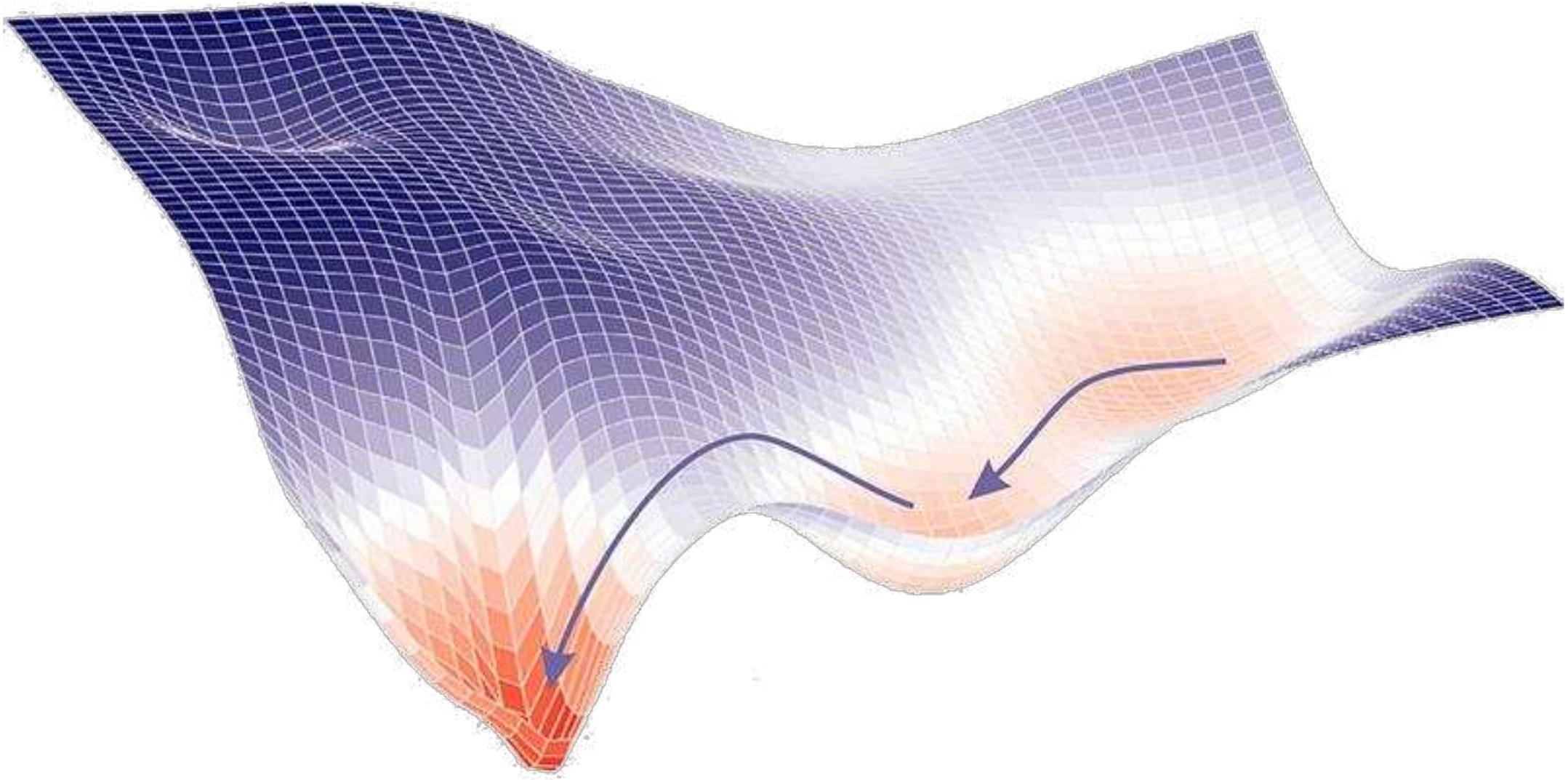
How to train a neural network



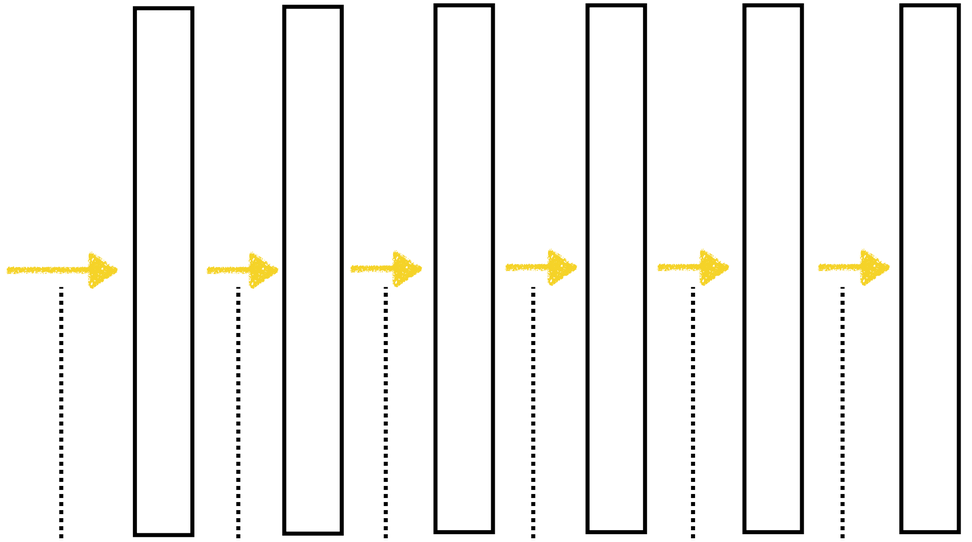
Deep learning

$\mathbf{y}^{(i)}$

`clown fish`

Learned

$\mathbf{x}^{(i)}$



Loss

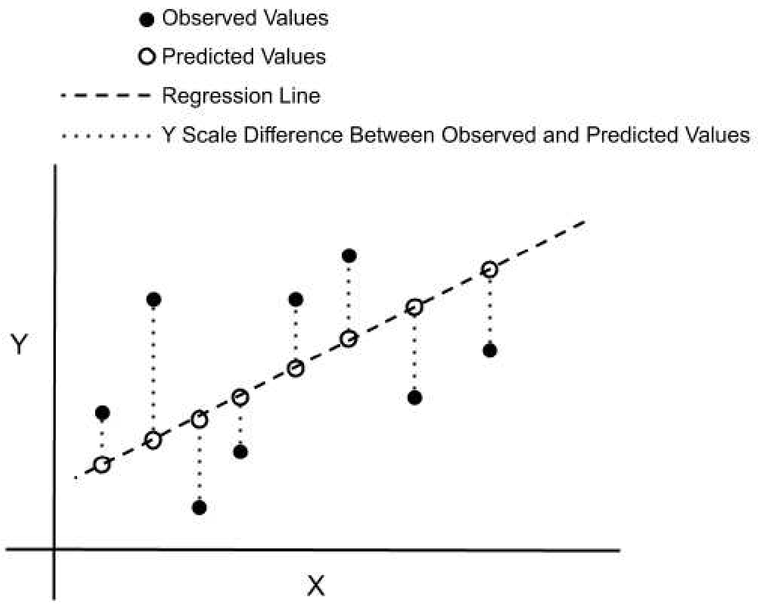
$\mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$

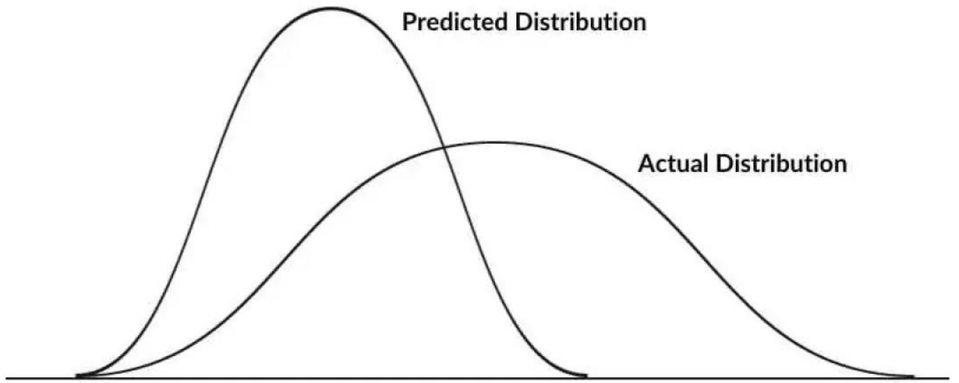
θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

Loss function

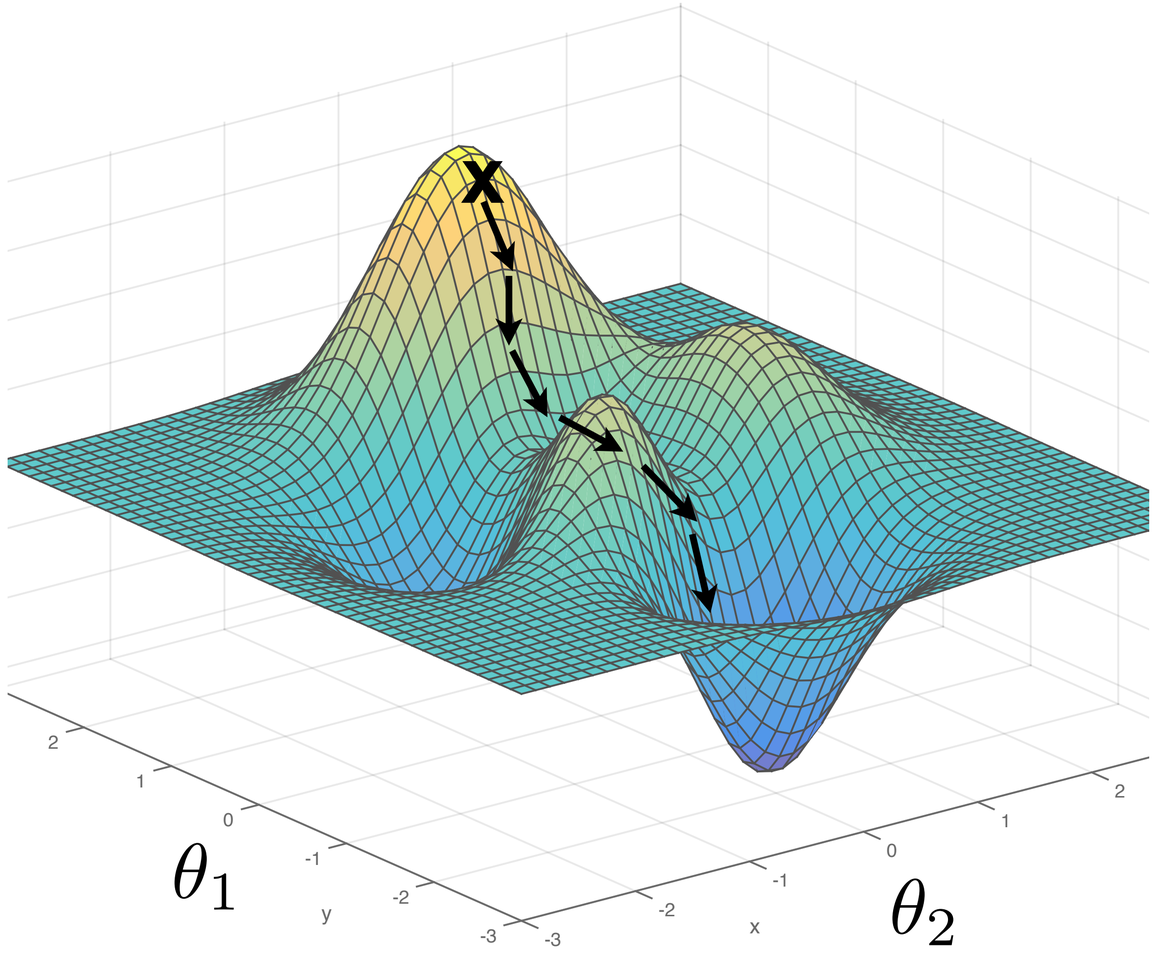
Task	Error type	Loss function	Note
Regression	Mean-squared error	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Easy to learn but sensitive to outliers (MSE, L2 loss)
	Mean absolute error	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	Robust to outliers but not differentiable (MAE, L1 loss)
Classification	Cross entropy = Log loss	$-\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] =$	Quantify the difference between two probability





Gradient descent

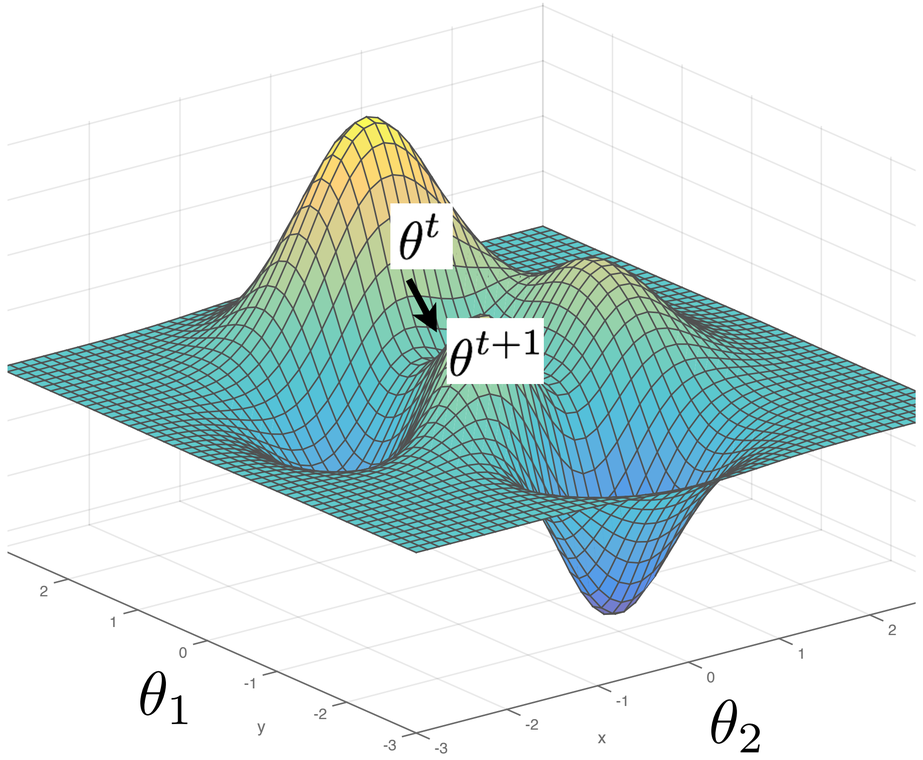
$J(\theta)$



$$\theta^* = \arg\min_{\theta} J(\theta)$$

Gradient descent

$J(\theta)$



$$\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})}_{J(\theta)}$$

One iteration of gradient descent:

$$\theta^{t+1} = \theta^t - \eta_t \left. \frac{\partial J(\theta)}{\partial \theta} \right|_{\theta=\theta^t}$$

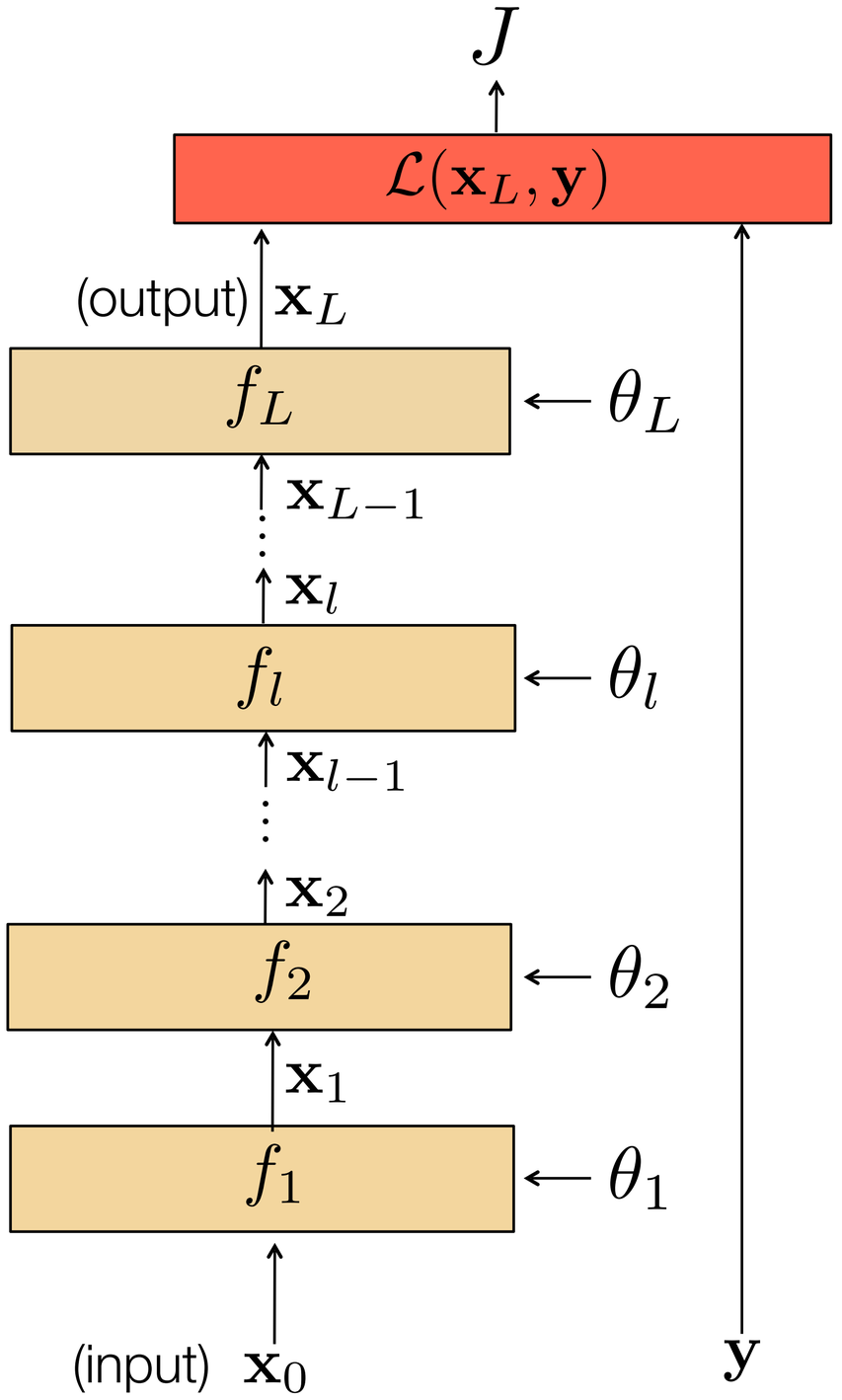
learning rate

Chains

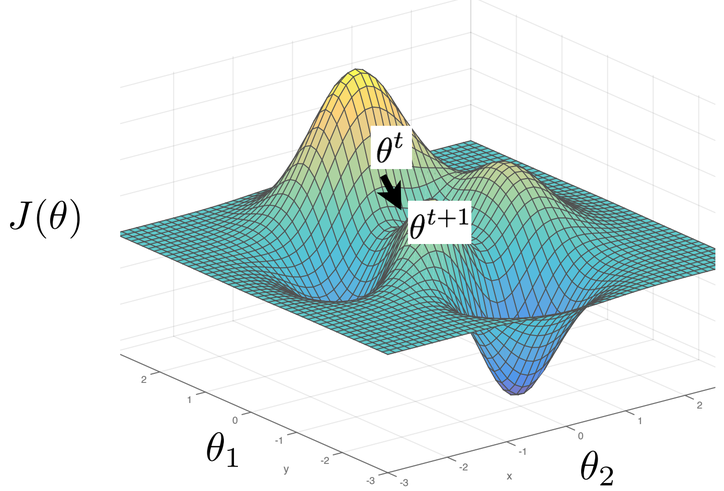
- Consider model with L layers. Layer l has vector of weights θ_l
- **Forward pass**: takes input $\mathbf{x}_{l-1}$ and passes it through each layer f_l :

$$\mathbf{x}_l = f_l(\mathbf{x}_{l-1}, \theta_l)$$

- An example of such a computation graph is an MLP
- **Loss function** $\mathcal{L}$ compares $\mathbf{x}_L$ to $\mathbf{y}$
- Overall cost is the sum of the losses over all training examples: $J = \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_L^{(i)}, \mathbf{y}^{(i)})$

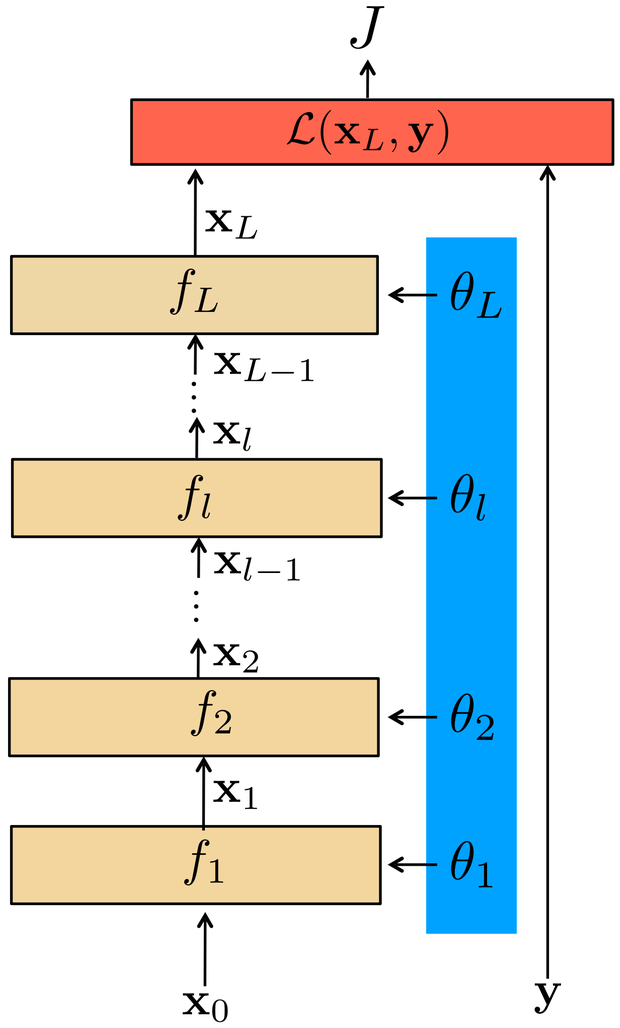


Gradient descent



- We need to compute gradients of the cost with respect to model parameters.

- By design, each layer will be differentiable with respect to its inputs (the inputs are the data and parameters)



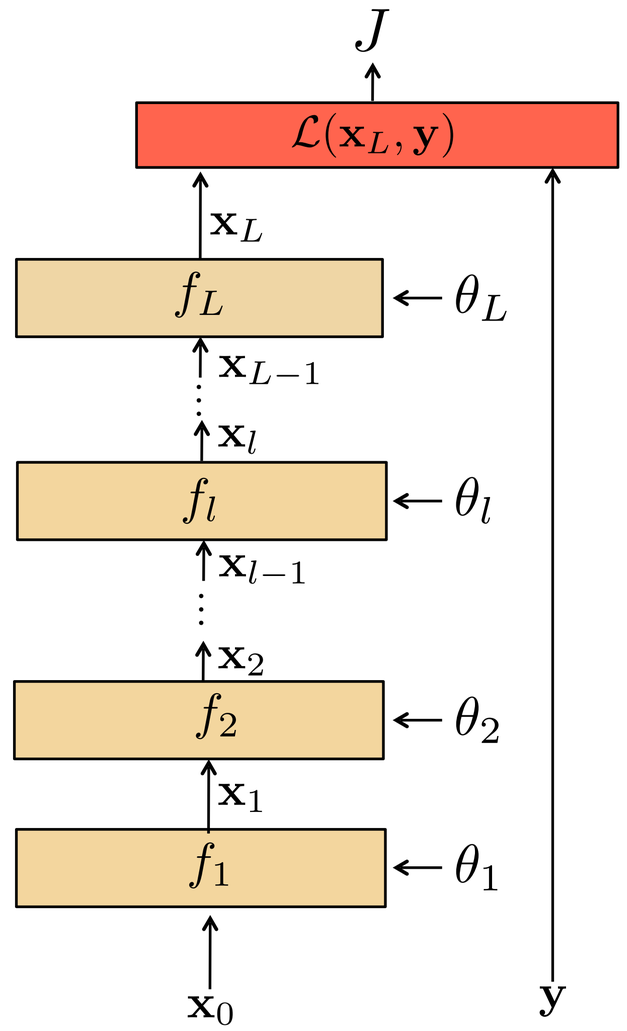
Computing gradients

To compute the gradients, we could start by writing the full energy J as a function of the model parameters.

$$J(\theta) = \sum_{i=1} \mathcal{L}(f_L(\dots f_2(f_1(\mathbf{x}_0^{(i)}, \theta_1), \theta_2), \dots \theta_L), \mathbf{y}^{(i)})$$

And then evaluate each partial derivatives separately…

$$\frac{\partial J}{\partial \theta_l}$$



instead, we can use the chain rule to derive a compact algorithm: **backpropagation**

Computing gradients

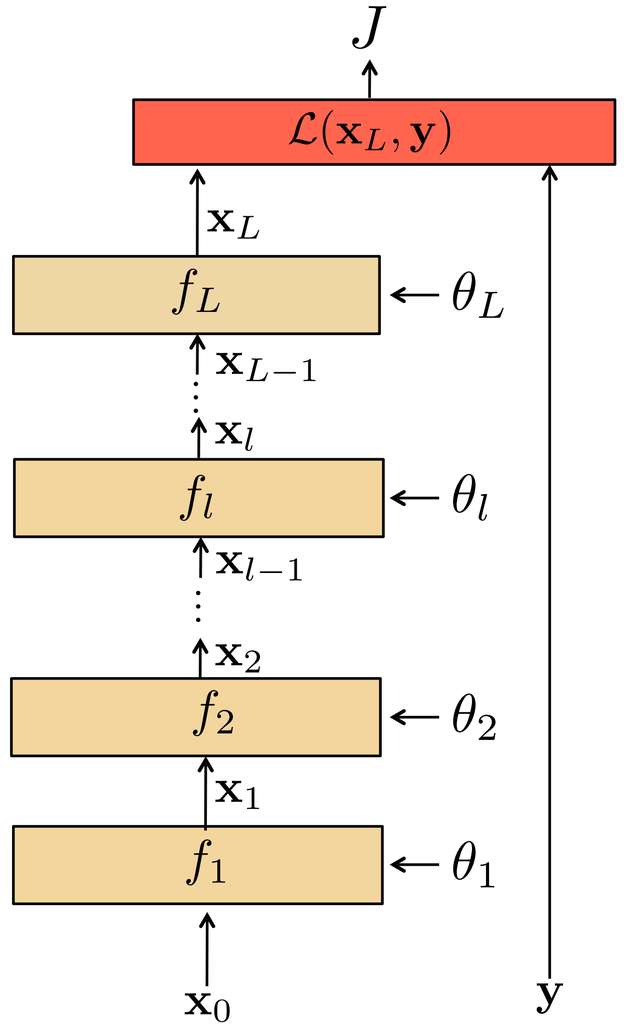
The loss J is the sum of the losses associated with each training example

$$J(\theta) = \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_L^{(i)}, \mathbf{y}^{(i)}; \theta)$$

Its gradient with respect to each of the network's parameters θ_i is:

$$\frac{\partial J(\theta)}{\partial \theta_i} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}(\mathbf{x}_L^{(i)}, \mathbf{y}^{(i)}; \theta)}{\partial \theta_i}$$

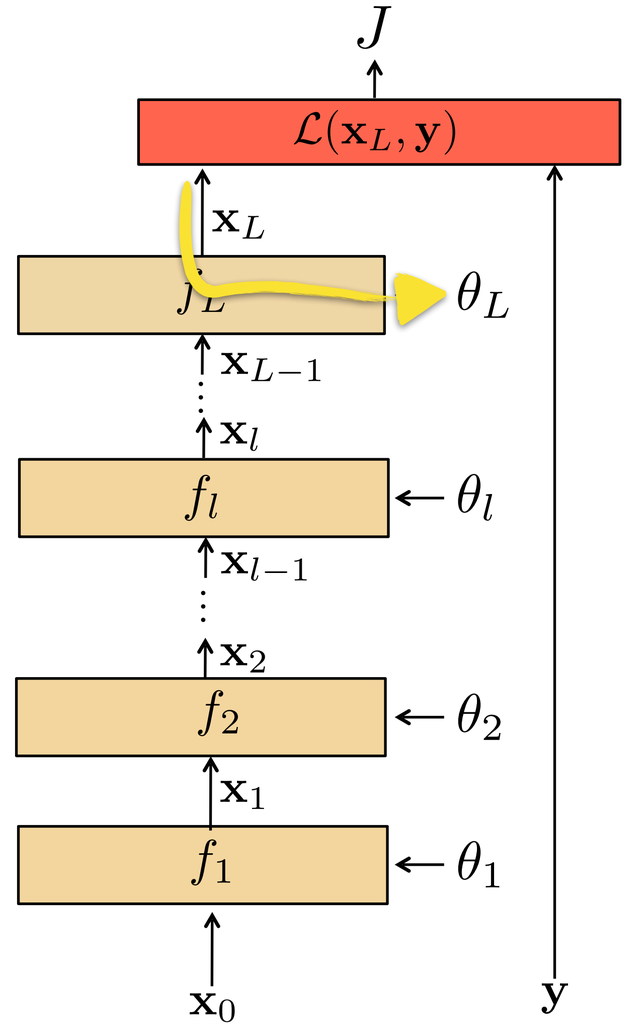
Aka how much J varies when the parameter θ_i is varied.



Computing gradients

To compute the parameter update for the last layer, we can use the **chain rule**:

$$\frac{\partial J}{\partial \theta_L} = \frac{\partial J}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \theta_L}$$



How much the loss changes when we change θ_i?

The change is the product between how much the loss changes when we change the output of the last layer and how much the output changes when we change the layer parameters.

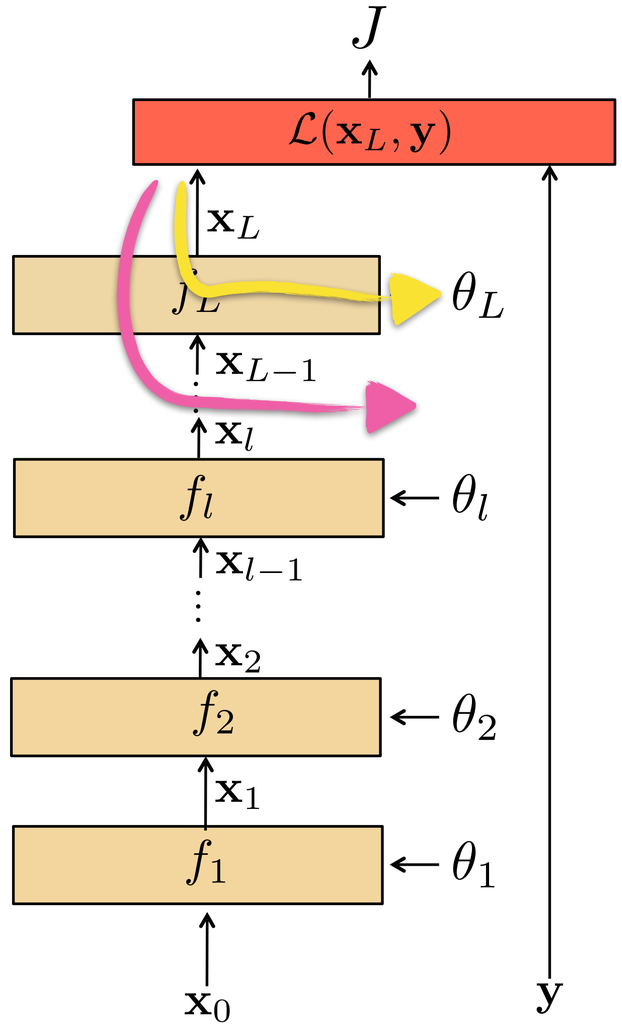
Computing gradients

To compute the parameter update for the last layer, we can use the **chain rule**:

$$\frac{\partial J}{\partial \theta_L} = \frac{\partial J}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \theta_L}$$

To compute the parameter update for the second-to-last layer:

$$\frac{\partial J}{\partial \theta_{L-1}} = \frac{\partial J}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}} \frac{\partial \mathbf{x}_{L-1}}{\partial \theta_{L-1}}$$

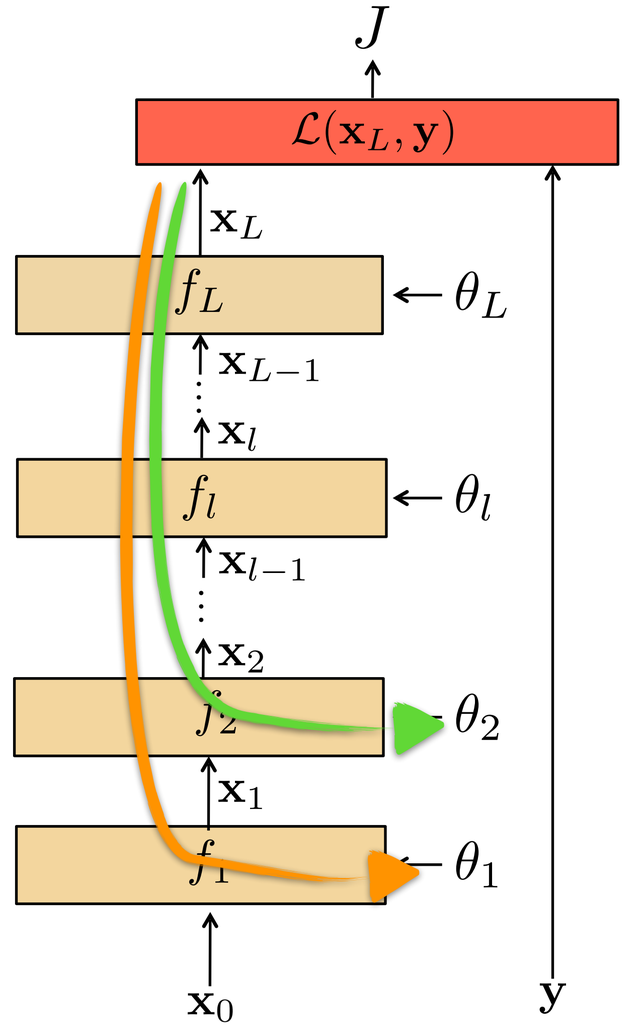


Computing gradients

To compute the parameter update for the 2nd and 1st layers:

$$\frac{\partial J}{\partial \theta_2} = \frac{\partial J}{\partial \mathbf{x}_L} \cdots \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}} \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} \frac{\partial \mathbf{x}_2}{\partial \theta_2}$$

$$\frac{\partial J}{\partial \theta_1} = \frac{\partial J}{\partial \mathbf{x}_L} \cdots \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}} \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1}{\partial \theta_1}$$



Blue terms are all shared! Can compute that product once and share it between these two equations.

The trick of backpropagation — reuse of computation

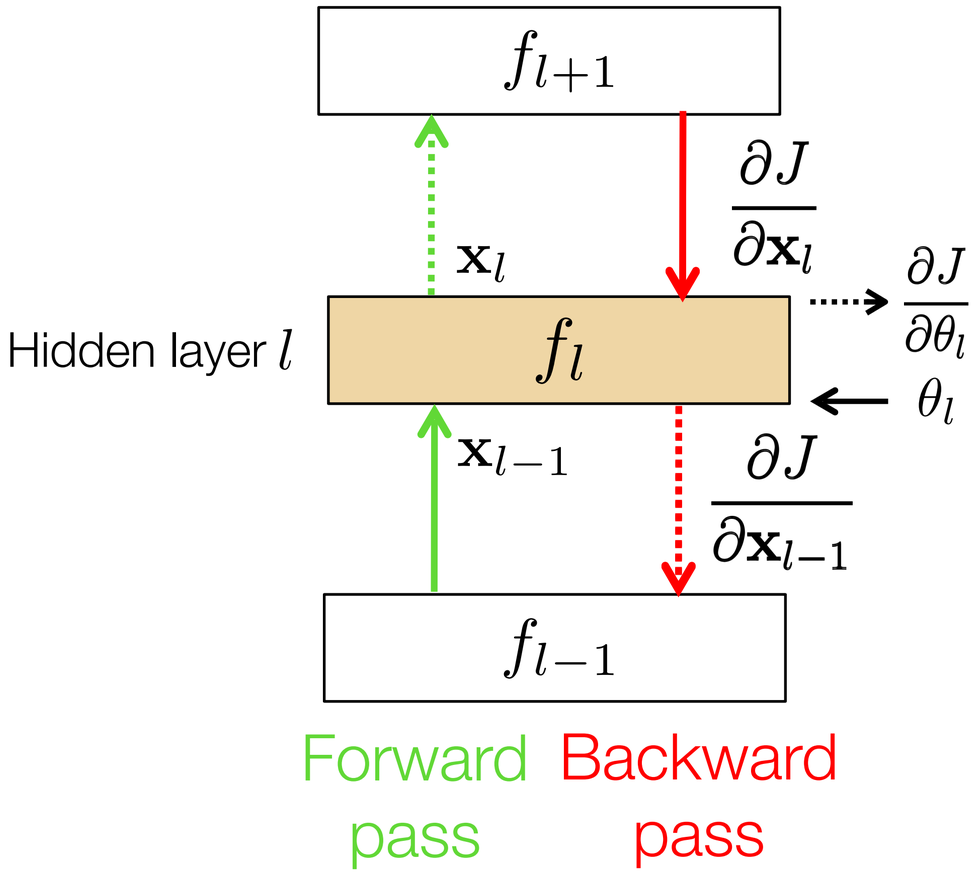
(aka dynamic programming)

Gradient w.r.t. loss at layer L-1 ⟶ $\frac{\partial J}{\partial \mathbf{x}_{L-1}} = \frac{\partial J}{\partial \mathbf{x}_L} \cdot \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_{L-1}}$ ⟵ Layer L's gradient

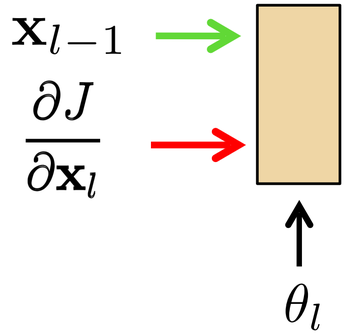
↑

Gradient w.r.t. loss at layer L

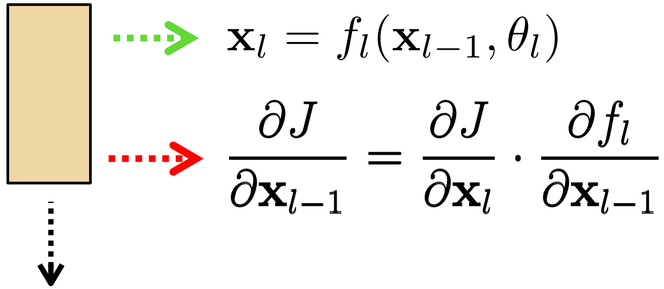
Backpropagation — Goal: to update parameters of layer l



- Layer l has three inputs (during training)



- And three outputs



- Given the inputs, we just need to evaluate:

$$f_l \qquad \frac{\partial f_l}{\partial \mathbf{x}_{l-1}} \qquad \frac{\partial f_l}{\partial \theta_l}$$

Backpropagation Summary

1. **Forward pass:** for each training example, compute the outputs for all layers:

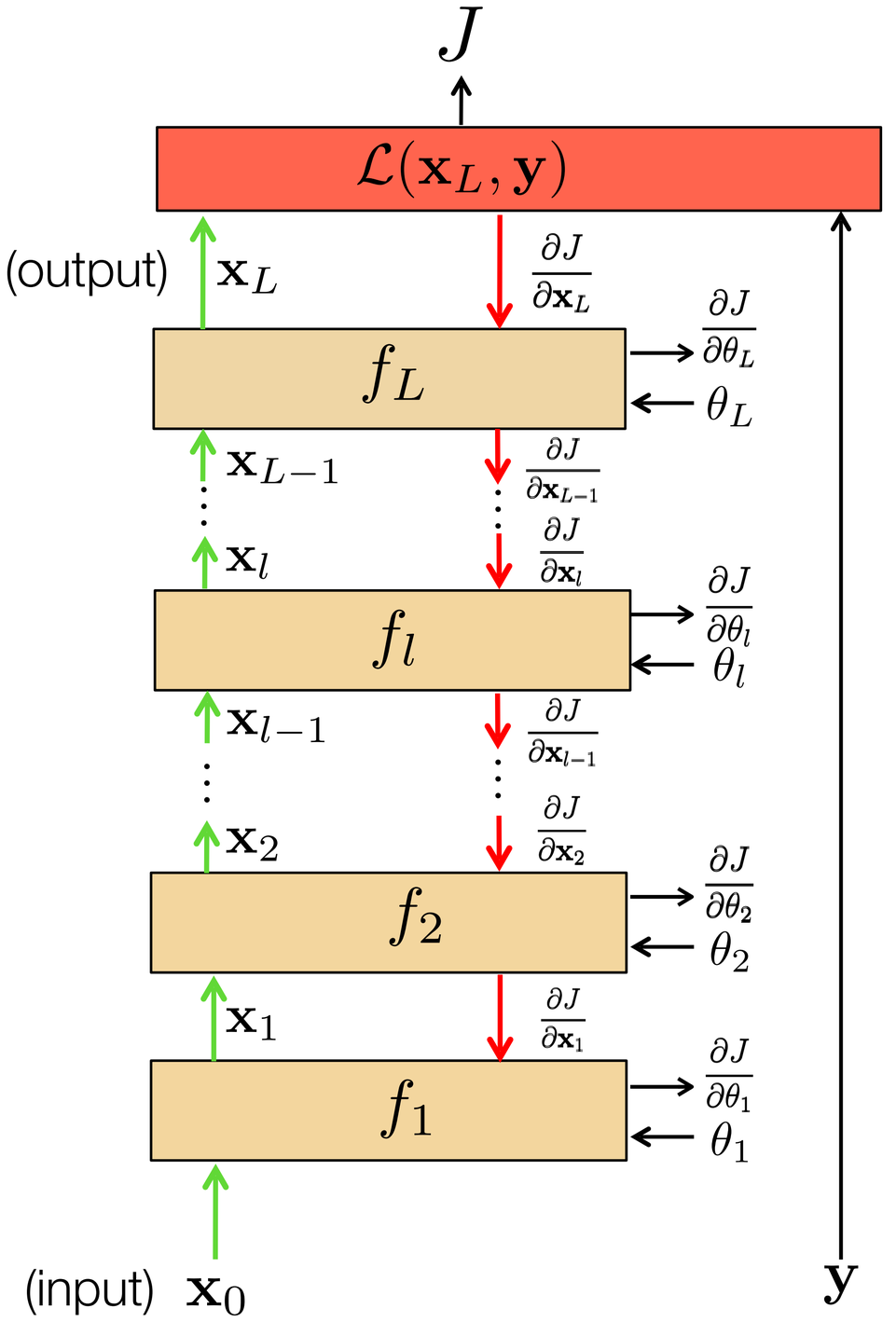
$$\mathbf{x}_l = f_l(\mathbf{x}_{l-1}, \theta_l)$$

2. **Backwards pass:** compute loss derivatives iteratively from top to bottom:

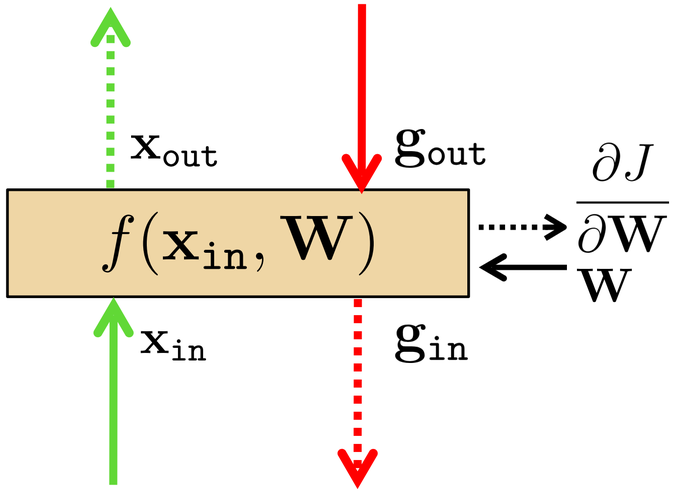
$$\frac{\partial J}{\partial \mathbf{x}_{l-1}} = \frac{\partial J}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l}{\partial \mathbf{x}_{l-1}}$$

3. **Parameter update:** Compute gradients w.r.t. weights, and update weights:

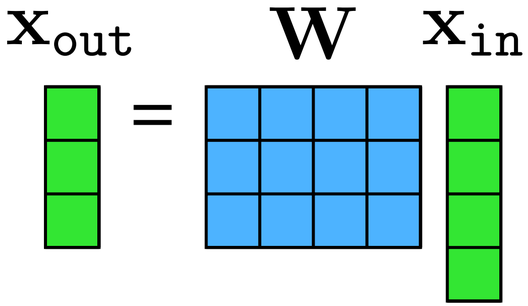
$$\frac{\partial J}{\partial \theta_l} = \frac{\partial J}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l}{\partial \theta_l}$$



Linear layer



- Forward propagation: $\mathbf{x}_{\text{out}} = f(\mathbf{x}_{\text{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\text{in}}$



With W being a matrix of size $|x_{out}| \times |x_{in}|$

- Backprop to input:

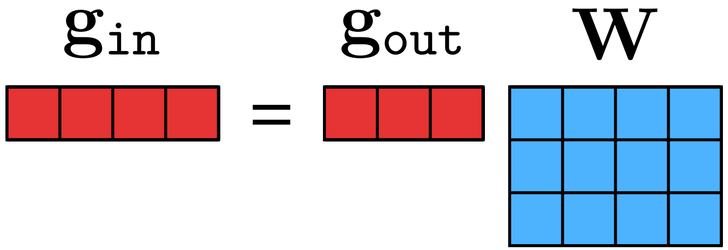
$$\mathbf{g}_{\text{in}} = \mathbf{g}_{\text{out}} \cdot \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{x}_{\text{in}}} = \mathbf{g}_{\text{out}} \cdot \frac{\partial \mathbf{x}_{\text{out}}}{\partial \mathbf{x}_{\text{in}}} \triangleq \mathbf{g}_{\text{out}} \cdot \mathbf{L}^{\mathbf{x}}$$

If we look at the i component of output x_{out}, with respect to the j component of the input, x_{in}:

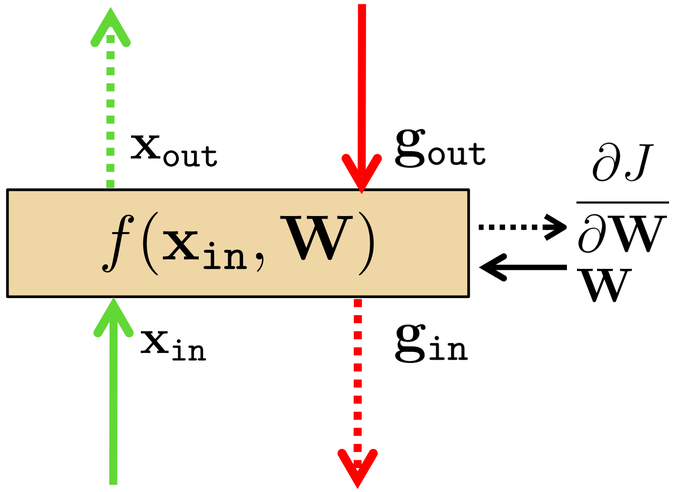
$$\frac{\partial \mathbf{x}_{\text{out}_i}}{\partial \mathbf{x}_{\text{in}_j}} = \mathbf{W}_{ij} \longrightarrow \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{x}_{\text{in}}} = \mathbf{W}$$

Therefore:

$$\boxed{\mathbf{g}_{\text{in}} = \mathbf{g}_{\text{out}} \cdot \mathbf{W}}$$



Linear layer



- Forward propagation: $\mathbf{x}_{\text{out}} = f(\mathbf{x}_{\text{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\text{in}}$
- Backprop to weights:

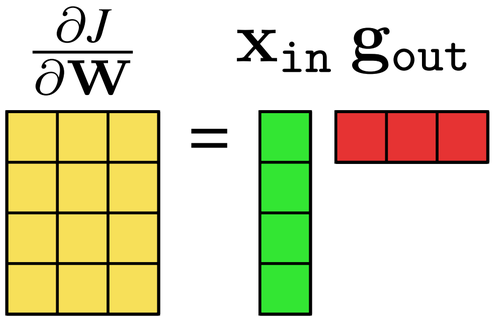
$$\frac{\partial J}{\partial \mathbf{W}} = \mathbf{g}_{\text{out}} \cdot \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{W}} = \mathbf{g}_{\text{out}} \cdot \frac{\partial \mathbf{x}_{\text{out}}}{\partial \mathbf{W}}$$

If we look at how the parameter W_{ij} changes the cost, only the i component of the output will change, therefore:

$$\frac{\partial J}{\partial \mathbf{W}_{ij}} = \frac{\partial J}{\partial \mathbf{x}_{\text{out}_i}} \cdot \frac{\partial \mathbf{x}_{\text{out}_i}}{\partial \mathbf{W}_{ij}} = \frac{\partial J}{\partial \mathbf{x}_{\text{out}_i}} \cdot \mathbf{x}_{\text{in}_j}$$

$$\frac{\partial \mathbf{x}_{\text{out}_i}}{\partial \mathbf{W}_{ij}} = \mathbf{x}_{\text{in}_j}$$

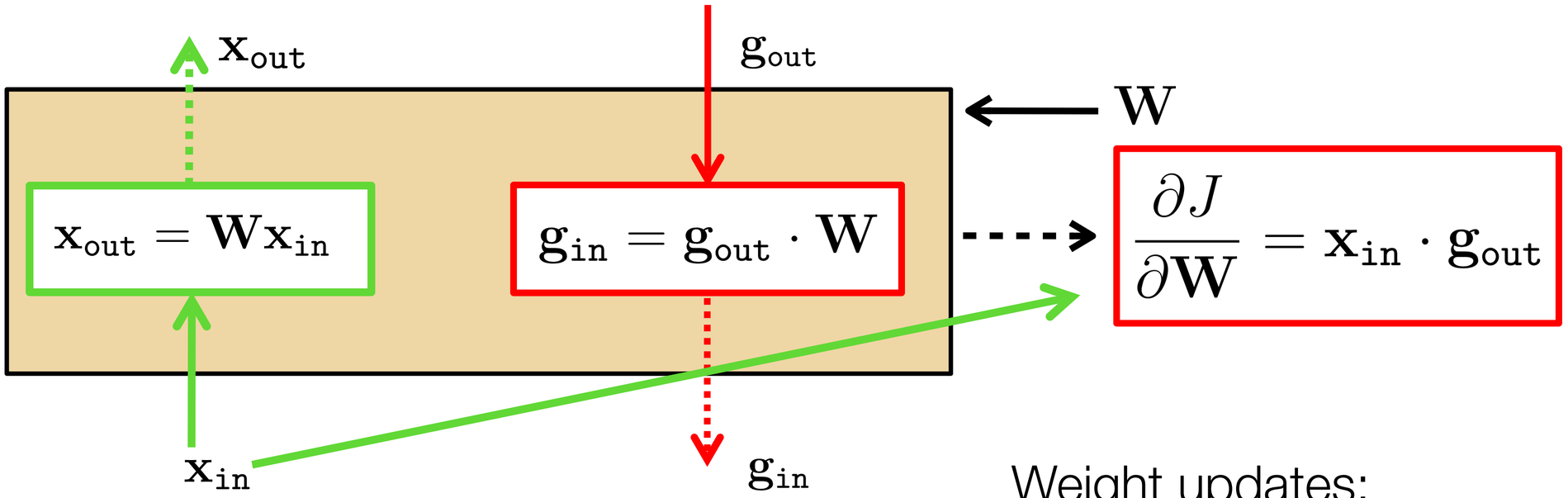
$$\frac{\partial J}{\partial \mathbf{W}} = \mathbf{x}_{\text{in}} \cdot \frac{\partial J}{\partial \mathbf{x}_{\text{out}}} = \mathbf{x}_{\text{in}} \cdot \mathbf{g}_{\text{out}}$$



And now we can update the weights:

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^{T}$$

Linear layer

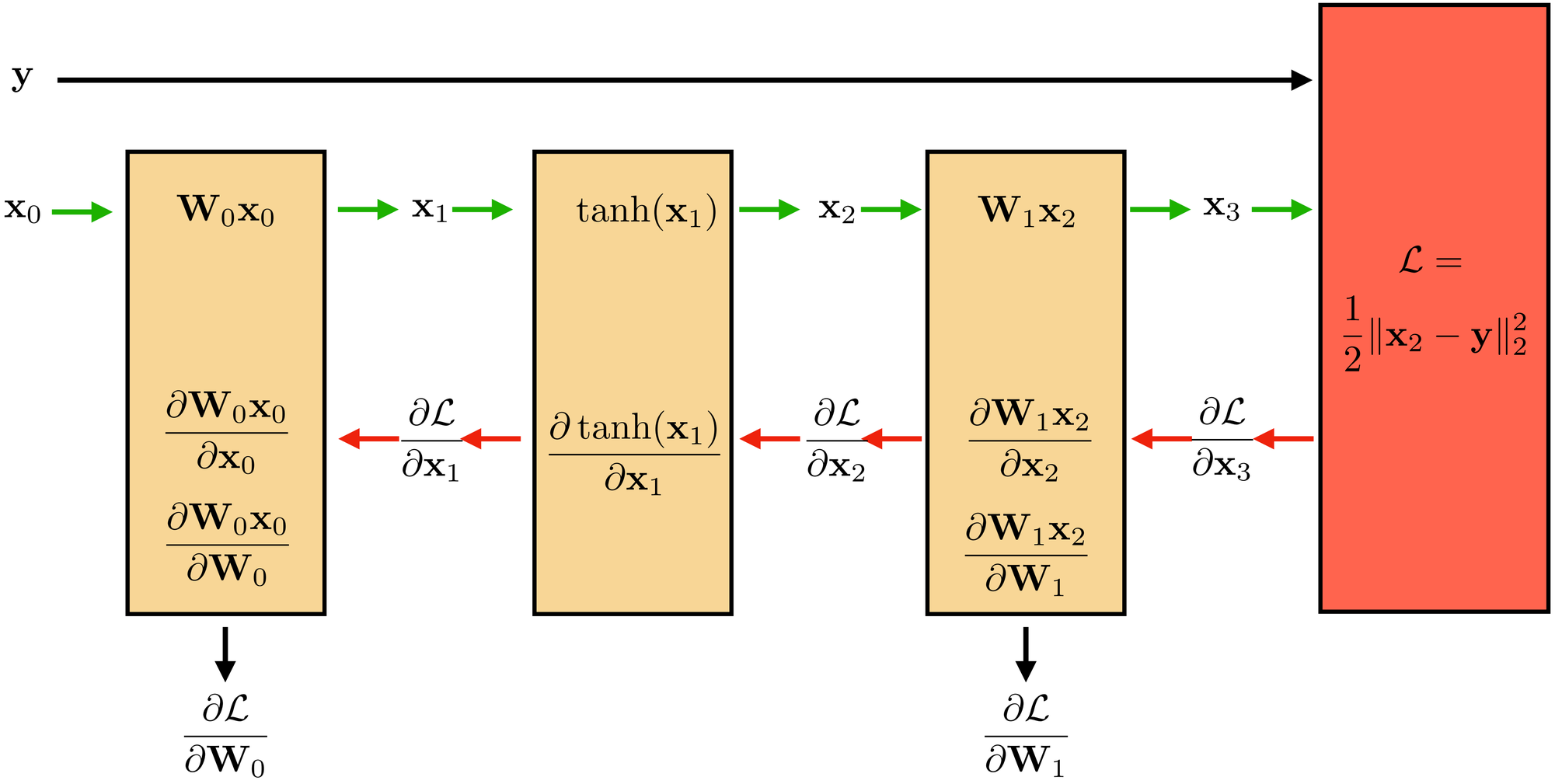


Weight updates:

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^{T}$$

Step by step solution

First, let's rewrite the network using the modular block notation:



We need to compute all these terms simply so we can find the weight updates at the bottom.

First we compute the derivative of the loss with respect to the output:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3} = \mathbf{x}_3 - \mathbf{y}$$

Now, by the chain rule, we can derive equations, working *backwards*, for each remaining term we need:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}\frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}\mathbf{W}_1$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}\frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}\frac{\partial \tanh(\mathbf{x}_1)}{\partial \mathbf{x}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}(1 - \tanh^2(\mathbf{x}_1))$$

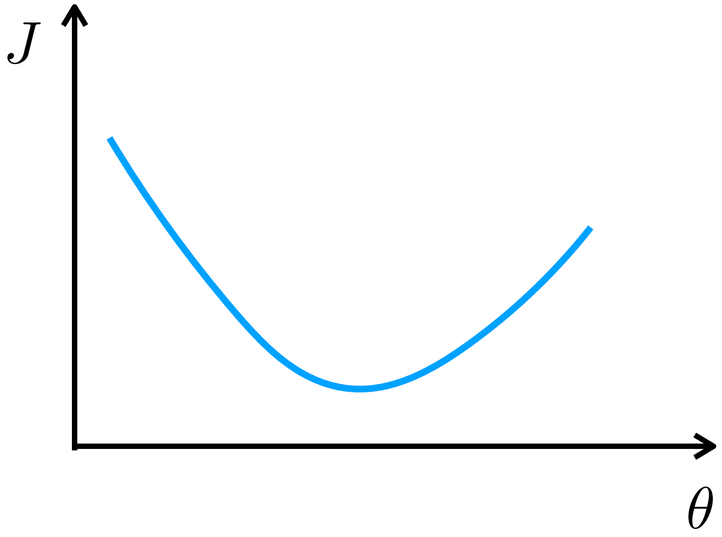
ending up with our two gradients needed for the weight update:

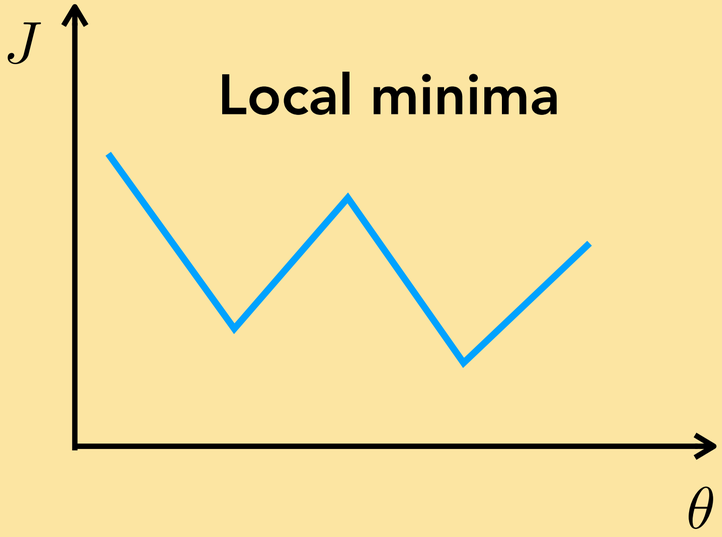
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}\frac{\partial \mathbf{x}_1}{\partial \mathbf{W}_0} = \mathbf{x}_0\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}$$

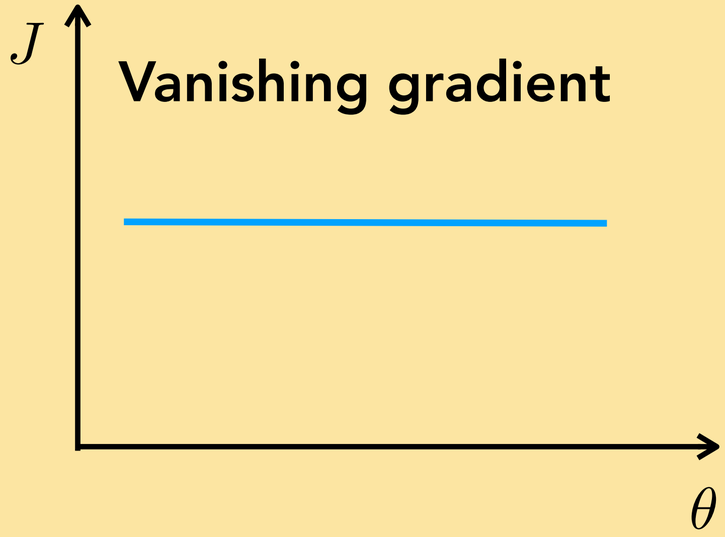
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}\frac{\partial \mathbf{x}_3}{\partial \mathbf{W}_1} = \mathbf{x}_2\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}$$

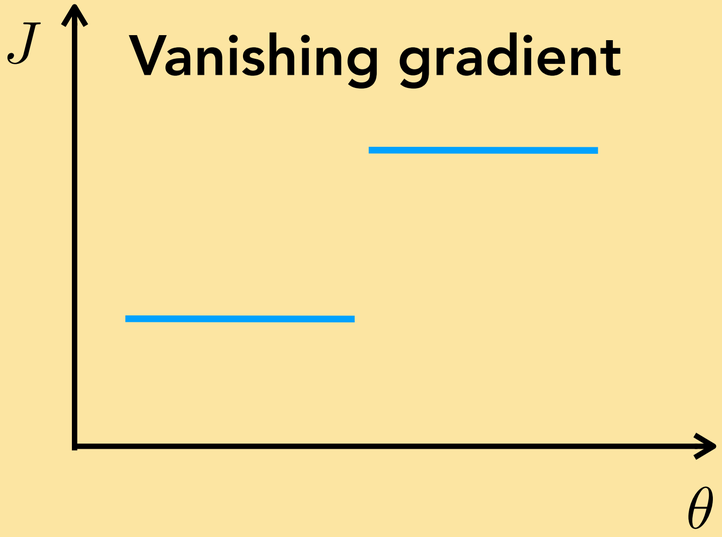
Notice the ordering of the two terms being multiplied here. The notation hides the details but you can write out all the indices to see that this is the correct ordering — or just check that the dimensions work out.

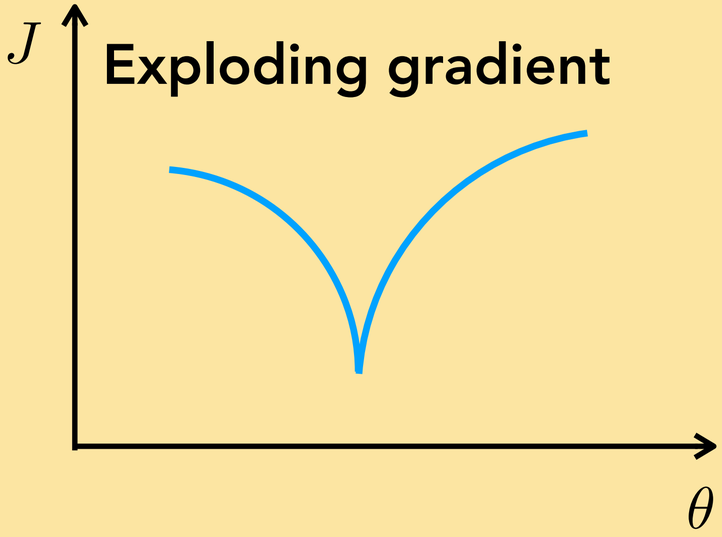
Which will be hard to optimize?

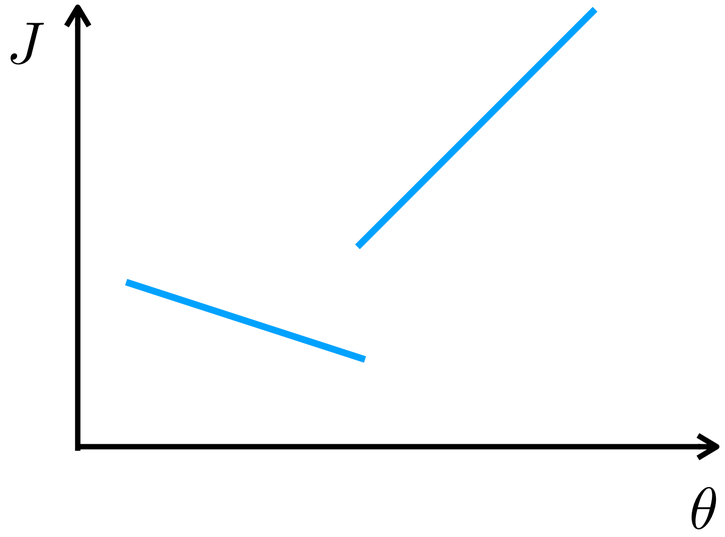












Stochastic Gradient Descent (SGD)

- Want to minimize overall loss function **J**, which is sum of individual losses over each example.
- In Stochastic gradient descent, compute gradient on sub-set (batch) of data.
  - If batchsize=1 then θ is updated after each example.
  - If batchsize=N (full set) then this is standard gradient descent.
- Gradient direction is noisy, relative to average over all examples (standard gradient descent).
- Advantages
  - Faster: approximate total gradient with small sample
  - Implicit regularizer
- Disadvantages
  - High variance, unstable updates

Momentum

- A heavy ball rolling down a hill, gains speed.
- Gradient steps biased to continue in direction of previous update:

$$\theta^{t+1} \leftarrow \theta^t - \eta \nabla f(\theta^t) - \alpha m^t$$

- Can help or hurt. Strength of momentum is a hyperparam.