Learning with neural networks

Dr. Xiaowei Huang

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

Up to now,

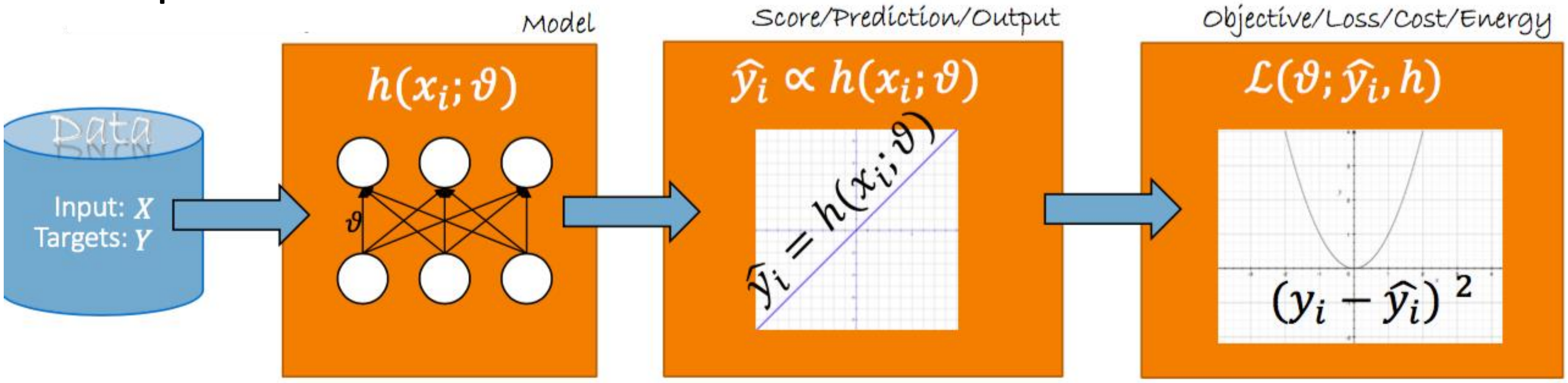
- Overview of Machine Learning
- Traditional Machine Learning Algorithms
- Deep learning
    - Introduction to Tensorflow
    - Introduction to Deep Learning
    - Functional view and features

Topics

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

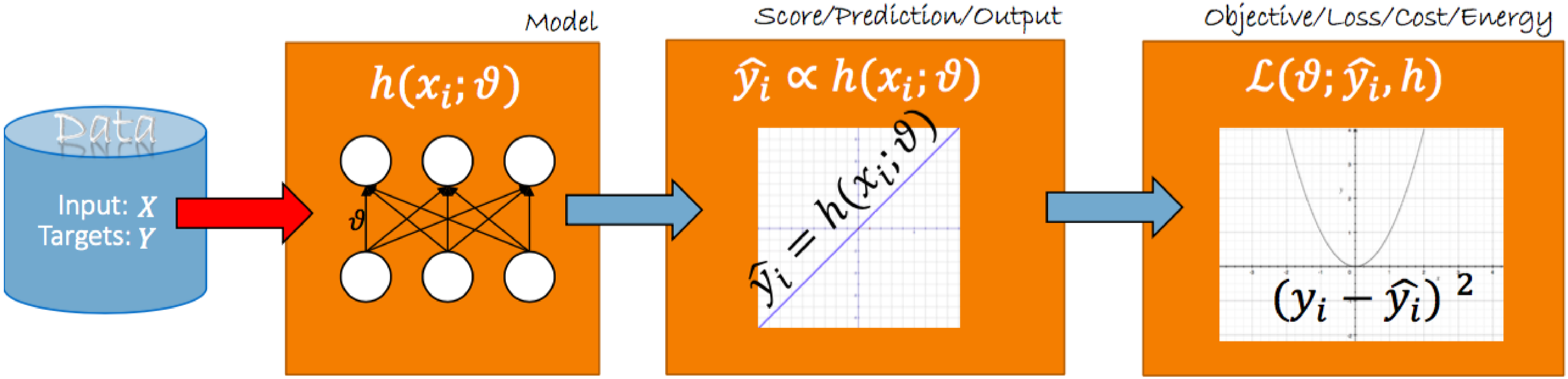
Forward computations

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



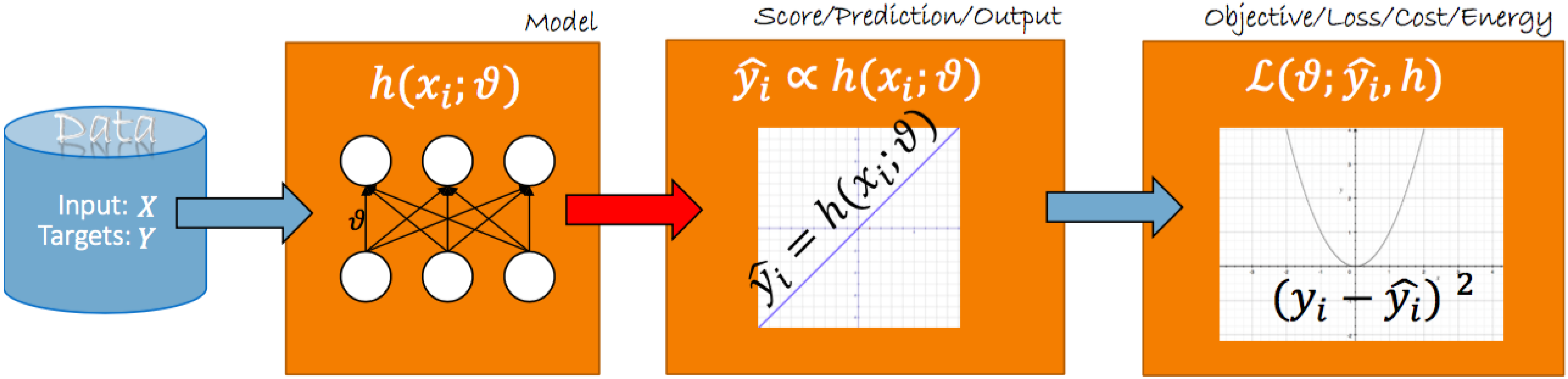
Forward computations

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



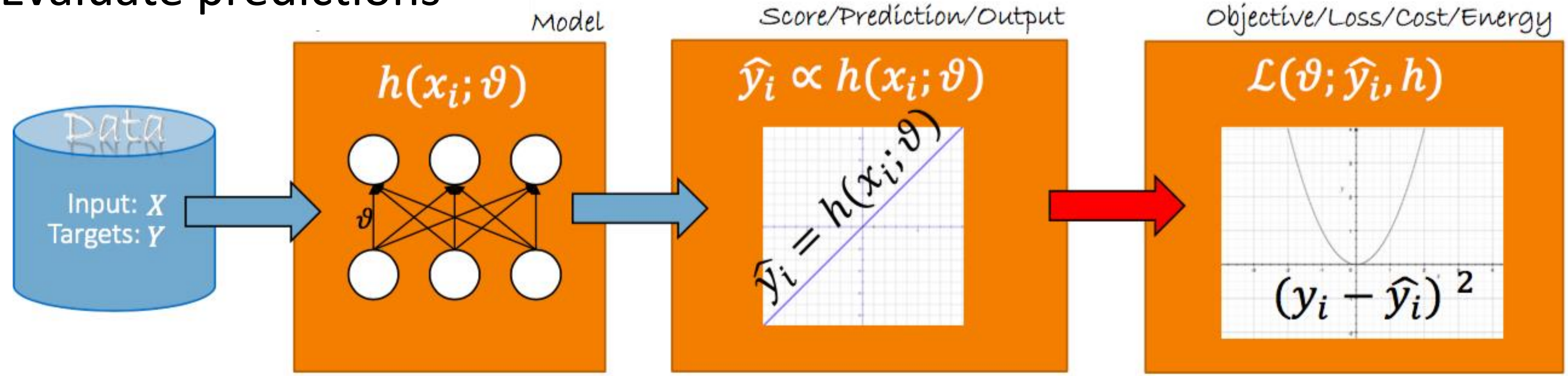
Forward computations

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



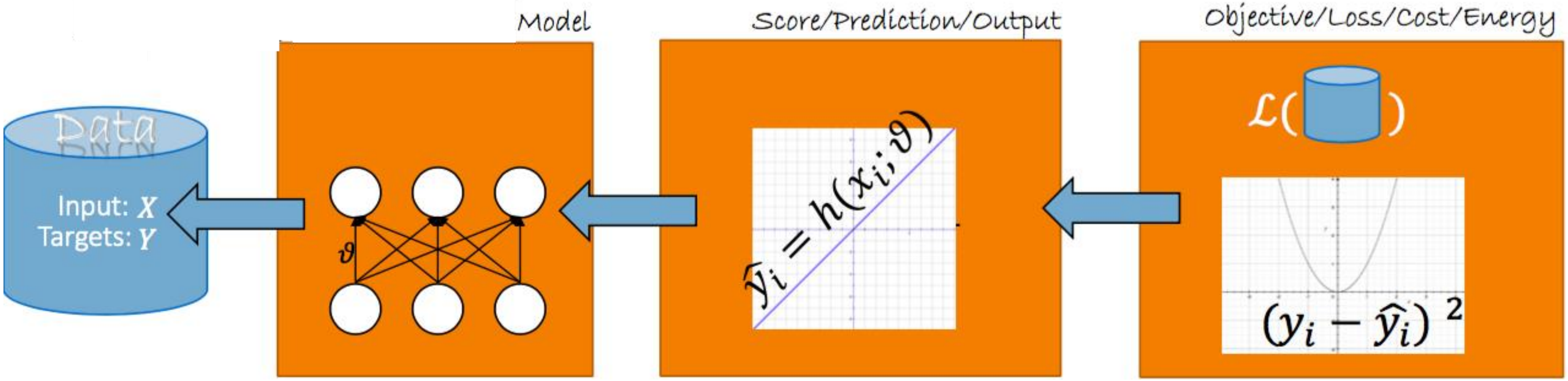
Forward computations

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



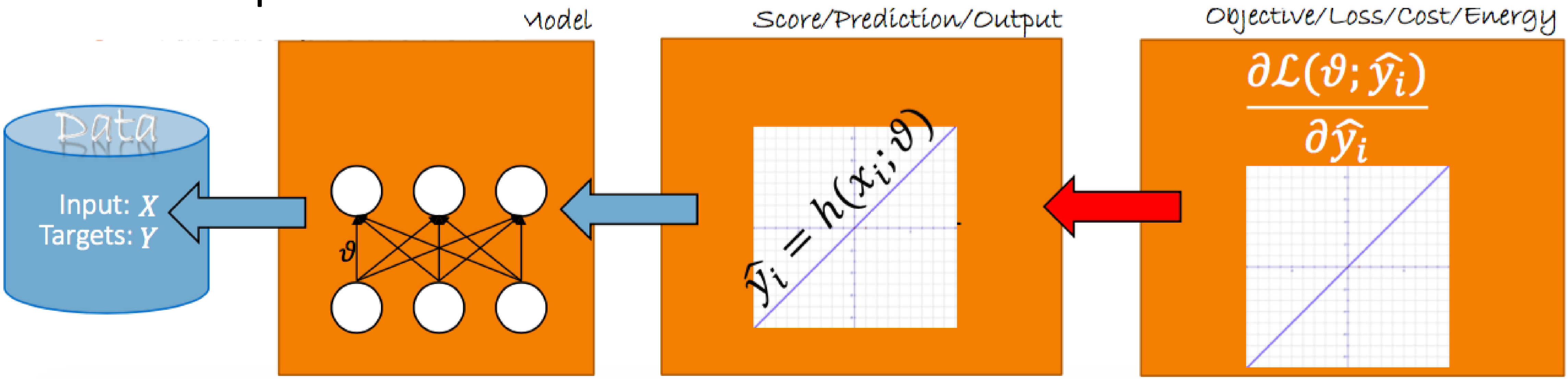
Backward computations

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



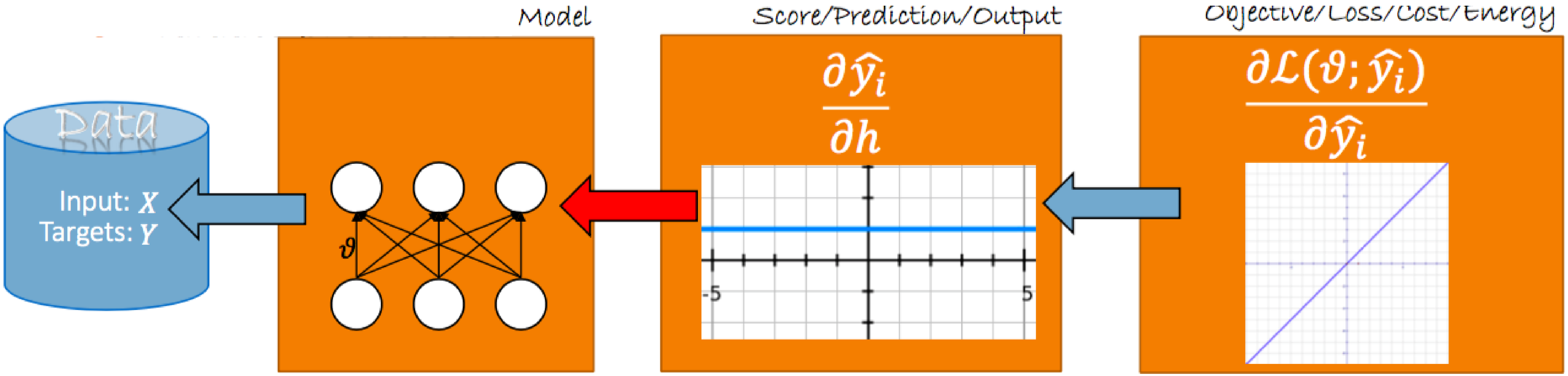
Backward computations

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



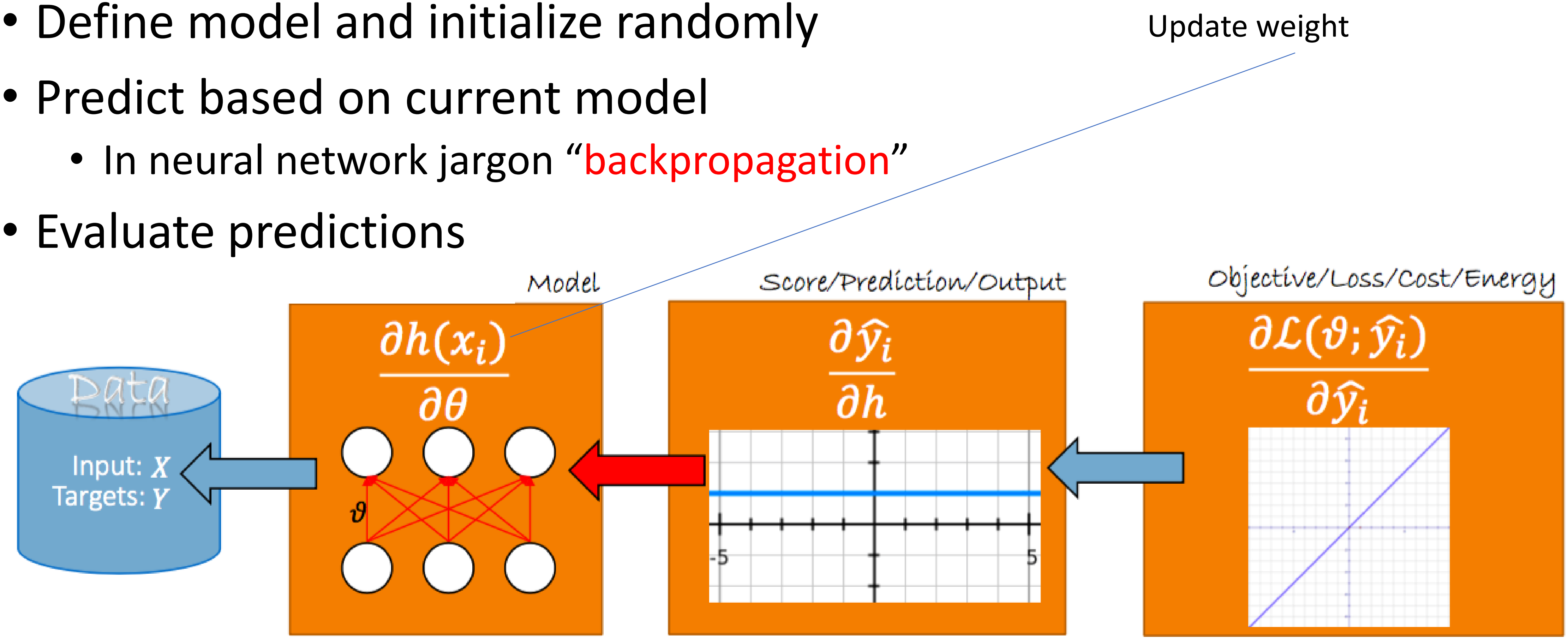
Backward computations

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



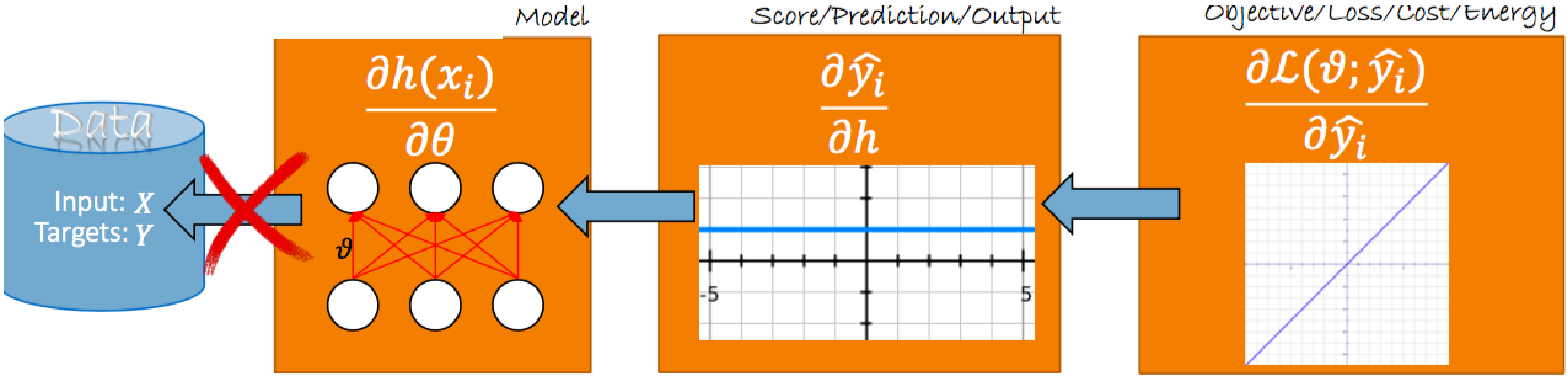
Backward computations

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



Backward computations

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



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

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

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

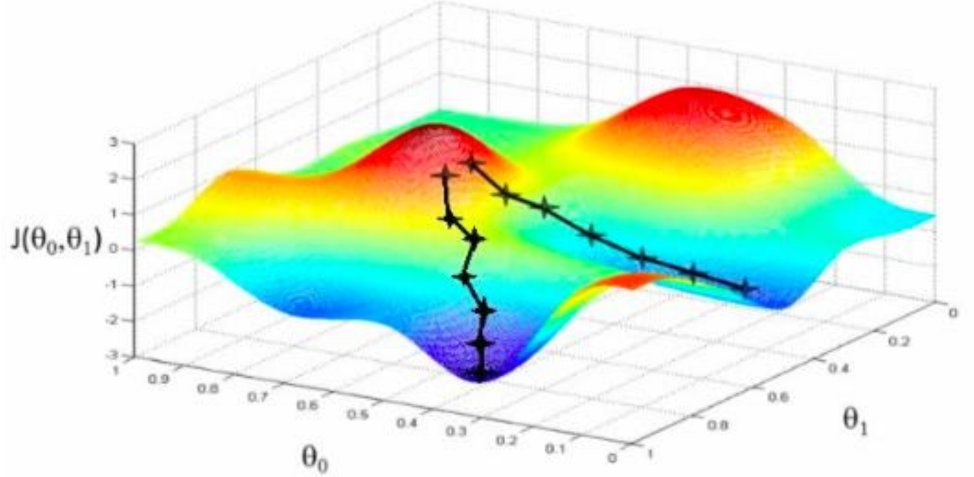
Note: In the training objective case, f is the loss, and the parameter x is \theta

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



Chain rule

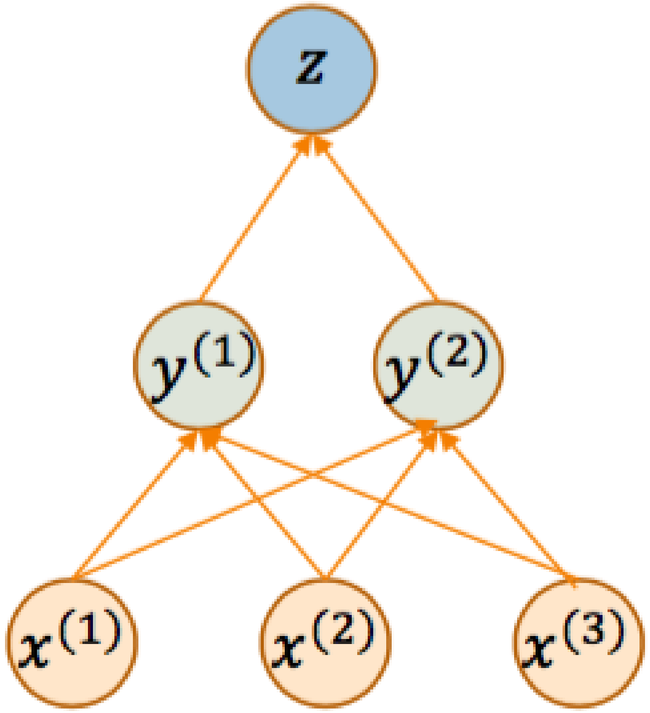
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

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

- 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



Chain rule

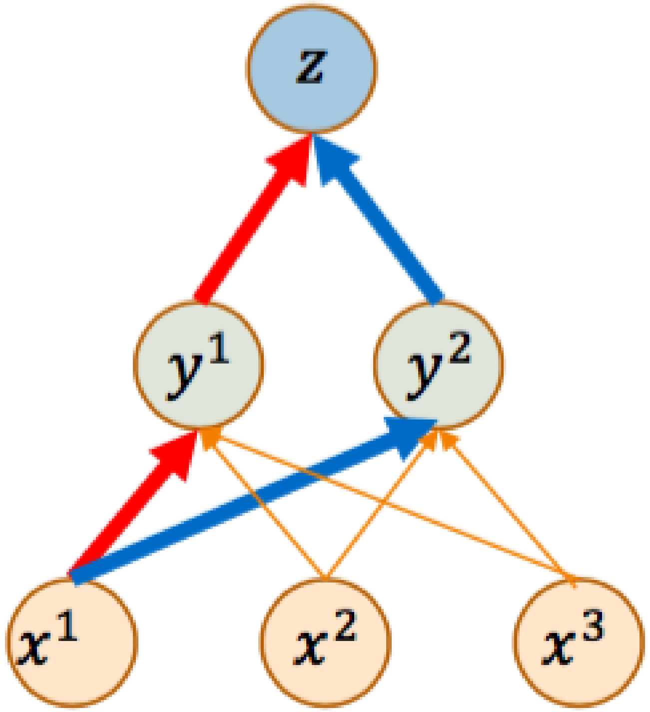
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

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

- 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

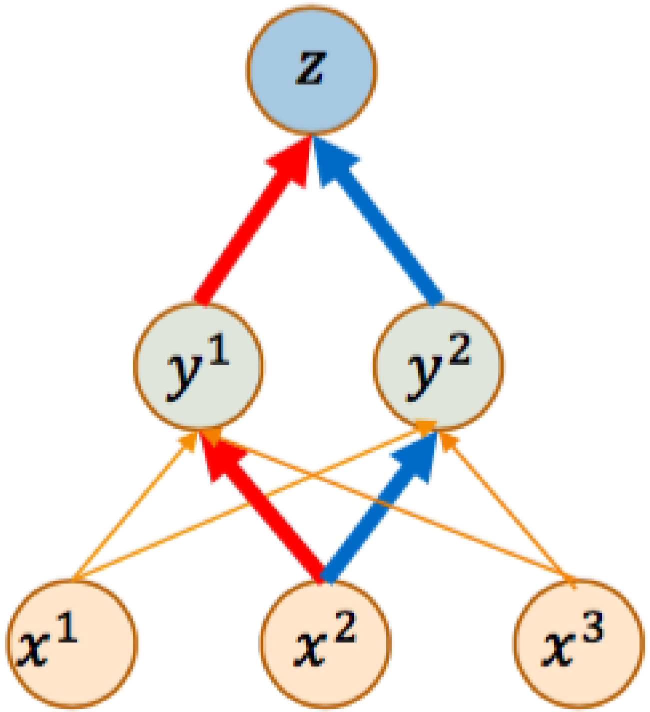
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

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

- 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

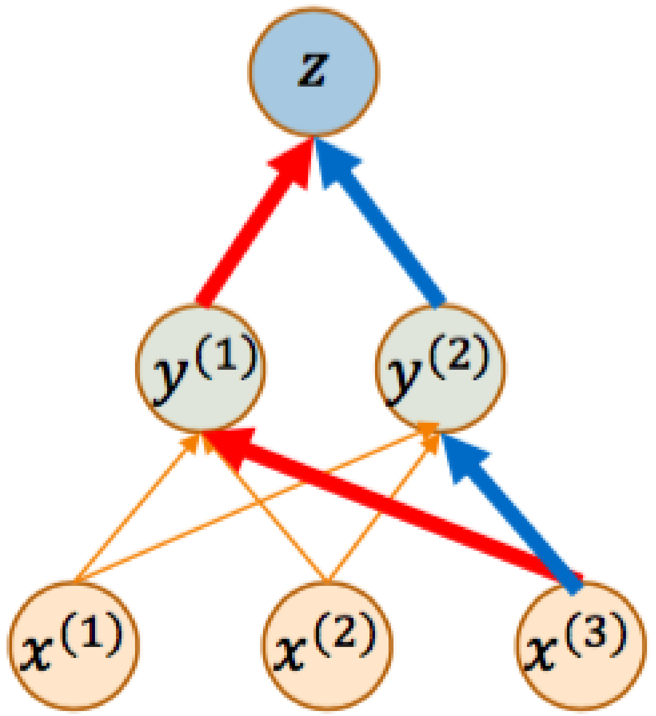
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars x, y, z

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

- When$x \in$ 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



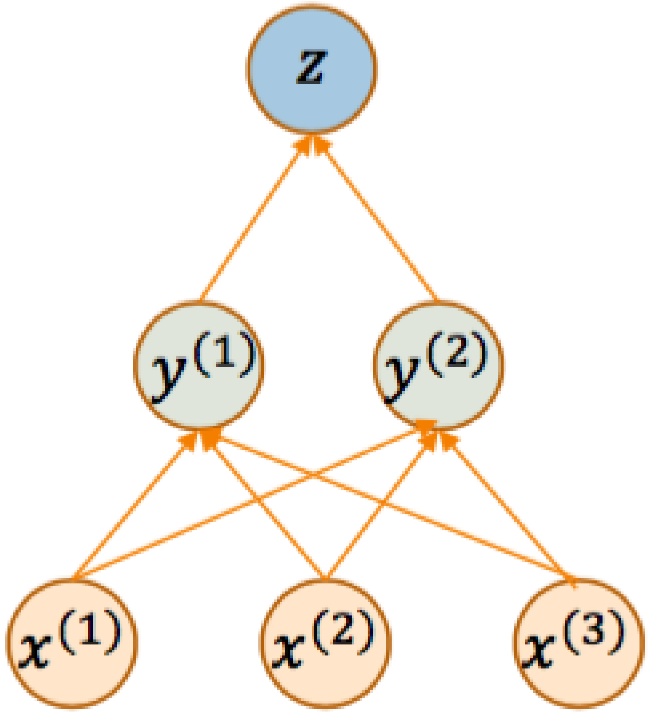
$$\frac{dz}{dx^3} = \frac{dz}{dy^1}\frac{dy^1}{dx^3} + \frac{dz}{dy^2}\frac{dy^2}{dx^3}$$

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

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

  - i.e., gradients from all possible paths
  - 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

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

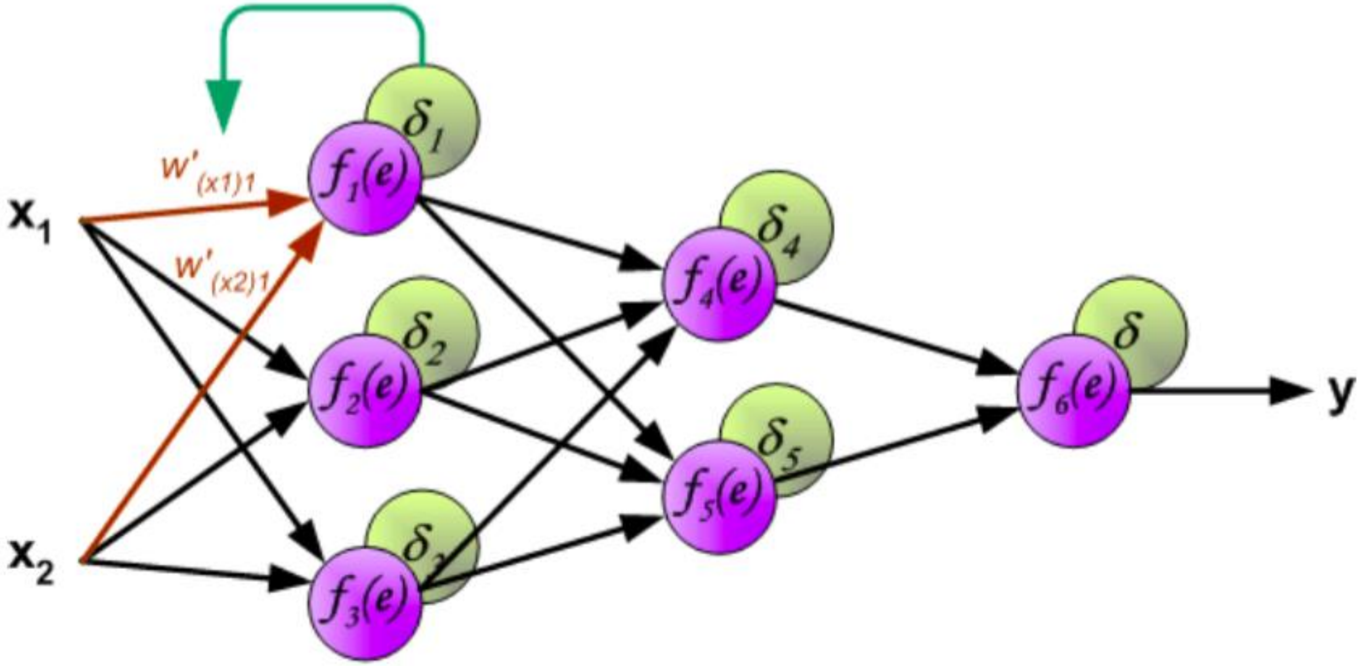
Chain rule in practice

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

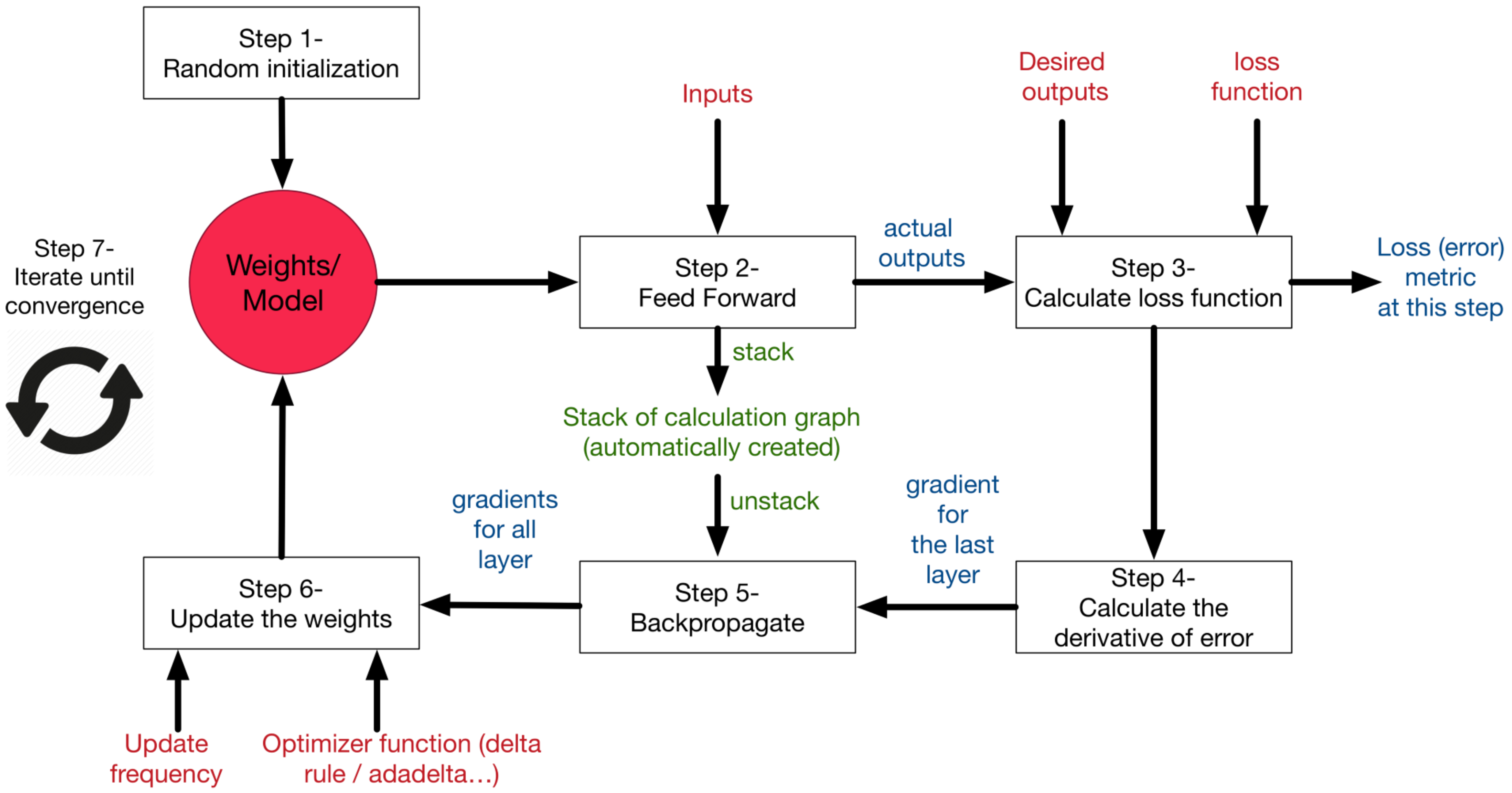
$$\frac{df}{dx} = \frac{d\,[\sin(y)]}{dg}\frac{d[0.5x^2]}{dx}$$

$$= \cos(0.5x^2) \cdot x$$

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

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

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

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

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

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

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

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

Regularization as soft constraint

- Showed by Lagrangian multiplier method

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

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

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

- Suppose λ^* is the corresponding optimal for max

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