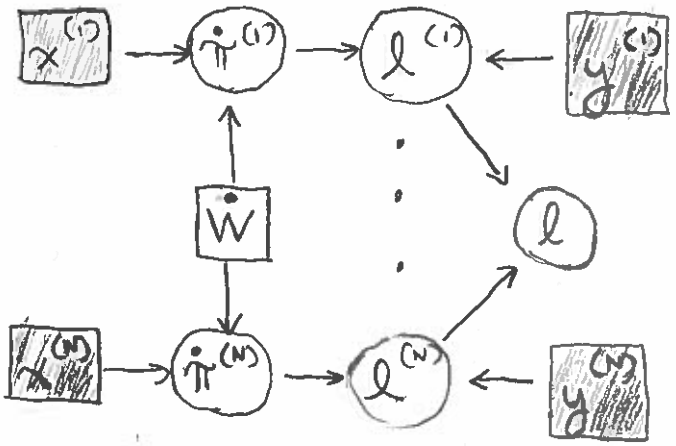
Minibatch Gradient Descent

(1) Consider the full causal diagram for training a 1-layer feedforward neural network:



For simplicity, suppose that W is a 1×1 (singleton) matrix with entry w. Thus, the goal of training is to compute $\hat{w} = \operatorname{argmin}_w \ell$.

(2) Let's suppose we're using the loss function for ordinary linear regression ("ordinary least squares" loss)

$$\ell = f(w) = \sum_{n=1}^{N} (y^{(n)} - x^{(n)} w)^2$$

Minibatch Gradient Descent

③ Now what if you have so many training examples that you can't fit them in memory? (i.e. N is very large). This poses a difficulty for gradient descent:

GradientDescent (loss function L, learning rate $\alpha \in \mathbb{R}$):

- initialize $w^{(0)}$ to some real number; $t \leftarrow 0$
- repeat until happy:
  - let update $\sigma^{(t)} \leftarrow -\alpha \cdot \frac{dL}{dw}(w^{(t)}) = -\alpha \cdot \sum_{n=1}^{N} \frac{dL^{(n)}}{dw}(w^{(t)})$
  - let next guess $w^{(t+1)} \leftarrow w^{(t)} + \sigma^{(t)}$
  - let $t \leftarrow t+1$

this is expensive to compute!

---

④ A common solution is to split the training data into bitesize "minibatches" of a fixed size. Create a partition $B_1, B_2, \ldots, B_p$ of the indices $\{1, \ldots, N\}$. Then:

MinibatchGradientDescent (L, α):

- initialize $w^{(0)}$ to some real number; $t \leftarrow 0$
- repeat until happy:
  - for i in $\{1, \ldots, p\}$:
    - let update $\sigma^{(t)} \leftarrow -\alpha \cdot \sum_{n \in B_i} \frac{dL^{(n)}}{dw}(w^{(t)})$
    - let next guess $w^{(t+1)} \leftarrow w^{(t)} + \sigma^{(t)}$
    - let $t \leftarrow t+1$

← this is typically called an epoch

Minibatch Gradient Descent

(5) This solves our memory problem, but at what cost? WHAT DID WE JUST DO? We wanted to compute

$$-\alpha \cdot \frac{dL}{dw}(w^{(t)})$$

but instead we computed:

$$-\alpha \cdot \sum_{n \in B_i} \frac{dL^{(n)}}{dw}(w^{(t)}) \neq -\alpha \cdot \frac{dL}{dw}(w^{(t)})$$

(6) Imagine we had the following data

n	$x^{(n)}$	$y^{(n)}$
1	2	4
2	4	9
3	-1	-2
4	5	15

and we split it into minibatches $B_1 = \{1, 2\}$

$B_2 = \{3, 4\}$

Minibatch Gradient Descent

(7) When we compute our first update (over minibatch B_1), we are computing the gradient

$$\sum_{n \in B_1} \frac{dL^{(n)}}{dw}(w^{(t)}) = \frac{dL^{(1)}}{dw}(w^{(t)}) + \frac{dL^{(2)}}{dw}(w^{(t)})$$

which is the gradient of loss function

$$L'(w) = L^{(1)}(w) + L^{(2)}(w)$$

---

(8) When we compute our second update (over minibatch B_2), we are computing the gradient

$$\sum_{n \in B_2} \frac{dL^{(n)}}{dw}(w^{(t)}) = \frac{dL^{(3)}}{dw}(w^{(t)}) + \frac{dL^{(4)}}{dw}(w^{(t)})$$

which is the gradient of loss function

$$L''(w) = L^{(3)}(w) + L^{(4)}(w)$$

---

(9) Notice that neither of these is our actual loss function:

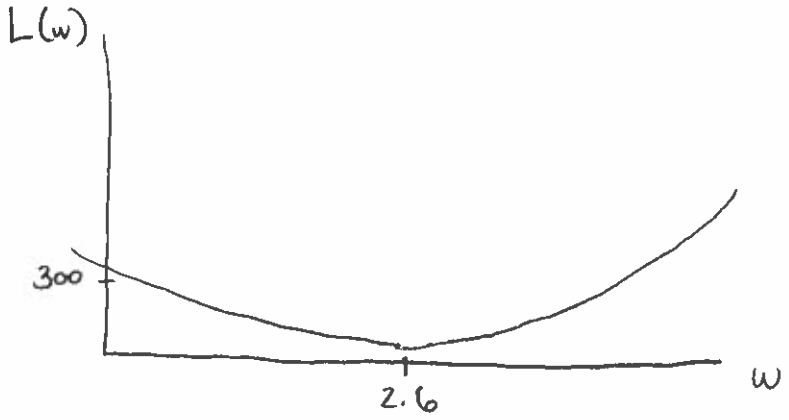
$$L(w) = \sum_{n=1}^{4} L^{(n)}(w) = (4-2w)^2 + (9-4w)^2 + (-2+w)^2 + (15-5w)^2$$

$$L'(w) = L^{(1)}(w) + L^{(2)}(w) = (4-2w)^2 + (9-4w)^2$$

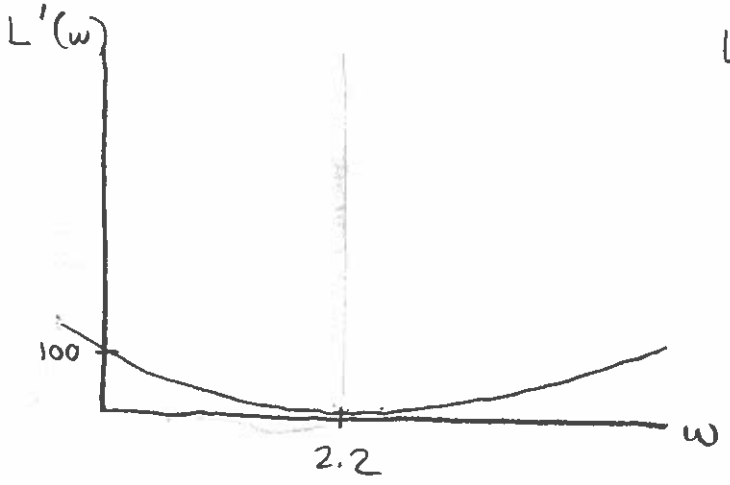
$$L''(w) = L^{(3)}(w) + L^{(4)}(w) = (-2+w)^2 + (15-5w)^2$$

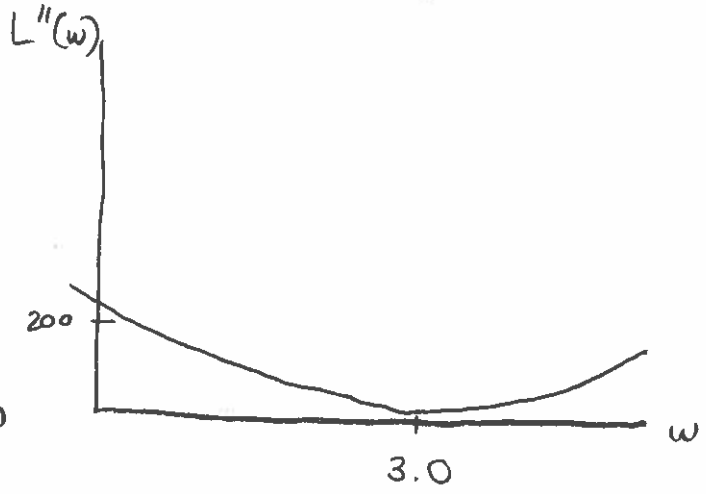
Minibatch Gradient Descent

(10) While $L(w)$ looks as follows:

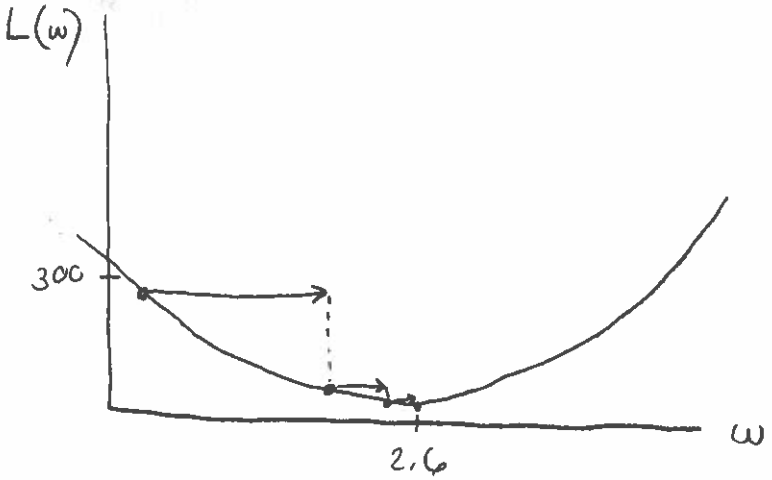


The other two losses are similar, but not the same:



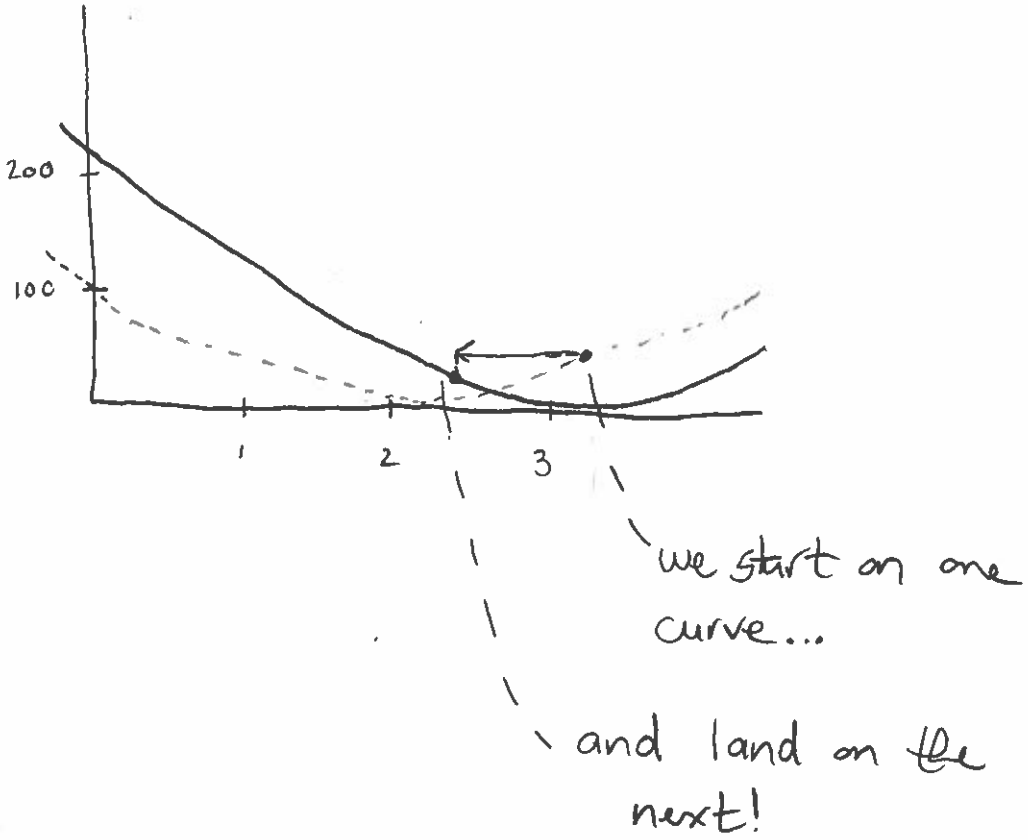


(11) Standard gradient descent repeatedly finds the gradient, then takes a step in that direction (with the magnitude of the step depending on the steepness of the gradient):



Minibatch Gradient Descent

(12) With minibatch descent, the grand beneath our feet changes with every step!



(13) Why should we expect this to approximate gradient descent on the true loss function?

Well, at the very least we know that if we assume that our data is drawn independently from an identical distribution, then as our batch size increases:

$$\sum_{n \in B_i} L(w) \longrightarrow \sum_{n=1}^{N} L(w)$$

Minibatch Gradient Descent

(14) So with a sufficiently large batch size, each minibatch loss function should be approximately equal to the true loss function. Thus minibatch gradient descent will behave similarly to gradient descent.

(15) What batch size should you use?

Consider the update computation:

$$w^{(t)} \leftarrow -\alpha \cdot \sum_{n \in B_i} \frac{dL^{(n)}}{dw}(w^{(t)})$$

This summation can be expressed as a matrix/tensor operation and thus can be better processed with batch sizes between 32 and 256 (too small underutilizes the parallel computation power of GPUs; too large overflows the capacity).

(16) Sometimes people observe that small batch sizes actually help reduce overfitting (i.e. the trained models generalize better because noisy pieces of data do not impact every single update computation).

However, with extremely small batch sizes, the training may be unstable (since the loss function can fluctuate wildly), so you need to worry about the learning rate more.