Intro to Deep Learning

Lecture 6: Optimization II

Recall Stochastic Gradient Descent

1. Pick example (x, y)
2. Pick ML model $f_w(\,.\,)$ with weights w
3. Pick loss function ℓ to minimize (e.g. squared)
4. Optimization becomes:

$$\mathscr{L}(w) = l\big(y, f_w(x)\big)$$

5. Gradient via chain rule

$$\nabla \mathscr{L}(w) = \ell'(y, f_w(x)) \times \nabla f_w(x)$$

Derivative of Loss

Gradient of Model

HOW?

Loss Functions

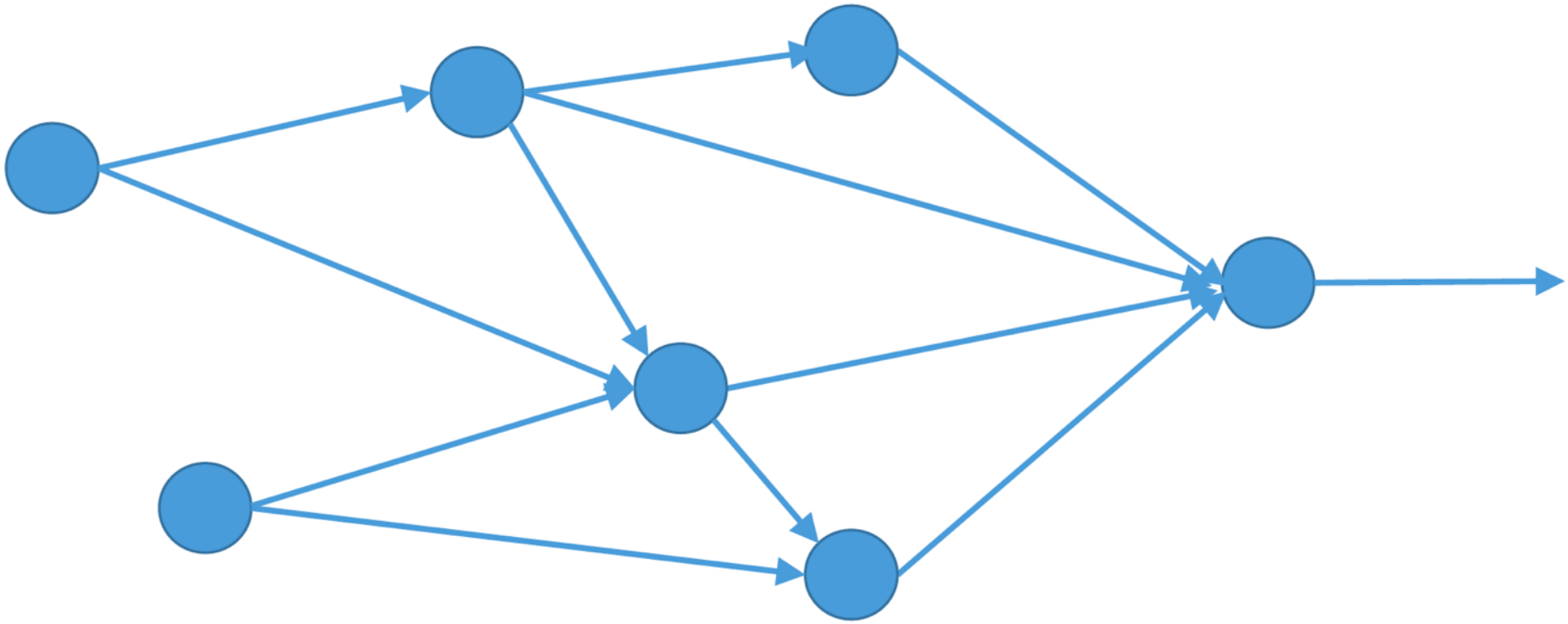
- Square loss: $\ell(y, f(x)) = \frac{1}{2}\big(f(x) - y\big)^2$

$$\ell'(y, f(x)) = \big(f(x) - y\big)$$

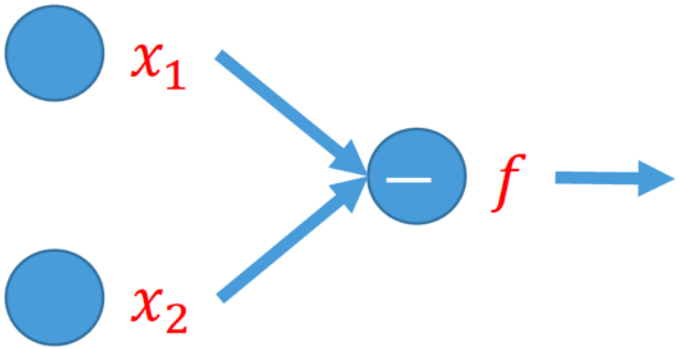
- Cross entropy loss

$$\ell\big(y, f(x)\big) = -\sum_{i \in [k]} y_i \log(p_i), \quad \text{where} \quad p_i = \frac{e^{f_i(x)}}{\sum_{j=1}^{n} e^{f_j(x)}}$$

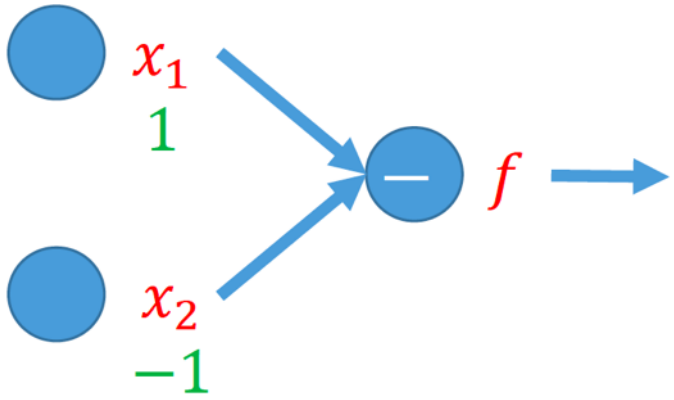
How to differentiate a neural net?



Answer: *Backpropagate*

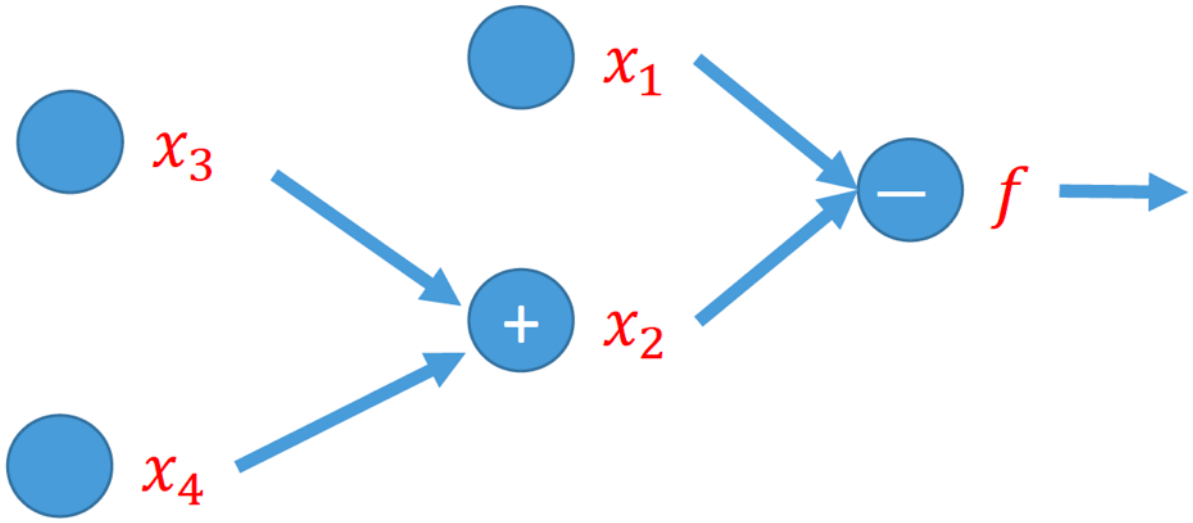


Function: $f = x_1 - x_2$

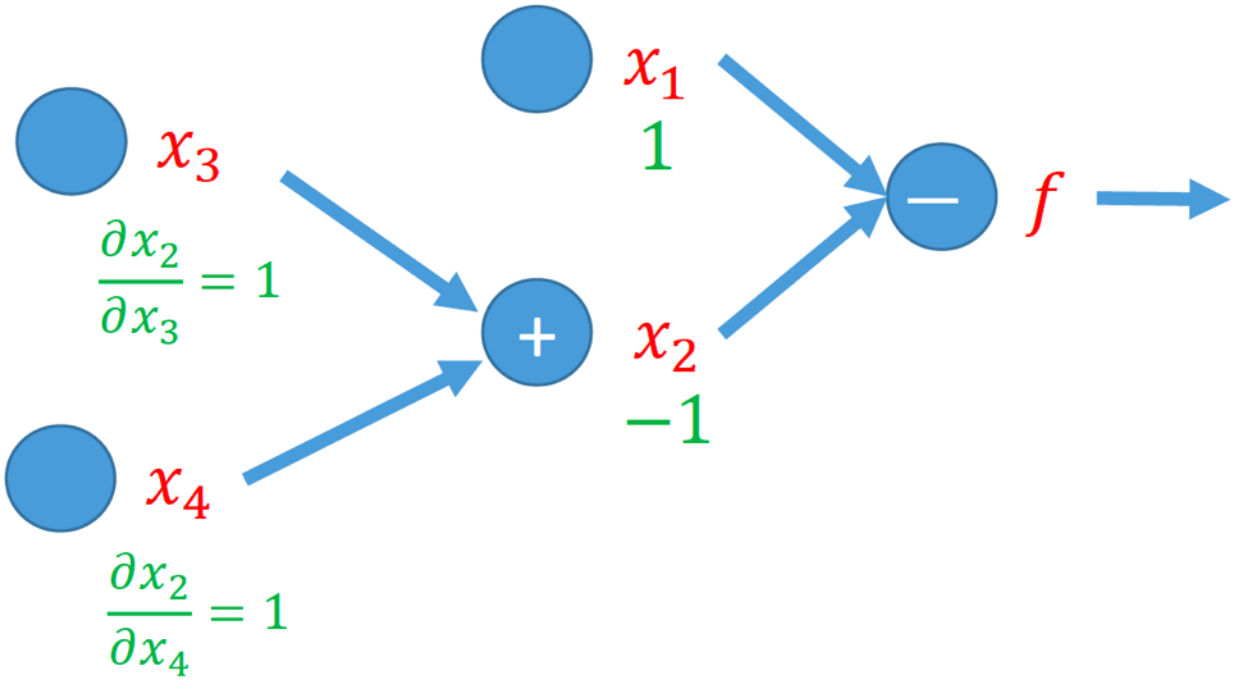


Function: $f = x_1 - x_2$

Gradient: $\frac{\partial f}{\partial x_1} = 1, \frac{\partial f}{\partial x_2} = -1$

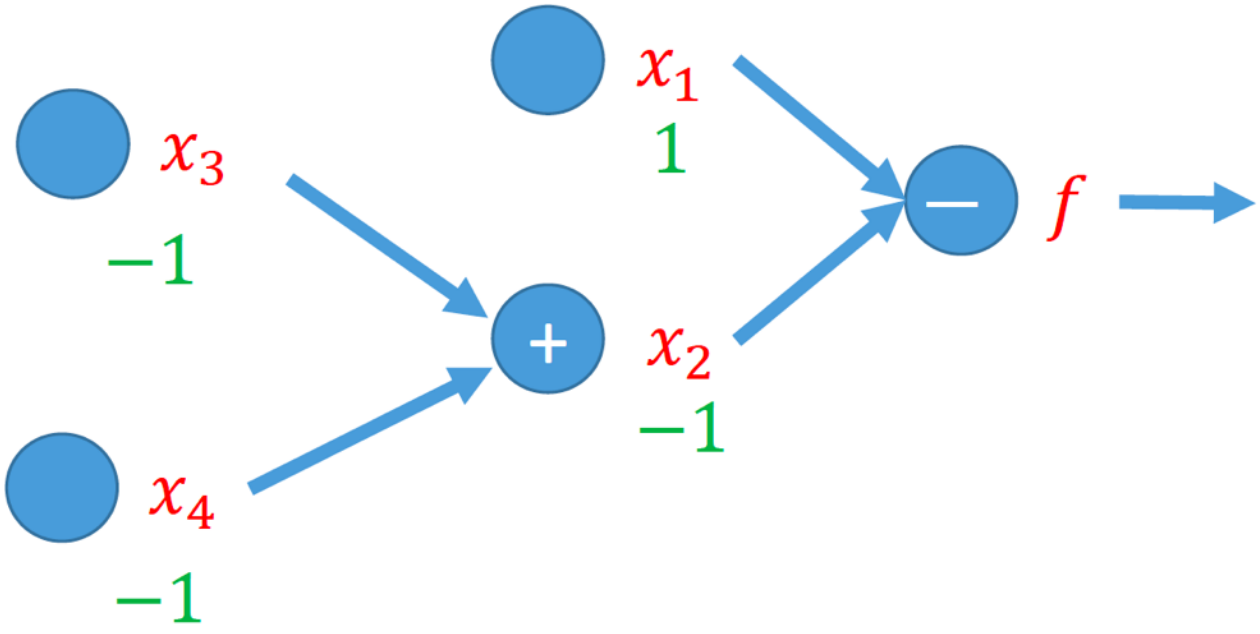


Function: $f = x_1 - x_2 = x_1 - (x_3 + x_4)$



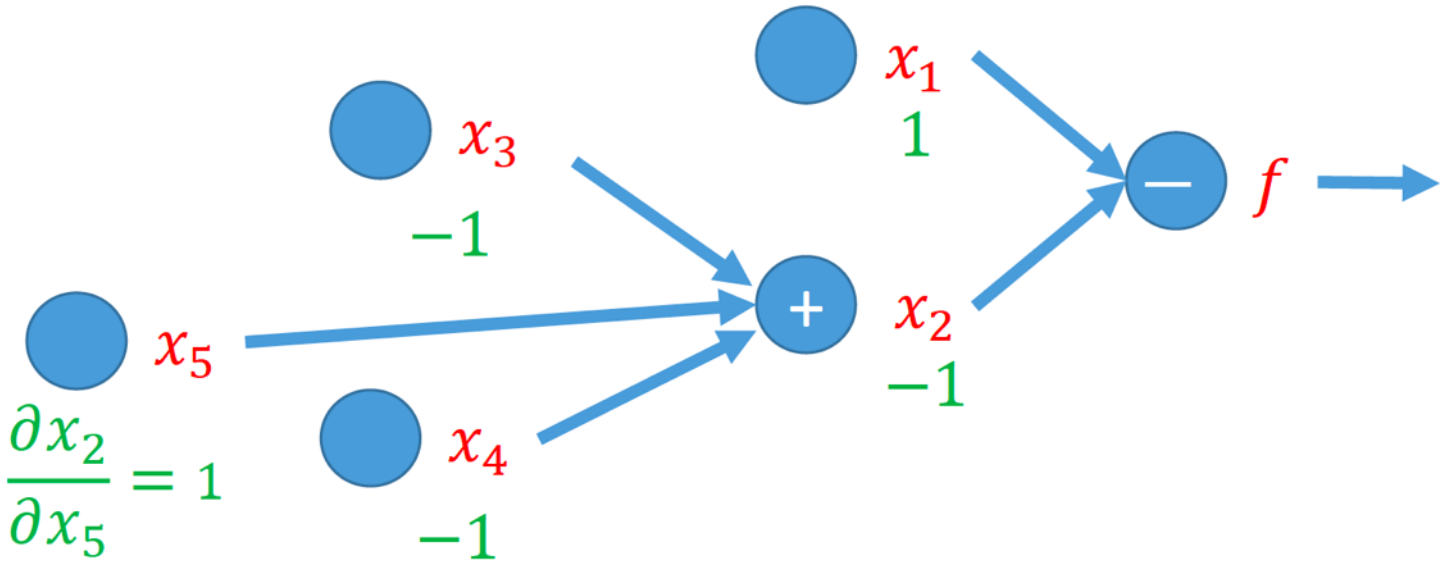
Function: $f = x_1 - x_2 = x_1 - (x_3 + x_4)$

Gradient: $\frac{\partial x_2}{\partial x_3} = 1, \frac{\partial x_2}{\partial x_4} = 1$. What about $\frac{\partial f}{\partial x_3}$?



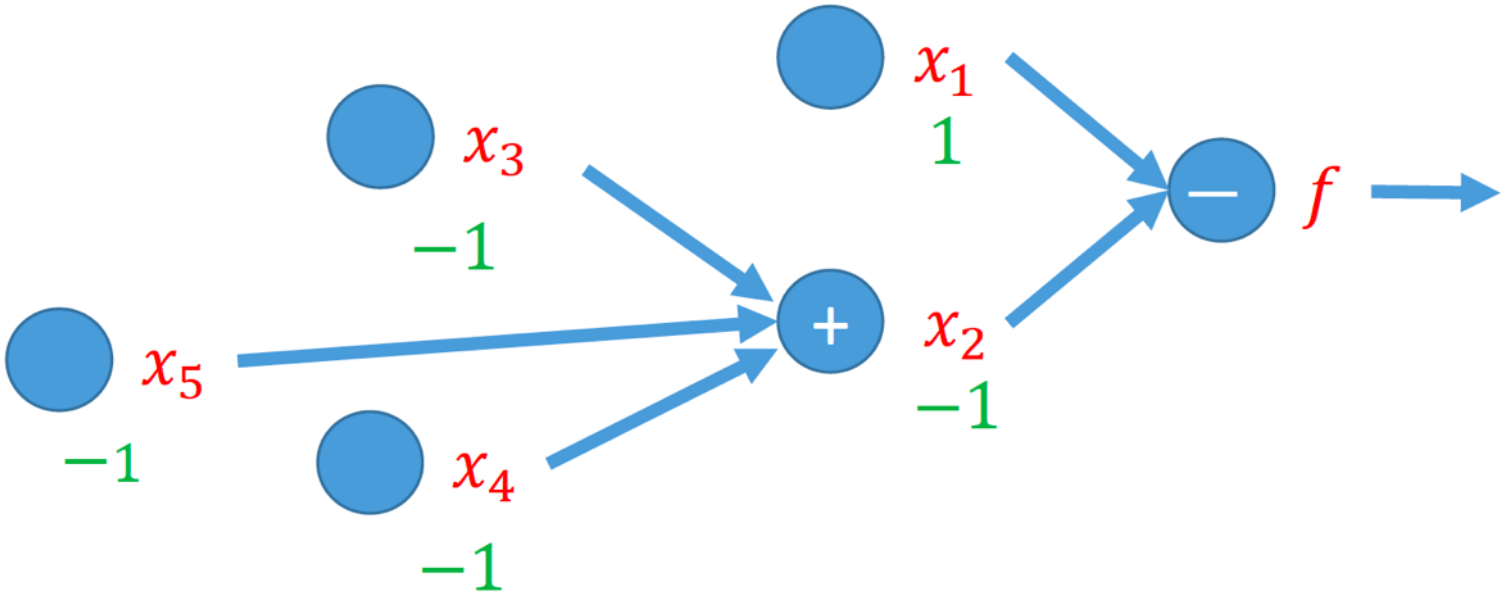
Function: $f = x_1 - x_2 = x_1 - (x_3 + x_4)$

Gradient: $\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial x_3} = -1$



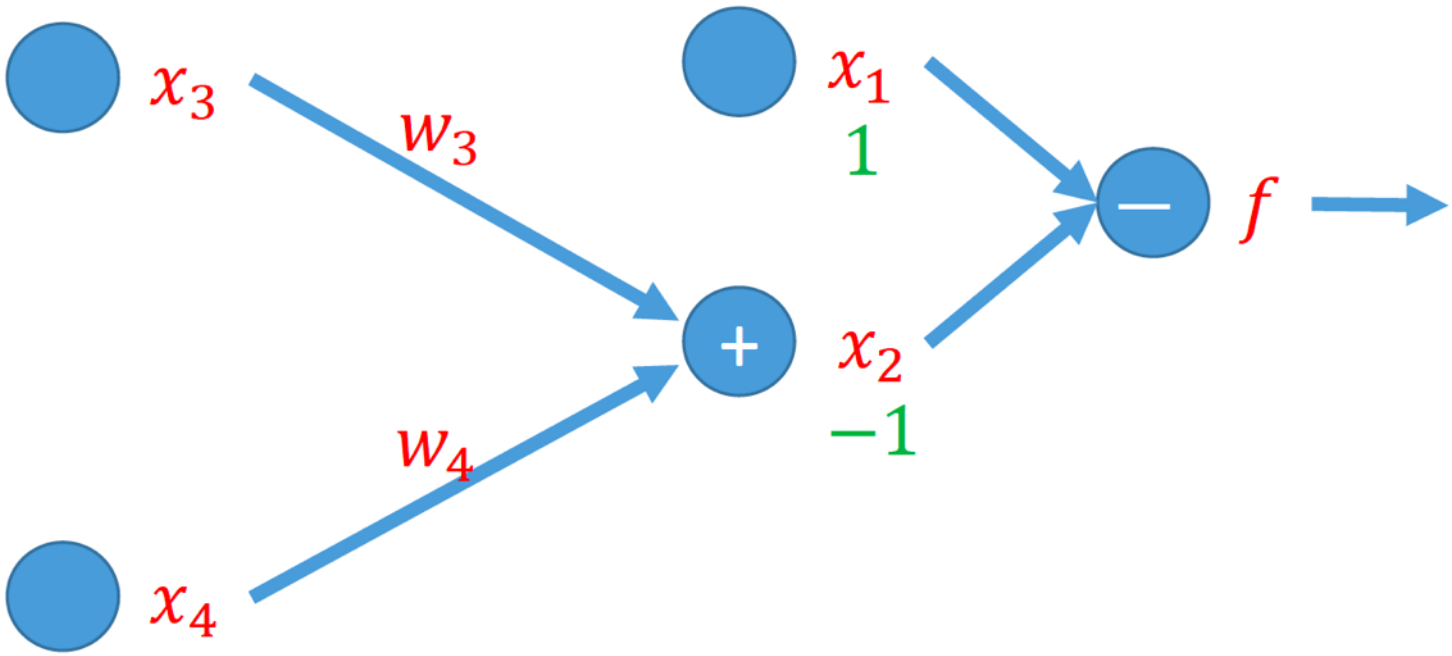
Function: $f = x_1 - x_2 = x_1 - (x_3 + x_5 + x_4)$

Gradient: $\frac{\partial x_2}{\partial x_5} = 1$

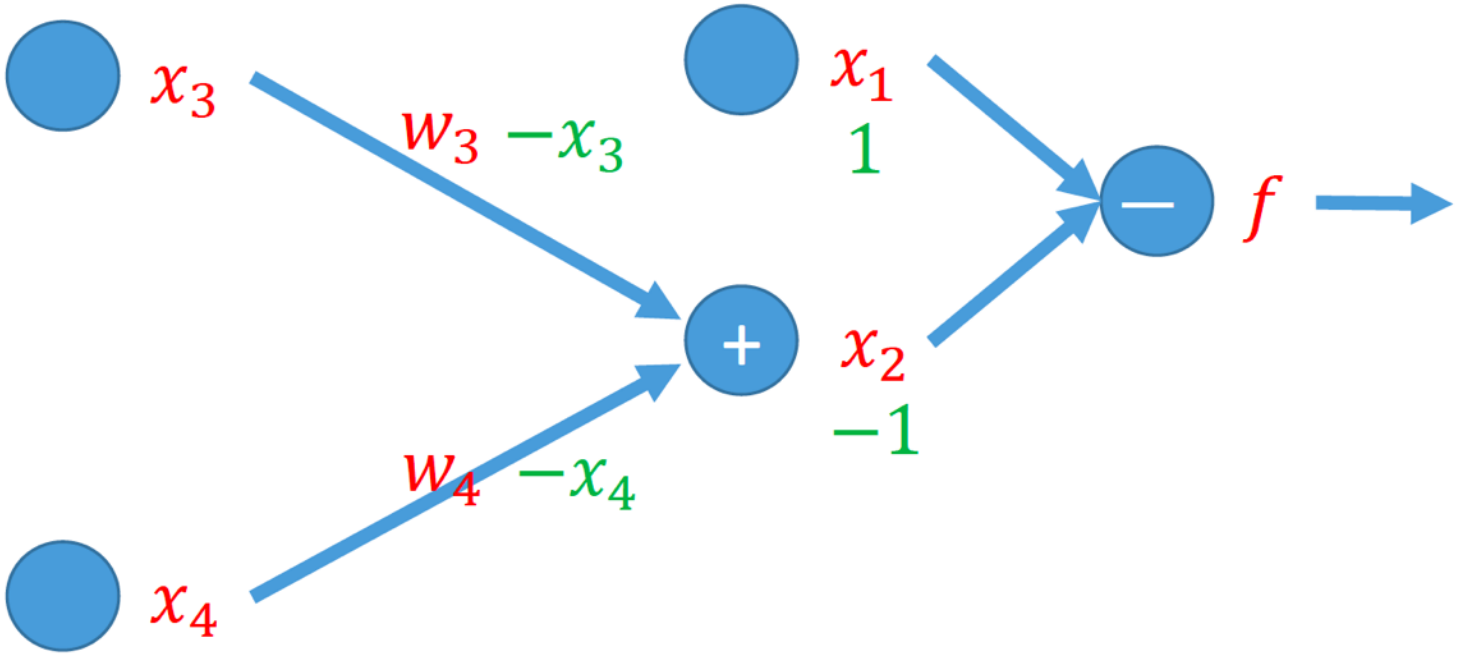


Function: $f = x_1 - x_2 = x_1 - (x_3 + x_5 + x_4)$

Gradient: $\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial x_5} = 1$

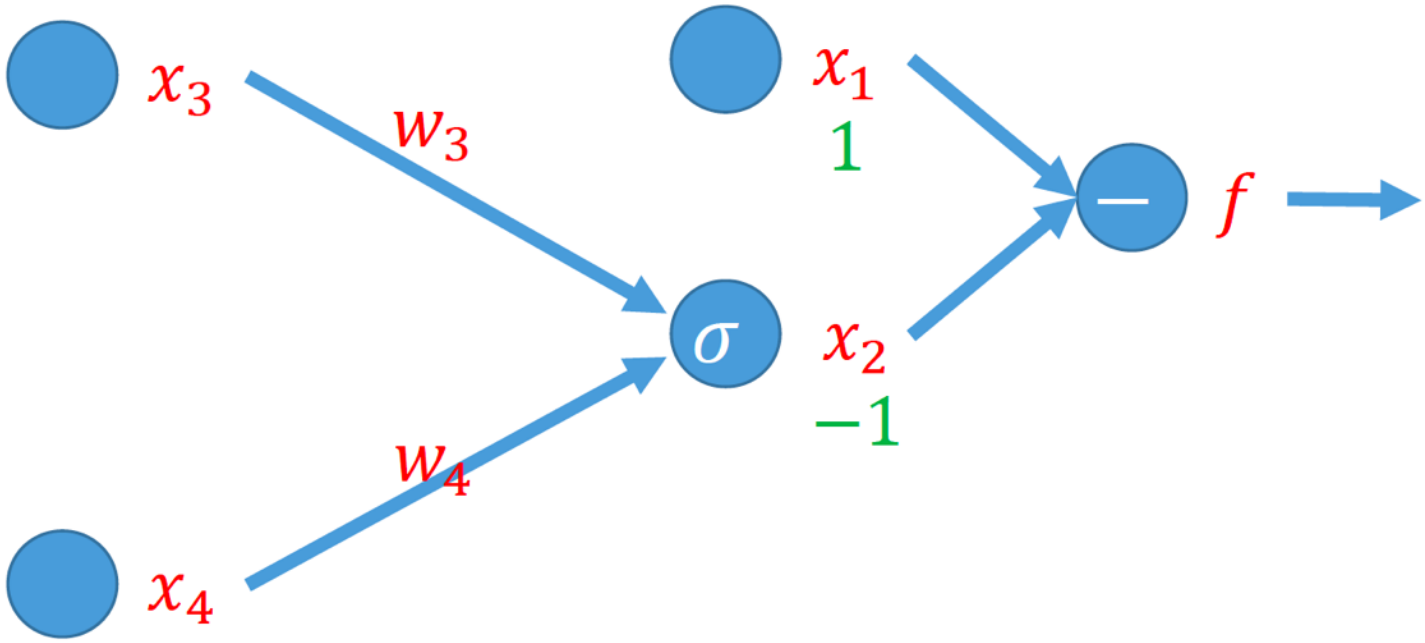


Function: $f = x_1 - x_2 = x_1 - (w_3 x_3 + w_4 x_4)$

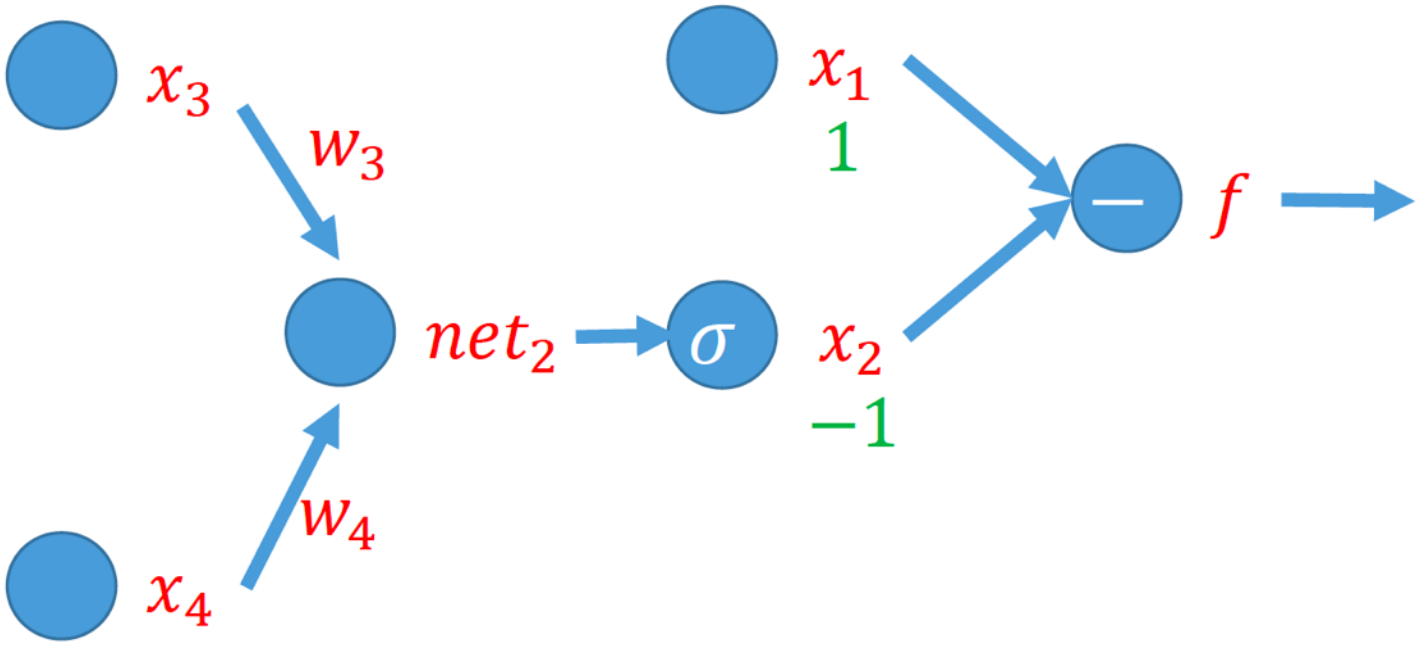


Function: $f = x_1 - x_2 = x_1 - (w_3 x_3 + w_4 x_4)$

Gradient: $\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial w_3} = -1 \times x_3 = -x_3$

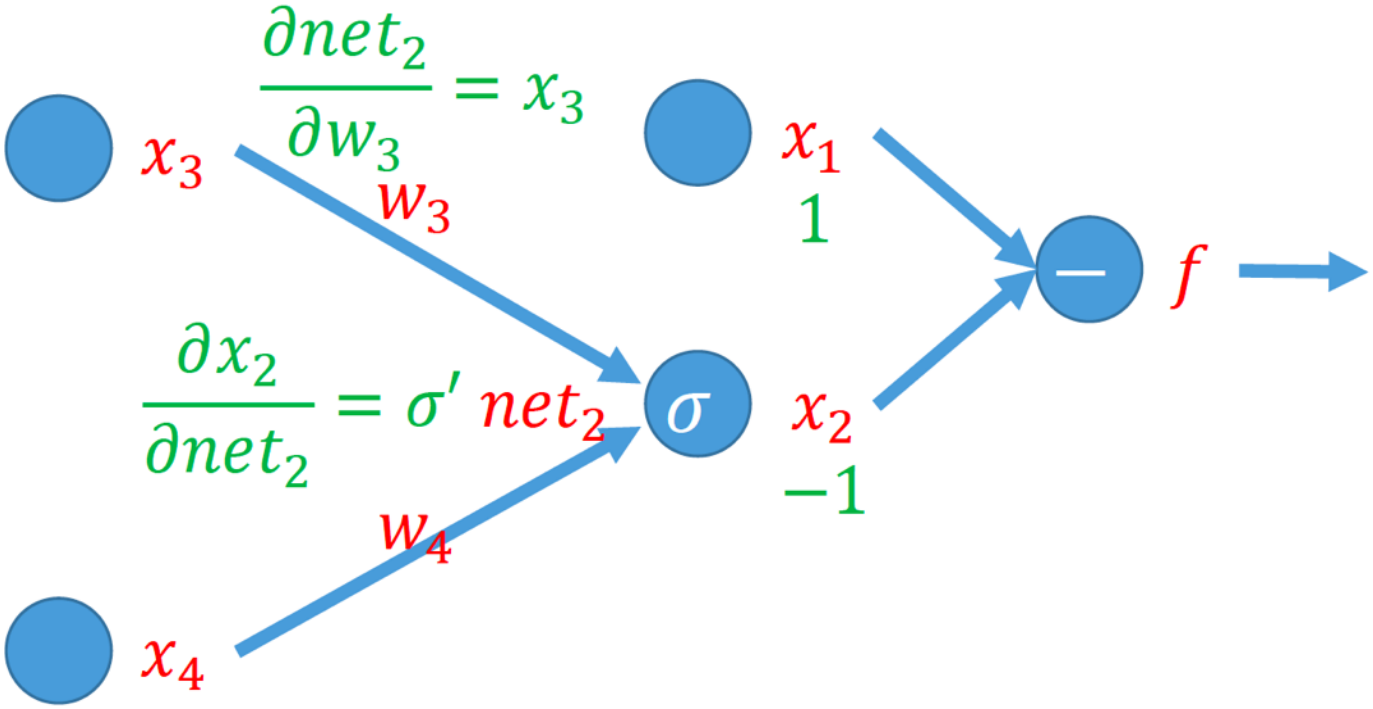


Function: $f = x_1 - x_2 = x_1 - \sigma(w_3 x_3 + w_4 x_4)$



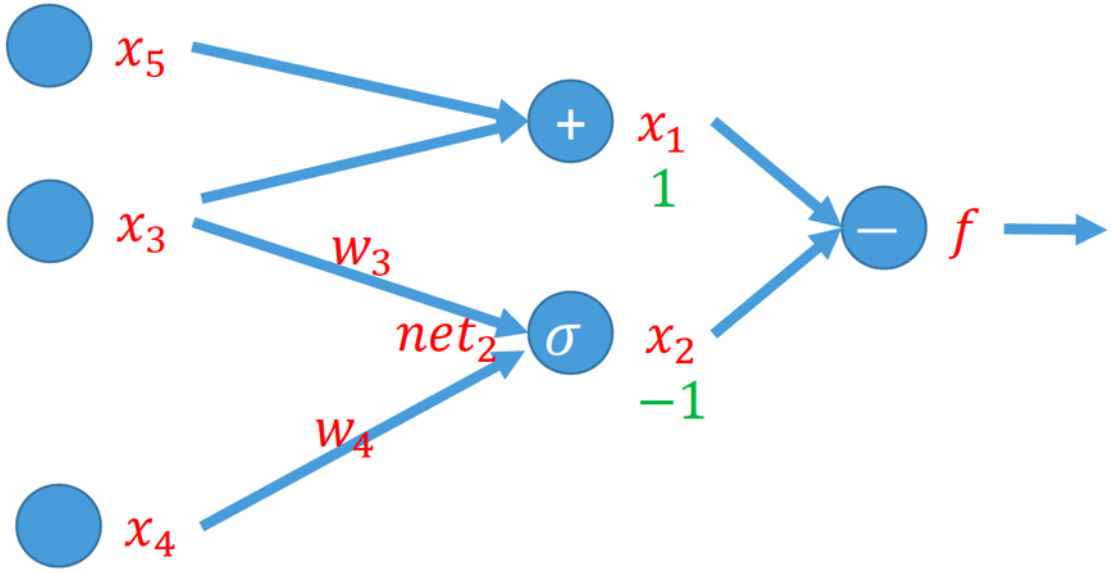
Function: $f = x_1 - x_2 = x_1 - \sigma(w_3 x_3 + w_4 x_4)$

Let $net_2 = w_3 x_3 + w_4 x_4$

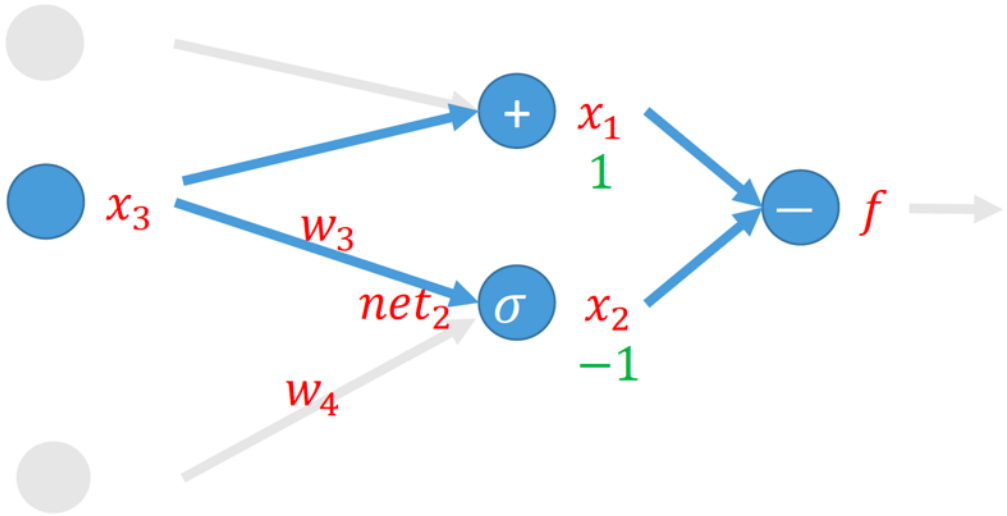


Function: $f = x_1 - x_2 = x_1 - \sigma(w_3 x_3 + w_4 x_4)$

Gradient: $\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial net_2}\frac{\partial net_2}{\partial w_3} = -1 \times \sigma' \times x_3 = -\sigma' x_3$



Function $f = x_1 - x_2 = (x_3 + x_5) - \sigma(w_3 x_3 + w_4 x_4)$

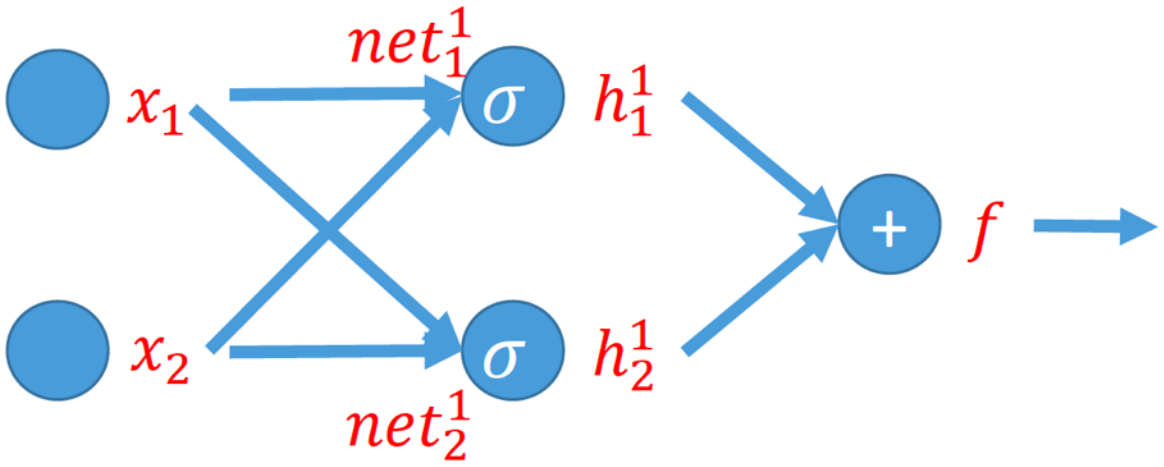


Function: $f = x_1 - x_2 = (x_3 + x_5) - \sigma(w_3 x_3 + w_4 x_4)$

Gradient: $\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial net_2}\frac{\partial net_2}{\partial x_3} + \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial x_3} = -1 \times \sigma' \times w_3 + 1 \times 1 = -\sigma' w_3 + 1$

Summary

- Forward to compute f
- Backward to compute the gradients



Activation Functions

- ReLU $\mathrm{ReLU}(x) = \max\{x, 0\}$

$$\mathrm{ReLU}'(x) = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

Sub-gradient used at $x = 0$

- Sigmoid $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$\sigma'(x) = \sigma(x) \cdot \big(1 - \sigma(x)\big)$$

- $\tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

Deep Neural Networks

Final linear layer

What does a deep net look like?

1. Pick weight matrices $W^{(1)}, W^{(2)}, \cdots, W^{(L)}, w$
2. Pick activation function σ
3. The neural net f predicts

$$f(x) = w^\top \sigma\Big(W^{(L)} \sigma\Big(W^{(L-1)} \cdots \sigma\Big(W^{(2)} \sigma\big(W^{(1)} x \big) \Big) \Big) \Big)$$

More Notations

$$f(x) = w^\top \sigma\Big(W^{(L)} \sigma\Big(W^{(L-1)} \cdots \sigma\Big(W^{(2)} \sigma\big(W^{(1)} x \big) \Big) \Big) \Big)$$

- Suppose $x \in \mathbb{R}^{d_0}$, $W^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$, and $w \in \mathbb{R}^{d_L}$
- Denote $z^{(0)} := x$, $a^{(i)} := W^{(i)} z^{(i-1)}$, and $z^{(i)} := \sigma(a^{(i)})$
- It's easy to see $a^{(i)}, z^{(i)} \in \mathbb{R}^{d_i}$
- Denote the j-th row of $W^{(i)}$ as $w^{(i;j:)} \in \mathbb{R}^{d_{i-1}}$
- Denote the j-th column of $W^{(i)}$ as $w^{(i;:j)} \in \mathbb{R}^{d_i}$

Backpropagation

What we have already learned

- $\frac{\partial f}{\partial w} = z^{(L)} \in \mathbb{R}^{d_L}$
- $\frac{\partial f}{\partial W^{(L)}} = \left(w \odot \sigma'(a^{(L)})\right) z^{(L-1)^\top}$
- Denote $\delta^{(L)} := \left(w \odot \sigma'(a^{(L)})\right)$, we have $\frac{\partial f}{\partial W^{(L)}} = \delta^{(L)} z^{(L-1)^\top}$

What about $\frac{\partial f}{\partial W^{(L-1)}}$?

Backpropagation

- $$\frac{\partial f}{\partial w^{(L-1;k:)}} = \frac{\partial f}{\partial z_k^{(L-1)}} \frac{\partial z_k^{(L-1)}}{\partial w^{(L-1;k:)}}$$
- $$\frac{\partial f}{\partial z_k^{(L-1)}} = w^{(L;:k)^\top} \left(w \odot \sigma'(a^{(L)}) \right) = w^{(L;:k)^\top} \delta^{(L)}$$
- $$\frac{\partial z_k^{(L-1)}}{\partial w^{(L-1;k:)}} = \sigma' \left(w^{(L-1;k:)^\top} z^{(L-2)} \right) z^{(L-2)^\top}$$
- $$\frac{\partial f}{\partial w^{(L-1;k:)}} = w^{(L;:k)^\top} \delta^{(L)} \, \sigma' \left(w^{(L-1;k:)^\top} z^{(L-2)} \right) z^{(L-2)^\top}$$

Backpropagation

- $$\frac{\partial f}{\partial W^{(L-1)}} = \left(\left(W^{(L)^\top} \delta^{(L)} \right) \odot \sigma'(a^{(L-1)}) \right) z^{(L-2)^\top}$$
- Denote

  $$\delta^{(L-1)} := \left(\left(W^{(L)^\top} \delta^{(L)} \right) \odot \sigma'(a^{(L-1)}) \right),$$

  we have $$\frac{\partial f}{\partial W^{(L-1)}} = \delta^{(L-1)} z^{(L-2)^\top}$$

Backpropagation

What we have already learned

- $\dfrac{\partial f}{\partial w} = z^{(L)} \in \mathbb{R}^{d_L}$
- $\dfrac{\partial f}{\partial W^{(L)}} = \left(w \odot \sigma'(a^{(L)})\right) z^{(L-1)^\top}$
- Denote $\delta^{(L)} := \left(w \odot \sigma'(a^{(L)})\right)$, we have $\dfrac{\partial f}{\partial W^{(L)}} = \delta^{(L)} z^{(L-1)^\top}$
- $\dfrac{\partial f}{\partial W^{(L-1)}} = \delta^{(L-1)} z^{(L-2)^\top}$

More Generally

- We have $\frac{\partial f}{\partial W^{(i)}} = \delta^{(i)} z^{(i-1)^\top}$
- where $\delta^{(L)} := \left(w \odot \sigma'(a^{(L)}) \right)$, and

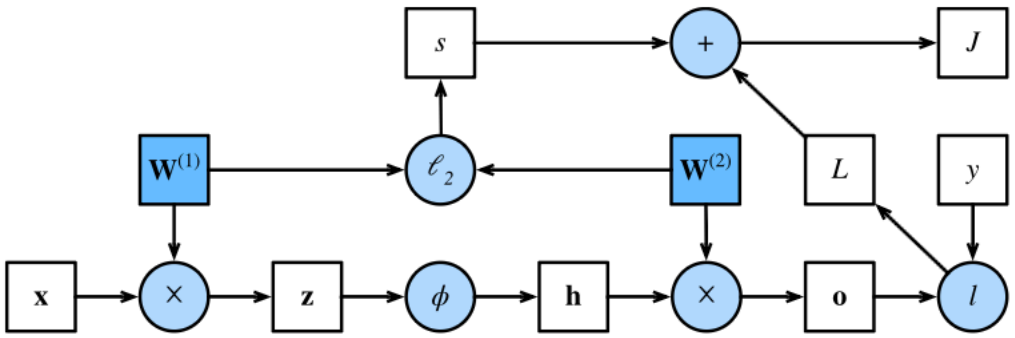
$$\delta^{(i-1)} := \left(\left(W^{(i)^\top} \delta^{(i)} \right) \odot \sigma'(a^{(i-1)}) \right)$$

People has to hand-code these back propagation updates in early days

Computation Graph

- Optimize

$$J = \ell(W^{(2)}\phi(W^{(1)}x), y) + \|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2$$




```
# Compute prediction and loss
pred = model(X)
loss = loss_fn(pred, y)

# Backpropagation
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

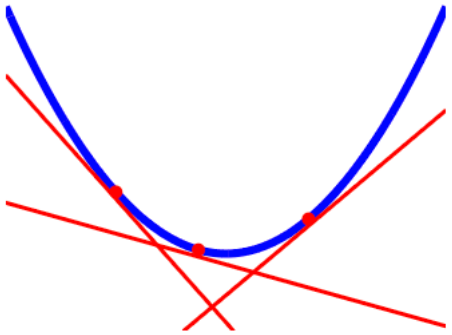
Computation Graph

- Forward pass:
  - Generate dynamic graph (at least for PyTorch)
  - Store a, z lists
- Backward pass:
  - loss.backward() calculate gradients
  - Release memory
- Inference: torch.no_grad()
  - No graph will be generated (and no need to store intermediate values)

Convex Optimization

- A (differentiable) function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex iff, for any $x_1, x_2 \in \mathbb{R}^d$, we have

$$f(x_1) \geq f(x_2) + \nabla f(x_2)^\top (x_1 - x_2)$$



Convex Optimization

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is L-smooth if for any $x_1, x_2 \in \mathbb{R}^d$

$$\|\nabla f(x_1) - \nabla f(x_2)\|_2 \leq L\|x_1 - x_2\|_2$$

- L-smooth implies that for any $x_1, x_2 \in \mathbb{R}^d$

$$f(x_1) \leq f(x_2) + \nabla f(x_2)^\top (x_1 - x_2) + \frac{L}{2}\|x_1 - x_2\|_2^2$$

GD for Convex Functions

- Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function that is also L-smooth
- Let $w^\star := \arg\min_w f(w)$ — the minimizer we try to find
- Consider the following gradient descent update

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

Step I: The Descend

- If $\eta \leq 1/L$, we then have

$$f(w_{t+1}) \leq f(w_t) - \frac{\eta}{2}\|\nabla f(w_t)\|_2^2$$

Function value decreases strictly unless $\nabla f(w_t) = 0$

Step II: Convergence Rate

We have

- $f(w_{t+1}) - f(w^\star) \leq \frac{1}{2\eta}\left(\|w_t - w^\star\|_2^2 - \|w_{t+1} - w^\star\|_2^2\right)$

- $f(w_T) - f(w^\star) \leq \frac{\|w_0 - w^\star\|_2^2}{2\eta T}$

  - To achieve ε error, we need $O\left(\frac{1}{\eta\varepsilon}\right) = O\left(\frac{L}{\varepsilon}\right)$ number of iterations

GD for Linear regression

Optimization becomes

$$f(w) = \frac{1}{2} \left\| y - Xw \right\|_2^2$$

- f is convex (actually quadratic)
- It's gradient equals to

$$\nabla f(w) = X^\top Xw - X^\top y$$

GD for Linear regression

- It's gradient equals to

$$\nabla f(w) = X^\top X w - X^\top y$$

- $\nabla f(w)$ is $\lambda_{\max}\left(X^\top X\right)$-Lipschitz

$$|\nabla f(w_1) - \nabla f(w_2)| = |X^\top X(w_1 - w_2)| \le \lambda_{\max}(X^\top X) \cdot \|w_1 - w_2\|_2$$

Select the learning rate as $\eta = 1/\lambda_{\max}(X^\top X)$ and run for $O\left(\lambda_{\max}(X^\top X)/\varepsilon\right)$ iterations ensures convergence up to an ε-optimal minimizer.

Only $O(\log(1/\varepsilon))$ iterations are needed if $\lambda_{\min}(X^\top X) > 0$, i.e., under strong convexity

Stochastic Gradient Descent

- The empirical risk minimization with m examples solves

$$\mathscr{L}(w) = \frac{1}{m} \sum_{i=1}^{m} \ell(y_i, g(x_i; w))$$

- This is a special case of minimizing avg of functions

$$\mathscr{L}(w) = \frac{1}{m} \sum_{i=1}^{m} f_i(w)$$

where $f_i(w) = \ell(y_i, g(x_i; w))$.

- Per-iteration cost is proportional to m.
- Question: Efficient ways to minimize avg of functions?

Stochastic Gradient Descent

Consider minimizing an average of functions

$$\min_x \frac{1}{m} \sum_{i=1}^{m} f_i(x)$$

Stochastic Gradient Descent

Consider minimizing an average of functions

$$\min_x \frac{1}{m} \sum_{i=1}^m f_i(x)$$

As $\nabla \sum_{i=1}^m f_i(x) = \sum_{i=1}^m \nabla f_i(x)$, gradient descent would repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m} \sum_{i=1}^m \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Stochastic Gradient Descent

Consider minimizing an average of functions

$$\min_x \frac{1}{m} \sum_{i=1}^m f_i(x)$$

As $\nabla \sum_{i=1}^m f_i(x) = \sum_{i=1}^m \nabla f_i(x)$, gradient descent would repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m} \sum_{i=1}^m \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

In comparison, stochastic gradient descent or SGD (or incremental gradient descent) repeats:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f_{i_k}(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration k

Two rules for choosing index i_k at iteration k:

- Randomized rule: choose $i_k \in \{1, \ldots, m\}$ uniformly at random
- Cyclic rule: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$

Two rules for choosing index i_k at iteration k:

- Randomized rule: choose $i_k \in \{1, \ldots, m\}$ uniformly at random
- Cyclic rule: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$

Randomized rule is more common in practice. For randomized rule, note that

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x)$$

so we can view SGD as using an unbiased estimate of the gradient at each step

Two rules for choosing index i_k at iteration k:

- Randomized rule: choose $i_k \in \{1, \ldots, m\}$ uniformly at random
- Cyclic rule: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$

Randomized rule is more common in practice. For randomized rule, note that

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x)$$

so we can view SGD as using an unbiased estimate of the gradient at each step

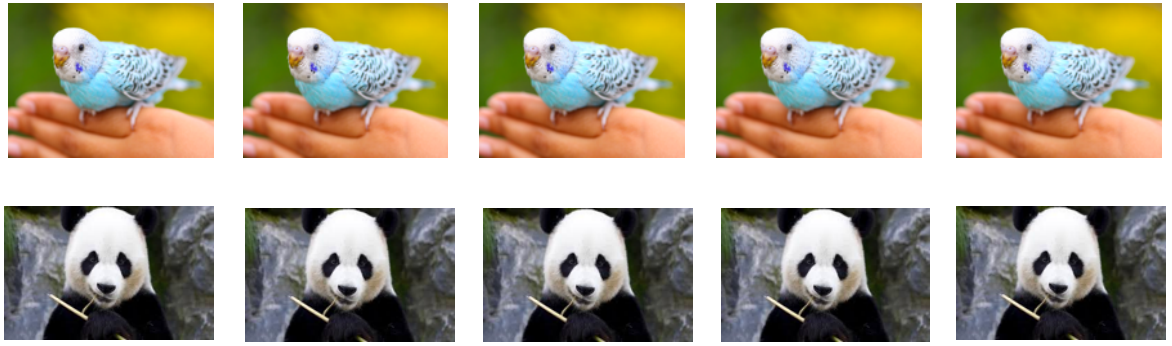
Main appeal of SGD:

- Iteration cost is independent of m (number of functions)
- Can also be a big savings in terms of memory useage

Intuition: Avoid Redundancy

- Example: Classify parrot vs monkey

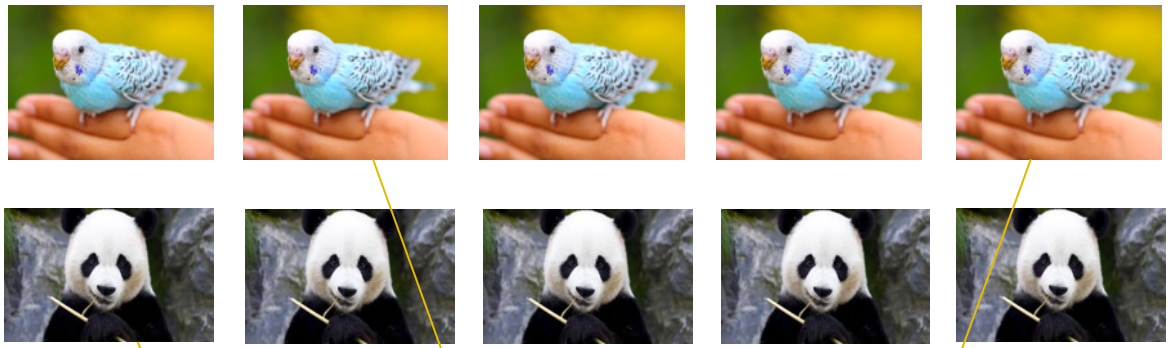
My Dataset



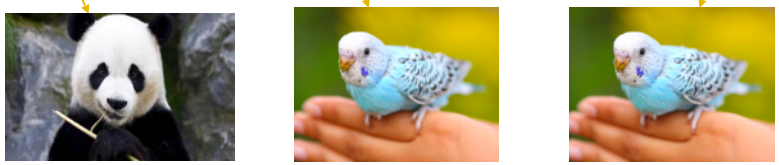
- Gradient descent: Use all 10 pictures
- ☹ These pictures are all duplicates
  - Duplicate gradients are redundant calculations

Making things efficient

My Dataset



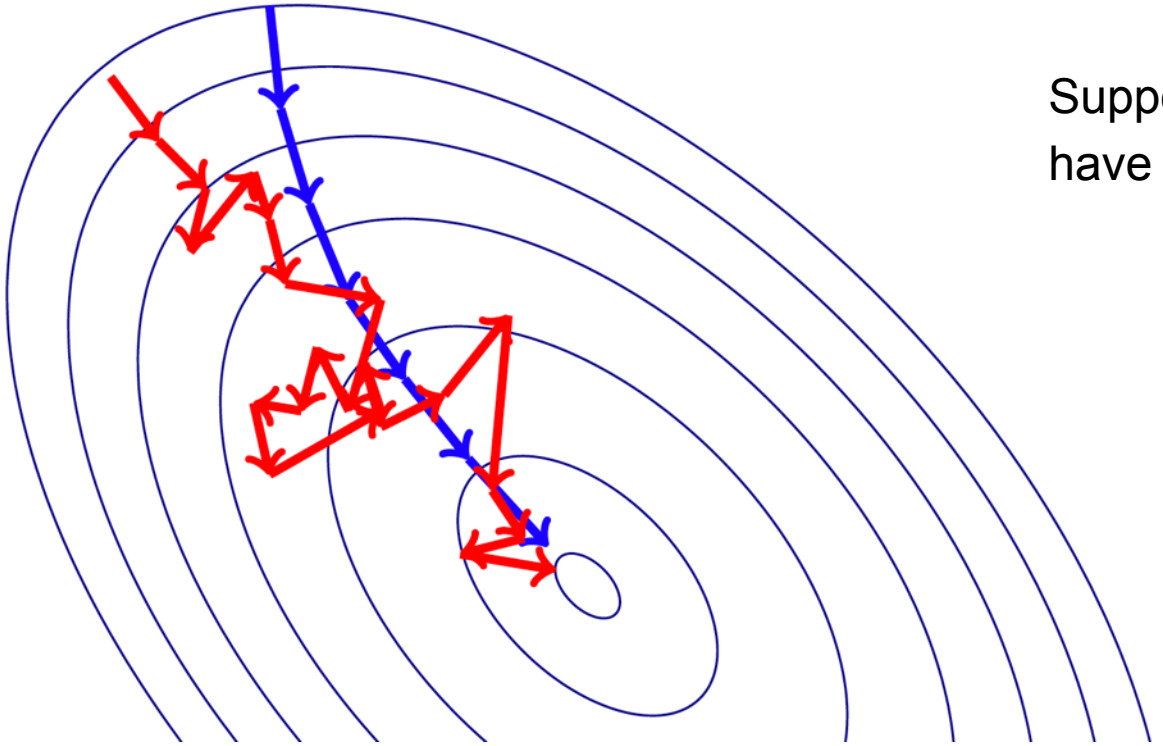
3 random samples



☺ These three samples contain all the info

Remark: In real data, examples are not duplicates but similar

Computation Cost



Suppose $x \in \mathbb{R}^d$ and we have m data points

- GD: $O(md)$
- SGD: $O(d)$

SGD can struggle when it gets close to the optimum

Mini-batch SGD

Also common is mini-batch stochastic gradient descent, where we choose a random subset $I_k \subseteq \{1, \ldots, m\}$, $|I_k| = b \ll m$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Mini-batch SGD

Also common is mini-batch stochastic gradient descent, where we choose a random subset $I_k \subseteq \{1, \ldots, m\}$, $|I_k| = b \ll m$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

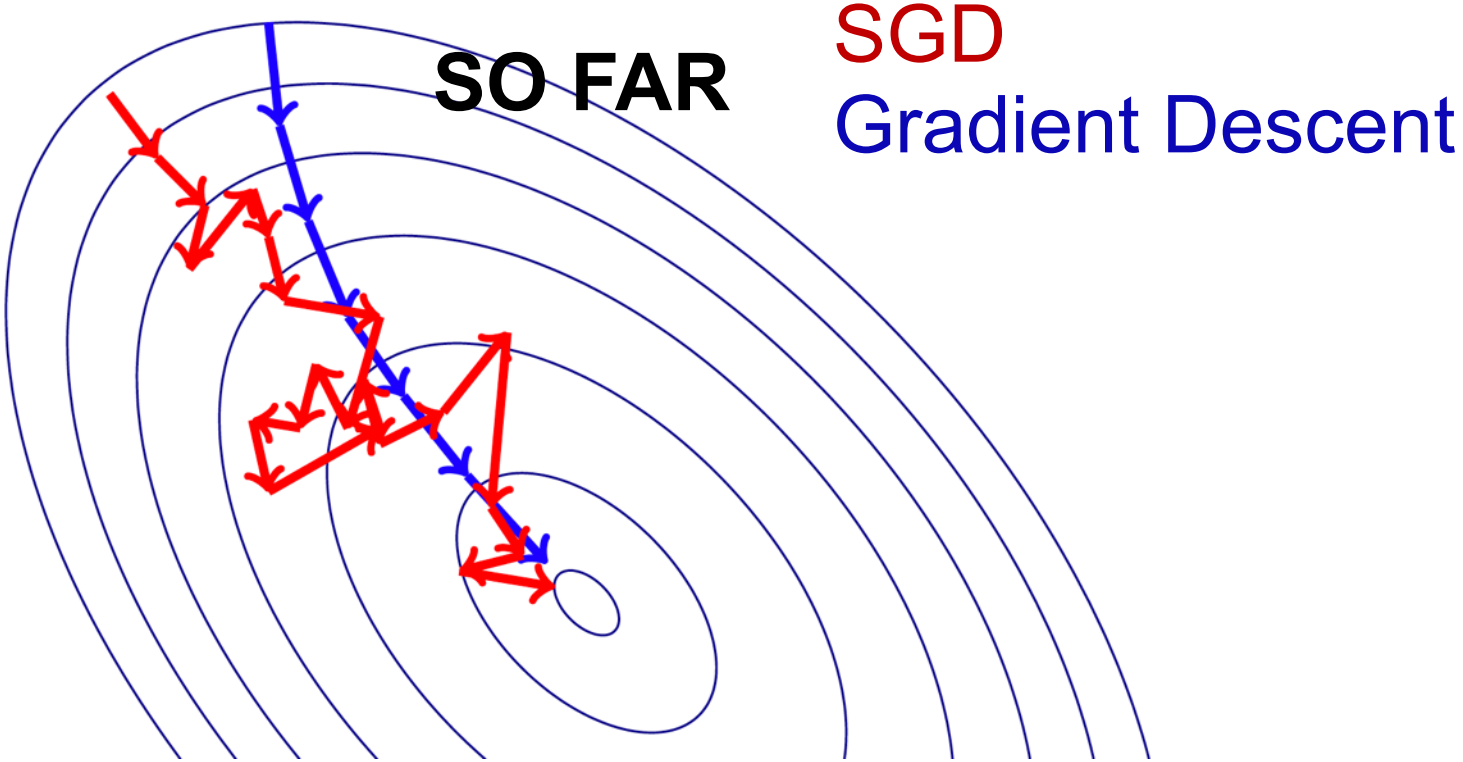
Again, we are approximating full gradient by an unbiased estimate:

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in I_k} \nabla f_i(x)\right] = \nabla f(x)$$

Computation Cost

- Suppose $x \in \mathbb{R}^d$, and we have m data points in total. Suppose the mini-match size is $b \in [m]$.
  - Gradient descent: $O(md)$
  - Stochastic gradient descent: $O(d)$
  - Mini-batch SGD: $O(bd)$

Going beyond SGD



How can we do better than minibatch SGD?

Optimization Algorithms

Many smart people are working on speeding up stuff!

Key ideas:

- Making the optimizer adapt to the data
- Take advantage of previous calculations

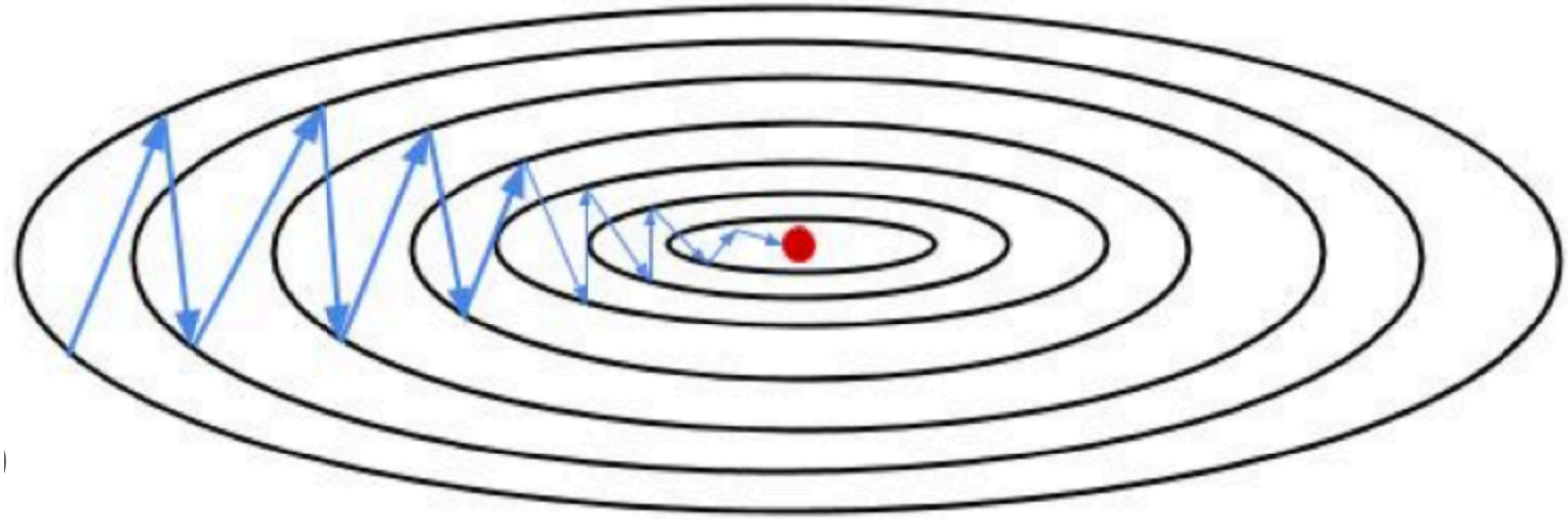
Optimizers:

- Momentum
- Adaptive Methods (AdaGrad, RMSProp, Adam)

Momentum

- The Momentum method is a method to accelerate learning using SGD

- In particular SGD suffers in the following scenarios:
  - Error surface has asymmetric curvature
  - The gradients are very noisy



(Stochastic) gradient descent can exhibit Zigzag behavior

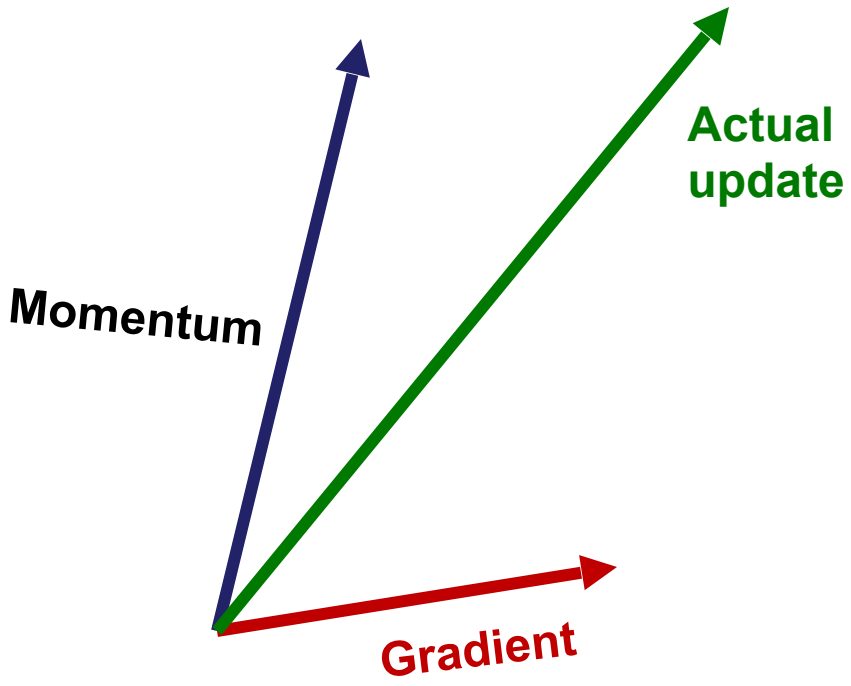
Momentum

- How do we solve this problem?
- Introduce a new variable m_t

- Think of m_t as the direction and speed by which the parameters (weights) move as the learning dynamics progresses

- Choose parameter $\beta \in (0,1)$. Denote $g_t = \nabla f_t$
  - $m_t = \beta m_{t-1} + g_{t-1}$
  - $\theta_t = \theta_{t-1} - \eta m_t$

Effective learning rate $\frac{\eta}{1-\beta}$


Actual
update
Momentum
Gradient

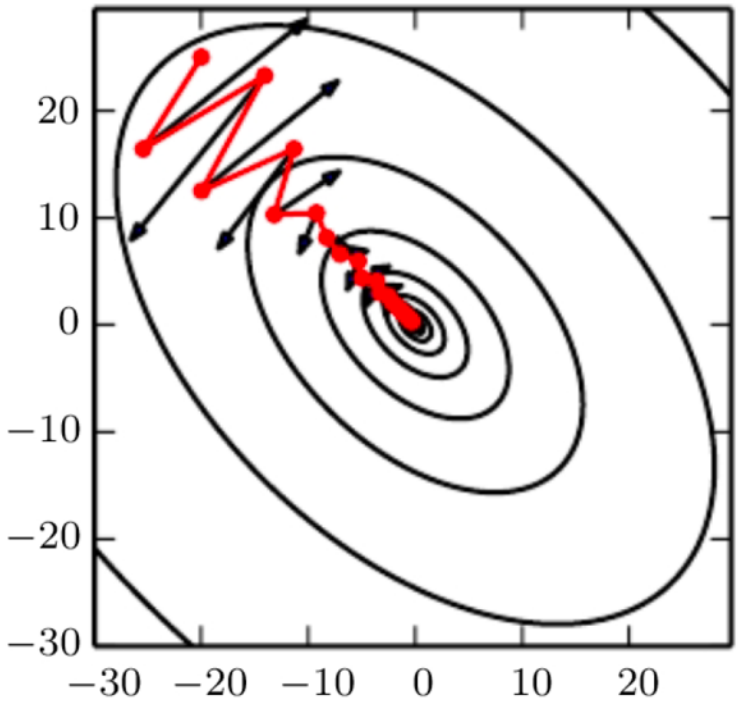
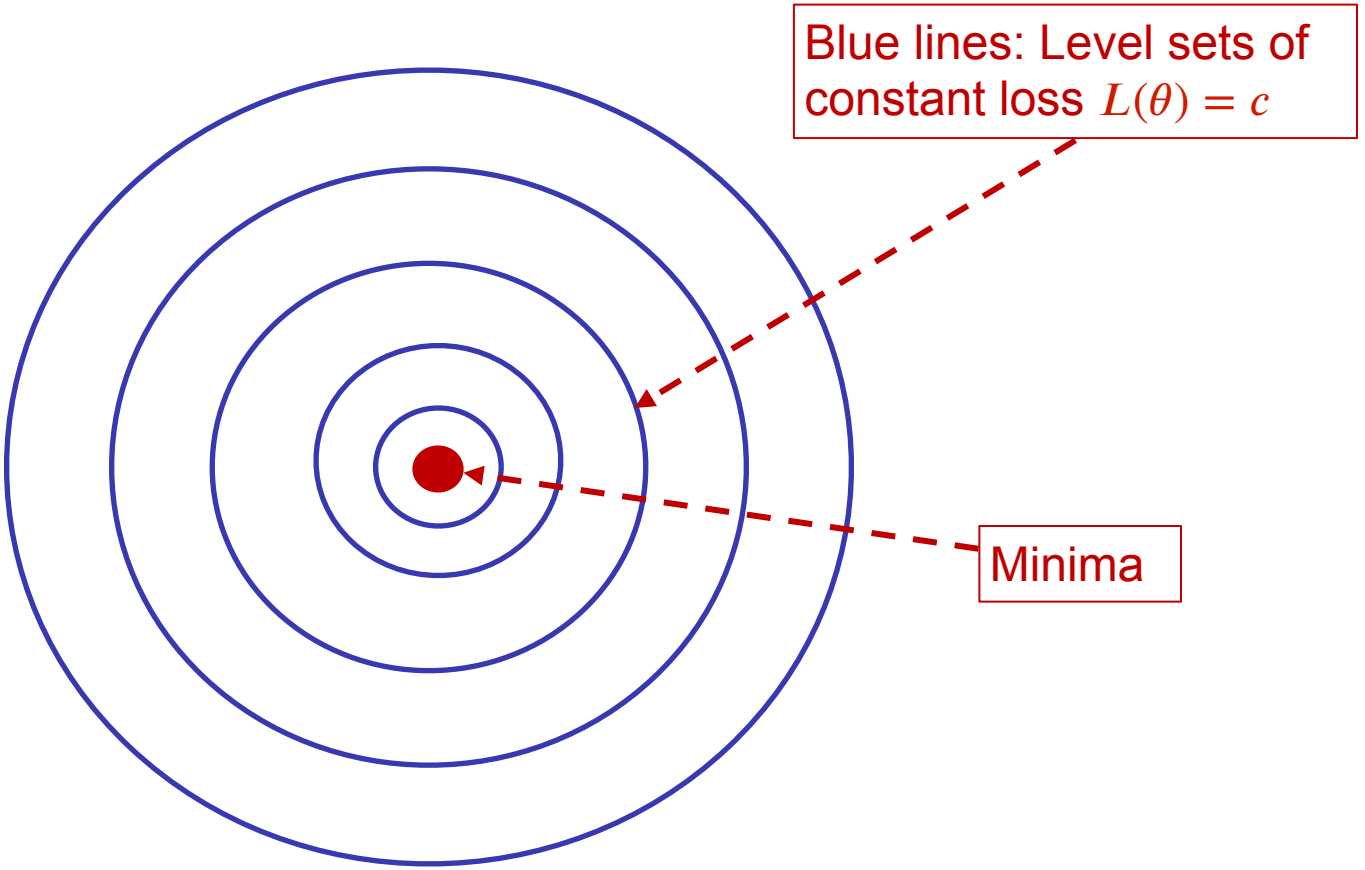


Illustration of how momentum traverses such an error surface better compared to Gradient Descent

Motivation

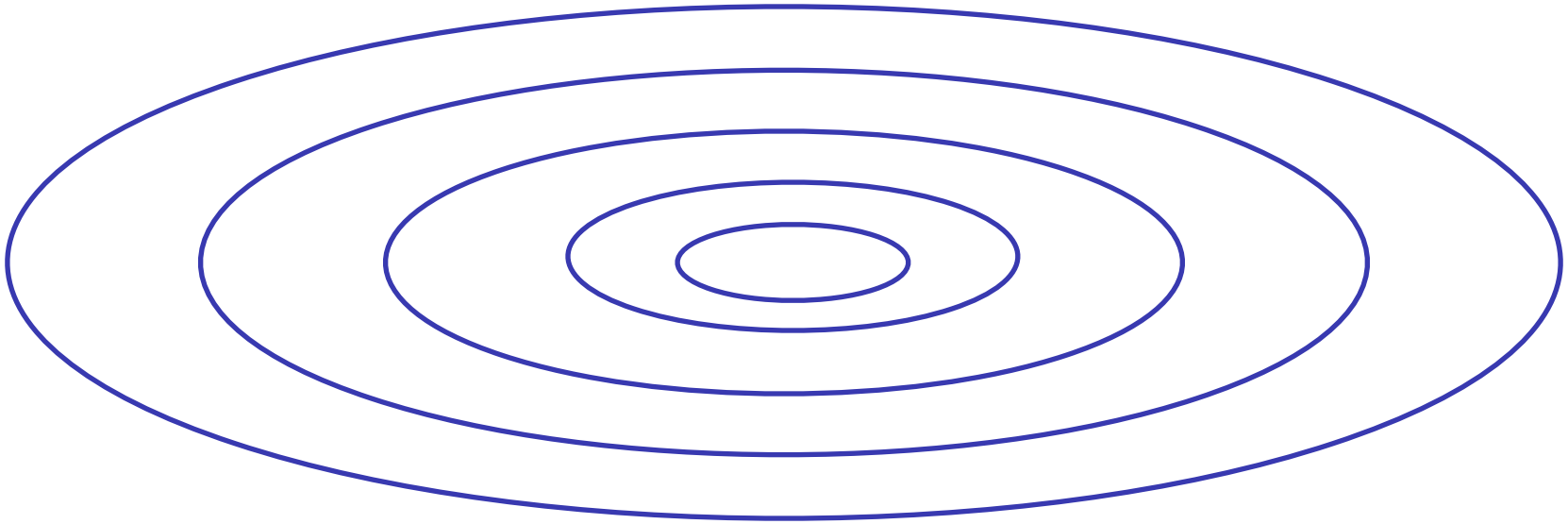
- Till now we assign the same learning rate to all features
- If the features vary in importance and frequency, why is this a good idea?
- It's probably not!

Loss landscape



Nice (all features are equally important)

Loss landscape



Not as nice!
Wish to have small learning rate on y axis
but large learning rate on x axis

AdaGrad (Adaptive Gradient)

- **Idea:** Downscale a model parameter by square-root of sum of squares of all its historical values

- Parameters that have large partial derivative of the loss: learning rates for them are rapidly declined

AdaGrad (Adaptive Gradient)

- For any $k \in [d]$, set $s_{t,k} = \sum_{i=1}^{t} g_{i,k}^2$
- Update $\theta_{t+1,k} = \theta_{t,k} - \eta \frac{1}{\sqrt{s_{t,k} + \varepsilon}} g_{t,k}$
- In matrix format:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{s_t + \varepsilon}} \odot g_t$$

RMSProp

- AdaGrad is good when the objective is convex.
- AdaGrad might shrink the learning rate too aggressively, we want to keep the history in mind
- We can adapt it to perform better in non-convex settings by accumulating an exponentially decaying average of the gradient
- Currently has about 7000+ citations on scholar, but was proposed **in a slide** in Geoffrey Hinton's Coursera course

RMSProp

- Set $s_{t,k} = \beta s_{t-1,k} + (1-\beta) g_{t-1,k}^2$

Exponentially weighted moving average

- Update $\theta_{t+1,k} = \theta_{t,k} - \eta \frac{1}{\sqrt{s_{t,k} + \varepsilon}} g_{t,k}$
- In matrix format:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{s_t + \varepsilon}} \odot g_t$$

Adam: ADAptive Moments

- We could have used RMSProp with momentum
- Use of Momentum with rescaling is not well motivated
- Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments

Adam: A method for stochastic optimization

DP Kingma, J Ba - arXiv preprint arXiv:1412.6980, 2014 - arxiv.org Paperpile

… **Adam** works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss AdaMax, a variant of **Adam** … Overall, we show that **Adam** is a versatile …

☆ Save 99 Cite Cited by 157542 Related articles All 27 versions Import into BibTeX »

Adam

- Set two parameters $\beta_1, \beta_2 \in (0,1)$
- Set
  - $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ Momentum
  - $s_t = \beta_2 s_{t-1} + (1-\beta_2) g_t^2$ Adaptive gradient
- Update $\theta_{t+1,k} = \theta_{t,k} - \eta \frac{1}{\sqrt{s_{t,k}} + \varepsilon} m_{t,k}$ for any $k \in [d]$
- In matrix format:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{\sqrt{s_t} + \varepsilon} \odot m_t$$

Acknowledgements

- A lot of content from online resources
  - Yingyu Liang's slides https://www.cs.princeton.edu/courses/archive/spring16/cos495/
  - Fei Fei Li's slides http://cs231n.stanford.edu/slides/2019/
  - Goodfellow's slides https://www.deeplearningbook.org/