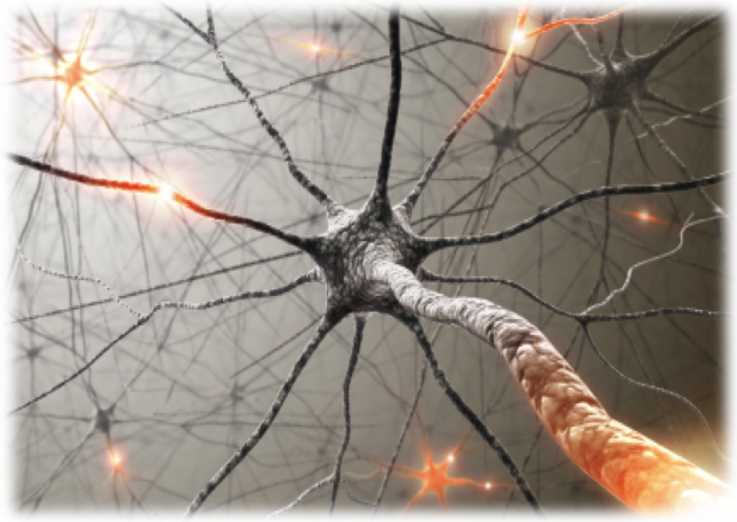
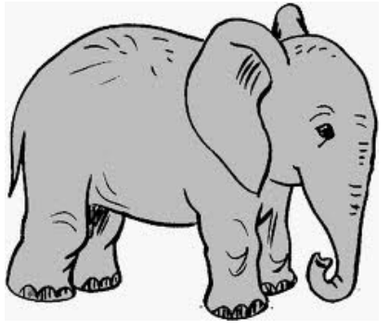
Machine learning: stochastic gradient descent



- In this module, we will introduce stochastic gradient descent.

Gradient descent is slow

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$



Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

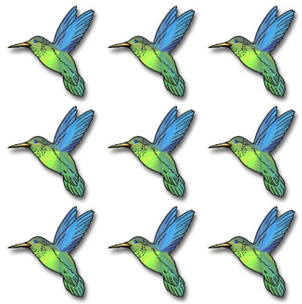
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

Problem: each iteration requires going over all training examples — expensive when have lots of data!

- So far, we've seen gradient descent as a general-purpose algorithm to optimize the training loss.
- But one problem with gradient descent is that it is slow.
- Recall that the training loss is a sum over the training data. If we have one million training examples, then each gradient computation requires going through those one million examples, and this must happen before we can make any progress.
- Can we make progress before seeing all the data?

Stochastic gradient descent

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$



Algorithm: stochastic gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

  For $(x, y) \in \mathcal{D}_{\text{train}}$:

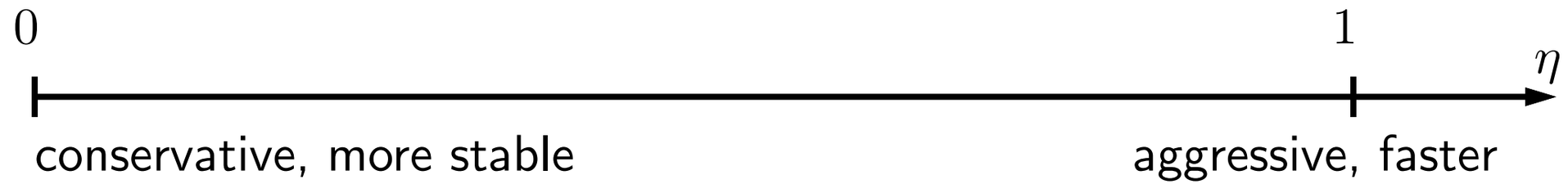
    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$

- The answer is **stochastic gradient descent** (SGD).
- Rather than looping through all the training examples to compute a single gradient and making one step, SGD loops through the examples (x, y) and updates the weights $\mathbf{w}$ based on **each** example.
- Each update is not as good because we're only looking at one example rather than all the examples, but we can make many more updates this way.
- Aside: there is a continuum between SGD and GD called minibatch SGD, where each update consists of an average over B examples.
- Aside: There are other variants of SGD. You can randomize the order in which you loop over the training data in each iteration. Think about why this is important if in your training data, you had all the positive examples first and the negative examples after that.

Step size

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

Question: what should η be?



Strategies:

- Constant: $\eta = 0.1$
- Decreasing: $\eta = 1/\sqrt{\#\text{ updates made so far}}$

- One remaining issue is choosing the step size, which in practice is quite important.
- Generally, larger step sizes are like driving fast. You can get faster convergence, but you might also get very unstable results and crash and burn.
- On the other hand, with smaller step sizes you get more stability, but you might get to your destination more slowly. Note that the weights do not change if $\eta = 0$
- A suggested form for the step size is to set the initial step size to 1 and let the step size decrease as the inverse of the square root of the number of updates we've taken so far.
- Aside: There are more sophisticated algorithms like AdaGrad and Adam that adapt the step size based on the data, so that you don't have to tweak it as much.
- Aside: There are some nice theoretical results showing that SGD is guaranteed to converge in this case (provided all your gradients are bounded).

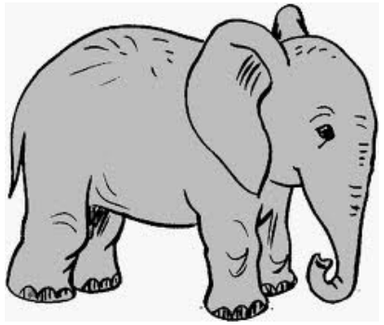
Stochastic gradient descent in Python

[code]

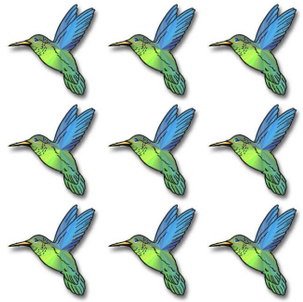
- Now let us code up stochastic gradient descent for linear regression in Python.
- First we generate a large enough dataset so that speed actually matters. We will also generate 1 million points according to $x \sim \mathcal{N}(0, I)$ and $y \sim \mathcal{N}(\mathbf{w}^* \cdot x, 1)$, where $\mathbf{w}^*$ is the true weight vector, but hidden to the algorithm.
- This way, we can diagnose whether the algorithm is actually working or not by checking whether it recovers something close to $\mathbf{w}^*$.
- Let's first run gradient descent, and watch that it makes progress but it is very slow.
- Now let us implement stochastic gradient descent. It is much faster.

Summary

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$



gradient descent



stochastic gradient descent

Key idea: stochastic updates

It's not about **quality**, it's about **quantity**.

- In summary, we've shown how stochastic gradient descent can be faster than gradient descent.
- Gradient just spends too much time refining its gradient (quality), while you can get a quick and dirty estimate just from one sample and make more updates (quantity).
- Of course, sometimes stochastic gradient descent can be unstable, and other techniques such as mini-batching can be used to stabilize it.