Practical Deep Learning

Episode 0, 2025

ML recap. Neural Nets 101

‘bout the course

First module:

- Layers and optimization (Conv, Transformer, ...)
- Popular vision and NLP tasks

Second module:

- Advanced stuff: LLM/Agents, Diffusion, RL, ...
- Elective topics (we’ll run a poll!)

Workload:

- One small assignment after each week’s practice
- It’s open-source! Go contribute :)

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:

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

Optimization (iterative):

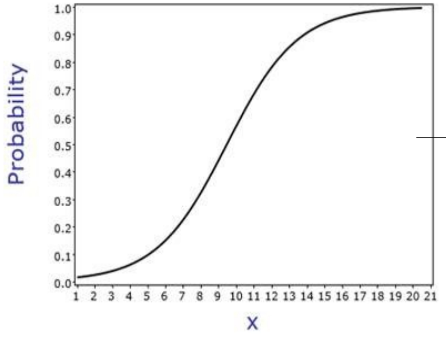
$$w_0 \leftarrow 0$$

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

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

Logistic Regression

X → Wx +b →



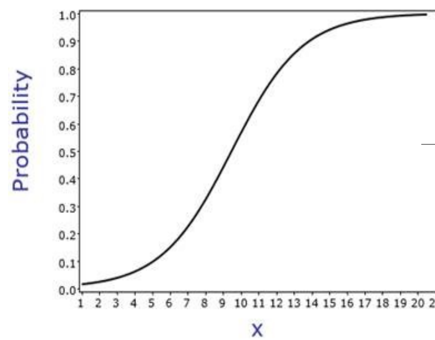
→ P(y)

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

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:

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

Objective function:

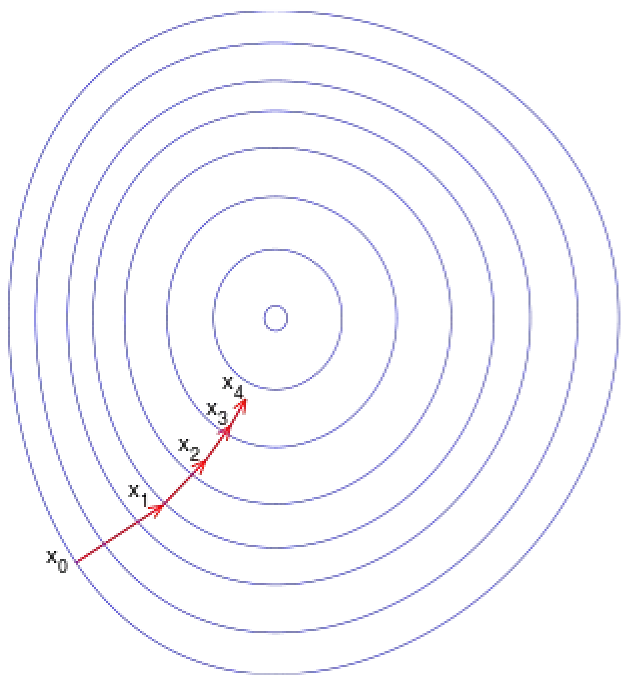
$$L = -\sum_i \log P\left(y_i^{correct} | x_i\right)$$

Gradient descent

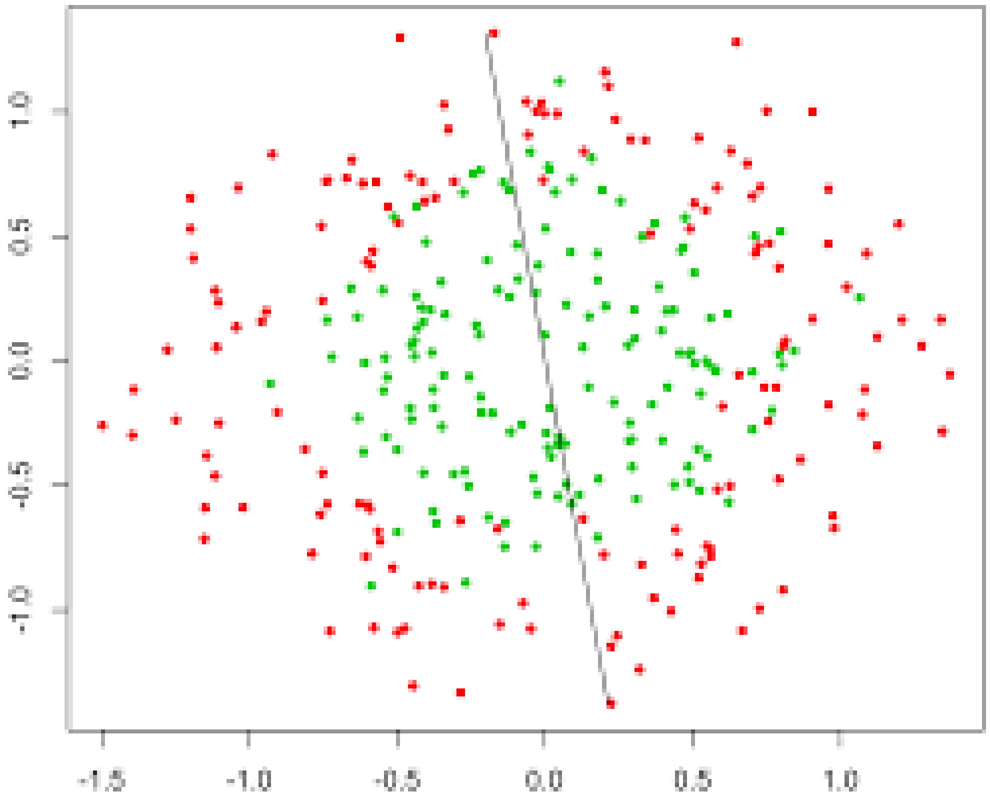
Update:

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

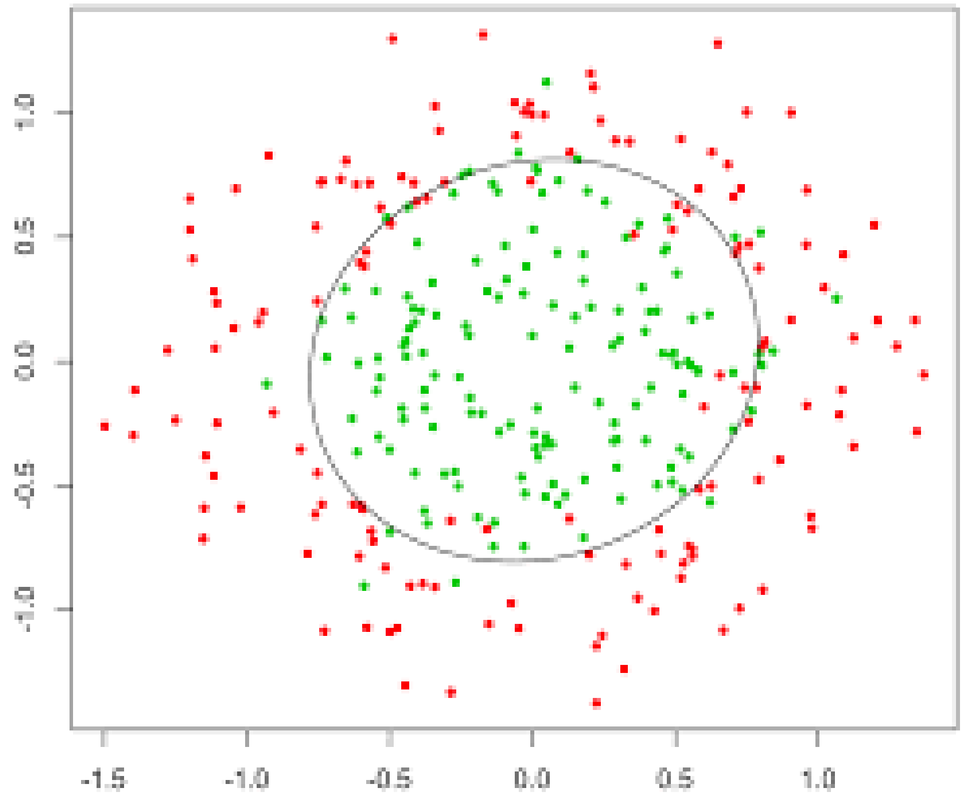
- a – learning rate a<<1
- L – loss function



Nonlinear dependencies



What we have



What we want

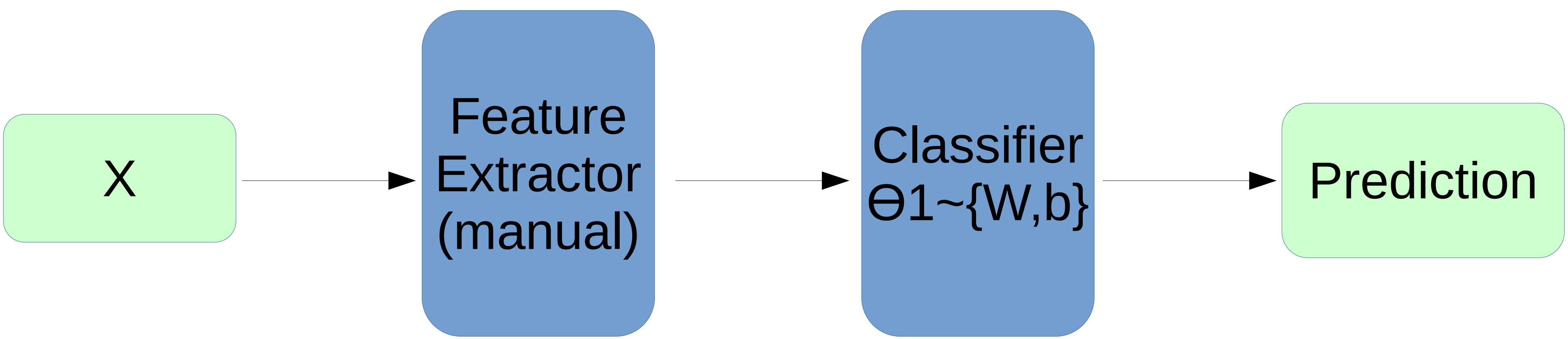
How to get that?

Feature extraction

Loss, for example:

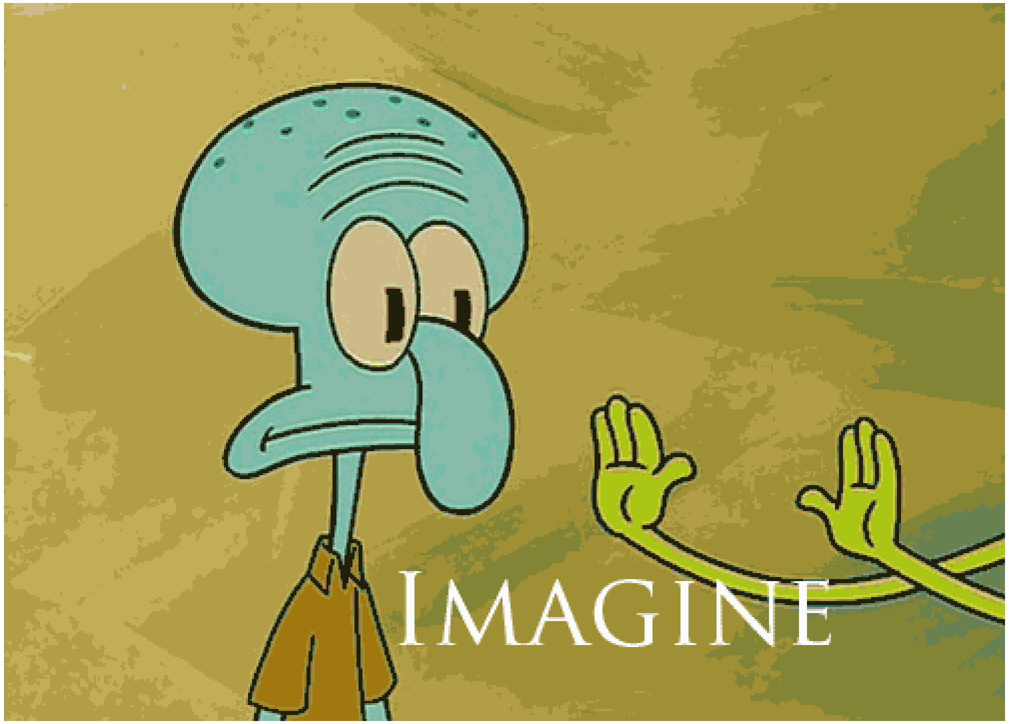
$$L = -\sum_i y_i \log P(y|x_i) + (1-y_i) \log(1 - P(y|x_i))$$

Model:



Training:

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



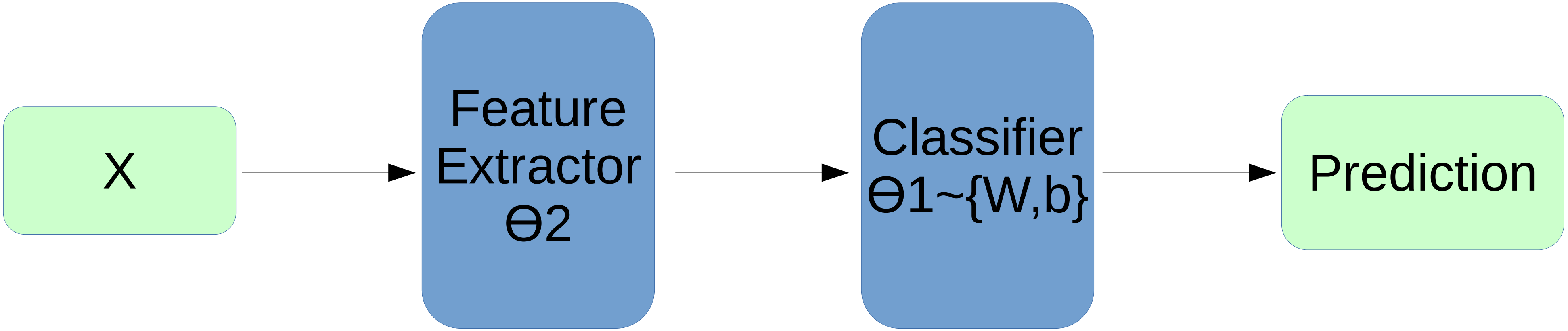
Features would tune to your problem automatically!

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:



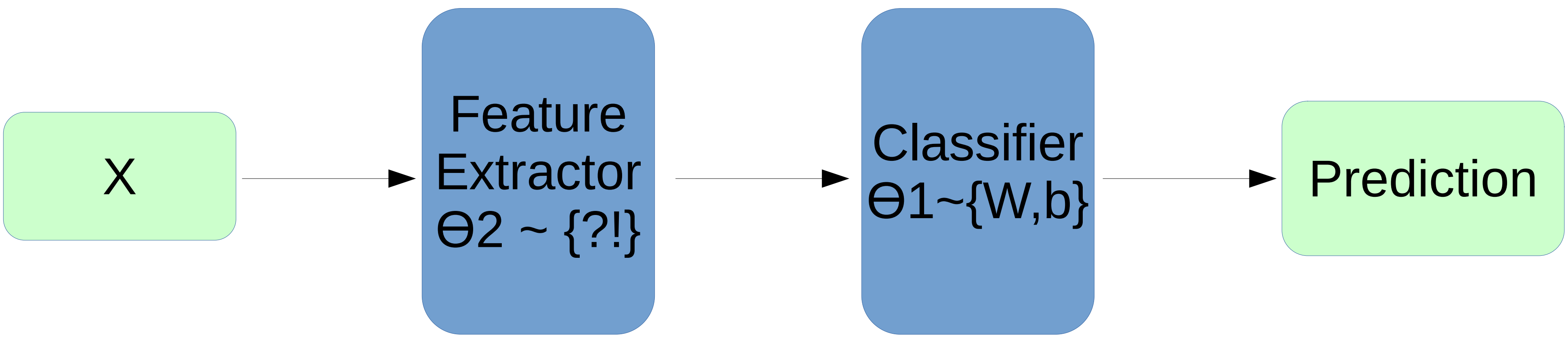
Training: ? $\underset{\theta_1}{argmin}\, L(y, P(y|x))$

What do we want, exactly?

Loss, for example:

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

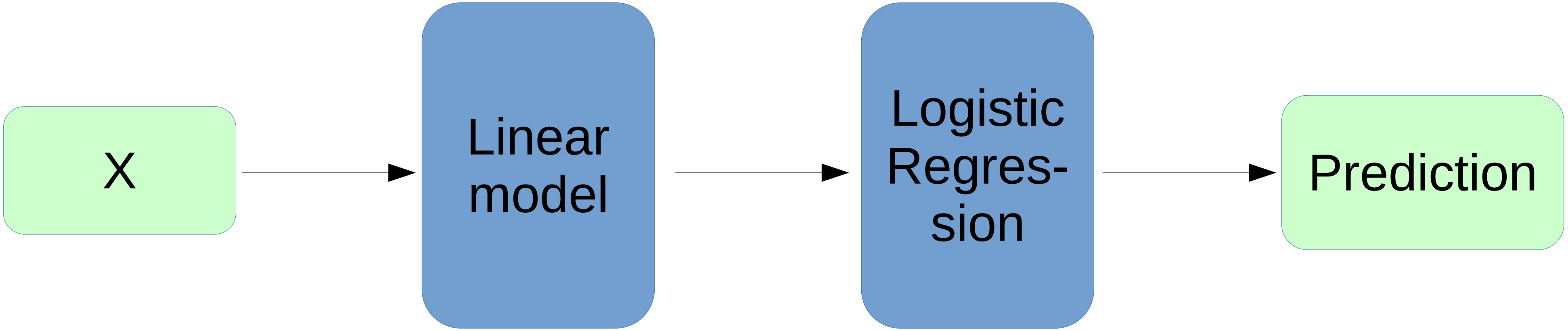
Model:



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

Try linear

Model:

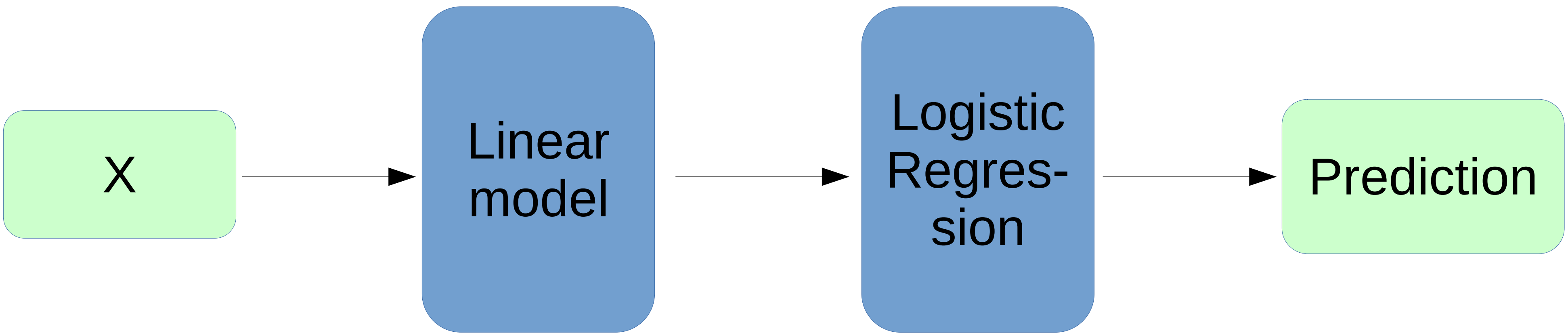


$$h_j = \sum_i w_{ij}^h x_i + b_j^h \quad j \in \{1,2,\ldots,n\}$$

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

Try linear

Model:



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

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

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

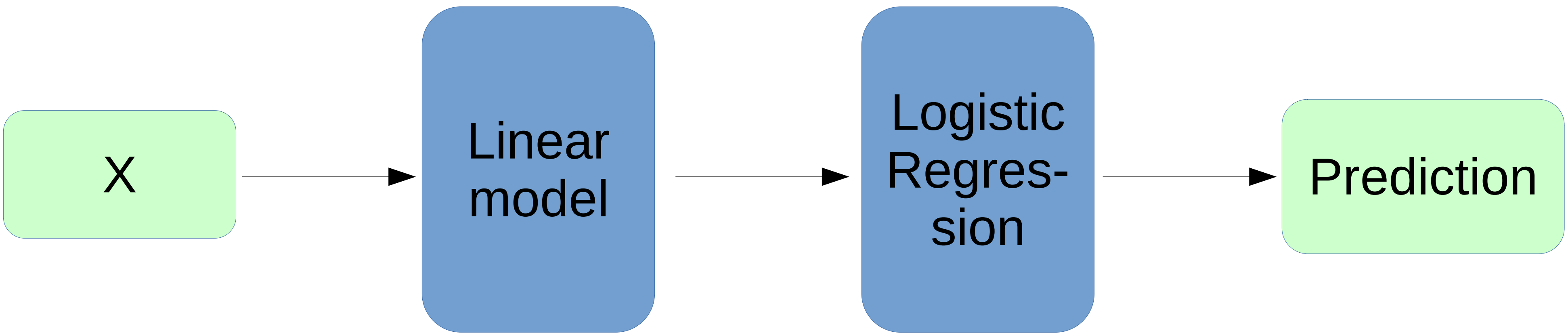
$$P(y|x) = \sigma\left(\sum_j w_j^o \left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

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

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

Try linear

Model:



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

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

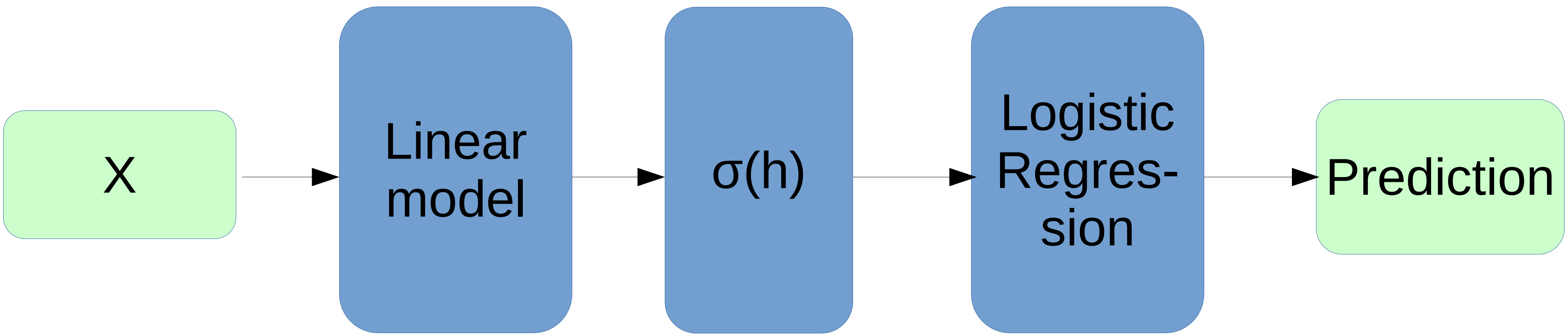
Output:

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

Is it any better than logistic regression?

Nonlinearity

Model:

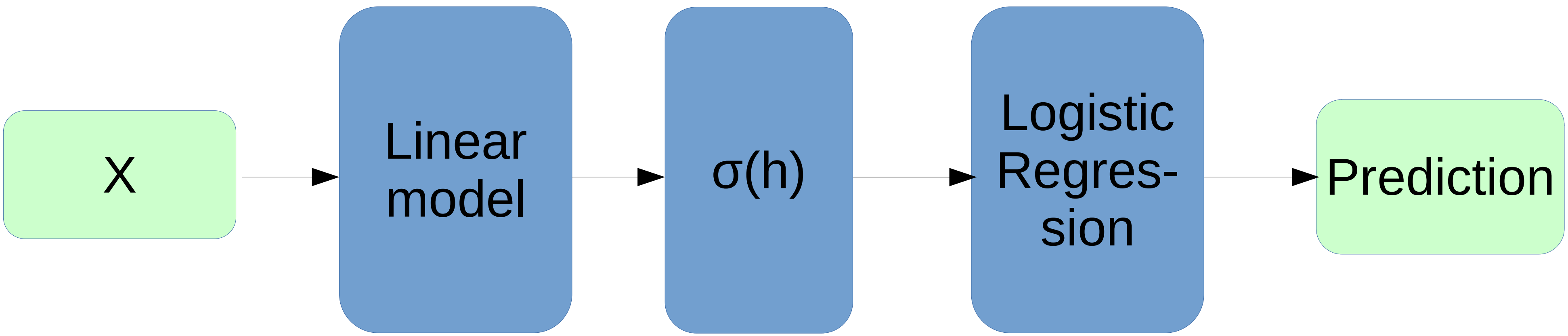


$$h_j = \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) \qquad y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$$

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

Nonlinearity

Model:



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

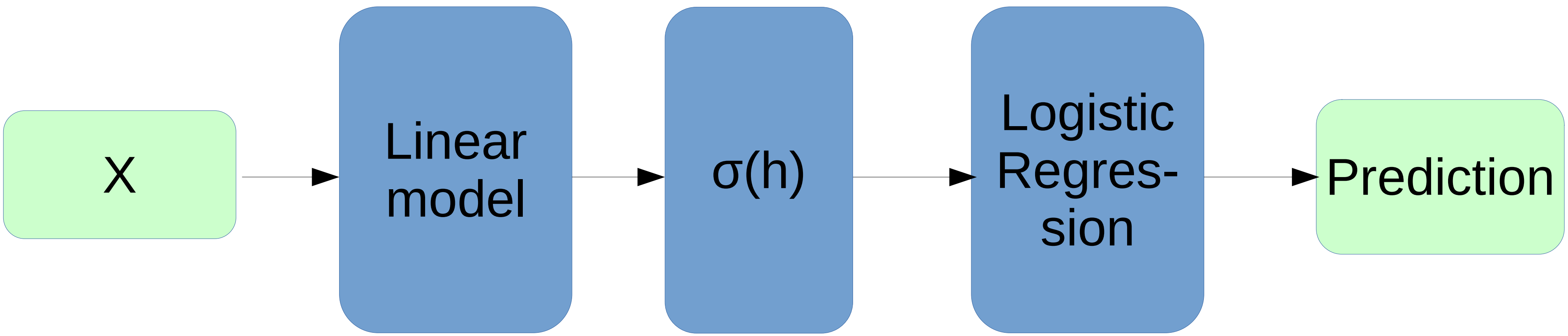
$$j\in\{1,2,\ldots,n\}$$

Output:

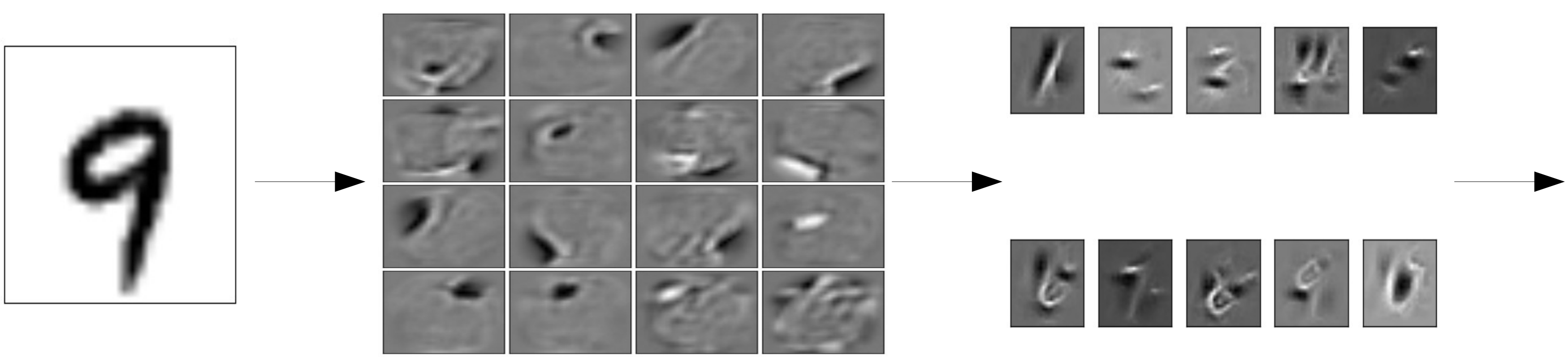
$$P(y|x)=\sigma\left(\sum_j w_j^o\,\sigma\left(\sum_i w_{ij}^h x_i+b_j^h\right)+b^o\right)$$

Nonlinearity

Model:



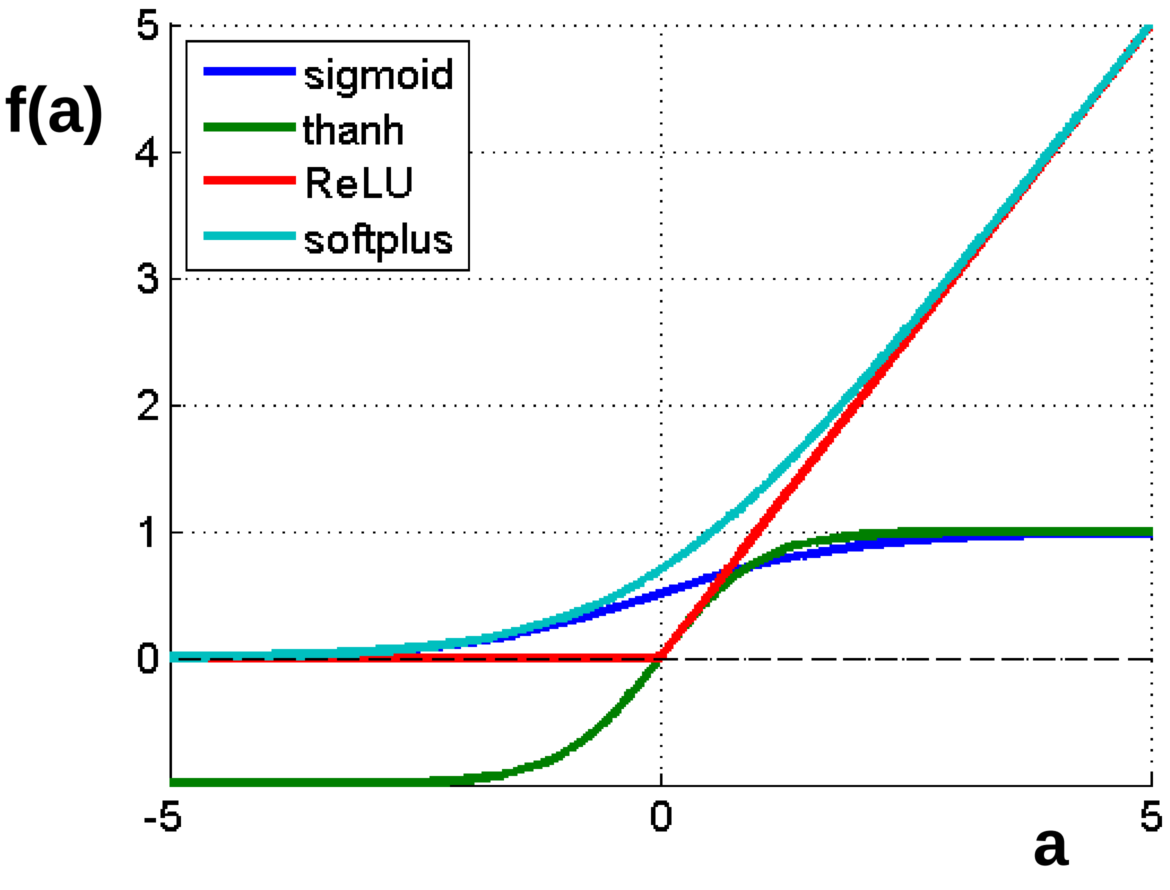
$$h_j = \sigma\left(\sum_{i \in \{1,2,\dots,n\}} w_{ij}^h x_i + b_j^h\right) \qquad y_{pred} = \sigma\left(\sum_j w_j^0 h_j + b^0\right)$$



P(“1”)
P(“2”)
...
P(“9”)

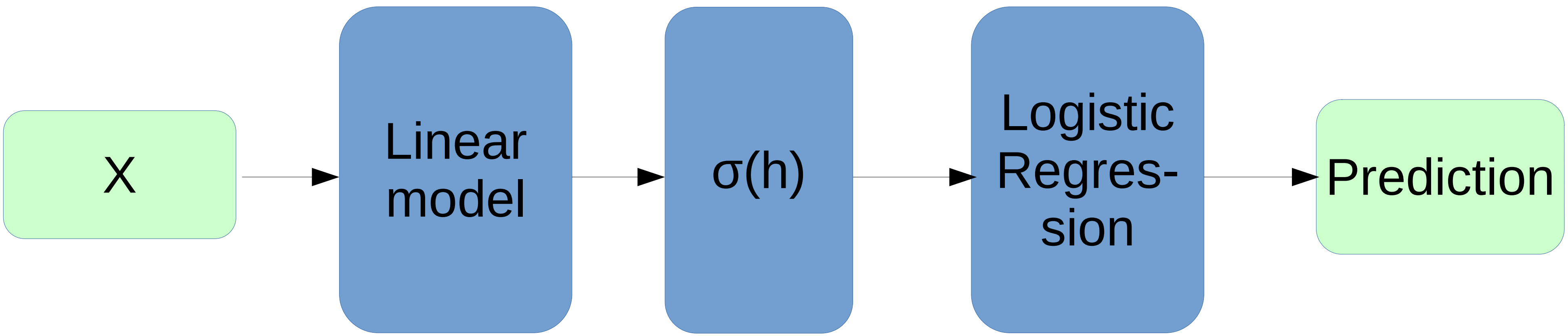
Nonlinearity

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



Training neural nets

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:

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

Part II: Backpropagation

Backpropagation

TL;DR: backprop = chain rule*

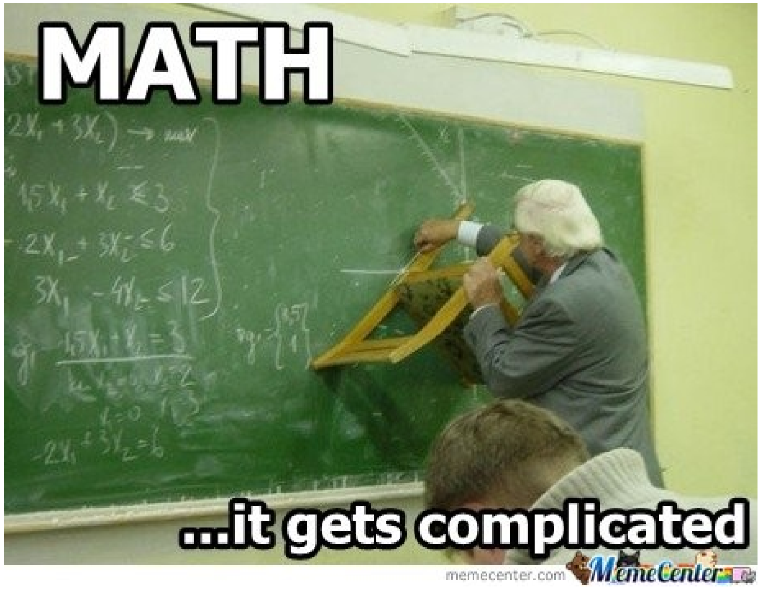
$$\frac{\partial f\left(g\left(x\right)\right)}{\partial x}=\frac{\partial f\left(g\left(x\right)\right)}{\partial g\left(x\right)}\cdot\frac{\partial g\left(x\right)}{\partial x}$$

Backpropagation

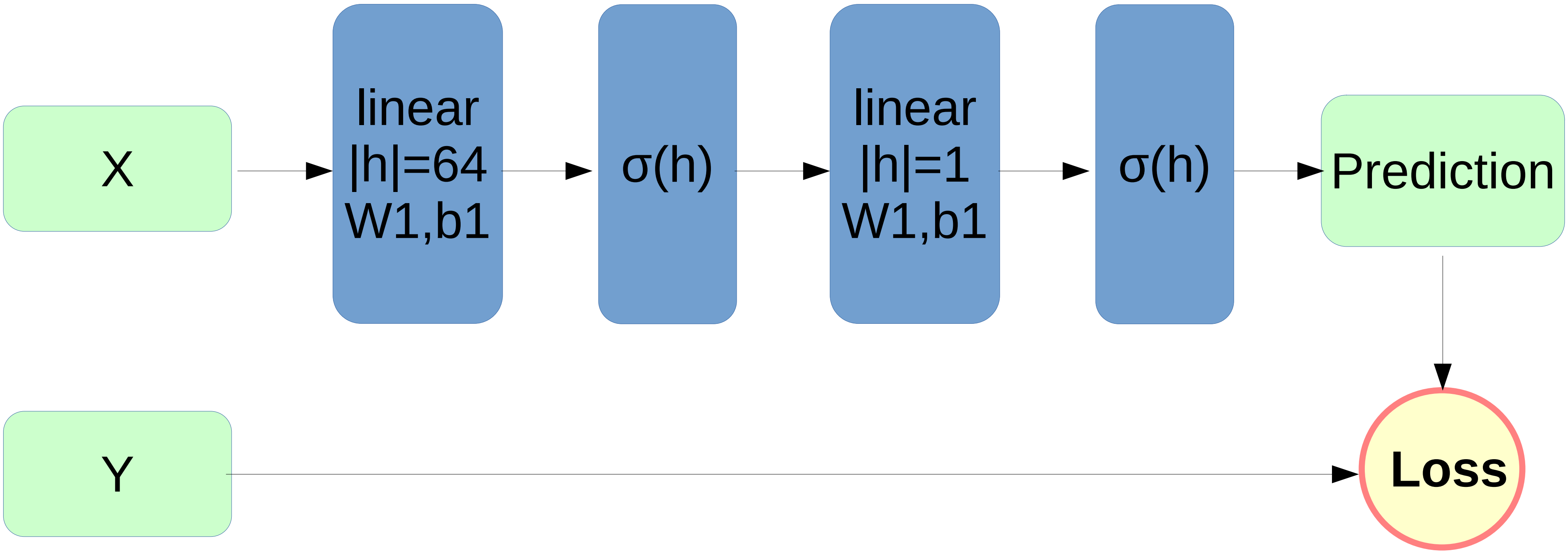
TL;DR: backprop = chain rule*

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

* g and x can be vectors/vectors/tensors

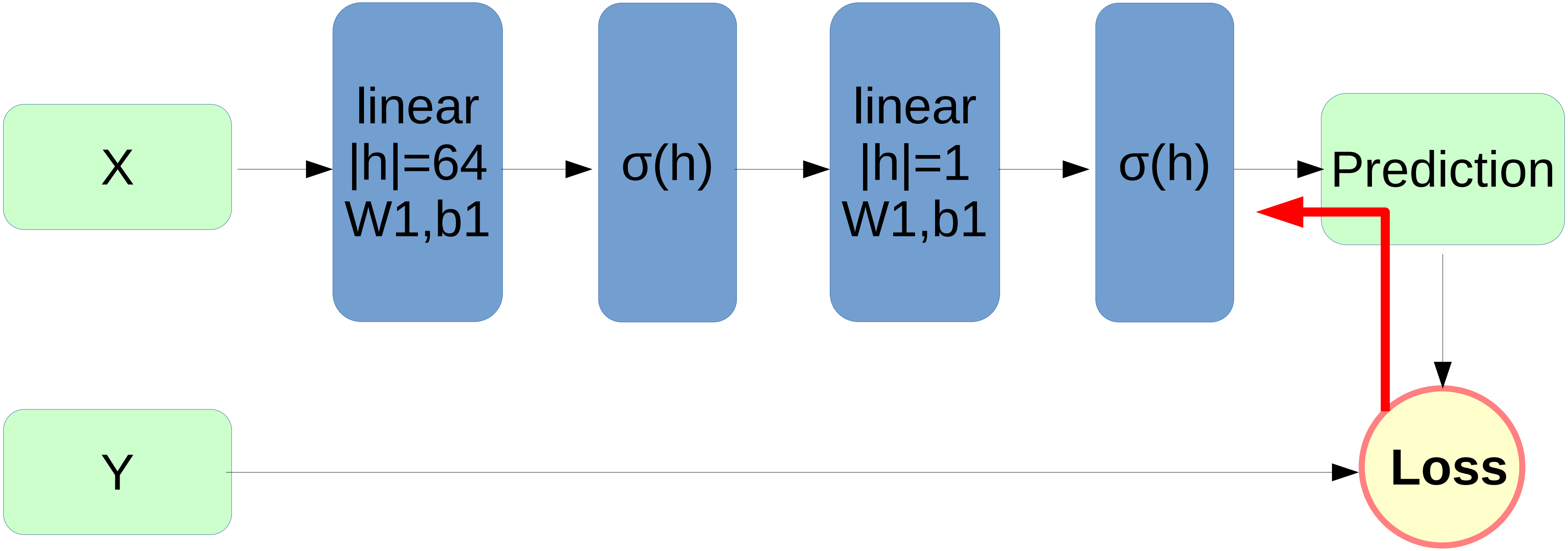


Backpropagation



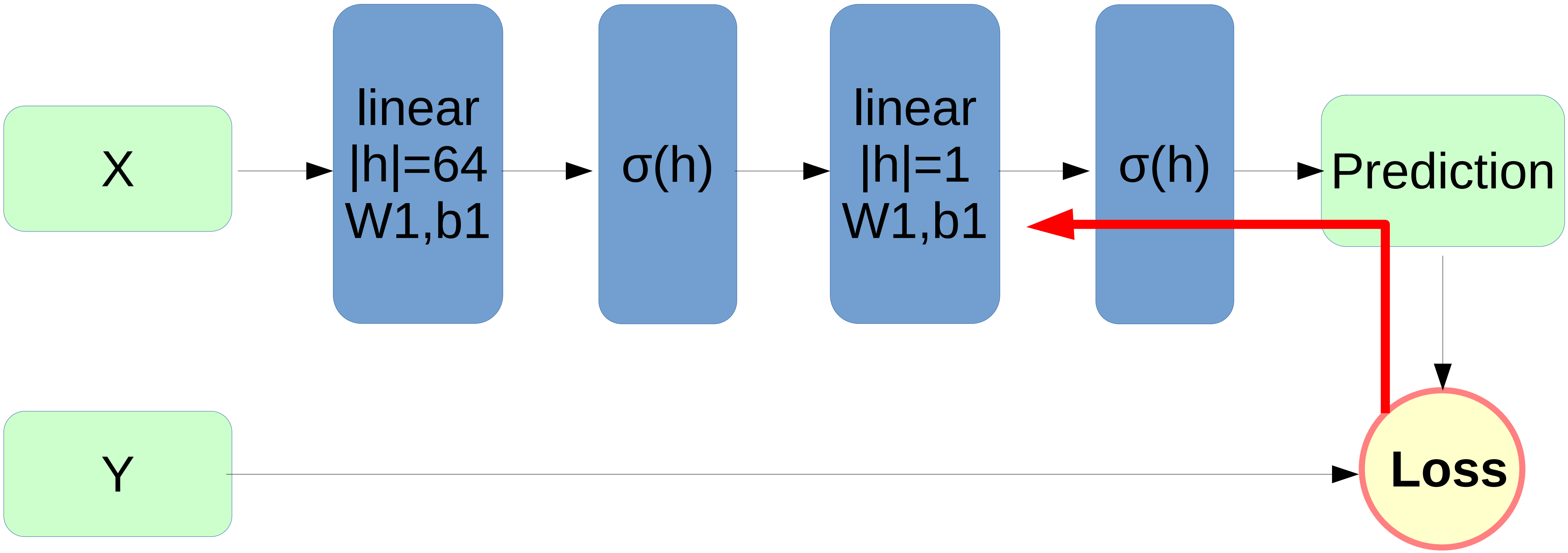
$$\frac{\partial L(\sigma(linear_{w2,b2}(\sigma(linear_{w1,b1}(x)))))}{\partial w1} = \ldots$$

Backpropagation



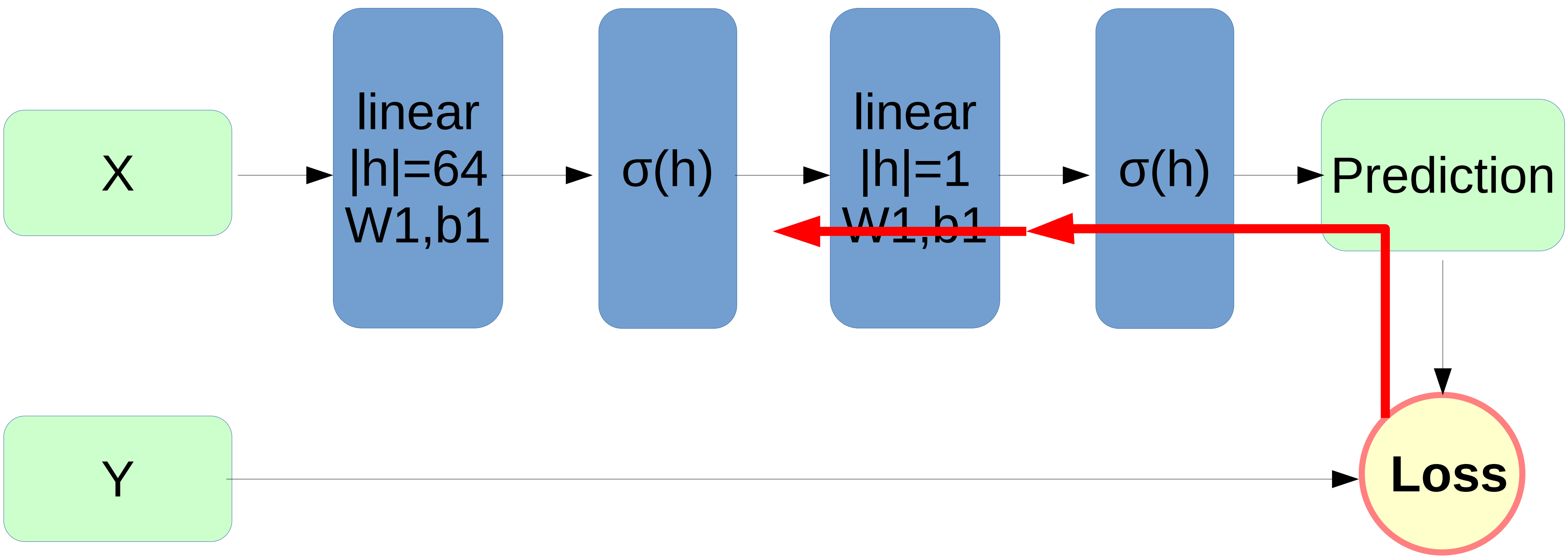
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot$$

Backpropagation



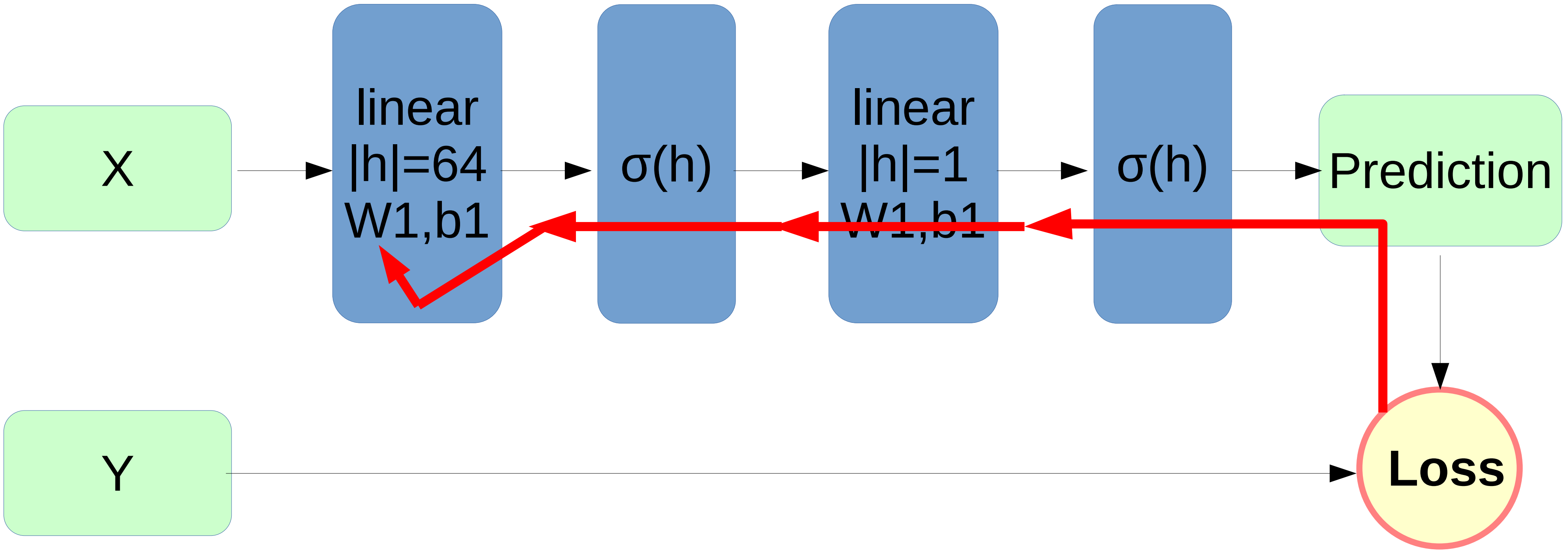
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial\, linear_{w2,b2}} \cdot$$

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 [batch size, features]

W [features, outputs]

b [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:

Hint: 1. figure out scalar case,
2. match shapes for matrices

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

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 [batch size, features]

W [features, outputs]

b [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:

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

Variable shapes:

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

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

Matrix derivatives

Let's compute:

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

Variable shapes:

X [batch size, features]

W [features, outputs]

b [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 (formulae)

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

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

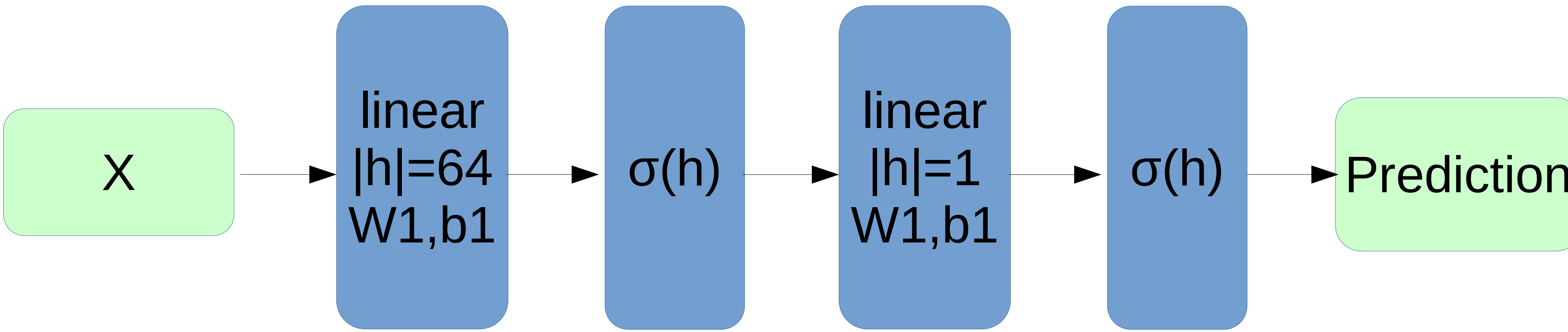
$$\frac{\partial L(X \times W + b)}{\partial W} = X^T \times \frac{\partial L}{\partial [X \times W + b]}$$

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

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

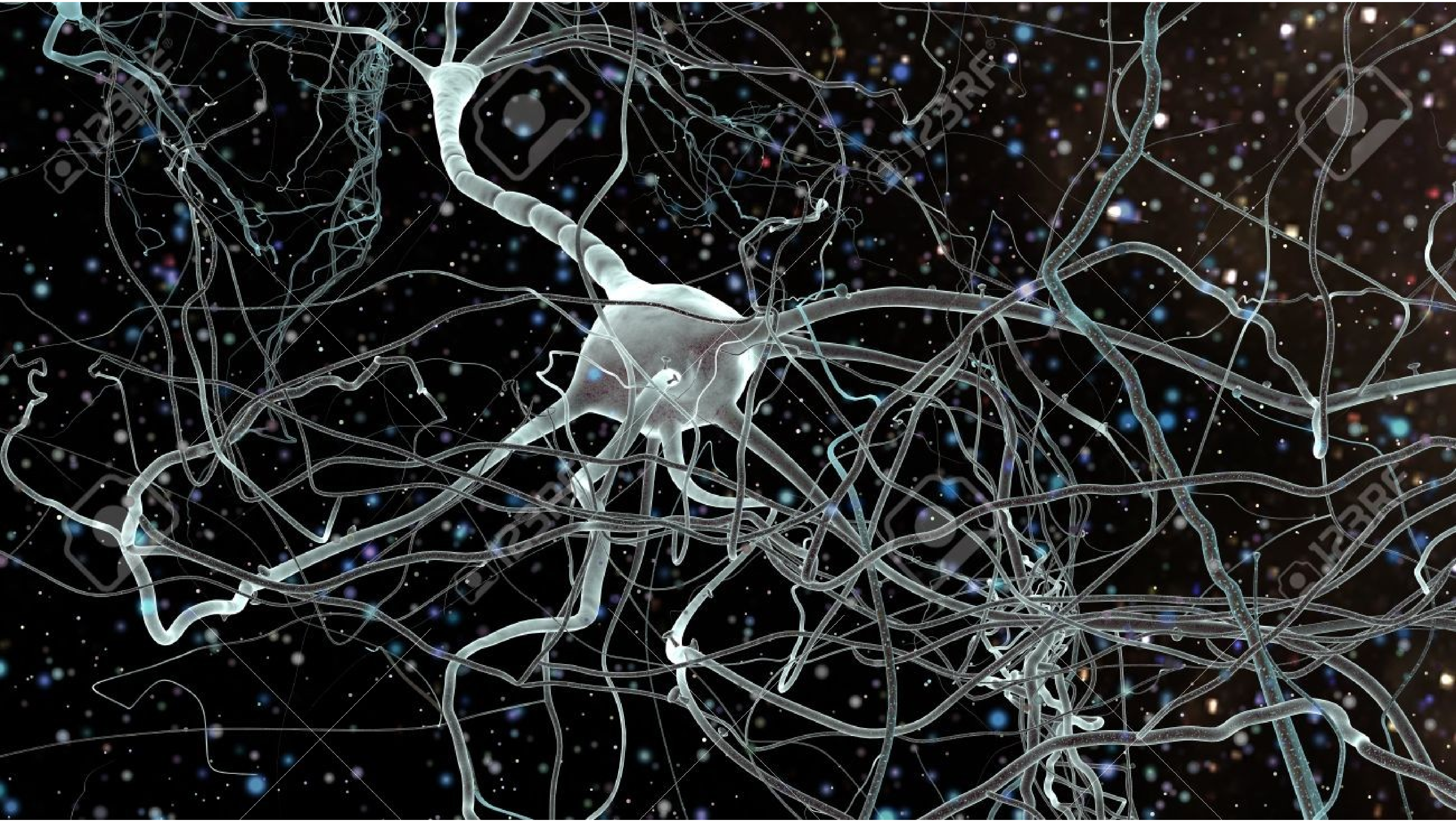
Back to neural networks

Model:

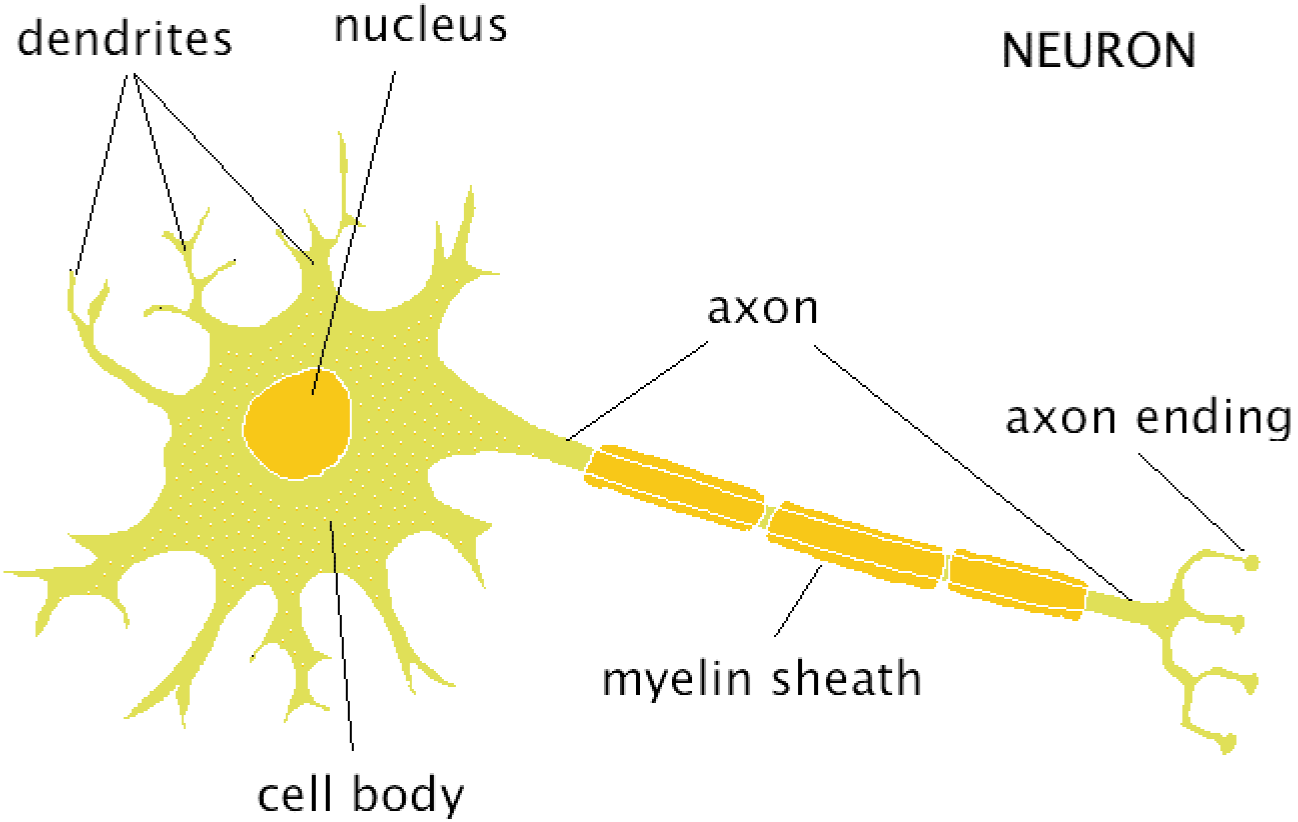


Training:

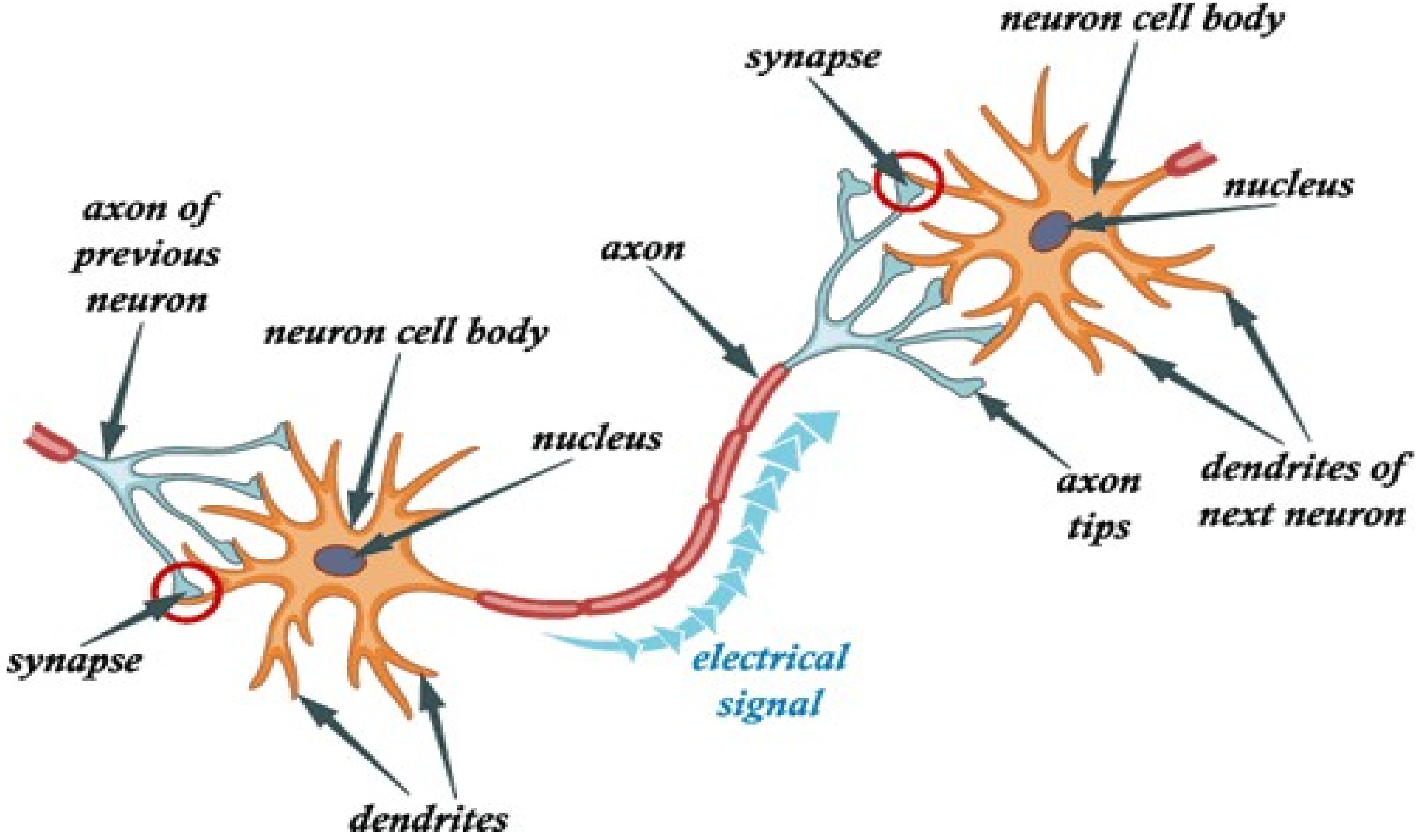
Part III: Biological inspiration



Biological inspiration

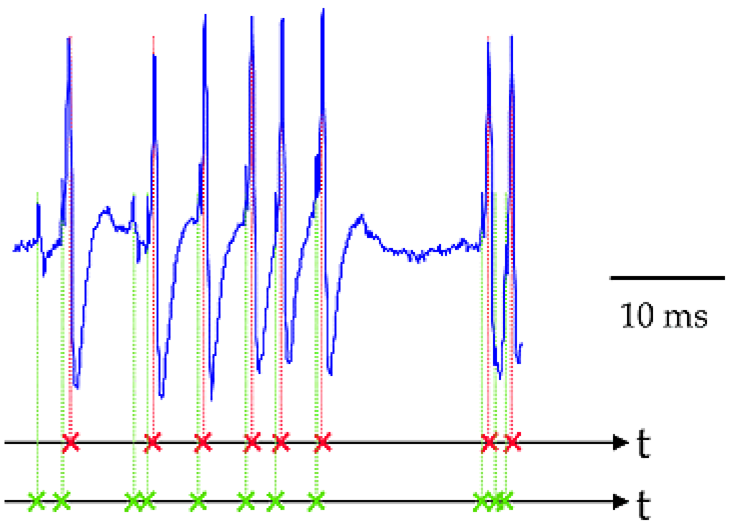


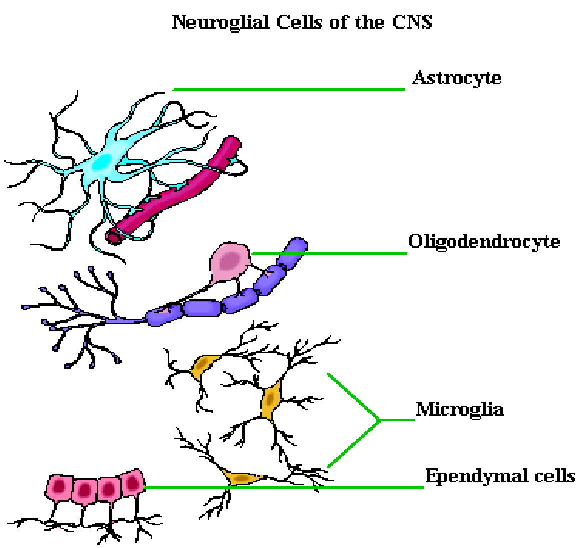
Biological inspiration



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





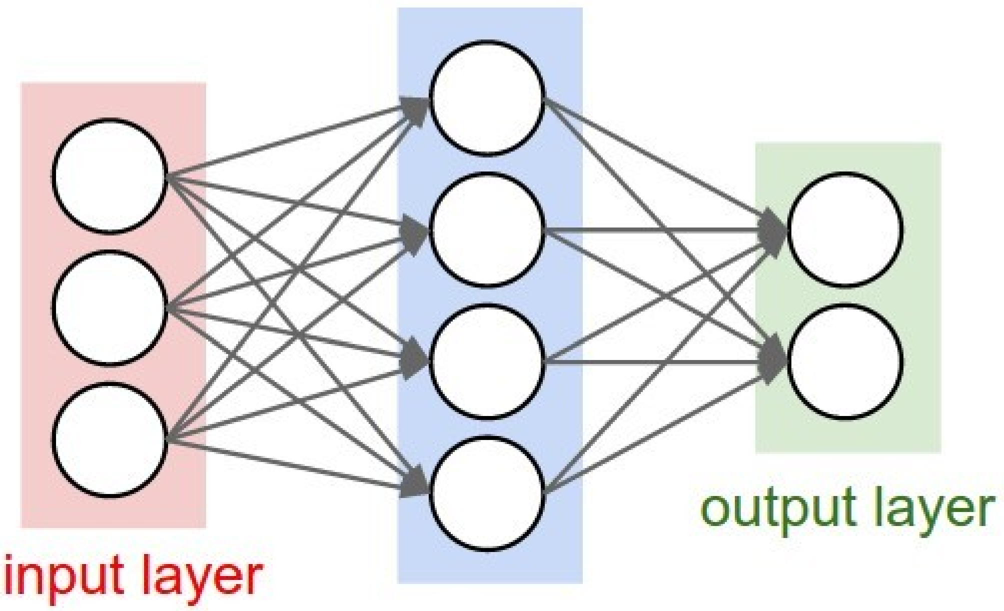
Connectionist phrasebook

- 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”

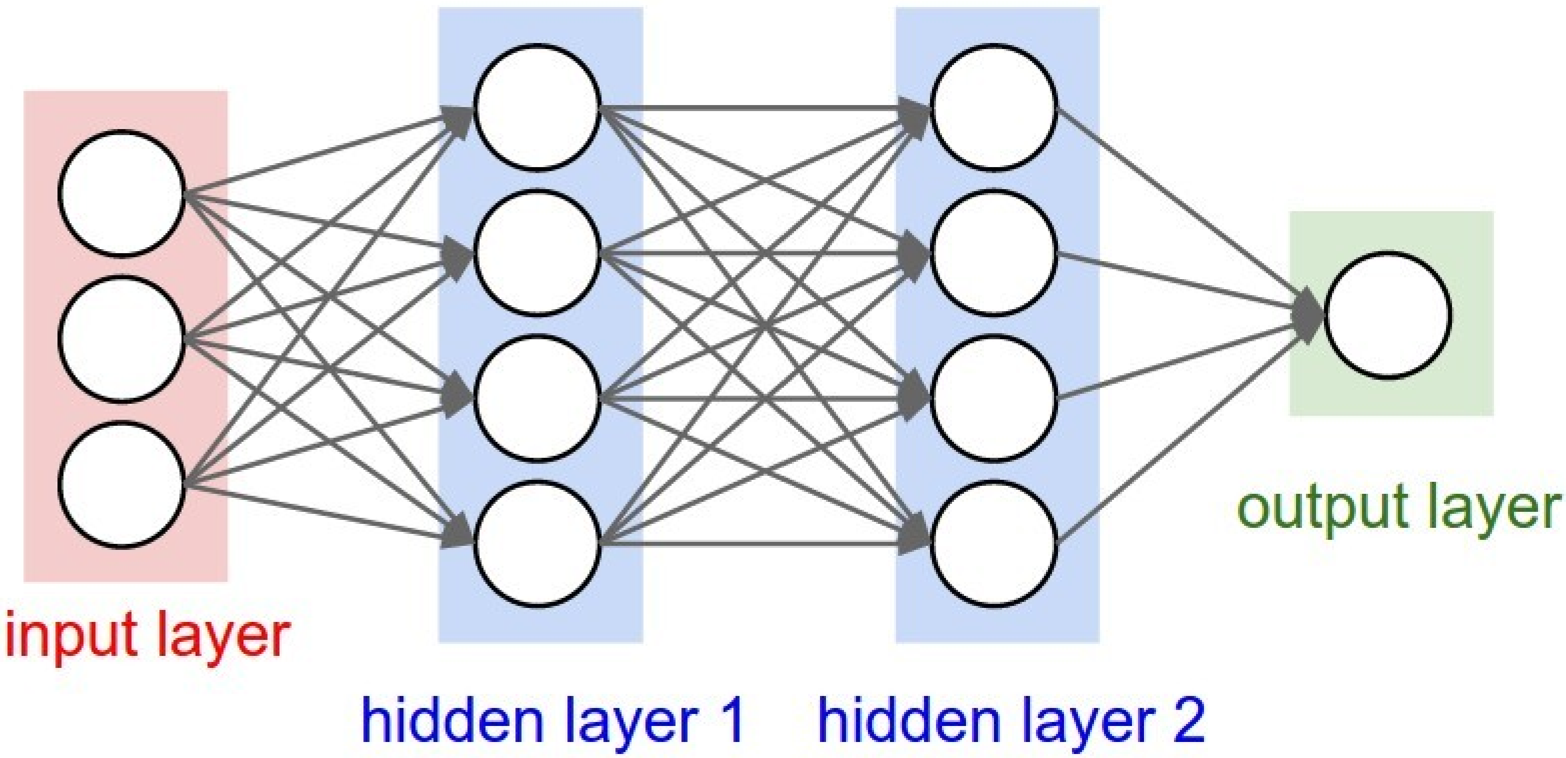
Connectionist phrasebook



Dense layer
with a nonlinearity

- “Train it via backprop!”

Connectionist phrasebook



How do we train it?

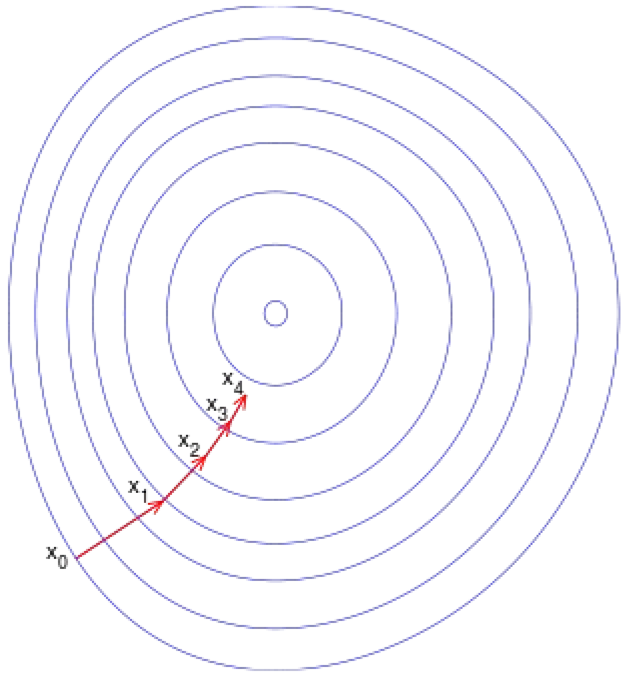
Part IV: Optimization

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?

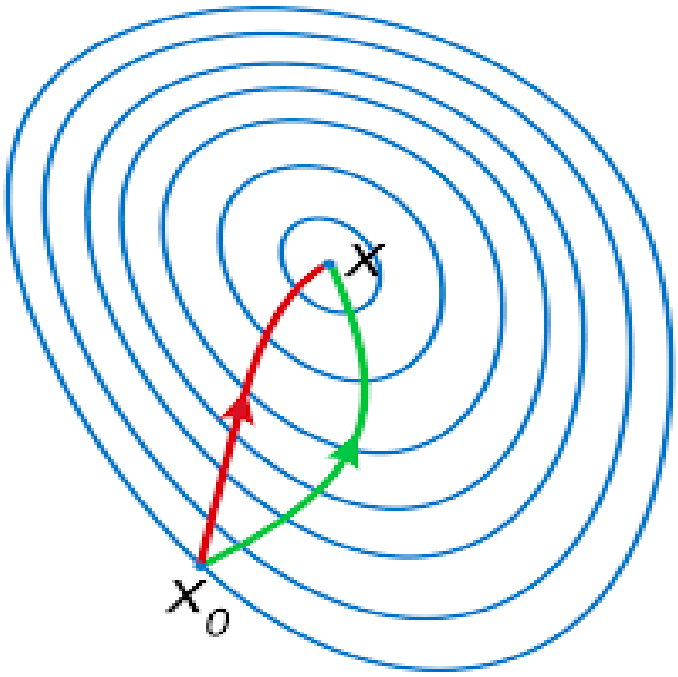
Newton-Raphson

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?

Stochastic gradient descent

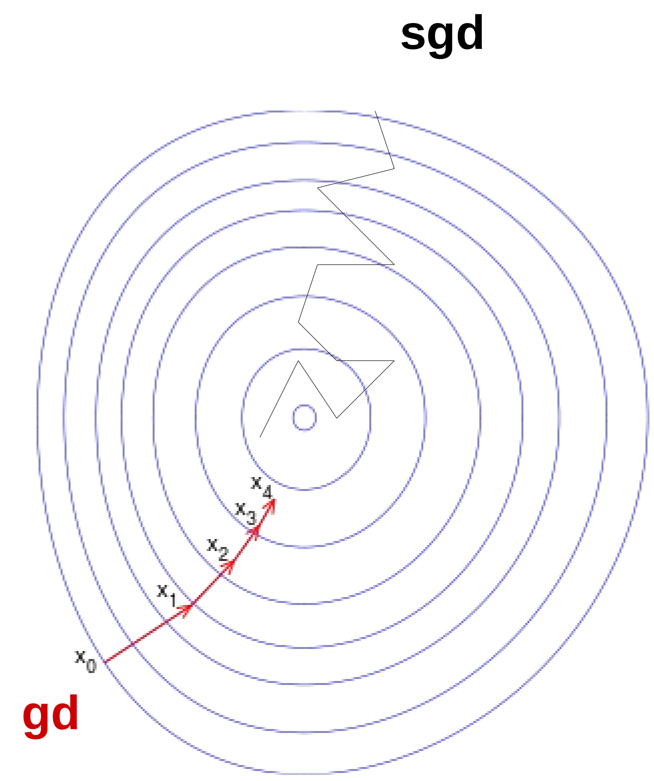
Loss function is mean over all data samples.

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

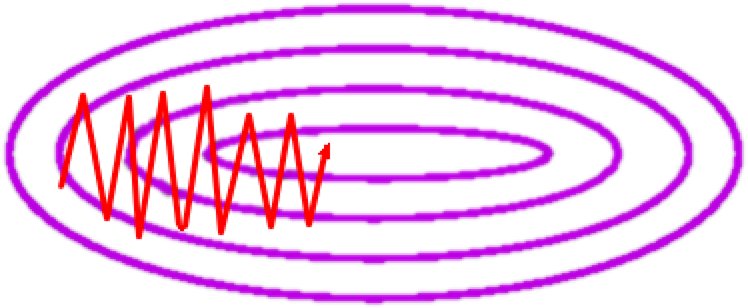
SGD with momentum

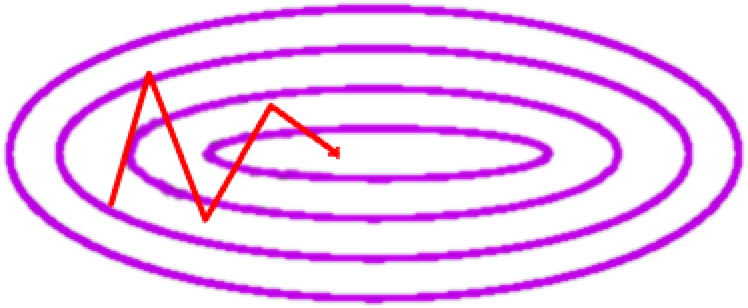
Idea: move towards "overall gradient direction",
Not just current gradient.

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

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

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





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)

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

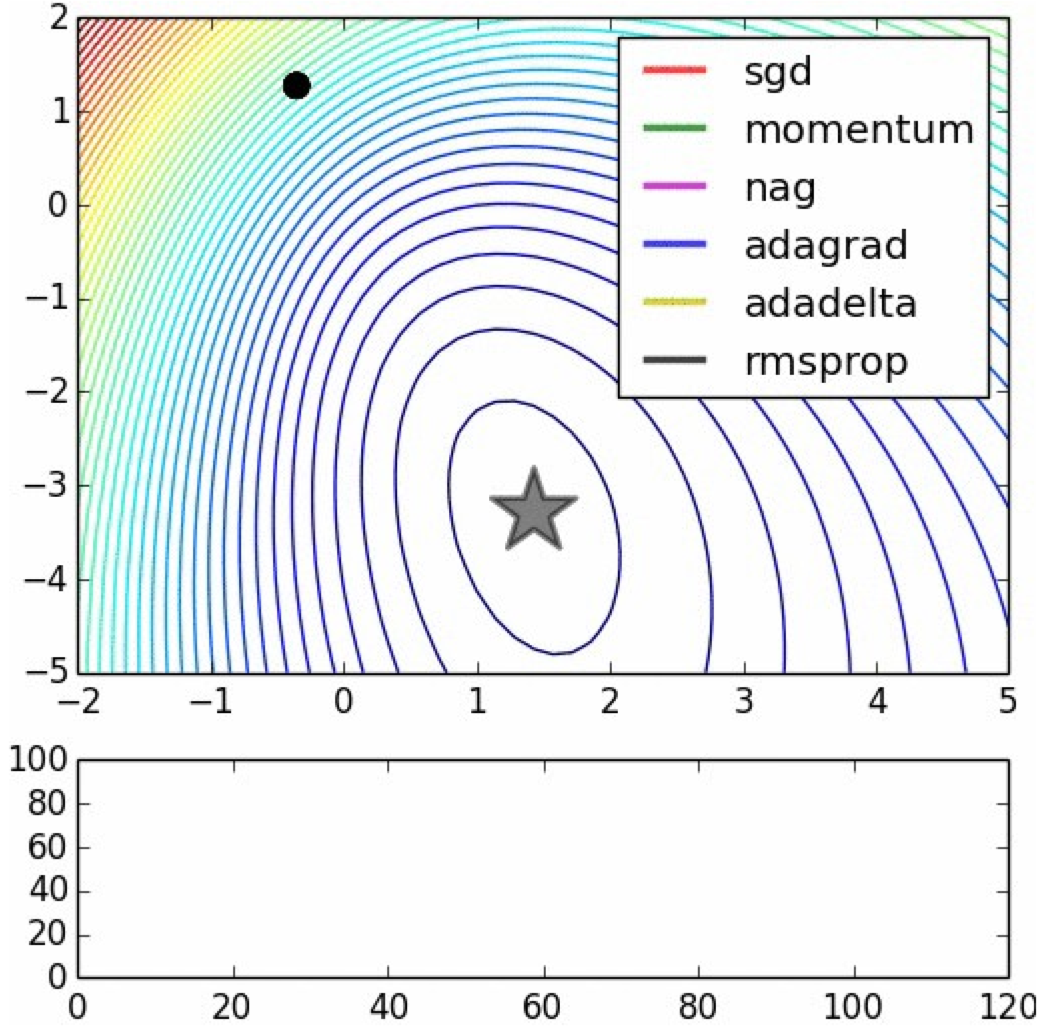
RMSProp

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

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

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

All together



Adam Optimizer

https://arxiv.org/abs/1412.6980

Update rule:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Adam Optimizer

https://arxiv.org/abs/1412.6980

≈ momentum

Update rule:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

≈ rmsprop

Adam Optimizer

https://arxiv.org/abs/1412.6980

≈ momentum

Update rule:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

≈ rmsprop

Statistics:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Moving averages of gradient and squared gradient.

(Optional) bias correction

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Newer stuff

Minor Adam improvements: Adan, AdEMAMix
Memory savings: Adafactor, Adam8bit, Adammini
Matrix-level loss geometry: Muon, Shampoo, Soap

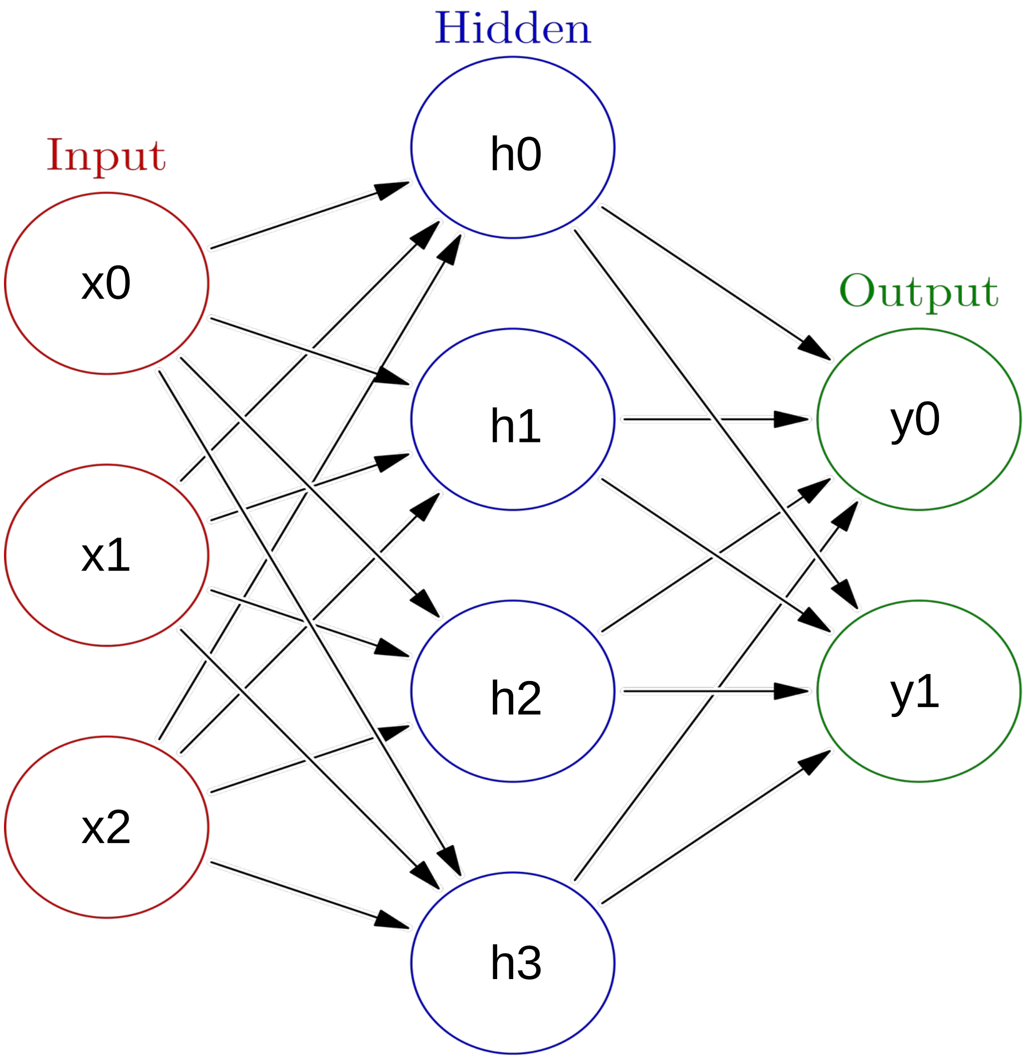
Other: LAMB(large batch), LION(sign), Prodigy(tuning-free)

Recent optimizer comparisons (September 2025):

Fantastic Pretraining Optimizers and Where to Find Them

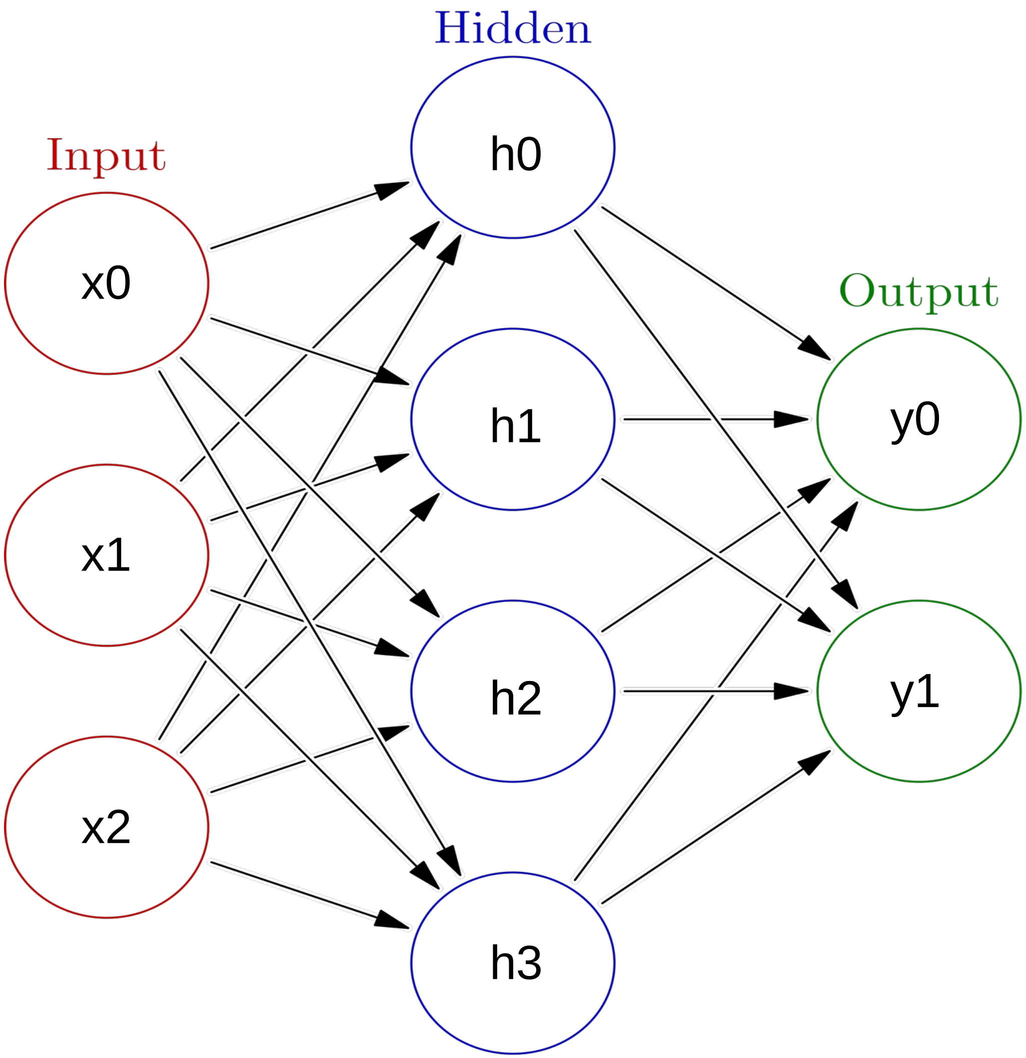
Benchmarking Optimizers for LLM Pretraining

Initialization, symmetry problem



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

Initialization, symmetry problem

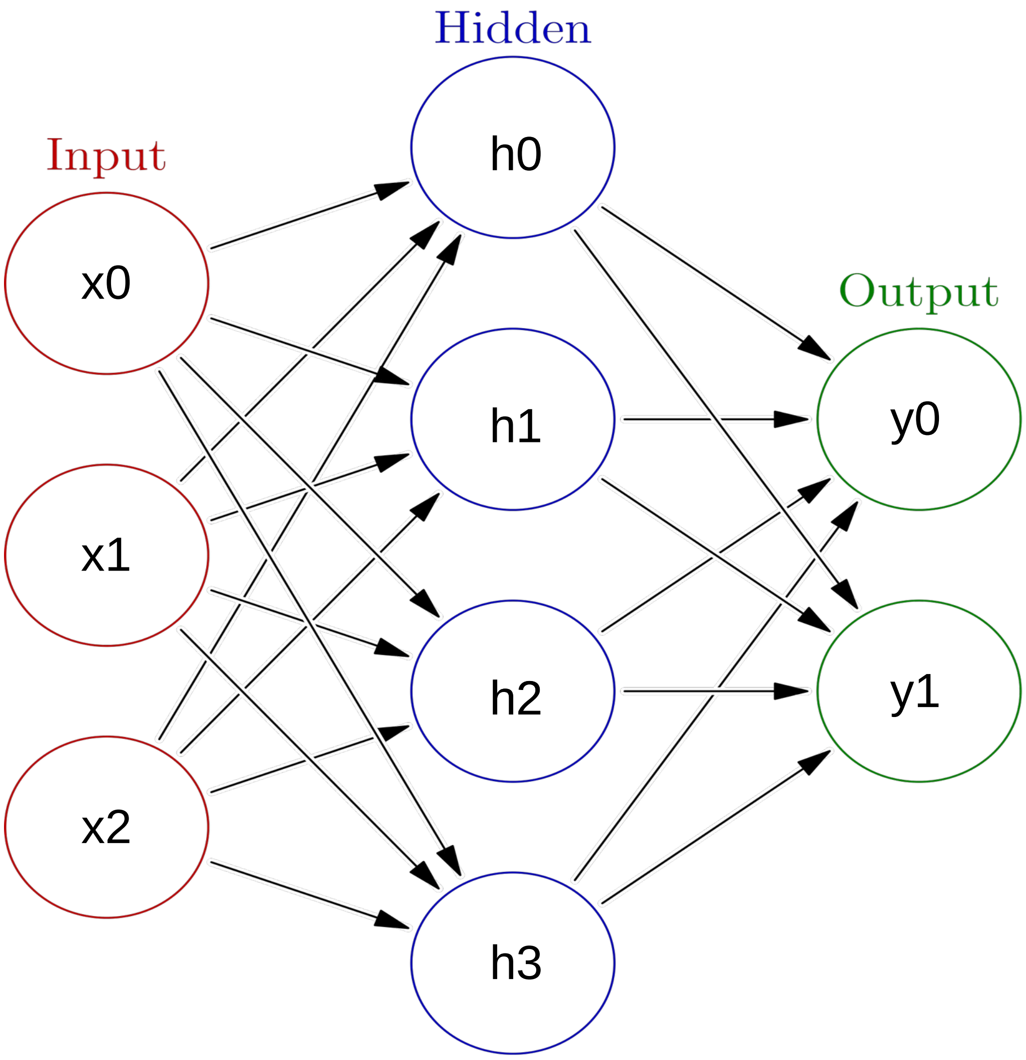


- Break the symmetry!
- Initialize with random numbers!

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

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

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?

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

Nuff

Let's go implement that!

Short break

playground.tensorflow.org