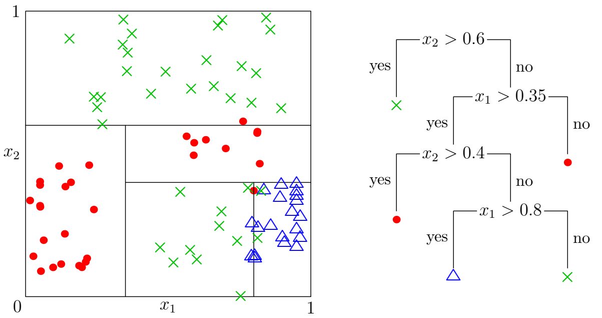
Advanced Machine Learning

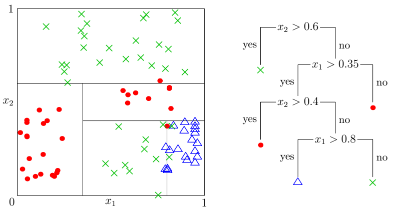
Boosting



Boosting, AdaBoost, Gradient Boosting

Outline

1. **Boosting**
2. AdaBoost
3. Gradient Boosting
4. Dropout



Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \hat{h}_i(\boldsymbol{x})$$

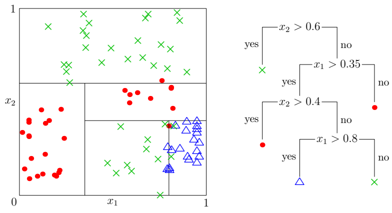
- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance
- The trick is to choose the weights, α_i
- Because the weak learners do little better than chance we (miraculously) **don't** overfit that much

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump), although usually we would use slightly deeper trees
- There are different algorithms for choosing the weights
  - adaboost—a classic algorithm for binary classification
  - gradient boosting—used for regression, trains a weak learner on the residual errors

Outline

1. Boosting
2. **AdaBoost**
3. Gradient Boosting
4. Dropout



Boosting a Binary Classifier

- Suppose we have a binary classification task with data $\mathcal{D} = \{(\boldsymbol{x}^\mu, y^\mu) | \mu = 1, 2, \dots, m\}$ with $y^\mu \in \{-1, 1\}$
- Our i^{th} weak learner provides a prediction $\hat{h}_i(\boldsymbol{x}^\mu) \in \{-1, 1\}$
- We ask, can we find a linear combination

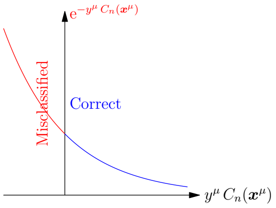
$$C_n(\boldsymbol{x}) = \alpha_1 \hat{h}_1(\boldsymbol{x}) + \alpha_2 \hat{h}_2(\boldsymbol{x}) + \cdots + \alpha_n \hat{h}_n(\boldsymbol{x})$$

- So that $\operatorname{sgn}(C_n(\boldsymbol{x}))$ is a strong learner?
- Note we want $y^\mu C_n(\boldsymbol{x}^\mu) > 0$

AdaBoost

- AdaBoost is a classic solution to this problem
- It assigns an "loss function"

$$L_n = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)}$$



- This punishes examples where there is an errors more than correct classifications

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$\begin{aligned}
L_n(\alpha_n) &= \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \hat{h}_n(\boldsymbol{x}^\mu))} \\
&= \sum_{\mu=1}^{m} w_n^\mu \mathrm{e}^{-\alpha_n y^\mu \hat{h}_n(\boldsymbol{x}^\mu)} = \mathrm{e}^{\alpha_n} \sum_{\mu:y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \mathrm{e}^{\alpha_n} + \mathrm{e}^{-\alpha_n} \sum_{\mu:y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu \mathrm{e}^{-\alpha_n} \\
&= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}) \sum_{\mu:y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu
\end{aligned}$$

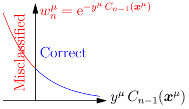
Choosing a Weak Classifier

- To minimise the loss

$$L_n(\alpha_n) = \mathrm{e}^{-\alpha_n}\sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}) \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

- We choose the weak learner with the lowest value of

$$\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$$



- That is, it misclassifies only where the other learners classify well

Choosing Weights

- We now choose the weight α_n to minimise the loss $L_n(\alpha_n)$

$$\frac{\partial L_n(\alpha_n)}{\partial \alpha_n} = \mathrm{e}^{\alpha_n} \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu - \mathrm{e}^{-\alpha_n} \sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = 0$$

- That is

$$\mathrm{e}^{2\alpha_n} = \frac{\sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \qquad \text{or} \qquad \alpha_n = \frac{1}{2}\log\left(\frac{\sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}\right)$$

Algorithm

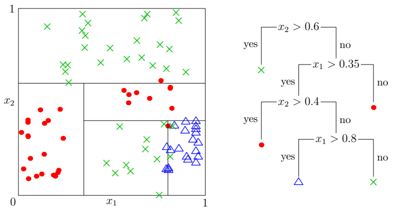
1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
3. Initially $w_1^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \hat{h}_n(\boldsymbol{x})$ where $\alpha_n = \frac{1}{2}\log\left(\frac{\sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}\right)$
6. Update $w_{n+1}^\mu = w_n^\mu \mathrm{e}^{-y^\mu \alpha_n \hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Performance

- Adaboost works well with weak learners, usually out-performing bagging
- It doesn't work well with strong learners (tends to over-fit)
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)
- It has fallen from fashion
- In contrast **gradient boosting** used for regression is very popular

Outline

1. Boosting
2. AdaBoost
3. **Gradient Boosting**
4. Dropout



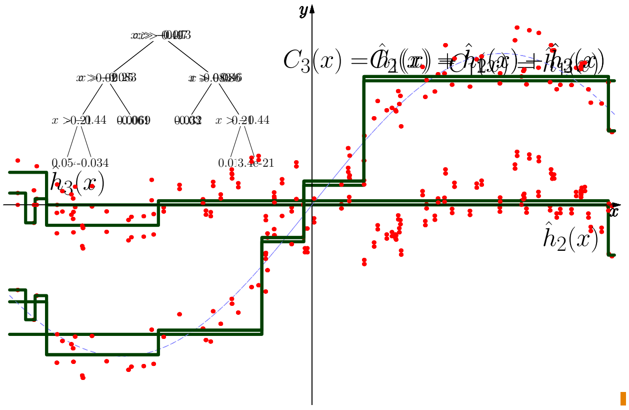
Gradient Boosting

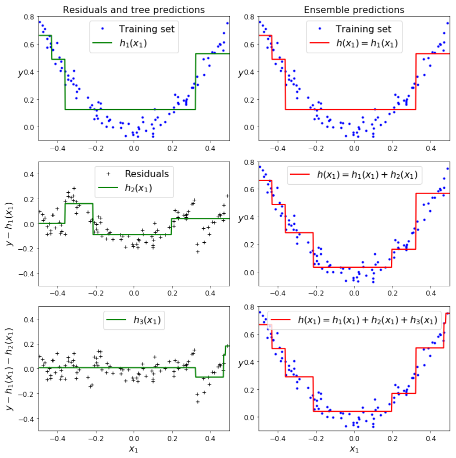
- In gradient boosting we again build a strong learner as a linear combination of weak learners

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \hat{h}_n(\boldsymbol{x})$$

- Gradient boosting used on regression (again using decision trees)
- At each step $\hat{h}_n(\boldsymbol{x})$ is trained to predict the **residual error**, $\Delta_{n-1} = y - C_{n-1}(\boldsymbol{x})$, (i.e. the target minus the current prediction)
- (This difference looks a bit like a gradient hence the rather confusing name)

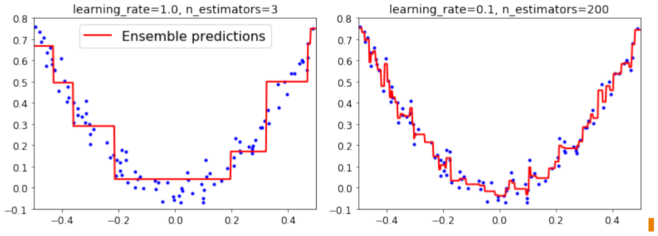
Fitting a Sin Wave





Keep On Going

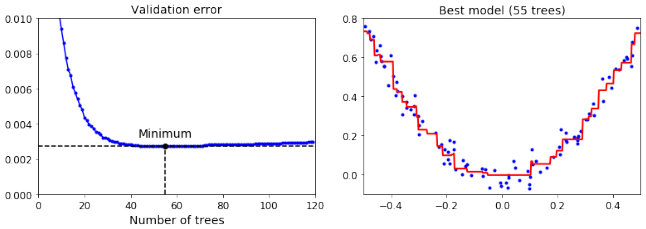
- We can keep on going



- But we will over-fit eventually

Early Stopping

- Like many algorithms we often get better results by early stopping



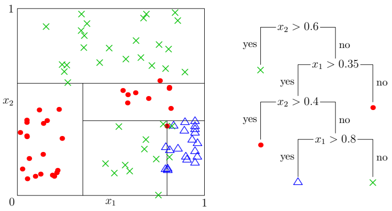
- Use cross-validation against a validation set to decide when to stop

XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast
- Rather a bodge of optimisation hacks
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better

Outline

1. Boosting
2. AdaBoost
3. Gradient Boosting
4. **Dropout**

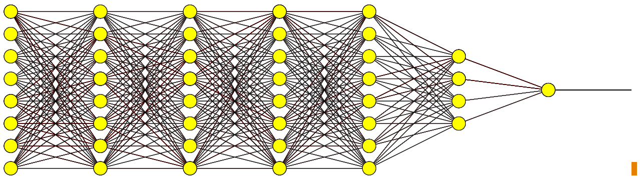


Ensembling in Deep Learning

- For most machine learning ensembling different machines usually gives a reasonable improvement in performance
- The machines should have roughly the same performance
- Of course, this comes at the price of having to train multiple machines
- One can try to train a machine to decide how to combine different machines (stacking), but beware, it is very easy to overfit
- Usually better to average predictions for regression or do majority voting for classification problems

Dropout

- For deep learning we can control for over-fitting using dropout



- This can be seen as ensembling lots of much simpler machines

Conclusion

- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)—probably
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model