Neural Nets, Other Fancy Algs

Lecture 17

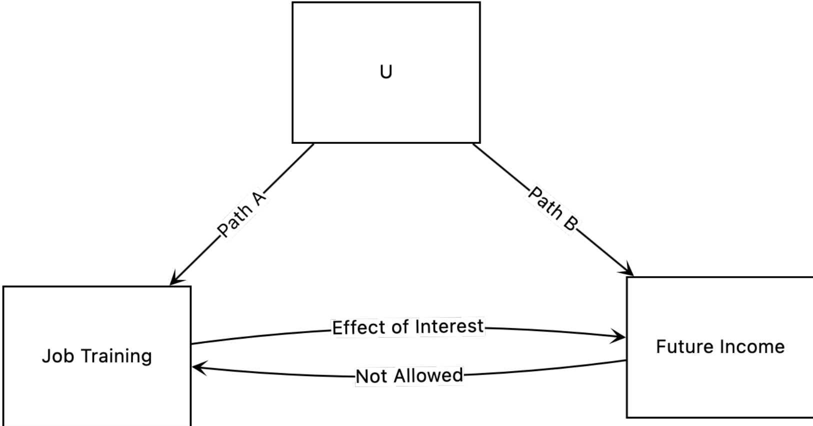
Connor Dowd

May 25th, 2021

Today's Class

1. Review
   - Targeting
   - Observational Methods
2. Neural Nets
   - Basic Idea
   - Activation Functions
   - Stochastic Gradient Descent
   - Backprop
   - Example
3. Other Aggressive ML things:
   - GANs
   - Evolutionary Algos

Review: Causality



Review: RCT

Randomization fixes this problem. It gives us the ability to say that no “unknown” factors drove the variable “X” which is “treatment status” – *because* we know the randomization drove it.

Review: TEs

An average treatment effect (ATE) is the average difference between the potential outcome under treatment, and the potential outcome under control.

Individual treatment effects are the *individual's* difference between potential outcomes under treatment and control.

Conditional Average Treatment Effects, are the average difference btween potential outcome under treatment and control *for some subgroup*

Review: Targeting

Targeting mostly consists of trying to identify groups with notable (large, negative, etc) CATEs.

Many methods. We've looked at "T-learners" – which make predictions about individual outcomes under treatment and control, then estimate differences.

With estimated treatment effects for individuals, we can decide who to *target*.

This needs solid out-of-sample performance to be useful.

- If you want to learn more – look into "Causal Forests"
  - Which take a slightly different approach.

Review: Observational Methods IV/RD

Some observational methods solve the basic problem of causal inference by trying to find something which is "as-if" random.

- Instrumental variables: find something that is unrelated to Y, but can drive X
- Regression Discontinuity: take position relative to a threshold as semi-random *for individuals in the vicinity of the threshold*

Both rely on the disconnect between outcome Y and "randomness". If there is a link between Y and the "as-if random" variable, there will be problems. We cannot prove this assumption *true*, we can only sometimes prove it false.

Review: Observational Methods DiD/SCM

Other observational methods try to counteract any bias that might exist.

- Diff-in-Diff establishes two primary sources of problems, and tries to eliminate them.
  - Can also upgrade to triple-diff, or Diff-in-RD, etc.
- SCM tries to improve on weighting schemes in DiD.

Both of these rely on other *untestable* assumptions. “Parallel trends”, etc.

Neural Nets

Basic “idea” is to build a structure ‘similar’ to a brain.[1]

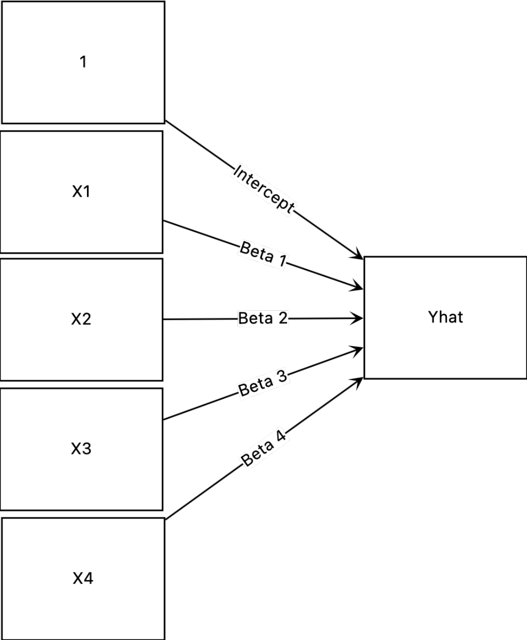
Brains are ‘good’ at predictive tasks, so maybe we can borrow their structure.

Stylized model:

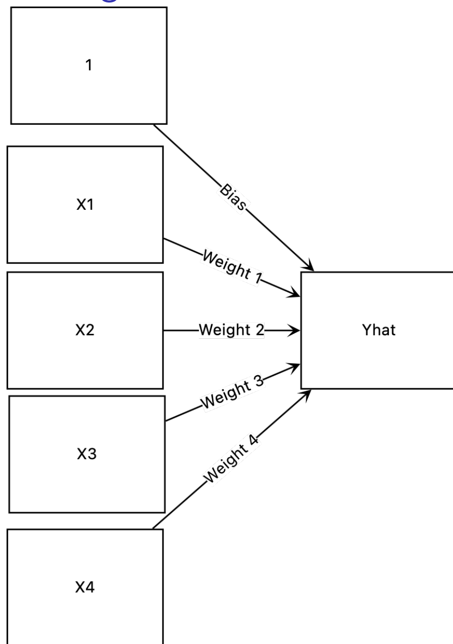
- Each neuron in the brain has some neurons that signal it and some which it signals
- Each neuron can be ‘inactive’ or activate (with some room for “intensity”)

[1] or a computer scientist’s idea of a brain

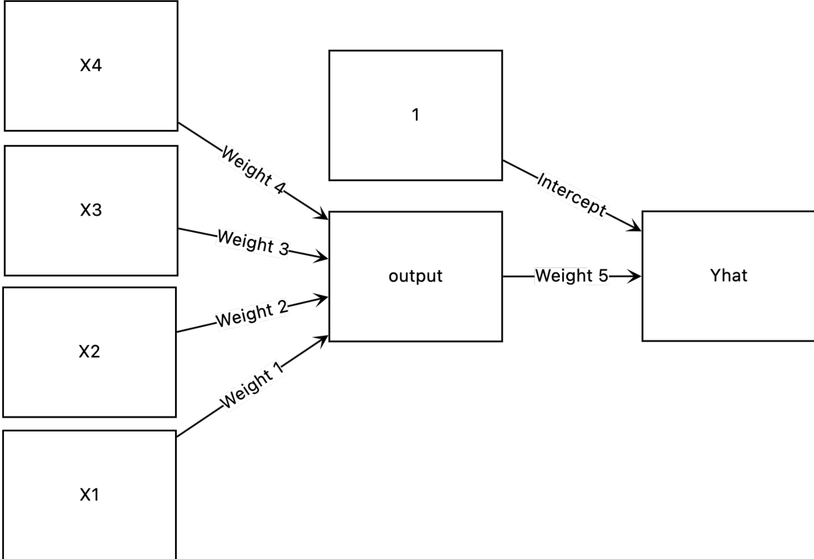
Linear Regression Diagram



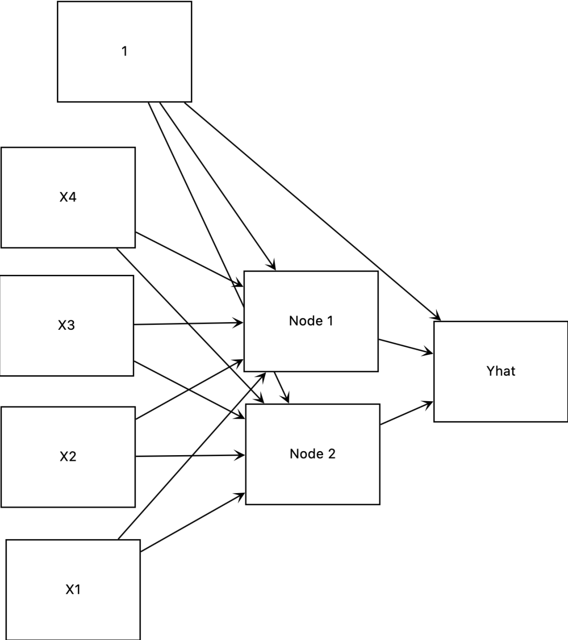
Linear Regresion New Names



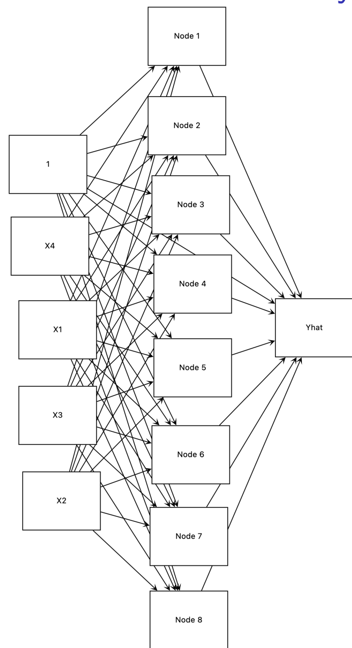
Neural Nets - Simplest



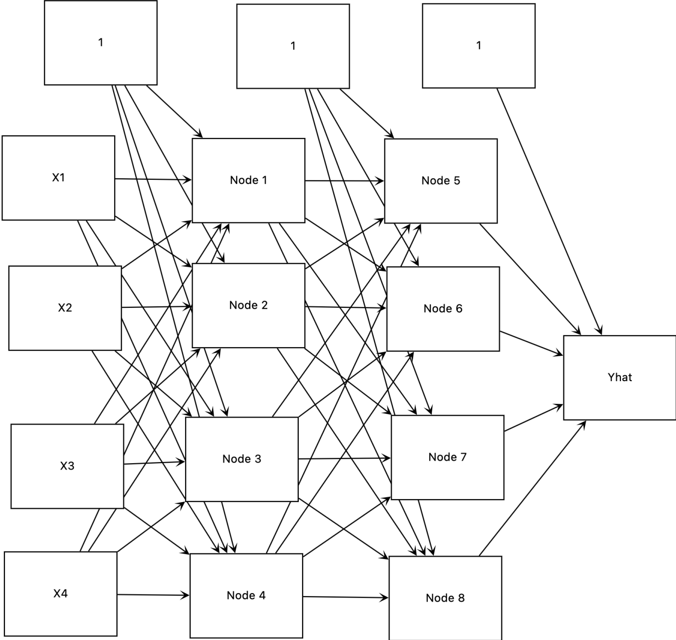
Neural Nets - Small Hidden Layer



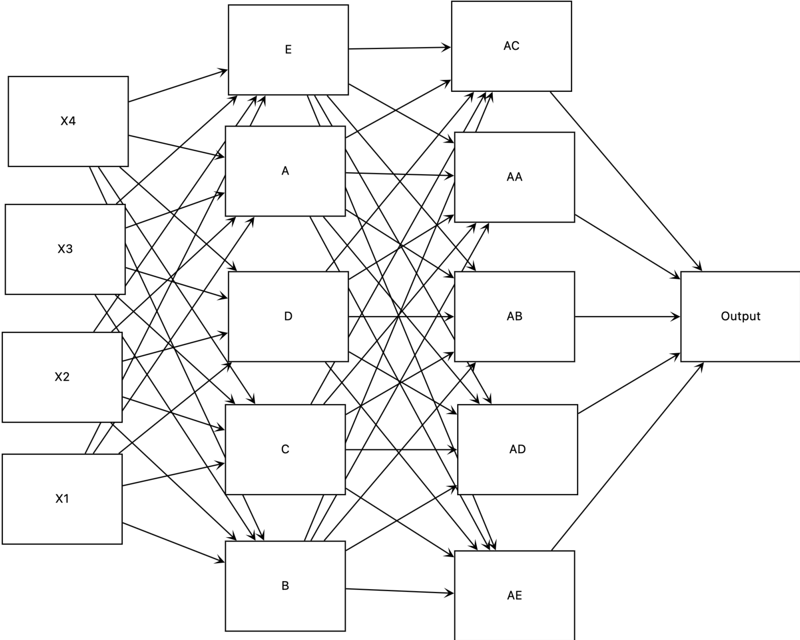
Neural Nets - Hidden Layer



Neural Nets - Two Hidden Layers



Neural Nets - Essence



Node Math

At each node, the neural net sums up its inputs (weights times the outputs from prior layer).

Then it passes that to an activation function to produce its own output. A common activation function is:

$$ReLU(x) = max(x, 0)$$

Activation Functions

Other common activation functions are sigmoid:

$$S(x) = \frac{1}{1 + e^{-x}}$$

Leaky ReLU:

$$LReLU(x) = max(0.01 * x, x)$$

Other Activation Functions

SoftMax

$$SoftMax(\mathbf{X})_i = \frac{e^{\mathbf{X}_i}}{\sum_{j=1}^{J} e^{\mathbf{X}_j}}$$

There are many more.

Node Math

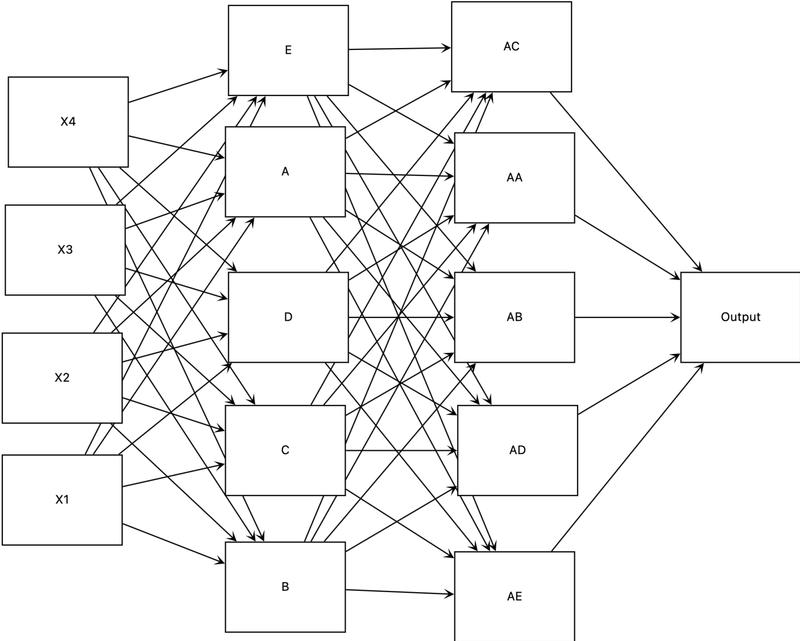
Each node gets a bunch of inputs X, a bunch of weights w, and spits out a single number. That number is a function of the vector Xw.

We apply the activation function either to the sum $\sum Xw$ or the full vector Xw.

Nodes

We repeat this at each node until we have a prediction.

Neural Nets



Estimating Neural Nets

For a given structure (layer setup, activation function), we need to select the weights that are best. **This is tricky**

Suppose we have:

- 4 variables
- Two hidden layers of 5 nodes
- A bias term for each layer.

We need to estimate: $(4+1)*5+(5+1)*5+(5+1)=61$ parameters. This number grows rapidly with the number of variables, the number of layers, and the number of nodes per layer.

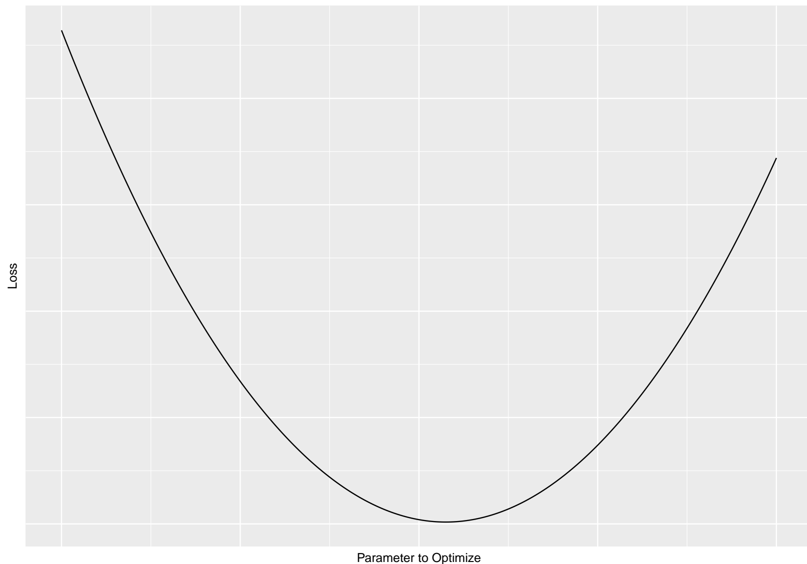
Estimating Neural Nets

Not only are there *many* parameters to estimate, there is no known closed form to solve this general problem.

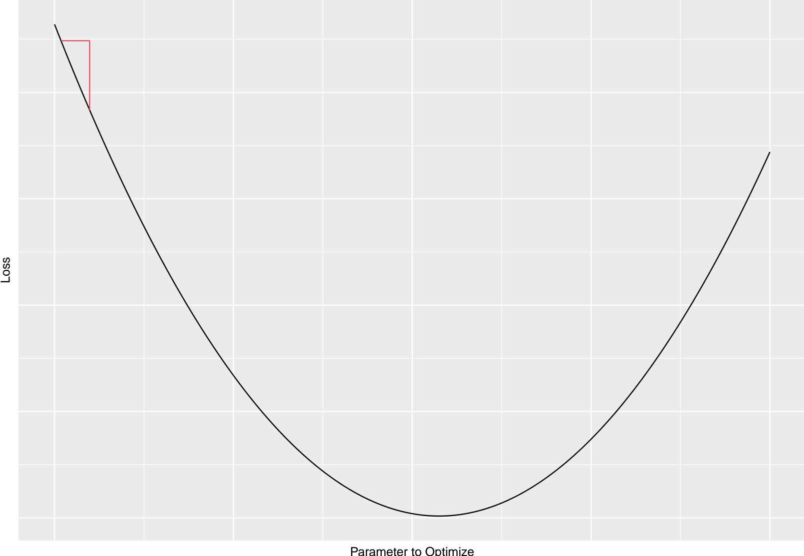
$\implies$ there is no analytical solution, so we need to use a computational optimization routine.

- Backpropagation

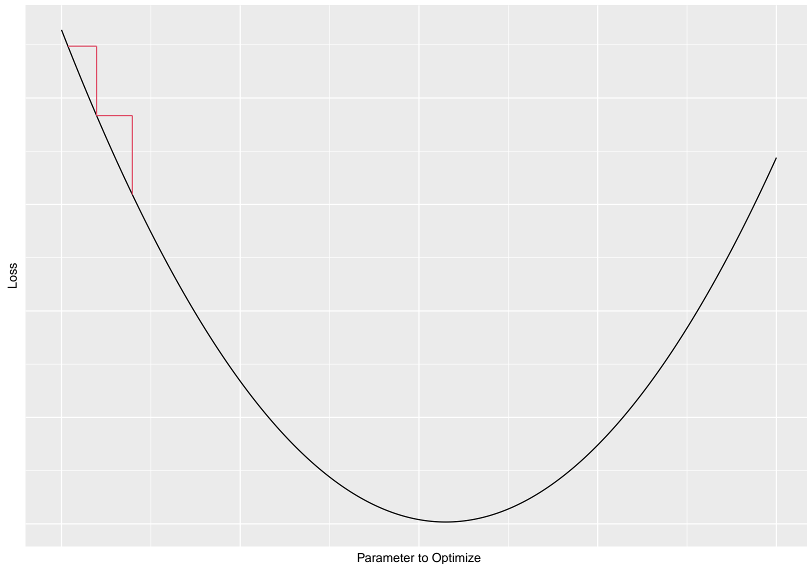
Optimization Routines, A brief detour



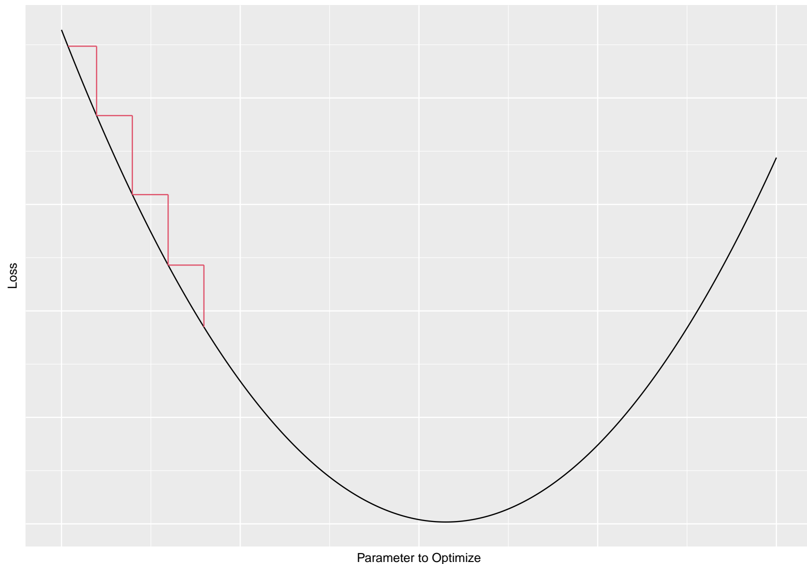
Optimization



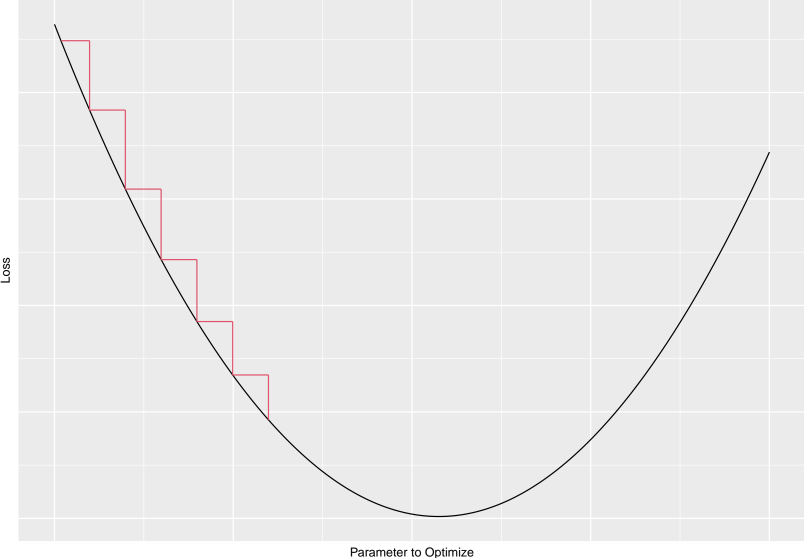
Optimization



Optimization



Optimization



Optimization - Pt 2

But how do we know which direction to go?

We use the data to estimate the slope whenever we stop. This tells us which way is "downhill".

- Gradient Descent

However, this can be very slow.

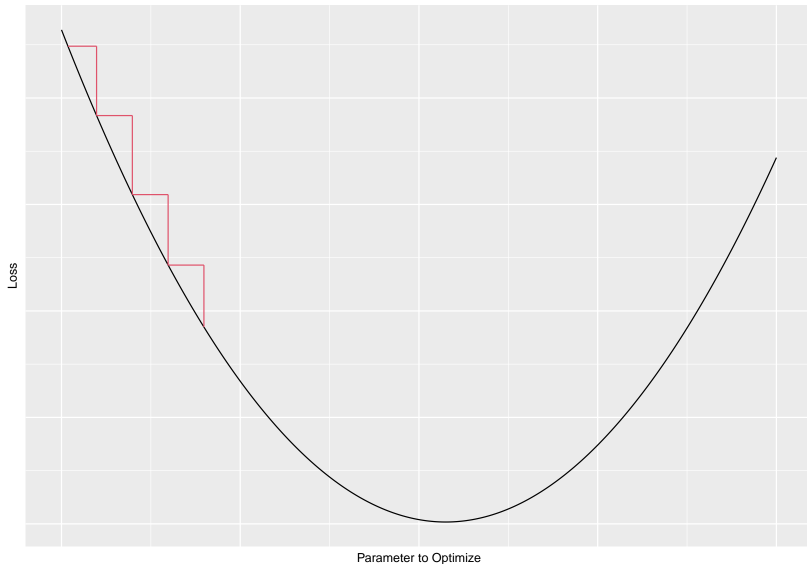
Can we speed up?

What if we only use *some* of the data to estimate the slope whenever we stop?

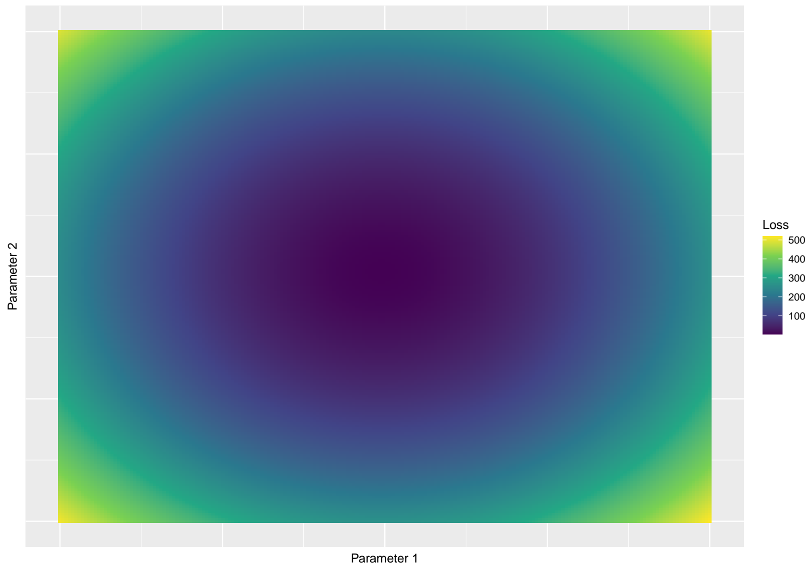
"Stochastic Gradient Descent"

- Faster *per iteration*
- Somewhat more erratic
  - Need more iterations?
- Usually faster overall anyhow

Optimization

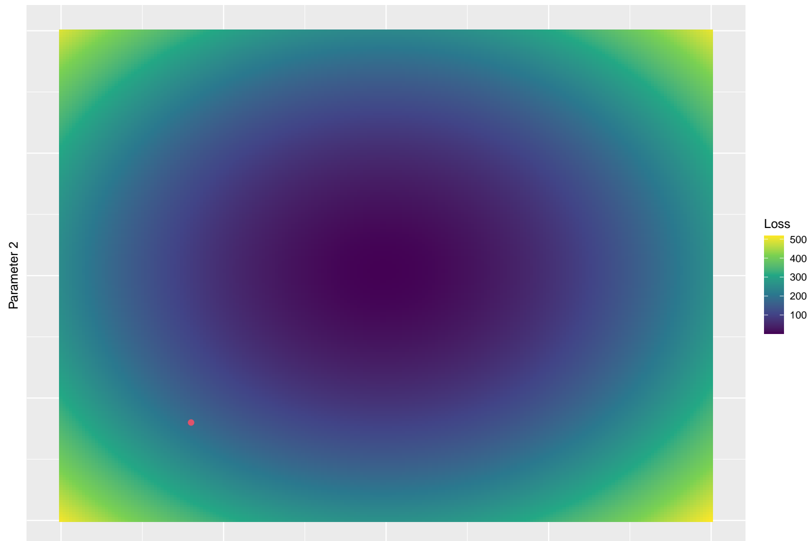


Optimization

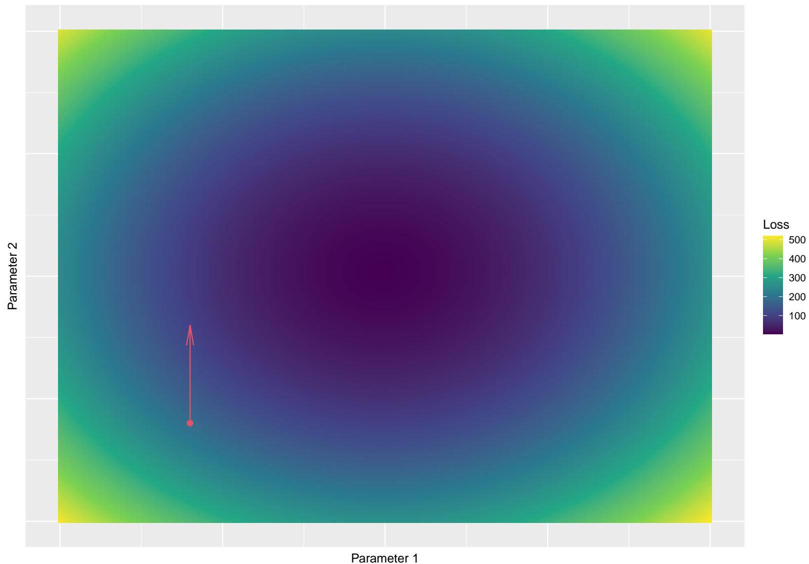


Optimizing in 2 Dimensions

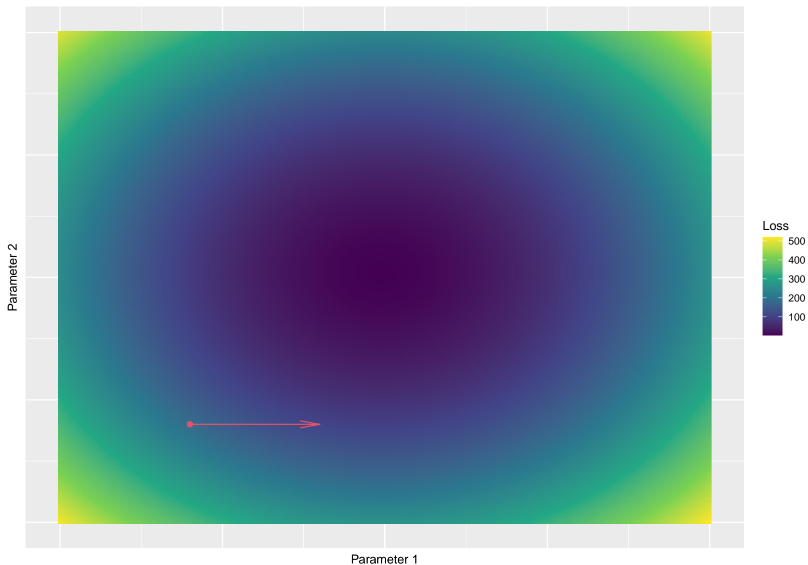
```
p2 = p1 + annotate("point",x=-3,y=-3,col=2,size=2)
p2
```



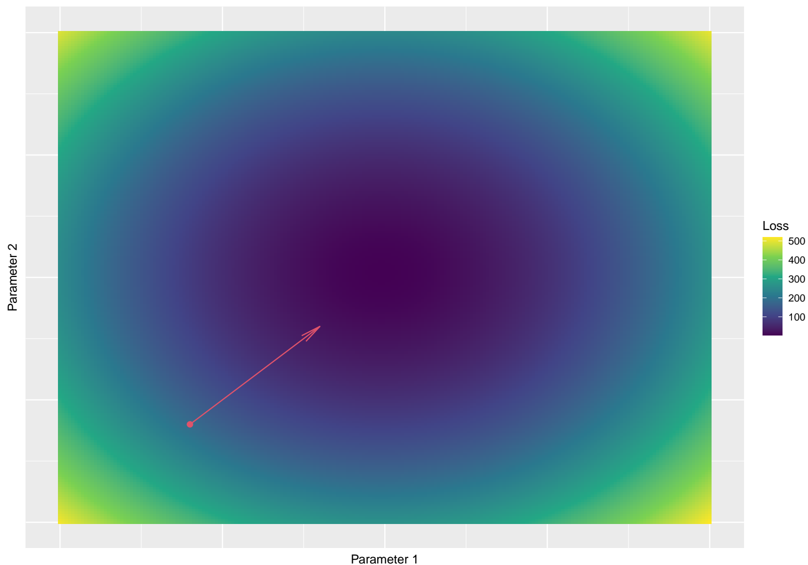
Optimizing in 2D



Optimizing in 2D



Optimizing in 2D



Optimizing in 64 dimensions

Each derivative we want to compute is more work.

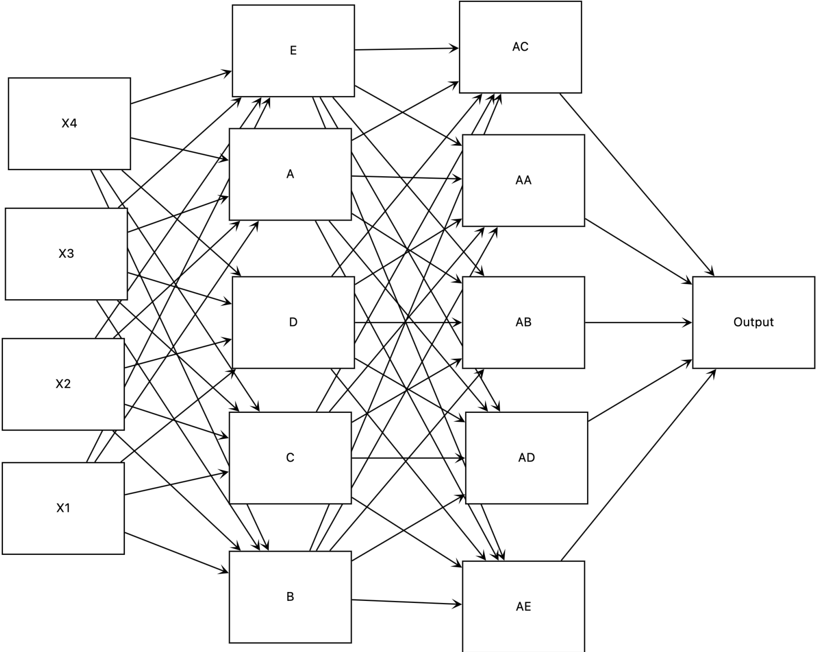
But beyond that, for linear regression, derivatives are straightforward.

The derivative

$$\frac{\partial \hat{Y}}{\partial \beta_1} = X_1$$

And the error derivatives are similarly easy.

Neural Nets



Backpropagation

The idea behind backpropagation is that we can understand the effect of the last set of nodes on our errors. Those derivatives are easy to compute.

And the derivatives for the second to last set of nodes depend *solely* on the last set of nodes. And so forth.

- We can "propagate" the error terms backwards from the output to the weights at the very beginning.
  - I promise this is a bigger reveal than it seems.
    - Go read more if you want more depth.
- This is backpropagation

Backpropagation

Backpropagation allows us to compute the derivatives of our error with respect to weights arbitrarily deep in our neural net.

This is *critical* for our ability to optimize those weights, and thus for us to estimate them.

Iris Data

```
iris = as_tibble(iris)
iris
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Spe
##           <dbl>       <dbl>        <dbl>       <dbl> <fo
##  1          5.1         3.5          1.4         0.2 set
##  2          4.9         3            1.4         0.2 set
##  3          4.7         3.2          1.3         0.2 set
##  4          4.6         3.1          1.5         0.2 set
##  5          5           3.6          1.4         0.2 set
##  6          5.4         3.9          1.7         0.4 set
##  7          4.6         3.4          1.4         0.3 set
##  8          5           3.4          1.5         0.2 set
##  9          4.4         2.9          1.4         0.2 set
## 10          4.9         3.1          1.5         0.1 set
## # ... with 140 more rows
```

Neural Nets - Benchmarking

```
holdout_ind = sample(nrow(iris),0.2*nrow(iris))
test = iris[holdout_ind,]
train = iris[-holdout_ind,]
library(ranger)
forest = ranger(Species~.,data=train)
pred_forest = predict(forest,data=test)$prediction
mean(pred_forest == test$Species)
```

```
## [1] 0.9333333
```

Basic Neural Nets

```
library(neuralnet)
nn0 = neuralnet(Species~.,data=train,hidden=0)
pred_nn0 = predict(nn0,newdata=test)
pred_nn0
```

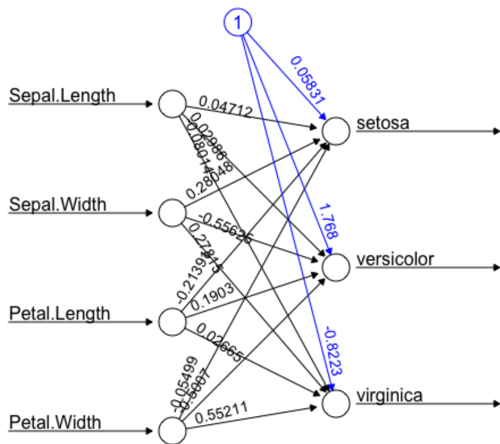
```
##                [,1]        [,2]        [,3]
##  [1,] -0.452203089  0.71351634  0.73886700
##  [2,]  1.106475284 -0.19269222  0.08652068
##  [3,]  1.011263108  0.01050910 -0.02198016
##  [4,]  0.820956863  0.45901557 -0.28007150
##  [5,]  0.832333881  0.42532347 -0.25779696
##  [6,]  0.225195432  0.38041967  0.39460787
##  [7,] -0.143312523  0.38277284  0.76072814
##  [8,]  0.883294918  0.13033955 -0.01358311
##  [9,] -0.086419263  0.10806558  0.97878385
## [10,]  1.009936748  0.13313195 -0.14278820
## [11,]  0.156939609  0.71857188  0.12416082
## [12,]  0.824137524  0.33243775 -0.15681579
```

Basic Neural Nets

```
class_index = apply(pred_nn0,1,which.max)
class_nn0 = levels(iris$Species)[class_index]
mean(class_nn0 == test$Species)
```

```
## [1] 0.8666667
```

Basic Neural Nets



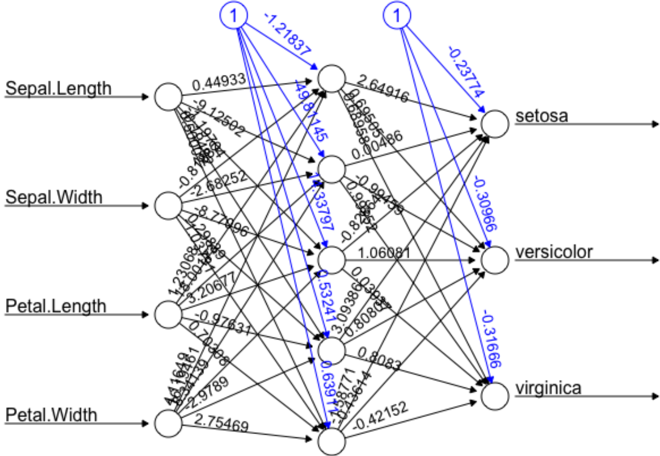
Error: 16.207105   Steps: 4987

Bigger NN

```
nn1 = neuralnet(Species~.,data=train,hidden=5)
pred_nn1 = predict(nn1,newdata=test)
class_index = apply(pred_nn1,1,which.max)
class_nn1 = levels(iris$Species)[class_index]
mean(class_nn1 == test$Species)
```

```
## [1] 0.9333333
```

Bigger NN

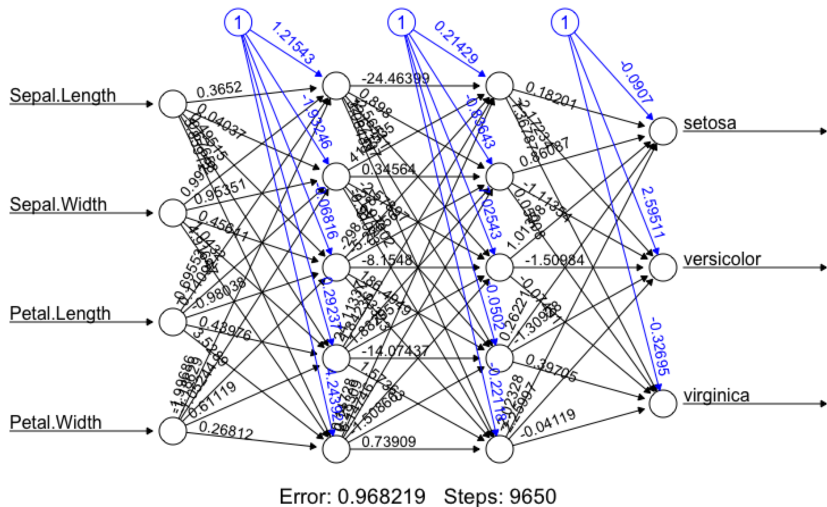


Two Layers

```
nn2 = neuralnet(Species~.,data=train,hidden=c(5,5))
pred_nn2 = predict(nn2,newdata=test)
class_index = apply(pred_nn2,1,which.max)
class_nn2 = levels(iris$Species)[class_index]
mean(class_nn2 == test$Species)
```

```
## [1] 0.9666667
```

Two Layers



Multinomial Deviance

A somewhat approximate function.

```
multinom_deviance = function(probs,actual,epsilon=0.00001)
  out = as.integer(actual)
  #Grab the probability corresponding to the event that hap
  prob_event = sapply(1:length(actual),
                      function(index) probs[index,out[index
  #Make sure the probability is between 0 and 1.
  if (epsilon) {
    prob_event[prob_event >= 1] = 1-epsilon
    prob_event[prob_event <= 0] = epsilon
  }
  -2*sum(log(prob_event))
}
```

OOS Deviances

```r
multinom_deviance(pred_nn0,test$Species)
```

```
## [1] 18.98888
```

```r
multinom_deviance(pred_nn1,test$Species)
```

```
## [1] 29.15948
```

```r
multinom_deviance(pred_nn2,test$Species)
```

```
## [1] 23.82508
```

Tuning Parameters

Like forests, trees, LASSO, KNN, and other models we've encountered, Neural Nets have several "tuning parameters" we need to pick in order to use the model.

- Forests: number of trees, tree depth, variable dropout rate
- Trees: tree depth
- LASSO: penalty parameter
- KNN: Number of nearby points to examine k
- Neural Nets: Number of hidden layers, connections between hidden layers, activation functions, nodes per layer, training thresholds, and more.

Choosing Tuning Parameters

As with the other models, we need some way to choose these parameters.

Once again, if we care about out-of-sample performance, we should use. . .

- Cross-validation.

There are (again) a wide variety of theoretical tools you could use. But for most settings, CV will do as well – and is an excellent swiss army knife.

Other ML algorithms

As should be transparent, these models can become arbitrarily complicated.

> *Why not add 10 more layers and 10 more nodes to each layer?*

Two problems become extremely important in ML

- How do I find good parameter estimates?
- How do I prevent overfitting?

Optimization

The first question is about the mechanics of an actual optimization routine.

The second question is about preventing that routine from going to far.

- All about optimization.

Optimization Routines

There are a variety of techniques for helping with this.

- Generative Aversarial Networks (GANs)
- Evolutionary Algorithms
- Etc

GANs

Essence:

1. Build a model that generates fake data
2. Build a model that tries to detect fake data
3. Have them compete, determine performance of each.
4. Iterate and repeat

- Each model has incentives to understand DGP well.
- Over 1000s of iterations, will converge to something that works well

Evolutionary Algorithms

Essence:

1. Build 1000 models. Test performance.
2. Remove the worst 200, duplicate the best 200. (Selection)
3. Transplant some features across different models ()
4. Randomly change some features of each model (Mutation)
5. Repeat

- Again – we are creating incentives for model performance. Over 1000s of iterations, can find things that work well.

Optimization

Optimization/estimation is a key element of more complicated models.

> *Linear models are everywhere because they are easy to estimate.*

The two models above are based on some of the most successful methods of optimization – evolution and competition.

- Used by well known entities like nature, capitalism, and science

Wrap up

Homework 7 is due tomorrow.

On Thursday I'll try to review the whole of this course in 90 minutes.

See you Thursday!