Deep learning
Episode 1

Neural networks 101

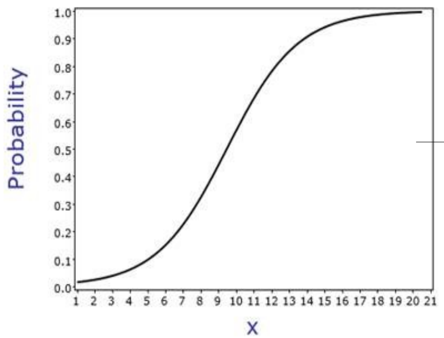
Maxim Borisyak, Alexander Panin, Andrey Ustyuzhanin

Yandex Data Factory

LAMBDA

British Hedgehog Preservation Society

Recap: logistic regression

X → Wx +b → 

→ P(y)

Gradient descent

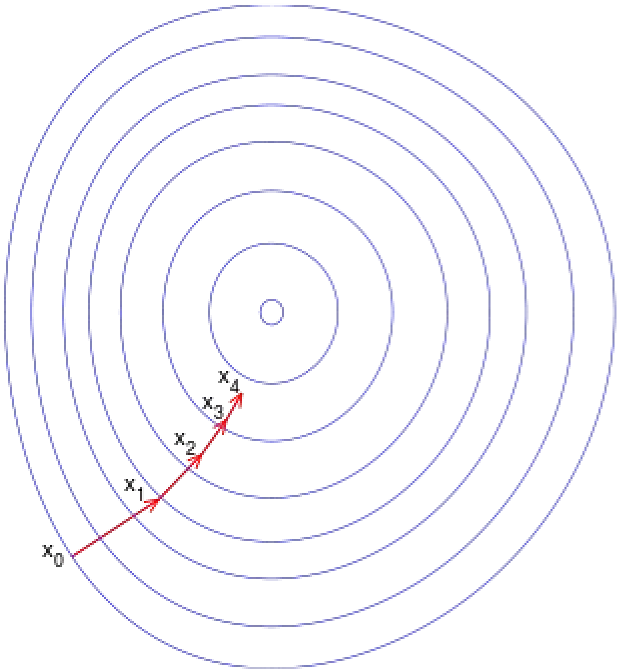
$$P(y|x)=\sigma(w\cdot x+b)$$

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

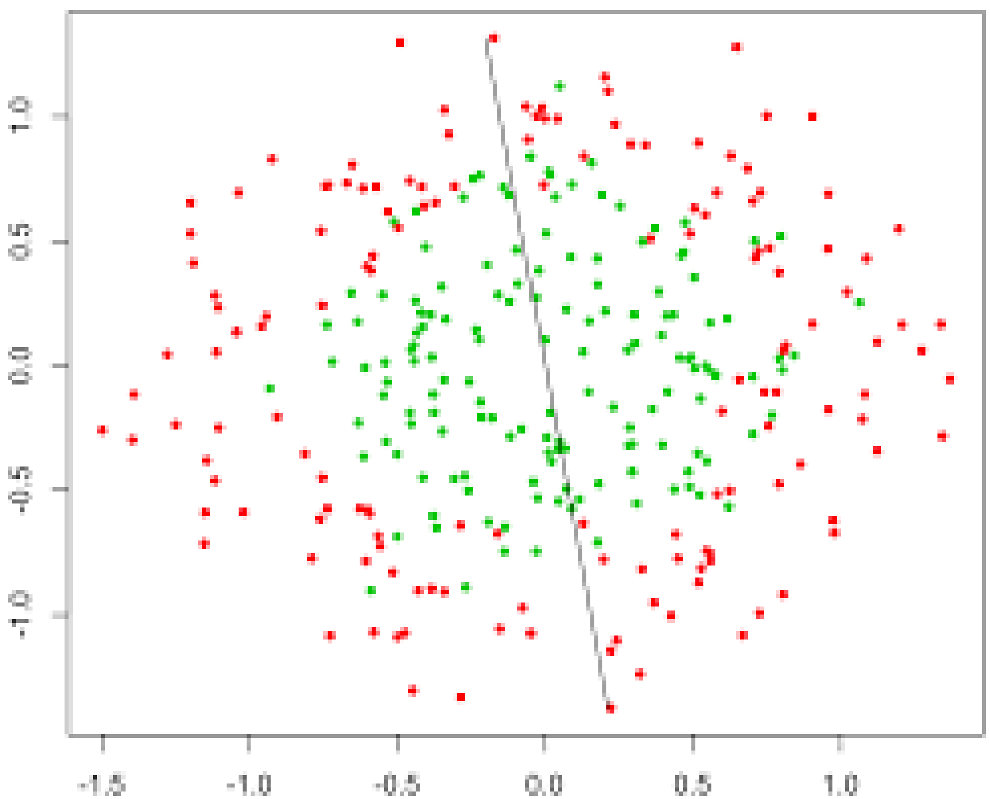
Repeat until convergence

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

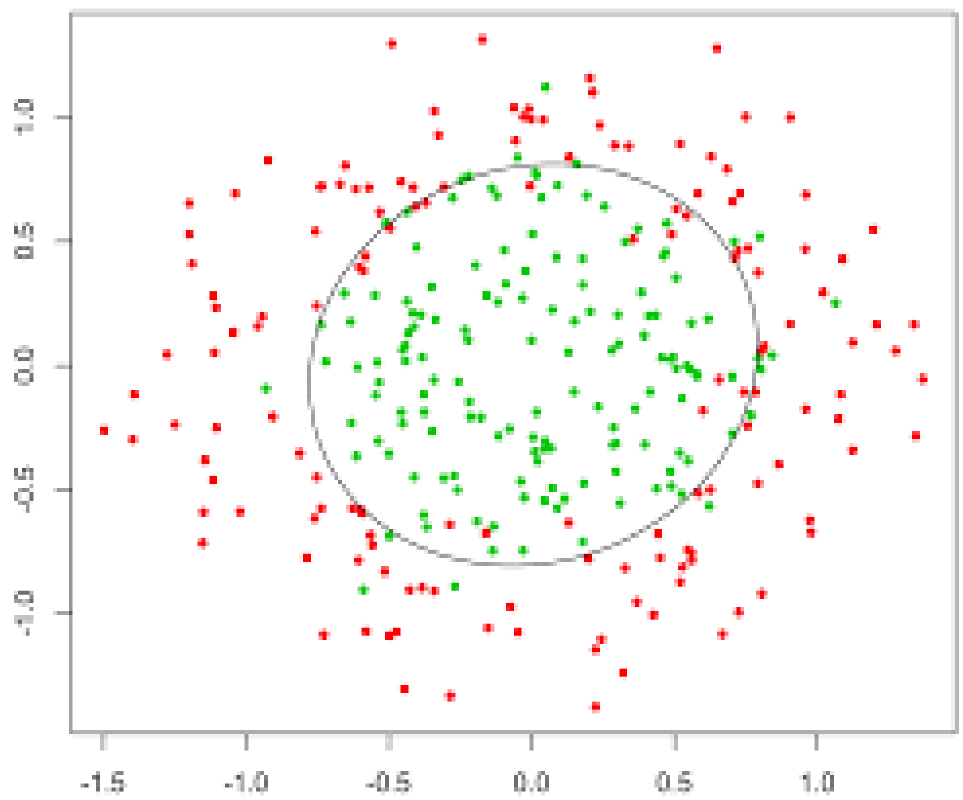
ϴ~{W,b}



Nonlinear dependencies



What we have



What we want

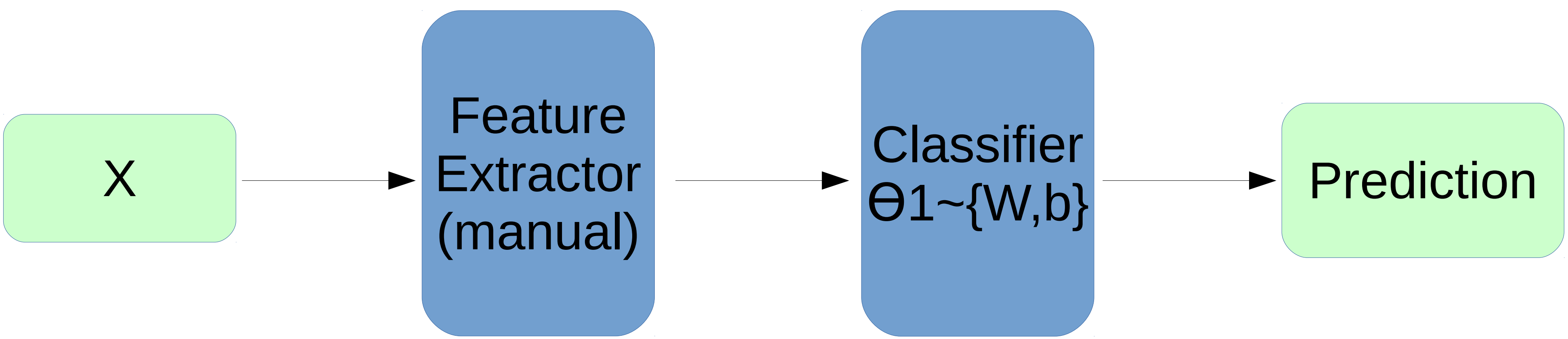
- How to get that?

Feature extraction

Loss, for example:

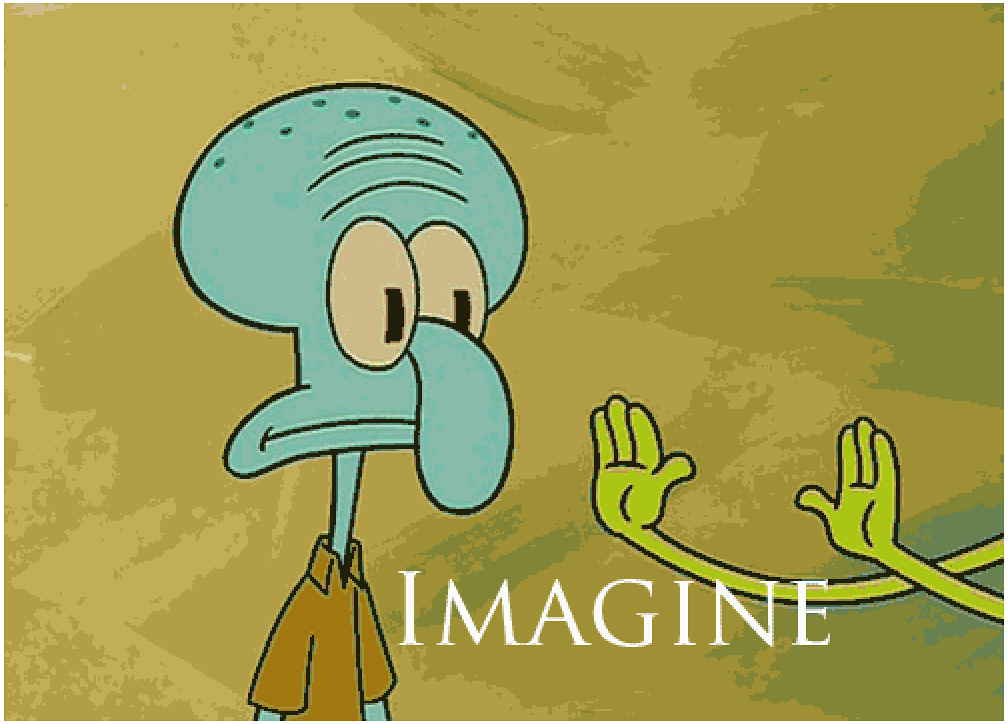
$$L = -\sum_i y_i \log P(y|x_i) + (1 - y_i) \log(1 - P(y|x_i))$$

Model:



Training:

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



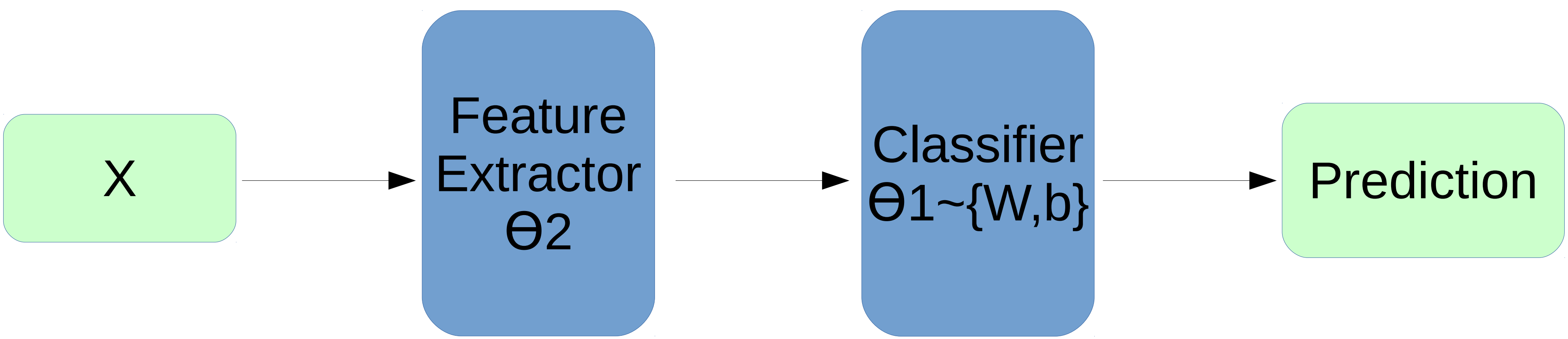
Features would tune to your problem automatically!

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:



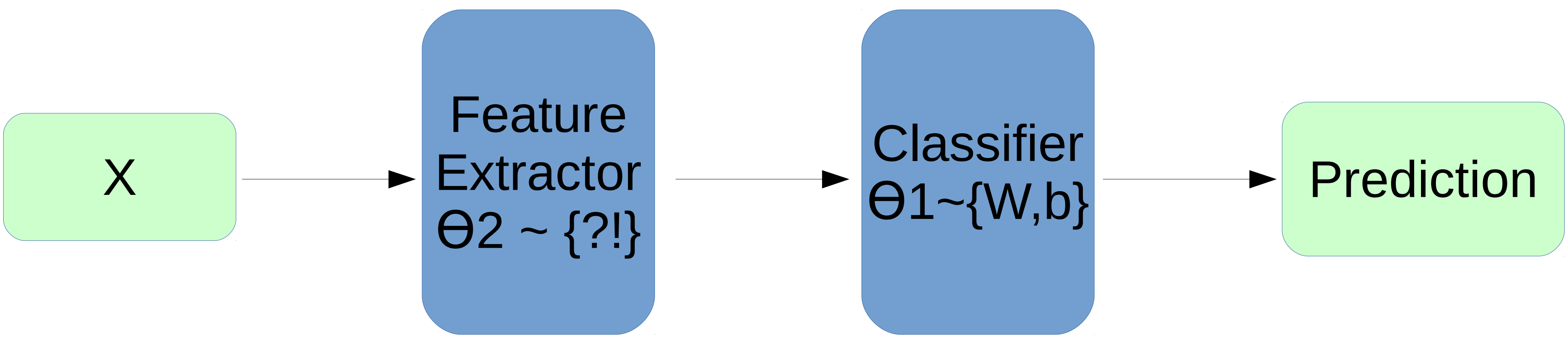
Training: ? $\underset{\theta_1}{argmin}\, L(y, P(y|x))$

What do we want, exactly?

Loss, for example:

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

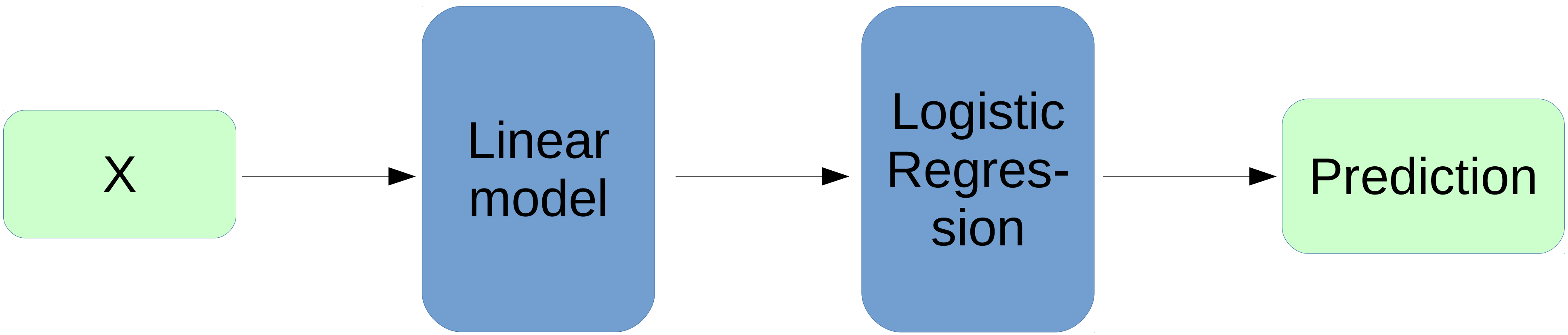
Model:



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

Try linear

Model:

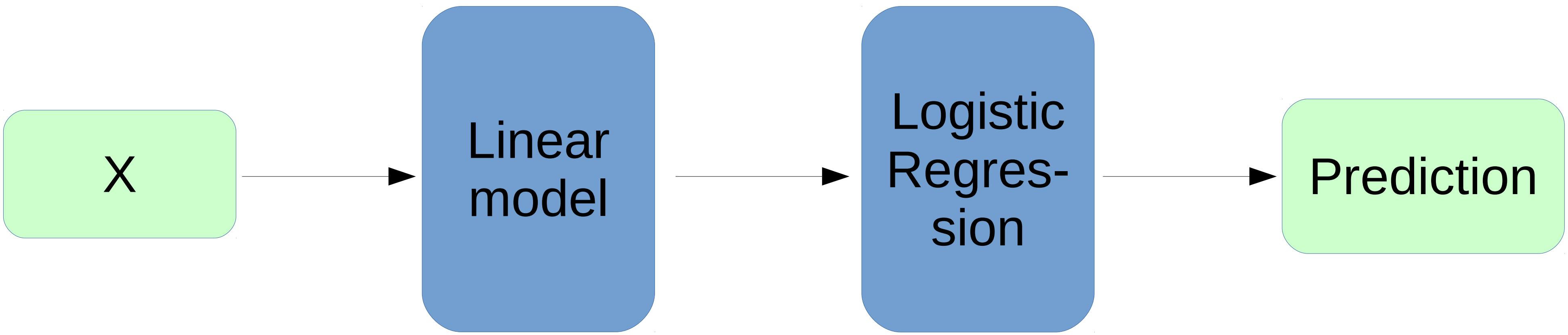


$$h_j = \sum_i w_{ij}^h x_i + b_j^h \qquad y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

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

Try linear

Model:



$$h_j = \sum_{i} w^h_{ij} x_i + b^h_j \qquad j \in \{1,2,\ldots,n\}$$

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

Output:

$$P(y|x) = \sigma\left(\sum_j w^o_j \left(\sum_i w^h_{ij} x_i + b^h_j\right) + b^o\right)$$

Is it any better than logistic regression?

Try linear

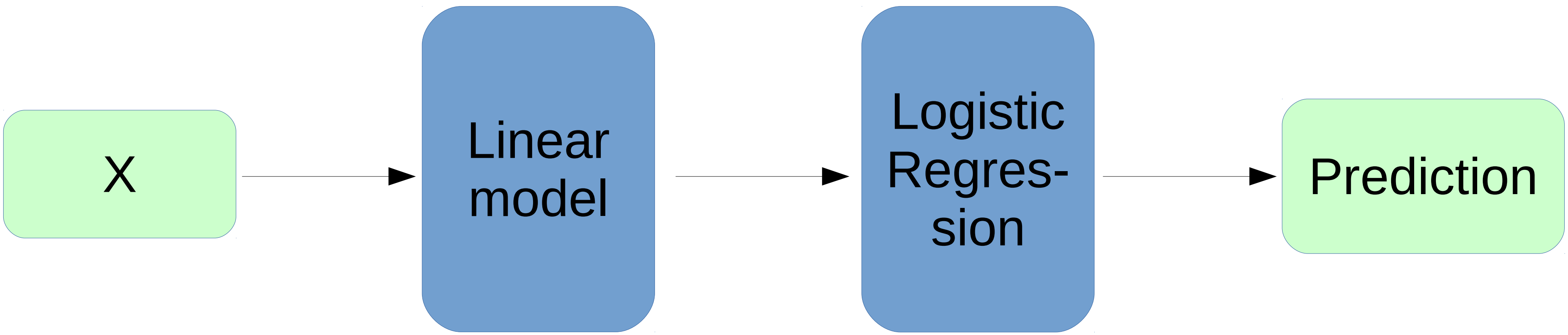
$$P(y|x)=\sigma\left(\sum_j w_j^o\left(\sum_i w_{ij}^h x_i+b_j^h\right)+b^o\right)$$

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

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

Try linear

Model:



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

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

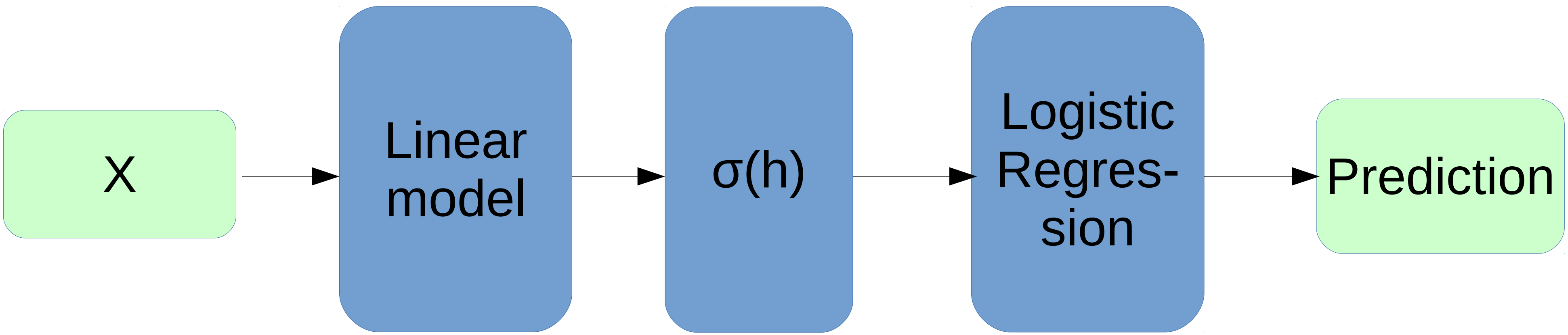
Output:

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

Is it any better than logistic regression?

Nonlinearity

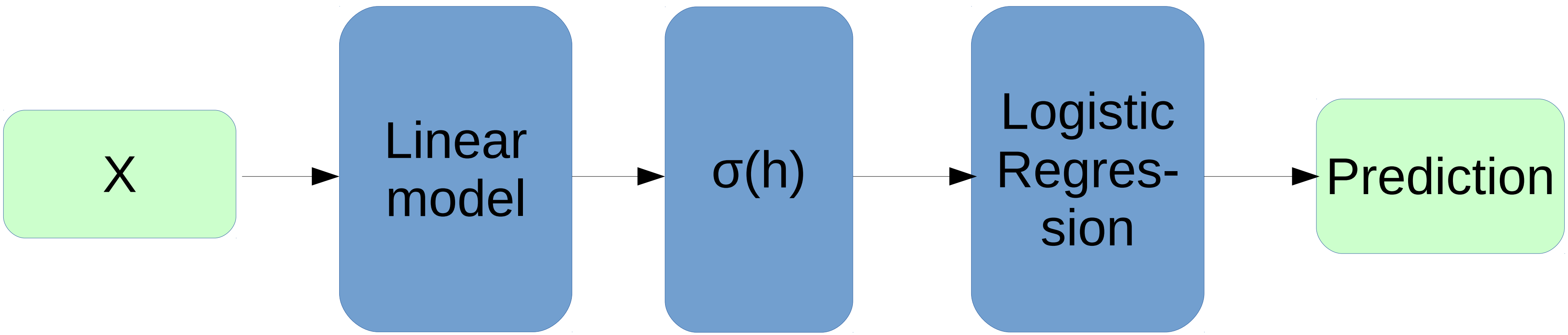
Model:



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

Nonlinearity

Model:



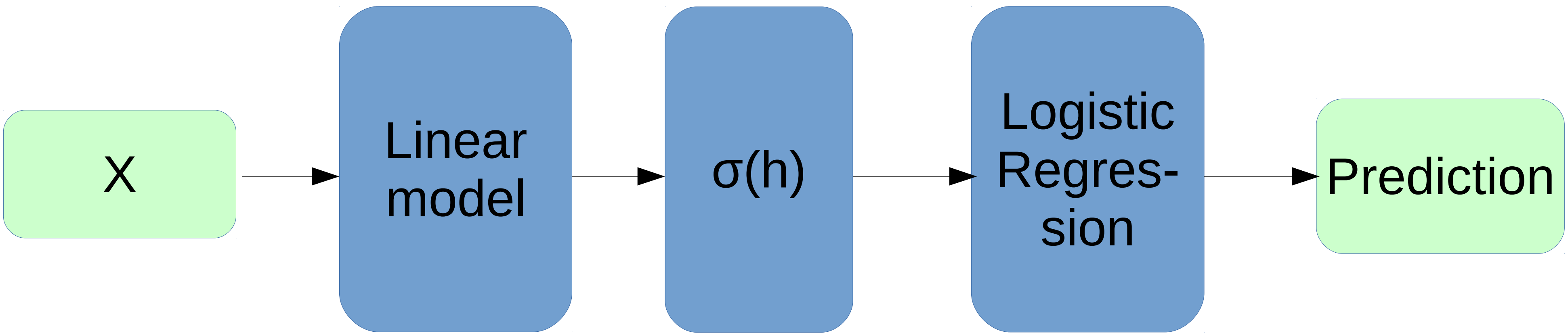
$$h_j = \sigma\left(\sum_{\substack{i \\ j \in \{1,2,\ldots,n\}}} w_{ij}^h x_i + b_j^h\right) \qquad y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

Output:

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

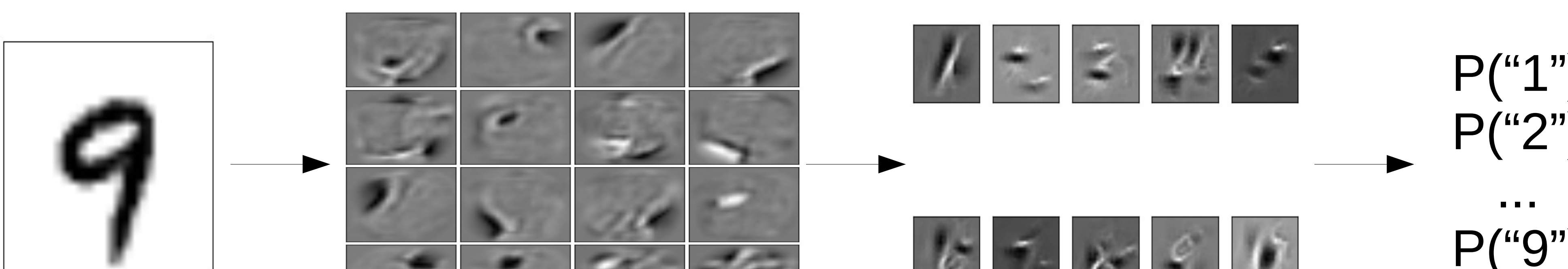
Nonlinearity

Model:



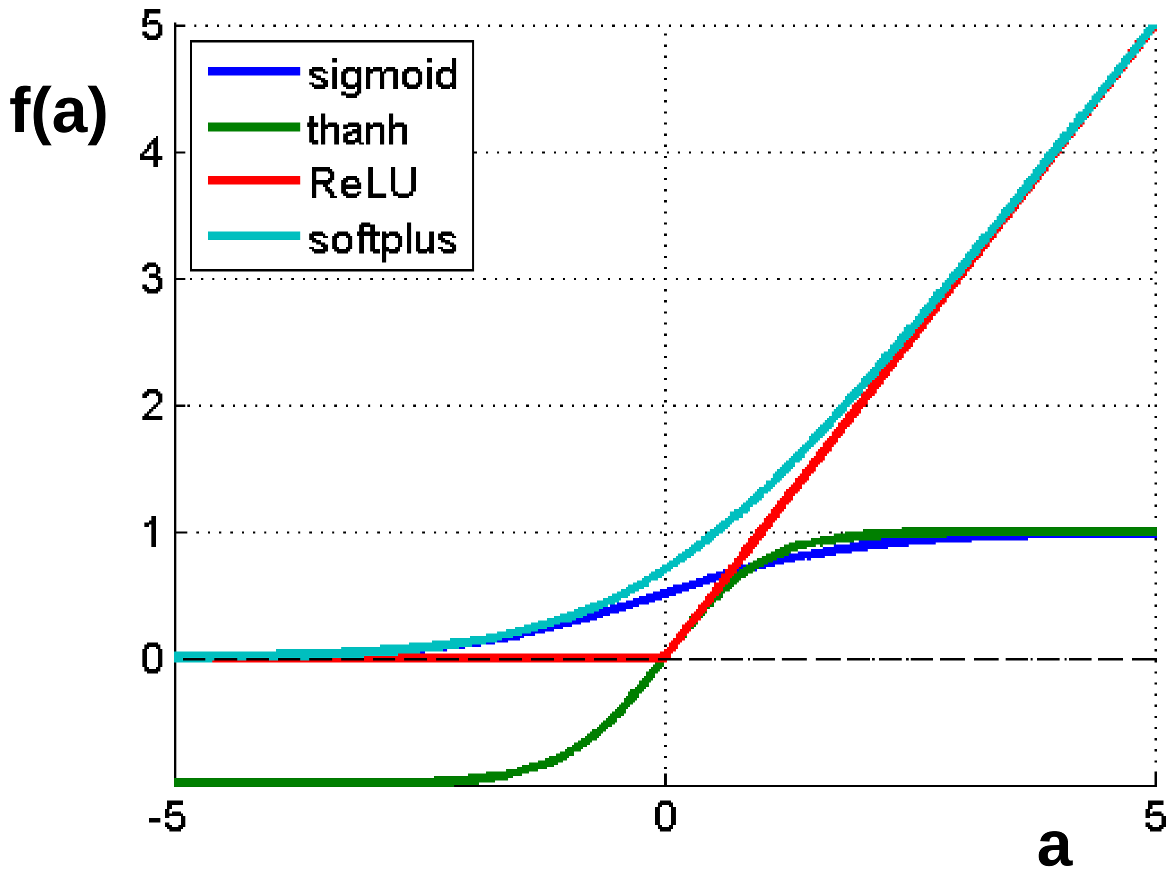
$$h_j = \sigma\left(\sum_{i\in\{1,2,\ldots,n\}} w^h_{ij} x_i + b^h_j\right)$$

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

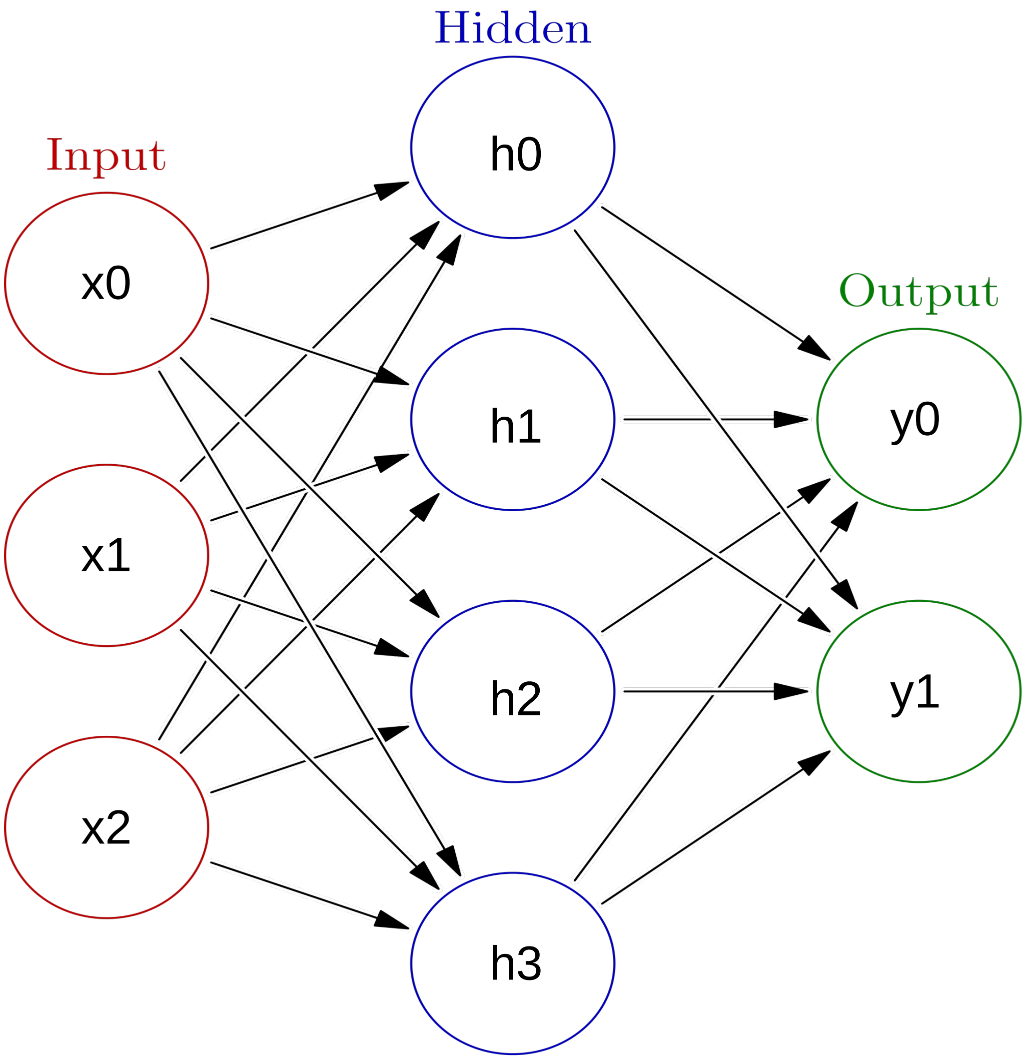


Nonlinearity

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



Initialization, symmetry problem

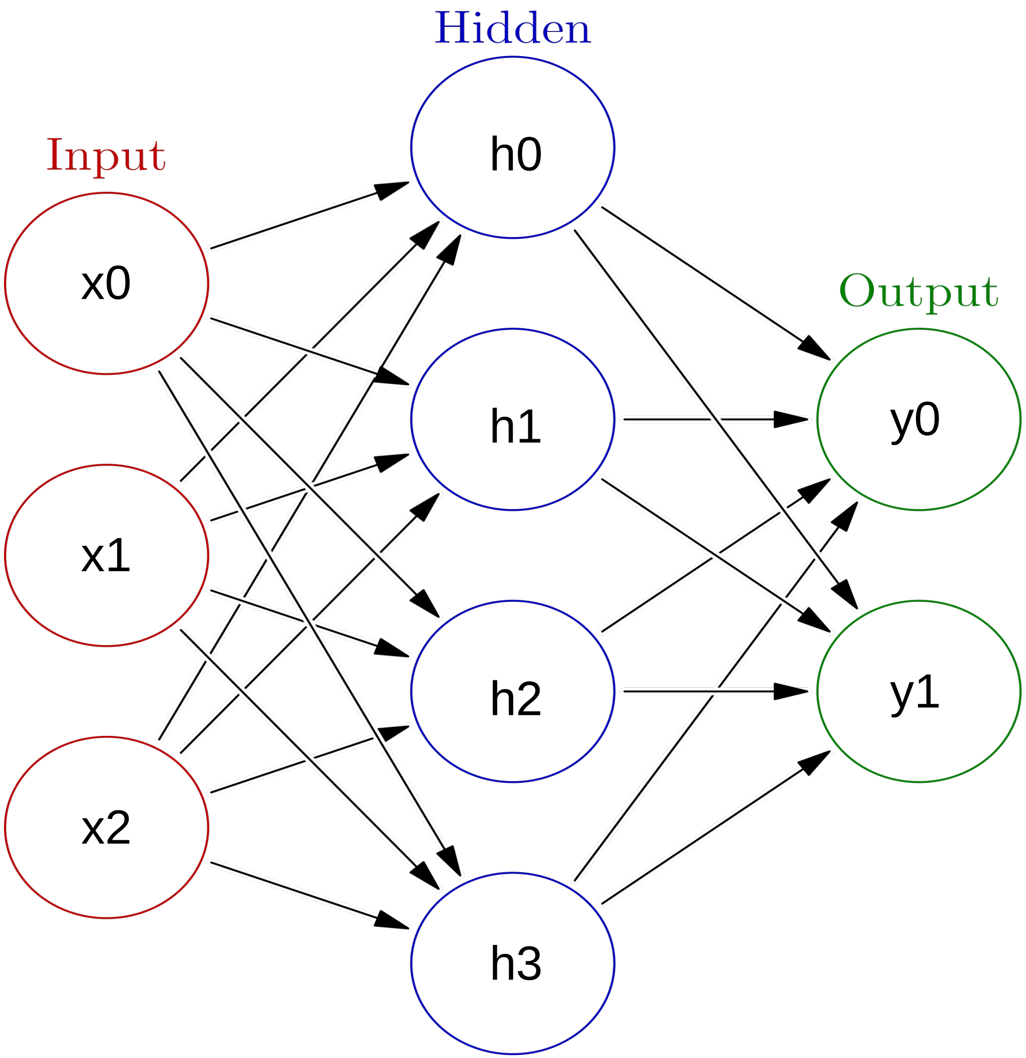


- Initialize with zeros

  $W \leftarrow 0$

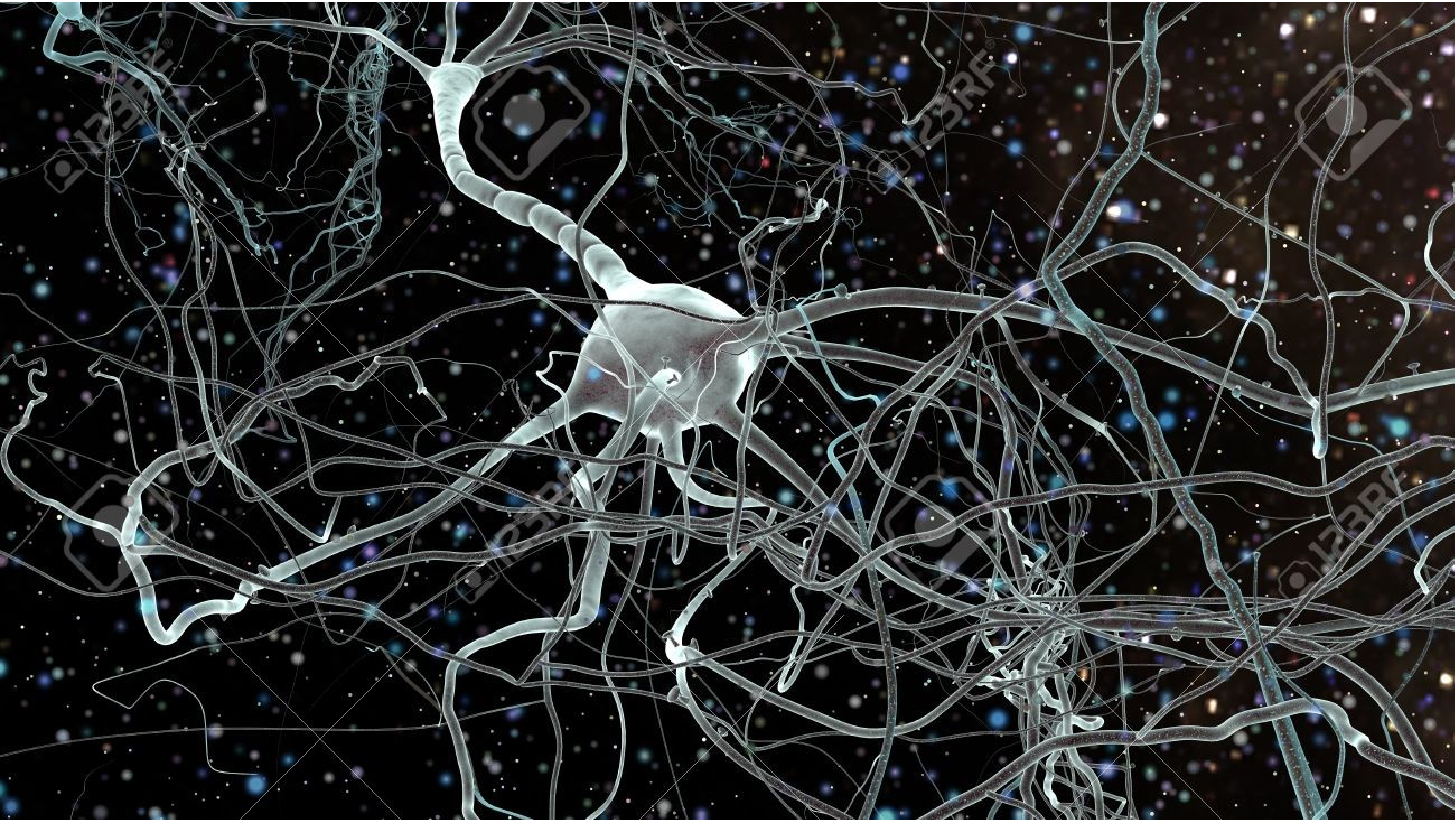
- What will the first step look like?

Initialization, symmetry problem

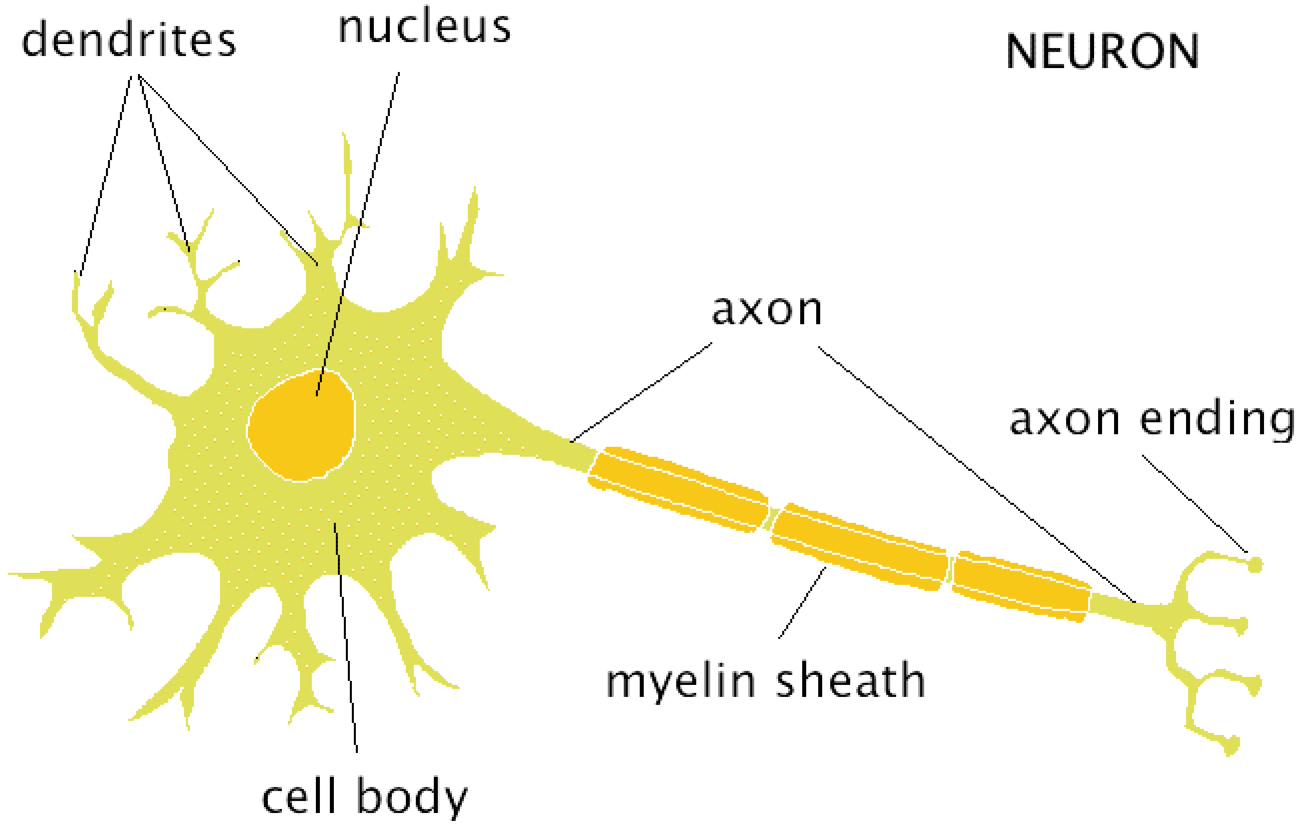


- Break the symmetry!
- Initialize with random numbers!
  $W \leftarrow N(0,0.01)$?
  $W \leftarrow U(0,0.1)$?
- Can get a bit better for deep NNs

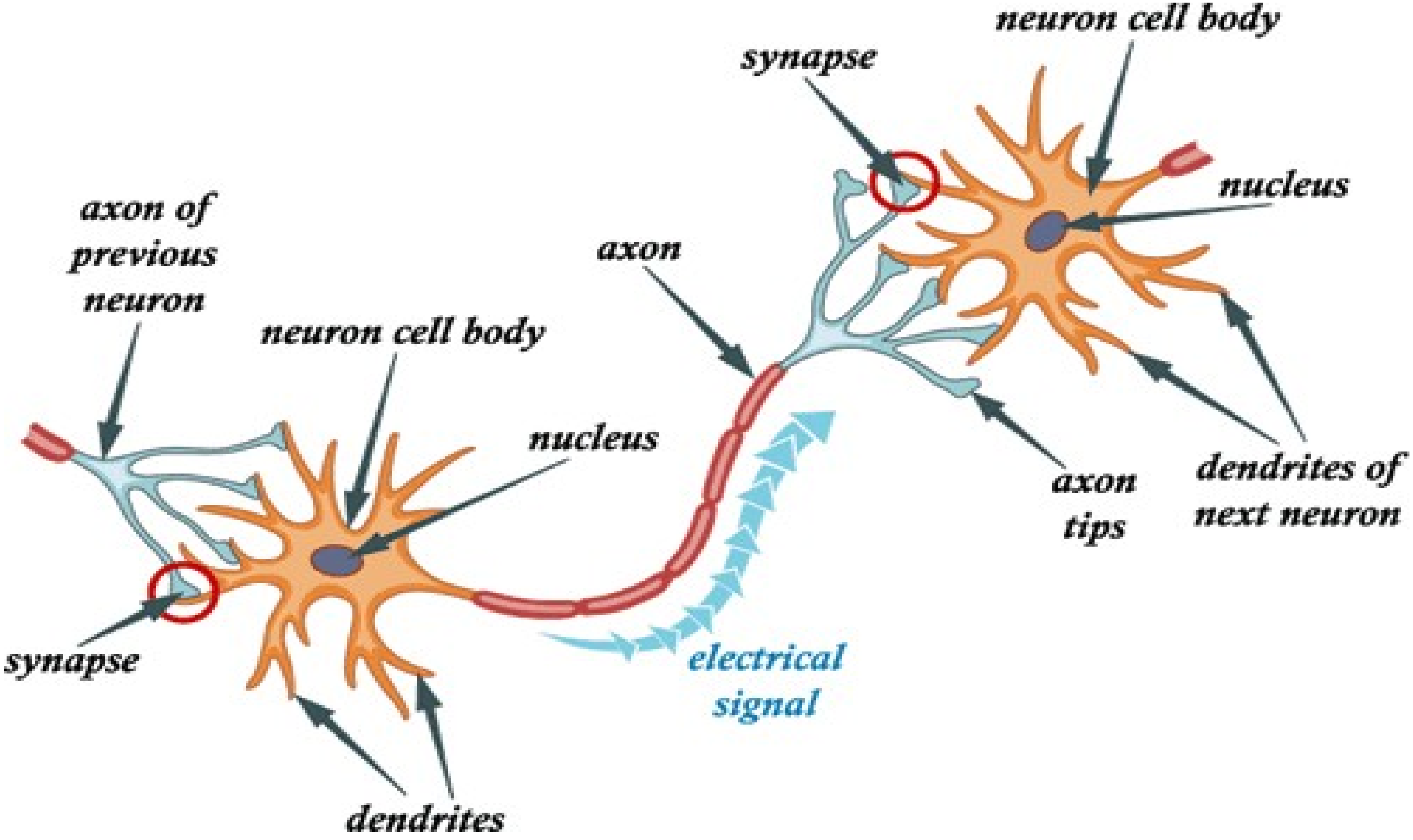
Biological inspiration



Biological inspiration

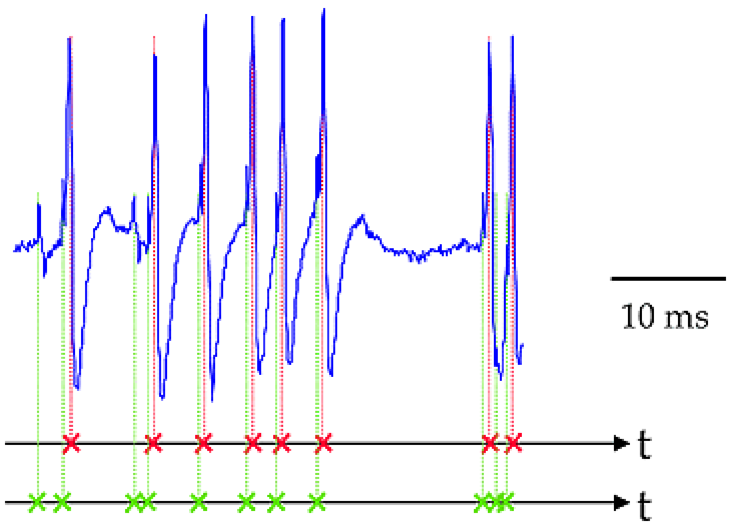


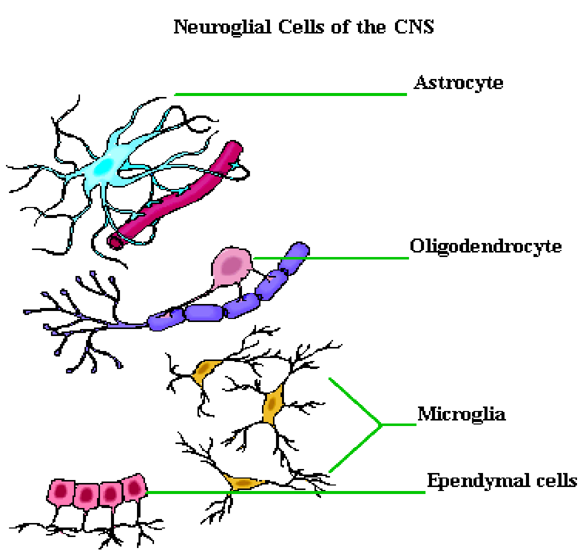
Biological inspiration



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

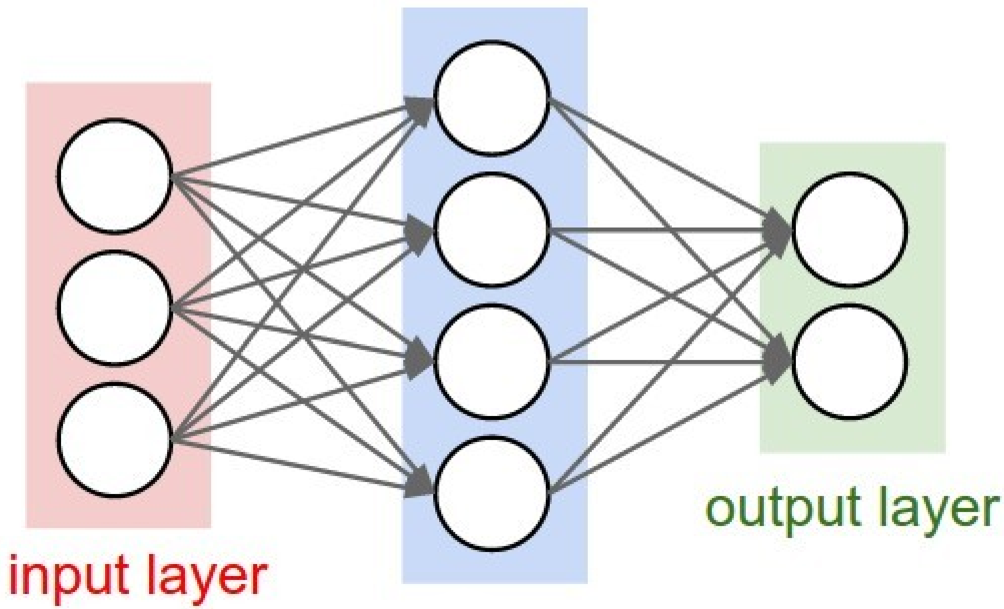




Connectionist phrasebook

- 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”

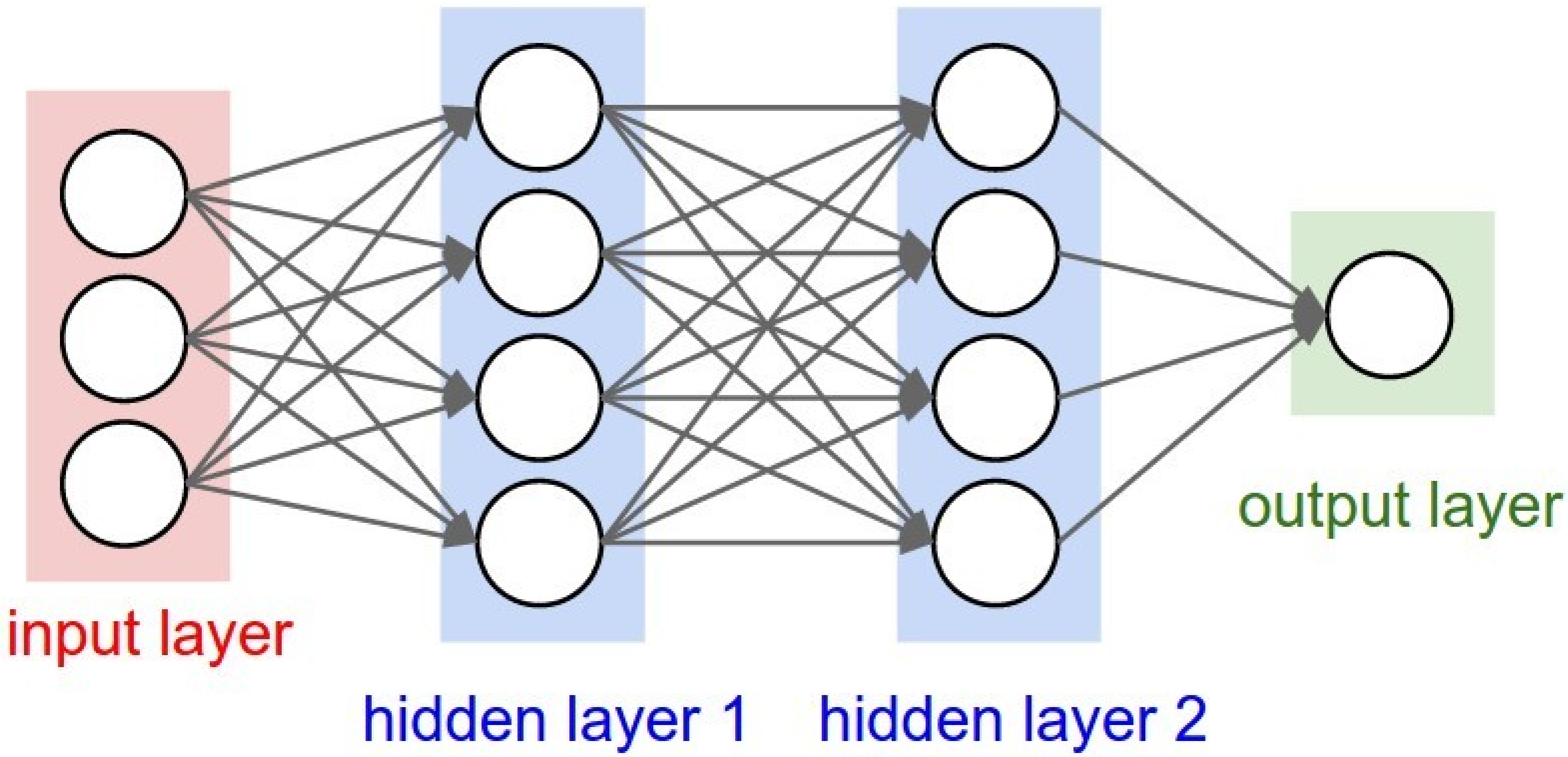
Connectionist phrasebook



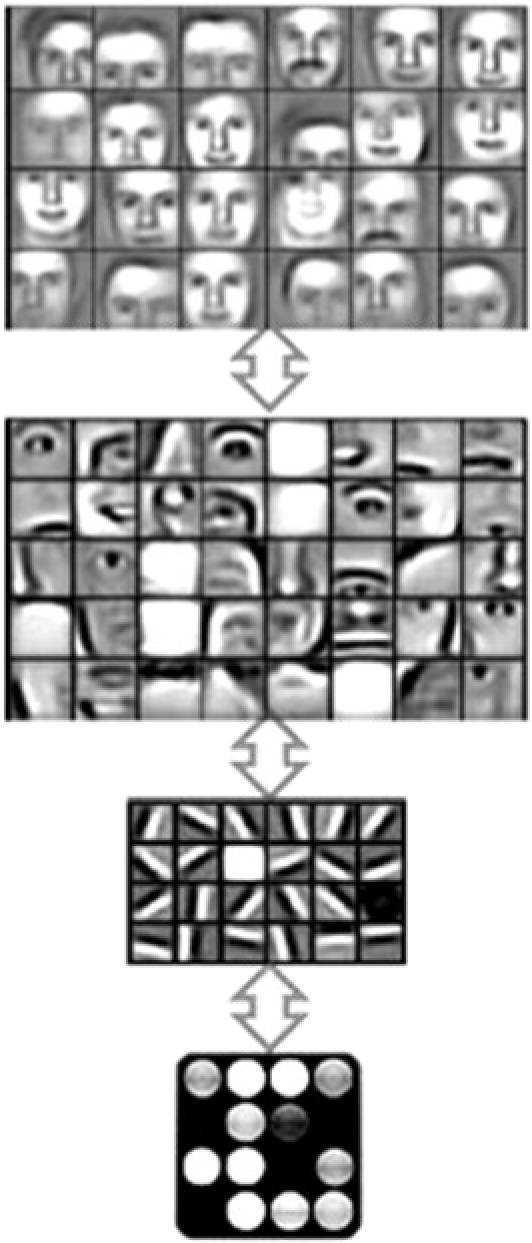
Dense layer
with a nonlinearity

- “Train it via backprop!”

Connectionist phrasebook



How do we train it?



Discrete Choices

⋮

Layer 2 Features

Layer 1 Features

Original Data

Potential caveats?

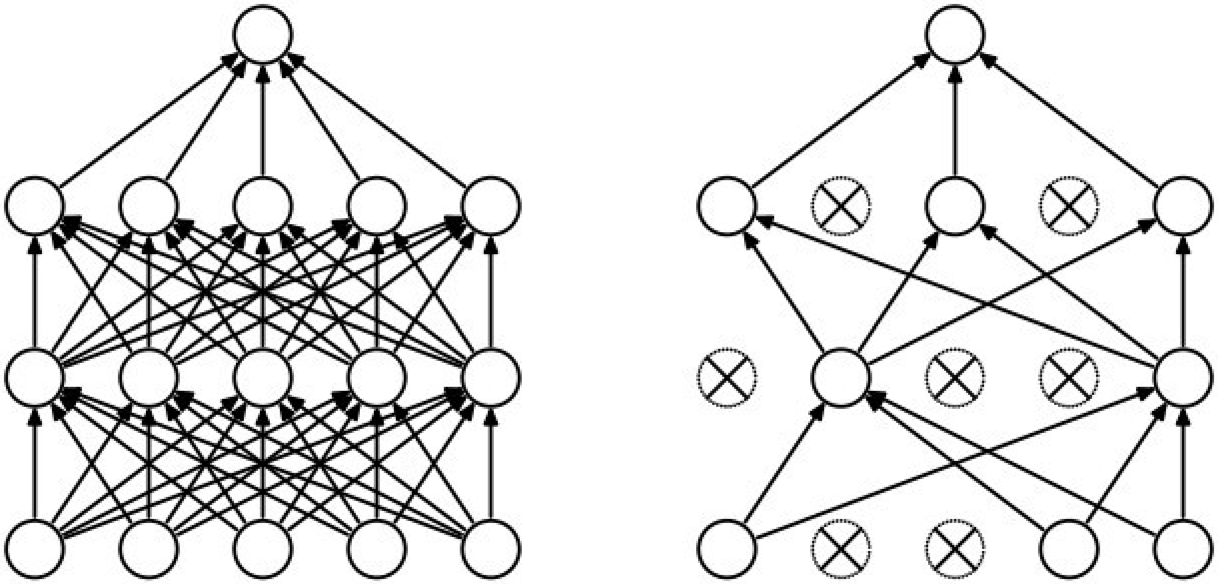
Potential caveats?

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

Regularization

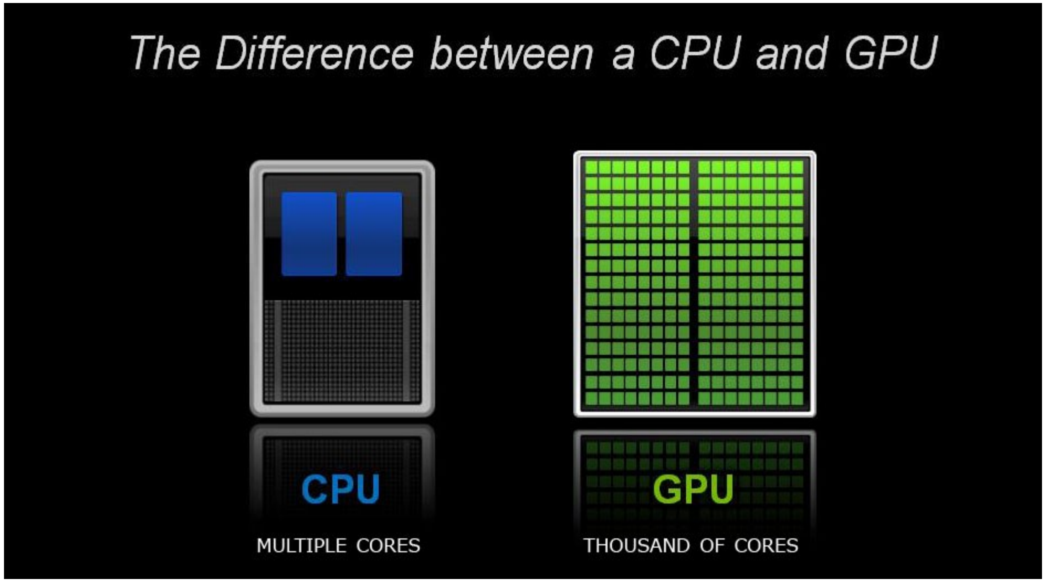
- L1, L2, as usual

- Dropout

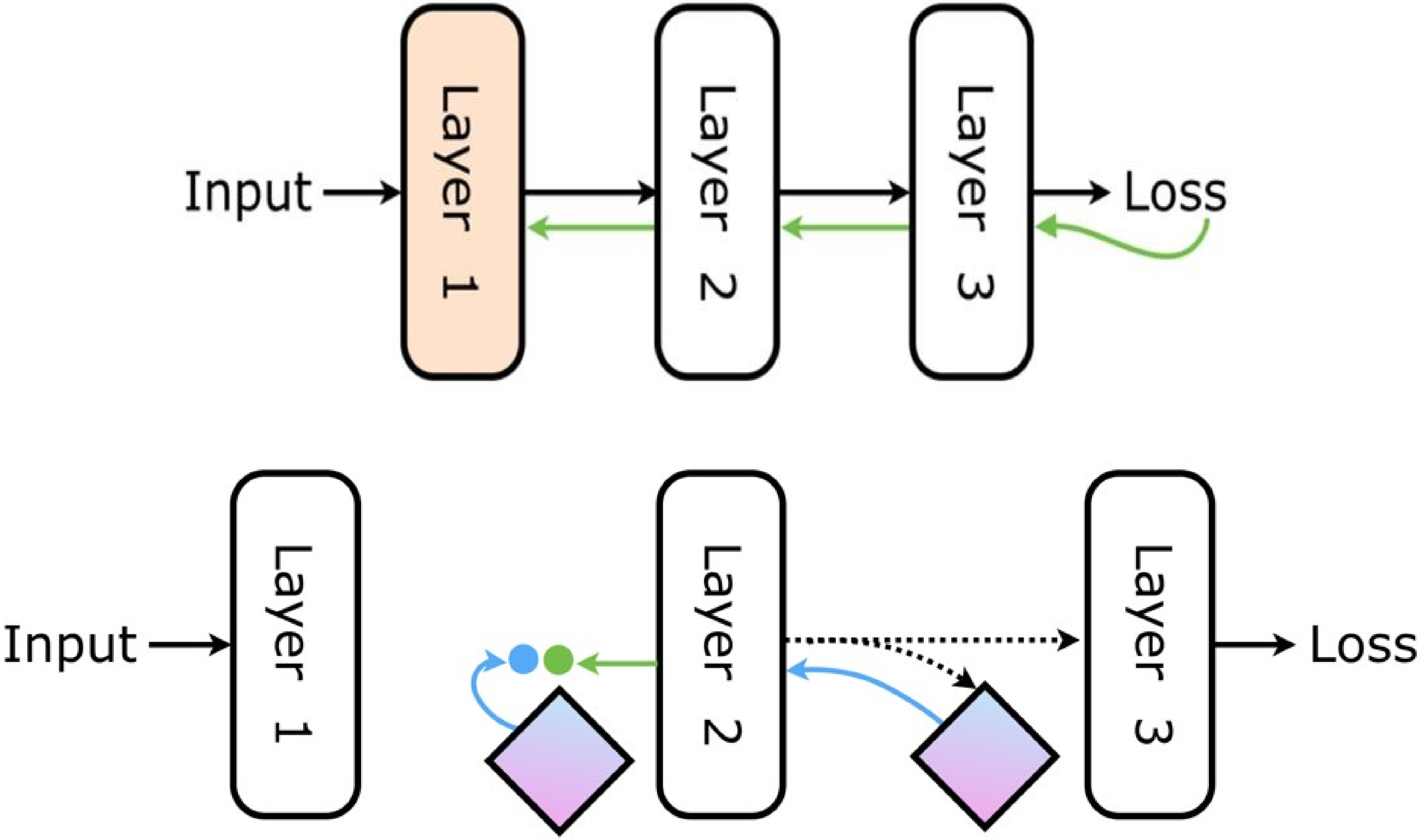


(a) Standard Neural Net (b) After applying dropout.

Computation



Is backprop the only choice?



Nuff

Let's code some neural networks!