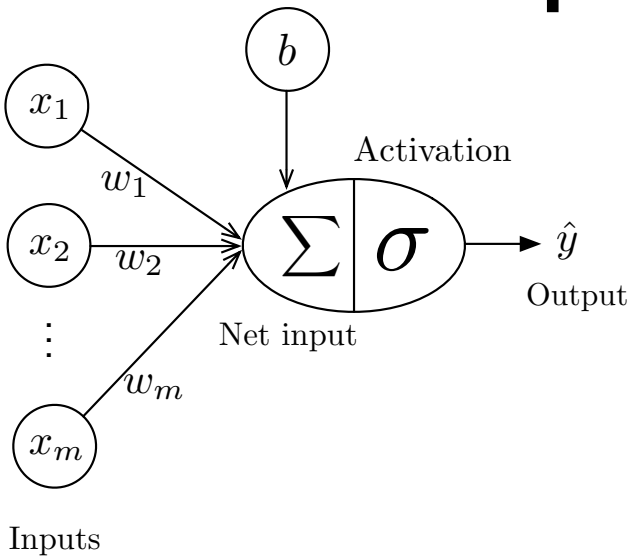
Training neural nets: Online, batch, and minibatch modes

Deep Learning & Applications

Perceptron Recap



$$\sigma\left(\sum_{i=1}^{m} x_i w_i + b\right) = \sigma\left(\mathbf{x}^T\mathbf{w} + b\right) = \hat{y}$$

$$\sigma(z) = \begin{cases} 0, & z \leq 0 \\ 1, & z > 0 \end{cases}$$

$$b = -\theta$$

Let $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) $\hat{y}^{[i]} := \sigma(\mathbf{x}^{[i]T}\mathbf{w} + b)$ ← Compute output (prediction)
      (b) $\text{err} := (y^{[i]} - \hat{y}^{[i]})$ ← Calculate error
      (c) $\mathbf{w} := \mathbf{w} + err \times \mathbf{x}^{[i]}$, $b := b + err$ ← Update parameters

General Learning Principle

Let $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

"On-line" mode

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) Compute output (prediction)
      (b) Calculate error
      (c) Update $\mathbf{w}, b$

This applies to all common neuron models and (deep) neural network architectures!

There are some variants of it, namely the "batch mode" and the "minibatch mode" which we will briefly go over in the next slides and then discuss more later

General Learning Principle

Let $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

"On-line" mode

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) Compute output (prediction)
      (b) Calculate error
      (c) Update $\mathbf{w}, b$

Batch mode

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. Initialize $\Delta\mathbf{w} := 0,\ \Delta b := 0$
   B. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) Compute output (prediction)
      (b) Calculate error
      (c) Update $\Delta\mathbf{w}, \Delta b$
   C. Update $\mathbf{w}, b$:
      $\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, b := +\Delta b$

General Learning Principle

Let $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

"On-line" mode

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) Compute output (prediction)
      (b) Calculate error
      (c) Update $\mathbf{w}, b$

In practice, we usually shuffle the dataset prior to each epoch to prevent cycles

Batch mode

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. Initialize $\Delta\mathbf{w} := 0,\ \Delta b := 0$
   B. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) Compute output (prediction)
      (b) Calculate error
      (c) Update $\Delta\mathbf{w}, \Delta b$
   C. Update $\mathbf{w}, b$:

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, b := +\Delta b$$

General Learning Principle

Let $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

"On-line" mode

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) Compute output (prediction)
      (b) Calculate error
      (c) Update $\mathbf{w}, b$

In practice, we usually shuffle the dataset prior to each epoch to prevent cycles

"On-line" mode II (alternative)

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For for t iterations:
   A. Pick random $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
      (a) Compute output (prediction)
      (b) Calculate error
      (c) Update $\mathbf{w}, b$

No shuffling required

(actually, not really stochastic because a fixed training set instead of sampling from the population)

General Learning Principle

Let $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

Minibatch mode

(mix between on-line and batch)

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   3. For every minibatch of size *k*:
      A. Initialize $\Delta \mathbf{w} := 0,\ \Delta b := 0$
      B. For every $\{\langle \mathbf{x}^{[i]}, y^{[i]} \rangle, ..., \langle \mathbf{x}^{[i+k]}, y^{[i+k]} \rangle\} \subset \mathcal{D}$:
         (a) Compute output (prediction)
         (b) Calculate error
         (c) Update $\Delta \mathbf{w}, \Delta b$
      C. Update $\mathbf{w}, b$:

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, b := +\Delta b$$

The most common mode in deep learning. Any ideas why?

General Learning Principle

Let $\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$

Minibatch mode

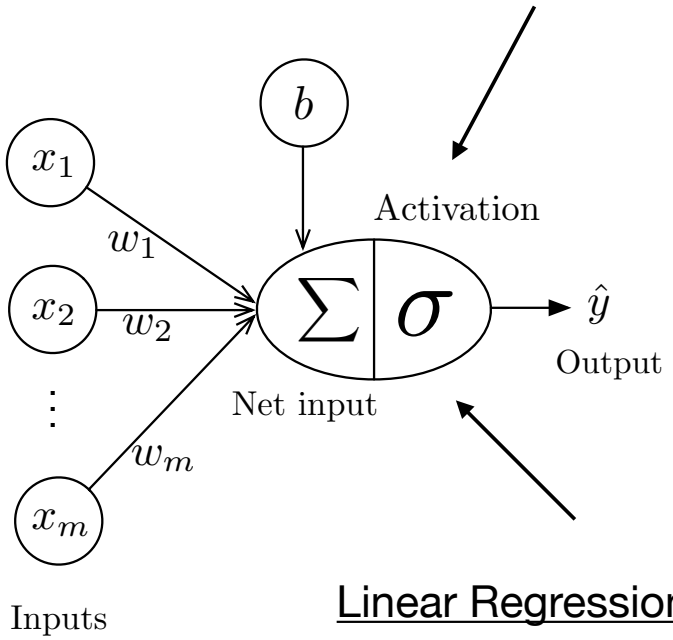
(mix between on-line and batch)

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
    3. For every minibatch of size *k*:
        A. Initialize $\Delta \mathbf{w} := 0, \ \Delta b := 0$
        B. For every $\{\langle \mathbf{x}^{[i]}, y^{[i]} \rangle, ..., \langle \mathbf{x}^{[i+k]}, y^{[i+k]} \rangle\} \subset \mathcal{D}$
            (a) Compute output (prediction)
            (b) Calculate error
            (c) Update $\Delta \mathbf{w}, \Delta b$
        C. Update $\mathbf{w}, b$:
            $\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, b := +\Delta b$

Most commonly used in DL, because

1. Choosing a subset (vs 1 example at a time) takes advantage of vectorization (faster iteration through epoch than on-line)
2. having fewer updates than "on-line" makes updates less noisy
3. makes more updates/ epoch than "batch" and is thus faster

Linear Regression



Perceptron: Activation function is the threshold function

The output is a binary label $\hat{y} \in \{0, 1\}$

You can think of linear regression as a linear neuron!

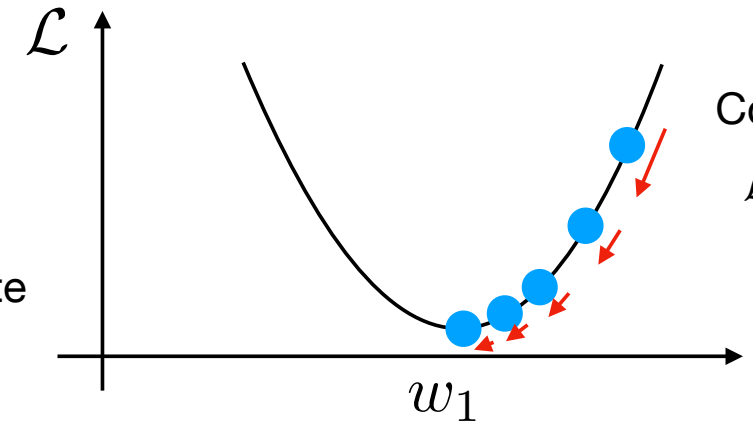
Linear Regression: Activation function is the identity function

$$\sigma(x) = x$$

The output is a real number $\hat{y} \in \mathbb{R}$

Gradient Descent

Learning rate and steepness of the gradient determine how much we update

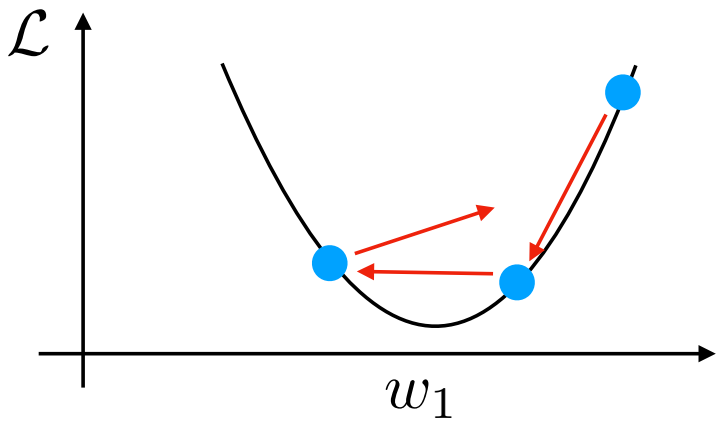


Convex loss function

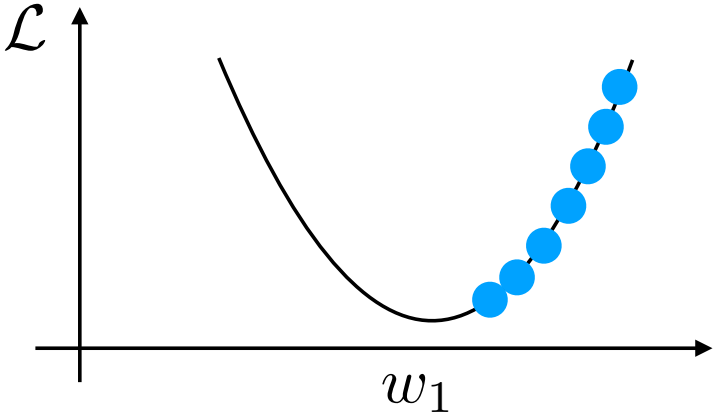
$$\mathcal{L}(\mathbf{w}, b) = \sum_i (\hat{y}^{[i]} - y^{[i]})^2$$

Gradient Descent

If the learning rate is too large, we can overshoot



If the learning rate is too small, convergence is very slow



(Least-Squares) Linear Regression

The update rule turns out to be this:

"On-line" mode

Perceptron learning rule

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$
      (a) $\hat{y}^{[i]} := \sigma(\mathbf{x}^{[i]T}\mathbf{w} + b)$
      (b) $\text{err} := (y^{[i]} - \hat{y}^{[i]})$
      (c) $\mathbf{w} := \mathbf{w} + err \times \mathbf{x}^{[i]}$
          $b := b + err$

Stochastic gradient descent

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$
      (a) $\hat{y}^{[i]} := \sigma(\mathbf{x}^{[i]T}\mathbf{w} + b)$
      (b) $\nabla_{\mathbf{w}}\mathcal{L} = (y^{[i]} - \hat{y}^{[i]})\mathbf{x}^{[i]}$
          $\nabla_b\mathcal{L} = (y^{[i]} - \hat{y}^{[i]})$
      (c) $\mathbf{w} := \mathbf{w} + \eta \times (-\nabla_{\mathbf{w}}\mathcal{L})$
          $b := b + \eta \times (-\nabla_b\mathcal{L})$

learning rate

negative gradient

Linear Regression Loss Derivative

$$\mathcal{L}(\mathbf{w}, b) = \sum_i (\hat{y}^{[i]} - y^{[i]})^2$$ Sum Squared Error (SSE) loss

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\partial}{\partial w_j} \sum_i (\hat{y}^{[i]} - y^{[i]})^2 \\
&= \frac{\partial}{\partial w_j} \sum_i (\sigma(\mathbf{w}^T \mathbf{x}^{[i]}) - y^{[i]})^2 \\
&= \sum_i 2(\sigma(\mathbf{w}^T \mathbf{x}^{[i]}) - y^{[i]}) \frac{\partial}{\partial w_j} (\sigma(\mathbf{w}^T \mathbf{x}^{[i]}) - y^{[i]}) \\
&= \sum_i 2(\sigma(\mathbf{w}^T \mathbf{x}^{[i]}) - y^{[i]}) \frac{d\sigma}{d(\mathbf{w}^T \mathbf{x}^{[i]})} \frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}^{[i]} \\
&= \sum_i 2(\sigma(\mathbf{w}^T \mathbf{x}^{[i]}) - y^{[i]}) \frac{d\sigma}{d(\mathbf{w}^T \mathbf{x}^{[i]})} x_j^{[i]} \\
&= \sum_i 2(\sigma(\mathbf{w}^T \mathbf{x}^{[i]}) - y^{[i]}) x_j^{[i]}
\end{aligned}$$

(Note that the activation function is the identity function in linear regression)

Linear Regression Loss Derivative (alt.)

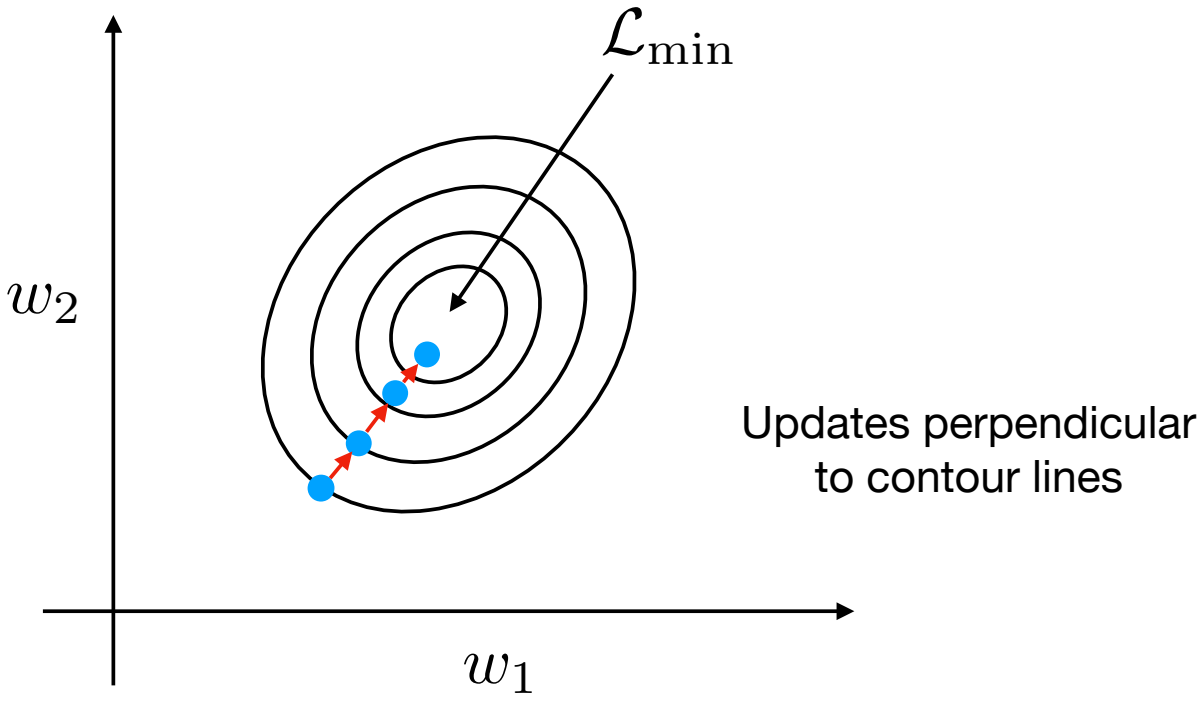
$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{2n}\sum_i (\hat{y}^{[i]} - y^{[i]})^2$$

Mean Squared Error (MSE) loss often scaled by factor 1/2 for convenience

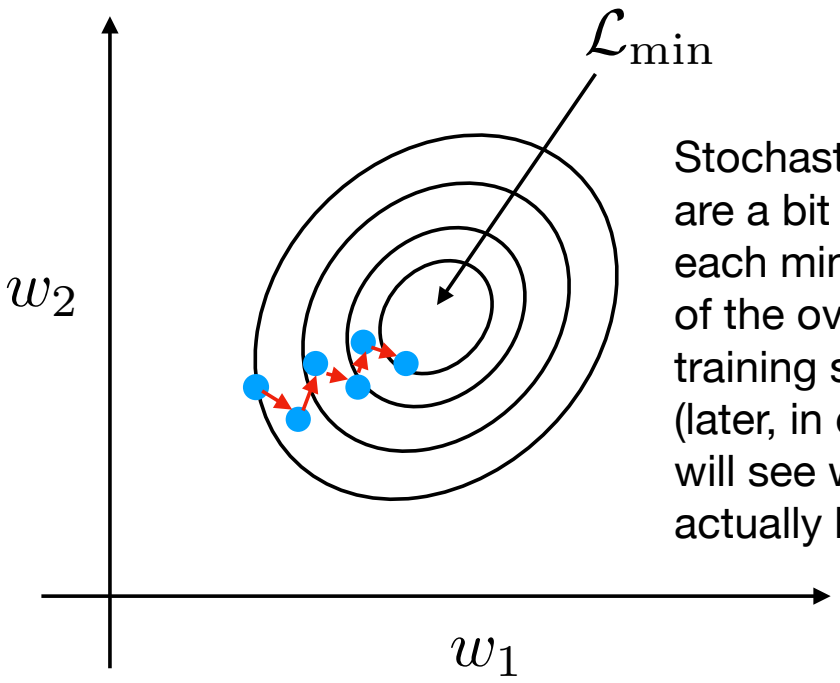
$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2n}\sum_i (\hat{y}^{[i]} - y^{[i]})^2 \\
&= \frac{\partial}{\partial w_j}\sum_i \frac{1}{2n}(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})^2 \\
&= \sum_i \frac{1}{n}(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})\frac{\partial}{\partial w_j}(\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]}) \\
&= \frac{1}{n}\sum_i (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})\frac{d\sigma}{d(\mathbf{w}^T\mathbf{x}^{[i]})}\frac{\partial}{\partial w_j}\mathbf{w}^T\mathbf{x}^{[i]} \\
&= \frac{1}{n}\sum_i (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})\frac{d\sigma}{d(\mathbf{w}^T\mathbf{x}^{[i]})}x_j^{[i]} \\
&= \frac{1}{n}\sum_i (\sigma(\mathbf{w}^T\mathbf{x}^{[i]}) - y^{[i]})x_j^{[i]}
\end{aligned}$$

(Note that the activation function is the identity function in linear regression)

Batch Gradient Descent as Surface Plot

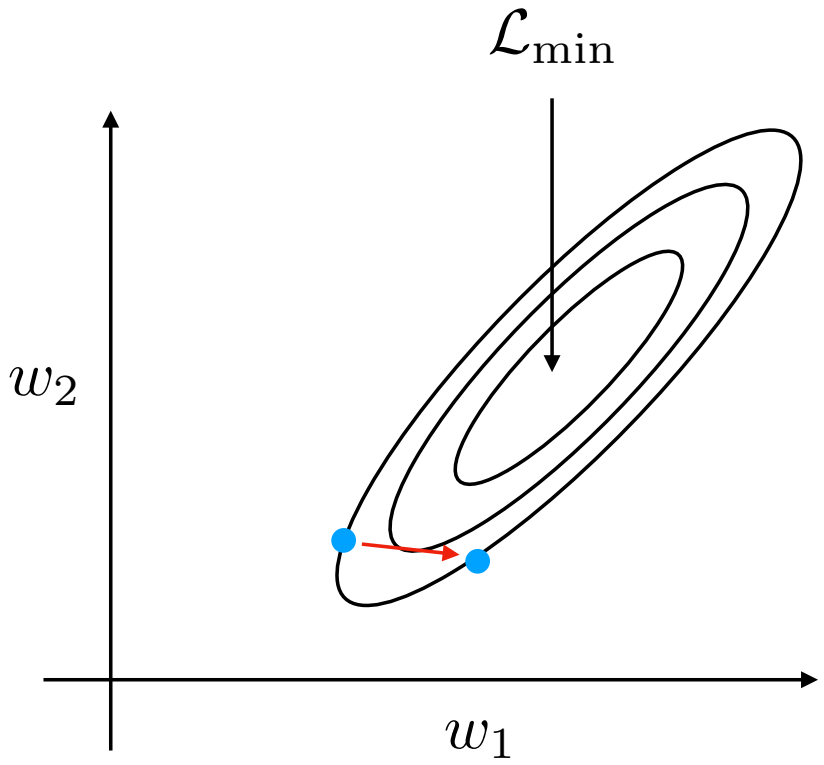


Stochastic Gradient Descent as Surface Plot



Stochastic updates
are a bit noisier, because
each minibatch is an approximation
of the overall loss on the
training set
(later, in deep neural nets, we
will see why noisier updates are
actually helpful)

Batch Gradient Descent as Surface Plot



If inputs are on very different scales some weights will update more than others ... and it will also harm convergence

(always normalize inputs!)