Artificial Neural Networks 3:

Deep Learning

Course: Computational Intelligence (TI2736-A)

Lecturer: Thomas Moerland

TU Delft

Recap: Machine Learning

= Function approximation

Today: focus on *parametric*, supervised learning

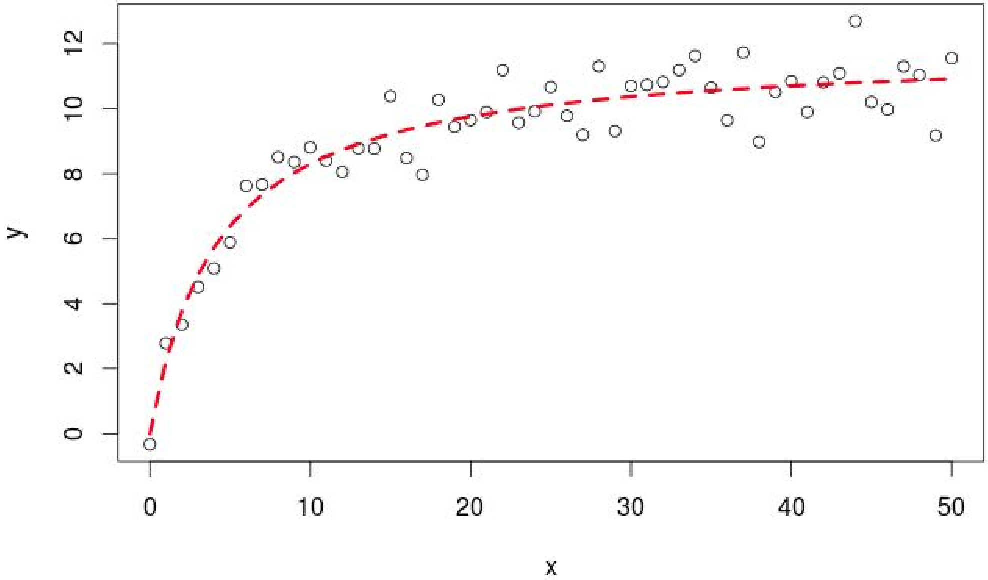
$$\mathbf{y = f(x;}\theta\mathbf{)}$$

Recap: Machine Learning

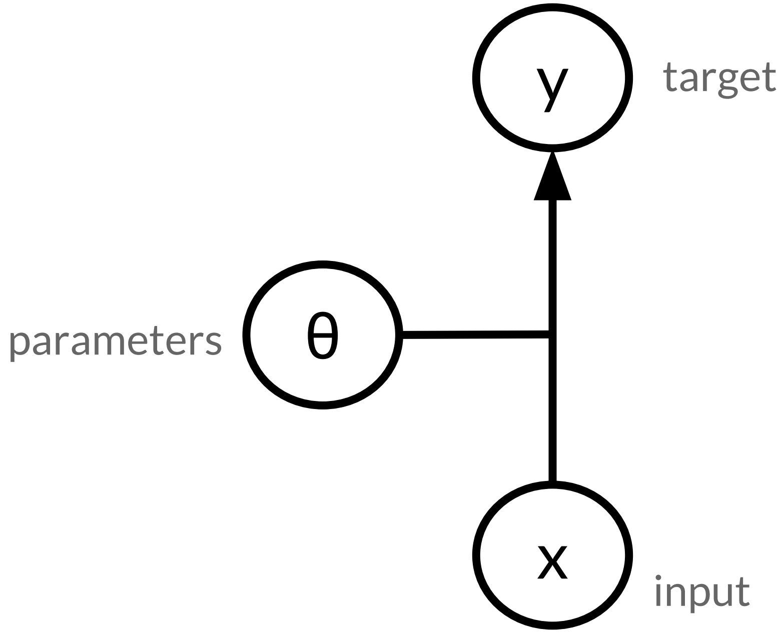
= Function approximation

Today: focus on *parametric*, supervised learning

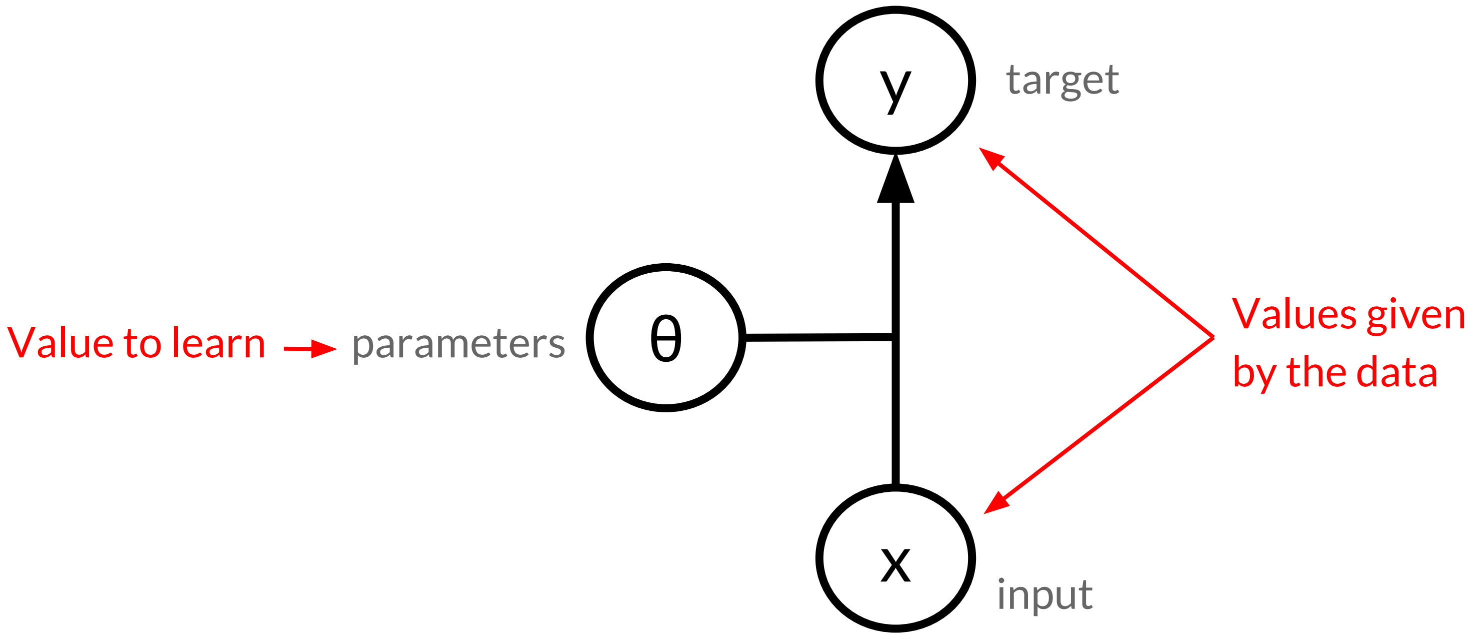
y = f(x;θ)



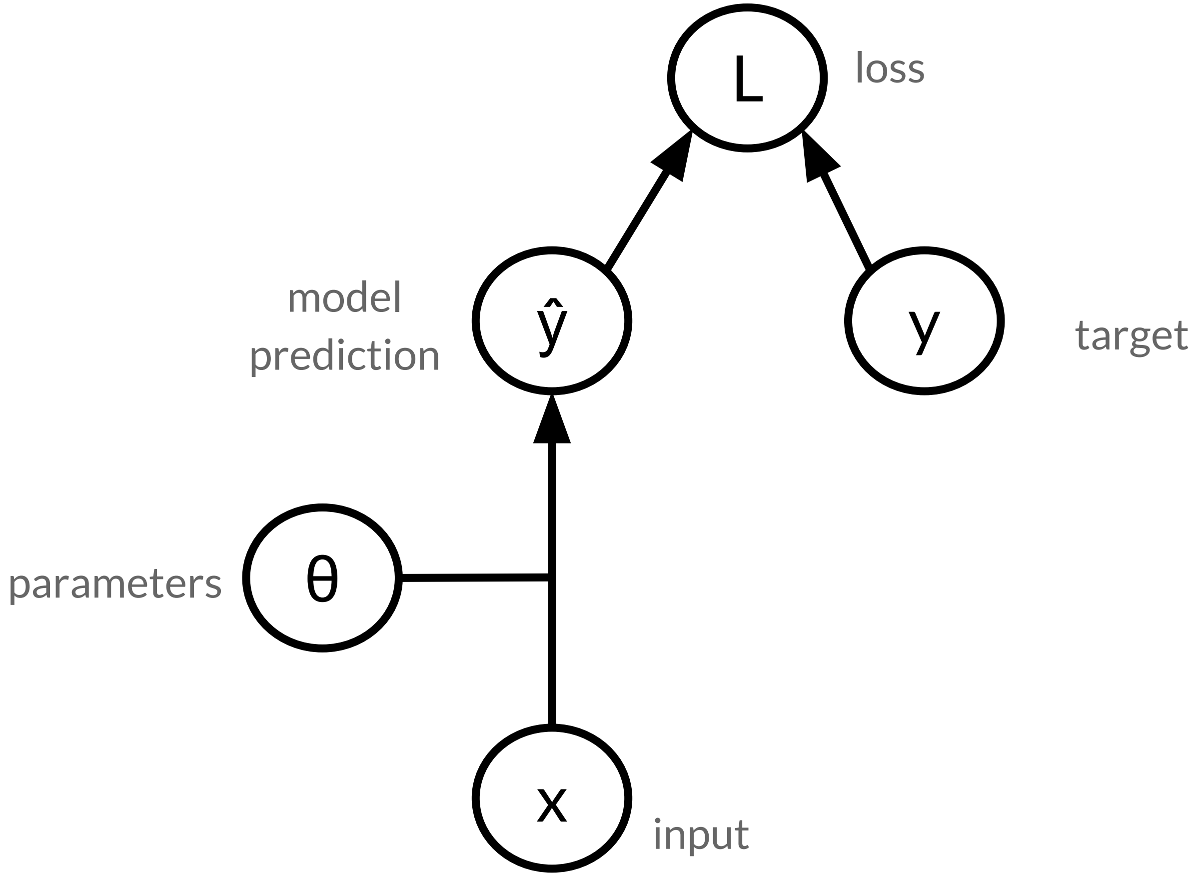
Recap: Machine Learning



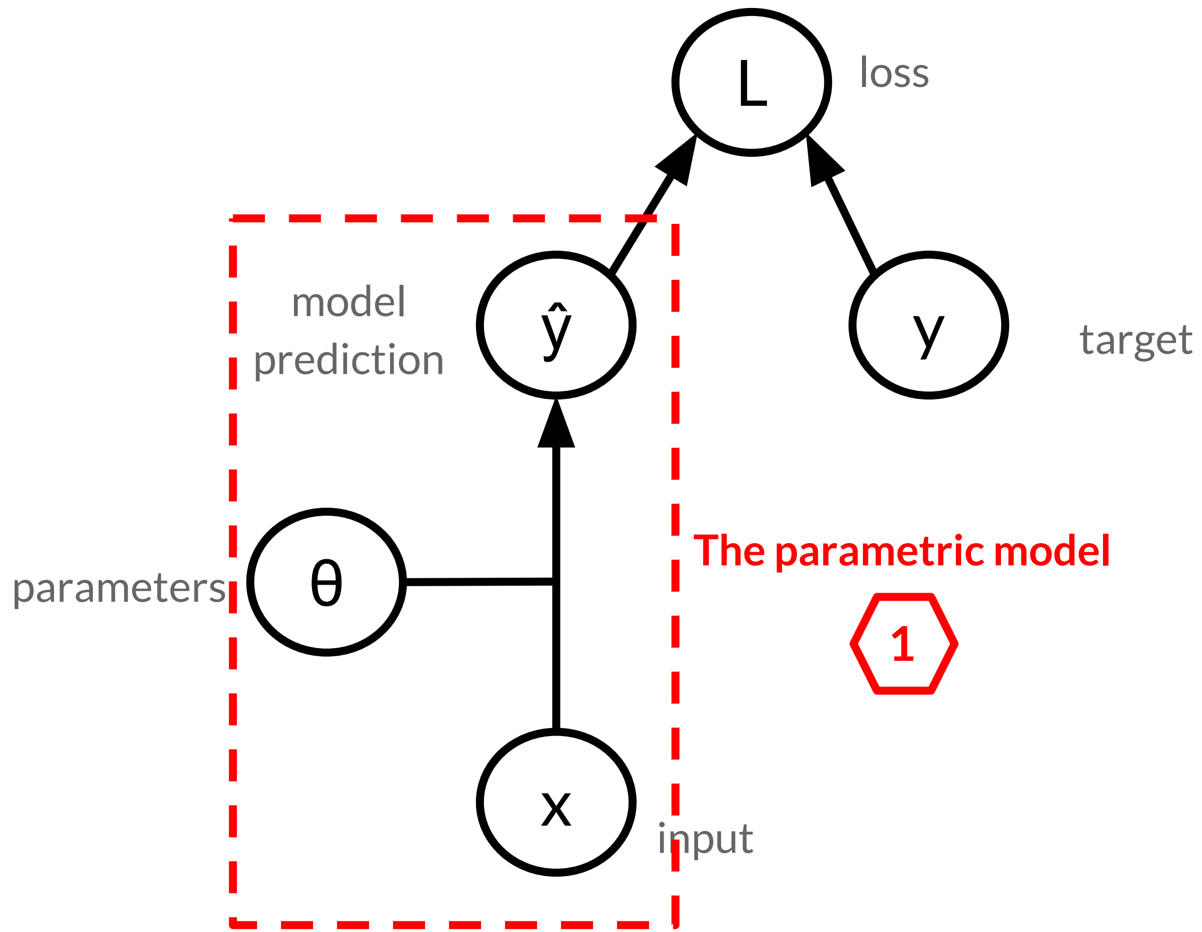
Recap: Machine Learning



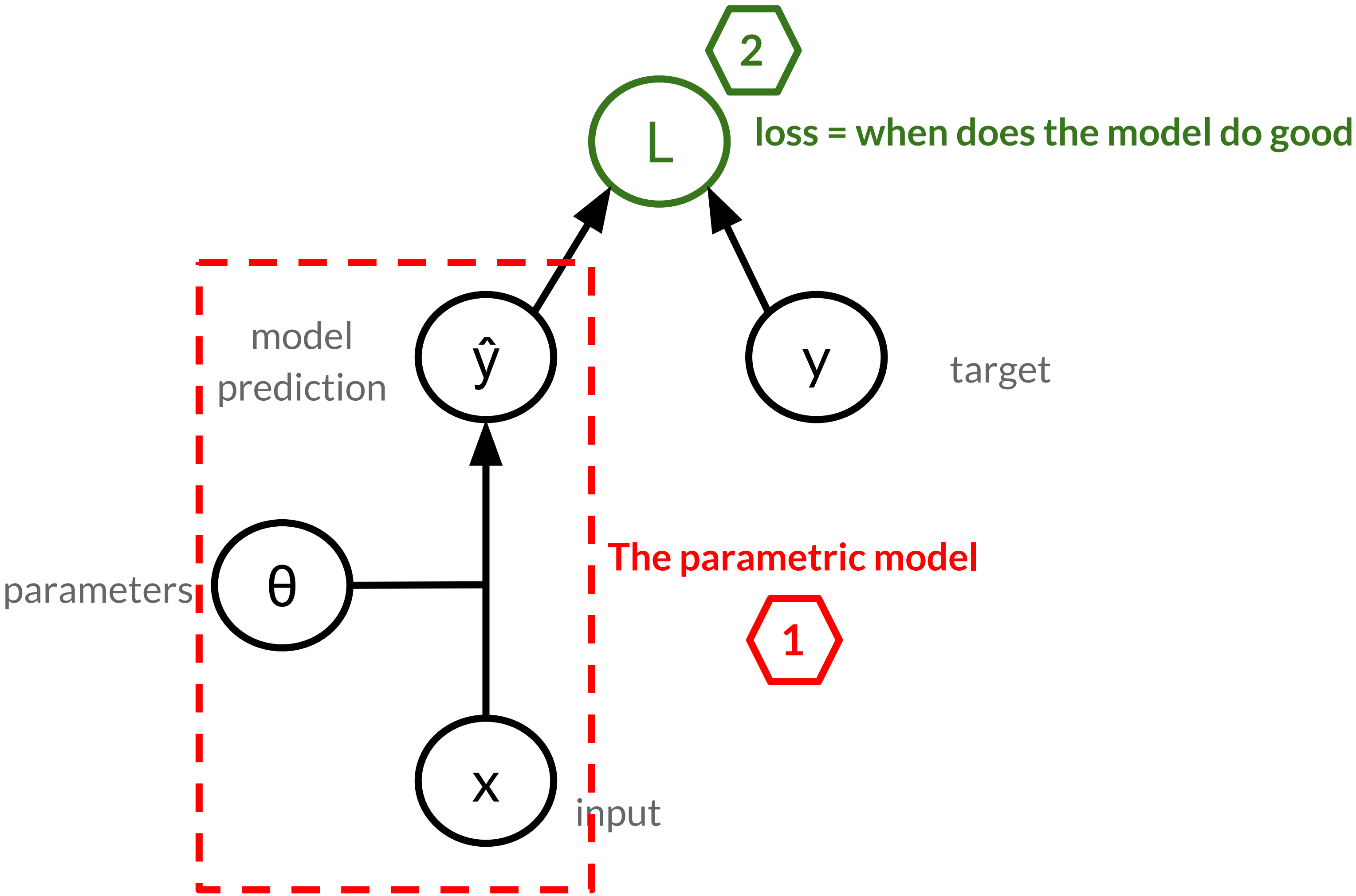
Recap: Machine Learning



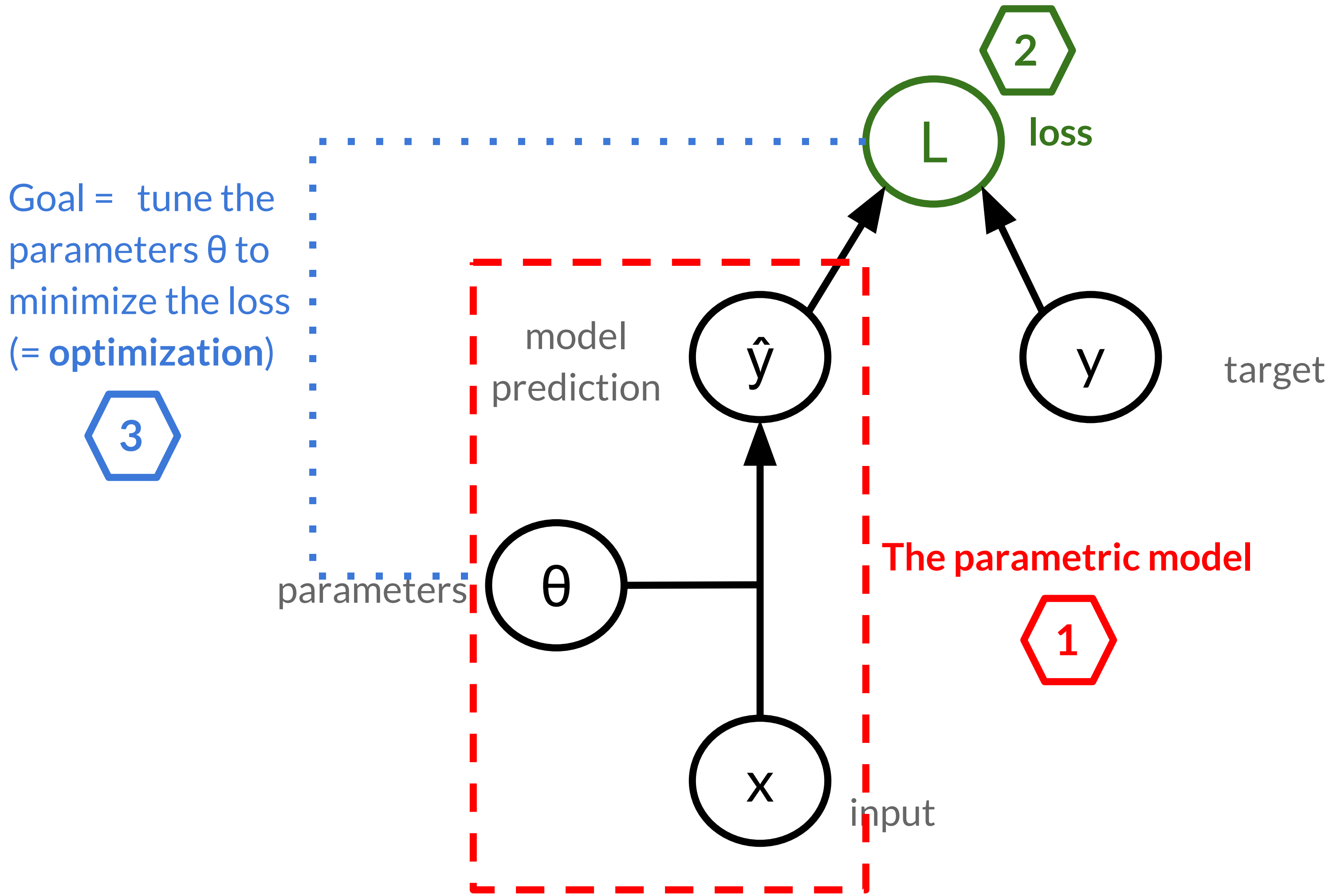
Recap: Machine Learning



Recap: Machine Learning



Recap: Machine Learning



Content for today

1. **The Feedforward Network**
   a. Artificial Neural Network (ANN): A Parametric Model

   b. Loss Functions

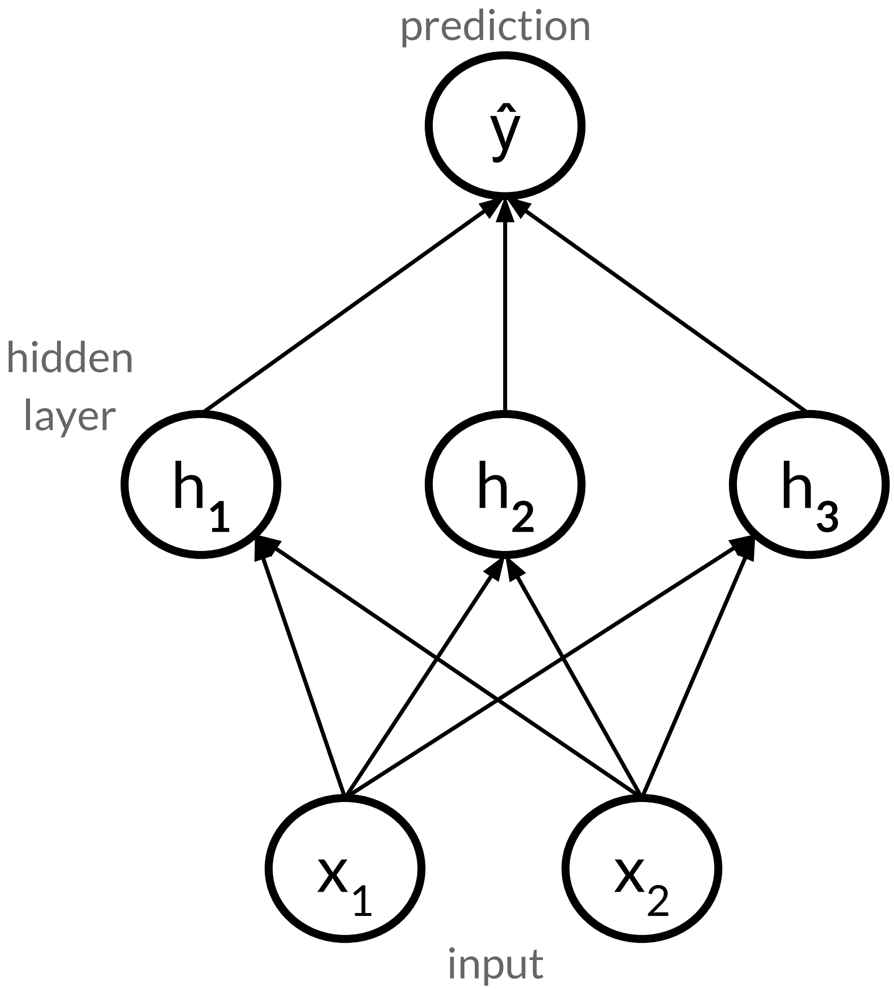
   c. Numerical Optimization

2. **Advanced Neural Network Architectures**
   a. Convolutional Neural Network (CNN)
   b. Recurrent Neural Network (RNN)

3. **Deep learning**

1. The Feedforward Network

ANN: A Parametric Model



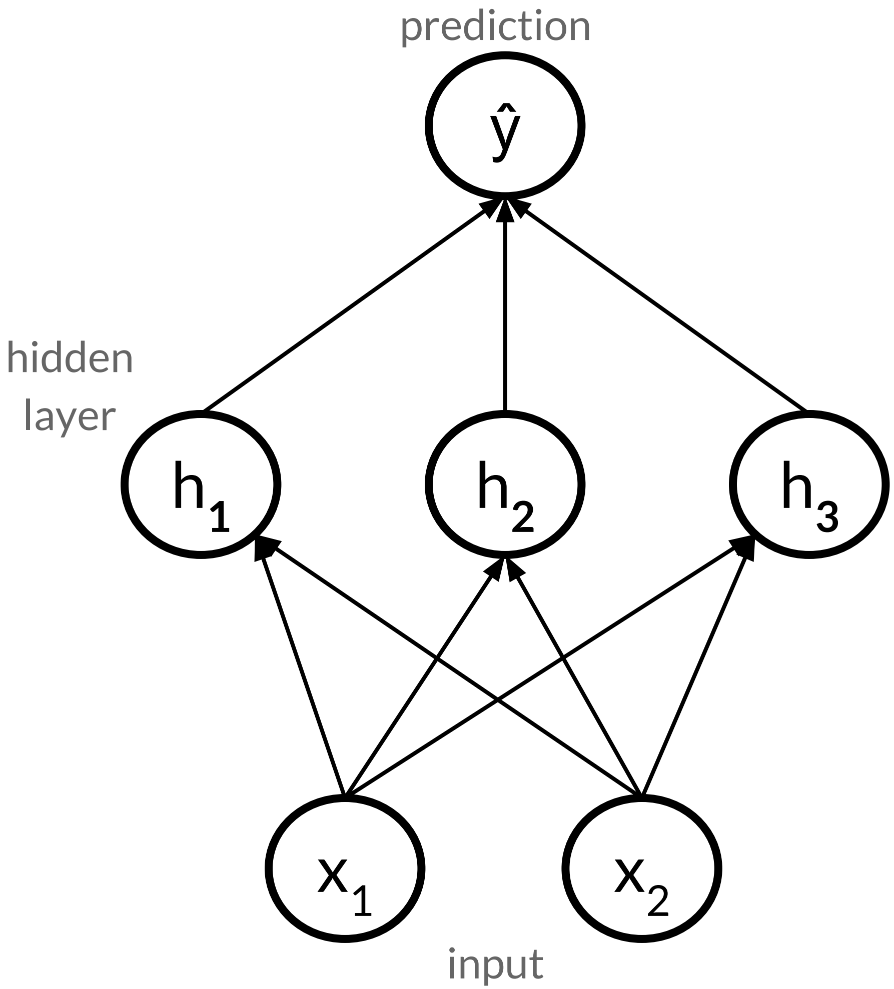
Artificial Neural Network (ANN)

=

stacked sequence of non-linear regressions

("*fully connected layers*")

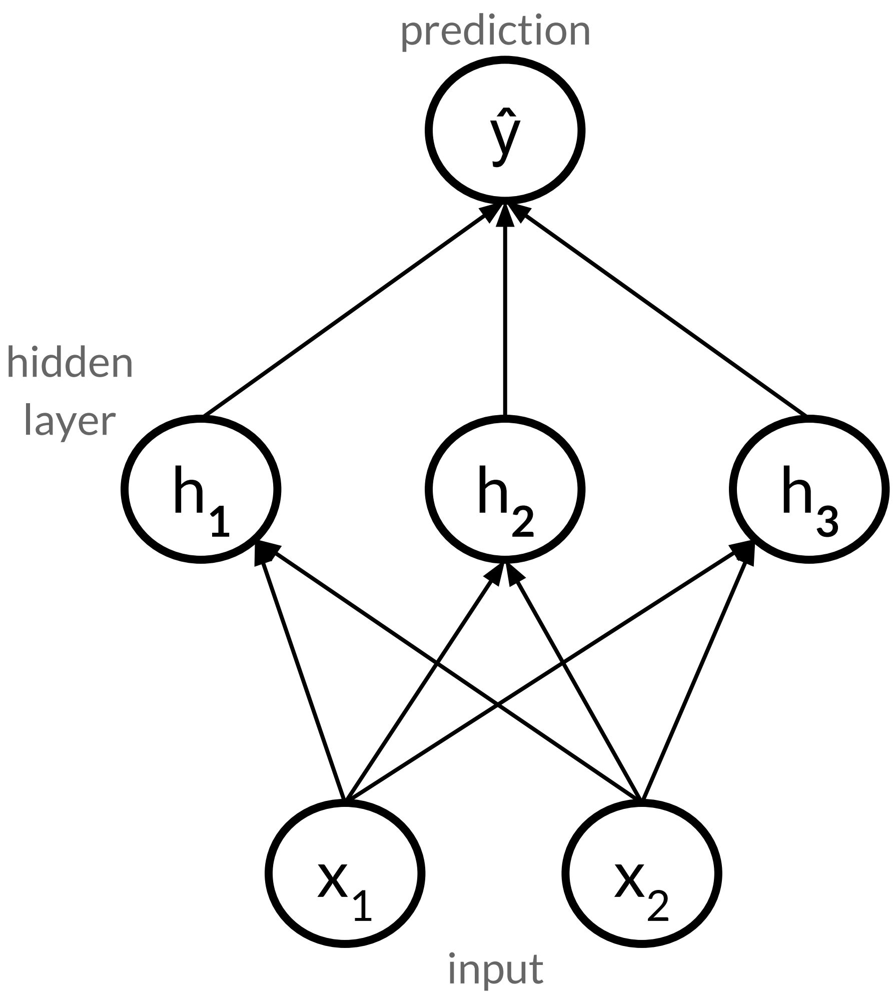
Artificial Neural Network structure



per layer:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}|\theta) = g^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + b^{(1)})$$

Artificial Neural Network structure

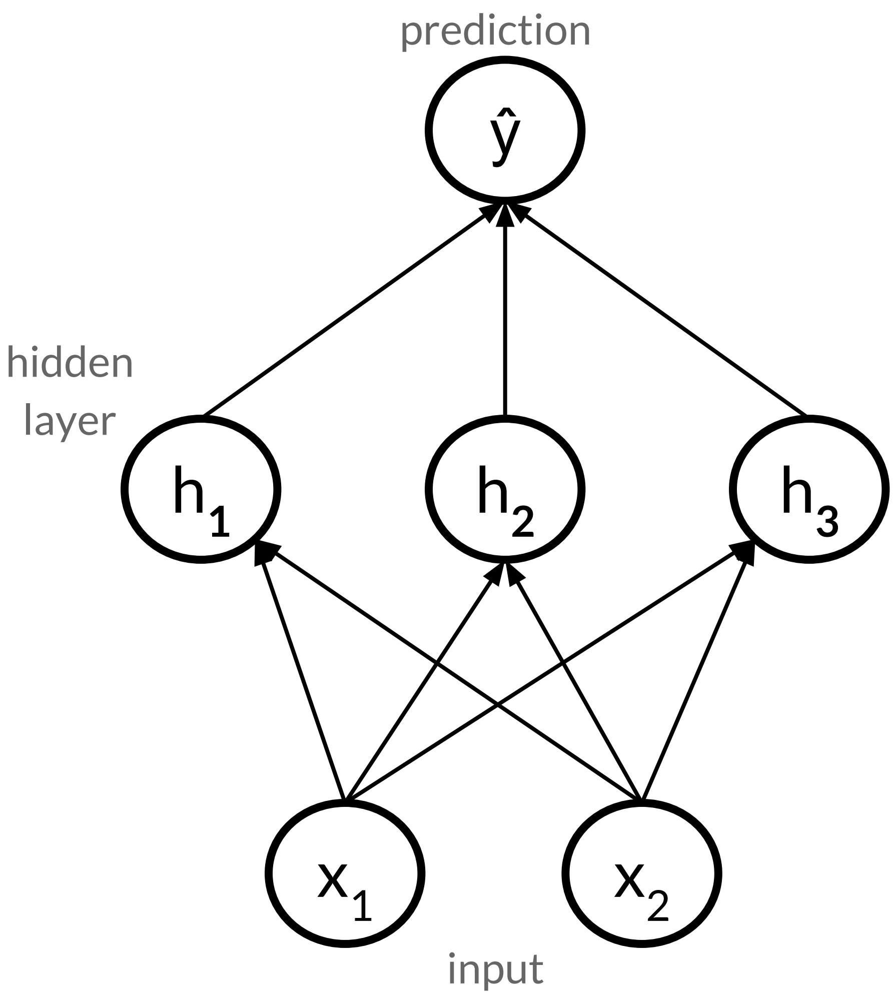


per layer:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}|\theta) = g^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + b^{(1)})$$

- layer number
- input (vector)
- output (vector)
- non-linear function (element-wise to vector)
- weight (matrix)
- bias (vector)

Artificial Neural Network structure



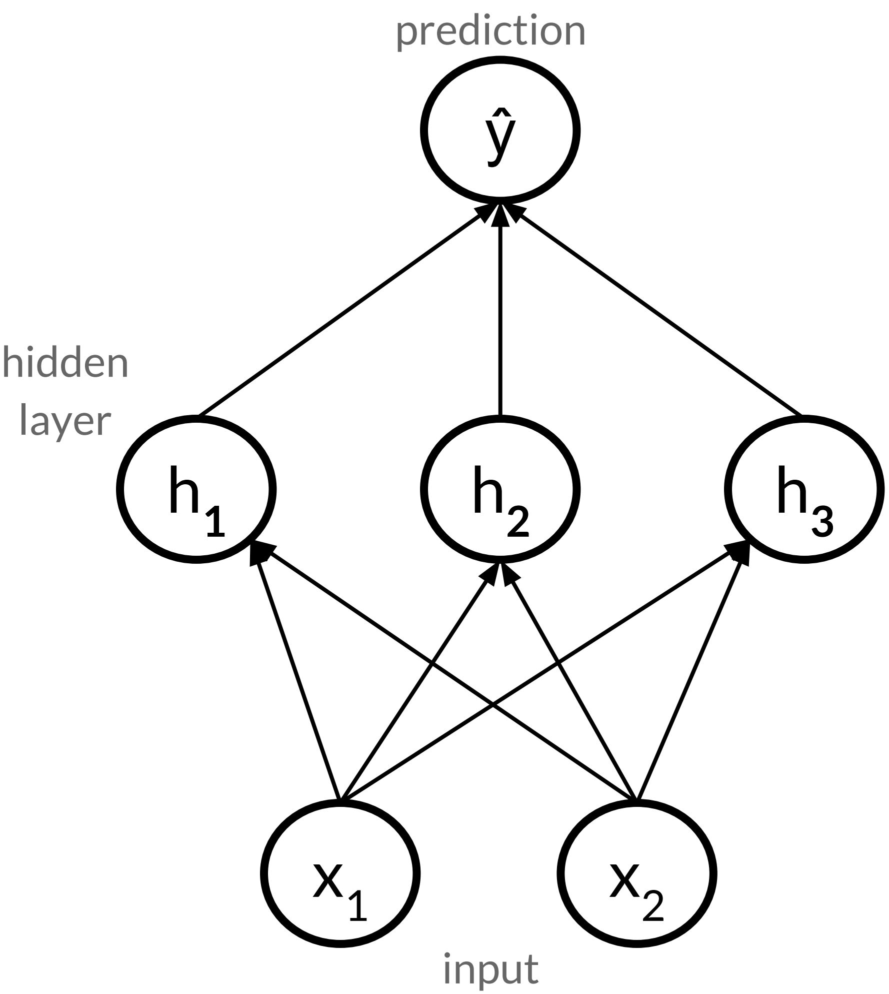
<u>per layer:</u>

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}|\theta) = g^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + b^{(1)})$$

- layer number
- input (vector)
- output (vector)
- non-linear function (element-wise)
- weight (matrix)
- bias (vector)

Q: How many parameters does this network have?

Artificial Neural Network structure



per layer:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}|\theta) = g^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + b^{(1)})$$

- layer number
- input (vector)
- output (vector)
- non-linear function (element-wise)
- weight (matrix)
- bias (vector)

Q: How many parameters does this network have?

A: 13

First layer: 6 weights + 3 biases

Second layer: 3 weights + 1 bias

Activation Functions

Q: *Why not stack multiple linear layers?*

A: Composition of linear transformations is still linear.

Activation Functions

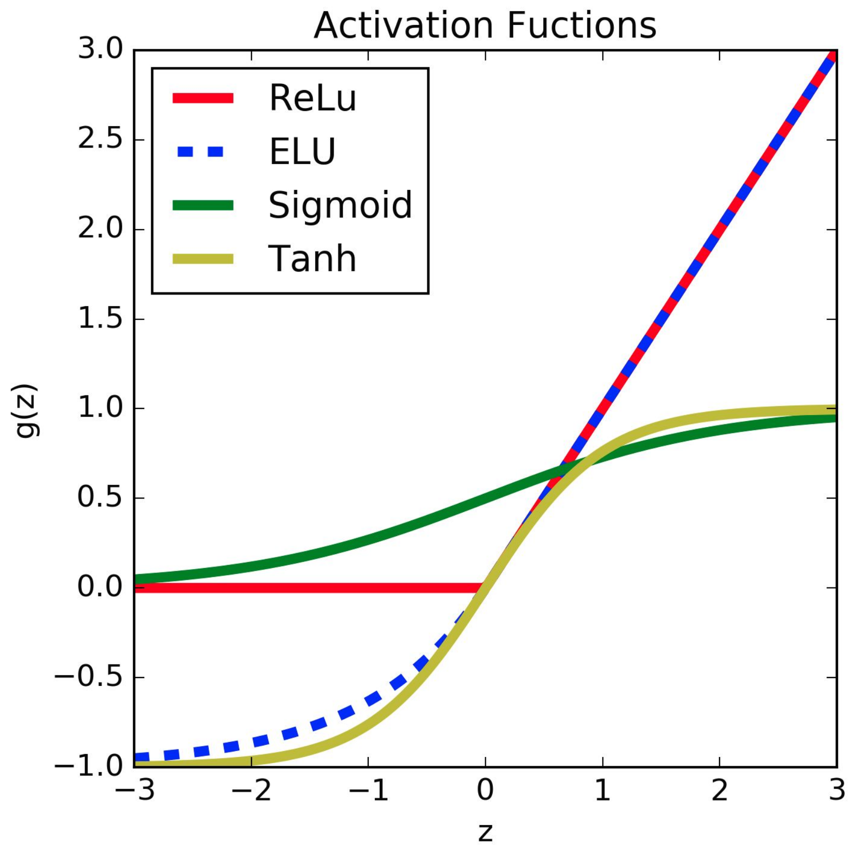
***Q:** Why not stack multiple linear layers?*

A: Composition of linear transformations is still linear.

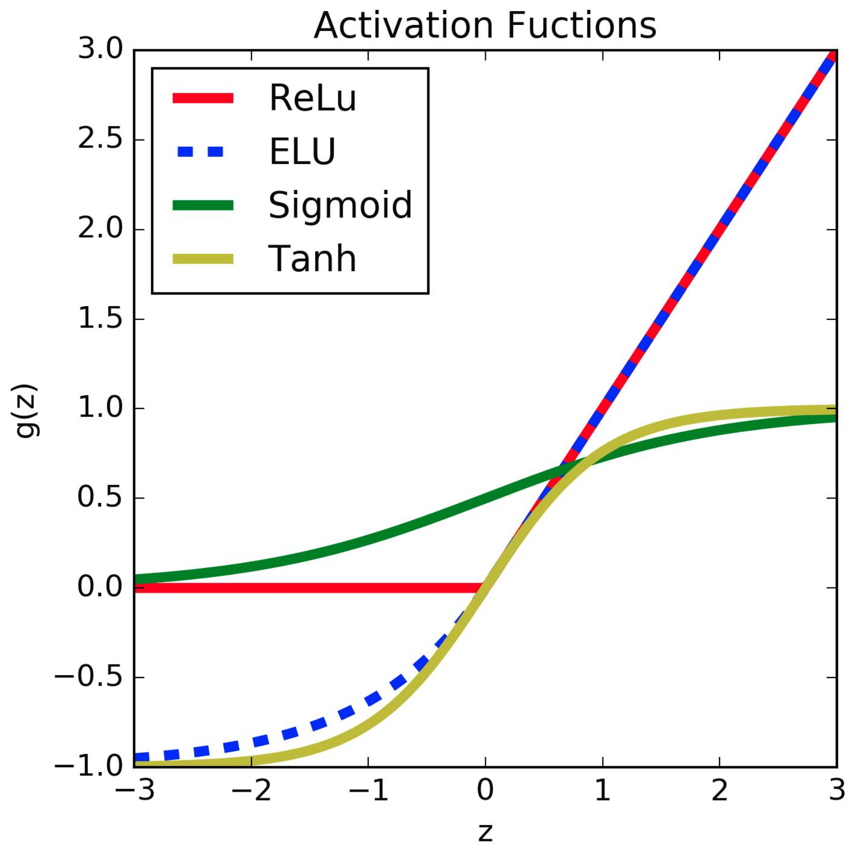
Activation function = non-linear transformation

1. Rectifier linear unit (**ReLu**): $g(z) = \begin{cases} 0, & \text{if } z < 0 \\ z, & \text{if } z \geq 0 \end{cases}$
2. Exponential linear unit (**ELU**): $g(z) = \begin{cases} e^z - 1, & \text{if } z < 0 \\ z, & \text{if } z \geq 0 \end{cases}$
3. **Sigmoid**: $g(z) = \dfrac{1}{1 + e^{-z}}$
4. Hyperbolic tangent (**Tanh**): $g(z) = \tanh(z)$

Activation Functions



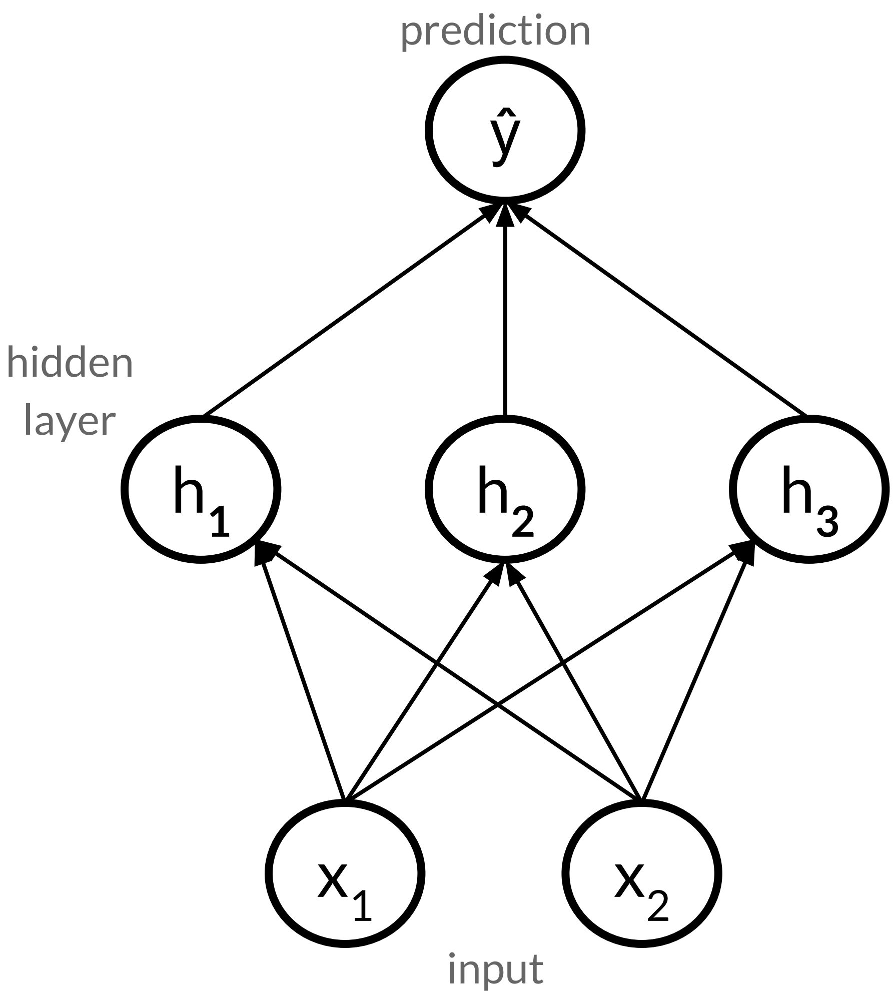
Activation Functions



1980-2010 : Sigmoid & Tanh. Problems: saturate (both sides) & hard to copy input

2010-now : ReLu & ELU (Partially linear functions): gradient flows more easily

ANN: Layer Stacking

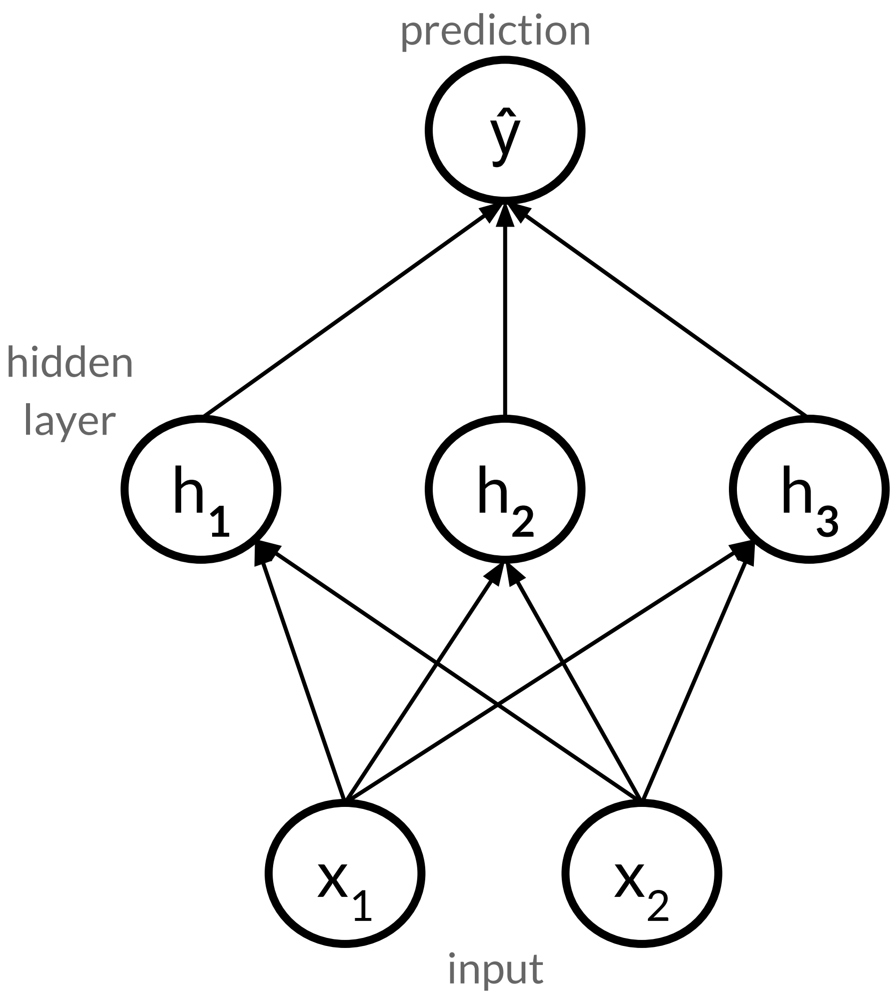


Idea:

Repeatedly apply the input to such a parametrized layer

$$\hat{y} = f^{(2)}(f^{(1)}(\mathbf{x}))$$

ANN: Layer Stacking



<u>Idea</u>:

Repeatedly apply the input to such a parametrized layer

$$\hat{y} = f^{(2)}(f^{(1)}(\mathbf{x}))$$

or, when fully written out

$$\hat{y} = f_\theta(\mathbf{x}) = \mathbf{W}^{(2)} g^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}$$

Note: In the last layer we do **not** apply a standard non-linearity g(). More about this in the loss function part.

B. Loss function

General idea:

1. Specify error measure between $\hat{y}$ (prediction) and y (true data target)
2. Minimize that quantity over the entire dataset

B. Loss function

General idea:

1. Specify error measure between ŷ (prediction) and y (true data target)
2. Minimize that quantity over the entire dataset

Two important considerations:

1. Type of y variable (regression vs classification)
2. Deterministic versus probabilistic loss

B. Loss function

1. Regression versus classification (= type of target variable (y))

B. Loss function

1. Regression versus classification (= type of target variable (y))

Target type (y)	Name	Prediction	Network output
Continuous	Regression	Number on real line	Direct prediction (1 head) or parameters of contin prob. distr.
Discrete	Classification	Class label out of a set	Usually one network head per class

B. Loss function

1. Regression versus classification (= type of target variable (y))

Target type (y)	**Name**	**Prediction**	**Network output**
Continuous	Regression	Number on real line	Direct prediction (1 head) or parameters of contin prob. distr.
Discrete	Classification	Class label out of a set	Usually one network head per class

Cardinal example: Regression on Mean-Squared Error (MSE)

$$\mathcal{L}(\theta|y,\mathbf{x}) = \mathbb{E}_{\mathcal{D}}\left[\Big(f(\mathbf{x};\theta) - y\Big)^2\right] = \frac{1}{N}\sum_{i=1}^{N}\Big(f(\mathbf{x}_i;\theta) - y_i\Big)^2$$

B. Loss function

1. Regression versus classification (= type of target variable (y))

Target type (y)	Name	Prediction	Network output
Continuous	Regression	Number on real line	Direct prediction (1 head) or parameters of contin prob. distr.
Discrete	Classification	Class label out of a set	Usually one network head per class

Cardinal example: Regression on Mean-Squared Error (MSE)

$$\mathcal{L}(\theta|y,\mathbf{x}) = \mathbb{E}_{\mathcal{D}}\left[\left(f(\mathbf{x};\theta) - y\right)^2\right] = \frac{1}{N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i;\theta) - y_i\right)^2$$

square the error

sum over whole dataset

prediction

true label

B. Loss function

1. Regression versus classification (= type of target variable (y))

Target type (y)	**Name**	**Prediction**	**Network output**
Continuous	Regression	Number on real line	Direct prediction (1 head) or parameters of contin prob. distr.
Discrete	Classification	Class label out of a set	Usually one network head per class

Cardinal example: Regression on Mean-Squared Error (MSE)

$$\mathcal{L}(\theta|y,\mathbf{x}) = \mathbb{E}_{\mathcal{D}}\left[\Big(f(\mathbf{x};\theta) - y\Big)^2\right] = \frac{1}{N}\sum_{i=1}^{N}\Big(f(\mathbf{x}_i;\theta) - y_i\Big)^2$$

square the error

sum over whole dataset

prediction

true label

Q: why the square of the error?

B. Loss function

1. Regression versus classification (= type of target variable (y))

Target type (y)	**Name**	**Prediction**	**Network output**
Continuous	Regression	Number on real line	Direct prediction (1 head) or parameters of contin prob. distr.
Discrete	Classification	Class label out of a set	Usually one network head per class

Cardinal example: Regression on Mean-Squared Error (MSE)

$$\mathcal{L}(\theta|y,\mathbf{x}) = \mathbb{E}_{\mathcal{D}}\left[\Big(f(\mathbf{x};\theta) - y\Big)^2\right] = \frac{1}{N}\sum_{i=1}^{N}\Big(f(\mathbf{x}_i;\theta) - y_i\Big)^2$$

square the error

sum over whole dataset

prediction

true label

Q: why the square of the error?

A: penalize positive **and** negative errors + easier derivative (compared to absolute error)

B. Loss function

2. Deterministic versus probabilistic loss

Main idea of probabilistic loss: The network predicts the *parameters of a probability distribution* out of which the observed y would be sampled, instead of predicting y directly.

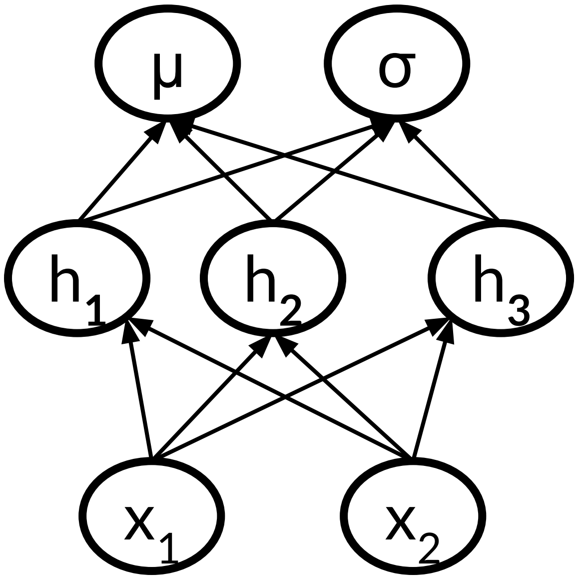
B. Loss function

2. Deterministic versus probabilistic loss

Main idea of probabilistic loss: The network predicts the *parameters of a probability distribution* out of which the observed y would be sampled, instead of predicting y directly.

For example:

$\hat{y} \sim N(.|\mu,\sigma)$ and



B. Loss function

2. Deterministic versus probabilistic loss

Main idea of probabilistic loss: The network predicts the *parameters of a probability distribution* out of which the observed y would be sampled, instead of predicting y directly.

Benefits:

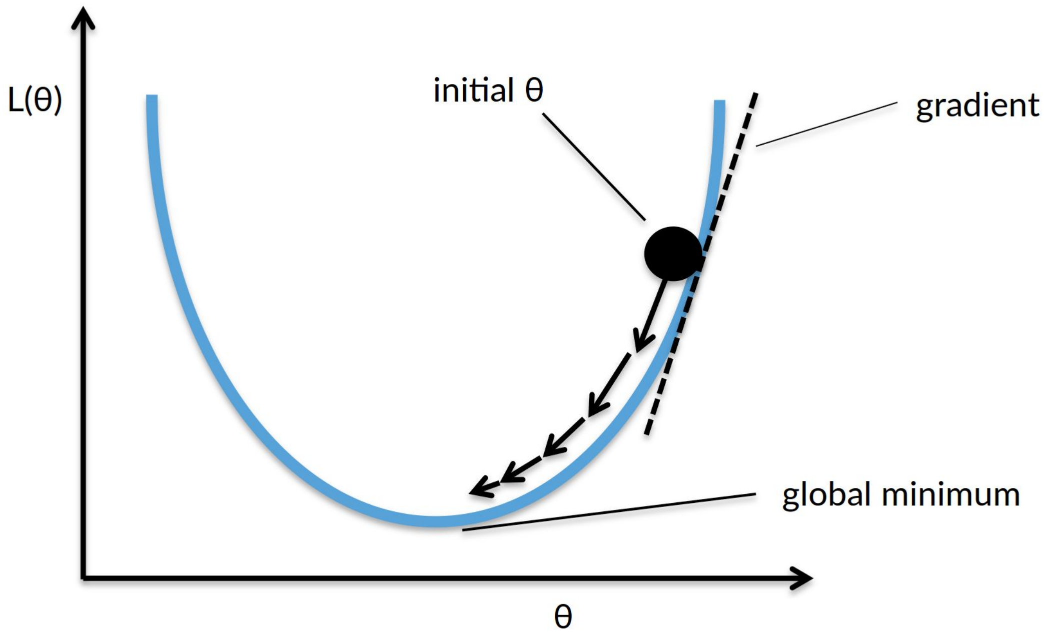
1. Model stochastic output & sensor noise
2. Directly have a loss function:

   '*Maximum likelihood estimation*' = learn a model that gives maximum probability to the observed data

See lecture notes for details (also for classification case)

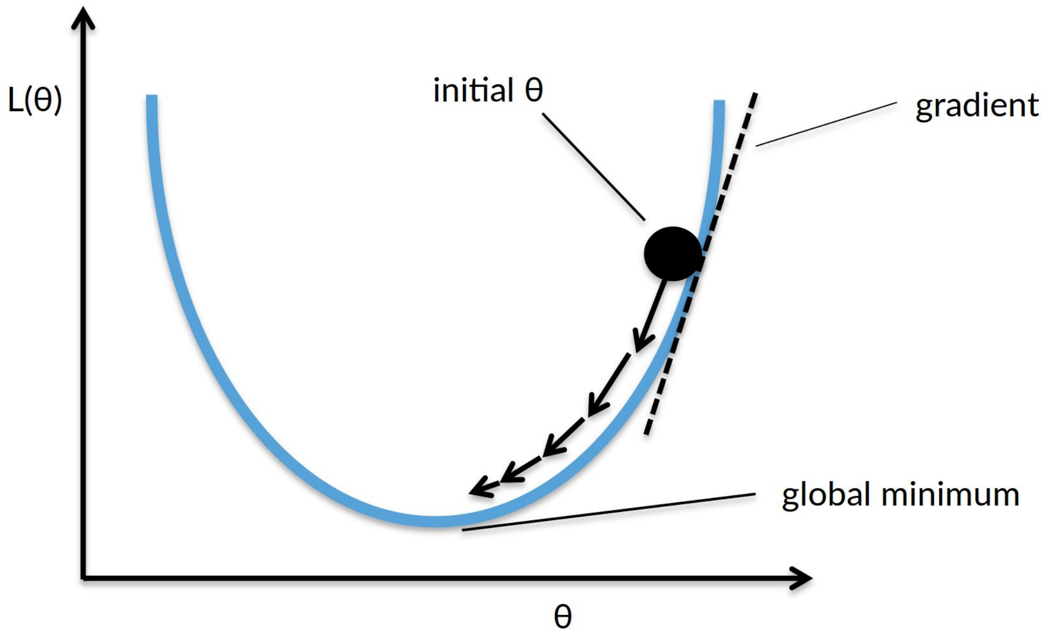
C. Numerical optimization

Gradient Descent



C. Numerical optimization

Gradient Descent

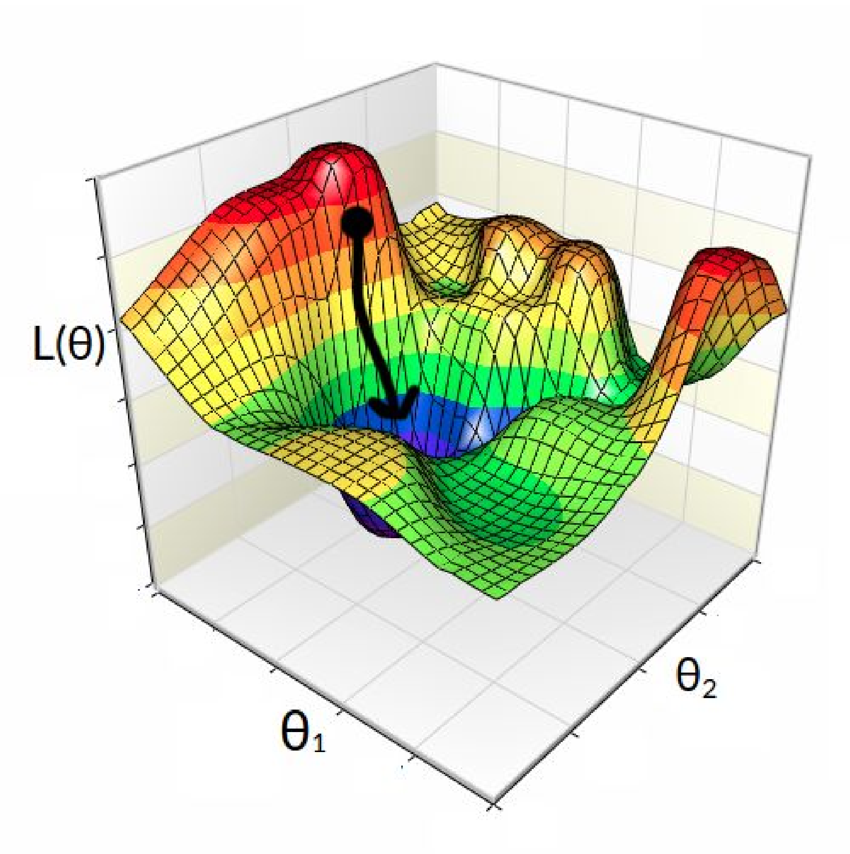


Update rule:

$$\theta' = \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

Learning rate

Non-Convex Objective Function



NN objective/cost

=

Non-convex

Learning rate = crucial

Too small : no progress

Too large : unstable

Importance of learning rate

Gradient Descent for Neural Networks

Two issues around the same problem:

How do we get the gradients in feasible computational time?

1. **Datasets are usually large**:

Solution: stochastic gradient descent (SGD)

2. **Networks are usually large**:

Solution: backpropagation ('backprop')

Stochastic Gradient Descent

True gradient is a sum over the entire dataset:

$$\nabla_\theta \mathcal{L}(\theta|y, \mathbf{x}) = \sum_{i=1}^{N} \nabla_\theta \Big(f(\mathbf{x_i}; \theta) - y_i \Big)^2$$

Dataset size (N) may be millions.

Stochastic Gradient Descent

True gradient is a sum over the entire dataset:

$$\nabla_\theta \mathcal{L}(\theta | y, \mathbf{x}) = \sum_{i=1}^{N} \nabla_\theta \Big(f(\mathbf{x_i}; \theta) - y_i \Big)^2$$

Dataset size (N) may be millions.

Solution: approximate the gradient with a sample from the dataset
(= a 'minibatch' per parameter update)

$$\mathbf{grad} = \sum_{i=1}^{m} \nabla_\theta \Big(f(\mathbf{x_i}; \theta) - y_i \Big)^2$$

Minibatch size (usually m=32 or m=64) stays fixed when dataset grows!

Backpropagation

First: How do we get the gradient anyway?

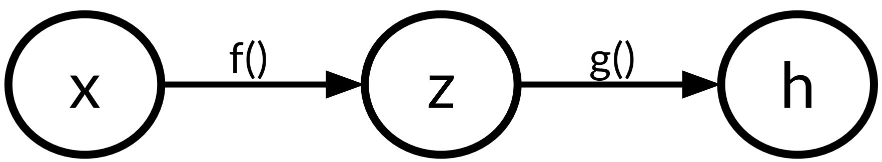
Backpropagation

First: How do we get the gradient anyway?

Required: Chain Rule of Calculus

Example:

z = f(x)   h = g(z)   -->   h = g(f(x))



How do we get dh/dx?

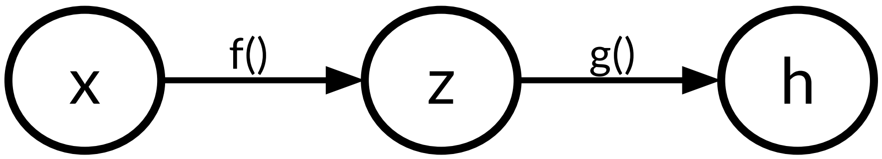
Backpropagation

First: How do we get the gradient anyway?

Required: Chain Rule of Calculus

Example:

z = f(x) h = g(z) --> h = g(f(x))



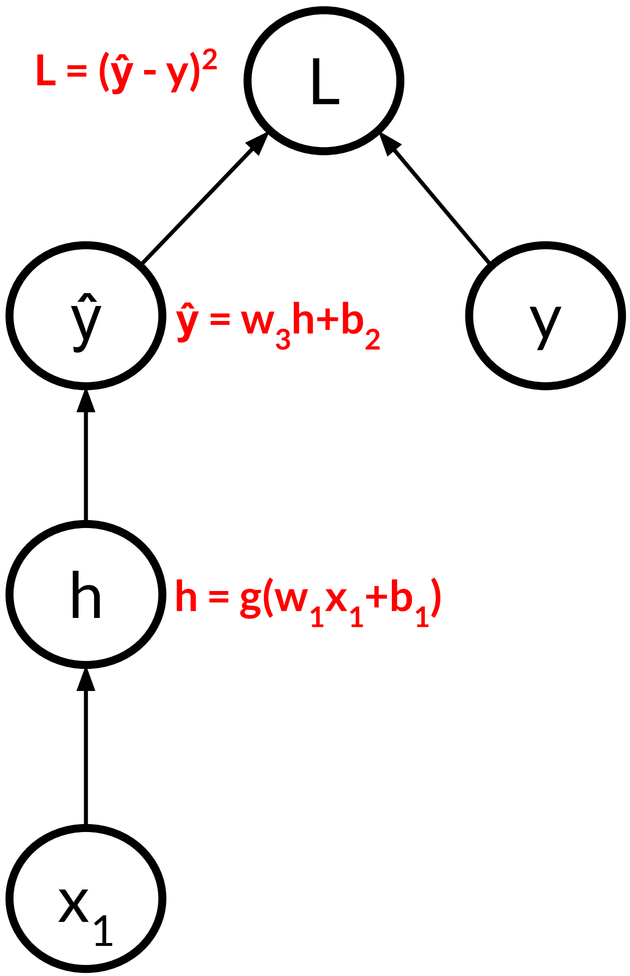
How do we get dh/dx?

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

chain = multiply the gradients of the subfunctions

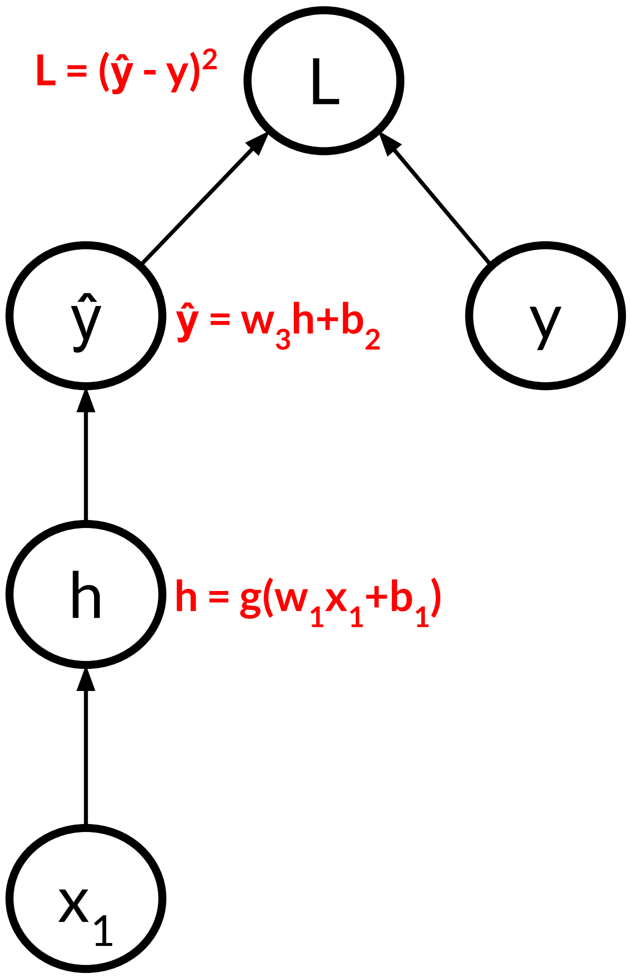
*(generalizes to case where **x,z** and **h** are vectors - need <u>partial derivatives</u> (see lecture notes))*

Class example: NN gradients



Q: To update weight w_1 we need dL/dw_1. Give dL/dw_1 (symbolic).

Class example: NN gradients

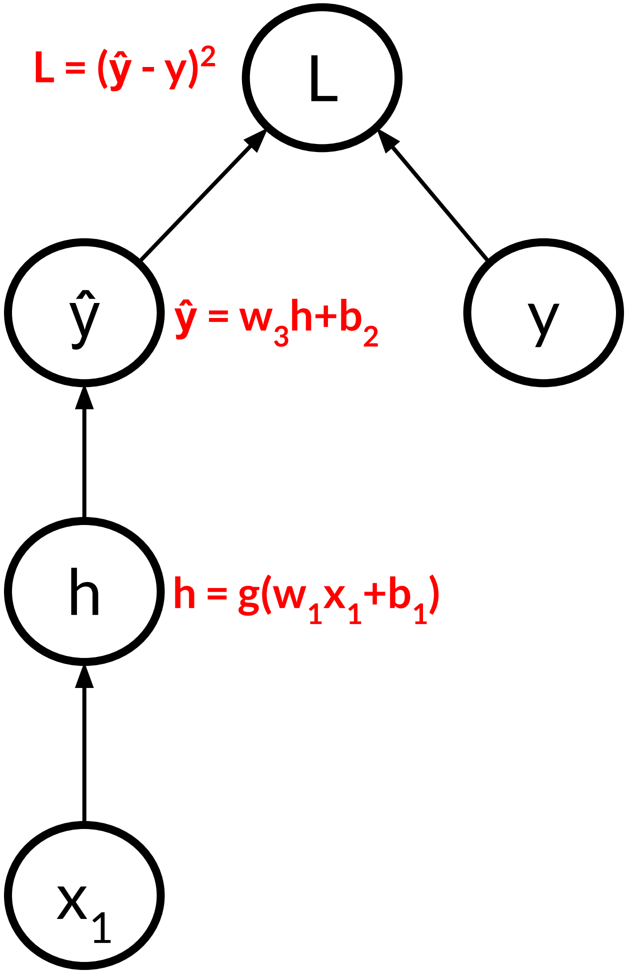


Q: To update weight w_1 we need dL/dw_1. Give dL/dw_1 (symbolic).

A:

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_1}$$

Class example: NN gradients

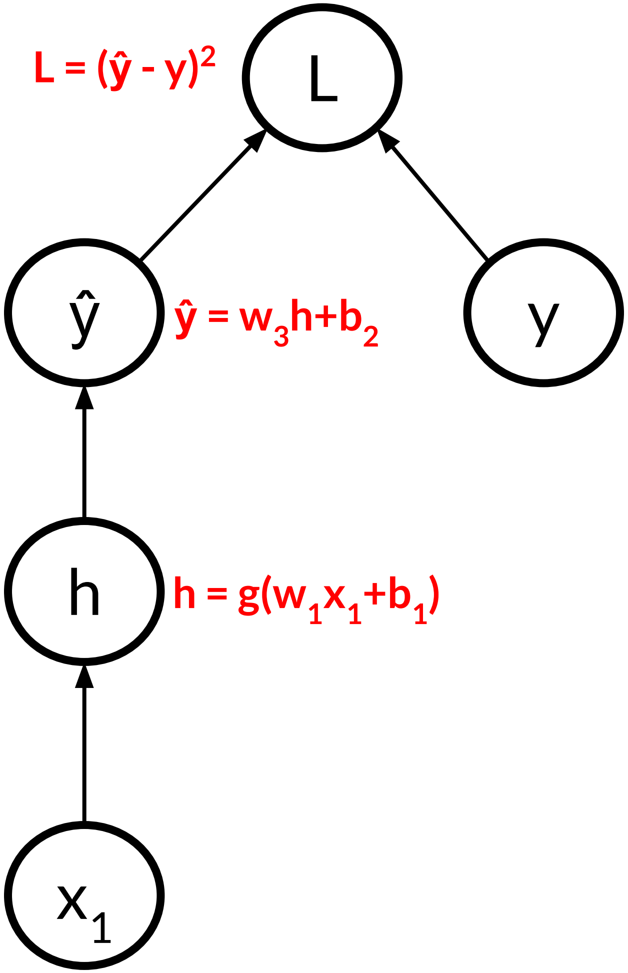


Q: To update weight w_1 we need dL/dw_1. Give dL/dw_1 (symbolic).

A:
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_1}$$

Q: Can you further write out dh/dw_1? (think about the non-linearity)

Class example: NN gradients



Q: To update weight w_1 we need dL/dw_1. Give dL/dw_1 (symbolic).

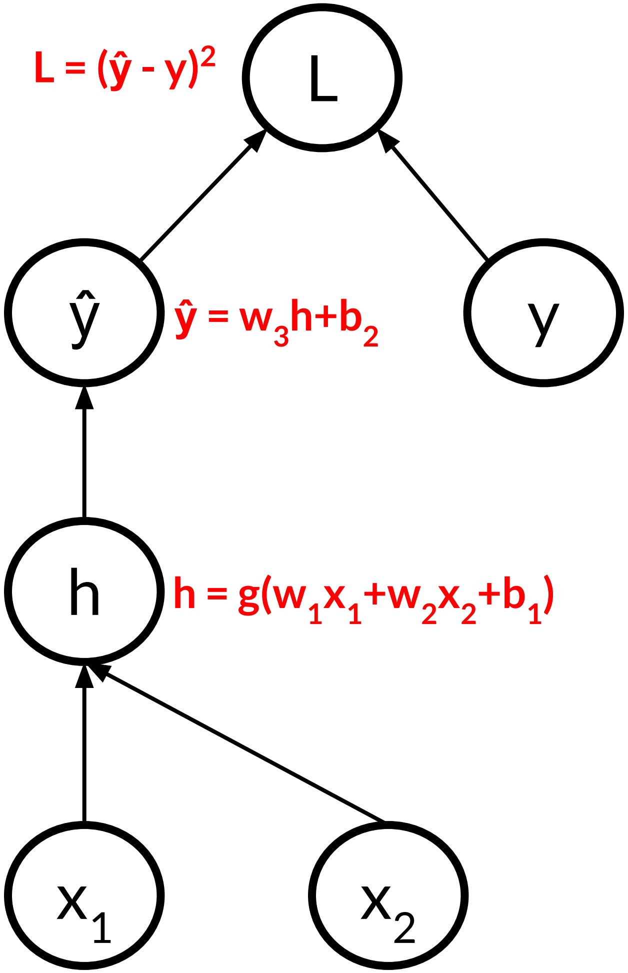
A: $$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_1}$$

Q: Can you further write out dh/dw_1? (think about the non-linearity)

$$z = w_1 x + b_1 \quad \text{and} \quad h = g(z)$$

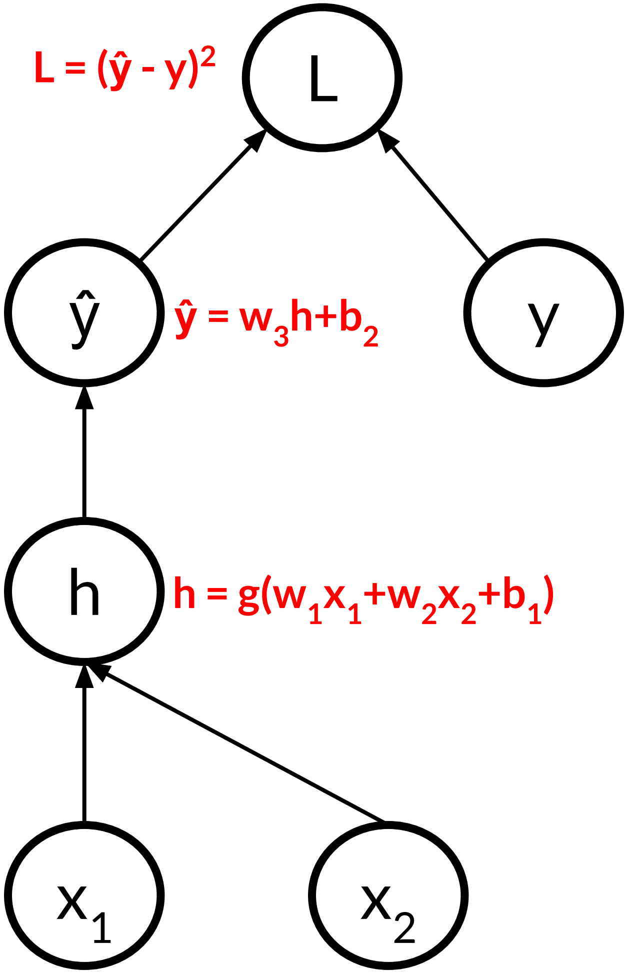
$$\frac{\partial h}{\partial w_1} = \frac{\partial h}{\partial z} \frac{\partial z}{\partial w_1}$$

Class example: NN gradients



Q: Now our input **x** is actually a vector of length 2. Can you give dL/dw_1 and dL/dw_2?

Class example: NN gradients



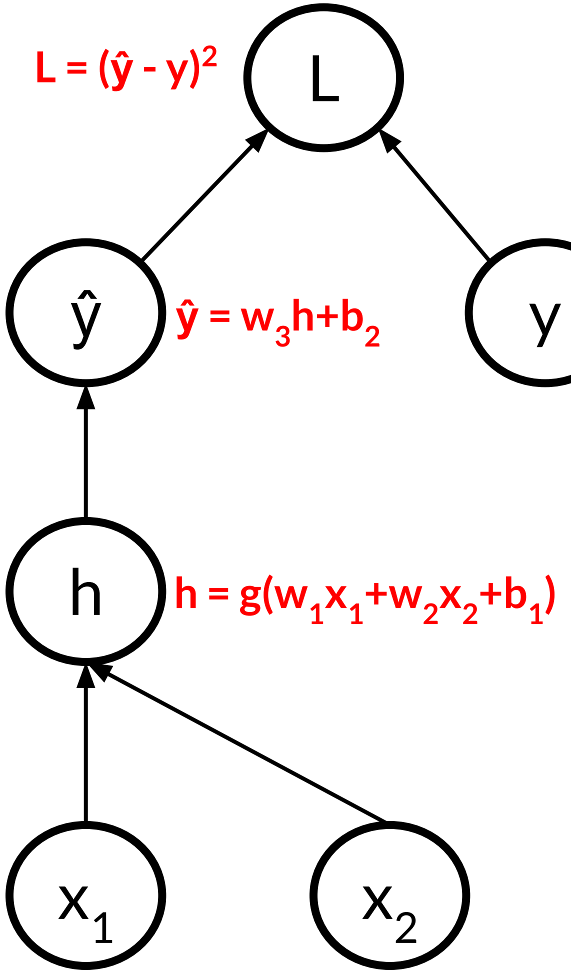
Q: Now our input **x** is actually a vector of length 2. Can you give dL/dw_1 and dL/dw_2?

A:

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_2}$$

Class example: NN gradients



Q: Now our input **x** is actually a vector of length 2. Can you give dL/dw$_1$ and dL/dw$_2$?

A:

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w_2}$$

large part of the gradient is the same

(= key idea of backpropagation)

Backpropagation

Main idea:

- Efficiently store gradients and re-use them by **walking backwards** through the network.

Backpropagation

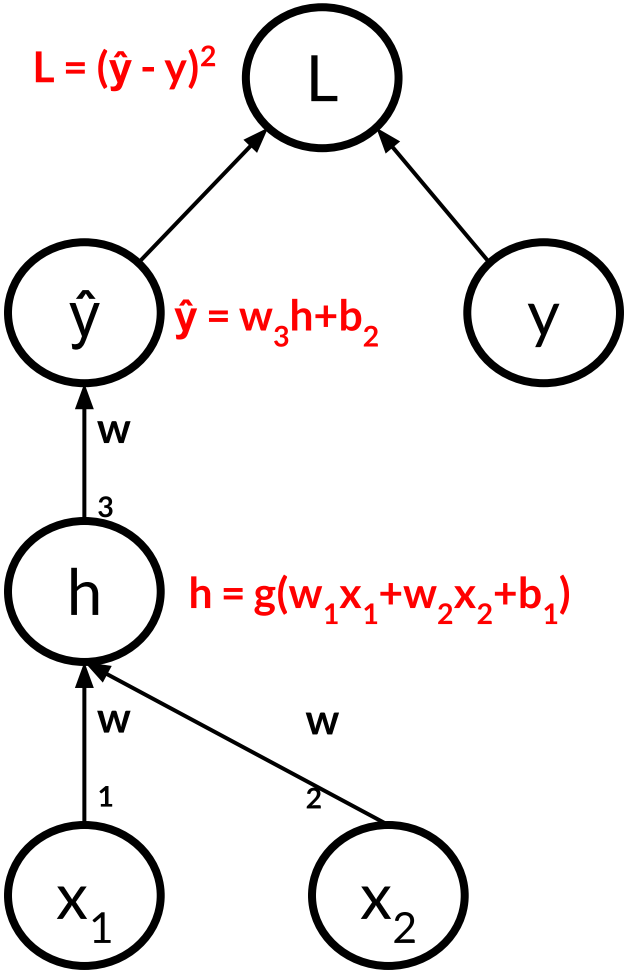
Main idea:

- Efficiently store gradients and re-use them by **walking backwards** through the network.

Backpropagation algorithm

$\mathbf{grad} = \nabla_{\hat{y}} \mathcal{L}$	differentiate the loss w.r.t. the network prediction
for d in $l..1$:	
$\quad \mathbf{grad} \leftarrow \nabla_{\mathbf{z}^{(d)}} \mathcal{L} = \mathbf{grad} \odot \frac{dg^{(d)}}{dz_j^{(d)}}$	propagate through non-linearity
$\quad \nabla_{\mathbf{b}^{(d)}} \mathcal{L} = \mathbf{grad}$	gradients for biases in layer d
$\quad \nabla_{\mathbf{W}^{(d)}} \mathcal{L} = \mathbf{grad} \cdot \mathbf{h}^{(d-1)}$	gradients for weights in layer d
$\quad \mathbf{grad} \leftarrow \nabla_{\mathbf{h}^{(d-1)}} \mathcal{L} = \mathbf{grad} \cdot \mathbf{W}^{(d)}$	propagate gradients to hidden units of next layer $d-1$

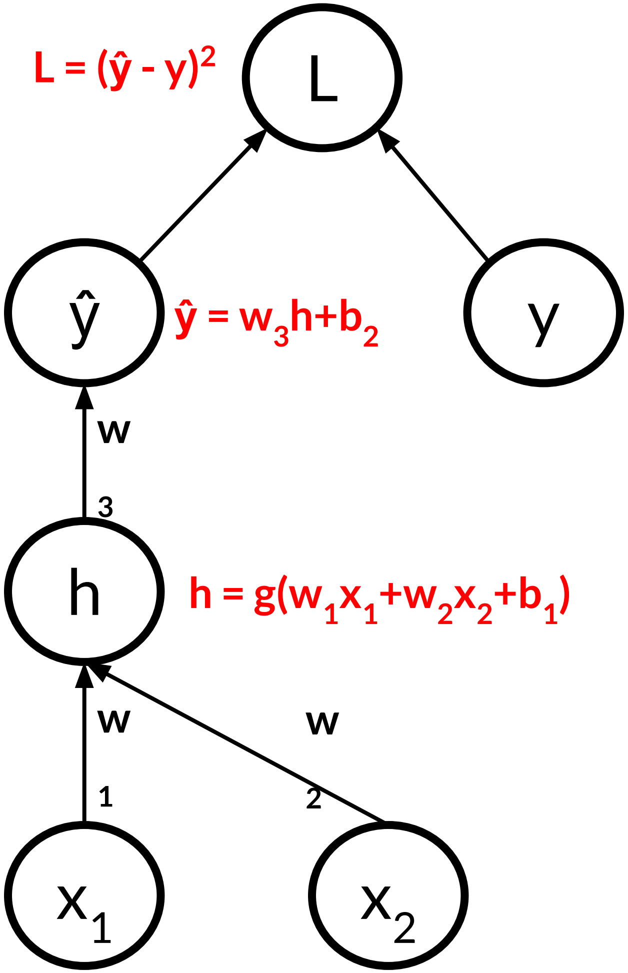
Class example: One full learning loop



Let's assume some data and initialize parameters:

$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu} = \max(0,z)$

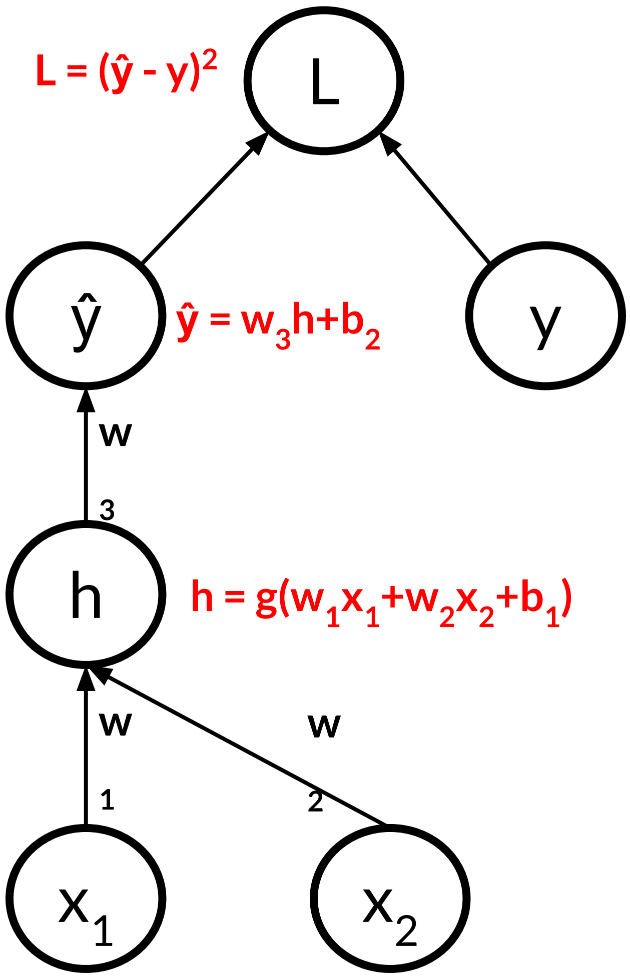
Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu} = \max(0,z)$

Q: Compute $\hat{y}$ (forward pass)

Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu} = \max(0,z)$

Q: Compute ŷ (forward pass)

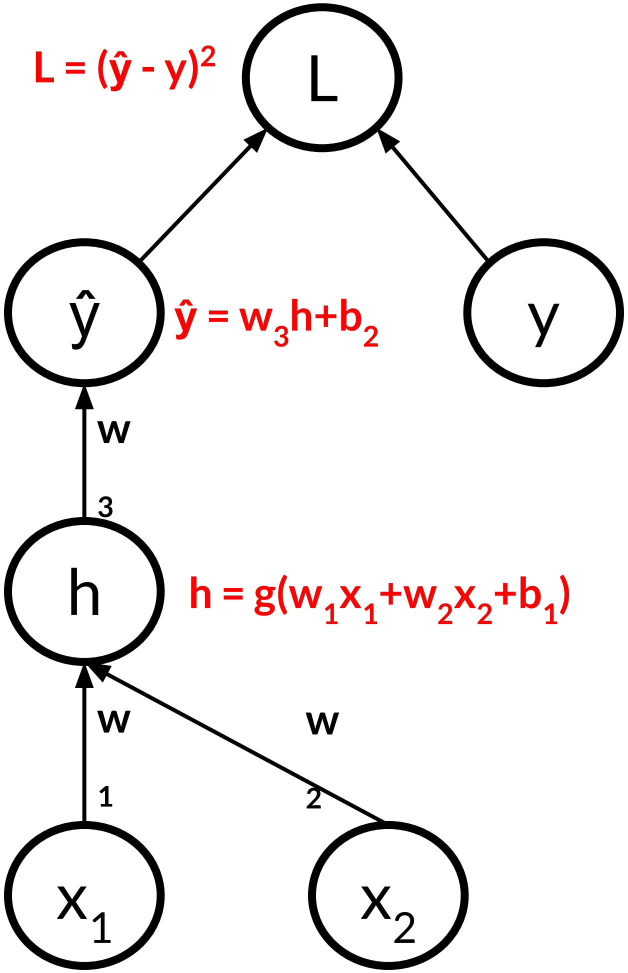
A:

$z = (2*1.5) + (-1*2) + 3 = 4$

$h = \max(0,4) = 4$

$\hat{y} = (2.5*4) - 2 = 8$

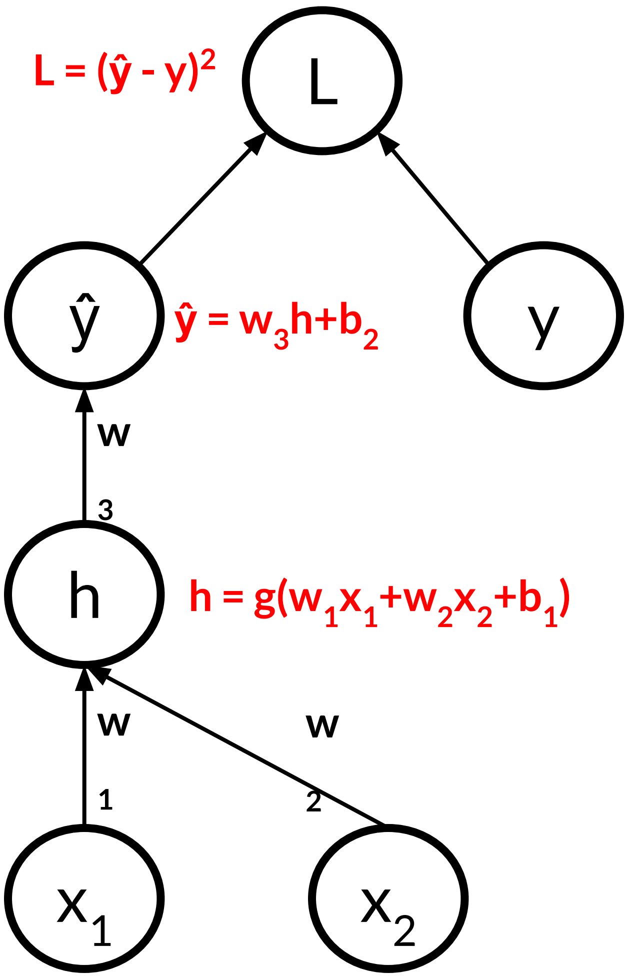
Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: We assume the squared loss $L = (\hat{y} - y)^2$. Compute the loss for this datapoint.

Class example: One full learning loop

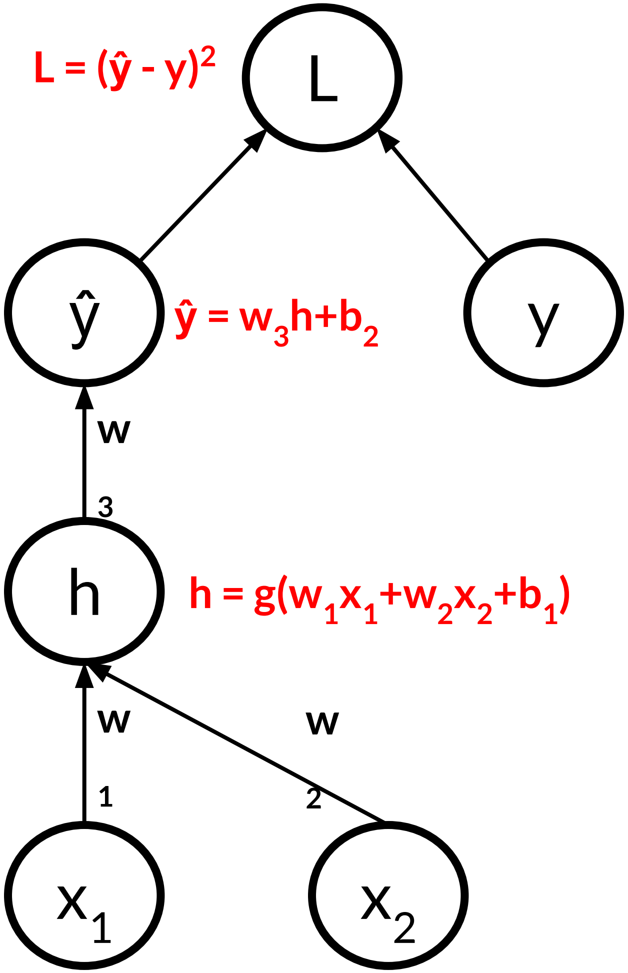


$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: We assume the squared loss $L = (\hat{y} - y)^2$. Compute the loss for this datapoint.

A: $L = (8 - 6)^2 = 4$

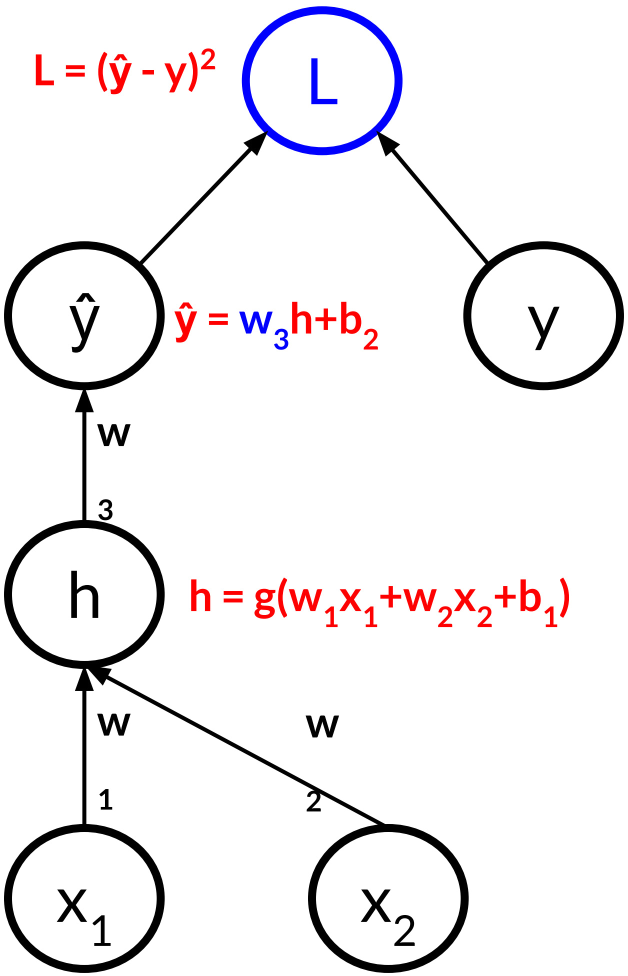
Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: Let backpropagate. Calculate dL/dw_3.

Class example: One full learning loop



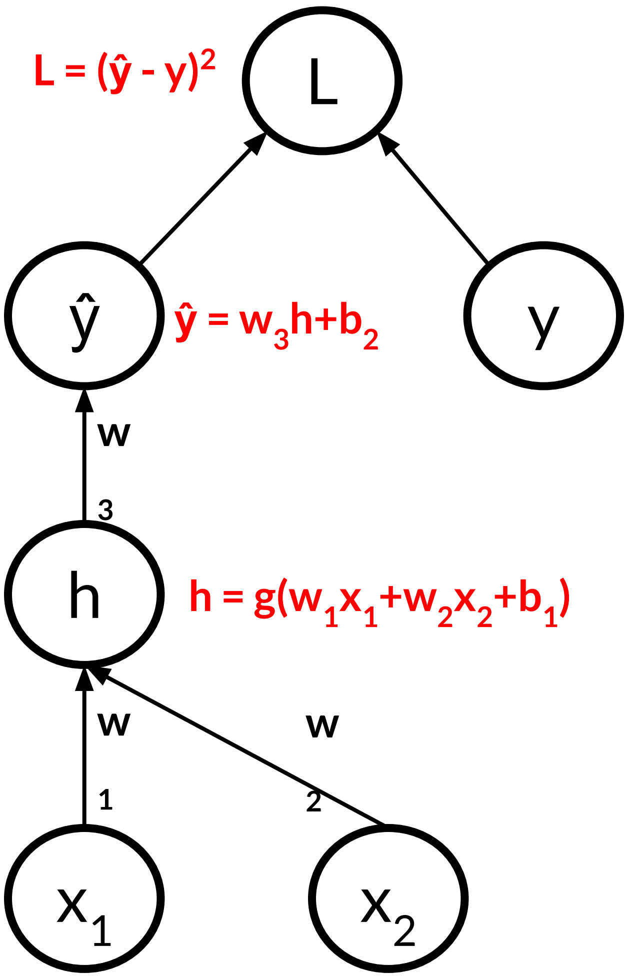
$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: Let backpropagate. Calculate dL/dw_3.

A:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_3} &= \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_3} \\ &= \frac{\partial}{\partial \hat{y}}(\hat{y} - y)^2 \frac{\partial}{\partial w_3}(w_3 h + b_2) \\ &= 2(\hat{y} - y) \cdot h \\ &= 2(8 - 6) \cdot 4 = 16\end{aligned}$$

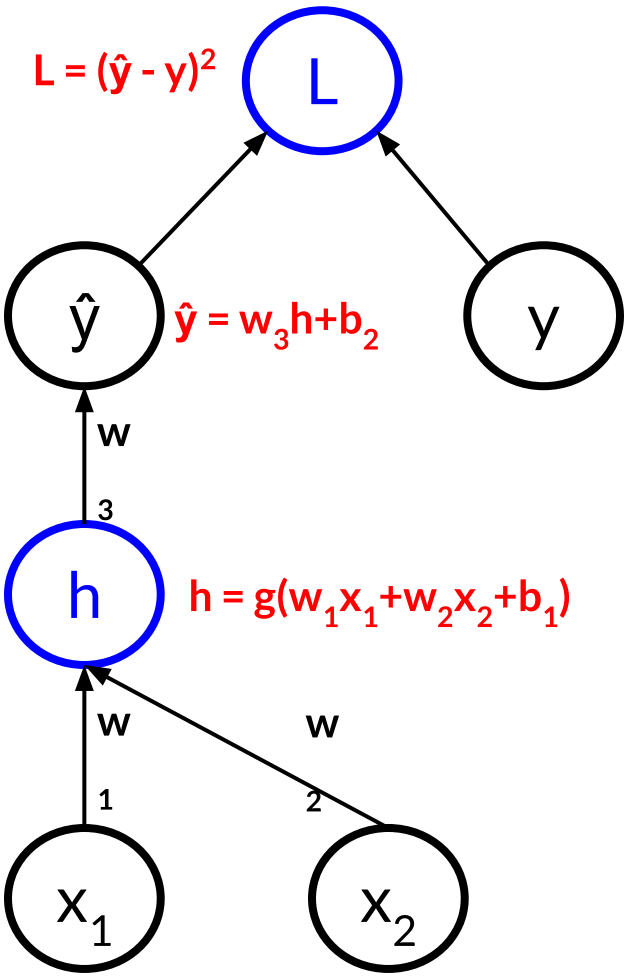
Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: Now for dL/dw_1 and dL/db_1

Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: Now for dL/dw_1 and dL/db_1

A:

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h}$$

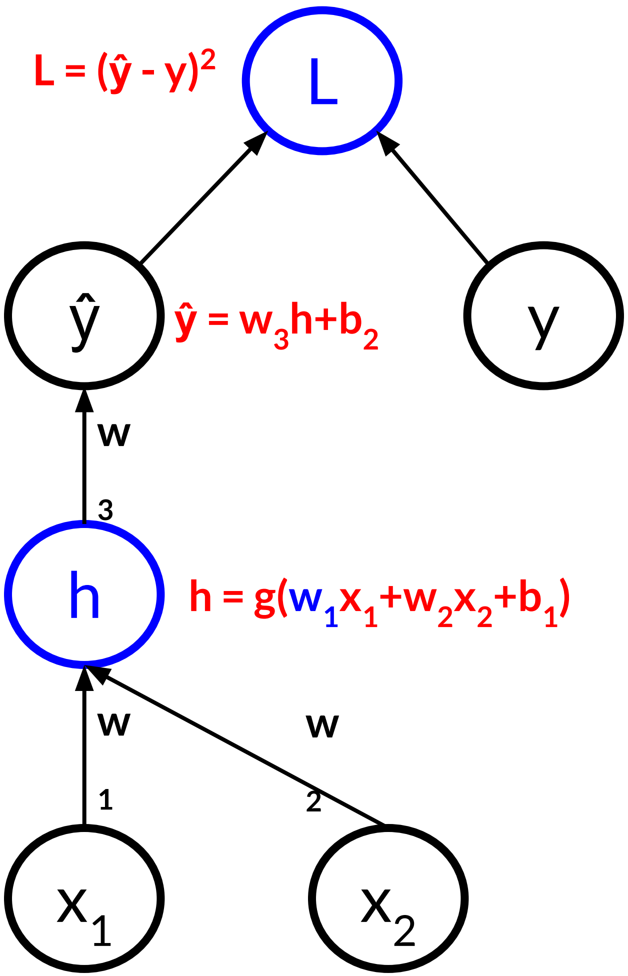
First through the top layer

$$= \frac{\partial}{\partial \hat{y}} (\hat{y} - y)^2 \frac{\partial}{\partial h} (w_3 h + b_2)$$

$$= 2(\hat{y} - y) \cdot w_3$$

$$= 2(8 - 6) \cdot 2.5 = 10$$

Class example: One full learning loop

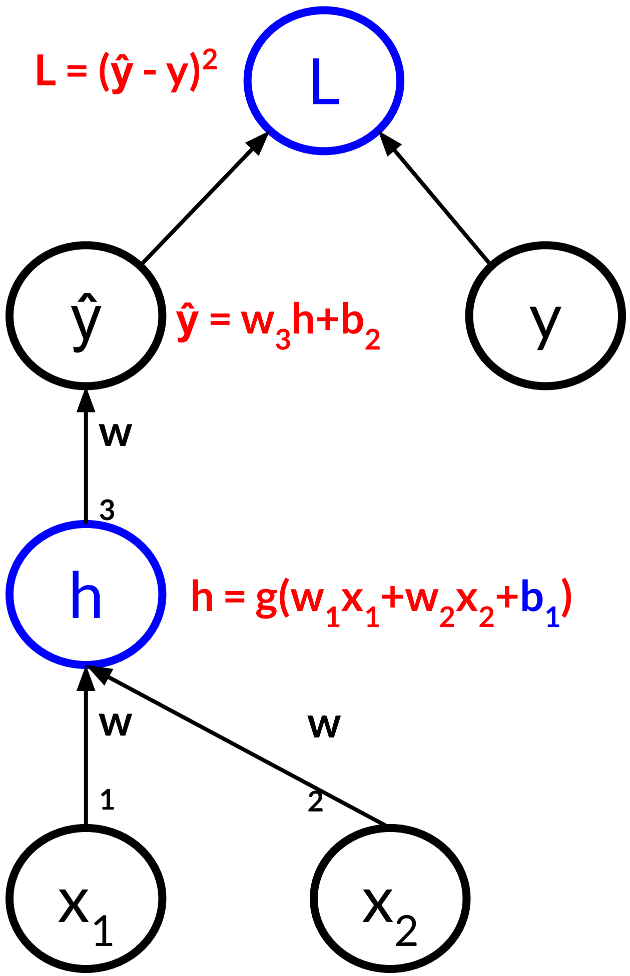


$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: Now for dL/dw_1 and dL/db_1

A:
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial w_1}$$
$$= 10 \cdot \frac{\partial}{\partial z}\max(0, z)\frac{\partial}{\partial w_1}(w_1 x_1 + w_2 x_2 + b_1)$$
$$= 10 \cdot 1 \cdot x_1$$
$$= 10 \cdot 2 = 20$$

Class example: One full learning loop



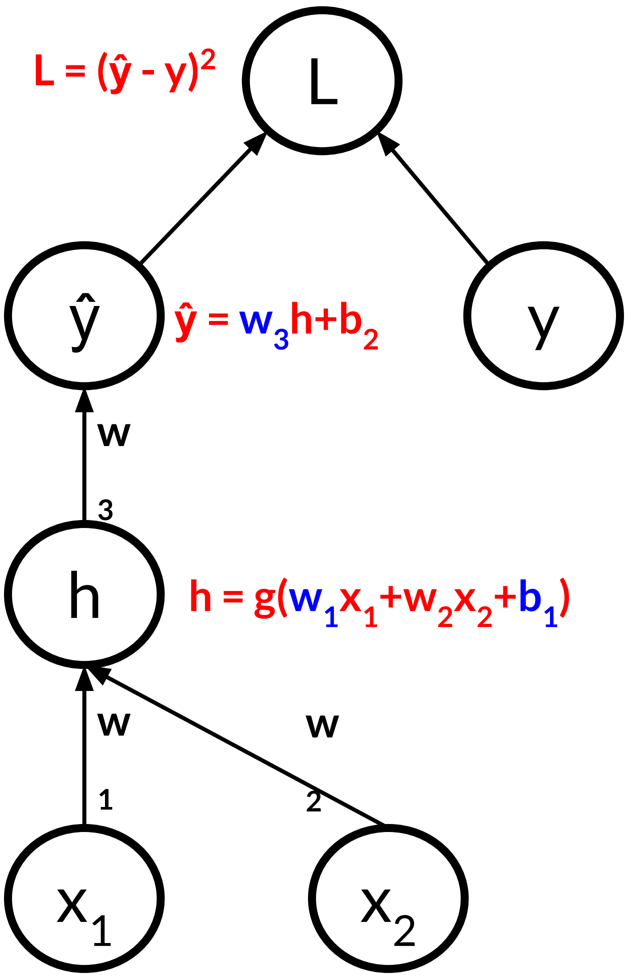
$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z)$ = ReLu	$\hat{y} = 8$

Q: Now for dL/dw_1 and dL/db_1

A:
$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial b_1}$$
$$= 10 \cdot \frac{\partial}{\partial b_1}(w_1 x_1 + w_2 x_2 + b_1)$$
$$= 10 \cdot 1 \cdot 1$$
$$= 10 \cdot 1 = 10$$

Re-use previous gradient

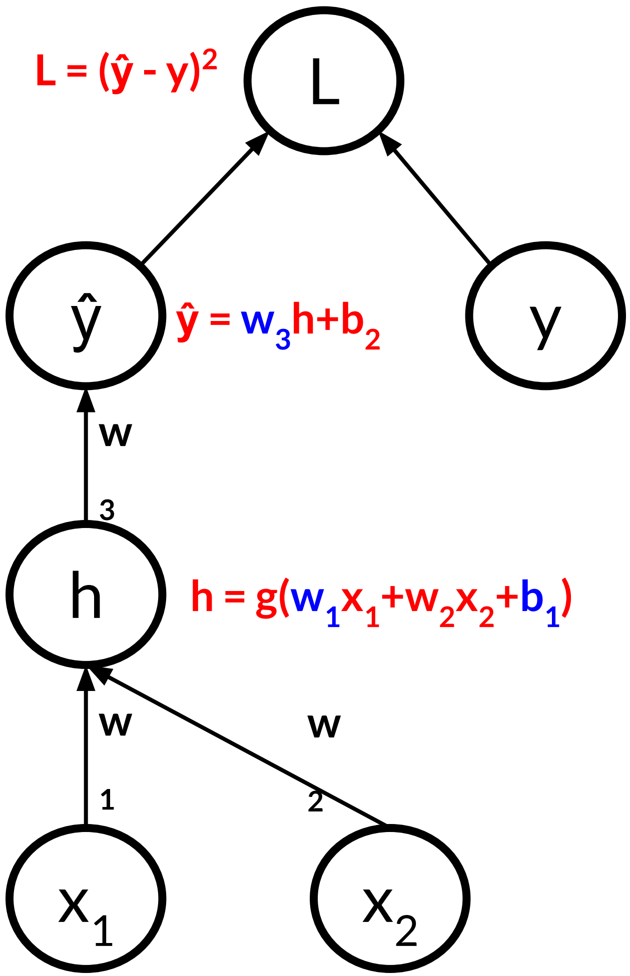
Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z)$ = ReLu	$\hat{y} = 8$

Q: So $dL/dw_3=16$, $dL/dw_1=20$ and $dL/db_1=10$. Update parameters, take learning rate 0.01.

Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.5$	$b_1 = 3$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.5$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: So $dL/dw_3=16$, $dL/dw_1=20$ and $dL/db_1=10$. Update parameters, take learning rate 0.01.

A:

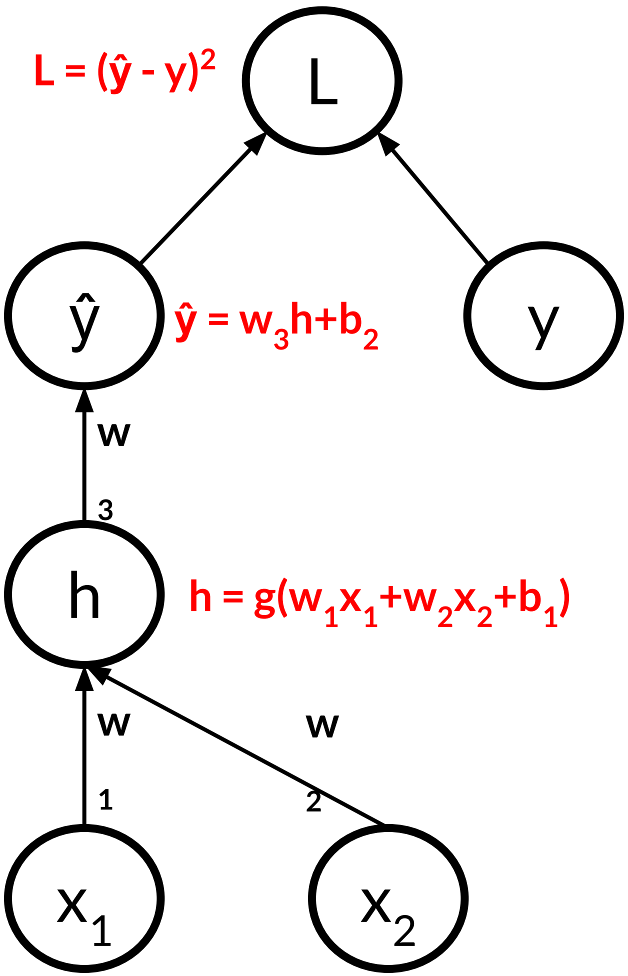
$w_1 = 1.5 - 0.01*20 = 1.3$

$b_1 = 3 - 0.01*10 = 2.9$

$w_3 = 2.5 - 0.01*16 = 2.34$

(Note: normally we update all parameters, i.e. w_2 and b_2 as well)

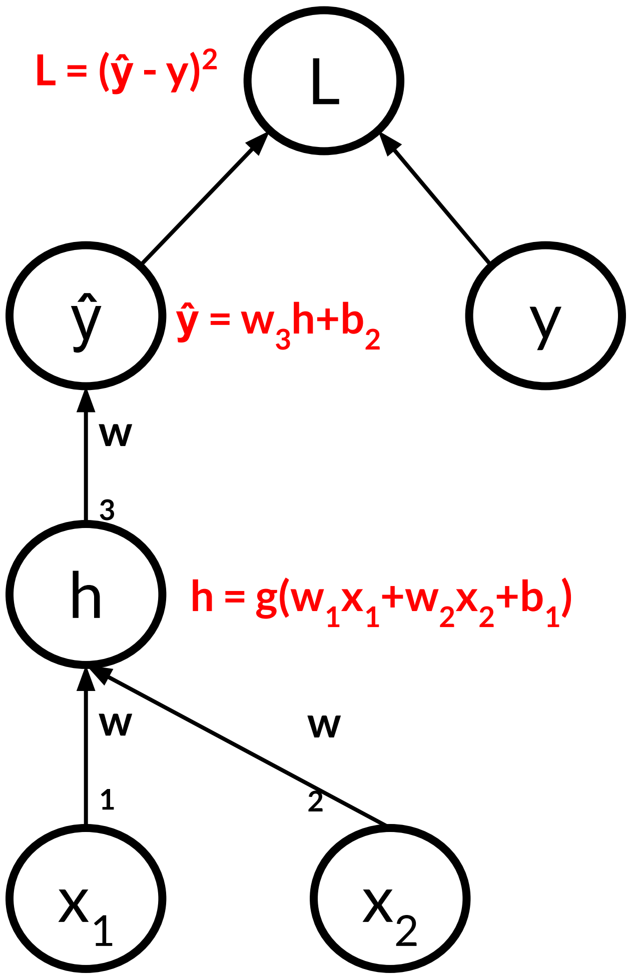
Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.3$	$b_1 = 2.9$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.34$	$g(z) = \text{ReLu}$	$\hat{y} = 8$

Q: So we have update the parameters. Did our prediction get better?

Class example: One full learning loop



$x_1 = 2$	$w_1 = 1.3$	$b_1 = 2.9$	$z = 4$
$x_2 = -1$	$w_2 = 2$	$b_2 = -2$	$h = 4$
$y = 6$	$w_3 = 2.34$	$g(z) = \text{ReLu}$	$\hat{y} = 6.19$

Q: So we have update the parameters. Did our prediction get better?

A: $z = (2*1.3) + (-1*2) + 2.9 \quad = 3.5$

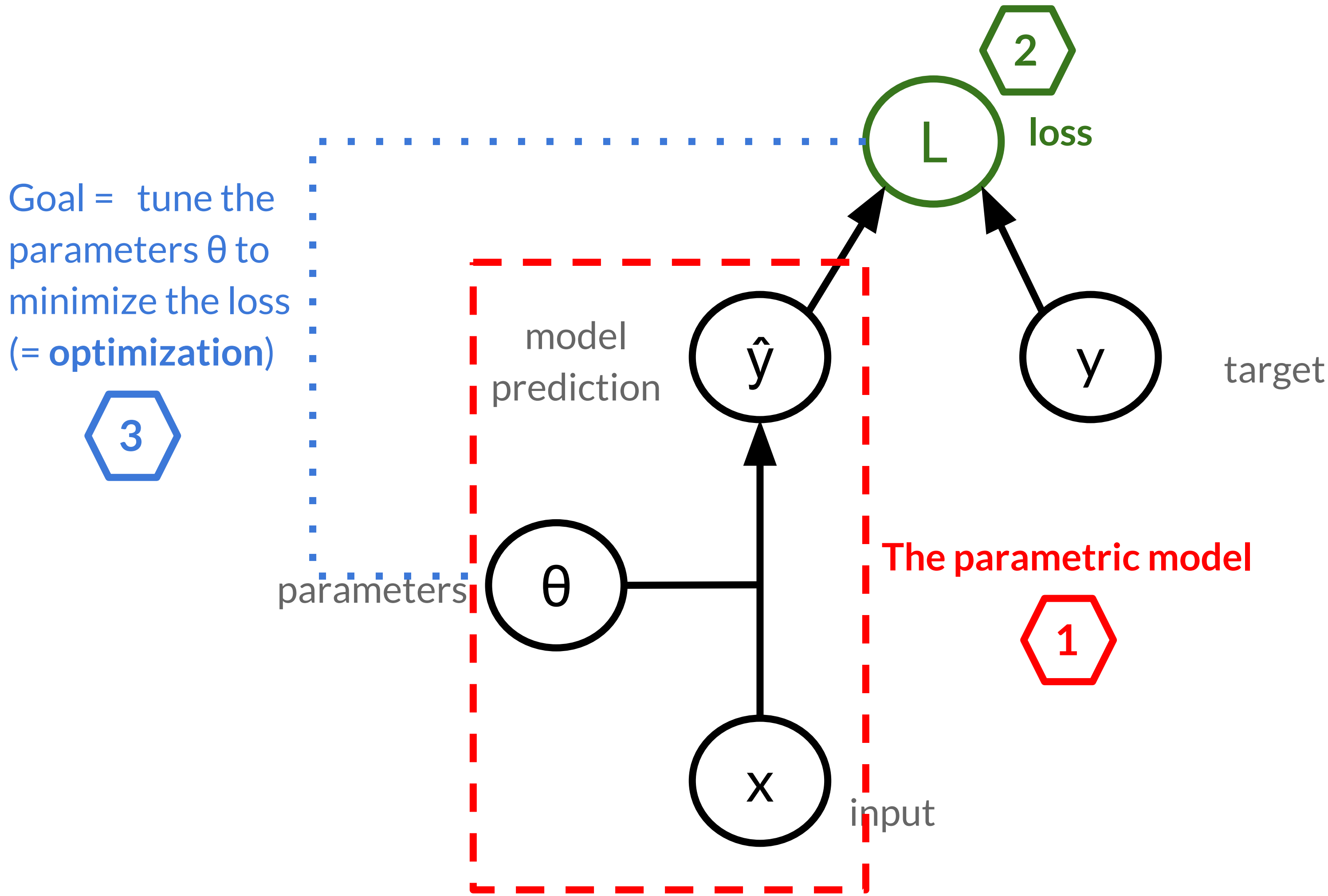
$h = \max(0,3.5) \quad = 3.5$

$\hat{y} = (2.34*3.5) - 2 \quad = 6.19$

<u>Yes, we got much closer!</u>

$(8 \rightarrow 6.19$, while true y is $6)$

Summary: You just manually trained a neural network (one learning loop)



Break

2. Advanced Neural Network Architectures

Advanced neural network architectures

1. Convolutional Neural Network (CNN)

= 'the NN solution to *space*'

2. Recurrent Neural Network (RNN)

= 'the NN solution to *time*/sequence'

Convolutional Neural Network (CNN)

Problem:
For high-dimensional input (e.g. images) fully connected layers have way too many parameters/connections.

Convolutional Neural Network (CNN)

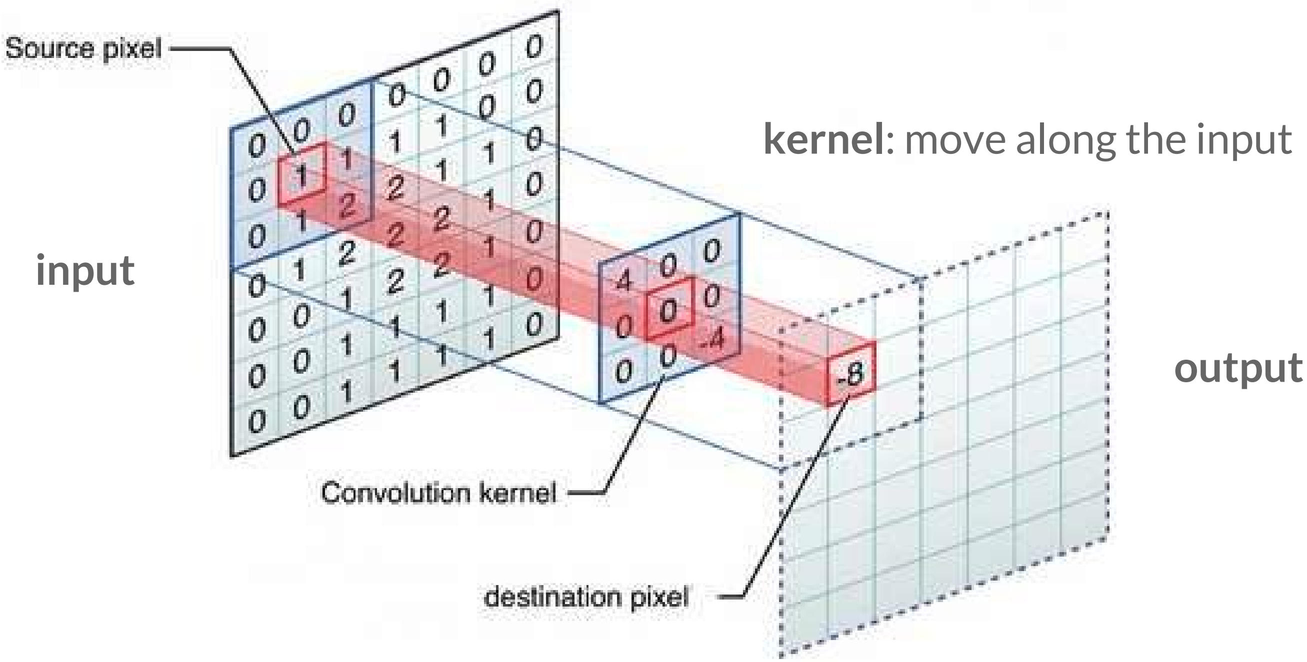
Problem:
For high-dimensional input (e.g. images) fully connected layers have way too many parameters/connections.

Solution:
Convolutions. Useful for data with *grid-like structure*, especially 2D/3D (computer vision), where *subpatterns re-appear throughout the grid*.

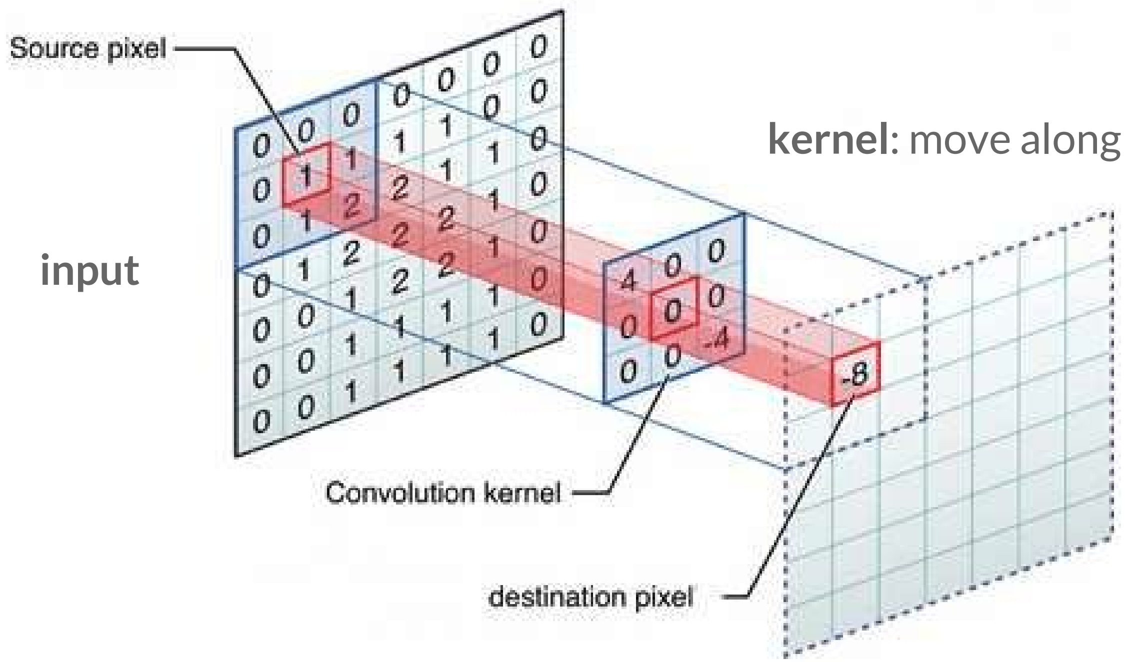
Underlying ideas:

1. *Local connectivity*: connect input only locally through small kernel
2. *Parameter sharing*: re-use (move) the kernel along the grid/image/video

Convolutional Neural Network (CNN)



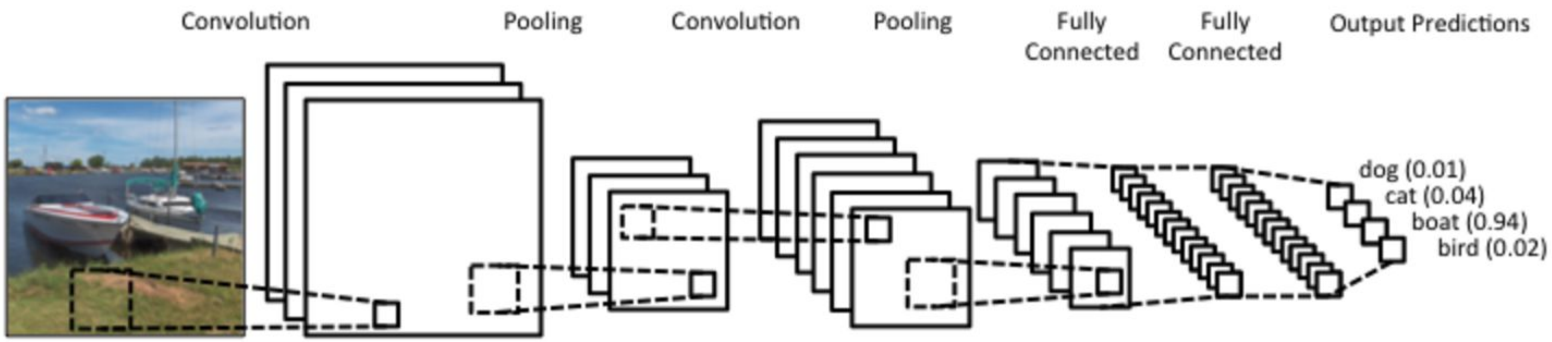
Convolutional Neural Network (CNN)



- Besides that similar to fully connected: take **linear combination with (kernel)** weights, then add **non-linearity**.
- But we **preserve the grid (2D/3D) structure** into the next layer.

Convolutional Neural Network (CNN)

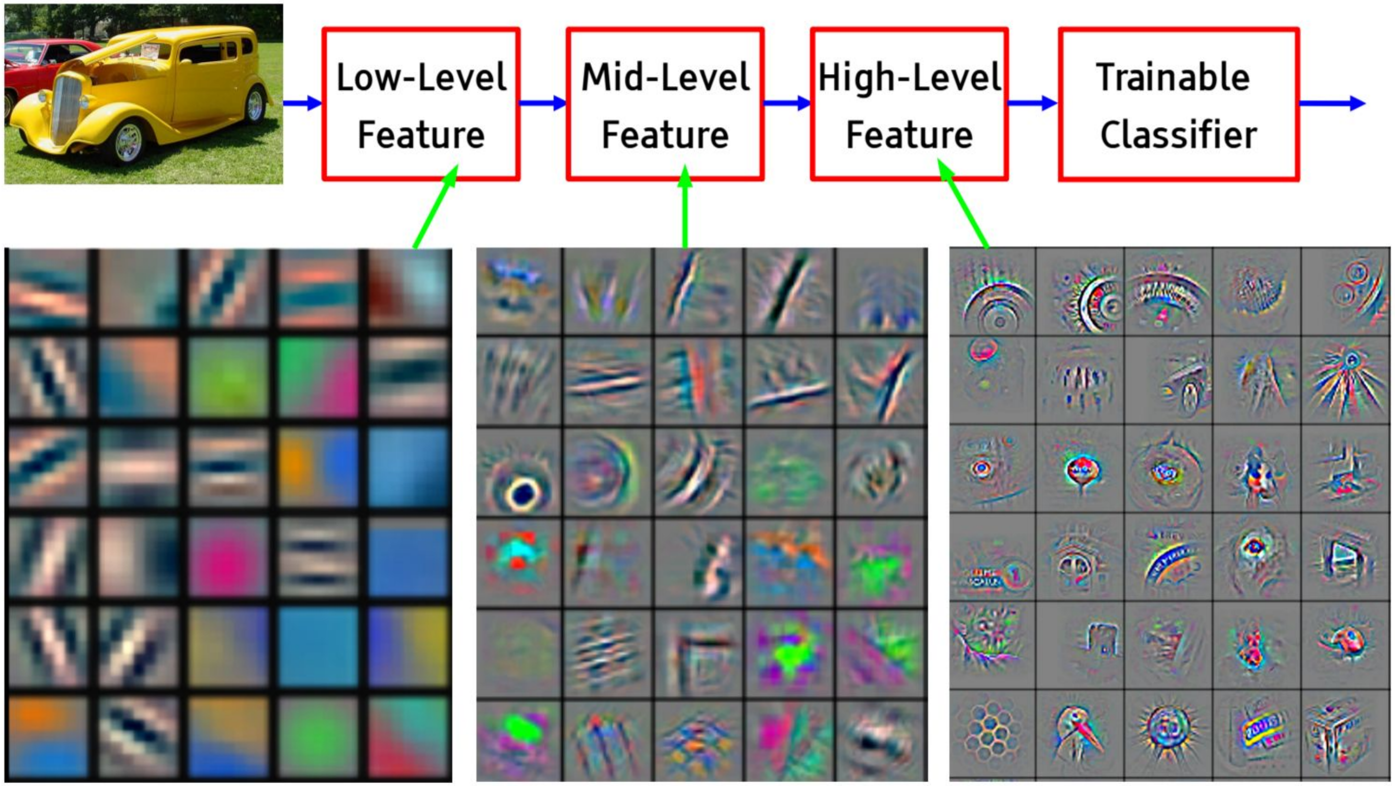
Stacking layers = **Hierarchy**



Note: The higher-up in the hierarchy, the wider the 'receptive field' in the original image.

Convolutional Neural Network (CNN)

Visualizing the Hierarchy



Zeiler, Matthew D., and Rob Fergus. Visualizing and understanding convolutional networks. 2013.

Convolutional Neural Network (CNN)

Convolution (& Pooling) = **effectively a very strong prior** on a fully connected layer:

- remove many weights (force to 0)
- tie the values of some others (parameter sharing)

Convolutional Neural Network (CNN)

Convolution (& Pooling) = **effectively a very strong prior** on a fully connected layer:

- remove many weights (force to 0)
- tie the values of some others (parameter sharing)

Q: Can you think of an example in which convolution would **not** work?

Convolutional Neural Network (CNN)

Convolution (& Pooling) = **effectively a very strong prior** on a fully connected layer:

- remove many weights (force to 0)
- tie the values of some others (parameter sharing)

Q: Can you think of an example in which convolution would **not** work?

A: When there is no spatio-temporal (i.e. grid-like) structure in the data.
For example, if **x** contains patient information (age, gender, medication, etc.), then it does not make sense to move a window along it (there is no repeating structure).

Recurrent Neural Network (RNN)

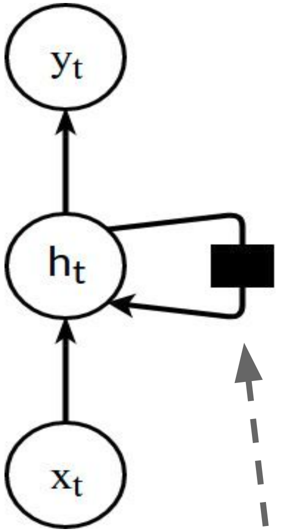
For sequential/temporal data

(text, video, audio, most real-world data is a sequence/stream)

Recurrent Neural Network (RNN)

For sequential/temporal data

(text, video, audio, most real-world data is a sequence/stream)

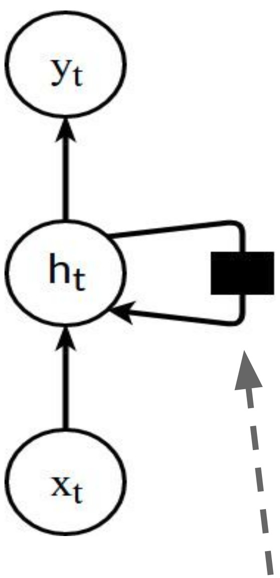


Feed information of previous step into next timestep

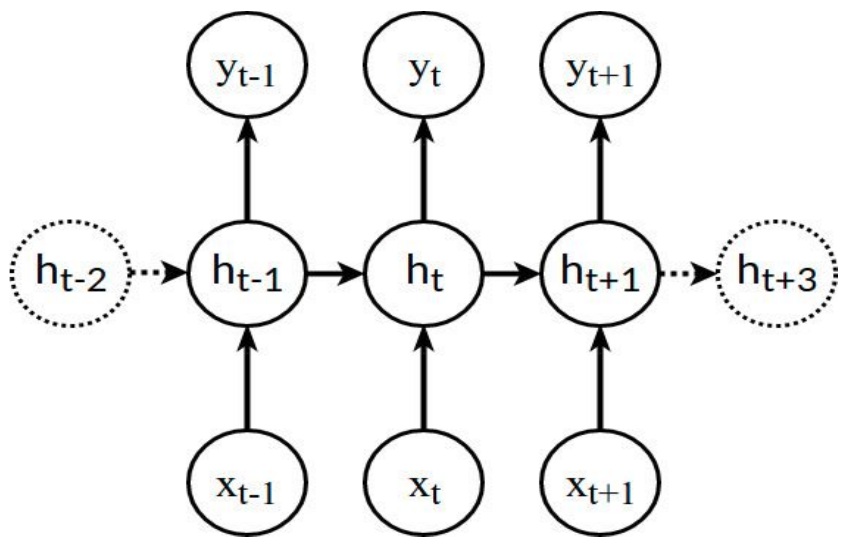
Recurrent Neural Network (RNN)

For sequential/temporal data

(text, video, audio, most real-world data is a sequence/stream)



Feed information of previous step into next timestep



Rolled out graph over time

RNN Training

Key idea:

- **Recurrent connection** between timesteps at the hidden level
- **Parameter sharing** (again): the recurrent parameters are the same at every timestep.

RNN Training

Key idea:

- **Recurrent connection** between timesteps at the hidden level
- **Parameter sharing** (again): the recurrent parameters are the same at every timestep.

But: How to train it?

RNN Training

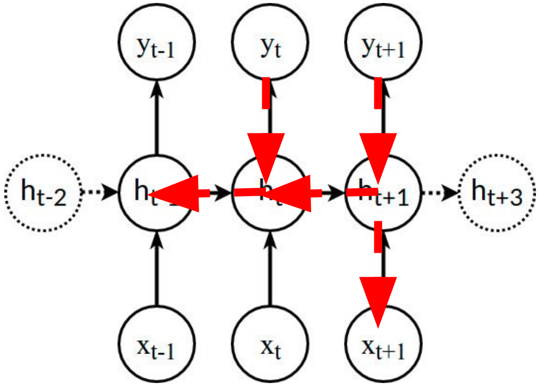
Key idea:

- **Recurrent connection** between timesteps at the hidden level
- **Parameter sharing** (again): the recurrent parameters are the same at every timestep.

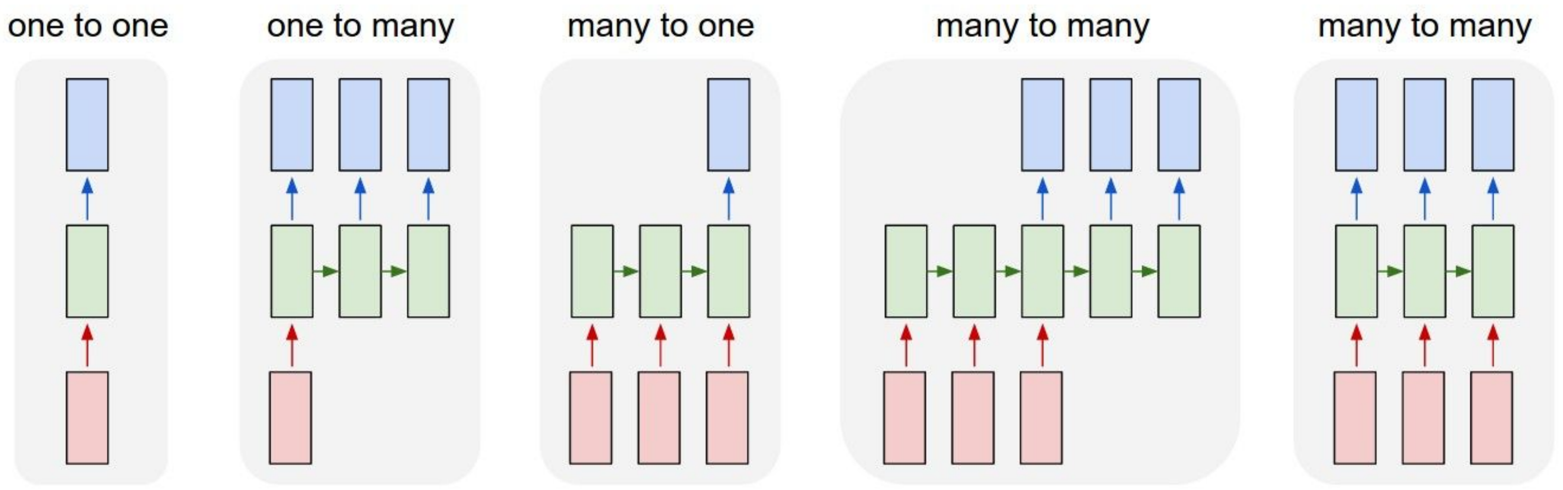
But: How to train it?

Backpropagation Through Time (BPPT)

Feed in the entire sequence - backpropagate loss through the recurrency (until the beginning)



RNN architecture variants



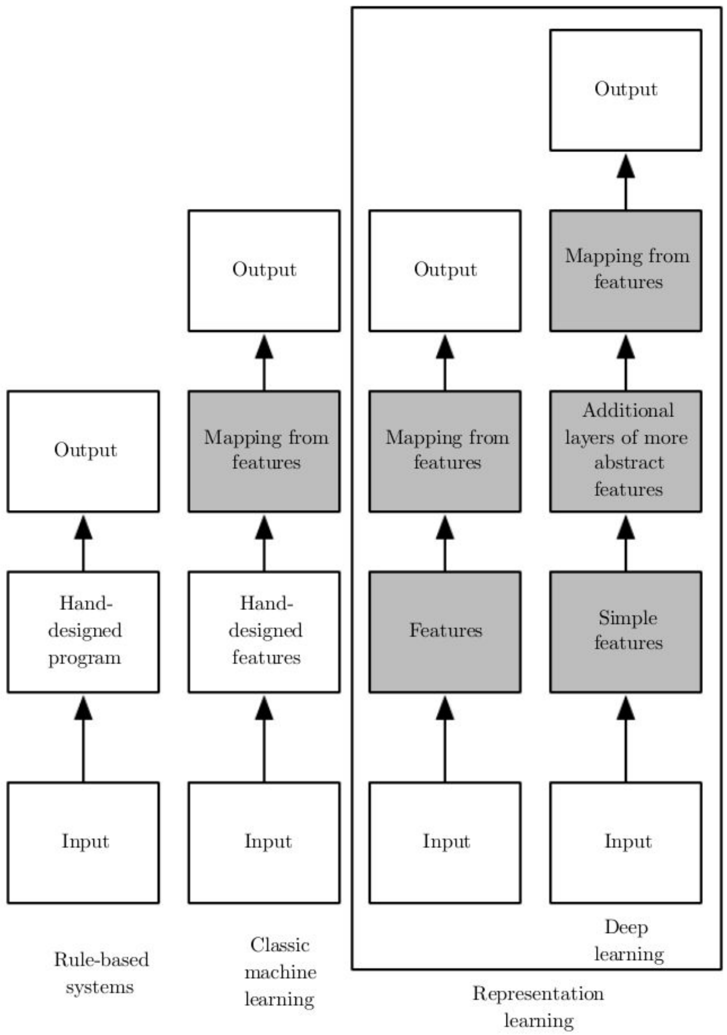
feedforward network

(e.g action classification)

(e.g. translation)

3. Deep Learning

Deep Learning



White box = hand designed

Grey box = learned

'End-to-end learning'

Deep Learning

“We have never seen machine learning or artificial intelligence technologies so quickly make an impact in industry.”

-- Kai Yu, Baidu

Deep learning =
stacking many neural network layers & training them end-to-end

(i.e. already discussed)

I. Illustration: Computer Vision

ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)

ImageNet dataset: 1.2 million pictures over 1000 classes.

$(x \rightarrow y)$

I. Illustration: Computer Vision

ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)

Year	Model name	Error rate	Details
Before 2012		**25.7%**	

I. Illustration: Computer Vision

ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)

Year	Model name	Error rate	Details
Before 2012		25.7%	
2012	**AlexNet**	**15.4%**	7 layers, GPU's, Relu activation

I. Illustration: Computer Vision

ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)

Year	Model name	Error rate	Details
Before 2012		25.7%	
2012	AlexNet	15.4%	7 layers, GPU's, Relu activation
2013	ZF Net	11.2%	Visualization by deconvolution
2014	VGG Net	7.3%	Deep (19 layers)
2015	GoogleNet	6.7%	Very deep (100 layers), Inception module
2015	**ResNet**	**3.4%**	Residual connections

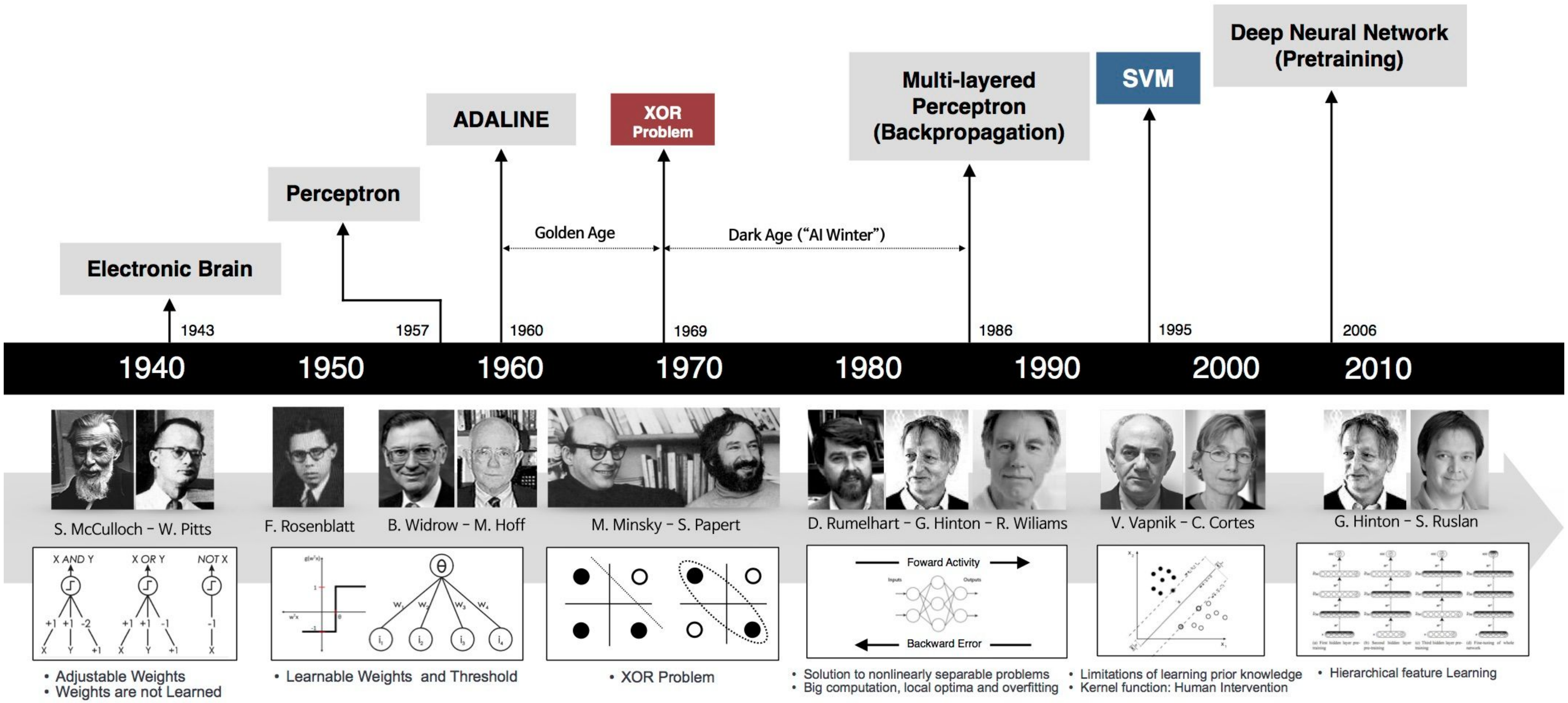
I. Illustration: Computer Vision

ILSVRC (ImageNet Large-Scale Visual Recognition Challenge)

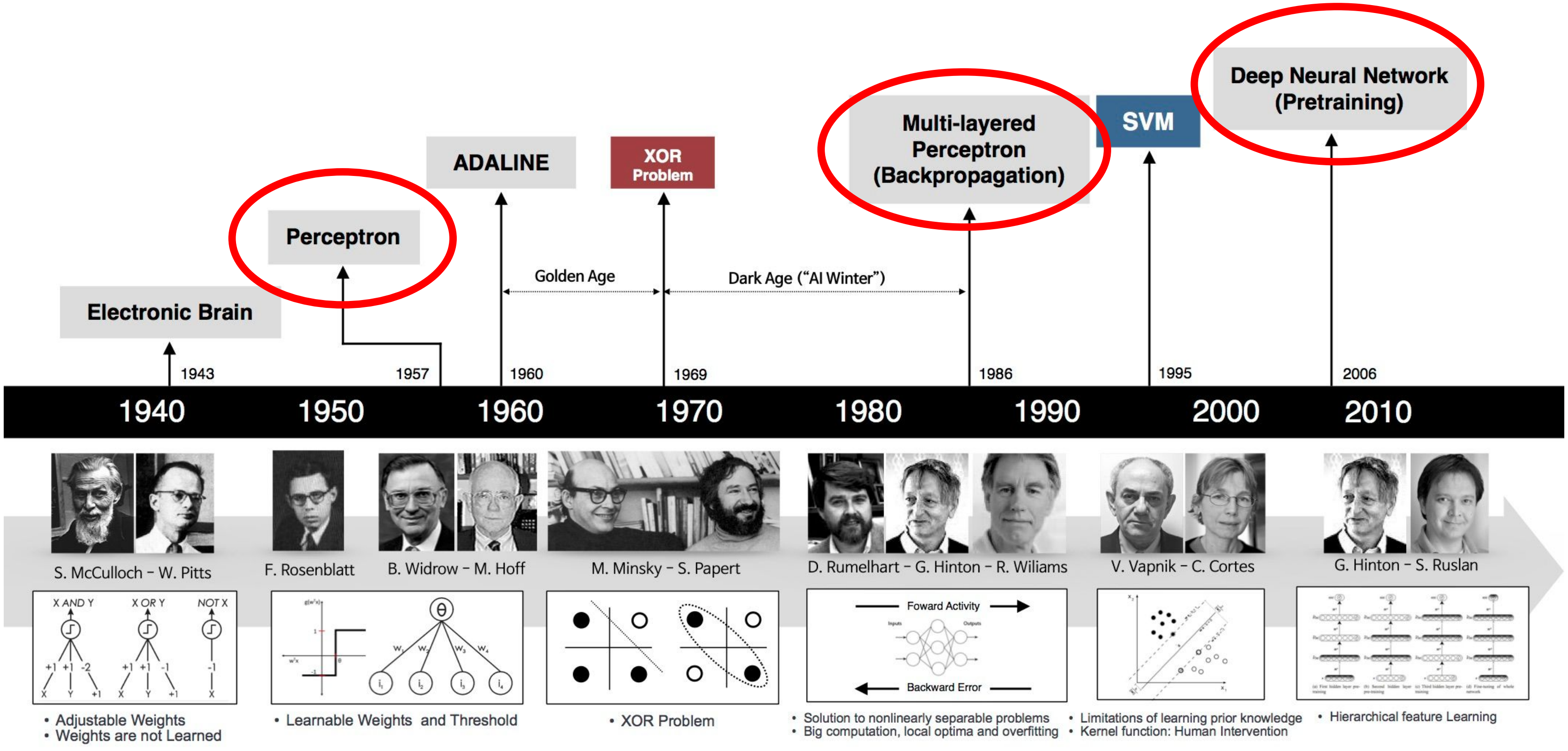
Year	Model name	Error rate	Details
Before 2012		25.7%	
2012	AlexNet	15.4%	7 layers, GPU's, Relu activation
2013	ZF Net	11.2%	Visualization by deconvolution
2014	VGG Net	7.3%	Deep (19 layers)
2015	GoogleNet	6.7%	Very deep (100 layers), Inception module
2015	ResNet	3.4%	Residual connections

Human **5~10%**

II. History of Neural Networks

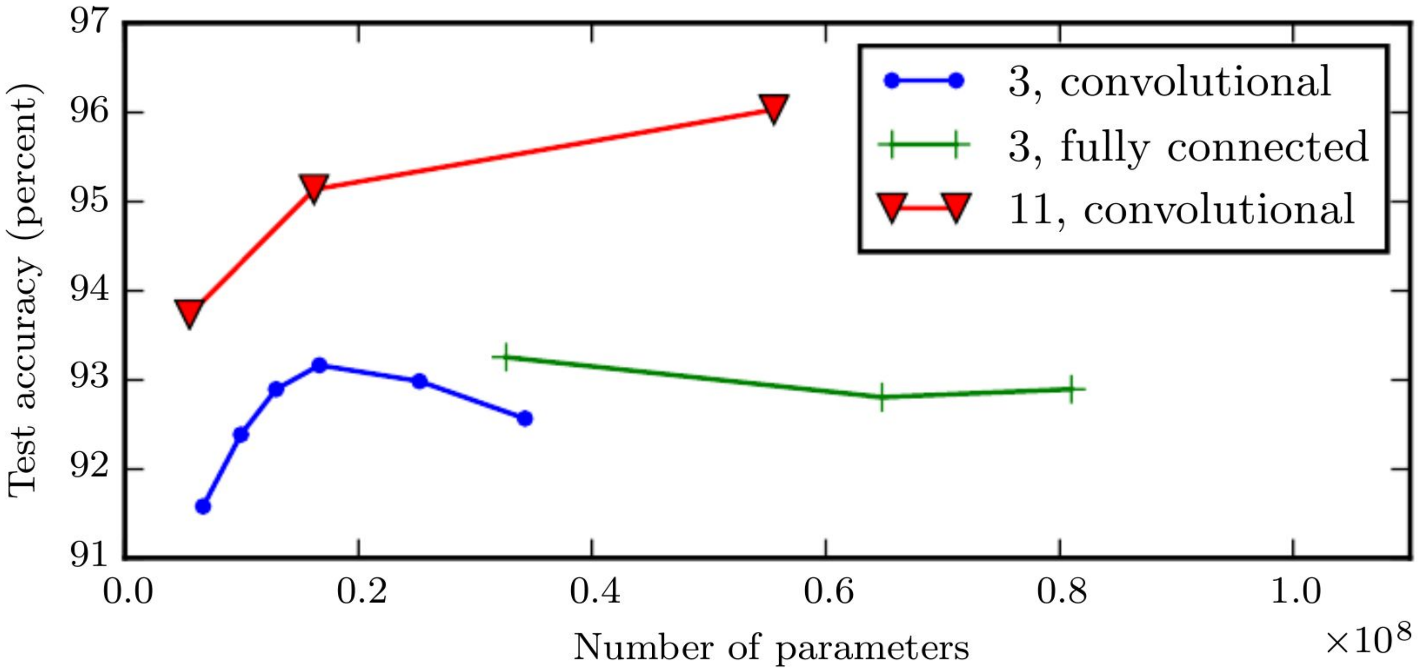


II. History of Neural Networks

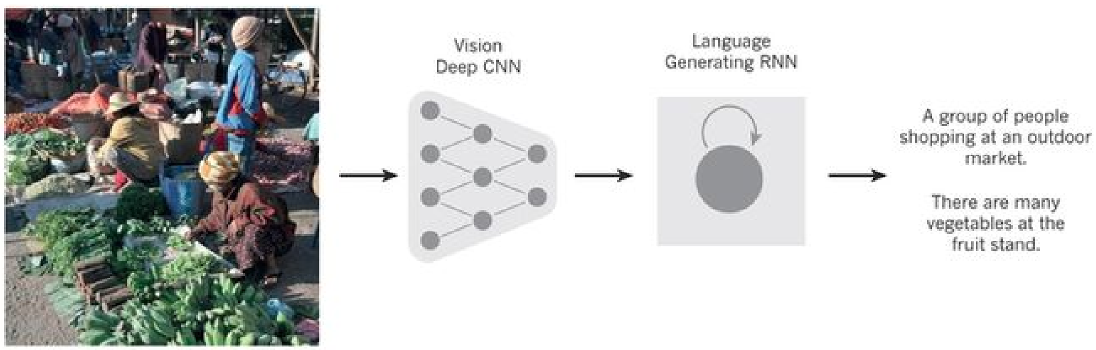


III. The Benefit of Depth

Depth is beneficial beyond just giving more parameters



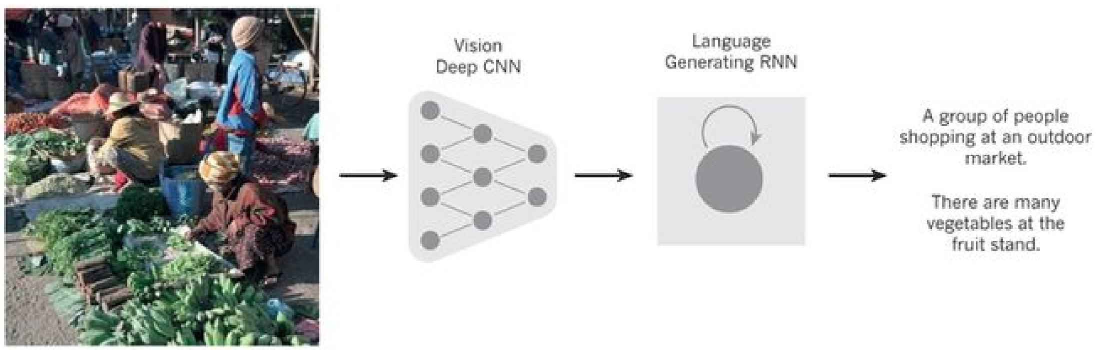
IV. Combining Layers



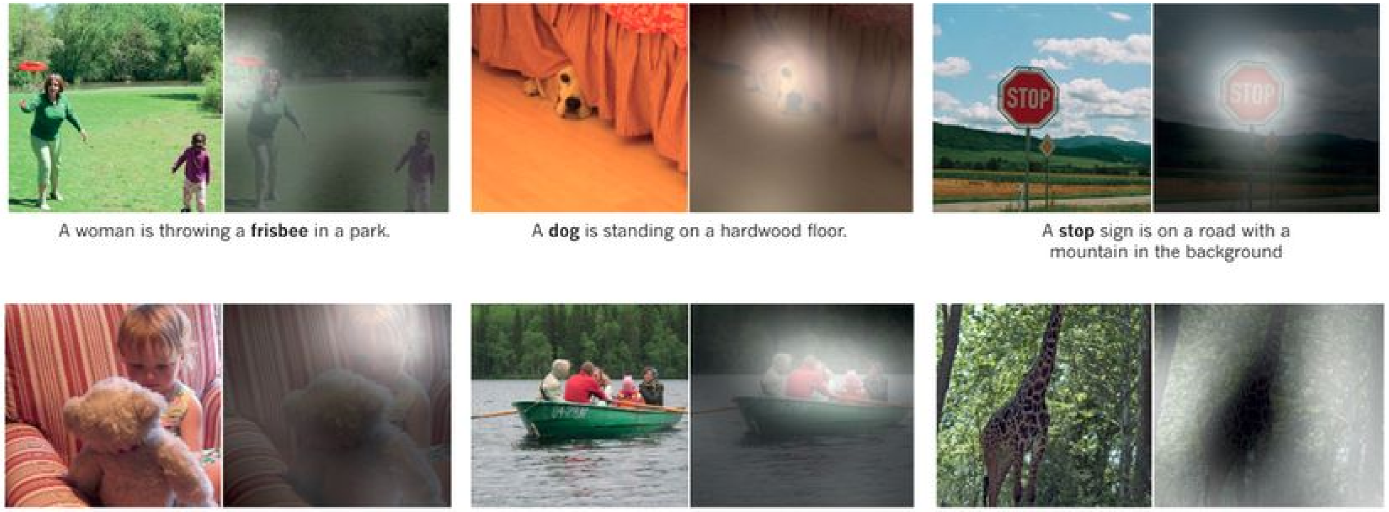
CNN
+
RNN

Karpathy A. Fei-Fei L. Deep Visual-Semantic Alignments for Generating Image Descriptions. 2015.

IV. Combining Layers



CNN
+
RNN



A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

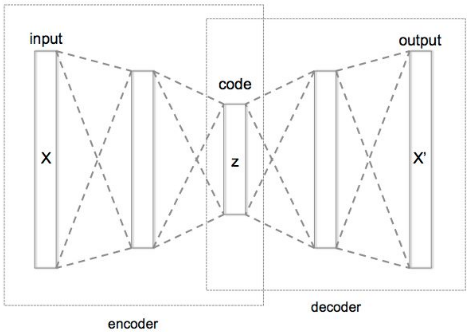
A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

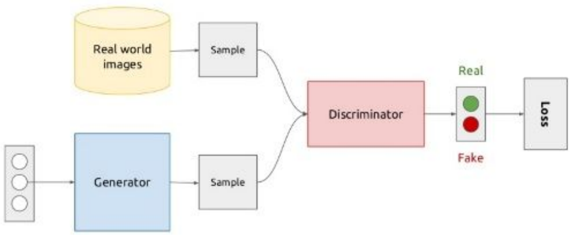
A giraffe standing in a forest with **trees** in the background.

Karpathy A. Fei-Fei L. Deep Visual-Semantic Alignments for Generating Image Descriptions. 2015.

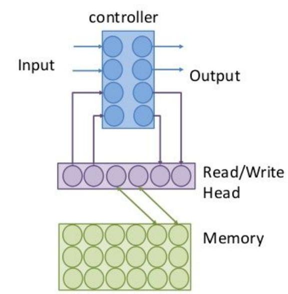
V. Deep Learning Research



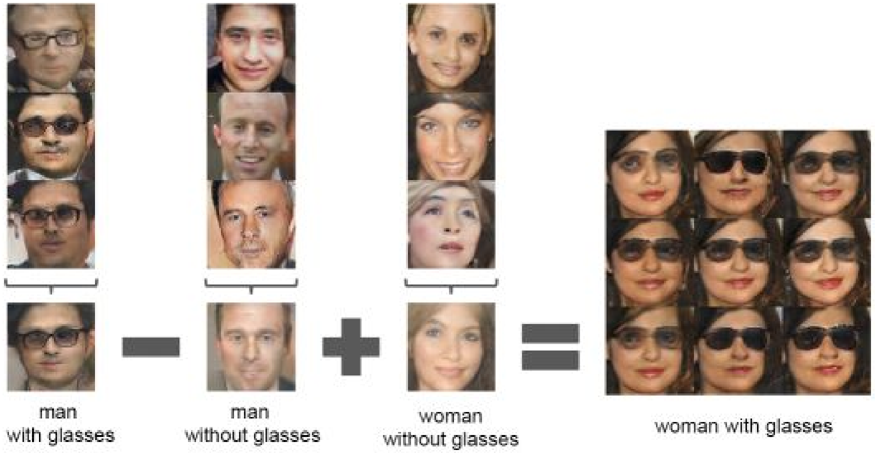
Autoencoders



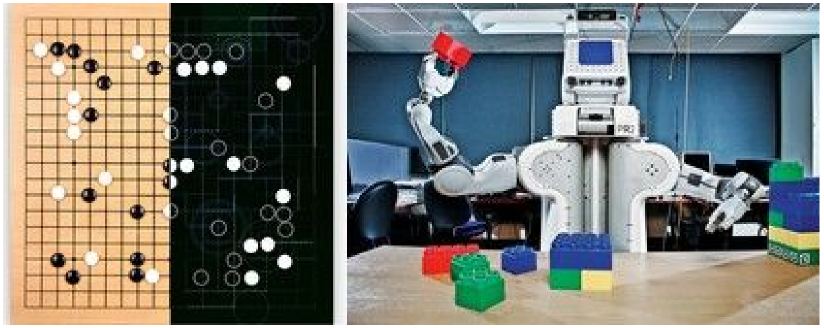
Adversarial Training



Neural Turing Machines



Deep Generative Models



Deep Reinforcement Learning

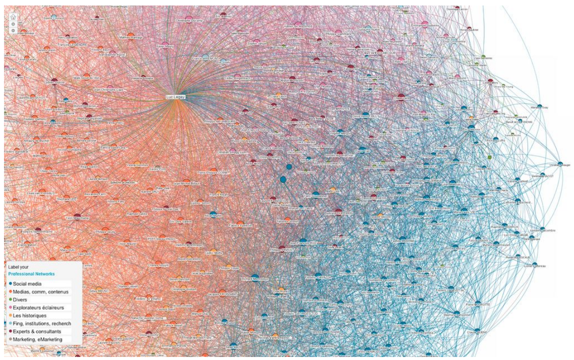
VI. The other pillars of deep learning

(Apart from the algorithms/math discussed in this lecture)

VI. The other pillars of deep learning

(Apart from the algorithms/math discussed in this lecture)

1. Data



2. Computation

3. Software

TensorFlow

theano

torch

Reading Material

1. **Lecture Notes**

2. **Deep learning book (**Free PDF: http://www.deeplearningbook.org/)

<u>Read</u>:

Fully connected layers: 6.0-6.1, 6.3

Loss functions: 6.2

Numerical Optimization: 4.0-4.3, 5.9, 6.5.1-4, 8.1-8.1.1

CNN: 9.0-9.4

RNN: 10.0,10.1,10.2.0,10.2.2

Deep learning: 1.0 (+ figure 1.5)