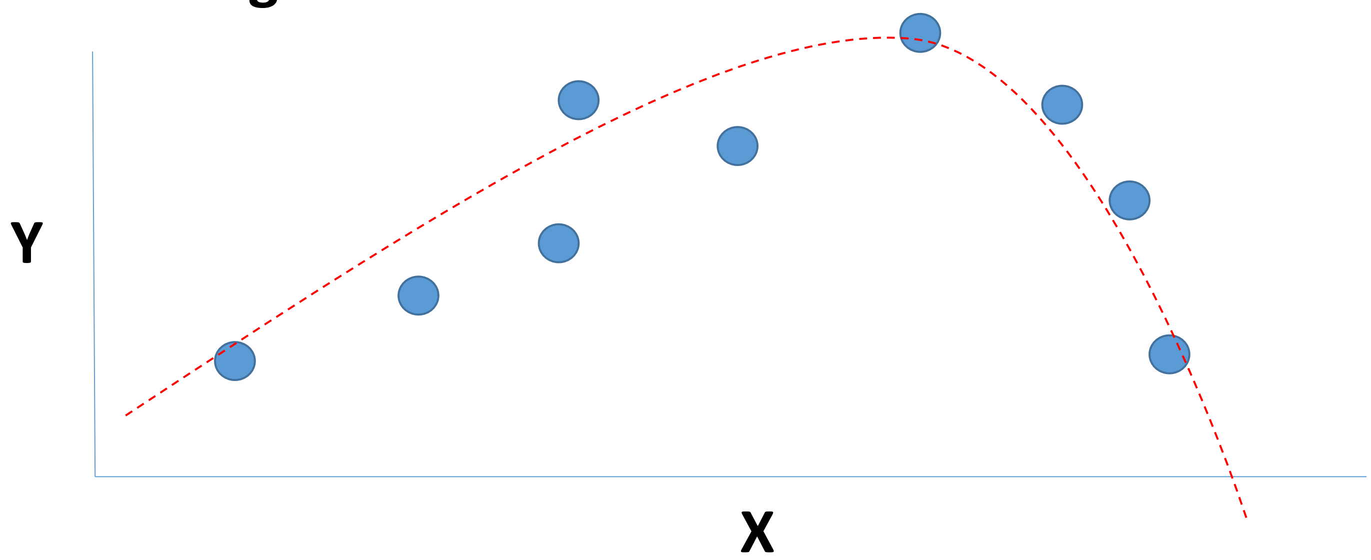
REGULARIZATION

Pick data **D**

The data defines a function to learn: $f(x) = y$

Typically, this is from $\mathbb{R}^d$ to $\mathbb{R}^d$.

This is called **regression**.

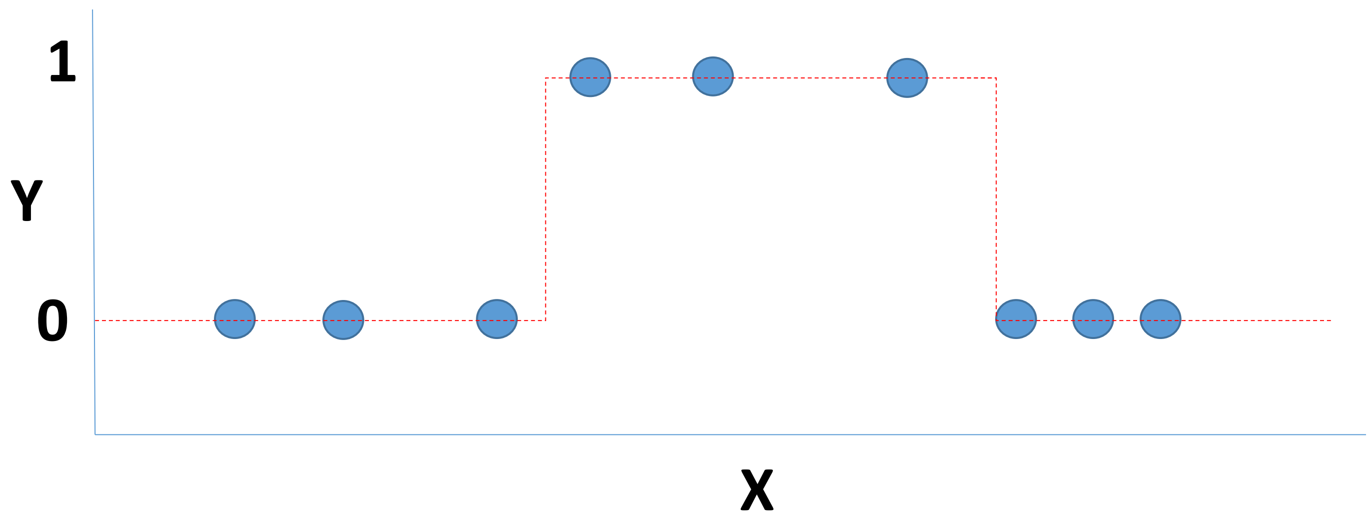


Pick data **D**

The data defines a function to learn: $f(x) = y$

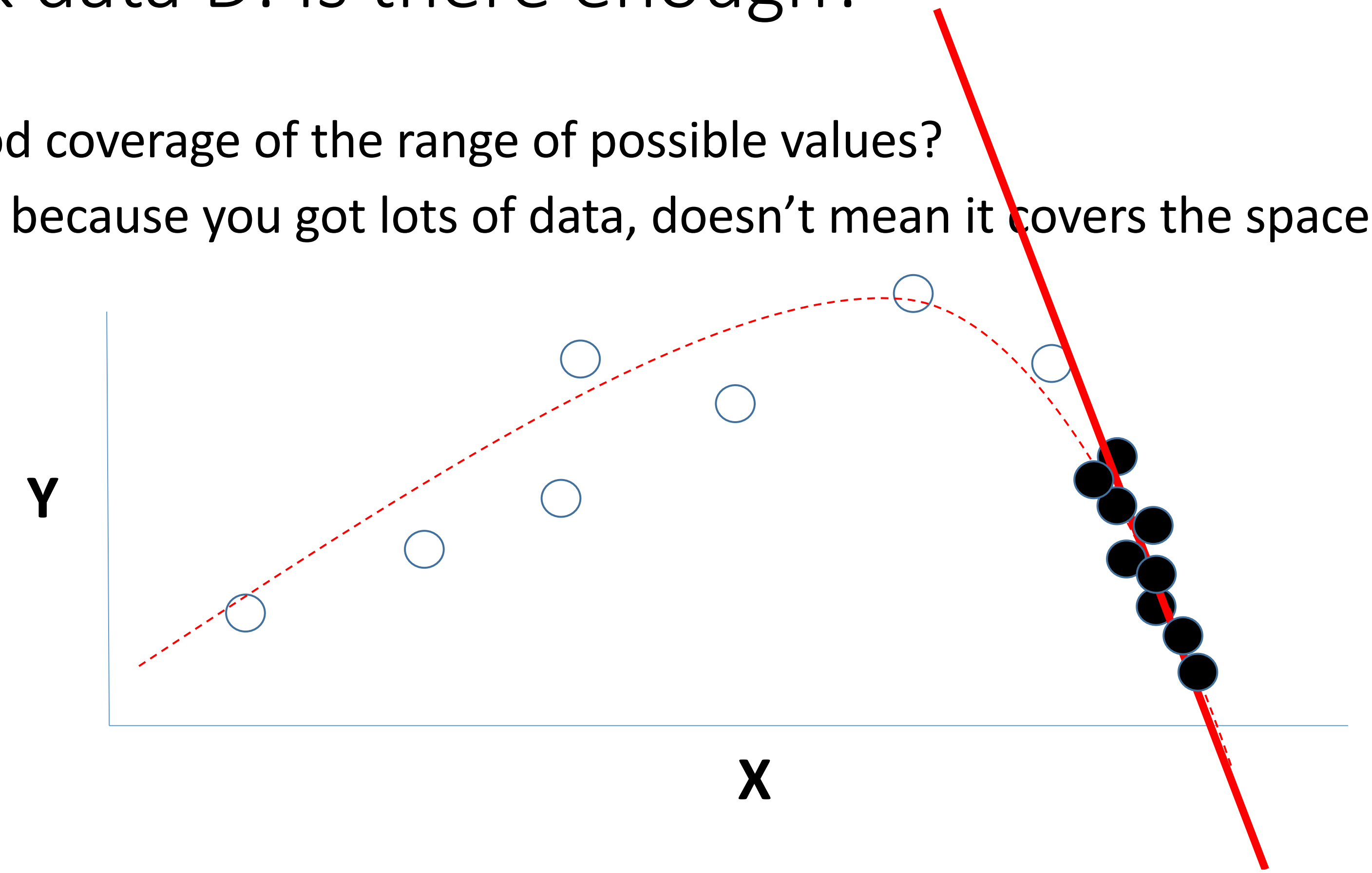
This can also be from $\mathbb{R}^d$ to a finite set of labels, e.g. {0,1}.

This is **classification**.



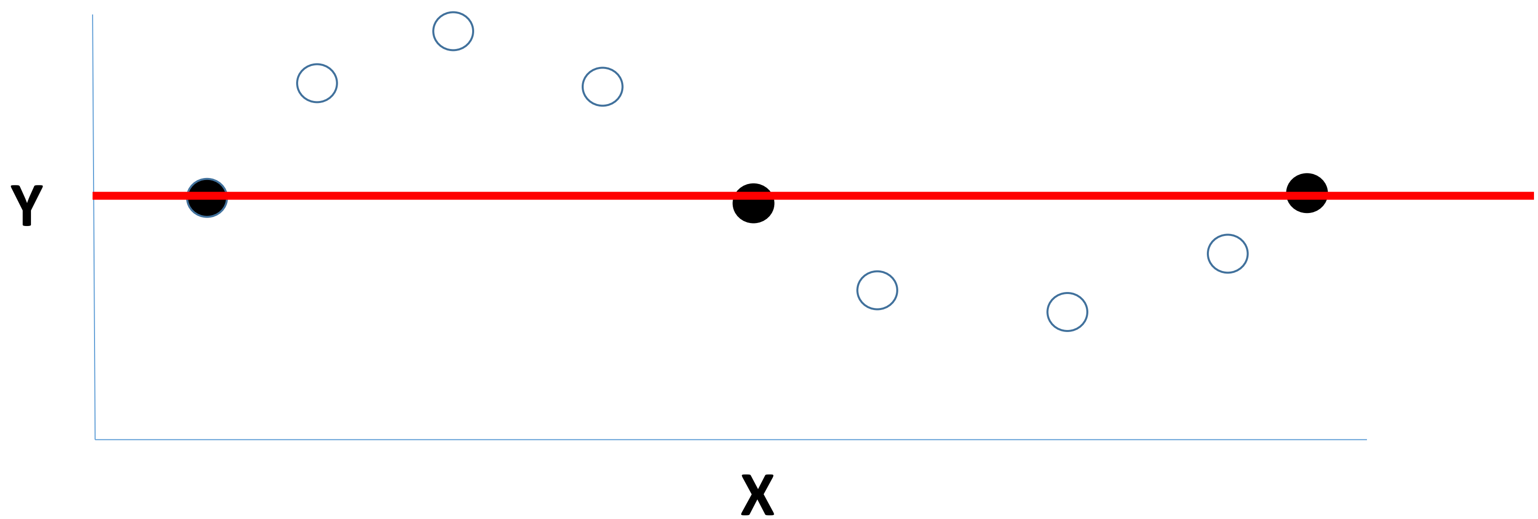
Pick data D: Is there enough?

- Good coverage of the range of possible values?
- Just because you got lots of data, doesn't mean it covers the space.



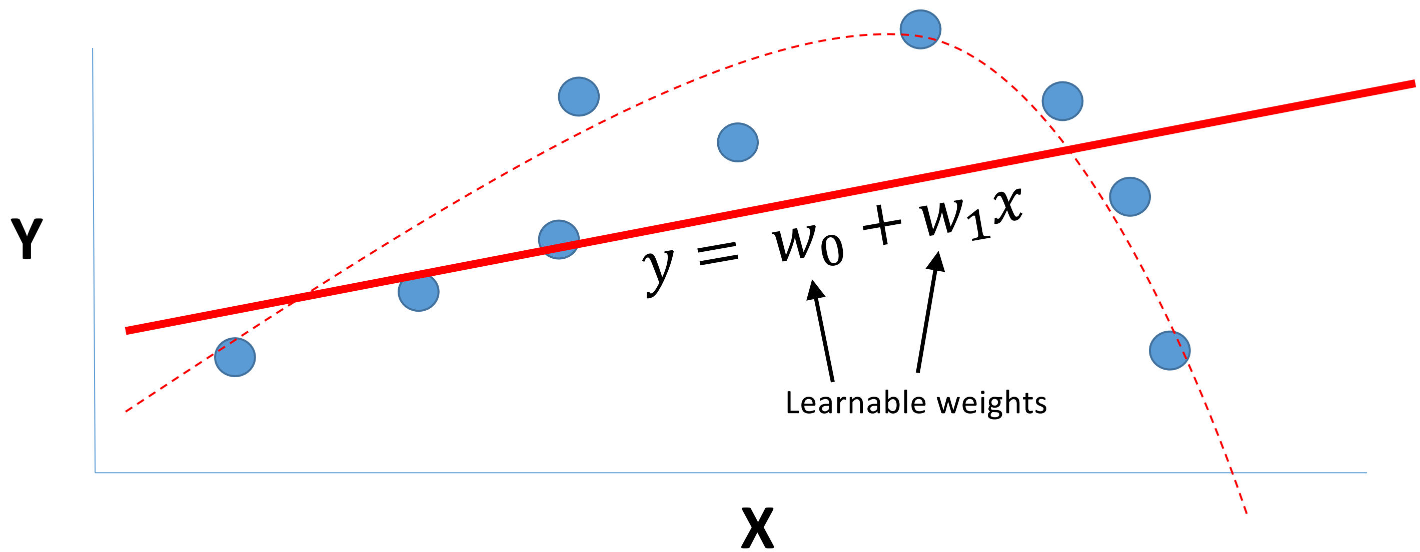
Pick data D: Is there enough?

- Enough sample density in the space?
- Just because you cover the range, doesn't mean you captured the function.



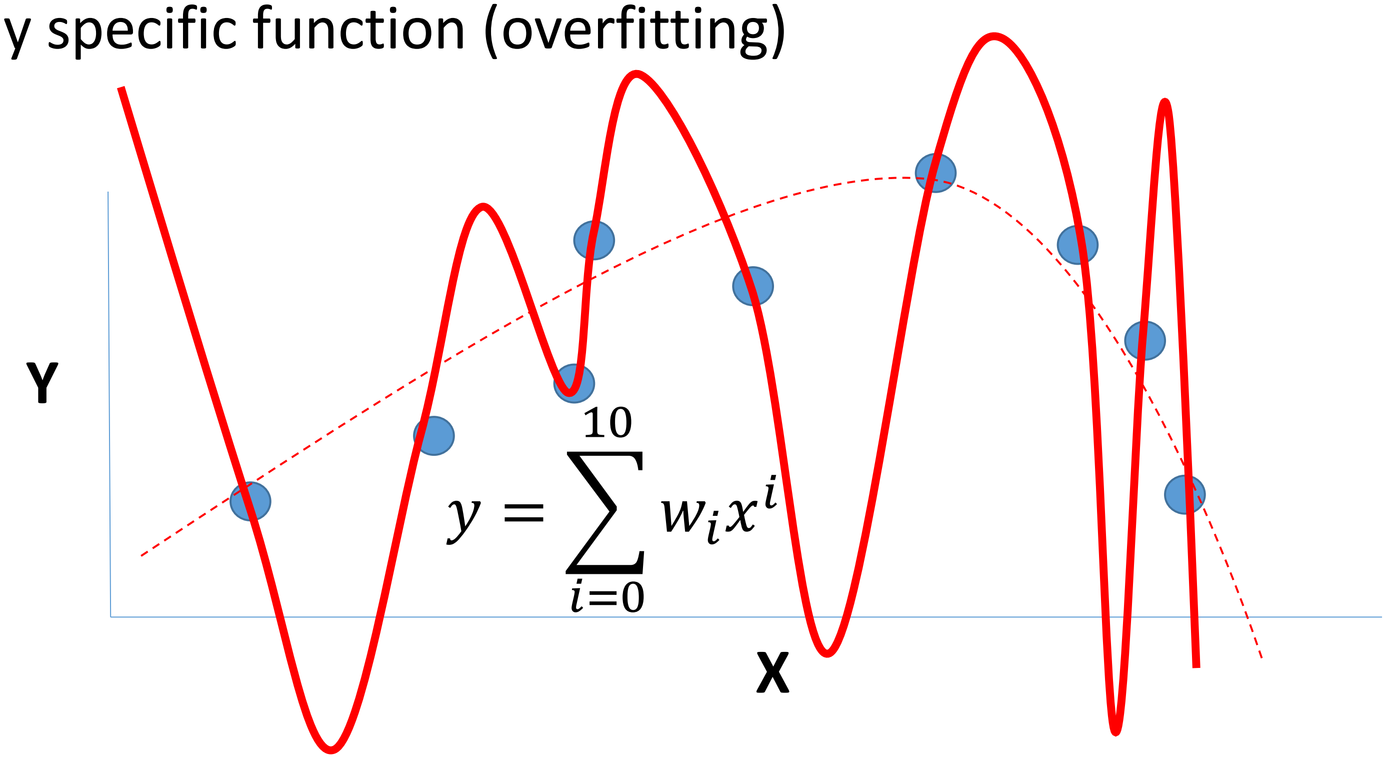
Fitting & Hypothesis space.

If a model's hypothesis space is too small, the true function is probably not in its vocabulary (underfitting)



Fitting & Hypothesis space

If a model's hypothesis space is too big, it can learn a crazy, overly specific function (overfitting)



Revisiting Overfitting

- Overfitting occurs when your model begins to “memorize” the training data
  - Can detect overfitting from an increasing gap between training and validation loss.
  - Performance on the training set improves, but performance on the validation set does not.

Dimensions and data

- The more dimensions your data has, the more data you need to cover the space
- The more dimensions, the more parameters your model needs (at least 1 per dimension)
- The more parameters, the more data you need to prevent overfitting
- Conclusion: You probably don't have enough data. You probably overfit somehow.

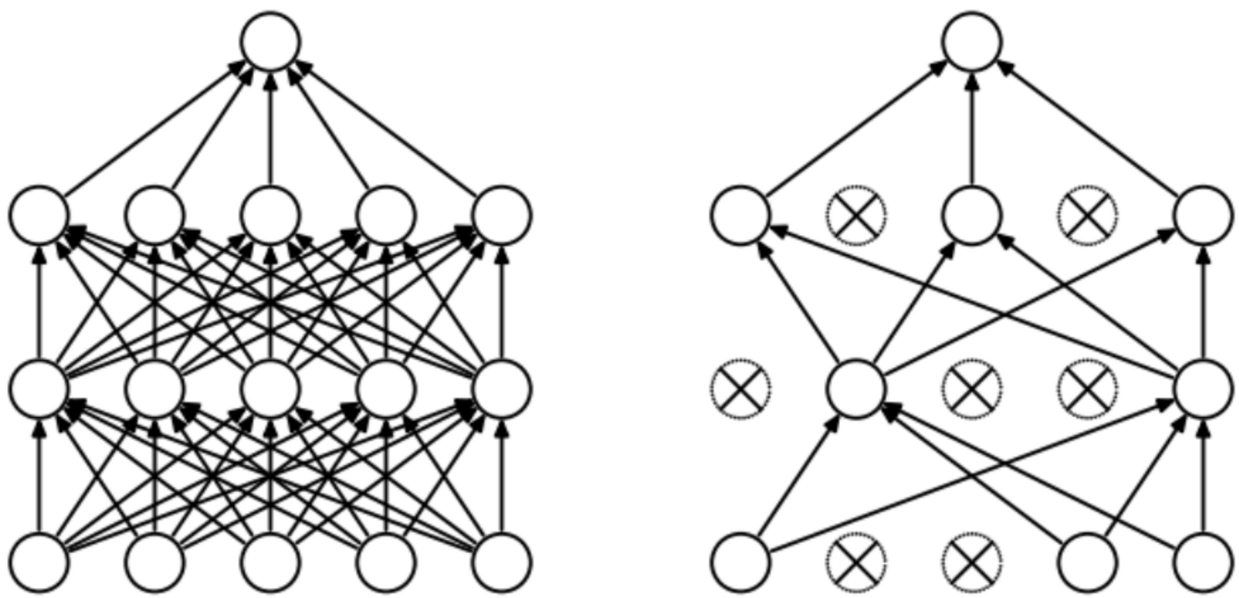
Resisting overfitting

- Add noise to the training process
- Stop before you start overfitting (Early stopping)
- Make the model smaller, somehow (regularization)
- Make the dataset bigger, somehow (augmentation)

Adding noise

- Stochastic Gradient Descent adds noise
- Changing or randomizing step sizes add noise
- Explicitly adding noise to the
  - input data
  - target labels
  - weights (e.g. Dropout)

Regularization via noise: Dropout



(a) Standard Neural Net (b) After applying dropout.

Validation

- Divide data into 3 sets: train, validate, test
- Train on the training data
- Every so often, evaluate on the validation set (which you don’t train on)
- If the loss stops getting better on validation data, stop training
- Only then, when you’re done, do you evaluate on the testing data

“traditional” regularization

- Big idea (**Occam’s Razor**) – Given two models with equal performance, prefer the *simpler* model.
  - E.g., models with fewer parameters or smaller coefficients

- Regularization can be applied to any loss function

$$L_R(X, Y; \theta) = L(X, Y; \theta) + \lambda R(\theta)$$

- The amount of regularization is controlled by the hyperparameter λ

L1- and L2-regularization

- Recall the l_p-norm:

$$\ell_p(\theta) = \sqrt[p]{\sum_{i=1}^{d} |\theta_i|^p}$$

- l_1-regularization penalizes high values of the l_1-norm of the model parameters:

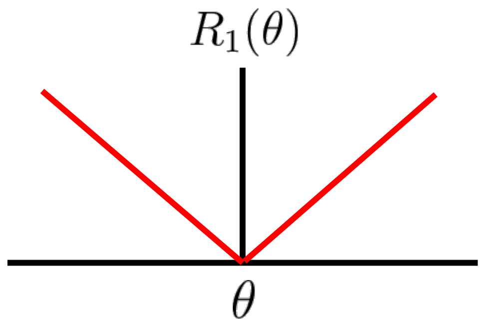
$$R_1(\theta) = \sum_{i=1}^{d} |\theta_i|$$

- l_2-regularization penalizes high values of the l_2-norm:

$$R_2(\theta) = \frac{1}{2} \sum_{i=1}^{d} |\theta_i|^2$$

L1-regularization and sparsity

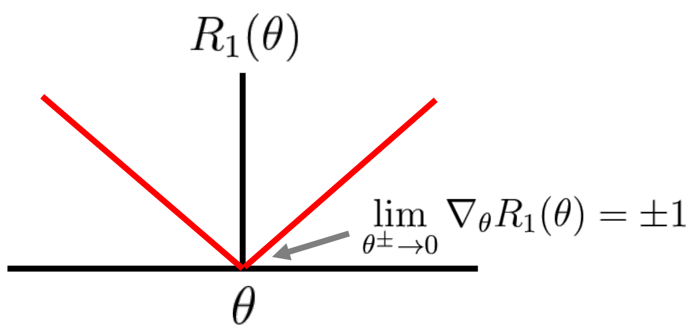
- The gradient of the L1-regularizer is bounded (between -1 and +1, inclusive) but not unique at $\theta = 0$.
- Arbitrarily set the gradient at this point to 0.
- The resulting function is the *sign* function



$$\nabla_\theta R_1(\theta) = \mathrm{sign}(\theta) = \begin{cases} +1, & \theta > 0 \\ 0, & \theta = 0 \\ -1, & \theta < 0 \end{cases}$$

L1-regularization and sparsity

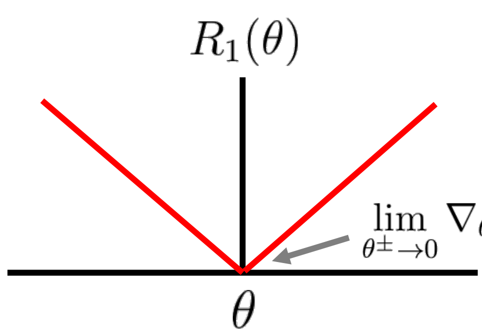
- L1-regularization encourages the model parameters to be *sparse*
  - This is a form of feature selection
  - Only features with non-zero coefficients contribute to the model's prediction
- This is because the gradient of L1-regularization moves model parameters towards 0 at a *constant* rate



L2-regularization and big weights

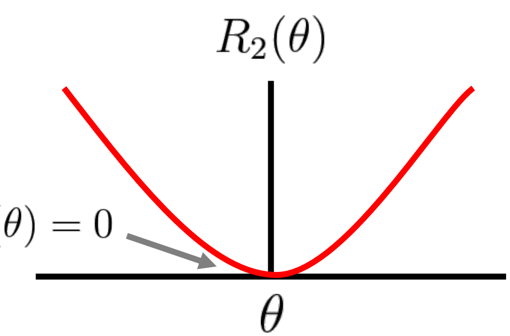
- L2-regularization encourages the model parameters to be *small*

- Why would this be?



$$\lim_{\theta^{\pm}\to 0} \nabla_\theta R_1(\theta) = \pm 1$$

$$\lim_{\theta^{\pm}\to 0} \nabla_\theta R_2(\theta) = 0$$



Regularization and offset (aka bias)

- Many ML models include a bias term, b.

- Example: A linear model: $h_\theta(x) = \theta^T x + b$

- Or equivalently, by augmenting θ and x, like we did with perceptrons…

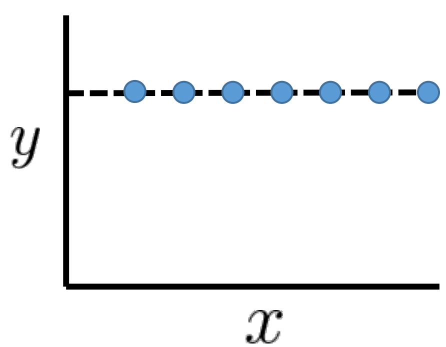
$$\theta' = [\theta_1, \theta_2, \ldots, \theta_d, b],\ x' = [x_1, x_2, \ldots, x_d, 1]$$

- What happens if we regularize the bias term?

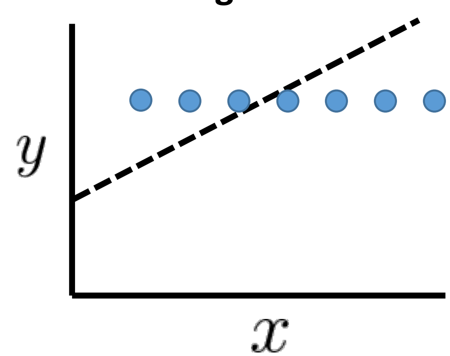
Regularization and offset (aka bias)

- Recall that “regularizing” a model parameter means encouraging that model parameter to tend towards 0.
- How would a linear model represent horizontal line?
- How does shrinking the bias affect its ability to do so?

No bias regularization



Bias regularization



Don’t regularize the bias term!

Data Augmentation

- Make perturbed copies of your data that vary in ways that should not change the value nature of the output function.

- This can help prevent spurious correlations between data and output.

- Example: Distinguishing clarinet sounds from flute sounds
  - Vary the pitch of each note by + or – 1%, 2%, 3%, 4%....
  - Add background noise of different kinds and at different dB
  - Time-stretch each note a bit
  - Delay or advance the onset of the note

- This can turn 1000 data points into 100,000.

Let’s look at “trees”

One other thing…"normalization"

What if....

Your training data looks like this?

High dynamic range
Very bright

(image from https://www.dreamstime.com/)

Your testing data looks like this?

Low dynamic range
Very dark

You might want to do this:

- Standardize your data:
  - Make sure that you have unit variance in your batch/dataset.
- Give your data the same range overall (e.g. center your values around the same center point)
- Decorrelate your variables (can be harder for images, if every pixel is a variable)

“Whitening your data”

- name comes from white noise.
- Here’s a definition from https://en.wikipedia.org/wiki/Whitening_transformation
- A **whitening transformation** or **sphering transformation** is a linear transformation that transforms a vector of random variables with a known covariance matrix into a set of new variables whose covariance is the identity matrix, meaning that they are uncorrelated and each have variance 1.[1] The transformation is called "whitening" because it changes the input vector into a white noise vector.

So....when/how to do this?

- At the dataset level?
- At the batch level?
- At the input?
- Or further into the network?

What is covariant shift?

- Well… we already talked about this potential issue between testing and training

- How can a similar issue happen between gradient descent steps for the input, if we're using minibatches?
- How can a similar issue happen to interior nodes even if we run on the same mini batch for two steps in a row?

Batchnorm: centering and scaling

- Normalize the data scale input to each node
- Subtract the mean value of the data
- Do this on a dimension-by-dimension basis
- Do this at every training step in gradient descent

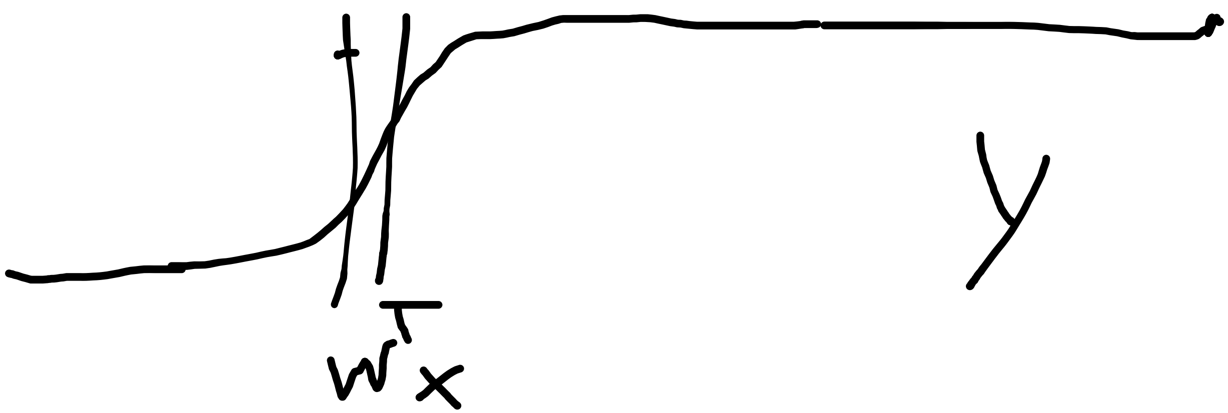
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$


$w^T x$
y

What to do once trained?

Replace batch statistics with the statistics over the entire training set.

$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x] + \epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}}\right)$$