Optimization Methods

Sabya D.
| Lighthouse Labs Instructor, Data Analytics

W06D3

Who am I >

Instructor: Dr. Sabya DG (sabya.datatech@gmail.com)

Director/Founder, Altius AI Solutions Inc.

-PhD in Theoretical Physics, Germany

-Postdoc: NUS, Singapore / UToronto

Superpowers:

- Microsoft Azure, Data Platform Solutions
- Big data Engineering, MLOPs
- Unsupervised ML / Statistical Methods

LinkedIn: **https://www.linkedin.com/in/sabyadg/**

"If you want to master something, teach it" ... Richard P. Feynman

Agenda

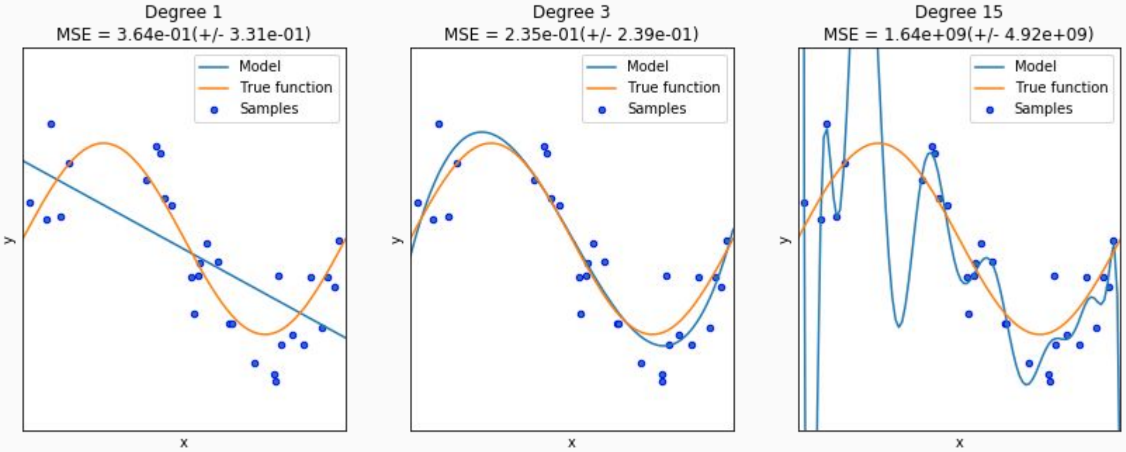
- Taming ML models
  - Issues & challenges in ML
  - Bias vs. Variance in ML
- Optimizing Loss functions:
  - Gradient descent
  - Stochastic gradient descent
- Regularization
  - L1/Lasso
  - L2/ Ridge
  - Elastic-Net/Dropout
- Demo 1

Part I

Taming ML models

Issues & Challenges in ML I

- Avoid overfitting and underfitting
- An over-fit model generalizes poorly: •An underfit model fails to capture th general trend



Underfit
High Bias

Just right
Low Bias
Low Variance

Overfit
High V

Bias & Variance | Classification

Bias to Variance Tradeoff

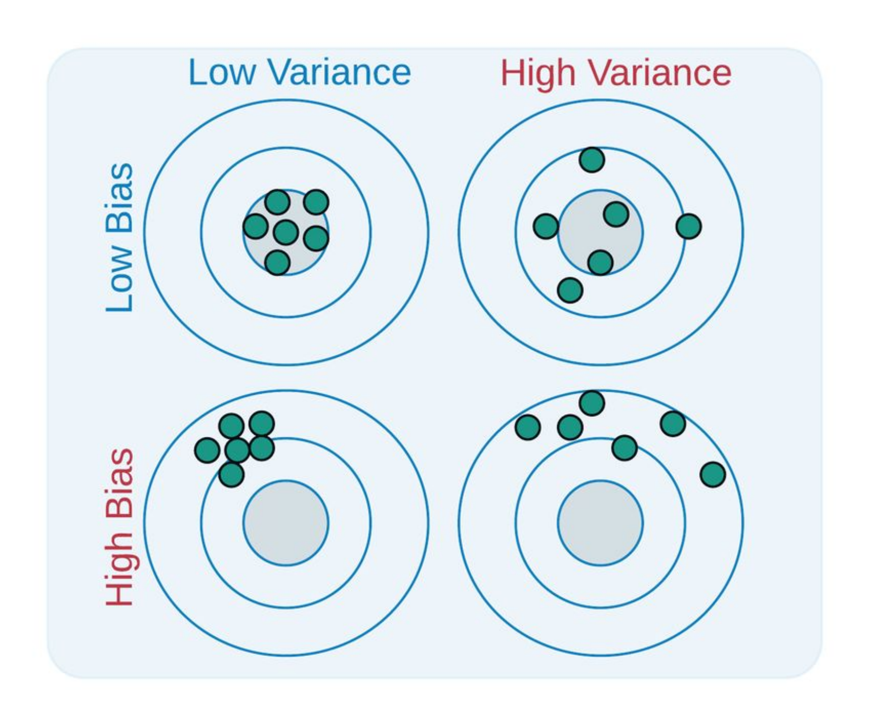
Bias (B) : Inability of a model to find the true relationship between the features and the label in the training dataset. Difference between actual and predicted values

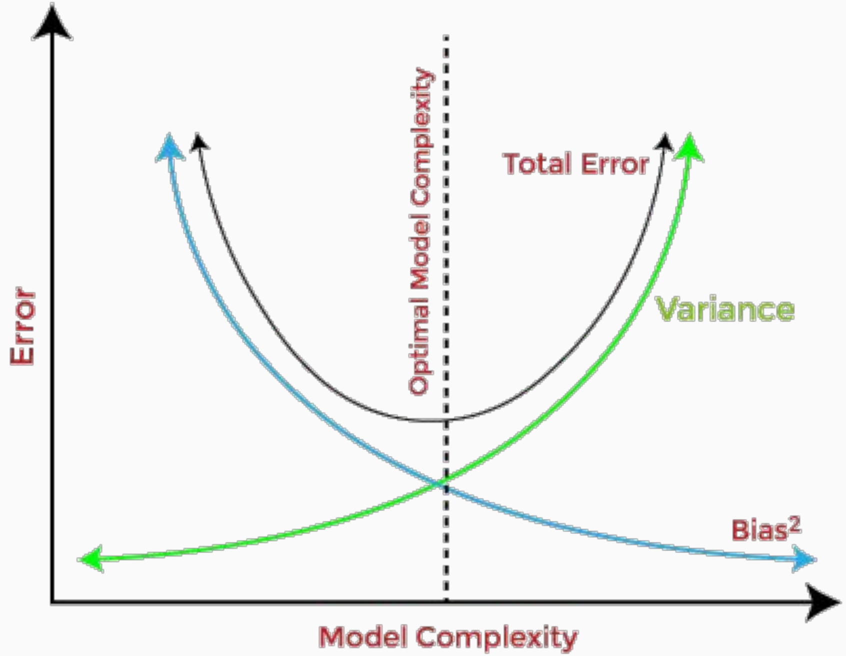
Variance (V) : Sensitivity of a model to change in data. Difference between performance errors when used on different datasets

Bias and Variance Tradeoff
- Overfit models : High V Low B
- Underfit models: Low V High B

Optimal fits: Model with the right fit has a balance between bias and variance

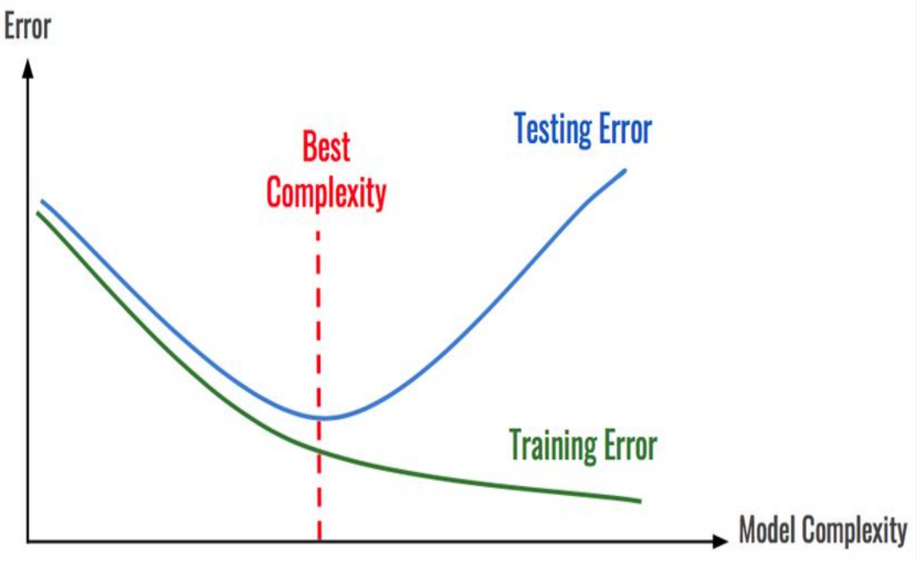
Bias vs. Variance contd.





Note: Important for interviews

- Data has two components components: *signal (pattern)* + *noise*
- **Goal of machine learning**: learn the signal, ignore the noise
- When the model is learning the noise, it is **overfitting**
- *Avoid Overfitting*
    - *Reduce features: [FS/DRTs]()*
    - *Reduce parameter space of fit parameters: Regularization*



Simple is better than complex, Complex is better than complicated

Overfitting vs. Underfitting

Overfit models:

a. Low performance error when used on training data
b. High performance error when used on test/validation/unseen data
c. Unpredictable performance (*)
d. More complex, have many (hyper)parameters

Under fit models

a. High performance error when used on training data: High Bias
b. High performance error when used test/validation/unseen data
c. Consistently bad performance
d. Simple, less parameters

Why we care about Bias/Generalizability ?

Taming ML models:

Feature Selection (See, W06D1)
- Filter methods
- Dimensional reduction

Regularization: Technique used to avoid overfitting by shrinking feature coefficients
- L1: Lasso Regressions
  - Shrinks coefficients of less important features to 0, and removes them
  - Generates simpler models
- L2: Ridge Regressions
  - Shrinks, but does not completely remove
  - Performs for models which depends on all features

Loss functions & Optimizations

Summary

Overfitting vs. Underfitting

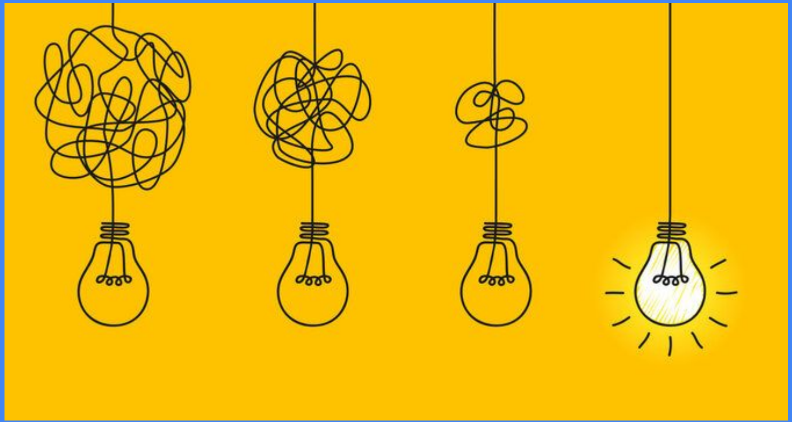
Simple vs. Complex

What is Bias & Variance ?

What is error, and its relations to Bias & Variance ?

Why we need regularization ?

How to choose an optimal fit ?



Part II

Regularization

Loss/Cost functions: Introductions

A loss/cost function is a function that quantifies how far away the predicted values are from the actual values.

- **Least/Mean squared error/L2 Loss:** MSE is used for regression problems. It quantifies the difference between predicted and actual values by taking the squared difference and then averaging over all examples.
- **MAE/L1 Loss**: Absolute of the difference over all samples.
- **Hinge loss:** Classification problems e.g. support vector machines (SVMs). [convex]
- **Logloss:** Classification problems e.g. logistic regression. [convex]
- **Huber Loss:** Combination of L1/L2 loss

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$MAE = \frac{1}{2}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

Regularization aka antidote to overfitting

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

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

Where $\lambda \geq 0$

L1/Lasso (least absolute shrinkage/selection operator)

- 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)

$$\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = V(\mathbf{X}, \mathbf{y}, \mathbf{w}) \quad + \quad \lambda \quad |\mathbf{w}|$$

$$= V(\mathbf{X}, \mathbf{y}, \mathbf{w}) \quad + \quad \lambda \quad \sum_{i=0}^{n} |w_i|$$

L2/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

$$\mathcal{L}(\mathbf{X},\mathbf{y},\mathbf{w}) = V(\mathbf{X},\mathbf{y},\mathbf{w}) \quad + \quad \lambda \quad \mathbf{w}^T\mathbf{w}$$

$$= V(\mathbf{X},\mathbf{y},\mathbf{w}) \quad + \quad \lambda \quad \sum_{i=0}^{n} w_i^2$$

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

See, Elastic net regularization - Wikipedia

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 when predicting unseen data (test set)

Summary

Anti-overfitting is regularization

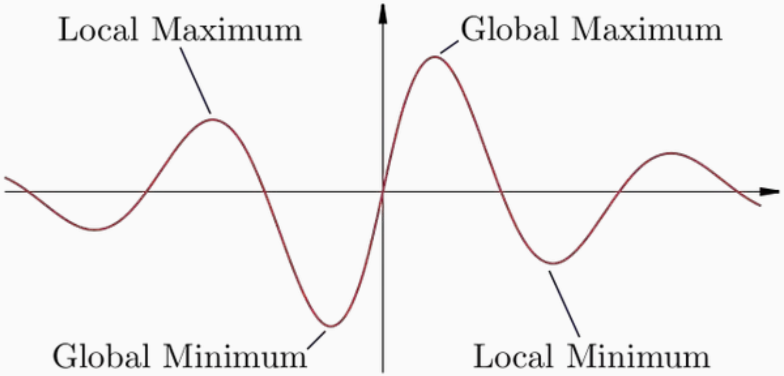
L1- can be used for feature selection.

L2- More useful, when all features are valuable.

Regularization penalizes the feature /parameter space disallowing functional search space, creating simpler models

Alternatives, can be data augmentation.

Questions ?



Part III

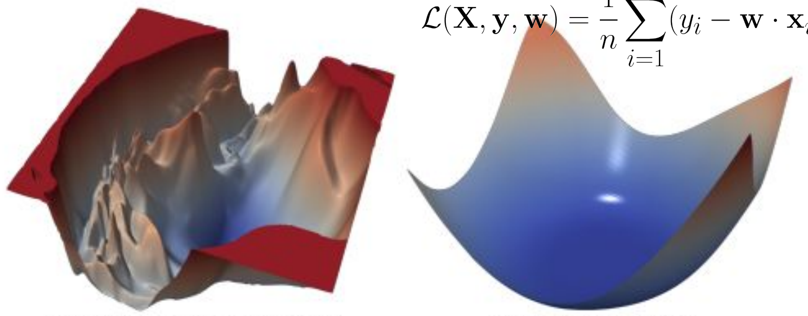
Optimizing Loss functions

Loss functions | Contd.

Properties of Loss functions

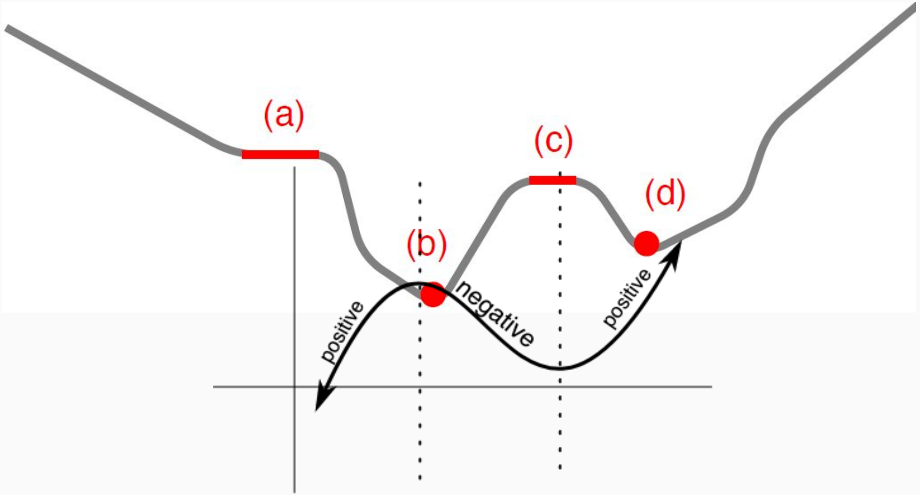
- Differentiability- MAE e.g. is non-differentiable. Differentiability is important because it allows the loss function to be minimized using gradient descent.
- Smoothness- A loss function is smooth if it changes gradually as the predicted values change.
- Convexity- Convex function guarantee 1 local minima. Easier to optimize. Loss landscapes of most complex models e..g NN’s are non-convex i.e. have >1 minima *harder to optimize*.

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

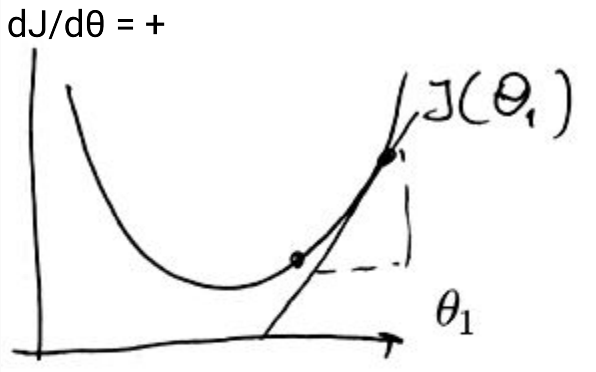


(a) ResNet-110, no skip connections (b) DenseNet, 121 layers

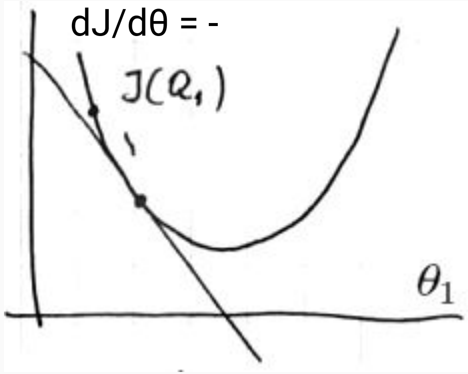
Figure 4: The loss surfaces of ResNet-110-noshort and DenseNet for CIFAR-10.



Gradient Descent



$$\theta_1' = \theta_1 - \beta$$



$$\theta_1' = \theta_1 + \beta$$

Intuition

Let's see how it works

- assuming that there's only one variable θ_1 and $\theta_1 \in \mathbb{R}$

$$\theta_1 = \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

let's have a look at the partial derivative:

- $\beta = \alpha \frac{d}{d\theta_1} J(\theta_1)$

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

Learning Rate

- when α is too small - we're taking very small steps - too slow
- when α is too large - we're taking too big steps and may miss the minimum

  in this case not only may it fail to converge, but even diverge!

Approaching the minimum

- As we're approaching the local minimum, it takes smaller and smaller steps
- If θ_1 is at the local minimum, then $\beta = 0$ and θ_1 won't change

Gradient Descent

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**

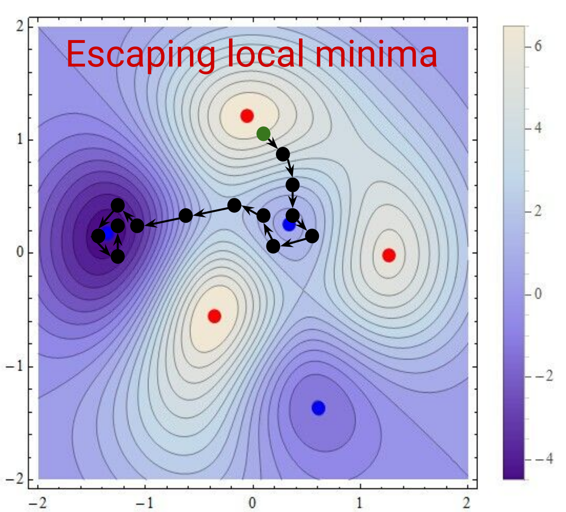
    The parameters are now better because they result in a lower loss. This is the goal of training

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

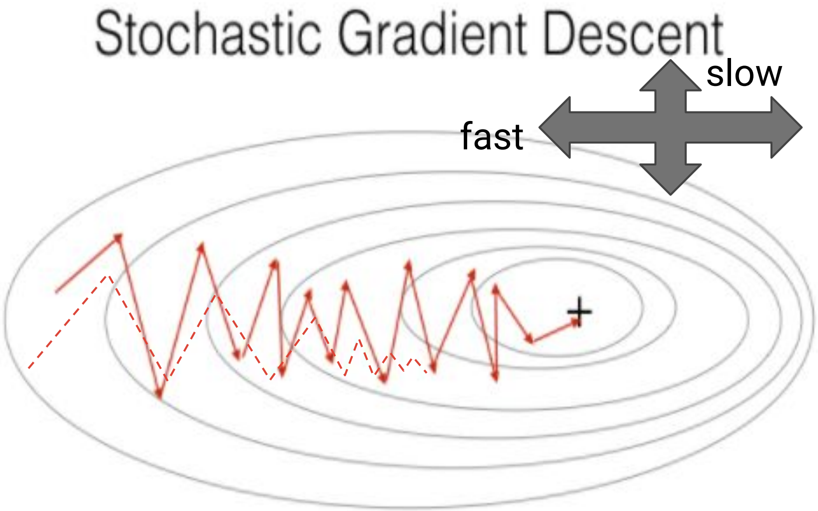
- **Learning rate:** Movement in gradient descent direction/update
    - Optimal setting is f(model parameters, loss function)
    - Empirically set i.e. trial & error
    - Can be annealed (changed online)
- **Epochs:** No. of times we update the weights
    - Constant value
    - Can keep updating until loss stops decreasing (i.e. loss has **converged**)

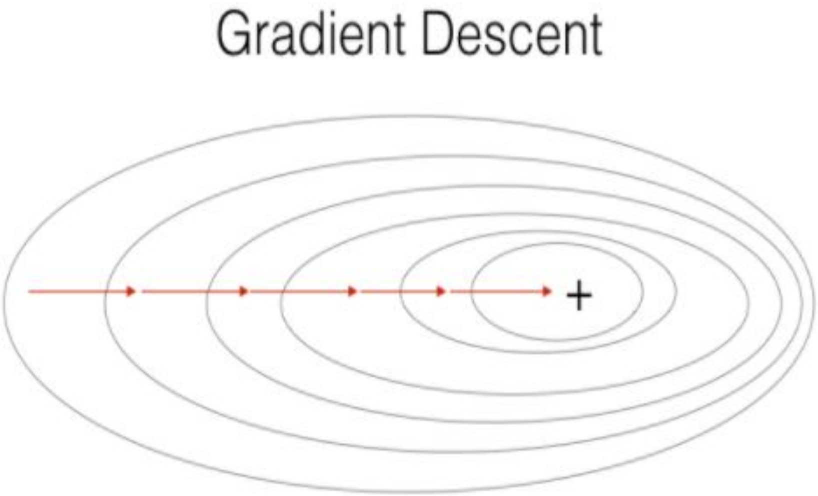
Stochastic gradient descent

- Issues with gradient descent:
  - Computationally expensive and slow (uses the entire **X** dataset for each weight update)
  - Can get stuck in bad local minima (never goes uphill)
- Solution: 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)



SGD vs. GD





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

Algorithm: Gradient Descent

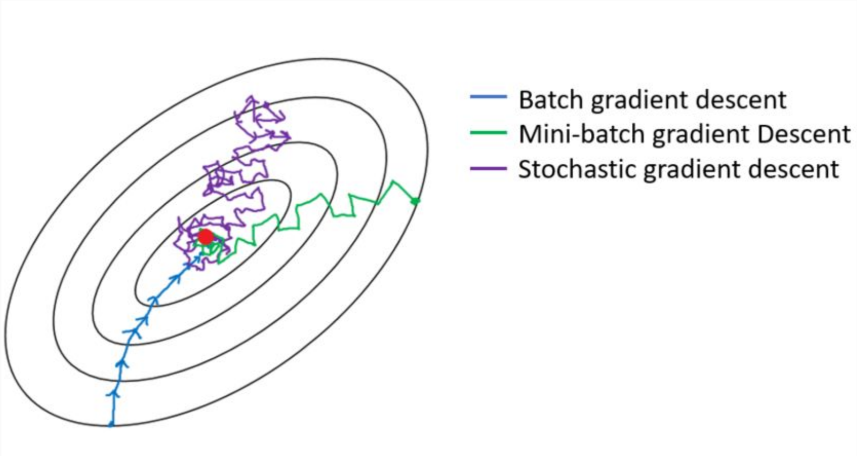
Initialize $\mathbf{w}$ (e.g. randomly)
for $epoch \in nEpochs$ **do**
  $\mathbf{w}_{grad} = \frac{\partial}{\partial \mathbf{w}} \mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w})$
  $\mathbf{w} = \mathbf{w} - \alpha \mathbf{w}_{grad}$
end

α denotes the Learning Rate

Mini-batch Gradient Descent

- Noise (i.e. randomness):
    - Small noise help find optimal solution by escaping local minima
    - Large noise slows training /sustained downhill progress
- Computational efficiency:
    - Big matrix multiplications - OOM errors/memory overhead
    - Small matrix multiplications - Time complexity bad

We often want something in between: estimate the gradient each iteration using *some, but not all* of the data



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

Summary

Convexity

Learning rate

Momentum

Epochs

Convergence

Questions ?

Demo (1)

Questions ?

Resources

Regularization Part 1: Ridge (L2) Regression - YouTube

Regularization Part 2: Lasso (L1) Regression - YouTube

Stochastic Gradient Descent, Clearly Explained!!! - YouTube

Regularization. What, Why, When, and How? | by Akash Shastri | Towards Data Science

Loss Functions. Loss functions explanations and… | by Tomer Kordonsky | Artificialis | Medium

REGULARIZATION: An important concept in Machine Learning | by Megha Mishra | Towards Data Science

Least squares - Wikipedia

Regularized least squares - Wikipedia

Gradient Descent: 3Blue1Brown

GD with momentum

Linear reg. With GD

Thanks!