**Lecture 1: The Learning Problem**

* I like this definition on the essence of machine learning:
  + 1. A pattern exists.
  + 2. We cannot pin it down mathematically.
  + 3. We have data on it.
* Reverse engineering the system… a good concept.
* Netflix Challenge is a good practical example.
* Every time you have a model
  + There is a hypothesis set, and there is a learning algorithm with which you implement
* The Perceptron Learning Algorithm:
  + Pick a misclassified point and update the weight vector
  + At iteration t = 1,2,3… pick a misclassified point from (x1, y1)…(xn,yn) and run a PLA iteration on it.
* The basic premise of learning:
  + Def: “Using a set of observations to uncover an underlying process.” Broad premise 🡪 many variations
  + Language is a FANTASTIC example of supervised/unsupervised
  + Review of reinforcement learning:
    - Instead of (input , correct output) we get (input, SOME output, grade for this output)
    - Example: playing games!
* Relation to statistics:
  + Statistics is a machine learning problem where the result is a probability distribution field

**Lecture 2: Is Learning Feasible?**

* Review of supervised learning terminology
  + Learning algorithm picks g (similar to) f from a hypothesis set H. The actual target function f(x) is unknown, and we’re just finding a function g from H that can approximate f.
  + Machine learning is impartial as to whether a mathematical formula *could* exist or not. Even if there is a mathematical function that is the best fit, there is no reason that an algorithm cannot approximate it.
  + Can we learn an unknown function? NO! The function can assume any value outside the data we have. Couldn’t the function assume any value it wants, if it’s truly unknown.
* Hoeffding’s Inequality
  + P[|v- u| > $ ] <= 2e^(-2($^2)N))
  + We’re saying that v is approximating u, but in reality it’s more u approximates v.

**Lecture 3: Linear Modeling**

* Disadvantages of the perceptron learning algorithm:
  + It can fluctuate GREATLY as you increase the samples and move the hypothesis h, since it’s moving the model by a weight.
* Modifications of perceptron: The ‘pocket’ algorithm
  + Because E(in) has all values known, you only keep the best iteration of the perceptron learning algorithm. So you only keep the hypothesis that works best. That seems to work very, very well.
* So what he did is do linear regression to discover best fit weights and use that on the Pocket algorithm as the initial weights.
* The problem with linear regression: sets whose weights are far off can skew the results. But it can be a good initial value for the perceptron algorithm.
* Limitatations of linear:
  + Circles, elipses, hyperbolas… etc can’t be used because they’re not linear!
  + The weights might not be linear. Example: diminishing returns on years in residence on credit… after 5 years it doesn’t really matter. So it’s not linear!
  + Linear classification, remember, is just determined by the sign of the weight combination
  + Algorithm only works because of LINEARITY in the weights.
* Nonlinear transformations (intro): Transforming the data nonlinearly. The nonlinear features will have a linear dependency.
  + Example: circles. Quadratics. This is essentially what we’re doing in nonlinear examples… it’s a linear transformation. Then once the transformation is made you can work linearly.
  + BUT there is a catch. (in the next lecture, he’ll explain it.) We’ll go over it after the theory of generalization.

**Lecture 4: Error and Noise**

* Example: He took x1^2 + y1^2 to get a linear transformation, so that we can apply a linear model (the perceptron among them) to the data. THEN we have to transform from the z-space to the input-space.
  + The nonlinear space is called the feature space, or Z-space.
  + The problem with moving to the z-space is that the inverse transformation is not a mapping necessarily. So you can’t simply transform every point back. You classify/interpret based on the points in the z-space.
  + This generalized to as many dimensions, so you can implement plenty of sophisticated surfaces.
* What transforms to what
  + x = (x0, x1, … xd) -🡪 z = (z0, z1, …. zd)
  + If you do the wrong transformation, you will end up with a VERY poor generalization.
  + X1, X2, … , Xn -🡪 z1, z2, …, zn
  + Y1, Y2, … , yn 🡪 y1, y2, …, yn
  + Weights: NO weights in the x-space when you do a non-linear transformation. The weights occur in the z-space. (we’ll call those w~)
  + g(x) (final hypothesis) = sign (w~T z) = sign(w~T(phi(x))
* Error measures:
  + “what does *h ~~ f* mean???”
  + Error measure: E(h,f) == Error of target function as it relates to actual function.
    - For linear models, always a pointwise definition: you define e(h(x), f(x))
    - Examples:
      * Squared error: e(h(x), f(x)) = (h(x) – f(x) ) ^2
      * Binary error: if(h(x) != f(x)] (this is exactly what we’re talking about with frequency/probability of error for classification)
    - In-sample error: E(in)(h) = 1/N SUM(n=1 🡪 N) [e(h(xn), f(xn))]
    - Out of sample error: E(out)(h) = Ex[e(h(x), f(x))] (expected value of error) Makes sense…
  + The learning diagram, revised: we get the x from the same probability distribution that generated those points. Test it from the same points drawn from the same distribution for Ein. Makes sense. So use the probability distribution to generate the training examples, and use it for error measurement.
* How to choose the error measure:
  + Types of error: false accept and false reject. (this is related to precision/recall).
  + How do we penalize each type? You can apply weights to every error type.
  + His claim: there is NO inherent merit to choosing one error function over another. All you’re really seeing is an application-specific reason for error measurement.
    - Example: error measure of fingerprinting for a discount versus fingerprinting for the CIA. The penalties for false accept and false reject *should be* TOTALLY different. The approach is normative.
    - The takeaway: the error measure should be specified **by the user.**
  + But user-specified error is not always known or knowable. Alternatives:
    - Resort to *plausible* measures. Analytic arguments that you can argue have merit. Examples: Squared error = Gaussian noise. That’s a plausible error measure.
    - Use *friendly measures*. Nice because they’re easy to use. closed-form solution. ?????
  + The error measure can minimize the final hypothesis and the algorithm used.
* Noisy targets:
  + In reality, these are the only targets that you’ll ever actually encounter.
  + The ‘target function’ is not always a *function.*
    - Example: two ‘identical’ customers with the exact same scalars in feature space can have different outputs. !!
  + So we use a target ‘distribution:’ instead of y = f(x), we use the P(y|x) . Bayesian…???
    - (x, y) is now generated by the joint distribution: P(x)P(y|x). Oh WAIT I get it.
    - Noisy target = deterministic target, + noise. So P(y|x) is not far from what we already have.
      * Deterministic target f(x) = E(y|x) plus noise y = f(x)
      * Deterministic target is a special case of a noisy target: P(y|x) is zero exept for y = f(x).
    - So instead of an unknown target function, we have an unknown target DISTRIBUTION. The target function (E(y|x)) + NOISE.
    - Distinction between f(x) and P(y|x) we get an unknown input distribution (P(x)) and we have an unknown target distribution (P(y|x)) to get the training examples: (x1, y1), … , (xn, yn)
    - We’re not trying to learn the input distribution.
    - So then we can merge P(x)P(y|x) to become P(x,y). But REMEMBER:
      * P(x, y) is NOT a target distribution. It is the result.
* Preamble to the theory of learning
  + What we know so far:
    - Learning is feasible. It is likely that E(out)(g) ~~ E(in)(g)
      * Is this learning? We need g to approximate f. What that means is that E(out)(g) ~~ 0 (the out of sample error approaches 0). Basically what the above is saying is that the out of sample error will get really close to the in sample error.
      * E(out)(g) ~~ 0 (WE LEARN WELL) is achieved through
        + E(out) (g) ~~ E(in)(g) (the in sample error approximates all error of all future examples) AND
        + E(in)(g) ~~ 0 (the error of a sample approaches 0. We get it right close to every time).
    - Learning is thus split into two questions:
      * 1. CAN we make sure that E(out)(g) is close enough to E(in)(g)? (we’re spending 2 weeks on this)
      * 2. CAN we make E(in)(g) small enough? (we’re spending 4 weeks on this)
    - “small enough” has been close to 0 for now. But it depends on your application. Ex: financial forecasting. The data is extremely noisy under ideal conditions, and the only reason we’re interested in forecasting is because the data is not entirely noisy. But there is NO way that we won’t have no noise in our data. If the error is smaller than 1/2, in this case, we’re in business. WOW.
  + What the theory will achieve:
    - Characterizing the feasibility of learning for infinite M.
    - Characterizing the tradeoff:
      * Model complexity goes up, and the Ein goes DOWN
      * Model complexity goes up, and the Eout – Ein goes UP.
      * This will set the foundation for regularization, which we’ll be doing in a bit.

**Lecture 5: Training versus Testing**

* The problem with training sets is that M 🡪 infinity for most cases (the perceptron is one of them)
  + Where did the M come from?
    - The bad events *B*m are “|E(in)(hm) – Eout(hm)| > e” : the probability of a bad situation is small.
    - The union bound: P[B1 or B2 or … or Bm]
  + Replacing M
    - We take the input space, and in the Perceptron case, we get an infinite number of h since there’s an infinite input space
    - Now let’s restrict ourselves only to the sample
    - As you vary the perceptron, you’re not going to notice until
  + Dichotomies
    - Function of h: {x1, x2… xN} 🡪 {-1, +1}
    - That’s a possible set of application of each member to the output y. So perceptrons with the same return of the function will be the same h.
    - (in a binary return, the largest number is 2^N. This is the upper bound.)]
* Let’s apply this to the perceptron
  + Mh(3)
    - You’d think the number of combinations is 2^3… and it is! We just pick the maxiumum. For now we don’t really care that there are some impossible perceptrons.
  + Mh(4)
    - There are now a couple of divisions that we cannot generate via a perceptron.
    - The growth function here is 14, not 16, since there are two divisions that you can’t make with a perceptron.
    - So the maximum of the growth function cannot be the maximum of divisions possible (2^4)… otherwise, the perceptron is limited
    - So the maximum number of hypotheses is stronger than those generated by the perceptron.
  + Mh(5), Mh(6)…
    - Yeah, it gets hard
* Examples of Hypotheses and growth functions
  + **1. Positive rays.** 
    - From a point on (at point a), all of the points larger go to +1, all the points smaller go to -1. So there’s a list of hypotheses that could occur.
    - Say we have a list of X = {x1, … , xn}. How many h < *H* can we have? What is the growth function?
      * The value of a along the segment determines the hypothesis
      * How many line segments?
      * We have N + 1 possible dichotomies. So the growth function for this is EXACTLY N +1.
  + **2. Positive intervals.**
    - Anything within that interval will be of one value in the dichotomy, anything outside it will not be.
    - How many dichotomies?
      * You choose 2 different line segments… an exact 1 to 1 mapping to dichotomies.
      * So the growth function is the number of ways to pick 2 segments from 1 segment… N+1 choose 2.
    - Mh(N) = (N+1 choose 2) +1 (for interval covering nothing)
    - = 1/2 N^2 + 1/2N + 1 (quadratic)
  + **3. Convex sets.** 
    - H is a set of convex regions… take any two points, form a convex region.
    - What is the growth function for the set of hypotheses?
      * You get a cloud of points…
      * You’re excluding a lot of dichotomies, however, because anything inside the convex region has to be the same. No donughts, basically
    - What if we arrange the points such that the maximum disjoint h is possible? Say… arrange your circles along the perimeter.
      * THEN if they’re on a perimeter, your growth function CAN be 2^N.
      * Now this is not the case for all points, of course, but we’ll just take the maximum, and pay the price for the security that we know that every set can be expressed by that maximum # of dichotomies.
      * When you get all possible dichotomies, it’s called **shattering.**
  + In general, more complex set 🡪 bigger growth function. More maximum dichotomies.
* Back to the big picture
  + If we replace the M in the inequality (the one based on LOLN) with the growth function of your points.
  + All you need to do is declare that mH(N) is a polynomial, and you are in business. Once you declare that the set of hypotheses is polynomial, you can generalize that learning is possible.
  + So can we just prove that mH(N) is polynomial?
  + **Def:** break point. The point (number of possible inputs in set) at which you fail to get all possible dichotomies.
    - **If no data set of size k can be shattered by H, then k is a break point for H.**
    - So mH(k) < 2^k
    - So for the perceptron, the break point is 4. At 4 points, there is a subset of the dichotomy that you cannot achieve.
    - The idea: “Just tell me the break point, and I will tell you the learning behavior.”
  + Examples of break points:
    - Positive rays: break point k = 2. (since there is no way for positive rays to go in between points).
    - Positive intervals: break point k =3. (we’re just plugging the number into the probability function and comparing it to 2^N).
    - Convex sets: break point never exists. If you shatter you can always achieve 2^N regardless of N.
    - **So if you have no break point, you growth function is 2^N.**
    - **At ANY break point 🡪 mH(N) IS POLYNOMIAL IN N.** what.
    - So all you need to know is that there is a break point to prove that learning is possible.
  + The ramifications: say we have a break point at 3. Now imagine N = 100. The number of breaks is ENORMOUS. So learning is super possible.
    - For x1, x2, x3, where the break point is 3, then you only have half the number of possible combinations for replacements

**Lecture 6: Theory of Generalization**

* Bounding mH(N)
  + We need to show that mH(N) is a polynomial. That’s all we need to know.
    - We show that mH(N) <= … <= … <= …. <= a polynomial
    - The key quantity: the breakpoints B(N, k). The maxiumum number of combinations of N points given that no k vectors can be the same
    - So B(N, k) = the maximum number of dichotomies on N points, with break point k
  + Recursive bound on B(N,k)
    - Consider the following table: x1, x2, … , xn
      * We try to fill with number of possible combinations
      * We’ll call a group these vectors S1 with all the rows that have a single extension. Number of rows we’ll call alpha
      * The group of vectors S2 have both extensions. S2 has two versions: S2- and S2+, each of which have Beta rows. So B(N, k) = alpha + 2Beta
    - We can say that alpha + beta <= B(N-1, k)
    - Estimating Beta by itself: focus on S2 = S2+ || S2-
    - If you have all possible combinations for N – 1 points, you have the same as or less than the number of possible combinations of k
    - So Beta <= B(N-1, k-1)
  + So let’s add them. B(N, k) <= B(N-1, k), + B(N-1, k-1). This property holds for B(N,k), so all we have to do is to compute the numerical value, and compute the upper bound for possible combinations of hypotheses mH(N) at some value N for breakpoint k
* The computation of B(N, k)
  + For k = 1, you can’t have both -1 and +1 for a dichotomy regardless of value N. so they ALL have to be either +1 or all have to be -1. We are stuck. Everything has to be one condition or the other.
  + For N = 1, how many patterns can we get for one column? 2. Since everything can be +1 or -1. The constraint for k is vacuous. Since you can’t have, say , 6 possible combinations k for N = 1 points.
  + OH so we’re solving for the Basic Case
    - And then our inductive step is the B(N,k) <= B(N-1, k) + B(N-1, k-1)
    - THAT’S why this definition is recursive.
  + It turns out that we can solve this VERY easily, numerically. We don’t have a formula yet cause it’s recursive, but hey.
  + EX: in the puzzle, you have 3 points and the breakpoint is 2, so by computing a simple recursion we get an awesome thing: simple math
* **Theorem:**  1) B(N,k) <= SUM (N CHOOSE i) [i = 0, i = k-1]
  + I mean come on, this looks very binomial.
  + We’re going to prove that, by induction, since it holds for all points N-1, it’ll hold for all points N
  + Inductive step
    - Assume that 1) holds for N-1, k-1 and N-1, k. Prove that 1) holds for N, k
    - Aka: argue that SUM (N CHOOSE i) [i = 0, i = k-1] == SUM (N-1 CHOOSE i) [i = 0, i = k-1] + SUM (N-1 CHOOSE i) [i = 0, i = k-2]
    - Okay it works out. We have an exact solution of the upper bound
* Review:
  + YES it’s a polynomial!! That’s awesome
    - But why? For a given H, the break point k is fixed
    - For a growth function, mH(N) <= SUM(N CHOOSE i)[i=0, k-1]. Therefore, the maximum power is N^k-1
    - We KNOW that k doesn’t change with N, so IT’S A POLYNOMIAL WITH N!!!!!
    - Three examples: (SUM(N CHOOSE i)[i=0, k-1])
    - 1) H is positive rays: (break point k = 2)
      * We computed the growth function mH(N) = N + 1. Let’s find the bound.
      * Sum from i-0 to i = 1 (N choose 0, N choose 1) … == N + 1!!!!!
    - 2) H is positive intervals: (break point k = 3)
      * We calculated that mH(N) = ½ N^2 + 1/2 N + 1.
      * …which is the SAME thing as the upper bound B(N,k)!!!!
    - 3) H is 2D Perceptrons.
      * The growth function is totally unknown… we just know the breakpoint k = 4
      * The mH(N) = ?
      * But we can bound the growth function <= 1/6N^3 + 5/6N + 1!! WOW that is useful.
  + Now let’s prove that we can replace M with mH(N) in the Hoeffding Inequality. How do we prove this? It’s 6 pages.
    - Pictorial proof, thank God :D
    - D 🡺 the space of Data Sets. We’ll be painting data sets red if they violate Hoeffding’s inequality. It’s a small quantity, since Hoeffding Inequality
    - We’re saying, for the union bound: you get a bad region that is disjoint. It fills up the canvas very quickly. You have an infinite amount of disjoint sets
    - In our new canvas, the VC Bound, you take the overlaps into consideration, and the total area is a small fraction of the whole thing, and you CAN learn.
    - The reason we introduce the growth function is to characterize the overlaps. That is, the points at which learning is impossible can be characterized
    - So this redundancy is captured by the growth function.
    - The argument is GREAT as long as we can move around the presence of Eout
  + What to do about Eout…
    - Instead of picking one sample, we’re going to pick two samples, independently.
    - We know that Eout and Ein track each other.
    - Let’s give the two samples names. Ein(h) and Ein’(h). Does Ein(h) track Ein’(h)? YES.
    - If we’re not looking at the bin at all, they’re tracking each other, but they get loosely disassociated as we increase the sample size. As I understand it, this is preferable to using Eout because Eout can be infinite.
  + Basically there’s stuff happening in this proof that holds true for any hypothesis set that has a breakpoint
    - P(|Ein(g) – Eout(G) \ > epsilon] <= 4 mH(2N) e^-1/8 epsilon^2 N. This is called the Vapnik-Chervonekis Inequality.

**Lecture 7: The VC Dimension**

* What is the VC Dimension?
  + Definition: The VC dimension of a hypothesis set H, denoted by dVC(H), is “the most points you can shatter.” This is “The largest value of N for which mH(N) = 2^N.
    - Example : Say the VC dimension is 15. Then if N<= dVC (H) 🡪 H can shatter N points. Then if N is equal to or even smaller than dVC(H), you can
    - If any value n, k > dVC, then N is a break point
  + In terms of the vc Dimension dVC:
    - mH(N) <= SUM(N CHOOSE i) [ i = 0, dVC]
    - so it seeeems like the VC dimension is k-1
  + Examples of the VC dimension
    - H is positive rays
      * VC dimension here is 1, since the breakpoint k is 2.
    - H is 2D perceptrons
      * The VC dimension is 3, since the breakpoint k is 4
    - H is convex sets
      * The VC dimension is infinite, since we can always achieve 2^N permutations. The VC dimension always gives an upper bound, just like usual.
  + VC dimension and learning:
    - If dVC (H) is finite 🡪 g <- H will generalize. The algorithm can learn.
    - This is independent of the learning algorithm. Right, makes sense.
      * …but doesn’t the learning algorithm determine the vc dimension?
      * OH it deals with the hypothesis sets!!! RIGHT.
    - The is also independent of the input distribution.
      * Right, we don’t have to worry about it, because the VC dimension computes the max, so it will hold with generalization.
    - The VC dimension is also independent of the target function.
      * Makes sense.
    - So the VC theorem: deals with only the **training examples, the hypothesis set, and the final hypothesis.**
      * **The VC inequality only deals with these parameters.**
* The VC dimension of perceptrons:
  + Example:
    - For d = 2, dVC = 3.
    - For d = 3, dVC = 4
    - dVC = d + 1
  + We’ll prove this:
    - One direction:
      * Construct a set N = d + 1 points in R^d. Let’s construct them such that we can shatter them.
      * X = [x1T, x2T, x3T… xd+1T]. This is a d-dimensional vector, and we have d+1 points.
      * He chose the simplest possible format of data points, such that X is invertible, since the determinant is one. This is an invertible normalized matrix
      * Can we shatter the data set?
        + Yes. Because for any data point, we can shatter the set with vector w: sign(Xw) = y. How do we do this? Easy! Just make Xw = y, so this works. This means that w = X^-1 y. of course. We can shatter any set of independent points.
      * So we can shatter d+1 points. Thus we are GUARANTEED that the VC dimension is at LEAST this big. Since we can shatter this big, we know that we can shatter at least this big, if not bigger.
      * But now we need to show that we cannot shatter ANY set of d + 2 points.
    - Other direction:
      * Let’s take d +2 points
      * We have more points than dimensions
        + Each of these guys comes from a d + 1 dimension
        + We know that the vectors must be linearly dependent
      * Therefore, one xj must be SUM(ai \* xi)[i != j] where not all the ai’s are 0’s. Definition of linear dependence.
      * So what?
        + Consider the following dichotomy: xi’s with nonzero ai get yi = sign(ai) and xj gets yj = -1. This is a dichotomy.
        + We must prove that no perceptron can implement such a dichotomy!
        + Let’s multiply the xj by any w. so it equals SUM(ai \* wT xi)[i != j]
        + So the sign of wT xi = sign ai, which means that ai wTxi > 0
        + This means that the sum must be greater than 0.
        + So the value of wTxj must be greater than 0, so the value of the perceptron must be +1. So it cannot be -1.
      * So therefore you CANNOT shatter ANY d+2 points beyond the VC dimension
    - So if we put it together, we proved that since dVC <= d+1 and dVC >= d +1,
    - So dVC = d + 1
  + Perceptron example:
    - What is d + 1 in the perceptron?
    - It is the number of parameters w0, w1, … , wd
* Interpreting the VC dimension
  + 1. Degrees of freedom
    - Parameters create the degrees of freedom. Think of a series of knobs, where you can turn them in different directions. Any hypothesis h is really just a combination of parameters w.
    - The # of parameters: analog degrees of freedom. If you pick a different threshold, you will get a different perceptron.
    - The VC dimension translated these into binary degrees of freedom.
      * We’re only concerned with how expressive the model is itself. How many we can shatter. What are the degrees of freedom of the model.
    - Example: The usual suspects
      * Positive rays: dVC = 1. One degree of freedom.
      * Positive intervals: dVC = 2. There are two degrees of freedom. Look at that.
    - Example: Not just parameters. (parameters may not contribute to degrees of freedom)
      * What if we feed outputs into perceptrons again and again? The first perceptron has 2 weights. Then the other perceptron has 2 more weights. We do this 4 times, and we have 8 weights. Degrees of freedom? Still just 2. Since these weights are very redundant in 2 dimensions.
      * So we can say there’s a bunch of parameters, but the VC dimension just asks: “how many points can I shatter?”
    - So the VC dimension dVC just gives the **effective** number of parameters. So you don’t look at the number of parameters, you look at the effective number of parameters.
  + 2) Number of data points needed
    - There are two small quantities in the VC inequalities:
      * P[|Ein(g) – Eout(g) | > <- ] <= the VC quantity
      * So we have 2 quantites: the probability, and the approximation error. So we want a particular epsilon and delta. We want to be a certain way away from Eout, and we want this to be correct a certain number of times.
    - So how does N depend on dVC?
      * Let’s look at N^d e^-N. This quantity is a caricature of the quantity, the VC quantity.
      * Let’s plot it for d = 4. We would like the probability to be very small.
      * Let’s plot it for d = 5. Oh MAN I see what he’s doing. The probability that something bad happens is exponentially worse. Oh god I see what’s going on.
      * How does N change with d? let’s draw the probability y on a log scale, since it really doesn’t matter if the probability is greater than 1.
      * So for d = 5, the probability only goes below 1 when we get above 20. Geez.
      * d = 6, d = 15, blah… as we increase the number ,the probabilities under 1 gets larger and larger. The bound (the number of examples to to
    - A practical observation: the actual quantity that we want to bound follows the same monotonicity as the bound. If you use a bigger VC dimension, the quantity that you get will be bigger. This sounds unproven. If you have a bigger VC dimension, you will need PLENTY more examples.
    - So in order to achieve a set of performance, as dimensionality increases
      * Observation: for a huge range of reasonable epsilon and delta, the rule of thumb is that you need **10 times** the vc dimension. That will get you in the region in which the probability is meaningful. WOW. That is HUGE.
* Rearranging things:
  + Start with the VC inequality: P[|Ein(g) – Eout(g) | > <- ] <= 4 mH(2N) e^-1/8 epsilon^2 N.
  + We can do this in two ways: solve for epsilon, or solve for delta. This is sooo law of large numbers.
  + Let’s talk about the good events: with probability 1- delta, |Eout – Ein\ <= omega(N, H, delta). So it’s just a restatement of that.
  + That’s called the **generalization bound.**
  + Some things:
    - Eout – Ein <= omega(N, H, delta). We got rid of the absolute value. Let’s just assume that Ein is always gonna be less than Eout. Let’s assume that Ein has an optimist bias. This doesn’t say that Eout is NEVER smaller than Ein. But let’s just make this a general rule for now.
    - So with probability > = 1 – delta, Eout – Ein <= omega
    - So with probability >- 1 – delta, Eout <= Ein + omega
      * This is very friendly because, now that we got rid of the absolute values, we can do expectation.
    - This will create our best technique for machine learning: regularization. Use Ein as a proxy for Eout. So it’s not Ein that effects Eout—it’s also omega that affects our Eout. Oh that’s a nice way of looking at it.

**Lecture 8: Bias-Variance Tradeoff**

* Approximation-generalization tradeoff
  + We would like to get a small Eout, and a good approximation of f (that holds out of sample)
  + More complex *H* 🡪 better chance of approximating f
  + Morecomplex *H* 🡪 better chance of generalizing f
  + Less complex H 🡪 better chance of generalizing out of sample
* So the bias-variance tradeoff is another way of quantifying the tradeoff of Ein, Eout and epsilon/delta
  + We decompose Eout into
    - 1. How well H can approximate f
    - 2. How well we can zoom in on a specific g in H
  + Applies to real-valued targets and uses squared error
* Let’s start
  + Start with Eout
    - Eout(g^(D)) = EXPECT[(g^(D)( x) – f(x))^2], where D is the data set
    - EXPECT[D] [Eout(g^D)]= ED[Ex [(g^(D)(x) – f(x))^2]
    - … = Ex[Ed [(g^(D)(x) – f(x))^2]]
    - Let’s focus on Ed [(g^(D)(x) – f(x))^2]
      * To evaluate, let’s define the “average” hypothesis, g-(x) = ED[G^(D)(x)] (average hypothesis for a given x in a given data set)
      * Imagine many data sets D1, D2, … , Dn
    - Using g-(x)
      * Ed [(g^(D)(x) – f(x))^2]
  + So the expected value with respect to the data set d of any given point x is the sum of the bias and the variance. OH DUH OH WAIT
    - This is = Ex[bias(x) + var(x)]. BAM.
  + The tradeoff of generalization versus approximation
* Oh I get this. Which one is better to maximize? Bias, or variance?
  + The equivalent question (since we don’t have the target function): which of the two are best to approximate, GIVEN that there is a bias/variance tradeoff and we only have a set.
  + The expected out of sample error is not the sum of both.
* So when you are in a learning situation, you are matching the model complexity to the data resources you have, NOT the target complexity.
  + OH I GET THIS.
  + If you get 100 examples, you choose the function to match the 100 examples best.
* Learning curves
  + Plotting the expected value of Eout and Ein
  + Given data set of size N
    - We know the expected out-of-sample error EXPd[Eout(g^D)]
    - We know the expected in-sample error EXPd[Eout(g^D)]
    - How do these vary with N?