Neural Computation

Neural Computation (extended)

Week 2:
Gradient Descent Methods and Linear Classification

Kashif Rajpoot

Outline

1. Gradient descent (GD)
2. Stochastic gradient descent (SGD)
3. Minibatch SGD
4. Linear classification

Gradient descent (GD)

Outline: GD

1. Introduction
2. Optimisation
3. Minimisation
4. Choosing a step size
5. GD for least squares regression

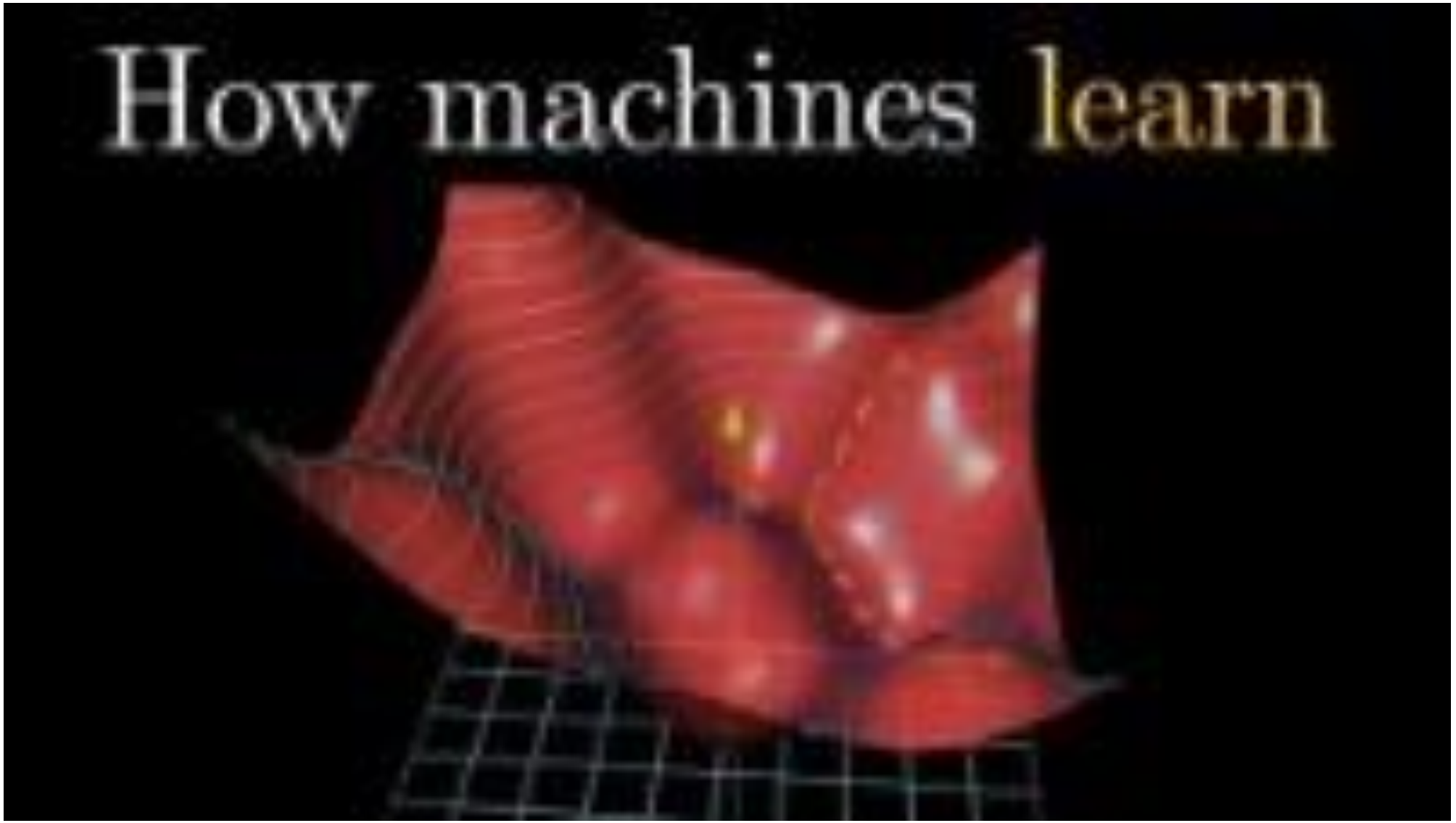
Gradient descent: introduction

Gradient descent: optimisation

- Optimisation algorithm
  1. Start with a point $\boldsymbol{w}$ (initial guess)
  2. Find a direction $\boldsymbol{d}$ to move on
  3. Determine how far (η) to move along $\boldsymbol{d}$
  4. Update: $\boldsymbol{w} = \boldsymbol{w} + \eta \boldsymbol{d}$

Minimizing the cost is like finding the lowest point in a hilly landscape

Gradient descent: introduction



https://www.youtube.com/watch?v=IHZwWFHWa-w&t=325s

Gradient descent: introduction

https://www.youtube.com/watch?v=qg4PchTECck

Gradient descent: introduction

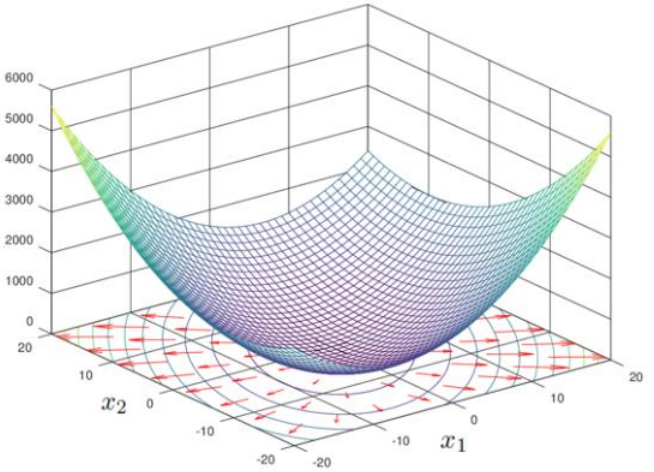
- The gradient vector at a point $\boldsymbol{x}$ points in the direction of greatest increase of the function f: each element of the gradient shows how fast $f(\boldsymbol{x})$ is changing w.r.t. each of the coordinate axes

- Example: consider

$$f(\boldsymbol{x}) = f(x_1, x_2) = 6x_1^2 + 4x_2^2 - 4x_1x_2$$

- The gradient vector is

$$\nabla f(x_1, x_2) = \begin{pmatrix} 12x_1 - 4x_2 \\ 8x_2 - 4x_1 \end{pmatrix}$$



f and its gradient vector field

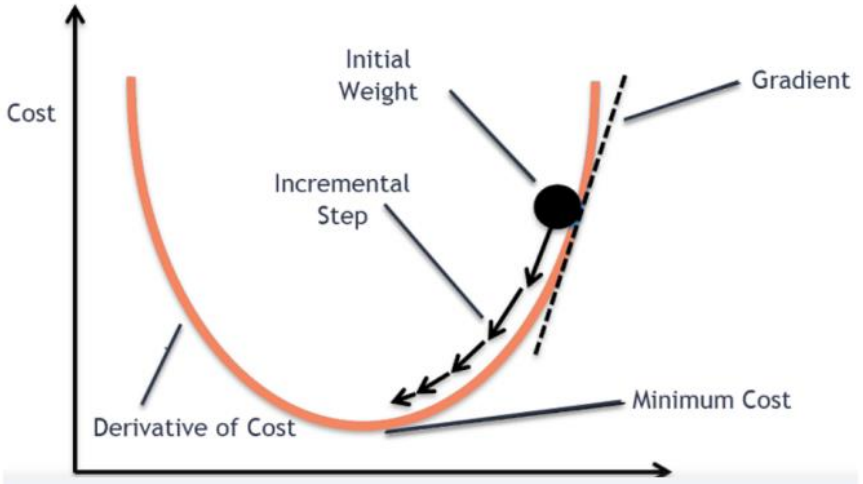
Gradient descent: minimisation

- Gradient descent is one of the simplest, but very general algorithm for minimising an objective function $C(\boldsymbol{w})$ (first proposed by Cauchy in 1847)
- It is an iterative algorithm, starting from $\boldsymbol{w}^{(0)}$ and producing a new $\boldsymbol{w}^{(t+1)}$ at each iteration as:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \nabla C\left(\boldsymbol{w}^{(t)}\right)$$

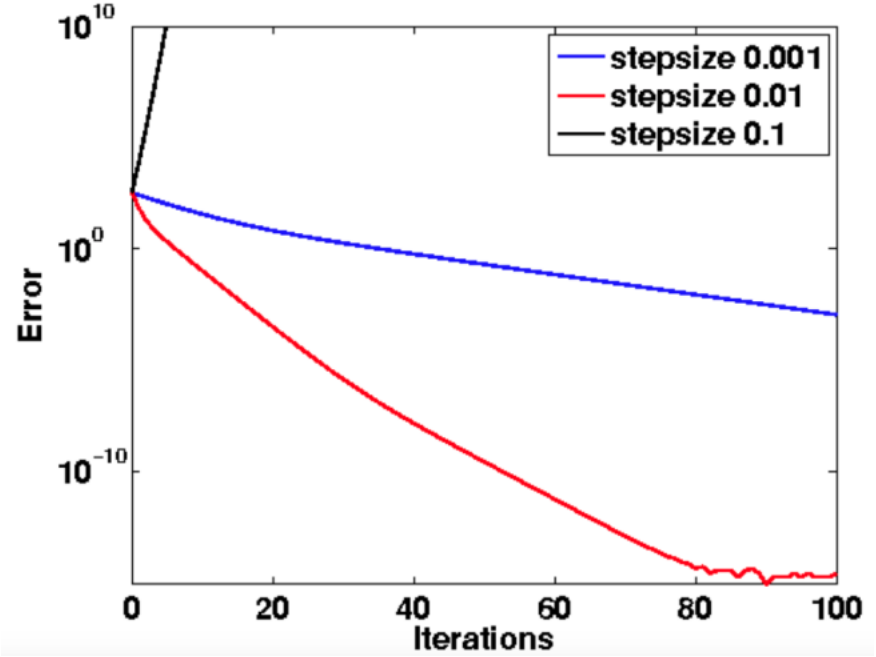
where $t = 0,1, \dots, T$

- $\eta_t > 0$ is the learning rate or step size



Gradient descent: choosing a step size

- Choosing a good step-size is important
- If step size is too large, algorithm may never converge
- If step size is too small, convergence may be very slow
- May want a time-varying step size



Gradient descent: least squares regression

- For least square regression, let's recall:

$$C(\boldsymbol{w}) = \frac{1}{2n}\left(\underbrace{\boldsymbol{w}^T X^T X \boldsymbol{w}}_{\text{quadratic}} - \underbrace{2\boldsymbol{w}^T X^T \boldsymbol{y}}_{\text{linear}} + \underbrace{\boldsymbol{y}^T \boldsymbol{y}}_{\text{constant}}\right)$$

- The gradient is computed as:

$$\nabla C(\boldsymbol{w}) = \frac{1}{n}(X^T X \boldsymbol{w} - X^T \boldsymbol{y})$$

- GD updates $\boldsymbol{w}^{(t)}$ by

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla C\left(\boldsymbol{w}^{(t)}\right)$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \frac{\eta}{n}\left(X^T X \boldsymbol{w}^{(t)} - X^T \boldsymbol{y}\right)$$

Simple to Implement!

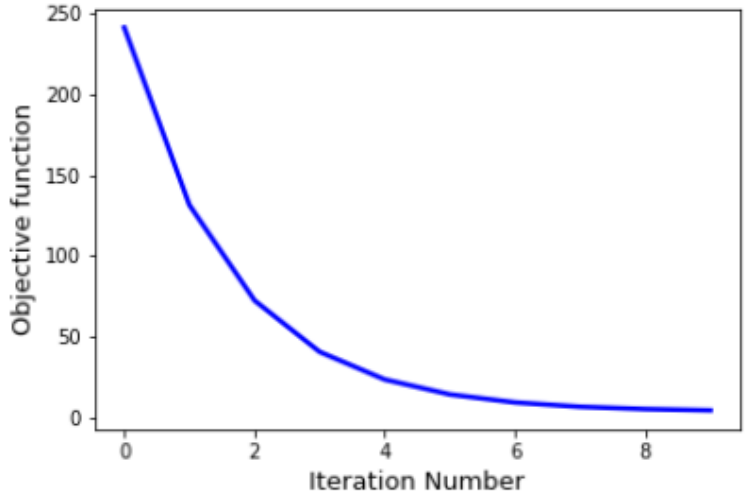
Gradient descent: toy example (regression)

- Let's recall our bus commute example where $n = 5$ and

$$\boldsymbol{y} = \begin{pmatrix} 25 \\ 33 \\ 15 \\ 45 \\ 22 \end{pmatrix}, X = \begin{pmatrix} 1 & 2.7 & 1 \\ 1 & 4.1 & 1 \\ 1 & 1.0 & 0 \\ 1 & 5.2 & 1 \\ 1 & 2.8 & 0 \end{pmatrix}, \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

- If we run GD with $\boldsymbol{w}^{(0)} = (0 \quad 0 \quad 0)^T$ and $\eta = 0.02$, then we get (with Python implementation)

```
the grad of 1-th iteration is [ -28.   -102.68  -20.6 ]
the weight of 1-th iteration is [0.56   2.0536 0.412 ]
the grad of 2-th iteration is [-20.703424  -75.2866144 -15.08816  ]
the weight of 2-th iteration is [0.97406848 3.55933229 0.7137632 ]
the grad of 3-th iteration is [-15.35018357 -55.1911618  -11.0449035 ]
the weight of 3-th iteration is [1.28107215 4.66315552 0.93466127]
the grad of 4-th iteration is [-11.42255963 -40.44941129  -8.07898669]
the weight of 4-th iteration is [1.50952334 5.47214375 1.096241  ]
the grad of 5-th iteration is [ -8.5407578  -29.6350914   -5.90339639]
the weight of 5-th iteration is [1.6803385  6.06484558 1.21430893]
the grad of 6-th iteration is [ -6.42616412 -21.70190135  -4.30758215]
the weight of 6-th iteration is [1.80886178 6.4988836  1.30046057]
the grad of 7-th iteration is [ -4.87438968 -15.88228366  -3.13708593]
the weight of 7-th iteration is [1.90634958 6.81652928 1.36320229]
the grad of 8-th iteration is [ -3.73549653 -11.61316462  -2.27859861]
the weight of 8-th iteration is [1.98105951 7.04879257 1.40877427]
the grad of 9-th iteration is [-2.89949141 -8.48147805 -1.64899757]
the weight of 9-th iteration is [2.03904933 7.21842213 1.44175422]
the grad of 10-th iteration is [-2.2856842  -6.18420209 -1.18730475]
the weight of 10-th iteration is [2.08476302 7.34210617 1.46550031]
```



Summary

- Optimisation and minimisation
- Step size choice
- Least squares regression via gradient descent

Stochastic gradient descent (SGD)

Outline: SGD

1. Introduction
2. Algorithm
3. SGD vs GD
4. Effect of learning rates

SGD: introduction

- GD is easy to implement since gradient computation is required
- GD is computationally expensive as it requires to go through all the examples
- Sum structure
  - $C(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n} C_i(\boldsymbol{w})$, $C_i(\boldsymbol{w})$ corresponds to the loss for i^{th} example

- Can we boost the optimisation by using the sum structure of C?

SGD: introduction

- Yes, and replace the computationally expensive term $\nabla C\left(\boldsymbol{w}^{(t)}\right)$ by a stochastic gradient computed on a random example

- At the t^{th} iteration, we randomly choose an index i_t uniformly from $\{1,2,\ldots,n\}$
- We compute a stochastic gradient $\nabla C_i\left(\boldsymbol{w}^{(t)}\right)$
- We update the model as follows

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \nabla C_i\left(\boldsymbol{w}^{(t)}\right)$$

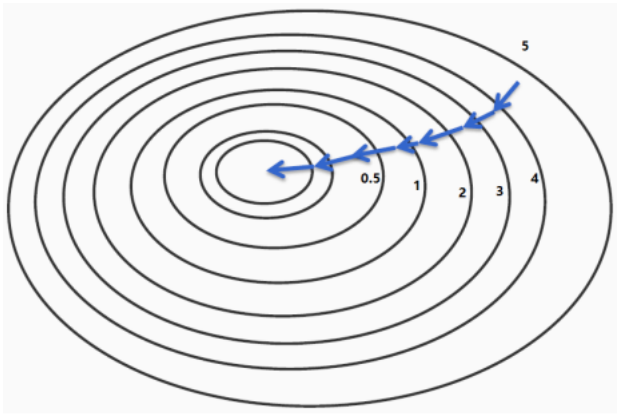
SGD: algorithm

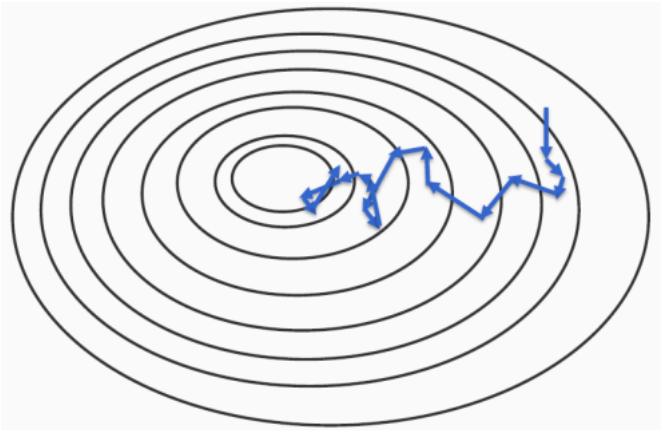
- Stochastic Gradient Descent (Robbins & Monro 1951)
  1. Initialise the weights $\boldsymbol{w}^{(0)}$
  2. For $t = 0,1, \dots, T$
     - Draw i_t from $\{1,2, \dots, n\}$ with equal probability
     - Compute stochastic gradient $\nabla C_i\left(\boldsymbol{w}^{(t)}\right)$ and update

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \nabla C_i\left(\boldsymbol{w}^{(t)}\right)$$

- ∇C_i is not a true gradient and therefore SGD is not as smooth as GD
- On-average, SGD estimates $\frac{1}{n}\sum_{i=1}^{n} C_i\left(\boldsymbol{w}^{(t)}\right) = \nabla C_i\left(\boldsymbol{w}^{(t)}\right)$
- In the long run, SGD would solve the optimisation problem

SGD vs GD

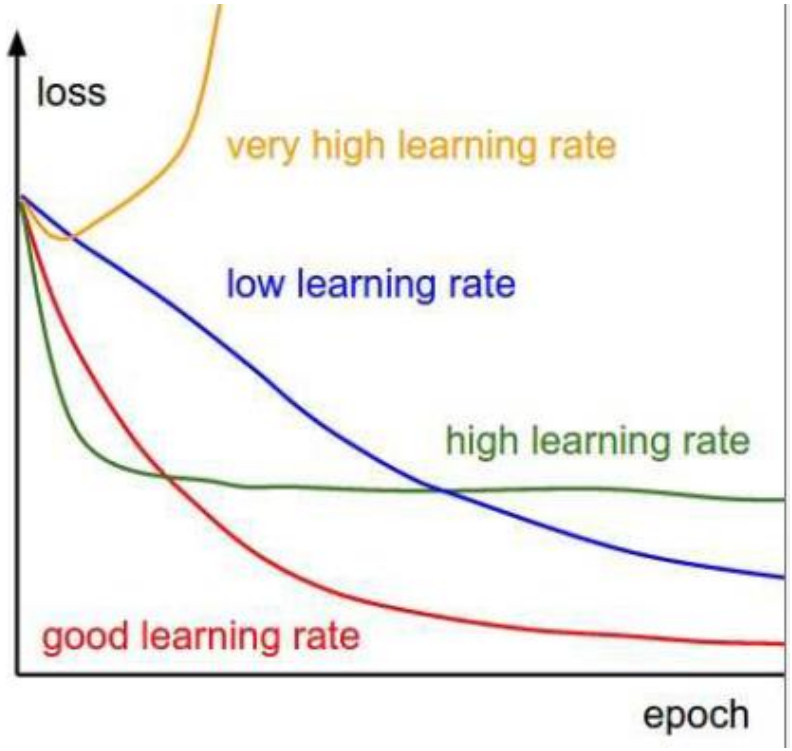




- GD requires more computations per iteration but makes a good progress per iteration.
  - It needs few iterations to get a good solution.
- SGD requires less computations per iteration but makes less update per iteration.
  - Therefore, it needs more iterations to get a good solution.
- GD and SGD cannot always dominate the other.
  - If we want high accuracy and n is small, then **GD** is better.
  - If we want moderate accuracy and n is large, then **SGD** is better.

SGD: effect of learning rates

- If we choose a low learning rate, then SGD would converge very slowly
- If we choose a large learning rate, then SGD would not go further as we run more and more iterations
- If we choose a huge learning rate, then SGD would become unstable
- A typical choice is $\eta_t = \frac{c}{\sqrt{t}}$, where c is a parameter needed to tune.



Summary

- SGD algorithm
- SGD vs GD
- Learning rate effect

Minibatch SGD

Outline: minibatch SGD

1. Introduction
2. Algorithm
3. Minibatch SGD vs SGD vs GD

Minibatch SGD

- Instead of using only one example to compute a stochastic gradient, we can use several examples to compute a stochastic gradient.
- We randomly select a batch of indices: $B_t \subseteq \{1,2,\dots,n\}$ and update the model

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \frac{\eta_t}{b}\sum_{i\in B_t} \nabla C_i\left(\boldsymbol{w}^{(t)}\right)$$

  where b is the batch size

- For example, if $b = 2$ and $B_t = \{2,6\}$ then

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \frac{\eta_t}{2}\left(\nabla C_2\left(\boldsymbol{w}^{(t)}\right) + \nabla C_6\left(\boldsymbol{w}^{(t)}\right)\right)$$

- If $b = 1$, it is clear that minibatch SGD is SGD.

Minibatch SGD: algorithm

- Let $\{\eta_t\}$ be a sequence of step sizes

Algorithm

1. Initialise the weights $\boldsymbol{w}^{(0)}$
2. For $t = 0,1, \dots, T$
   - Randomly select a batch $B_t \subseteq \{1,2, \dots, n\}$ of size b
   - Compute stochastic gradient $\nabla C_i\left(\boldsymbol{w}^{(t)}\right)$ with $i \in B_t$ and update

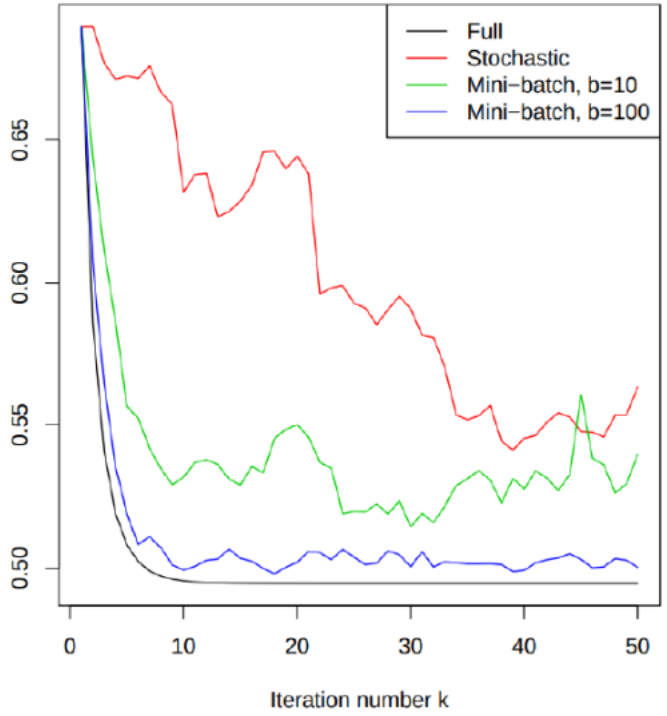
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \frac{\eta_t}{b} \sum_{i \in B_t} \nabla C_i\left(\boldsymbol{w}^{(t)}\right)$$

Minibatch SGD: minibatch selection

- There are two ways to sample the minibatch B_t
  - sampling with replacement: we put it back in the bag after selecting the index (if $b = 2$, it is possible to get $B_t \subseteq \{2,2\}$)
  - sampling without replacement: we do not put it back in the bag after selecting the index (if $b = 2$, it is not possible to get $B_t \subseteq \{2,2\}$)

Minibatch SGD vs SGD vs GD

- Minibatch SGD requires more computations than SGD per iteration to build stochastic gradient.
- Minibatch SGD is more accurate than SGD. Therefore, it converges faster w.r.t. the iteration number.
- We need to balance the accuracy and computation by choosing an appropriate b.
  - Typical choices: $b = 32; b = 64; b = 128$.



Summary

- Algorithm
- Minibatch SGD vs SGD vs GD

Linear classification

Outline: linear classification

1. Introduction
2. 0-1 loss
3. Margin-based loss
4. Surrogate loss functions
5. SGD for linear classification

Linear classification

- Suppose we have

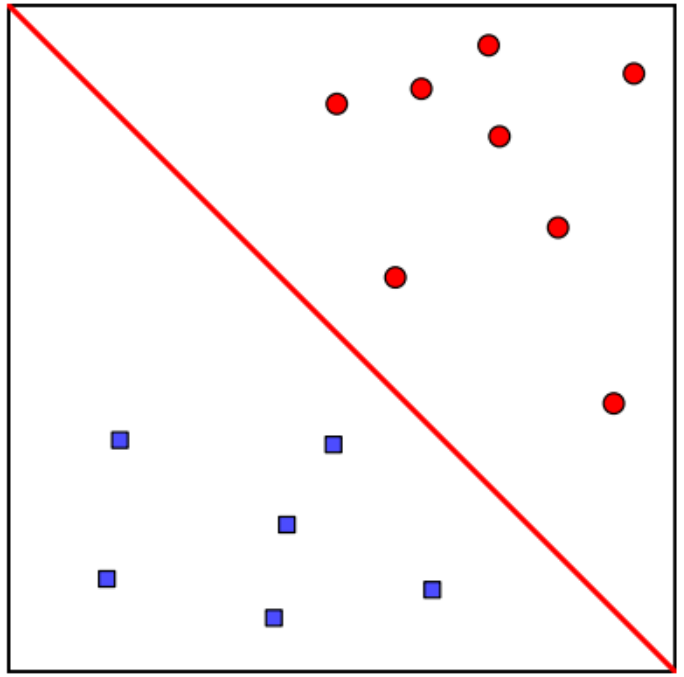
$$D = \{(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), \ldots, (\boldsymbol{x}^n, y^n)\}$$

and

$$y^i \in \{-1, +1\}$$

- We want to build a linear model to separate positive examples from negative examples
  - positive examples are on one-hand of the linear function
  - negative examples are on the other hand.
- A linear model $\boldsymbol{x} \longmapsto \boldsymbol{w}^T \boldsymbol{x}$ outputs a real number, using which we can predict as

$$\hat{y} = \text{sgn}(\boldsymbol{w}^T \boldsymbol{x}) = \begin{cases} +1, & \text{if } \boldsymbol{w}^T \boldsymbol{x} > 0 \\ -1, & \text{otherwise} \end{cases}$$



Linear classification: 0-1 loss

- The performance of a model $\boldsymbol{w}$ on $(\boldsymbol{x}, y)$ can be measured by the 0-1 loss

$$L(\hat{y}, y) = \mathbb{I}[\hat{y} \neq y] = \begin{cases} 1, & \text{if } \hat{y} \neq y \\ 0, & \text{otherwise} \end{cases}$$

- The behaviour of a model on D can be measured by

$$C(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left[\text{sgn}\left(\boldsymbol{w}^T \boldsymbol{x}^i\right) \neq y^i\right]$$

- The minimisation of $C(\boldsymbol{w})$ is challenging since it is not continuous!
- We need to find a surrogate of $C(\boldsymbol{w})$ which is easy to minimise!

Linear classification: margin-based loss

$$\hat{y} = \begin{cases} +1, \\ -1, \end{cases} \quad \begin{matrix} \text{if } \boldsymbol{w}^T\boldsymbol{x} > 0 \\ \text{otherwise} \end{matrix} \Rightarrow \begin{cases} y = \hat{y} = +1, & y = +1, y\boldsymbol{w}^T\boldsymbol{x} > 0 \\ y = \hat{y} = -1, & y = -1, y\boldsymbol{w}^T\boldsymbol{x} \geq 0 \\ y = +1, \hat{y} = -1, & y = +1, y\boldsymbol{w}^T\boldsymbol{x} \leq 0 \\ y = -1, \hat{y} = +1, & y = -1, y\boldsymbol{w}^T\boldsymbol{x} < 0 \end{cases}$$

- $\hat{y} \neq y$ amounts to saying $y\boldsymbol{w}^T\boldsymbol{x} \leq 0$
- Margin
  - The margin of a model $\boldsymbol{w}$ on an example $(\boldsymbol{x}, y)$ is defined as $y\boldsymbol{w}^T\boldsymbol{x}$.
- A model with a positive margin means a correct prediction
- A model with a negative margin means an incorrect prediction
- This further motivates a model with large margin: a large margin means the model is robust in making a correct prediction.

Linear classification: margin-based loss

- We consider the loss function of the form

$$L(\hat{y}, y) = g(y\hat{y})$$

  where g is decreasing.

- We want to minimise the loss $L(\hat{y}, y) = g(y\hat{y})$. Since g is decreasing, minimising L means maximising the margin.
  - Maximising the margin means getting a model with good performance.
  - If g is differentiable, then we can get an “easy” optimisation problem.

Linear classification: surrogate loss (1)

- We mainly consider

$$g(t) = \frac{1}{2}(\max\{0,1-t\})^2 = \begin{cases} 0, & \text{if } t \geq 1 \\ \frac{1}{2}(1-t)^2, & \text{otherwise} \end{cases}$$

- The loss function becomes

$$L(\hat{y}, y) = \frac{1}{2}(\max\{0,1-y\hat{y}\})^2 = \frac{1}{2}(\max\{0,1-y\boldsymbol{w}^T\boldsymbol{x}\})^2$$

- The behaviour on D is quantified by

$$C(\boldsymbol{w}) = \frac{1}{2n}\sum_{i=1}^{n}\left(\max\left\{0,1-y^i\boldsymbol{w}^T\boldsymbol{x}^i\right\}\right)^2$$

Linear classification: minimisation

- By first-order necessary optimality condition, the optimal $\boldsymbol{w}^*$ satisfies $\nabla C(\boldsymbol{w}^*) = 0$
- The gradient can be computed by

$$\begin{cases} 0, \\ \frac{1}{2}\left(1 - y^i \boldsymbol{w}^T \boldsymbol{x}^i\right)^2, \end{cases} \Rightarrow \quad \nabla C_i(\boldsymbol{w}) = \begin{cases} 0, & \text{if } y^i \boldsymbol{w}^T \boldsymbol{x}^i \geq 1 \\ y^i\left(y^i \boldsymbol{w}^T \boldsymbol{x}^i - 1\right)\boldsymbol{x}^i, & \text{otherwise} \end{cases}$$

- We further get

$$\nabla C_i(\boldsymbol{w}) = \begin{cases} 0, & \text{if } y^i \boldsymbol{w}^T \boldsymbol{x}^i \geq 1 \\ \left(\boldsymbol{w}^T \boldsymbol{x}^i - y^i\right)\boldsymbol{x}^i, & \text{otherwise} \end{cases}$$

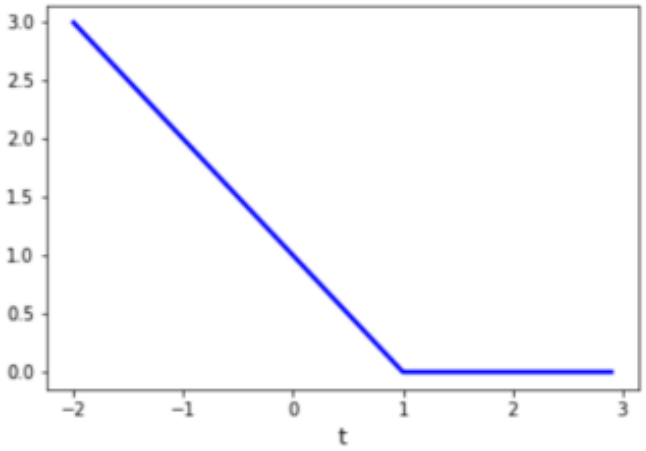
- The first-order optimality condition implies

$$\frac{1}{n}\sum_{i=1}^{n} \nabla C_i(\boldsymbol{w}^*) = 0$$

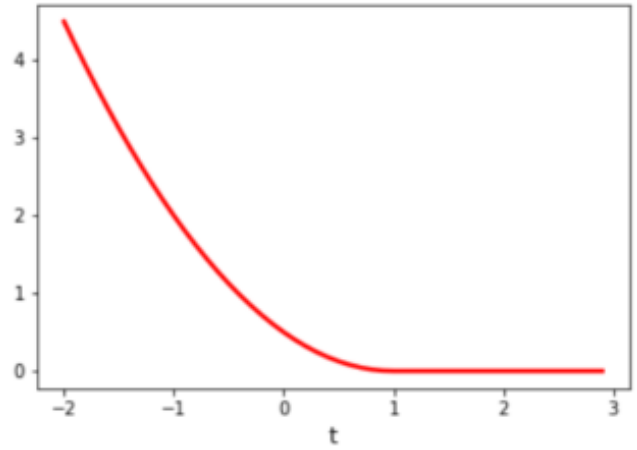
- Difficult to solve since this is a nonlinear problem!

Linear classification: surrogate loss (2)

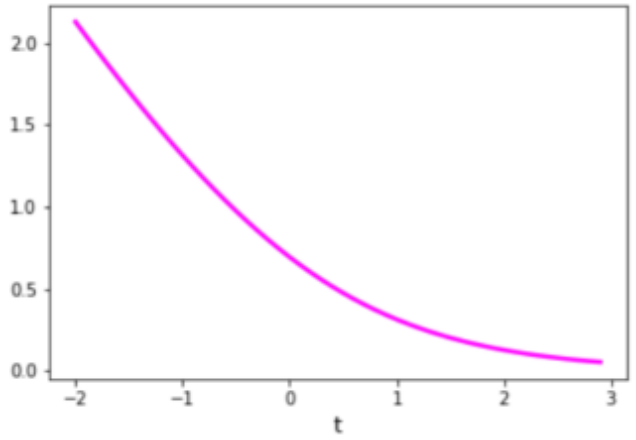
- There are some other choices, including:
  - $g(t) = \max\{0,1 - t\}$
  - $g(t) = \log(1 + \exp(-t))$



$g(t) = \max\{0,1 - t\}$



$g(t) = \frac{1}{2}(\max\{0,1 - t\})^2$



$g(t) = \log(1 + \exp(-t))$

Linear classification: surrogate loss (3)

- If we consider $g(t) = \max\{0, 1 - t\}$, then this leads to the following optimisation problem

$$\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \left(\max\left\{0, 1 - y^i \boldsymbol{w}^T \boldsymbol{x}^i\right\} \right)$$

  which corresponds to the support vector machine.

- If we consider $g(t) = \log(1 + \exp(-t))$, then this leads to

$$\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp(-y^i \boldsymbol{w}^T \boldsymbol{x}^i) \right)$$

  which corresponds to another popular method called logistic regression.

- We recover different popular methods by considering different margin-based loss functions!

SGD for linear classification

- Suppose we consider the following loss function for linear classification

$$C_i(\boldsymbol{w}) = \begin{cases} 0, & \text{if } y^i \boldsymbol{w}^T \boldsymbol{x}^i \geq 1 \\ \frac{1}{2}\left(1 - y^i \boldsymbol{w}^T \boldsymbol{x}^i\right)^2, & \text{otherwise} \end{cases}$$

- If we choose the index i_t, then the update should be

$$\boldsymbol{w}^{(t+1)} = \begin{cases} \boldsymbol{w}^{(t)}, & \text{if } y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t} \geq 1 \\ \boldsymbol{w}^{(t)} - \eta_t \left(y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t} - 1\right) y^{i_t} \boldsymbol{x}^{i_t}, & \text{otherwise} \end{cases}$$

SGD for linear classification: algorithm

for $t = 0,1, \ldots, T$ do

i_t =random index from $\{1,2, \ldots, n\}$

if $y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t} \geq 1$ then

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)}$$

else

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \left(1 - y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t}\right) y^{i_t} \boldsymbol{x}^{i_t}$$

SGD for linear classification: discussion

- Given the linear classification model

$$\boldsymbol{w}^{(t+1)} = \begin{cases} \boldsymbol{w}^{(t)}, & \text{if } y^{i_t}\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{i_t} \geq 1 \\ \boldsymbol{w}^{(t)} + \eta_t\left(y^{i_t} - \boldsymbol{w}^{(t)^T}\boldsymbol{x}^{i_t}\right)\boldsymbol{x}^{i_t}, & \text{otherwise} \end{cases}$$

- Case 1: $y^{i_t}\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{i_t} \geq 1$, the model needs no update
  - i.e., model is doing well on this example, no need to update
- Case 2: $y^{i_t}\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{i_t} < 1$ or alternatively $1 > y^{i_t}\boldsymbol{w}^{(t)^T}\boldsymbol{x}^{i_t}$, the model needs an update
  - The margin is smaller than 1
  - i.e., we're not doing well on this example, so need to update

SGD for linear classification: discussion

- We know that:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \alpha y^{i_t} \boldsymbol{x}^{i_t}$$

where $\alpha = \eta_t \left(1 - y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t}\right) > 0$

- Thus:

$$y^{i_t} \boldsymbol{w}^{(t+1)^T} \boldsymbol{x}^{i_t} = y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t} + \alpha y^{i_t} y^{i_t} \left\| \boldsymbol{x}^{i_t} \right\|^2$$
$$> y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t}$$

- The margin of $\boldsymbol{w}^{(t+1)}$ at $\left(\boldsymbol{x}^{i_t}, y^{i_t}\right)$ is larger than the margin of $\boldsymbol{w}^{(t)}$!

SGD for linear classification: question 1

- If we consider linear classification with the hinge loss

$$C_i(\boldsymbol{w}) = \max\{0, 1 - y^i \boldsymbol{w}^T \boldsymbol{x}^i\}$$

- What is the formula for SGD update?

$$\boldsymbol{w}^{(t+1)} = \begin{cases} \boldsymbol{w}^{(t)}, & \text{if } y^{i_t} \boldsymbol{w}^{(t)^T} \boldsymbol{x}^{i_t} \geq 1 \\ \boldsymbol{w}^{(t)} + \eta_t y^{i_t} \boldsymbol{x}^{i_t}, & \text{otherwise} \end{cases}$$

SGD for linear classification: question 2

- If we consider regularisation in the loss function

$$C_i(\boldsymbol{w}) = \frac{1}{2}\left(\max\left\{0, 1 - y^i \boldsymbol{w}^T \boldsymbol{x}^i\right\}\right)^2 + \frac{1}{2}\lambda\|\boldsymbol{w}\|_2^2$$

- What is the formula for SGD update?

$$\boldsymbol{w}^{(t+1)} = \begin{cases} \boldsymbol{w}^{(t)} - \lambda \boldsymbol{w}^{(t)} & \text{if } y^{i_t} {\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{i_t} \geq 1 \\ \boldsymbol{w}^{(t)} - \lambda \boldsymbol{w}^{(t)} + \eta_t \left(1 - y^{i_t} {\boldsymbol{w}^{(t)}}^T \boldsymbol{x}^{i_t}\right) y^{i_t} \boldsymbol{x}^{i_t}, & \text{otherwise} \end{cases}$$

Summary

- Loss functions
- SGD for linear classification

Summary and reading materials

Week 2: summary

- Gradient descent (GD)
- Stochastic gradient descent (SGD)
- Minibatch SGD
- Linear classification

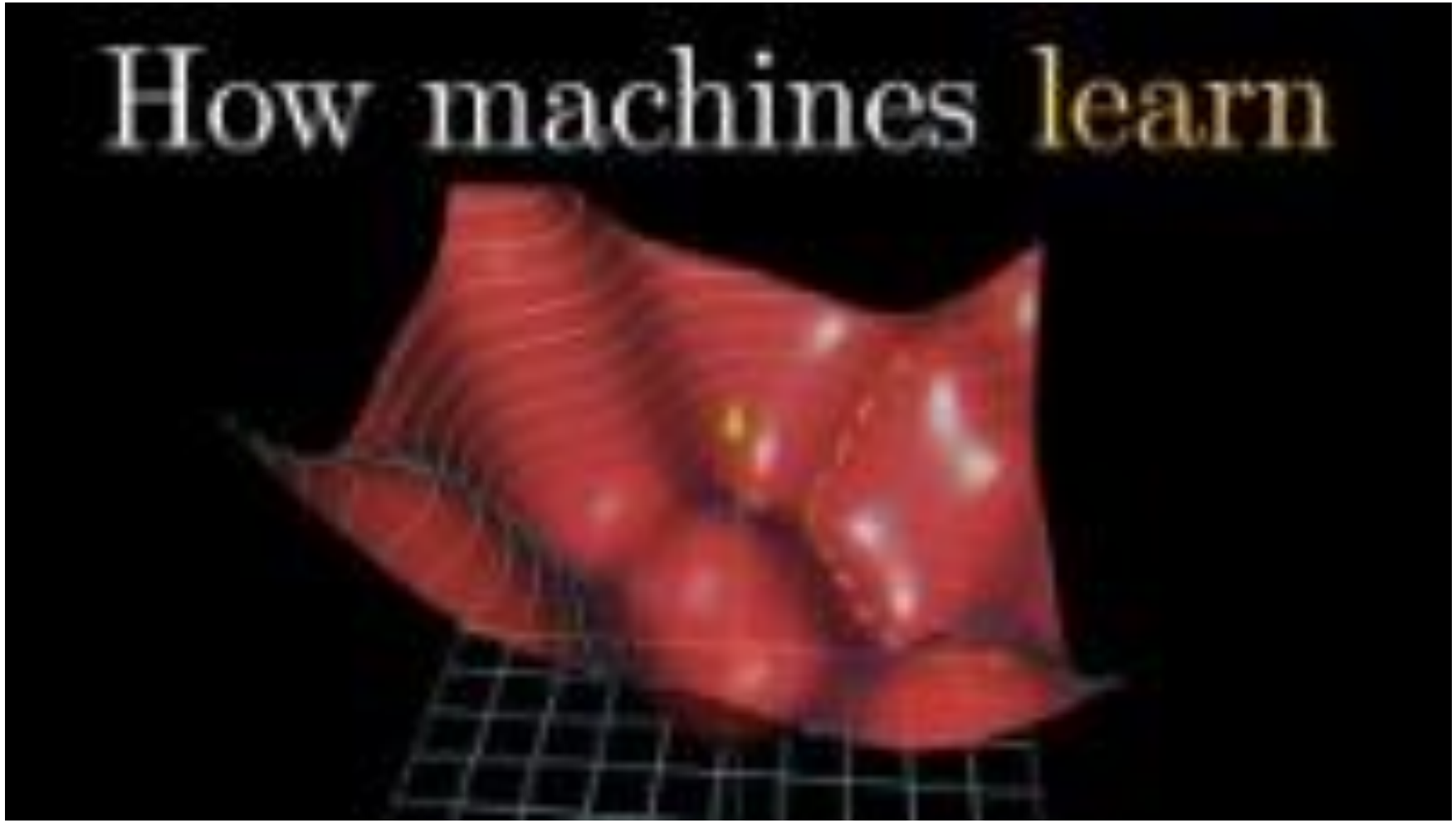
Further reading

- Gradient descent and SGD
  - Section 4.3 of the Deep Learning book
  - Section 4.5 of the Deep Learning book
  - Section 5.9 of the Deep Learning book

- Linear classification
  - Section 4.1.1 of the Pattern Recognition and Machine Learning book
  - Section 4.1.7 of the Pattern Recognition and Machine Learning book

Further watching (optional)

- Optional, but recommended



https://www.youtube.com/watch?v=IHZwWFHWa-w

Practical

- Week 2 lab on GD for Linear Regression
  - See wk2_lab_gradient_descent.ipynb on Canvas
  - Complete the missing bits in this code to complete the solution for linear regression solution via gradient descent (GD)

Credits

- Parts of the material are derived from Yunwen Lei’s materials.