STA 4241 Lecture, Week 14

Ensemble approaches

- Ensemble approaches
  - Super learner
- **Introduction to neural networks**
  - Projection pursuit regression
  - Hidden layer neural network
- When to use neural networks and why do they work?

Projection pursuit regression

- Before discussing neural networks, we discuss an approach called projection pursuit regression (PPR)
  - Many similarities to neural networks
  - Neural networks build on the ideas of projection pursuit regression
- The main idea of these approaches is to find new features, which are linear combinations of the original features
- Then run a highly nonlinear regression on these new features

Projection pursuit regression

- As is typically the case, our goal will be to estimate $\mathbb{E}(Y|X) = f(X)$
- The projection pursuit regression model is defined as

$$f(X) = \sum_{m=1}^{M} g_m(\omega_m^T X)$$

- This looks a lot like a generalized additive model
- Additive in the derived features, $V_m = \omega_m^T X$

Projection pursuit regression

- In this model there are two key unknown parameters to be estimated
  - The directions or weights given by ω_m
  - The nonlinear functions $g_m(\cdot)$
- The model simultaneously tries to find directions and functions that fit the data well
  - Most approaches we've considered have done only one of these at a time
  - Principle components regression is a special case where $g_m(x) = x$ and ω_m is chosen via PCA
  - Generalized additive models are the special case where $m = p$ and $\omega_m X = X_m$

Projection pursuit regression

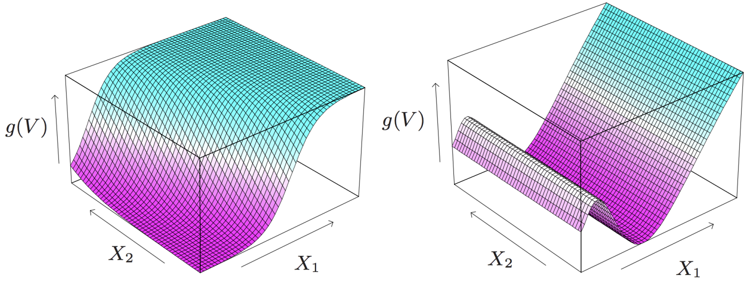
- This class of models is extremely large
  - Can account for high degrees of nonlinearity or non-additivity
  - Additivity in the derived features does not imply additivity of the original ones
- For instance, $X_1 X_2 = [(X_1 + X_2)^2 - (X_1 - X_2)^2]/4$
- Amazingly it can be shown that for a large enough M and correct functions g_m, **any** continuous function can be approximated arbitrarily well by the PPR model
  - PPR is a universal approximator

Projection pursuit regression

- This universal approximation property is quite incredible
  - Very flexible approach
- The main drawback is in interpretation
  - Not unique to PPR
  - Most machine learning approaches have this limitation
- PPR is best used for making predictions, not for inferring structural features of the model

Projection pursuit regression

- The $g_m(\cdot)$ functions are called ridge functions
- Below are two example ridge function / direction combinations when we have two covariates



Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning. New York: springer.

Projection pursuit regression

- So how do we estimate these complex models?
- We can aim to minimize the squared error

$$\sum_{i=1}^{n} \left\{ Y_i - \sum_{m=1}^{M} g_m(\omega_m^T X_i) \right\}^2$$

  with respect to the functions g_m and vectors ω_m for $m = 1, \ldots, M$

- Need to impose some structure on the g_m functions as well
  - Avoid overfitting

Projection pursuit regression

- To simplify, let's first assume $M = 1$ so we only have one term
  - This is called a single index model and is widely used in some fields
- To simultaneously estimate g and ω we use an iterative procedure
  - First estimate g, then conditionally on that, estimate ω
  - Repeat until convergence
- Each of these two individual steps is relatively straightforward

Projection pursuit regression

- Suppose we know ω and want to estimate g
- Therefore we know $V = \omega^T X$ and we just need to estimate $g(V)$
- This is simply a one-dimensional smoothing problem!
  - See lectures 10 and 11 for a wide range of ways to solve those
- Smoothing splines and local regression are most convenient for computational reasons

Projection pursuit regression

- Now suppose we have estimated g and need to estimate ω conditional on g
- Letting $\omega_{(t)}$ be the t^{th} iterate of an algorithm to update ω with g fixed. We can approximate $g(\omega^T X_i)$ using

$$g(\omega^T X_i) \approx g(\omega_{(t)}^T X_i) + g'(\omega_{(t)}^T X_i)(\omega - \omega_{(t)})^T X_i$$

- Therefore we can write

$$\sum_{i=1}^{n} \Big\{ Y_i - g(\omega^T X_i) \Big\}^2 \approx$$

$$\sum_{i=1}^{n} g'(\omega_{(t)}^T X_i)^2 \left\{ \left(\omega_{(t)}^T X_i + \frac{Y_i - g(\omega_{(t)}^T X_i)}{g'(\omega_{(t)}^T X_i)} \right) - \omega^T X_i \right\}^2 .$$

Projection pursuit regression

- Interestingly this is simply a weighted least squares problem to solve for ω
  - The responses are $\omega_{(t)}^T X_i + \frac{Y_i - g(\omega_{(t)}^T X_i)}{g'(\omega_{(t)}^T X_i)}$
  - The inputs are the X_i's
  - The weights are $g'(\omega_{(t)}^T X_i)^2$
- Can use standard software for weighted least squares to solve this step
- Can iterate between this step and the previous step until convergence

Additional considerations

- This is how the problem is solved for $M = 1$
- When $M > 1$ we first fit the first function in this manner, then solve for (g_2, ω_2) conditionally on that first step to capture any additional signal, and so on
  - Forward stagewise model building
  - Similar to boosting
- The number of derived features M is an important tuning parameter
  - Stop when additional functions don't improve model fit substantially
  - Use cross-validation to find M

Difference between PPR and neural networks

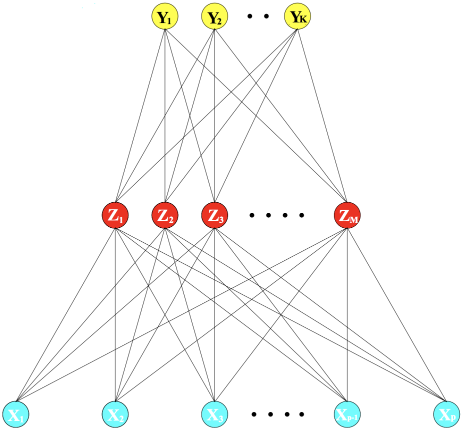
- Neural networks and PPR are very similar, but have a couple of key differences
- Neural networks do not estimate $g_m(\cdot)$ functions, but rather choose them a priori
- This would seem to be a restriction
    - Learning the functions adaptively should improve performance
- Neural networks use a much larger M typically
    - PPR usually restricts to M between 5 and 10
    - Neural networks will use much more

Neural networks

- For neural networks we need to adopt some new terminology
- We refer to our covariates as inputs, X_j
- The outputs are given by $Y_1, \ldots, Y_K$
  - For regression with a single response, we only have $K = 1$
  - For classification where the outcome has K classes, we have K outputs, each corresponding to a $0/1$ variable denoting membership into a particular class
- Derived features Z_m are called the hidden layers, because they are not directly observed
  - They are functions of our inputs

Neural networks

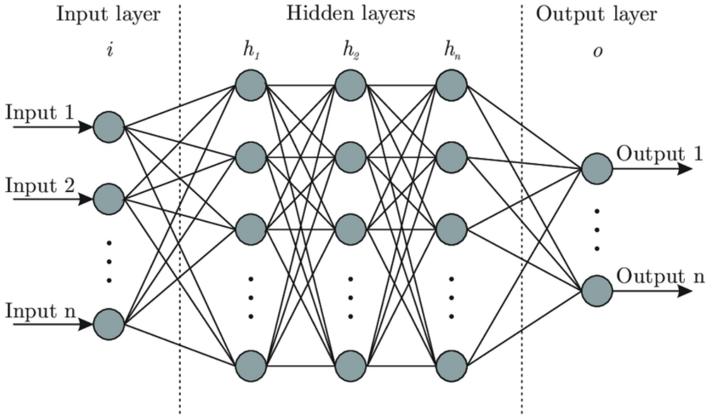
- This can be seen easily via a diagram
  - This represents a neural network with only one hidden layer



Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning. New York: springer.

Neural networks

- Can also have many hidden layers, but we will restrict attention mostly to the single hidden layer case



Neural networks

- Mathematically, we can write the single hidden layer model as follows

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X), \quad m = 1, \ldots, M$$

$$T_k = \beta_{0k} + \beta_k^T Z, \quad k = 1, \ldots, K$$

$$f_k(X) = g_k(T), \quad k = 1, \ldots, K$$

- This looks very similar to projection pursuit
  - We have replaced $g_m(\cdot)$ with a pre-specified function $\sigma(\cdot)$
  - $\sigma(\cdot)$ is called the activation function

Neural networks

- The most common choice for $\sigma(\cdot)$ is the sigmoid function

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

- Our outputs T_k are functions of our nonlinear derived features
- The final line of our model translates our outputs T_k to the scale of our outcome
  - For regression and continuous outcomes, we set $g_k(T_k) = T_k$
  - For classification, we can use the softmax function that gives us probabilities of class membership that sum to 1

$$g_k(T_k) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}$$

Neural networks

- This looks like a very fancy and complicated model, but ultimately all we have is a very nonlinear and non-additive regression model
- By setting M large, we can capture an extremely wide range of possible functions using this formulation
- Additional complexity can be added by including more hidden layers
  - Important tuning parameter
  - Many ways to add more layers
  - Fitting the best neural network is something of an art

Estimating neural networks

- There are a huge number of parameters we have introduced, which we can denote by θ
- These consist of

$$\{\alpha_{0m}, \alpha_m; m = 1, \ldots, M\}; \qquad M(p+1) \text{ weights}$$

$$\{\beta_{0k}, \beta_k; k = 1, \ldots, K\}; \qquad K(M+1) \text{ weights}$$

- For continuous outcomes we aim to minimize

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{n} \{Y_{ik} - f_k(X_i)\}^2$$

- Categorical outcomes can use cross-entropy (not covered here), but all other ideas apply directly

Estimating neural networks

- We will use a method called gradient descent to optimize this function
    - Or it's stochastic extension
- We don't want the value of θ that is the global minimizer of $R(\theta)$
    - This solution would be overfit
- Regularization is induced in one of two ways
    - Early stopping of the optimization algorithm (indirect penalty)
    - Applying a penalty term

Brief overview of gradient descent

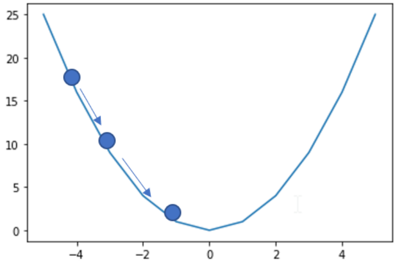
- Gradient descent is an iterative method for finding local minima or maxima of a differentiable function
- If we are currently at $\theta_{(t)}$ then we set

$$\theta_{(t+1)} = \theta_{(t)} - \gamma R'(\theta_{(t)})$$

- The main idea is to move θ in the opposite direction of the derivative as this will take you towards the minimum
- This is easiest to see visually in one dimension

Brief overview of gradient descent

- If we start to the left of the minimizing value the derivative is negative
- Therefore we move to the right to get closer to the minimizing value
    - Repeat this until convergence



Neural networks

- Neural networks use gradient descent to find the weights in the model
- In practice they use stochastic gradient descent which speeds up computation
    - Randomly select a subset of data points to use when calculating the gradient
- One form of penalization is to stop this algorithm before convergence
- This seems like an odd way to induce regularization, but in practice it does well at shrinking the model towards linearity
- In special cases there is an explicit connection between early stopping and ridge regression

Neural networks

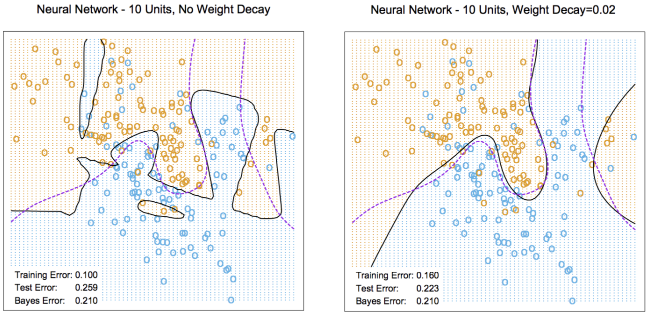
- Weight decay is another approach to regularizing neural networks
  - Similar to more standard penalization approaches
- Now we aim to minimize

$$R(\theta) + J(\theta) = R(\theta) + \lambda \sum_{k,m} \beta_{km}^2 + \lambda \sum_{m,l} \alpha_{ml}^2$$

- This places a ridge penalty on the weights and shrinks them towards zero
- The same gradient descent approaches apply with this penalty

Neural networks

- Here we see on an example that the weight decay can substantially improve model fit



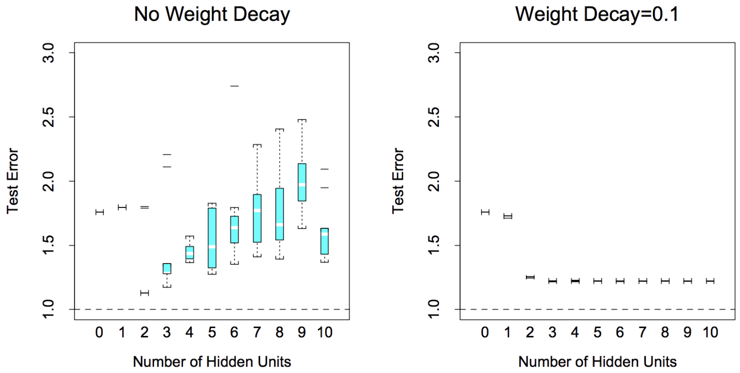
Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning. New York: springer.

Practical issues with neural networks

- Starting values can heavily influence the performance
    - Different starting weights lead to different solutions due to local minima
    - Typically random values near zero are used
    - Overly large starting values lead to poor performance
- Standardize the inputs before running the algorithm
    - Makes it easier to assign starting values
    - Penalizes all inputs equally
- How many hidden layers / nodes does the neural network have?
    - Using too many is better than too few as long as appropriate regularization is used

Simulated example

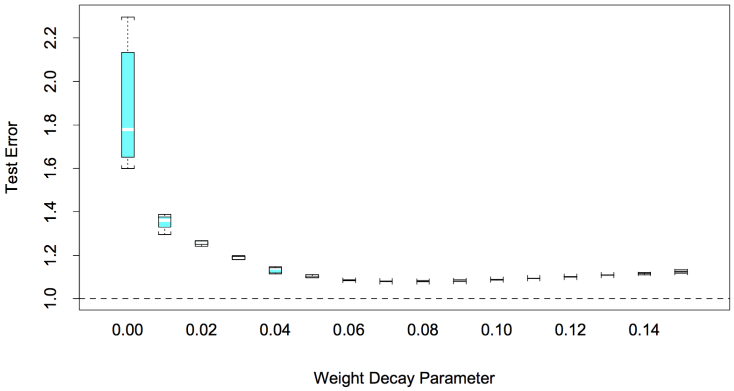
- Here is a simulated example comparing weight decay and no weight decay for a single hidden layer neural network
  - Box plots show the variability across different starting values



Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning. New York: springer.

Simulated example

- Clearly weight decay has a big impact on the results



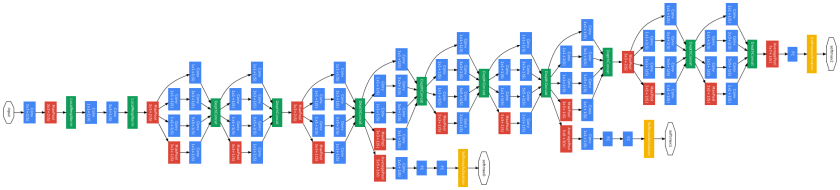
Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning. New York: springer.

Neural networks

- There are many types of neural networks with varying complexities that are tailored towards certain tasks
- Deep neural networks have many hidden layers, each with potentially many hidden nodes
- Convolutional neural networks are popular for certain uses of neural networks
  - Image classification
  - Pattern recognition

Neural networks

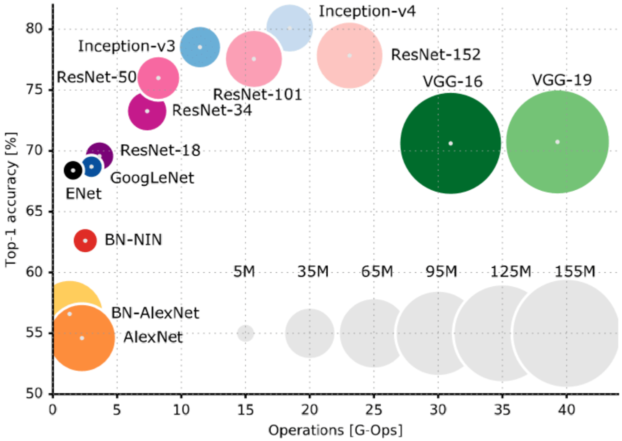
- Many neural networks in practice have a very large set of nodes and hidden layers
- Below is an example of one such network that has 22 layers



Going deeper with convolutions, Szegedy et al. (2015)

Neural networks

- Neural networks with incredibly large numbers of parameters are now being built
  - Some neural networks have over a billion parameters!

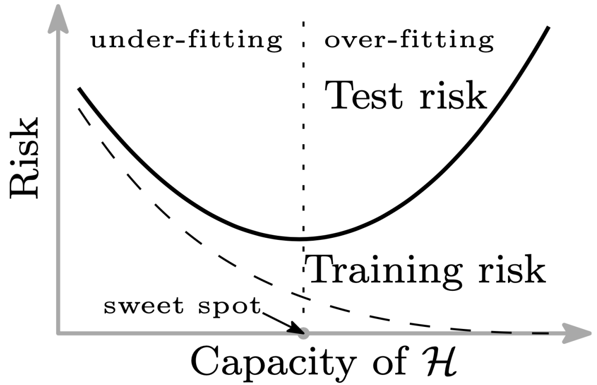


Neural networks

- You might be asking yourself how does this all work
- How can we have so many parameters in our model and avoid overfitting
- Everything so far in our class suggested a bias variance trade-off
  - Too few parameters can lead to bias
  - Too many leads to overfitting and bad generalization to new data
- Interestingly, many neural network models perfectly interpolate the data
  - Having training error zero

Neural networks

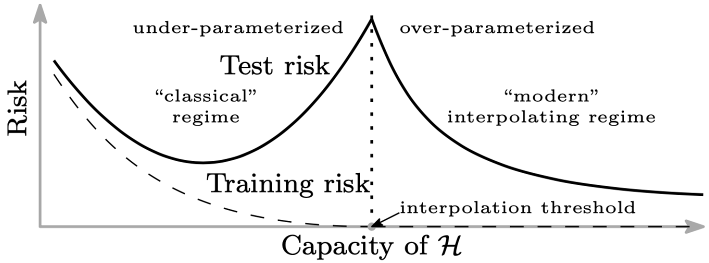
- That goes against what we have seen in class, where we usually observed the following



- Training error rates of zero are usually not ideal for a model

Neural networks

- Amazingly, if we go to more complex models that still have training error rates of zero, we can sometimes get improved testing error rates!
  - Called double descent
  - Which descent is lower depends on the specific problem
- The interpolation threshold is the point at which training error becomes zero

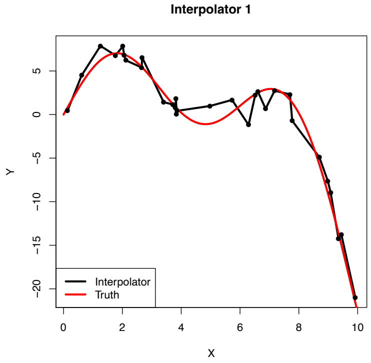


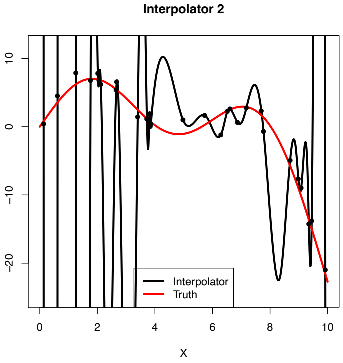
Neural networks

- Whaaaaaaaat?
- How can it be that these models with training error zero that perfectly interpolate the training data will generalize well to new data sets?
- How good or bad an interpolating model is depends on a few things
  - How smooth is the interpolator
  - What is the signal to noise ratio in your model

Interpolators

- Clearly some interpolators are better than others
  - The left one is actually quite good here



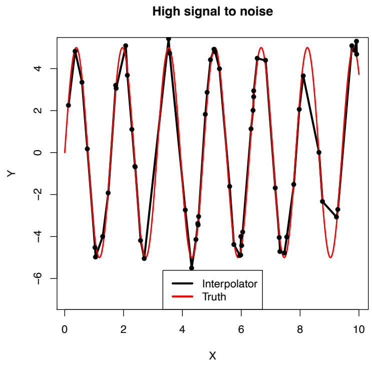


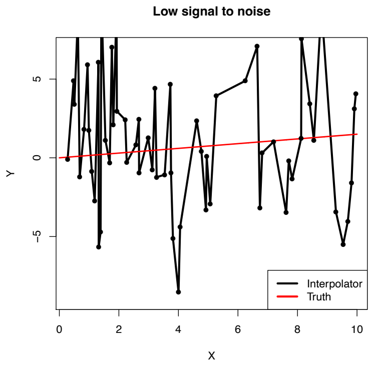
Interpolators

- By considering increasingly complex models, we are considering many possible interpolating functions
- If we choose the one that is the smoothest, then we might end up with a good function that will generalize well
- Whether or not this will outperform simpler models depends on the context
    - Let's highlight now some examples with different signal to noise ratios

Interpolators

- Signal to noise ratio makes a huge impact on the performance of interpolating functions
  - High SNR leads to interpolators that generalize well



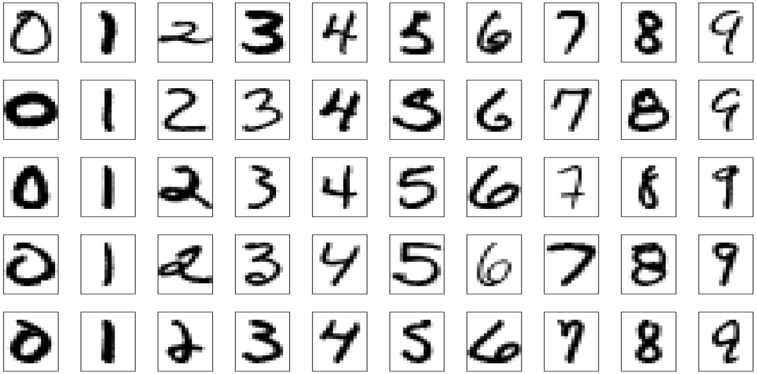


Neural networks

- Neural networks and related complex models tend to work well in situations with a fairly large sample size and high signal to noise ratios
- Can handle extremely complex, nonlinear prediction problems
- Only useful for prediction
    - Neural networks are a black box
    - Don't help us understand the physical processes underlying the data

Neural networks

- Can handle very non-standard types of prediction problems
- A classical data set aims to classify zip code numbers from handwritten numbers



Hastie, T., Tibshirani, R. and Friedman, J. (2009). The elements of statistical learning. New York: springer.

Neural networks

- This image classification is one with a very high signal to noise ratio
    - Clearly we can look at these numbers and classify them correctly almost 100% of the time
- Getting a model to classify this well is a difficult task, but one that neural networks are well suited to
    - The input data are the pixels of an image
    - Correct classification relies on extracting features from these pixels that help in classification
    - Likely these are nonlinear and complex functions of the pixels
- Neural networks have achieved classification rates of over 99% for this problem!

Neural networks

- In summary, neural networks are a very powerful tool in certain situations
- There are many decisions to be made when making and training a neural network that influence performance
- They are not an all encompassing solution for every problem
    - In many cases, simpler models are easier to fit and work better
- We as statisticians, computer scientists, etc. are still trying to better understand these mathematically to gain insight into exactly how they work