Learning with neural networks

Dr. Xiaowei Huang

https://cgi.csc.liv.ac.uk/~xiaowei/

Up to now,

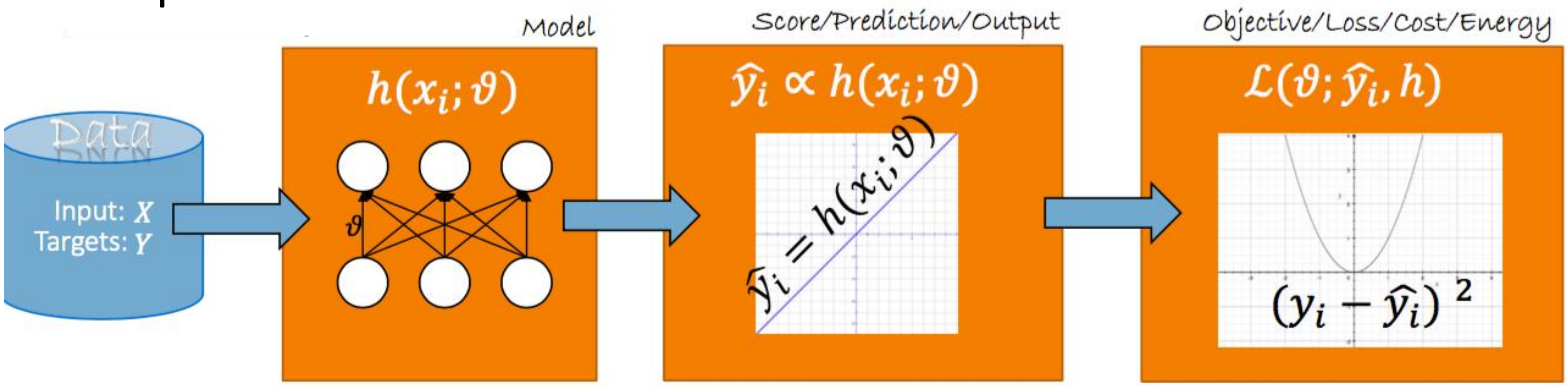
- Traditional Machine Learning Algorithms
- Deep learning
    - Introduction to Deep Learning
    - Functional view and features

Topics

- Forward and backward computation
- Back-propogation and chain rule
- Regularization

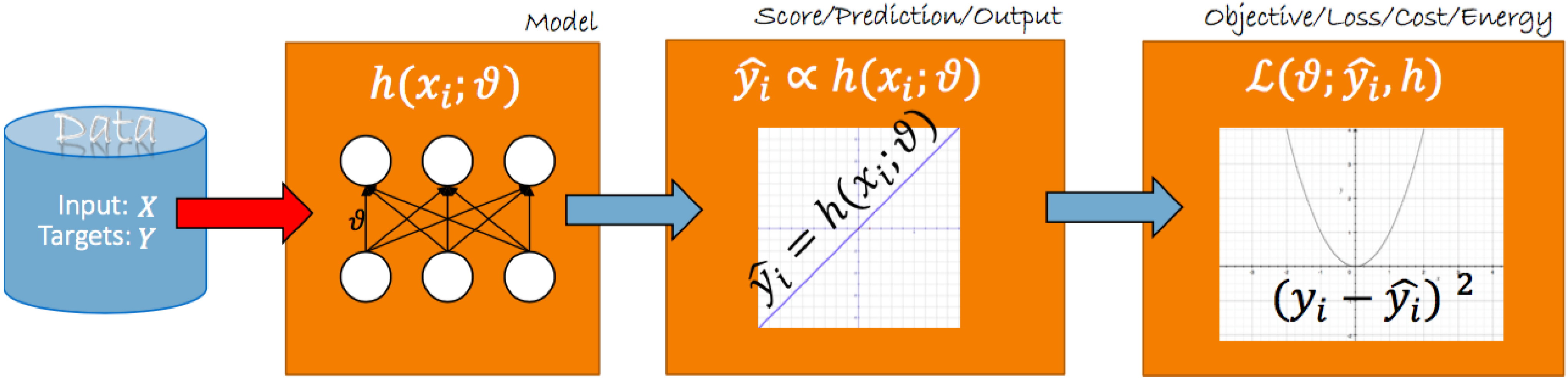
Initialisation

- Collect annotated data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon “forward propagation”
- Evaluate predictions and update model weights



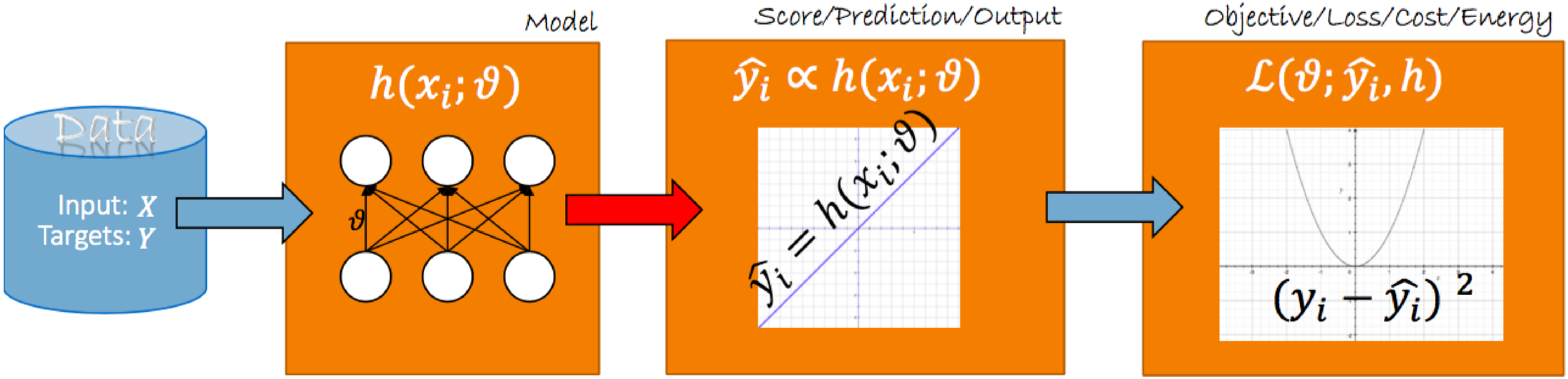
Forward computations

- Collect annotated data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon “forward propagation”
- Evaluate predictions and update model weights



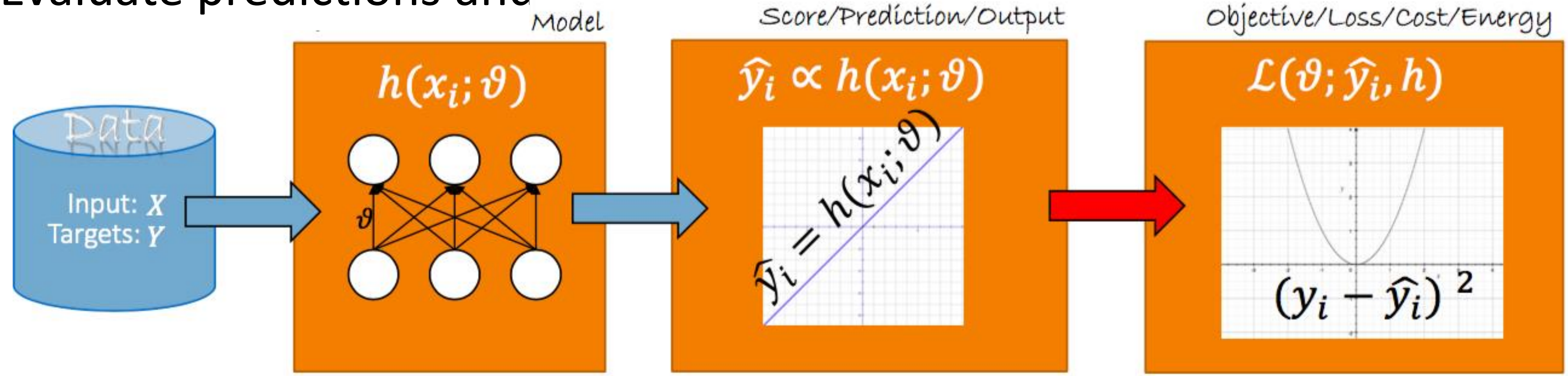
Forward computations

- Collect annotated data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon "forward propagation"
- Evaluate predictions and update model weights



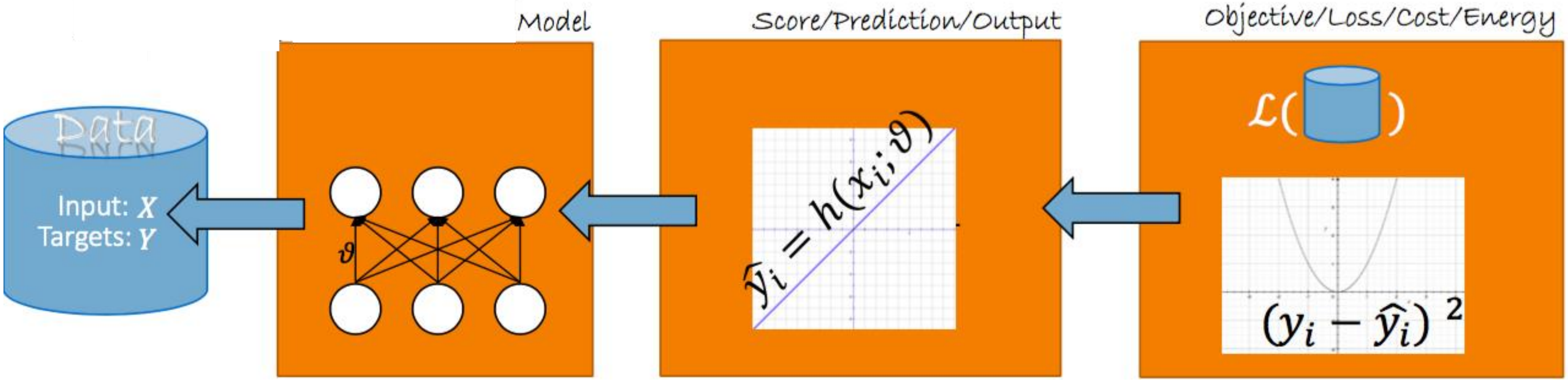
Forward computations

- Collect annotated data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon "forward propagation"
- Evaluate predictions and update model weights



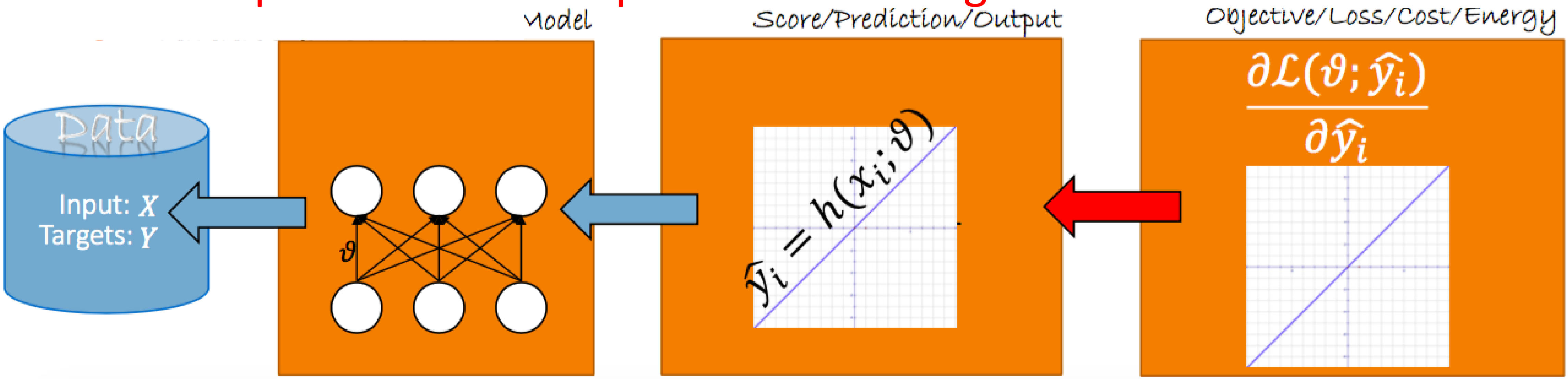
Backward computations

- Collect gradient data
- Define model and initialize randomly
- Predict based on current model
    - In neural network jargon "backpropagation"
- Evaluate predictions and update model weights



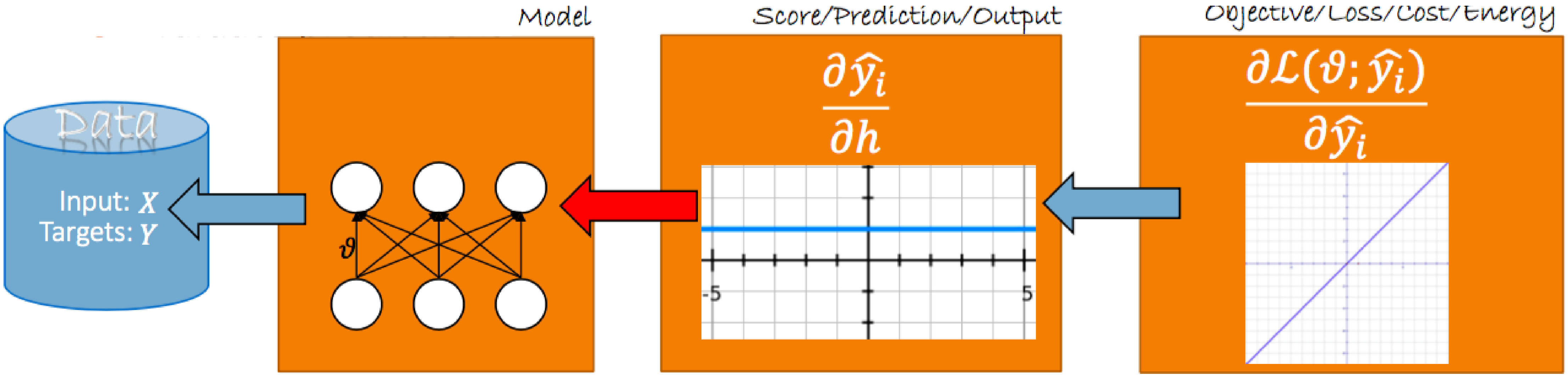
Backward computations

- Collect gradient data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon “backpropagation”
- Evaluate predictions and update model weights



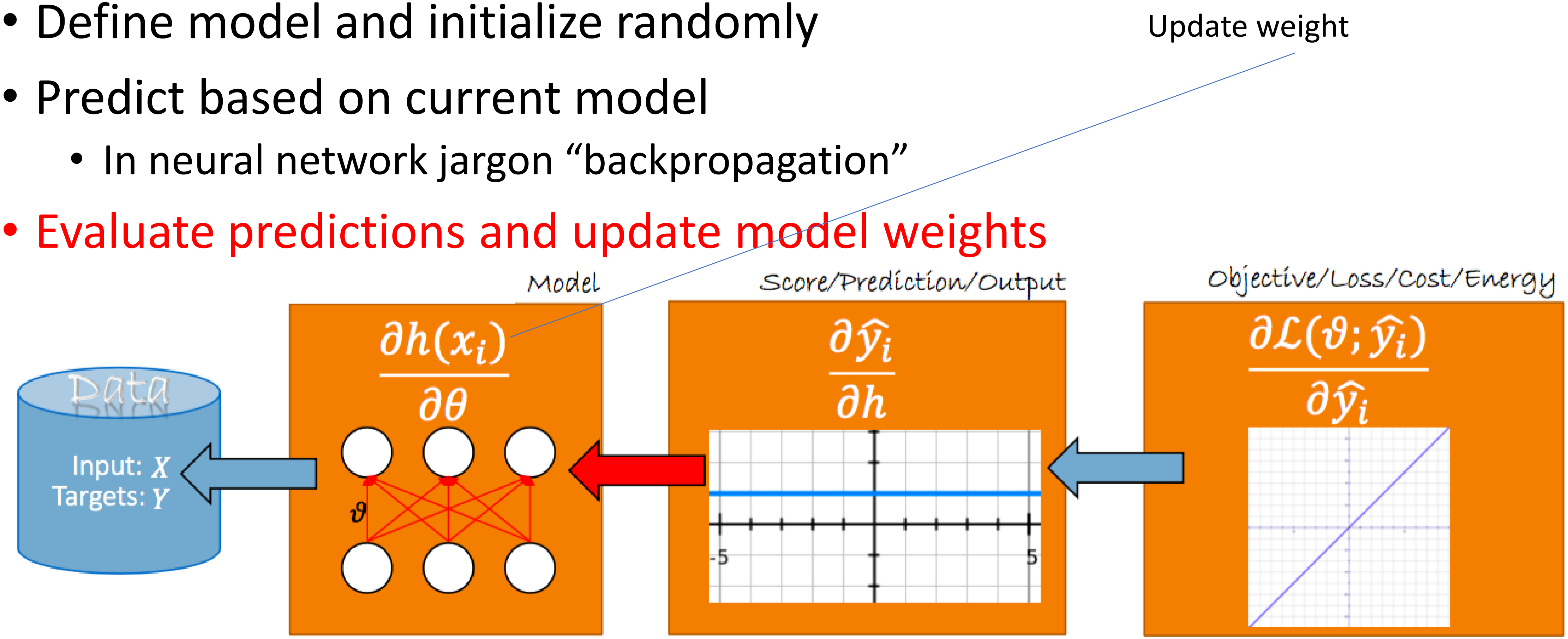
Backward computations

- Collect gradient data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon "backpropagation"
- Evaluate predictions and update model weights



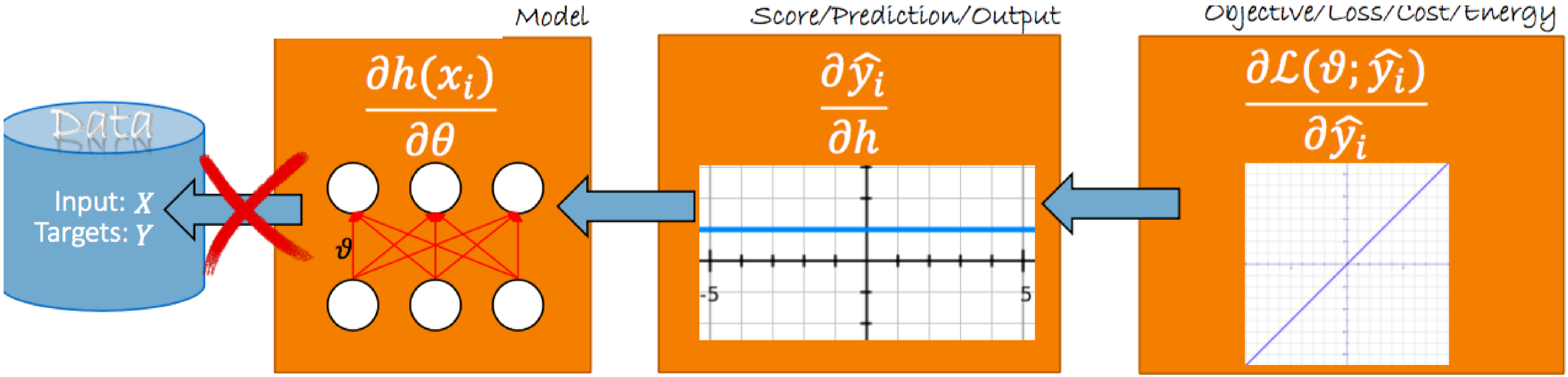
Backward computations

- Collect gradient data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon "backpropagation"
- Evaluate predictions and update model weights



Backward computations

- Collect gradient data
- Define model and initialize randomly
- Predict based on current model
  - In neural network jargon “backpropagation”
- Evaluate predictions and update model weights



Recall: Training Objective

Given training corpus $\{X, Y\}$ find optimal parameters

Find an optimal model parameterised over θ

Ground truth

Prediction

$$\theta^* \leftarrow \arg\min_\theta \sum_{(x,y)\subseteq(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L}))$$

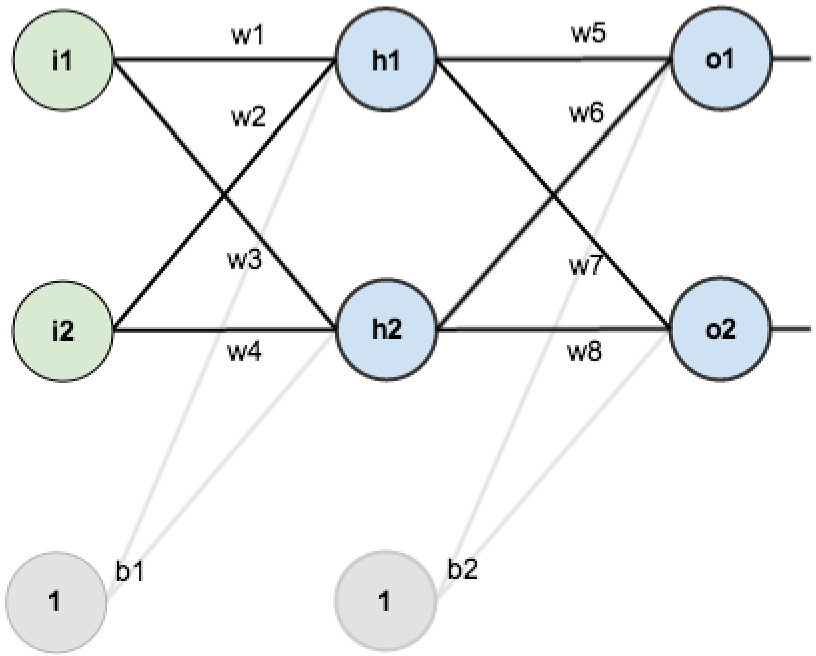
accumulated loss

Loss function

Forward Computation

Example

Network:



Loss function:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Example

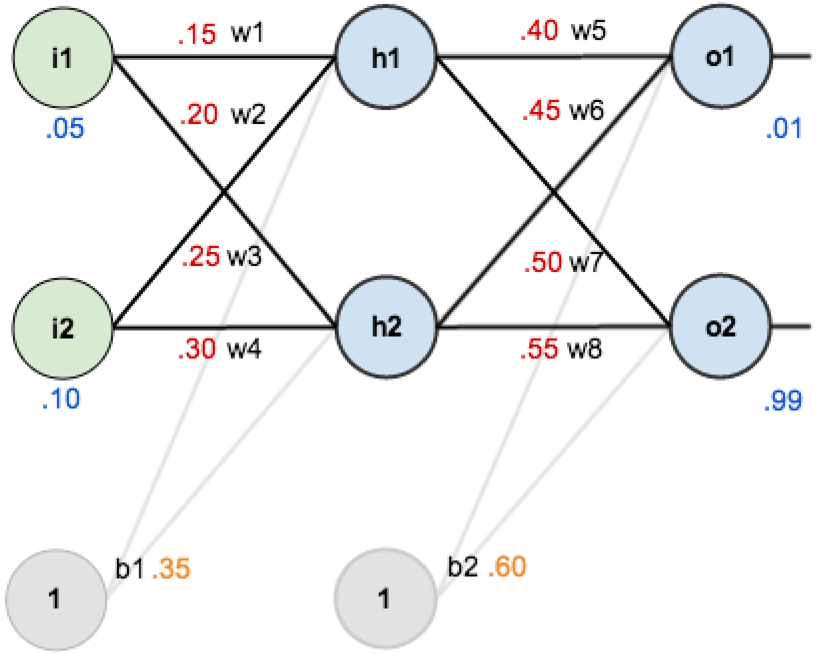
- Training data (a single sample)
  - given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

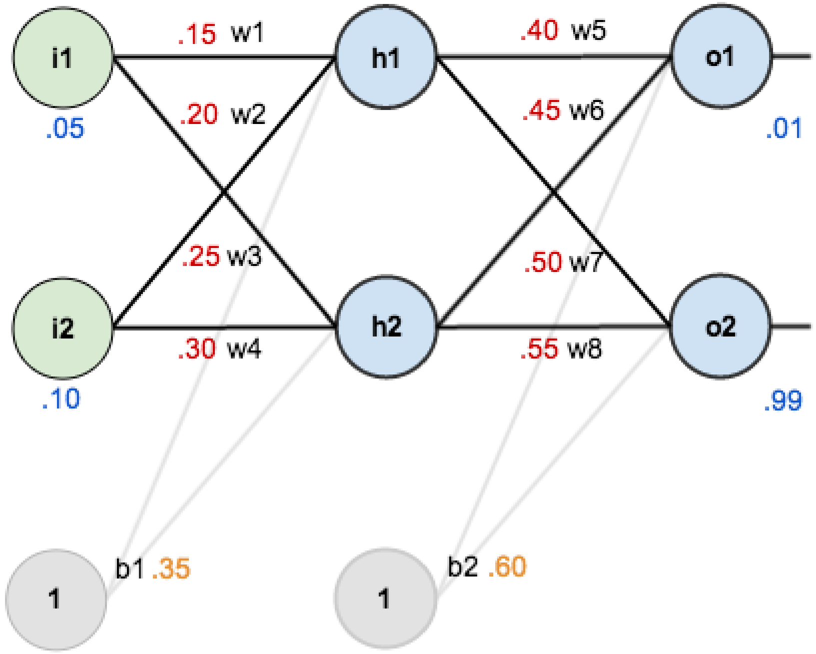
$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$



Example

- Training data (a single sample)
  - given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

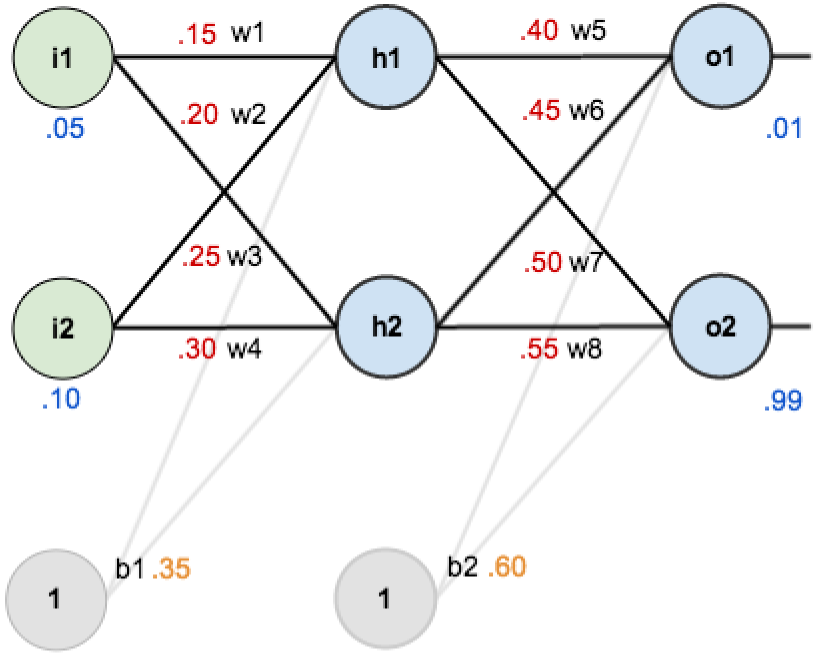
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

Example

- Training data (a single sample)
  - given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.



$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

Backward Propogation

Recall: Minimizing with multiple dimensional inputs

- We often minimize functions with multiple-dimensional inputs

$$f : \mathrm{R}^n \to \mathrm{R}$$

- For minimization to make sense there must still be only one (scalar) output

Functions with multiple inputs

- Partial derivatives

$$\frac{\partial}{\partial x_i} f(x)$$

measures how *f* changes as only variable x_i increases at point ***x***

- Gradient generalizes notion of derivative where derivative is wrt a vector
- Gradient is vector containing all of the partial derivatives denoted

$$\nabla_x f(x) = (\frac{\partial}{\partial x_1} f(x), \ldots, \frac{\partial}{\partial x_n} f(x))$$

Note: In the training objective case,

f is the loss $\sum_{(x,y)\in(X,Y)} l(y, a_L(x;\theta_{1,\ldots,L}))$

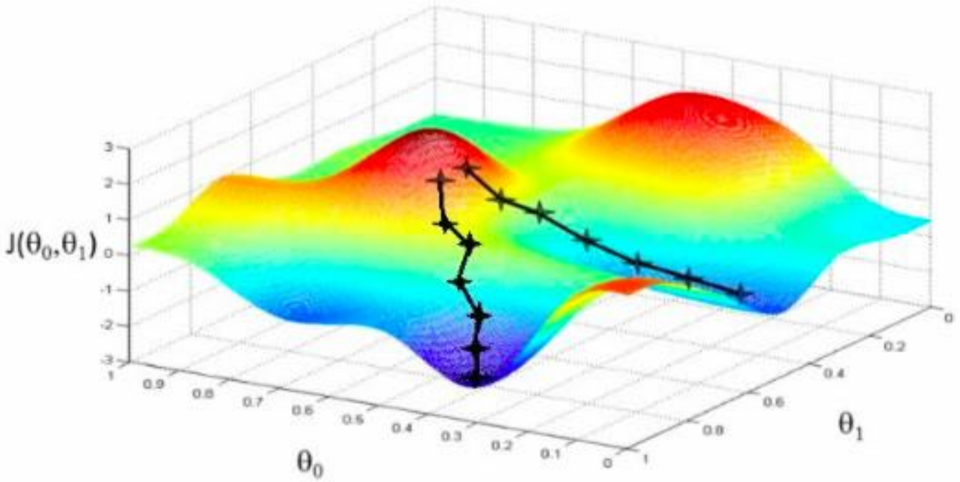
the parameter x is θ

Optimization through Gradient Descent

- As with many model, we optimize our neural network with Gradient Descent

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla_\theta \mathcal{L}$$

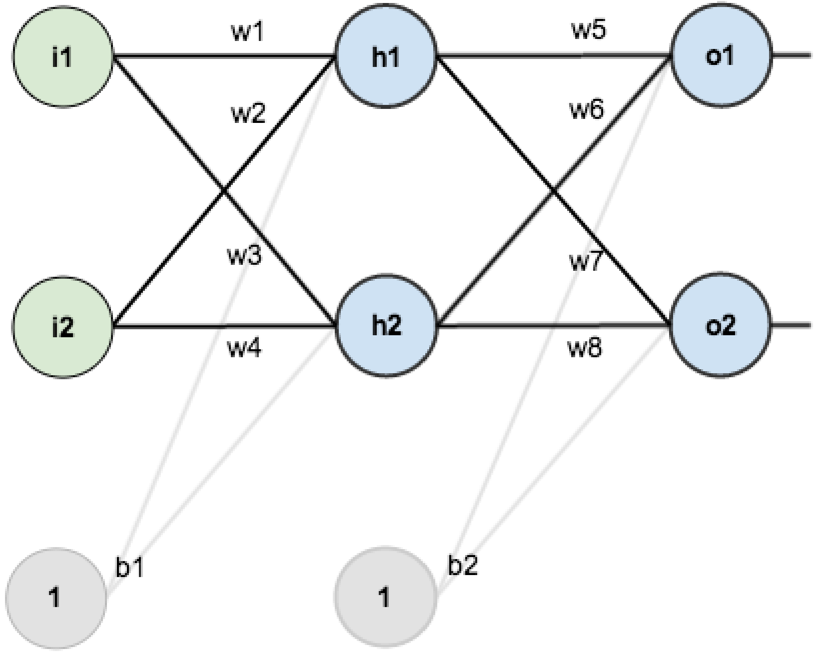
- The most important component in this formulation is the gradient
- Backpropagation to the rescue
  - The backward computations of network return the gradients
  - How to make the backward computations



Weight Update

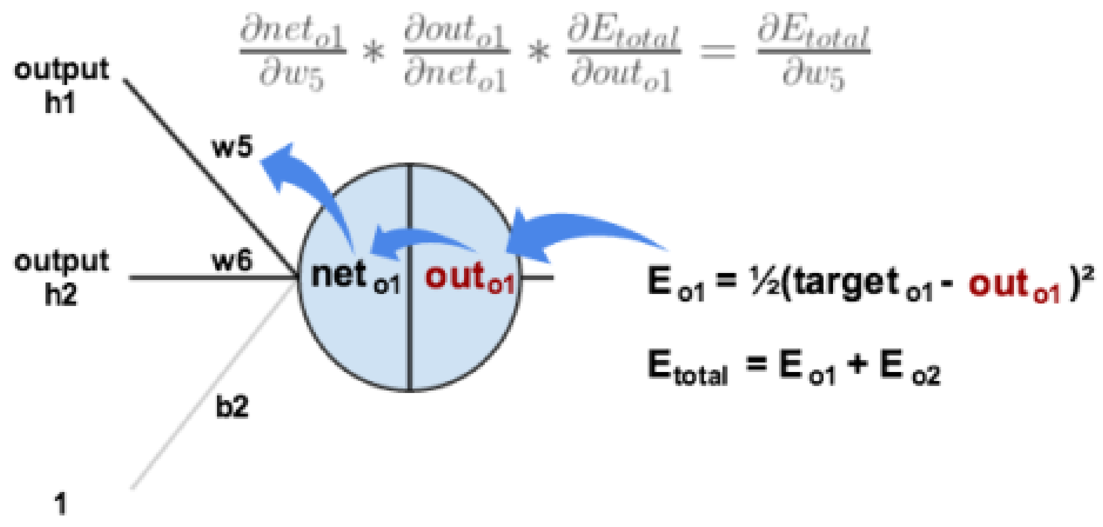
$$w_5^+ = w_5 - \eta \frac{\partial E_{total}}{\partial w_5}$$

$$w_1^+ = w_1 - \eta \frac{\partial E_{total}}{\partial w_1}$$

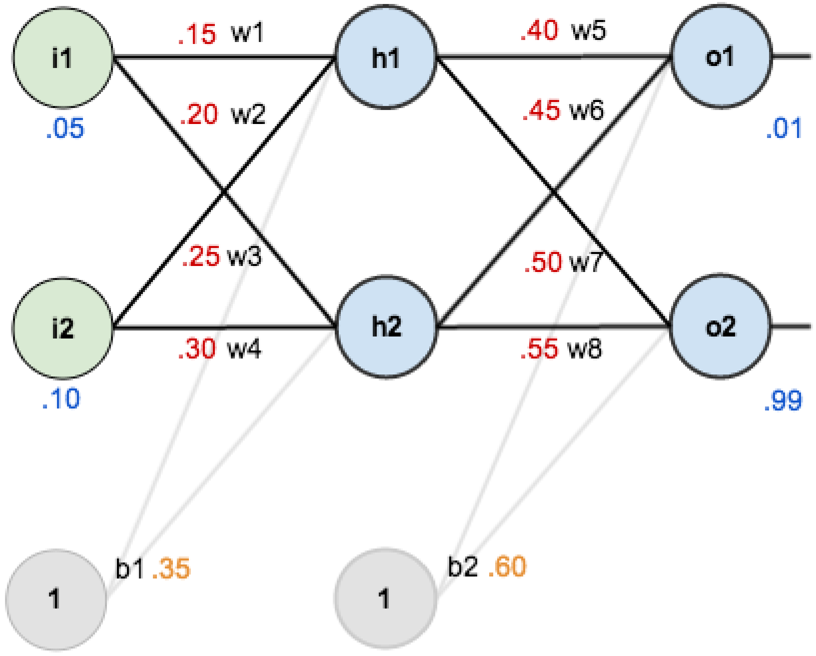


where $E_{total} = \sum \frac{1}{2}(target - output)^2$

Example



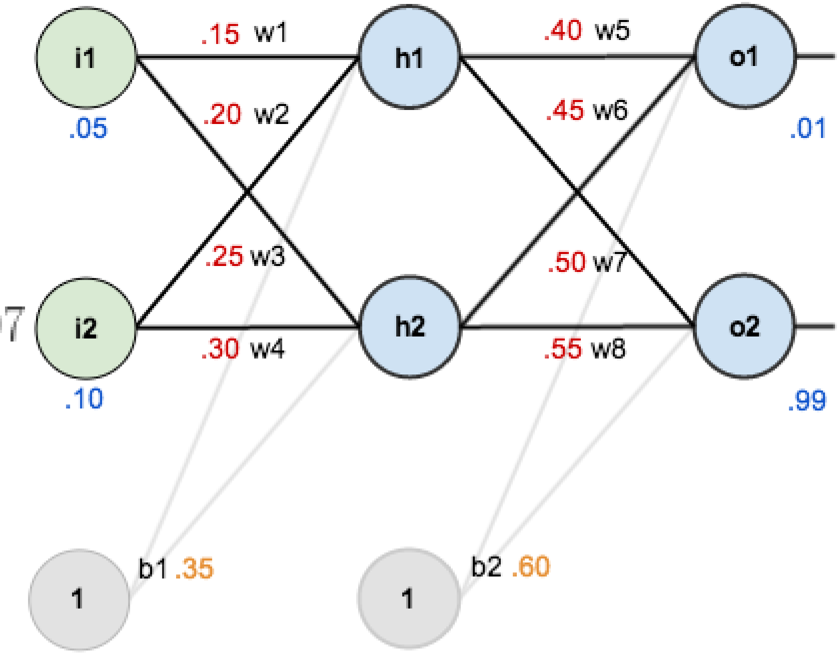
$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$
output h1
w5
output h2
w6
b2
1
net_{o1}
out_{o1}
$E_{o1} = ½(target_{o1} - out_{o1})^2$
$E_{total} = E_{o1} + E_{o2}$



i1
.05
i2
.10
.15 w1
.20 w2
.25 w3
.30 w4
h1
h2
.40 w5
.45 w6
.50 w7
.55 w8
o1
.01
o2
.99
1
b1 .35
1
b2 .60

Example

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$



1. $\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$

   $\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$

2. $out_{o1} = \frac{1}{1+e^{-net_{o1}}}$

   $\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$

3. $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

   $\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$

Example

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

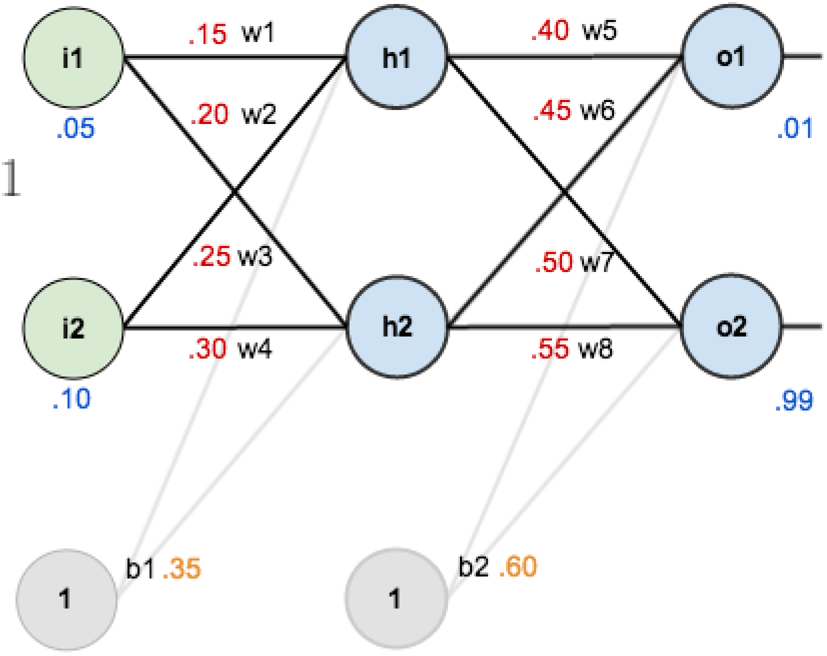
$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

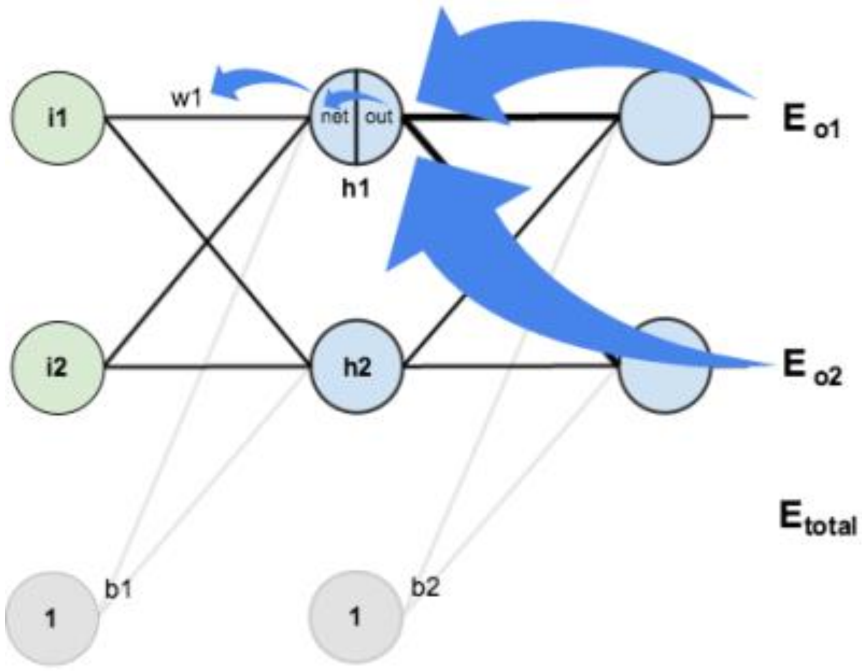
$$w_8^+ = 0.561370121$$



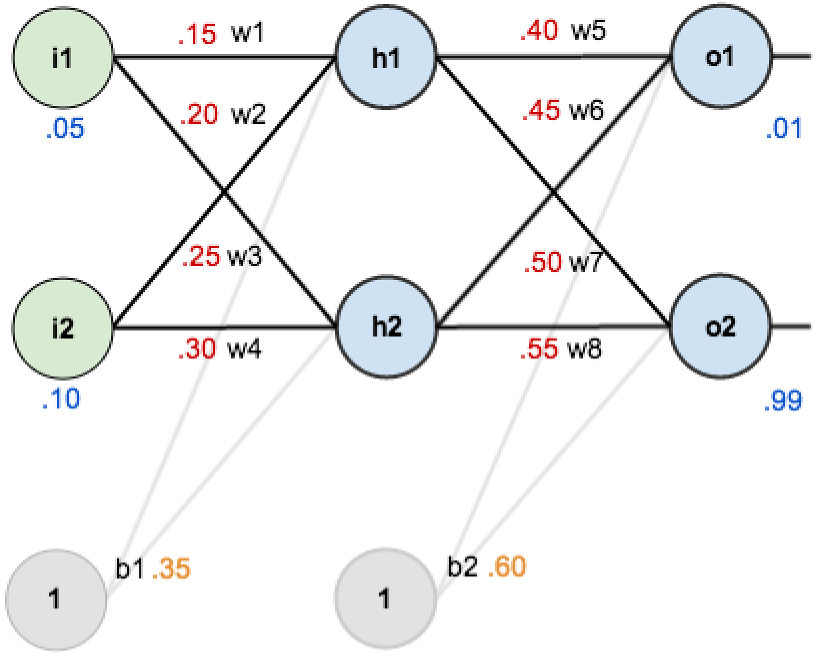
Example

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$$E_{total} = E_{o1} + E_{o2}$$



Chain rule

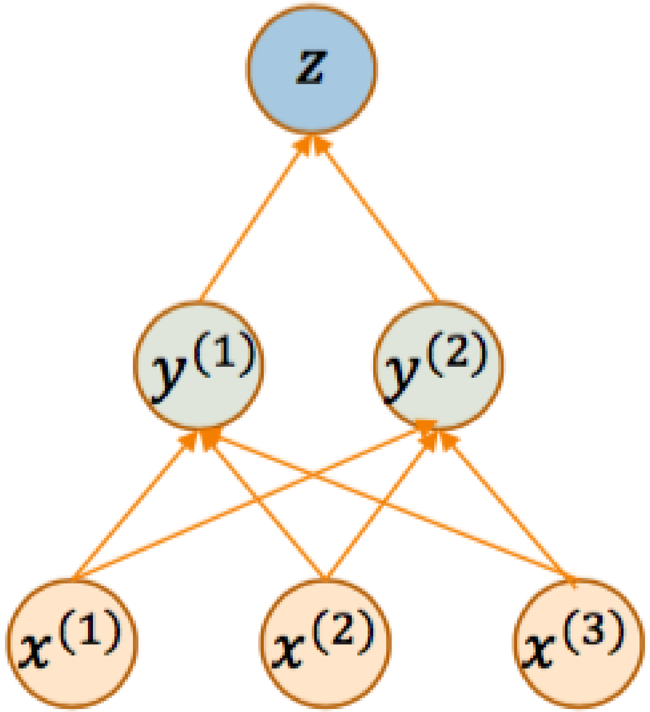
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- When $x \in$ R^m,$y \in$ R^n. $z \in$R

$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx_i}$$

  - i.e., gradients from all possible paths



Chain rule

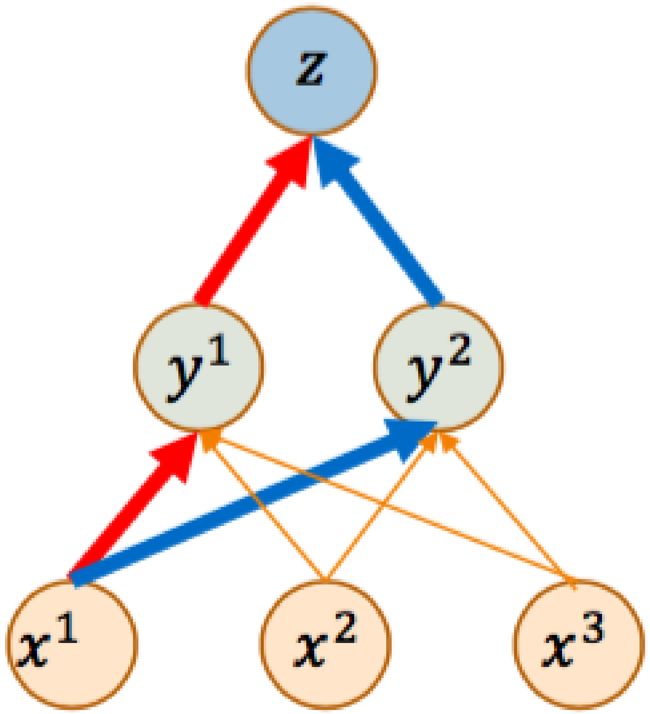
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- When $x \in \mathrm{R}^m$,$y \in \mathrm{R}^n$. $z \in \mathrm{R}$

$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx_i}$$

  - i.e., gradients from all possible paths



$$\frac{dz}{dx^1} = \frac{dz}{dy^1}\frac{dy^1}{dx^1} + \frac{dz}{dy^2}\frac{dy^2}{dx^1}$$

Chain rule

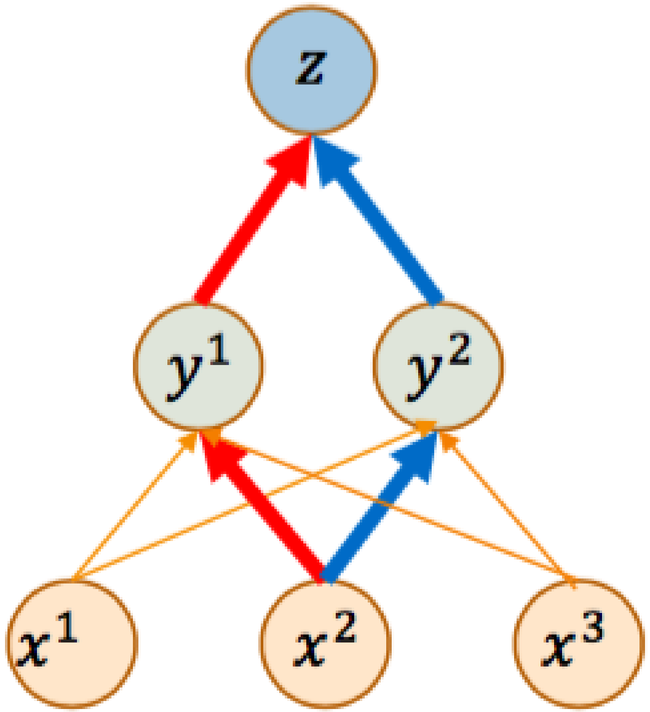
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- When $x \in \mathrm{R}^m, y \in \mathrm{R}^n . z \in \mathrm{R}$

$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx_i}$$

  - i.e., gradients from all possible paths



$$\frac{dz}{dx^2} = \frac{dz}{dy^1}\frac{dy^1}{dx^2} + \frac{dz}{dy^2}\frac{dy^2}{dx^2}$$

Chain rule

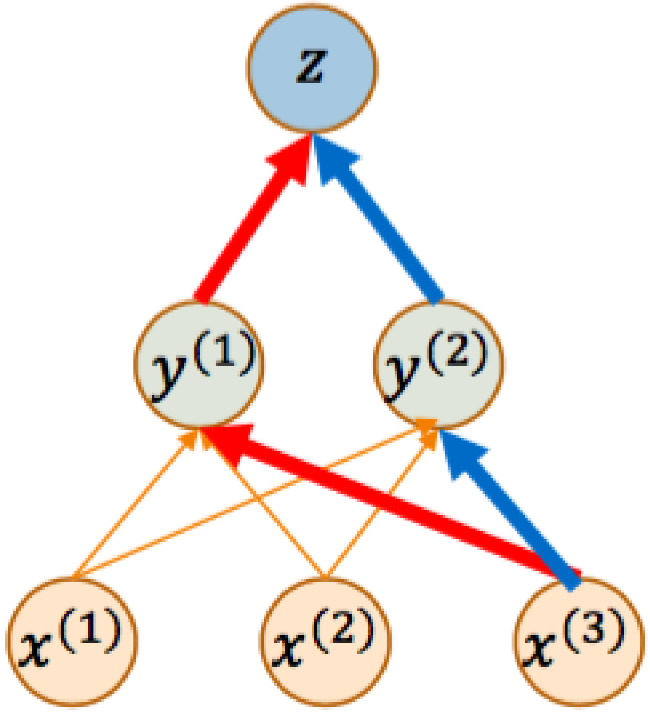
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- When $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, $z \in \mathbb{R}$

$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx_i}$$

  - i.e., gradients from all possible paths



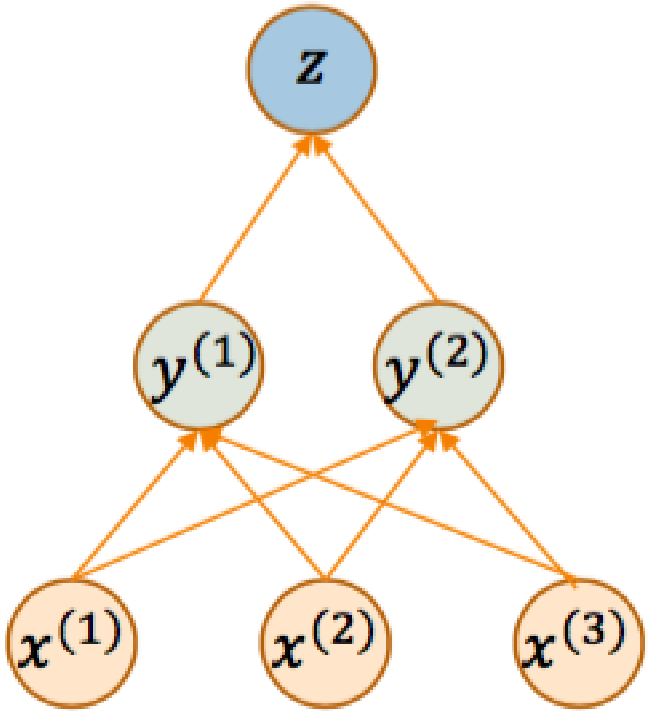
$$\frac{dz}{dx^3} = \frac{dz}{dy^1}\frac{dy^1}{dx^3} + \frac{dz}{dy^2}\frac{dy^2}{dx^3}$$

Chain rule

- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z : $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$
- When $x \in R^m, y \in R^n, z \in R$

$$\frac{dz}{dx_i} = \sum_j \frac{dz}{dy_j}\frac{dy_j}{dx_i}$$

  - i.e., gradients from all possible paths
  - or in vector notation $\boxed{\frac{dz}{d\boldsymbol{x}} = \left(\frac{d\boldsymbol{y}}{d\boldsymbol{x}}\right)^T \cdot \frac{dz}{d\boldsymbol{y}}}$
  - $\frac{d\boldsymbol{y}}{d\boldsymbol{x}}$ is the Jacobian



The Jacobian

- When $x \in$ R^3, $y \in$ R^2

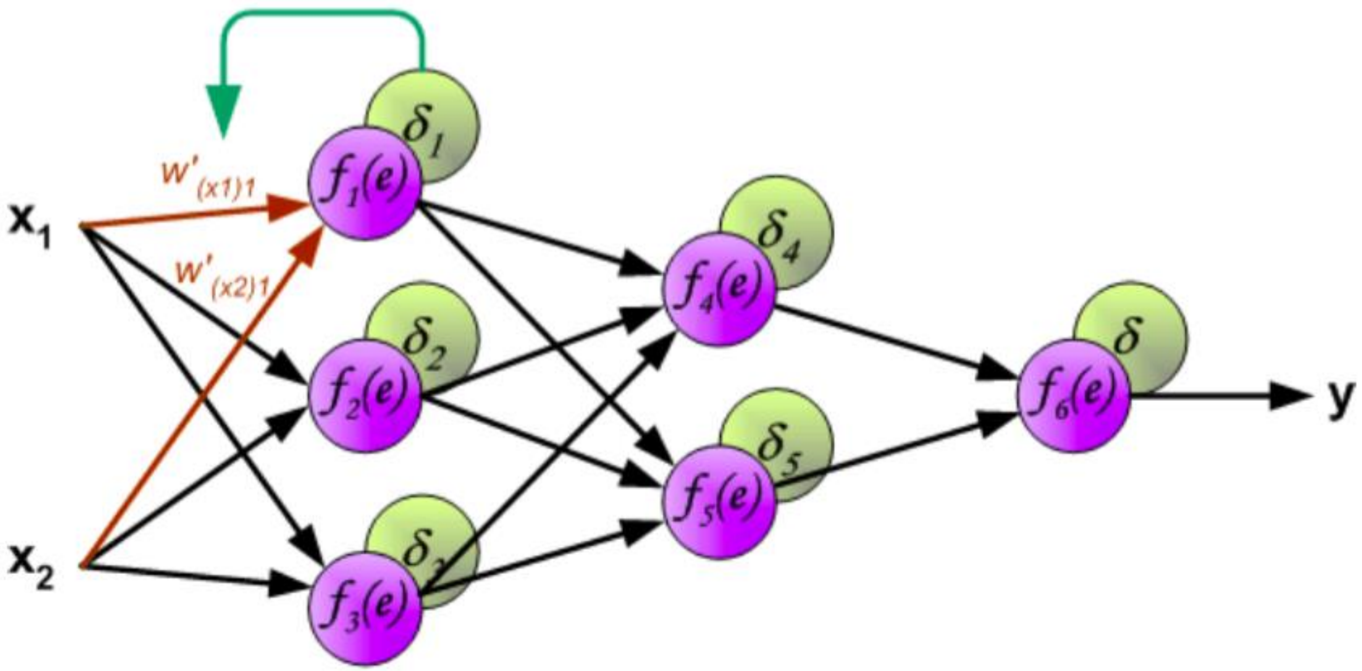
$$J(y(x)) = \frac{d\boldsymbol{y}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial y^{(1)}}{\partial x^{(1)}} & \frac{\partial y^{(1)}}{\partial x^{(2)}} & \frac{\partial y^{(1)}}{\partial x^{(3)}} \\ \frac{\partial y^{(2)}}{\partial x^{(1)}} & \frac{\partial y^{(2)}}{\partial x^{(2)}} & \frac{\partial y^{(2)}}{\partial x^{(3)}} \end{bmatrix}$$

Chain rule in practice

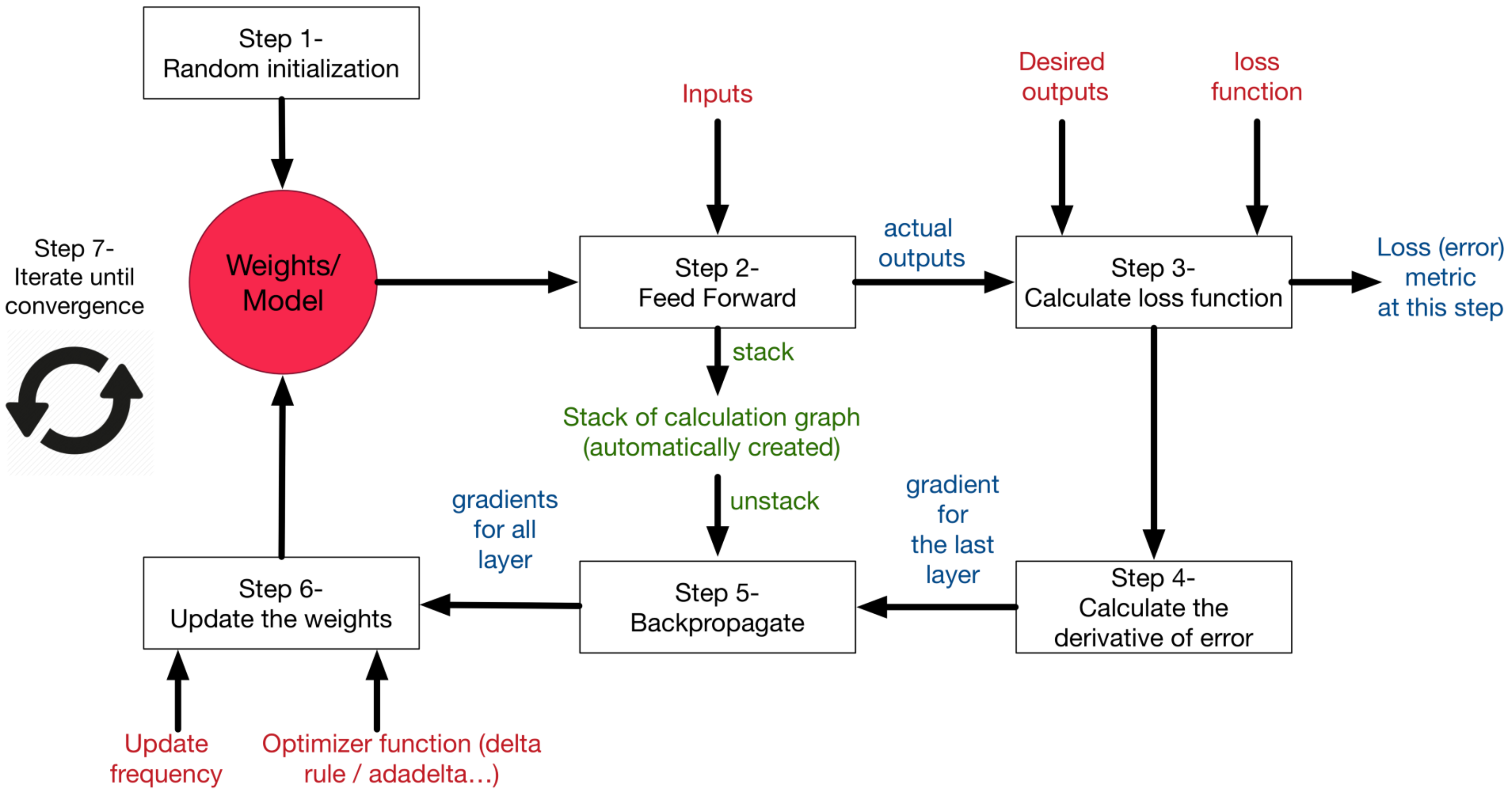
- f(y) =sin(y) ,y=$g(x)$ =0.5x^2

$$\frac{df}{dx} = \frac{d\,[\sin(y)]}{dg}\frac{d[0.5x^2]}{dx}$$
$$= \cos(0.5x^2) \cdot x$$

Backpropagation



General Workflow



Regularization as Constraints

Recall: what is regularization?

- In general: any method to prevent overfitting or help the optimization

- Specifically: additional terms in the training optimization objective to prevent overfitting or help the optimization

Recall: Overfitting

- Key: empirical loss and expected loss are different

- Smaller the data set, larger the difference between the two
- Larger the hypothesis class, easier to find a hypothesis that fits the difference between the two
  - Thus has small training error but large test error (overfitting)

- Larger data set helps
- Throwing away useless hypotheses also helps (regularization)

Regularization as hard constraint

- Training objective

$$\min_f \hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i)$$

$$\text{subject to: } f \in \mathcal{H}$$

- When parametrized

$$\min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

$$\text{subject to: } \theta \in \Omega$$

Adding constraints

Regularization as hard constraint

- When Ω measured by some quantity R

$$\min_{\theta} \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

$$\text{subject to: } R(\theta) \leq r$$

- Example: l_2 regularization

$$\min_{\theta} \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

$$\text{subject to: } ||\theta||_2^2 \leq r^2$$

Adding constraints

Regularization as soft constraint

- The hard-constraint optimization is equivalent to soft-constraint

$$\min_{\theta} \hat{L}_R(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i) + \lambda^* R(\theta)$$

for some regularization parameter $\lambda^* > 0$

- Example: l_2 regularization

$$\min_{\theta} \hat{L}_R(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i) + \lambda^* ||\theta||_2^2$$

Adding constraints

Regularization as soft constraint

- Showed by Lagrangian multiplier method

Adding constraints

$$\mathcal{L}(\theta, \lambda) := \hat{L}(\theta) + \lambda[R(\theta) - r]$$

- Suppose θ^* is the optimal for hard-constraint optimization

$$\theta^* = \operatorname*{argmin}_{\theta} \max_{\lambda \geq 0} \mathcal{L}(\theta, \lambda) := \hat{L}(\theta) + \lambda[R(\theta) - r]$$

- Suppose λ^* is the corresponding optimal for max

$$\theta^* = \operatorname*{argmin}_{\theta} \mathcal{L}(\theta, \lambda^*) := \hat{L}(\theta) + \lambda^*[R(\theta) - r]$$