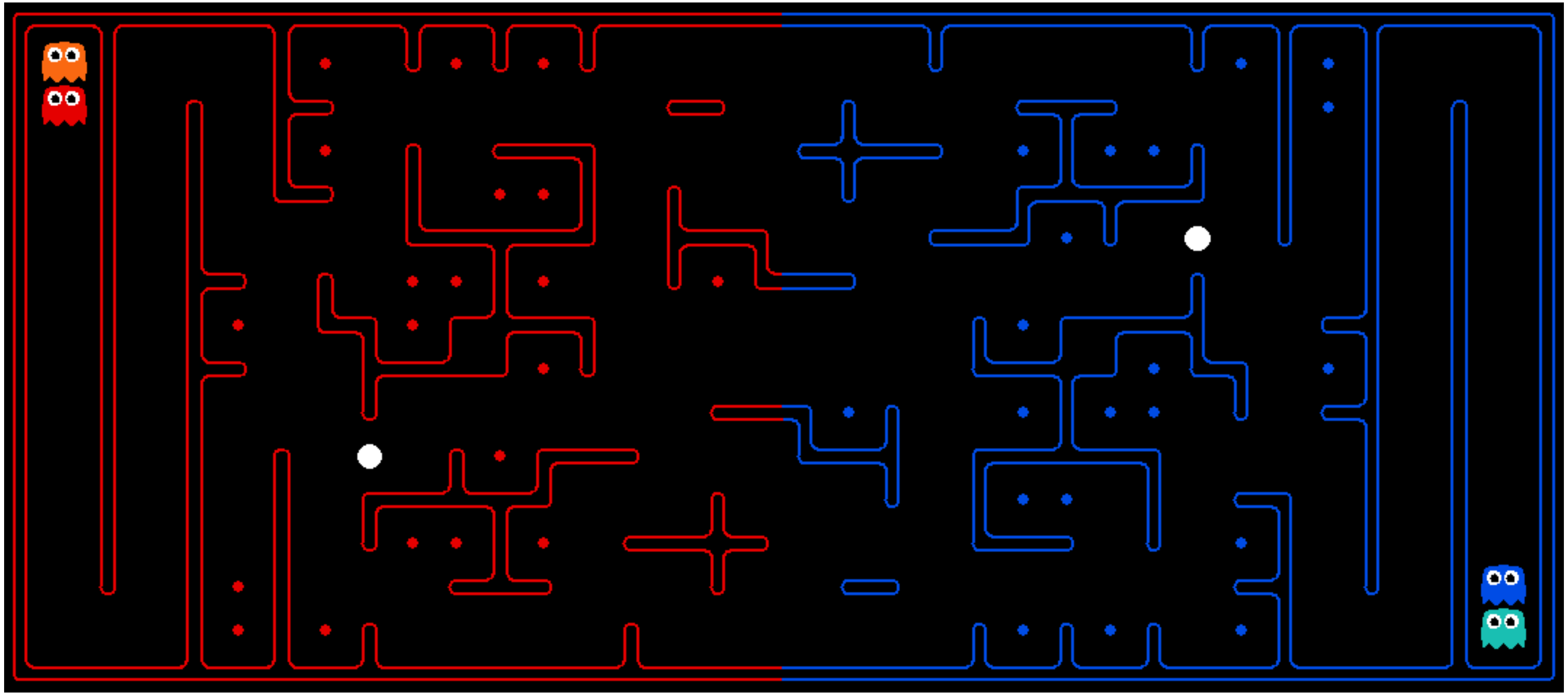
Announcements

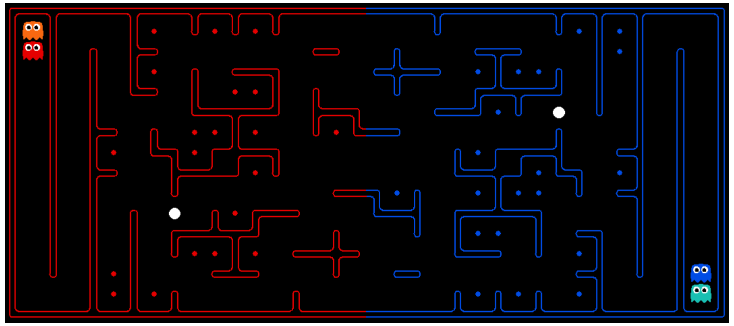
- Project 5 Ghostbusters
  - Due tomorrow/Friday 4/15 at 5pm
- Homework 9
  - Released soon, due Tuesday 4/19 at 11:59pm
- Cal Day – Robot Learning Lab Open House
  - Saturday 10am-1pm
  - 3rd floor Sutardja Dai Hall
  - Robot demos of knot tying, high-fives, fist-pumps, hugs

Final Contest!



- Challenges: Long term strategy, multiple agents, adversarial utilities, uncertainty about other agents' positions, plans, etc.

Final Contest Extra Credit



- Extra Credit Based on Final Ranking (deadline 4/24 11:59pm)
  - 1st place: 2 points on the final
  - 2nd and 3rd place: 1.5 points on the final
  - 4th to 10th place: 1 point on the final
  - 0.25 points on the final per staff bot beaten in the final ranking.

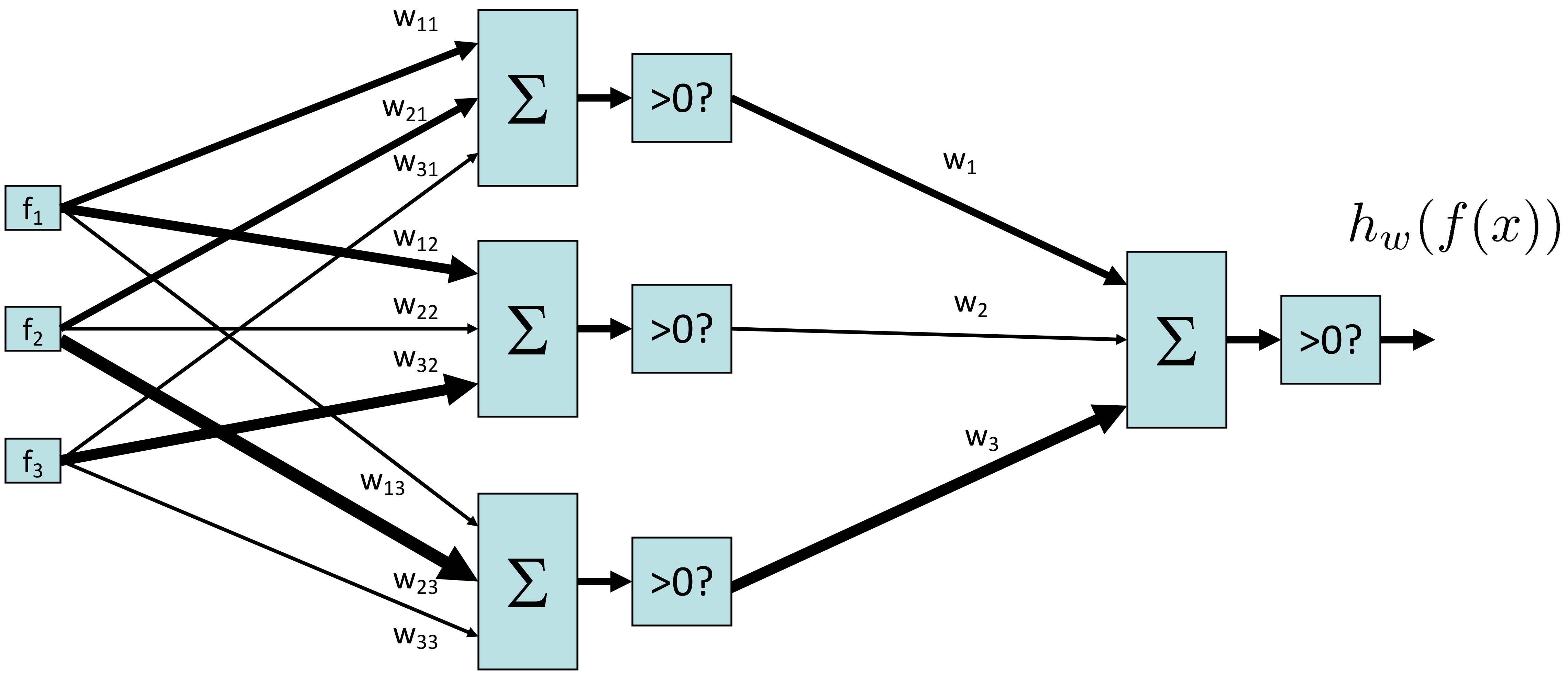
- In addition, every night from 4/17 through 4/24 at 11:59pm we will temporarily close submissions to compute an intermediate ranking, and there is also extra credit:
  - Top 20: 0.5 points on the final
  - Top 10: 1 point on the final

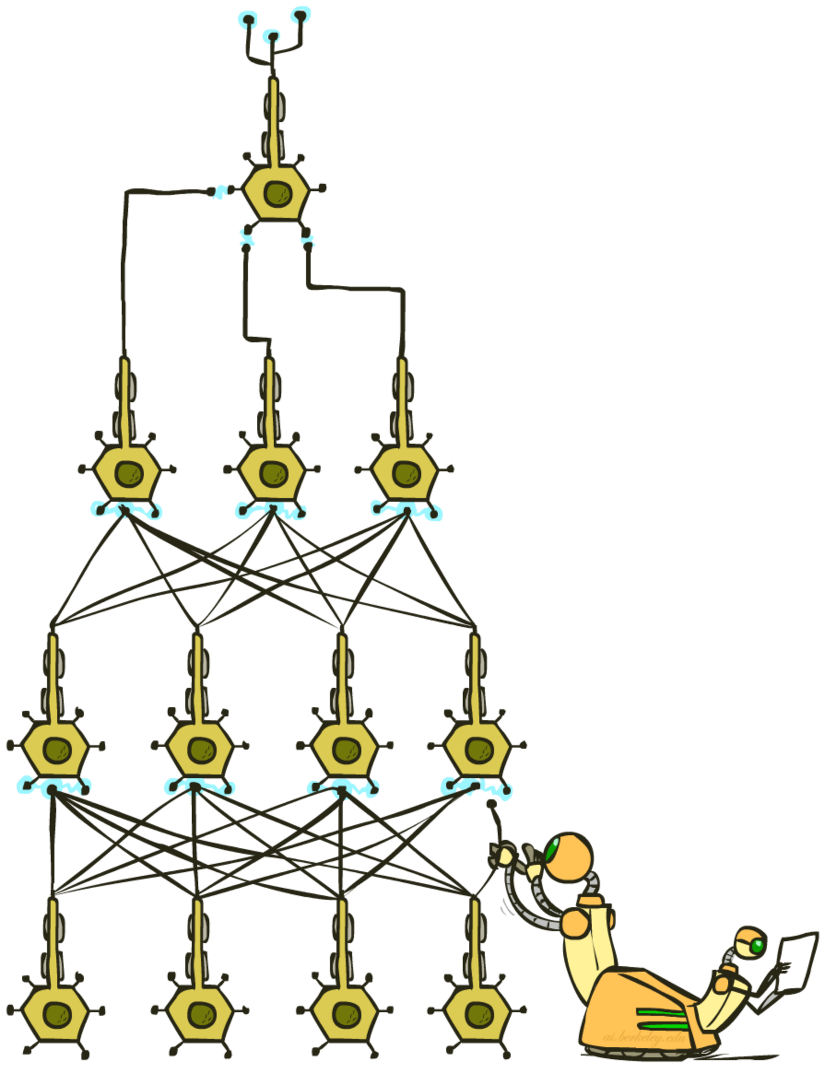
- Note that the ranking rewards are not cumulative, you will only get points for the highest ranking category you achieve during the contest.

Announcements

- Project 6:
  - Out Thursday assuming all goes well.
- HW10: Out today.
- Final Contest after Thanksgiving Break
- Today: Last “technical” lecture.
  - Remaining lectures will be applications.
  - 11/17: Vision (Alexei Efros)
  - 11/22: Natural Language Processing (Adam)
  - 11/29: Deep Robotics (Michael)
- My office hours this week: Election discussion (and anything else, of course)
- Adam’s office hours this week: Deep Learning Frameworks

Two-Layer Perceptron Network





CS 188: Artificial Intelligence

Deep Learning II

Instructors: Adam Janin & Josh Hug --- University of California, Berkeley

[These slides were created by Josh Hug, Dan Klein, Pieter Abbeel, Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

Local Search

- Simple, general idea:
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit
  - Neighbors = small perturbations of w

- Properties
  - Plateaus and local optima

→How to escape plateaus and find a good local optimum?

→How to deal with very large parameter vectors? E.g., $w \in \mathbb{R}^{1billion}$

Loss Functions

- Measurement: Zero-One Loss

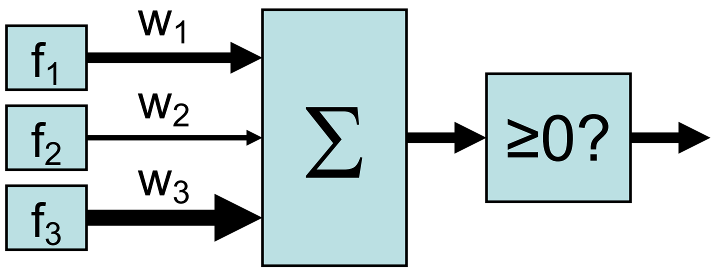
$$l^{\text{acc}}(w) = \frac{1}{m}\sum_{i=1}^{m}\left(\text{sign}(w^\top f(x^{(i)})) == y^{(i)}\right)$$

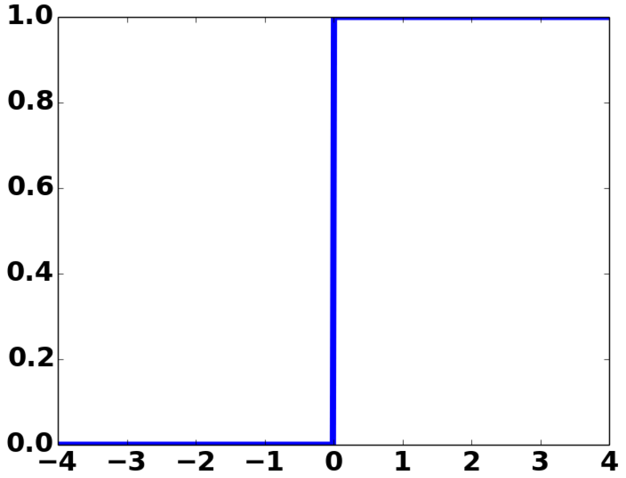
- Zero-one Loss isn't smooth:
  - Small changes to weight vector have no effect on loss.
  - Hard to optimize.

Note: $w^T f(x) = w \cdot f(x)$, both notations are common.

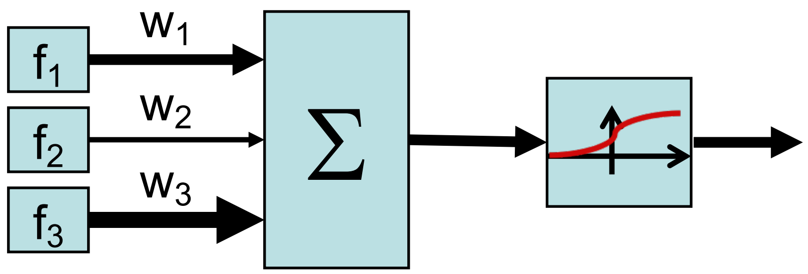
Softmax Activation Function

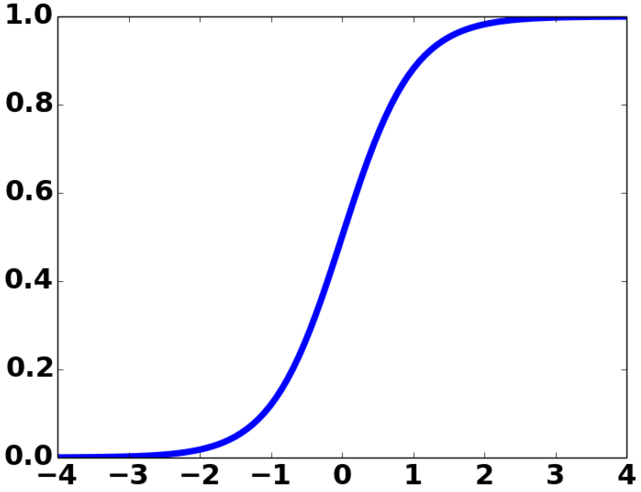
- Step Function: Outputs 1 if dot product $w^T f(x)$ is ≥ 0.



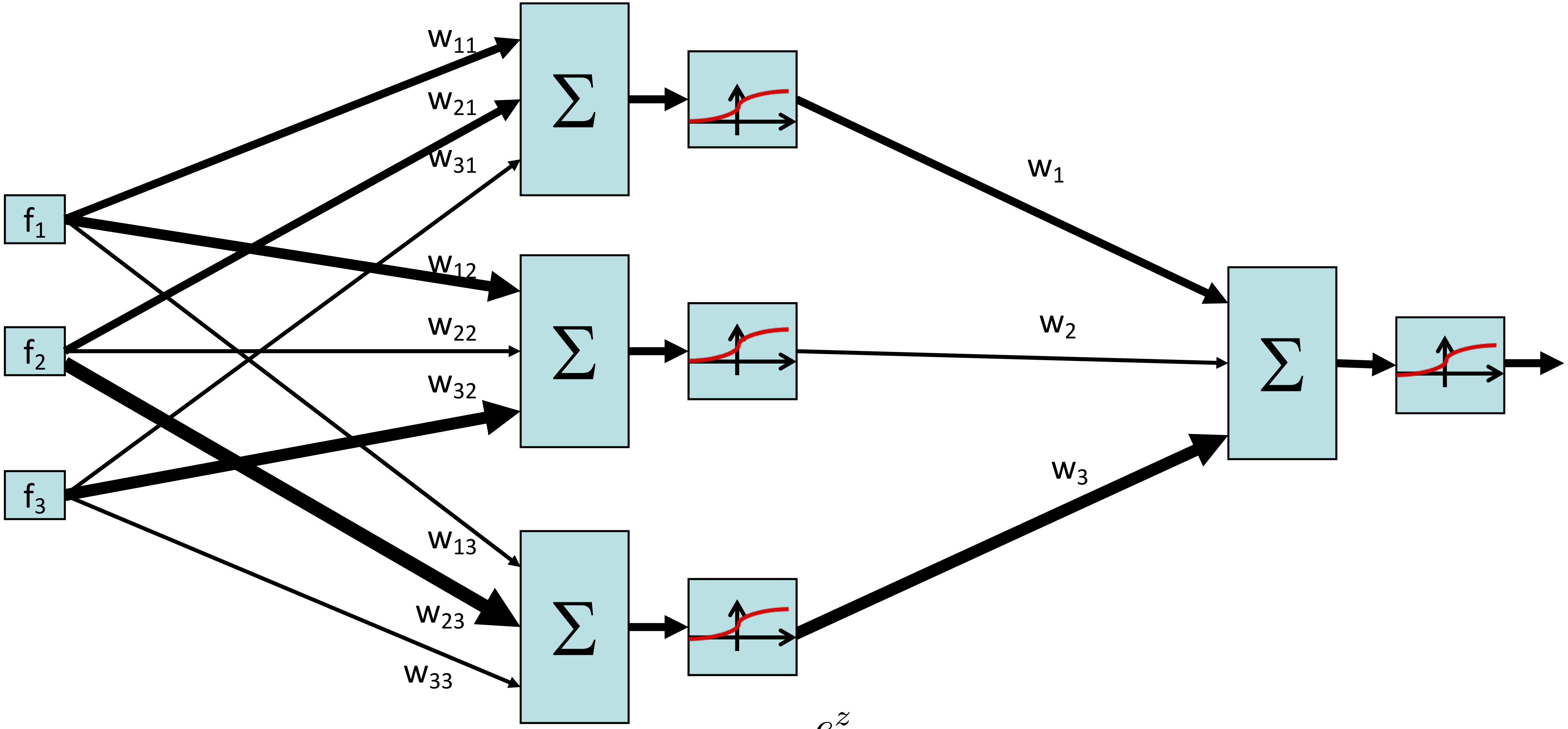


- Softmax Function: Dot product $z = w^T f(x)$ results in activation of $\frac{e^z}{e^z + e^{-z}}$
  - Can think of as probability of positive class.





Two-Layer Neural Network w/Softmax



$$z \rightarrow \frac{e^z}{e^z + e^{-z}}$$

Softmax

- Score for y=1: $w^\top f(x)$ Score for y=-1: $-w^\top f(x)$

- Probability of label:

$$p(y=1|f(x);w) = \frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

$$p(y=-1|f(x);w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

- Objective: $$l(w) = \prod_{i=1}^{m} p(y=y^{(i)}|f(x^{(i)});w)$$

- Log: $$ll(w) = \sum_{i=1}^{m} \log p(y=y^{(i)}|f(x^{(i)});w)$$

Goal: Find $\arg\min_{w} \; -ll(w)$

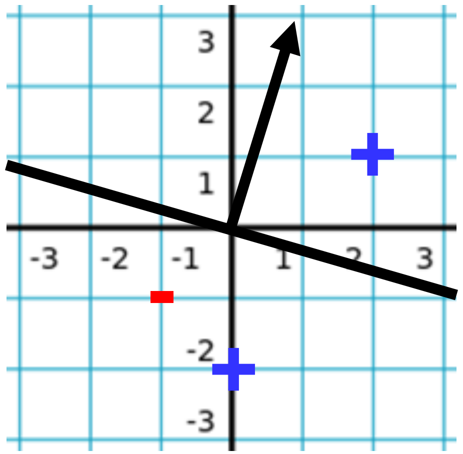
Loss Under Softmax Activation Function

- Softmax Function: Dot product $z = w^T f(x)$ results in activation of $\frac{e^z}{e^z + e^{-z}}$
  - Can think of as probability of positive class.
- Loss function for Softmax Activation Function:
  - Major advantage: Changes smoothly with parameter changes. Possible to optimize using standard numerical techniques (e.g. gradient descent).

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$

Function Optimization

- What function are we trying to minimize for the problem below?
  - $w = [1,3], x^{(1)} = [2,1], x^{(2)} = [0,-2], x^{(3)} = [-1,-1], y^{(1)} = +1, y^{(2)} = +1, y^{(3)} = -1$



$$p(y=1|f(x);w) = \frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

$$p(y=-1|f(x);w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)}|f(x^{(i)});w)$$

Function Optimization

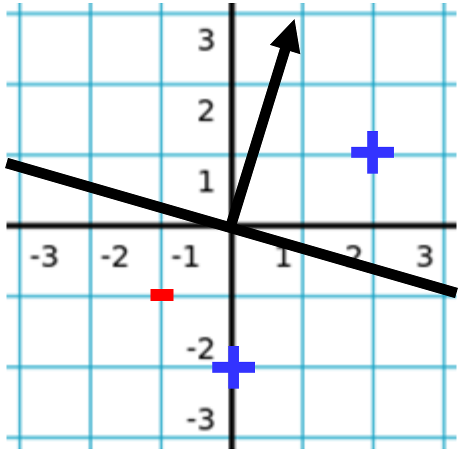
- What function are we trying to minimize for the problem below?
  - $w = [1, 3], x^{(1)} = [2, 1], x^{(2)} = [0, -2], x^{(3)} = [-1, -1], y^{(1)} = +1, y^{(2)} = +1, y^{(3)} = -1$

$$\sum_{i=1}^{3}$$

$$w^T x^{(1)} = 2w_1 + w_2 \rightarrow p\left(y = y^{(1)}\right) = \log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2} + e^{-2w_1-w_2}}\right)$$

$$w^T x^{(2)} = -2w_2 \rightarrow p\left(y = y^{(2)}\right) = \log\left(\frac{e^{-2w_2}}{e^{-2w_2} + e^{2w_2}}\right)$$

$$w^T x^{(3)} = -w_1 - w_2 \rightarrow p\left(y = y^{(3)}\right) = \log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2} + e^{-w_1-w_2}}\right)$$



Note: current weight vector [1, 3 is irrelevant]

$$-ll(w) = -\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2} + e^{-2w_1-w_2}}\right) - \log\left(\frac{e^{-2w_2}}{e^{-2w_2} + e^{2w_2}}\right) - \log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2} + e^{-w_1-w_2}}\right)$$

$$p(y = 1 | f(x); w) = \frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

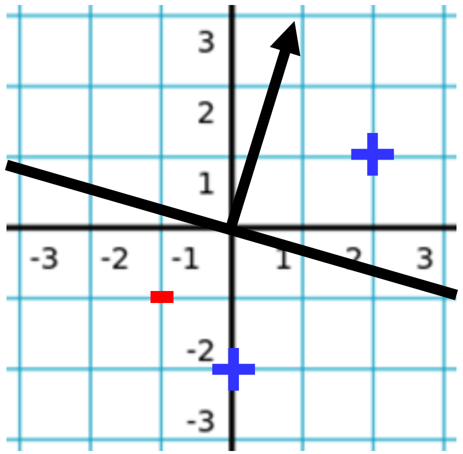
$$p(y = -1 | f(x); w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$

Function Optimization

- Finding the best weight vector for the positive class is equivalent to finding w_1 and w_2 such that the function below is minimized.

$$\min_{w_1,w_2}\left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)\right)$$



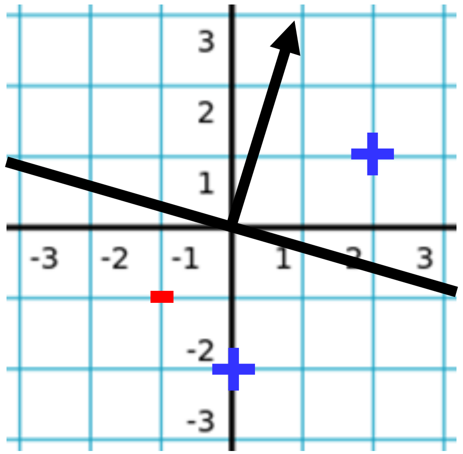
$$p(y=1|f(x);w)=\frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)}+e^{-w^\top f(x)}}$$

$$p(y=-1|f(x);w)=\frac{e^{-w^\top f(x)}}{e^{w^\top f(x)}+e^{-w^\top f(x)}}$$

$$ll(w)=\sum_{i=1}^{m}\log p(y=y^{(i)}|f(x^{(i)});w)$$

Function Optimization

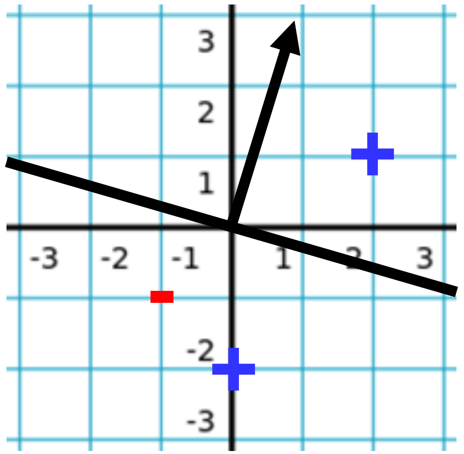
- Minimizing a multi-variate function g:
  - Purely analytic approach: Requires closed-form expression for function. Not sensible for functions of a billion variables or complex functions like our ll function.
  - Gradient descent: Calculate gradient ∇g, and move in same direction as gradient (scaled by learning rate alpha): $\boldsymbol{w_{new}} = \boldsymbol{w_{old}} - \boldsymbol{\alpha} * \nabla \mathrm{g}(\mathbf{w})$



$$\min_{w_1, w_2} \left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2} + e^{-2w_1-w_2}}\right) - \log\left(\frac{e^{-2w_2}}{e^{-2w_2} + e^{2w_2}}\right) - \log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2} + e^{-w_1-w_2}}\right) \right)$$

Function Optimization

- Minimizing a multi-variate function g:
  - Purely analytic approach: Requires closed-form expression for function. Not sensible for functions of a billion variables or complex functions like our ll function.
  - Gradient descent: Calculate gradient ∇g, and move in same direction as gradient (scaled by learning rate alpha): $\boldsymbol{w_{new}} = \boldsymbol{w_{old}} - \boldsymbol{\alpha} * \nabla \mathrm{g}(\mathbf{w})$
  - Suppose $\nabla g|_{\boldsymbol{w_{old}}=[1,3]} = [0, 4]$, and our learning rate is 0.01, what will be $\boldsymbol{w_{new}}$?

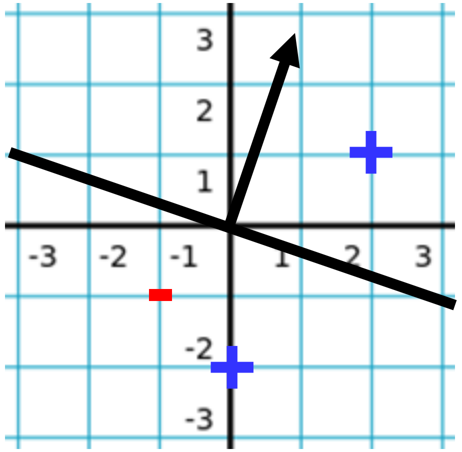


$$\min_{\mathrm{w}_1,\mathrm{w}_2}\left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)\right)$$

Function Optimization

- Minimizing a multi-variate function g:
  - Purely analytic approach: Requires closed-form expression for function. Not sensible for functions of a billion variables or complex functions like our ll function.
  - Gradient descent: Calculate gradient ∇g, and move in same direction as gradient (scaled by learning rate alpha): $\boldsymbol{w_{new}} = \boldsymbol{w_{old}} - \boldsymbol{\alpha} * \nabla \mathrm{g}(\mathbf{w})$
  - Suppose $\nabla g|_{\boldsymbol{w_{old}}=[1,3]} = [0, 4]$, and our learning rate is 0.01, what will be $\boldsymbol{w_{new}}$?

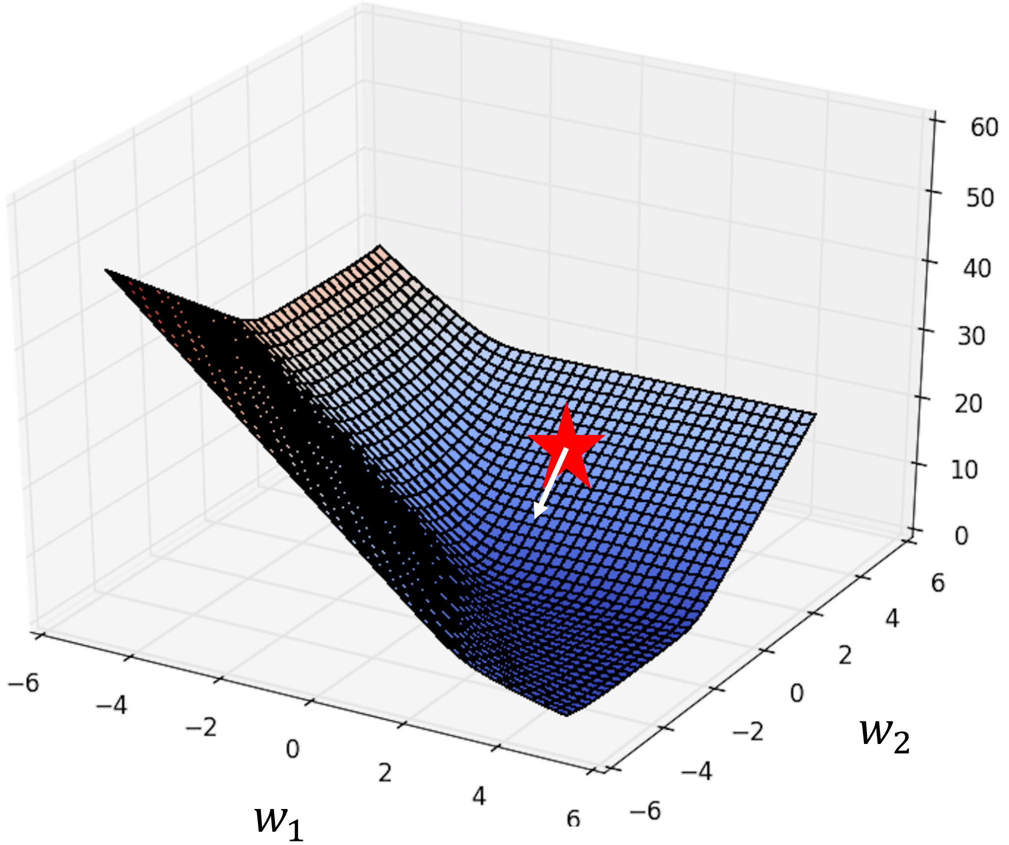
  - $\boldsymbol{w_{new}} = [1, 3] - [0, 4] * 0.01 = [1, 2.96]$

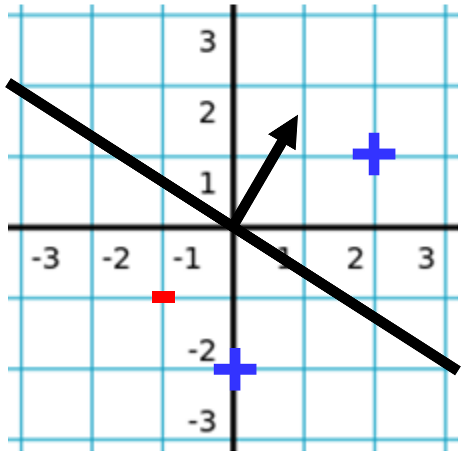


$$\min_{w_1, w_2} \left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2} + e^{-2w_1-w_2}}\right) - \log\left(\frac{e^{-2w_2}}{e^{-2w_2} + e^{2w_2}}\right) - \log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2} + e^{-w_1-w_2}}\right)\right)$$

Gradient Descent Visualization

- We can plot ll(w).

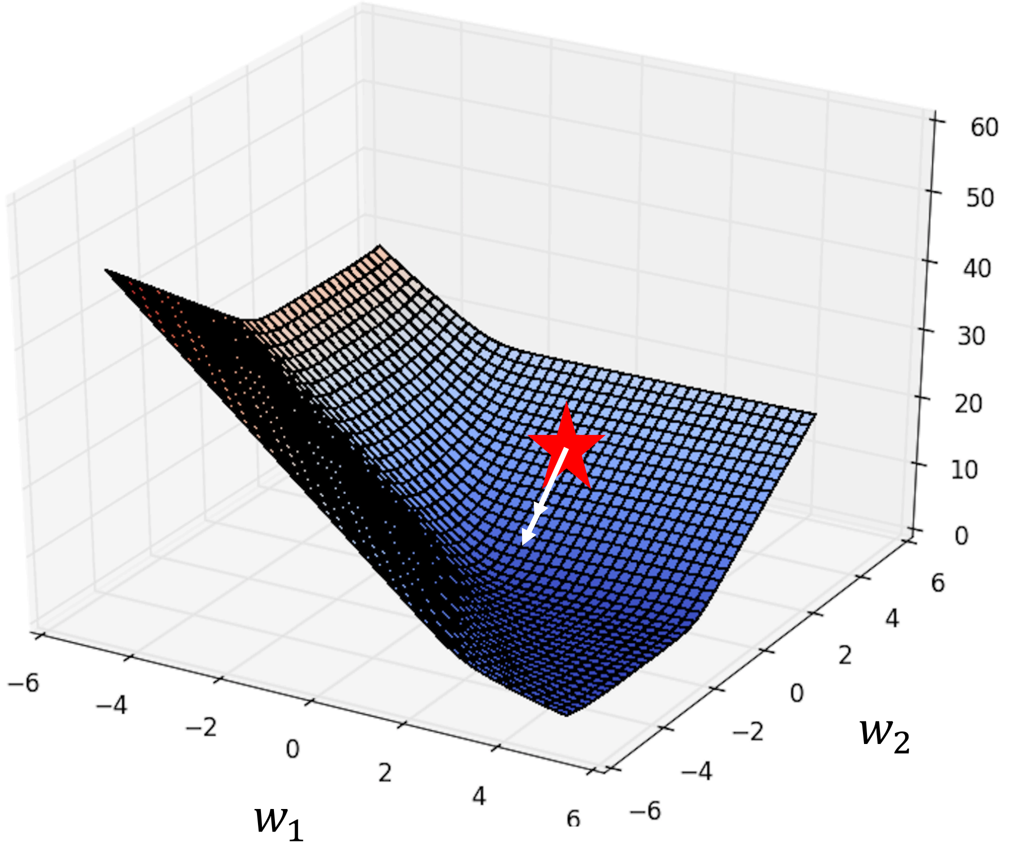


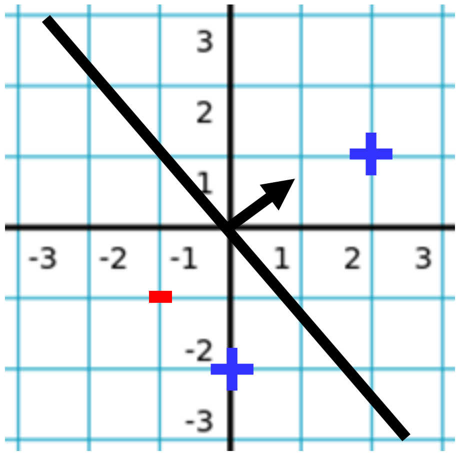


$$\min_{w_1,w_2}\left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)\right)$$

Gradient Descent Visualization

- We can plot ll(w).

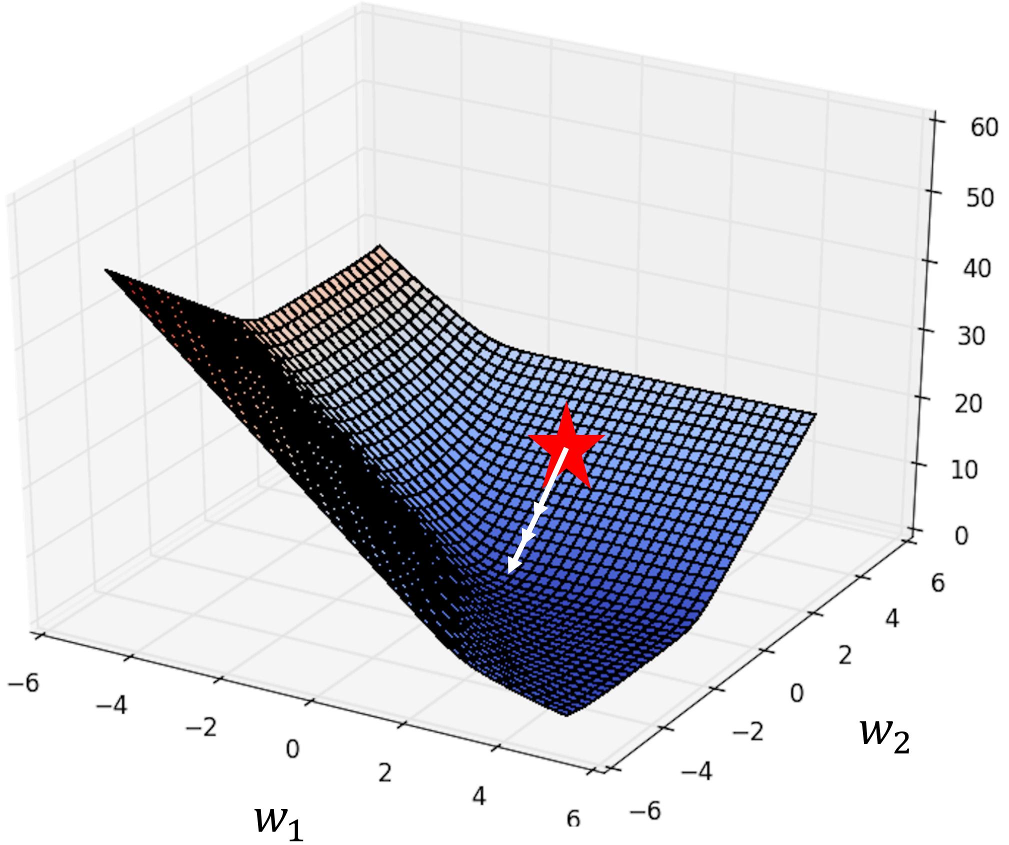


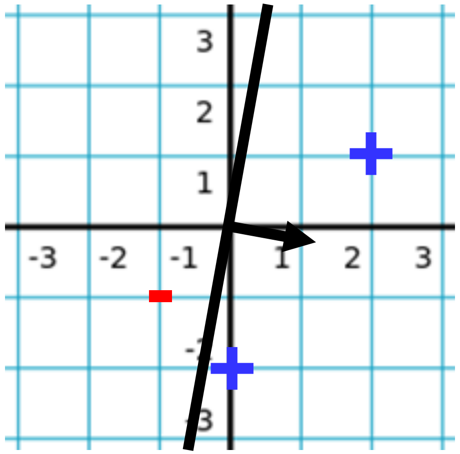


$$\min_{w_1,w_2}\left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)\right)$$

Gradient Descent Visualization

- We can plot ll(w).

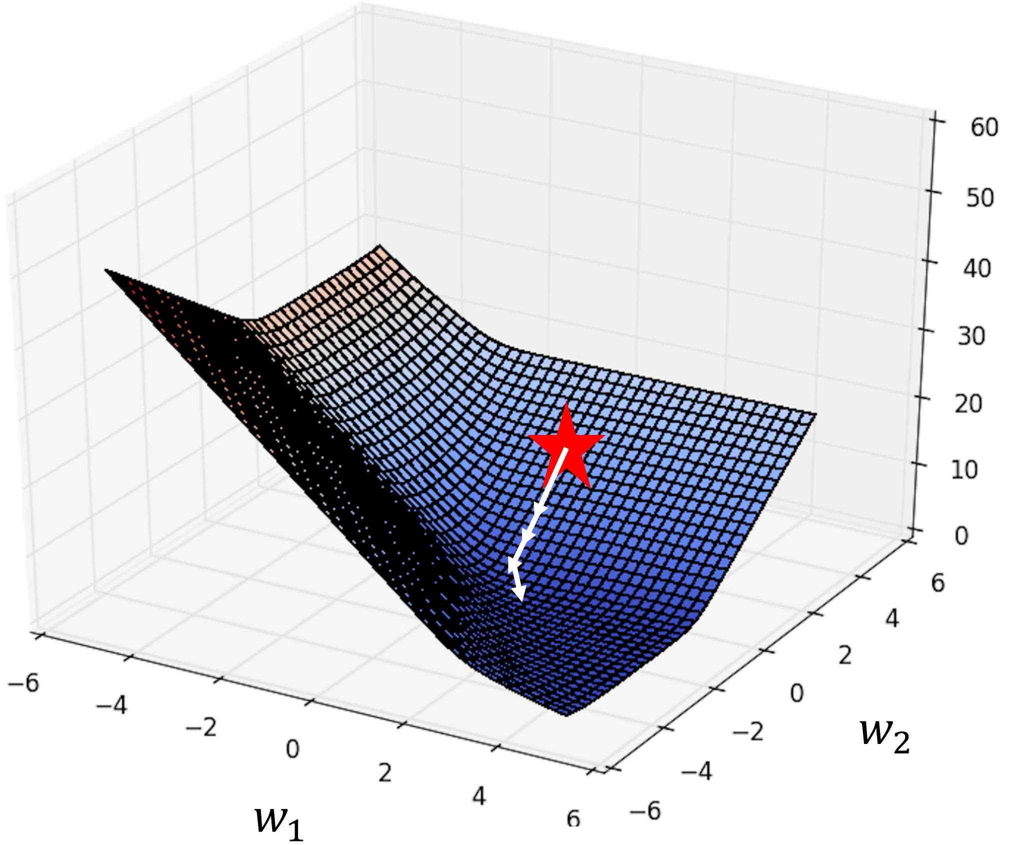


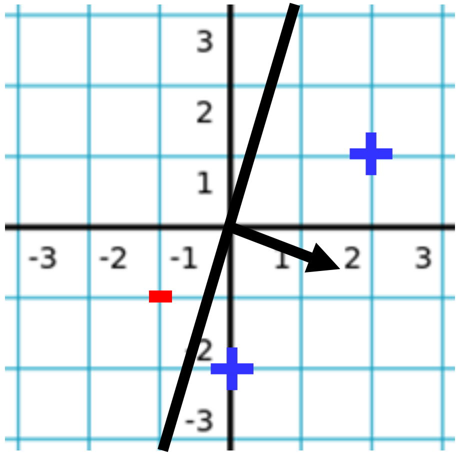


$$\min_{w_1,w_2}\left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)\right)$$

Gradient Descent Visualization

- We can plot ll(w).

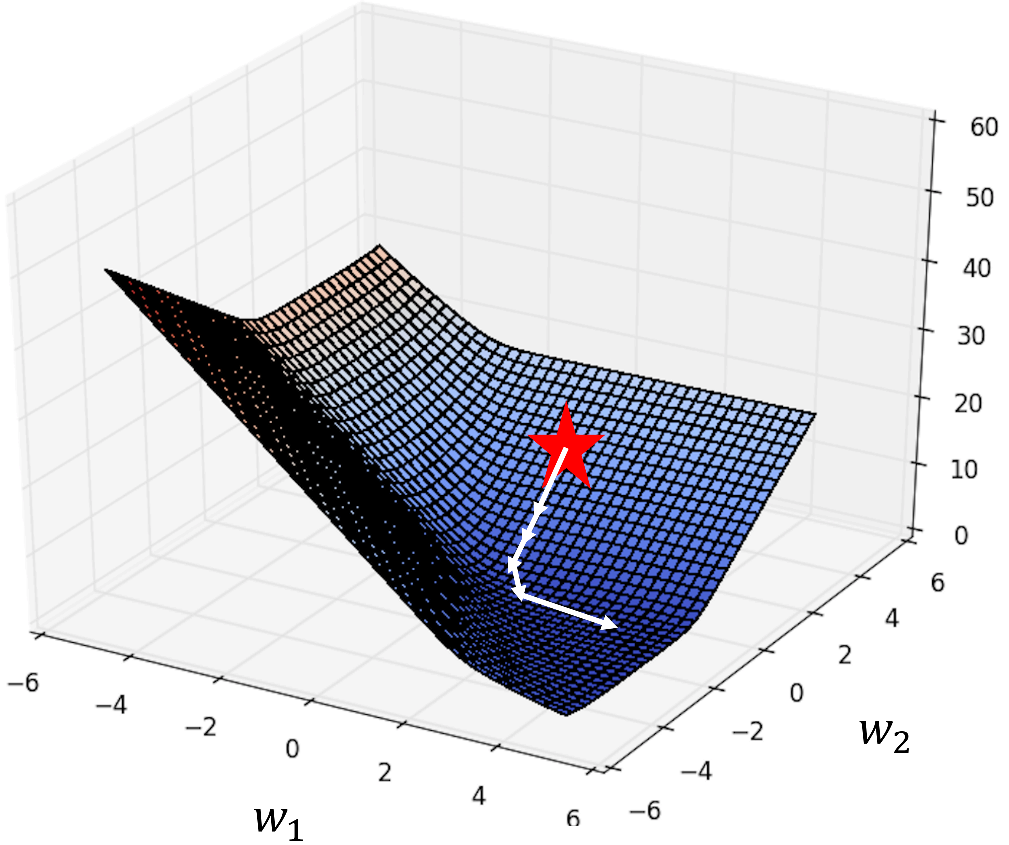


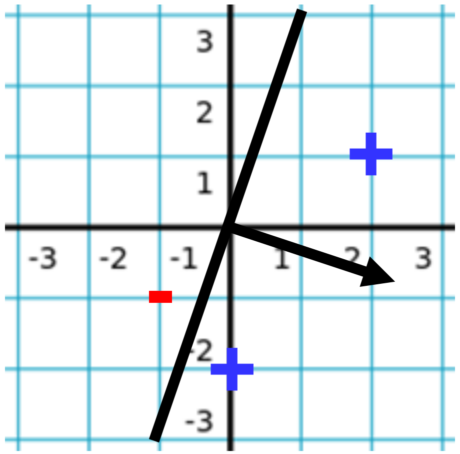


$$\min_{w_1,w_2}\left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)\right)$$

Gradient Descent Visualization

- We can plot ll(w).





$$\min_{w_1, w_2} \left(-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2} + e^{-2w_1-w_2}}\right) - \log\left(\frac{e^{-2w_2}}{e^{-2w_2} + e^{2w_2}}\right) - \log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2} + e^{-w_1-w_2}}\right)\right)$$

Gradient Descent

- Punchline: If we can somehow compute our gradient, we can use gradient descent.
- How do we compute the gradient?
  - Purely analytically, e.g. input the function into a symbolic Mathematics tool like Mathematica: $-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)$

(In the real world, this wouldn't be hand-typed!)

```
In[4]:= g[w1_, w2_] = -Log[Exp[2*w1 + w2] / (Exp[2*w1 + w2] + Exp[-2*w1 - w2])] -
          Log[Exp[-2*w2] / (Exp[-2*w2] + Exp[2*w2])] -
          Log[Exp[w1 + w2] / (Exp[w1 + w2] + Exp[-w1 - w2])]
```

Out[4]= $-\mathrm{Log}\left[\frac{e^{-2\,w2}}{e^{-2\,w2}+e^{2\,w2}}\right]-\mathrm{Log}\left[\frac{e^{w1+w2}}{e^{-w1-w2}+e^{w1+w2}}\right]-\mathrm{Log}\left[\frac{e^{2\,w1+w2}}{e^{-2\,w1-w2}+e^{2\,w1+w2}}\right]$

```
In[6]:= gradient[w1_, w2_] = Simplify[Grad[g[w1, w2], {w1, w2}]]
```

Out[6]= $\left\{-\frac{2\left(3+2\,e^{2\,(w1+w2)}+e^{4\,w1+2\,w2}\right)}{\left(1+e^{2\,(w1+w2)}\right)\left(1+e^{4\,w1+2\,w2}\right)},\ \frac{2\left(-2-e^{2\,(w1+w2)}-e^{4\,w1+2\,w2}+e^{2\,w1+6\,w2}+e^{4\,w1+6\,w2}+2\,e^{6\,w1+8\,w2}\right)}{\left(1+e^{4\,w2}\right)\left(1+e^{2\,(w1+w2)}\right)\left(1+e^{4\,w1+2\,w2}\right)}\right\}$

```
In[16]:= gradient[1., 3.]

Out[16]= {-0.000852292, 3.99921}
```

Gradient Descent

- Punchline: If we can somehow compute our gradient, we can use gradient descent.
- How do we compute the gradient?
  - Purely analytically, e.g. input the function into a symbolic Mathematics tool like Mathematica: $-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)$
  - Finite difference approximation (example on next slide).

Finite Difference Approximation

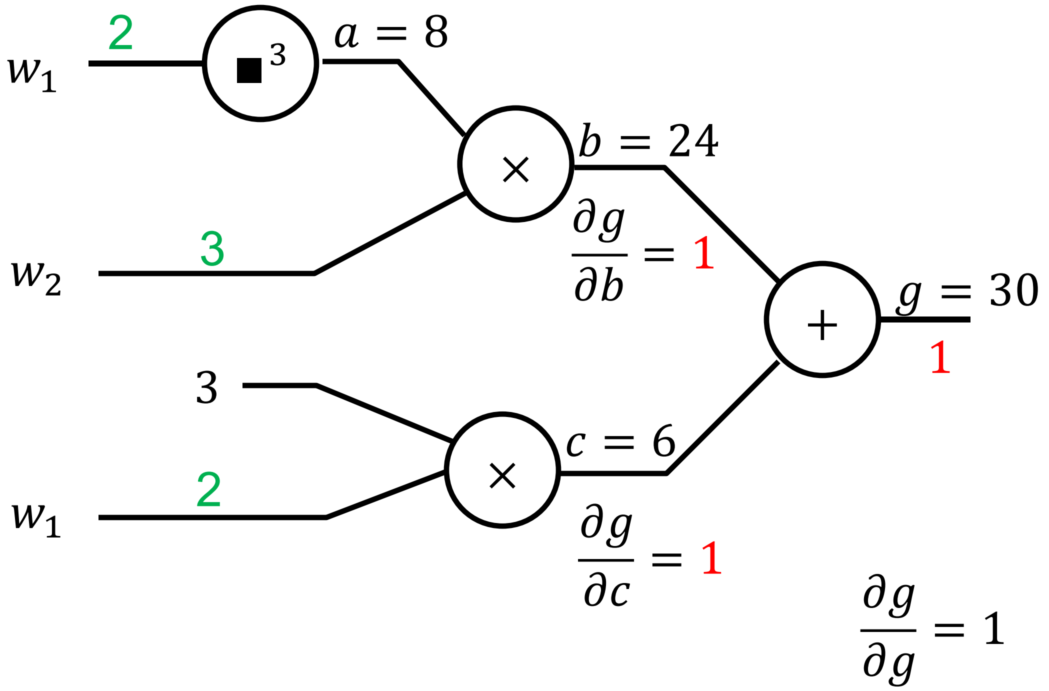
- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and we want to compute the gradient at 3, 4.
- Analytic solution: $\nabla g(\boldsymbol{w}) = (3w_1^2 w_2 + 3)i + w_1^3 j$
  - Gradient is $\nabla g(\boldsymbol{w}) = (3 \times 3^2 \times 4 + 3)i + 3^3 j = 111i + 27j$
- Finite difference approximation: $\nabla g(\boldsymbol{w}) \approx \left[\frac{g(\mathrm{w}_1 + \mathrm{h}, \mathrm{w}_2) - \mathrm{g}(\mathrm{w}_1, \mathrm{w}_2)}{\mathrm{h}}, \frac{g(\mathrm{w}_1, \mathrm{w}_2 + h) - \mathrm{g}(\mathrm{w}_1, \mathrm{w}_2)}{\mathrm{h}}\right]$
  - $g([3,4]) = 3^3 \times 4 + 3 \times 3 = 117$
  - $g([3.1,4]) = 3.1^3 \times 4 + 3 \times 3.1 = 128.464$
  - $g([3,4.1]) = 3^3 \times 4.1 + 3 \times 3 = 119.7$
  - $\nabla g(\boldsymbol{w}) \approx \left[\frac{128.464 - 117}{0.1}, \frac{119.7 - 117}{0.1}\right] = [114.64, 27]$

Gradient Descent

- Punchline: If we can somehow compute our gradient, we can use gradient descent.
- How do we compute the gradient?
  - Purely analytically, e.g. input the function into a symbolic Mathematics tool like Mathematica: $-\log\left(\frac{e^{2w_1+w_2}}{e^{2w_1+w_2}+e^{-2w_1-w_2}}\right)-\log\left(\frac{e^{-2w_2}}{e^{-2w_2}+e^{2w_2}}\right)-\log\left(\frac{e^{w_1+w_2}}{e^{w_1+w_2}+e^{-w_1-w_2}}\right)$
  - Finite difference approximation.
  - Back propagation (example on next slide).

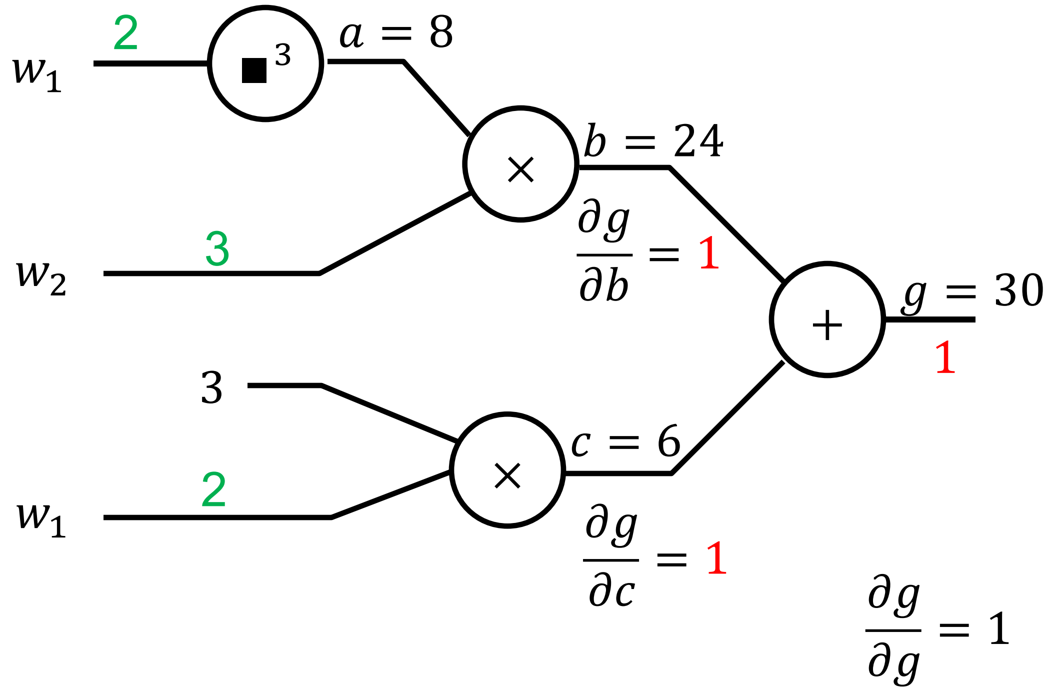
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$



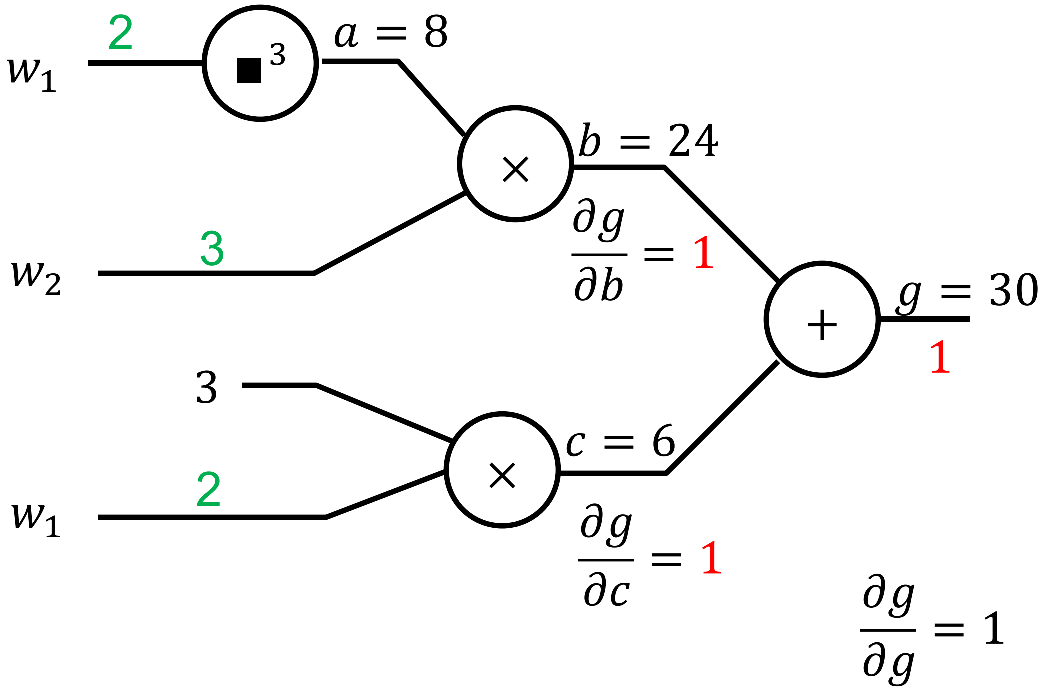
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a}$



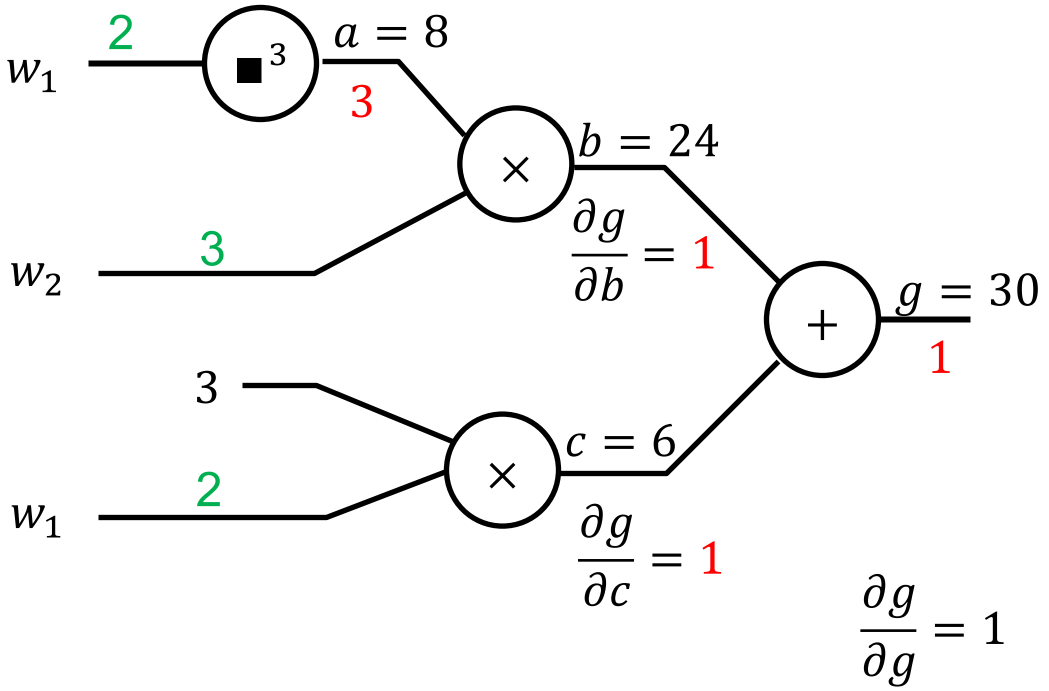
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} =?????$



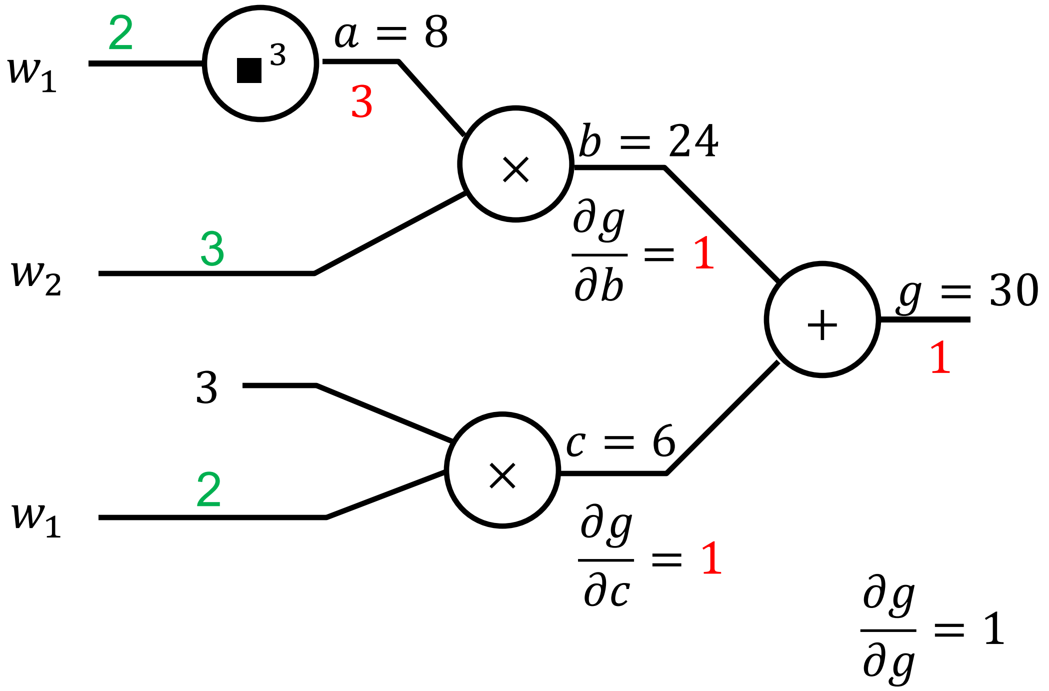
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$



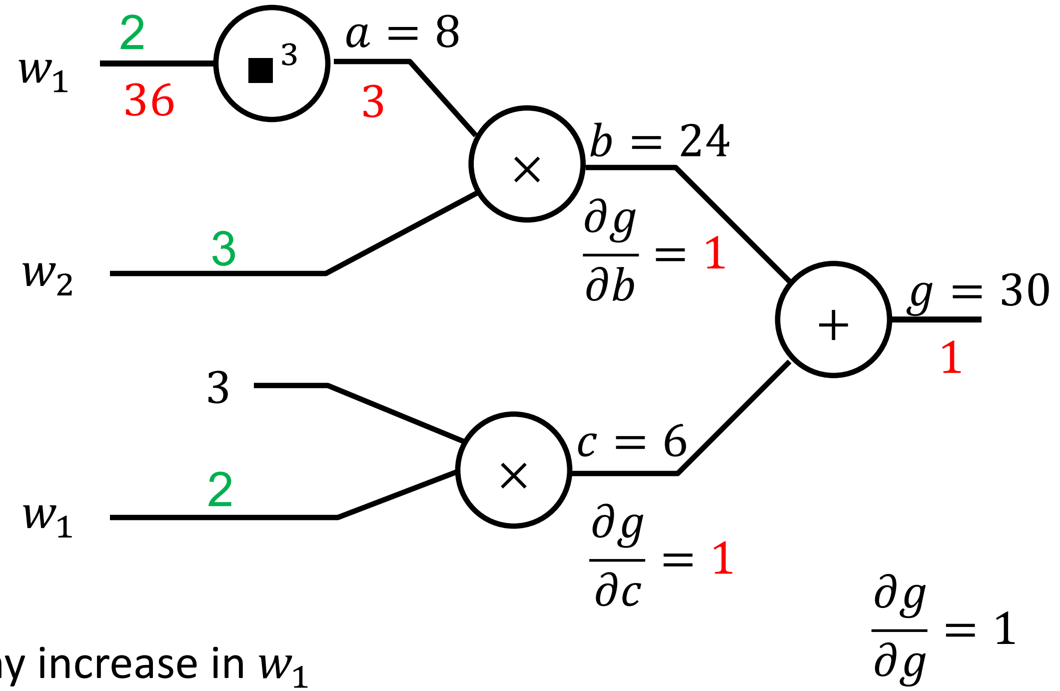
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = ?????$



Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

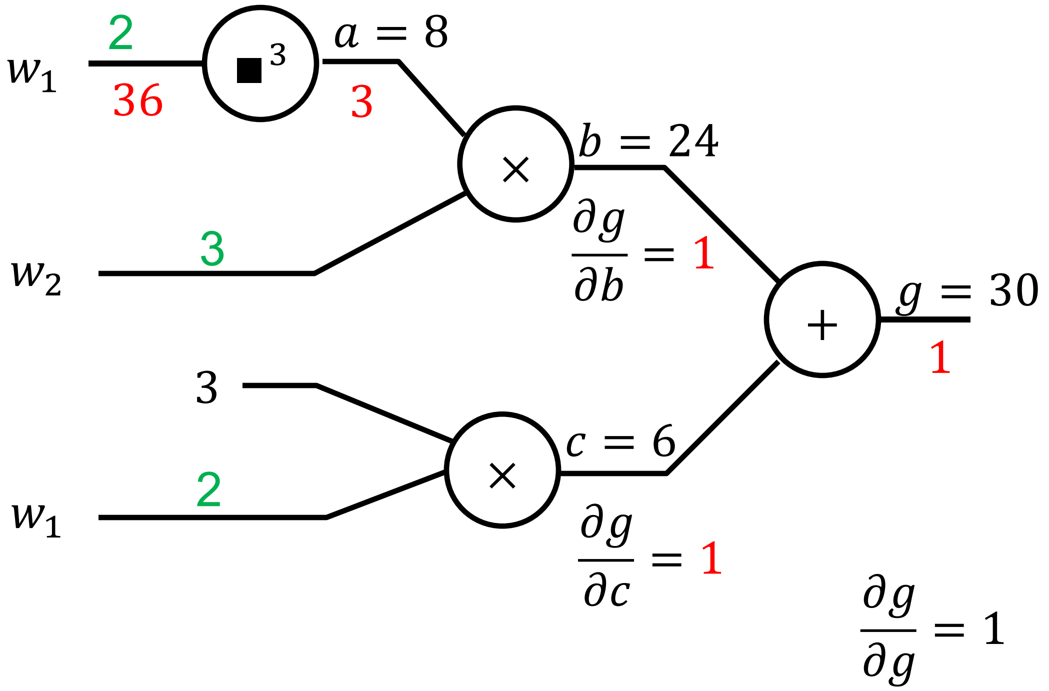
- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$



Interpretation: A tiny increase in w_1 will result in an approximately $36w_1$ increase in g due to this cube function.

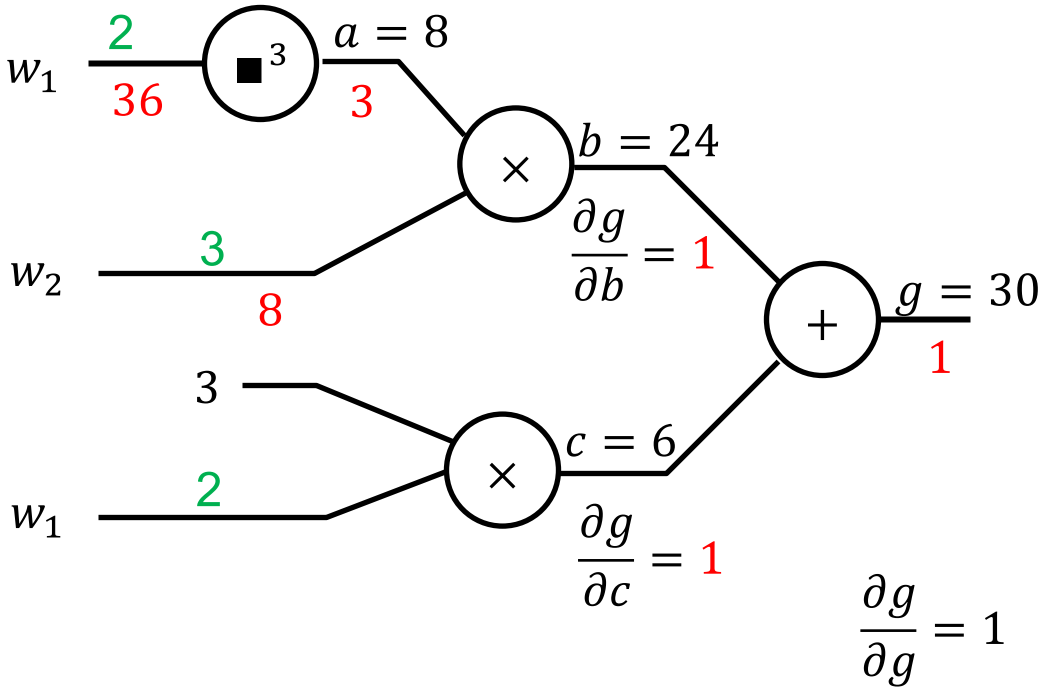
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g / \partial w_1$ and $\partial g / \partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
- $\frac{\partial g}{\partial w_2} = ???$ Hint: $b = a \times 3$ may be useful.



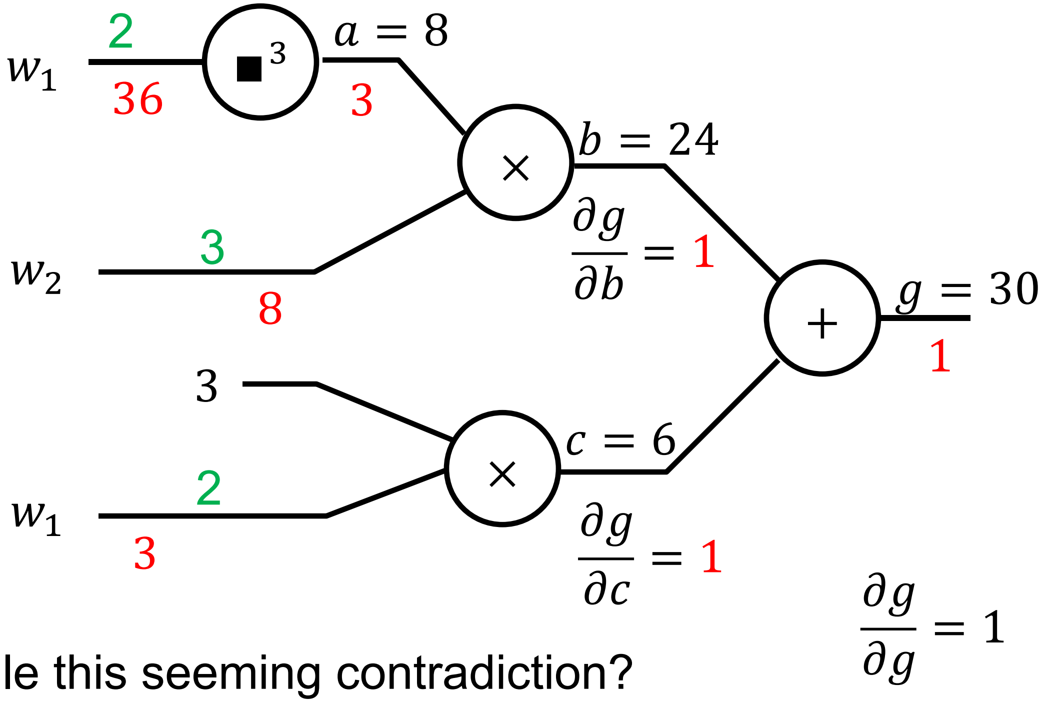
Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\partial g/\partial w_1$ and $\partial g/\partial w_2$.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
  - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$



Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

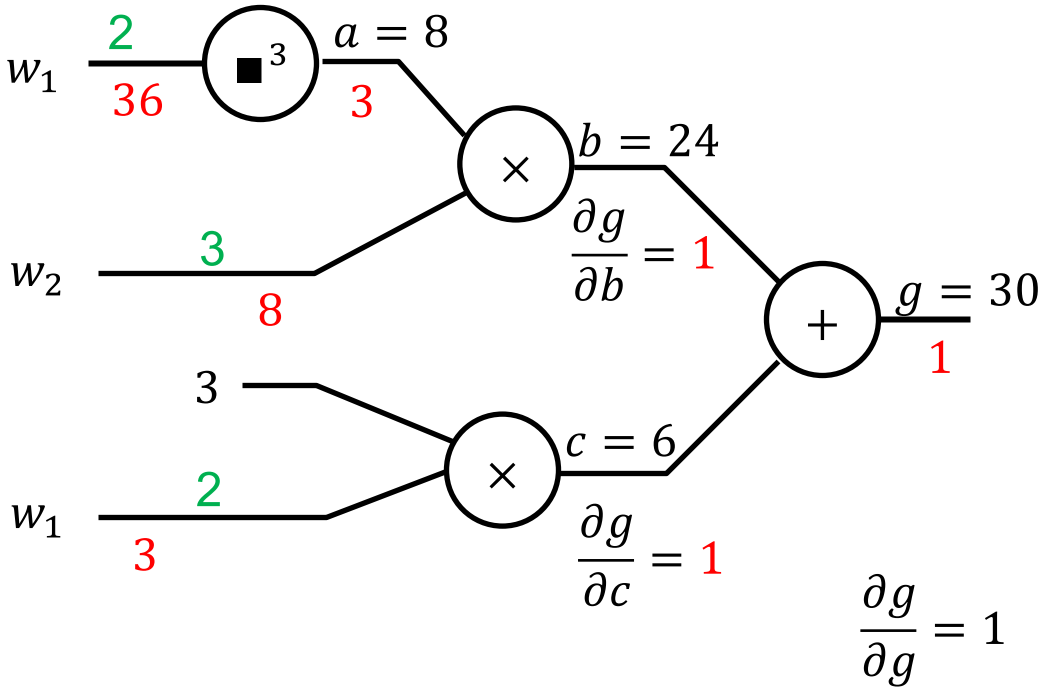
- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1\,\frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
  - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial w_2} = 1\,\frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
- $c = 3w_1$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c}\frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$



How do we reconcile this seeming contradiction? Top partial derivative means cube function contributes $36w_1$ and bottom p.d. means product contributes $3w_1$ so add them.

Back Propagation: $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(\boldsymbol{w}) = w_1^3 w_2 + 3w_1$ and want the gradient at $\boldsymbol{w} = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial a} = 1\frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
  - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b}\frac{\partial b}{\partial w_2} = 1\frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a}\frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
- $c = 3w_1$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c}\frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$



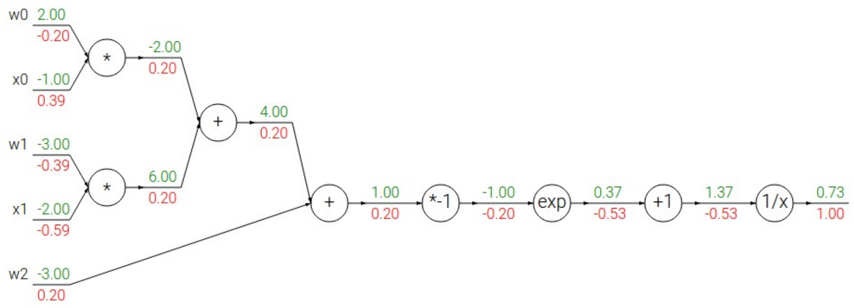
$$\nabla g = \left[\frac{\partial g}{\partial w_1}, \frac{\partial g}{\partial w_2}\right] = [39, 8]$$

Gradient Descent

- Punchline: If we can somehow compute our gradient, we can use gradient descent.
- How do we compute the gradient?
  - Purely analytically.
    - Gives exact symbolic answer. Infeasible for functions of lots of parameters or input values.
  - Finite difference approximation.
    - Gives approximation, very easy to implement.
    - Runtime for ll: $O(NM)$, where N is the number of parameters, and M is number of data points.
  - Back propagation.
    - Gives exact answer, difficult to implement.
    - Runtime for ll: $O(NM)$

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$

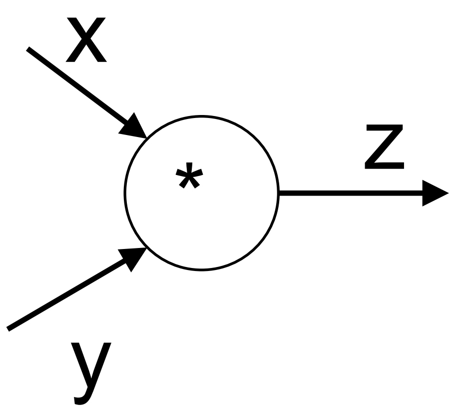
BP Implementation: forward/backward API



Graph (or Net) object. *(Rough psuedo code)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

BP Implementation: forward/backward API




```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

(x,y,z are scalars)

Optimizations and Modifications

- Mini-batching and Stochastic Gradient Descent
- Multiclass
- Activation Functions
- Dropout
- For more: See Spring 2016 Deep Learning Lecture 2
  - Covers a briefly some additional modern techniques.

Stochastic Gradient Descent

- Idea: Rather than looking at the gradient of the loss function on ALL data points, consider loss for only ONE data point.
- Update Algorithm:
  - Repeat for each data point:
    - Compute $\nabla ll_i(\mathbf{w})$, where $ll_i(w) = \log p\left(y = y^{(i)} \mid f\left(x^{(i)}; w\right)\right)$
    - $\mathbf{w} = \mathbf{w} - \boldsymbol{\alpha} * \nabla ll_i(\mathbf{w})$
- Typically will iterate over all data points before reusing any data points.
- See 189 for why this works.

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$

Mini-batches

- Idea: Rather than looking at the gradient of the loss function on ALL data points, consider loss for only some random subset.
- Update Algorithm:
  - For each batch of data points B_j:
    - Compute $\nabla ll_j(\mathbf{w})$, *where* $ll_j(w) = \sum_{i \in B_j} log\, p\left(y = y^{(i)} \,\middle|\, f\left(x^{(i)}; w\right)\right.$
    - $\mathbf{w} = \mathbf{w} - \boldsymbol{\alpha} * \nabla ll_i(\mathbf{w})$
- Example: $B_1 = \{1, \dots, k\}, B_2 = \{k+1, \dots, 2k\}, \dots$
  - More generally: Pick a set of random data points for each batch – why?
- Typically will iterate over all batches before returning reusing any data.

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$

Optimizations and Modifications

- Mini-batching and Stochastic Gradient Descent
- Multiclass
- Activation Functions
- Dropout
- For more: See Spring 2016 Deep Learning Lecture 2
  - Covers briefly some additional techniques.

Multi-class Softmax

- 3-class softmax – classes A, B, C
  - 3 weight vectors: w_A, w_B, w_C

- Probability of label A: (similar for B, C)

$$p(y = A|f(x); w) = \frac{e^{w_A^\top f(x)}}{e^{w_A^\top f(x)} + e^{w_B^\top f(x)} + e^{w_C^\top f(x)}}$$

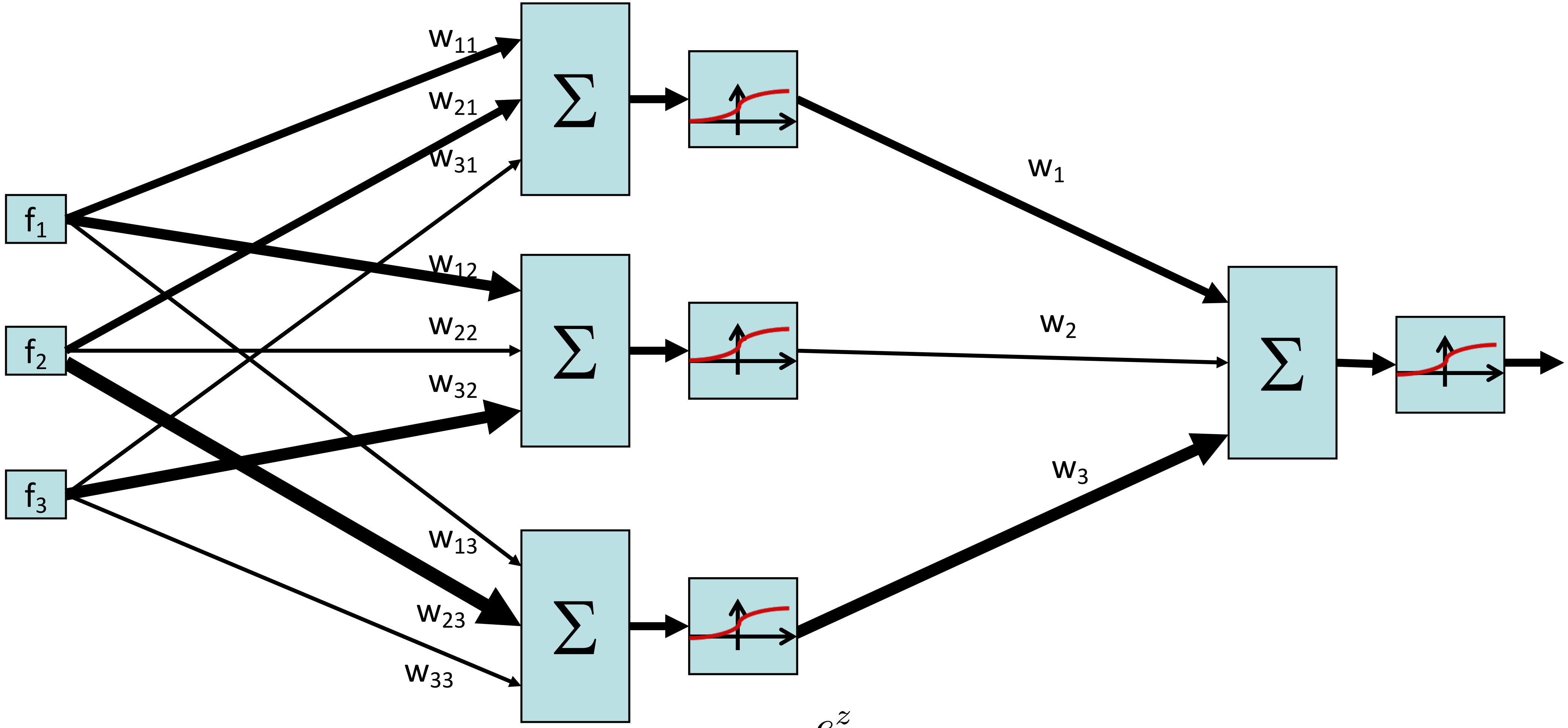
- Objective: $l(w) = \prod_{i=1}^{m} p(y = y^{(i)}|f(x^{(i)}; w)$

- Log: $ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)}|f(x^{(i)}; w)$

Optimizations and Modifications

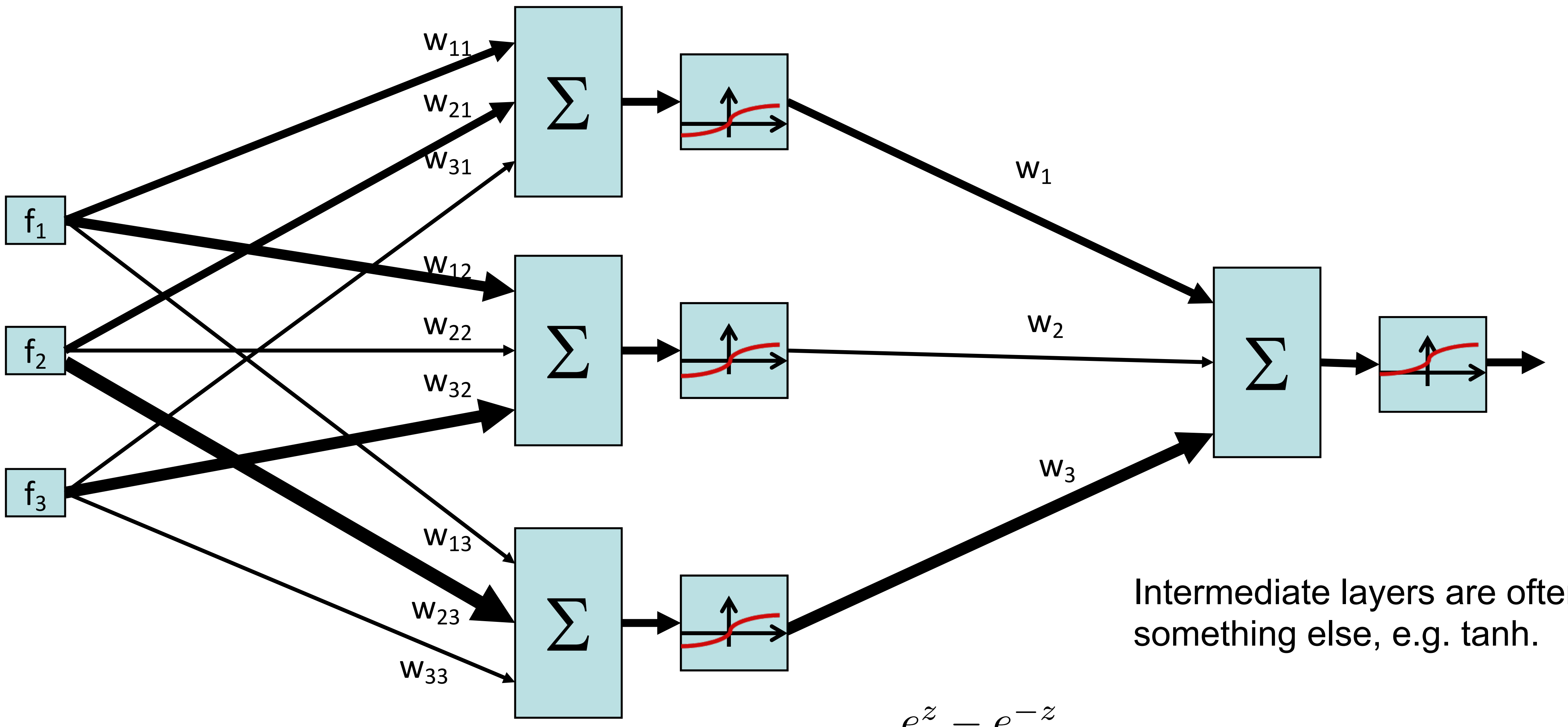
- Mini-batching and Stochastic Gradient Descent
- Multiclass
- Activation Functions
- Dropout
- For more: See Spring 2016 Deep Learning Lecture 2
  - Covers briefly some additional techniques.

Two-Layer Neural Network



$$z \rightarrow \frac{e^z}{e^z + e^{-z}}$$

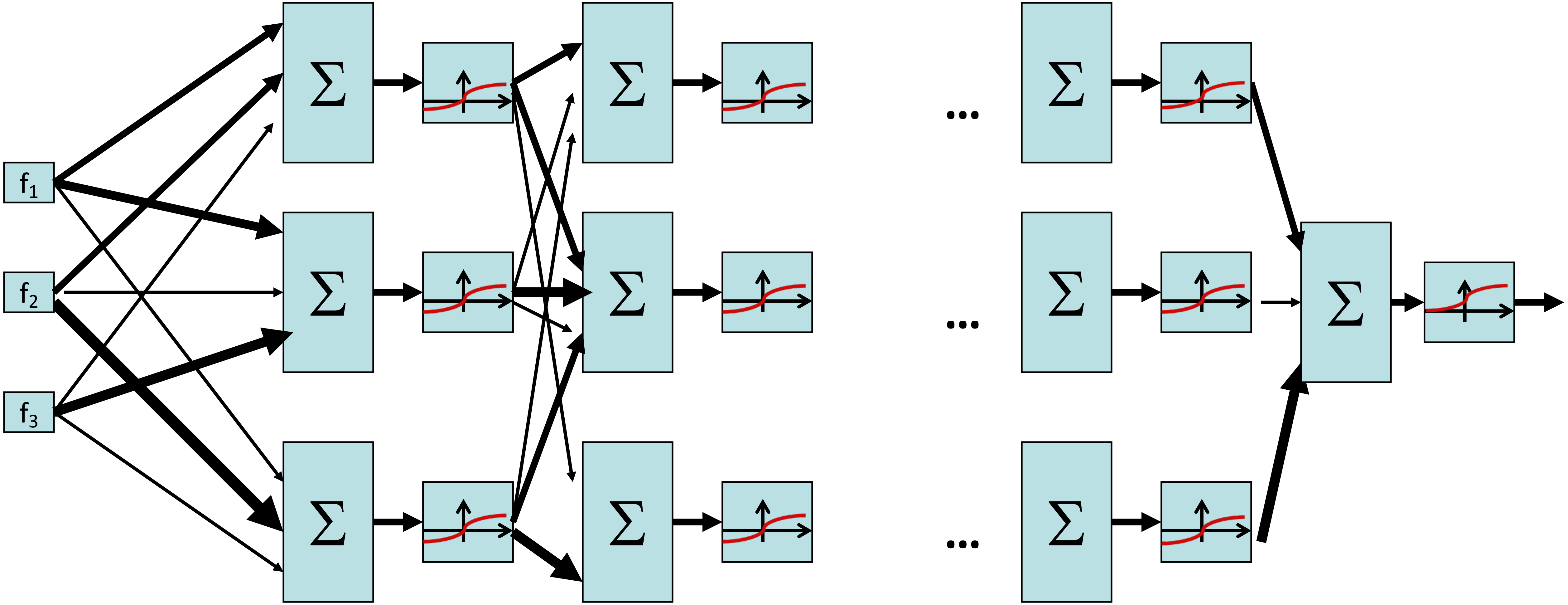
Two-Layer Neural Network



Intermediate layers are often something else, e.g. tanh.

$$z \to \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

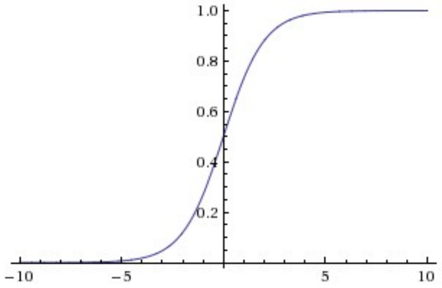
N-Layer Neural Network



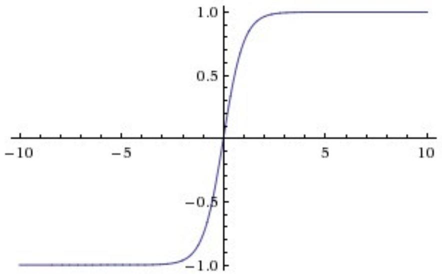
Activation Functions

Sigmoid

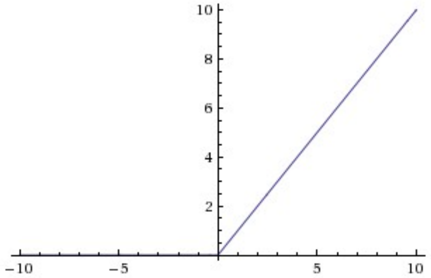
$$\sigma(x) = 1/(1 + e^{-x})$$



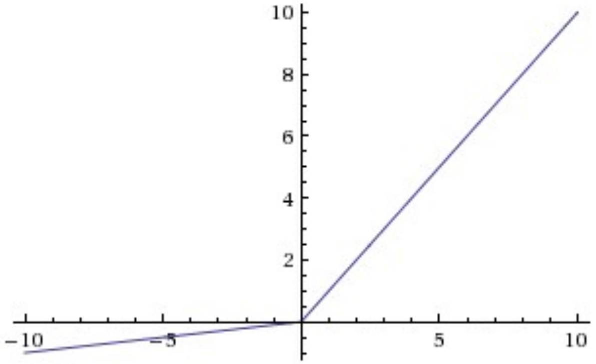
tanh tanh(x)



ReLU max(0,x)



Leaky ReLU
max(0.1x, x)

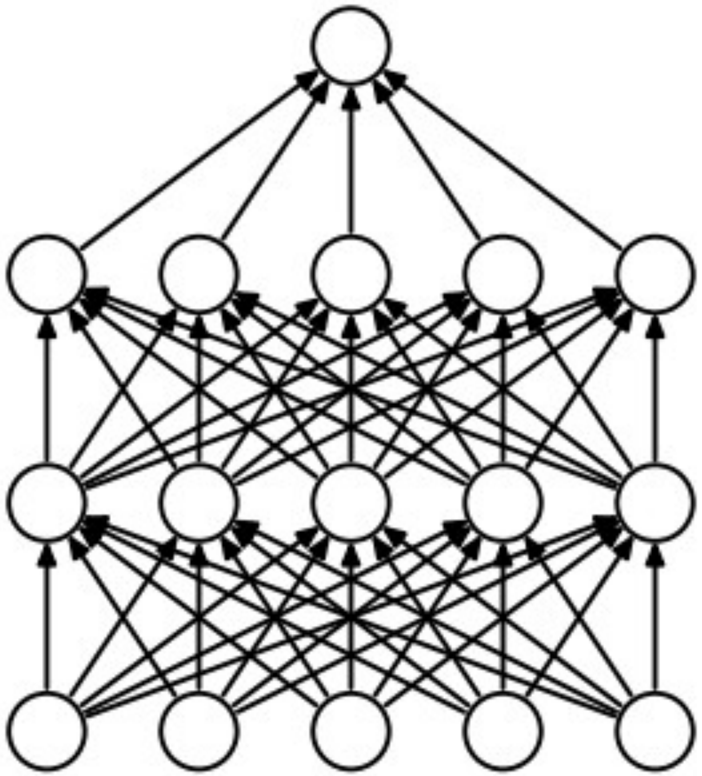


Optimizations and Modifications

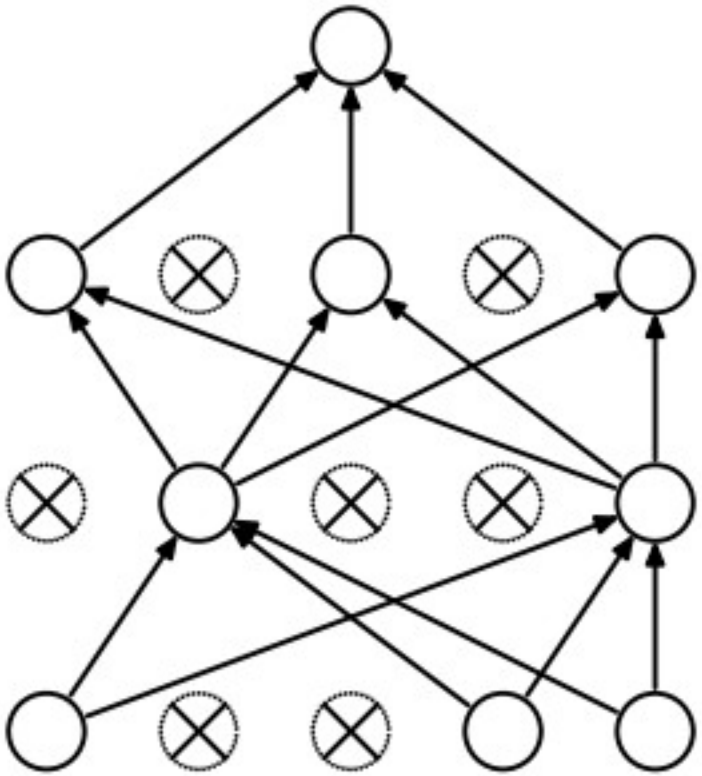
- Mini-batching and Stochastic Gradient Descent
- Multiclass
- Activation Functions
- Dropout
- For more: See Spring 2016 Deep Learning Lecture 2
  - Covers (very briefly) some additional techniques: initialization, batch normalization, gradient descent with momentum.
  - May discuss a few of these during the last lecture.
- 189 goes into MUCH more detail.
  - Reminder: Come in with your linear algebra really solid!

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net

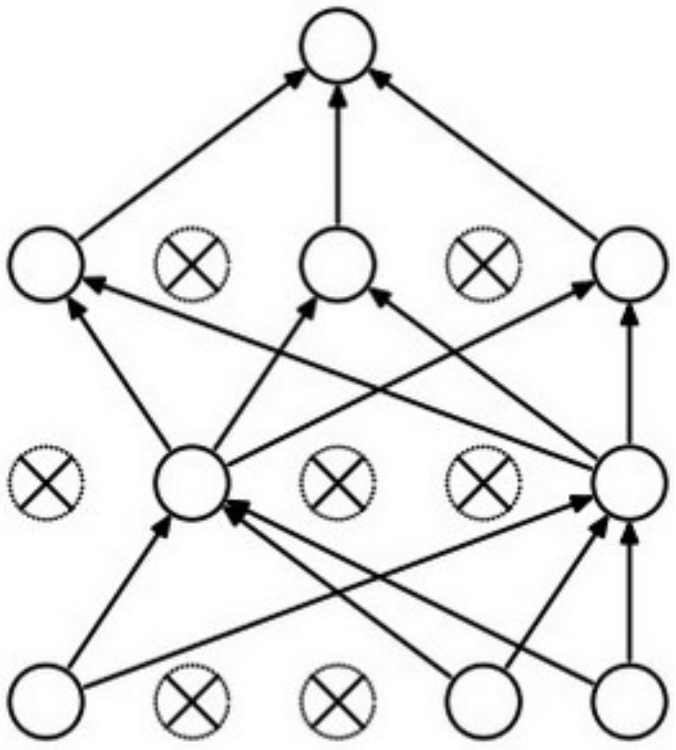


(b) After applying dropout.

[Srivastava et al., 2014]

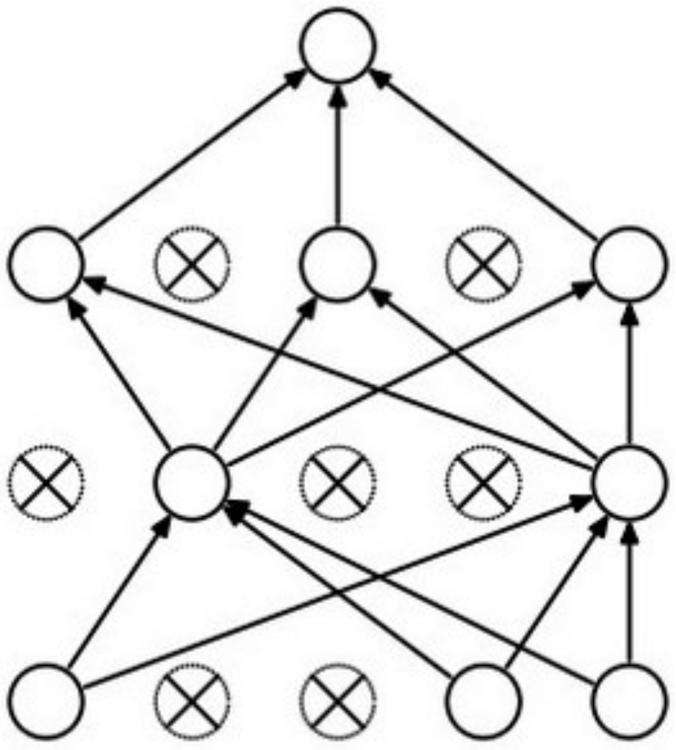
Waaaait a second…

How could this possibly be a good idea?



Waaaait a second…

How could this possibly be a good idea?



Forces the network to have a redundant representation.

Neural Network Playground

- Let’s see some Neural Networks in action: (Link)
  - We’ll now know what most (but not all) of the options mean.

Deep Learning Frameworks

TensorFlow

Theano

Torch

CAFFE

Computation Graph Toolkit (CGT)

For more: Adam’s Office Hours (Thursday 11:30 – 12:30)