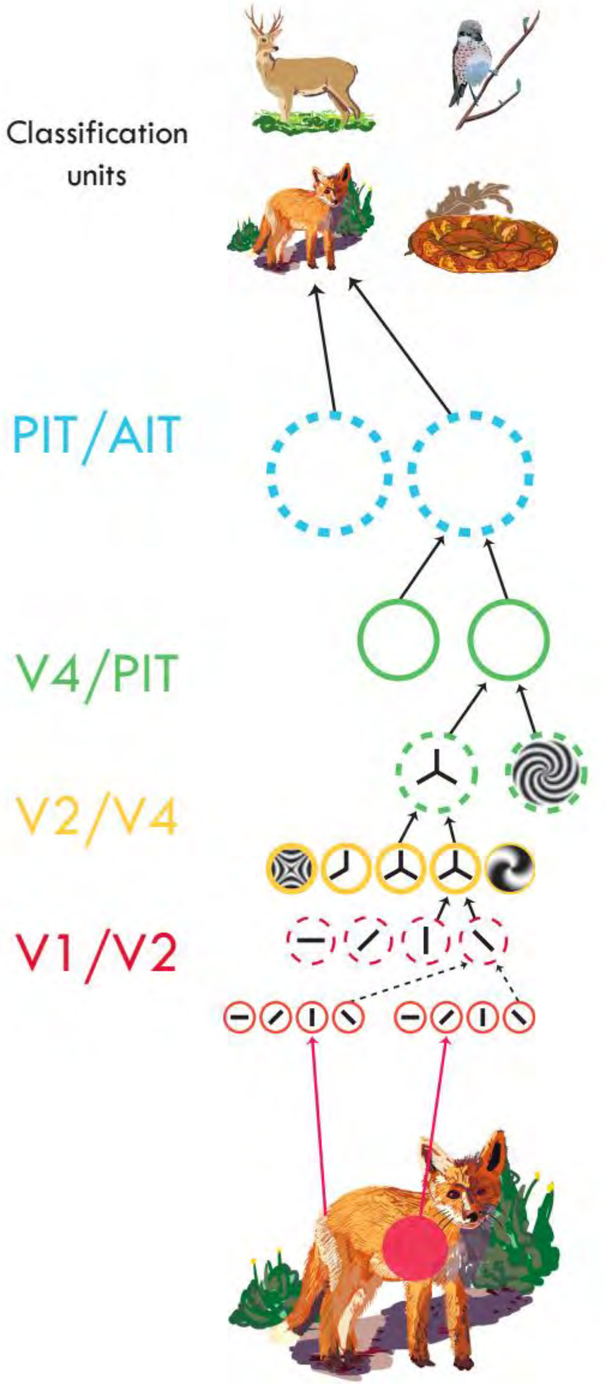
人工智能

深度学习基础I：神经网络

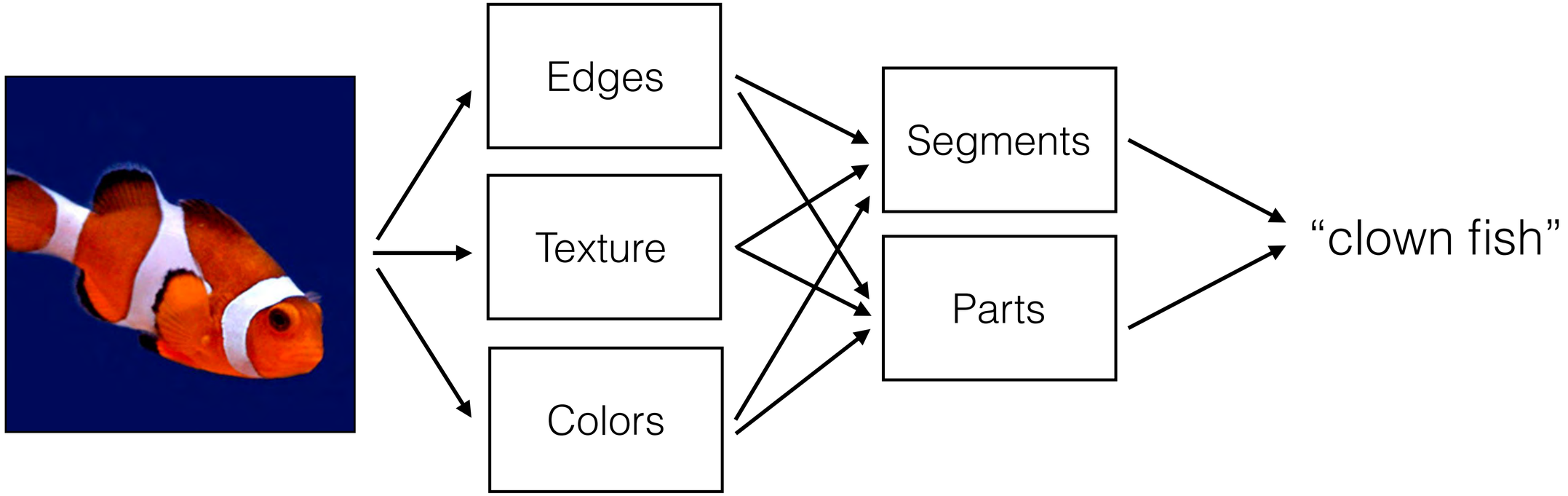
COMP130207.01

[Slide Credit to Phillip Isola]

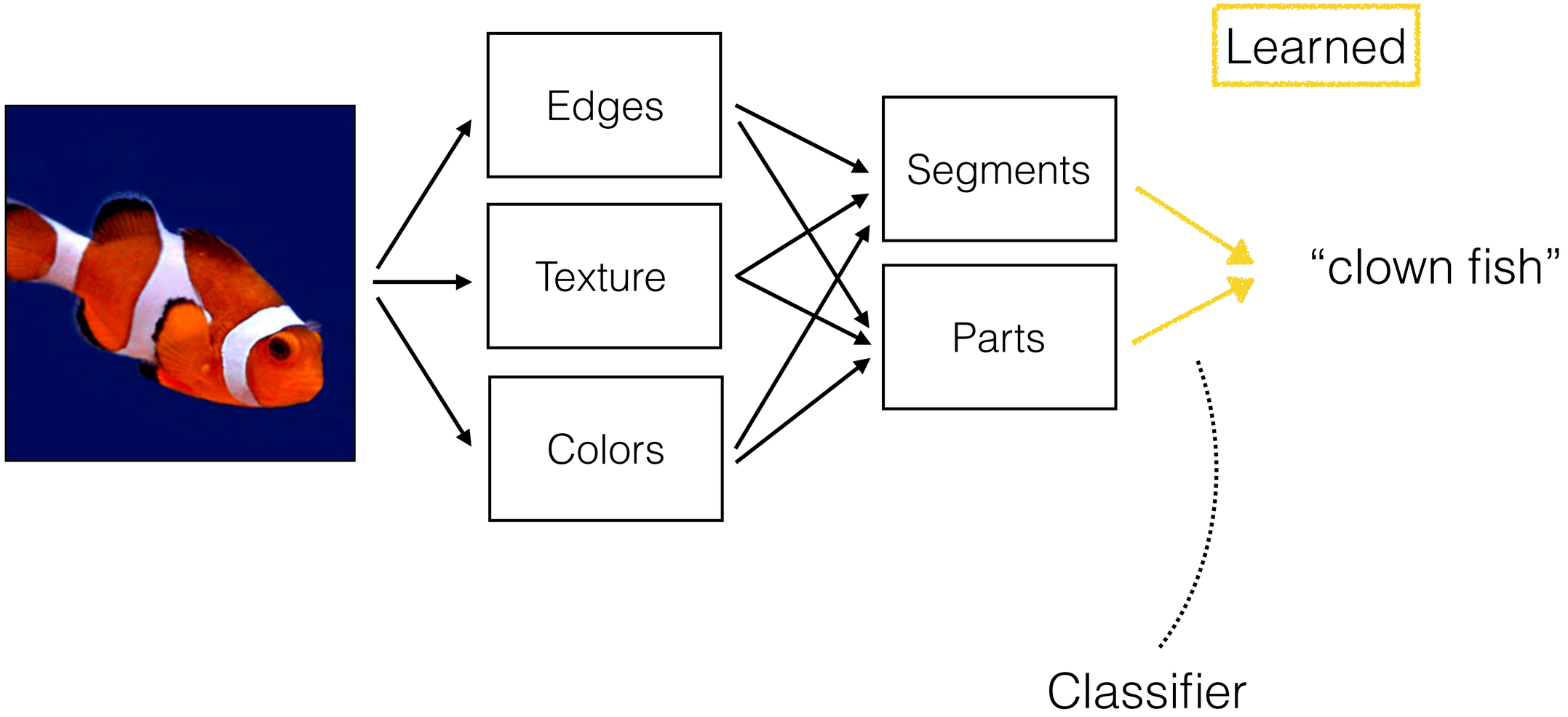


Serre, 2014

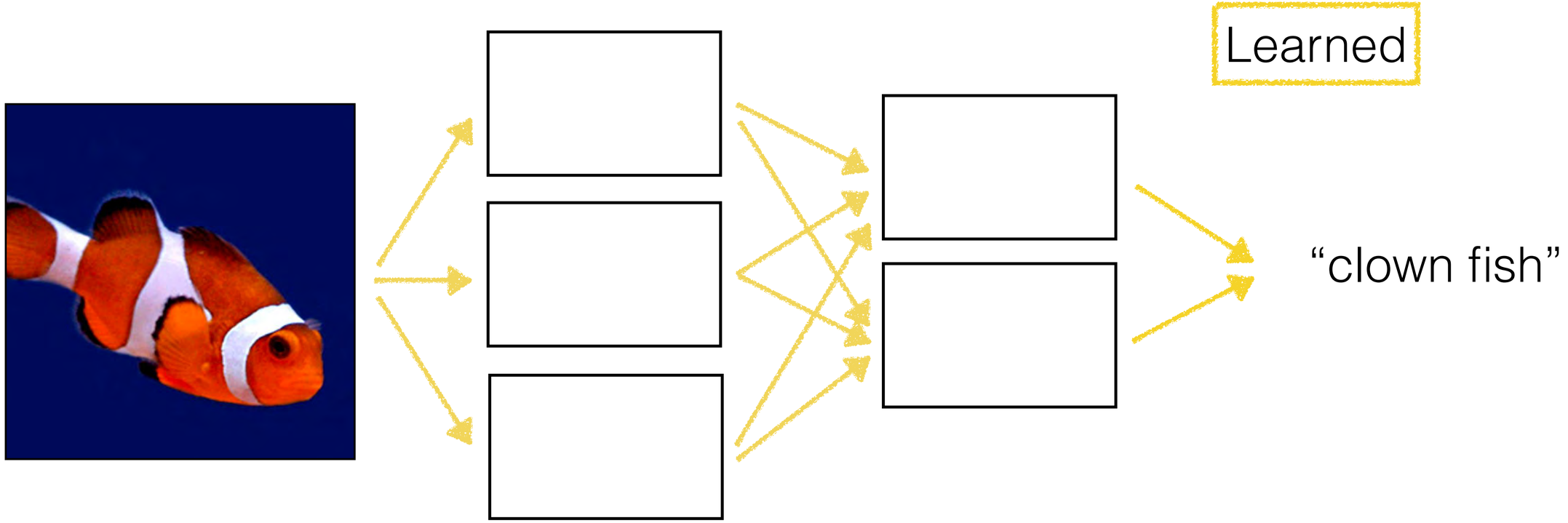
图像分类



图像分类

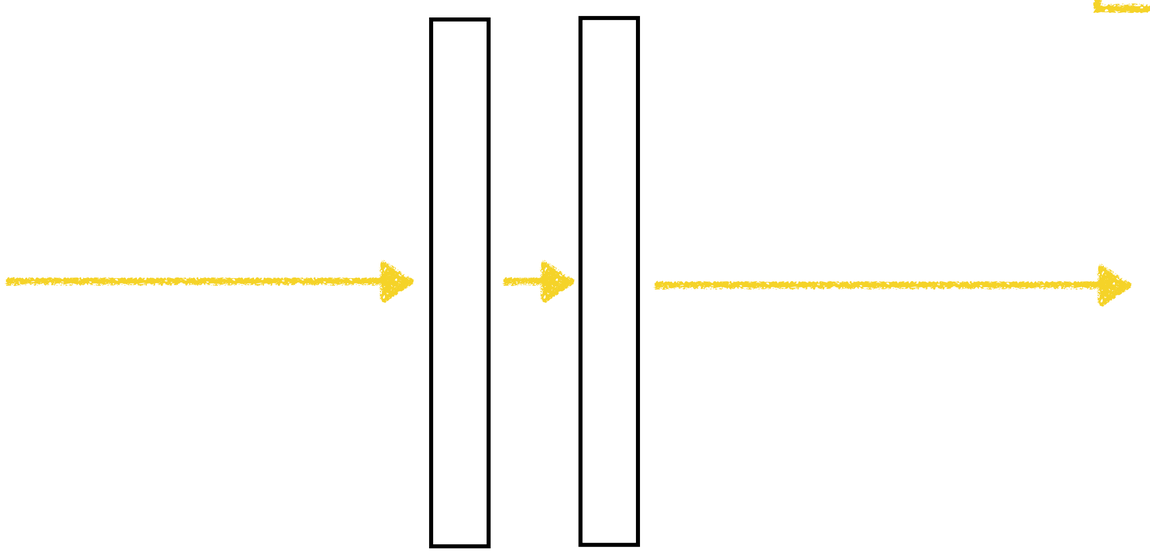

Learned
Edges
Texture
Colors
Segments
Parts
“clown fish”
Classifier

图像分类



深度学习

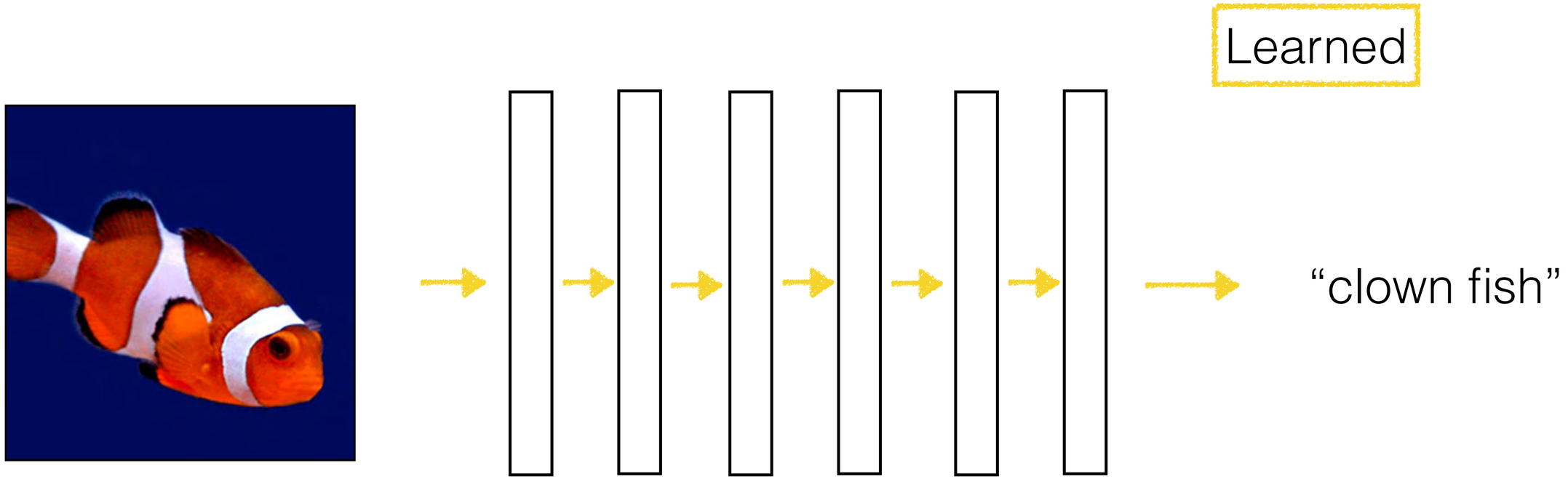
Learned



“clown fish”

Neural net

深度学习



Deep neural net

深度学习

Learned

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

"clown fish"

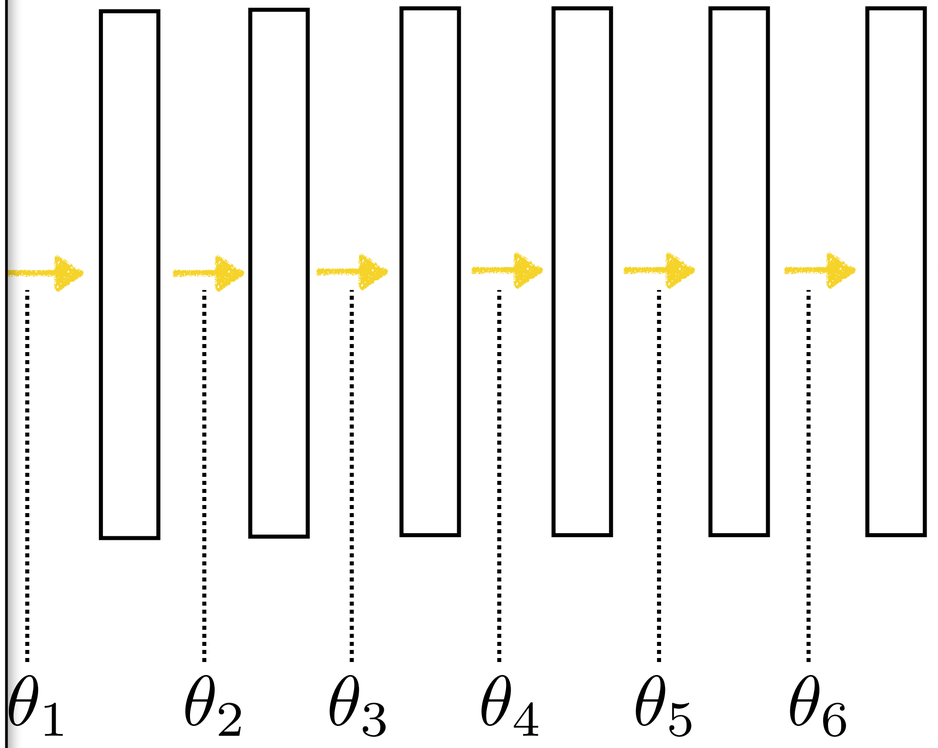
Training data

$\mathbf{x}$ y


"Fish"


"Grizzly"

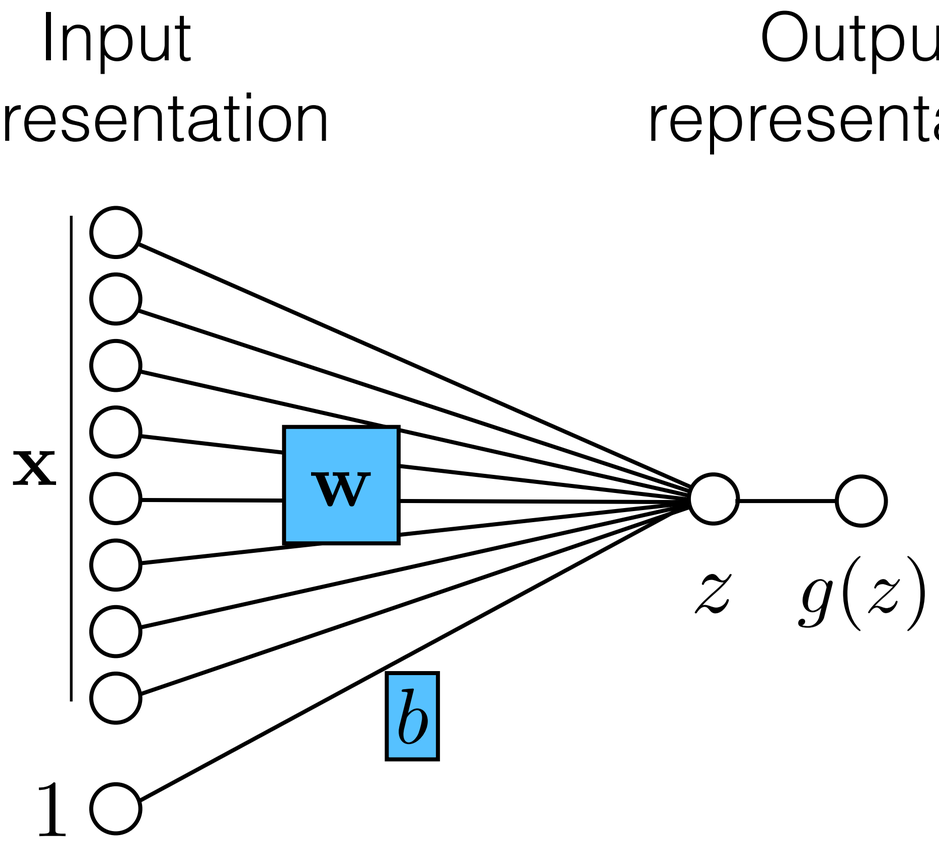

"Chameleon"



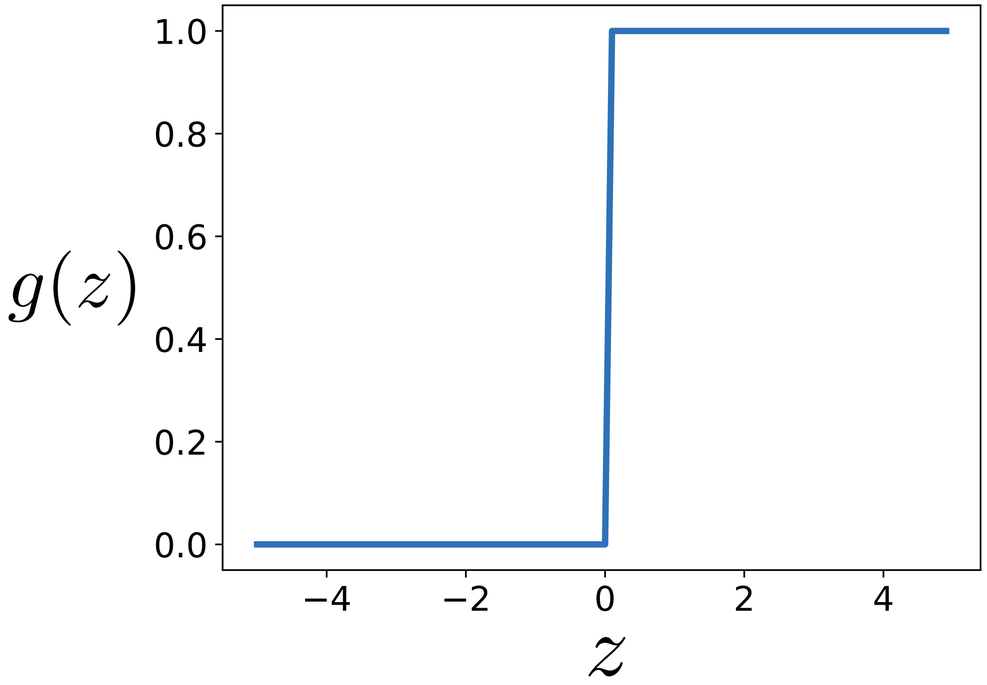
Loss $\mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$

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

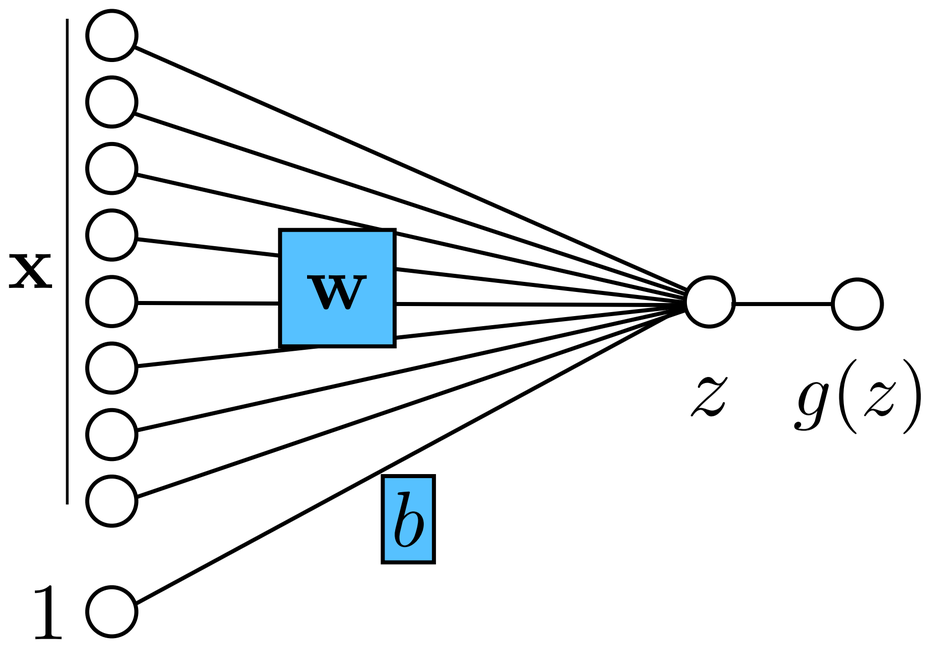
一层神经网络



$$g(z) = \begin{cases} 1, & \text{if} \quad z > 0 \\ 0, & \text{otherwise} \end{cases}$$

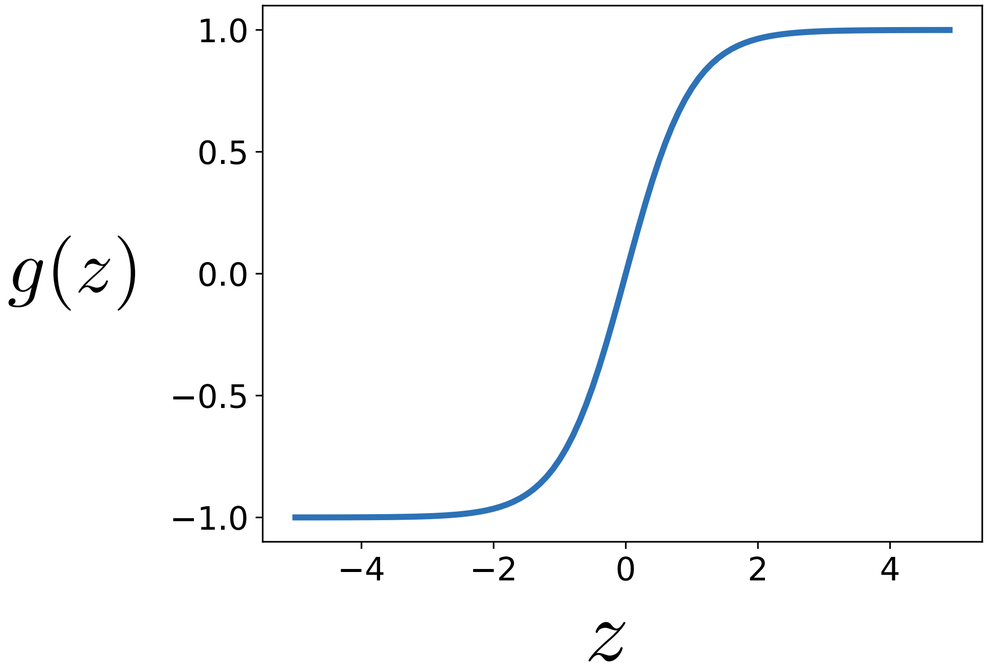


一层神经网络-非线性单元



Tanh

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

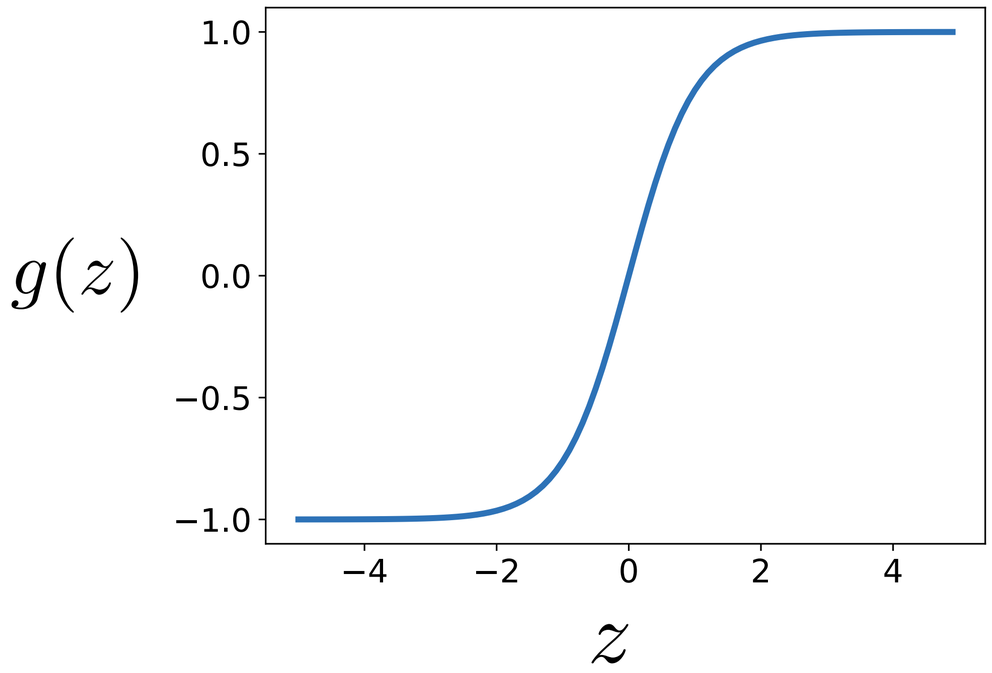


一层神经网络-非线性单元

- Bounded between [-1,+1]
- Saturation for large +/- inputs
- Gradients go to zero
- Outputs centered at 0
- tanh(z) = 2 sigmoid(2z) –1

Tanh

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

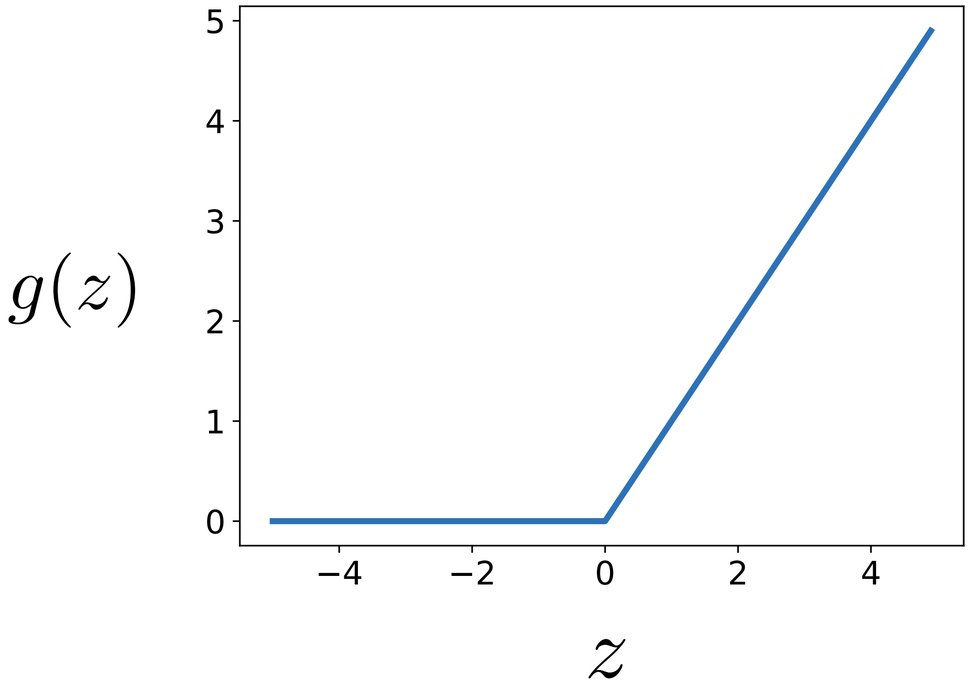


一层神经网络-非线性单元

- Unbounded output (on positive side)
- Efficient to implement: $\frac{\partial g}{\partial z} = \begin{cases} 0, & \text{if} \quad z < 0 \\ 1, & \text{if} \quad z \geq 0 \end{cases}$
- Also seems to help convergence (see 6x speedup vs tanh in [Krizhevsky et al.])
- Drawback: if strongly in negative region, unit is dead forever (no gradient).
- Default choice: widely used in current models.

Rectified linear unit (ReLU)

$$g(z) = \max(0, z)$$

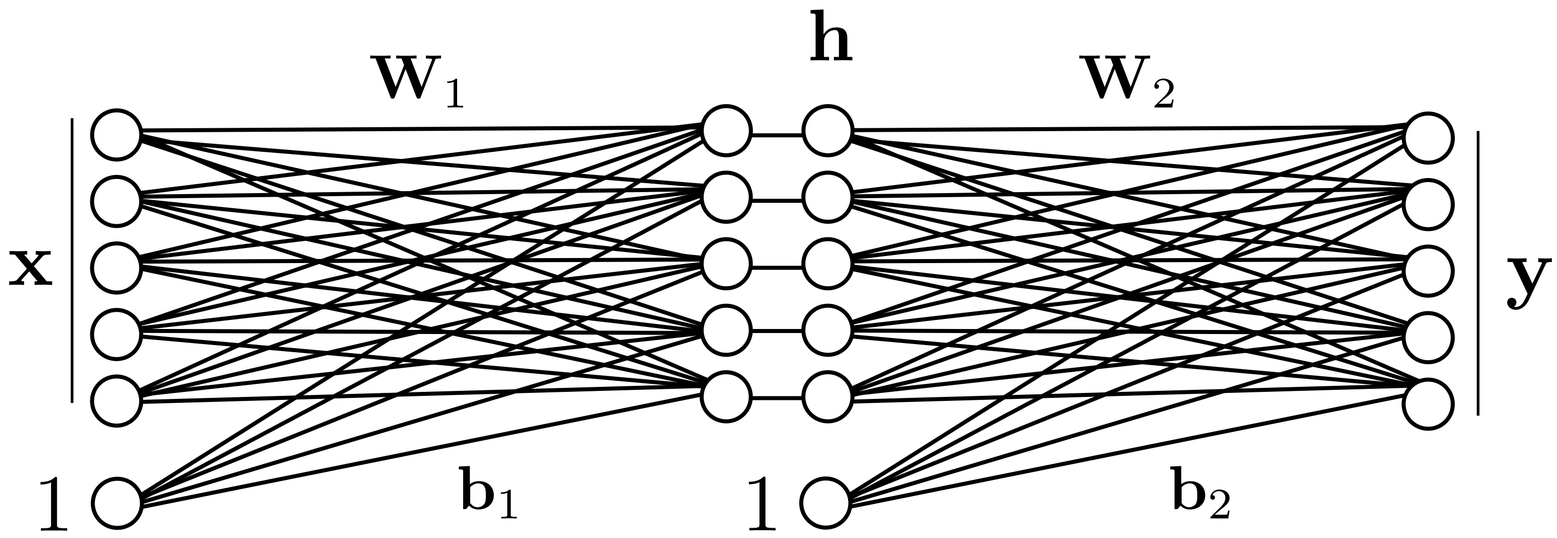


神经网络堆叠

Input representation

Intermediate representation

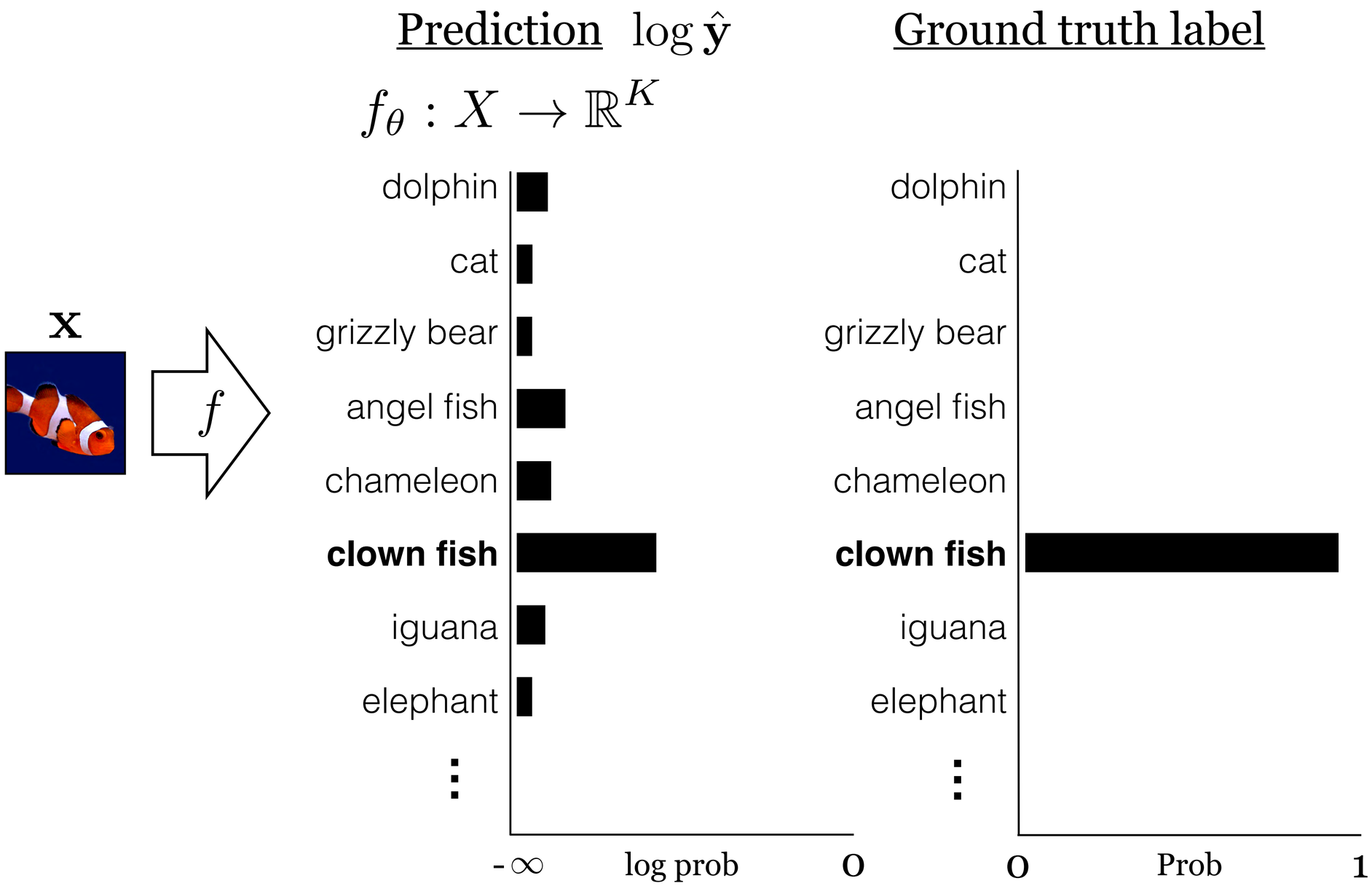
Output representation



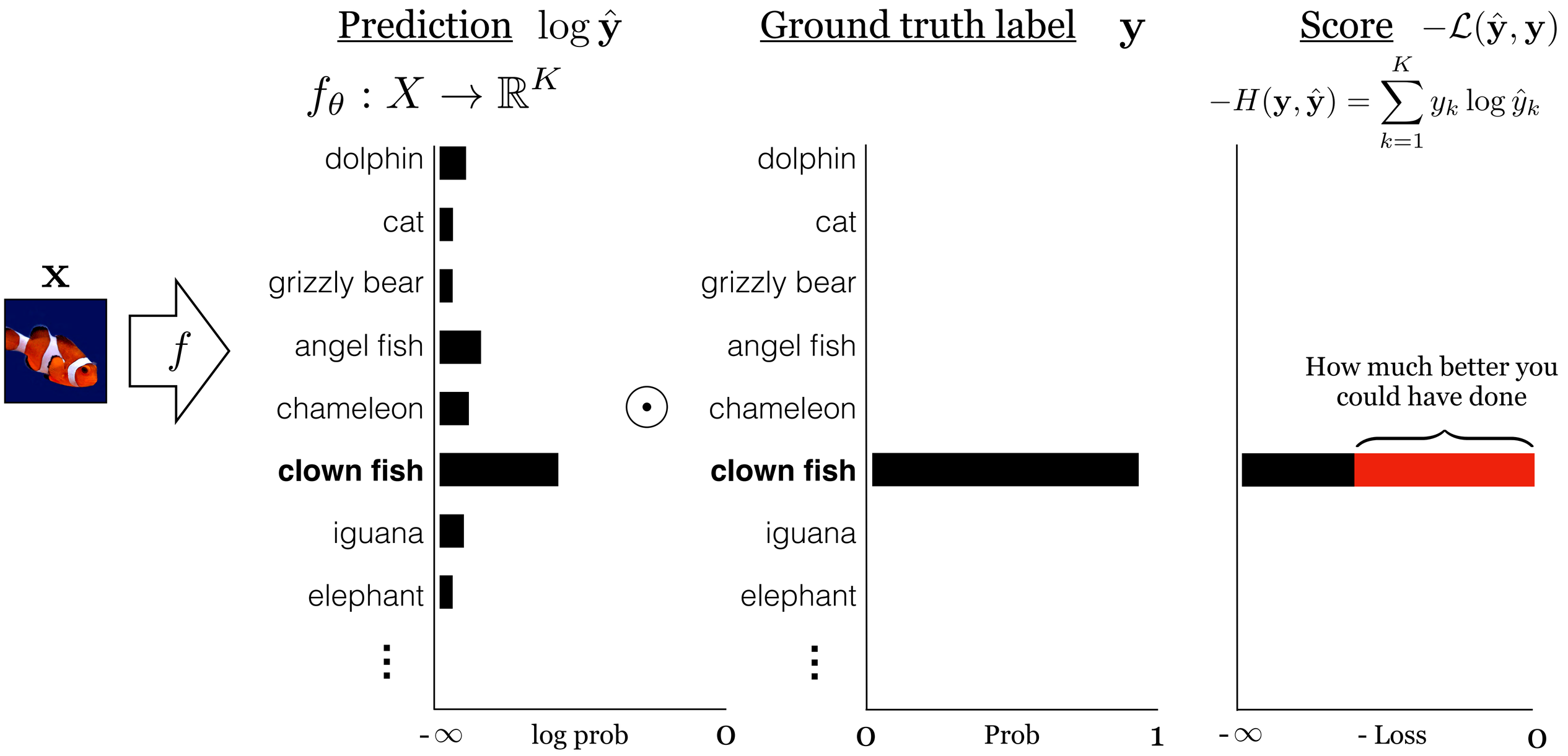
$$\mathbf{h} = g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \qquad \mathbf{y} = g(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2)$$

$$\theta = \{\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{b}_1, \ldots, \mathbf{b}_L\}$$

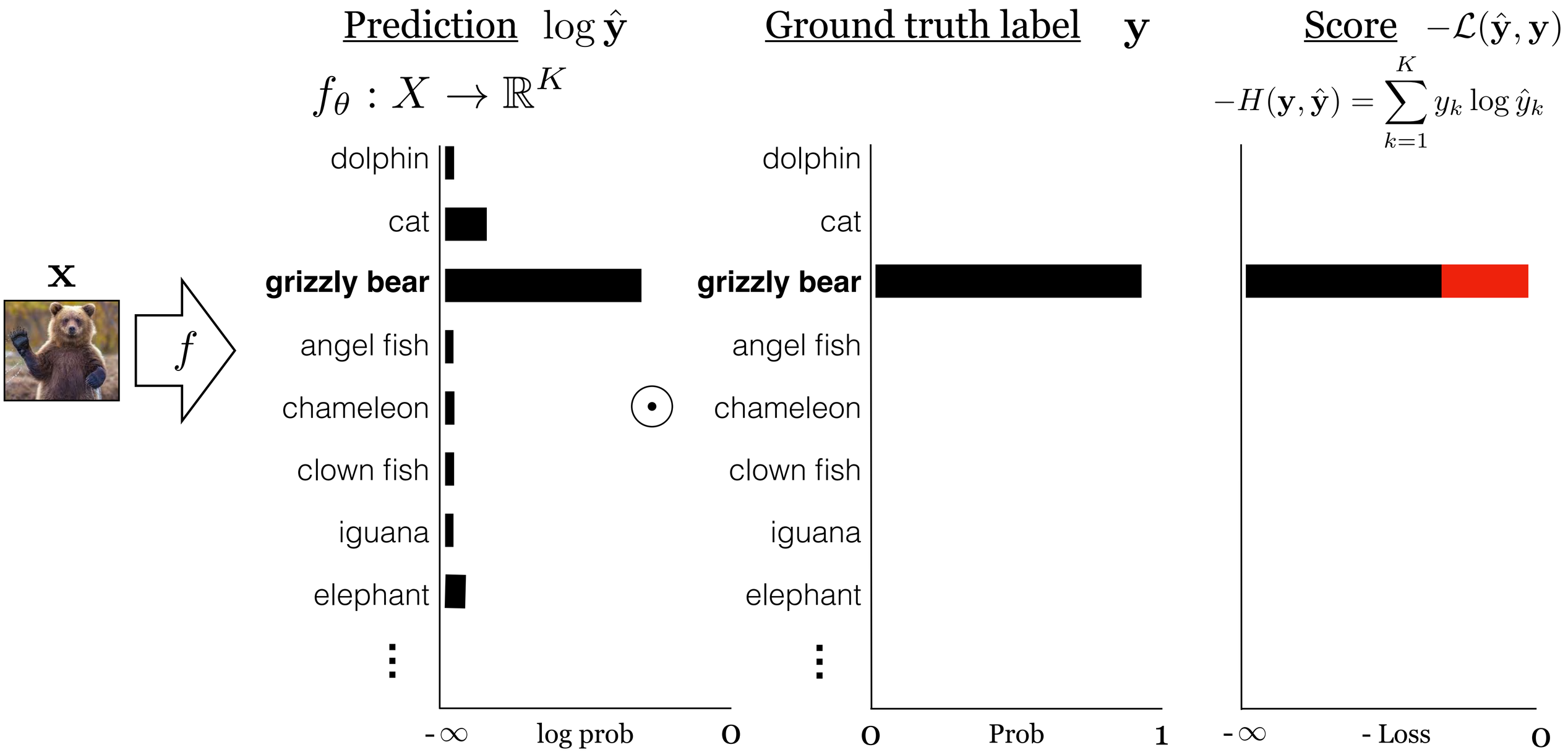
预测结果


Prediction $\log \hat{\mathbf{y}}$
Ground truth label
$f_\theta : X \to \mathbb{R}^K$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
$-\infty$
log prob
0
Prob
1

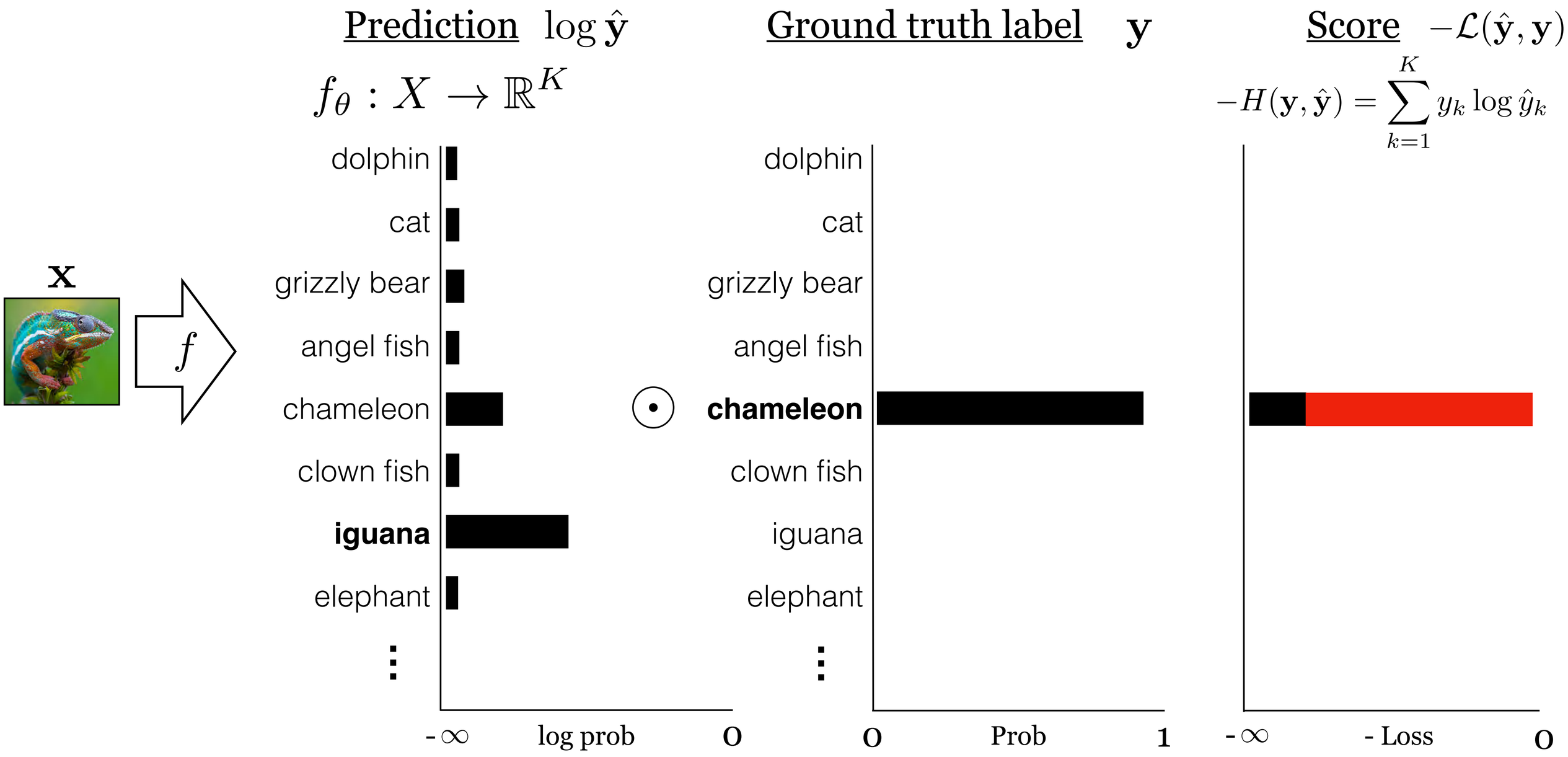
预测结果


Prediction $\log \hat{\mathbf{y}}$
Ground truth label $\mathbf{y}$
Score $-\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
$f_\theta : X \to \mathbb{R}^K$
$-H(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{K} y_k \log \hat{y}_k$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
How much better you could have done
$-\infty$
log prob
0
0
Prob
1
$-\infty$
- Loss
0

预测结果


Prediction $\log \hat{\mathbf{y}}$
Ground truth label $\mathbf{y}$
Score $-\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
$f_\theta : X \to \mathbb{R}^K$
$-H(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{K} y_k \log \hat{y}_k$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
$-\infty$
log prob
0
0
Prob
1
$-\infty$
- Loss
0

预测结果


Prediction $\log \hat{\mathbf{y}}$
Ground truth label $\mathbf{y}$
Score $-\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$
$f_\theta : X \to \mathbb{R}^K$
$-H(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{k=1}^{K} y_k \log \hat{y}_k$
$\mathbf{x}$
f
dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
$-\infty$
log prob
0
0
Prob
1
$-\infty$
- Loss
0

预测结果

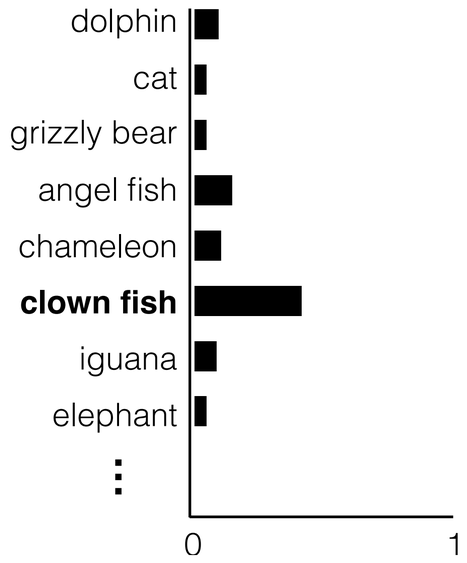
$$f_\theta : X \to \mathbb{R}^K$$

$$\mathbf{z} = f_\theta(\mathbf{x})$$ ← **logits**: vector of K scores, one for each class

$$\hat{\mathbf{y}} = \texttt{softmax}(\mathbf{z})$$ ← squash into a non-negative vector that sums to 1 — i.e. **a probability mass function!**

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

$\hat{\mathbf{y}} =$



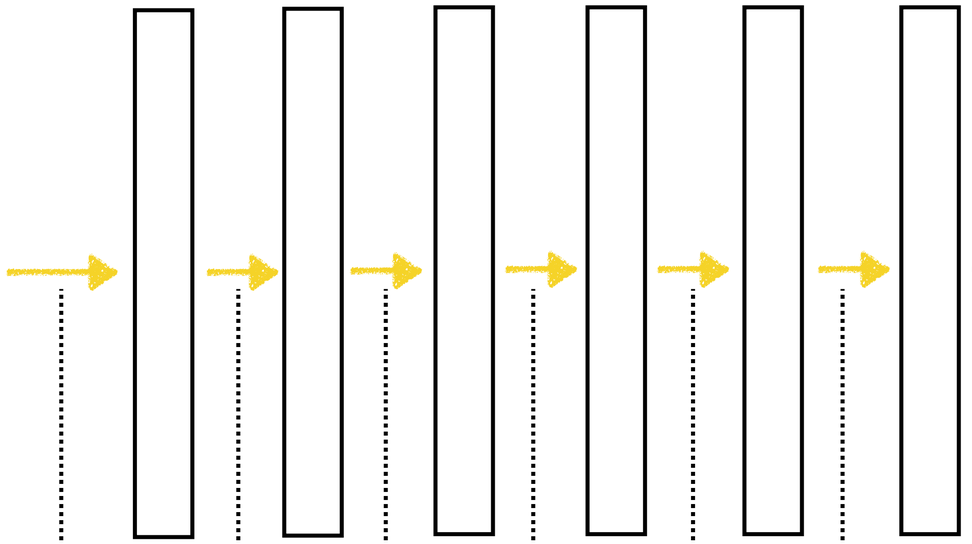
深度学习

Learned

$\mathbf{y}^{(1)}$

"clown fish"

$\mathbf{x}^{(1)}$



Loss

$\mathcal{L}(f_\theta(\mathbf{x}^{(1)}), \mathbf{y}^{(1)})$

θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

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

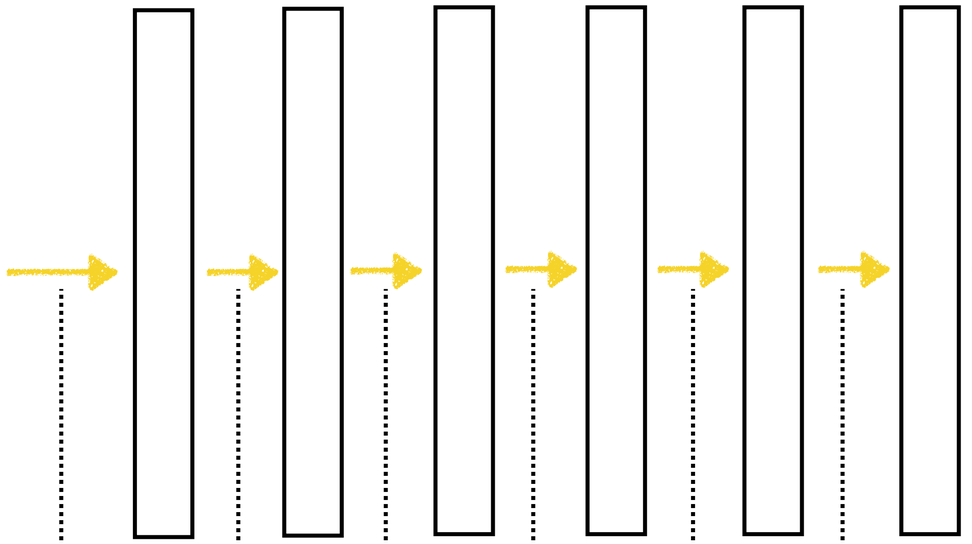
深度学习

Learned

$\mathbf{y}^{(2)}$

"grizzly bear"

$\mathbf{x}^{(2)}$



Loss

$\mathcal{L}(f_\theta(\mathbf{x}^{(2)}), \mathbf{y}^{(2)})$

θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

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

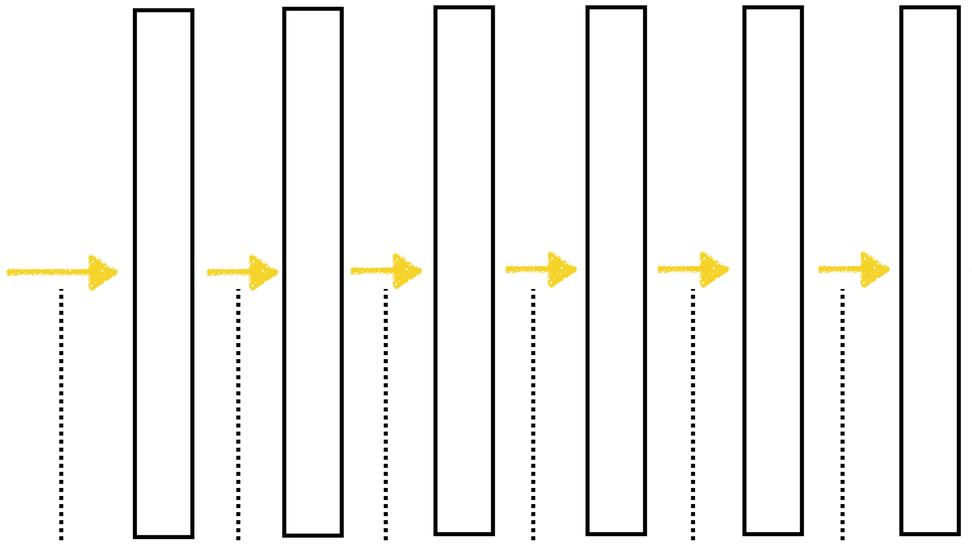
深度学习

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

"chameleon"

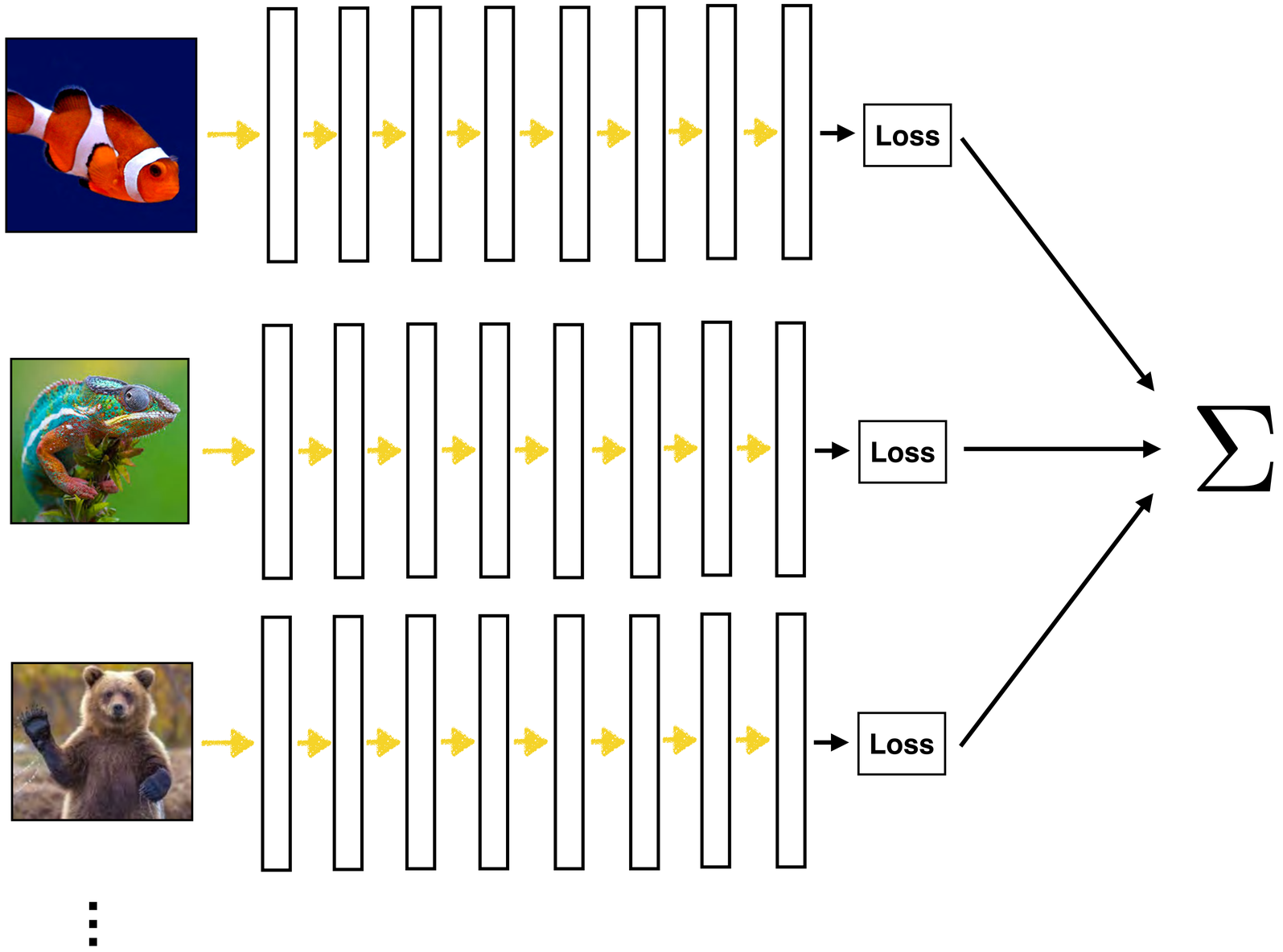
Learned

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

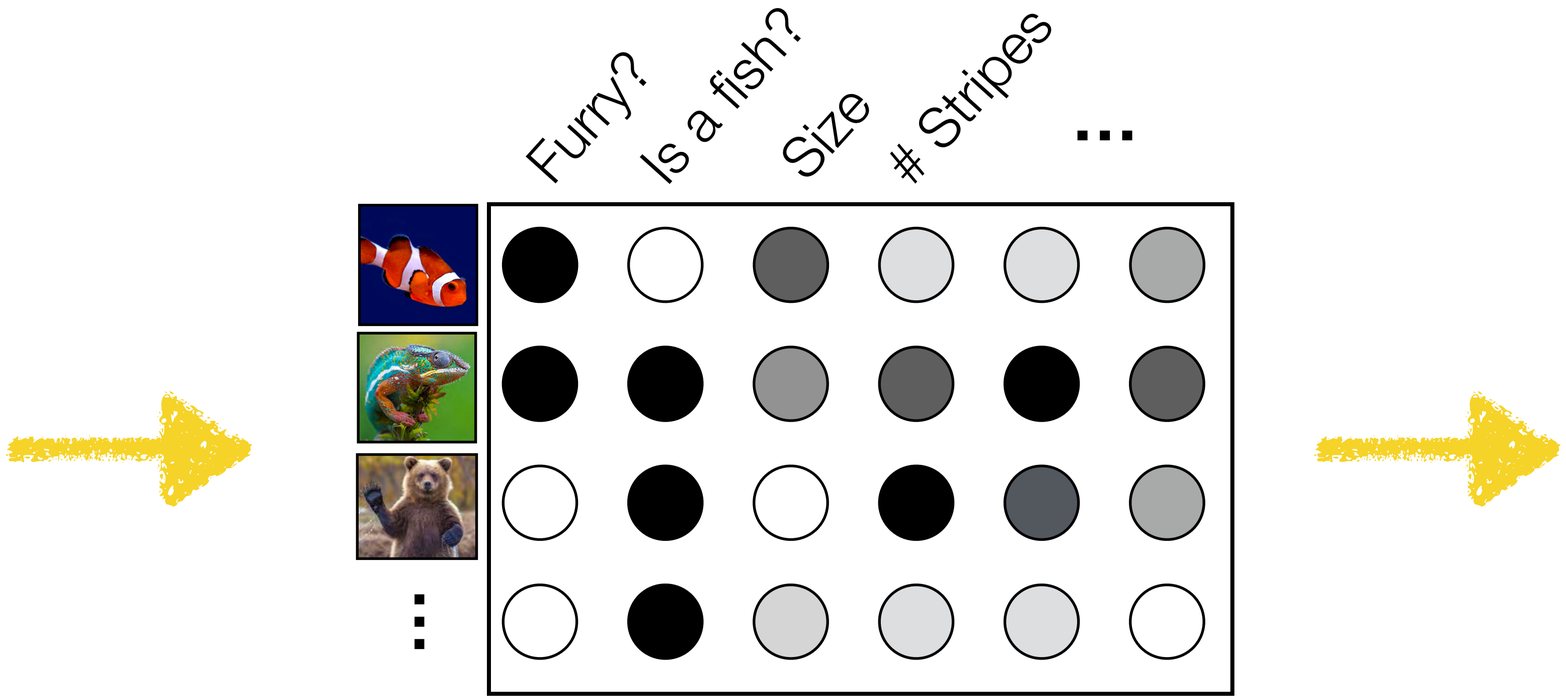


$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$$

深度学习（Batch）

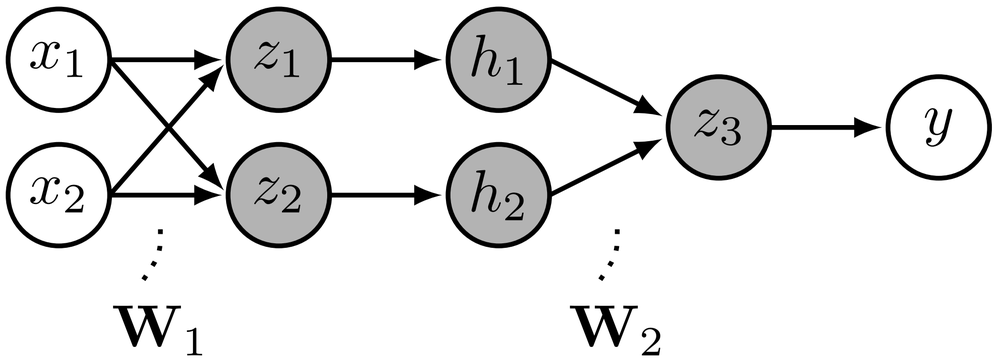


张量Tensor



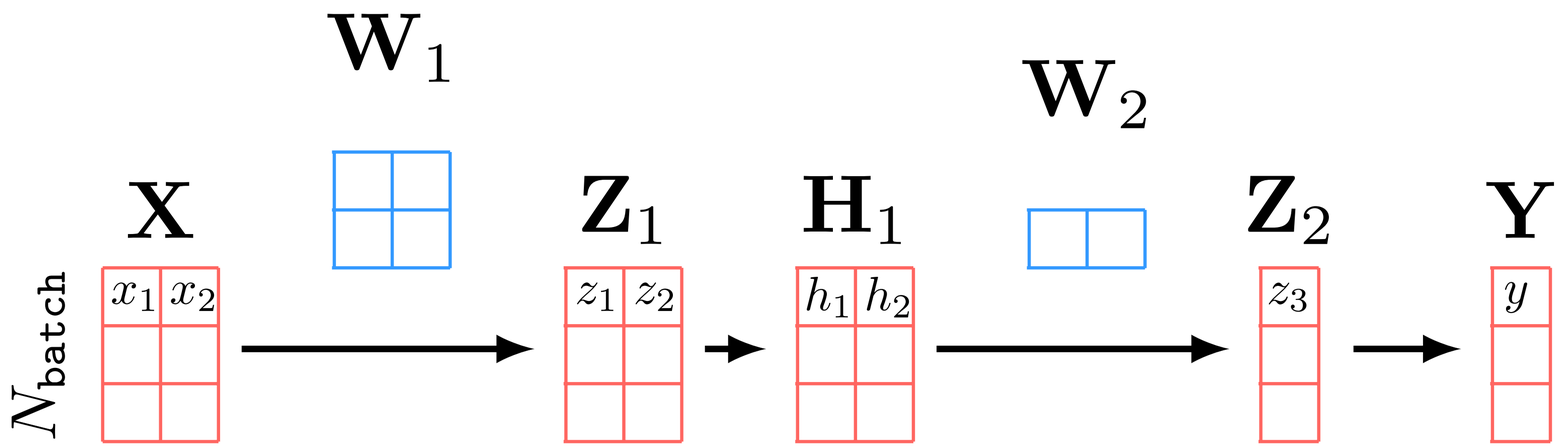
Each layer is a representation of the data

张量Tensor



$$\mathbf{z} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$
$$\mathbf{h} = g(\mathbf{z})$$
$$z_3 = \mathbf{W}_2\mathbf{h} + b_2$$
$$y = 1(z_3 > 0)$$

Tensor processing with batch size = 3:



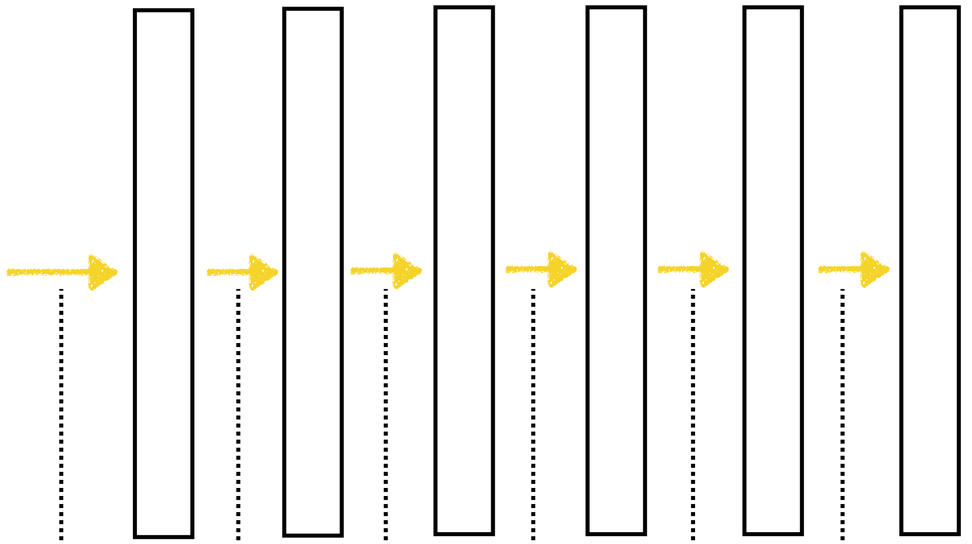
深度学习

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

"chameleon"

Learned

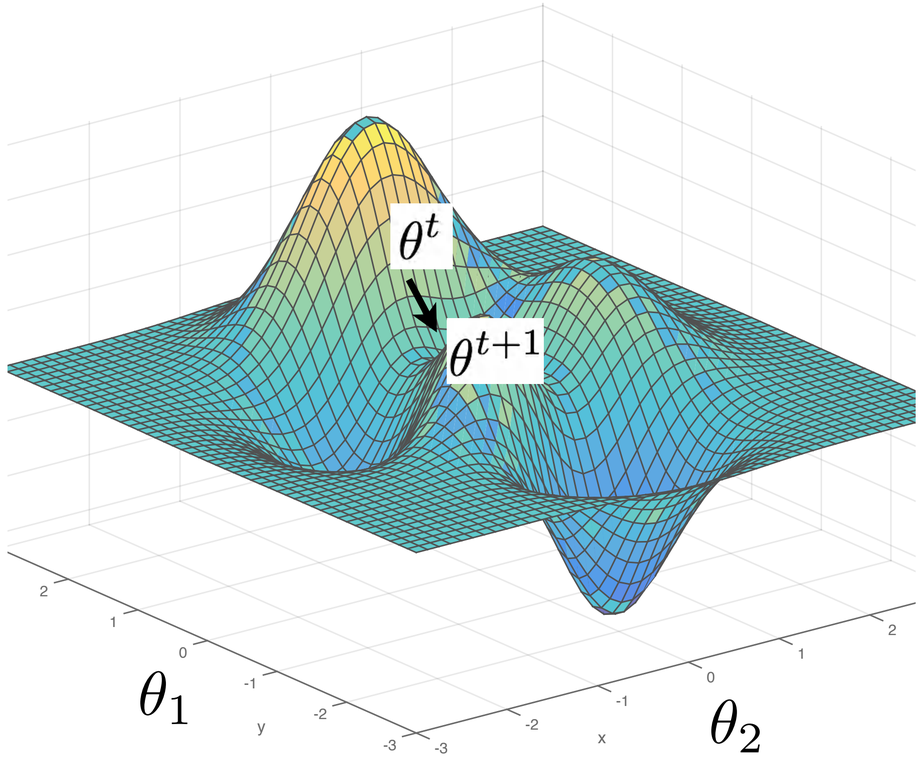
$\mathbf{x}^{(i)}$



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

梯度下降

$J(\theta)$



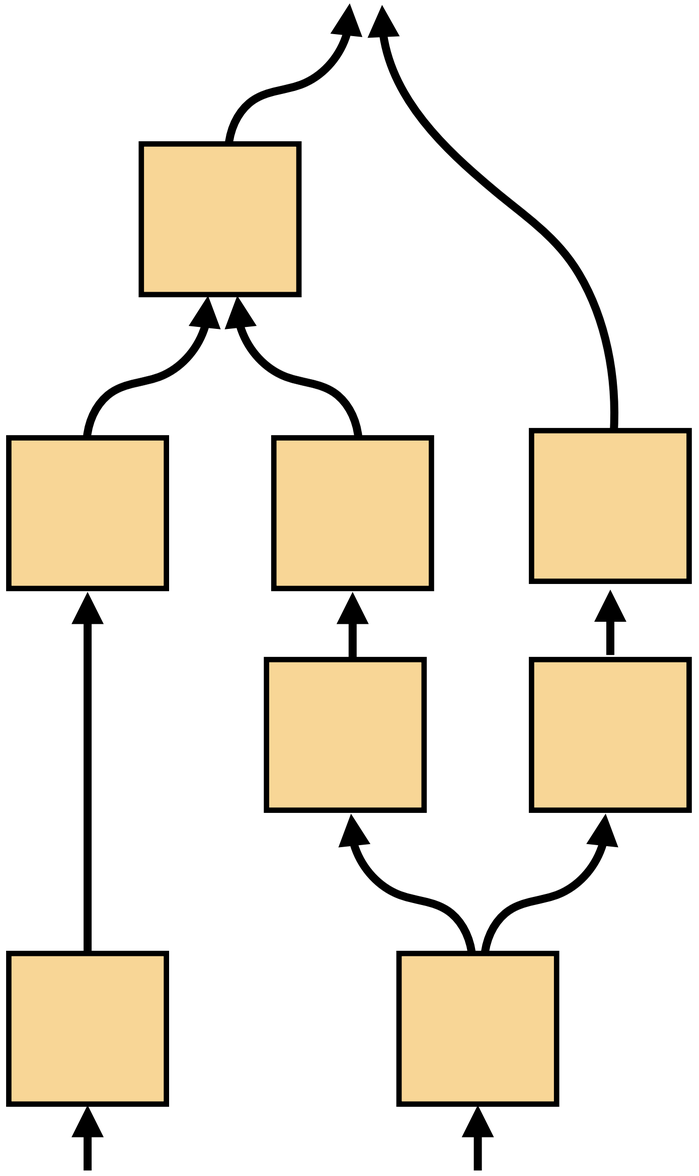
$$\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})}_{J(\theta)}$$

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

计算图



A graph of functional transformations, nodes (□), that when strung together perform some useful computation.

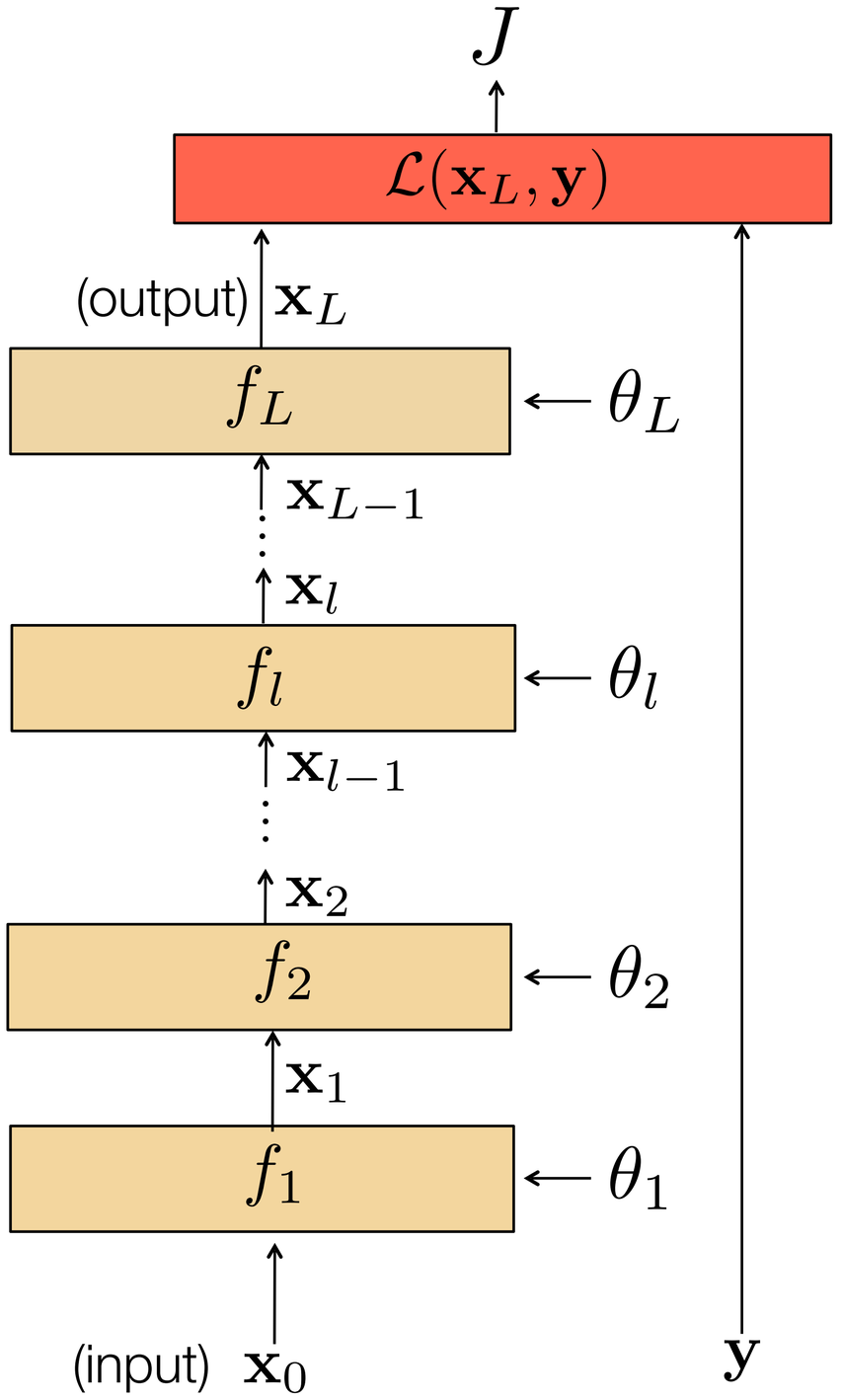
Deep learning deals (primarily) with computation graphs that take the form of **directed acyclic graphs** (DAGs), and for which each node is differentiable.

链式法则

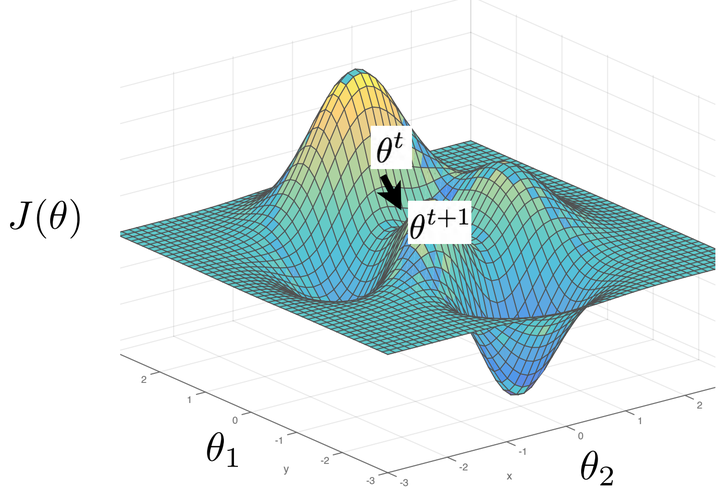
- Consider model with L layers. Layer l has vector of weights θ_l
- **Forward pass**: takes input $\mathbf{x}_{l-1}$ and passes it through each layer f_l :

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

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

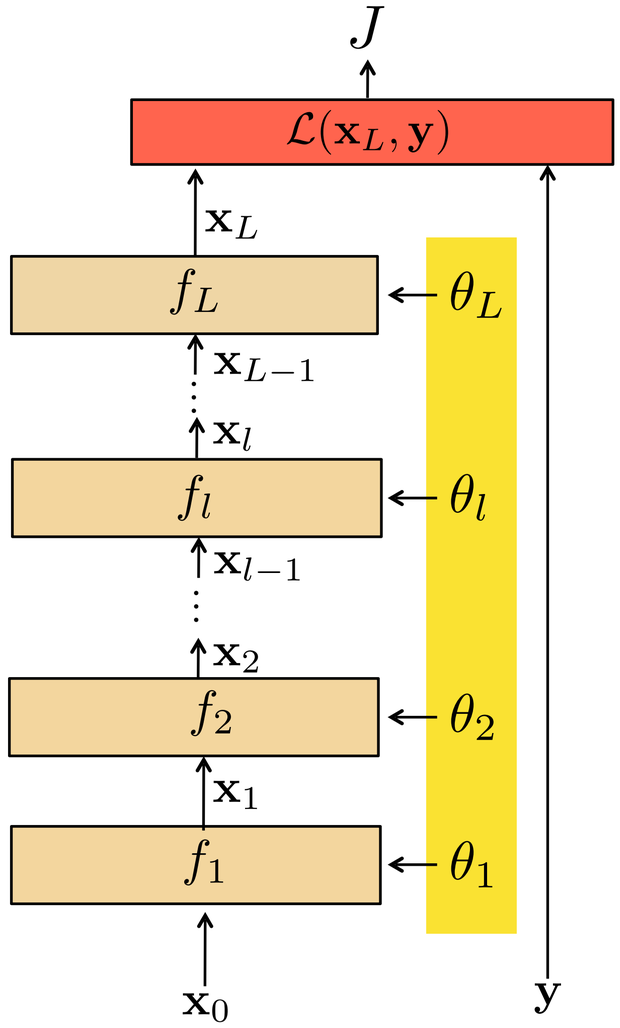


基于链式法则的梯度下降



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

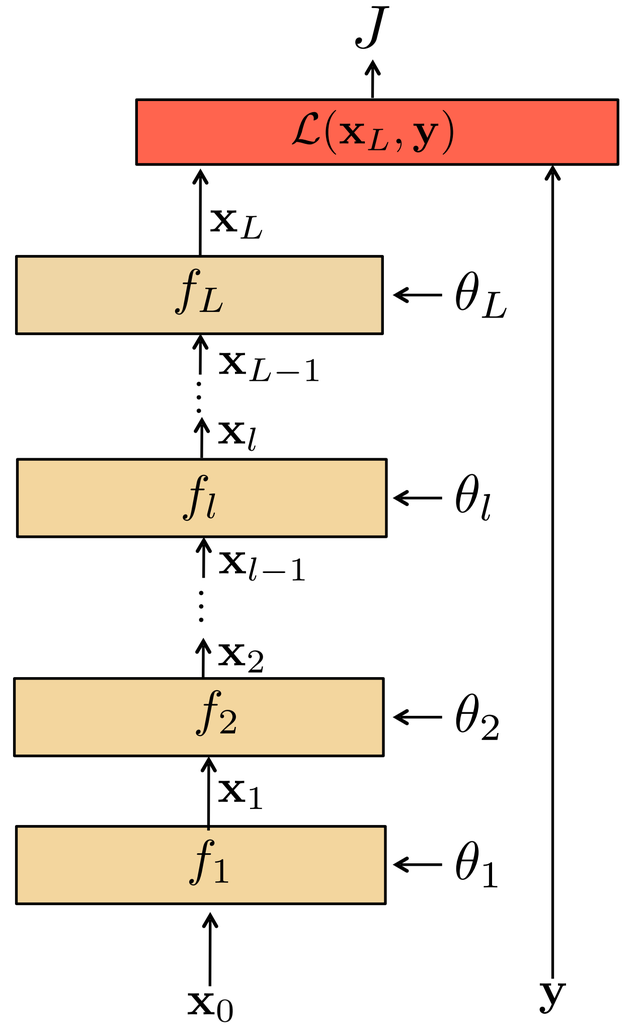
- By design, each layer will be differentiable with respect to its inputs (the inputs are the data and parameters)



链式法则-梯度的计算

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**

背景知识

- $\mathbf{x}$ column vector of size $[n \times 1]$:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

- We now define a function on vector $\mathbf{x}$: $\mathbf{y} = f(\mathbf{x})$
- If y is a scalar, then

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{pmatrix}$$

The derivative of y is a row vector of size $[1 \times n]$

- If $\mathbf{y}$ is a vector $[m \times 1]$, then (*Jacobian formulation*):

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

The derivative of y is a matrix of size $[m \times n]$

(m rows and n columns)

背景知识

- If y is a scalar and $\mathbf{X}$ is a matrix of size $[n \times m]$, then

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{n1}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y}{\partial x_{1m}} & \frac{\partial y}{\partial x_{2m}} & \cdots & \frac{\partial y}{\partial x_{nm}} \end{pmatrix}$$

The output is a matrix of size $[m \times n]$

Wikipedia: The three types of derivatives that have not been considered are those involving vectors-by-matrices, matrices-by-vectors, and matrices-by-matrices. These are not as widely considered and a notation is not widely agreed upon.

背景知识

- Chain rule:

For the function: $h(\mathbf{x}) = f(g(\mathbf{x}))$

Its derivative is: $h'(\mathbf{x}) = f'(g(\mathbf{x}))g'(\mathbf{x})$

and writing $\mathbf{z} = f(\mathbf{u})$, and $\mathbf{u} = g(\mathbf{x})$:

$$\left.\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}} = \left.\frac{\partial \mathbf{z}}{\partial \mathbf{u}}\right|_{\mathbf{u}=g(\mathbf{a})} \cdot \left.\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}}$$

$[m \times n] \qquad [m \times p] \qquad [p \times n]$

with $p =$ length of vector $\mathbf{u} = |\mathbf{u}|$, $m = |\mathbf{z}|$, and $n = |\mathbf{x}|$

Example, if $|\mathbf{z}| = 1$, $|\mathbf{u}| = 2$, $|\mathbf{x}| = 4$

$h'(\mathbf{x}) =$  $=$

链式法则-梯度的计算

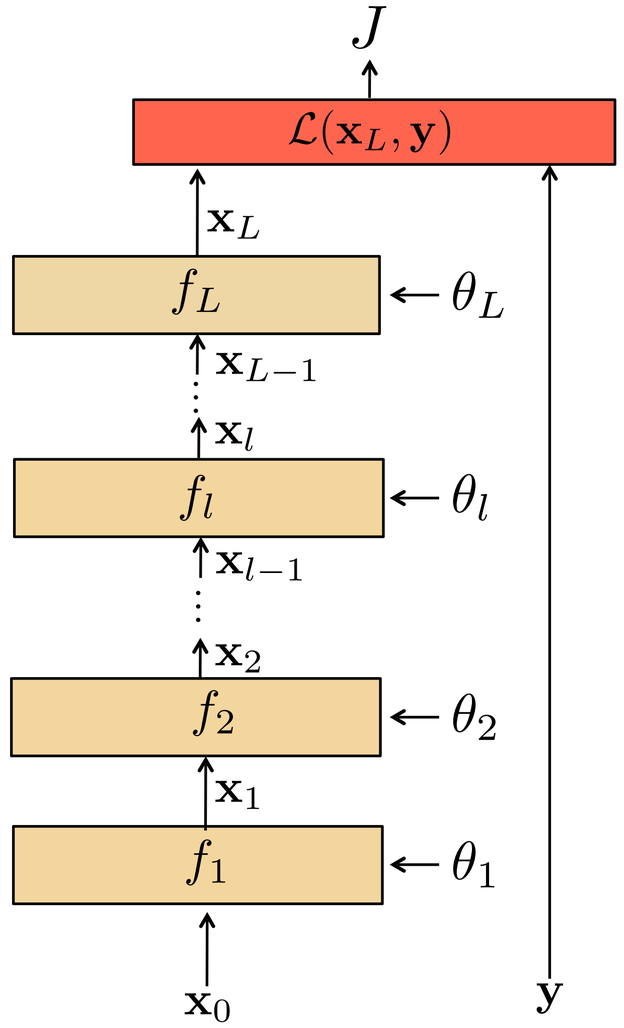
The loss J is the sum of the losses associated with each training example

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

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

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

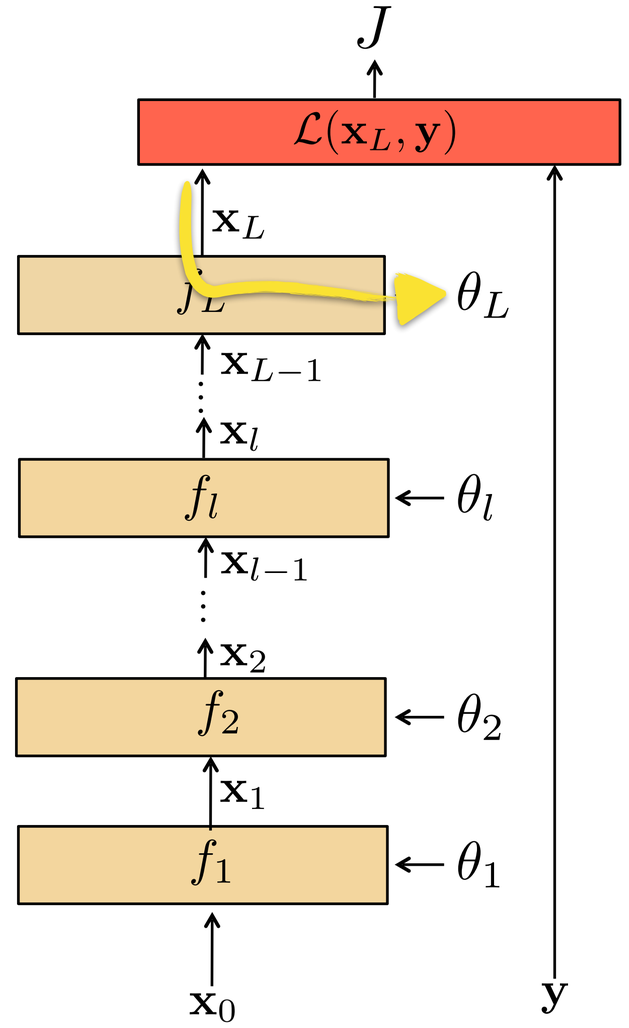
is how much J varies when the parameter θ_i is varied.



链式法则-梯度的计算

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

$$\frac{\partial \mathcal{L}}{\partial \theta_L} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \theta_L}$$



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.

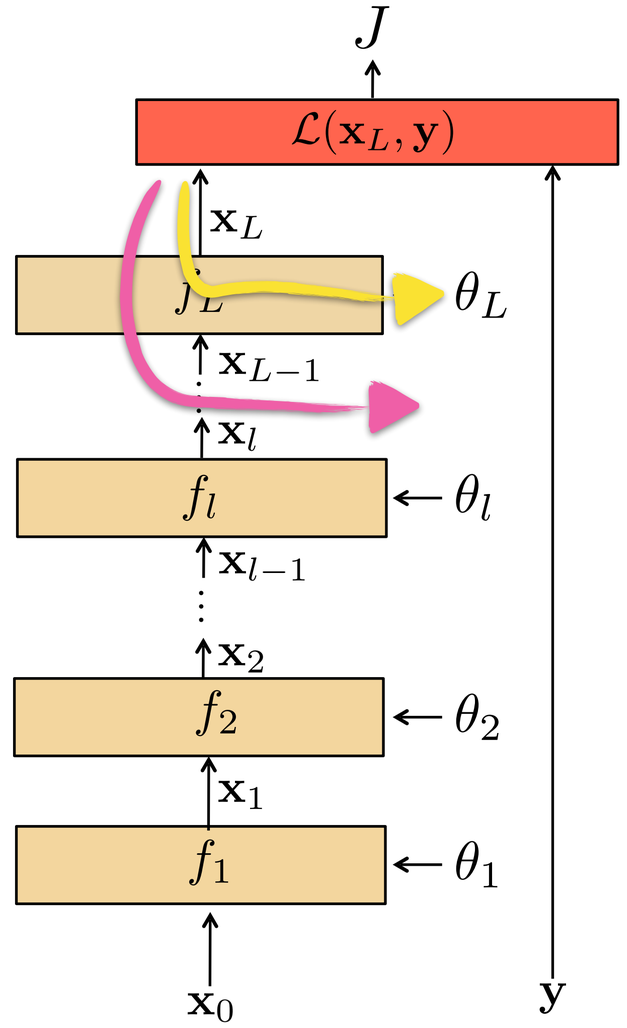
链式法则-梯度的计算

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

$$\frac{\partial \mathcal{L}}{\partial \theta_L} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \theta_L}$$

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

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



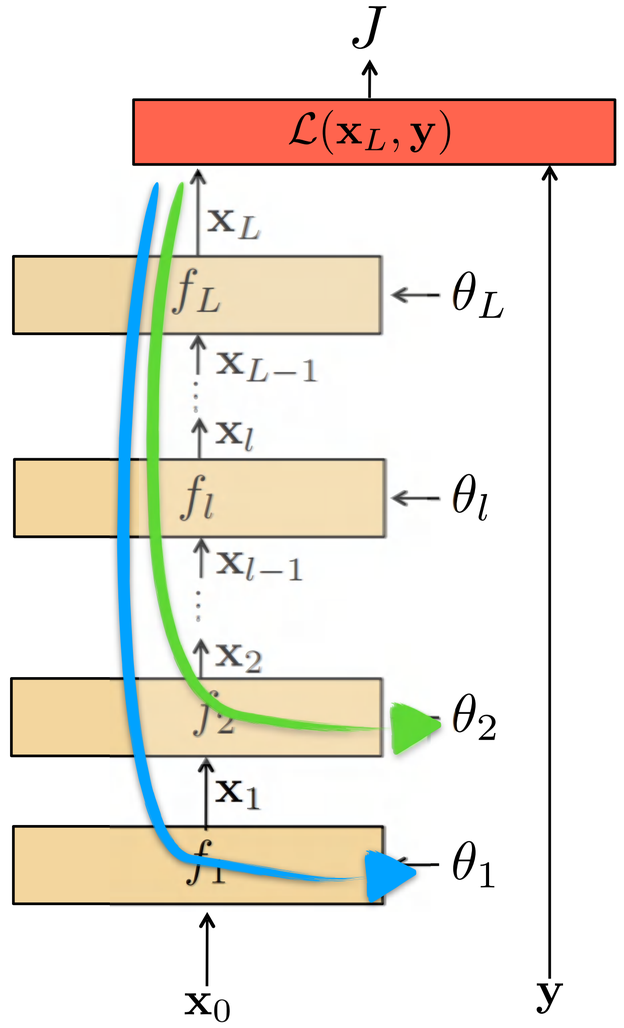
链式法则-梯度的计算

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

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

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

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



链式法则-梯度的计算

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l+1}} \rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l}} \rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l-1}}$$

This can be computed iteratively. We start at the top (L) and we can compute the gradient at layer l-1

If we have the value of $\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l}}$ we can compute the gradient at the layer below as:

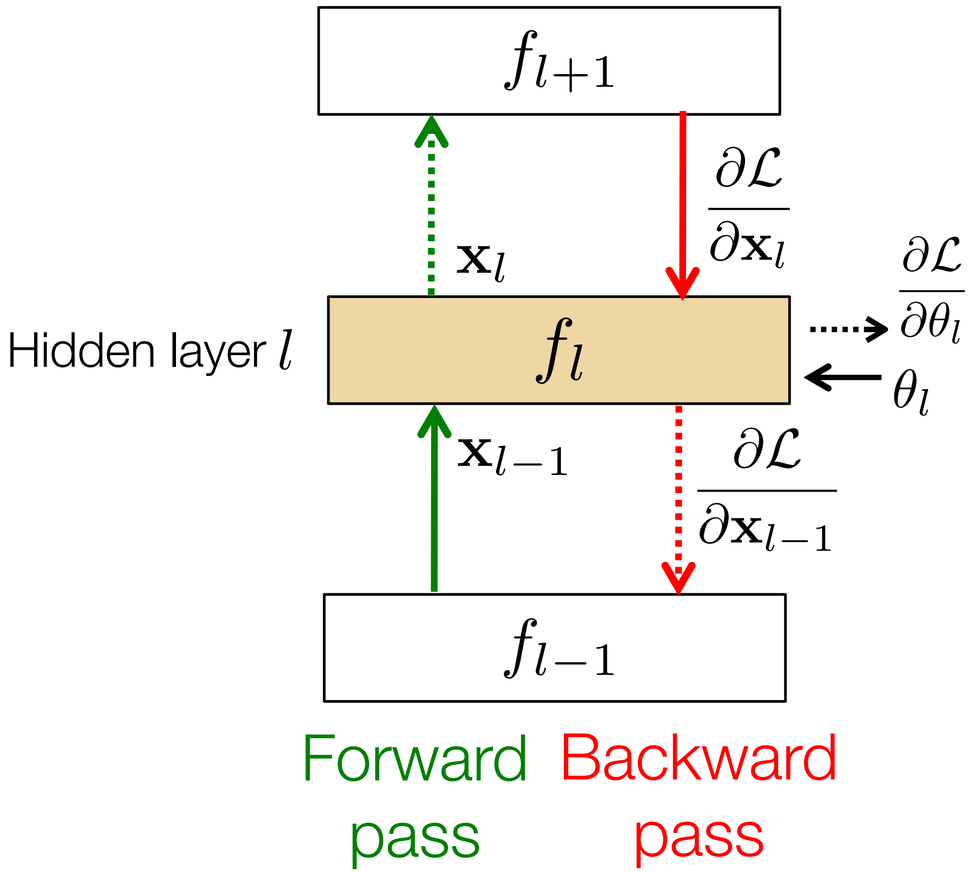
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l}} \cdot \frac{\partial \mathbf{x}_{l}}{\partial \mathbf{x}_{l-1}}$$

Gradient w.r.t. loss at layer l-1

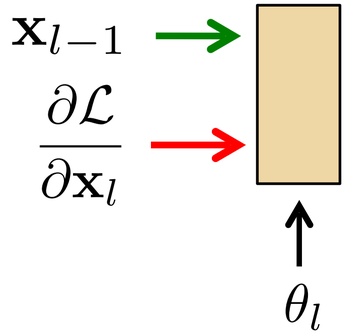
Gradient w.r.t. loss at layer l

Layer l's gradient

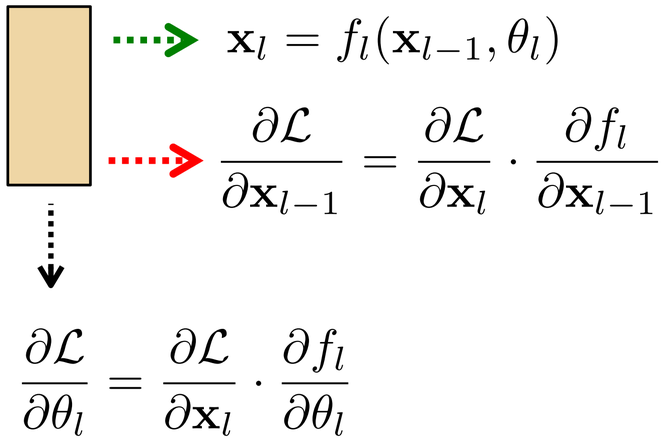
链式法则-梯度的计算



- Layer l has three inputs (during training)



- And three outputs



- Given the inputs, we just need to evaluate:

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

后向传播

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

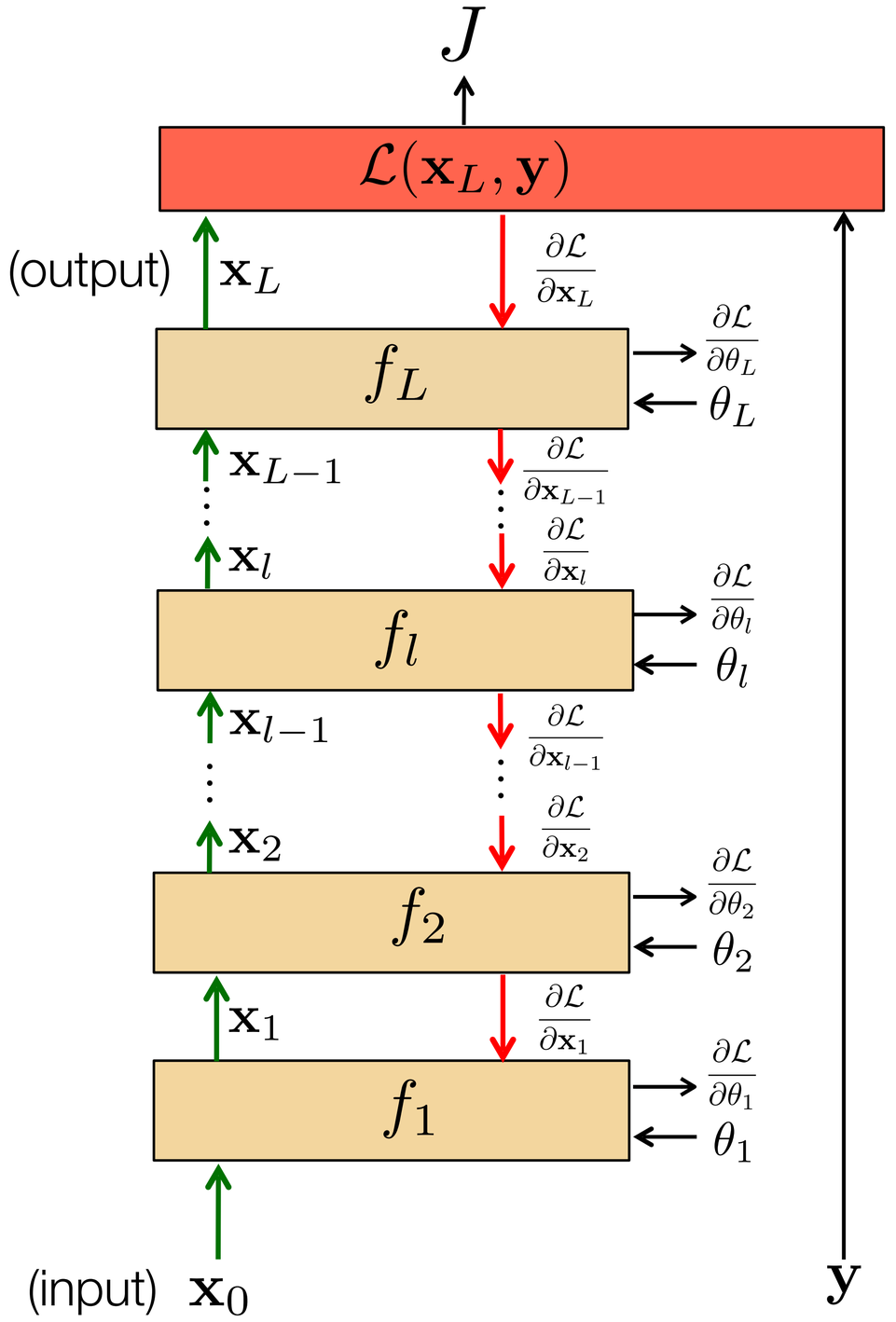
$$\mathbf{x}_l = f_l(\mathbf{x}_{l-1}, \theta_l)$$

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

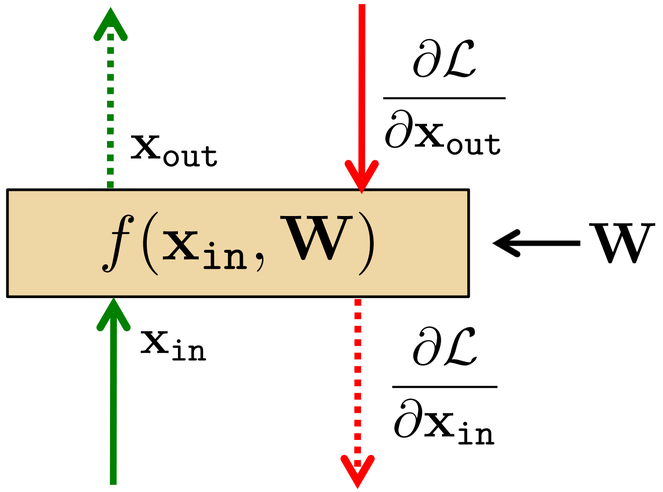
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_l} \cdot \frac{\partial f_l}{\partial \mathbf{x}_{l-1}}$$

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

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



线性层



- 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:

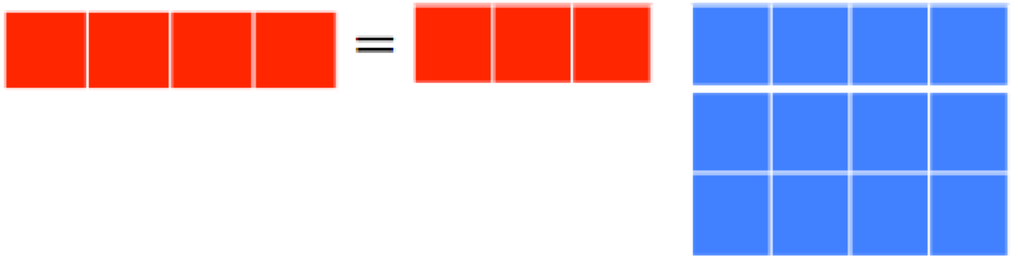
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{x}_{\text{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \frac{\partial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

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

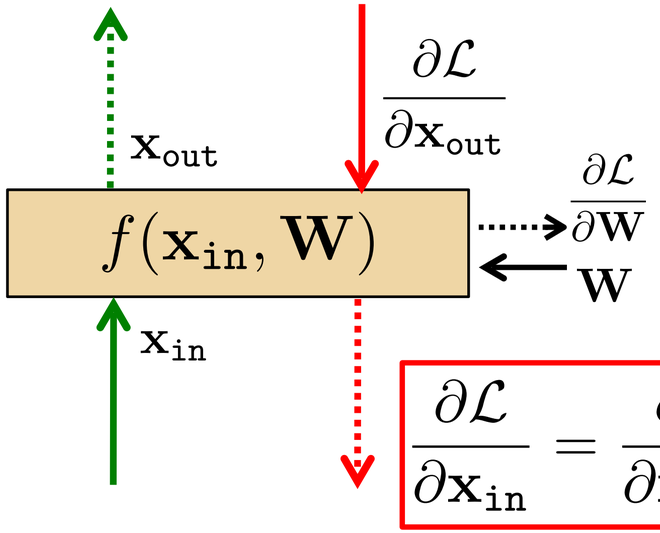
$$\frac{\partial \mathbf{x}_{\text{out}_i}}{\partial \mathbf{x}_{\text{in}_j}} = \mathbf{W}_{ij} \longrightarrow \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{x}_{\text{in}}} = \mathbf{W}$$

Therefore:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \mathbf{W}}$$

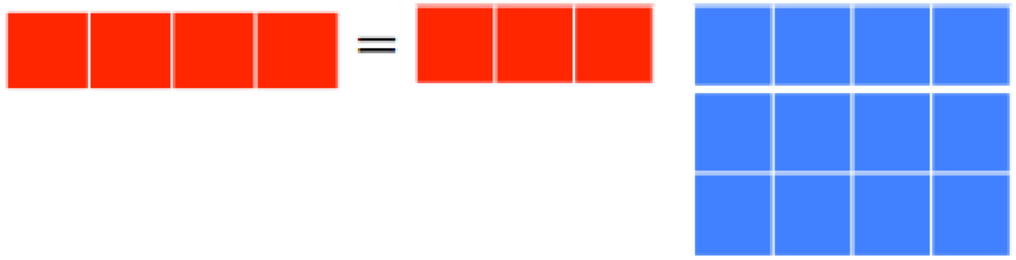


线性层



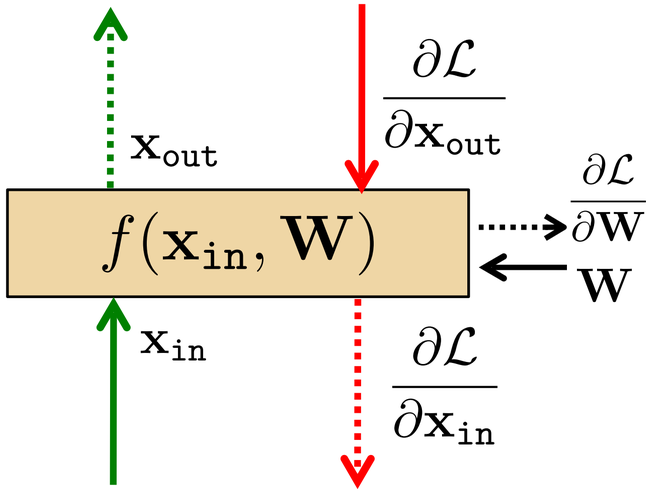
- Forward propagation: $\mathbf{x}_{\text{out}} = f(\mathbf{x}_{\text{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\text{in}}$
- Backprop to input:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \mathbf{W}$$



Now let's see how we use the set of outputs to compute the weights update equation (backprop to the weights).

线性层



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

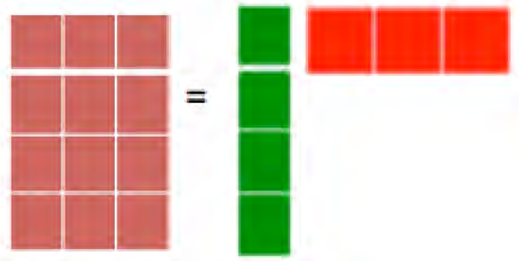
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \frac{\partial \mathbf{x}_{\text{out}}}{\partial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

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

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

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

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{x}_{\text{in}} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}}}$$

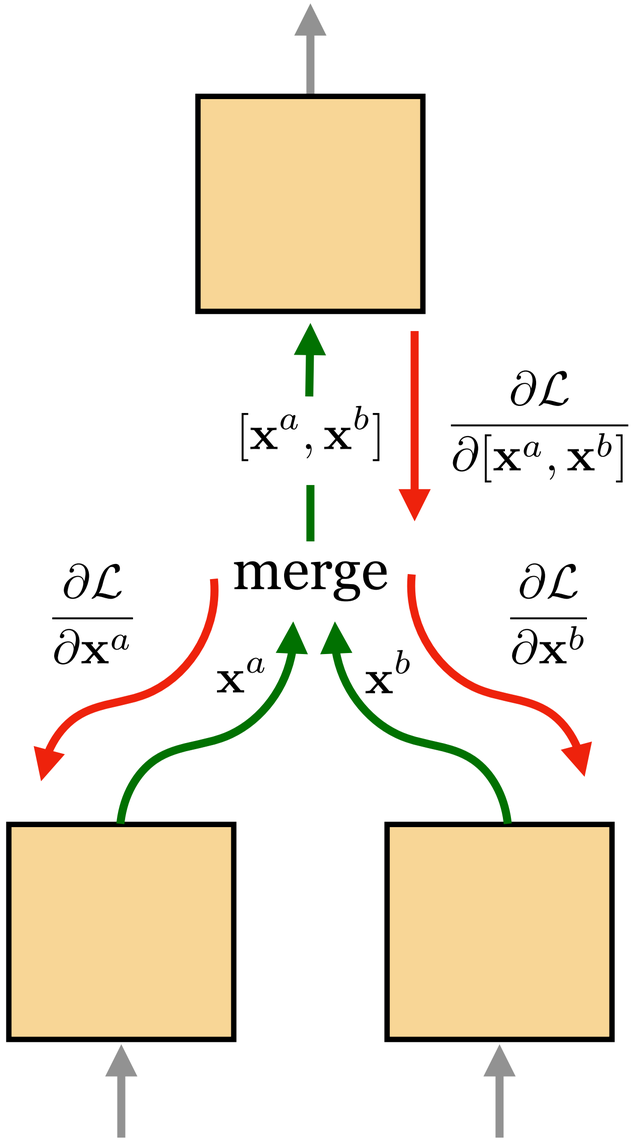


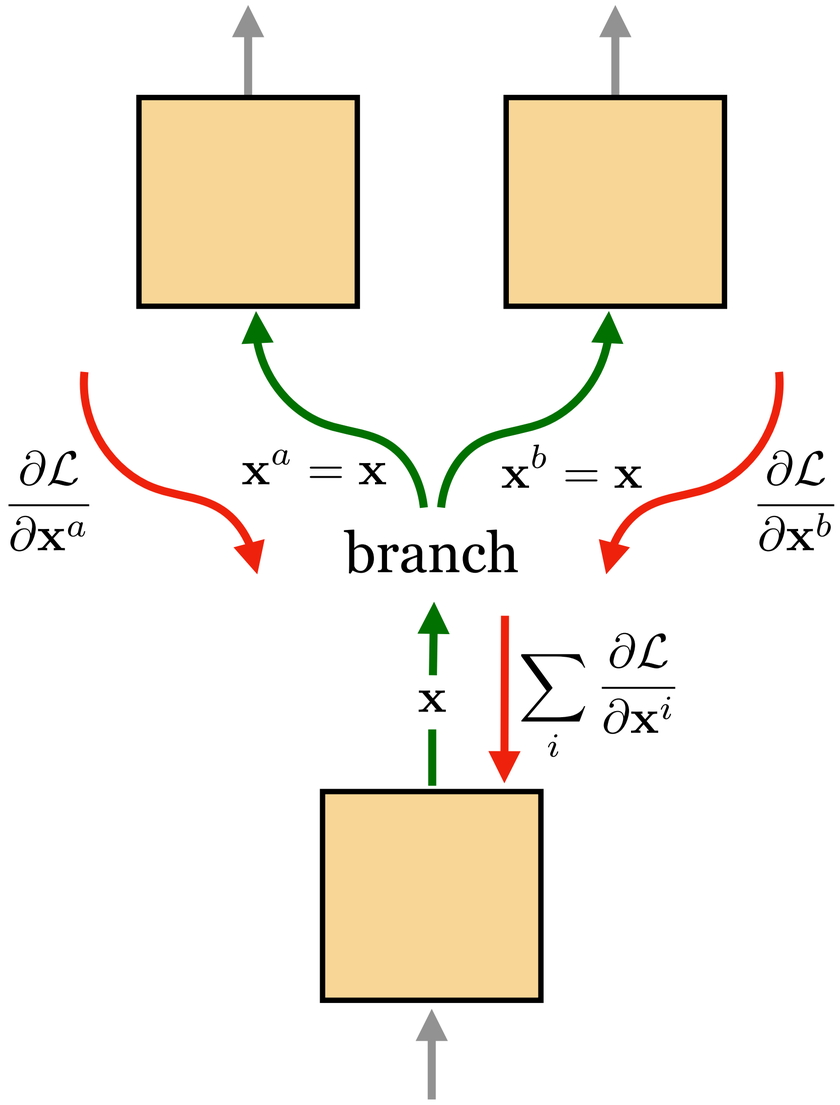
And now we can update the weights (by summing over all the training examples):

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

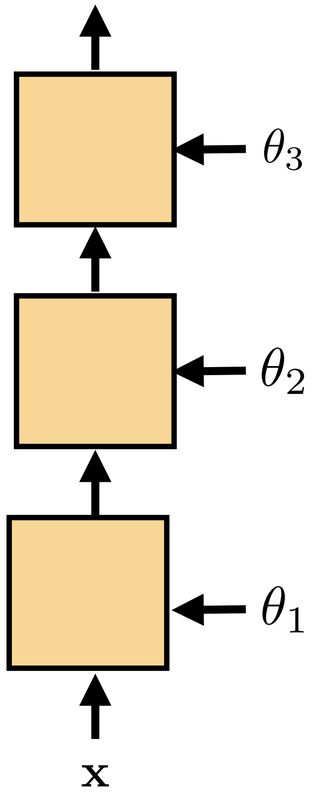
(sum over all training examples to get J)

计算图

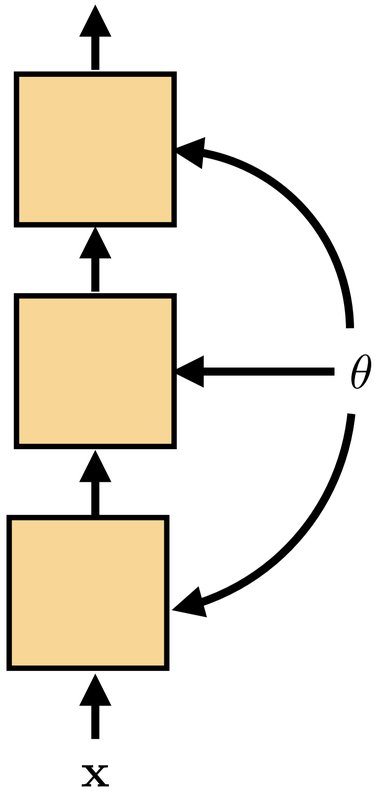

$[\mathbf{x}^a, \mathbf{x}^b]$
$\frac{\partial \mathcal{L}}{\partial [\mathbf{x}^a, \mathbf{x}^b]}$
merge
$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^a}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^b}$
$\mathbf{x}^a$
$\mathbf{x}^b$

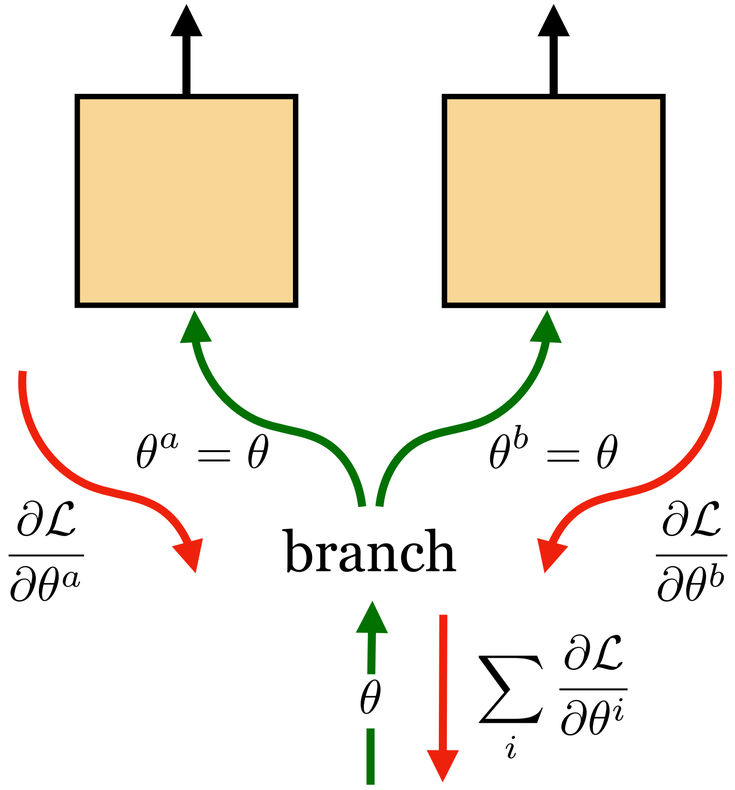

$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^a}$
$\mathbf{x}^a = \mathbf{x}$
$\mathbf{x}^b = \mathbf{x}$
$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^b}$
branch
$\mathbf{x}$
$\sum_i \frac{\partial \mathcal{L}}{\partial \mathbf{x}^i}$

特征共享



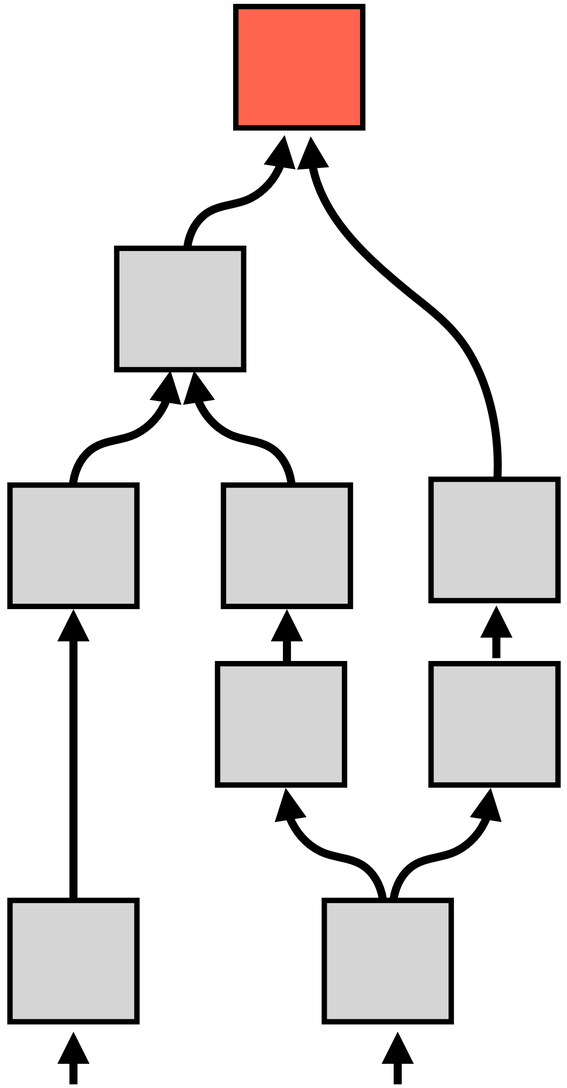
特征共享



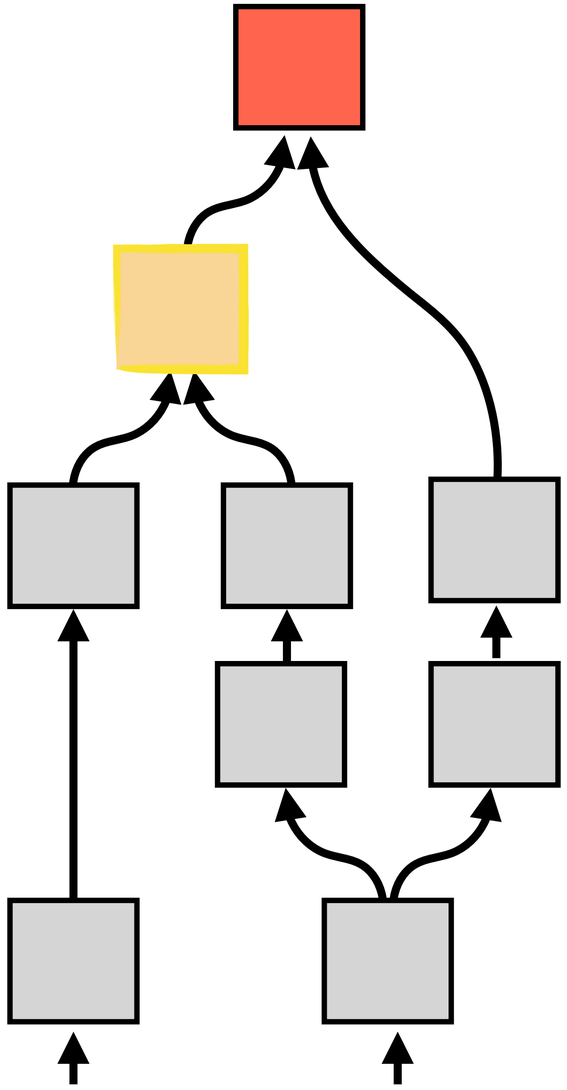


Parameter sharing —> sum gradients

后向传播可对任意节点求导



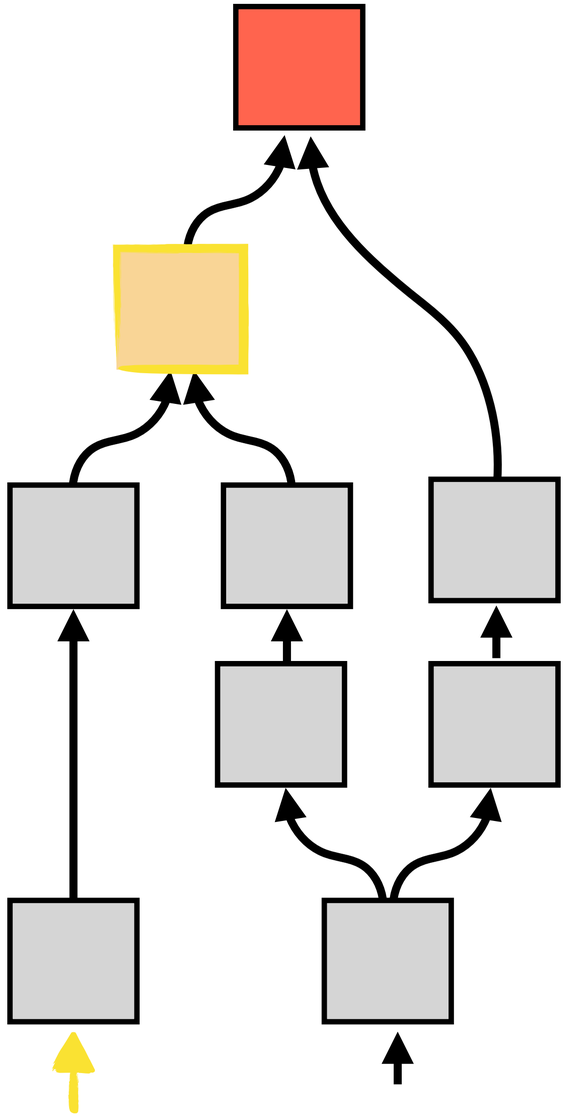
后向传播可对任意节点求导



$$\frac{\partial \blacksquare}{\partial \blacksquare}$$

How the loss changes when the input data changes

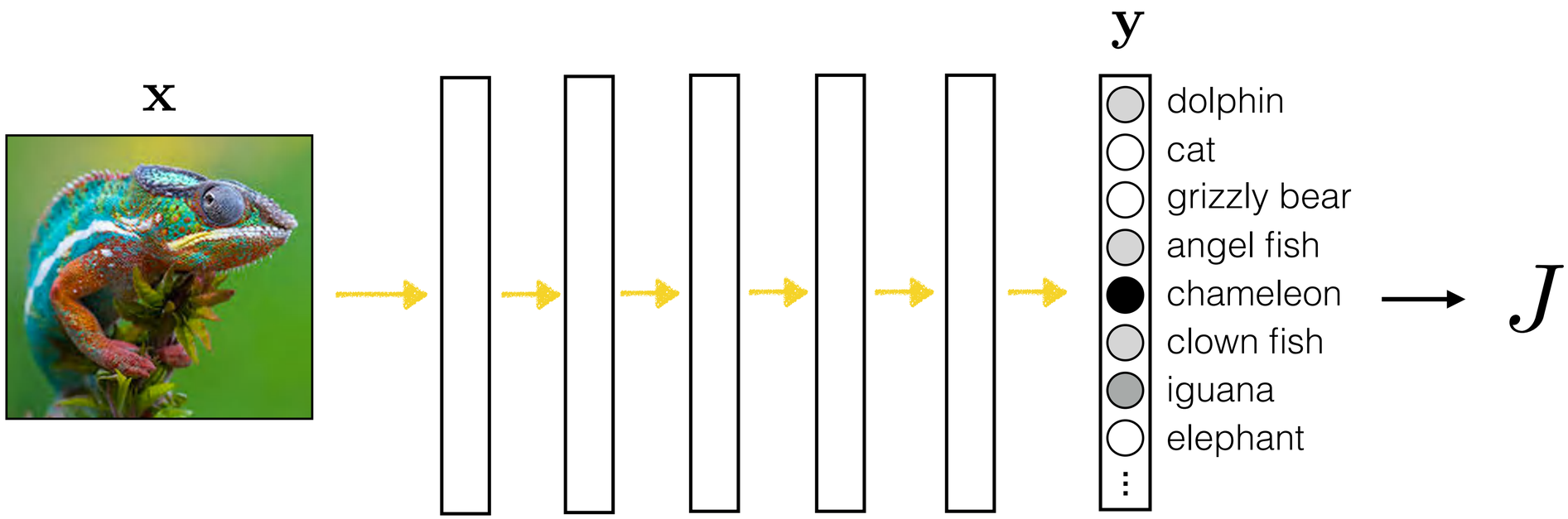
后向传播可对任意节点求导



$\frac{\partial \square}{\partial \square}$ How the loss changes when the functional node highlighted changes

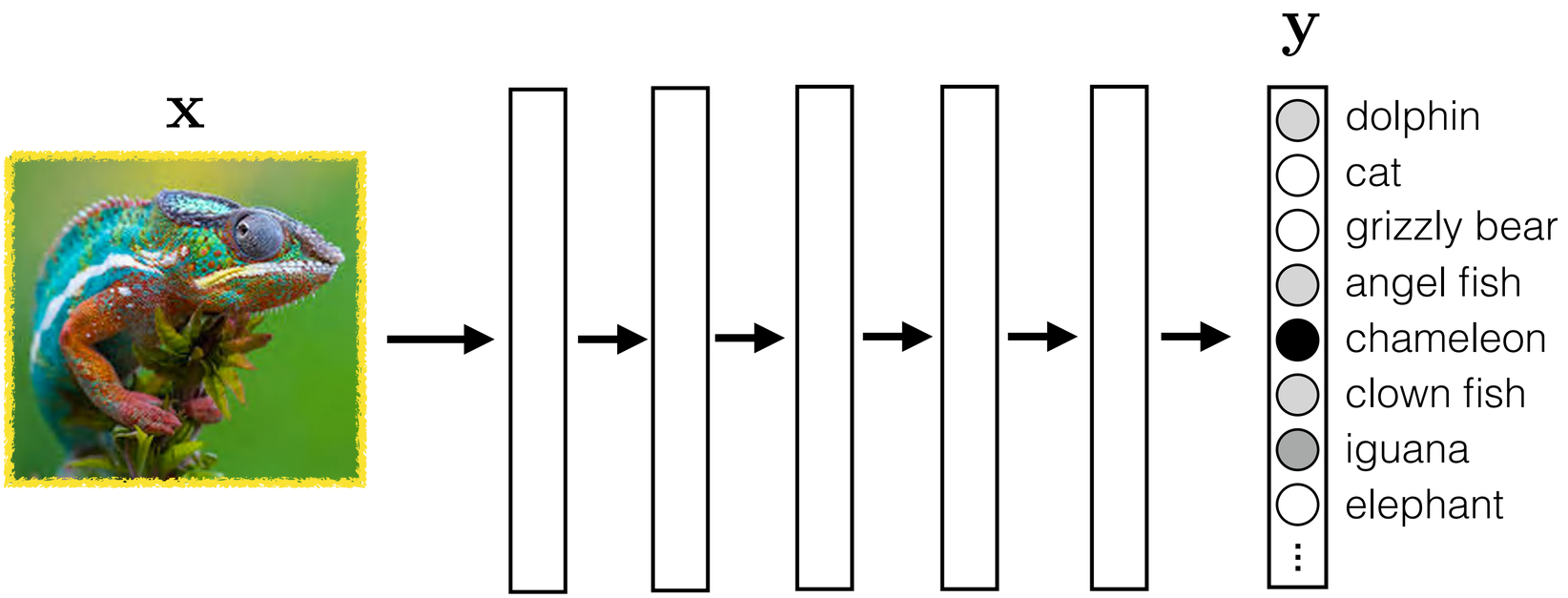
$\frac{\partial \square}{\partial \uparrow}$ How the cost changes when the input data changes

后向传播对变量求导



$\frac{\partial J}{\partial \theta}$ ← How much the total cost is increased or decreased by changing the parameters.

后向传播可对输入求导



$$\frac{\partial y_j}{\partial \mathbf{x}}$$ ← How much the “chameleon” score is increased or decreased by changing the image pixels.

可视化

$$\arg\max_{\mathbf{x}} y_j$$

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \eta \frac{\partial y_j(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}^k}$$

可视化

$$\arg\max_{\mathbf{x}} y_j + \lambda R(\mathbf{x})$$

$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \eta \frac{\partial (y_j(\mathbf{x}) + \lambda R(\mathbf{x}))}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}^k}$$

Make an image that maximizes the “cat” output neuron:

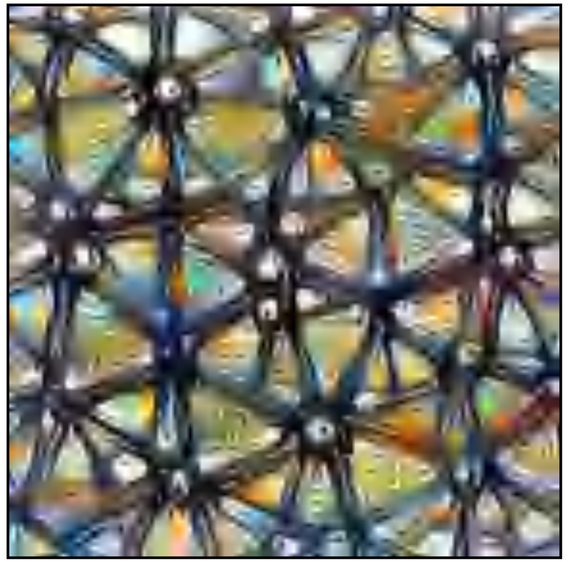
[https://distill.pub/2017/feature-visualization/]

可视化

$$\arg\max_{\mathbf{x}} h_{l_j} + \lambda R(\mathbf{x})$$

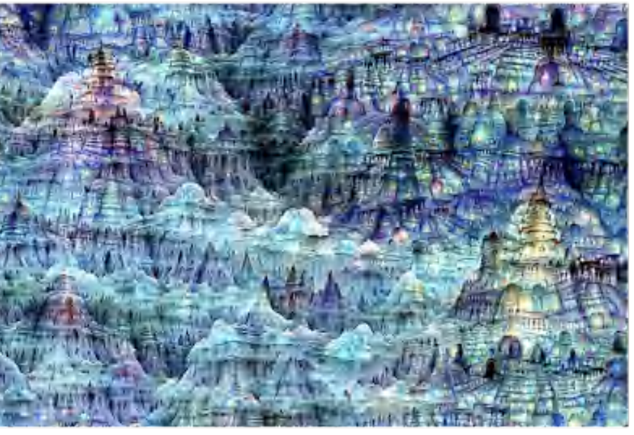
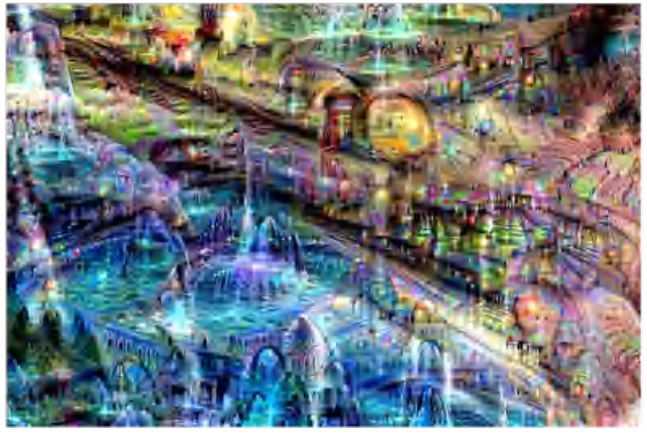
$$\mathbf{x}^{k+1} \leftarrow \mathbf{x}^k + \eta \frac{\partial (h_{l_j}(\mathbf{x}) + \lambda R(\mathbf{x}))}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}^k}$$

Make an image that maximizes the value of neuron j on layer l of the network:



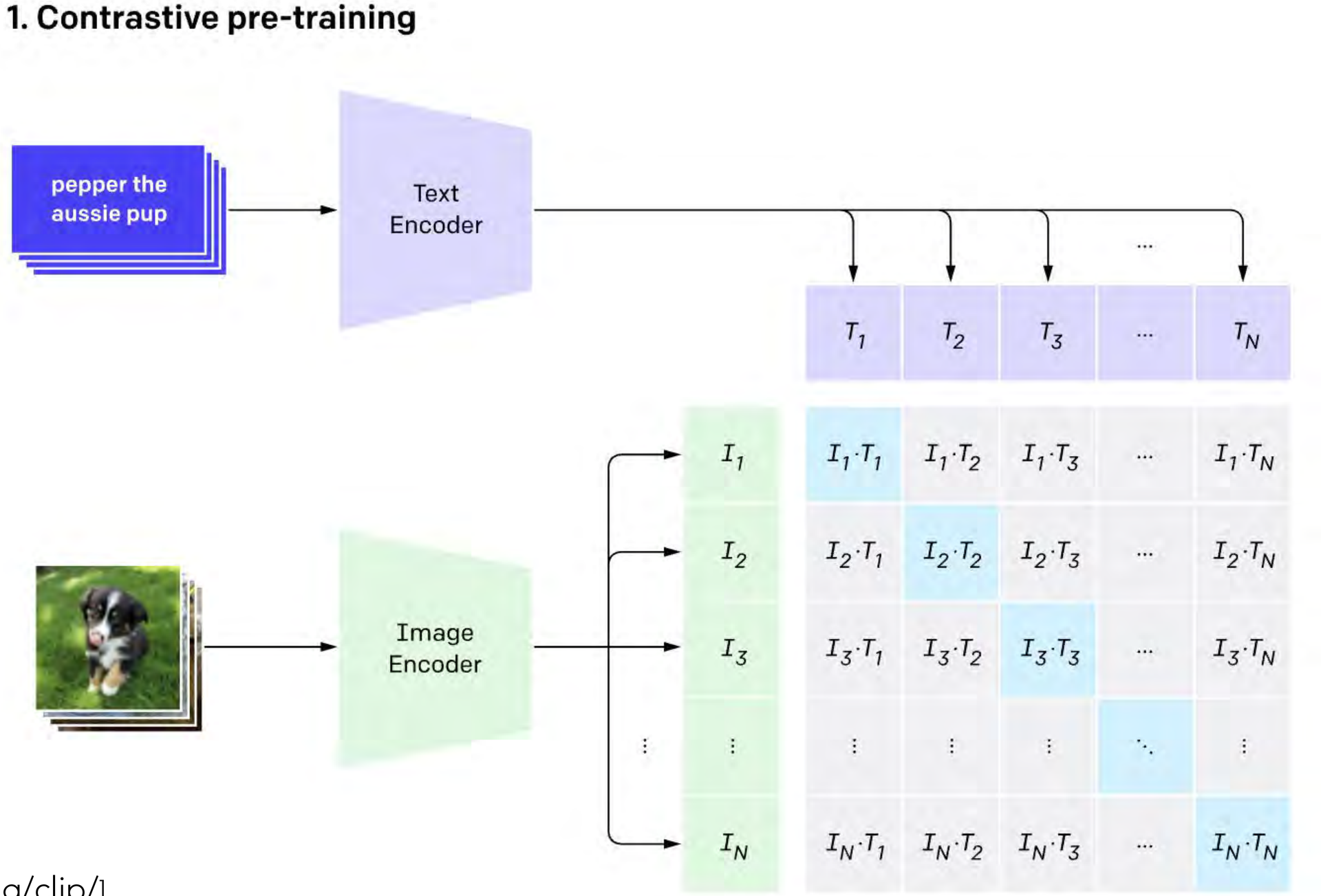
[https://distill.pub/2017/feature-visualization/]



“Deep dream” [https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html]

CLIP

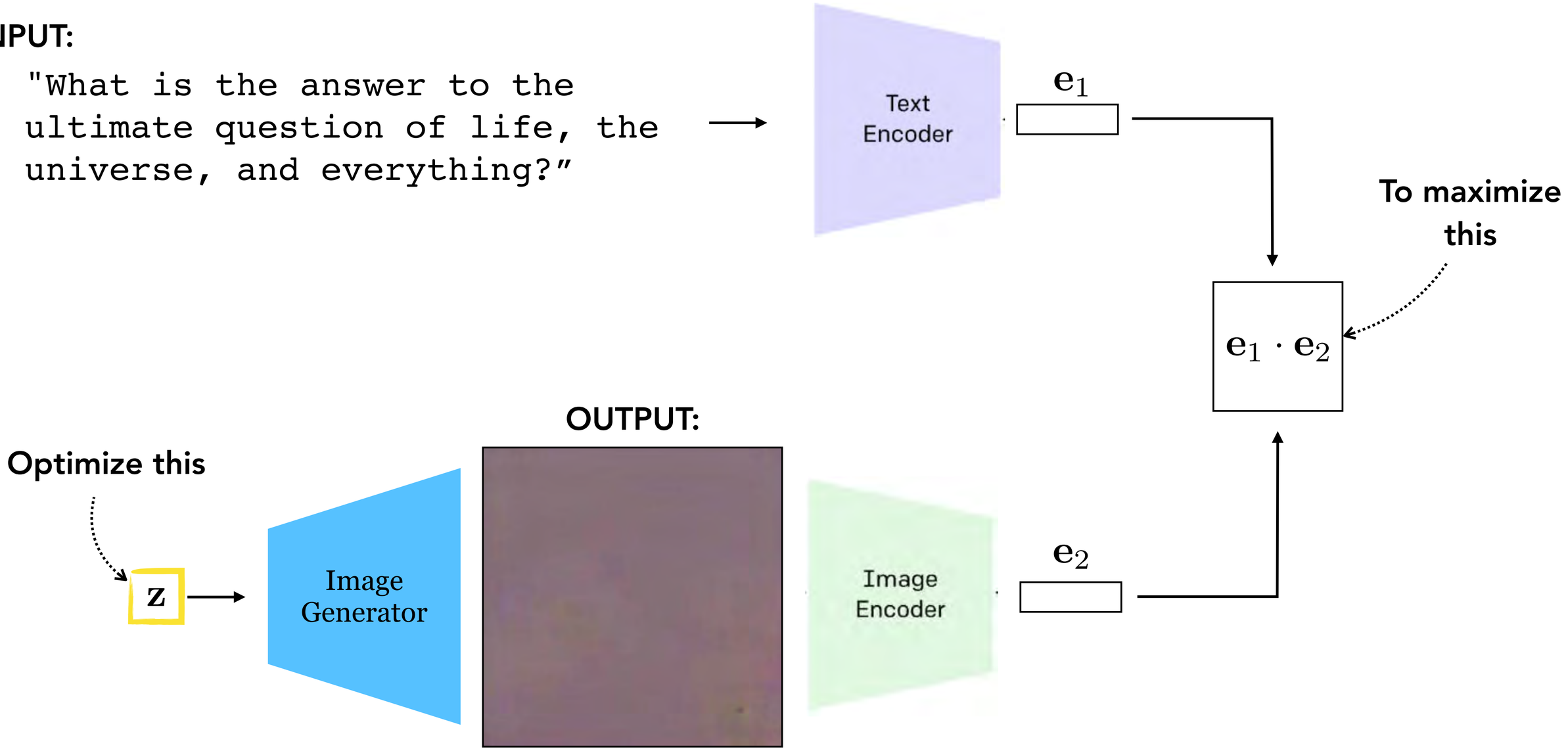


[https://openai.com/blog/clip/]

CLIP+GAN

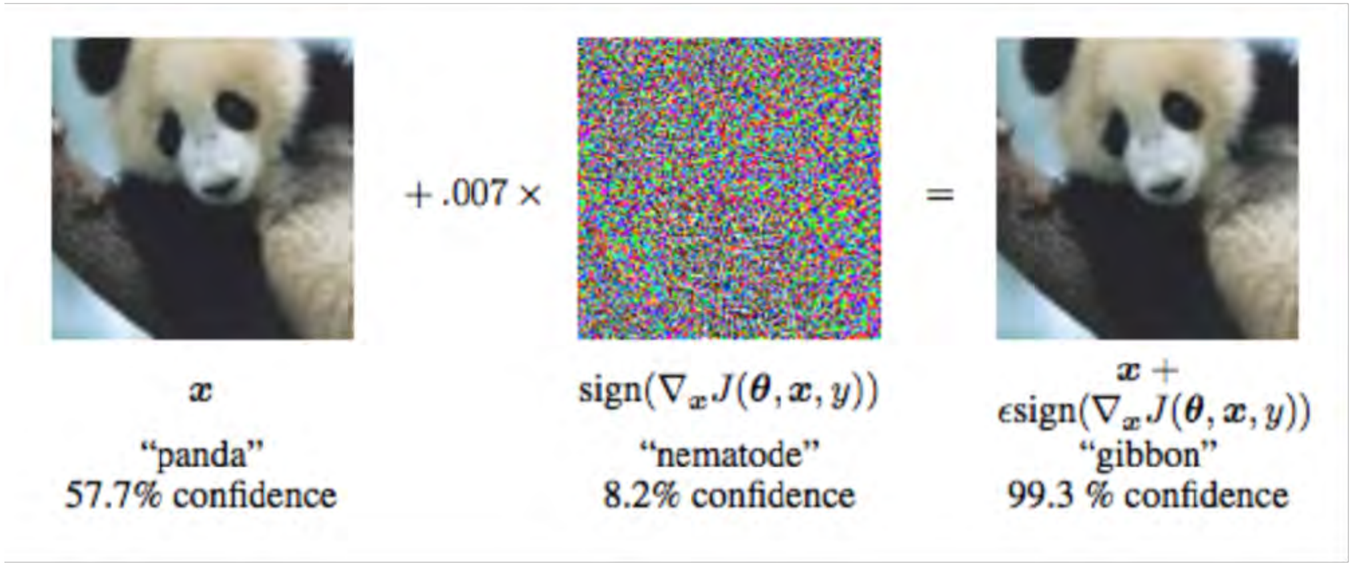
INPUT:

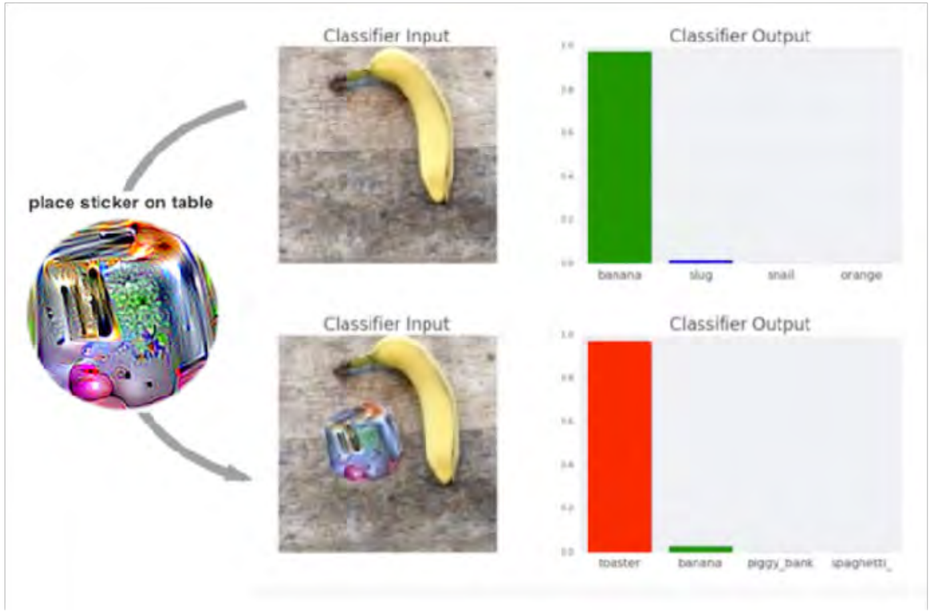
"What is the answer to the ultimate question of life, the universe, and everything?"



Code: https://colab.research.google.com/drive/1_4PQqzM_0KKytCzWtn-ZPi4cCa5bwK2F?usp=sharing

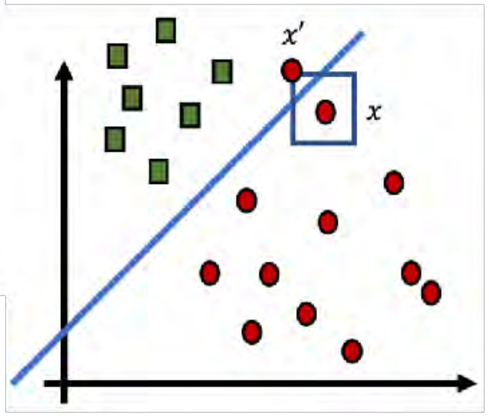
对抗样本



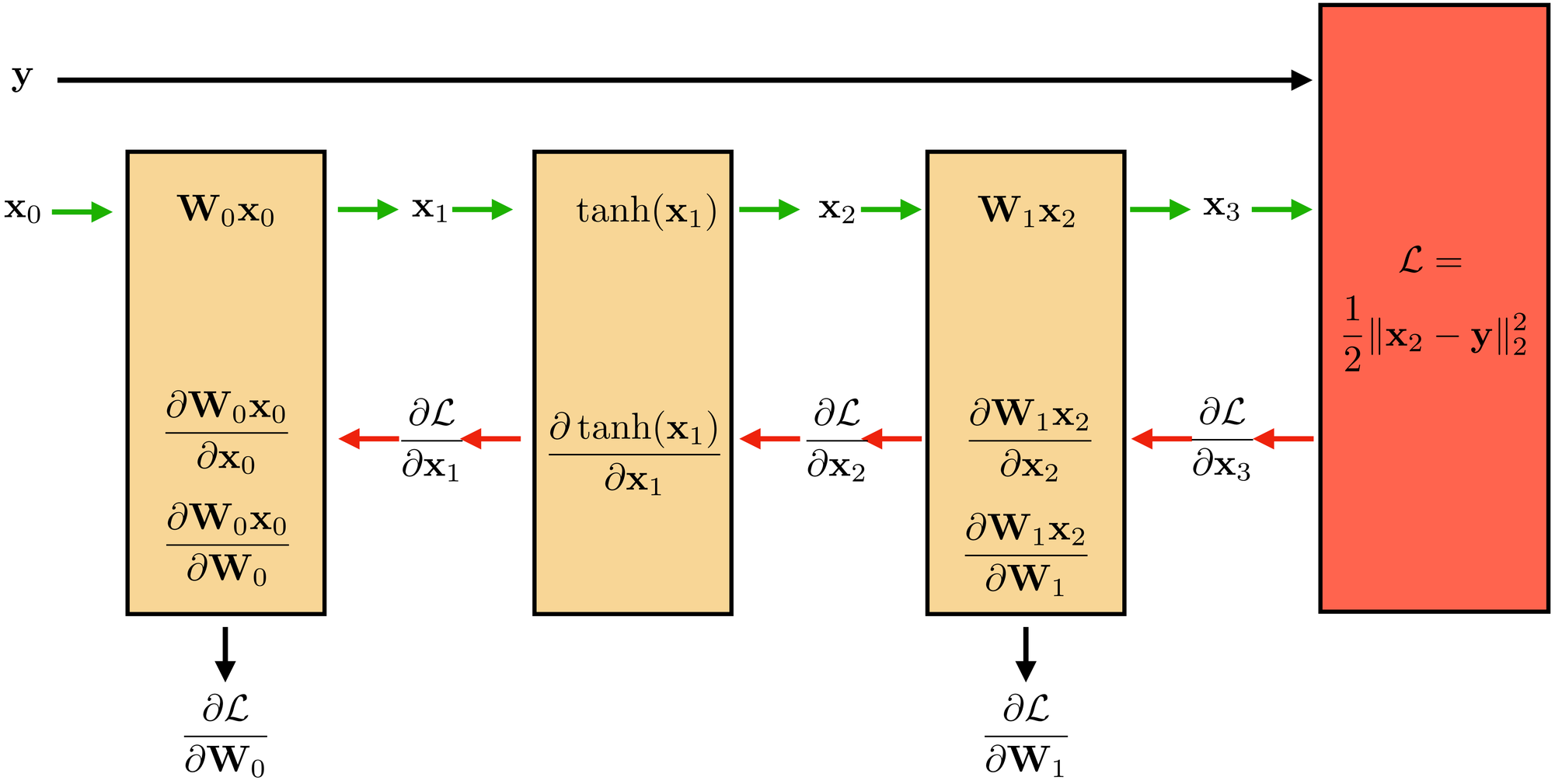


- Attacker's knowledge:
- Suppose model F with parameter θ is given to attacker;

- Goal:
- For a test sample x with label y (true)
- Find δ such that $F(x+\delta) \neq y$



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.

Our goal is to perform the following two updates:

$$\mathbf{W}_0^{k+1} = \mathbf{W}_0^k + \eta \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} \right)^T$$

$$\mathbf{W}_1^{k+1} = \mathbf{W}_1^k + \eta \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} \right)^T$$

where W^k are the weights at some iteration k of gradient descent given by the first slide:

$$\mathbf{W}_0^k = \begin{pmatrix} 1 & -3 \\ 0.2 & 1 \end{pmatrix} \qquad \mathbf{W}_1^k = \begin{pmatrix} 1 & -1 \end{pmatrix}$$

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.

The values for input vector x_0 and target y are also given by the first slide:

$$\mathbf{x}_0 = \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} \qquad \mathbf{y} = 0.5$$

Finally, we simply plug these values into our equations and compute the numerical updates:

Forward pass:

$$\mathbf{x}_1 = \mathbf{W}_0\mathbf{x}_0 = \begin{pmatrix} 1 & -3 \\ 0.2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}$$

$$\mathbf{x}_2 = \tanh(\mathbf{x}_1) = \begin{pmatrix} 0.604 \\ 0.291 \end{pmatrix}$$

$$\mathbf{x}_3 = \mathbf{W}_1\mathbf{x}_2 = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} 0.604 \\ 0.291 \end{pmatrix} = 0.313$$

$$\mathcal{L} = \frac{1}{2}(\mathbf{x}_3 - \mathbf{y})^2 = 0.017$$

Backward pass:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3} = \mathbf{x}_3 - \mathbf{y} = -0.1869$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}\mathbf{W}_1 = -0.1869 \begin{pmatrix} 1 & -1 \end{pmatrix} = \begin{pmatrix} -0.1869 & 0.1869 \end{pmatrix}$$

diagonal matrix because tanh is a pointwise operation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}(1 - \tanh^2(\mathbf{x}_1)) = \begin{pmatrix} -0.1869 & 0.1869 \end{pmatrix} \begin{pmatrix} 1 - \tanh^2(0.7) & 0 \\ 0 & 1 - \tanh^2(0.3) \end{pmatrix} = \begin{pmatrix} -0.1186 & 0.171 \end{pmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} = \mathbf{x}_0 \frac{\partial \mathcal{L}}{\partial \mathbf{x}_1} = \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} \begin{pmatrix} -0.1186 & 0.171 \end{pmatrix} = \begin{pmatrix} -0.1186 & 0.171 \\ -0.01186 & 0.0171 \end{pmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \mathbf{x}_2 \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3} = \begin{pmatrix} 0.604 \\ 0.291 \end{pmatrix} \begin{pmatrix} -0.1186 \end{pmatrix} = \begin{pmatrix} -0.113 \\ -0.054 \end{pmatrix}$$

Gradient updates:

$$\begin{aligned}\mathbf{W}_0^{k+1} &= \mathbf{W}_0^k + \eta\left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0}\right)^T \\ &= \begin{pmatrix} 1 & -3 \\ 0.2 & 1 \end{pmatrix} - 0.2 \begin{pmatrix} -0.1186 & 0.171 \\ -0.01186 & 0.0171 \end{pmatrix} \\ &= \begin{pmatrix} 1.02 & -3.0 \\ 0.17 & 1.0 \end{pmatrix}\end{aligned}$$

$$\begin{aligned}\mathbf{W}_1^{k+1} &= \mathbf{W}_1^k + \eta\left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}\right)^T \\ &= \begin{pmatrix} 1 & -1 \end{pmatrix} - 0.2 \begin{pmatrix} -0.113 & -0.054 \end{pmatrix} \\ &= \begin{pmatrix} 1.02 & -0.989 \end{pmatrix}\end{aligned}$$