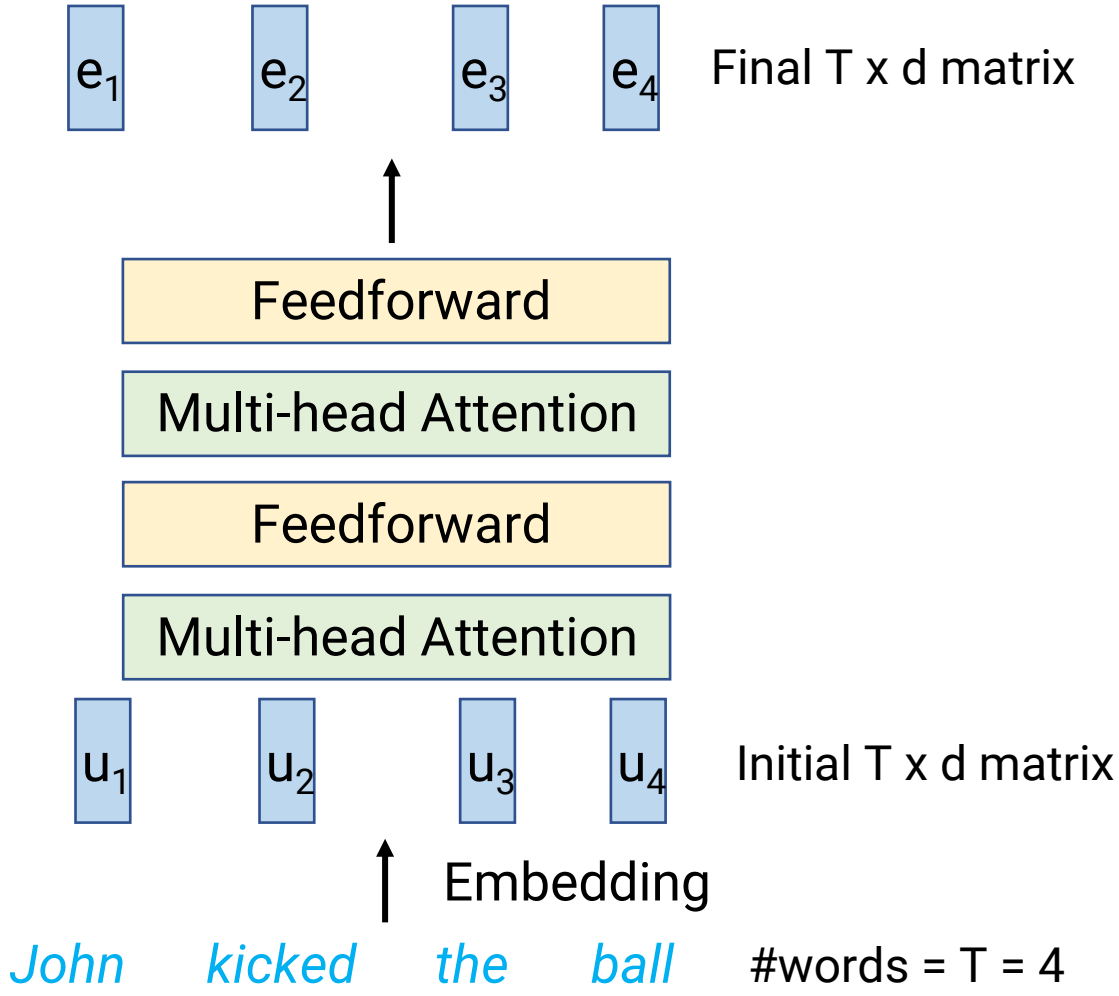
Finishing Transformers & Pretraining, Decision Trees, Ensembles

Robin Jia

USC CSCI 467, Spring 2023

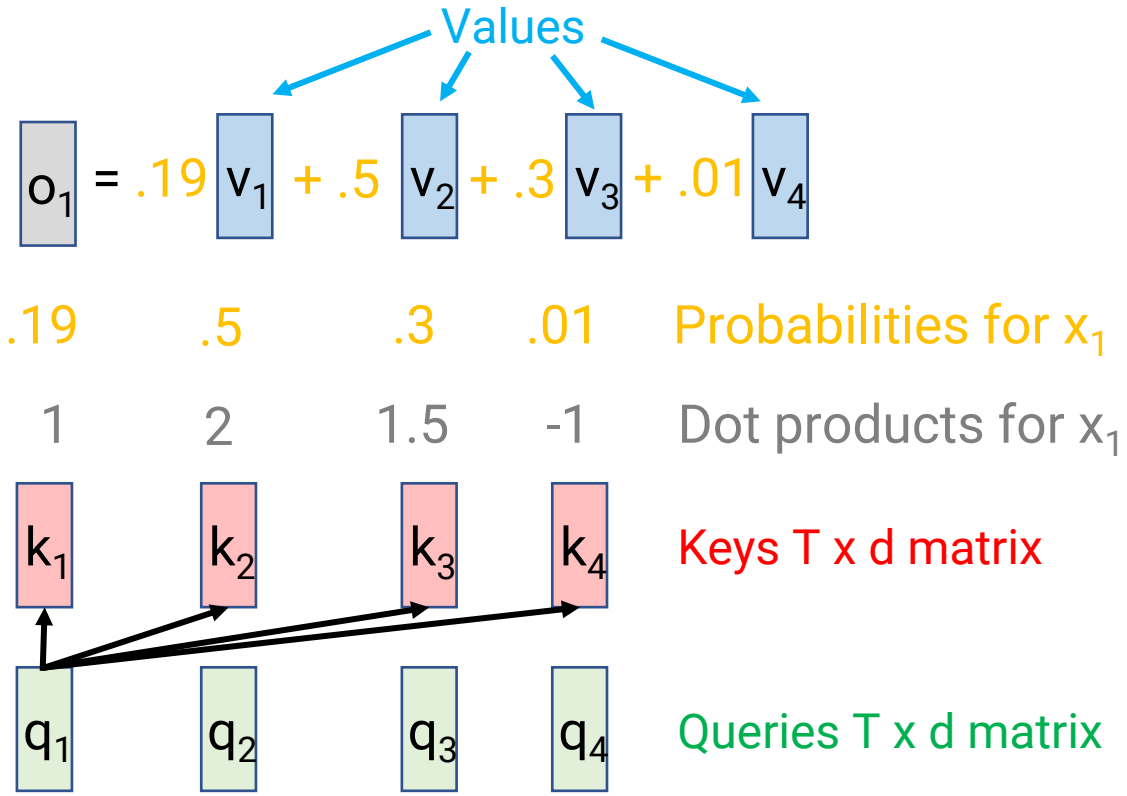
March 7, 2023

Review: Transformers



- One transformer consists of
  - Embeddings for each word of size d
    - Let T =#words, so initially T x d matrix
  - Alternating layers of
    - "Multi-headed" attention layer
    - Feedforward layer
    - Both take in T x d matrix and output a new T x d matrix
  - Plus some bells and whistles
    - Residual connections & LayerNorm
    - Byte pair encoding tokenization

Review: Multi-headed Attention

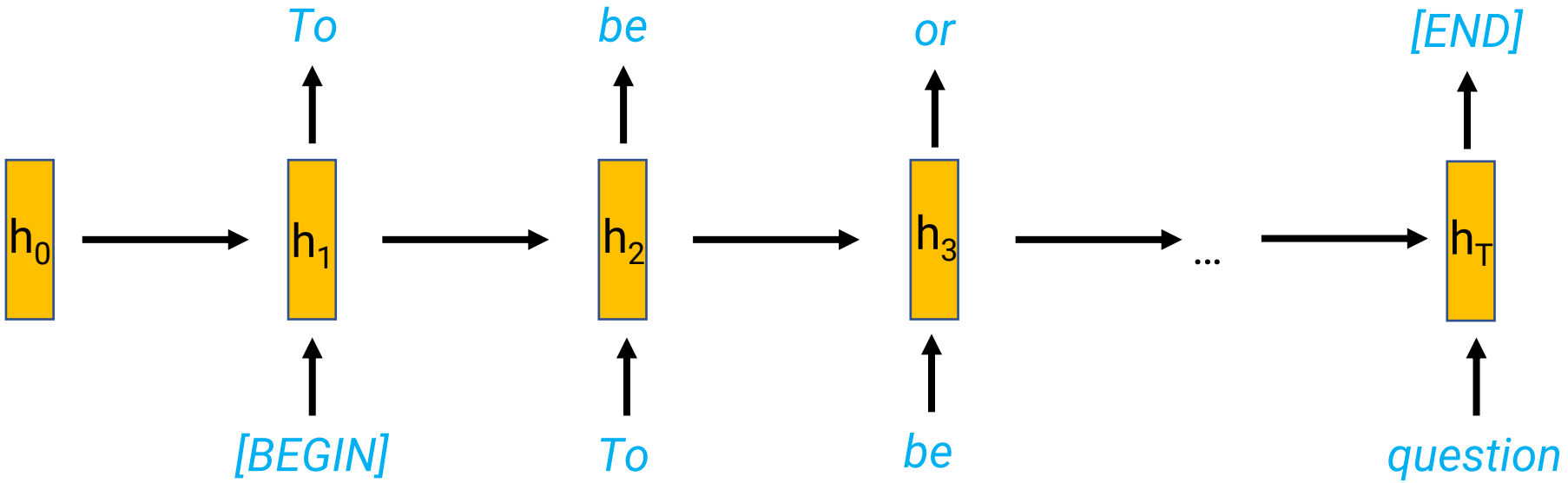


- At each head, apply 3 separate linear layers to input matrix X to get
  - Query matrix Q
  - Keys K
  - Values V
  - Each linear layer maps from dimension d to dimension d_{attn}
- Compute Q x K^T (T x T matrix)
  - Each entry is dot product of one query vector with one key vector
- Normalize each row with softmax to get matrix of probabilities P
- Output = P x V

Outline

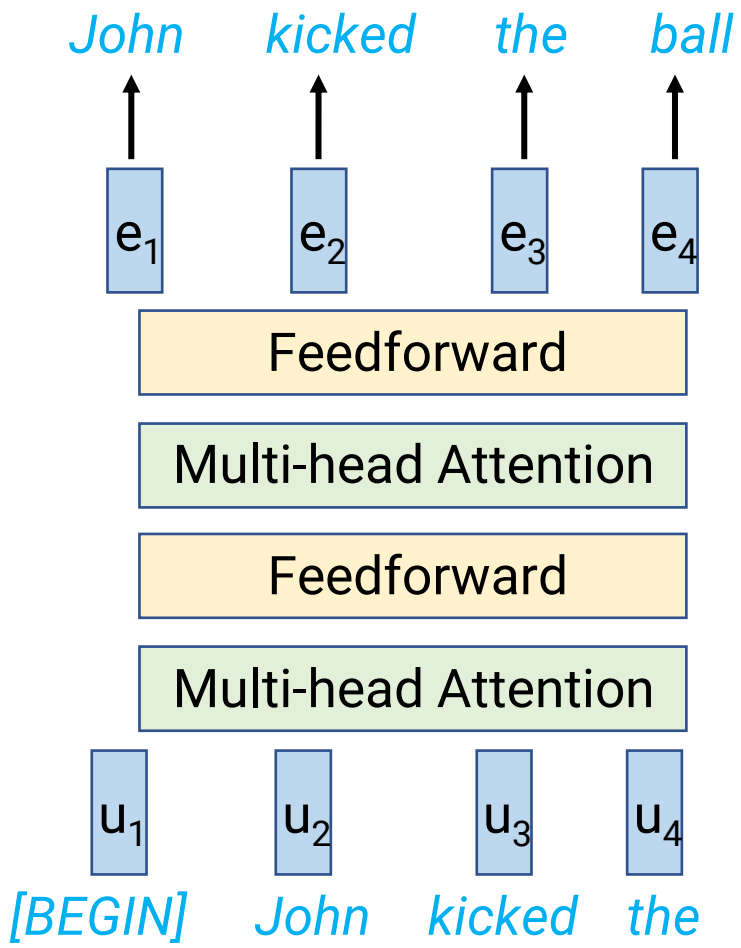
- Transformer decoders
- Pretraining
- Decision Trees and Ensemble learning

Review: RNN Decoder Language Models

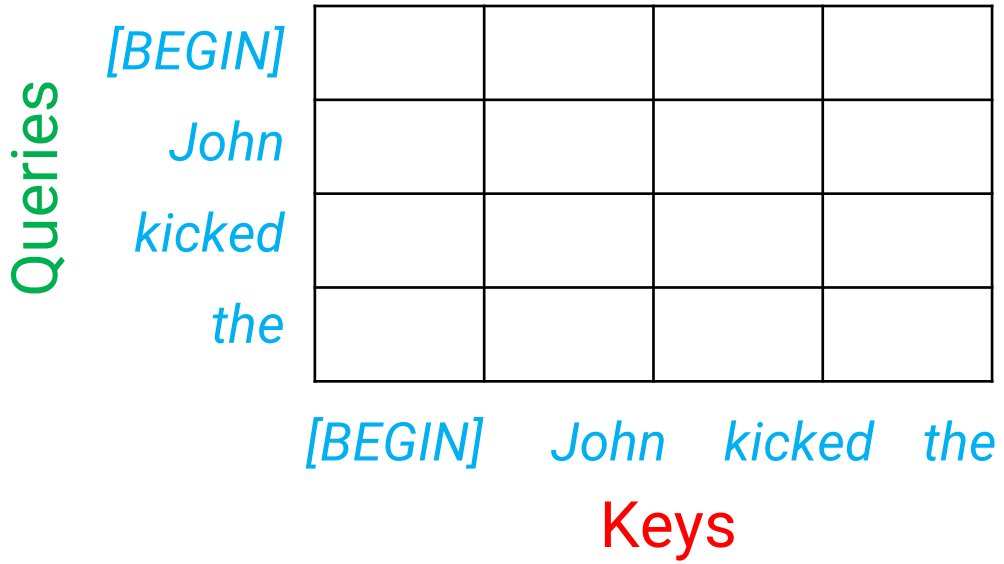


- At each step, predict the next word given current hidden state
- Must happen in series at both training and test time
  - Each hidden state depends on the previous hidden state

Transformer autoregressive decoders



- How to do autoregressive language modeling?
- Test-time
  - At time t, attend to positions 1 through t
  - Happens in series



Transformer autoregressive decoders

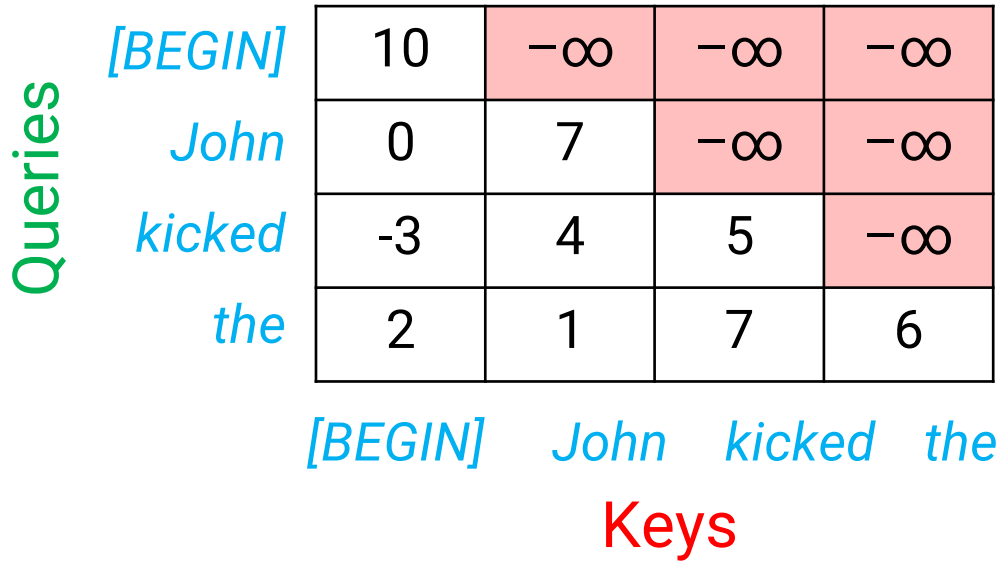
- How to do autoregressive language modeling?
- Training time: Masked attention trick
  - Recall: Attention computes Q x K^T (T x T matrix), then does softmax
  - But if generating autoregressively, time t can only attend to times 1 through t
  - Solution: Overwrite Q x K^T to be $-\infty$ when query index < key index
  - **All timesteps happen in parallel**

Transformer autoregressive decoders

- How to do autoregressive language modeling?
- Training time: Masked attention trick
  - Recall: Attention computes Q x K^T (T x T matrix), then does softmax
  - But if generating autoregressively, time t can only attend to times 1 through t
  - Solution: Overwrite Q x K^T to be $-\infty$ when query index < key index
  - **All timesteps happen in parallel**

Transformer autoregressive decoders

- How to do autoregressive language modeling?
- Training time: Masked attention trick
  - Recall: Attention computes Q x K^T (T x T matrix), then does softmax
  - But if generating autoregressively, time t can only attend to times 1 through t
  - Solution: Overwrite Q x K^T to be $-\infty$ when query index < key index
  - **All timesteps happen in parallel**

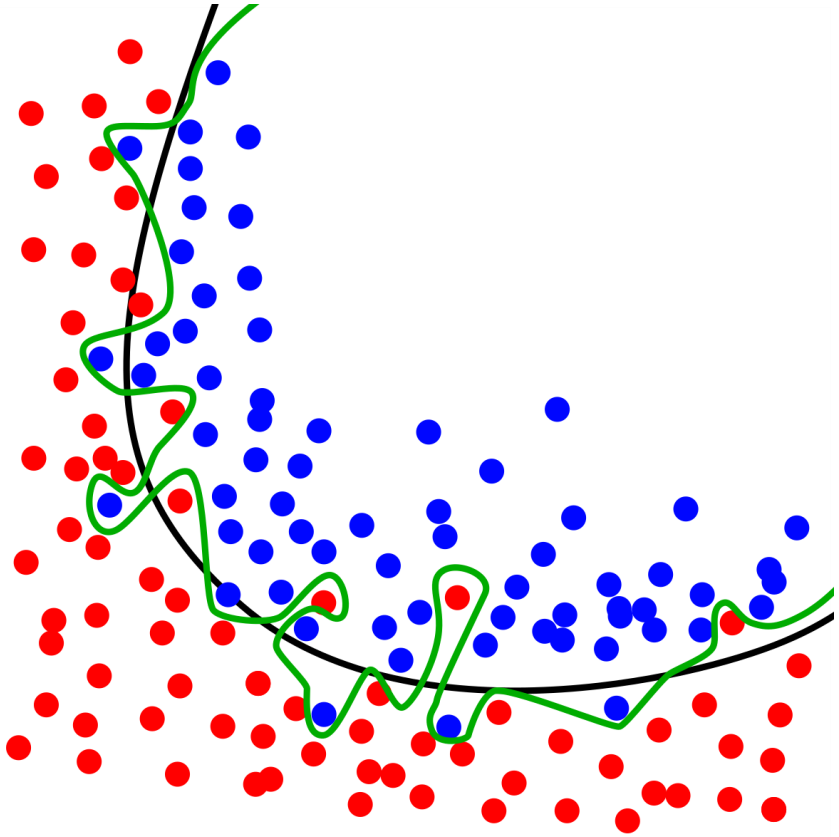


Outline

- Transformers (“Attention is all you need”)
  - Replacing recurrence with attention
  - All the bells and whistles
- Pretraining
  - Frozen features (ImageNet)
  - Fine-tuning (Masked language modeling)

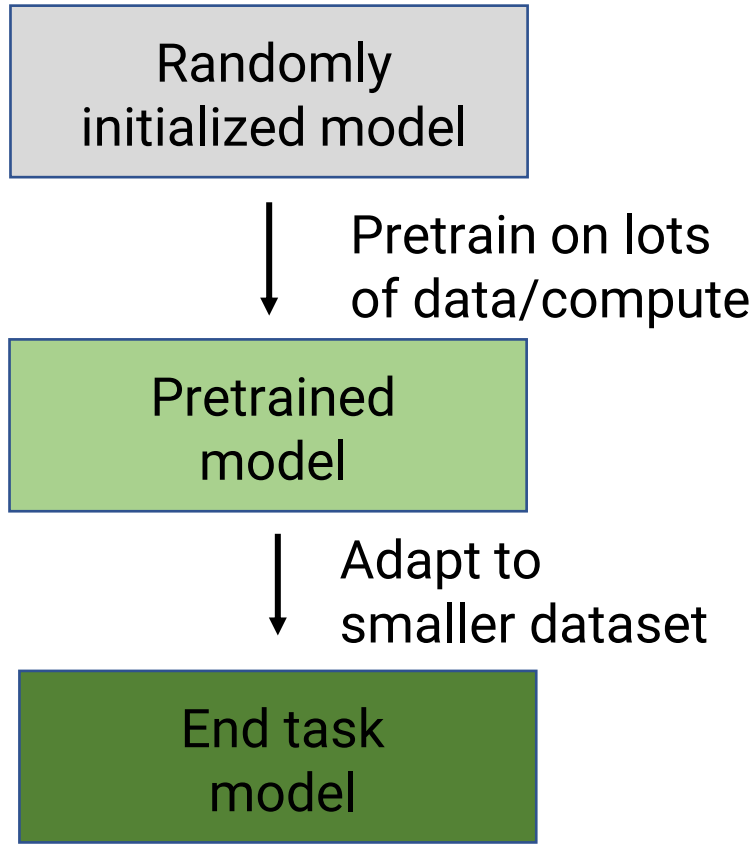
Neural Networks and Scale

- Neural networks are very expressive, but have tons of parameters
  - Very easy to overfit a small training dataset
- Traditionally, neural networks were viewed as flexible but very **"sample-inefficient"**: they need many training examples to be good
  - Computationally expensive
  - Training at scale often uses GPUs



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

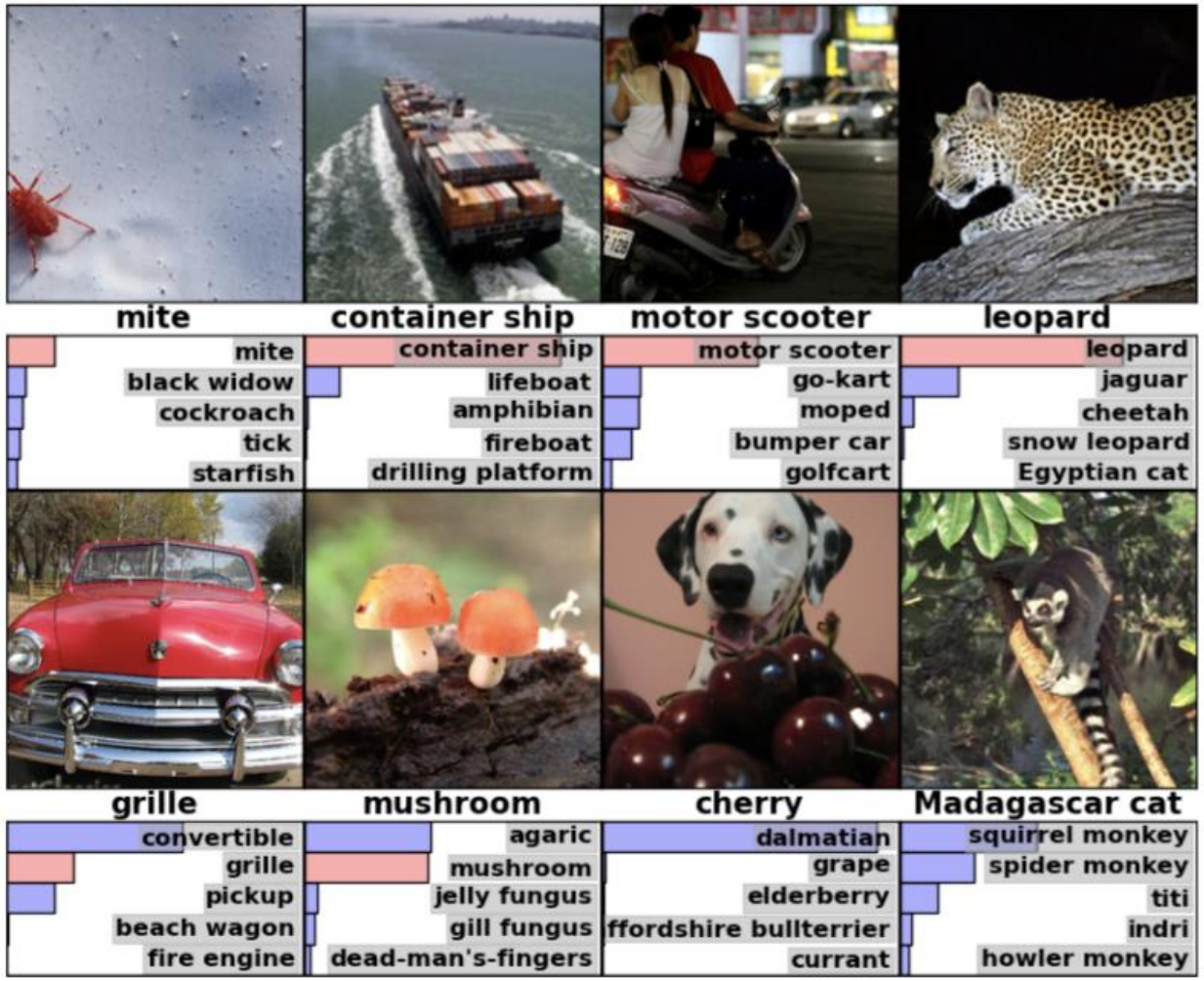
Text (years?)

Houses

Lens flare?

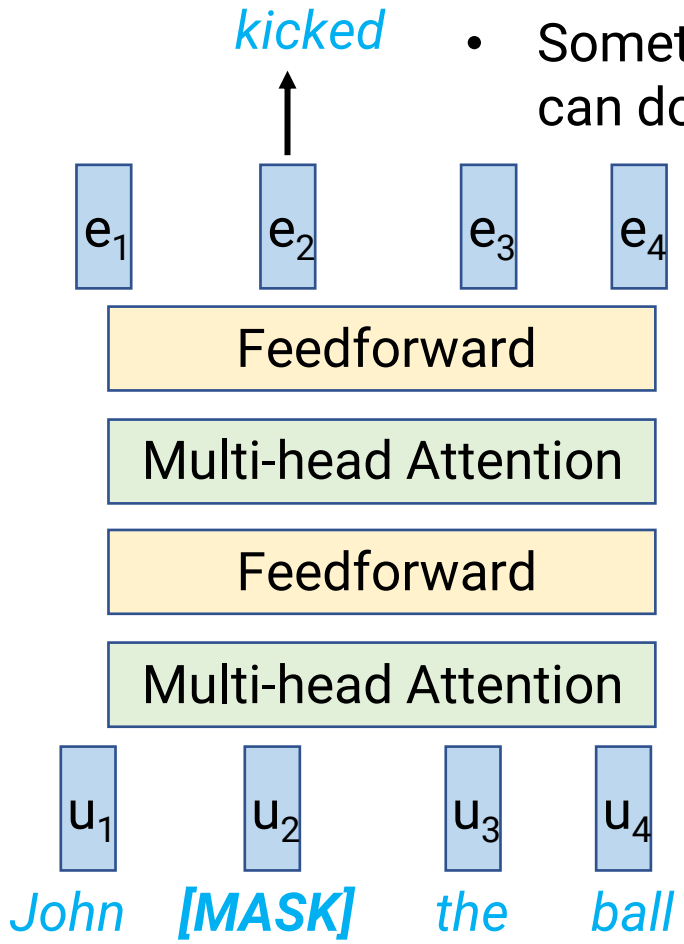
Features learned by AlexNet trained on ImageNet

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

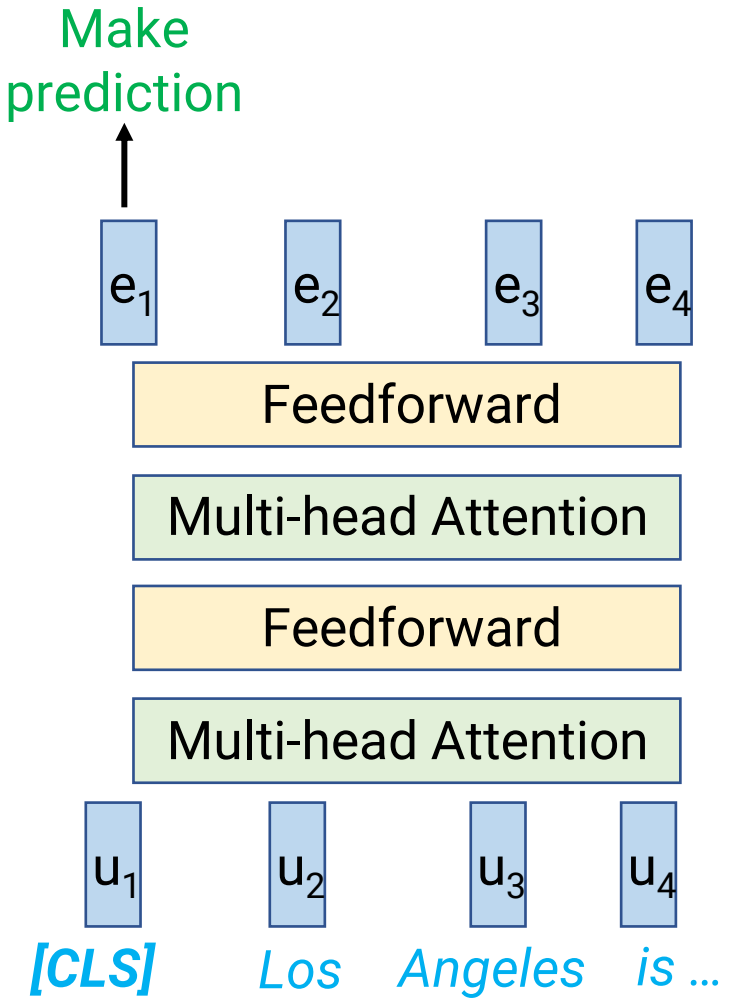
Masked Language Modeling (MLM)



- 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???

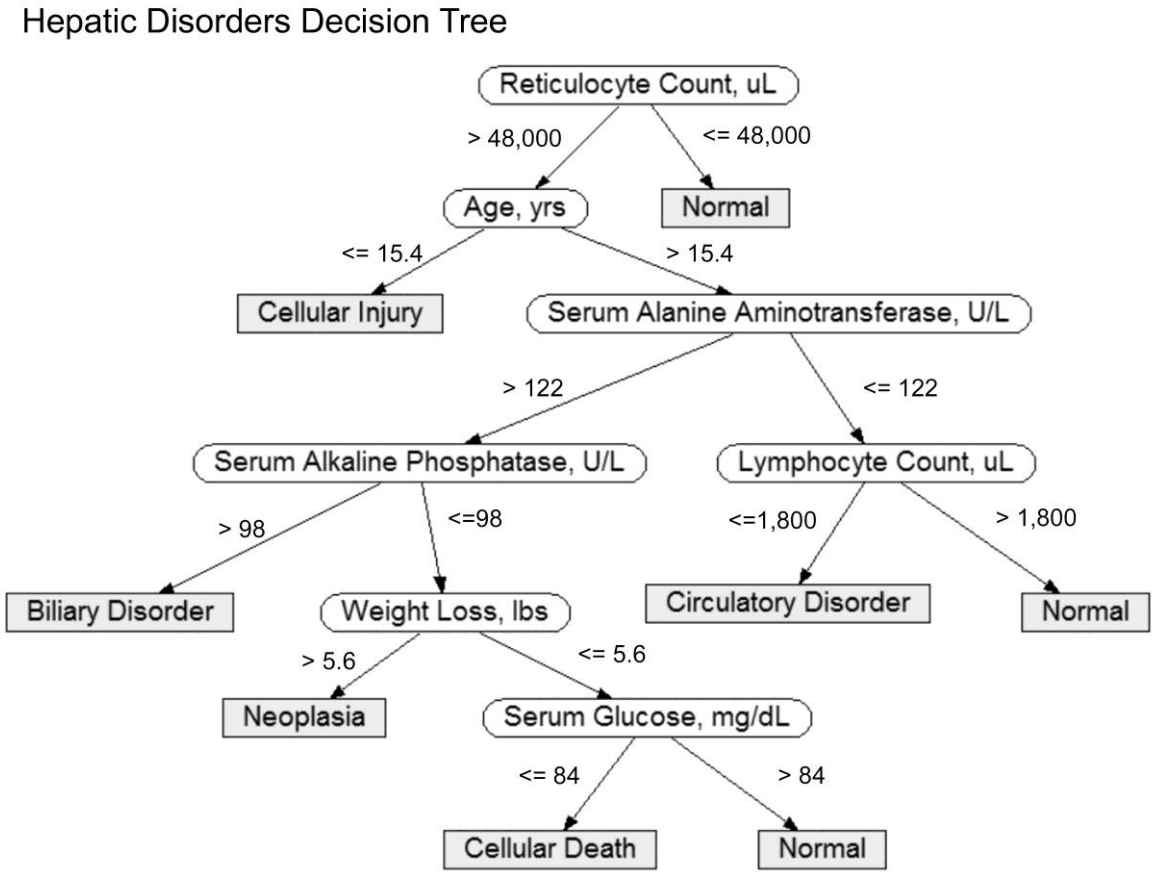
- ChatGPT appears to be a fine-tuned language model
  - Pretrained on autoregressive language modeling
  - Then fine-tuned with a method called RLHF (reinforcement learning from human feedback)
  - We'll return to this when we talk about reinforcement learning!

Announcements

- Midterm exam this Thursday!
  - All logistics information in Edstem post
- Project progress reports due Thursday March 23
  - Expectations on the website
  - CARC computing resource

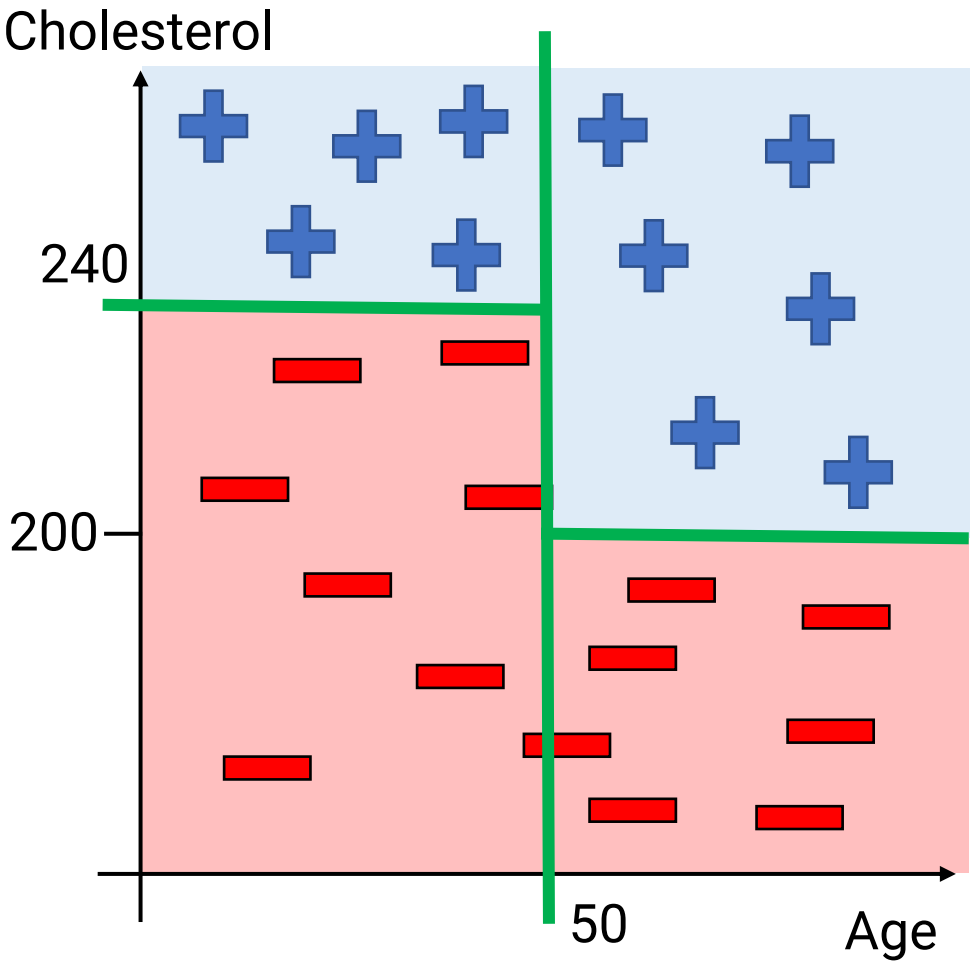
Modeling decision making

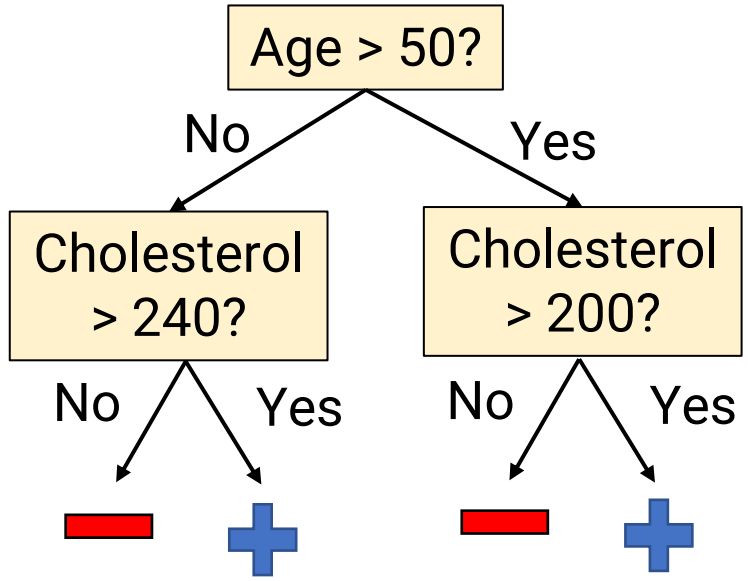
- Human experts make complex decisions and predictions every day
  - E.g., Given observations about a patient, what disease do they have?
- Can we build models that emulate the human decision-making process?



Decision Trees

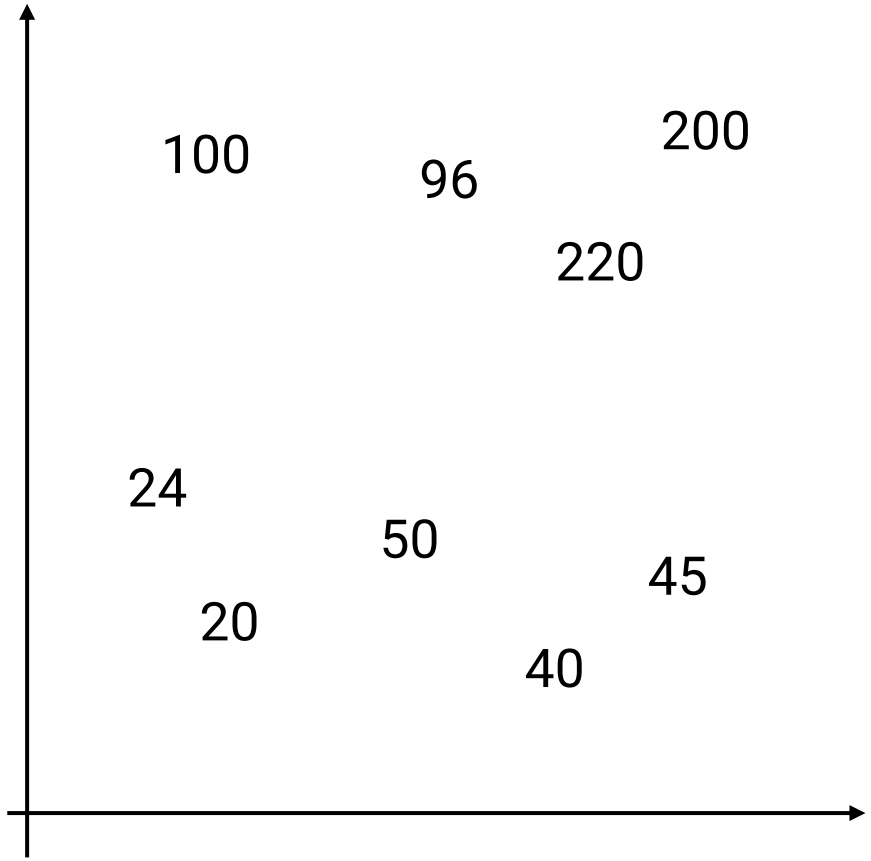
- At each node, split on one feature
- Remember the best output at each leaf node
  - Classification: Majority class
  - Regression: Mean within node
- Given new example, find which leaf node it belongs to and predict the associated output
- Interpretable!





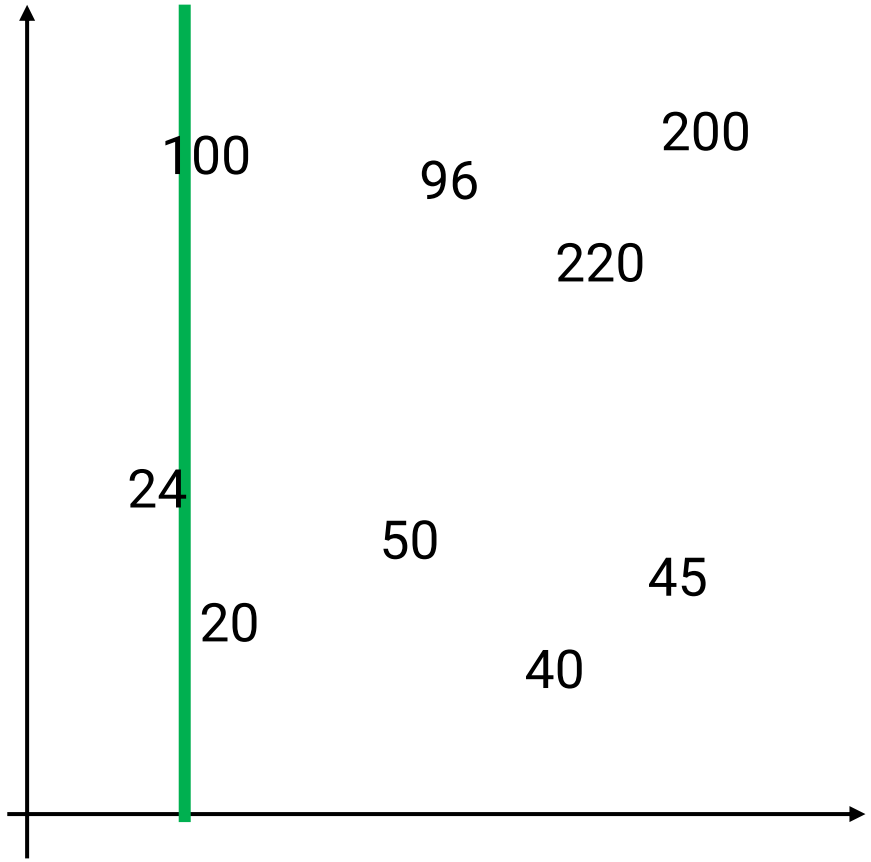
Learning Decision Trees for Regression

- 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



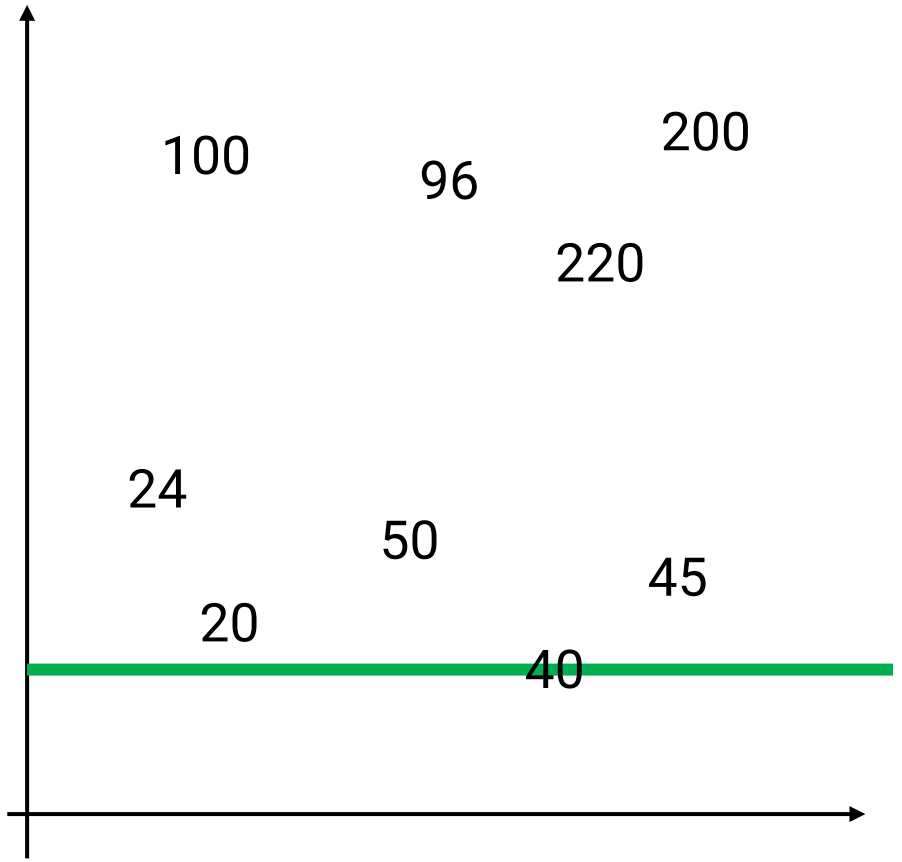
Learning Decision Trees for Regression

- 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



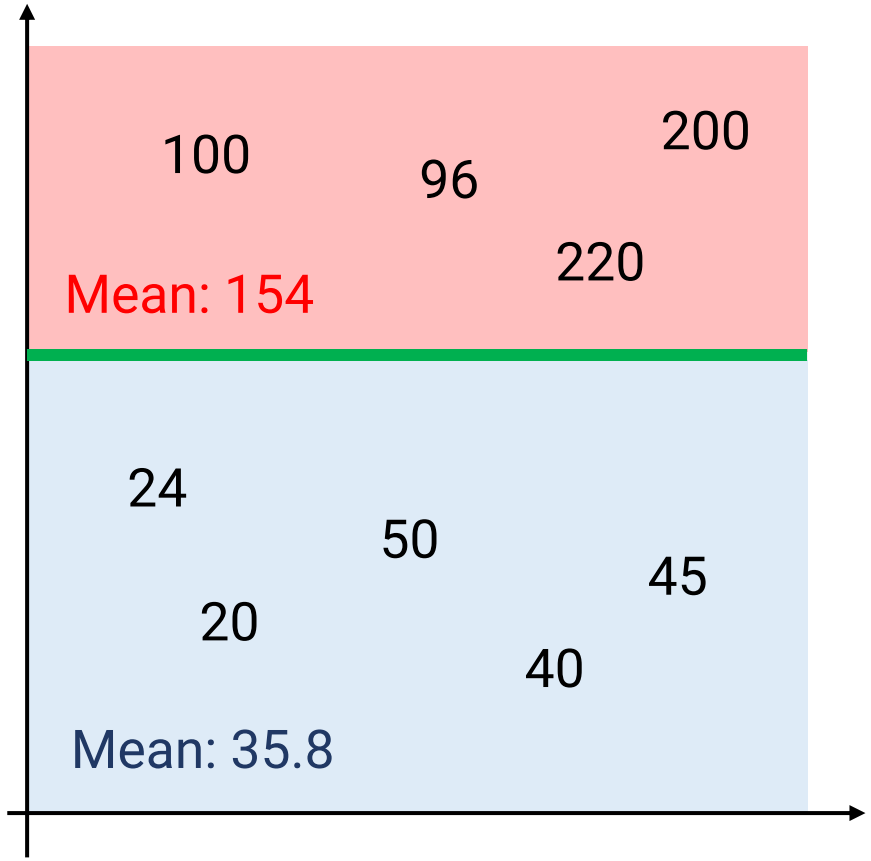
Learning Decision Trees for Regression

- 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



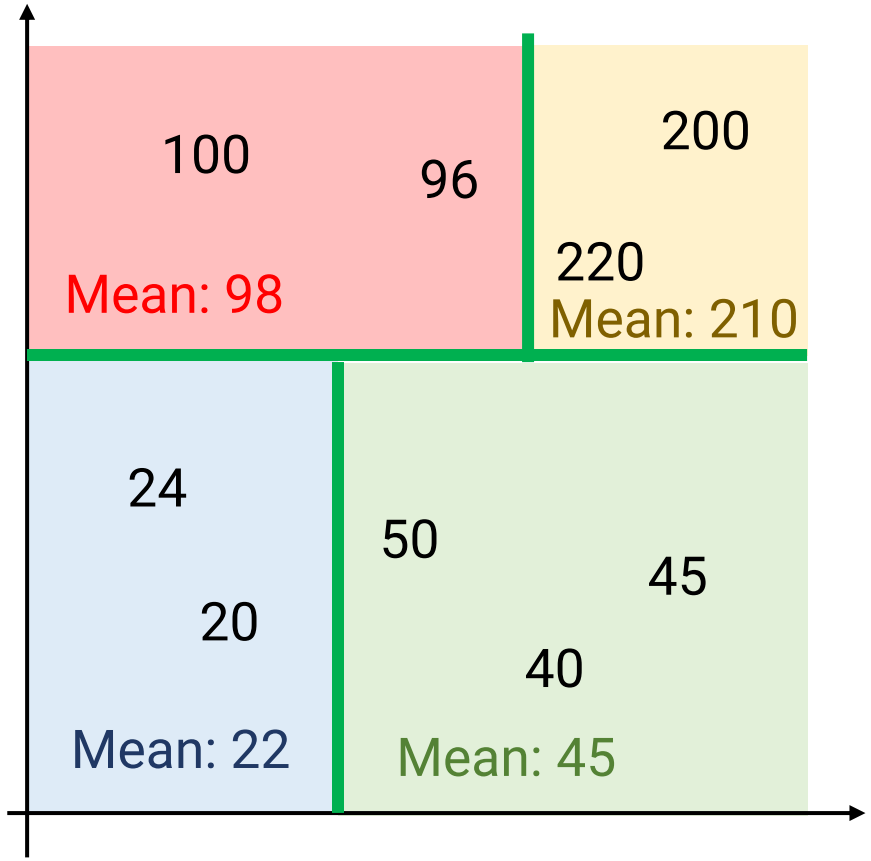
Learning Decision Trees for Regression

- 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



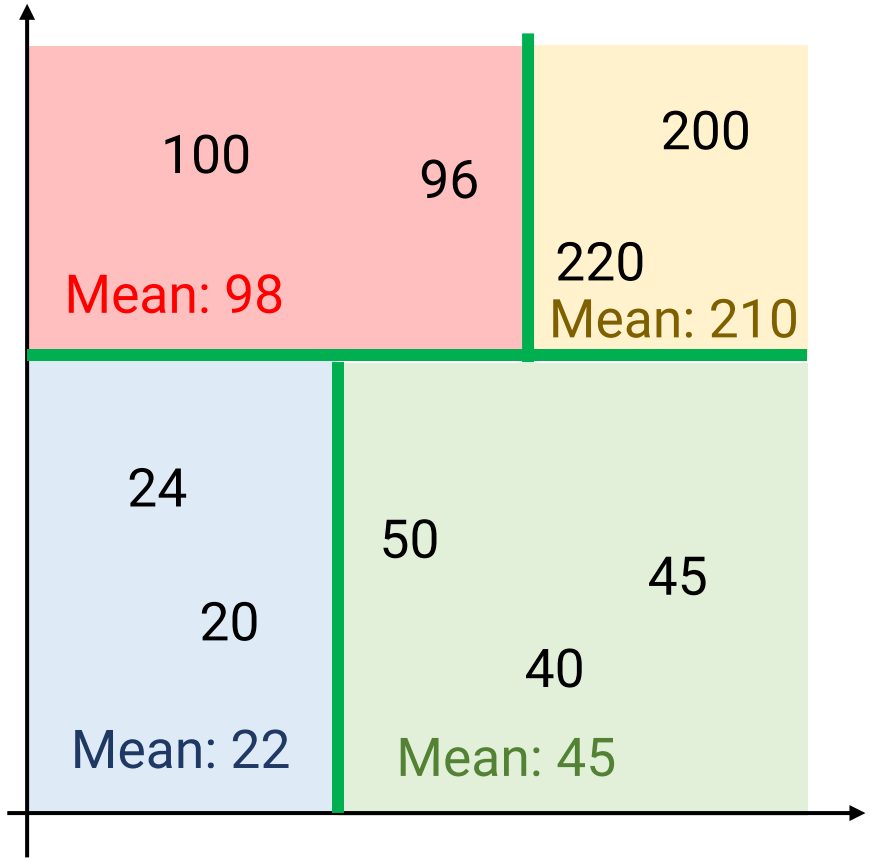
Learning Decision Trees for Regression

- 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



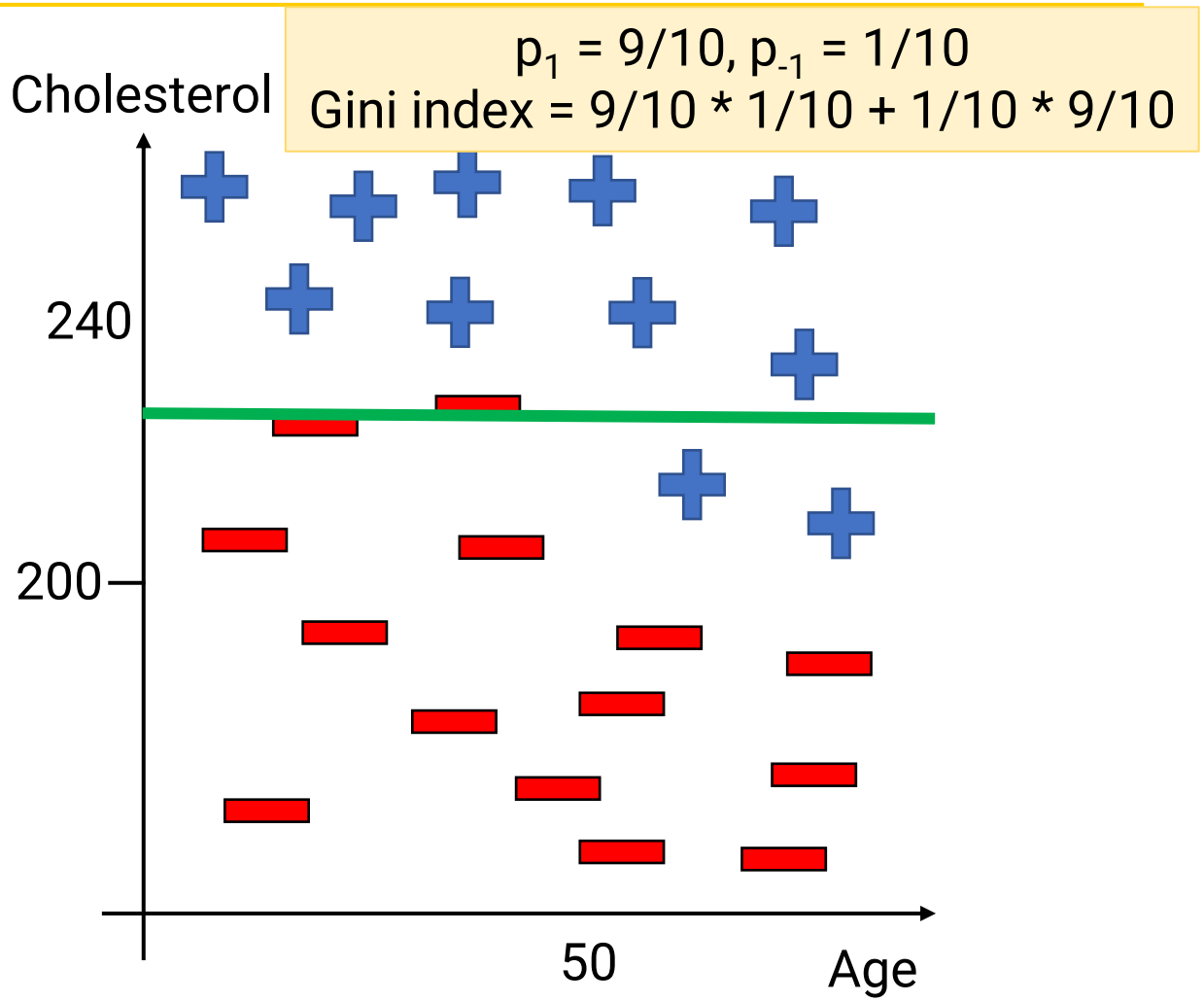
Learning Decision Trees for Regression

- 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)



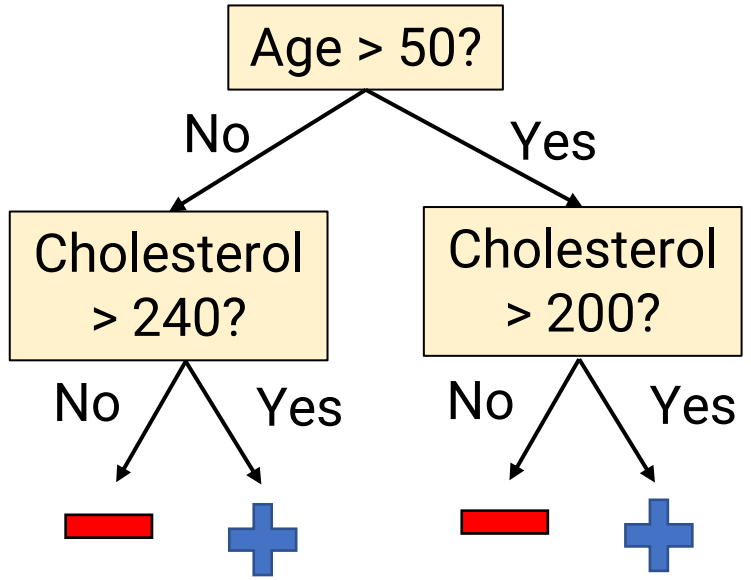
Learning decision trees for classification

- 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

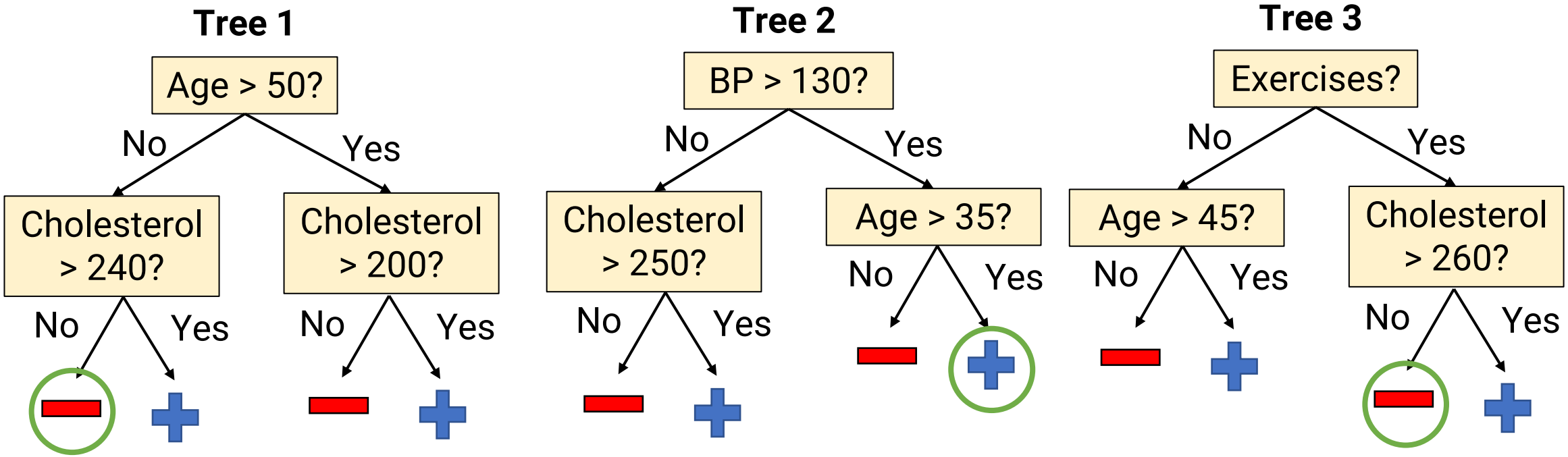


Handling Missing Features

- 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

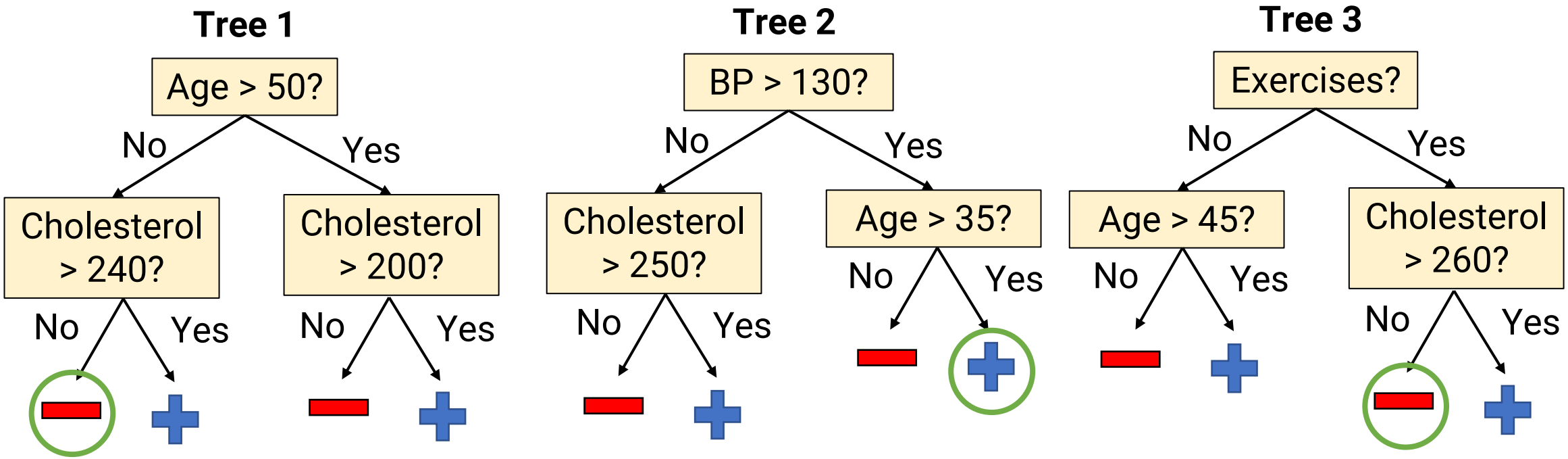


Ensembling



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

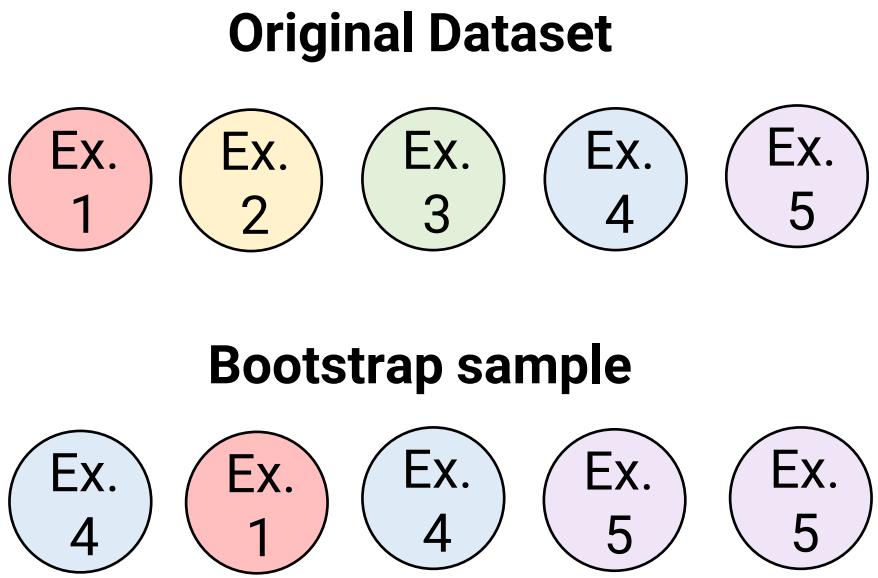
Ensembling and Trees



- 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

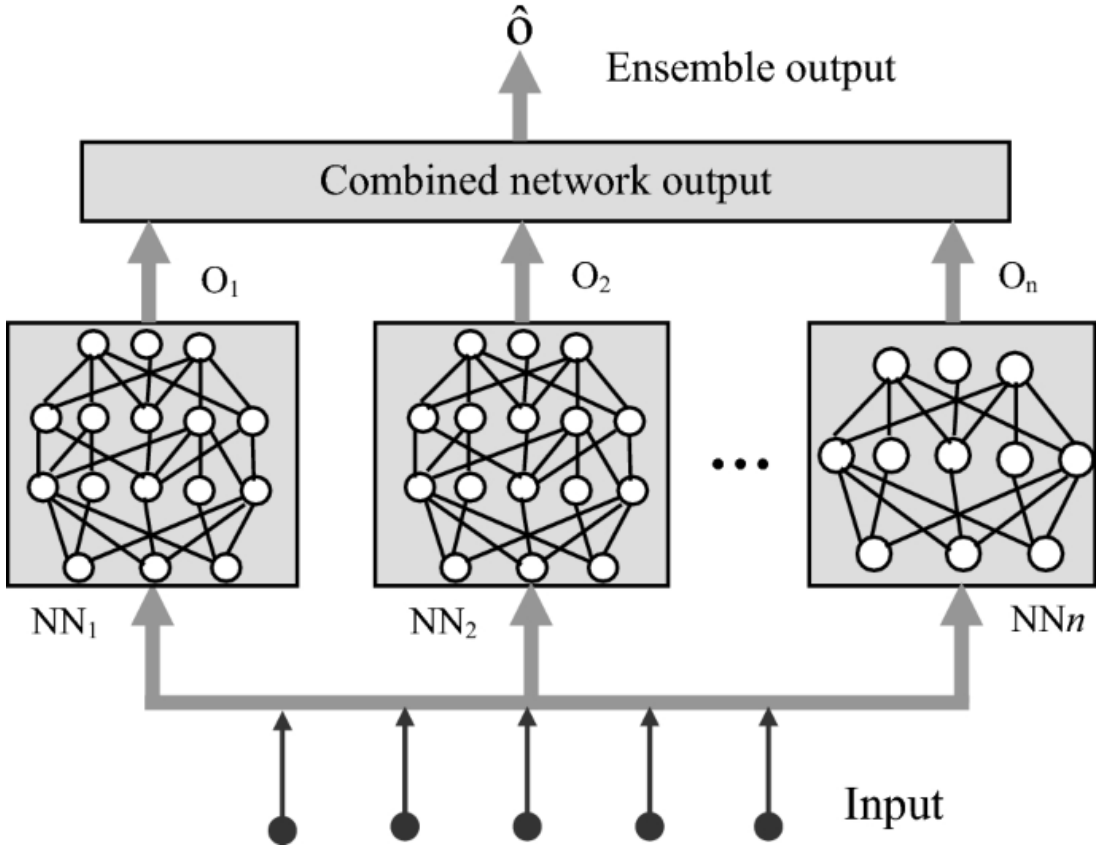


Random Forests

- Goal: Make the individual trees in the ensemble more different
  - Thus, all elements of the ensemble are complementary
  - Or: Reduce correlation between trees = lower variance
- 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
- In practice: Very common to ensemble neural networks!
  - Compute vs. accuracy trade-off



Conclusion

- Transformers as decoders
  - Attention can only attend to past/present, not future
  - At training time, handle this with clever masking
- 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