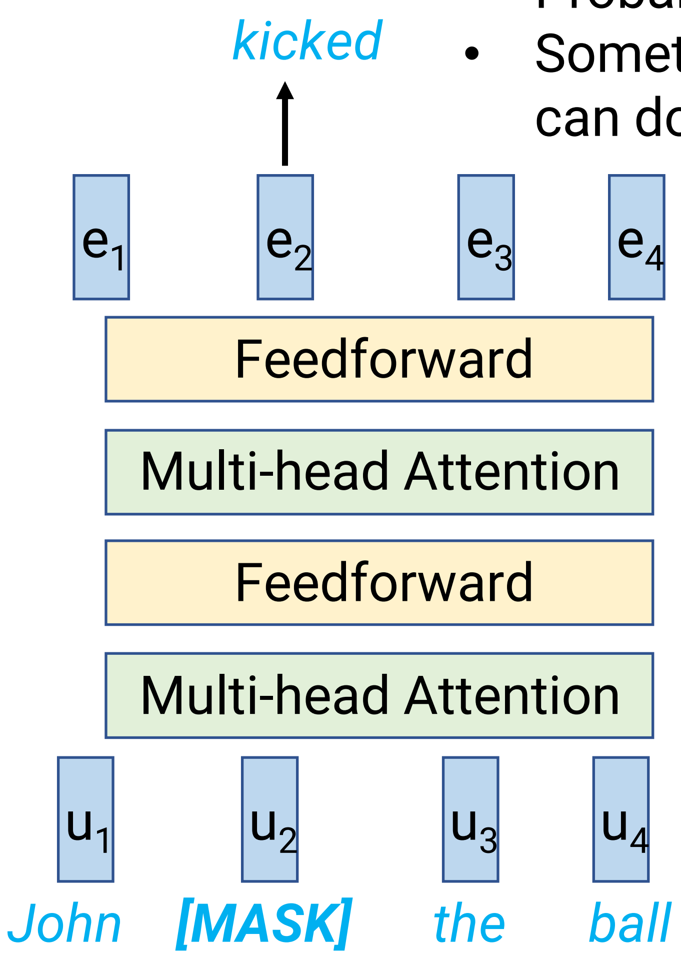
Features learned by AlexNet trained on ImageNet

ImageNet Features



- ImageNet dataset: **14M** images, 1000-way classification
- Most applications don't have this much data
- **But the same features are still useful**
- Using "frozen" pretrained features
  - Get a (small) dataset for your task
  - Generate features from ImageNet-trained model on this data
  - Train linear classifier (or shallow neural network) using ImageNet features

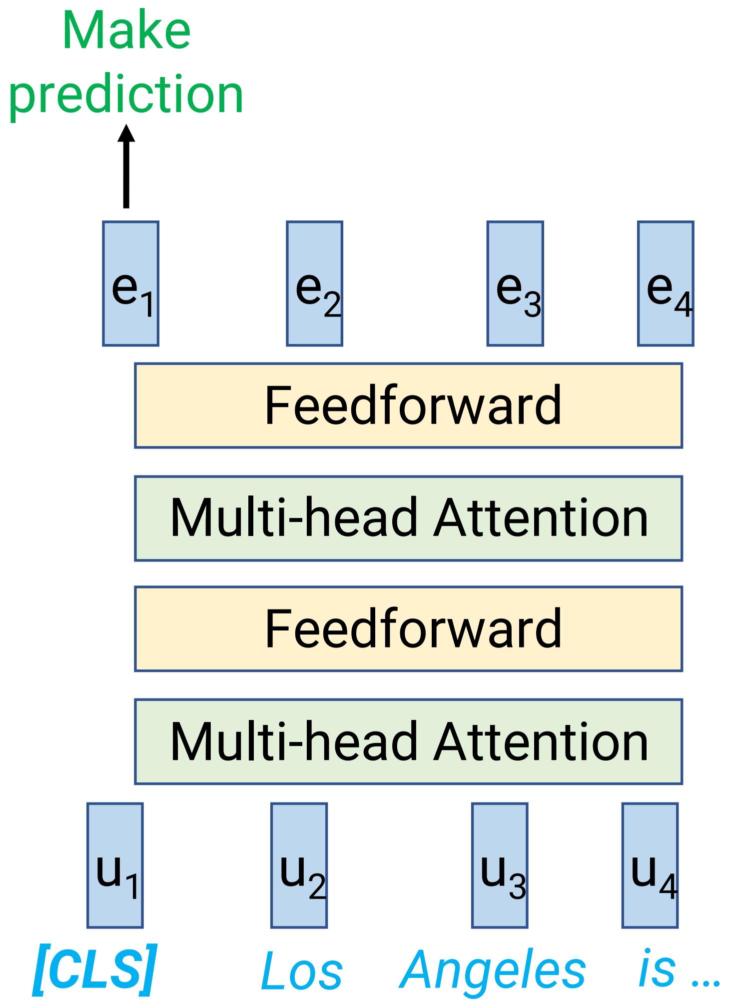
Masked Language Modeling (MLM)



- Probably a verb
- Something a person can do to a ball

- MLM: Randomly mask some words, train model to predict what's missing
  - Doing this well requires understanding grammar, world knowledge, etc.
  - Get training data just by grabbing any text and randomly delete words
  - Thus: Crawl internet for text data
- Transformers are good fit due to scalability
  - Large matrix multiplications are highly optimized on GPUs/TPUs
  - Don't need lots of operations happening in series (like RNNs)
- Most famous example: BERT

Fine-tuning



- Initialize parameters with BERT
  - BERT was trained to expect every input to start with a special token called [CLS]
- Add parameters that take in the output at the [CLS] position and make prediction
- Keep training all parameters ("fine-tune") on the new task
- Point: BERT provides very good initialization for SGD

What about ChatGPT and GPT-4???

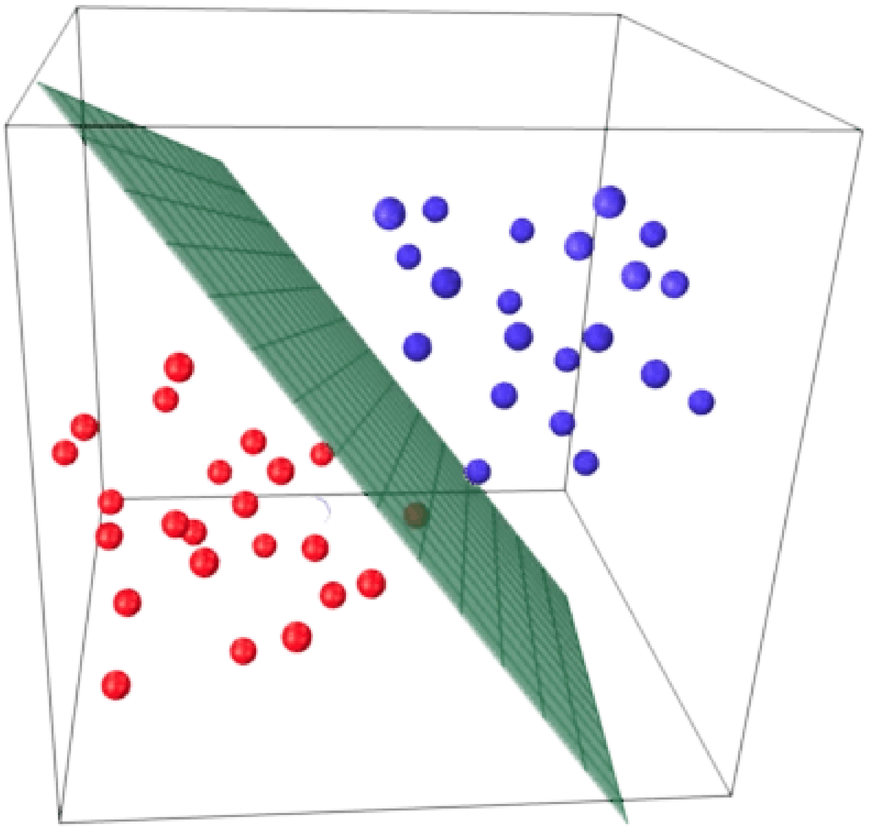
- ChatGPT appears to be a fine-tuned language model
  - Pretrained on autoregressive language modeling
  - Then fine-tuned with a method called RLHF (reinforcement learning from human feedback)
  - We'll return to this when we talk about reinforcement learning!
- GPT-4 is rumored to be an *ensemble* of many similar Transformer models
  - More an ensembling in a few minutes

Announcements

- Project progress reports due October 31
- HW2 grades released, solutions on blackboard
- HW3 to be released shortly
- Thursday: We start unsupervised learning, back to iPad + lecture notes

Previously: Reliance on Linear Layers

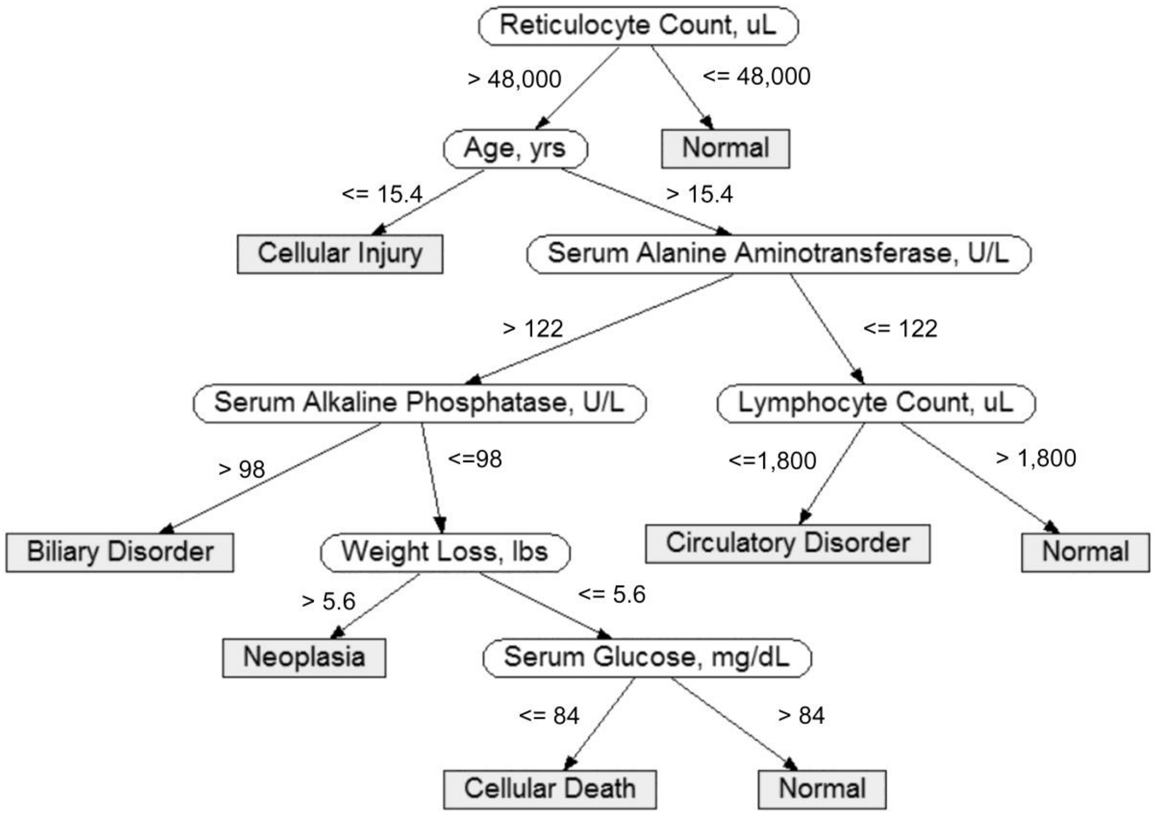
- Linear models
  - Linear regression, logistic regression, softmax regression
  - Classification: Decision boundary is defined by
    $$w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b = 0$$
  - Note: Combination of ***every*** feature x_i
    - Not necessarily how humans make decisions
    - Can be hard to understand why a prediction was made
- Neural networks
  - Linear layers are core building blocks
  - Final decision boundary is linear function of learned features



Modeling decision making

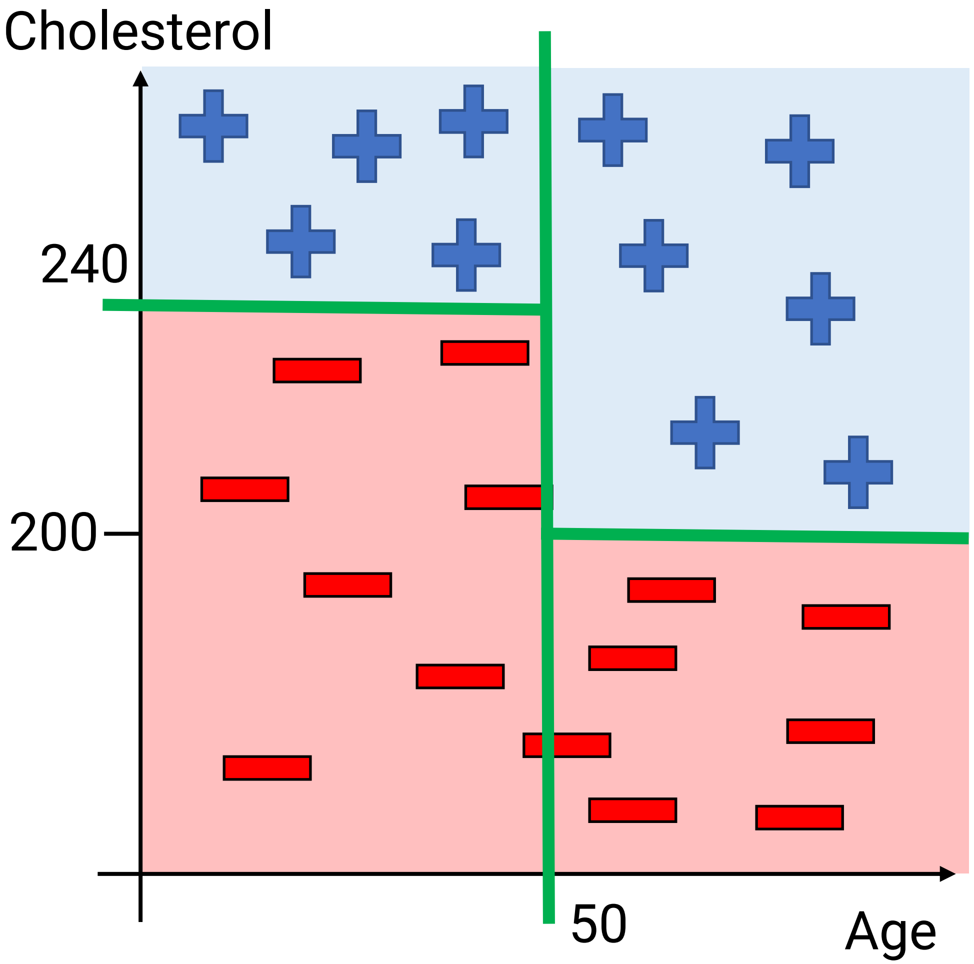
- Human experts make complex decisions and predictions every day
  - E.g., Given observations about a patient, what disease do they have?
- Doesn't really look like a linear function; more like a flow chart
- Can we build models that emulate the human decision-making process?

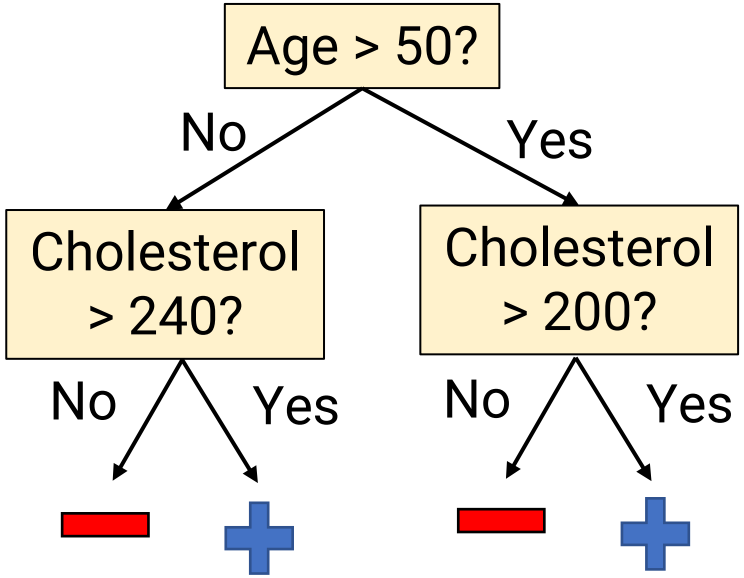
Hepatic Disorders Decision Tree



Decision Trees

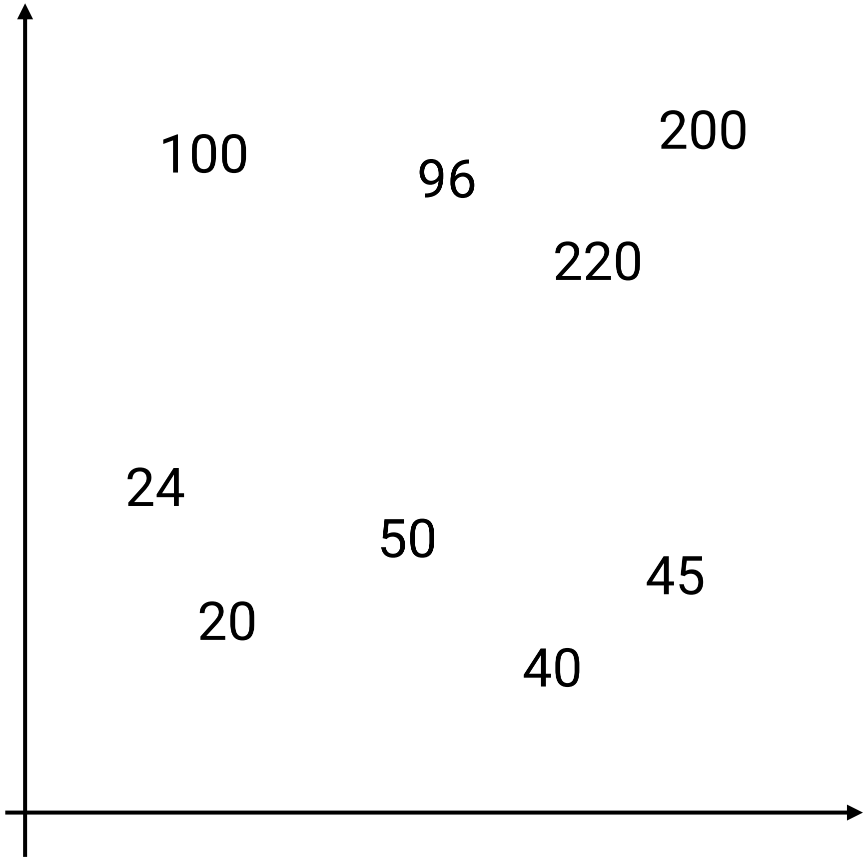
- At each node, split on one feature
- Remember the best output at each leaf node
  - Classification: Majority class
  - Regression: Mean within node
- Given new example, find which leaf node it belongs to and predict the associated output
- Interpretable!





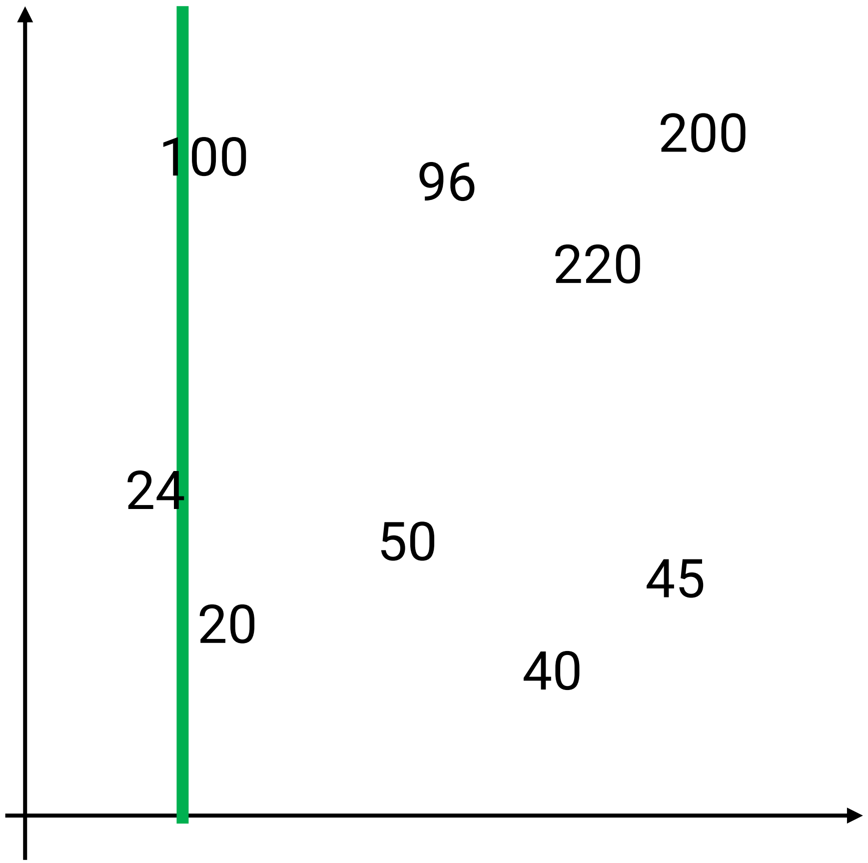
Learning Decision Trees for Regression

- At each node, decide:
  - Which feature to use
  - Which threshold to split on
- Strategy
  - Try each feature and all possible splits
  - Greedily choose split that minimizes error



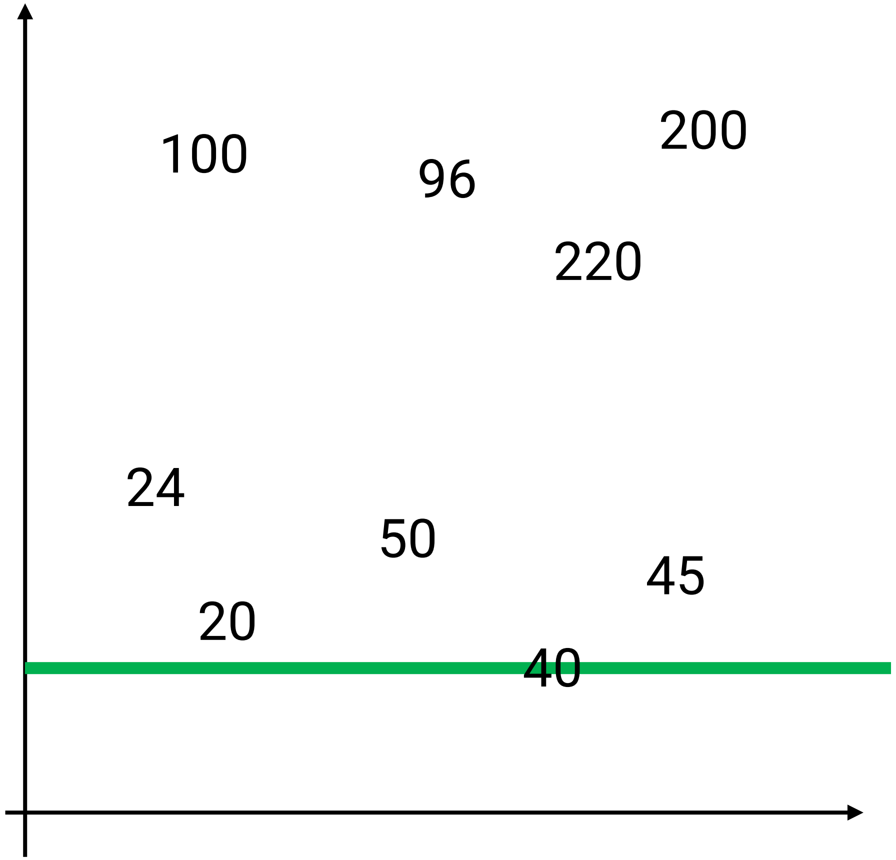
Learning Decision Trees for Regression

- At each node, decide:
  - Which feature to use
  - Which threshold to split on
- Strategy
  - Try each feature and all possible splits
  - Greedily choose split that minimizes error



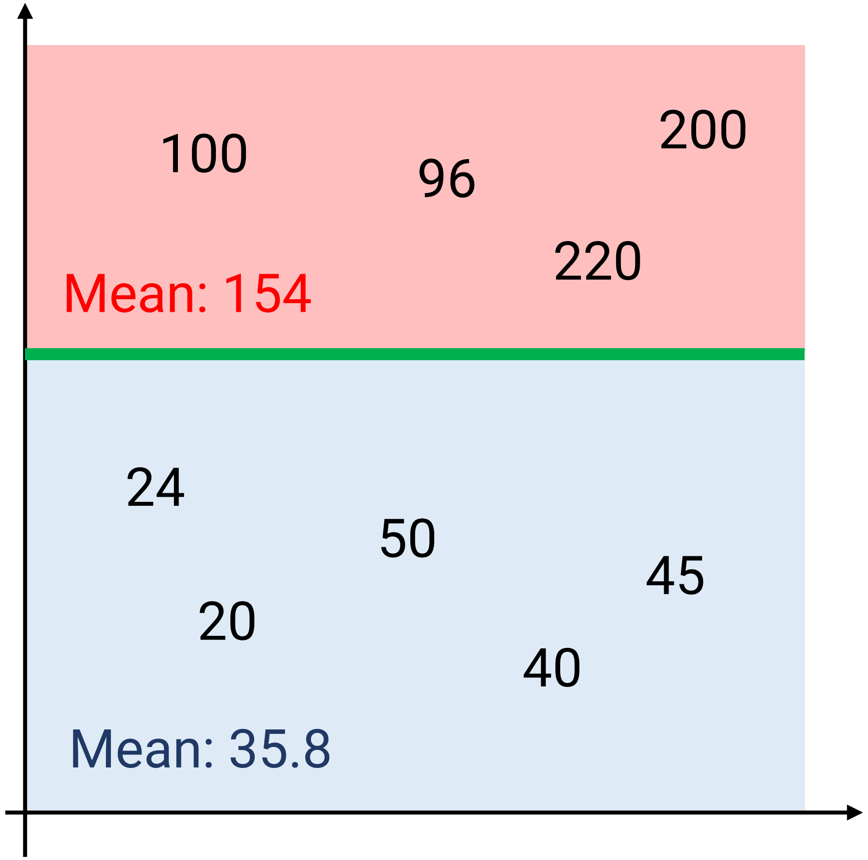
Learning Decision Trees for Regression

- At each node, decide:
  - Which feature to use
  - Which threshold to split on
- Strategy
  - Try each feature and all possible splits
  - Greedily choose split that minimizes error



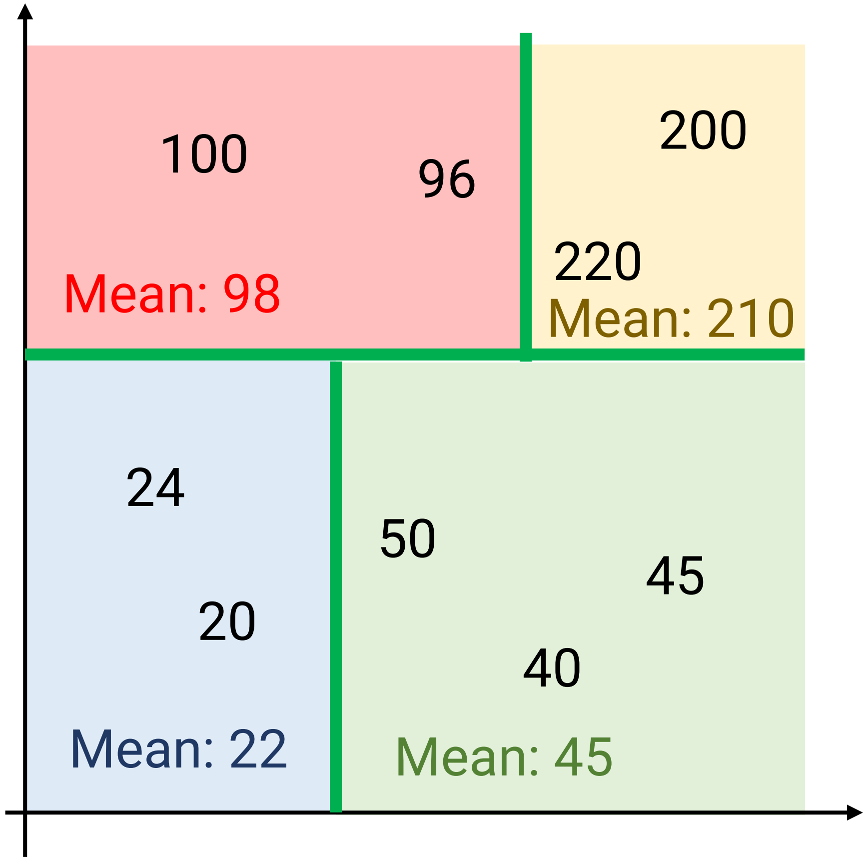
Learning Decision Trees for Regression

- At each node, decide:
  - Which feature to use
  - Which threshold to split on
- Strategy
  - Try each feature and all possible splits
  - Greedily choose split that minimizes error



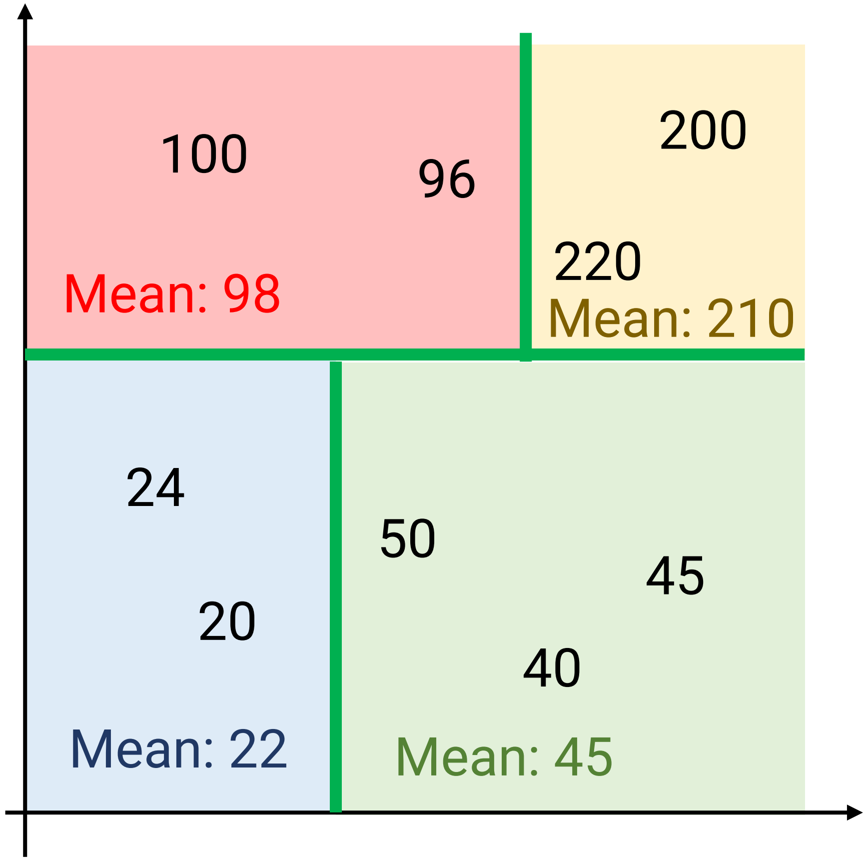
Learning Decision Trees for Regression

- At each node, decide:
  - Which feature to use
  - Which threshold to split on
- Strategy
  - Try each feature and all possible splits
  - Greedily choose split that minimizes error



Learning Decision Trees for Regression

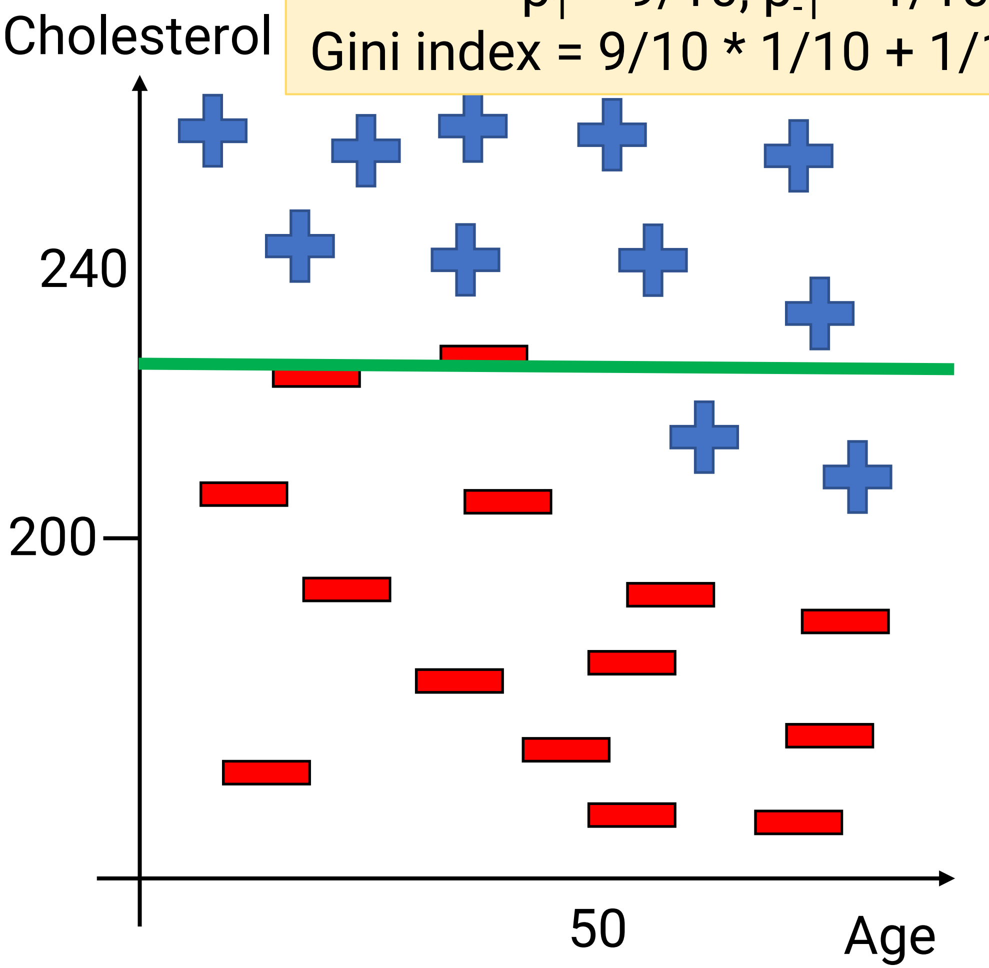
- When do we stop splitting?
  - If we split forever to nodes of size 1, we overfit
  - Heuristic stopping criteria
    - Minimum number of examples per node
    - Maximum depth of tree
  - Can go back afterwards and “prune” tree (i.e., merge nodes back together)



Learning decision trees for classification

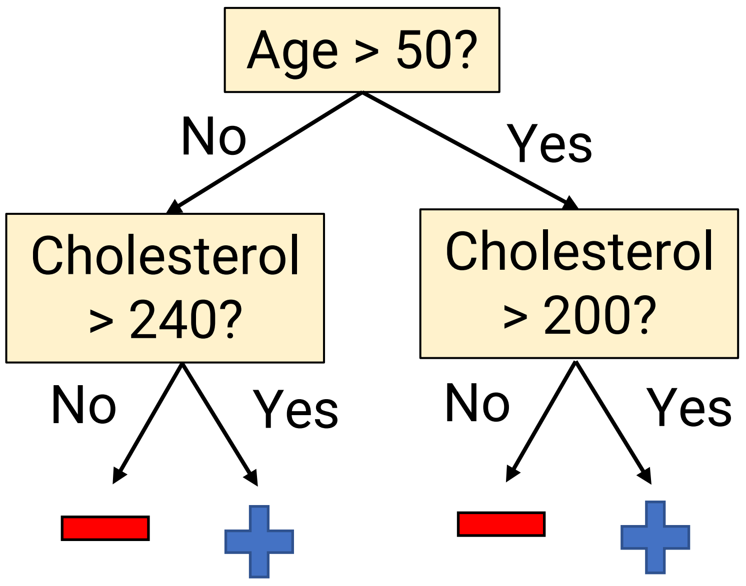
- Basic idea is the same
- But how do we measure the goodness of a split?
  - Option 1: Accuracy of majority classifier
  - Option 2: Gini index $\sum_{c=1}^{C} p_c(1 - p_c)$
    - p_c = Empirical probability of class c within the current node
    - Equals expected number of errors if you classify with the empirical distribution

$p_1 = 9/10$, $p_{-1} = 1/10$
Gini index = $9/10 * 1/10 + 1/10 * 9/10$

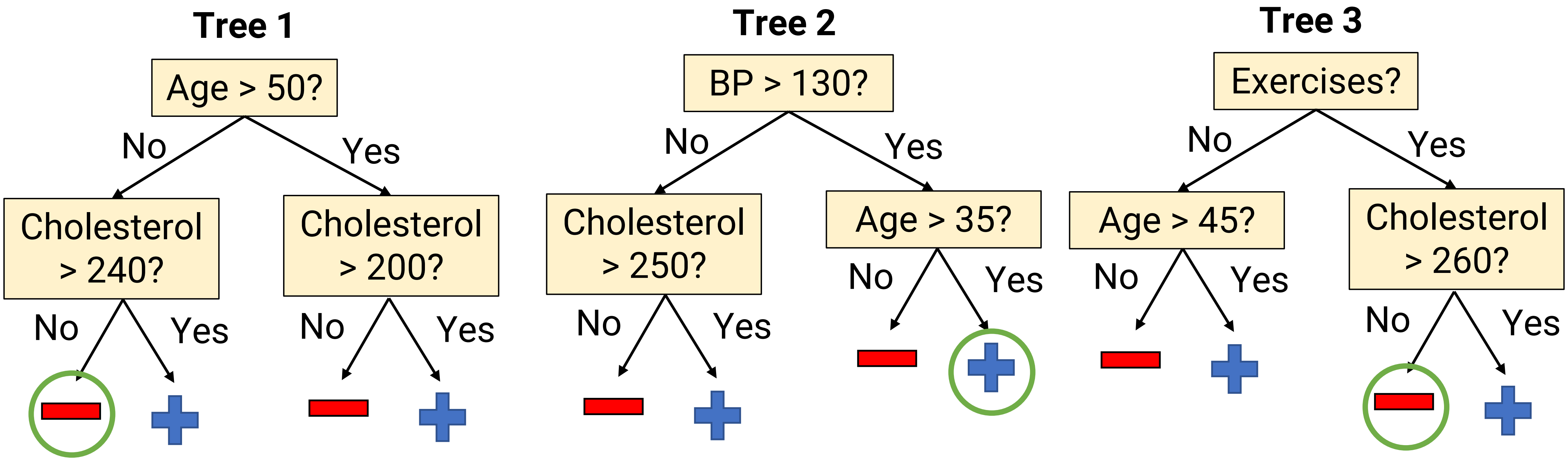


Handling Missing Features

- Some examples may be missing some features
  - E.g., For some patients, you didn't measure cholesterol level
  - What to do at a node where you split on cholesterol?
- Idea: Surrogate variables
  - During training, at each node, check which features act as **surrogates** of the feature you're using (i.e., lead to similar splits)
  - If original feature is missing, use a surrogate feature
  - E.g., If "blood pressure > 130" is correlated with "Cholesterol > 240", use blood pressure as surrogate for patients without cholesterol measurement

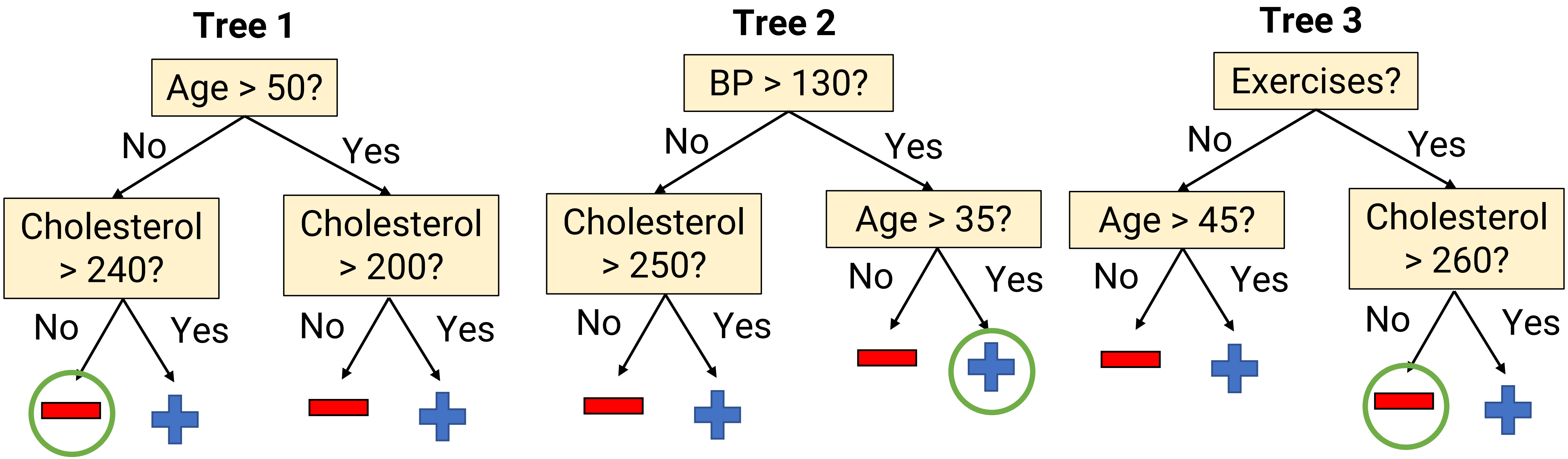


Ensembling



- Create an "ensemble" of multiple models (e.g., multiple trees)
- Make final prediction by averaging/majority vote

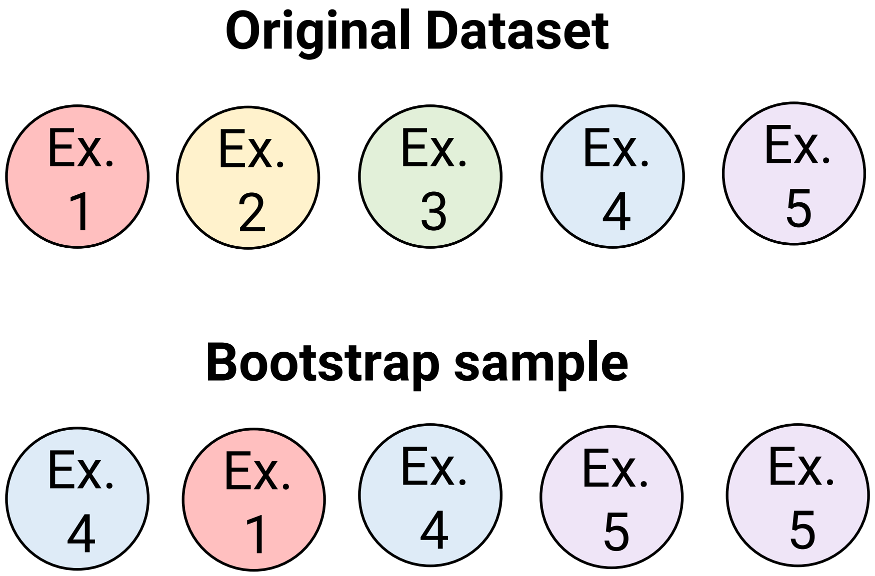
Ensembling and Trees



- An individual tree can capture complex patterns, but should not be too deep to avoid overfitting
- Thus it can only depend on a handful of features
- An ensemble of trees can leverage more features

Bagging

- How do you learn different trees from the same dataset?
- Idea: Randomly resample the dataset!
  - Given dataset with n examples, sample a new dataset of n examples **with replacement**
    - Also known as "Bootstrapping"
  - In expectation, each new dataset contains 63% of the original dataset, with some examples duplicated
  - Learn a tree on each resampled dataset

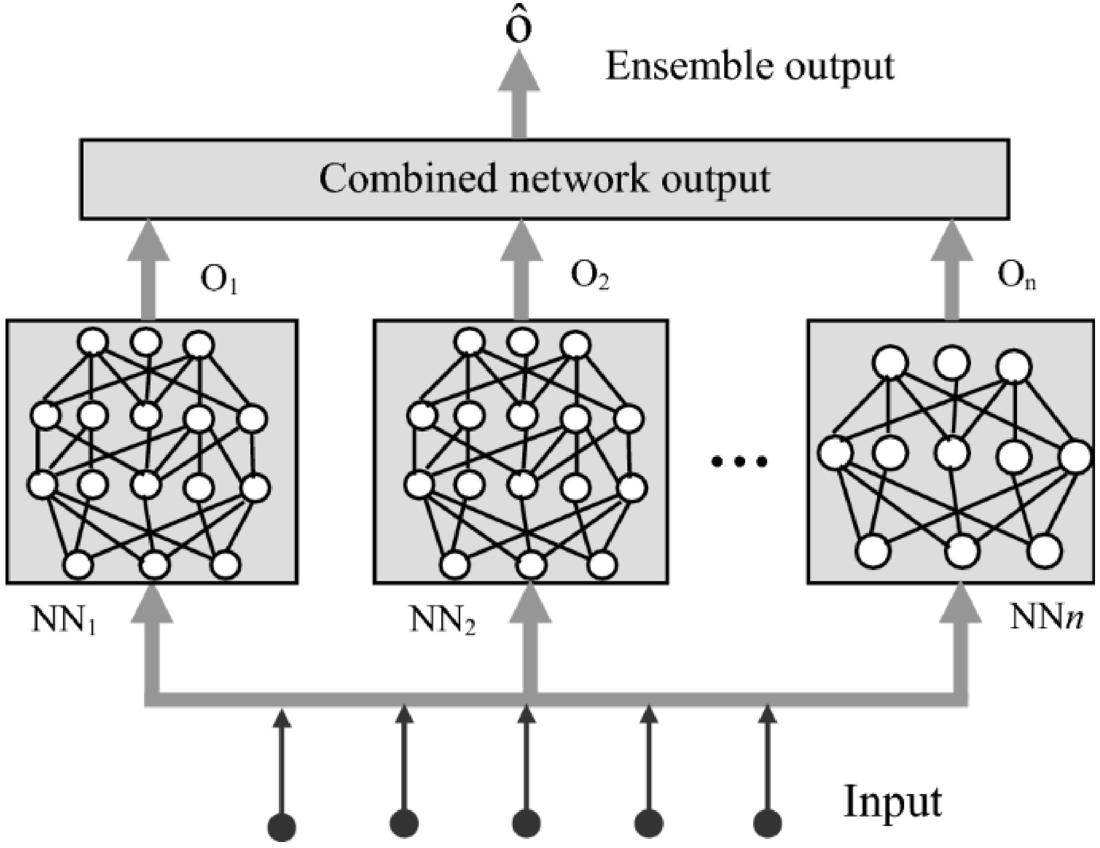


Random Forests

- Goal: Make the individual trees in the ensemble more different
  - Thus, all elements of the ensemble are complementary
- Simple strategy: Before each split, choose a random subset of features as candidates for splitting
  - Something like $\sqrt{d}$ features if *d* total features
  - Can even be randomly choosing 1 feature
- Very good general-purpose learners in practice!

Ensembles and neural networks

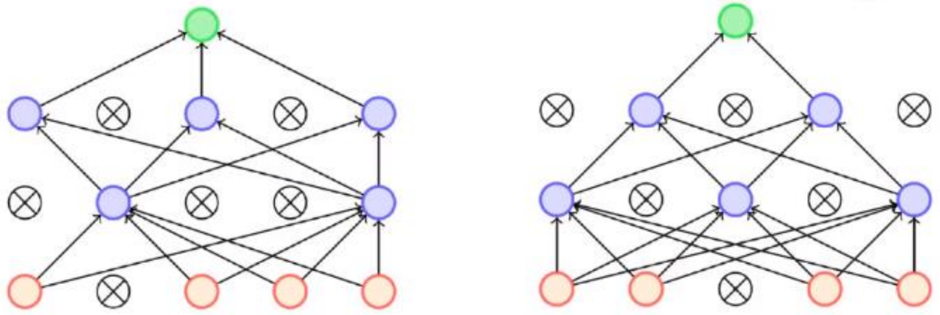
- Random Forest: Each member of ensemble differs due to random resampling of data & feature choice
- Neural Networks: Already have randomness
  - Initialization
  - Order of examples for SGD
  - Dropout
  - So, bagging is not necessary
- In practice: Very common to ensemble neural networks!
  - Compute vs. accuracy trade-off
  - Rumor: GPT-4 is an ensemble of 8 Transformers with 220 billion parameters each



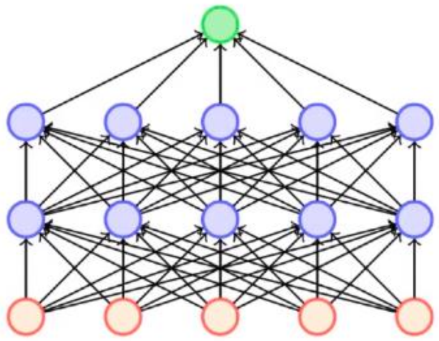
Dropout as an Ensemble

- Why does Dropout work? One explanation: **It learns a sort of ensemble**
- Training time
  - At each iteration, randomly drop out each neuron with probability *p*
  - Each iteration trains a weaker “subnetwork” instead of full network
- Test time
  - All neurons are active
  - Result is an average/ensemble of all the subnetworks
  - Note: Not exactly an ensemble in the usual sense because different subnetworks share parameters

Training time: Many “subnetworks”



Test time: Full network is average/ensemble of all subnetworks



Conclusion

- Pretraining
  - First train on large labeled or unlabeled datasets
  - Features learned are useful for other tasks with less data
- Decision trees
  - Human-interpretable decision making
  - Pairs well with ensembling, leading to random forests
- Ensembling also commonly applied to neural networks