Deep learning

Episode 1

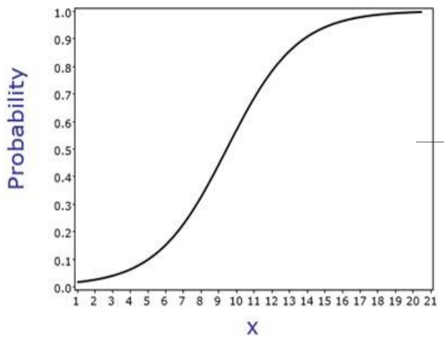
Neural networks 101

Yandex Data Factory

LAMBDA

British Hedgehog Preservation Society

Recap: logistic regression

X → Wx +b →  → P(y)

Gradient descent

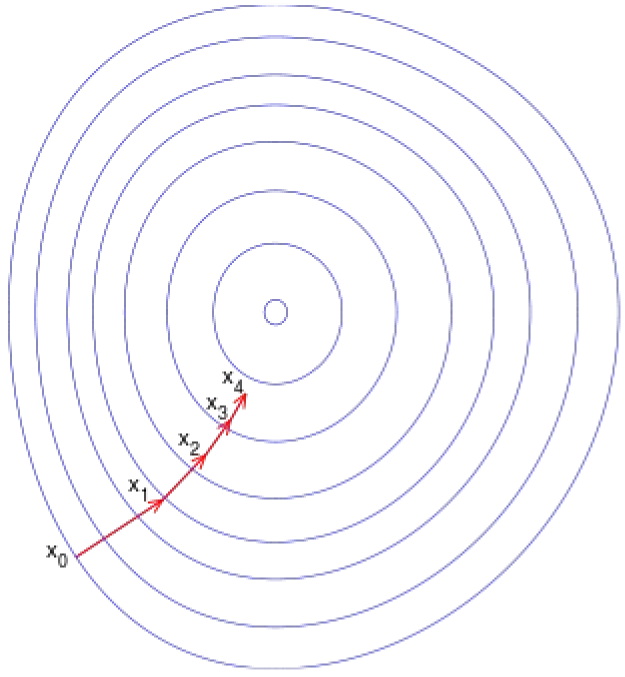
$$y_{pred}(\bar{x}) = \sigma(\bar{w} \cdot \bar{x} + b)$$

$$L(y, y_{pred}) = -(y \cdot \log y_{pred} + (1-y) \cdot \log(1-y_{pred}))$$

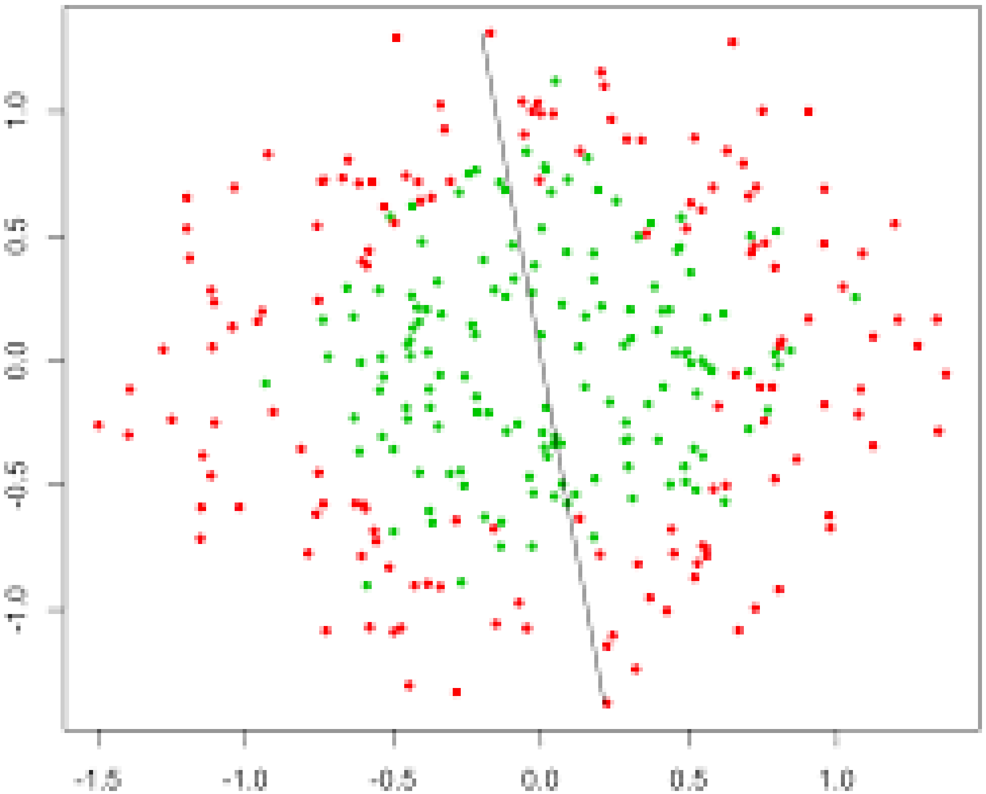
Repeat until convergence

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial L(y, y_{pred})}{\partial \theta_j}$$

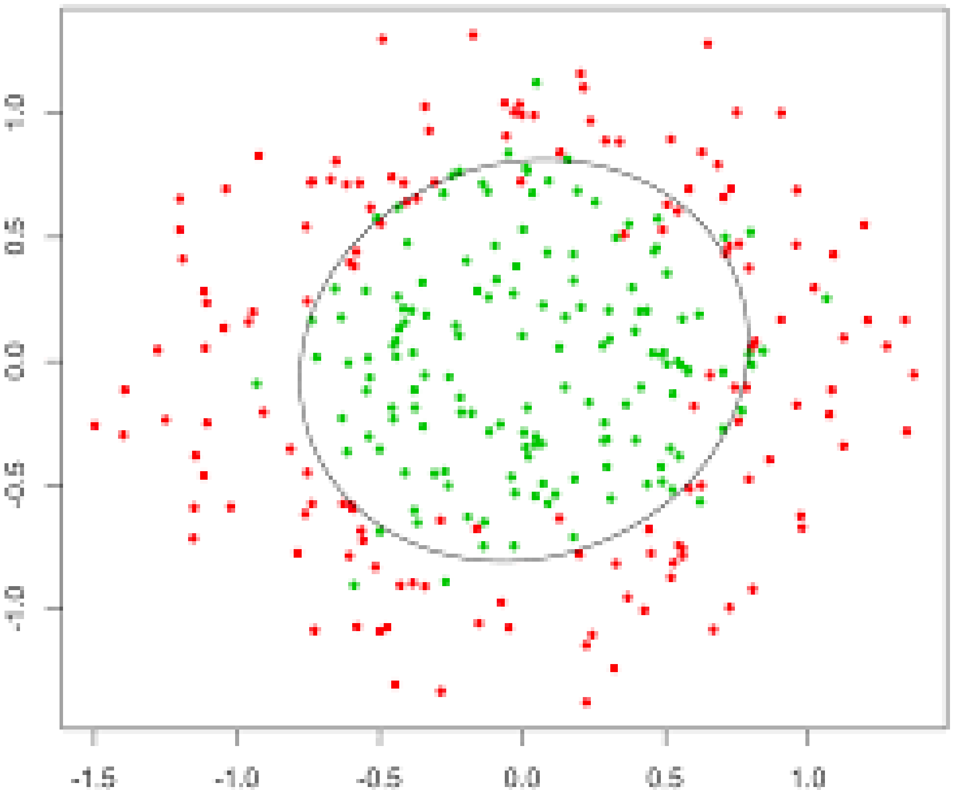
ϴ~{W,b}



Nonlinear dependencies



What we have



What we want

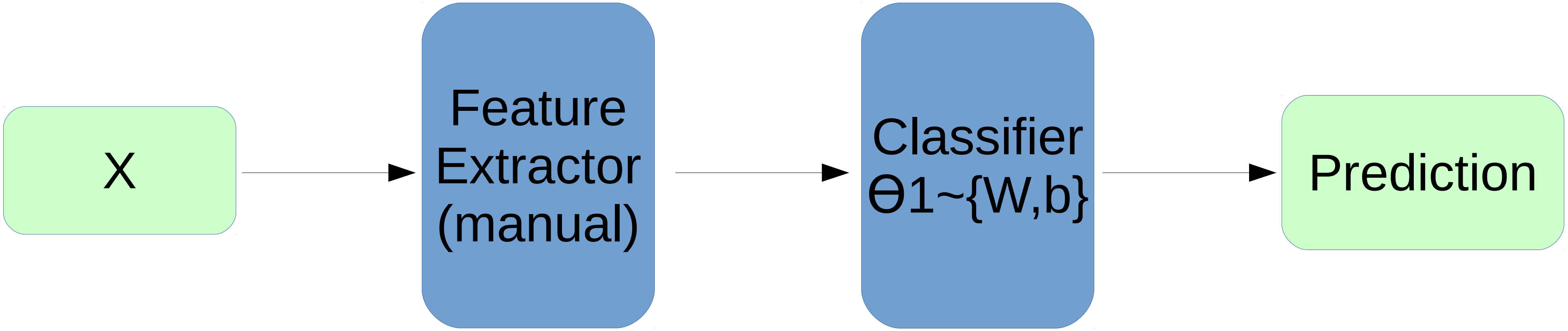
- How to get that?

Feature extraction

Loss, for example:

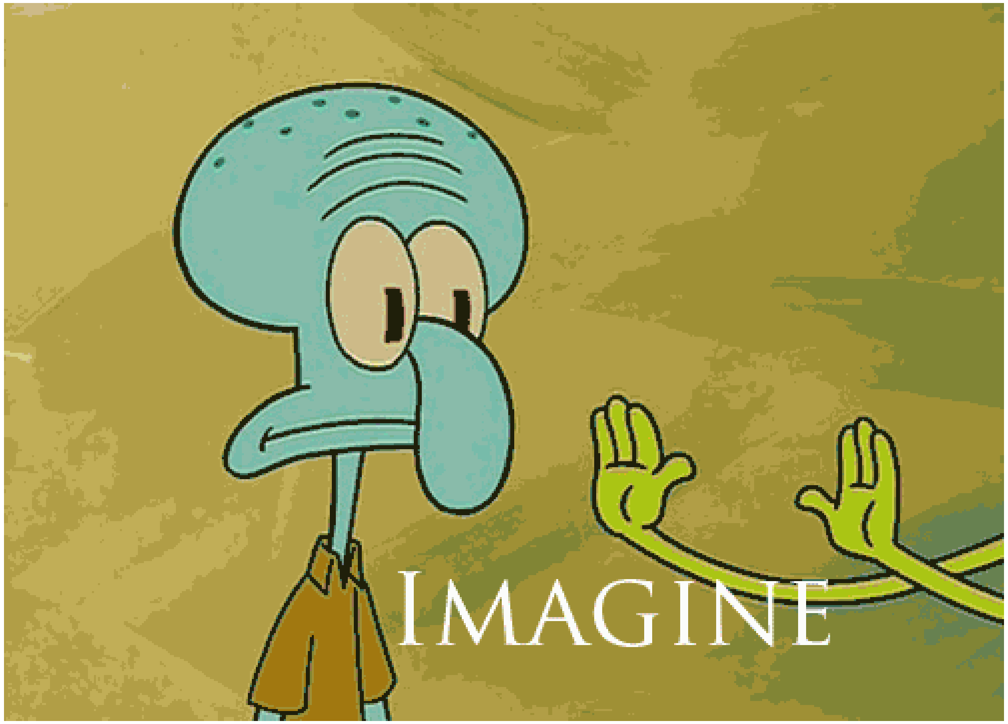
$$L(y, y_{pred}) = -\left(y \cdot \log y_{pred} + (1-y) \cdot \log(1-y_{pred})\right)$$

Model:



Training:

$$\underset{\theta_1}{argmin}\, L(y, y_{pred}(x))$$



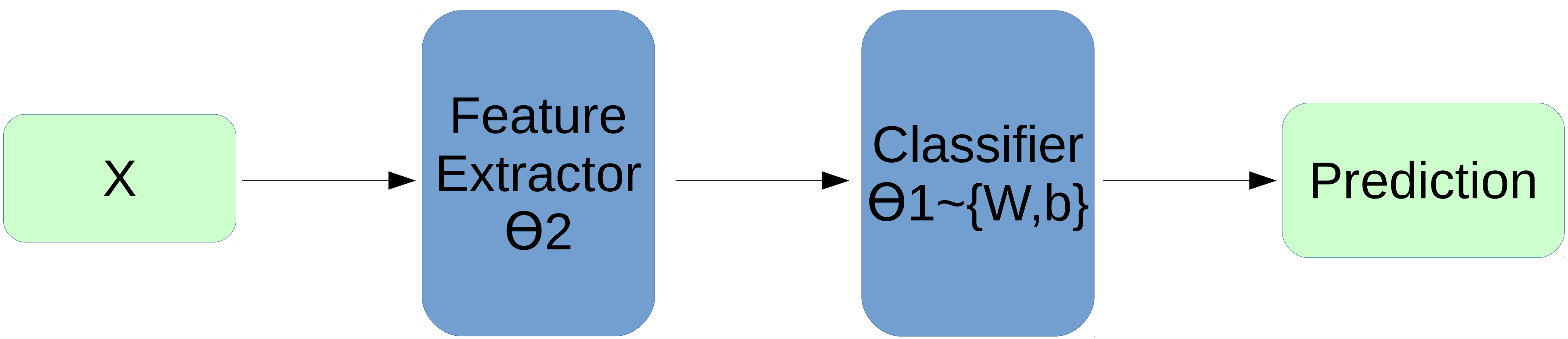
Features would tune to your problem automatically!

What do we want, exactly?

Loss, for example:

$$L(y, y_{pred}) = -\left(y \cdot \log y_{pred} + (1-y) \cdot \log(1-y_{pred})\right)$$

Model:



Training: ?

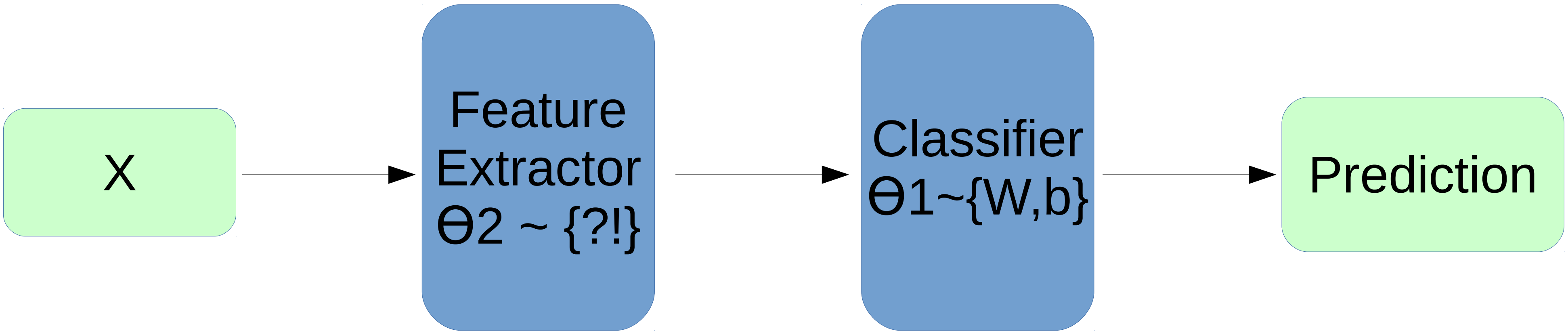
$$\underset{\theta_1}{argmin}\, L(y, y_{pred}(x))$$

What do we want, exactly?

Loss, for example:

$$L(y, y_{pred}) = -(y \cdot \log y_{pred} + (1-y) \cdot \log(1-y_{pred}))$$

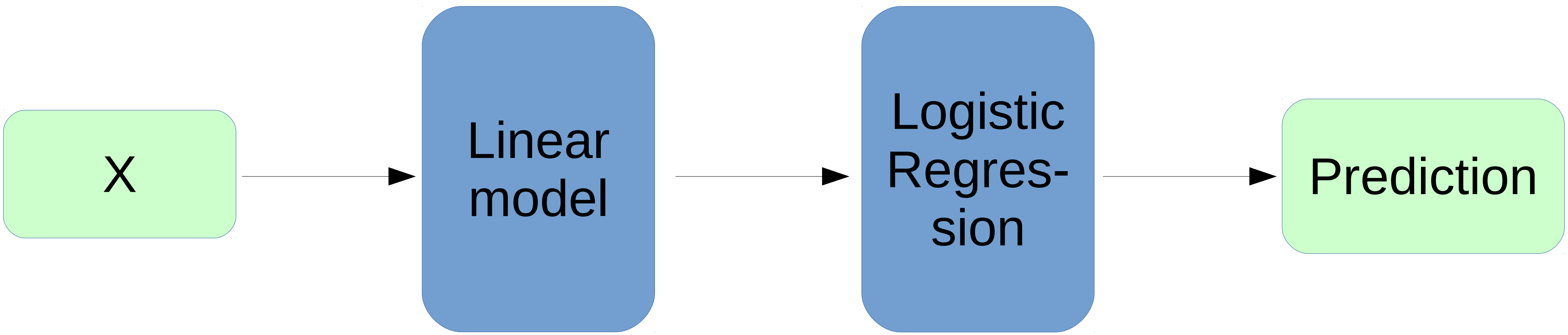
Model:



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

Try linear

Model:

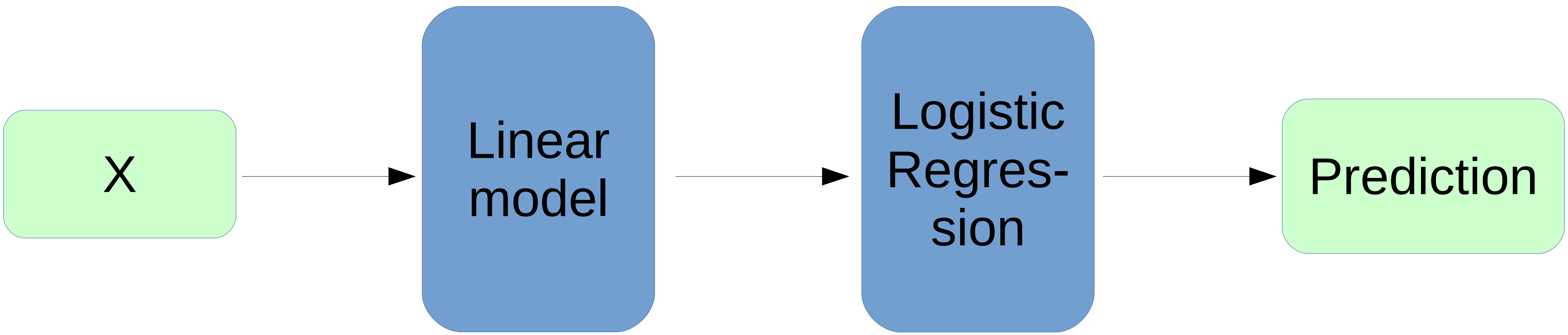


$$h_j = \sum_i w_{ij}^h x_i + b_j^h \quad j \in \{1, 2, \dots, n\}$$

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

Try linear

Model:



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

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

Output:

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

Is it any better than logistic regression?

Try linear

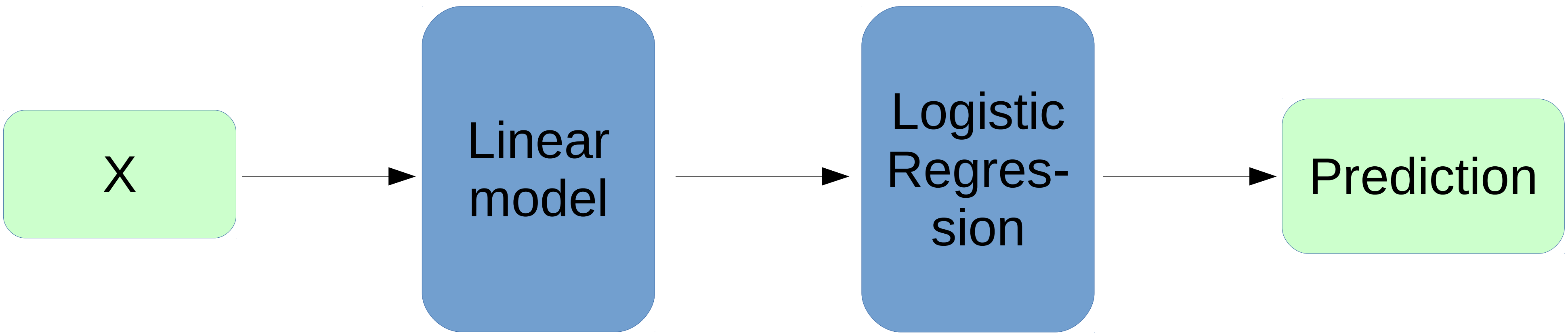
$$y_{pred} = \sigma\left(\sum_j w_j^o \left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

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

$$y_{pred} = \sigma\left(\sum_i w'_i x_i + b'\right)$$

Try linear

Model:



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

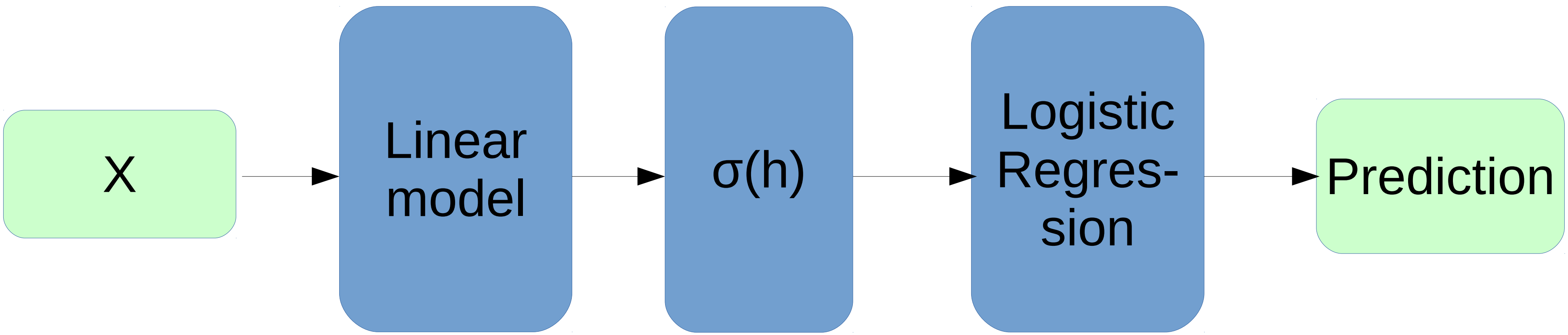
$$j \in \{1, 2, \ldots, n\}$$

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

Is it any better than logistic regression?

Nonlinearity

Model:

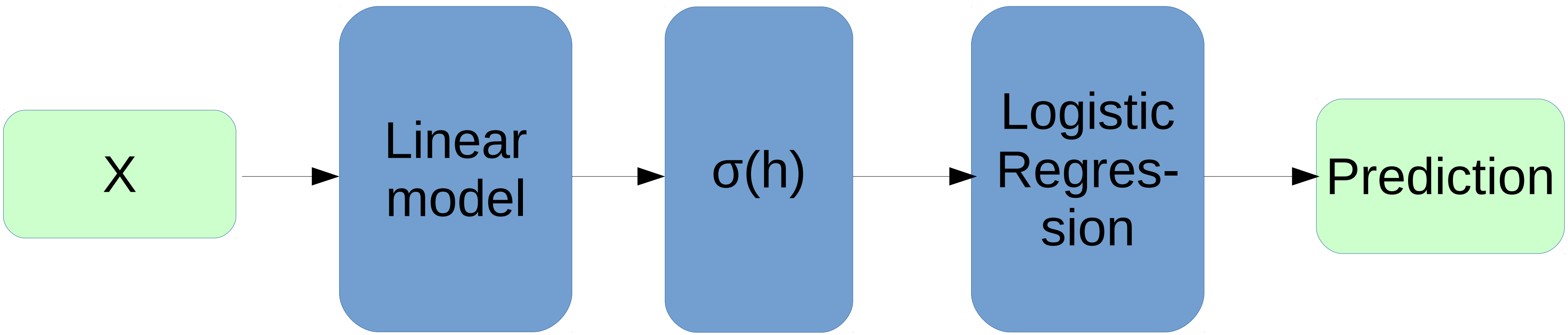


$$h_j = \sigma\left(\sum_{i} w_{ij}^{h} x_i + b_j^{h}\right) \qquad y_{pred} = \sigma\left(\sum_{j} w_j^{o} h_j + b^{o}\right)$$

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

Nonlinearity

Model:



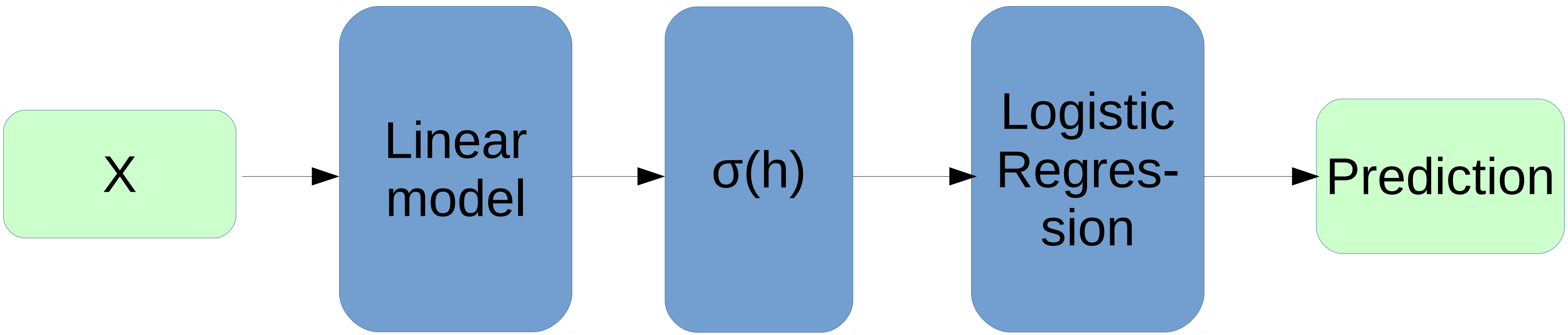
$$h_j = \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) \qquad y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

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

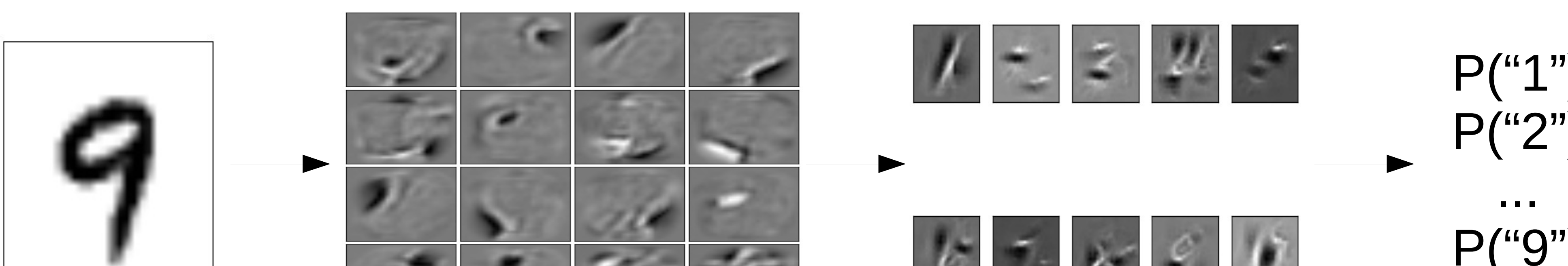
Output: $$y_{pred} = \sigma\left(\sum_j w_j^o \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

Nonlinearity

Model:

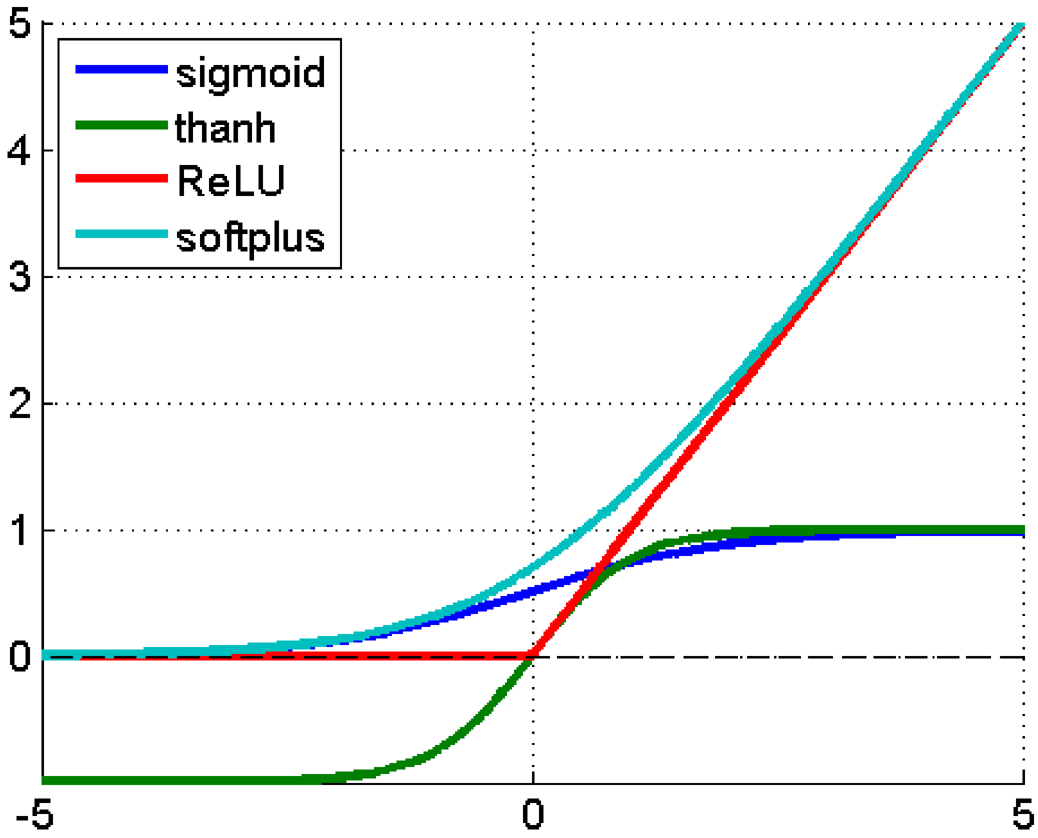


$$h_j = \sigma\left(\sum_{i\in\{1,2,\ldots,n\}} w_{ij}^{h} x_i + b_j^{h}\right) \qquad y_{pred} = \sigma\left(\sum_{j} w_j^{o} h_j + b^{o}\right)$$

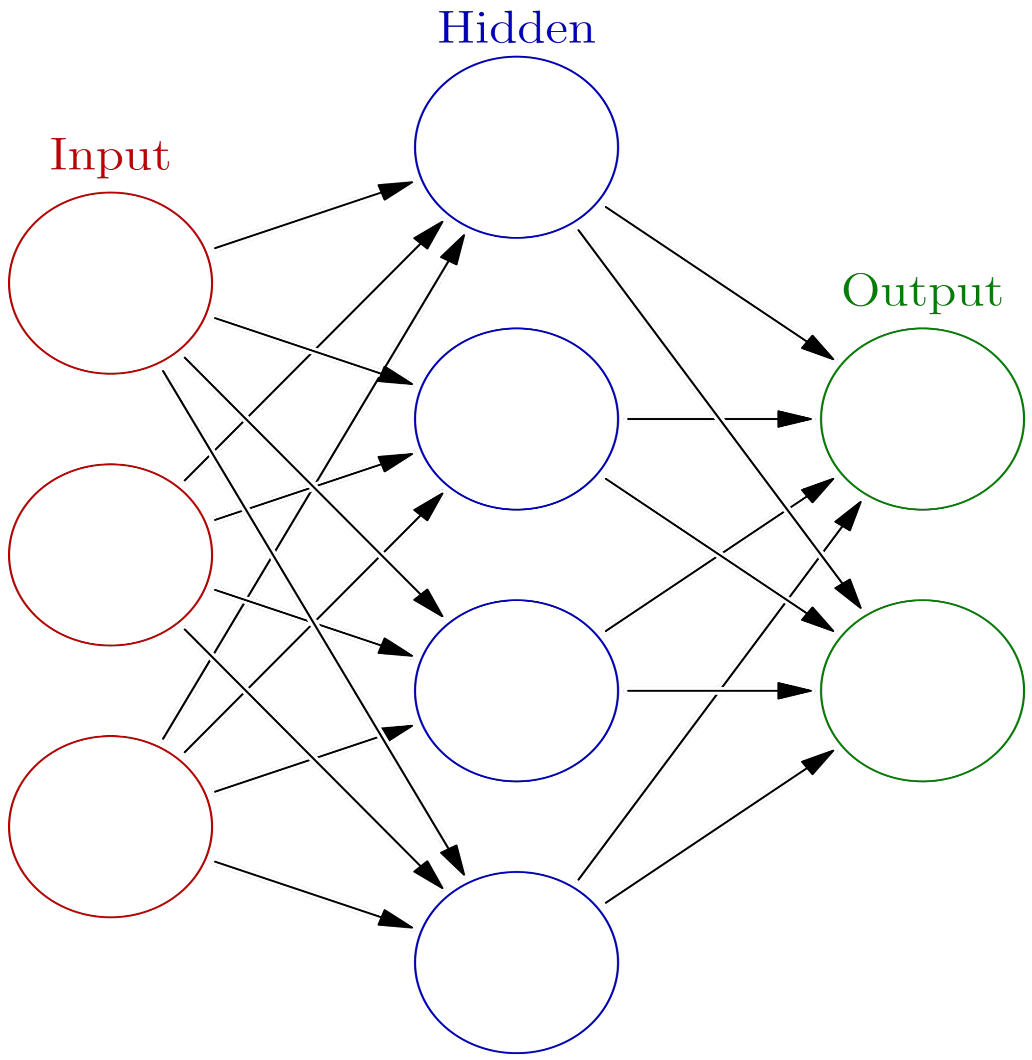


Nonlinearity

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

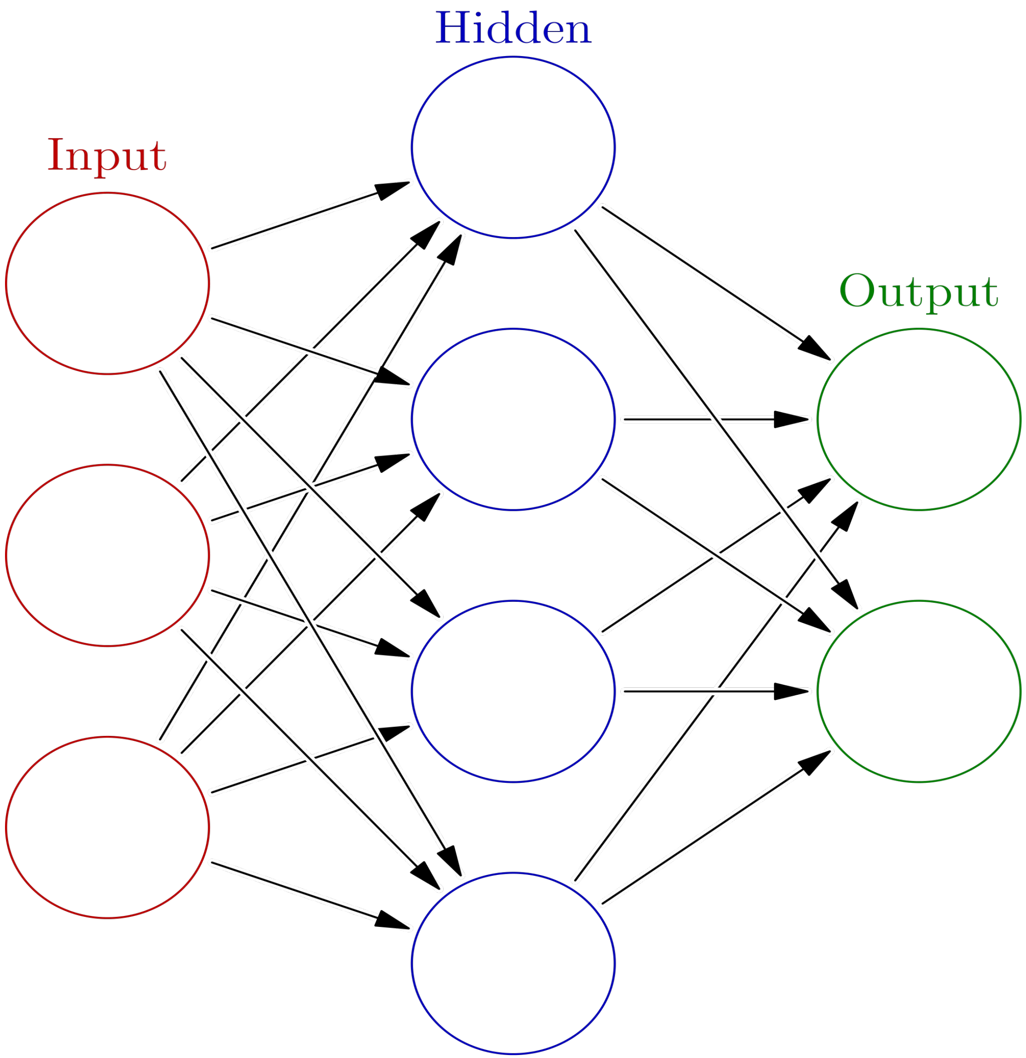


Initialization, symmetry problem



- Initialize with zeros
  $W \leftarrow 0$
- What will the first step look like?

Initialization, symmetry problem



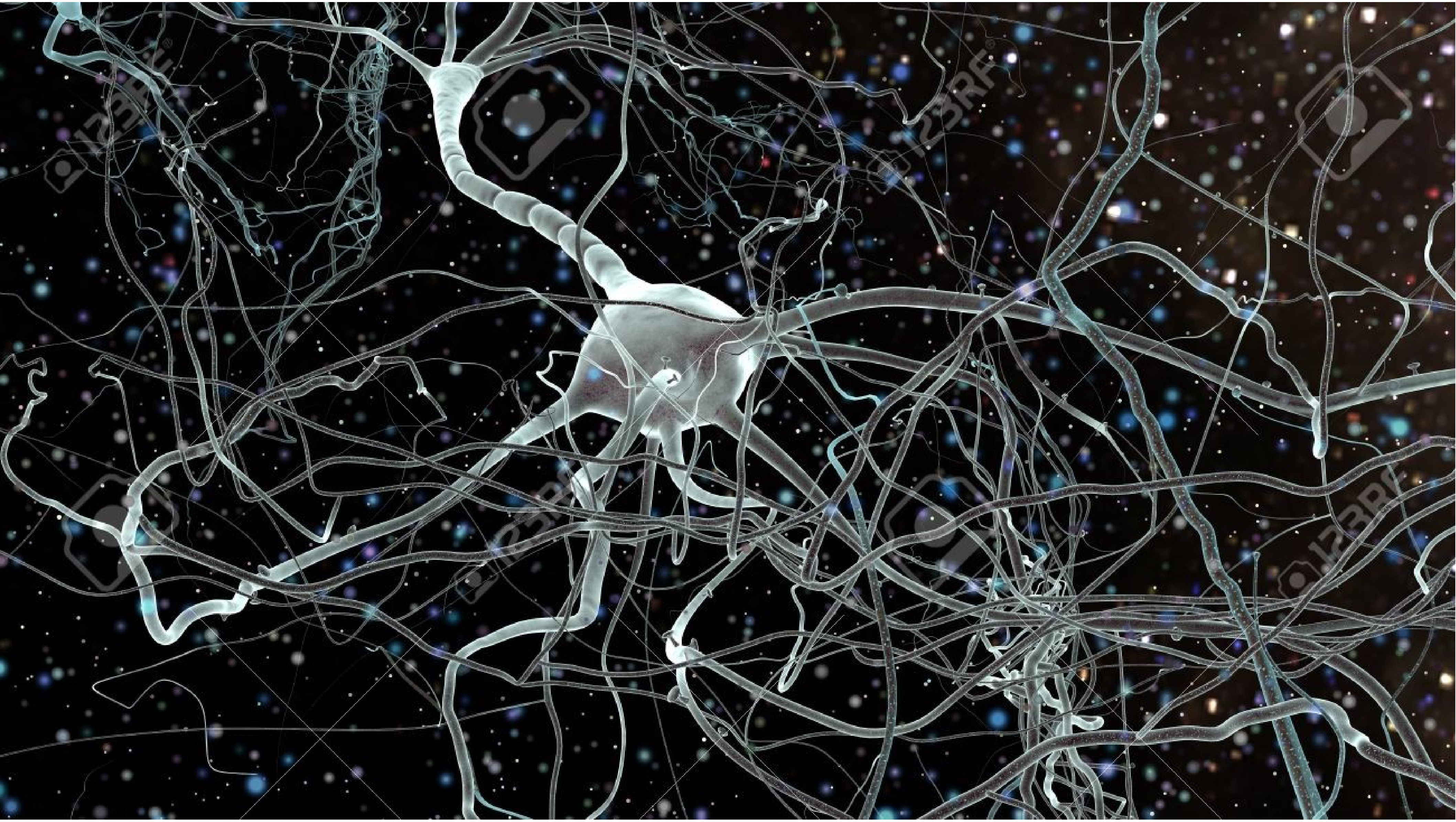
- Break the symmetry!
- Initialize with random numbers!

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

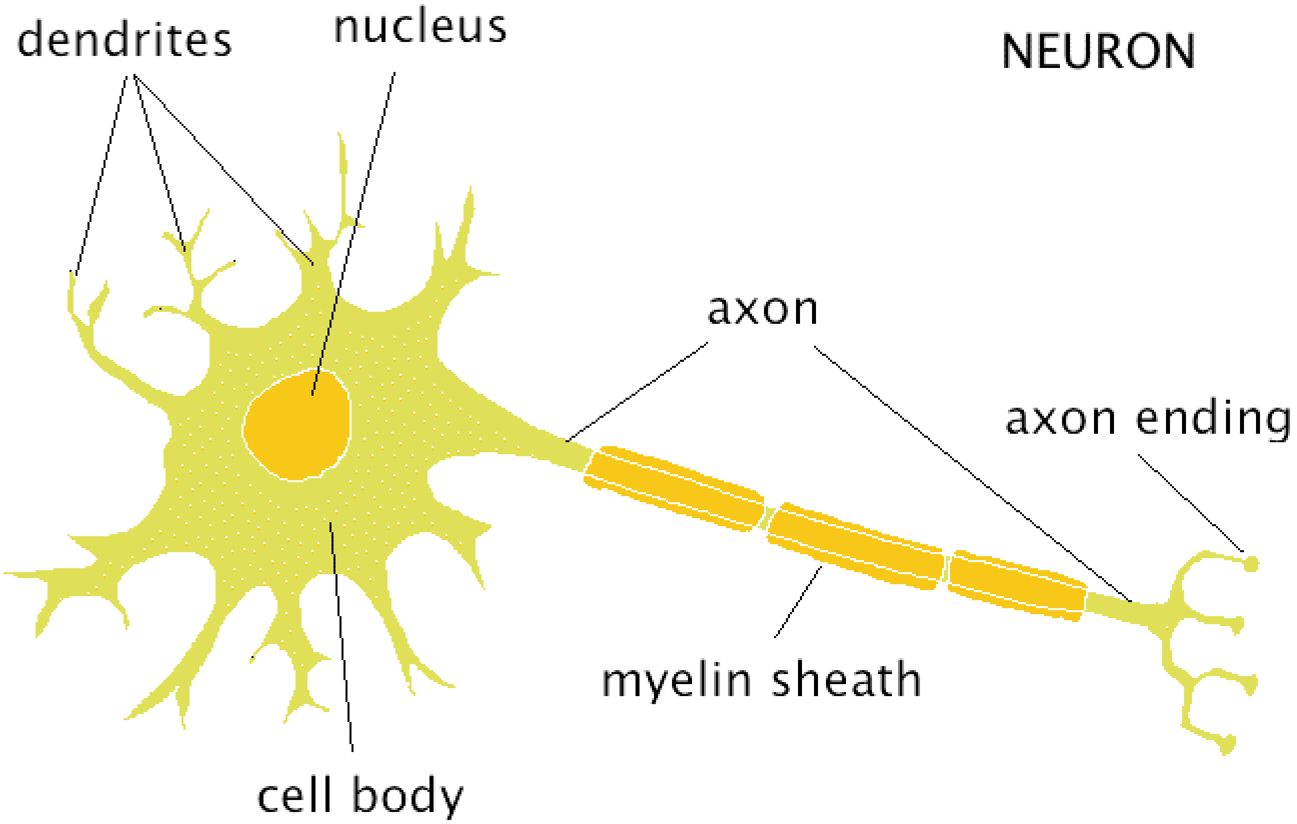
  $W \leftarrow U(0,0.1)$?

- Can get a bit better for deep NNs

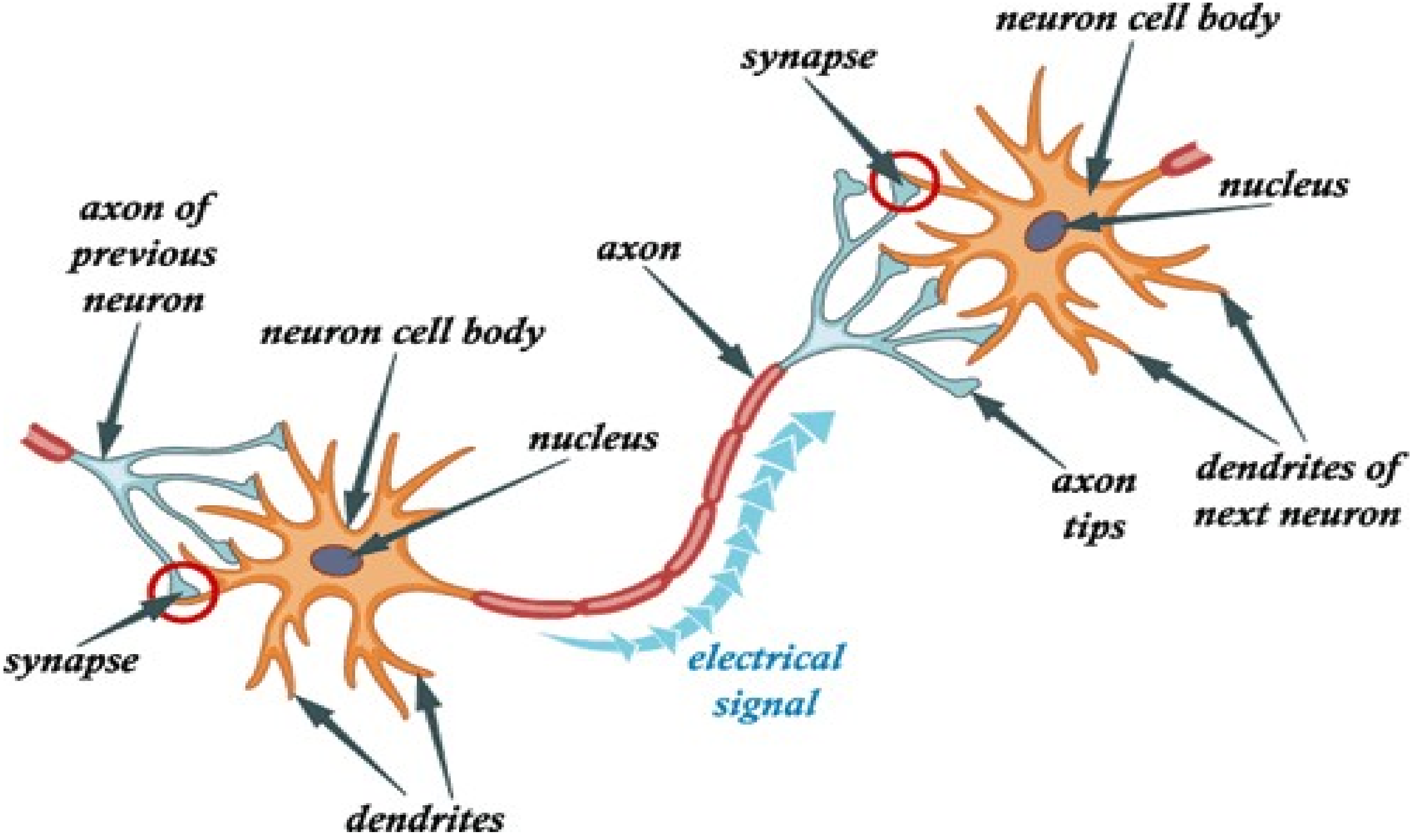
Biological inspiration



Biological inspiration



Biological inspiration



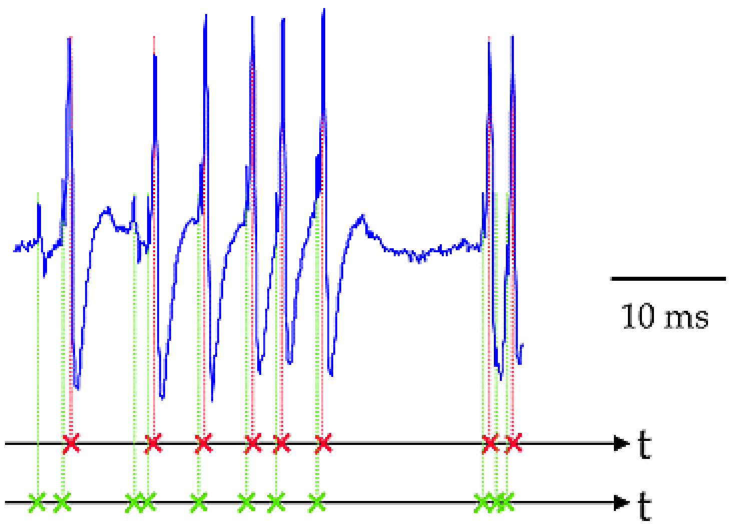
Not actual neurons :)

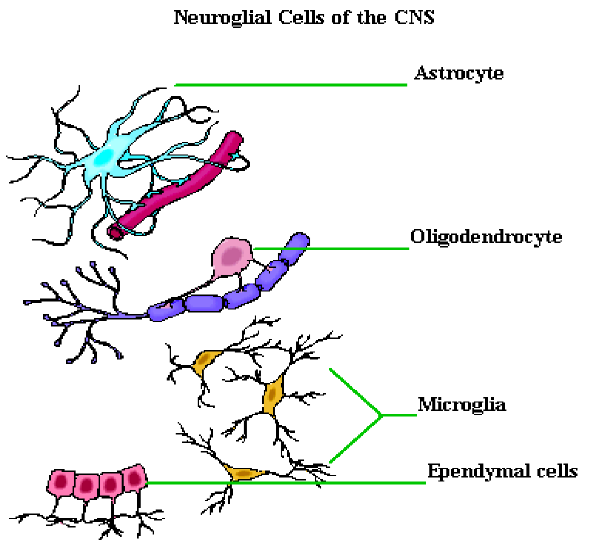
- Neurons react in “spikes”,
 not real numbers

- Neurons maintain/change
 their states over time

- No one knows for sure how
 they “train”

- Neuroglial cells are important
 But noone knows, why

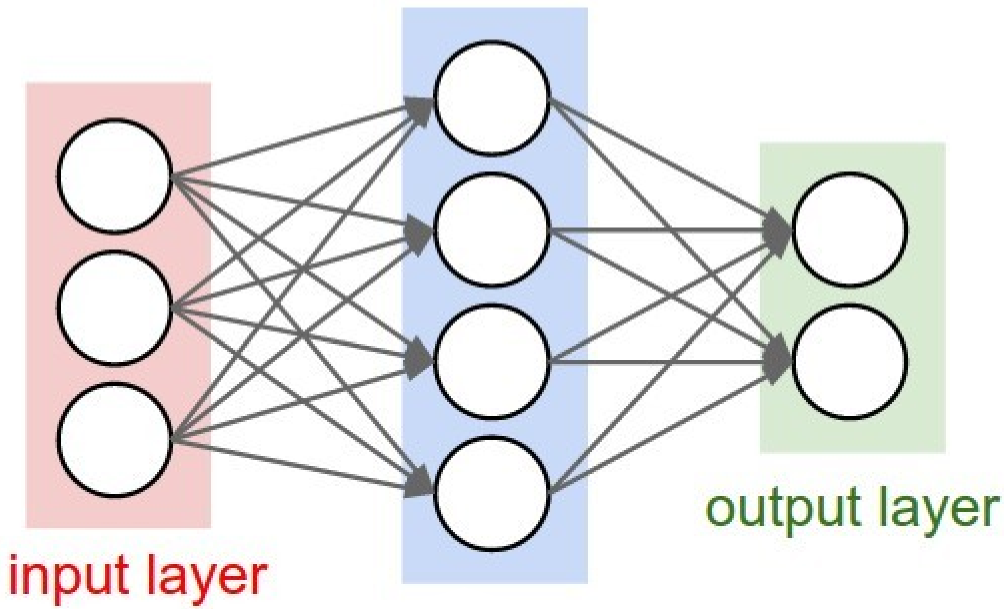




Connectionist phrasebook

- 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”

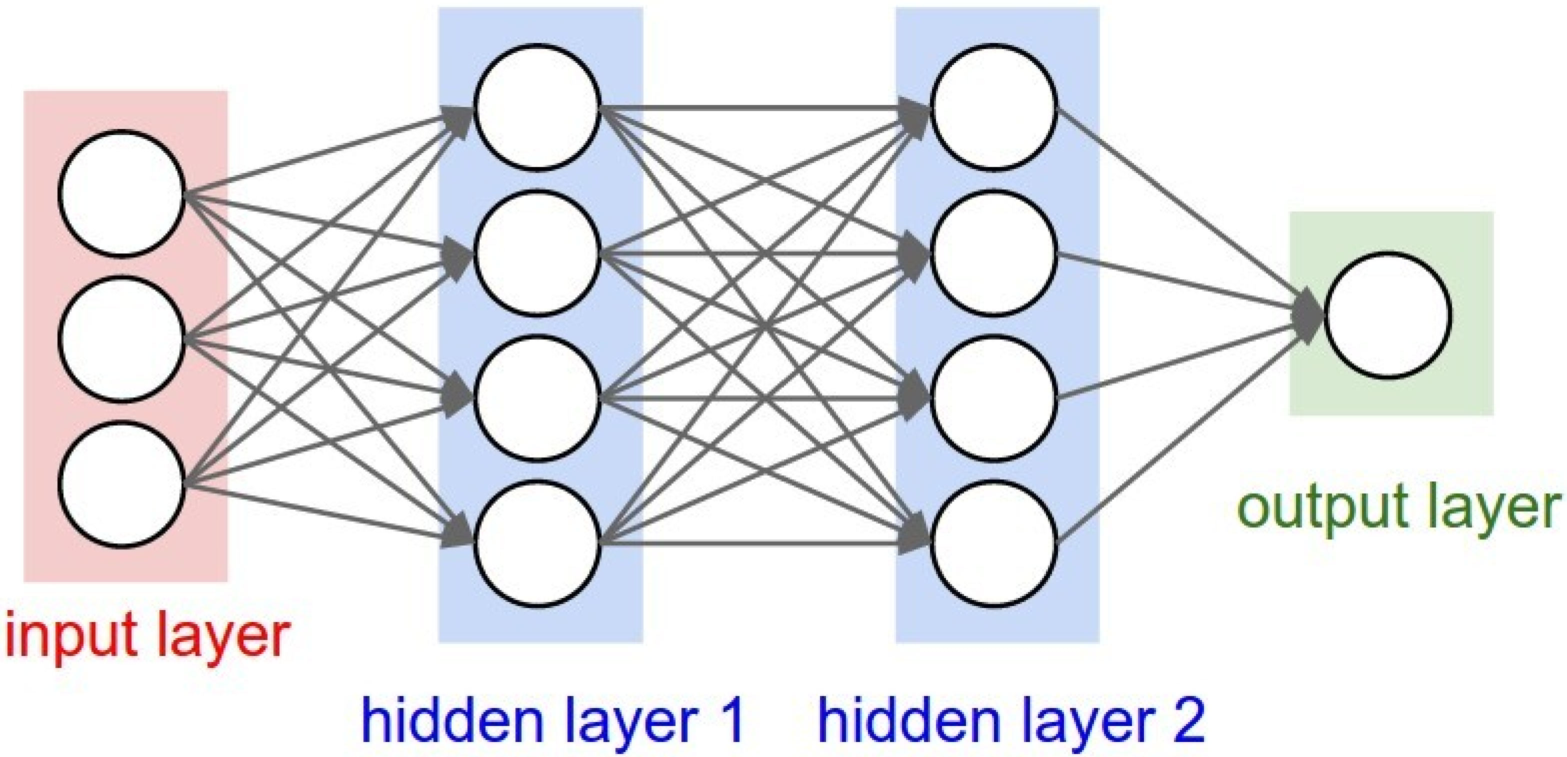
Connectionist phrasebook



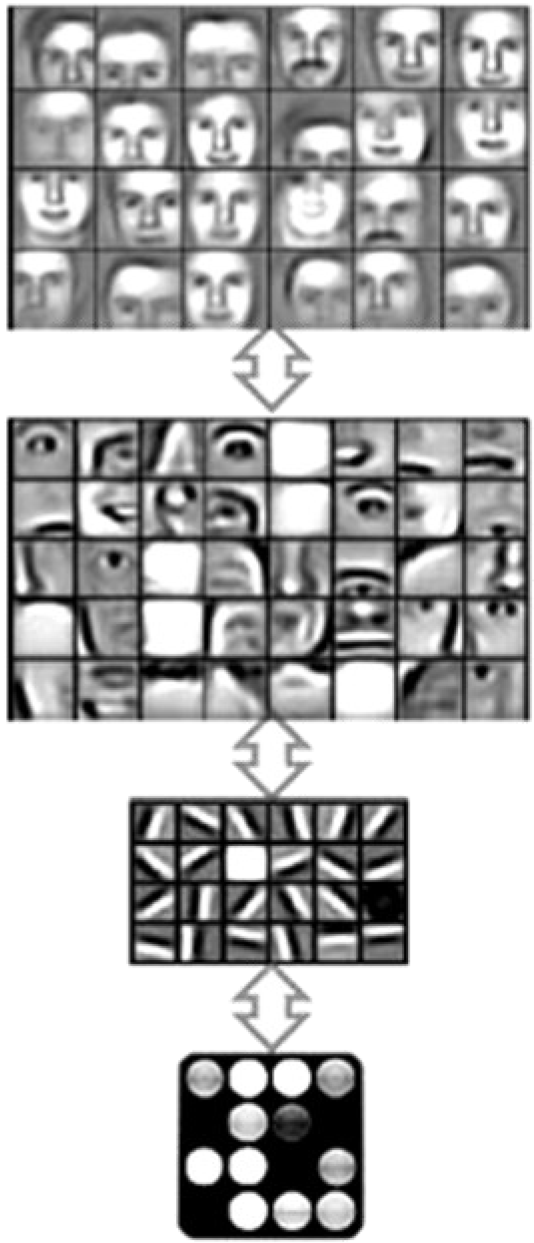
Dense layer
with a nonlinearity

- “Train it via backprop!”

Connectionist phrasebook



How do we train it?



Discrete Choices

⋮

Layer 2 Features

Layer 1 Features

Original Data

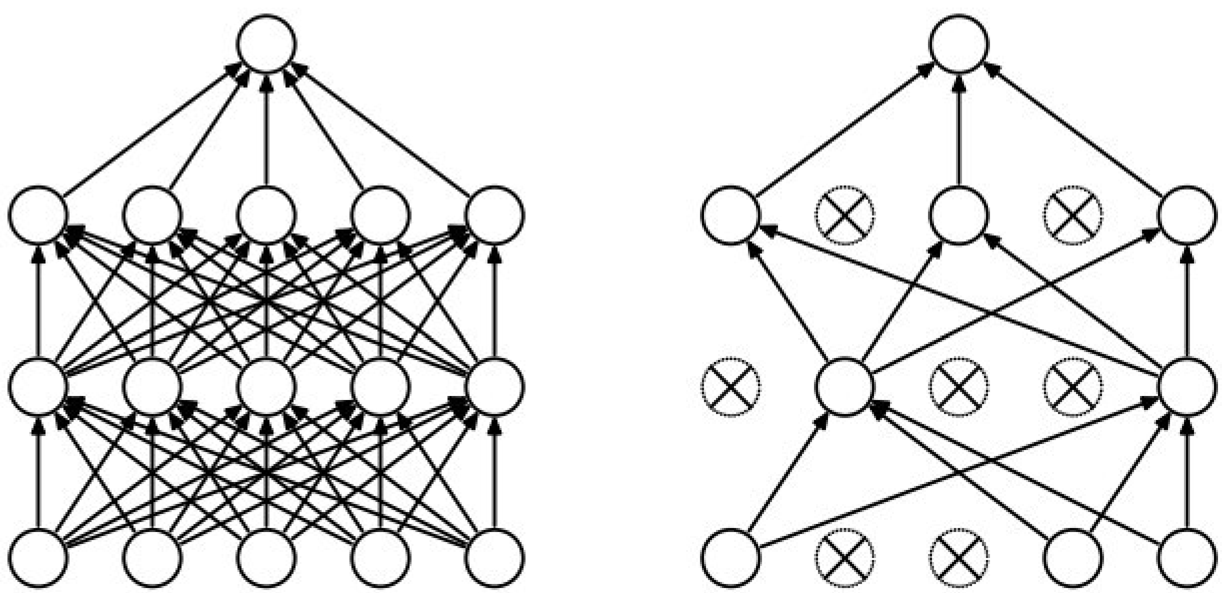
Potential caveats?

Potential caveats?

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

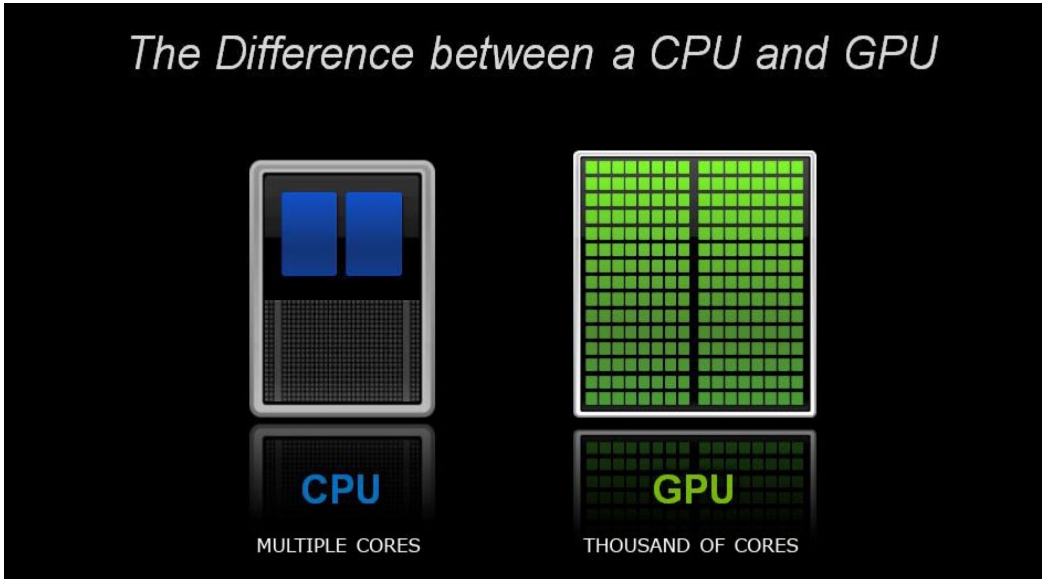
Regularization

- L1, L2, as usual
- Dropout

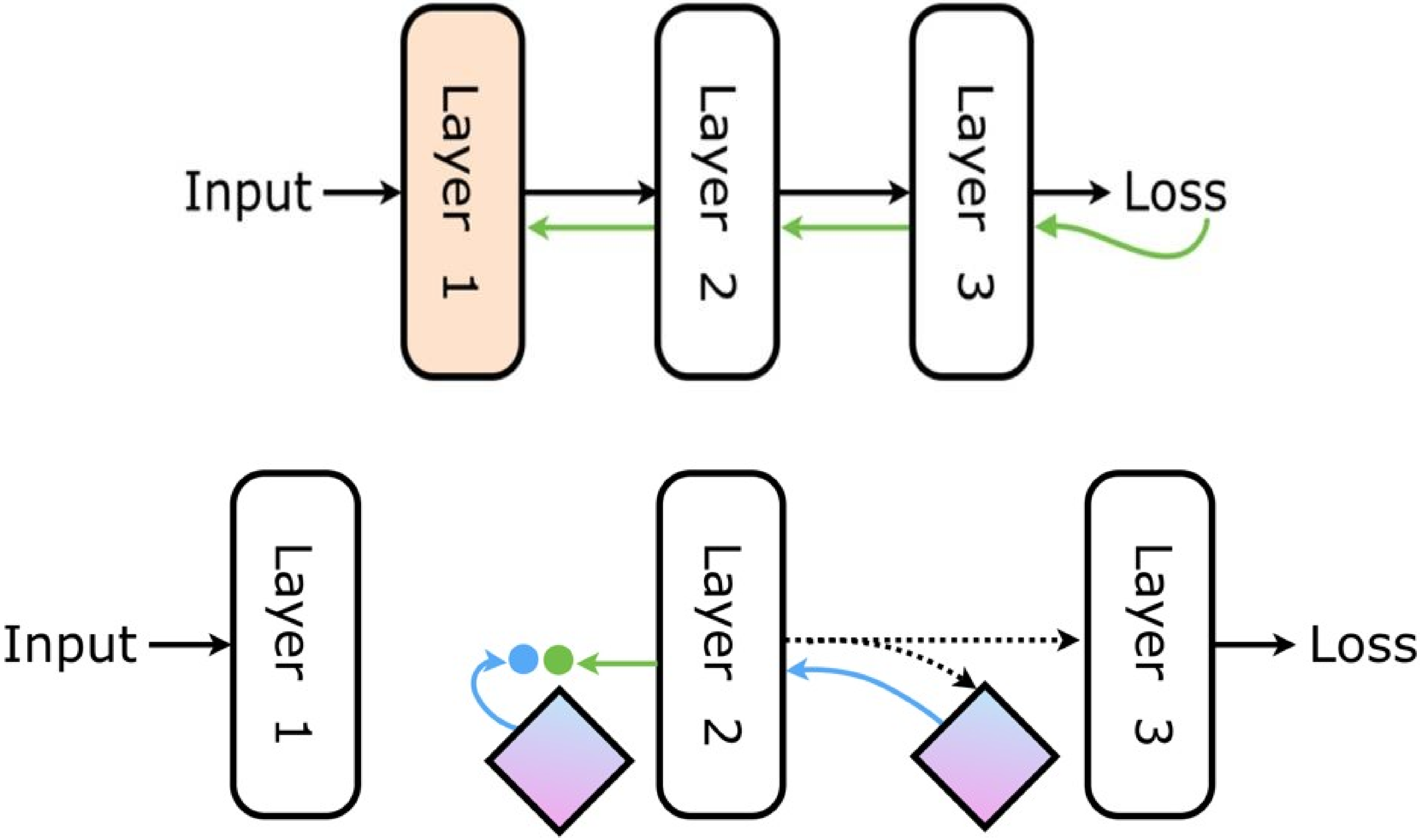


(a) Standard Neural Net (b) After applying dropout.

Computation



Is backprop the only choice?



Nuff

Let's code some neural networks!