Neural Networks

Kaiqi Zhao
The University of Auckland

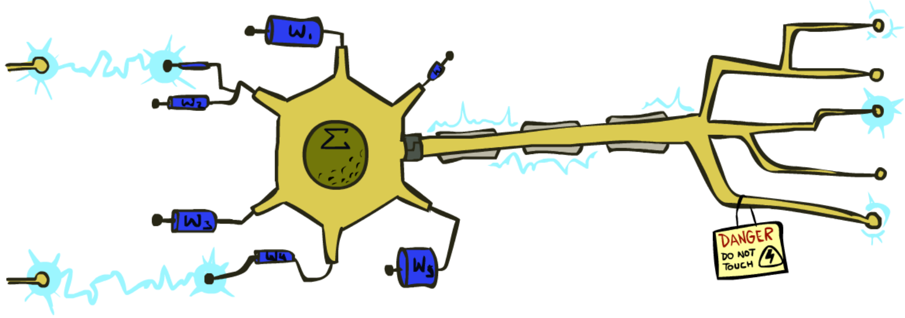
Slides are partially based on the materials from Stanford University and UC Berkeley

Content

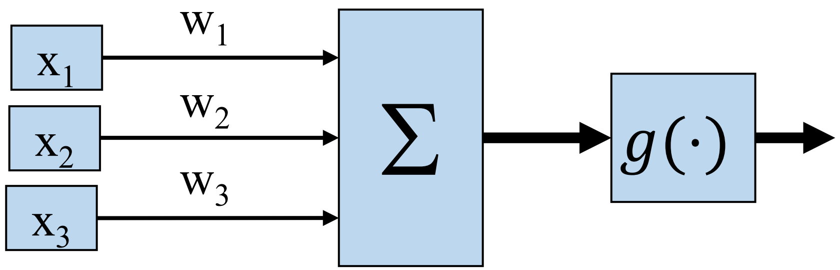
- Logistic Regression
- Artificial Neural Networks
- Non-linearity and Activation Functions
- Regularization
- Optimization

Motivation from Neuroscience

- Biological Neural Network
  - a group of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive



- An Artificial Neuron is composed of:
  - Input feature values
  - Weights for each input feature
  - Activation function $g(\cdot)$



$$\hat{y} = g(w^T x) = g\left(\sum_i w_i x_i\right)$$

Recap: Linear Regression

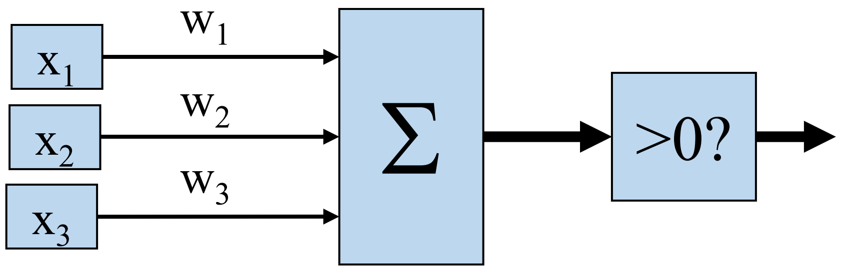
- Inputs are feature values
- Each feature has a weight
- The output is a real-value number!

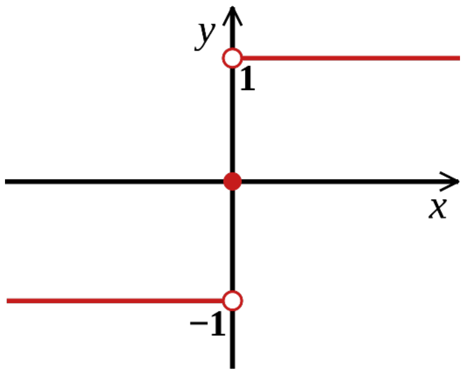
$$\hat{y} = w^T x = \sum_i w_i x_i$$

- Can we modify this and make it work for a binary classification problem?
  - How if we place a sign function (activation function) to make the prediction?

$$\hat{y} = Sign(w^T x)$$

- The output label would be
  - Positive (+1), if $w^T x > 0$
  - Negative (-1), otherwise





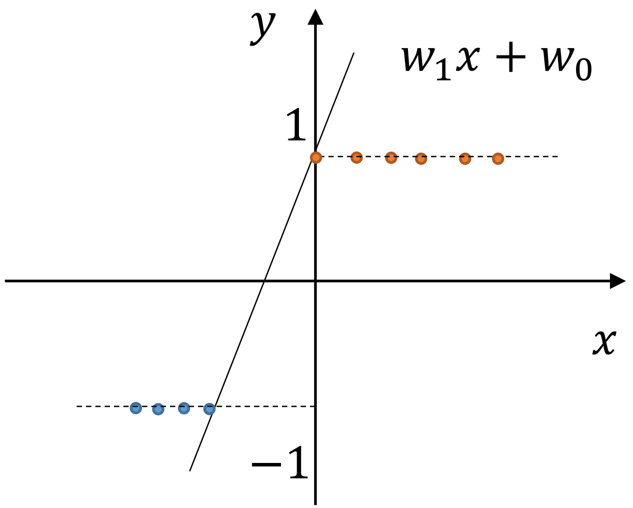
$Sign(x)$

A Simple Linear Classification Model

- Inputs are feature values
- Each feature has a weight
- A sign activation function applies to the weighted sum of the features, which outputs discrete class values

$$\hat{y} = Sign(w^T x) = Sign\left(\sum_i w_i x_i\right)$$

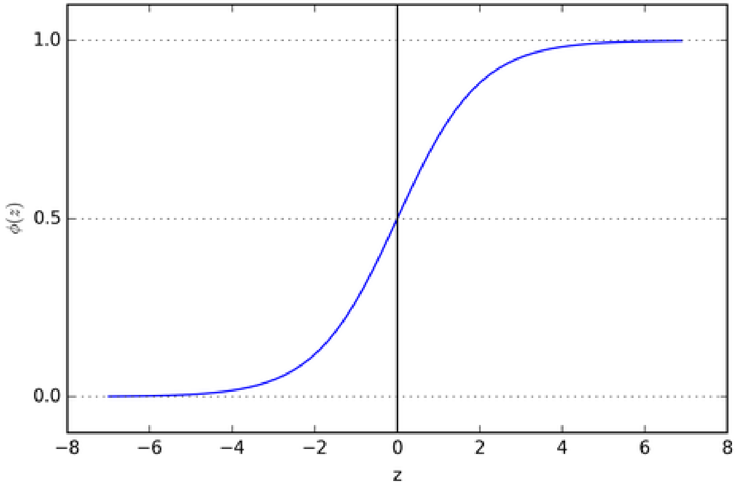
- This is an artificial neuron!



Sigmoid Function

- The sign function $sign(x)$ is not differentiable at $x = 0$
- Do we have any walkaround?
- Consider the problem as a probabilistic classification problem – we output a probability value of each class label provided $w^T x$ instead of the exact class labels
- We want
  - If $z = w^T x$ is very positive → probability goes to 1
  - If $z = w^T x$ is very negative → probability goes to 0
- Use the Sigmoid (logistic) function on z

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Logistic Regression

- Given that the sigmoid function outputs value within [0, 1], it could be considered as the probability of positive label for a binary classification problem.
- We can use maximum likelihood estimation – find the parameters w which maximizes the probability of all training examples being classified correctly:

$$w_{ML} = \underset{w}{\operatorname{argmax}}\, log\mathcal{L}(w) = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with

$$P\left(y^{(i)} = +1 \middle| x^{(i)}; w\right) = \frac{1}{1 + e^{-w^T x^{(i)}}}$$

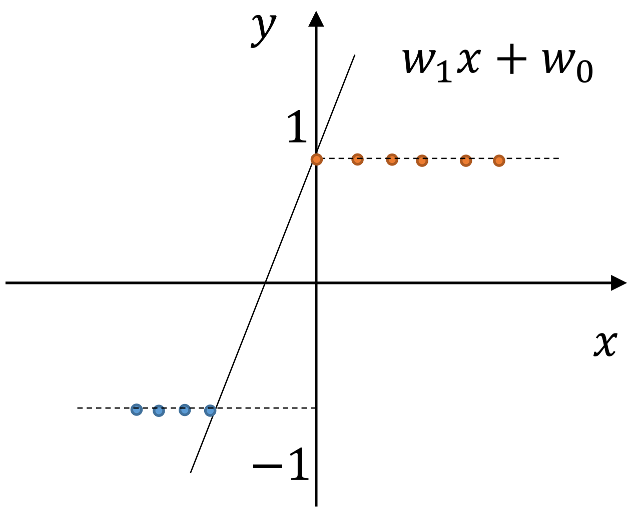
$$P\left(y^{(i)} = -1 \middle| x^{(i)}; w\right) = 1 - \frac{1}{1 + e^{-w^T x^{(i)}}} = \frac{e^{-w^T x^{(i)}}}{1 + e^{-w^T x^{(i)}}}$$

= Logistic Regression

Example – Prediction

- Let's consider the following data set of 1 feature. Let the weights $w_0 = 1, w_1 = 2$ be the best you get from maximum likelihood

i	x	y
1	-2	-1
2	0	+1
…	…	…



- Make prediction by calculating the probability:

$$P\left(y^{(1)} = +1 \middle| x^{(1)}; w\right) = \frac{1}{1 + e^{-(-2*2+1)}} = \frac{1}{1 + e^{3}} \approx 4.7\%$$

$$P\left(y^{(1)} = -1 \middle| x^{(1)}; w\right) = 1 - P\left(y^{(1)} = +1 \middle| x^{(1)}; w\right) \approx 95.3\%$$

$$P\left(y^{(2)} = +1 \middle| x^{(2)}; w\right) = \frac{1}{1 + e^{-1}} \approx 73.1\%$$

$$P\left(y^{(2)} = -1 \middle| x^{(2)}; w\right) = 1 - P\left(y^{(2)} = +1 \middle| x^{(2)}; w\right) \approx 26.9\%$$

$$P\left(y^{(i)} = +1 \middle| x^{(i)}; w\right) = \frac{1}{1 + e^{-w^T x^{(i)}}}$$

Logistic Regression

- The previous definition of Logistic Regression treats the two classes differently. In fact, you could also define it with its equivalent form:

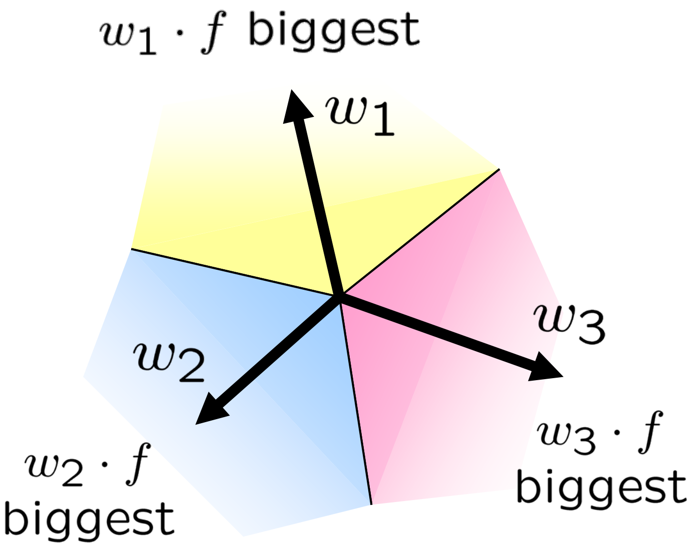
$$P\left(y^{(i)} = +1 \middle| x^{(i)}; w_1, w_2\right) = \frac{e^{w_1^T x^{(i)}}}{e^{w_1^T x^{(i)}} + e^{w_2^T x^{(i)}}}$$

$$P\left(y^{(i)} = -1 \middle| x^{(i)}; w_1, w_2\right) = \frac{e^{w_2^T x^{(i)}}}{e^{w_1^T x^{(i)}} + e^{w_2^T x^{(i)}}}$$

- In this way, the two classes are treated in the same way
  - We have one weight vector for each class (w_1 for class +1, and w_2 for class -1)
  - The conditional probability of each class only depend on the feature $x^{(i)}$ and its weight, the denominator is only a normalization term.
  - With this form, it is easier for us to extend Logistic Regression to multi-class problems.

Multiclass Logistic Regression

- Multi-class linear classification
  - A weight vector for each class: w_c
  - Score for a class y: $w_c^T x$
  - Prediction with the highest score: $y = \underset{c \in C}{\operatorname{argmax}} \, w_c^T x$
- Suppose we have class labels $\{1, 2, 3\}$
- How to make the scores into probabilities?
  - Extend the binary logistic regression
  - Let $z_1 = w_1^T x$, $z_2 = w_2^T x$, $z_3 = w_3^T x$



$$\underbrace{z_1, z_2, z_3}_{\text{original outputs}} \rightarrow \underbrace{\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}, \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}}_{\text{Softmax transformation}}$$

Multiclass Logistic Regression

- Maximum likelihood estimation:

$$w_{ML} = \underset{w}{\operatorname{argmax}}\, log\mathcal{L}(w) = \underset{w}{\operatorname{argmax}} \sum_{i} \log P(y^{(i)} | x^{(i)}; w)$$

with

$$P\left(y^{(i)} = c \middle| x^{(i)}; w\right) = \frac{e^{w_c x^{(i)}}}{\sum_{c'} e^{w_{c'} x^{(i)}}}$$

= Multi-Class Logistic Regression

Finding the Weights of Logistic Regression

- Optimization
  - i.e., how do we solve:

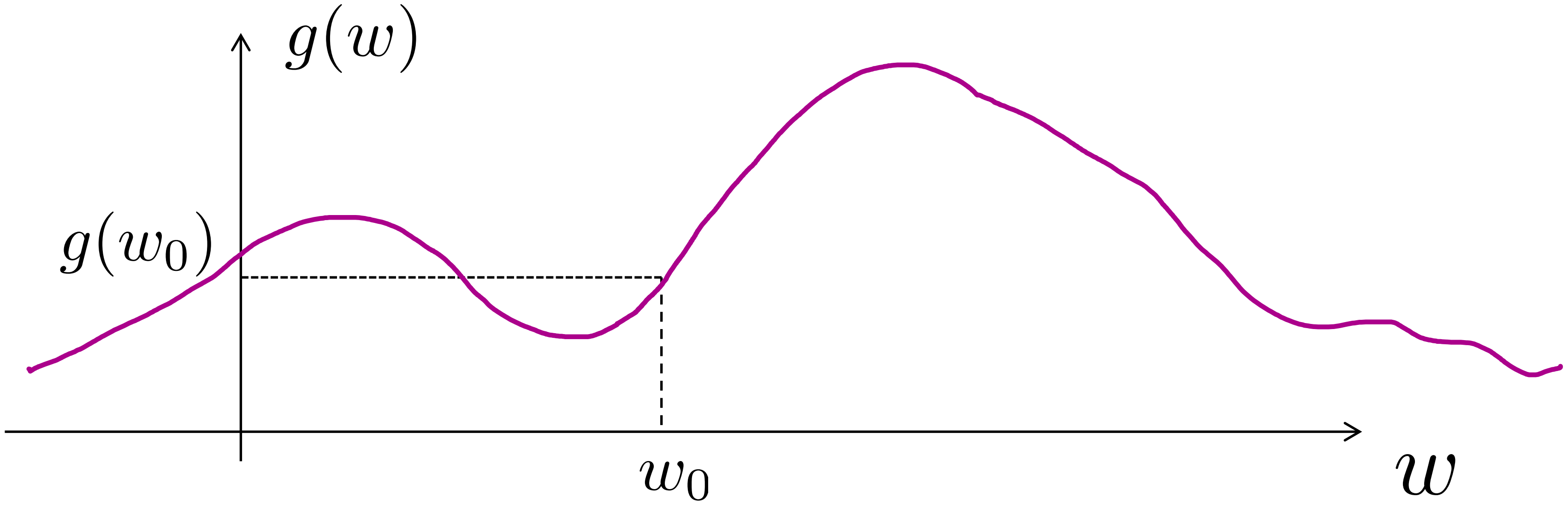
$$w_{ML} = \underset{w}{\operatorname{argmax}}\, log\mathcal{L}(w) = \underset{w}{\operatorname{argmax}} \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

Hill Climbing

- General idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What's particularly tricky when hill-climbing for multi-class logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?
    - Select the direction which is steepest to move!

Finding the Steepest Direction | 1-D Optimization



- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction

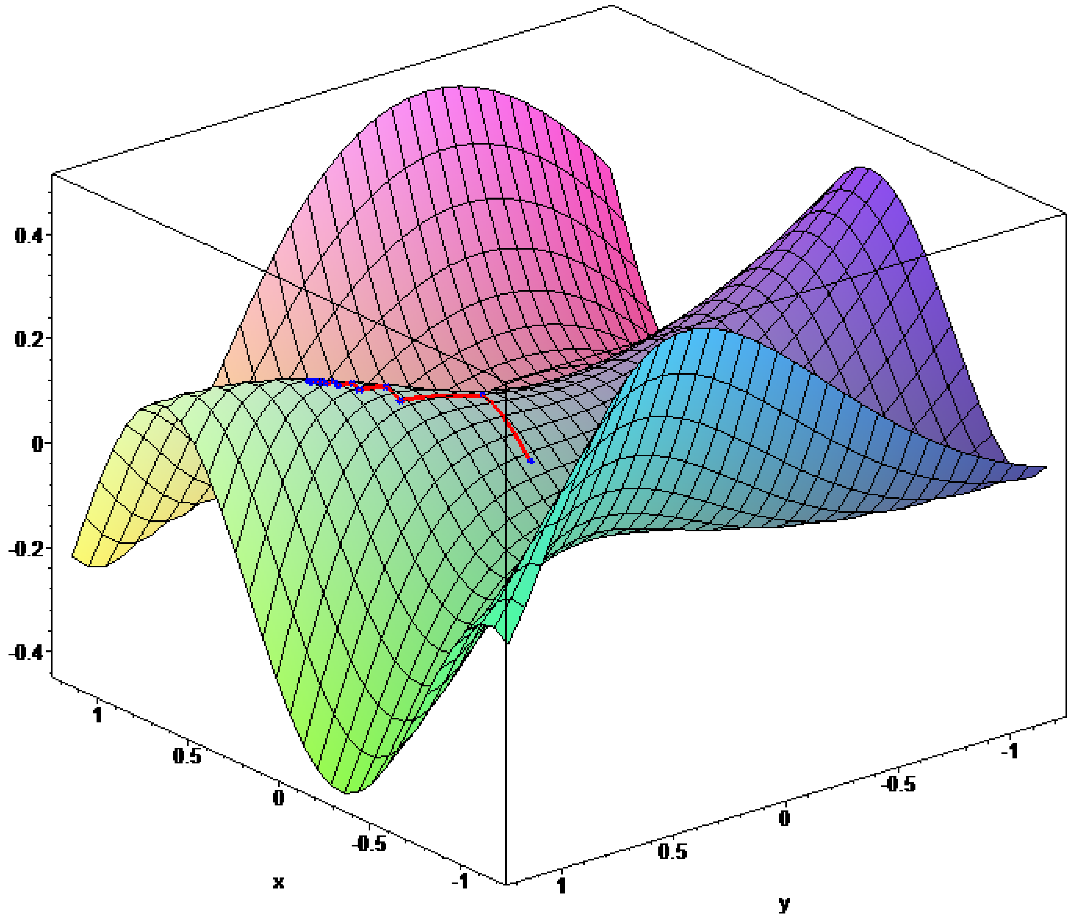
- Or, evaluate derivative: $$\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$

Finding the Steepest Direction | 2-D Optimization

Gradient provides the steepest direction in a 2D or higher dimensional optimization problem.

Let the two parameters to be estimated be w_1 and w_2, and $g(w_1, w_2)$ be the function you want to maximize/minimize. Then, the gradient at $w_1 = a, w_2 = \text{b}$ is

$$\nabla g(a,b) = \begin{bmatrix} \dfrac{\partial g}{\partial w_1}(a,b) \\ \dfrac{\partial g}{\partial w_2}(a,b) \end{bmatrix}$$



Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$, let $w_1 = a$ and $w_2 = b$ currently

Updates:

$$w_1 \leftarrow a + \alpha * \frac{\partial g}{\partial w_1}(a, b)$$

$$w_2 \leftarrow b + \alpha * \frac{\partial g}{\partial w_2}(a, b)$$

α is learning rate that controls how fast we step

Updates: in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(a, b)$$

with $\nabla_w g(a, b) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(a, b) \\ \frac{\partial g}{\partial w_2}(a, b) \end{bmatrix}$ = gradient

Gradient Ascent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction

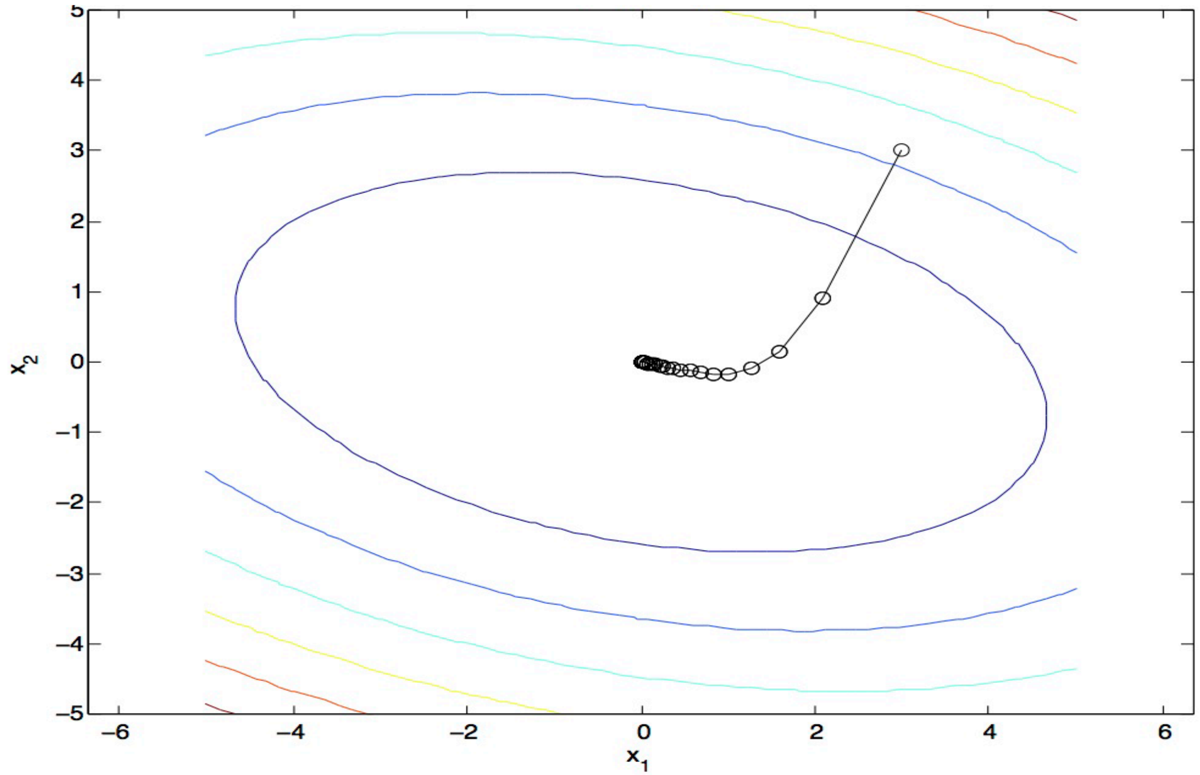


Figure source: Mathworks

Optimization Procedure: Gradient Ascent

- init $w = w_0$
- for iteration $t = 1, 2, \ldots$

$$w_t \leftarrow w_{t-1} + \alpha * \nabla g(w_{t-1})$$

- α : learning rate – the parameter that controls how fast we climb the hill
- A small learning rate – slow learning
- A large learning rate – might miss the peak!

Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w log\mathcal{L}(w) = \max_w \underbrace{\sum_i \log P(y^{(i)}|x^{(i)}; w)}_{g(w)}$$

- init $w = w_0$
- for iteration $t = 1, 2, \ldots$

$$w_t \leftarrow w_{t-1} + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w_{t-1})$$

Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_w log\mathcal{L}(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

Observation: once gradient on one training example has been computed, might as well incorporate before computing next one

- init $w = w_0$
- for iteration $t = 1, 2, \ldots$
  - pick random training example *j*

    $$w_t \leftarrow w_{t-1} + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w_{t-1})$$

Stochastic Gradient Ascent on the Log Likelihood Objective

$$\max_{w} log\mathcal{L}(w) = \max_{w} \sum_{i} \log P(y^{(i)}|x^{(i)}; w)$$

Observation: gradient over small sets of training examples (= mini-batch) can be computed in parallel, might as well do that instead of a single one

- init $w = w_0$
- for iteration $t = 1, 2, \ldots$
  - pick random **subset** of training examples J

$$w_t \leftarrow w_{t-1} + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w_{t-1})$$

Stochastic Gradient Ascent on the Log Likelihood Objective

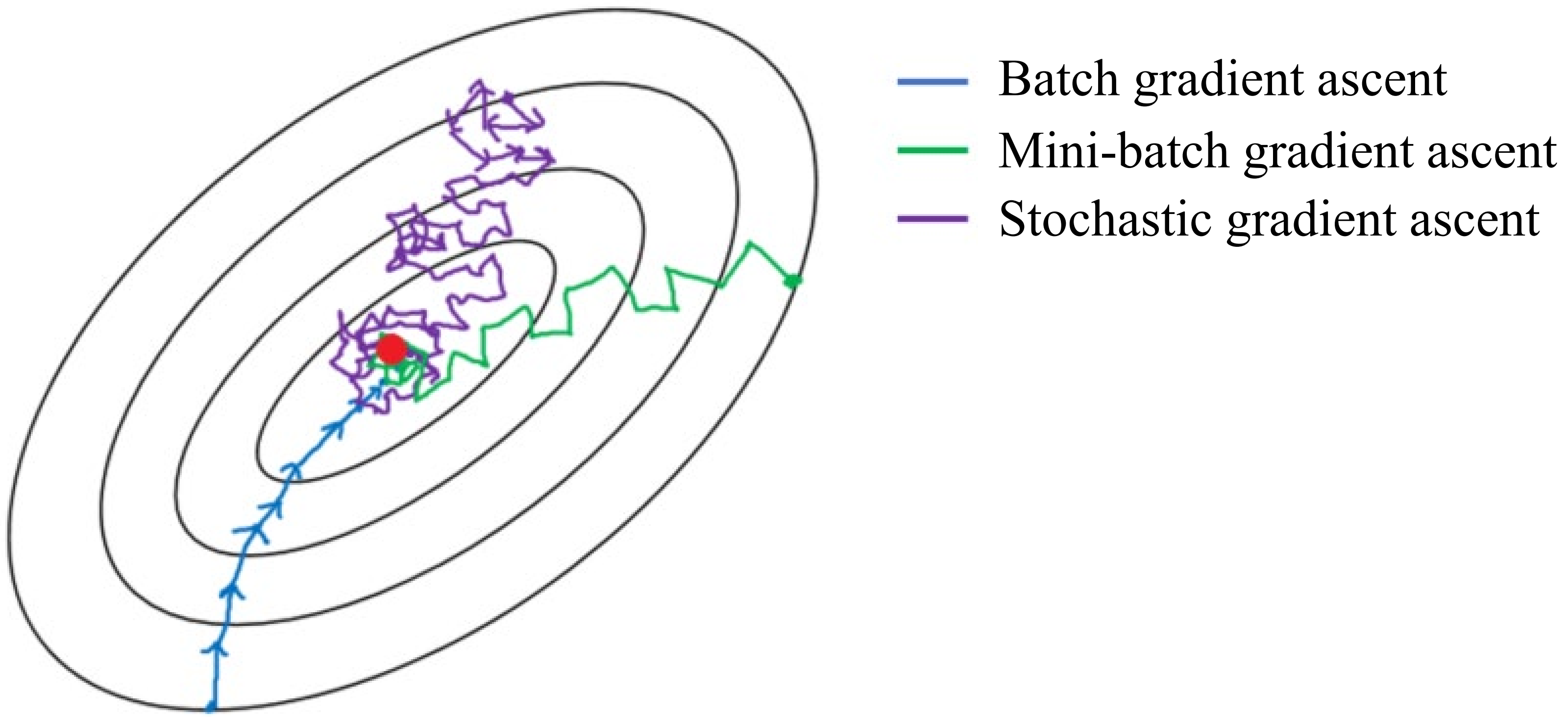


Image adapted from: https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461

Cross-Entropy Loss

- In maximum likelihood estimation, we optimize

$$w_{ML} = \underset{w}{\operatorname{argmax}}\, log\mathcal{L}(w) = \underset{w}{\operatorname{argmax}} \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

- Let $\left[y_1^{(i)}, y_2^{(i)}, \ldots, y_C^{(i)}\right]^T$ be a one-hot vector, where $y_k^{(i)} = 1$ if the class label of the i-th instance is k, i.e., $y^{(i)} = k$, otherwise, $y_k^{(i)} = 0$

- We can rewrite the likelihood: $\mathcal{L}(w) = \prod_i \prod_k P\left(k | x^{(i)}; w\right)^{y_k^{(i)}}$

- Thus, maximizing the log-likelihood is equivalent to minimize the loss (Cross-Entropy)

$$CE(w) = -log\mathcal{L}(w) = -\sum_i \sum_k y_k^{(i)} \log P\left(k \middle| x^{(i)}; w\right)$$

Example

- We have two different weight vectors (w_1 and w_2) results in the following predictions of weather (sunny, rain, windy):

i	Predictions with w_1 (S, R, W)	Predictions with w_2 (S, R, W)	$\boldsymbol{y}$ (S, R, W)
1	0.7, 0.2, 0.1	0.4, 0.3, 0.3	1, 0, 0
2	0.1, 0.1, 0.8	0.2, 0.1, 0.7	0, 0, 1
3	0.3, 0.3, 0.4	0.2, 0.2, 0.6	0, 1, 0

$$CE(w) = -\sum_i \sum_k y_k^{(i)} \log P\left(k \middle| x^{(i)}; w\right)$$

Weight vector w_1

Sample 1 loss:

$$-1 * \log 0.7 - 0 * \log 0.2 - 0 * \log 0.1 \approx 0.357$$

Sample 2 loss:

$$-0 * \log 0.1 - 0 * \log 0.1 - 1 * \log 0.8 \approx 0.223$$

Sample 3 loss:

$$-0 * \log 0.3 - 1 * \log 0.3 - 0 * \log 0.4 \approx 1.201$$

$CE(w_1) = 0.357 + 0.223 + 1.201 = 1.781$

Weight vector w_2

Sample 1 loss: $-1 * \log 0.4 \approx 0.916$

Sample 2 loss: $-1 * \log 0.7 \approx 0.357$

Sample 3 loss: $-1 * \log 0.2 \approx 1.609$

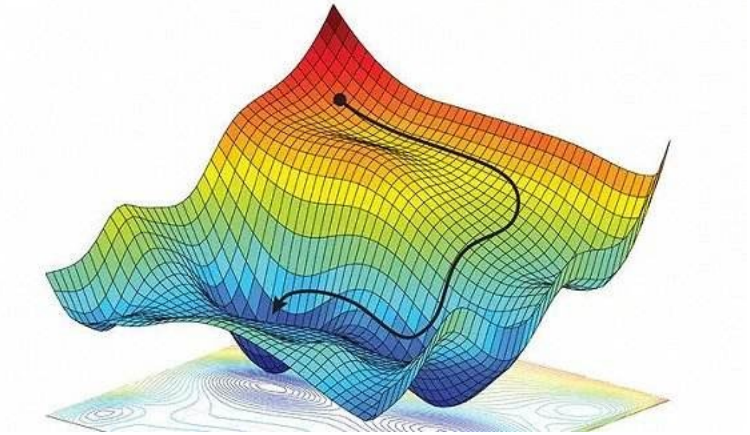
$CE(w_2) = 0.916 + 0.357 + 1.609 = 2.882$

Gradient Descent

$$\underset{w}{\operatorname{argmin}}\, CE(w) = \underset{w}{\operatorname{argmin}} - \sum_i \sum_k y_k^{(i)} \log P\left(\left(k \middle| x^{(i)}; w\right)\right)$$

- To minimize cross-entropy, we can use gradient descent – going downhill
- Gradient descent goes towards the opposite direction of gradient ascent

$$w \leftarrow w - \alpha * \nabla CE(w)$$



- Variants:
  - Stochastic Gradient Descent (SGD)
  - Mini-batch SGD