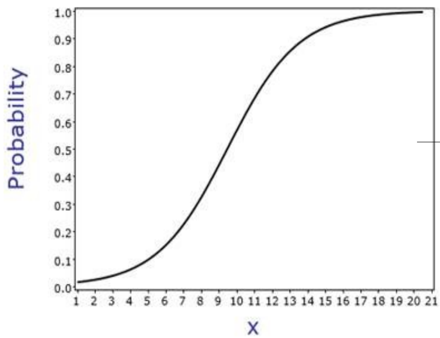
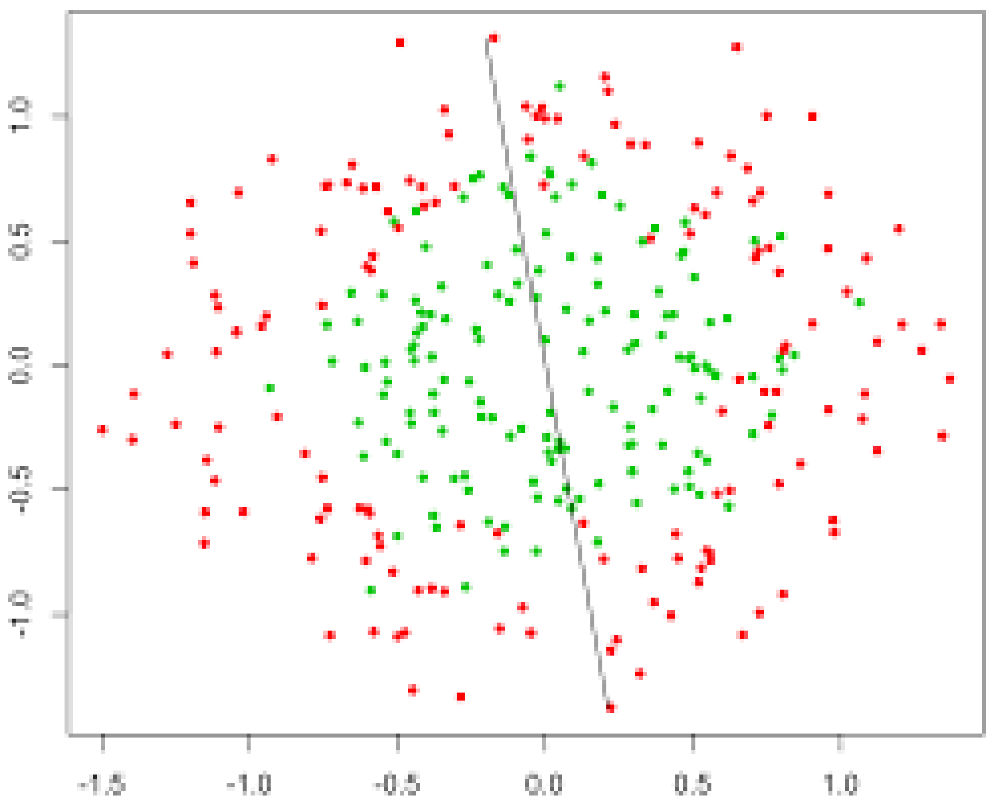
Intro to deep learning

Recap: logistic regression

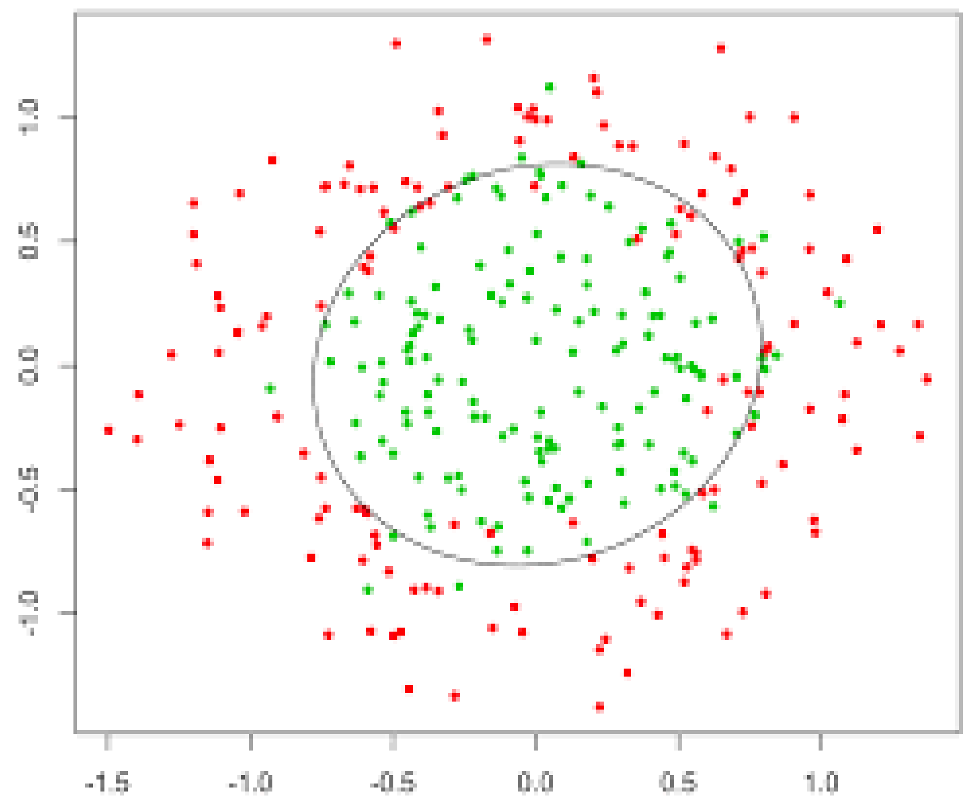
X → Wx +b → 

→ P(y)

Nonlinear dependencies



What we have



What we wan

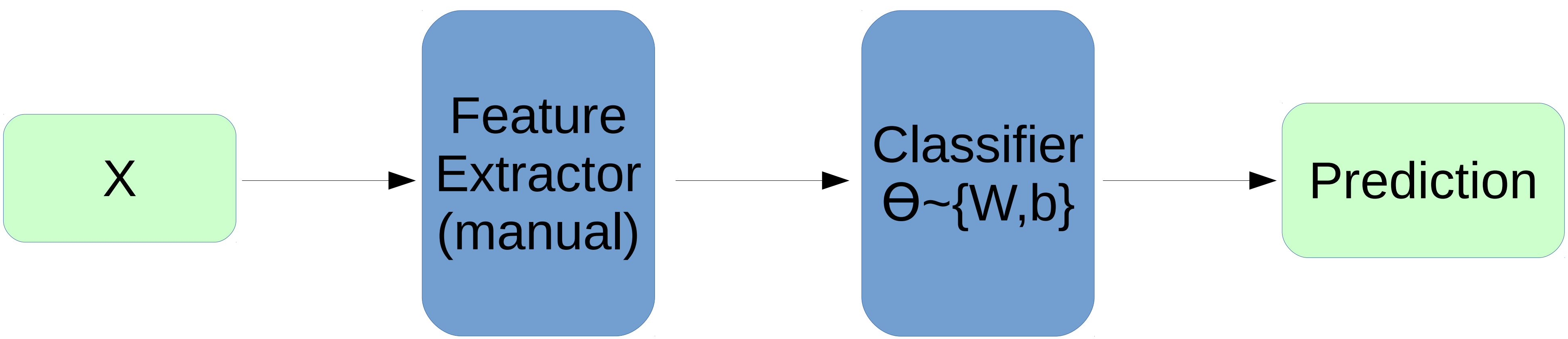
- How to get that?

Feature extraction

Loss, for example:

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

Model:



Training:

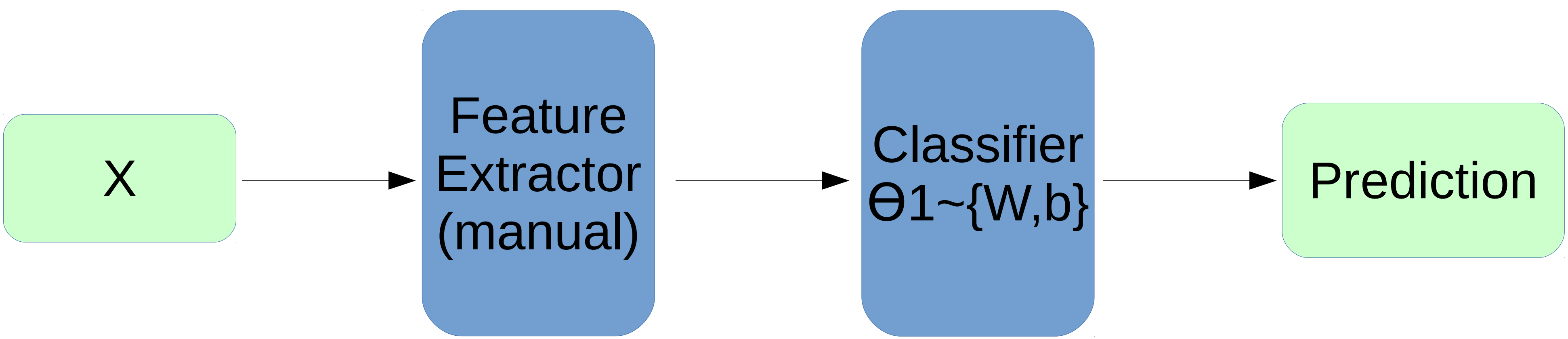
$$argmin_{\theta} L(y, y_{pred}(X, \theta))$$

Feature extraction

Loss, for example:

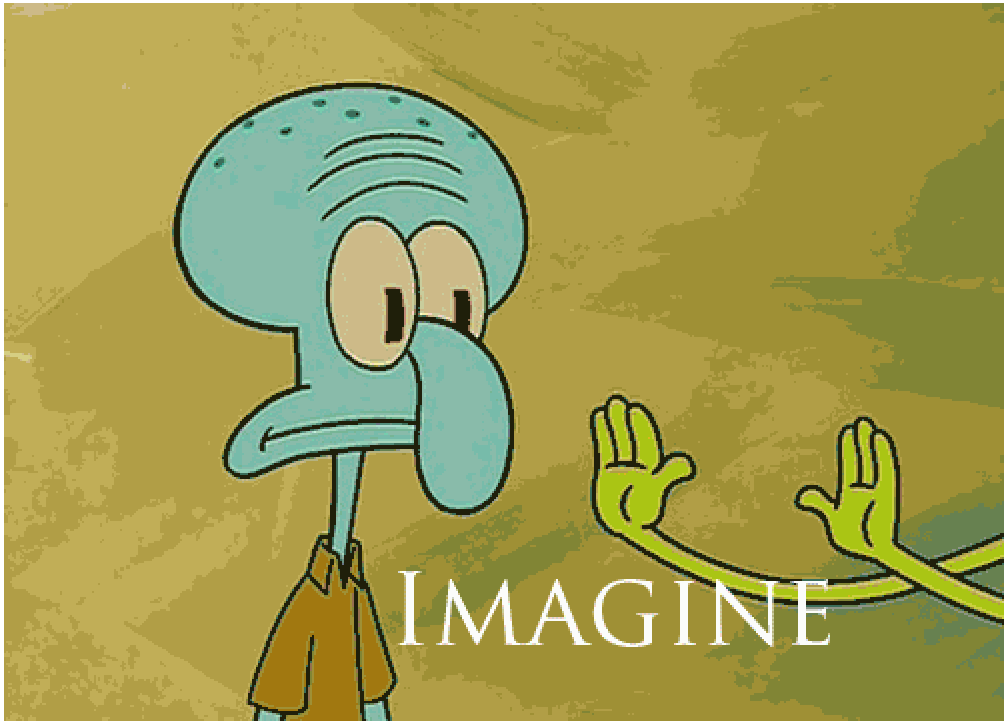
$$L(y, y_{pred}) = y \cdot \log y_{pred} + (1 - y) \cdot \log(1 - y_{pred})$$

Model:



Gradient:

$$\frac{\delta L(y, y_{pred}(X, \theta 1))}{\delta \theta 1}$$



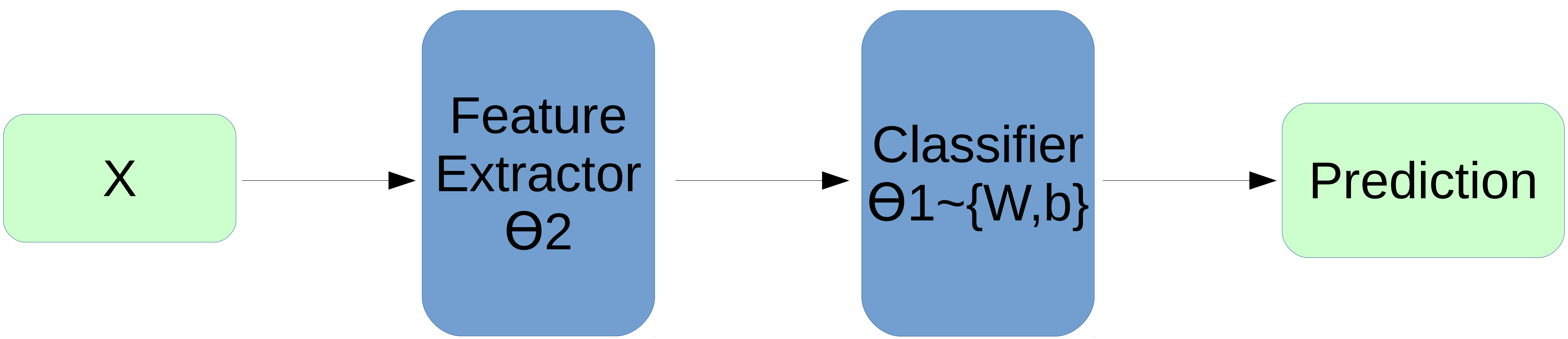
Features would tune to your problem automatically!

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:



Training: ?

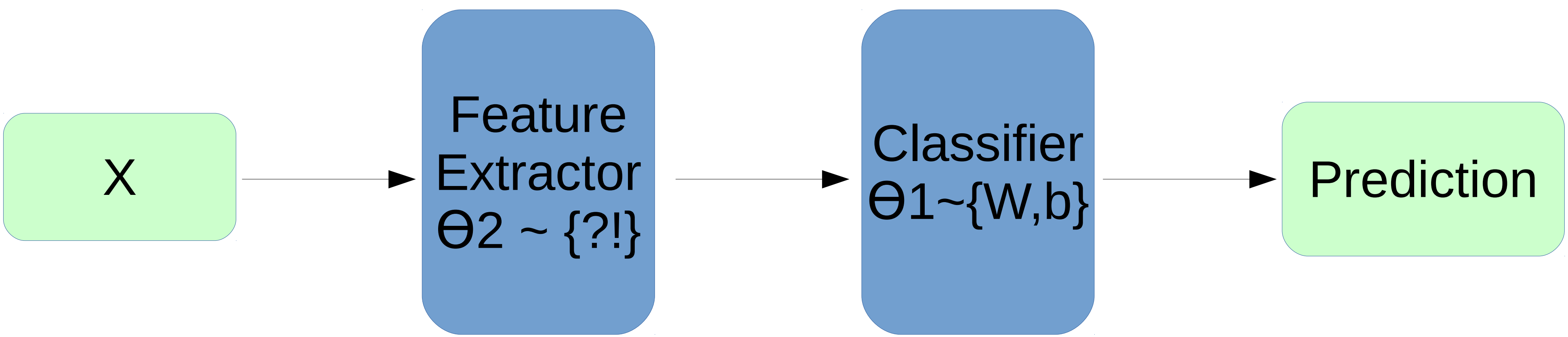
$$argmin_{\theta_1} L(y, y_{pred}(X, \theta_{1,} \theta_2))$$

What do we want, exactly?

Loss, for example:

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

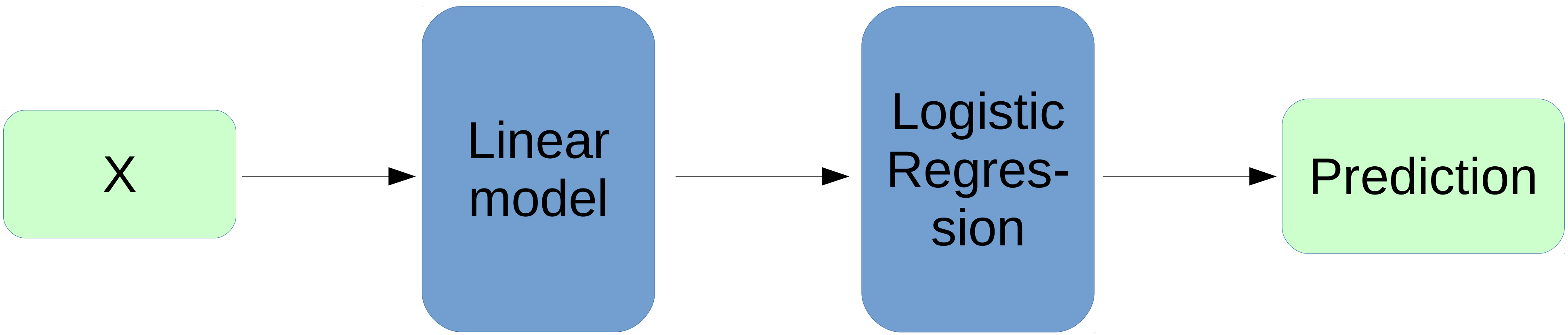
Model:



Gradients: $$\frac{\delta L(y, y_{pred}(X, \theta_{1,} \theta_2))}{\delta \theta_2} \quad \frac{\delta L(y, y_{pred}(X, \theta_{1,} \theta_2))}{\delta \theta_1}$$

Try linear

Model:



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

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

Output:

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

Is it any better than logistic regression?

Try linear

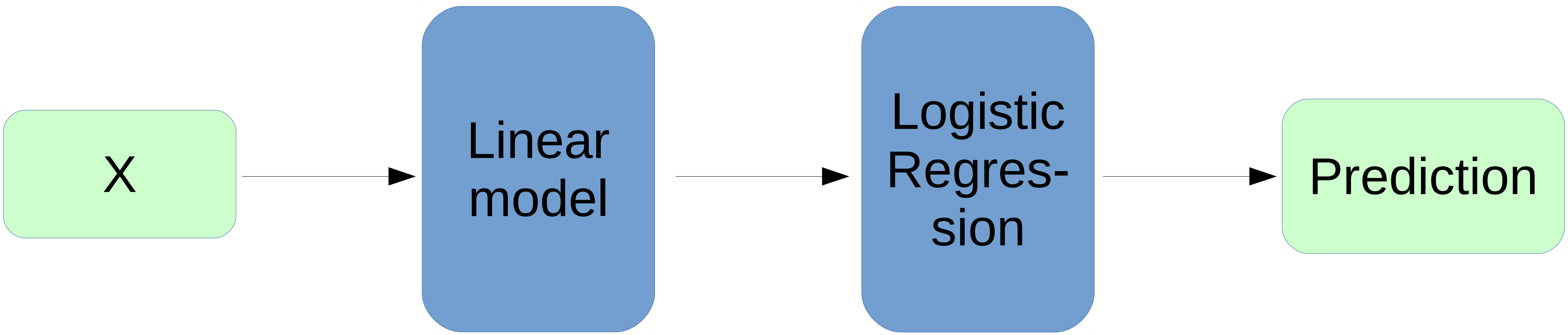
$$y_{pred} = \sigma\left(\sum_j w_j^o \left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

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

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

Try linear

Model:



$$h_j = \sum_i w_{ij}^h x_i + b_j^h$$

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

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

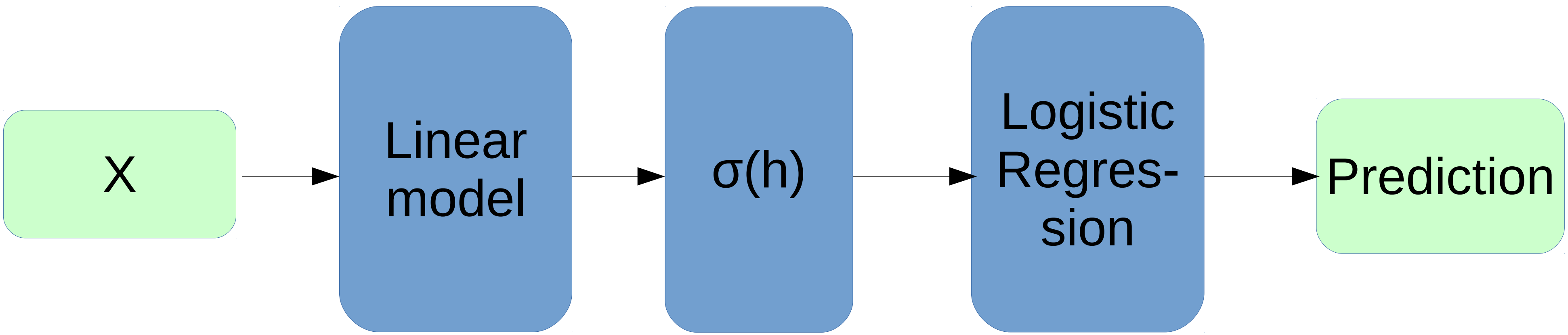
Output:

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

Is it any better than logistic regression?

Nonlinearity

Model:



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

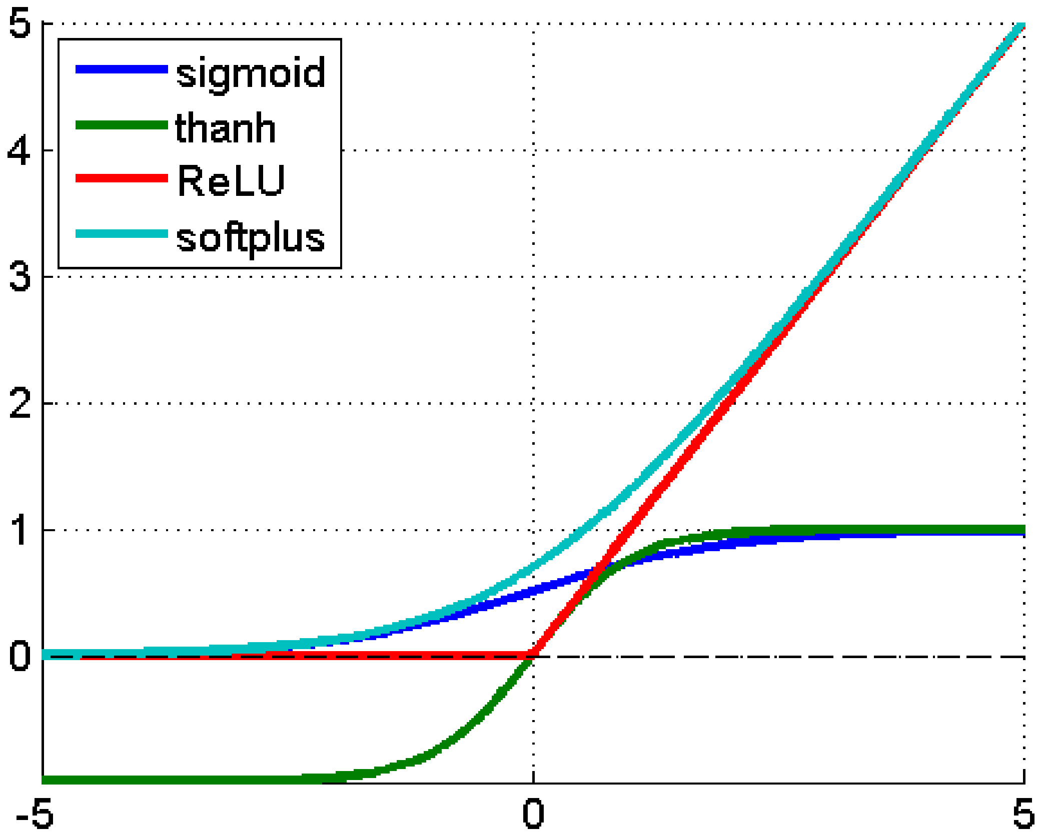
Gradients:

$$\frac{\delta L\left(y, y_{pred}\left(X, w_j^o, b^o, w_{ij}^h, b_j^h\right)\right)}{\delta w_j^o, \delta b^o, \delta w_{ij}^h, \delta b_j^h}$$

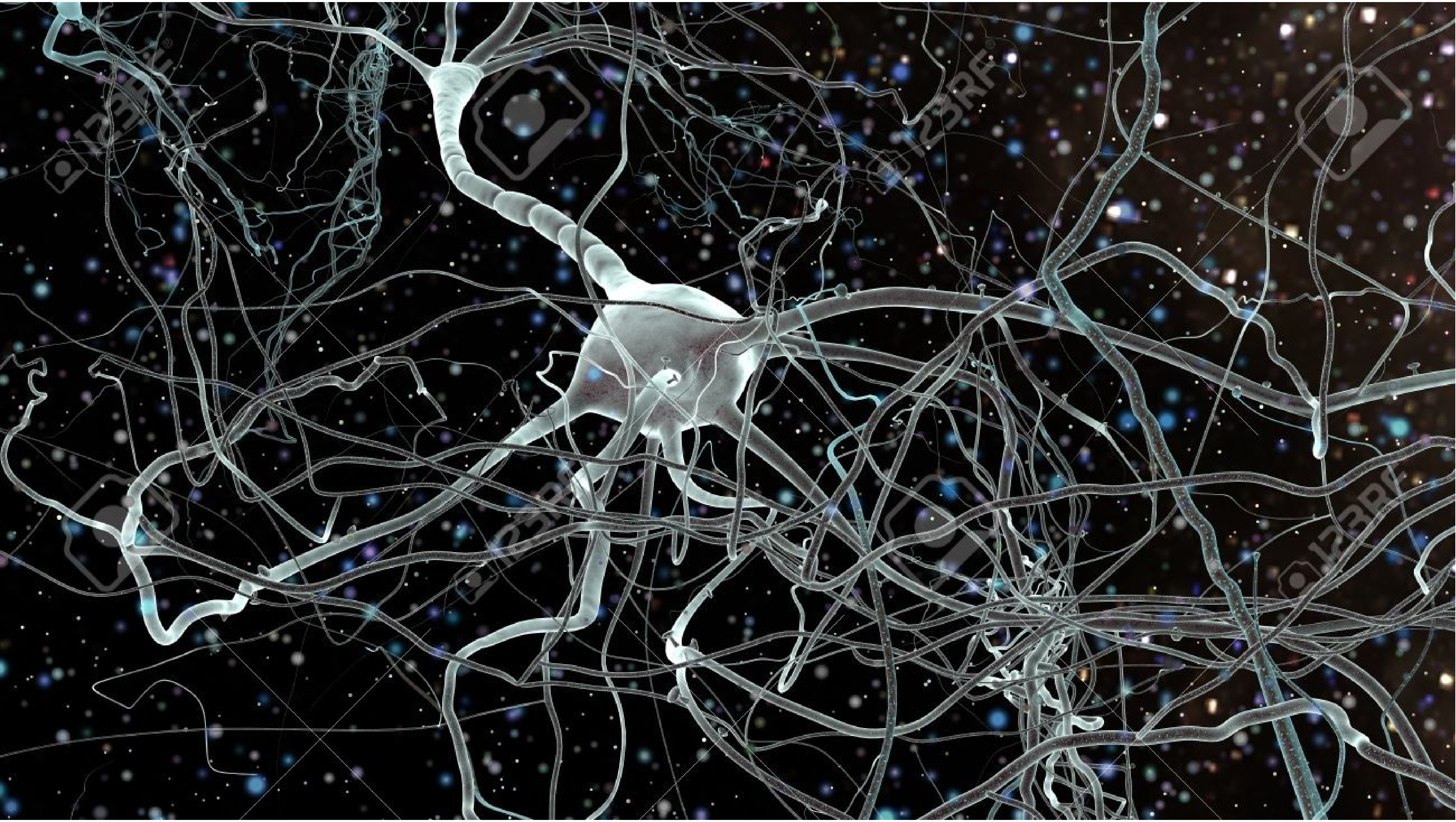
Nonlinearity

- f(a) = 1/(1+e^a)
- f(a) = tanh(a)

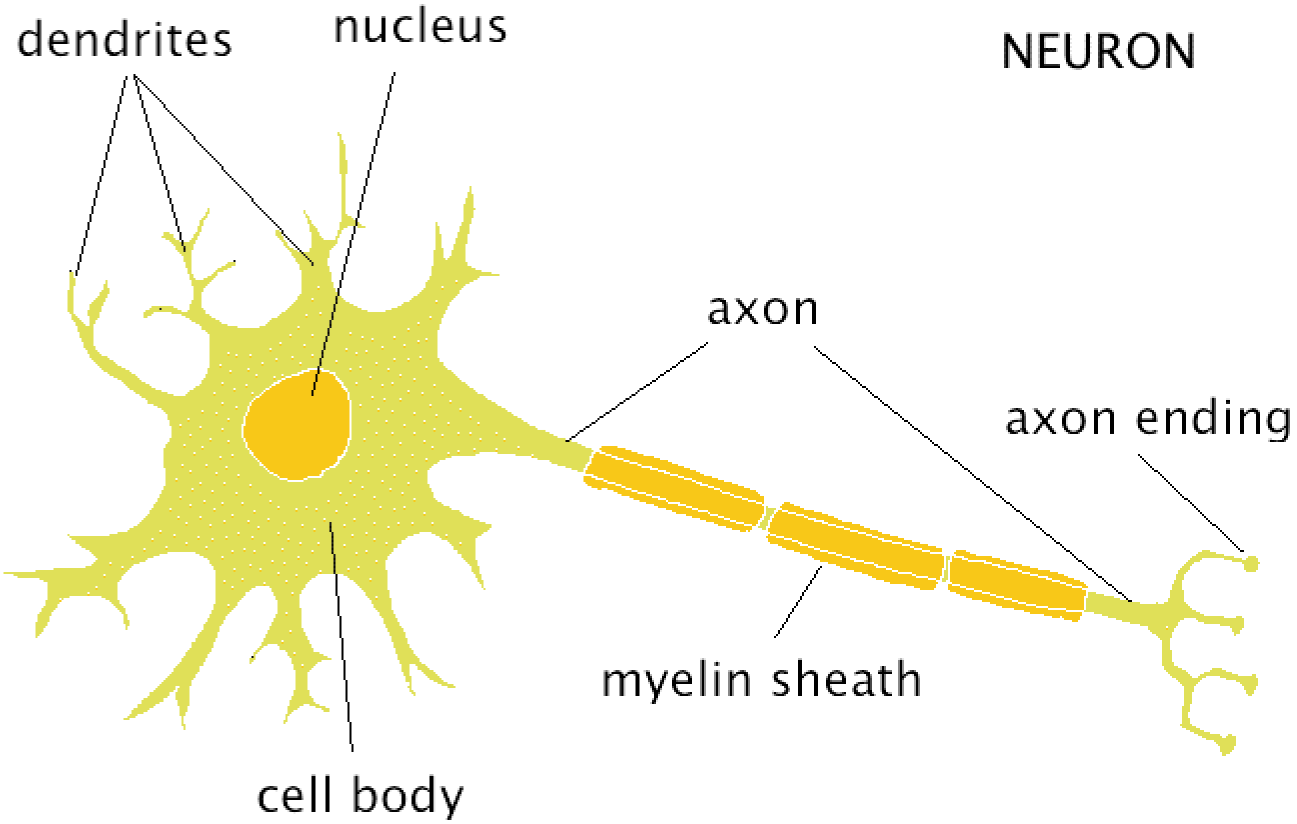
- f(a) = max(0,a)
- f(a) = log(1+e^x)



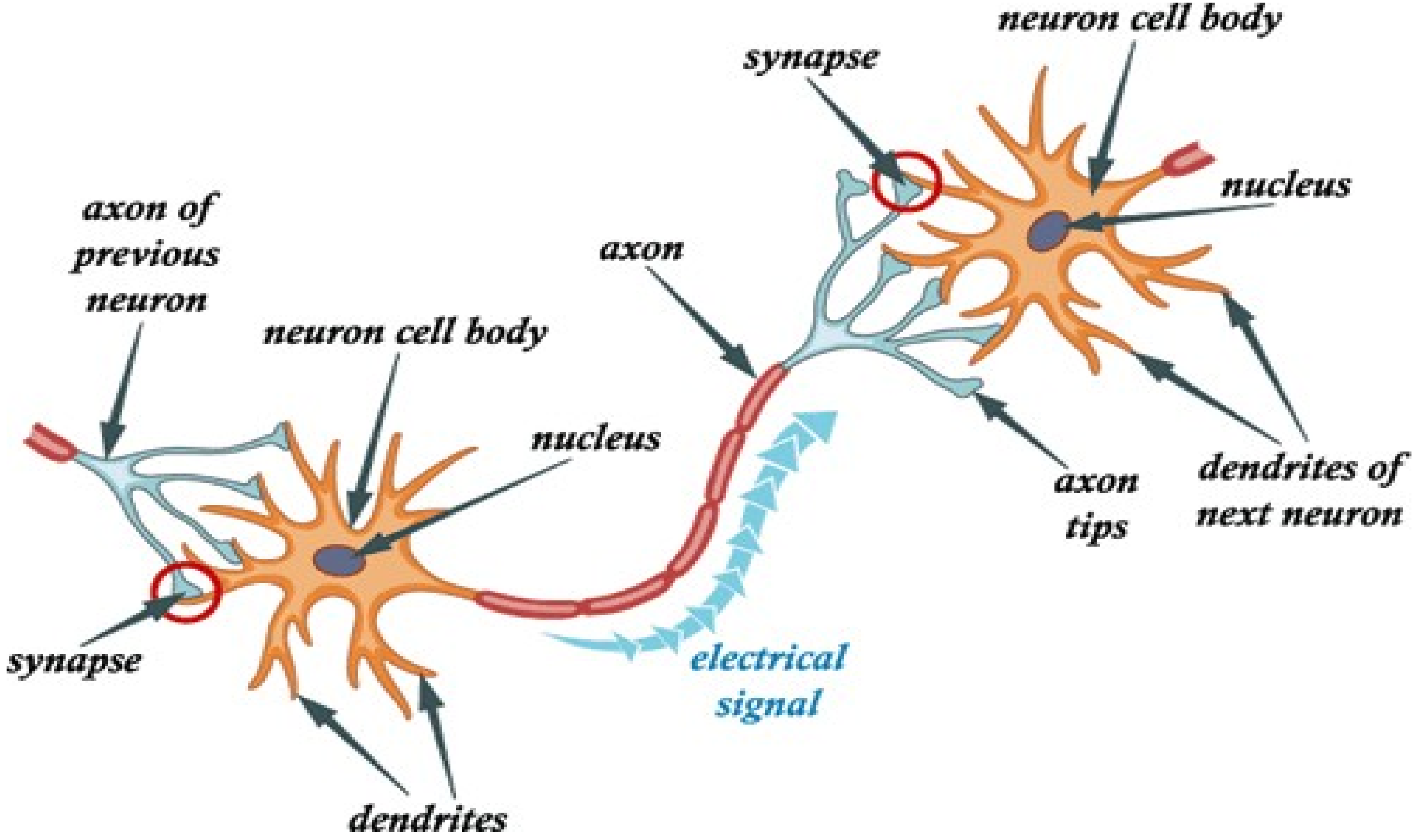
Biological inspiration



Biological inspiration

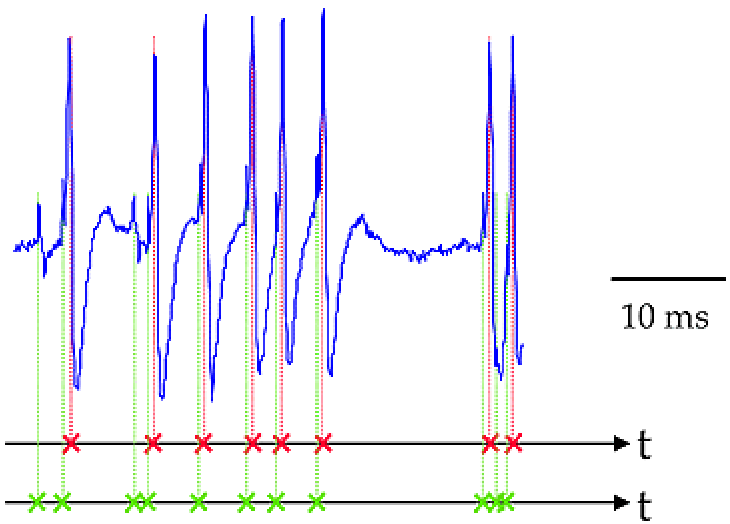


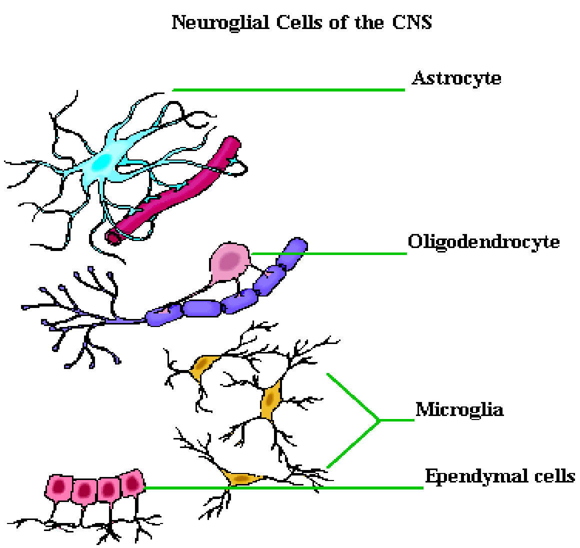
Biological inspiration



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





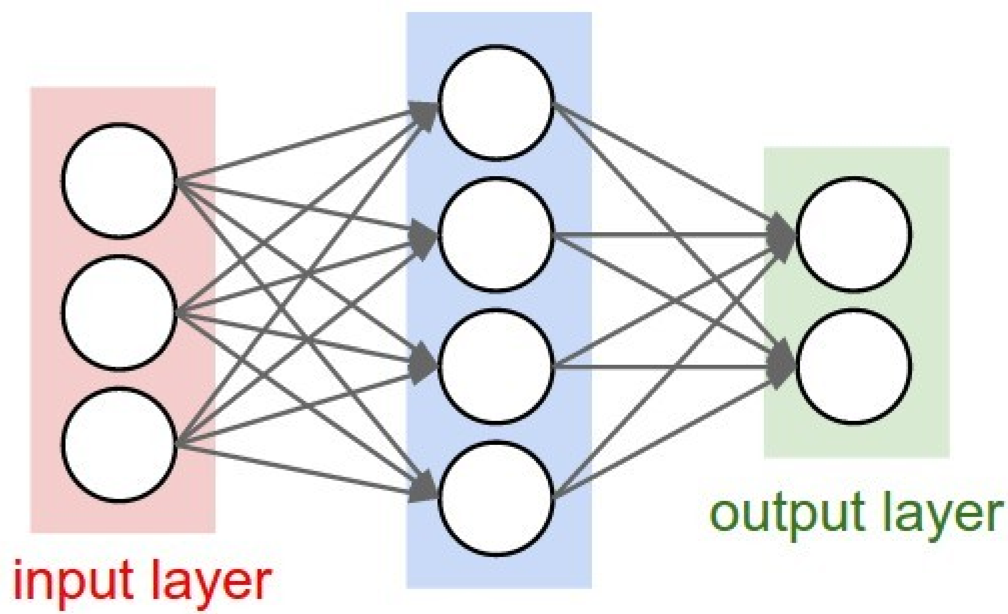
Connectionist phrasebook

- 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”

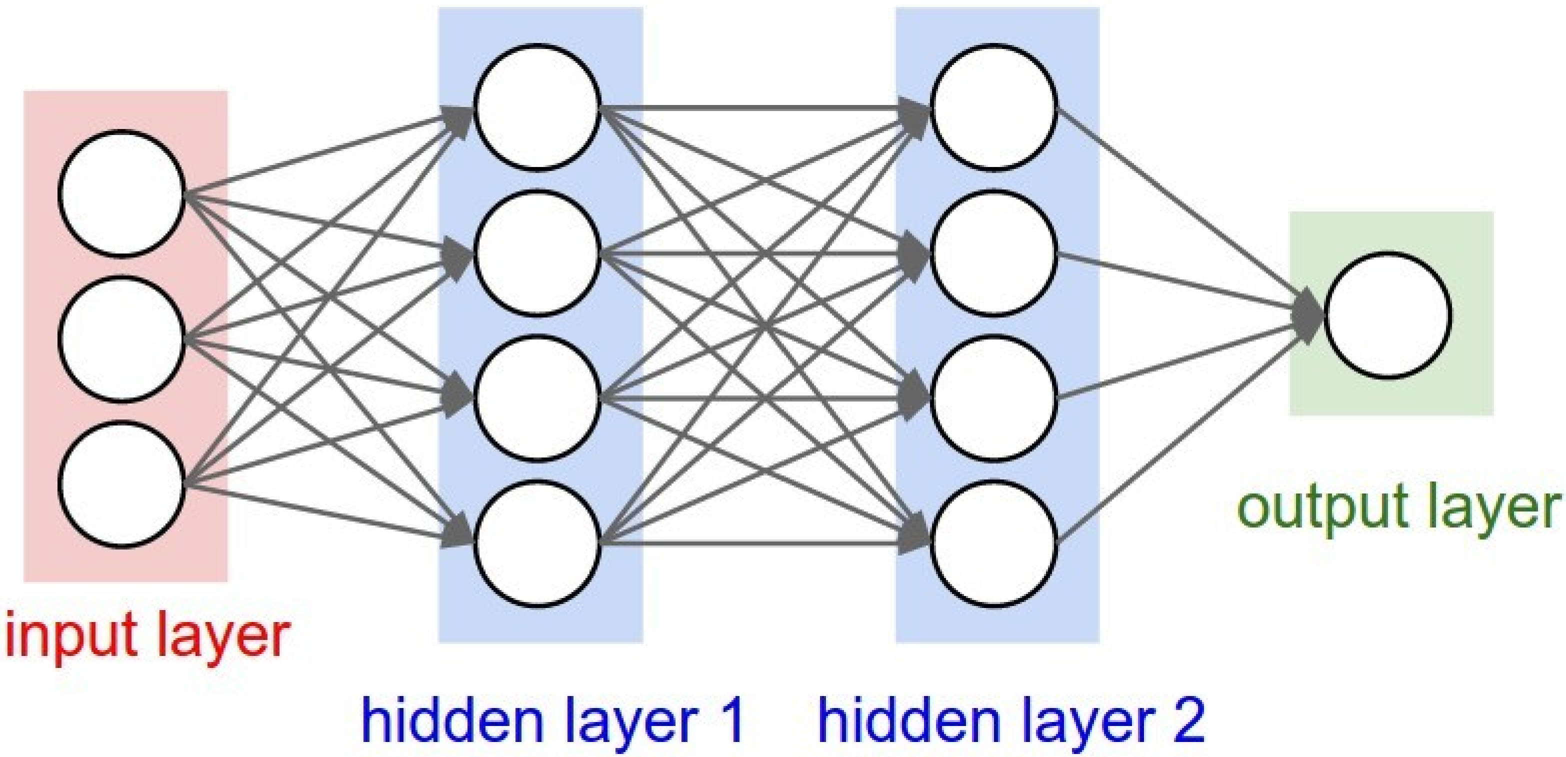
Connectionist phrasebook



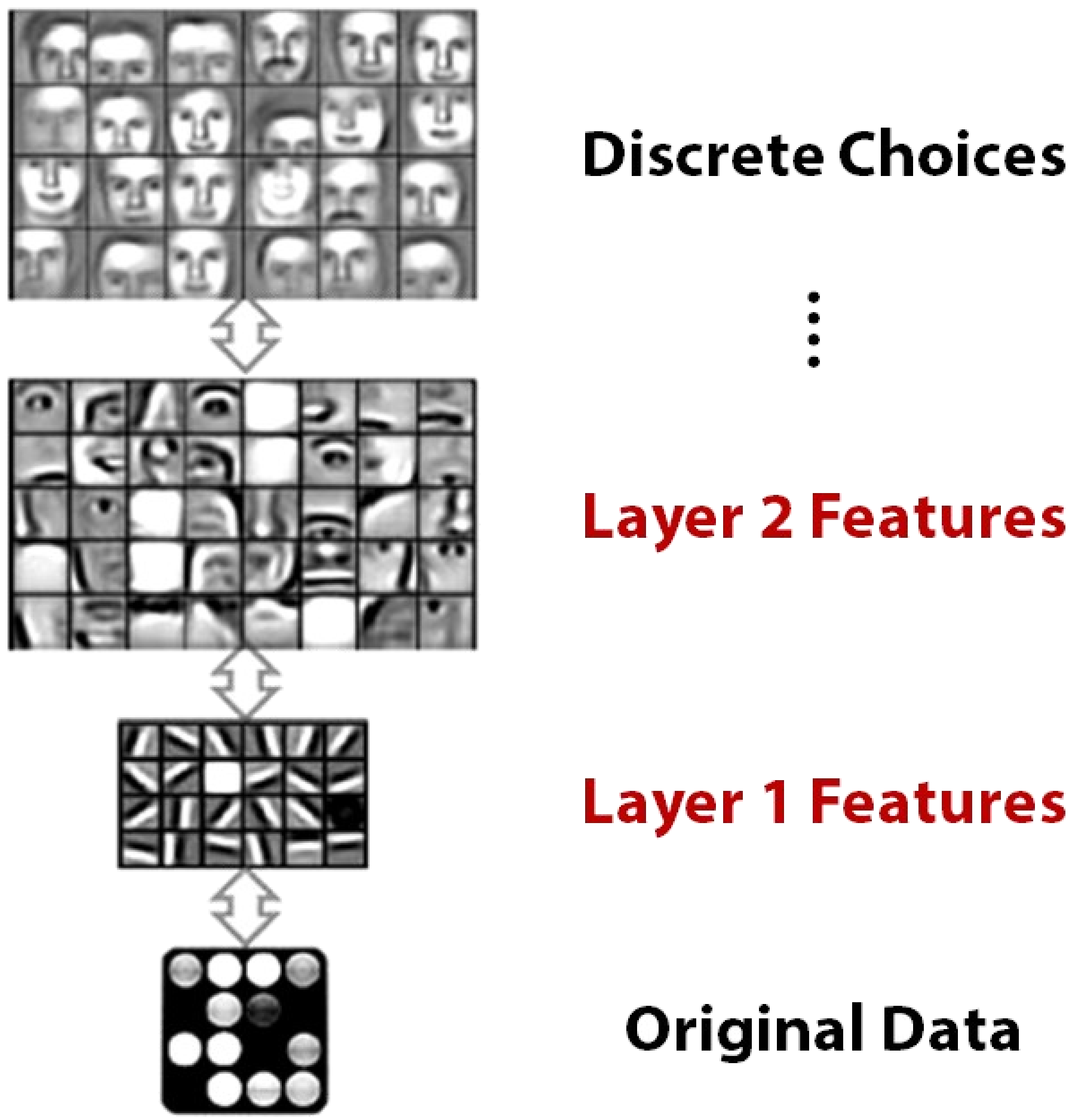
Dense layer
with a nonlinearity

- “Train it via backprop!”

Connectionist phrasebook



How do we train it?


Discrete Choices
Layer 2 Features
Layer 1 Features
Original Data

Potential caveats?

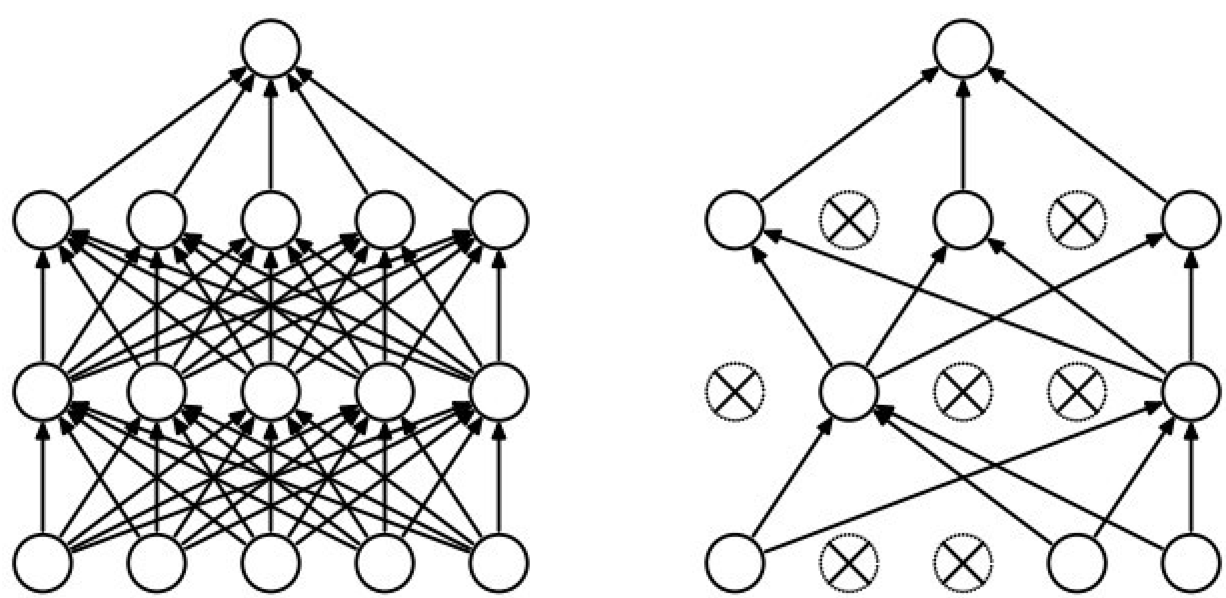
Potential caveats?

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

Regularization

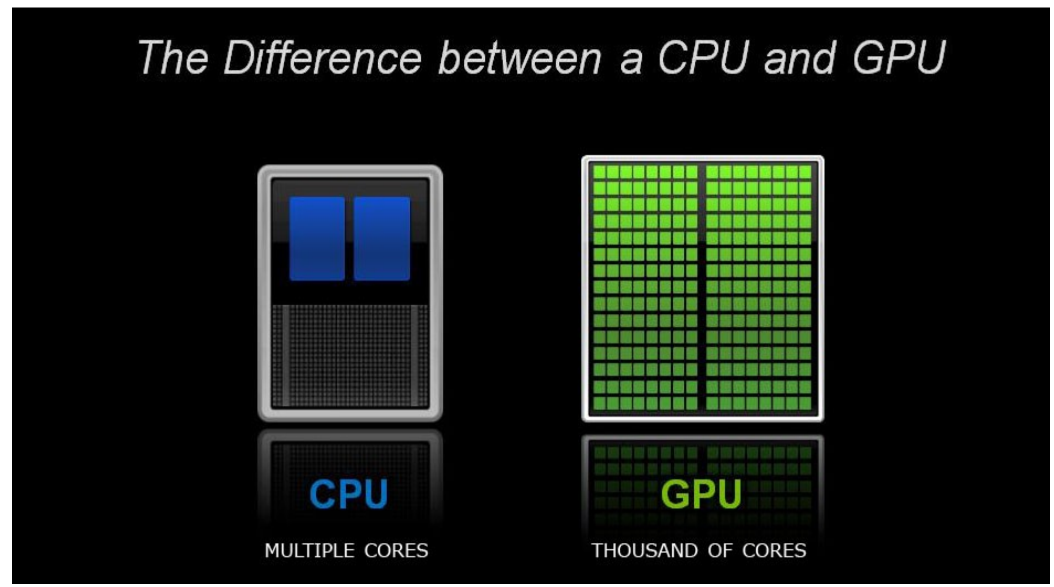
- L1, L2, as usual

- Dropout



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

Computation



Application: Image recognition

“Dog”

Application: Image recognition

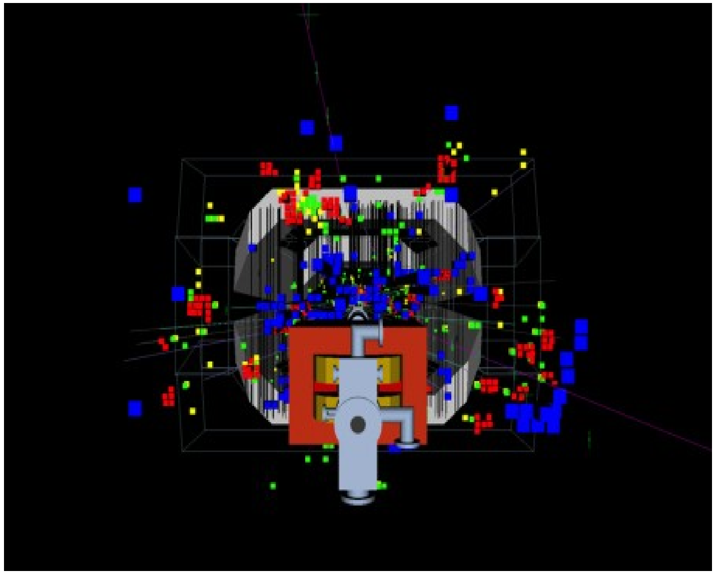
“Dog”

<a particular kind of dog>

“Dog tongue”

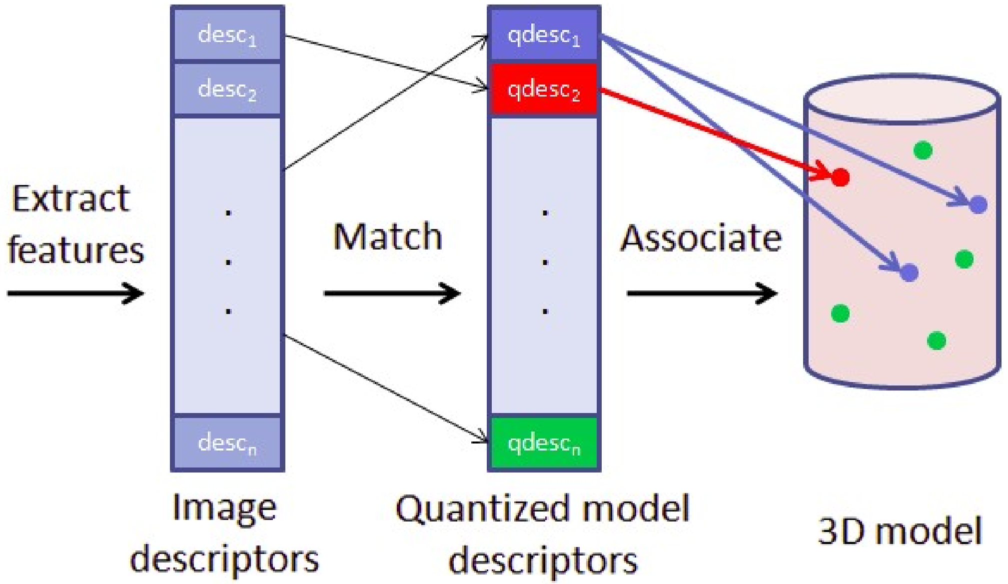
“Animal sadism”

Application: Image recognition



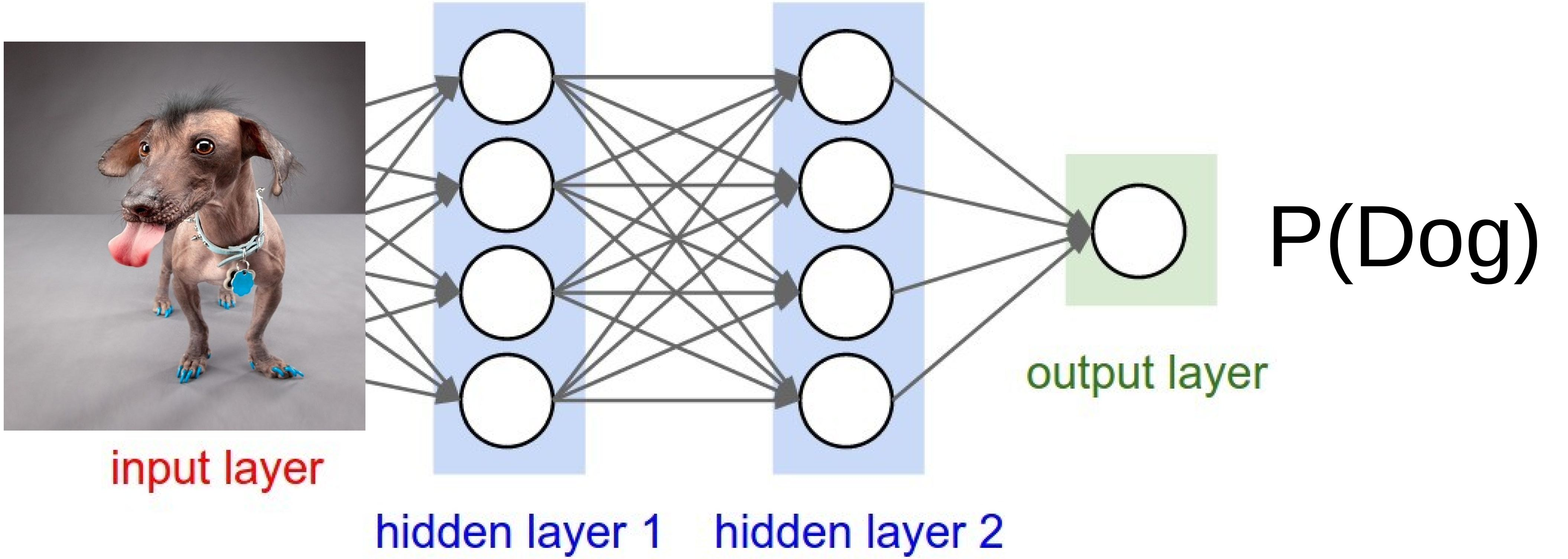
$K^0_S \rightarrow \pi^+ \pi^-$

Classical approach

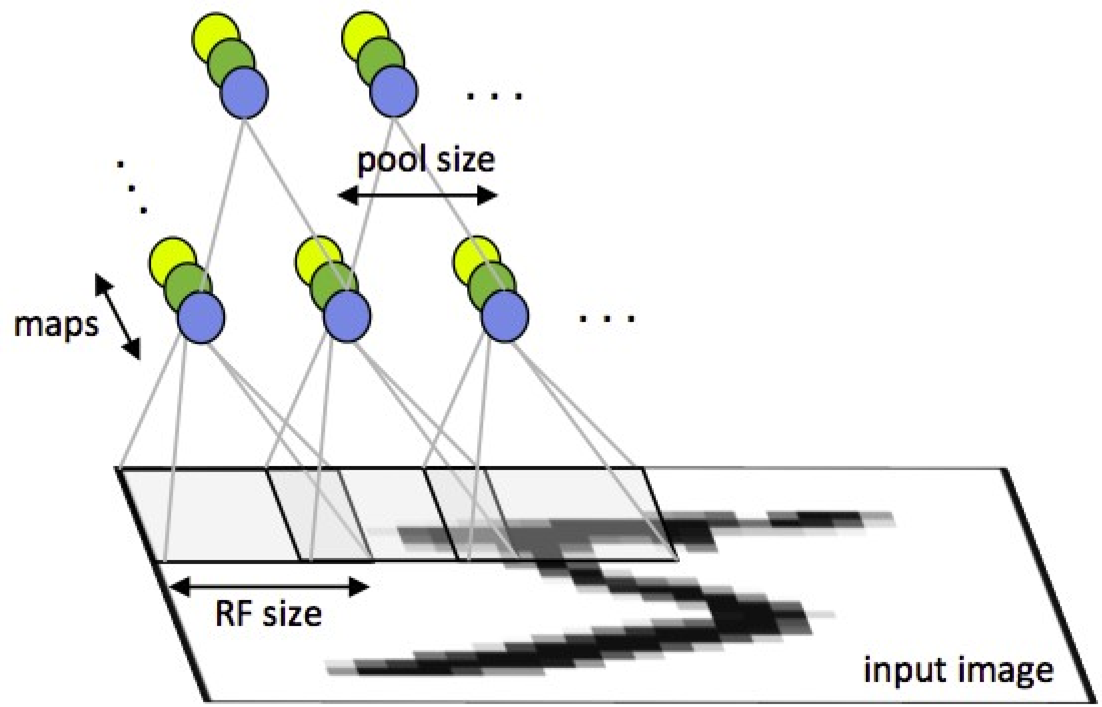


“Dog”

NN approach

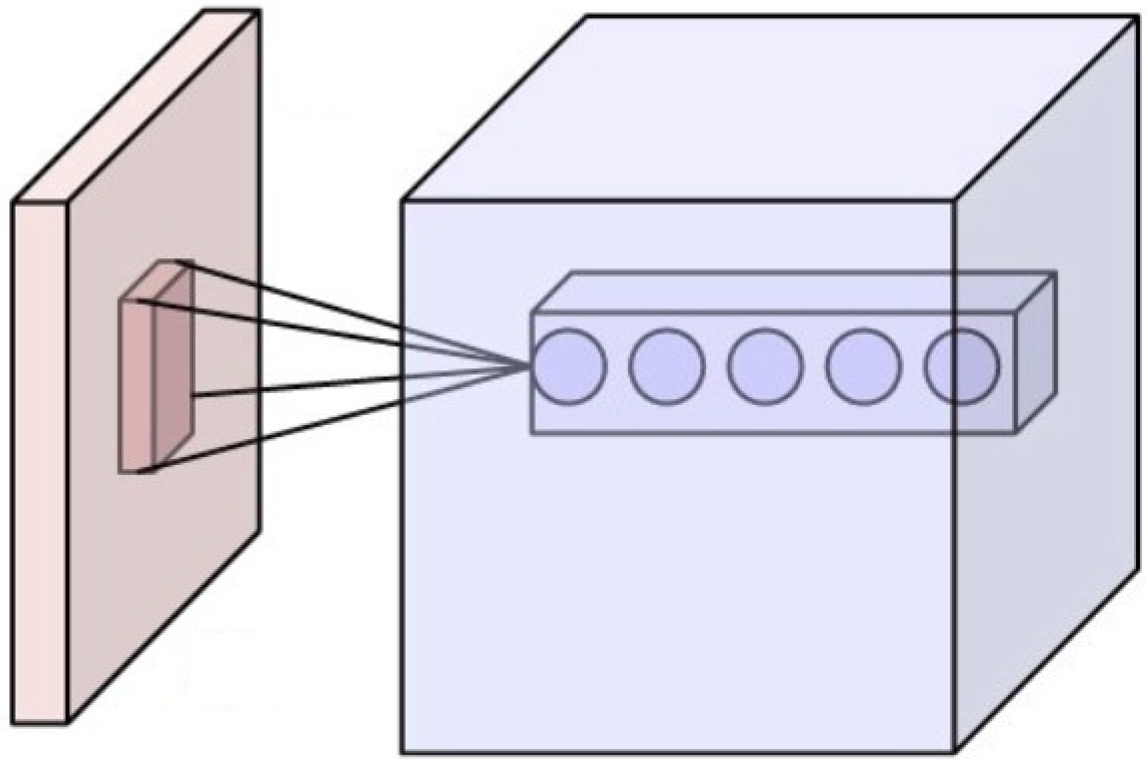


Convolutional NNs



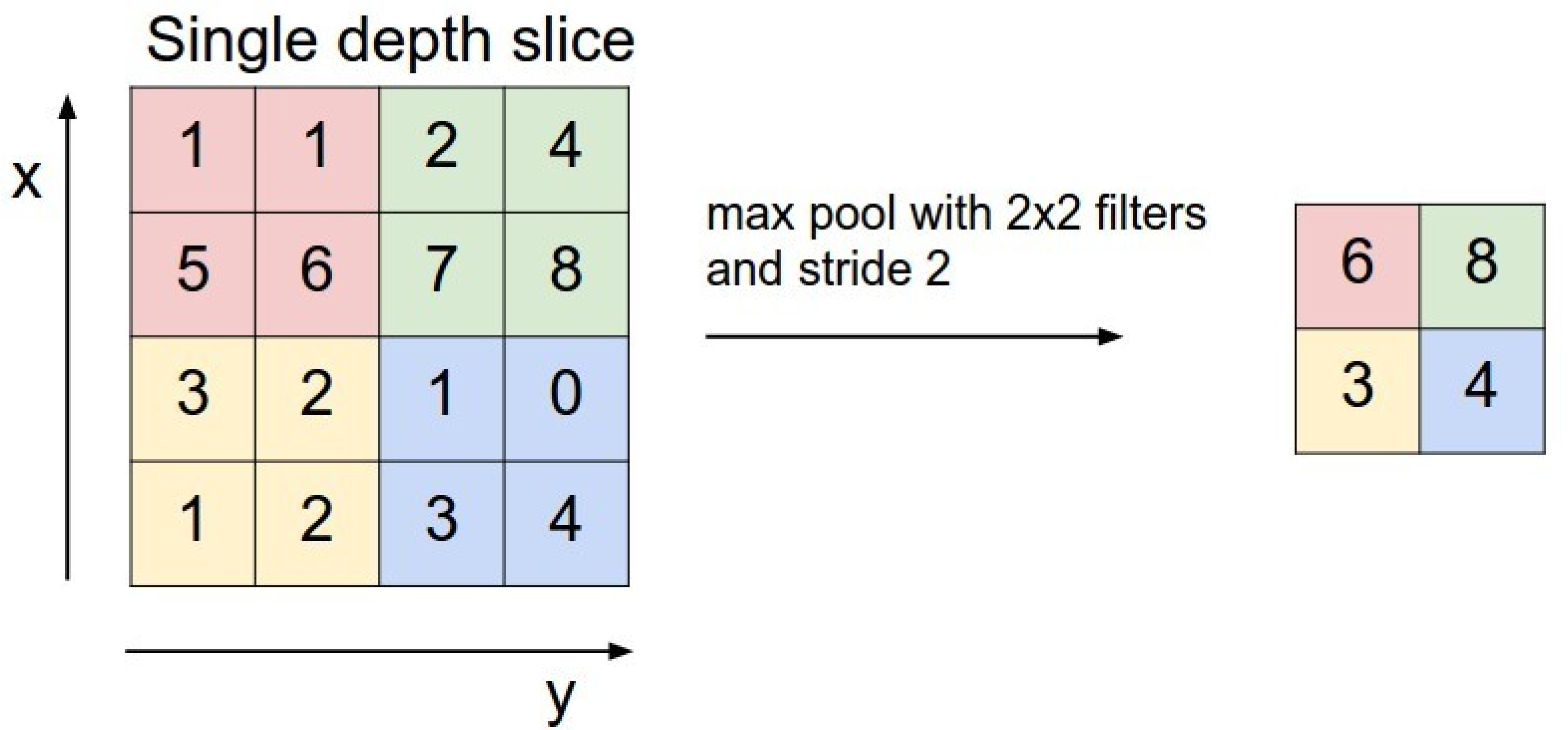
Intuition: how cat-like is this square?

Convolutional NNs



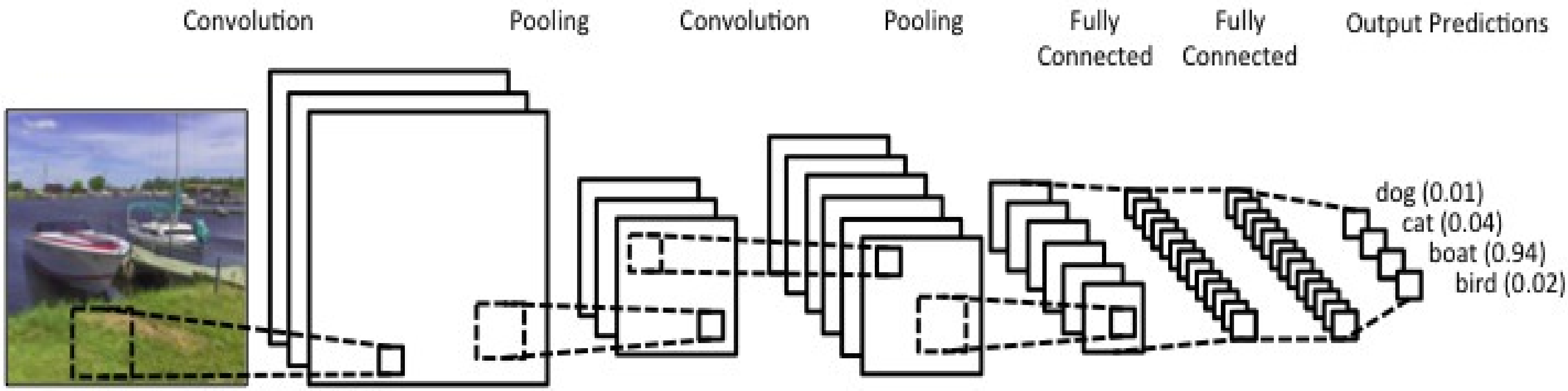
Intuition: how cat-like is this square?

Convolutional NNs



Intuition: What is the max cat-likelihood over this area?

Convolutional NNs



Gradient descent

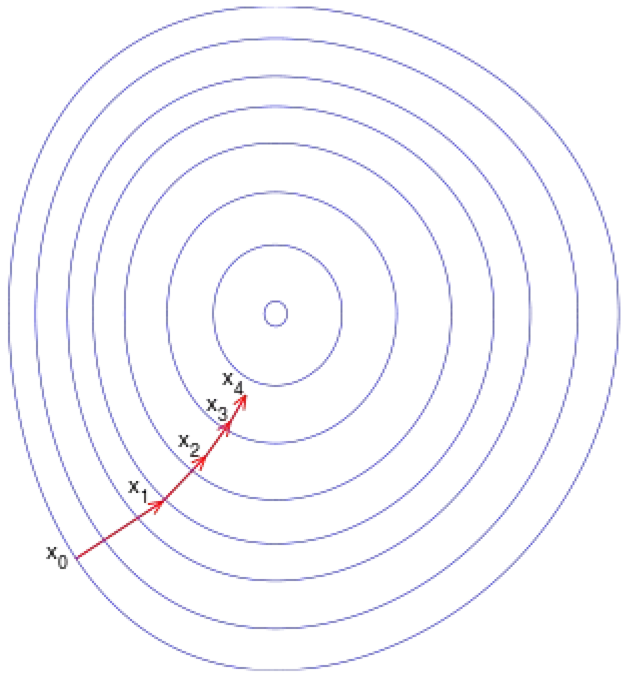
Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

$$(\text{for } j = 1 \text{ and } j = 0)$$

}



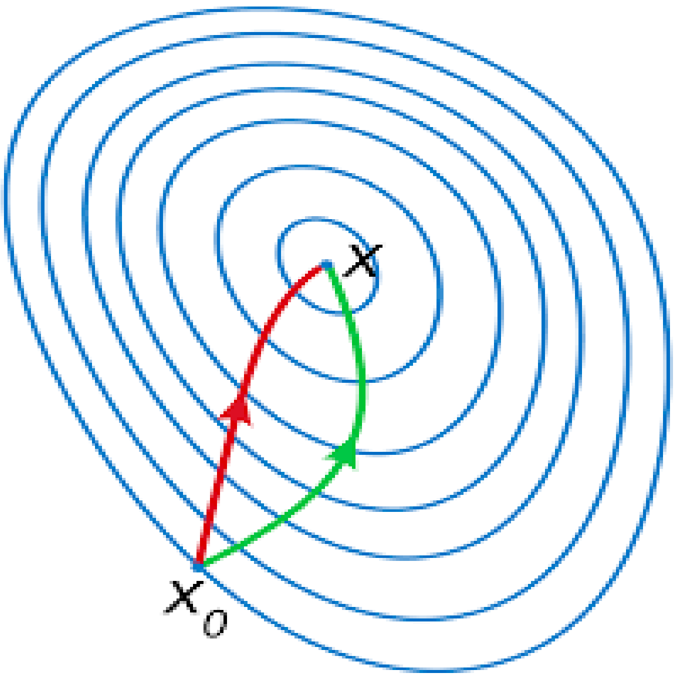
Newton-Raphson

Parameter update

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma\left[\mathbf{H}f(\mathbf{x}_n)\right]^{-1}\nabla f(\mathbf{x}_n).$$

Hessian:

$$\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\[2.2ex] \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\[2.2ex] \vdots & \vdots & \ddots & \vdots \\[2.2ex] \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$



Red: Newton-Raphson
Green: gradient descent

Any drawbacks?

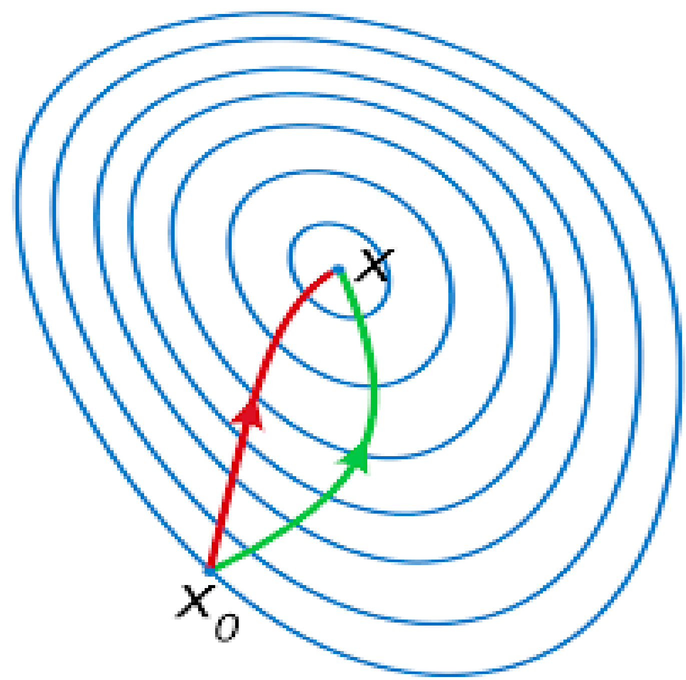
Newton-Raphson

Parameter update

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma[\mathbf{H}f(\mathbf{x}_n)]^{-1}\nabla f(\mathbf{x}_n).$$

Hessian:

$$\begin{bmatrix} \frac{\partial^2 f}{} & \frac{\partial^2 f}{} & \cdots & \frac{\partial^2 f}{} \\ \frac{}{\partial x_n\,\partial x_1} & \frac{}{\partial x_n\,\partial x_2} & \cdots & \frac{}{\partial x_n^2} \end{bmatrix}$$



Red: Newton-Raphson
Green: gradient descent

Impractical for large NNs

SGD with momentum

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

$$\Delta w := \eta \nabla Q_i(w) + \alpha \Delta w$$

$$w := w - \Delta w$$

AdaGrad

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

$$Let\ g_{\tau,j} = \frac{\delta L}{\delta w_j}\ on\ \tau_{th}\ tick$$

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2$$

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j.$$

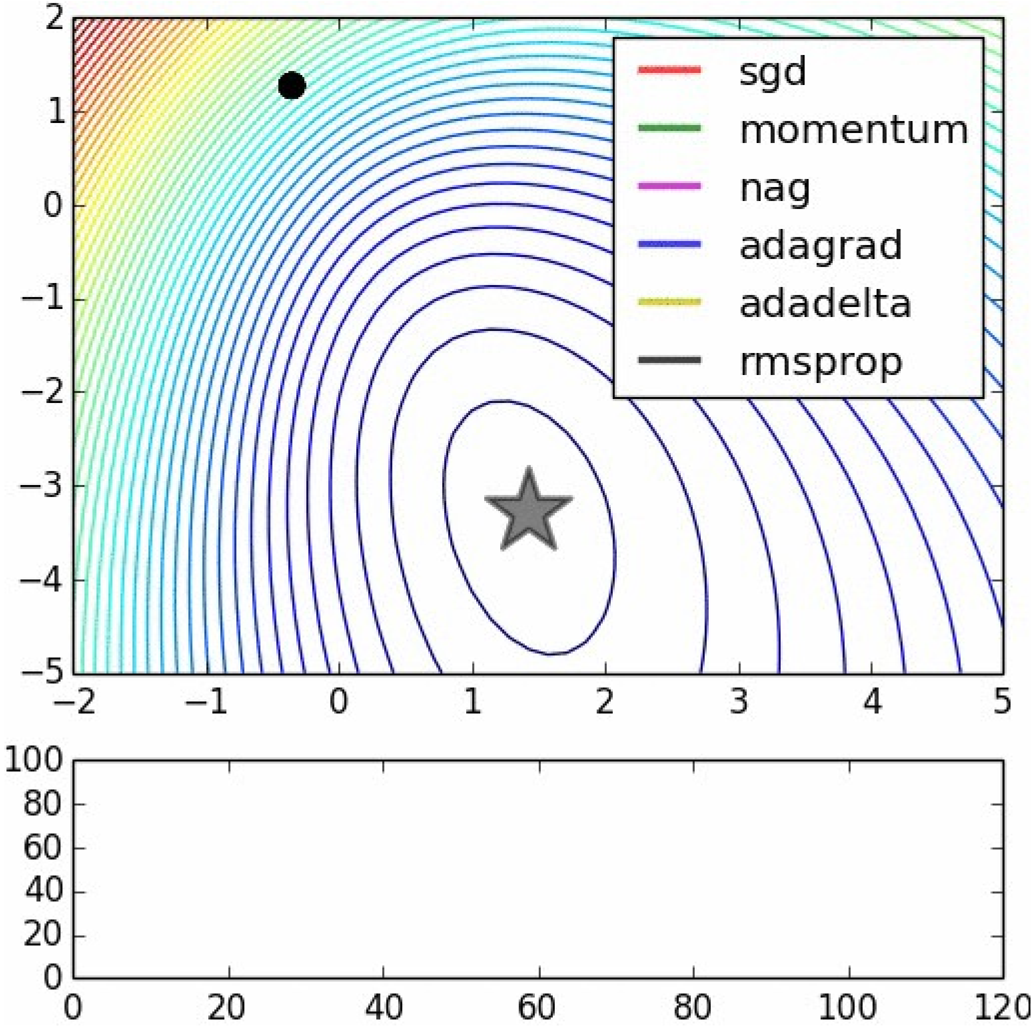
RMSProp

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

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

Alltogether



Moar stuff

- Adadelta
- Adam
- Adamax
- Hessian-free
- Nesterov-momentum

Nuff

Let's code some neural networks!