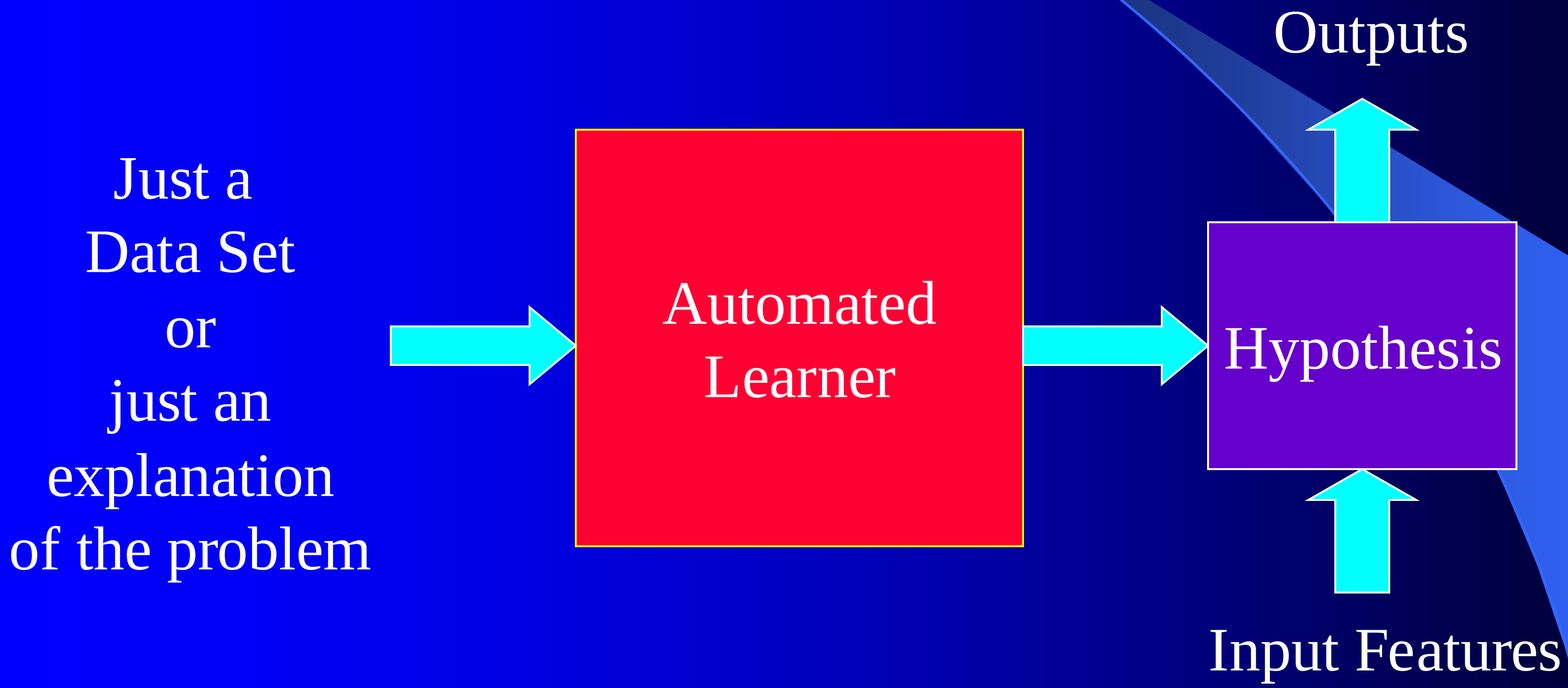
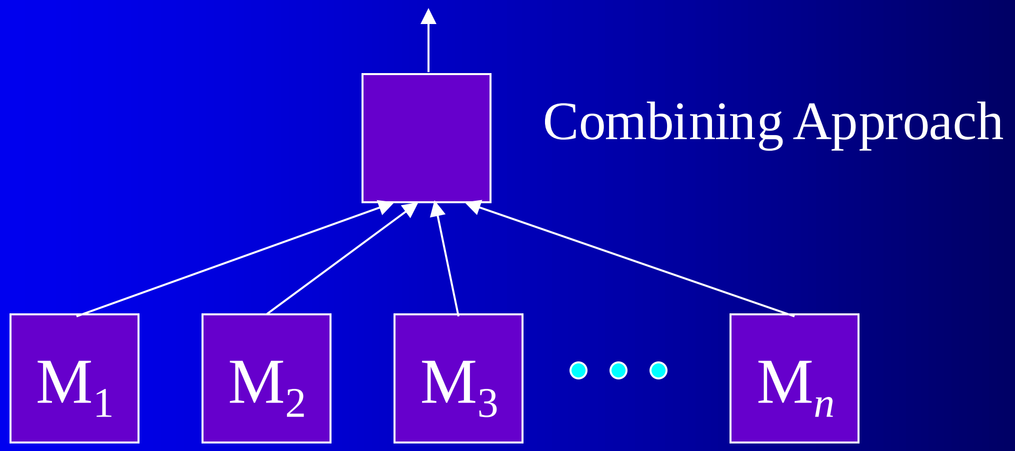
Ensembles

A "Holy Grail" of Machine Learning



Ensembles

- Multiple diverse models (Base Models) are trained on the same task and then their outputs are combined to come up with a final output
- Most commonly base models are variations of the same ML algorithm with different training sets or hyperparameters
- The specific overfit of each base model can be averaged out
- If models are diverse (uncorrelated errors) then even if the individual models are weak generalizers, the ensemble can be very accurate
- Many different Ensemble approaches
  - Bagging, Boosting, Stacking, Gating/Mixture of Experts, Wagging, Mimicking, Heuristic Weighted Voting, Combinations



Ensembles are Scriptural

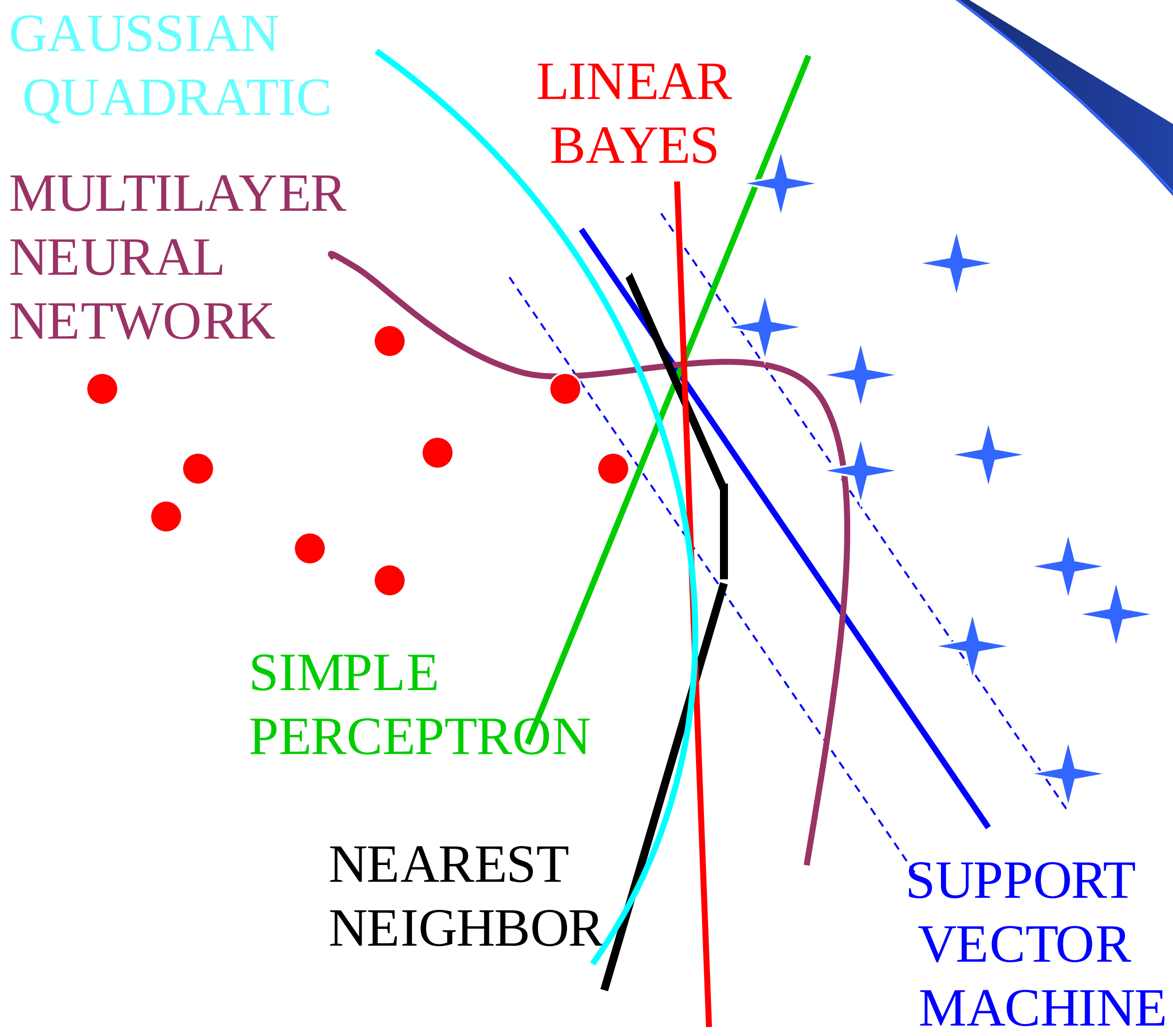
Mosiah 29:26, 27 Now it is not common that the voice of the people desireth anything contrary to that which is right; but it is common for the lesser part of the people to desire that which is not right; therefore this shall ye observe and make it your law--to do your business by the voice of the people.

And if the time comes that the voice of the people doth choose iniquity, then is the time that the judgments of God will come upon you; yea, then is the time he will visit you with great destruction even as he has hitherto visited this land.

Bias vs. Variance

- Learning models can have error based on two basic issues: Bias and Variance
  - "Bias" measures the basic capacity of a learning approach to fit the task
  - "Variance" measures the extent to which different hypotheses trained using a learning approach will vary based on training set variations and hyperparameters
- MLPs trained with backprop have low bias error because they can fit tasks well, but can have relatively high variance error because each model might fall into odd nuances (overfit) based on training set choice, initial weights, and other parameters – Typical with the more complex models
- Naïve Bayes has high bias error (doesn't fit that well), but has no variance error
- We would like low bias error *and* low variance error
- *Ensembles using multiple trained models with high-variance and low-bias error can average out the variance, leaving just the bias*

Some classifiers



Amplifying Weak Learners

- Combining weak learners
  - Assume *n* induced models which are independent of each other with each having accuracy of about 60% on a two class problem. While one model is not dependable, if a majority of a group of these lean in one direction, then we can have higher confidence.
  - If all *n* give the same class output then you can be confident it is correct with probability $1-(1-.6)^n$. For $n=10$, confidence would be 99.4%.
  - Normally not independent (e.g. similar training sets). If all *n* were the same hypothesis, then no advantage could be gained.
  - Also, unlikely that all *n* would give the same output, but if a majority did, then still get an overall accuracy better than the base accuracy of the models
  - If *m* models say class 1 and *w* models say class 2, then

    P(majority_class) = 1 – Binomial(n, min(m,w), .6)

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

Bagging

- Bootstrap aggregating (Bagging)
- Induce *n* learners using the same initial parameters
- Each training set is chosen uniformly at random with replacement from the original data set, training sets might be 2/3rds of the data set – still need to save some separate data for testing
- All *n* hypotheses have an equal vote for classifying novel instances
- Great way to improve overall accuracy by decreasing variance. Consistent significant empirical improvement
- Does not overfit (whereas boosting may), but may be more conservative overall on accuracy improvements
- Bigger *n* the better (diminishing), but need to consider efficiency trade-off
- Typically used with the same learning algorithm and thus best for those which tend to give more diverse hypotheses based on initial random conditions
- Could use other schemes to improve the diversity between learners
  - Different initial hyperparameters, sampling approaches, etc.
  - Different learning algorithms
  - The more diversity the better - (yet often used just with the same learning algorithm and different training subsets)

Boosting - AdaBoost

- Boosting by resampling - Each TS_t is chosen randomly with distribution D_t with replacement from the original training data. D_1 has all instances equally likely to be chosen. Typically each TS_t is the same size as the original data set.
  - Induce first model. Change D_{t+1} so that instances which are mis-classified by the current model on its current TS have a higher probability of being chosen for future training sets.
  - Keep training new models until stopping criteria met
    - *n* models induced – not best approach
    - Overall Accuracy levels out on validation set
    - Most recent model has accuracy less than 50% on its TS
- All models vote, but each model's vote is scaled by its accuracy on the training set it was trained on
- Boosting is more aggressive than bagging on accuracy but in some cases can overfit and do worse – can theoretically converge to training set
  - On average better than bagging, but worse for some tasks
  - In rare cases can be worse than the non-ensemble approach
  - Bagging can be trained in parallel, Boosting requires sequential training

Boosting

- Another approach to boosting is to have each base model train on the entire training set but have the ML algorithm take each current instance weighting into account during learning.
- How might you do that for
  - MLPs
  - Decision Trees
  - *k*-NN
- Then still have final models vote each weighted by its accuracy

Boosting

- Another approach to boosting is to have each base model train on the entire training set but have the ML algorithm take each current instance weighting into account during learning.
- How might you do that for
  - MLPs – Scale learning rate by weight
  - Decision Trees – instance membership is scaled by weight
  - *k*-NN – node vote is scaled by weight
- Then still have final models vote each weighted by its accuracy

Ensemble Creation Approaches

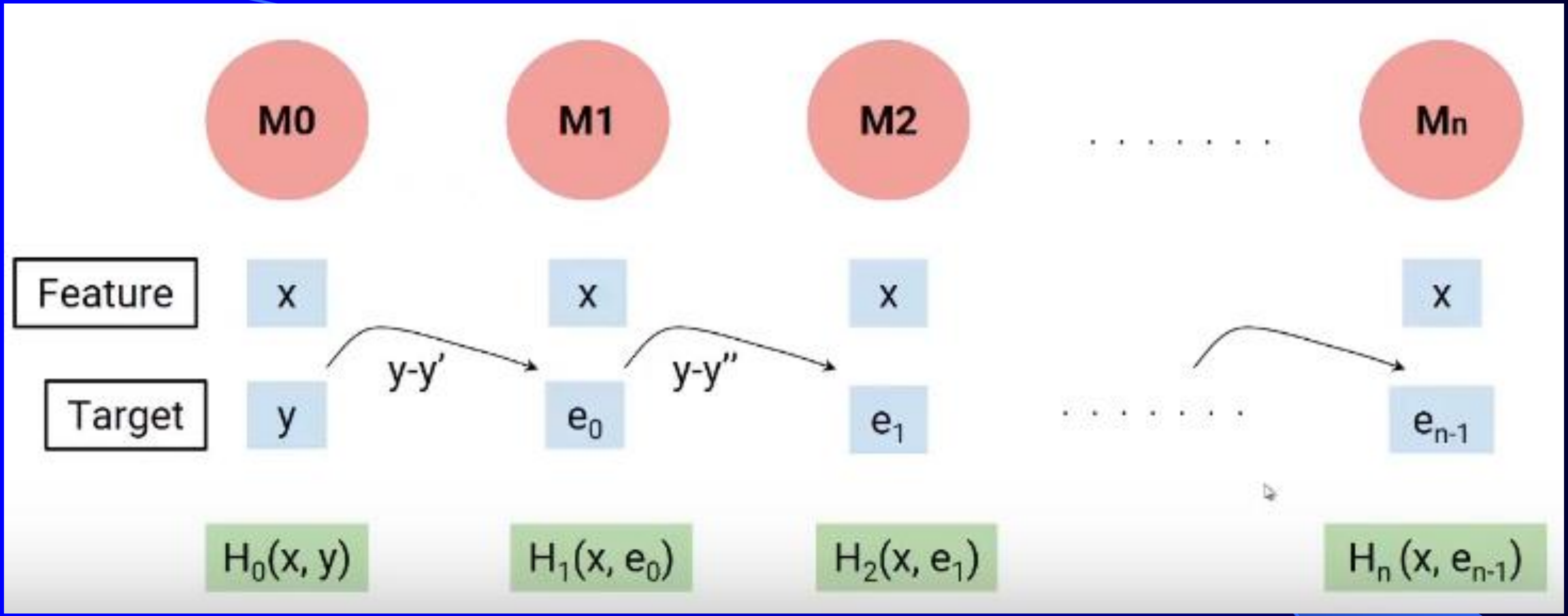
- A good goal is to get less correlated errors between models
- Injecting randomness – initial weights, different learning parameters, etc.
- Different Training sets – Bagging, Boosting, different features, etc.
- Different subset of features for each model
- Forcing differences – different objective functions, auxiliary tasks
- Different machine learning models
  - Obvious, but surprisingly it is used less often
  - More work to get all the models running, creating compatible data formats, etc.
- One aspect of *COD* (Classifier Output Distance) research - which algorithms are most different and thus most effective to ensemble

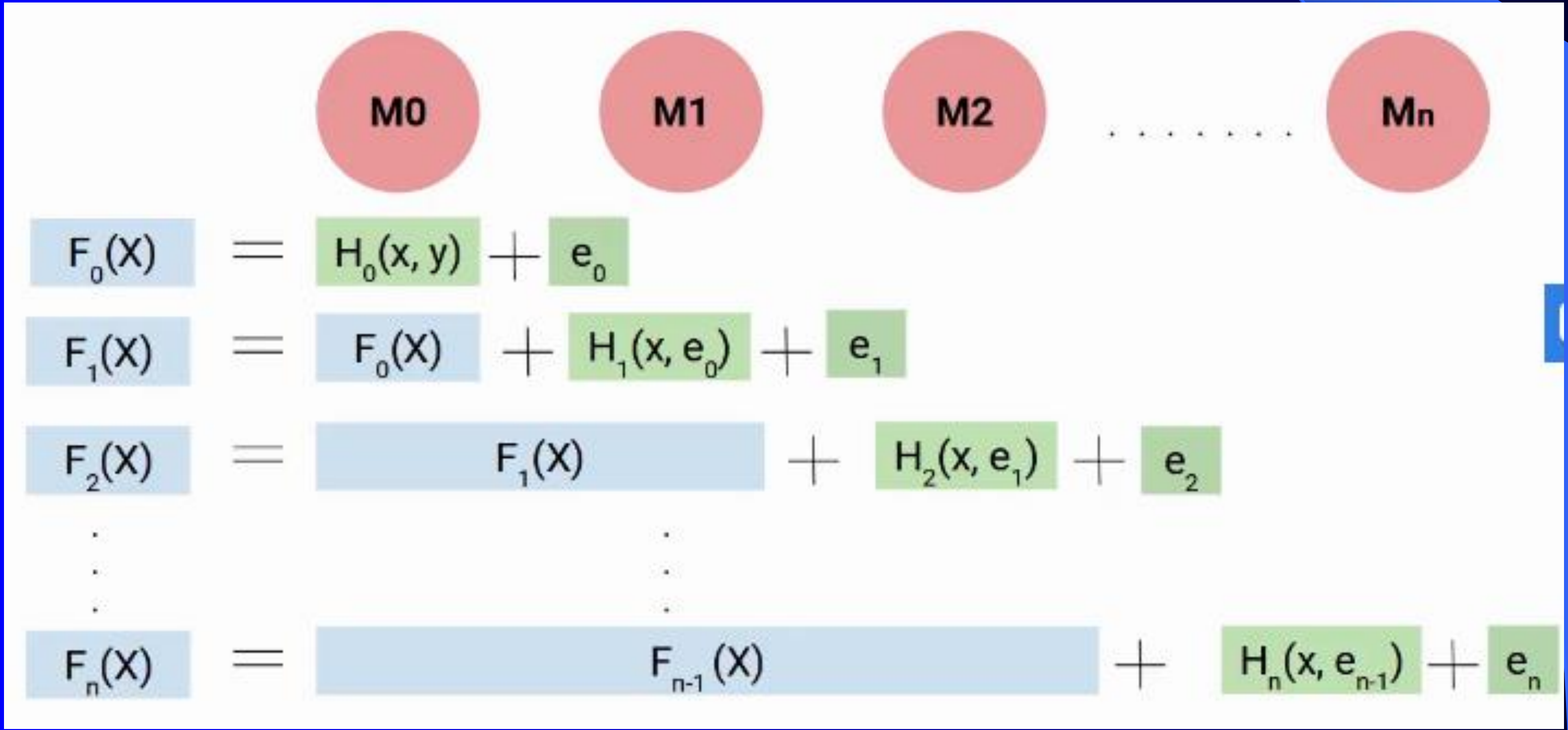
Ensemble Combining Approaches

- Unweighted Voting (e.g. Bagging)
- Weighted voting – based on accuracy, etc. (e.g. Boosting)
- Stacking - Learn the combination function
  - Higher order possibilities
  - Which algorithm should be used for the stacker
  - Must match the input/output data types between models
  - Stacking the stack, etc.
- Gating function/Mixture of Experts – The gating function uses the input features to decide which expert or combination (weighted) of experts to use in the vote with experts being strong in different parts of the input space
- Heuristic Weighted Voting – differs for each instance

Brief Intro to Gradient Boosting

- Often used with decision trees and regression
- Common winner in task competitions
- Train first model F_1 with the basic training set
- Train next model h creating updated ensemble model $F_{m+1} = F_m + h$
- But, train h using the *residual/error* (the difference between the target and the current output of F_m)
  - For h, change training instance (x,y) to $(x, y - F_m(x))$
  - Each new model learns to output and cancel the remaining error from the previous model, leaving less error with each model
  - Learning focuses on instances where the latest F_m has higher error
- Also learns each model's weighting coefficient γ with gradient descent to minimize chosen loss function (SSE common)
- Once trained, F_m no longer changes, and we keep adding new h's until remaining error is almost gone or test error begins to increase
- Final output is the weighted sum of the models





$$F_0(X) = H_0(x, y) + e_0$$

$$F_1(X) = F_0(X) + H_1(x, e_0) + e_1$$

$$F_2(X) = F_1(X) + H_2(x, e_1) + e_2$$

$$\vdots$$

$$F_n(X) = F_{n-1}(X) + H_n(x, e_{n-1}) + e_n$$

Brief Intro to Gradient Boosting

$$F_{n+1}(X) = F_n(X) + \gamma_n H(x, e_n)$$

$$\begin{aligned}
F_0(X) &= \gamma_0 H_0(x, y) + e_0 \\
F_1(X) &= F_0(X) + \gamma_1 H_1(x, e_0) + e_1 \\
F_2(X) &= F_1(X) + \gamma_2 H_2(x, e_1) + e_2 \\
&\vdots \\
F_n(X) &= F_{n-1}(X) + \gamma_n H_n(x, e_{n-1}) + e_n
\end{aligned}$$

- How to combine? Each model's weighting/voting coefficient γ_i is learned with gradient descent to minimize loss as models are created, then frozen
- XGBoost – eXtreme Gradient Boosting – a popular and successful software library for efficient parallel gradient boosting implementations

An Example of Gradient Boosting

1) Fit a shallow regression tree T_1 to the data
   - the first model is $M_1 = T_1$
   - The shortcomings of the model are given by the negative gradients.
2) Fit a tree T_2 to the negative gradients
   - The second model is: $M_2 = M_1 + \eta\gamma_2 T_2$
   - η is a learning rate (e.g. .1) to encourage more models
   - γ_i is optimized, then frozen, so that M_i best fits the data
3) Continue adding models (trees) until stopping criteria met
4) The final model is $M_{\text{final}} = M_{\text{final-1}} + \eta\gamma_{\text{final}} T_{\text{final}}$

Gradient Boosting Notes

- Avoid overfit (and maintain relatively weak models) by:
  - Tree constraints (e.g. max depth usually 4-8)
  - Tree models learn different γ for each leaf node
  - Stochastic Gradient Boosting - rows and/or columns dropped for current TS – commonly creating TS for *h* by choosing 50% of DS without replacement
  - Shrinkage – new *h's* scaled by smaller learning rate (e.g. .1), leading to a larger number of iterative models (slower learning but better generalization)
  - Early stopping (validation set)
  - Regularization - Standard L1 and L2 on weight magnitudes

Dropout – Overfit avoidance

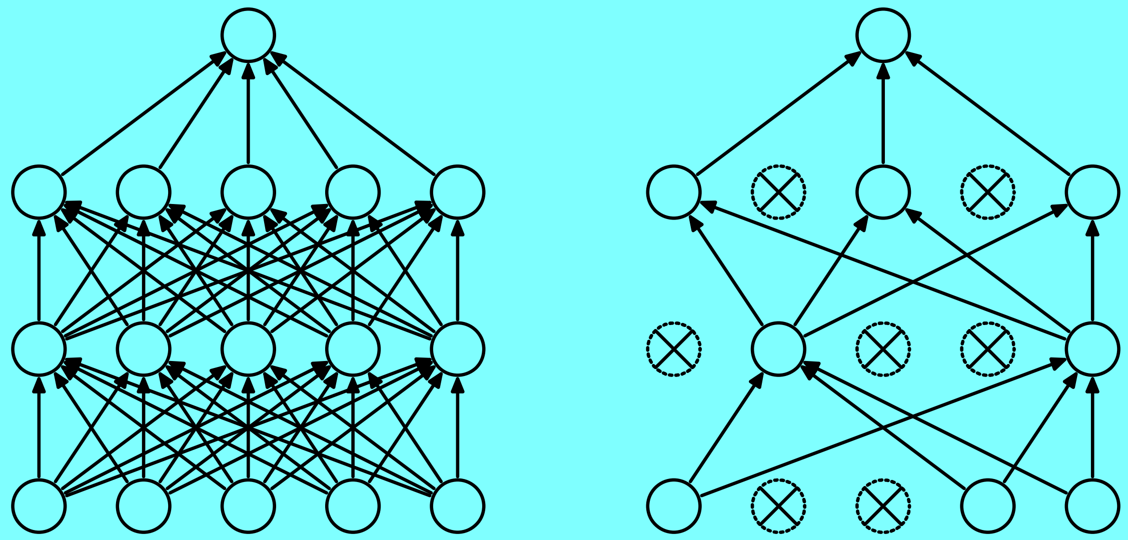


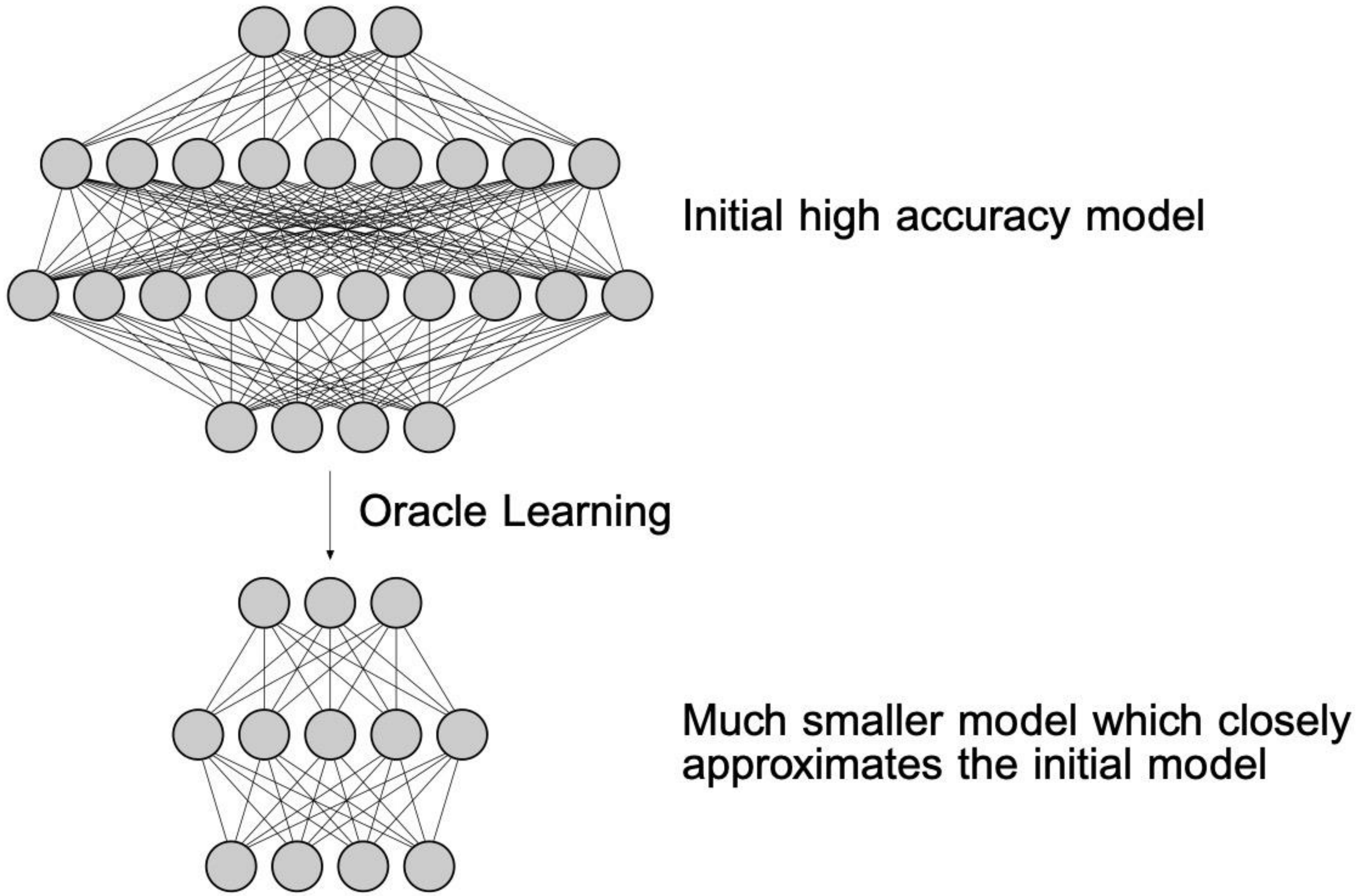
Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- For each instance during training temporarily drop any hidden or input node and its connections with probability *p* and then train
- Final network just has all averaged weights (actually scaled by 1-*p* since that better matches the expected values at training time)
- Works as if ensembling 2^n different network substructures
- Lots of other variations – Dropconnect, etc.

Ensemble Summary

- Other Models – Random Forests, Boosted stumps, Cascading, Arbitration, Delegation, PDDAGS (Parallel Decision DAGs), Bayesian Model Averaging and Combination, Clustering Ensemble, etc.
- End-to-end ensembles, combiner participates in base model training
- Efficiency Issues
  - Wagging (Weight Averaging) - Multi-layer? - Dropout
  - Mimicking - Oracle Learning, semi-supervised
- Great way to decrease variance/overfit
- Almost always gain accuracy improvements with Ensembles

Oracle Learning


Initial high accuracy model
Oracle Learning
Much smaller model which closely approximates the initial model