W04D4

Optimization

Instructor: Eric Elmoznino

Outline for today

- Issues and challenges in ML
- Optimization
    - Gradient descent
    - Stochastic (and mini-batch) gradient descent
- Break
- Regularization
    - L1 (Lasso) regularization
    - L2 (Ridge) regularization
    - Elastic Net regularization

Issues and challenges in ML

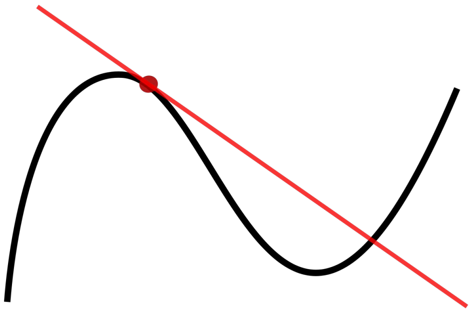
Machine learning at its worst

- Machine learning models fit the data we give them
- If the data is sexist, the model will be too. If the data is racist, the model will be too. Any **bias** in the model is a pattern that the model will learn
- Amazon trained a recruiting model that exhibited gender bias
- COMPAS predicted risk of repeating a crime and exhibited racial bias
- We want models to be fair and objective, so this is a big issue
- The bias is not in the model; it is in the *data*
- Be wary of thinking of these models as “artificial intelligence”. They don’t think, they find patterns in data

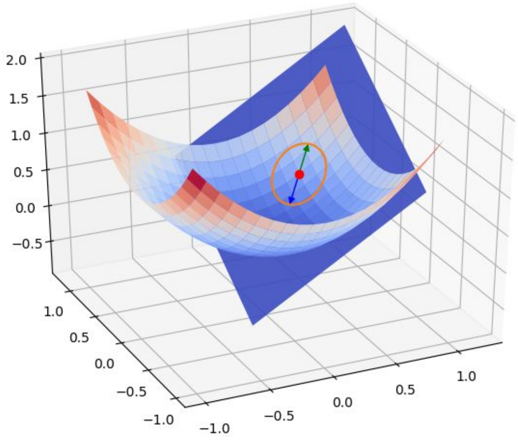
Optimization

The gradient: multidimensional derivative

Derivative for 1D independent variable (slope)



Derivative (gradient) for >1D independent variable (slope + steepest direction)

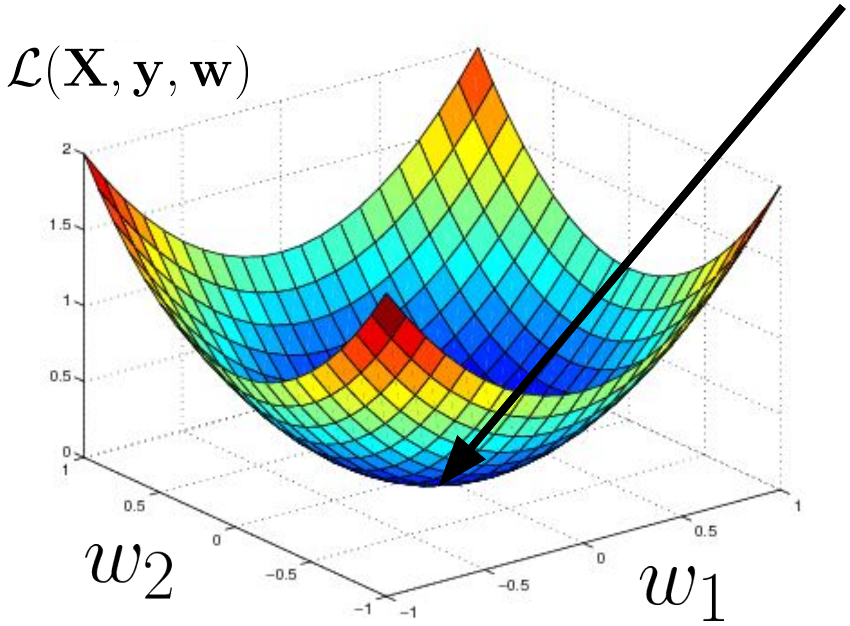


Derivative of function with respect to a vector

$$\frac{\partial}{\partial \mathbf{v}} f(\mathbf{v}, \ldots) = \begin{bmatrix} \frac{\partial}{\partial v_1} f(\mathbf{v}, \ldots) \\ \frac{\partial}{\partial v_2} f(\mathbf{v}, \ldots) \\ \vdots \\ \frac{\partial}{\partial v_n} f(\mathbf{v}, \ldots) \end{bmatrix}$$

Interpretation: *f* would increase most if **v** moved in this direction

Motivation and intuition



Lowest loss (best performance)

$\rightarrow$ Optimal weight vector $\hat{\mathbf{w}}$

$\rightarrow$ Occurs when $\mathcal{L}$ is flat

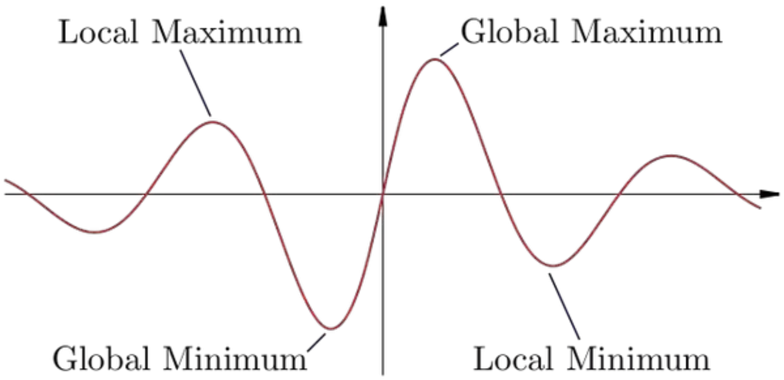
$\rightarrow$ $\mathcal{L}$ is flat when $\dfrac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = 0$

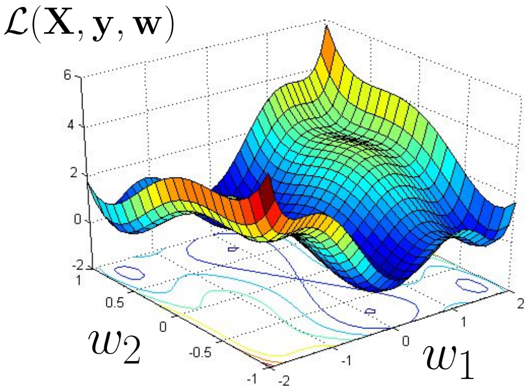
Just solve for $\hat{\mathbf{w}}$, right!?
For linear regression:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Motivation and intuition

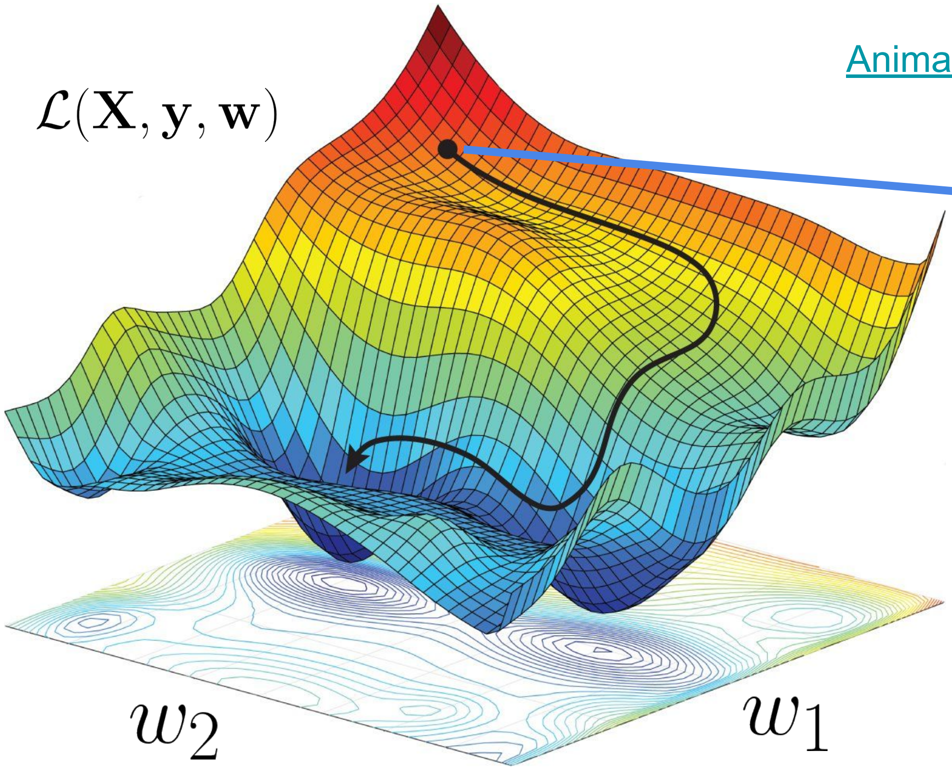
- Sometimes, we can’t solve for the **w** that makes the derivative (gradient) of the loss function 0 (e.g. deep neural network)
- Other times, there are multiple solutions where the gradient will be zero and the loss function is in a valley (i.e. loss function is not *convex*)





Gradient descent: going downhill

[Animation (near the bottom)]



Gradient descent

Loss function with respect to data, labels, and parameters

$$\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w})$$

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w})$$

Gradient: derivative of loss with respect to parameters

Example for linear regression

$$\hat{y}_i = \mathbf{w}\mathbf{x}_i$$

$$\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}\mathbf{x}_i)^2$$

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{-2}{n}\sum_{i=1}^{n}\mathbf{x}_i(y_i - \mathbf{w}\mathbf{x}_i)$$

Gradient descent procedure

1) Initialize the parameters (e.g. randomly)
2) Compute the gradient of the loss function with respect to the parameters

   *The loss function would **increase** if we moved the parameters in the direction of this gradient*

3) Move the parameters in the **opposite** direction of the gradient so that the loss function would **decrease**

   The parameters are now better because they result in a lower loss. This is the goal of training

4) Repeat (2) and (3) several times so that the loss gradually decreases

We can always use gradient descent to optimize a set of parameters, so long as the loss function is **differentiable** with respect to those parameters

Gradient descent decisions

- How much do we move in the opposite direction of the gradient each update iteration? This is called the **learning rate**
  - Optimal setting depends on your model and the loss function
  - Don’t know the optimal value beforehand. Try different values, see which is best (grid search with cross-validation)
  - Doesn’t necessarily need to be a constant value (learning rate scheduling/annealing)
- How many update iterations do we want to perform?
  - Can use a constant value. Number of times we update using the entire dataset is called the number of **epochs**
  - Can keep updating until loss stops decreasing (i.e. loss has **converged**)

Gradient descent algorithm

Algorithm: Gradient Descent

Initialize $\mathbf{w}$ (e.g. randomly)

for $epoch \in nEpochs$ **do**

$$\mathbf{w}_{grad} = \frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w})$$

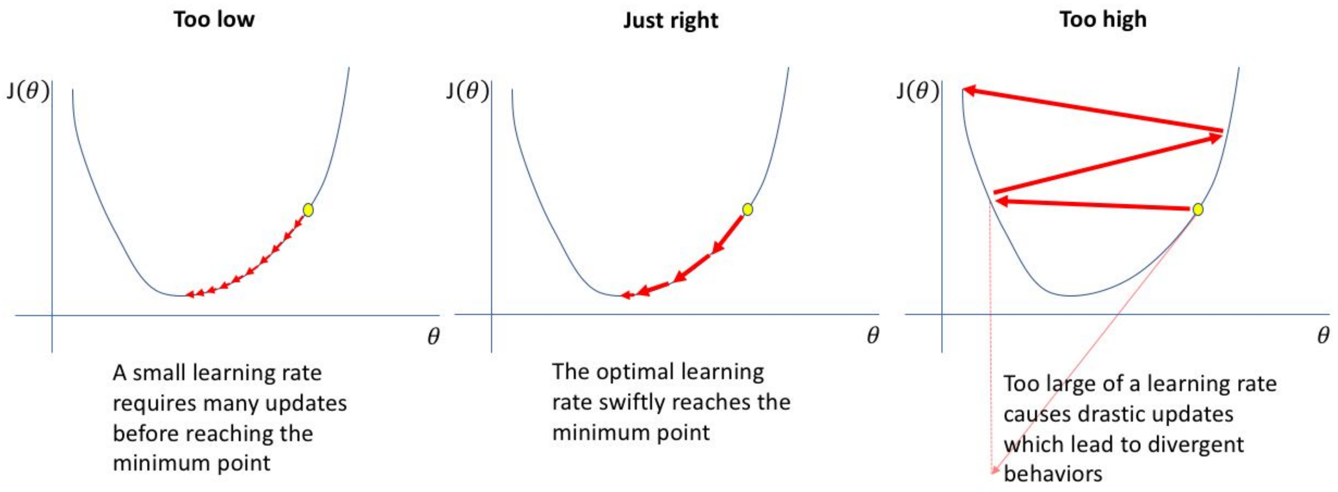
$$\mathbf{w} = \mathbf{w} - \alpha \mathbf{w}_{grad}$$

end

α denotes the Learning Rate

Effect of the learning rate

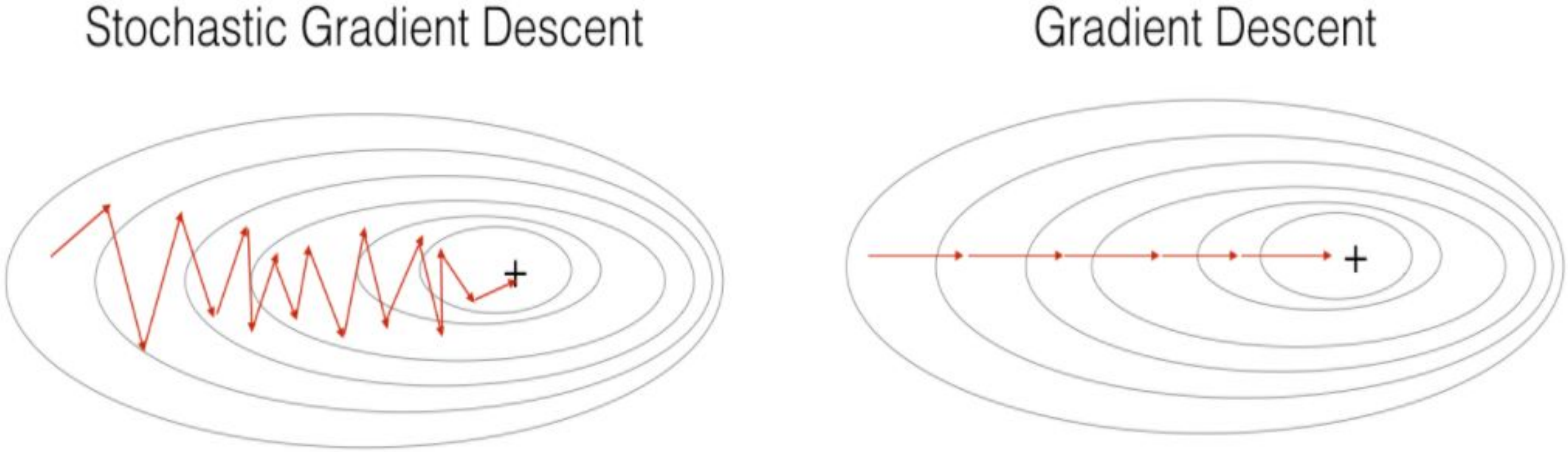
- Learning rate too large: end up missing the local minima (overshooting)
- Learning rate too small: learning takes too long (too many iterations)



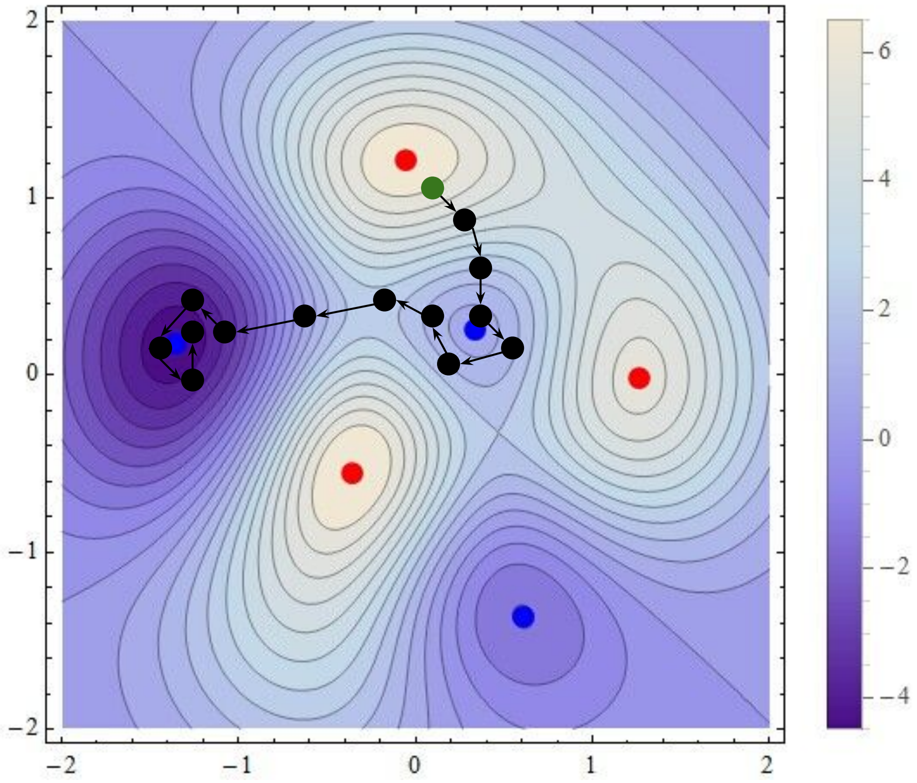
Stochastic gradient descent

- Gradient descent can be computationally expensive and slow (uses the entire **X** dataset each weight update)
- Stochastic gradient descent: randomly select *one* data point (row in **X**) and update using its gradient
- Results in noisy approximate of the true (full dataset) gradient
- In practice, using noisy updates converges to better solutions that are closer to the *global minimum* (can escape bad local minima)

Stochastic gradient descent



Stochastic gradient descent: escaping bad minima



Stochastic gradient descent algorithm

Algorithm: Stochastic Gradient Descent

Initialize $\mathbf{w}$ (e.g. randomly)

for $epoch \in nEpochs$ **do**

  shuffle $\mathbf{X}$

  for $\mathbf{x}_i \in \mathbf{X}$ **do**

    $\mathbf{w}_{grad} = \frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i, \mathbf{w})$

    $\mathbf{w} = \mathbf{w} - \alpha \mathbf{w}_{grad}$

  end

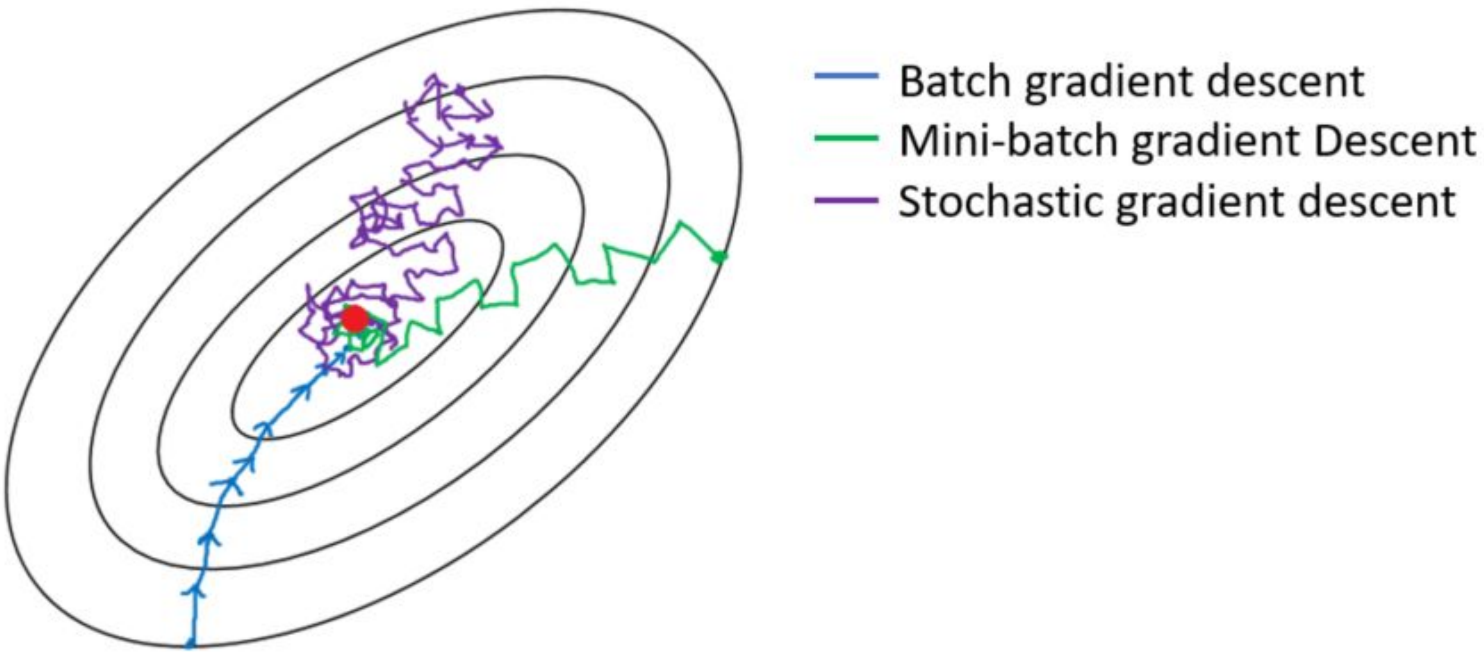
end

α denotes the Learning Rate

Mini-batch gradient descent

- A bit of noise can help us find the optimal solution by escaping local minima
- Too much noise can slow training due to lack of sustained downhill progress
- Big matrix multiplications may make our computers run out of memory
- Small matrix multiplications may not make efficient use of parallel computing
- **We often want something in between**: estimate the gradient each iteration using some, but not all, of the data

Mini-batch gradient descent



Mini-batch gradient descent algorithm

Algorithm: Mini-Batch Gradient Descent

Initialize $\mathbf{w}$ (e.g. randomly)

for $epoch \in nEpochs$ **do**

  shuffle $\mathbf{X}$

  for $\mathbf{X}_{i:i+BS} \in \mathbf{X}$, *with i increasing by BS at a time* **do**

    $\mathbf{w}_{grad} = \frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{X}_{i:i+BS}, \mathbf{y}_{i:i+BS}, \mathbf{w})$

    $\mathbf{w} = \mathbf{w} - \alpha \mathbf{w}_{grad}$

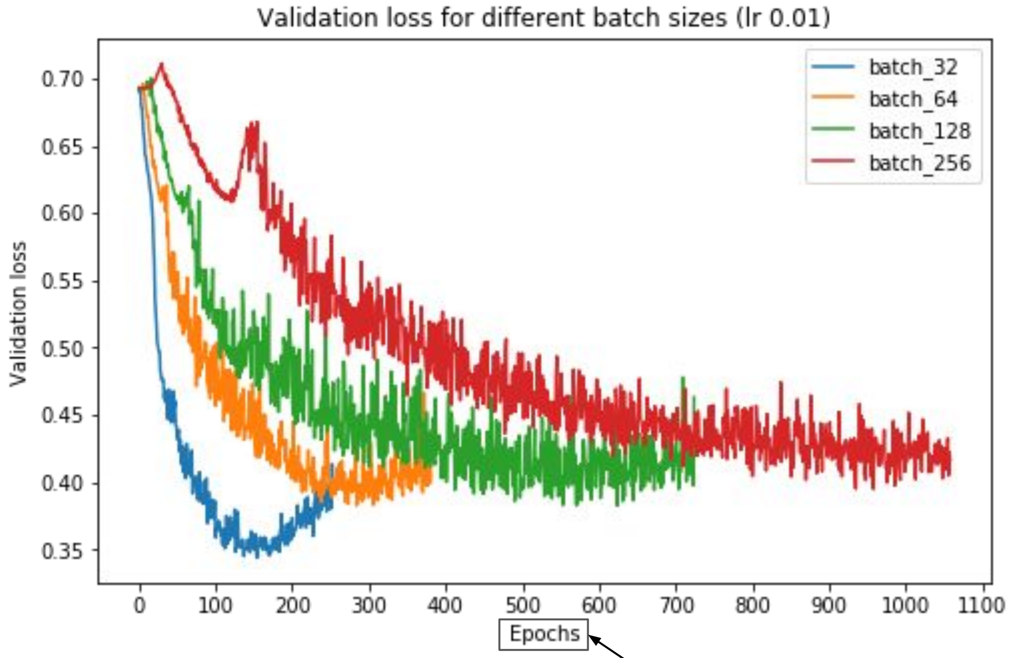
  end

end

α denotes the Learning Rate

BS denotes the Batch Size

Effect of the batch size

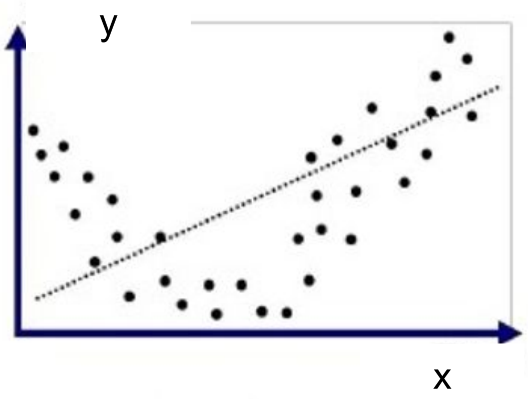


Not the same as # of iterations (updates)

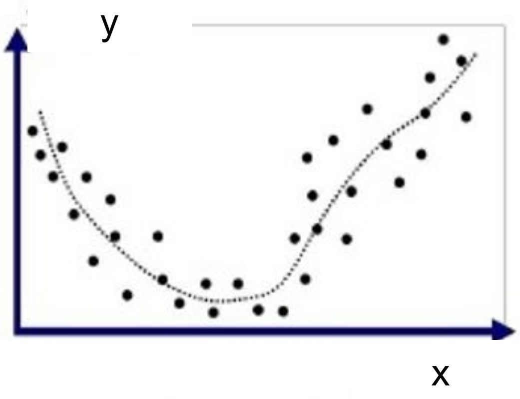
Note: graph is for a deep neural network (many local minima, noise is more important)

Regularization

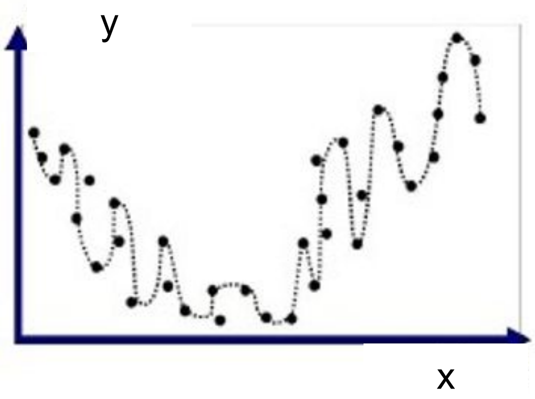
Motivation



Underfitted



Good Fit/Robust

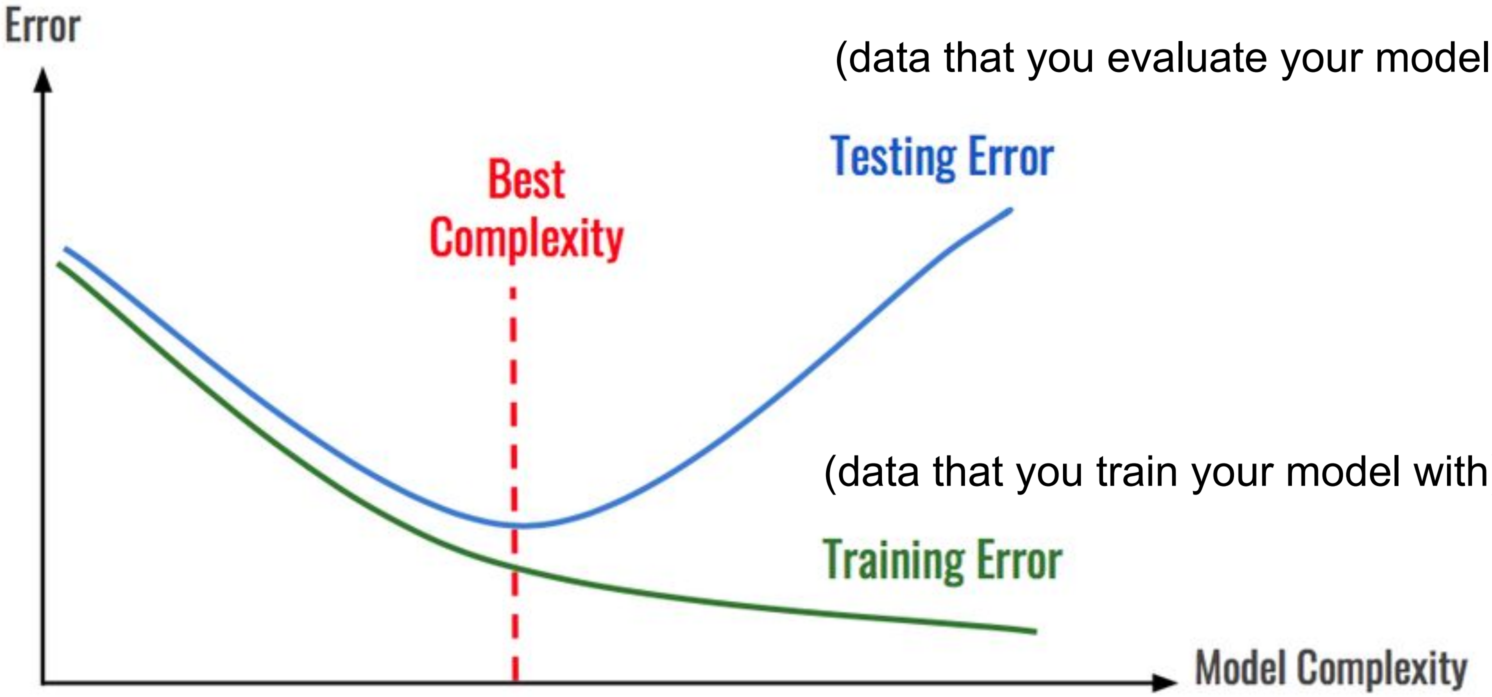


Overfitted

Motivation

- Data has two components components: *signal (pattern)* + *noise*
- Example: predicting house prices from # of bedrooms, area, age, etc.
    - Signal: degree to which these features influence the price
    - Noise: random variation, or variation due to unknown features
- **Goal of machine learning**: model the pattern, ignore the noise
- When the model is fitting (trying to predict) the noise, we say that it is **overfitting**
- Overfitting is undesirable, because the noise is random and therefore won't be the same on new data seen out in the real world

Detecting overfitting



(data that you evaluate your model on)

(data that you train your model with)

What is model complexity?

- The space of functions a model can learn
- Influenced by:
    - Model architecture (structure, type)
    - Model flexibility (e.g. number of parameters)
    - The particular solution we converge to (i.e. final parameters)
- Example for linear regression architecture: parameters come from weights connecting features to dependent variable
    - Complexity of linear regression models increases as the number of parameters increases
    - The number of parameters increases with the number of features you use (dimensionality)

Combating overfitting

- Option 1: use a less powerful model
- Option 2: reduce the number of parameters
  - For linear regression, corresponds to reducing the number of features (dimensionality reduction or feature selection)
- Option 3: limit the parameter space (effectively reducing the space of possible functions the model can learn)
  - A common way of doing this is to use parameter **regularization**

Regularization

- Constrain the parameter space by adding an additional loss term on the model parameters
- With this *weight penalty*, parameters can no longer vary freely

$$\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \underbrace{V(\mathbf{X}, \mathbf{y}, \mathbf{w})}_{\text{prediction error}} \quad + \quad \lambda \quad \underbrace{R(\mathbf{w})}_{\text{weight penalty}}$$

$$\text{Where } \lambda \geq 0$$

Ridge regression

- Uses an *L2 penalty* (penalizes weights based on their *squared sum*)
- In practice:
    - Prevents overfitting when there is a lot of collinearity (correlation) between the features
    - Model has reduced variance (more consistent model for small variations in the data)
    - Irrelevant features get small weights, instead of being used by the model to fit noise

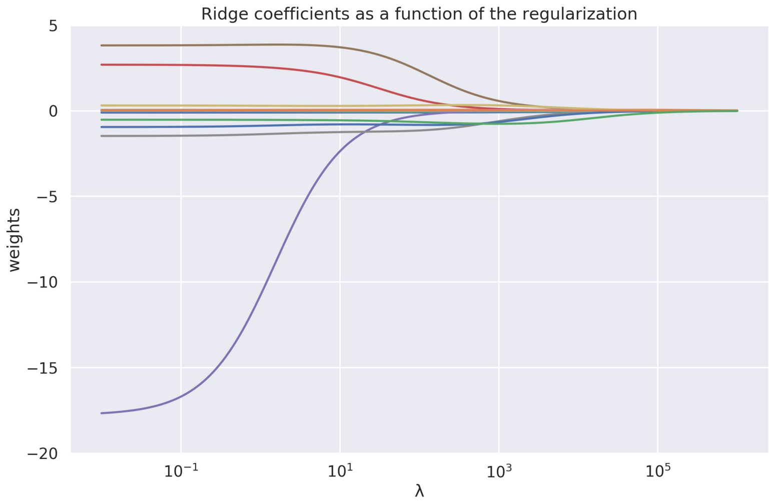
$$
\begin{aligned}
\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) &= V(\mathbf{X}, \mathbf{y}, \mathbf{w}) \quad + \quad \lambda \quad \mathbf{w}^T \mathbf{w} \\
&= V(\mathbf{X}, \mathbf{y}, \mathbf{w}) \quad + \quad \lambda \quad \sum_{i=0}^{n} w_i^2
\end{aligned}
$$

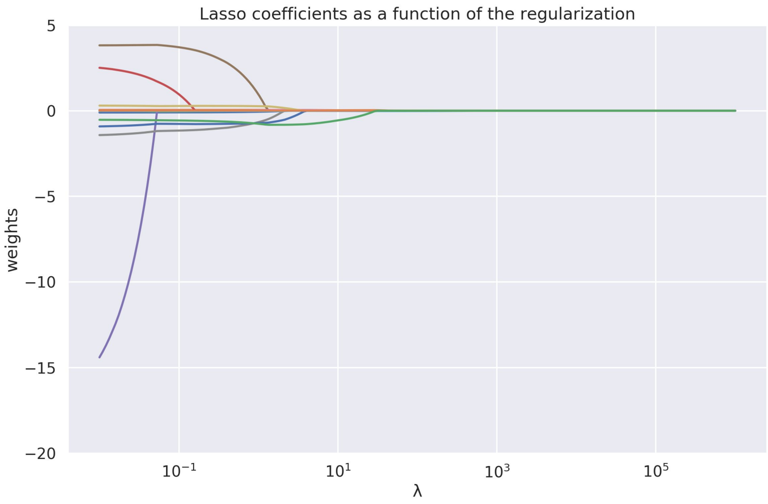
Lasso regression

- Uses an *L1 penalty* (penalizes weights based on their *absolute value sum*)
- In practice:
    - Prevents overfitting when there is a lot of collinearity (correlation) between the features
    - Model has reduced variance (more consistent model for small variations in the data)
    - Irrelevant features get weights of 0, instead of being used by the model to fit noise
    - Can be used for feature selection (pick the features with > 0 weight)

$$\begin{aligned}\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) &= V(\mathbf{X}, \mathbf{y}, \mathbf{w}) \quad + \quad \lambda \quad |\mathbf{w}| \\ &= V(\mathbf{X}, \mathbf{y}, \mathbf{w}) \quad + \quad \lambda \quad \sum_{i=0}^{n} |w_i|\end{aligned}$$

Ridge vs. Lasso





Elastic net regression

- Generally, we care about getting the best performance on the test set. We don't care if we do it using L1 or L2 regularization
- Elastic net uses both, each with their own λ

Picking λ

- If λ is too small, we can overfit (model too complex)
- If λ is too large, we can underfit (model ignores prediction error)
- Like all other hyperparameters, the simplest way to pick it is to try a lot of values and see which works best (grid search with cross-validation)