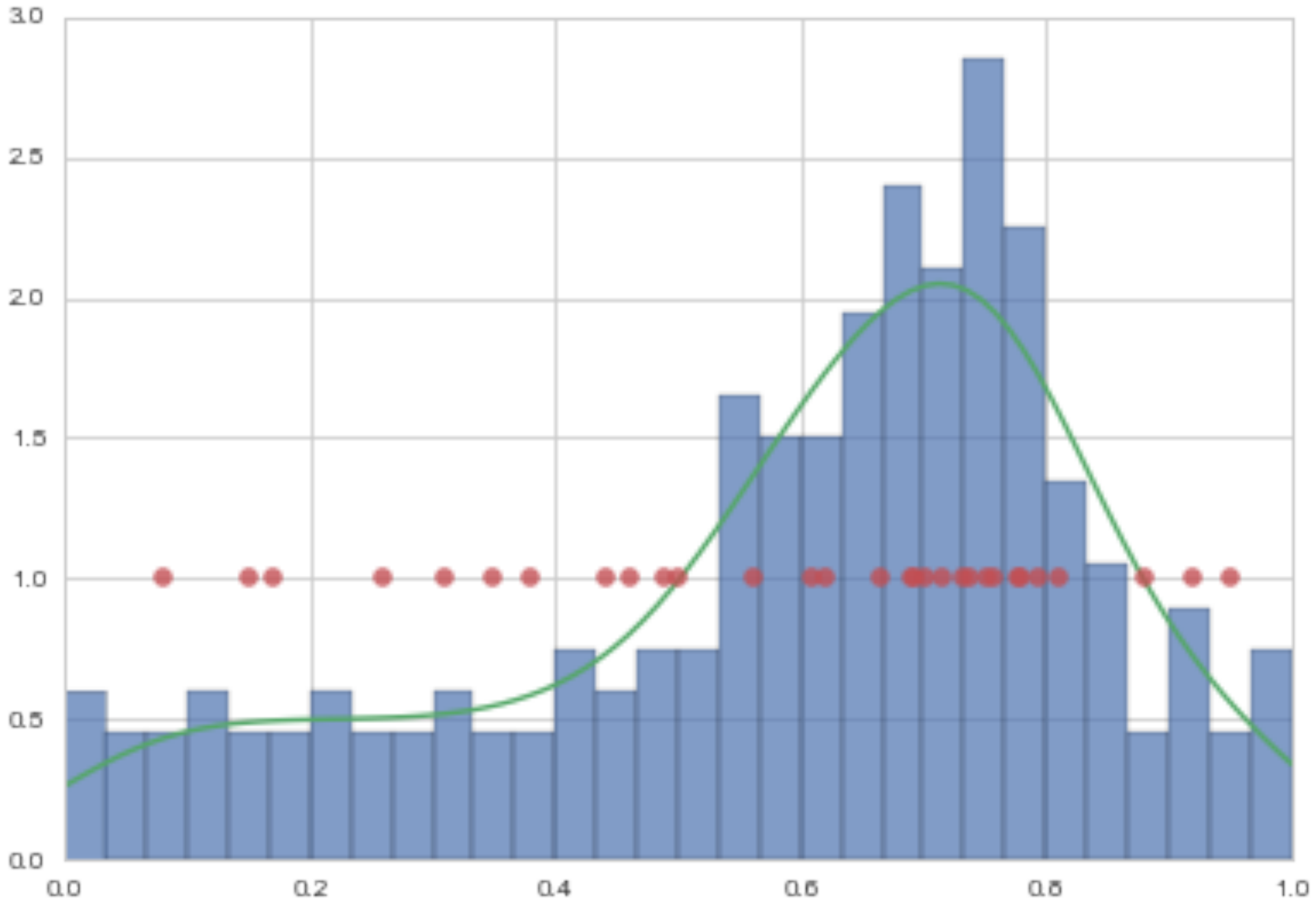
Lecture 7

Machine Learning

BackPropagation for Logistic Regression

Last Times:

- Machine learning, especially supervised learning
- Bias, variance, and overfitting
- Minimized an objective function, called error or cost or risk
- Gradient Descent, SGD on Empirical Risk
- We introduced the test set



Statement of the Learning Problem

The sample must be representative of the population!

$$A : R_{\mathcal{D}}(g) \;\; smallest\, on\, \mathcal{H}$$
$$B : R_{out}(g) \approx R_{\mathcal{D}}(g)$$

A: Empirical risk estimates in-sample risk.
B: Thus the out of sample risk is also small.

LLN: Expectations -> sample averages

$$E_p[R] = \int R(x)p(x)dx = \lim_{n \to \infty} \frac{1}{N} \sum_{x_i \sim p} R(x_i)$$

Empirical Risk Minimization:

$$R_{\mathcal{D}} = E_p[R] \sim \frac{1}{N} \sum_{x_i \sim p} R(x_i)$$

on training set(sample) $\mathcal{D}$.

What we'd really like: population

i.e. out of sample RISK

$$R_{out}(h, y) = E_{p(x)}[R(h(x), y)] = \int dx p(x)(h(x) - y)^2 (e.g.).$$

$$\langle R_{out} \rangle = E_{p(x,y)}[R(h(x), y)] = \int dydx\, p(x, y)R(h(x), y)$$

$$= \int dydxp(y \mid x)p(x)R(h(x), y) = \int dxp(x)E_{p(y|x)}[R(h(x), y)]$$

- This is an average over our sampling distribution, if we had it
- What do we do?

Fit hypothesis $h = g_{\mathcal{D}}$, where $\mathcal{D}$ is our training sample.

Then we'd like

$$\langle R_{out} \rangle = E_{\mathcal{D}}[R_{out}(g_{\mathcal{D}}, y)].$$

But:

Gradient Descent.

For a particular sample, we want:

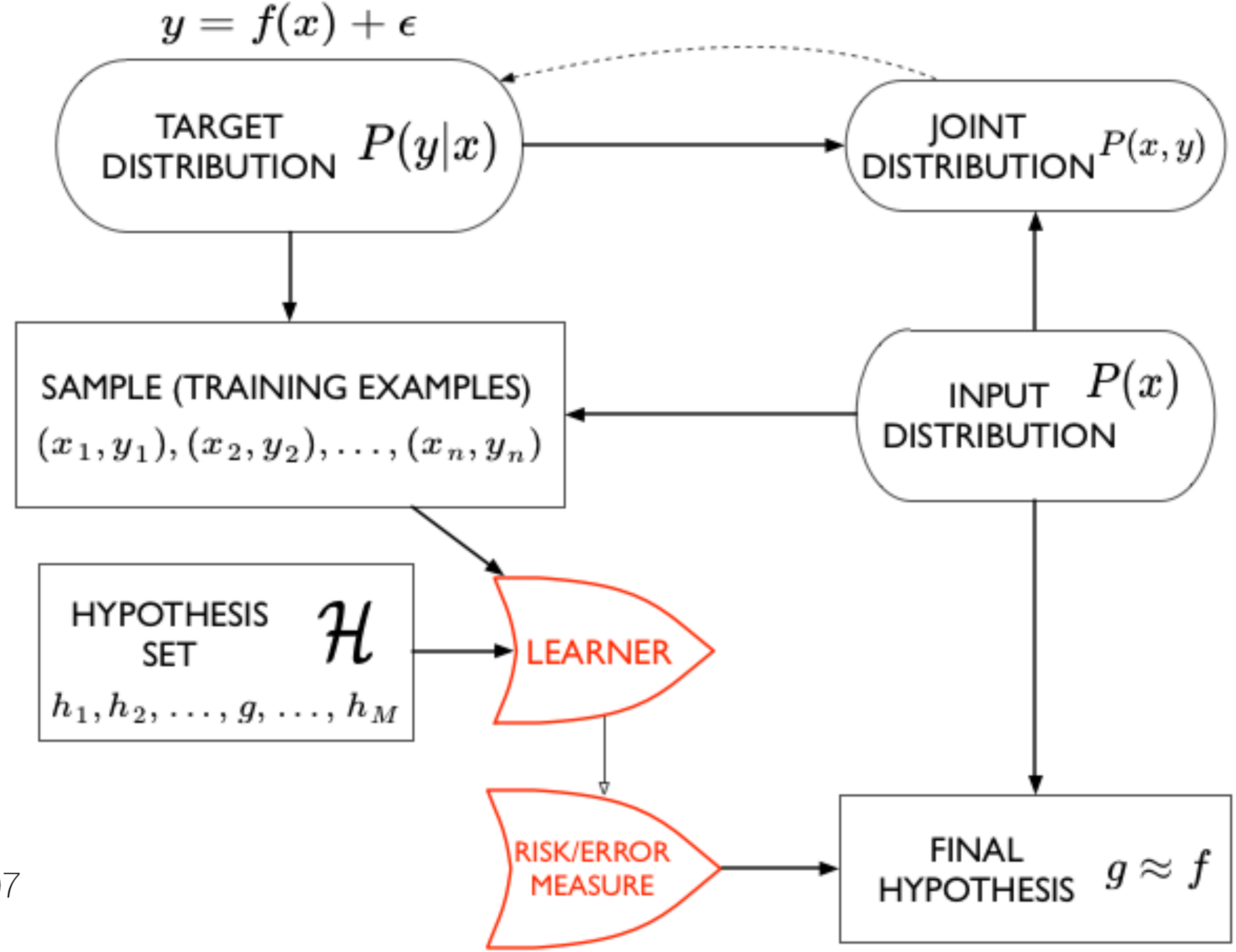
$$\nabla_h R_{out}(h, y) = \int dx p(x) \nabla_h R_{out}(h(x), y) (e.g.).$$

LLN: $= \nabla_h \frac{1}{N} \sum_{i \in pop} R_{out}(h(x_i), y_i) \sim \nabla_h \frac{1}{N} \sum_{i \in \mathcal{D}} R_{in}(h(x_i), y_i)$

SGD takes gradient inside sum

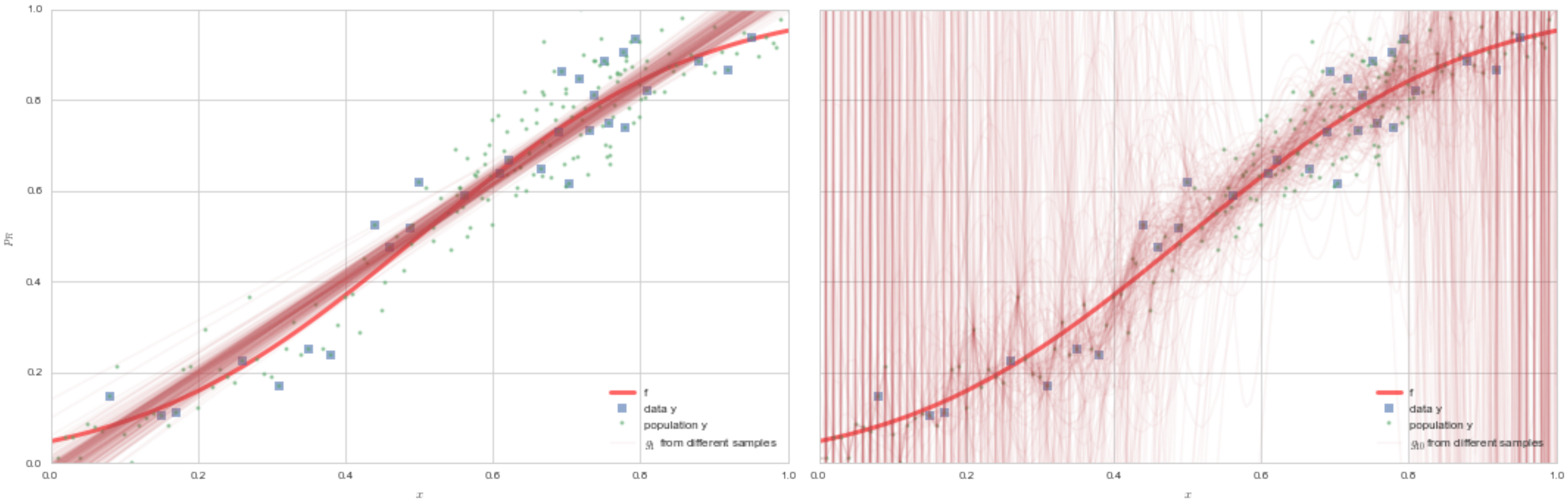
Empirical Risk Minimization

- But we only have the in-sample risk
- Furthermore its an empirical risk
- And its not even a full on empirical distribution, as N is usually quite finite

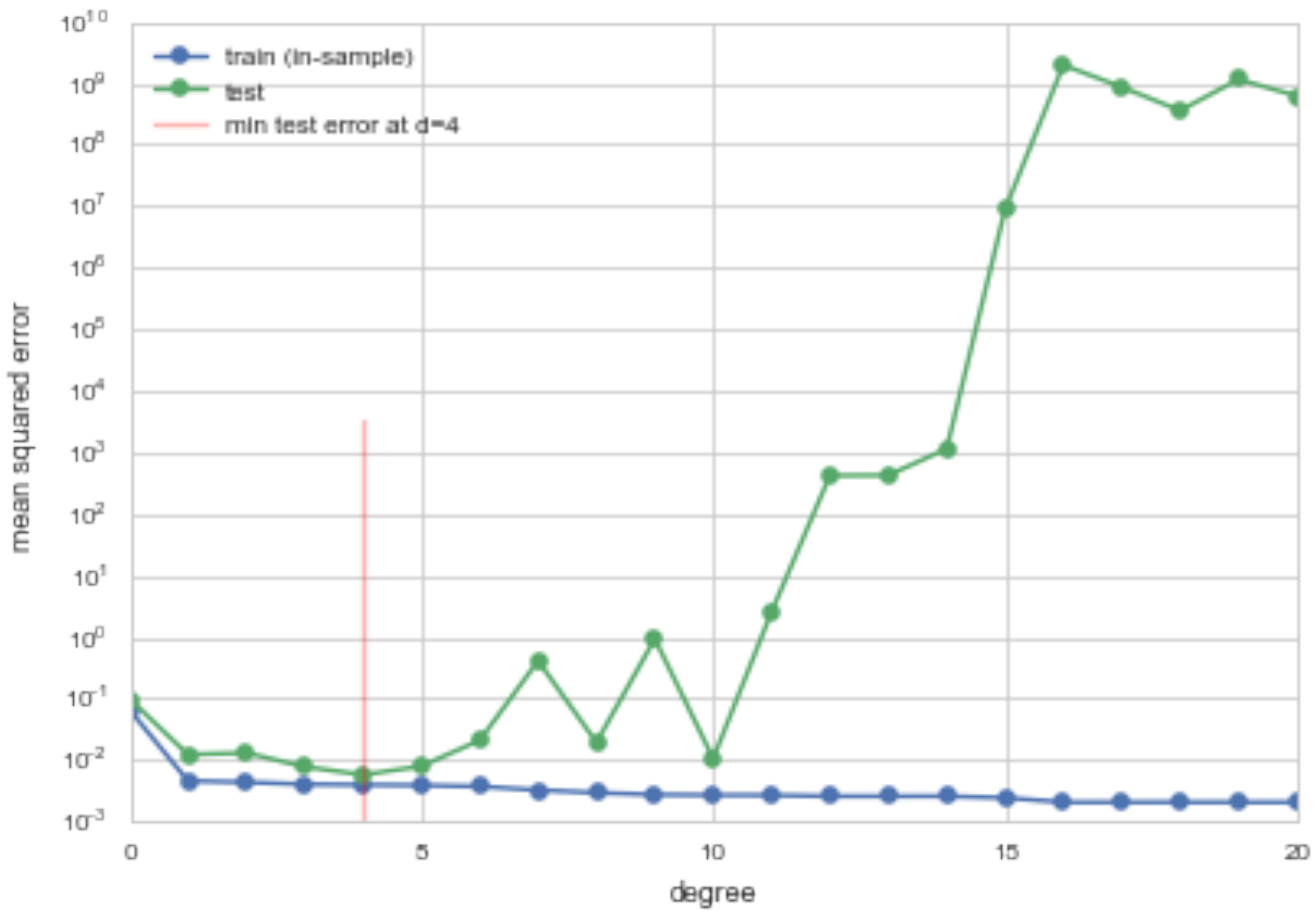

$y = f(x) + \epsilon$
TARGET DISTRIBUTION $P(y|x)$
JOINT DISTRIBUTION $P(x, y)$
SAMPLE (TRAINING EXAMPLES)
$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
INPUT DISTRIBUTION $P(x)$
HYPOTHESIS SET $\mathcal{H}$
$h_1, h_2, \ldots, g, \ldots, h_M$
LEARNER
RISK/ERROR MEASURE
FINAL HYPOTHESIS $g \approx f$

UNDERFITTING (Bias)

vs OVERFITTING (Variance)



BALANCE THE COMPLEXITY



Is this still a test set?

Trouble:

- no discussion on the error bars on our error estimates
- "visually fitting" a value of $d \implies$ contaminated test set.

The moment we **use it in the learning process, it is not a test set**.

Is in-sample

Approximating out-of-sample?

Hoeffding's inequality

population fraction μ, sample drawn with replacement, fraction ν:

$$P(|\nu - \mu| > \epsilon) \leq 2e^{-2\epsilon^2 N}$$

For hypothesis h, identify 1 with $h(x_i) \neq f(x_i)$ at sample x_i. Then μ, ν are population/sample error rates. Then,

$$P(|R_{in}(h) - R_{out}(h)| > \epsilon) \leq 2e^{-2\epsilon^2 N}$$

- Hoeffding inequality holds ONCE we have picked a hypothesis h, as we need it to label the 1 and 0s.
- But over the training set we one by one pick all the models in the hypothesis space
- best fit g is among the h in $\mathcal{H}$, g must be h_1 OR h_2 OR....Say **effectively** M such choices:

$$P(|R_{in}(g) - R_{out}(g)| \geq \epsilon) <= \sum_{h_i \in \mathcal{H}} P(|R_{in}(h_i) - R_{out}(h_i)| \geq \epsilon) <= 2\,M\,e^{-2\epsilon^2 N}$$

Hoeffding, repharased:

Now let $\delta = 2\,M\,e^{-2\epsilon^2 N}$.

Then, **with probability** $1-\delta$:

$$R_{out} <= R_{in} + \sqrt{\frac{1}{2N}ln(\frac{2M}{\delta})}$$

For finite effective hypothesis set size M, $R_{out} \sim R_{in}$ as N larger..

Training vs Test

- training error approximates out-of-sample error slowly
- is test set just another sample like the training sample?
- key observation: test set is looking at only one hypothesis because the fitting is already done on the training set. So $M = 1$ for this sample!

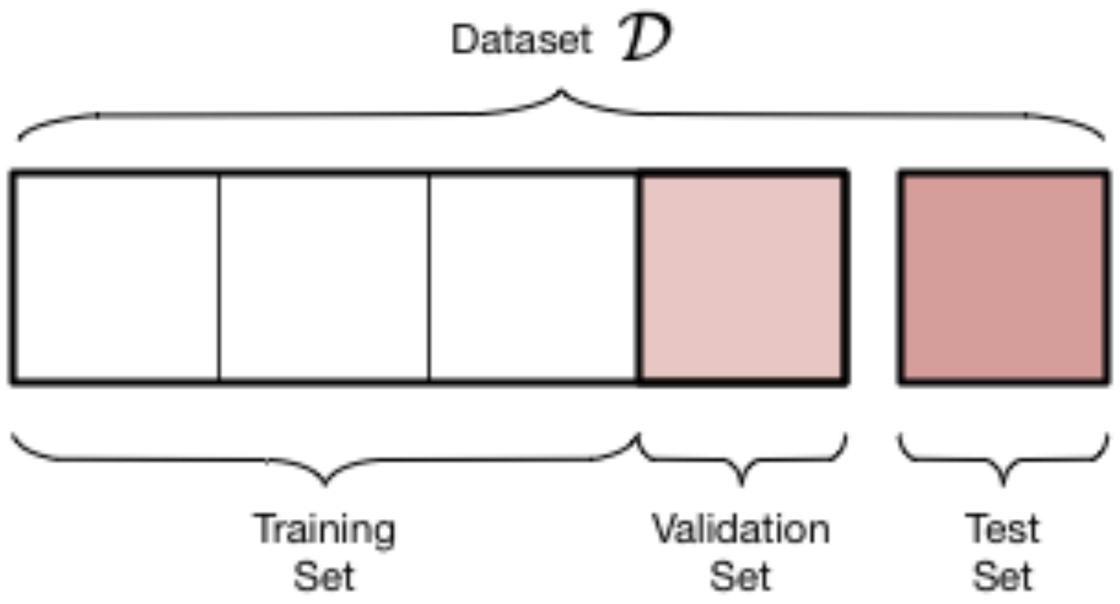
$$R_{out} <= R_{in} + \sqrt{\frac{1}{2N_{test}} ln(\frac{2}{\delta})}$$

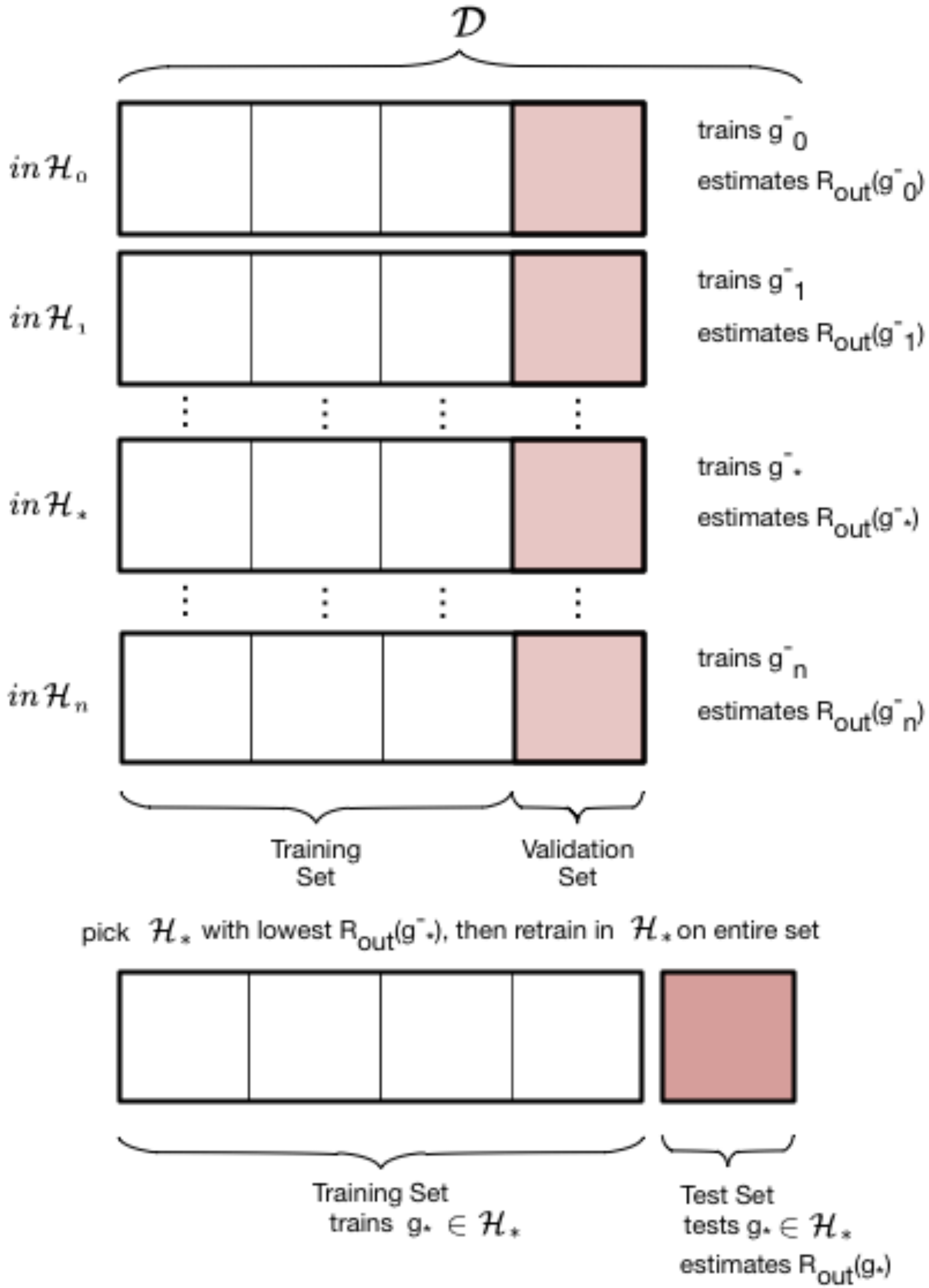
Training vs Test

- the test set does not have an optimistic bias like the training set(thats why the larger effective M factor)
- once you start fitting for things like d on the test set, you cant call it a test set any more since we lose tight guarantee.
- test set has a cost of less data in the training set and must thus fit a less complex model.

VALIDATION

- train-test not enough as we *fit* for d on test set and contaminate it
- thus do train-validate-test





If we dont fit a hyperparameter

- first assume that the validation set is acting like a test set.
- validation risk or error is an unbiased estimate of the out of sample risk.
- Hoeffding bound for a validation set is then identical to that of the test set.

usually we want to fit a hyperparameter

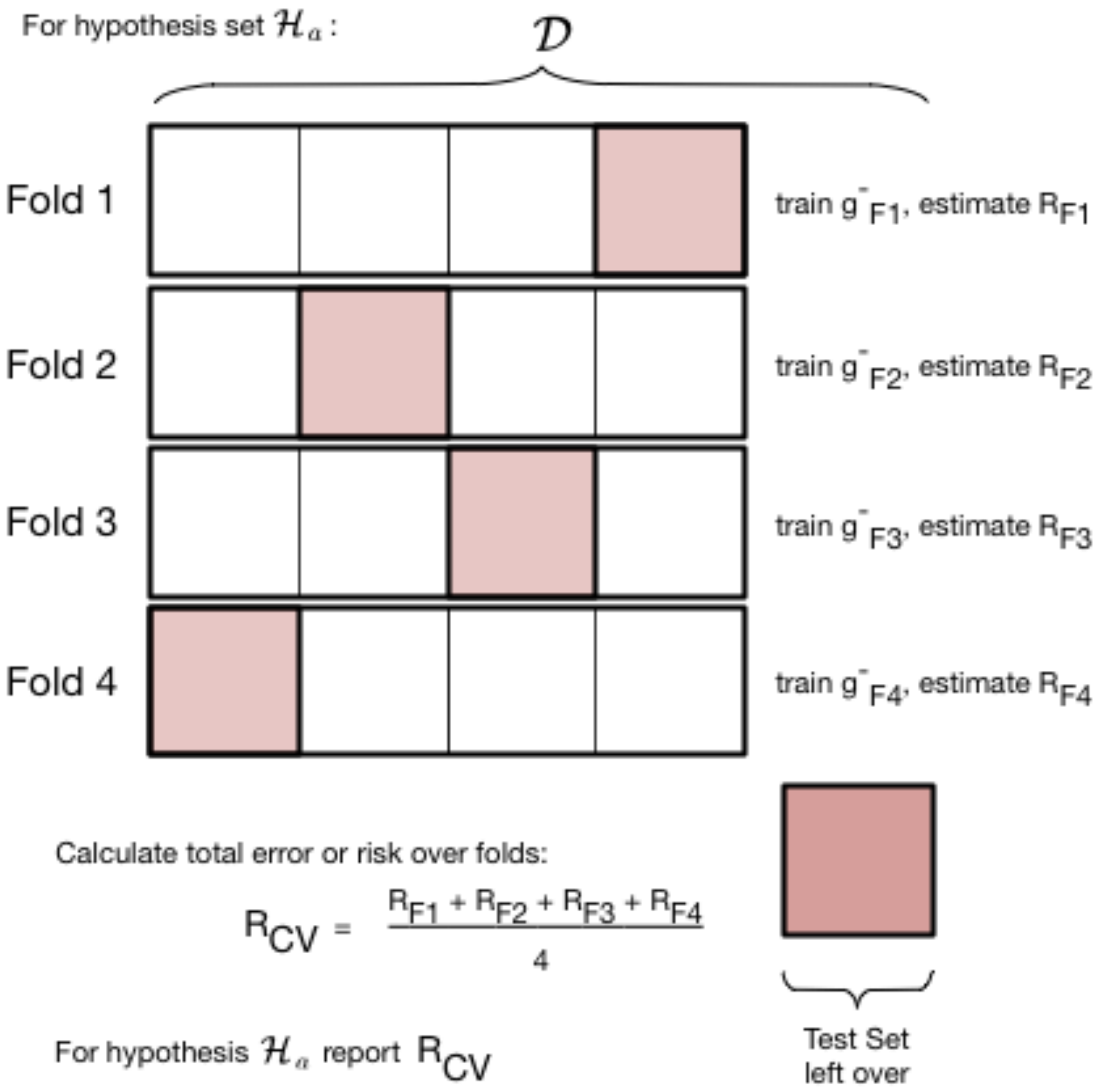
- we **wrongly** already attempted to do on our previous test set.
- choose the d, g^* combination with the lowest validation set risk.
- $R_{val}(g^{-*}, d^*)$ has an optimistic bias since d effectively fit on validation set
- its Hoeffding bound must now take into account the grid-size as the effective size of the hypothesis space.

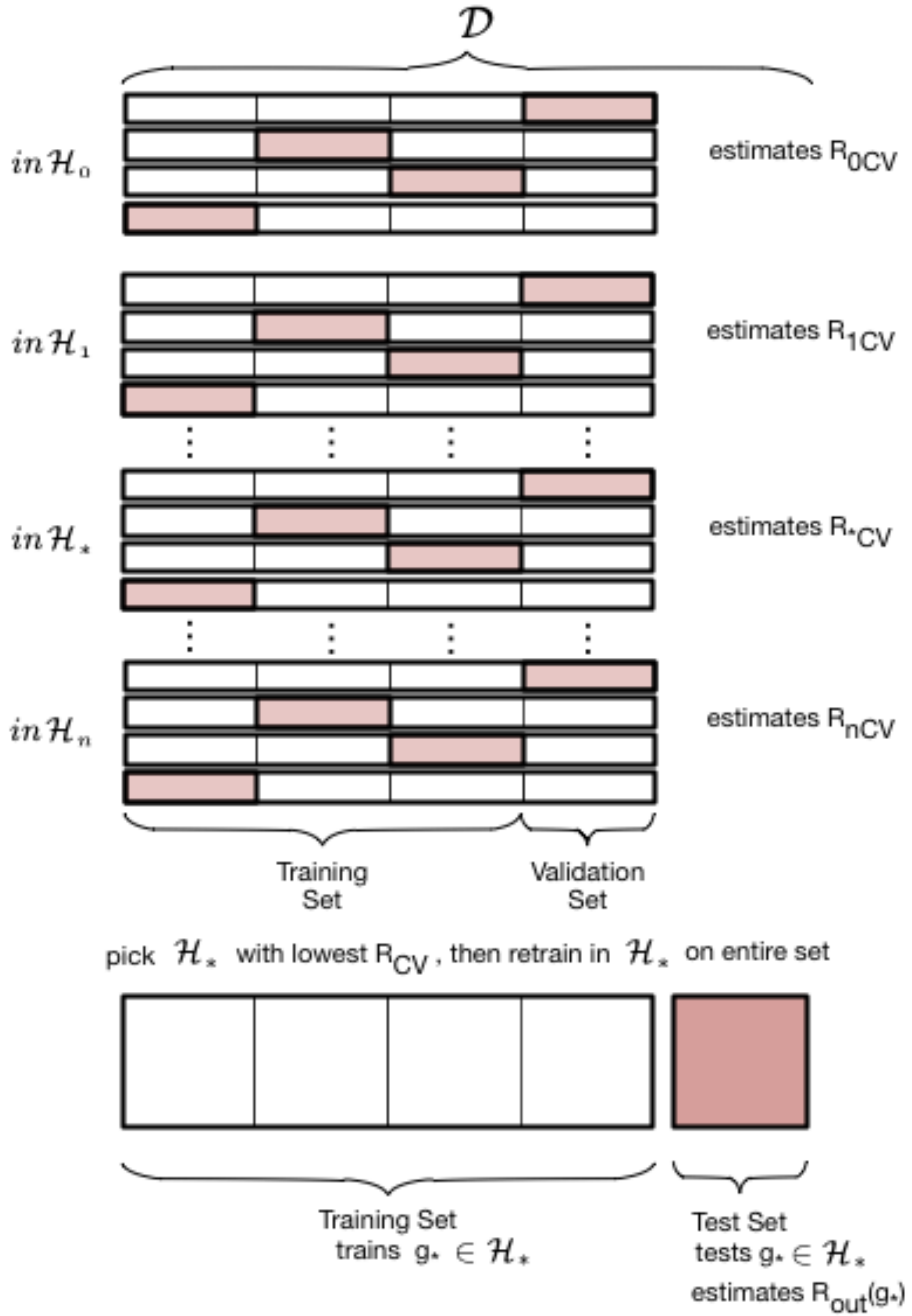
- this size from hyperparameters is typically a smaller size than that from parameters.

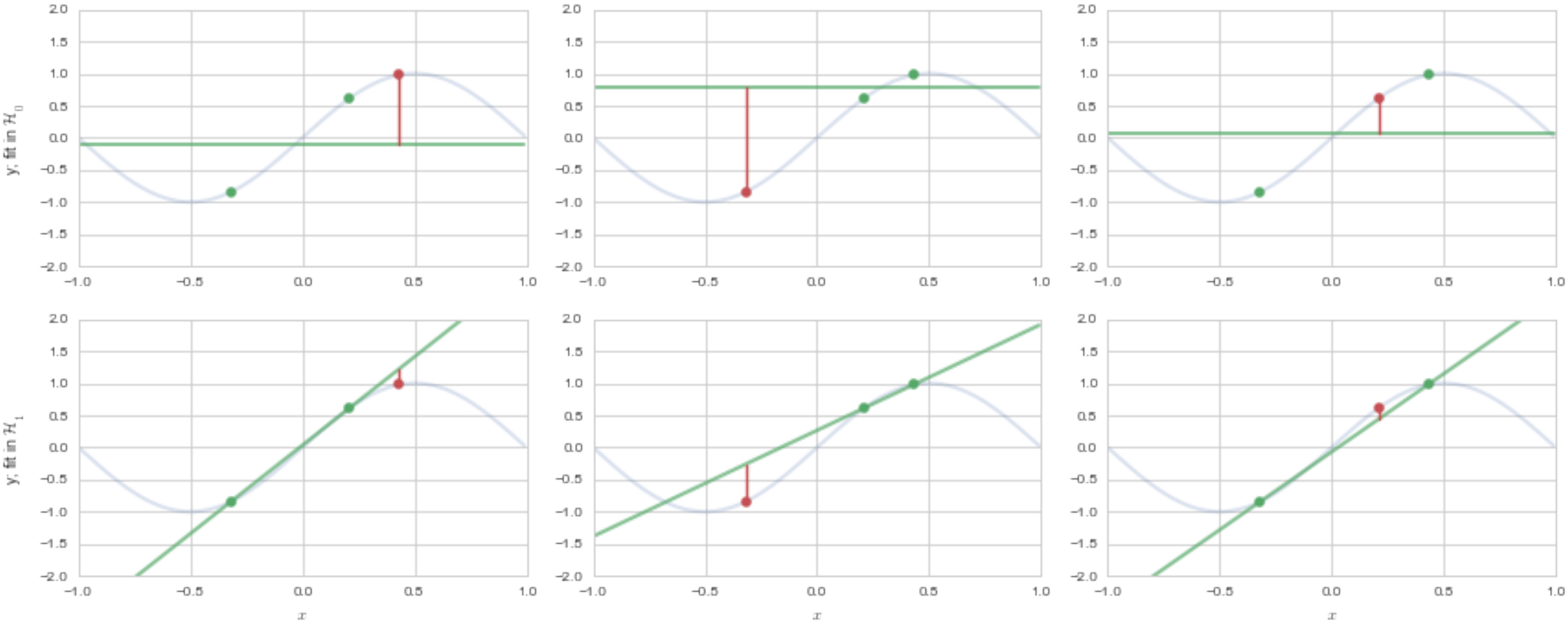
Retrain on entire set!

- finally retrain on the entire train+validation set using the appropriate (g^{-*}, d^*) combination.
- works as training for a given hypothesis space with more data typically reduces the risk even further.

CROSS-VALIDATION





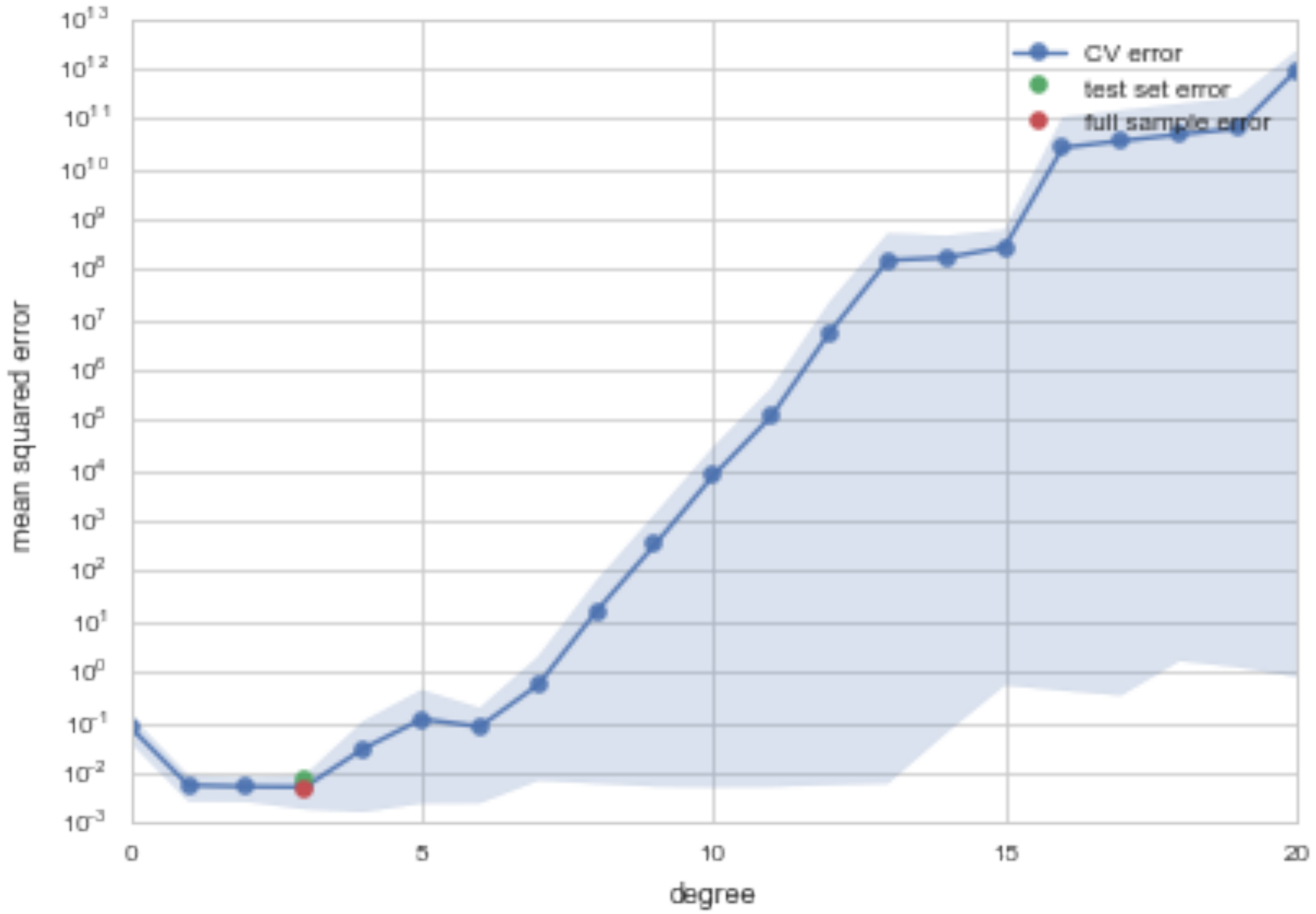

y; fit in $\mathcal{H}_0$
y; fit in $\mathcal{H}_1$
x

CROSS-VALIDATION

is

- a resampling method
- robust to outlier validation set
- allows for larger training sets
- allows for error estimates

Here we find $d = 3$.



Cross Validation considerations

- validation process as one that estimates R_{out} directly, on the validation set.
- It's critical use is in the model selection process.
- once you do that you can estimate R_{out} using the test set as usual, but now you have also got the benefit of a robust average and error bars.
- key subtlety: in the risk averaging process, you are actually

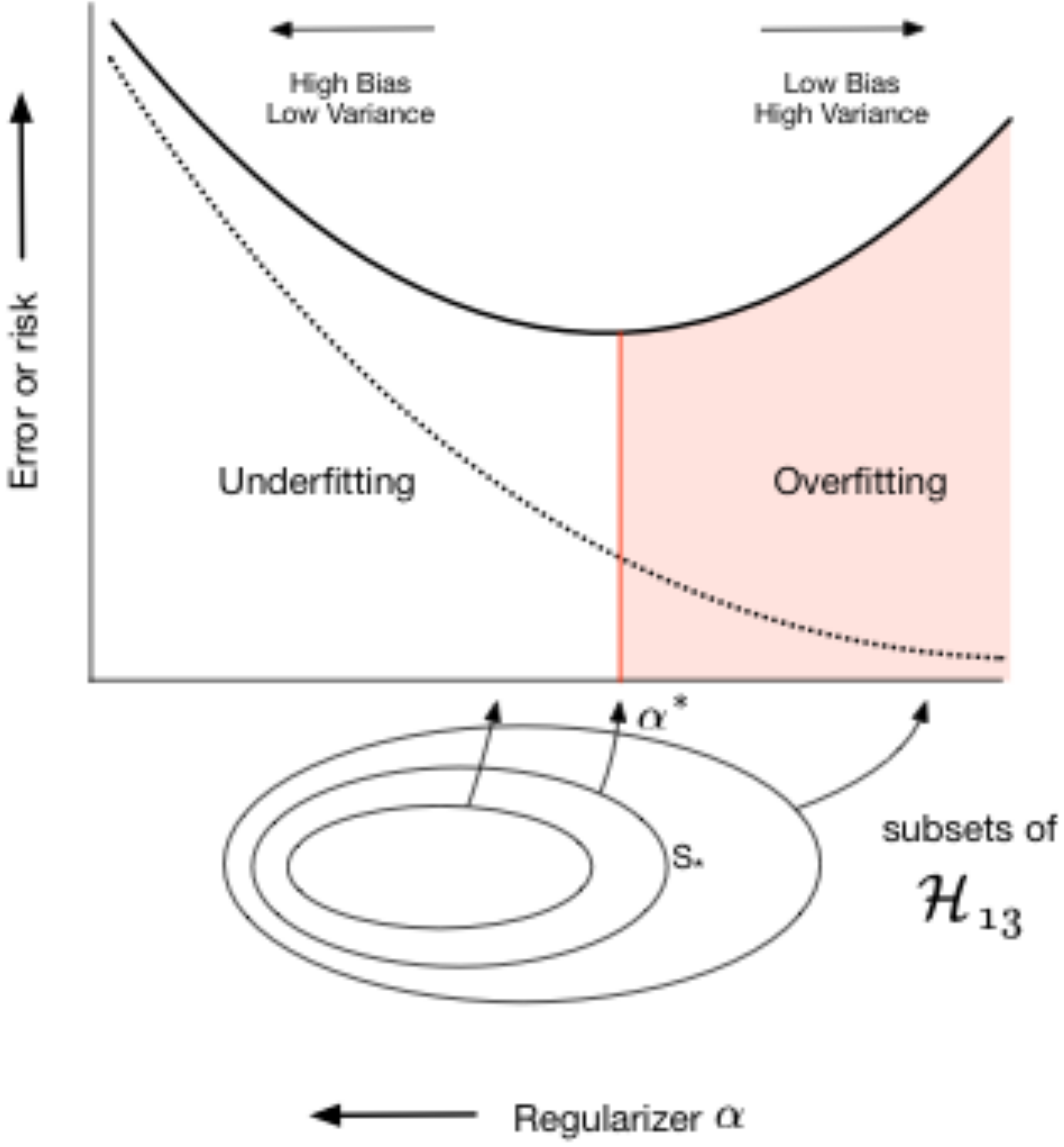
REGULARIZATION

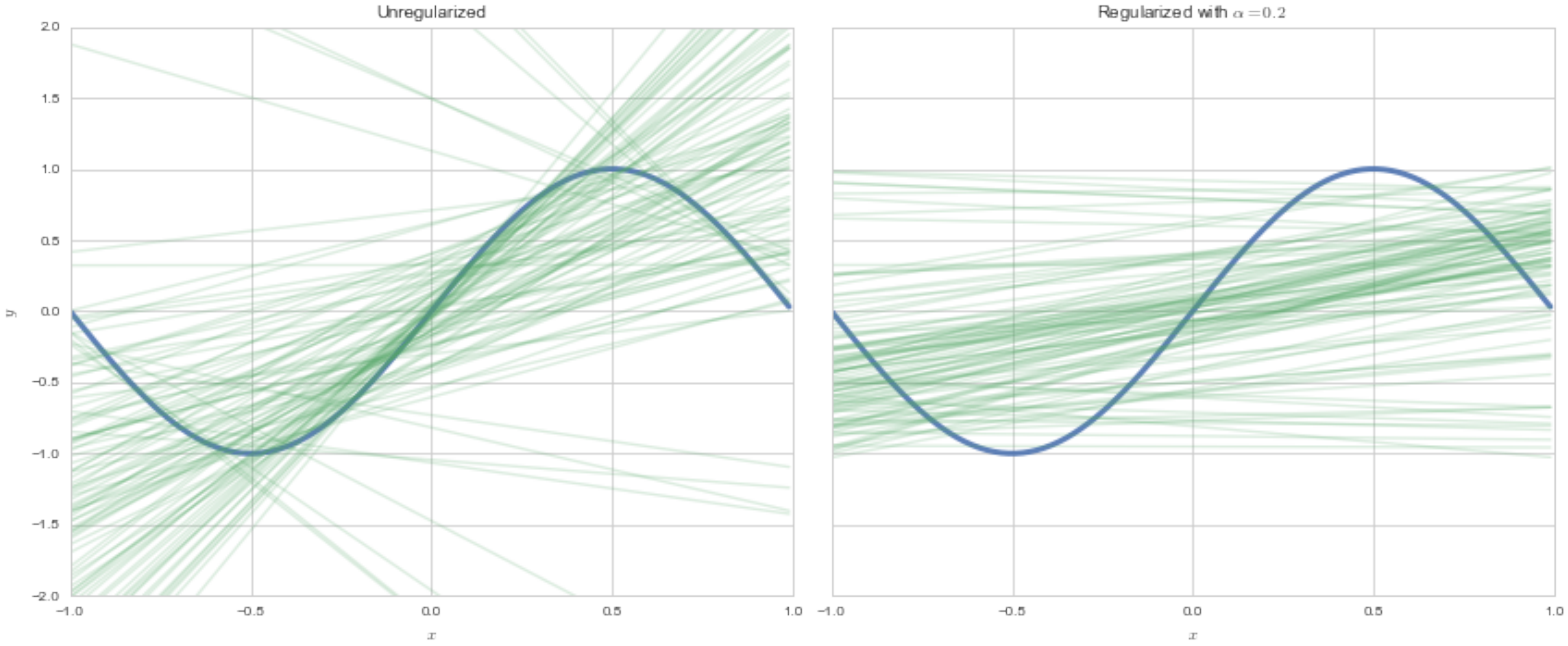
Keep higher a-priori complexity and impose a

complexity penalty

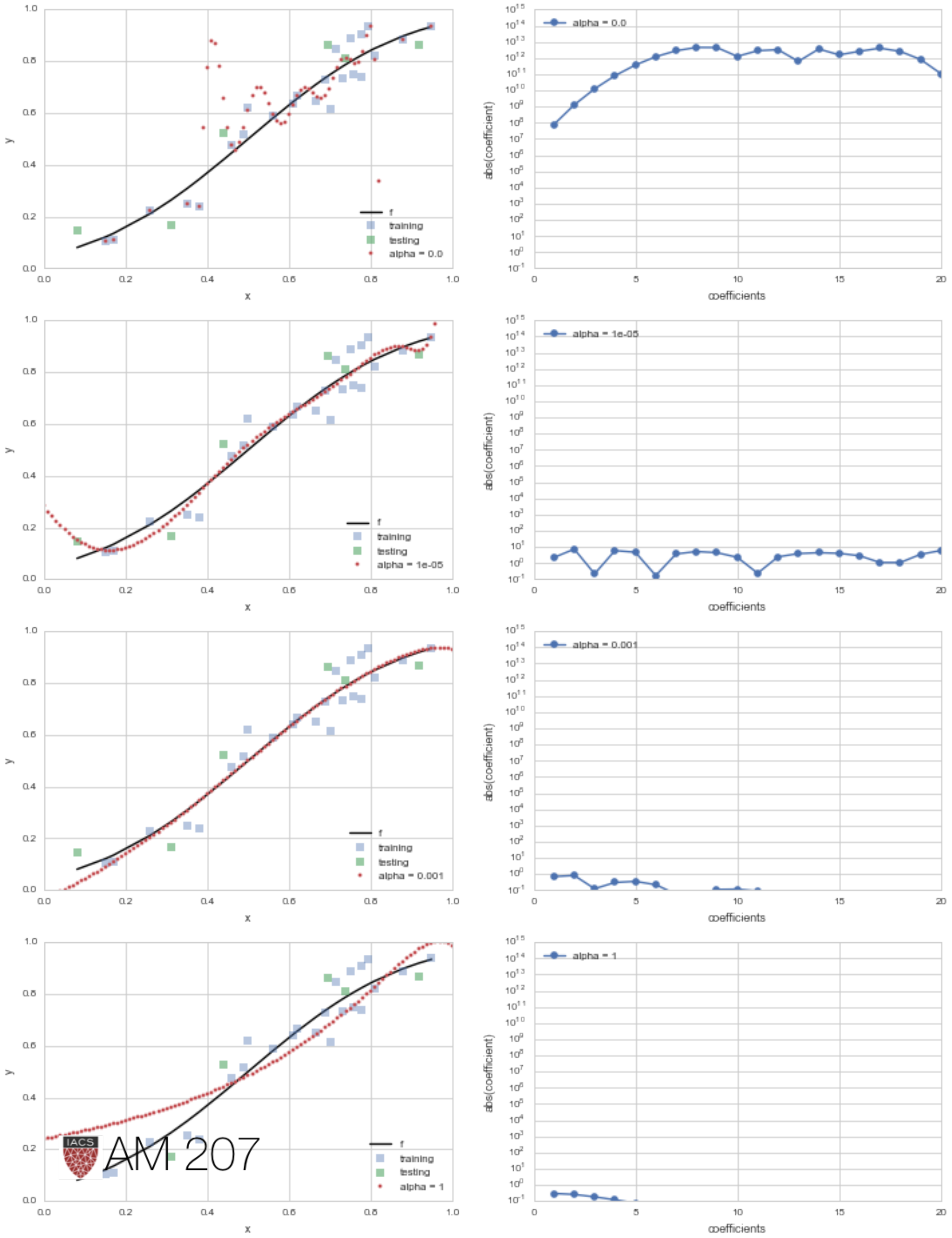
on risk instead, to choose a SUBSET of $\mathcal{H}_{big}$. We'll make the coefficients small:

$$\sum_{i=0}^{j} \theta_i^2 < C.$$





Unregularized
Regularized with $\alpha = 0.2$
y
x
x
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.0
−0.5
0.0
0.5
1.0



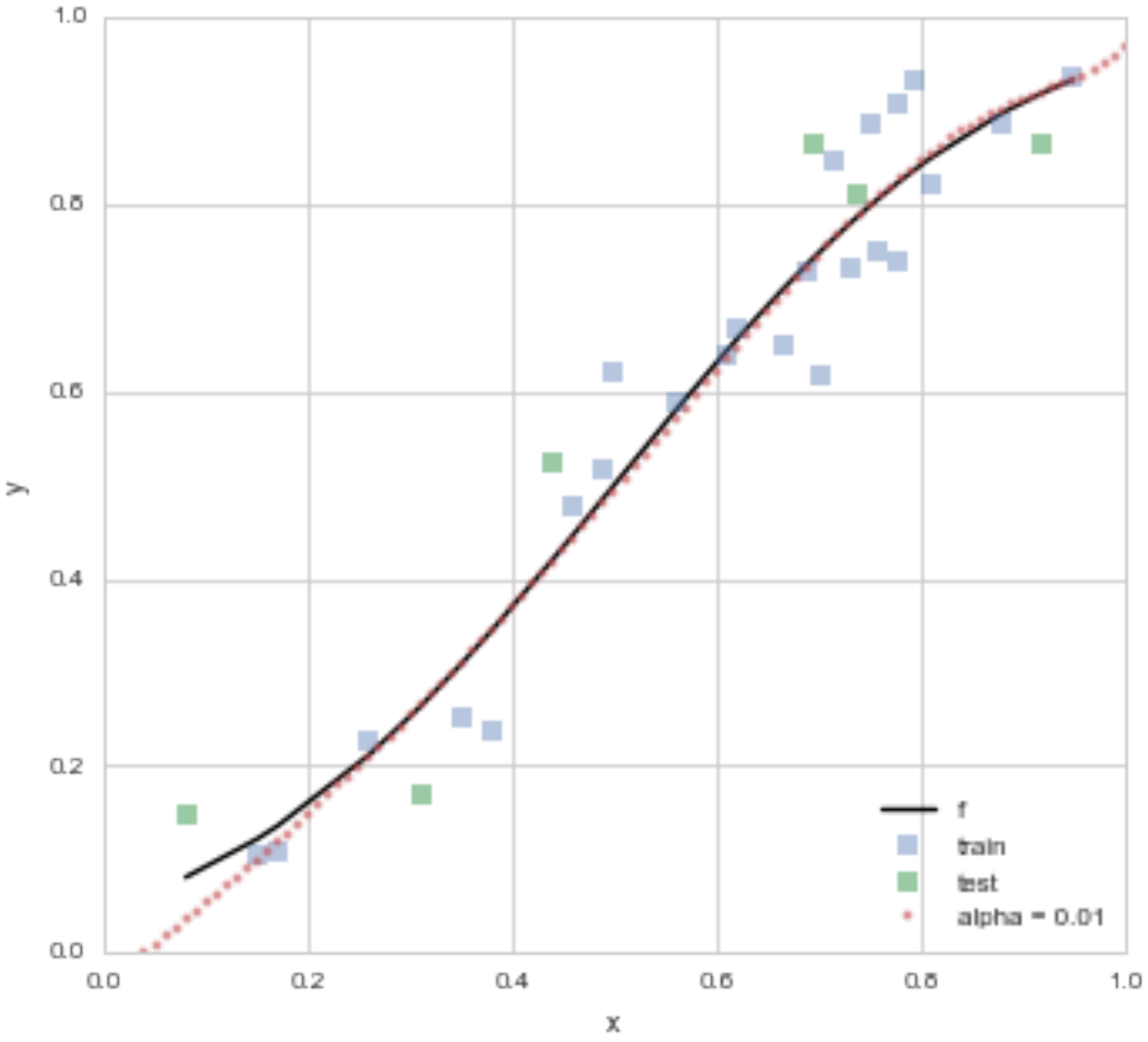
AM 207

REGULARIZATION

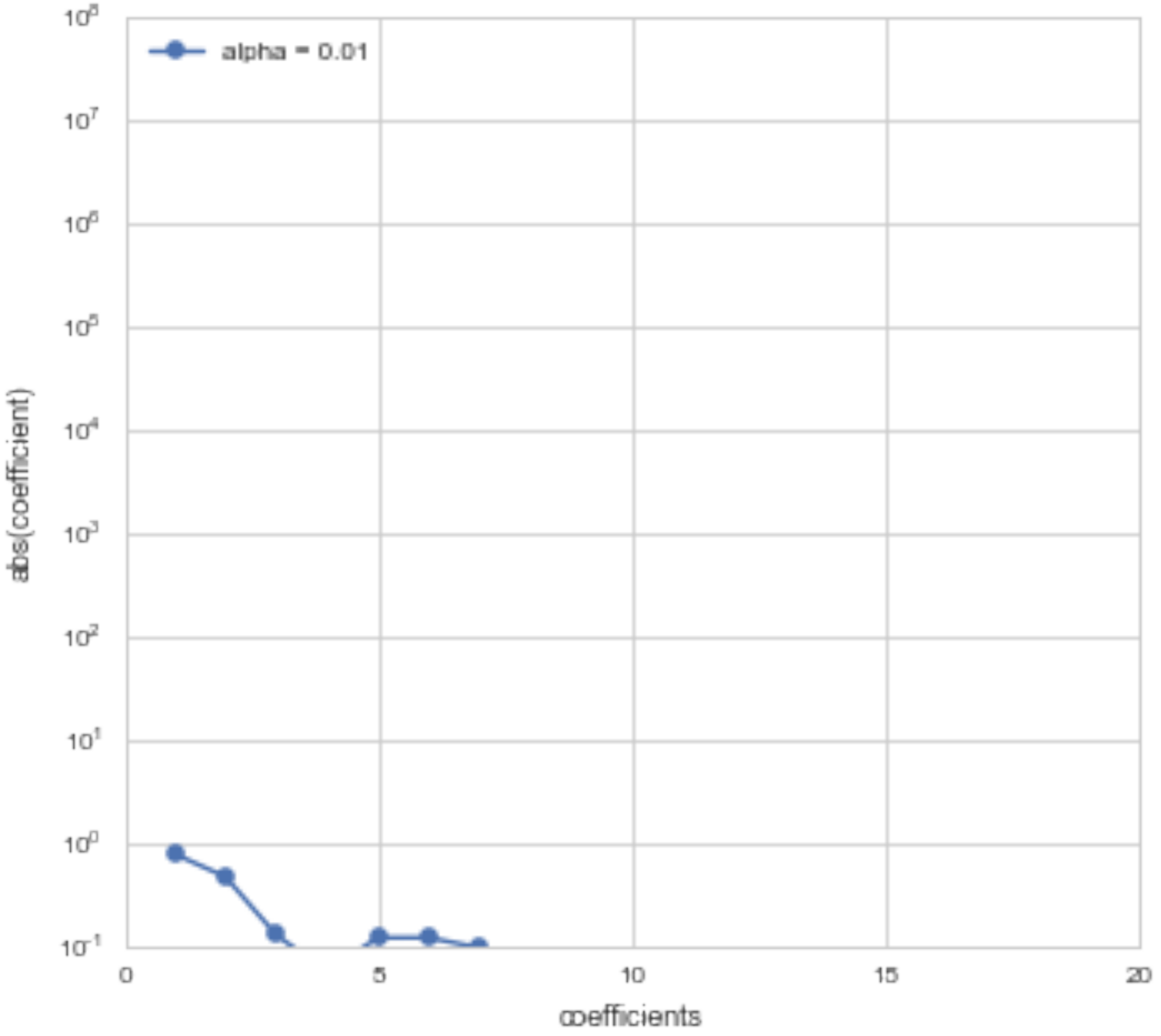
$$\mathcal{R}(h_j) = \sum_{y_i \in \mathcal{D}} (y_i - h_j(x_i))^2 + \alpha \sum_{i=0}^{j} \theta_i^2.$$

As we increase α, coefficients go towards 0.

Lasso uses $\alpha \sum_{i=0}^{j} |\theta_i|$, sets coefficients to exactly 0.



1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
f
train
test
alpha = 0.01



alpha = 0.01
abs(coefficient)
coefficients
0
5
10
15
20

MLE for Logistic Regression

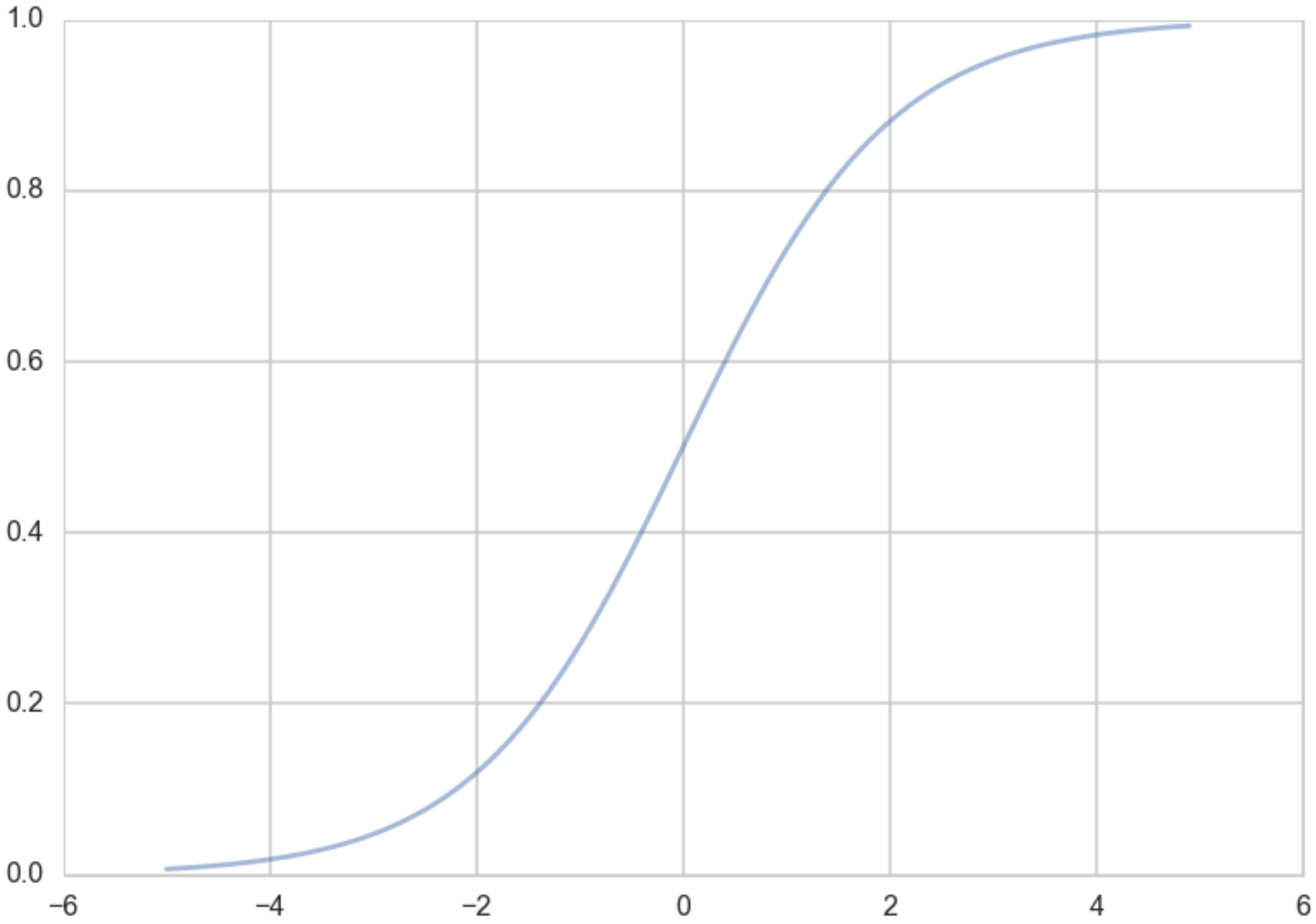
- example of a Generalized Linear Model (GLM)
- "Squeeze" linear regression through a **Sigmoid** function
- this bounds the output to be a probability
- What is the sampling Distribution?

Sigmoid function

This function is plotted below:

```
h = lambda z: 1./(1+np.exp(-z))
zs=np.arange(-5,5,0.1)
plt.plot(zs, h(zs), alpha=0.5);
```

Identify: $z = \mathbf{w} \cdot \mathbf{x}$. and $h(\mathbf{w} \cdot \mathbf{x})$ with the probability that the sample is a '1' ($y = 1$).



Then, the conditional probabilities of $y = 1$ or $y = 0$ given a particular sample's features $\mathbf{x}$ are:

$$P(y = 1|\mathbf{x}) = h(\mathbf{w} \cdot \mathbf{x})$$
$$P(y = 0|\mathbf{x}) = 1 - h(\mathbf{w} \cdot \mathbf{x}).$$

These two can be written together as

$$P(y|\mathbf{x}, \mathbf{w}) = h(\mathbf{w} \cdot \mathbf{x})^y (1 - h(\mathbf{w} \cdot \mathbf{x}))^{(1-y)}$$

BERNOULLI!!

Multiplying over the samples we get:

$$P(y|\mathbf{x},\mathbf{w}) = P(\{y_i\}|\{\mathbf{x}_i\},\mathbf{w}) = \prod_{y_i \in \mathcal{D}} P(y_i|\mathbf{x}_i,\mathbf{w}) = \prod_{y_i \in \mathcal{D}} h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i} \left(1 - h(\mathbf{w}\cdot\mathbf{x}_i)\right)^{(1-y_i)}$$

A noisy y is to imagine that our data $\mathcal{D}$ was generated from a joint probability distribution $P(x,y)$. Thus we need to model y at a given x, written as $P(y \mid x)$, and since $P(x)$ is also a probability distribution, we have:

$$P(x,y) = P(y \mid x)P(x),$$

Indeed its important to realize that a particular sample can be thought of as a draw from some "true" probability distribution.

maximum likelihood estimation maximises the **likelihood of the sample y**,

$$\mathcal{L} = P(y \mid \mathbf{x}, \mathbf{w}).$$

Again, we can equivalently maximize

$$\ell = log(P(y \mid \mathbf{x}, \mathbf{w}))$$

Thus

$$\begin{aligned}
\ell &= log\left(\prod_{y_i \in \mathcal{D}} h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}(1-h(\mathbf{w}\cdot\mathbf{x}_i))^{(1-y_i)}\right) \\
&= \sum_{y_i \in \mathcal{D}} log\left(h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i}(1-h(\mathbf{w}\cdot\mathbf{x}_i))^{(1-y_i)}\right) \\
&= \sum_{y_i \in \mathcal{D}} log\, h(\mathbf{w}\cdot\mathbf{x}_i)^{y_i} + log\,(1-h(\mathbf{w}\cdot\mathbf{x}_i))^{(1-y_i)} \\
&= \sum_{y_i \in \mathcal{D}} (y_i log(h(\mathbf{w}\cdot\mathbf{x})) + (1-y_i) log(1-h(\mathbf{w}\cdot\mathbf{x})))
\end{aligned}$$

NLL

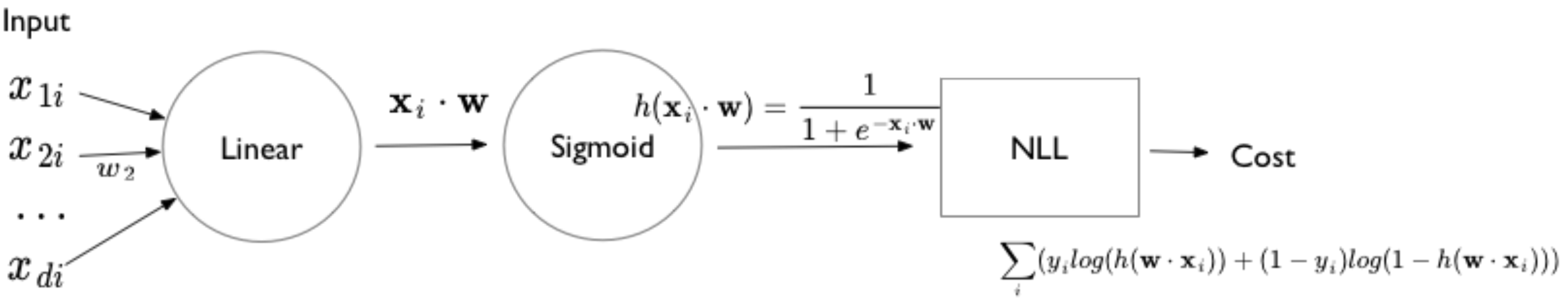
The negative of this log likelihood (NLL), also called *cross-entropy*.

$$NLL = -\sum_{y_i \in \mathcal{D}} (y_i log(h(\mathbf{w} \cdot \mathbf{x})) + (1 - y_i) log(1 - h(\mathbf{w} \cdot \mathbf{x})))$$

Gradient: $\nabla_{\mathbf{w}} NLL = \sum_i \mathbf{x}_i^T (p_i - y_i) = \mathbf{X}^T \cdot (\mathbf{p} - \mathbf{w})$

Hessian: $H = \mathbf{X}^T diag(p_i(1 - p_i))\mathbf{X}$ positive definite $\implies$ convex

Units based diagram



Softmax formulation

- Identify p_i and $1 - p_i$ as two separate probabilities constrained to add to 1. That is $p_{1i} = p_i ; p_{2i} = 1 - p_i$.
- $p_{1i} = \dfrac{e^{\mathbf{w}_1 \cdot \mathbf{x}}}{e^{\mathbf{w}_1 \cdot \mathbf{x}} + e^{\mathbf{w}_2 \cdot \mathbf{x}}}$
- $p_{2i} = \dfrac{e^{\mathbf{w}_2 \cdot \mathbf{x}}}{e^{\mathbf{w}_1 \cdot \mathbf{x}} + e^{\mathbf{w}_2 \cdot \mathbf{x}}}$
- Can translate coefficients by fixed amount ψ without any change

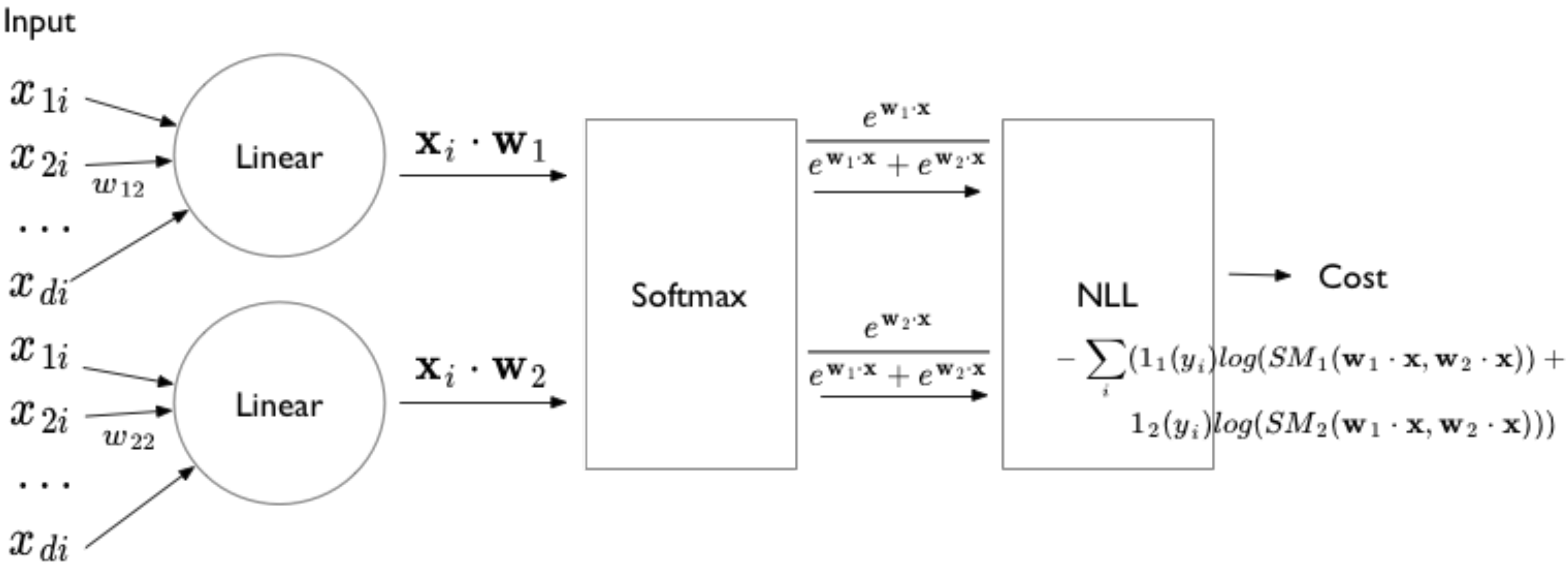
NLL and gradients for Softmax

$$\mathcal{L} = \prod_i p_{1i}^{1_1(y_i)} p_{2i}^{1_2(y_i)}$$

$$NLL = -\sum_i \left(1_1(y_i) log(p_{1i}) + 1_2(y_i) log(p_{2i})\right)$$

$$\frac{\partial NLL}{\partial \mathbf{w}_1} = -\sum_i \mathbf{x}_i (y_i - p_{1i}), \frac{\partial NLL}{\partial \mathbf{w}_2} = -\sum_i \mathbf{x}_i (y_i - p_{2i})$$

Units diagram for Softmax

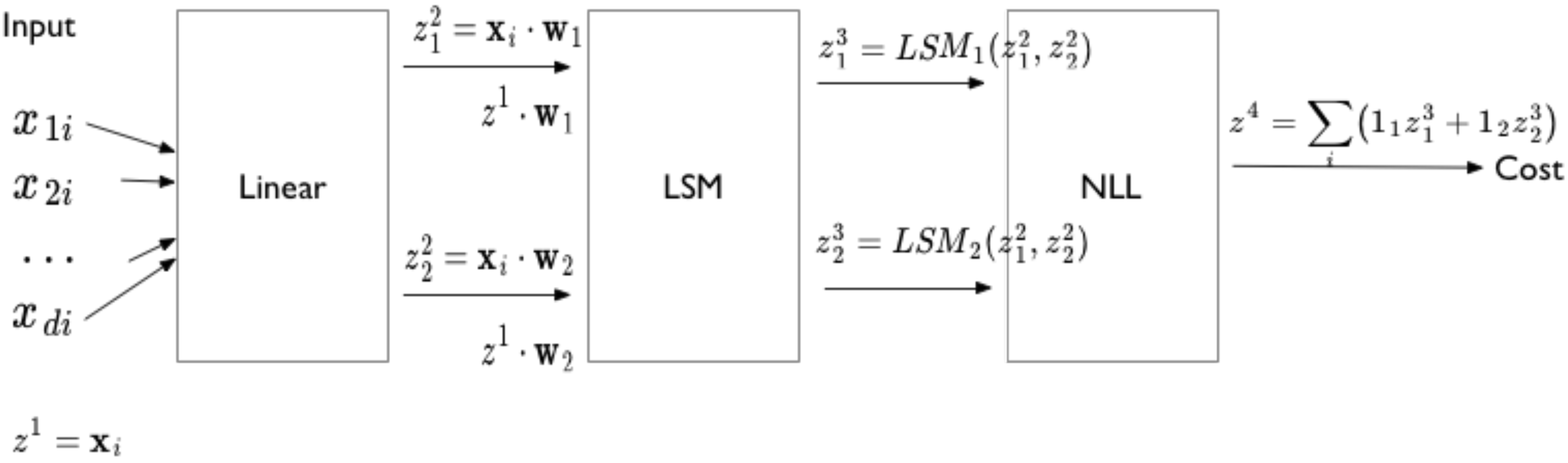


Rewrite NLL

$$NLL = -\sum_i \left(1_1(y_i)LSM_1(\mathbf{w}_1 \cdot \mathbf{x}, \mathbf{w}_2 \cdot \mathbf{x}) + 1_2(y_i)LSM_2(\mathbf{w}_1 \cdot \mathbf{x}, \mathbf{w}_2 \cdot \mathbf{x})\right)$$

where $SM_1 = \frac{e^{\mathbf{w}_1 \cdot \mathbf{x}}}{e^{\mathbf{w}_1 \cdot \mathbf{x}} + e^{\mathbf{w}_2 \cdot \mathbf{x}}}$ puts the first argument in the numerator. Ditto for LSM_1 which is simply $log(SM_1)$.

Units diagram Again



Equations, layer by layer

$$\mathbf{z}^1 = \mathbf{x}_i$$

$$\mathbf{z}^2 = (z_1^2, z_2^2) = \left(\mathbf{w}_1 \cdot \mathbf{x}_i, \mathbf{w}_2 \cdot \mathbf{x}_i\right) = \left(\mathbf{w}_1 \cdot \mathbf{z}_i^1, \mathbf{w}_2 \cdot \mathbf{z}_i^1\right)$$

$$\mathbf{z}^3 = (z_1^3, z_2^3) = \left(LSM_1(z_1^2, z_2^2), LSM_2(z_1^2, z_2^2)\right)$$

$$z^4 = NLL(\mathbf{z}^3) = NLL(z_1^3, z_2^3) = -\sum_i \left(1_1(y_i)z_1^3(i) + 1_2(y_i)z_1^3(i)\right)$$

Reverse Mode Differentiation

$$Cost = f^{Loss}(\mathbf{f}^3(\mathbf{f}^2(\mathbf{f}^1(\mathbf{x}))))$$

$$\nabla_{\mathbf{x}} Cost = \frac{\partial f^{Loss}}{\partial \mathbf{f}^3} \frac{\partial \mathbf{f}^3}{\partial \mathbf{f}^2} \frac{\partial \mathbf{f}^2}{\partial \mathbf{f}^1} \frac{\partial \mathbf{f}^1}{\partial \mathbf{x}}$$

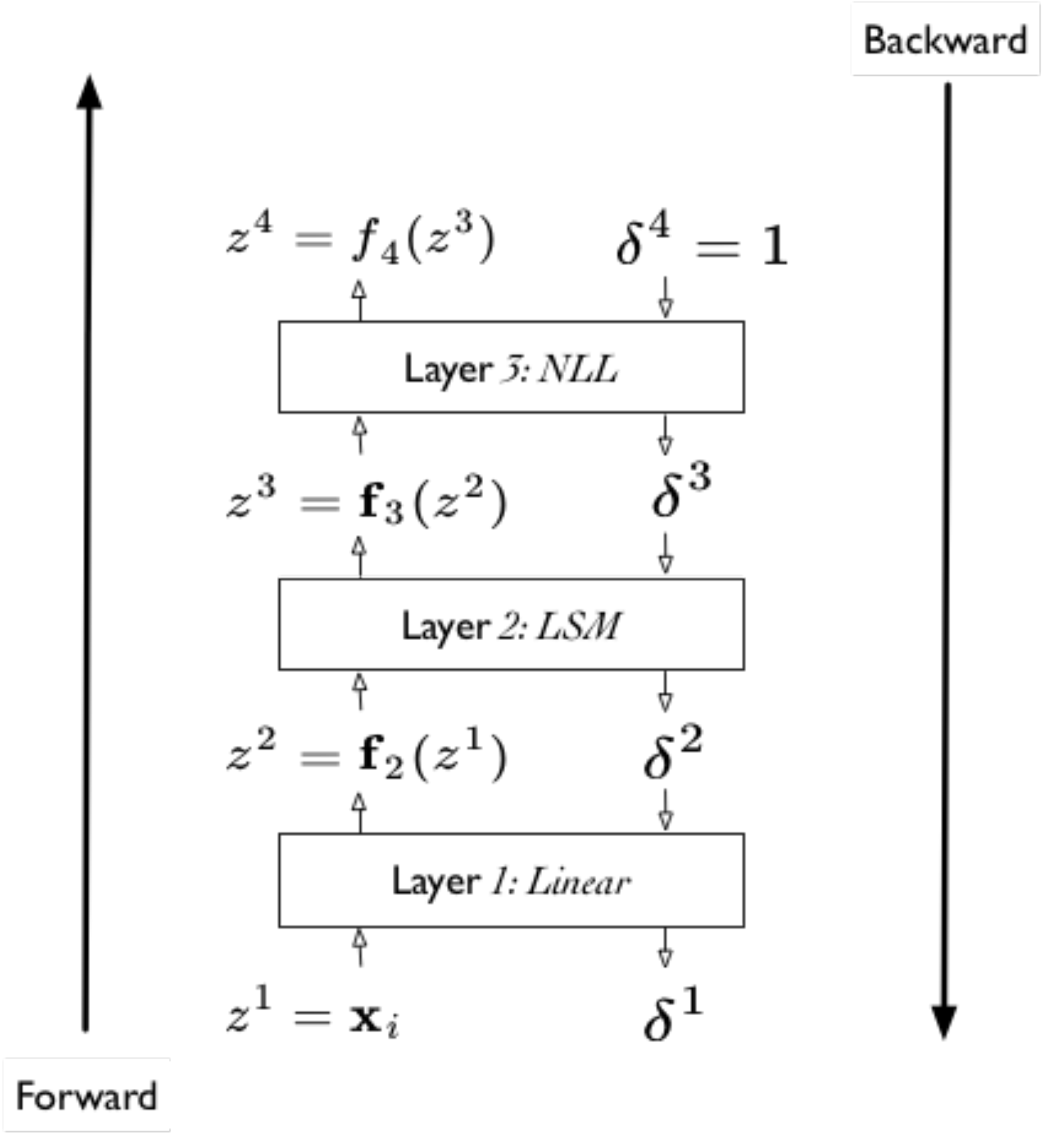
Write as:

$$\nabla_{\mathbf{x}} Cost = (((\frac{\partial f^{Loss}}{\partial \mathbf{f}^3} \frac{\partial \mathbf{f}^3}{\partial \mathbf{f}^2}) \frac{\partial \mathbf{f}^2}{\partial \mathbf{f}^1}) \frac{\partial \mathbf{f}^1}{\partial \mathbf{x}})$$

From Reverse Mode to Back Propagation

- Recursive Structure
- Always a vector times a Jacobian
- We add a "cost layer" to z^4. The derivative of this layer with respect to z^4 will always be 1.
- We then propagate this derivative back.

Layer Cake



Backward
$z^4 = f_4(z^3)$
$\delta^4 = 1$
Layer 3: NLL
$z^3 = \mathbf{f}_3(z^2)$
δ^3
Layer 2: LSM
$z^2 = \mathbf{f}_2(z^1)$
δ^2
Layer 1: Linear
$z^1 = \mathbf{x}_i$
δ^1
Forward

Backpropagation

RULE1: FORWARD (`.forward` in pytorch) $\mathbf{z}^{l+1} = \mathbf{f}^l(\mathbf{z}^l)$

RULE2: BACKWARD (`.backward` in pytorch)

$\delta^l = \frac{\partial C}{\partial \mathbf{z}^l}$ or $\delta^l_u = \frac{\partial C}{\partial z^l_u}.$

$$\delta^l_u = \frac{\partial C}{\partial z^l_u} = \sum_v \frac{\partial C}{\partial z^{l+1}_v} \frac{\partial z^{l+1}_v}{\partial z^l_u} = \sum_v \delta^{l+1}_v \frac{\partial z^{l+1}_v}{\partial z^l_u}$$

In particular:

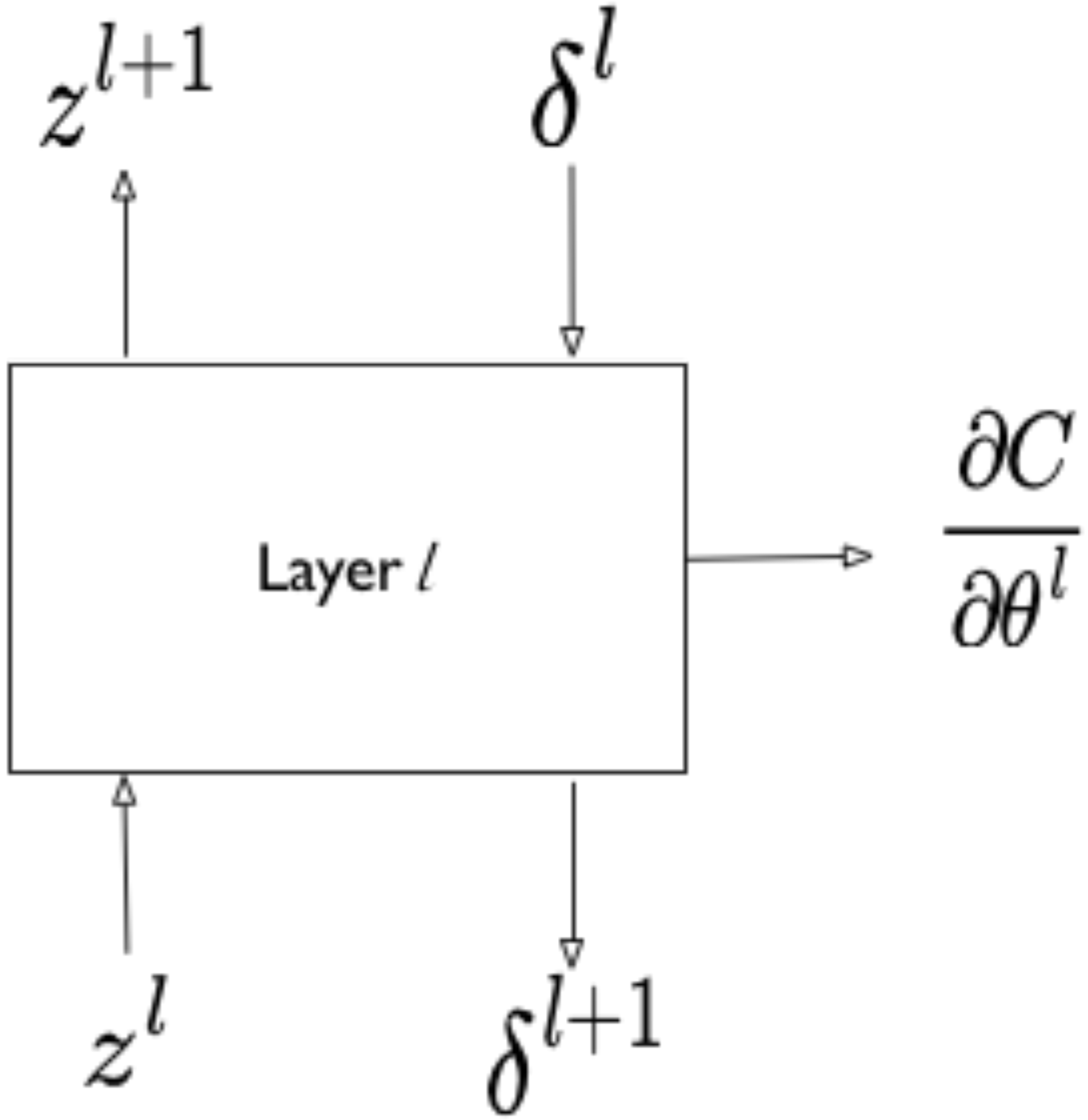
$$\delta_u^3 = \frac{\partial z^4}{\partial z_u^3} = \frac{\partial C}{\partial z_u^3}$$

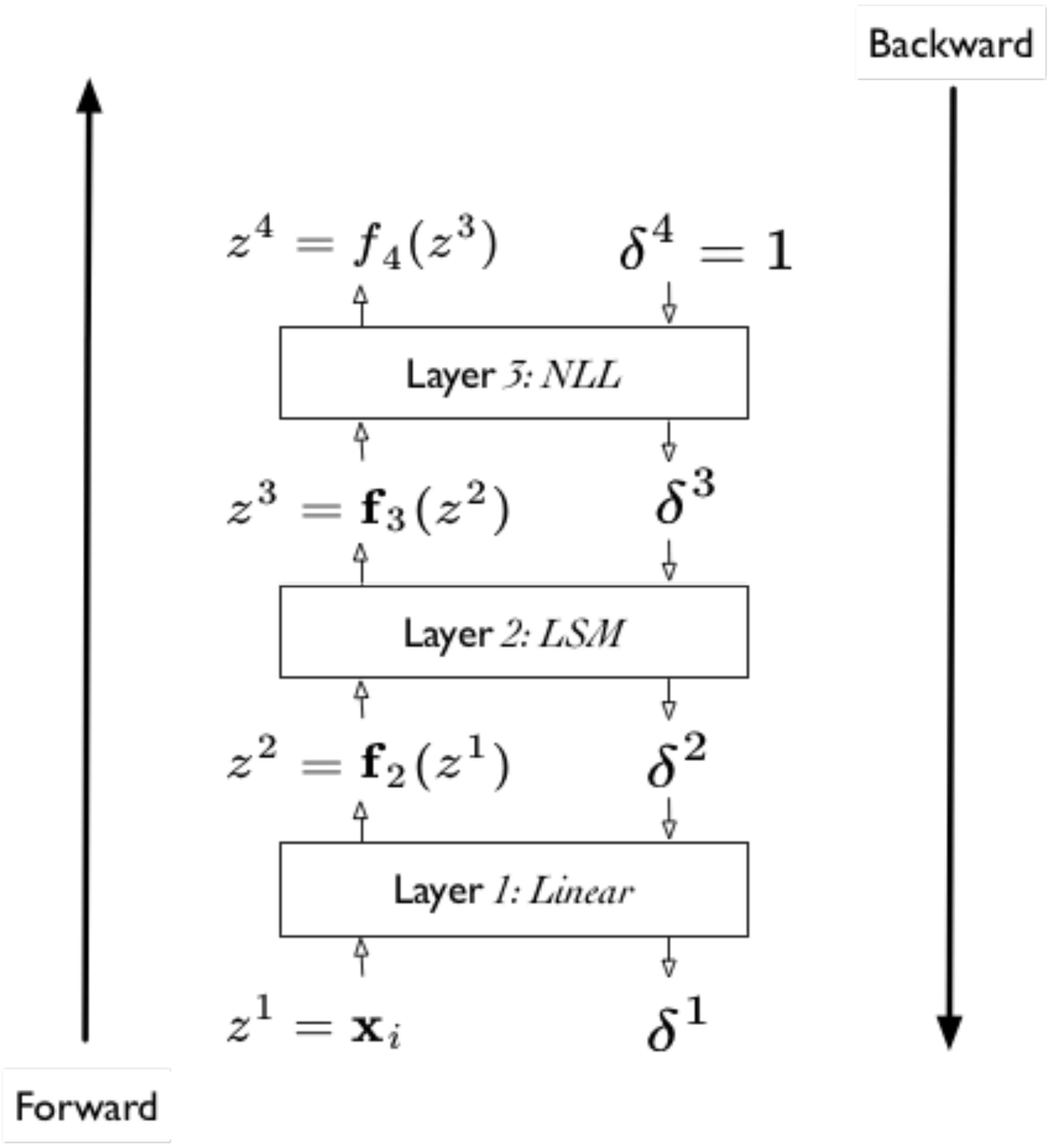
RULE 3: PARAMETERS

$$\frac{\partial C}{\partial \theta^l} = \sum_u \frac{\partial C}{\partial z_u^{l+1}} \frac{\partial z_u^{l+1}}{\partial \theta^l} = \sum_u \delta_u^{l+1} \frac{\partial z_u^{l+1}}{\partial \theta^l}$$

(backward pass is thus also used to fill the `variable.grad` parts of parameters in pytorch)

THATS IT! Write your Own Layer




Backward
$z^4 = f_4(z^3)$
$\delta^4 = 1$
Layer 3: NLL
$z^3 = \mathbf{f}_3(z^2)$
δ^3
Layer 2: LSM
$z^2 = \mathbf{f}_2(z^1)$
δ^2
Layer 1: Linear
$z^1 = \mathbf{x}_i$
δ^1
Forward