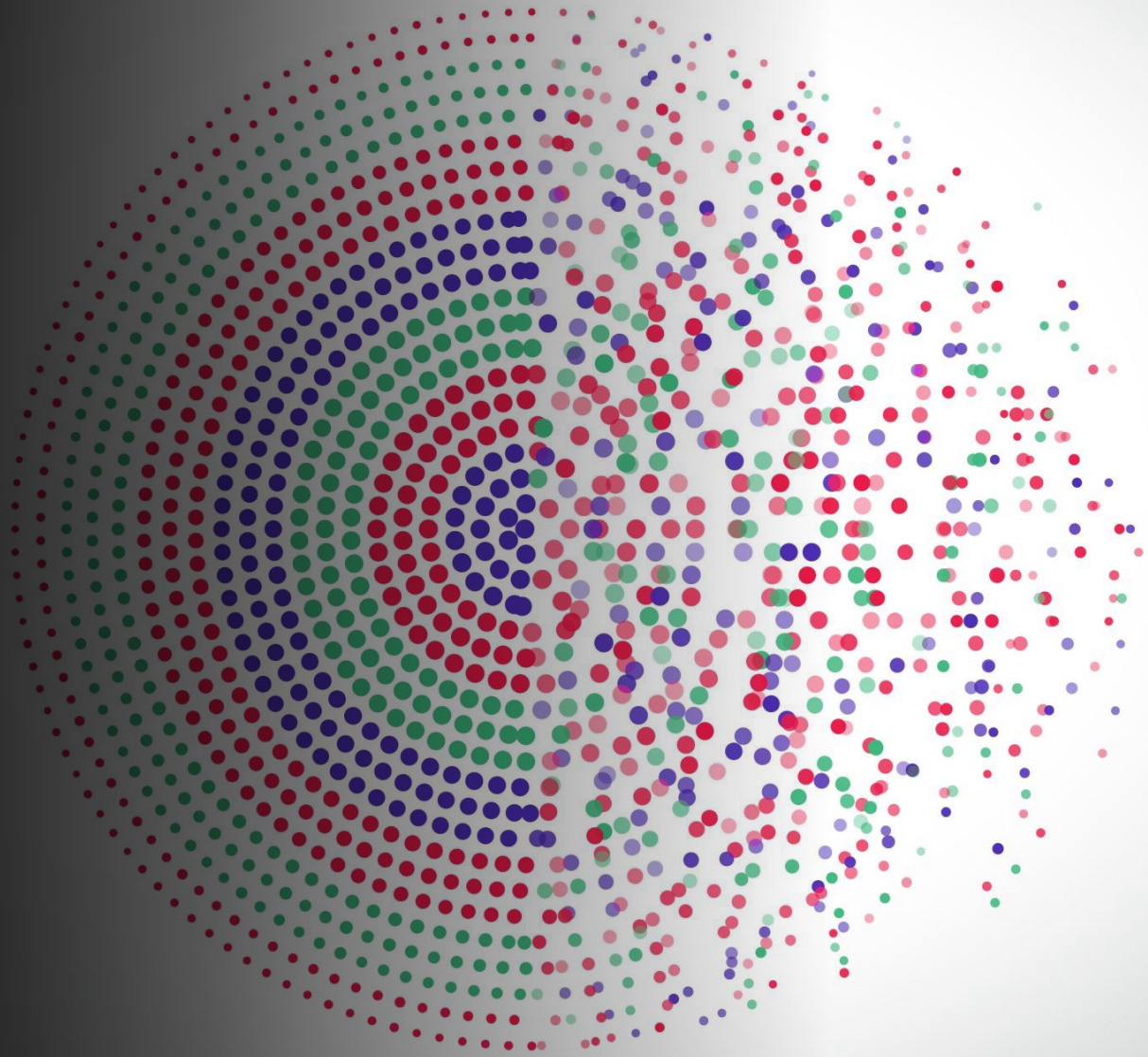
# Lecture 18 – Deep Learning: Training Algorithm



Prof Xiaowei Huang

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

(Attendance Code: **453995**)

# Up to now,

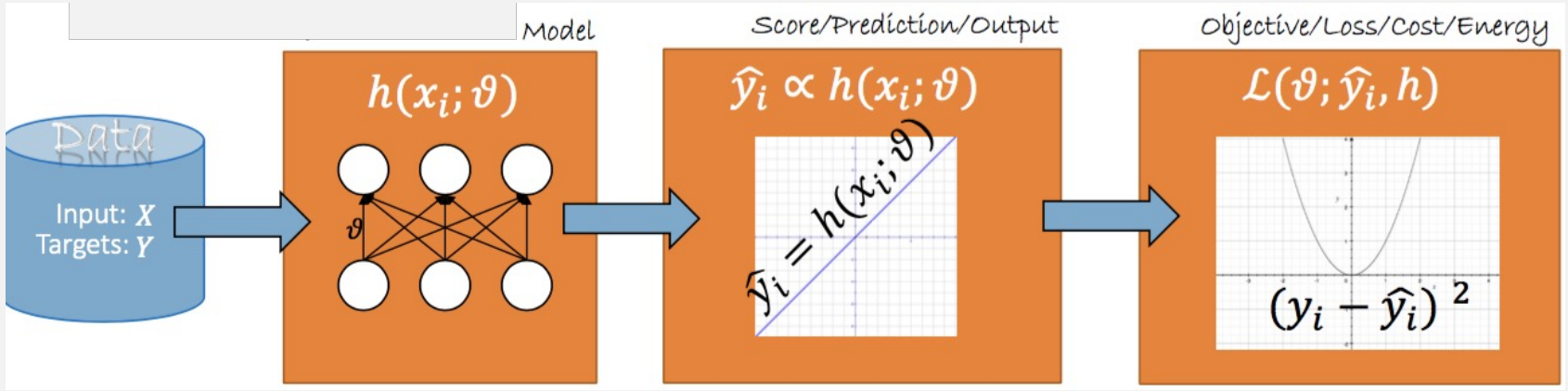
- Traditional Machine Learning Algorithms
- Adversarial Attack and Defence
- Deep learning
  - Introduction to Deep Learning
  - Functional view and features

# Topics

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

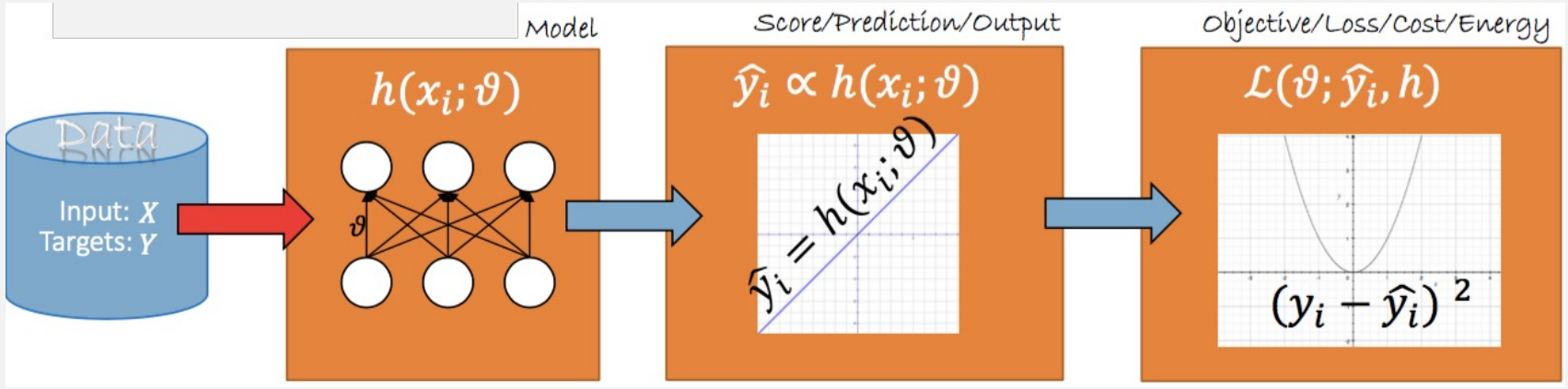
# Forward computations

- Collect annotated data



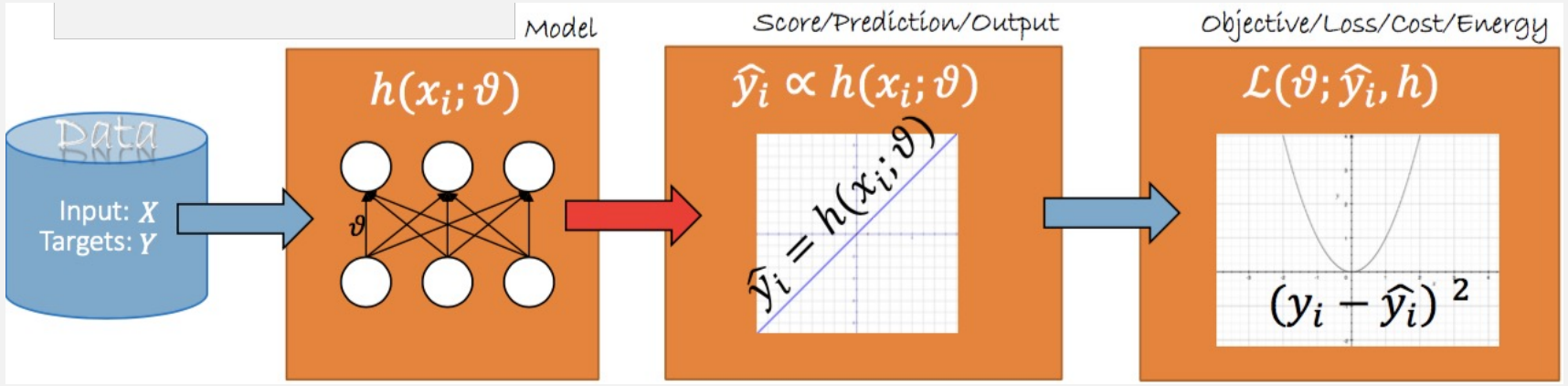
# Forward computations

- Collect annotated data
- Define model and initialize randomly



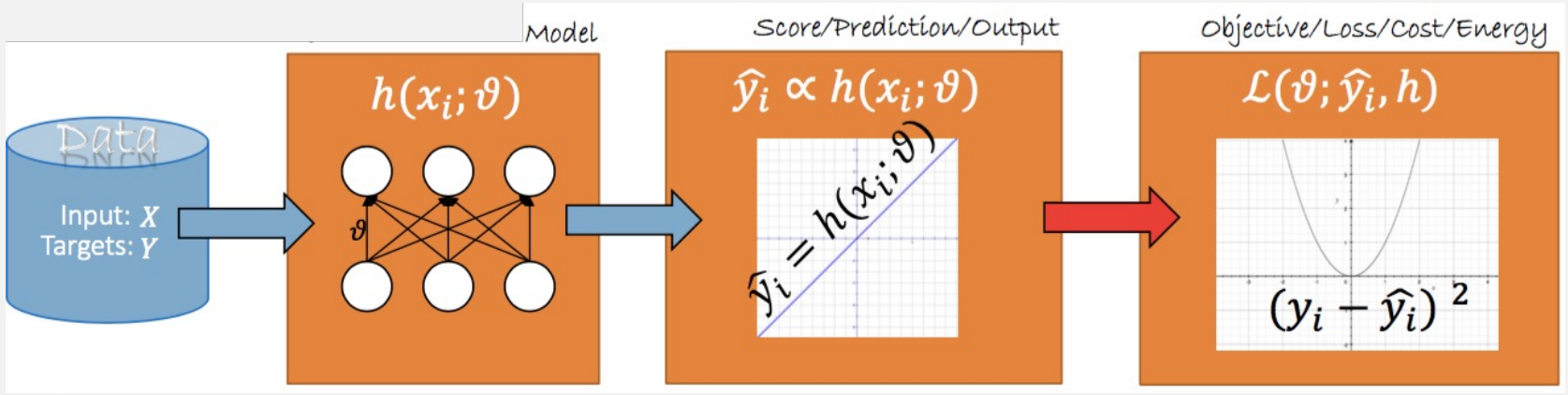
# Forward computations

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



# Forward computations

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



# Recall: Training Objective

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

Find an optimal model parameterized over $\theta$

Ground truth

Prediction

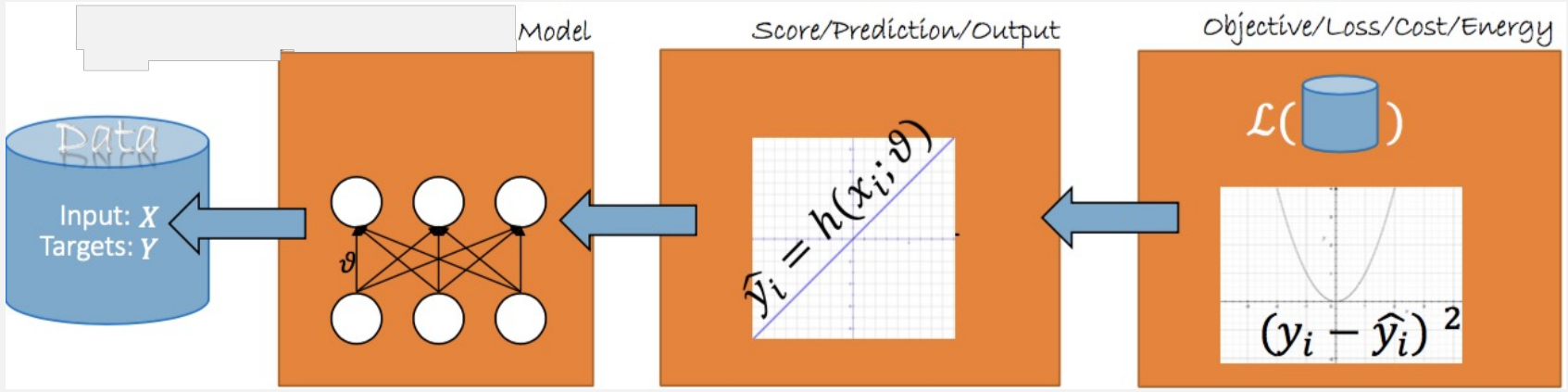
$$\theta^* \leftarrow \arg\min_\theta \sum_{(x,y)\subseteq(X,Y)} \ell(y, a_L(x; \theta_{1,\ldots,L}))$$

accumulated loss

Loss function

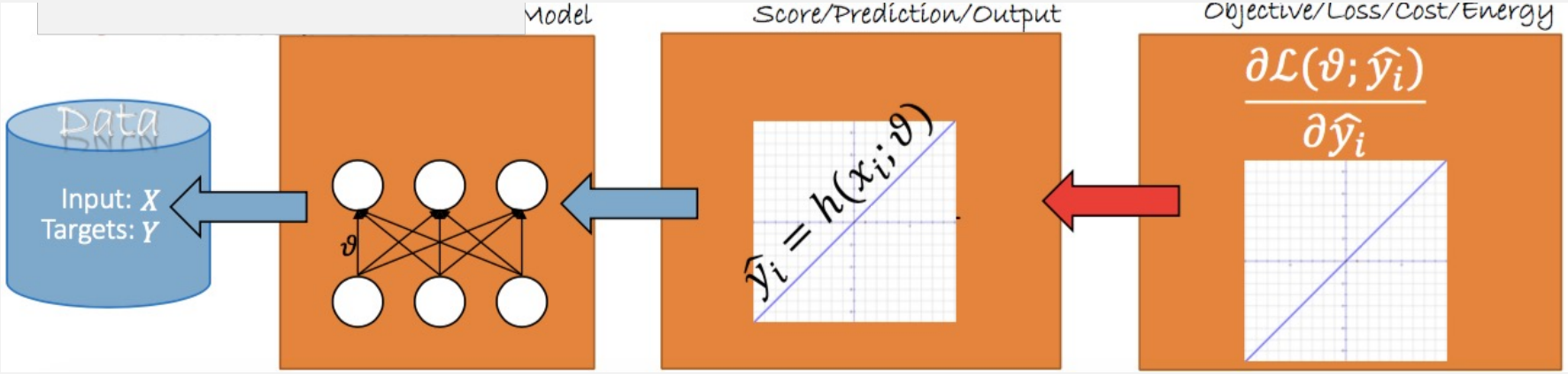
# Backward computations

- Compute loss



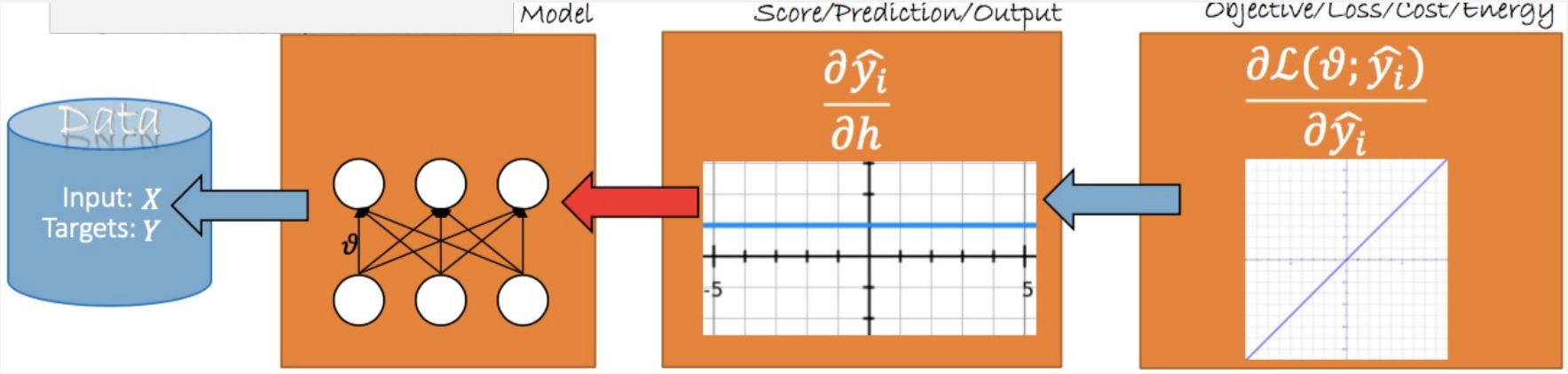
# Backward computations

- Compute loss
- Computer loss gradient over weights



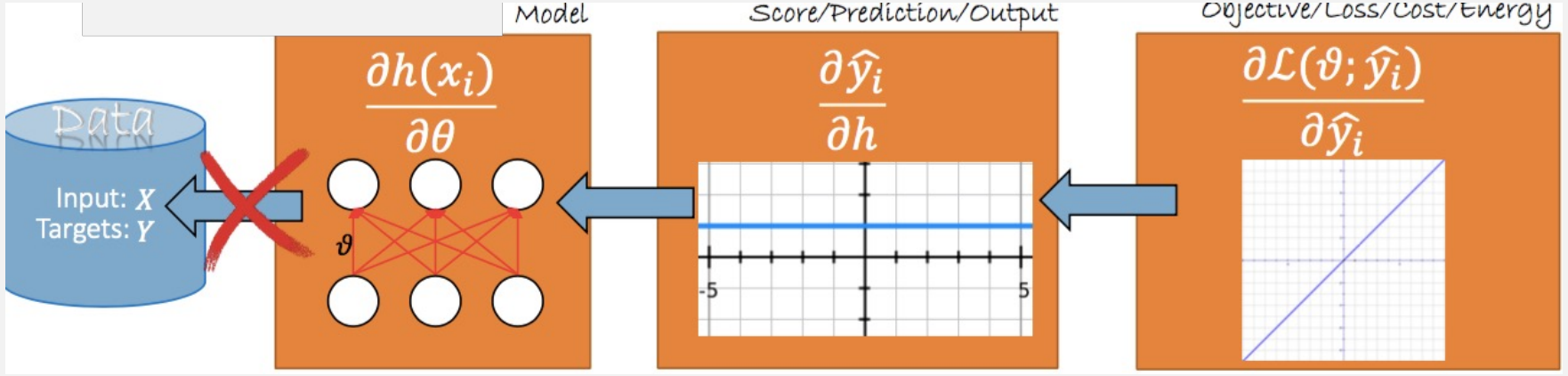
# Backward computations

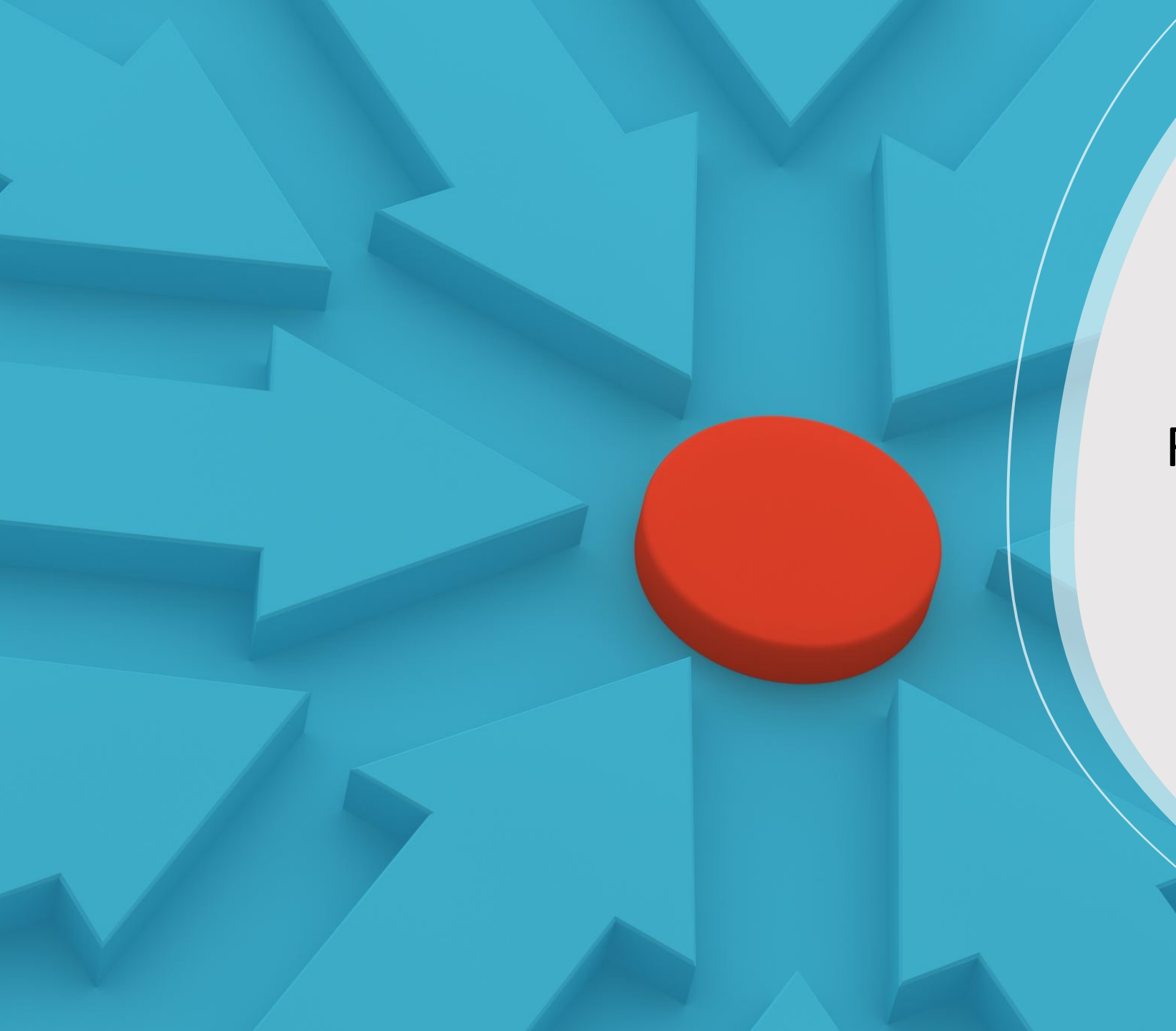
- Compute loss
- Computer loss gradient over weights
- Update weights



# Backward computations

- Compute loss
- Computer loss gradient over weights
- Update weights
- Completion of a round of forward-backward computation

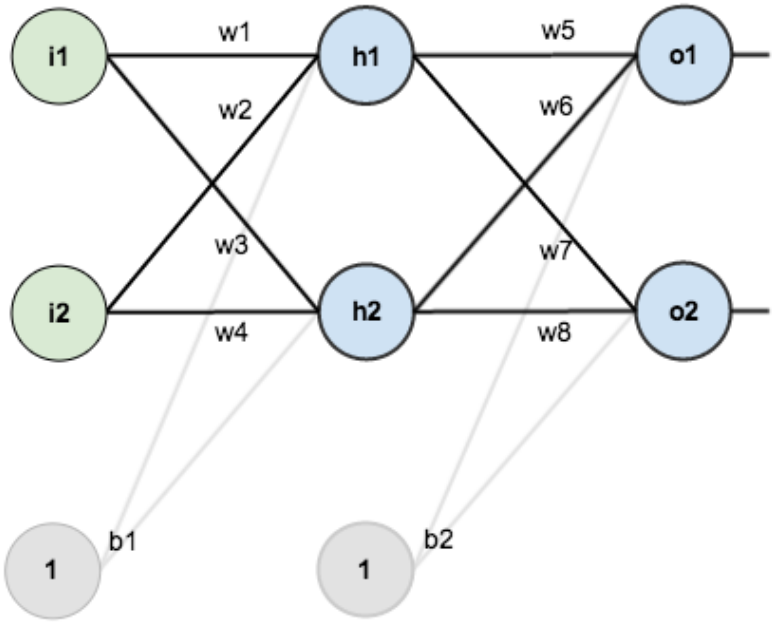




# Forward Computation

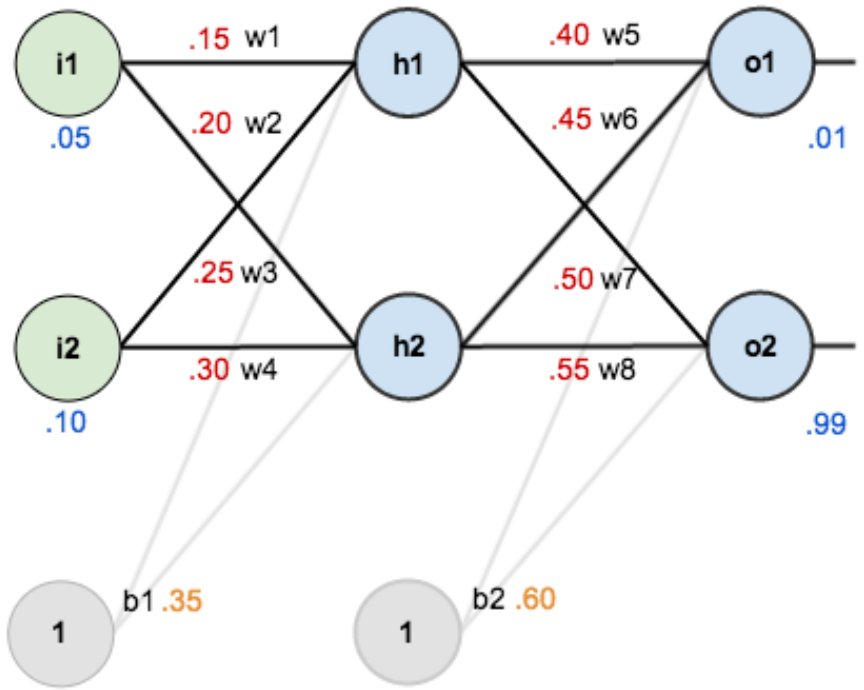
# Example

Network:




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

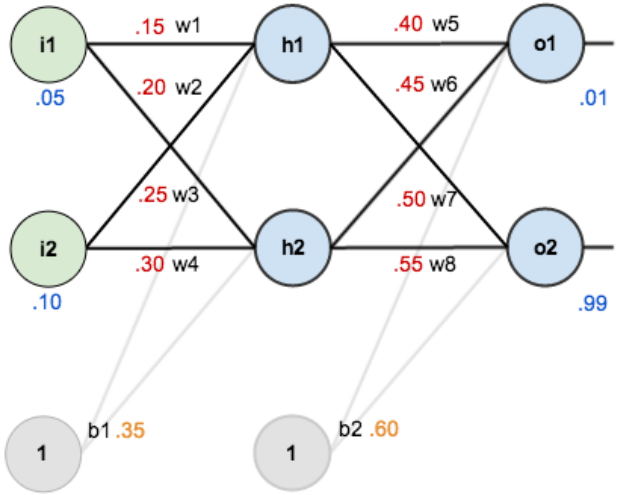
# Example




- 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* be the value before activation function
  - *out* be the value after activation function
  - Activation function is sigmoid function

# 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* be the value before activation function
  - *out* be the value after activation function
  - Activation function is sigmoid function

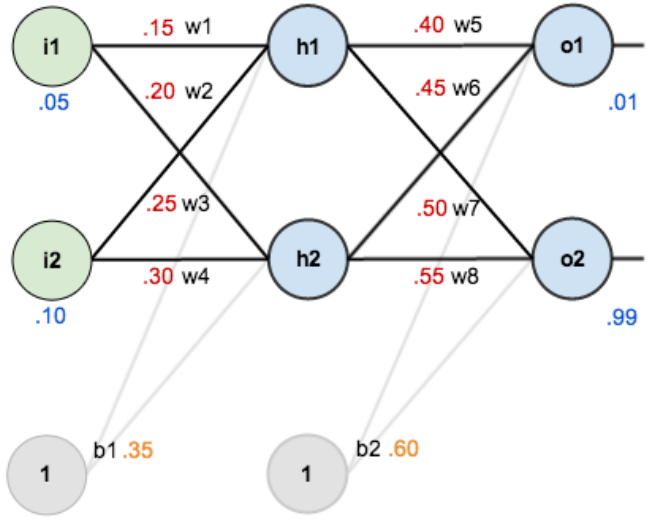



$$
\begin{aligned}
net_{h1} &= w_1 * i_1 + w_2 * i_2 + b_1 * 1 \\
&= 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775 \\
out_{h1} &= \frac{1}{1+e^{-net_{h1}}} \\
&= \frac{1}{1+e^{-net_{0.3775}}} = 0.593269992
\end{aligned}
$$

$$
out_{h2} = 0.596884378
$$

# 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.

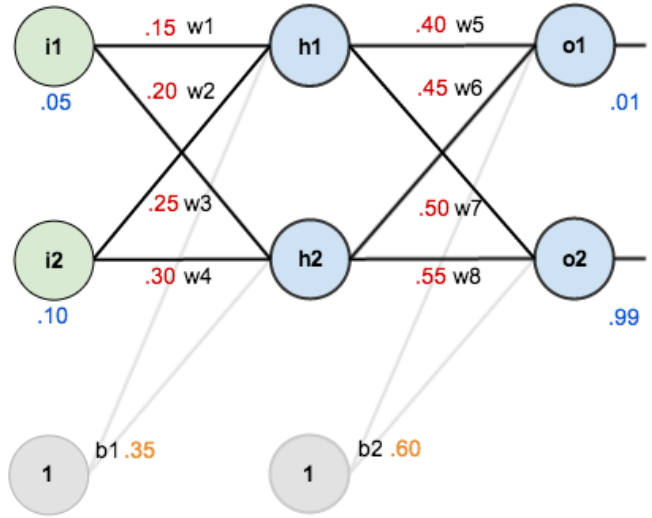



$$
\begin{aligned}
net_{o1} &= w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1 \\
&= 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967 \\
out_{o1} &= \frac{1}{1+e^{-net_{o1}}} \\
&= \frac{1}{1+e^{-1.105905967}} = 0.75136507
\end{aligned}
$$

$$
out_{o2} = 0.772928465
$$

# 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.




$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.751366507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

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

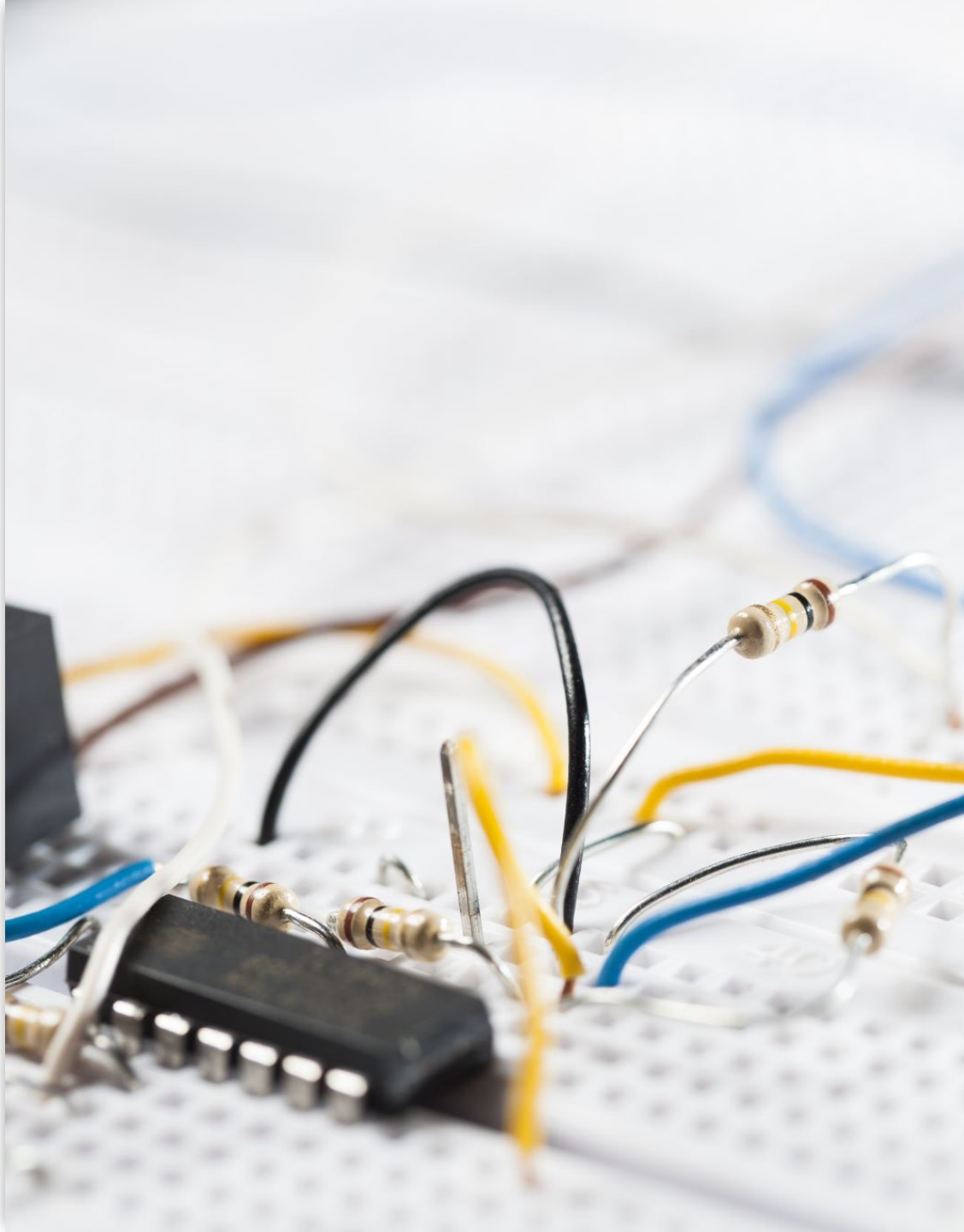
Backward Propogation

# Recall: Minimizing with multiple dimensional inputs

- We often minimize functions with multiple-dimensional inputs

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

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



# Functions with multiple inputs

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 $\theta$

- Partial derivatives

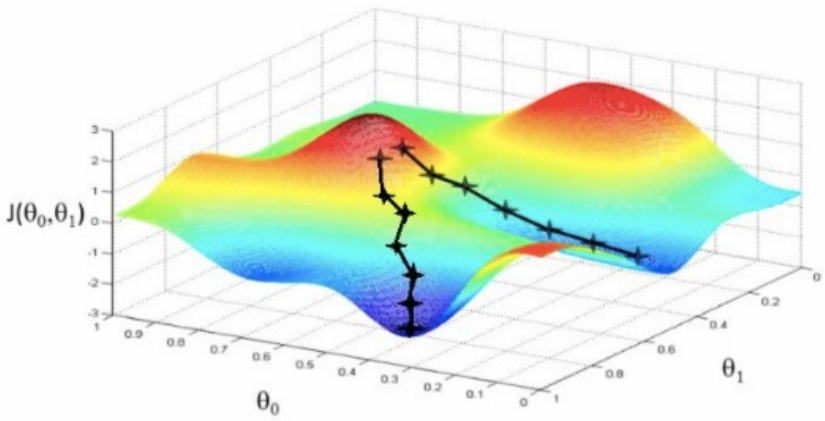
$$\frac{\partial}{\partial x_i} f(x)$$

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

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

# Optimization through Gradient Descent




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

- The most important component in this formulation is the gradient

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

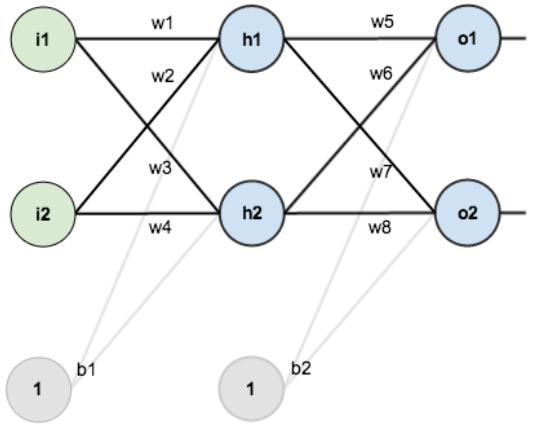
- Backpropagation to the rescue
  - The backward computations of network return the gradients
  - How to make the backward computations

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




How to calculate partial derivatives?

# Chain rule

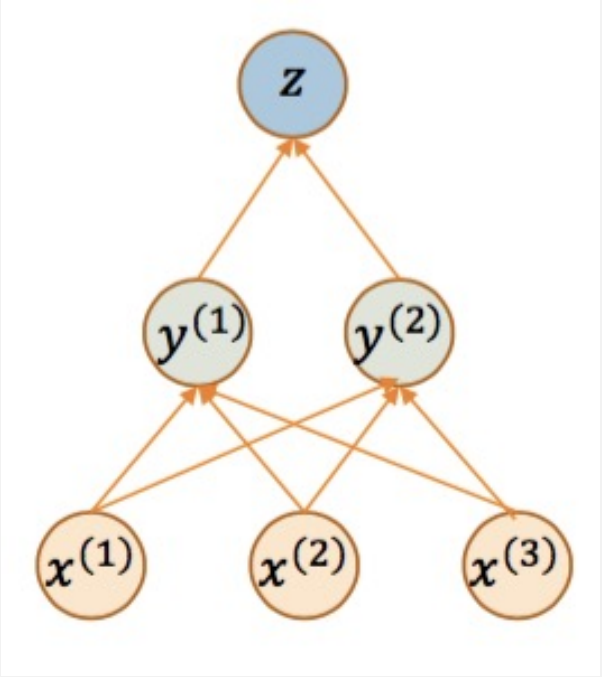
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars $x$, $y$, $z$

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

- 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

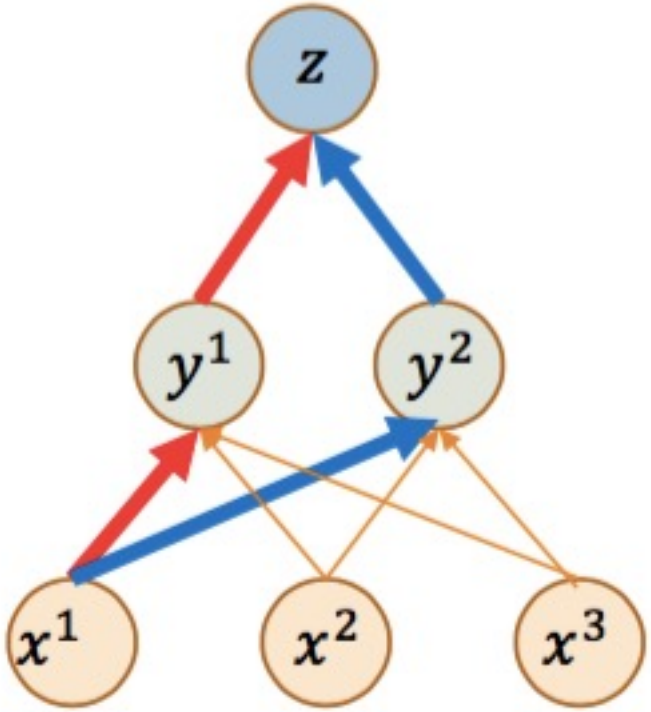
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars $x, y, z$

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

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

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

  - i.e., gradients from all possible paths




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

# Chain rule

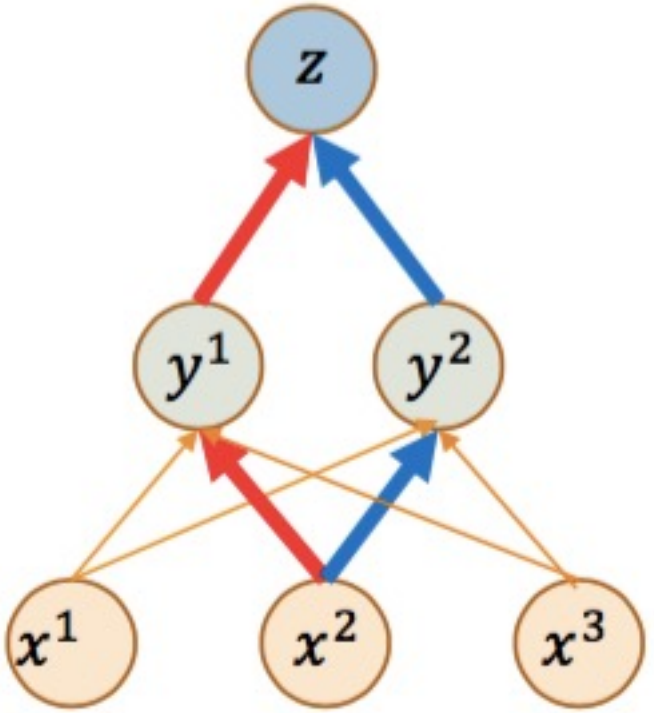
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars $x, y, z$

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

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

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

  - i.e., gradients from all possible paths




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

# Chain rule

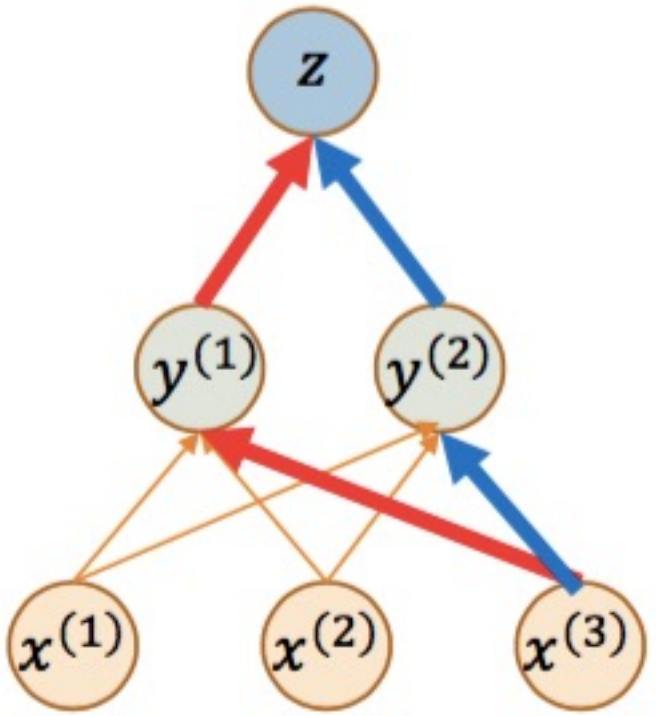
- Assume a nested function, $z = f(y)$ and y= g(x)
- Chain Rule for scalars $x, y, z$

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

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

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

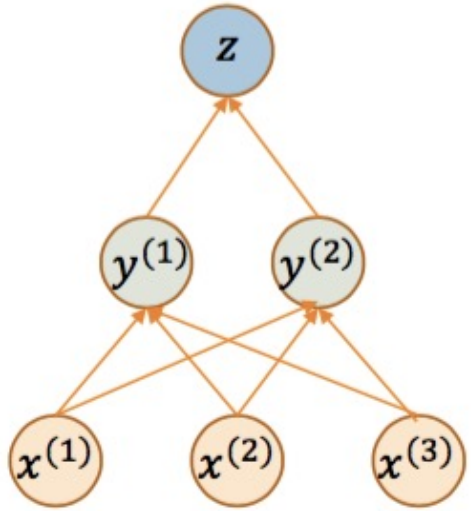
  - i.e., gradients from all possible paths




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

# Chain rule

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




- or in vector notation

$$\frac{dz}{dx} = \left(\frac{dy}{dx}\right)^T \cdot \frac{dz}{dy}$$

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

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

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

- $\frac{dy}{dx}$ is the Jacobian

# The Jacobian

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

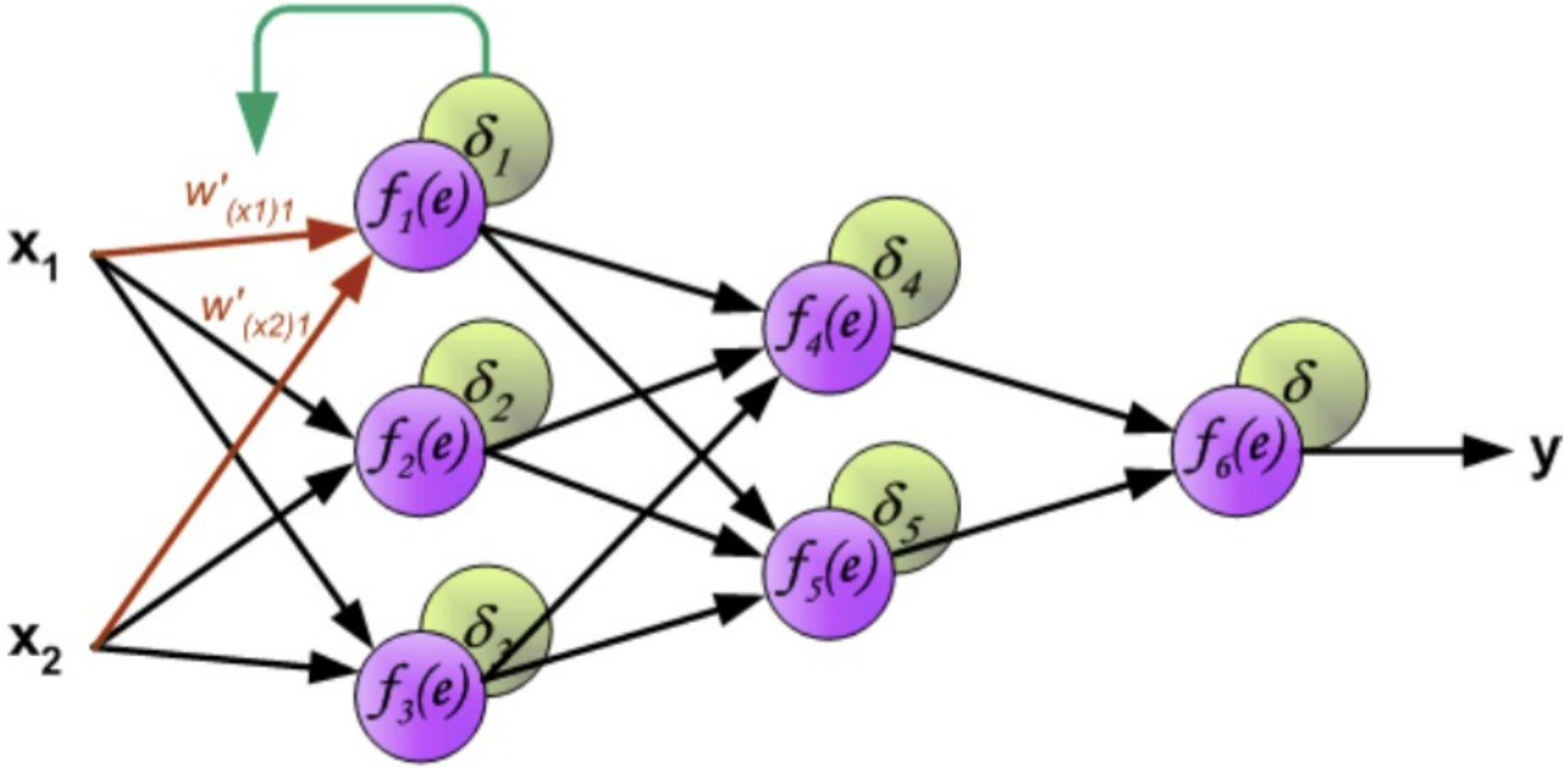
$$J(y(x)) = \frac{dy}{dx} = \begin{bmatrix} \frac{\partial y^{(1)}}{\partial x^{(1)}} & \frac{\partial y^{(1)}}{\partial x^{(2)}} & \frac{\partial y^{(1)}}{\partial x^{(3)}} \\ \frac{\partial y^{(2)}}{\partial x^{(1)}} & \frac{\partial y^{(2)}}{\partial x^{(2)}} & \frac{\partial y^{(2)}}{\partial x^{(3)}}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

# Chain rule in practice

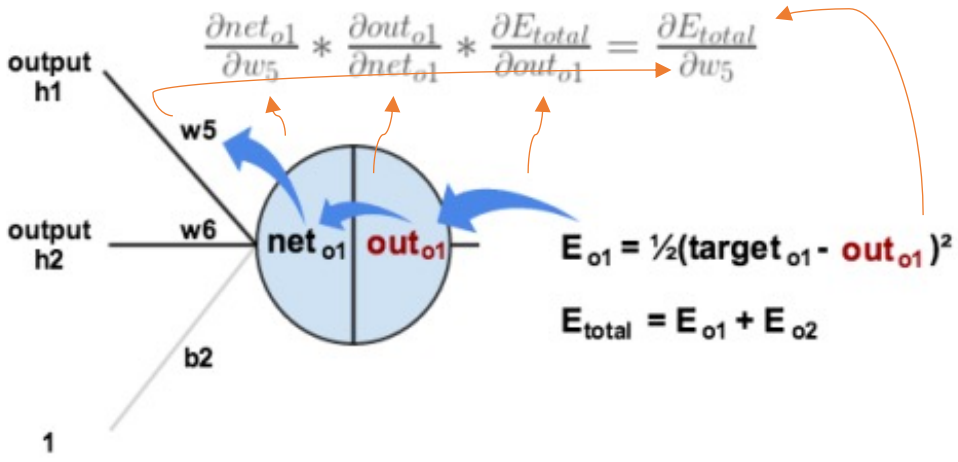
- f(y) =sin($y$) ,$y$=$g(x)$ =0.5$x^2$

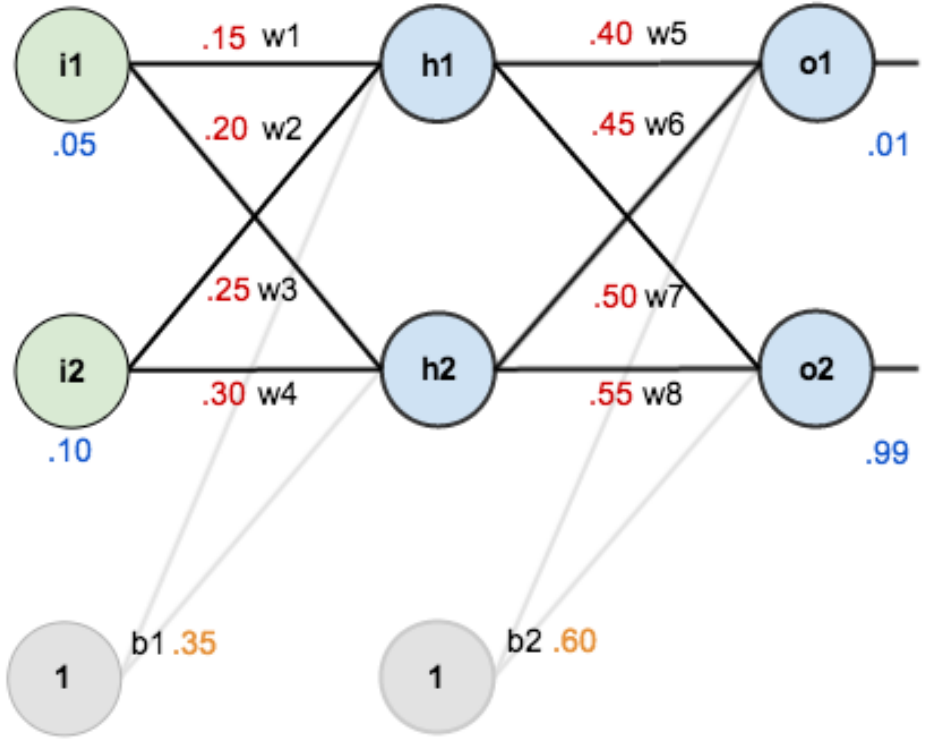
$$\frac{df}{dx} = \frac{d\,[\sin(y)]}{dg}\frac{d[0.5x^2]}{dx}$$
$$= \cos(0.5x^2) \cdot x$$

# Backpropagation



# 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
net o1
out o1
b2
1
E o1 = ½(target o1 - out o1)²
E total = E o1 + E o2


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

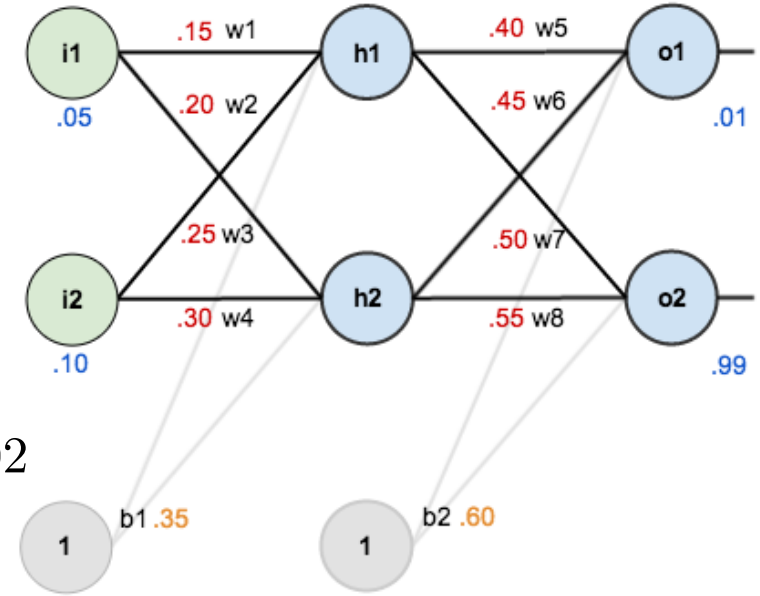
# Example

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

$$\begin{aligned} \frac{\partial E_{total}}{\partial out_{o1}} &= 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0 \\ &= -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507 \end{aligned}$$

$$\begin{aligned} out_{o1} &= \frac{1}{1+e^{-net_{o1}}} \\ \frac{\partial out_{o1}}{\partial net_{o1}} &= out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602 \end{aligned}$$

$$\begin{aligned} net_{o1} &= w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1 \\ \frac{\partial net_{o1}}{\partial w_5} &= 1 * out_{h1} * w_5^{1-1} + 0 + 0 = out_{h1} = 0.593269992 \end{aligned}$$



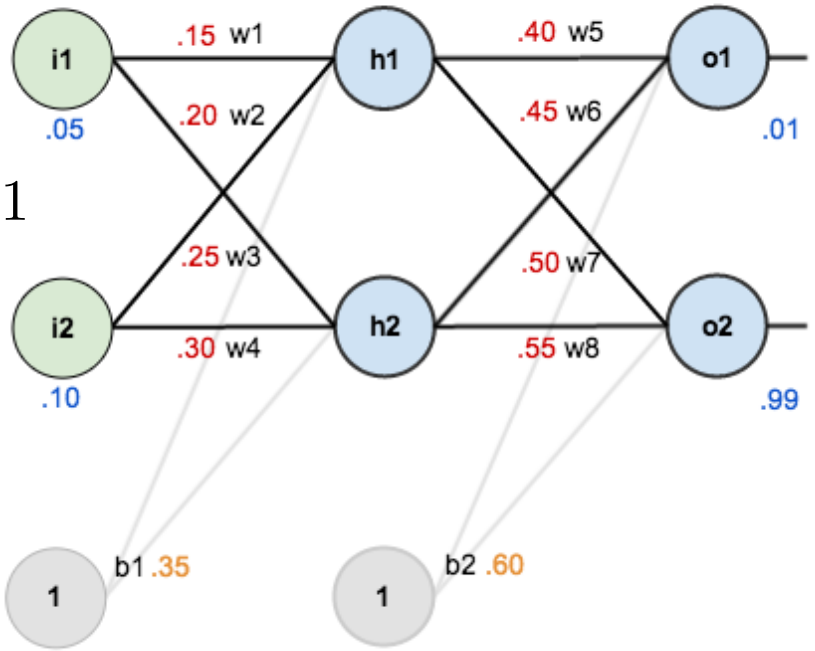
# Example

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

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

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

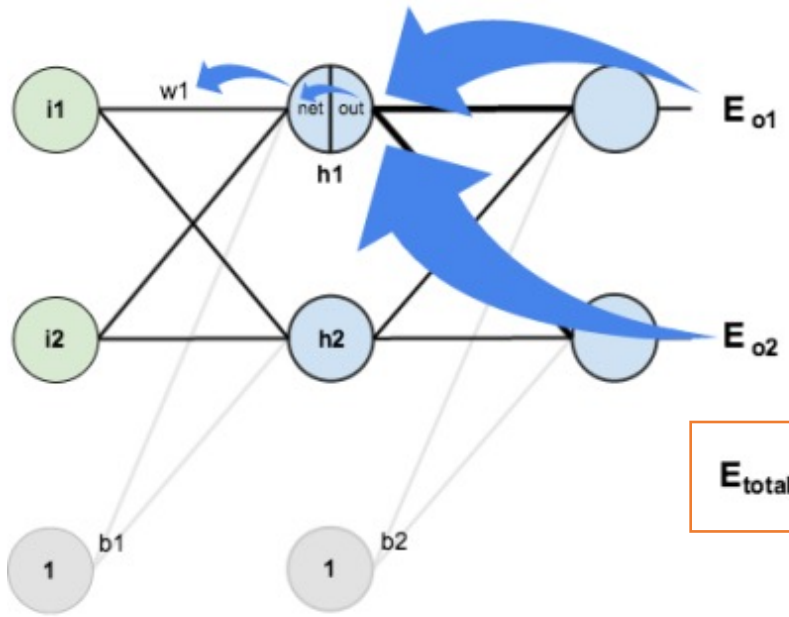
$$w_6^+ = 0.408666186$$
$$w_7^+ = 0.511301270$$
$$w_8^+ = 0.561370121$$



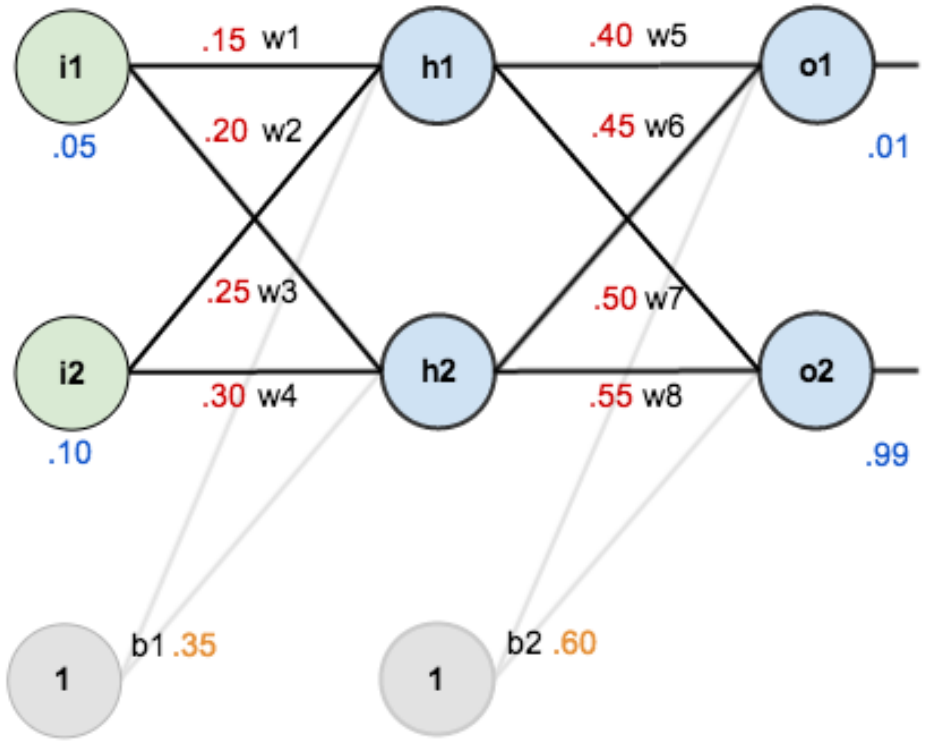
# Example

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

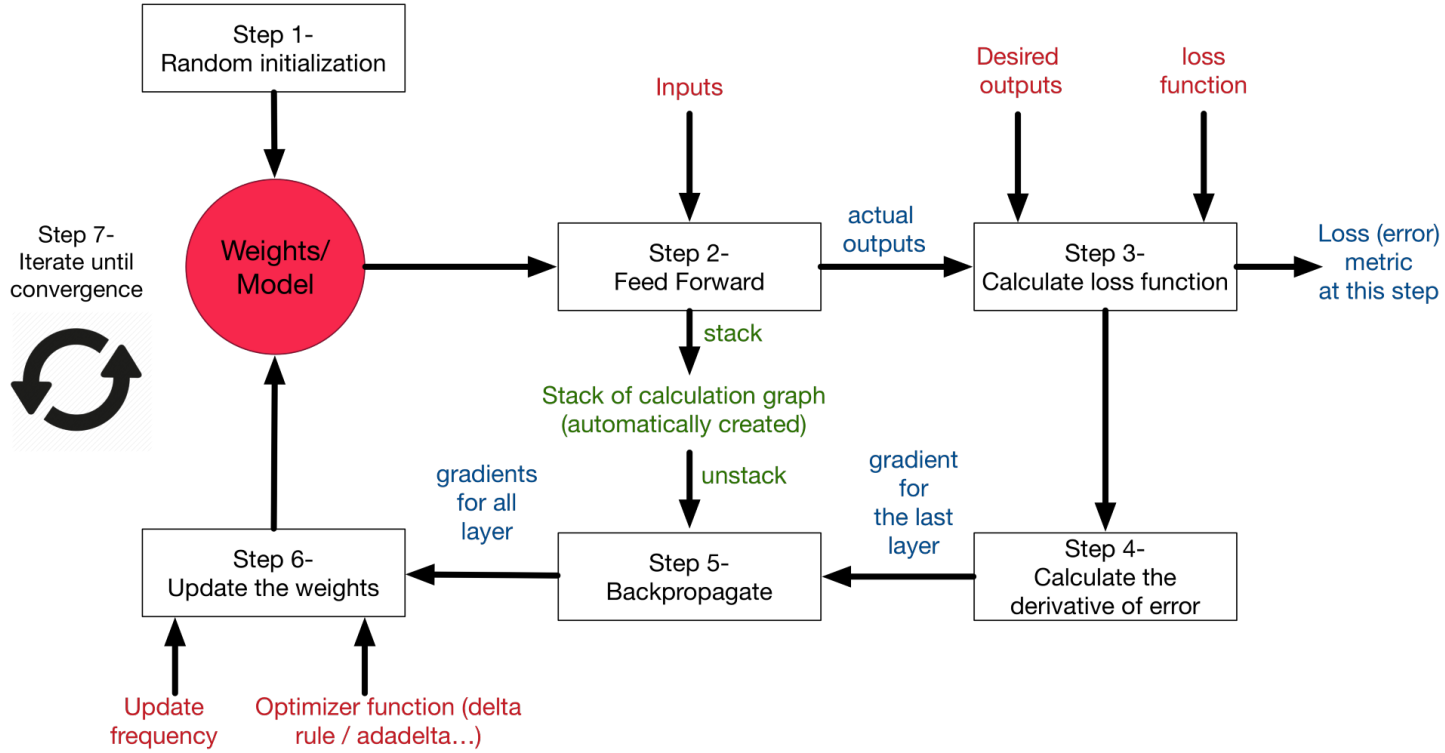
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

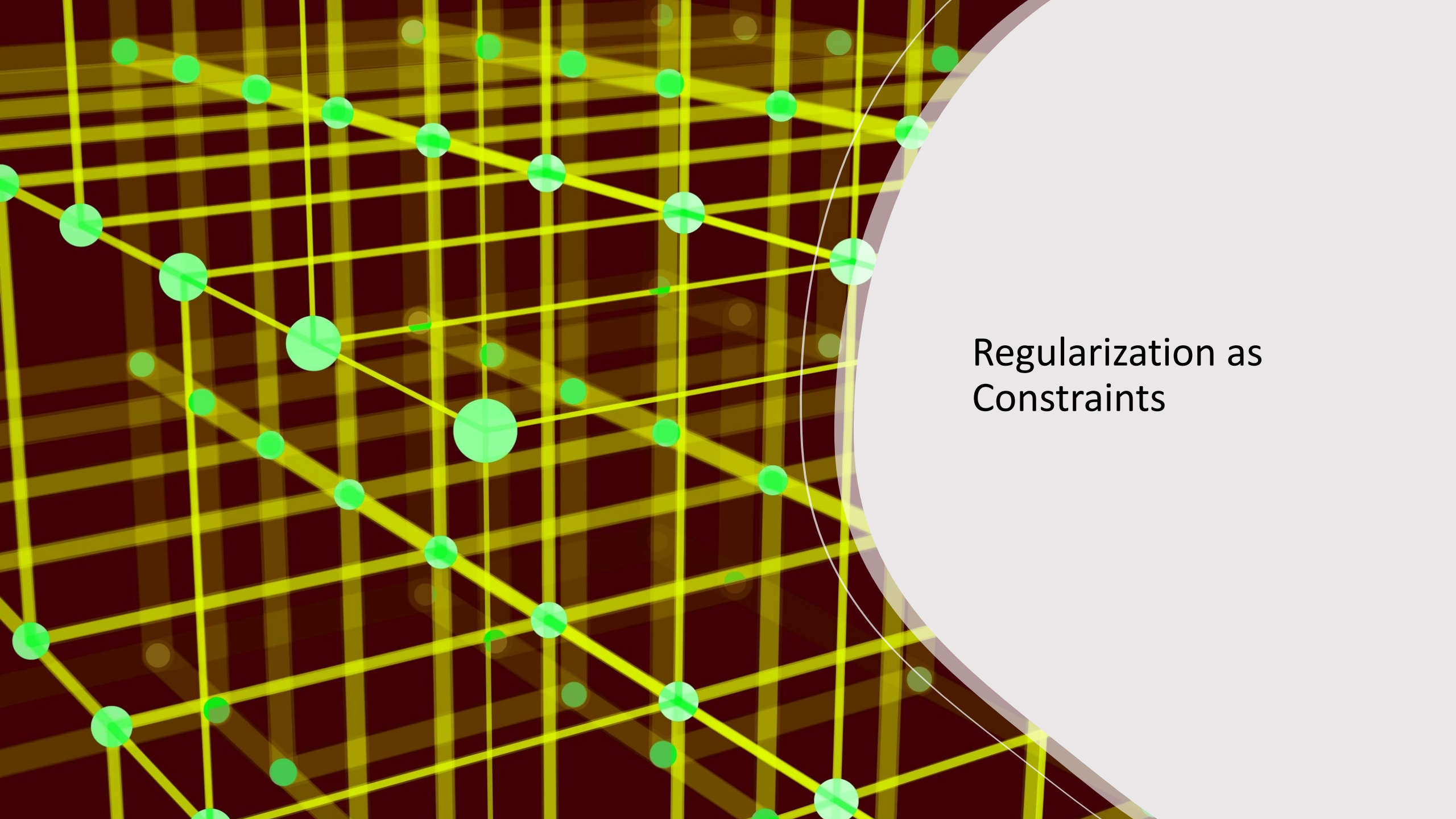



$$E_{total} = E_{o1} + E_{o2}$$



# 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

- The smaller the data set, the larger the difference between the two
- The larger the hypothesis class, the 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 $\Omega$ 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

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

Adding constraints

- Suppose $\theta^*$ is the optimal for hard-constraint optimization

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

- Suppose $\lambda^*$ is the corresponding optimal for max

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