WEEK 5.1

OPTIMIZATION

→ is important in ML bc. we often convert a learning problem to a continuous optimization problem

i.e. - Linear regression (optimize log likelihood, which turned out we can do it w/ an analytical soln)
- Logistic regression
- SVMs
- Neural networks

→ We get an error fn $E(w)$ which we want to minimize

→ If E is completely unconstrained, minimization is impossible. All we could do is search through all possible values w.

↳ BUT if E is continuous, then measuring $E(w)$ at one point gives information about E at nearby points.

ROLE OF DERIVATIVES

→ If we wiggle w_k and keep everything else the same, what happens to the error in this local area I'm in?

↳ Calculus has the answer: $\frac{\partial E}{\partial w_k}$

↳ So we use a differentiable cost fn E and compute partial derivatives of each parameter

↳ The vector of partial derivatives is called the gradient of the error:

$$\nabla E = \frac{\partial E}{\partial \vec{w}} = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n} \right)$$

It points in the direction of steepest error descent in weight space

NUMERICAL OPTIMIZATION ALGO.

→ tries to solve the general problem: $\min_w E(w)$

→ Most commonly, a numerical optimization procedure takes 2 inputs:

1) a procedure that computes $E(w)$

2) a procedure that computes the partial derivative $\frac{\partial E}{\partial w_j}$

→ are iterative; they generate a sequence of points

→ One example is the **gradient descent algorithm**

```
1  initialize w          → start w/ a guess for the weight matrix w
2  while E(w) is unacceptably high
3      calculate g ← ∂E/∂w     → calculate gradient of E given w
4      w ← w − ηg              → adjust the weight matrix by a small distance in the direction in weight space along which E decreases most rapidly i.e. in the direction of −g
5  endwhile
6  return w
```

↳ The idea of gradient descent is that to minimise an error fn w/ respect to the parameters, we want to take small steps in a downhill direction

bc. the gradient is not uniform, and if we take too big a step, we may end up going uphill again.

↳ η is known as the step size/learning rate

- If η is too small, it'll take too long to reach the optimum
- If η is too large, the gradient descent algo. may oscillate

↳ There is a simple heuristic for choosing η → "BOLD-DRIVER" approach

```
1  initialize w, η
2  initialize e ← E(w);   g ← ∇E(w)
3  while η > 0
4      w_1 ← w − ηg
5      e_1 = E(w_1);   g_1 = ∇E
6      if e_1 ≥ e      → If we are increasing our error, we reduce our step size by half and recalculate again
7          η = η / 2
8      else
9          η = 1.01η;   w ← w_1;   g ← g_1;   e = e_1
10 endwhile            → If moving in the right direction, then we cautiously increase our learning rate
11 return w
```

BATCH vs. ONLINE GRADIENT DESCENT

→ **Batch learning** updates the model parameters after evaluating ALL training instances

which is: $\frac{\partial E}{\partial w} = \sum_i \frac{\partial E_i}{\partial w}$

- The algos. in the prev. page go through all the training instance before updating model param. → HUGE COMPUTATION!
- Can give you a more powerful optimization methods
- Easier to analyze

→ **Online learning** updates the model parameters after evaluating each training instance

- One type of this is called "stochastic gradient ascent", which is when we have picked the training instance randomly:

```
1  initialize w
2  while E(w) is unacceptably high
3      Pick j as uniform random int. in 1..N
4      Calculate g ← ∂E_j/∂w
5      w ← w − ηg
6  endwhile
7  return w
```

- It is more feasible for huge datasets
- May have the ability to jump over local optima

→ You can do a 'mix' of batch & online, where you update the model parameters after evaluating, say, 10% of the training instances.

PROBLEMS W/ GRADIENT DESCENT

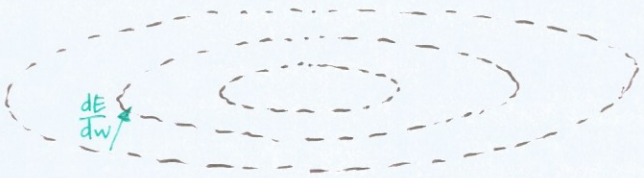
1) Shallow Valleys

Gradient descent goes very slowly once it hits the shallow valley.

↳ One way to deal w/ this is momentum

2) Curved Error Surfaces

→ The gradient might not point towards the optimum because of curvature



→ Local curvature is measured by the Hessian matrix

3) Local Minima

→ If we follow gradient descent, we'll get stuck in a local minima (Gradient is 0, so you stop)

→ Can use momentum to help get over this

→ Certain fns such as squared error / logistic regression likelihood are convex, meaning that the 2nd derivative is always (+)ive.

↳ Any local minimum is global

Hence, if an optimisation problem is convex, then the surface of the objective fn has only one minimum (the global minimum)