ML @ URL

Episode 1

Intro to deep learning

Yandex Data Factory

LAMBDA

British Hedgehog Preservation Society

Linear Regression

Model:

$$X \longrightarrow Wx + b \longrightarrow Y^{pred}$$

Objective function:

$$L = \sum_i (y_i - y_i^{pred})^2$$

Optimization (exact):

$$w = (X^T X)^{-1} X^T y$$

Linear Regression

Model:

$$X \longrightarrow Wx + b \longrightarrow Y^{pred}$$

Objective function:

$$L = \sum_i (y_i - y_i^{pred})^2$$

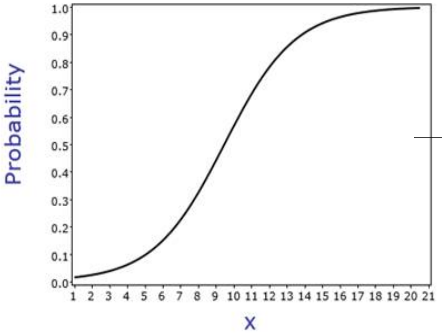
Optimization (iterative):

$$w_0 \leftarrow 0$$

$$w_{i+1} \leftarrow w_i - \alpha \frac{\partial L}{\partial W}$$

$$\frac{\partial L}{\partial W} = \sum_i -2x(y_i - (wx_i + b))$$

Logistic Regression

X → Wx +b →  → P(y)

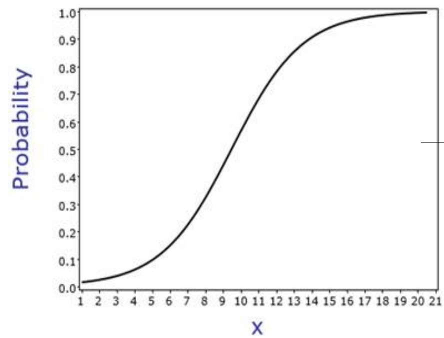
$$P(y) = \sigma(Wx + b)$$

Objective function ?

Logistic Regression

Model:

X → Wx +b →



→ P(y|x)

Objective function:

$$L = -\sum_i y_i \log P(y|x_i) + (1 - y_i) \log (1 - P(y|x_i))$$

Optimization (iterative):

You guessed it!

Logistic Regression

Model:

$$X \longrightarrow \begin{matrix} a_{[y=a]} = W_a x + b_a \\ a_{[y=b]} = W_b x + b_b \\ a_{[y=c]} = W_c x + b_c \end{matrix} \longrightarrow \frac{e^{a_{[y=class]}}}{\sum_j e^{a_{[y=j]}}} \longrightarrow \begin{matrix} P(y=a|X) \\ P(y=b|X) \\ P(y=c|X) \end{matrix}$$

Objective function:

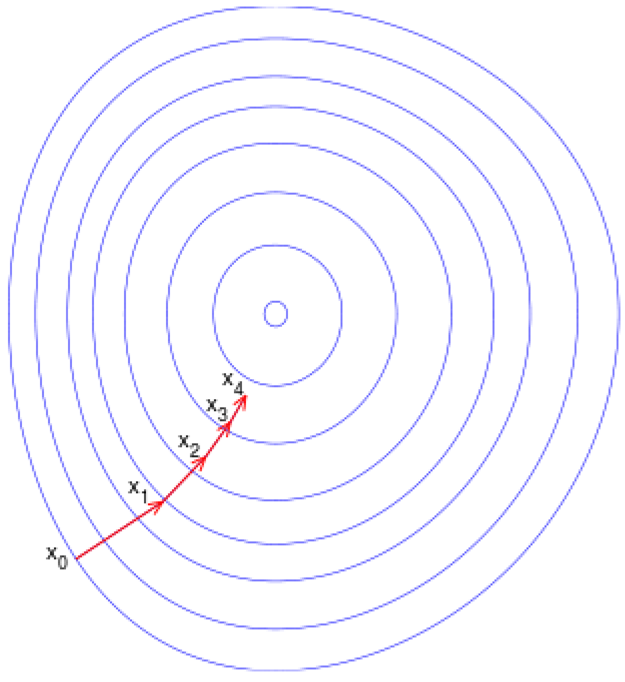
$$L = -\sum_i \sum_{class} [y_i = class] \log P(y = class | x_i)$$

Gradient descent

Update:

$$w_{i+1} \leftarrow w_i - \alpha \frac{\partial L}{\partial w}$$

- a – learning rate a<<1
- L – loss function



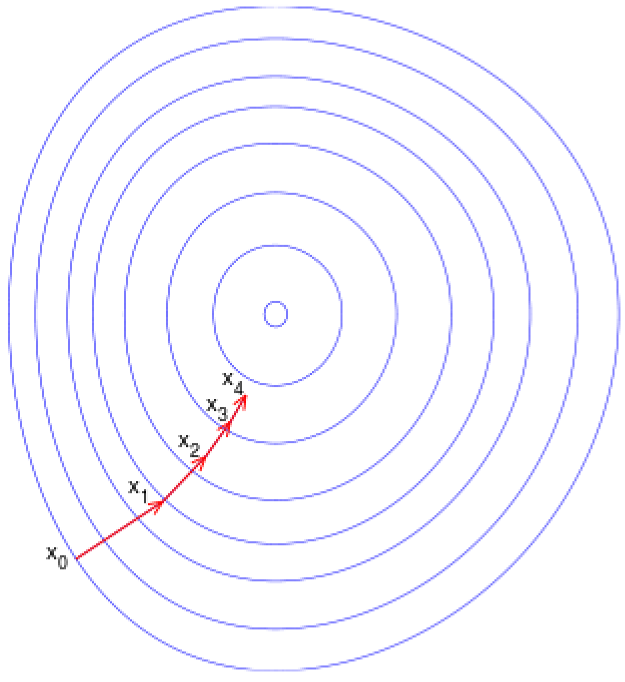
Can we do better?

Gradient descent

Update:

$$w_{i+1} \leftarrow w_i - \alpha \frac{\partial L}{\partial w}$$

- a – learning rate a<<1
- L – loss function



Can we do better?

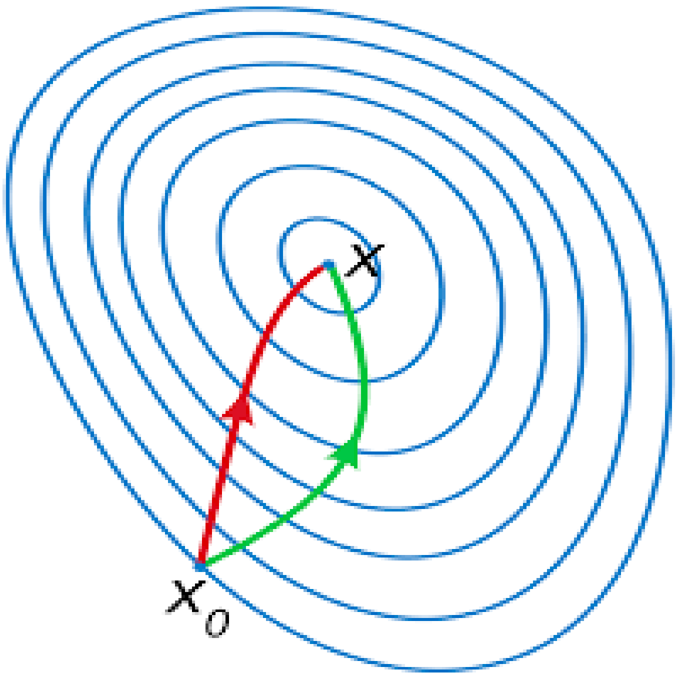
Newton-Raphson

Parameter update

$$w_{i+1} \leftarrow w_i - \alpha H_L^{-1} \frac{\partial L}{\partial w}$$

Hessian:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$



Red: Newton-Raphson
Green: gradient descent

Any drawbacks?

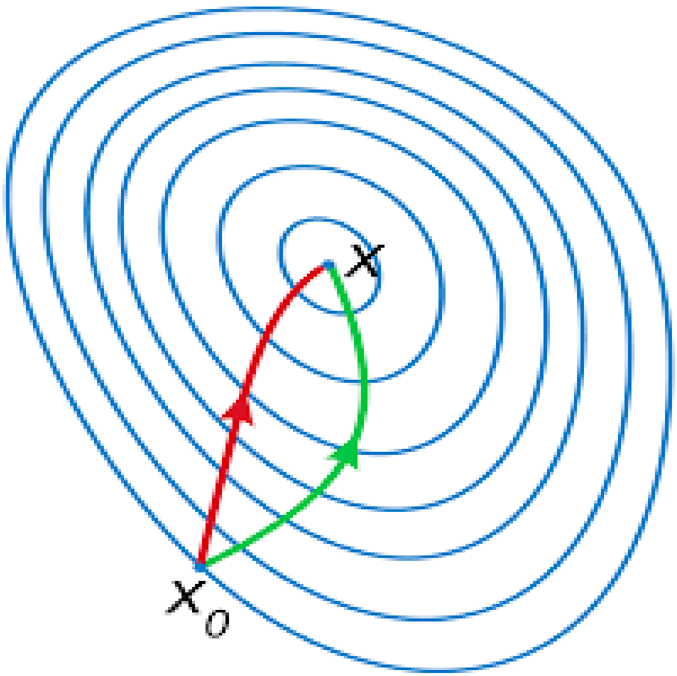
Newton-Raphson

Parameter update

$$w_{i+1} \leftarrow w_i - \alpha H_L^{-1} \frac{\partial L}{\partial w}$$

Hessian:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$



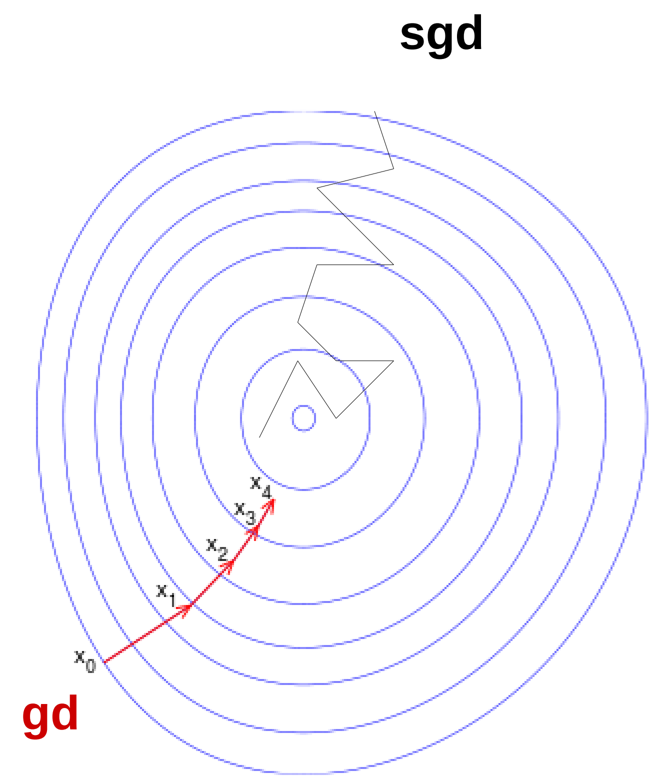
Red: Newton-Raphson
Green: gradient descent

Quadratic time/memory!

Stochastic gradient descent

Loss function is mean over all data samples.

Approximate with 1 or few random samples.



Update:

$$w_{i+1} \leftarrow w_i - \alpha E \frac{\partial L}{\partial w}$$

- E – expectation
- Learning rate should decrease

SGD with momentum

Idea: move towards "overall gradient direction",
Not just current gradient.

$$w_0 \leftarrow 0 \,;\, v_0 \leftarrow 0$$

$$\nu_{i+1} \leftarrow \alpha \frac{\partial L}{\partial w} + \mu \nu_i$$

$$w_{i+1} \leftarrow w_i - \nu_{i+1}$$

Helps for noisy gradient / canyon problem

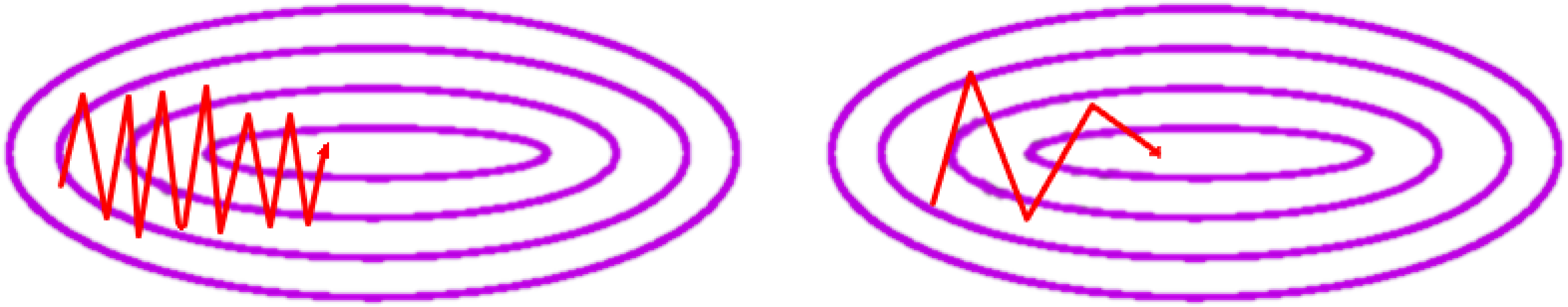
SGD with momentum

Idea: move towards “overall gradient direction”,
Not just current gradient.

$$w_0 \leftarrow 0\,;\, v_0 \leftarrow 0$$

$$\nu_{i+1} \leftarrow \alpha \frac{\partial L}{\partial w} + \mu \nu_i$$

$$w_{i+1} \leftarrow w_i - \nu_{i+1}$$



AdaGrad

Idea: decrease learning rate individually for each parameter in proportion to sum of it's gradients so far.

$$G_t = \sum_{\tau=1}^{t} \left[\frac{\partial L}{\partial w} \right]^2$$

"Total update path length"
(for each parameter)

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \frac{\partial L}{\partial w}$$

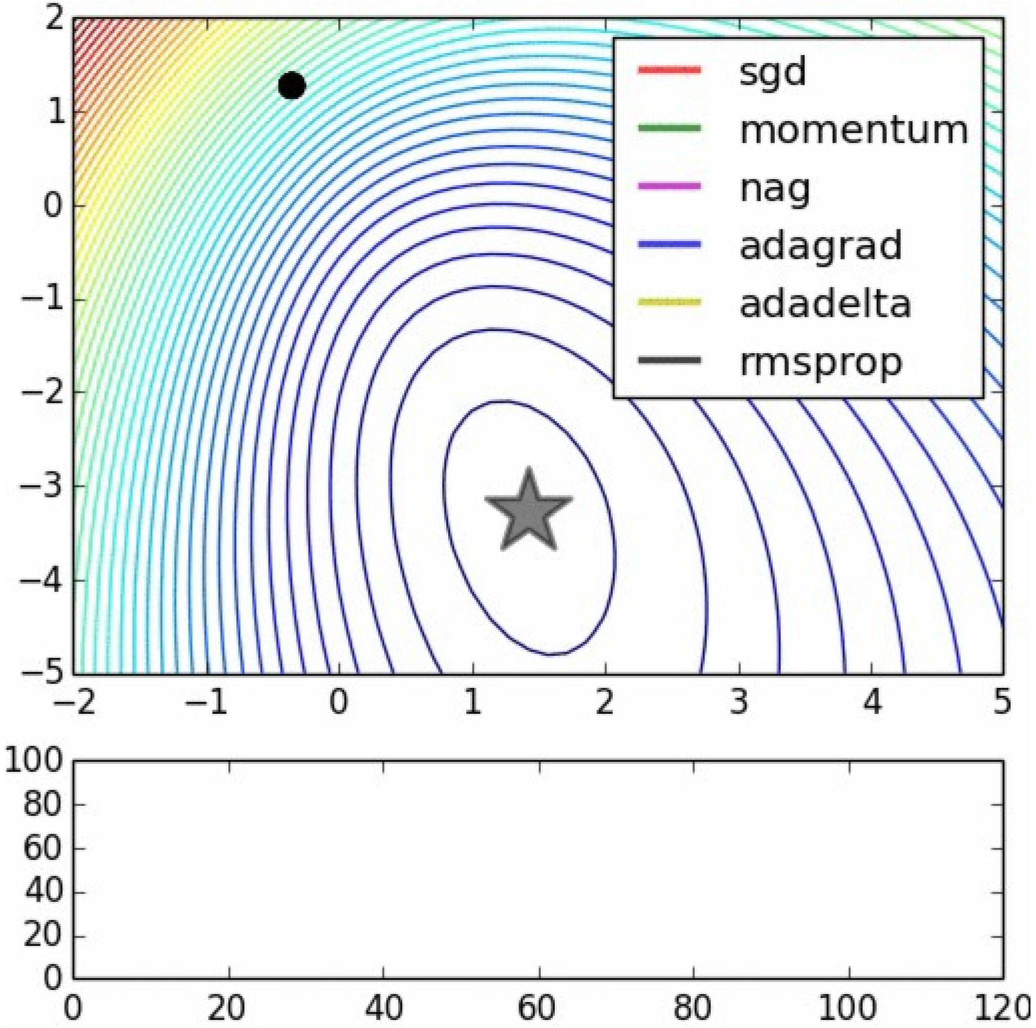
RMSProp

Idea: make sure all gradient steps have approximately same magnitude (by keeping moving average of magnitude)

$$ms_{t+1} = \gamma \cdot ms_t + (1-\gamma) \| \frac{\partial L}{\partial w} \|^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{ms + \epsilon}} \frac{\partial L}{\partial w}$$

Alltogether



Moar stuff

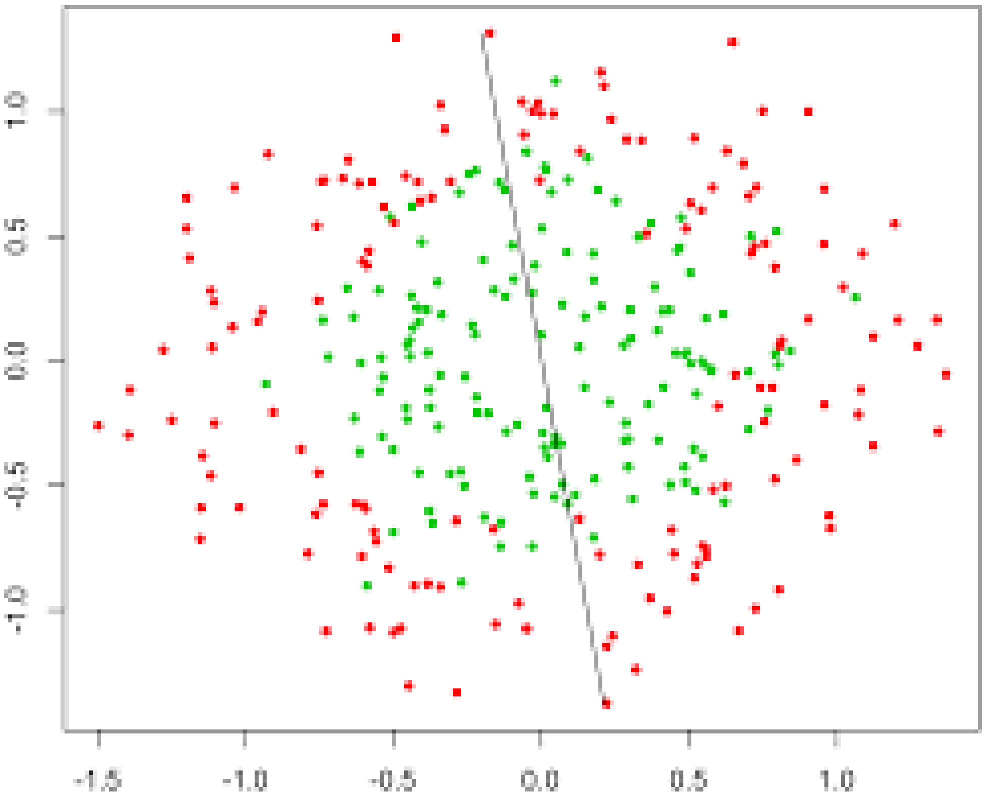
Without Hessian

- Adadelta ~ adagrad with window
- Adam ~ rmsprop + momentum
- Nesterov-momentum
- Hessian-free (narrow)
- Conjugate gradients

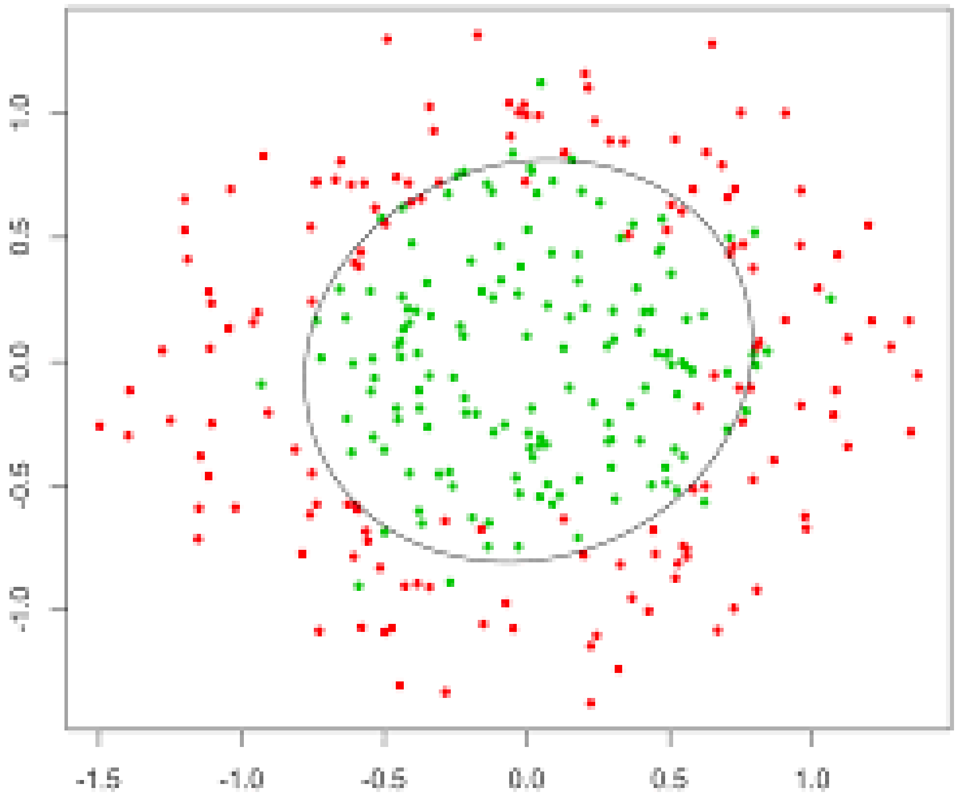
Estimate inverse Hessian

- BFGS
- L-BFGS
- ****-BFGS

Nonlinear dependencies



What we have



What we want

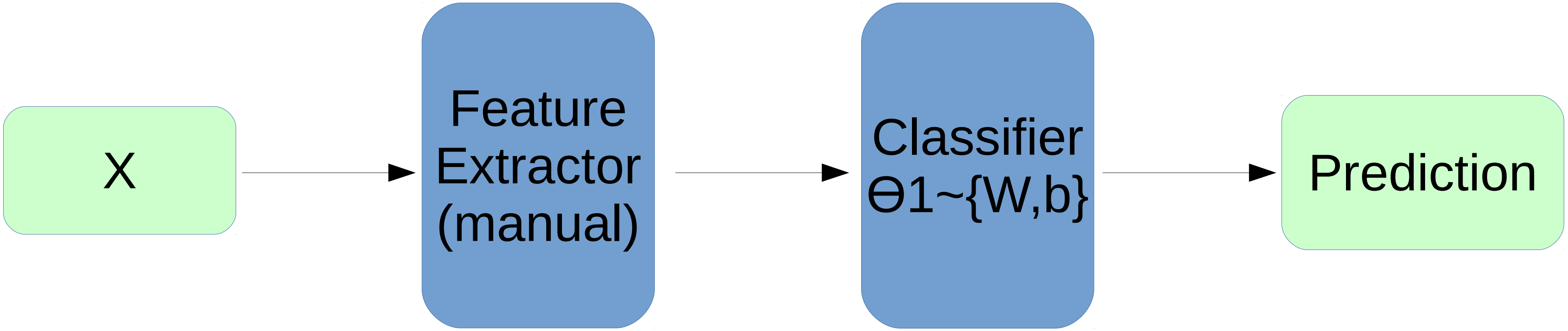
- How to get that?

Feature extraction

Loss, for example:

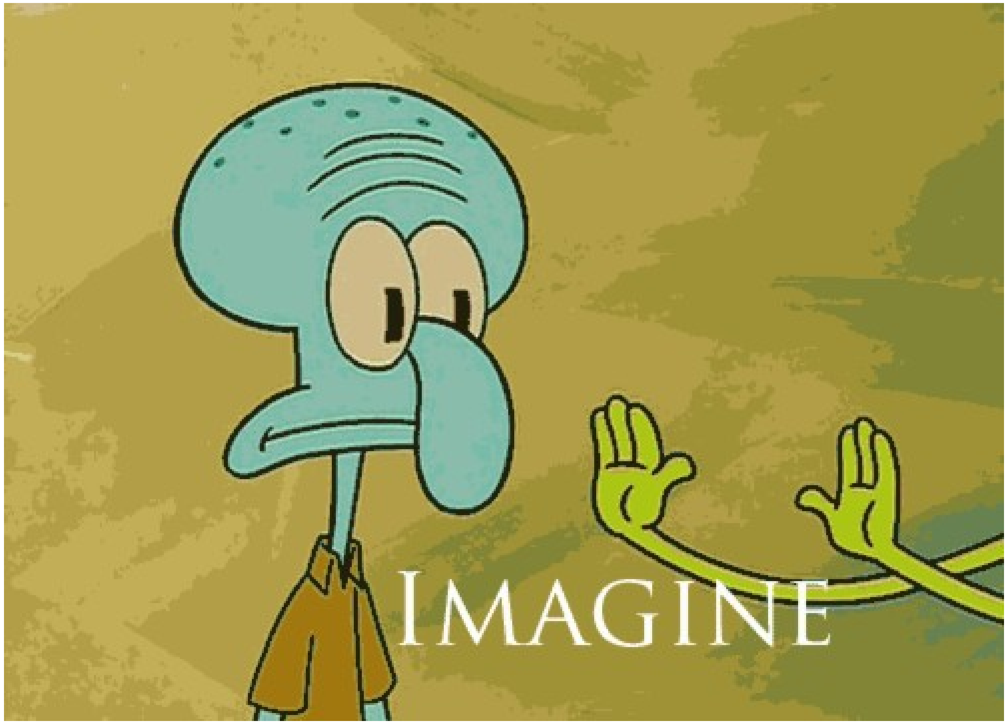
$$L = -\sum_i y_i \log P(y|x_i) + (1 - y_i) \log(1 - P(y|x_i))$$

Model:



Training:

$$\underset{\theta_1}{argmin}\, L(y, P(y|x))$$



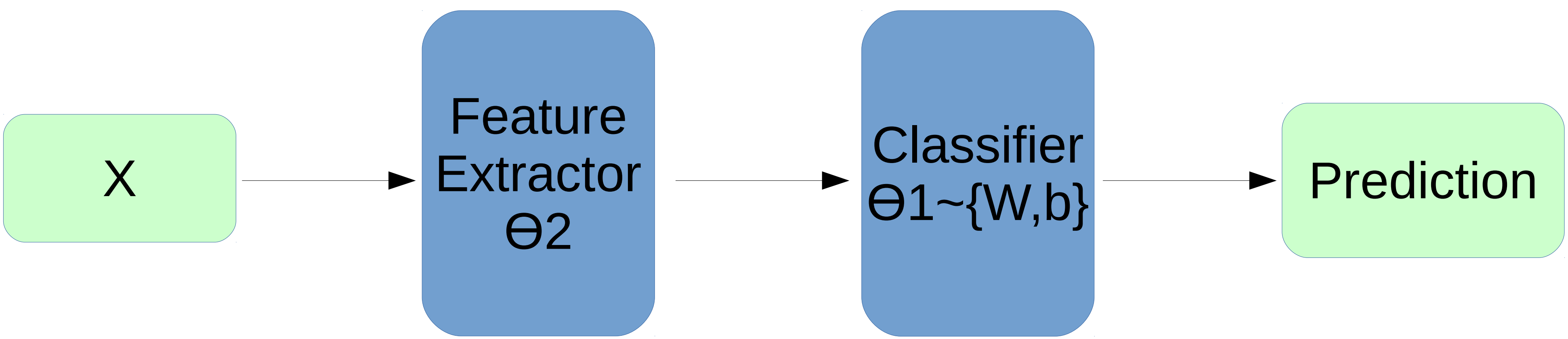
Features would tune to your problem automatically!

What do we want, exactly?

Loss, for example:

$$L = -\sum_i y_i \log P(y|x_i) + (1 - y_i) \log(1 - P(y|x_i))$$

Model:



Training: ?

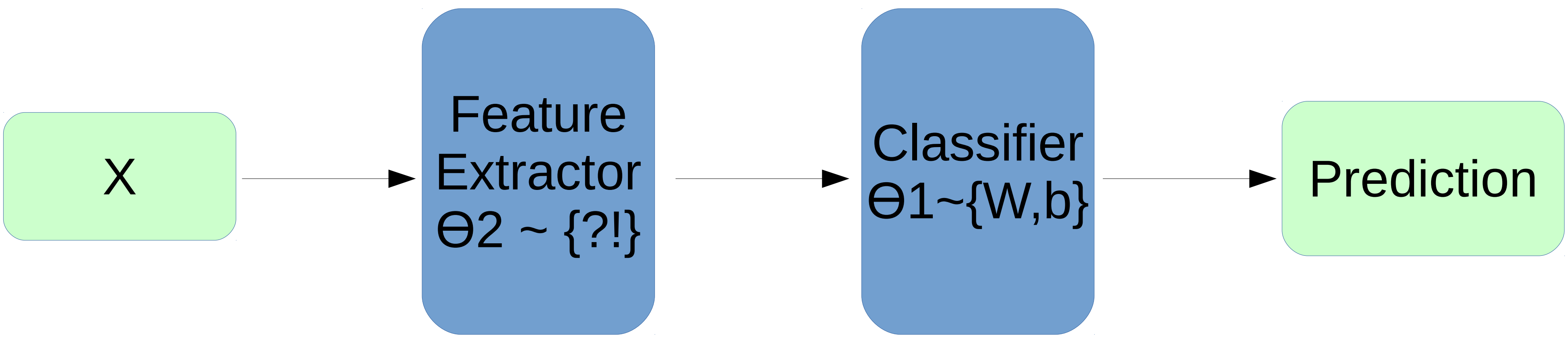
$$\underset{\theta_1}{\operatorname{argmin}} L(y, P(y|x))$$

What do we want, exactly?

Loss, for example:

$$L = -\sum_i y_i \log P(y|x_i) + (1 - y_i) \log(1 - P(y|x_i))$$

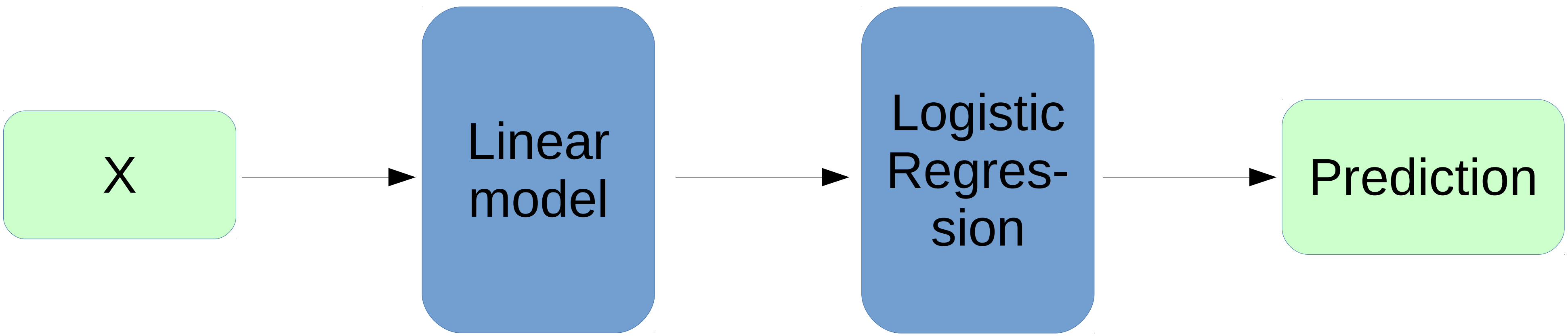
Model:



Gradients: $\underset{\theta_2}{argmin}\, L(y, P(y|x))$ $\underset{\theta_1}{argmin}\, L(y, P(y|x))$

Try linear

Model:

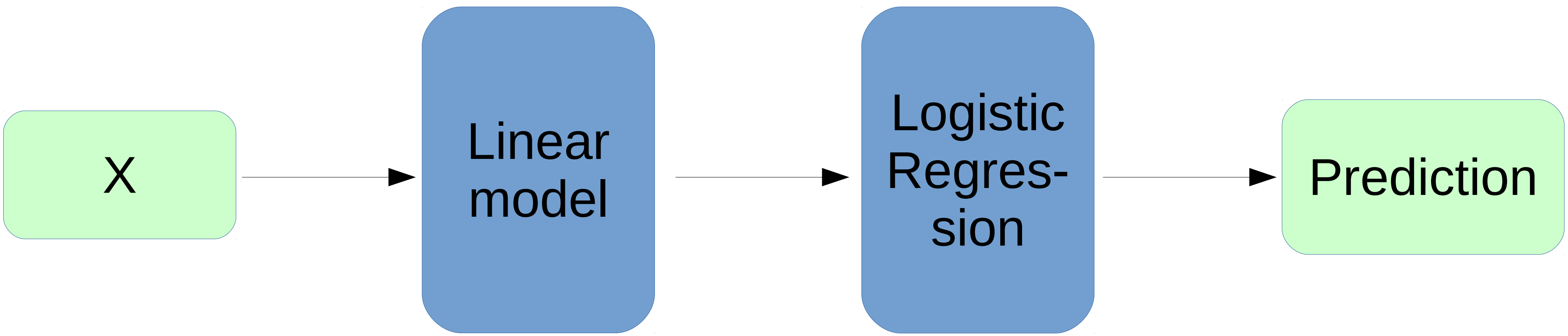


$$h_j = \sum_{i} w_{ij}^h x_i + b_j^h \qquad j \in \{1,2,\dots,n\}$$

$$y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

Try linear

Model:



$$h_j = \sum_{i} w_{ij}^{h} x_i + b_j^{h}$$
$$j \in \{1, 2, \ldots, n\}$$

$$y_{pred} = \sigma\left(\sum_{j} w_j^{o} h_j + b^{o}\right)$$

Output:

$$P(y|x) = \sigma\left(\sum_{j} w_j^{o} \left(\sum_{i} w_{ij}^{h} x_i + b_j^{h}\right) + b^{o}\right)$$

Is it any better than logistic regression?

Try linear

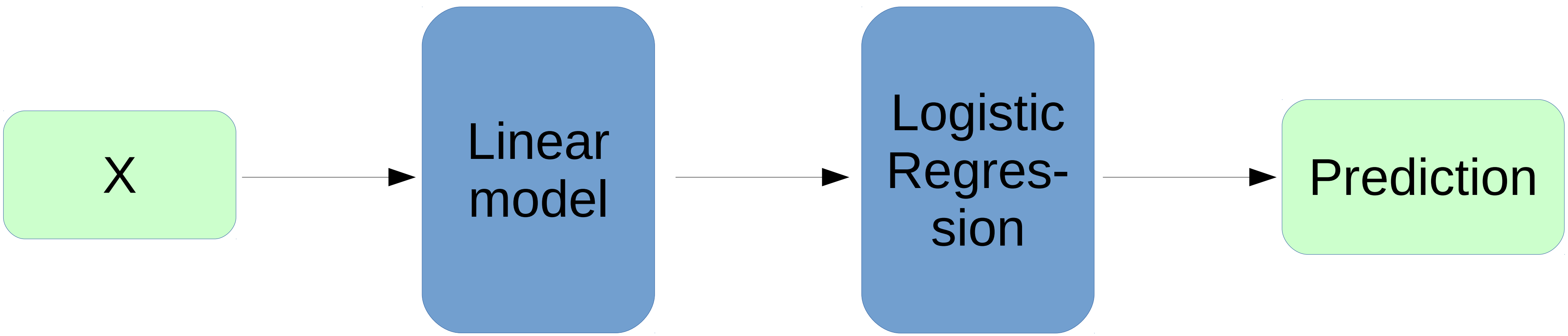
$$P(y|x) = \sigma\left(\sum_j w_j^o \left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

$$w'_i = \sum_j w_j^o w_{ij}^h \qquad b' = \sum_j w_j^o b_j^h + b^o$$

$$P(y|x) = \sigma\left(\sum_i w'_i x_i + b'\right)$$

Try linear

Model:



$$h_j = \sum_i w_{ij}^h x_i + b_j^h \qquad y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

$$j \in \{1, 2, \ldots, n\}$$

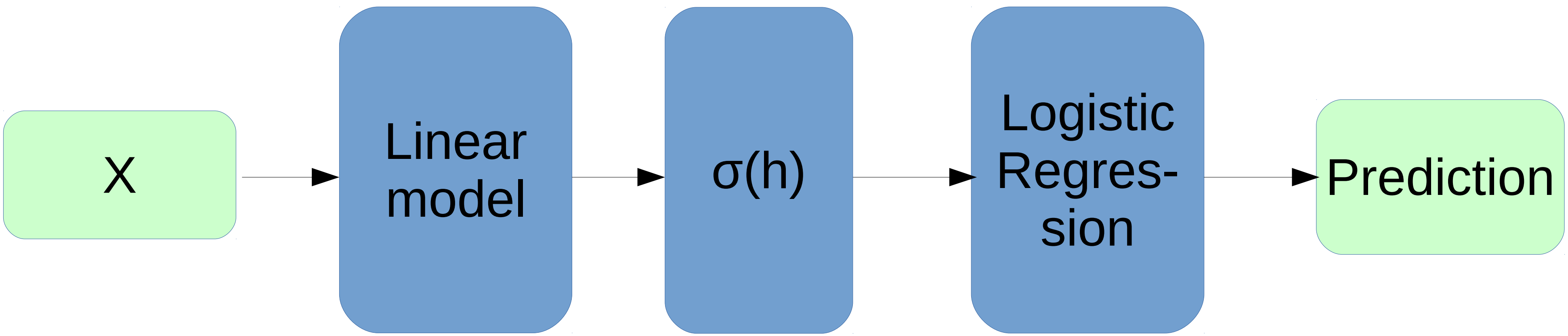
Output:

$$P(y|x) = \sigma\left(\sum_j w_j^o \left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

Is it any better than logistic regression?

Nonlinearity

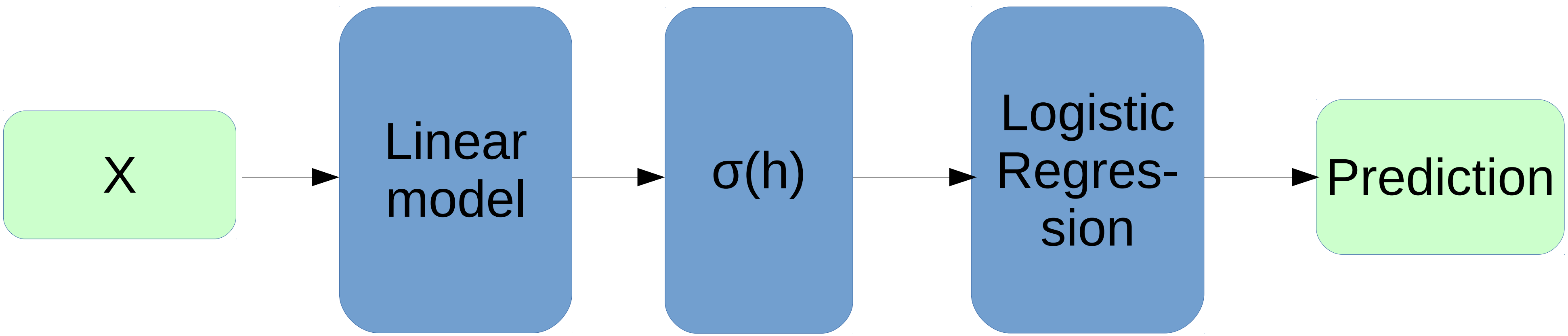
Model:



$$h_j = \sigma\left(\sum_{\substack{i \\ j \in \{1,2,\ldots,n\}}} w_{ij}^h x_i + b_j^h\right) \qquad y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

Nonlinearity

Model:



$$h_j = \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right)$$
$$j \in \{1,2,\ldots,n\}$$

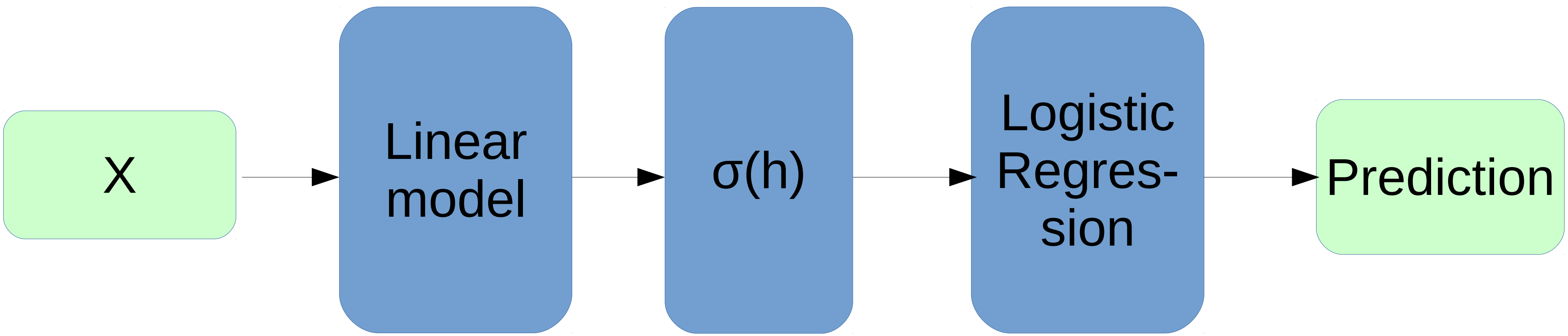
$$y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

Output:

$$P(y|x) = \sigma\left(\sum_j w_j^o \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

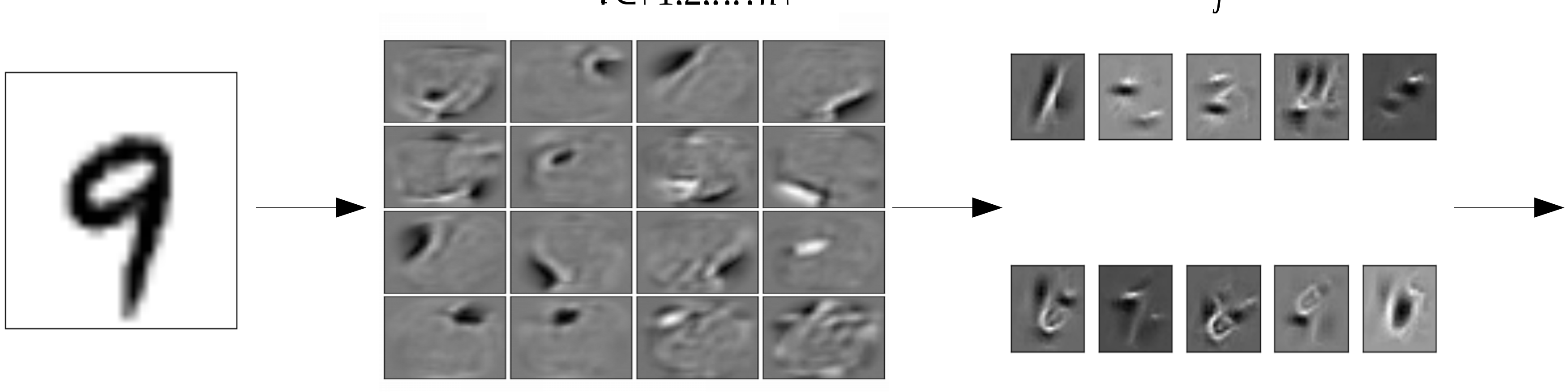
Nonlinearity

Model:



$$h_j = \sigma\left(\sum_{i \in \{1.2.\ldots.n\}} w_{ij}^{h} x_i + b_j^{h}\right)$$

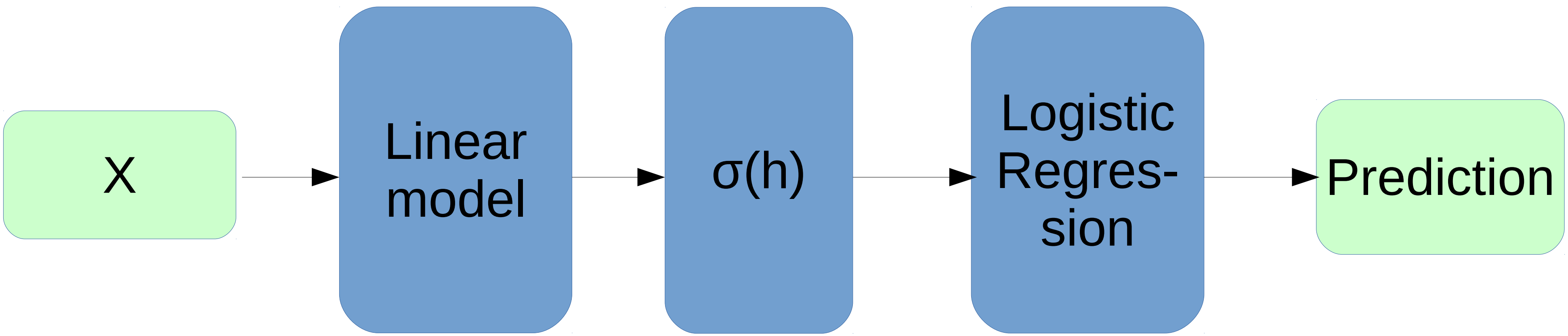
$$y_{pred} = \sigma\left(\sum_{j} w_j^{o} h_j + b^{o}\right)$$



P(“1”)
P(“2”)
...
P(“9”)

Nonlinearity

Model:



Output:

$$P(y|x)=\sigma\left(\sum_j w_j^o \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

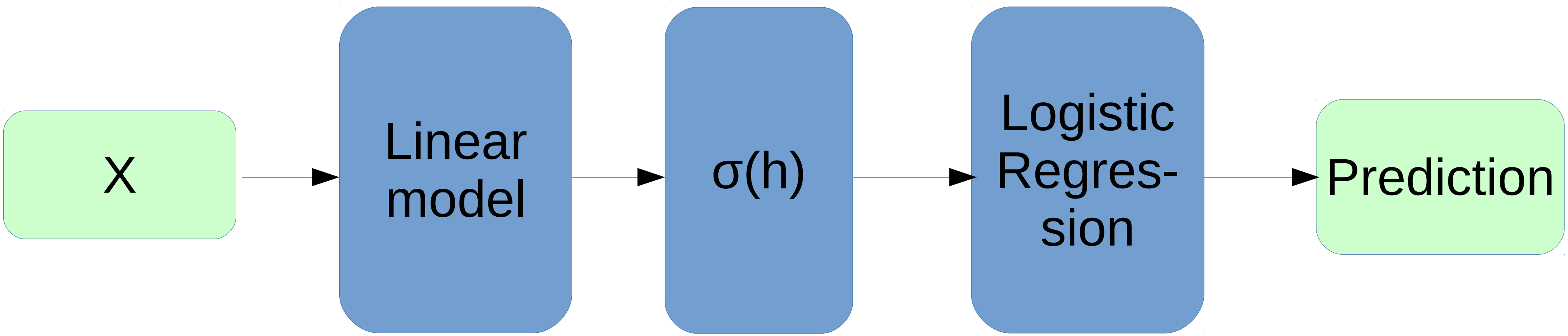
Training:

$w := w - \alpha$

Gradient of what?
w.r.t. what?

Nonlinearity

Model:



Output:

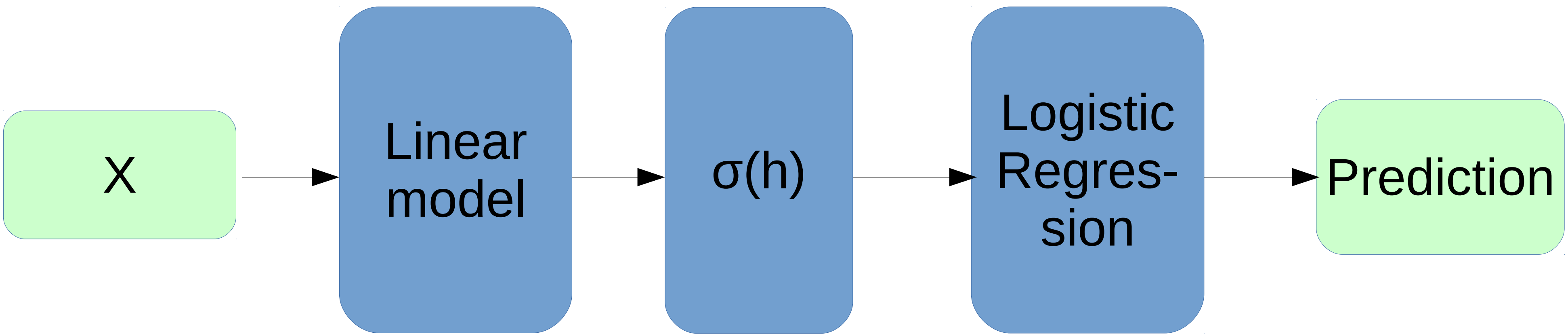
$$P(y|x)=\sigma\left(\sum_j w_j^o \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

Training:

$$w := w - \alpha \frac{\partial \, E_{x_i, y_i} Loss(y, P(y|x))}{\partial w}$$

Nonlinearity

Model:



Output:

$$P(y|x)=\sigma\left(\sum_j w_j^o \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

Training:

$$w := w - \alpha \frac{\partial \, E_{x_i, y_i} - \log P_w(y_i|x_i)}{\partial w}$$

Losses:
(task-dependent)
crossentropy
MSE, MAE

Backpropagation

TL;DR: backprop = chain rule*

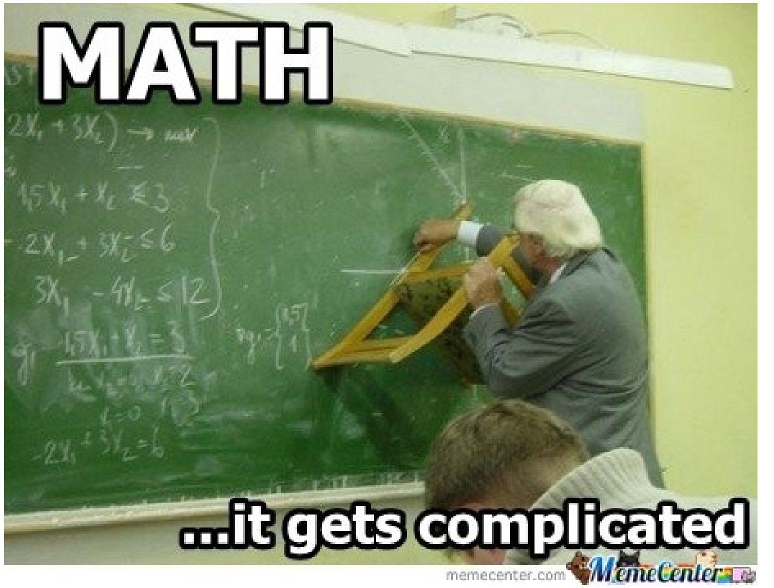
$$\frac{\partial f\left(g\left(x\right)\right)}{\partial x} = \frac{\partial f\left(g\left(x\right)\right)}{\partial g\left(x\right)} \cdot \frac{\partial g\left(x\right)}{\partial x}$$

Backpropagation

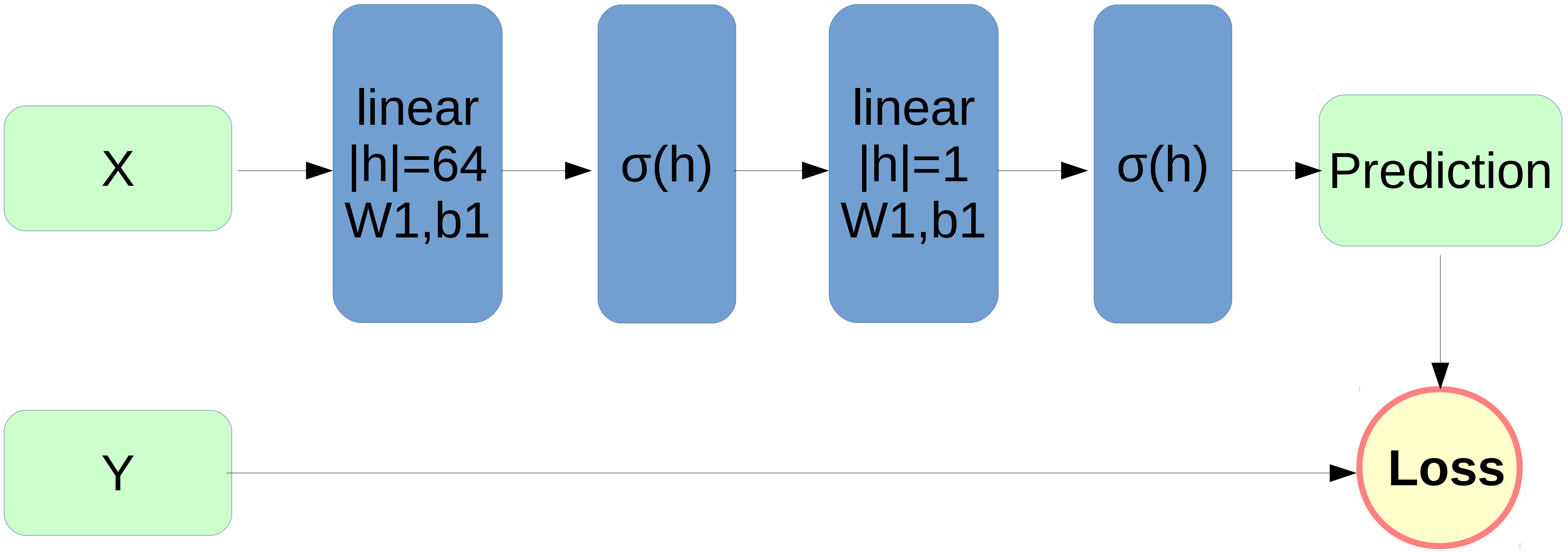
TL;DR: backprop = chain rule*

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

* g and x can be vectors/vectors/tensors

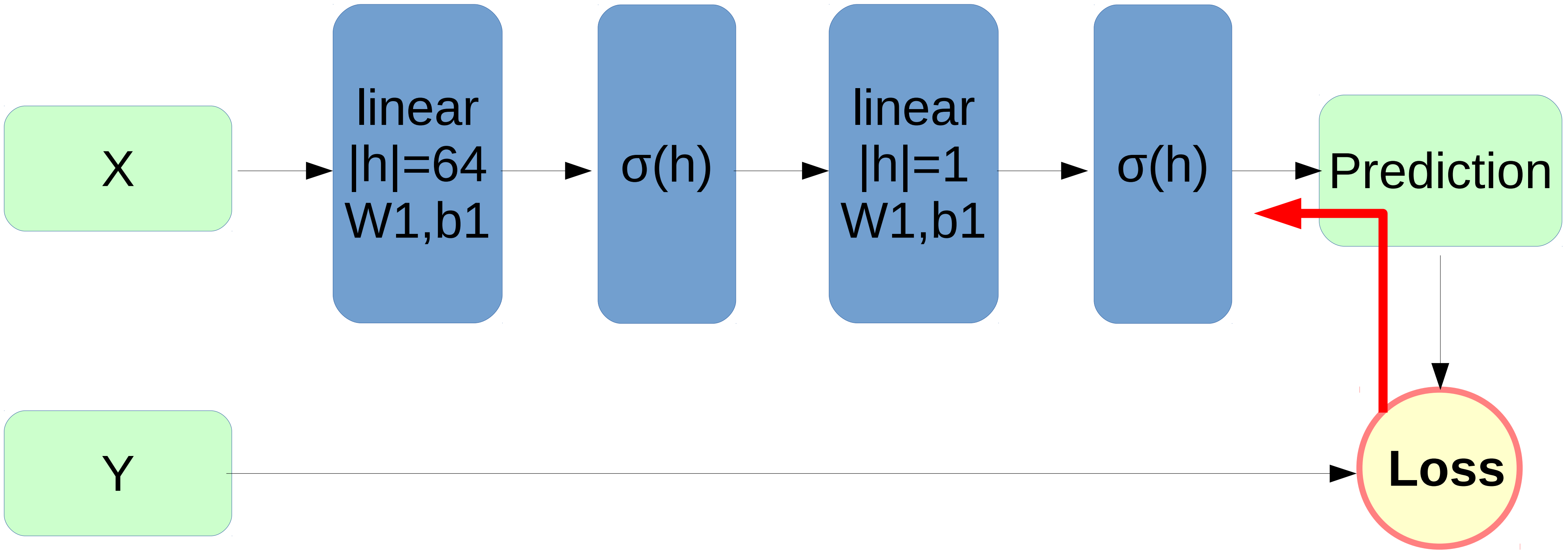


Backpropagation



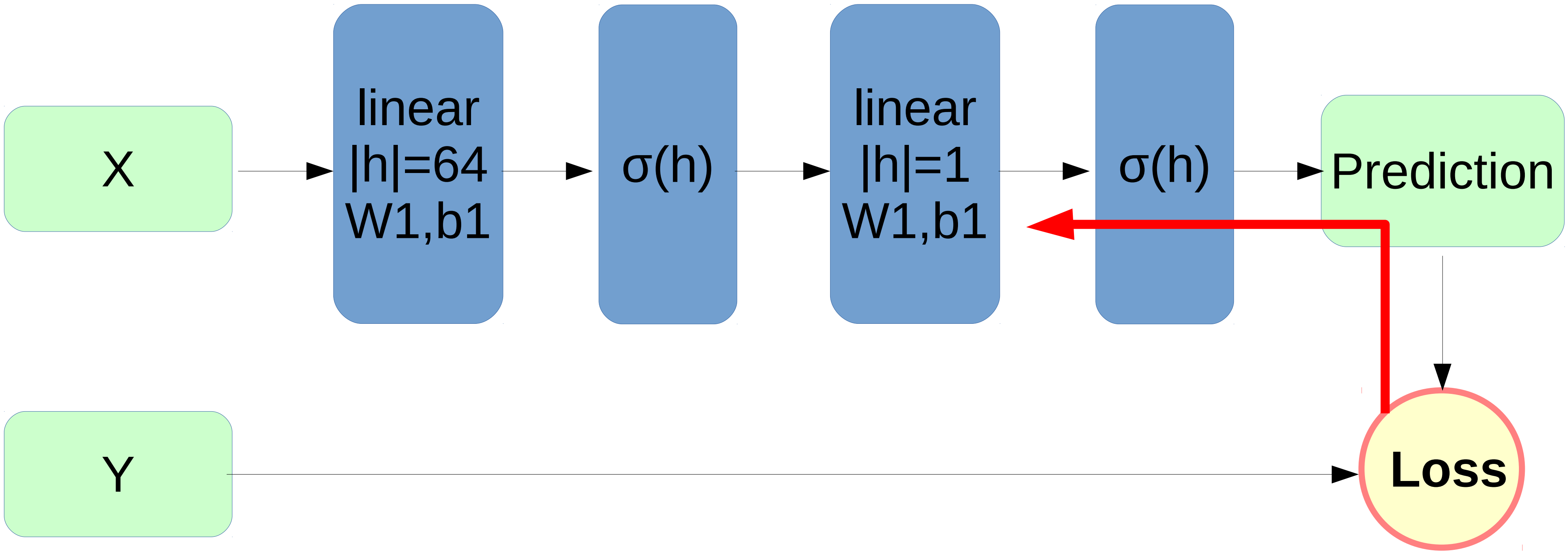
$$\frac{\partial L(\sigma(linear_{w2,b2}(\sigma(linear_{w1,b1}(x)))))}{\partial w1} = \ldots$$

Backpropagation



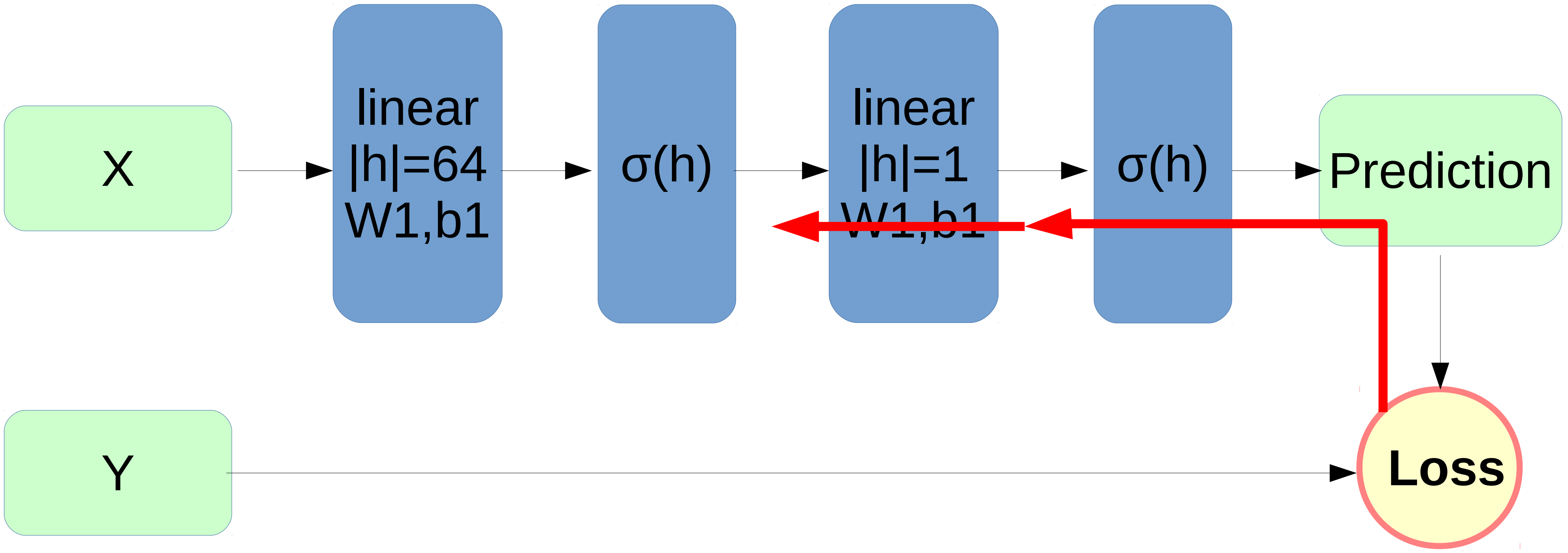
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot$$

Backpropagation



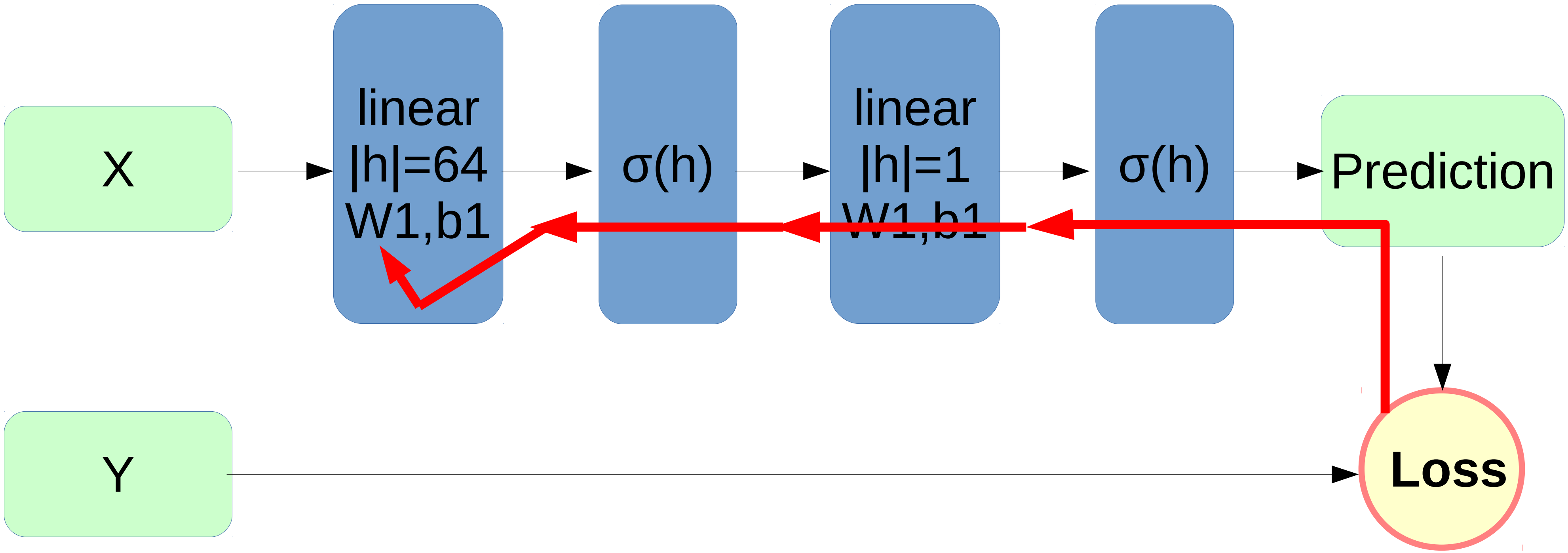
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial \, linear_{w2,b2}} \cdot$$

Backpropagation



$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial \, linear_{w2,b2}} \cdot \frac{\partial \, linear_{w2,b2}}{\partial \sigma} \cdot$$

Backpropagation



$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial\, linear_{w2,b2}} \cdot \frac{\partial\, linear_{w2,b2}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial\, linear_{w1,b1}} \cdot \frac{\partial\, linear_{w1,b1}}{\partial w1}$$

Matrix derivatives

Let's compute:

$$\frac{\partial L(X \times W + b)}{\partial X} = \frac{\partial L(X \times W + b)}{\partial [X \times W + b]} \times \boxed{\textbf{What?}}$$

Variable shapes:

X	W	b
[batch size, features]	[features, outputs]	[outputs]

$$\frac{\partial L(X \times W + b)}{\partial X}$$

[batch size, features]

$$\frac{\partial L(X \times W + b)}{X \times W + b}$$

[batch size, outputs]

Matrix derivatives

Let's compute:

$$\frac{\partial L(X \times W + b)}{\partial X} = \frac{\partial L(X \times W + b)}{\partial [X \times W + b]} \times W^T$$

Variable shapes:

X	W	b
[batch size, features]	[features, outputs]	[outputs]

$\frac{\partial L(X \times W + b)}{\partial X}$	$\frac{\partial L(X \times W + b)}{X \times W + b}$
[batch size, features]	[batch size, outputs]

Matrix derivatives (words)

$$\text{Gradient of } \sum_i \log p(y_i|x_i, w) = \sum_i \text{gradient} \log p(y_i|x_i, w)$$

$$\text{linear over X}: \frac{\partial L}{\partial[X \times W + b]} \times W^T$$

$$\text{linear over W}: \frac{1}{\|X\|} \cdot X^T \times \frac{\partial L}{\partial[X \times W + b]}$$

$$\text{sigmoid}: \frac{\partial L}{\partial \sigma(x)} \cdot [\sigma(x) \cdot (1 - \sigma(x))]$$

Works for any kind of x
(scalar, vector, matrix, tensor)

Matrix derivatives (formulae)

$$\frac{\partial \sum_i \log p(y_i|x_i,w)}{\partial w} = \frac{\sum_i \partial \log p(y_i|x_i,w)}{\partial w}$$

$$\frac{\partial L(X \times W + b)}{\partial X} = \frac{\partial L}{\partial [X \times W + b]} \times W^T$$

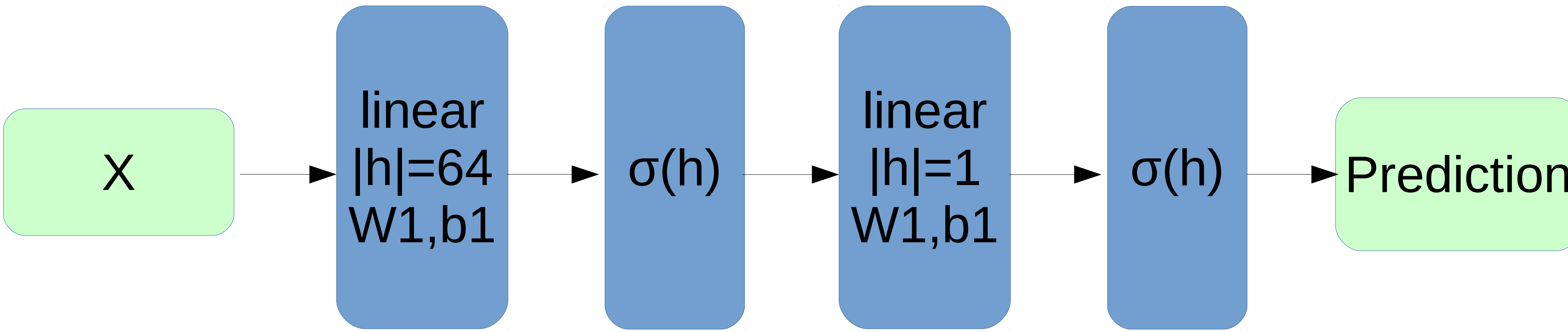
$$\frac{\partial L(X \times W + b)}{\partial W} = X^T \times \frac{\partial L}{\partial [X \times W + b]}$$

$$\frac{\partial L(\sigma(x))}{\partial x} = \frac{\partial L}{\partial \sigma(x)} \cdot [\sigma(x) \cdot (1 - \sigma(x))]$$

Works for any kind of x
(scalar, vector, matrix, tensor)

Back to neural networks

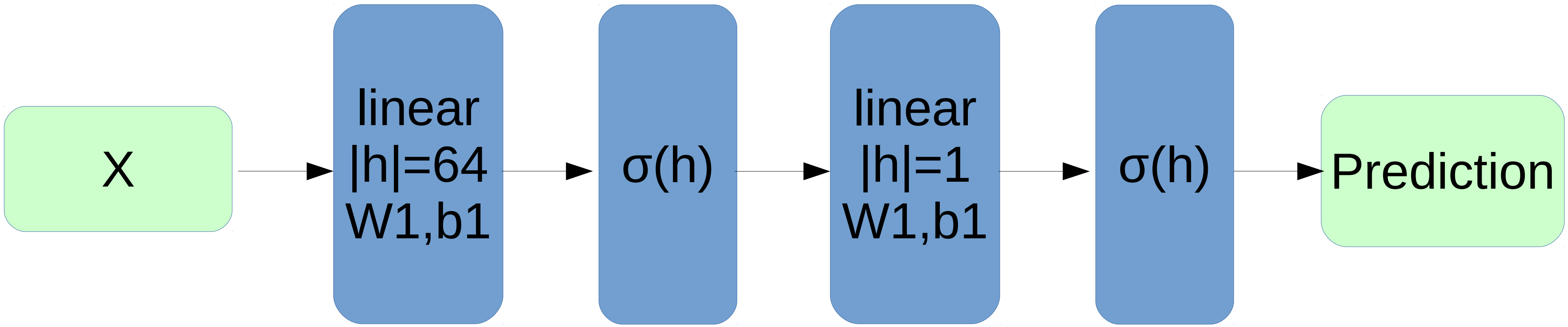
Model:



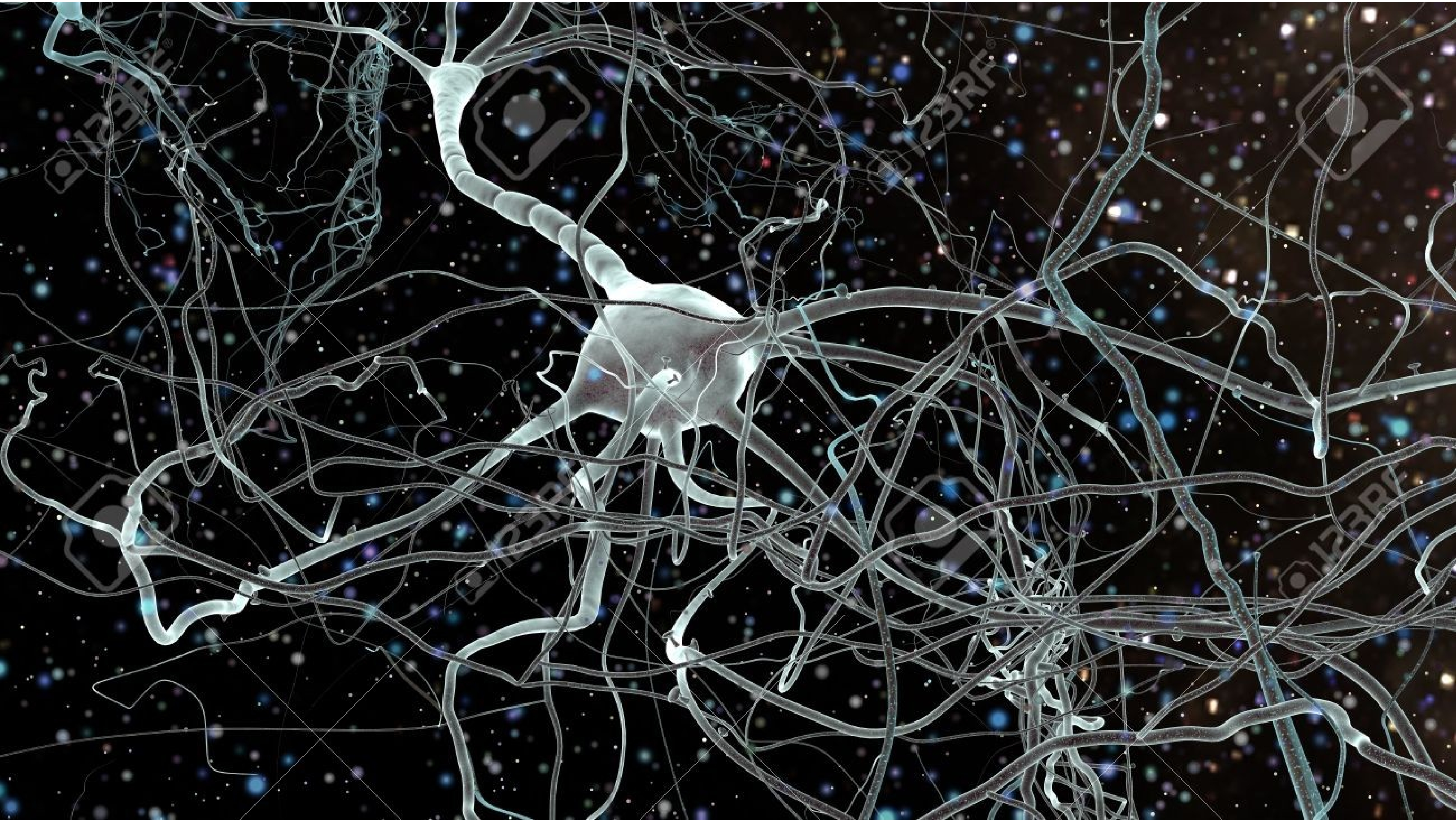
Training:

Back to neural networks

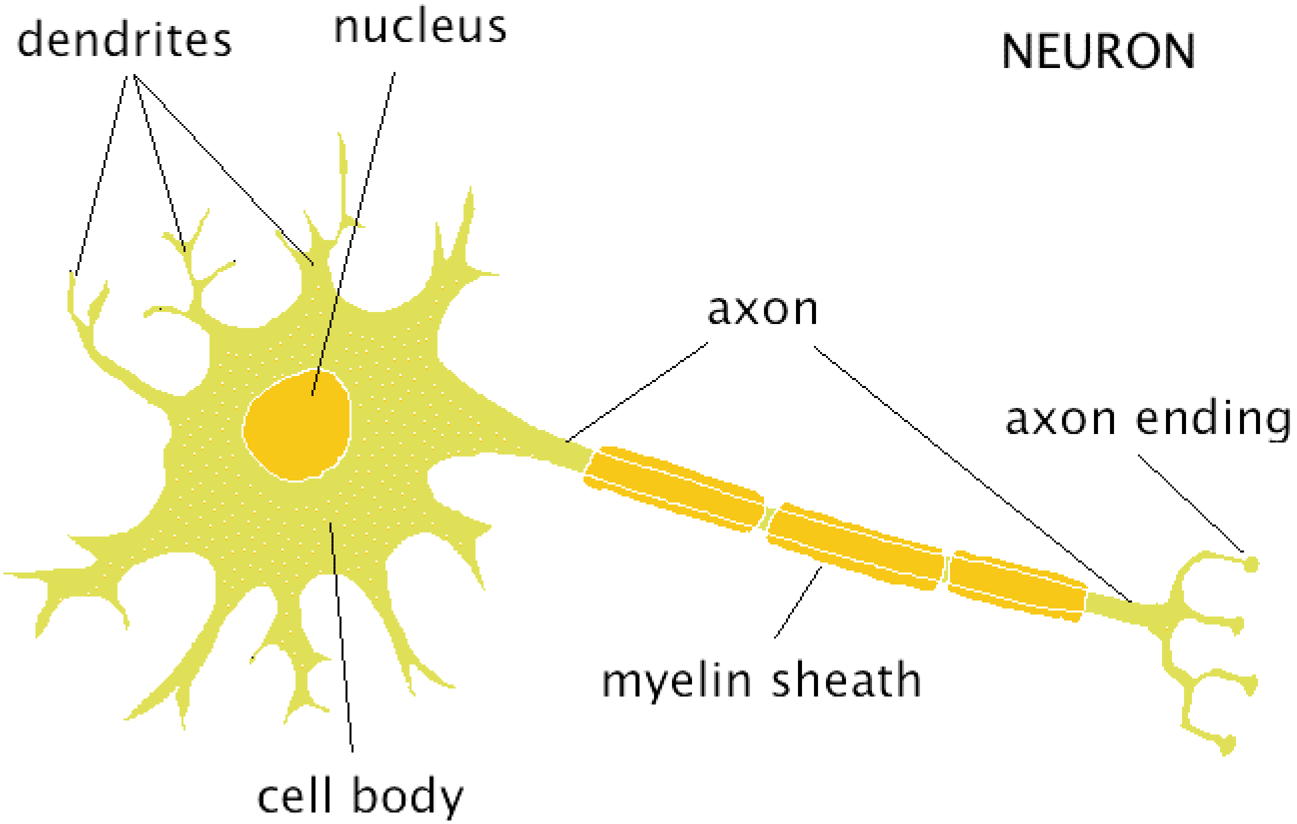
Model:



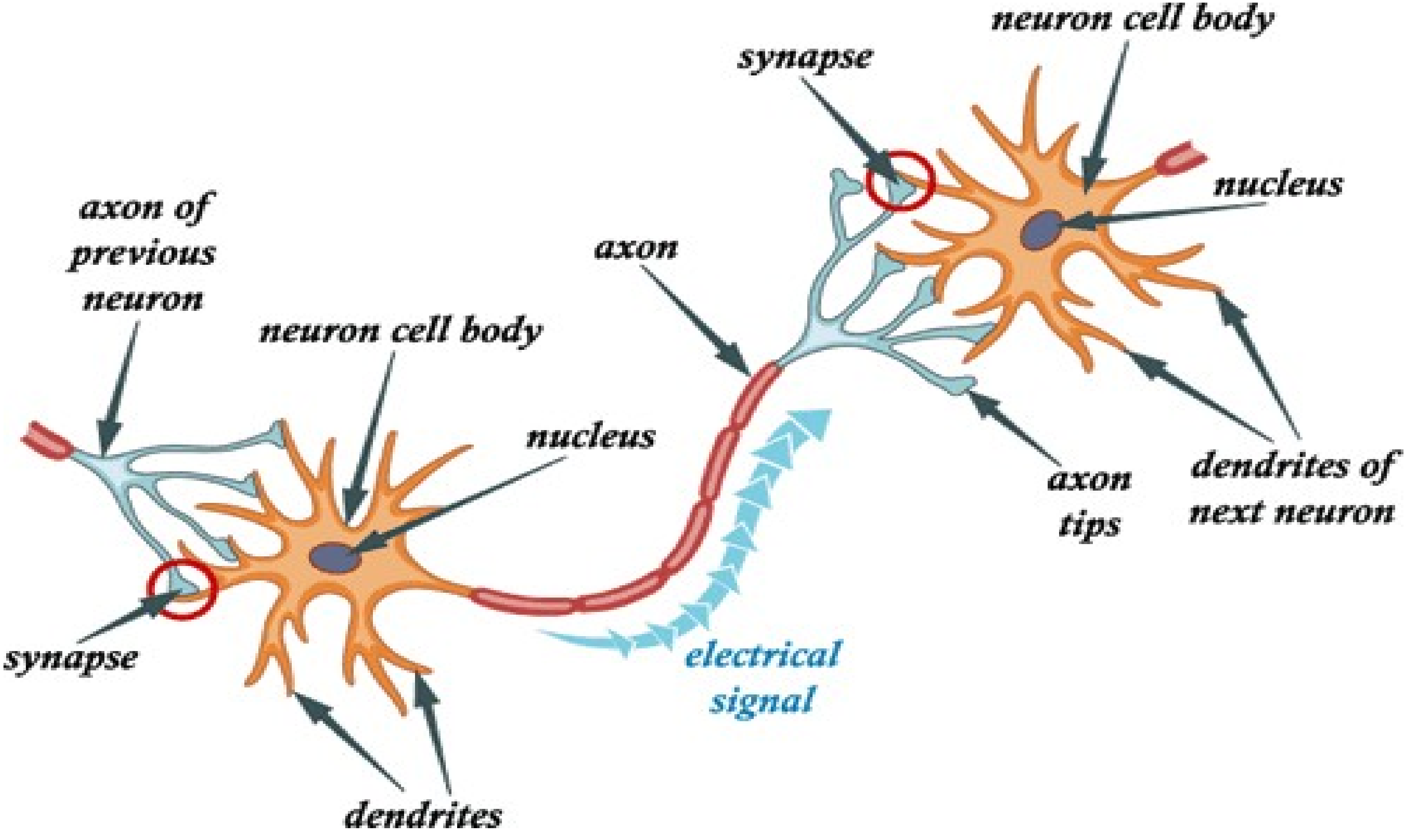
Biological inspiration



Biological inspiration

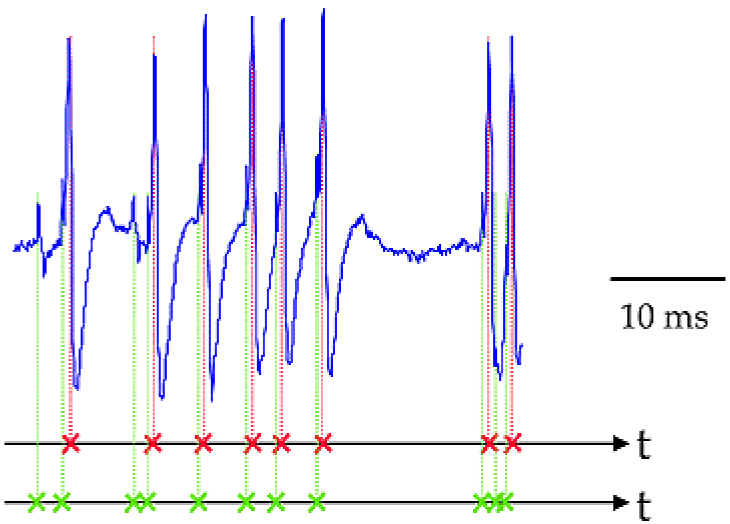


Biological inspiration



Not actual neurons :)

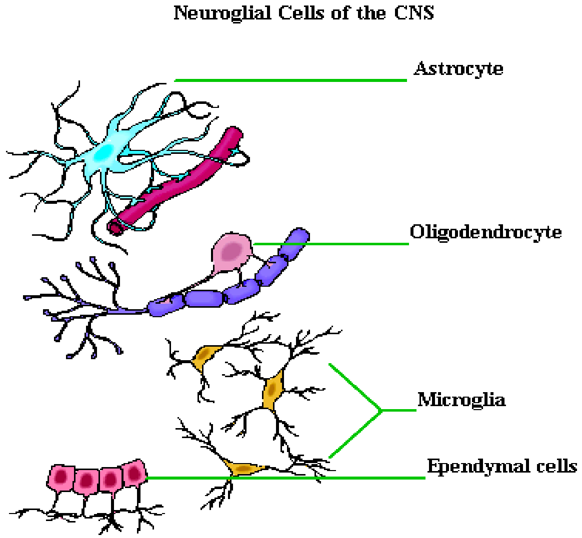
- Neurons react in “spikes”,
 not real numbers



- Neurons maintain/change
 their states over time

- No one knows for sure how they “train”

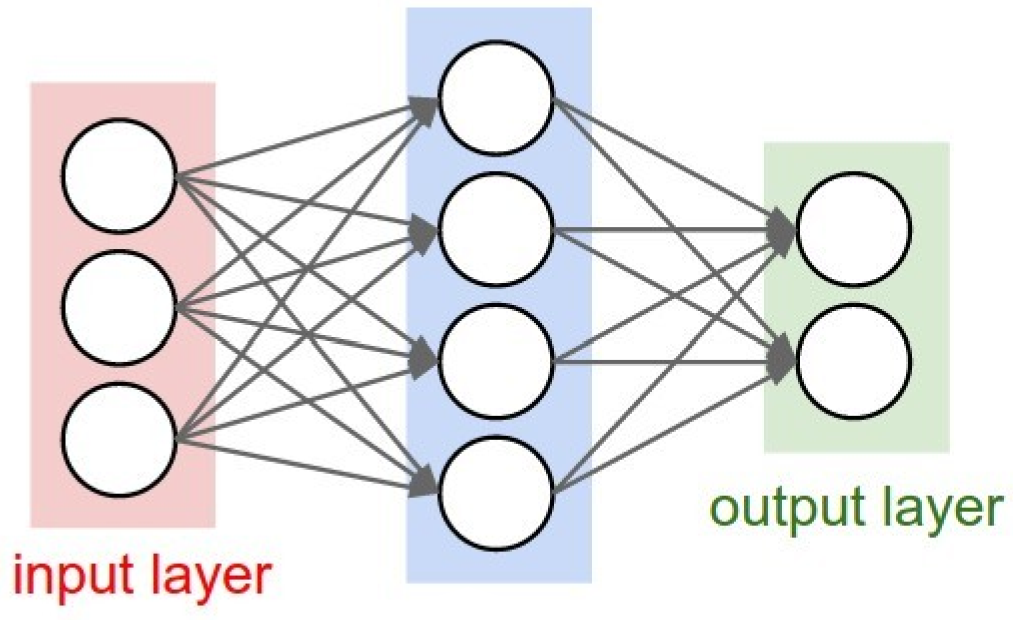
- Neuroglial cells are important
 But noone knows, why



Connectionist phrasebook

- Layer – a building block for NNs :
  - “Dense layer”: $f(x) = Wx+b$
  - “Nonlinearity layer”: $f(x) = \sigma(x)$
  - Input layer, output layer
  - A few more we gonna cover later
- Activation – layer output
  - i.e. some intermediate signal in the NN
- Backpropagation – a fancy word for “chain rule”

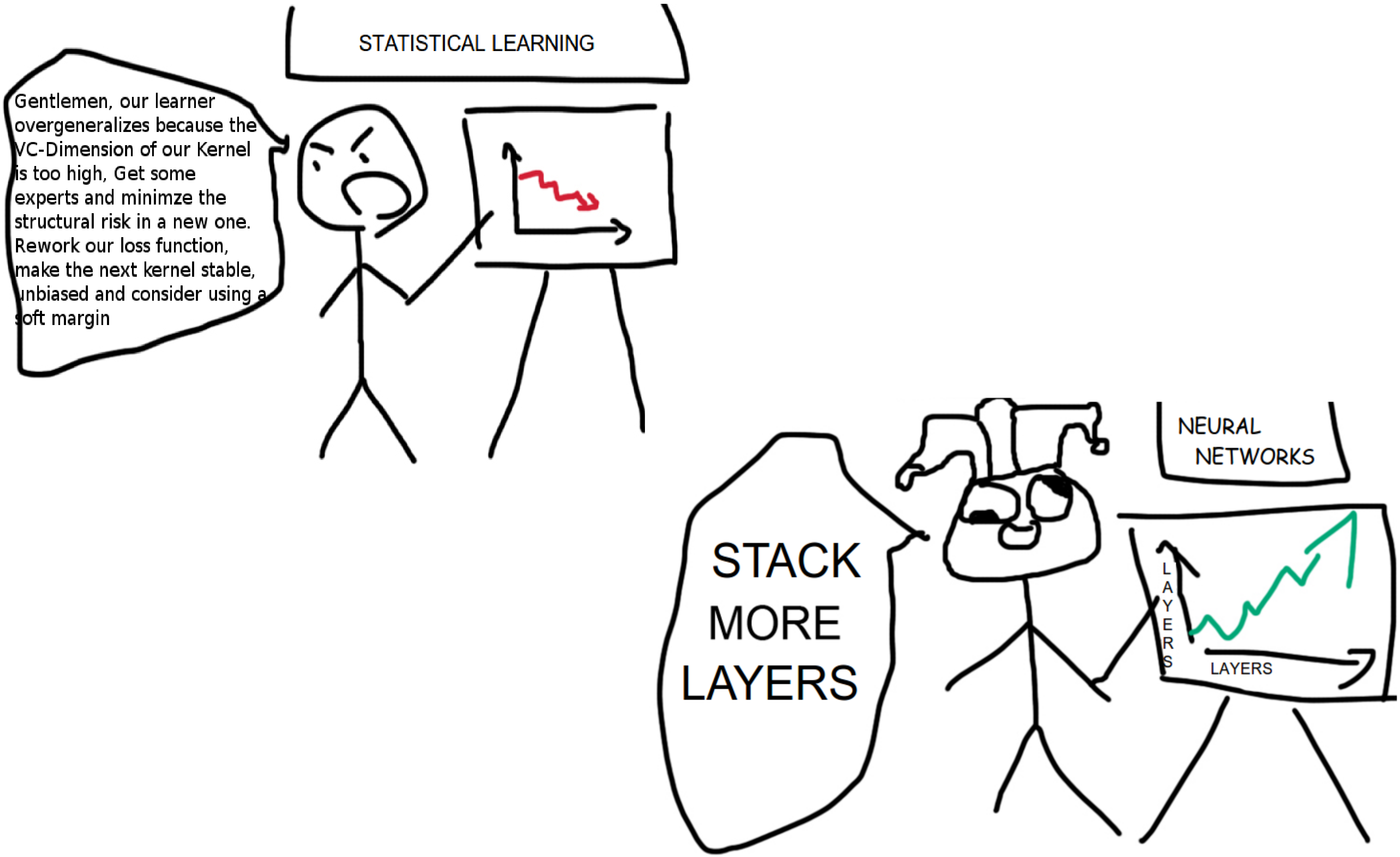
Connectionist phrasebook



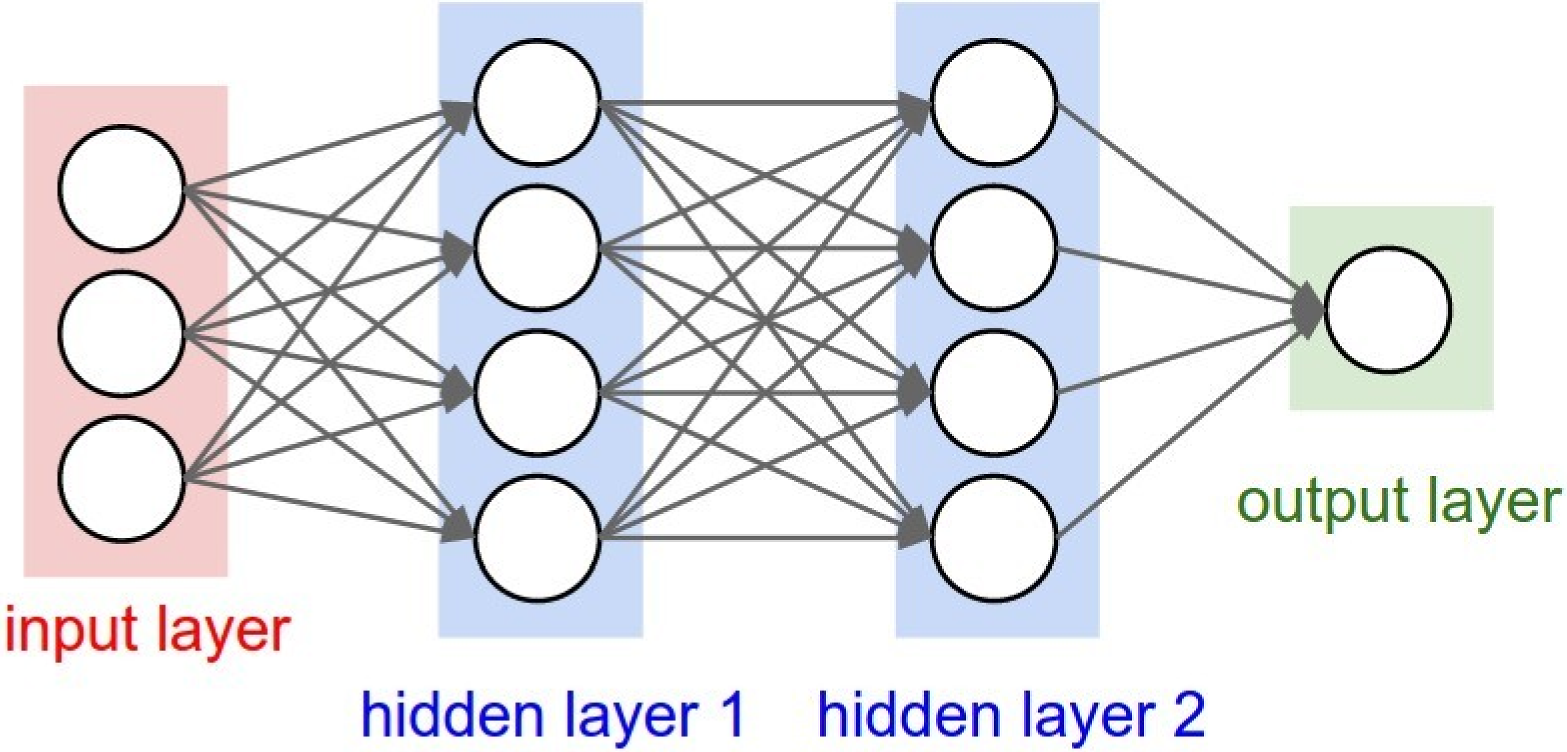
Dense layer
with a nonlinearity

- “Train it via backprop!”

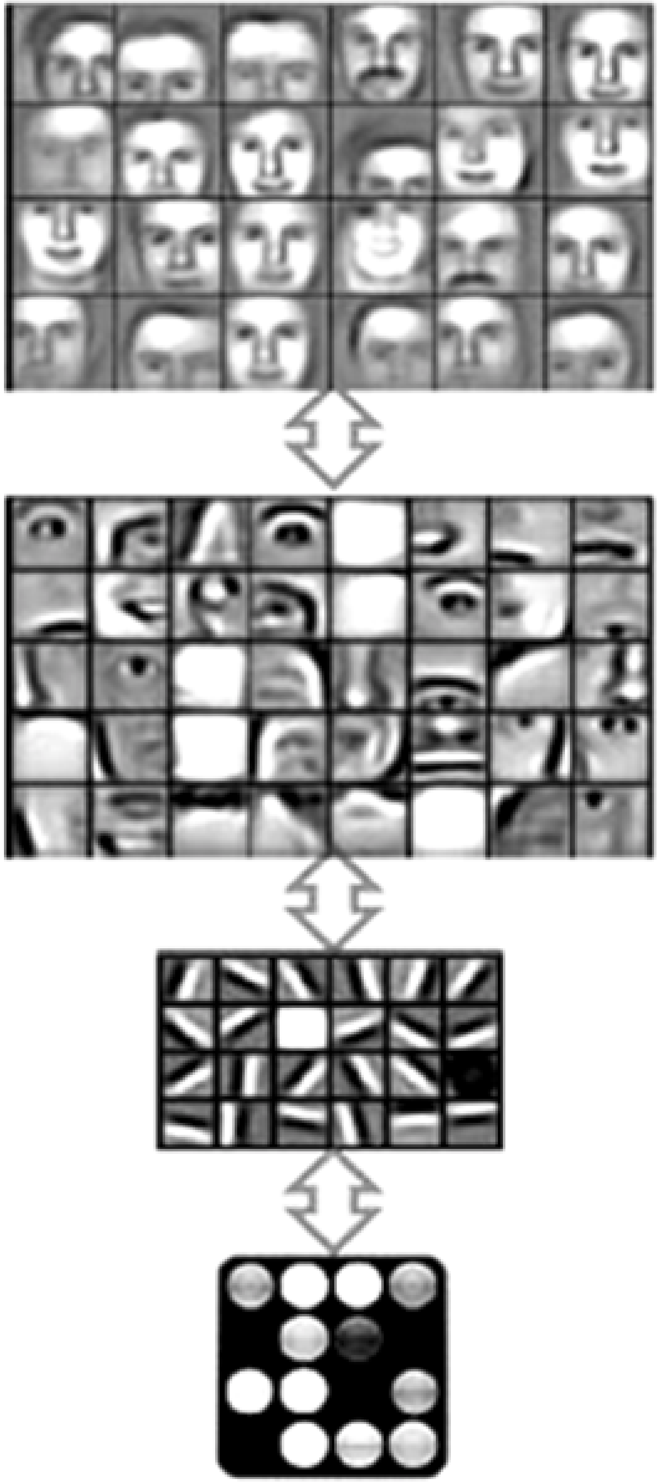
More layers


STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS

More layers



How do we train it?



Discrete Choices

⋮

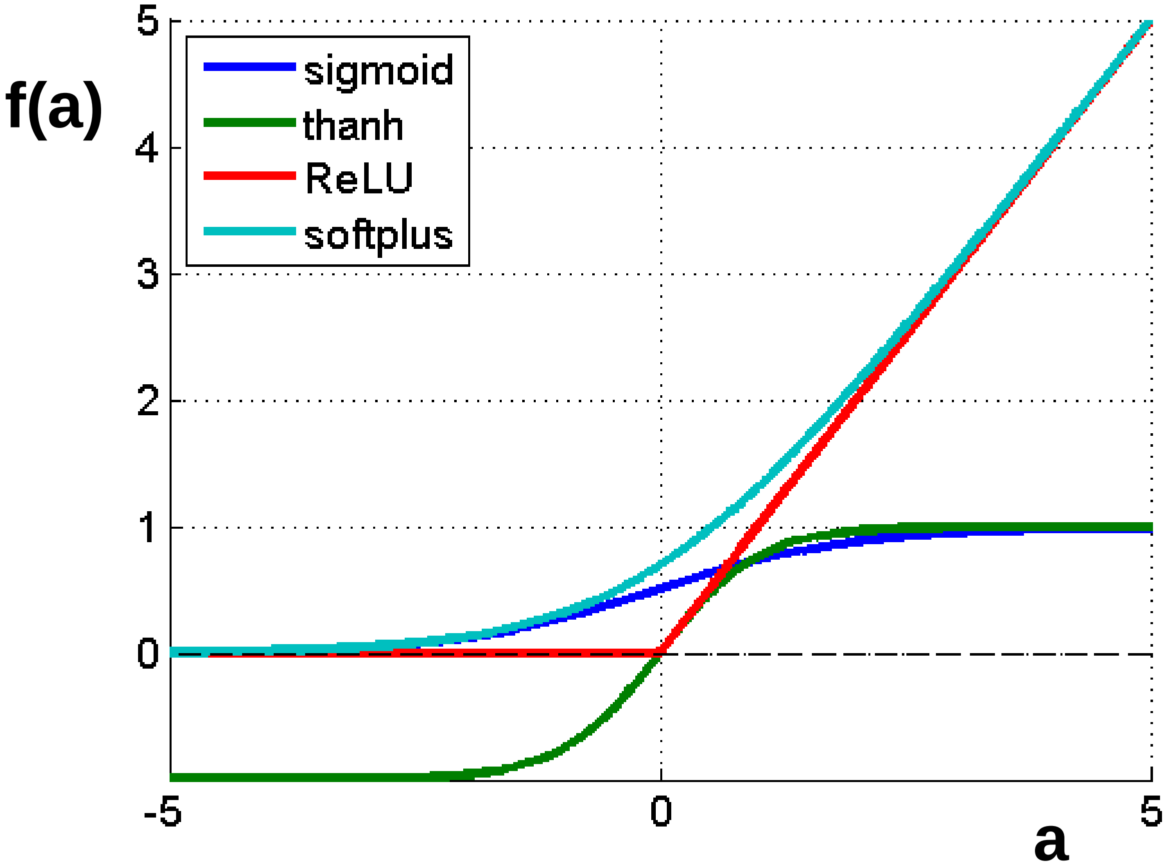
Layer 2 Features

Layer 1 Features

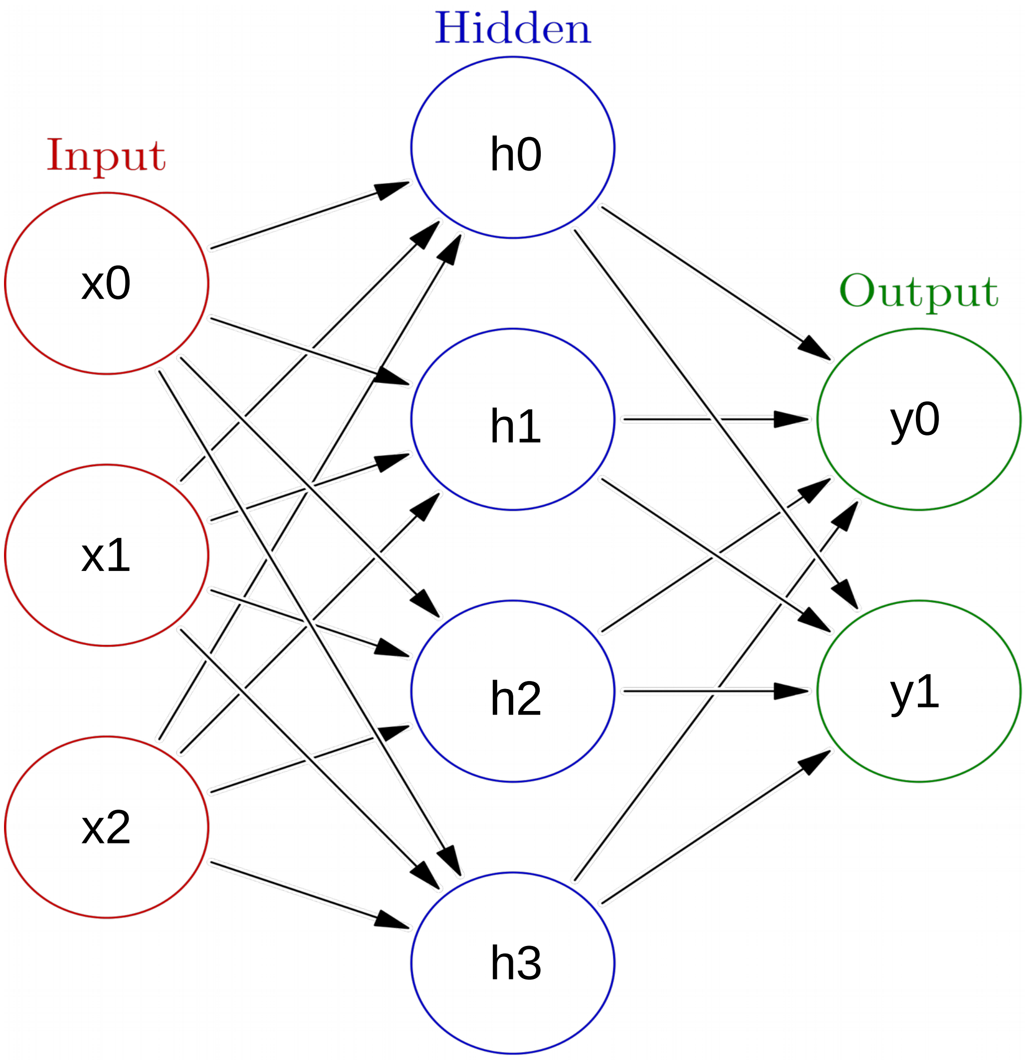
Original Data

Nonlinearity

- f(a) = 1/(1+e^a)
- f(a) = tanh(a)
- f(a) = max(0,a)
- f(a) = log(1+e^a)



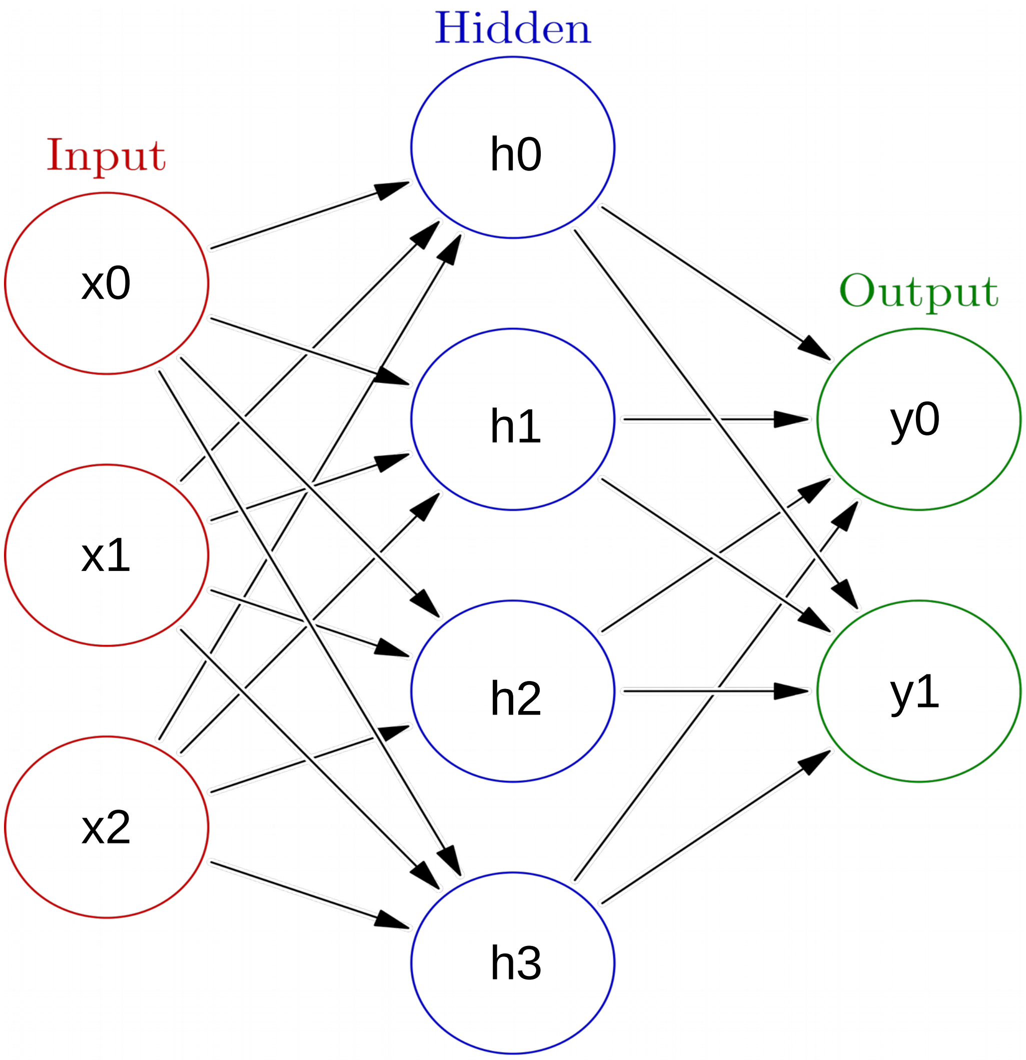
Initialization, symmetry problem



- Initialize with zeros
  $W \leftarrow 0$

- What will the first step look like?

Initialization, symmetry problem



- Break the symmetry!
- Initialize with random numbers!

  $W \leftarrow N(0,0.01)$?

  $W \leftarrow U(0,0.1)$?
- Can get a bit better for deep NNs

Potential caveats?

Potential caveats?

- Hardcore overfitting
- No “golden standard” for architecture
- Computationally heavy

You gonna code this

today