Primer on ML [2]

Overview

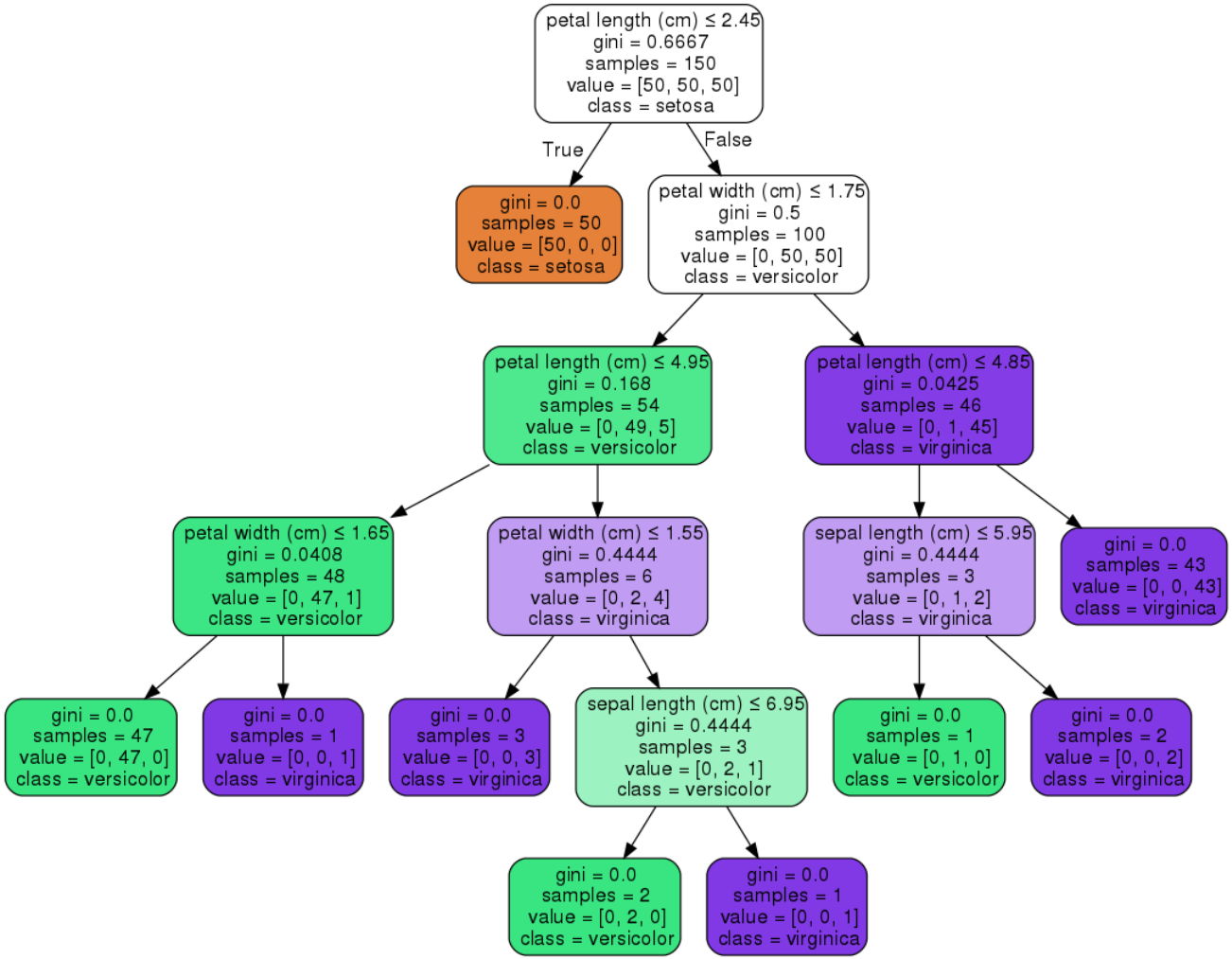
- Ensembles
  - Motivation
  - Bootstrap & Bagging $\rightarrow$ Random Forest
  - Boosting $\rightarrow$ BDT
- Neural Nets
  - Motivation
  - Building
  - Training
  - Special tricks
- $*$ Bonus
  - Convolutional NN
  - Recurrent NN

next lecture

- Thrilling applications in HEP

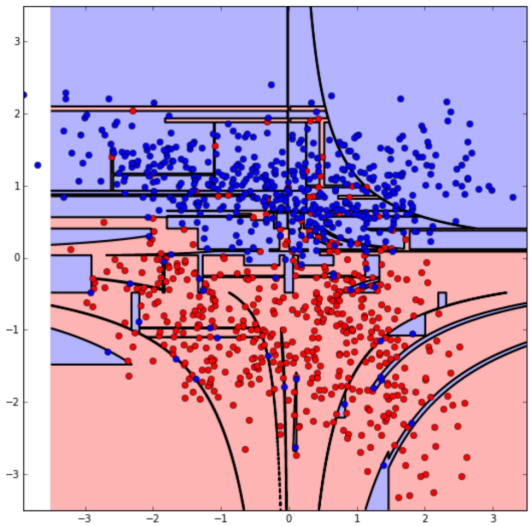
Trees [recap]

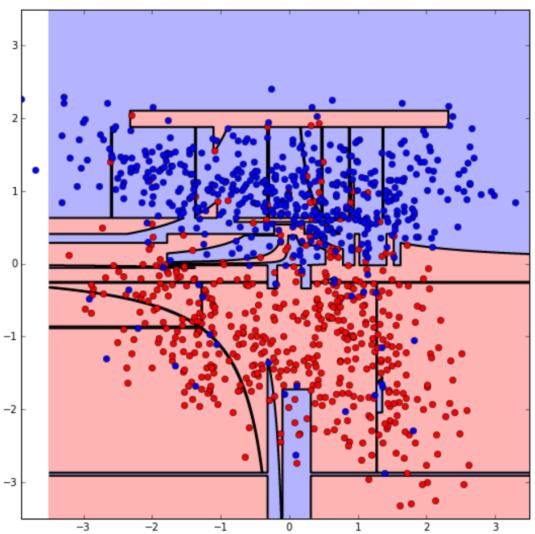
Trees [recap]


petal length (cm) ≤ 2.45
gini = 0.6667
samples = 150
value = [50, 50, 50]
class = setosa
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) ≤ 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
petal length (cm) ≤ 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
petal length (cm) ≤ 4.85
gini = 0.0425
samples = 46
value = [0, 1, 45]
class = virginica
petal width (cm) ≤ 1.65
gini = 0.0408
samples = 48
value = [0, 47, 1]
class = versicolor
petal width (cm) ≤ 1.55
gini = 0.4444
samples = 6
value = [0, 2, 4]
class = virginica
sepal length (cm) ≤ 5.95
gini = 0.4444
samples = 3
value = [0, 1, 2]
class = virginica
gini = 0.0
samples = 43
value = [0, 0, 43]
class = virginica
gini = 0.0
samples = 47
value = [0, 47, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica
gini = 0.0
samples = 3
value = [0, 0, 3]
class = virginica
sepal length (cm) ≤ 6.95
gini = 0.4444
samples = 3
value = [0, 2, 1]
class = versicolor
gini = 0.0
samples = 1
value = [0, 1, 0]
class = versicolor
gini = 0.0
samples = 2
value = [0, 0, 2]
class = virginica
gini = 0.0
samples = 2
value = [0, 2, 0]
class = versicolor
gini = 0.0
samples = 1
value = [0, 0, 1]
class = virginica

Trees [recap]

+ interpretable
+ categorical features
+ don't care about feature linear dependence
+ no need to scale inputs
- weak learners
- can overfit
- large variance with sample variation



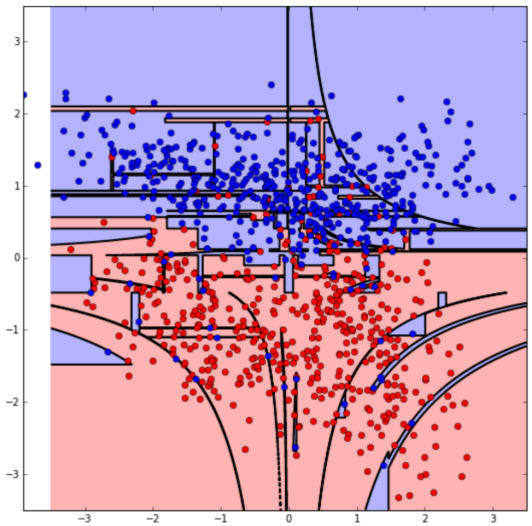


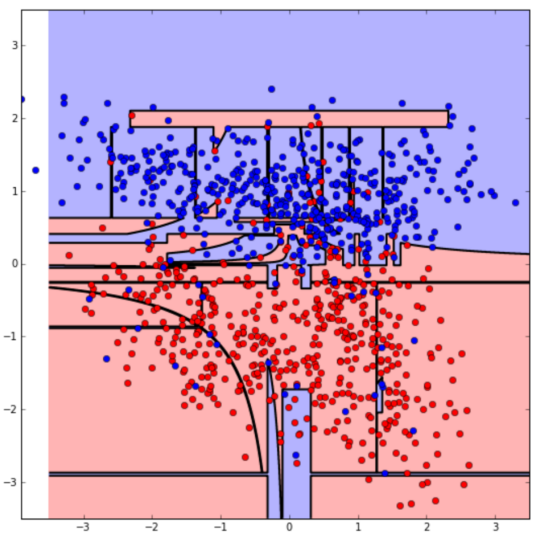
Trees [recap]

+ interpretable
+ categorical features
+ don't care about feature linear dependence
+ no need to scale inputs
- weak learners
- can overfit
- large variance with sample variation

Maybe combine to produce a better algorithm?

Yes!





Ensembles

more on bias-variance in ML [link]

Ensembles

Motivation

with data set variation:

$$\text{Error}\,[\,\text{model}\,] = \text{Noise} + \text{Bias} + \text{Variance}$$

Noise - performance of ideal model (noise in data)

Bias - average deviation from ideal model's prediction

- ▷ smaller for complex models

Variance - average sensitivity to sample replacement

- ▷ smaller for simple models

Ensembles

Motivation

with data set variation:

$$\text{Error}\,[\,\text{model}\,] = \text{Noise} + \text{Bias} + \text{Variance}$$

$$\text{Error}\,[\,\frac{1}{N}\sum_i \text{model}_i\,] = \text{Noise} + \text{Bias} + \frac{1}{N}\cdot\text{Variance} + \text{Correlation}$$

Noise - performance of ideal model (noise in data)

Bias - average deviation from ideal model's prediction

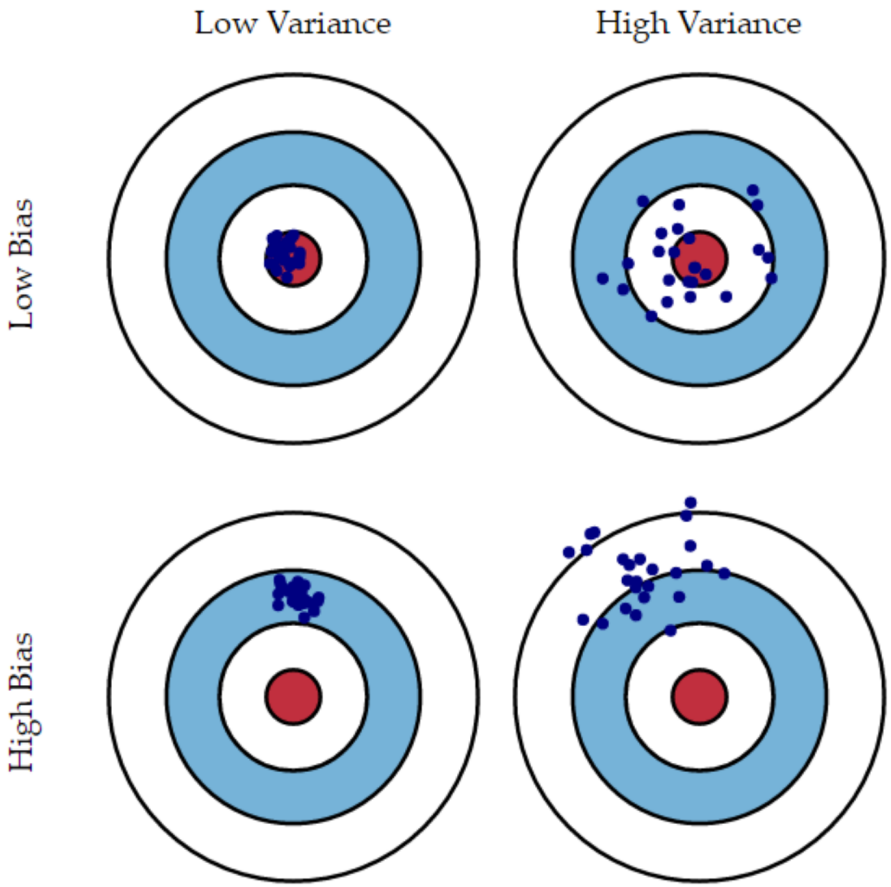
- ▷ smaller for complex models

Variance - average sensitivity to sample replacement

- ▷ smaller for simple models

Ensembles

Motivation



Ensembles

Motivation

$$\text{Error}\left[\frac{1}{N}\sum_i \text{model}_i\right] = \text{Noise} + \text{Bias} + \frac{1}{N}\cdot\text{Variance} + \text{Correlation}$$

Can we reduce Correlation?

Ensembles

Motivation

$$\text{Error}\left[\frac{1}{N}\sum_i \text{model}_i\right] = \text{Noise} + \text{Bias} + \frac{1}{N}\cdot\text{Variance} + \text{Correlation}$$

Can we reduce Correlation? Yes

- Bagging
  - train on subsample of events
- Random subspace method (feature bagging)
  - train on subsample of features

Ensembles

Motivation

$$\text{Error}\left[\frac{1}{N}\sum_i \text{model}_i\right] = \text{Noise} + \text{Bias} + \frac{1}{N}\cdot\text{Variance} + \text{Correlation}$$

Can we reduce Correlation? Yes

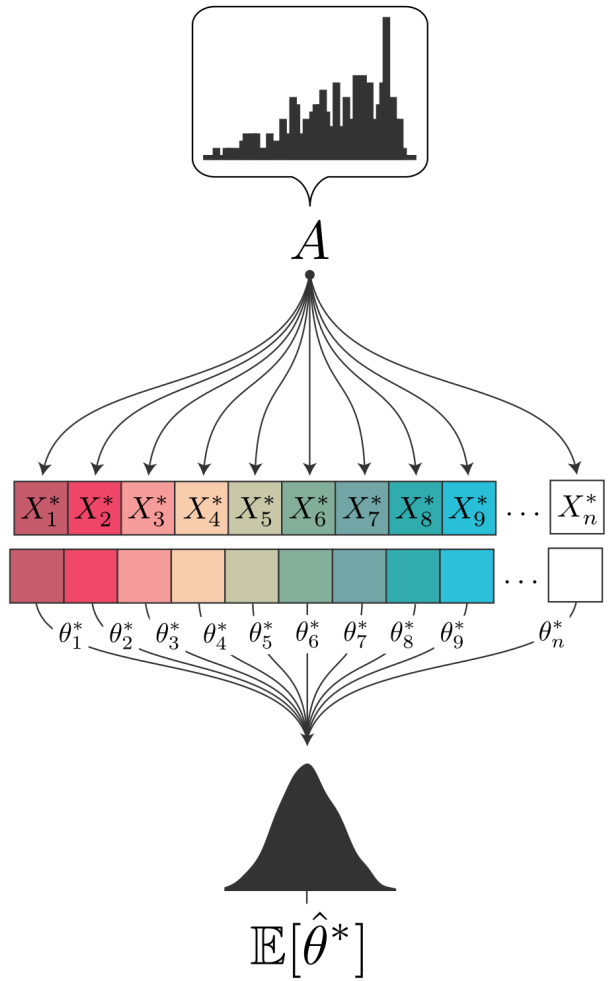
- Bagging
  - train on subsample of events $\leftarrow$ hyperparameter :(
- Random subspace method (feature bagging)
  - train on subsample of features

more on that [link]

Ensembles

Bootstrap

- same-size sampling with replacement
- widely used in
  - confidence intervals
  - hypothesis tests



Random Forest

sklearn guide [link]

Random Forest

$$\text{RF} = \frac{1}{N}\sum_{i=1}^{N} \text{tree}_i + \text{bootstrap each tree} + \text{bagging each split}$$

→ for $i = 1, 2, \ldots N$:

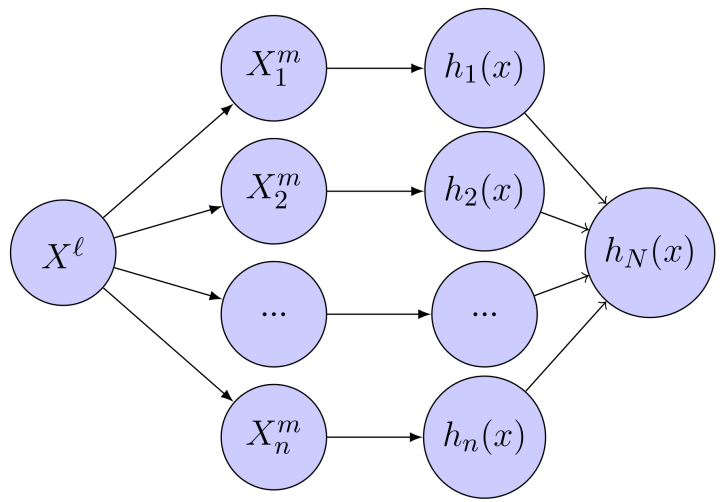
- bootstrap sample X_i
- while !stopping criteria:
  - pick p random features out of d
  - make split

→ prediction:

r $\text{RF}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} \text{tree}_i(\mathbf{x})$

c $\text{RF}(\mathbf{x}) = \text{sign}\frac{1}{N}\sum_{i=1}^{N} \text{tree}_i(\mathbf{x})$

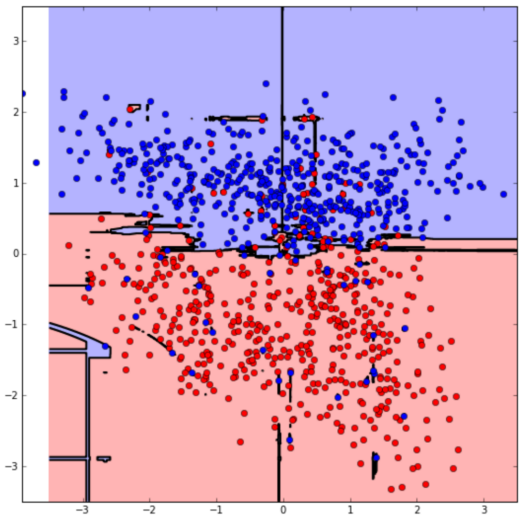
NB: proba average in sklearn

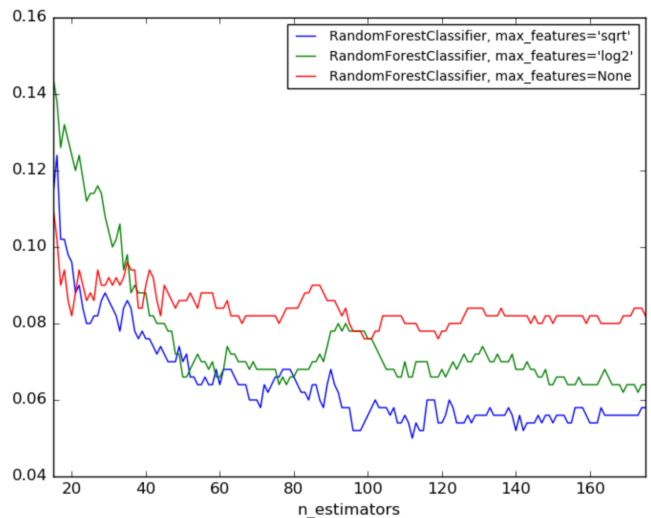


Random Forest

pros & cons

+ trees' pros
+ less overfitting with increasing N
+ better capturing of non-linearities
+ parallelized
+ out-of-bag score
+ more robust (than single tree)
- not lightweight
- previous trees' improvements lost



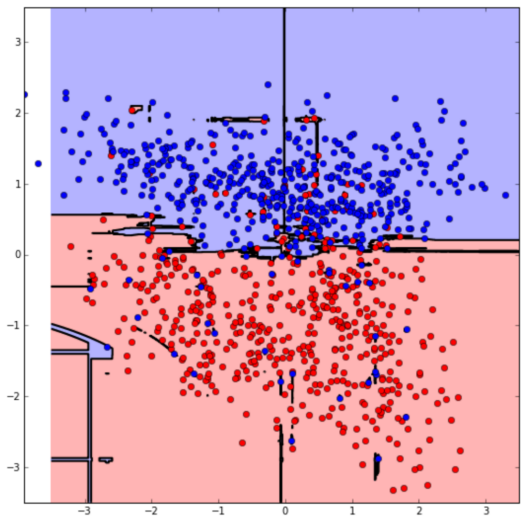


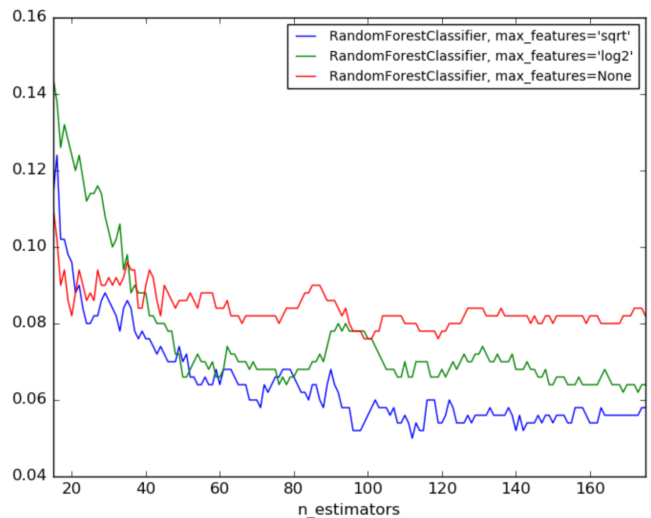
Random Forest

pros & cons

+ trees' pros
+ less overfitting with increasing N
+ better capturing of non-linearities
+ parallelized
+ out-of-bag score
+ more robust (than single tree)
- not lightweight
- previous trees' improvements lost

Boosting





Gradient Boosting

Gradient Boosting

Regression: simplified

Init. learner e.g. $a_0 = h_0 = \frac{1}{N} \sum_i y_i$

Add learners greedily to improve results of previous ones:

1. residuals:

   $$s_i = y_i - \sum_{n=1}^{N-1} h_n(\mathbf{x}) = y_i - a_{N-1}(\mathbf{x}_i)$$

2. train next algorithm on residuals:

   $$h_N = \arg\min_{h \in H} \sum_i (h(x_i) - s_i)^2$$

3. add to composition:

   $$a_N = a_{N-1} + h_N$$

Gradient Boosting

Regression: simplified

Init. learner e.g. $a_0 = h_0 = \frac{1}{N} \sum_i y_i$

Add learners greedily to improve results of previous ones:

1. residuals:

   $s_i = y_i - \sum_{n=1}^{N-1} h_n(\mathbf{x}) = y_i - a_{N-1}(\mathbf{x}_i)$ ← looks like L2 gradient

2. train next algorithm on residuals:

   $h_N = \arg\min_{h \in H} \sum_i (h(x_i) - s_i)^2$

3. add to composition:

   $a_N = a_{N-1} + h_N$

Gradient Boosting

Regression: simplified

Init. learner e.g. $a_0 = h_0 = \frac{1}{N} \sum_i y_i$

Add learners greedily to improve results of previous ones:

1. residuals:

   $s_i = y_i - \sum_{n=1}^{N-1} h_n(\mathbf{x}) = y_i - a_{N-1}(\mathbf{x}_i)$ ← looks like L2 gradient

2. train next algorithm on residuals: ← and here we learn this gradient

   $h_N = \arg\min_{h \in H} \sum_i (h(x_i) - s_i)^2$

3. add to composition:

   $a_N = a_{N-1} + h_N$

Gradient Boosting

Regression: simplified

Init. learner e.g. $a_0 = h_0 = \frac{1}{N} \sum_i y_i$

Add learners greedily to improve results of previous ones:

1. residuals:

   $s_i = y_i - \sum_{n=1}^{N-1} h_n(\mathbf{x}) = y_i - a_{N-1}(\mathbf{x}_i)$ ← looks like L2 gradient

2. train next algorithm on residuals: ← and here we learn this gradient

   $h_N = \arg\min_{h \in H} \sum_i (h(x_i) - s_i)^2$

3. add to composition: ← and finally descend in algorithm space

   $a_N = a_{N-1} + h_N$

Gradient Boosting

GBM [Friedman, 2001]

- More general approach:

Define $\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h(x_i)) \to \min_{\gamma, h}$

At each iteration:

1. $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=a_{N-1}(x_i)}$
2. regression on s_i:

   $h_N = \arg\min_{h \in H} \sum_{i=1}^{\ell} (h(x_i) - s_i)^2$
3. optimal γ with plain grad. descent:

   $\gamma_N = \arg\min_{\gamma} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h_N(x_i))$
4. $a_N \leftarrow a_{N-1} + \gamma_N h_N$

Gradient Boosting

GBM [Friedman, 2001]

- More general approach:

Define $\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h(x_i)) \to \min_{\gamma, h}$

At each iteration:

1. $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=a_{N-1}(x_i)}$
2. regression on s_i:

   $h_N = \arg\min_{h \in H} \sum_{i=1}^{\ell} (h(x_i) - s_i)^2$
3. optimal γ with plain grad. descent:

   $\gamma_N = \arg\min_{\gamma} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h_N(x_i))$
4. $a_N \leftarrow a_{N-1} + \gamma_N h_N$

- classification?

Gradient Boosting

GBM [Friedman, 2001]

- More general approach:

Define $\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h(x_i)) \to \min_{\gamma, h}$

At each iteration:

1. $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=a_{N-1}(x_i)}$
2. regression on s_i:

$h_N = \arg\min_{h \in H} \sum_{i=1}^{\ell} (h(x_i) - s_i)^2$

3. optimal γ with plain grad. descent:

$\gamma_N = \arg\min_{\gamma} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h_N(x_i))$

4. $a_N \leftarrow a_{N-1} + \gamma_N h_N$

- classification?
- BDT = GB with trees
- but you can boost whatever you want*

Gradient Boosting

GBM [Friedman, 2001]

- More general approach:

Define $\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h(x_i)) \to \min_{\gamma, h}$

At each iteration:

1. $s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=a_{N-1}(x_i)}$
2. regression on s_i:

   $h_N = \arg\min_{h \in H} \sum_{i=1}^{\ell} (h(x_i) - s_i)^2$
3. optimal γ with plain grad. descent:

   $\gamma_N = \arg\min_{\gamma} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma h_N(x_i))$
4. $a_N \leftarrow a_{N-1} + \gamma_N h_N$

- classification?
- BDT = GB with trees
- but you can boost whatever you want*
- modern libraries are slightly different

intuition and formulas [1], [2]
and nice visuals [3], [4]

BDT

pros & cons

\+ damn good performance

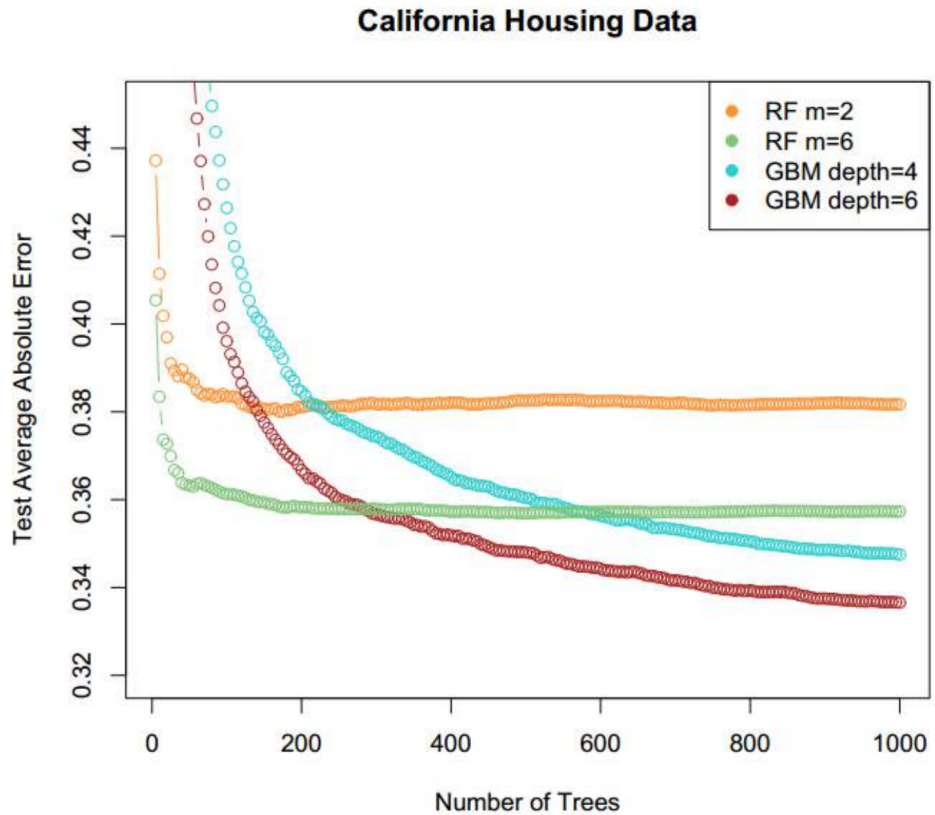
\- not interpretable (directly)

\- overfits

You might want try these ones:

- LightGBM
- XGBoost
- CatBoost

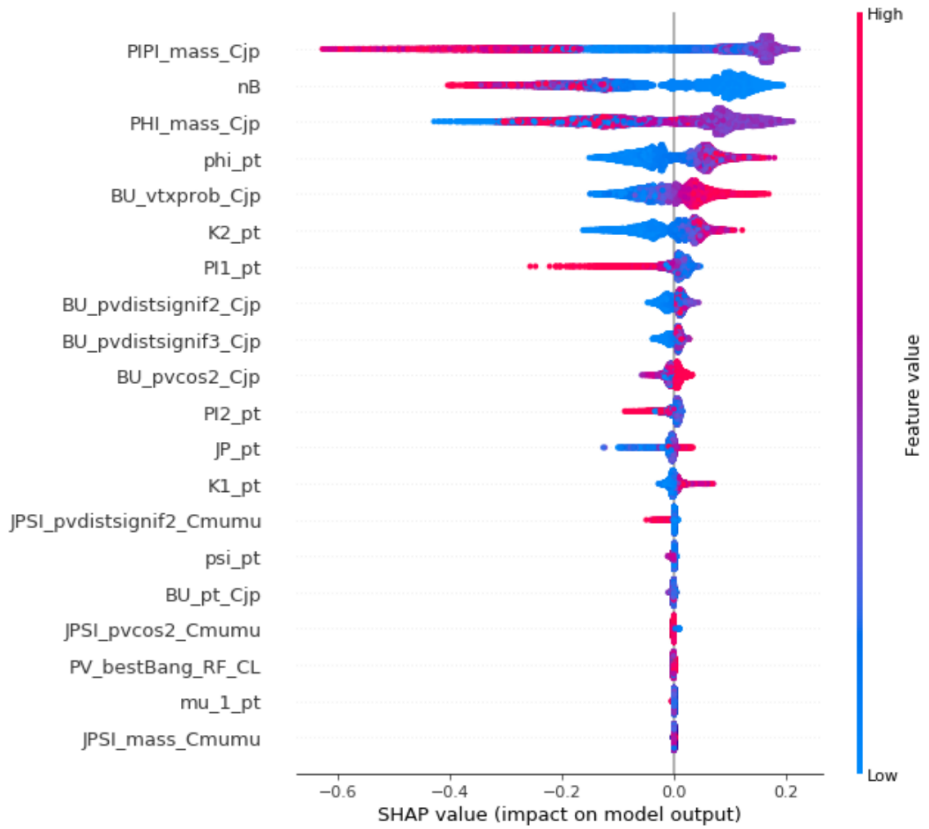
? AdaBoost (used in TMVA)



Sidenote

Feature importance

- Linear models:
  - Weights' values
- Trees:
  - Gain
  - Frequency
- But you should probably use SHAP values [link]
  - Makes BDT interpretable!

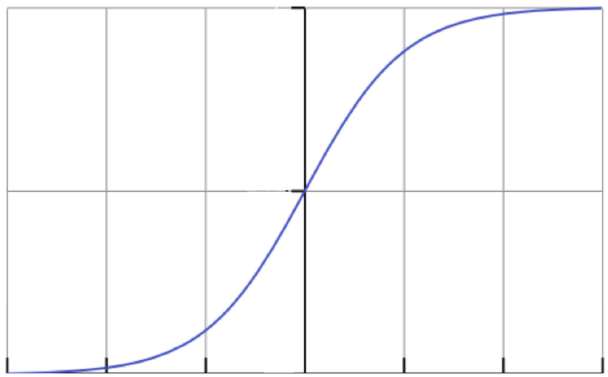



WE NEED TO GO
DEEPER

the following are the slides from [1], [2]
huge credit to them!

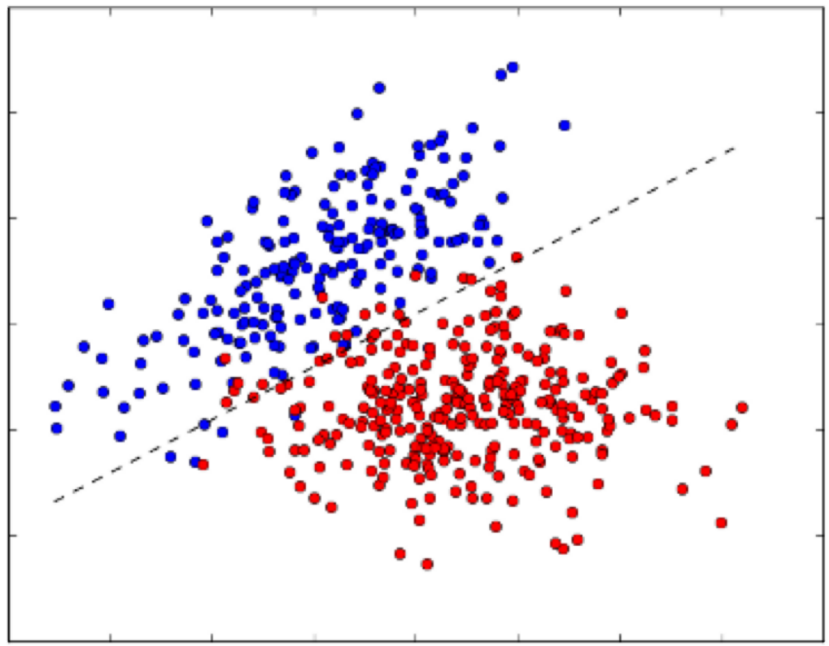
Principles of linear vs. nonlinear models

Recap: linear models

$W \cdot x + b \longrightarrow$  $\longrightarrow P(y = +1|x)$

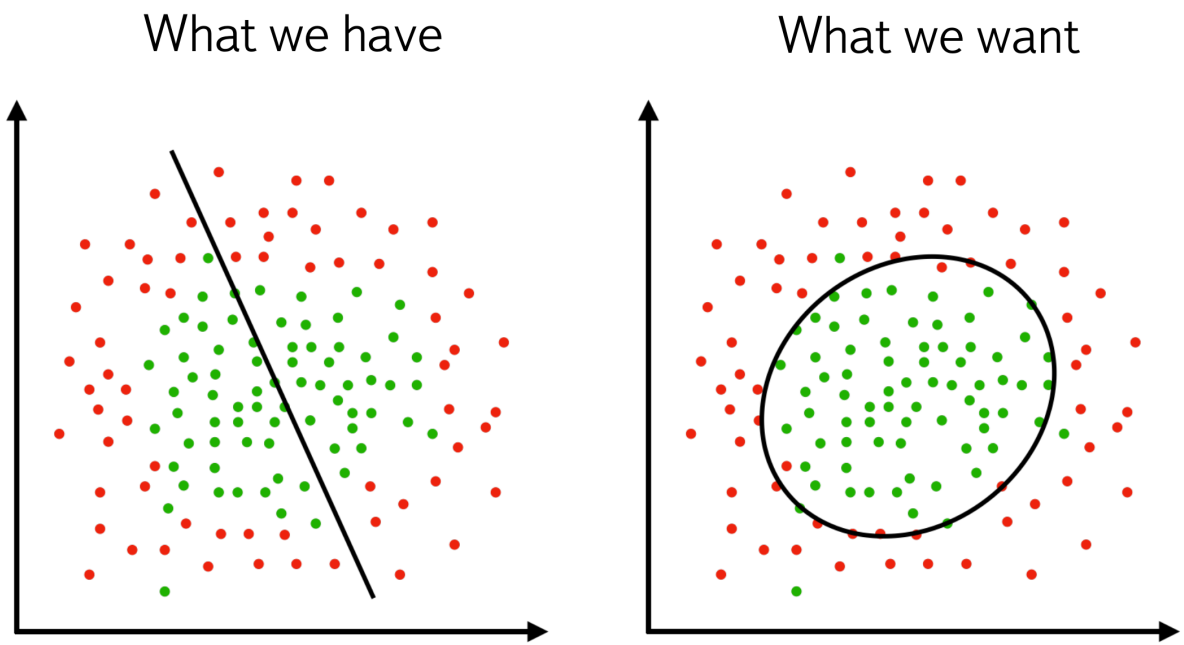
- x: features vector
- W, b: model slope and intercept

Linear dependency



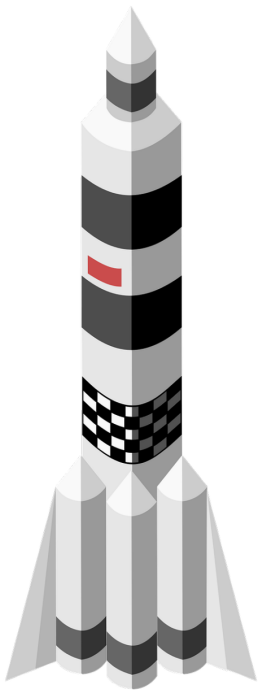
5

Nonlinear dependencies



6

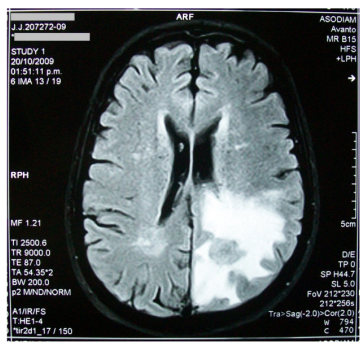
Extremely nonlinear dependencies



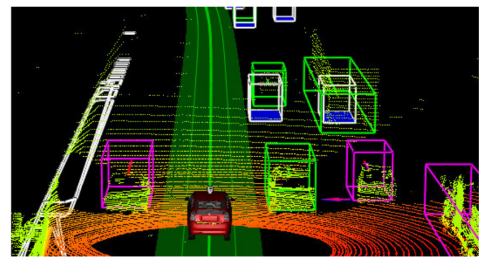
rocket

Most of the dependencies in this world!

cat

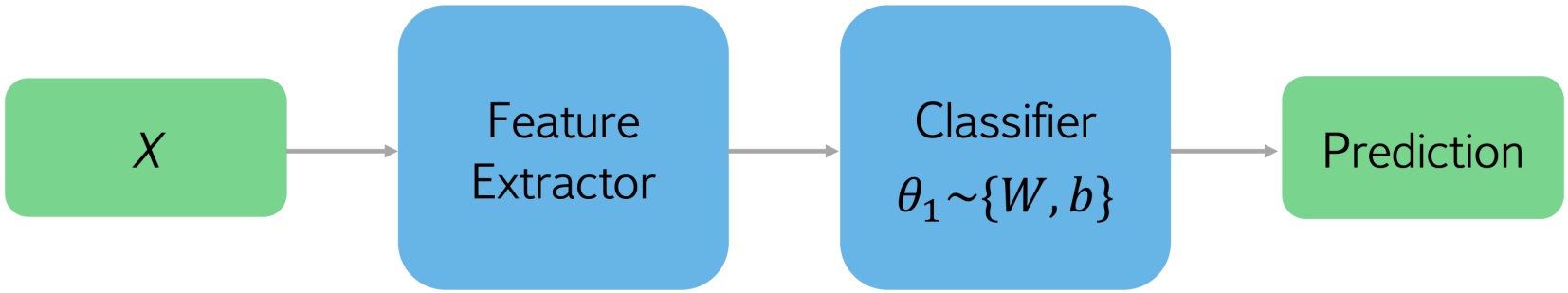


Brain tumors
as seen on MRI



Self-driving LiDAR [8]

Extremely nonlinear dependencies



- Decouple feature extractor from the classifier
- Training and inference really can have multiple stages!

9

Feature *extraction?*

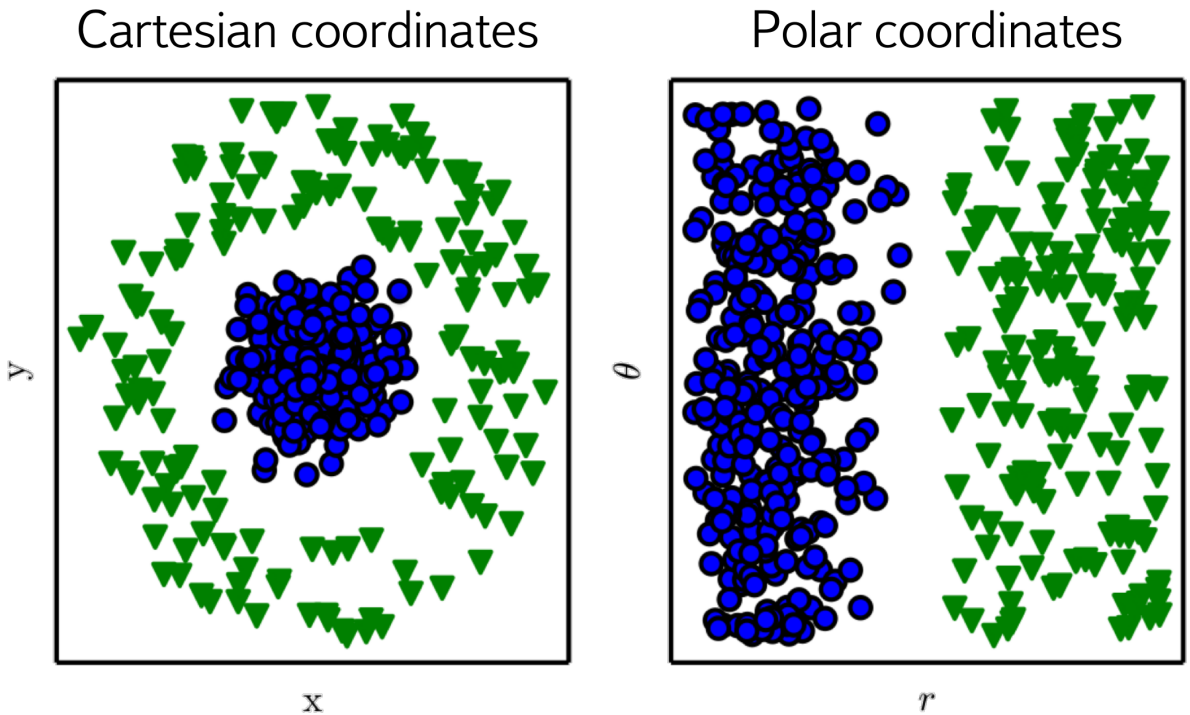
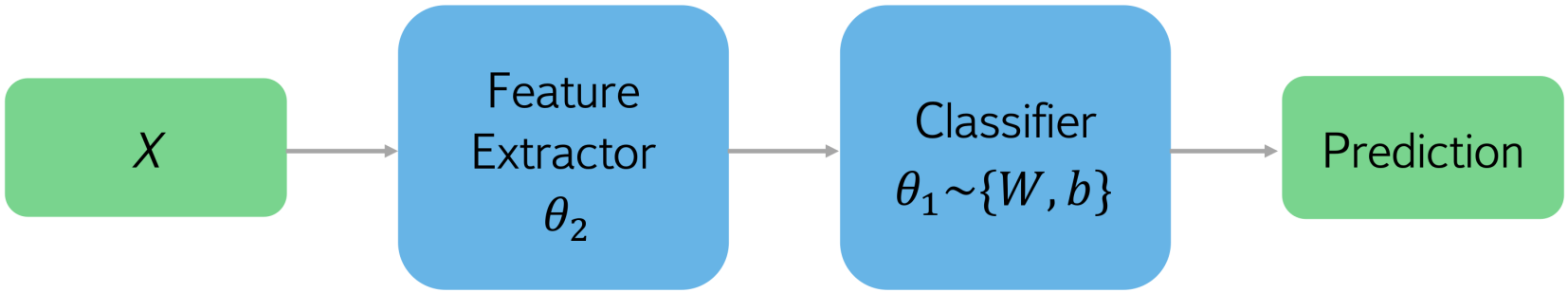


Image: Ian Goodfellow et al.

10

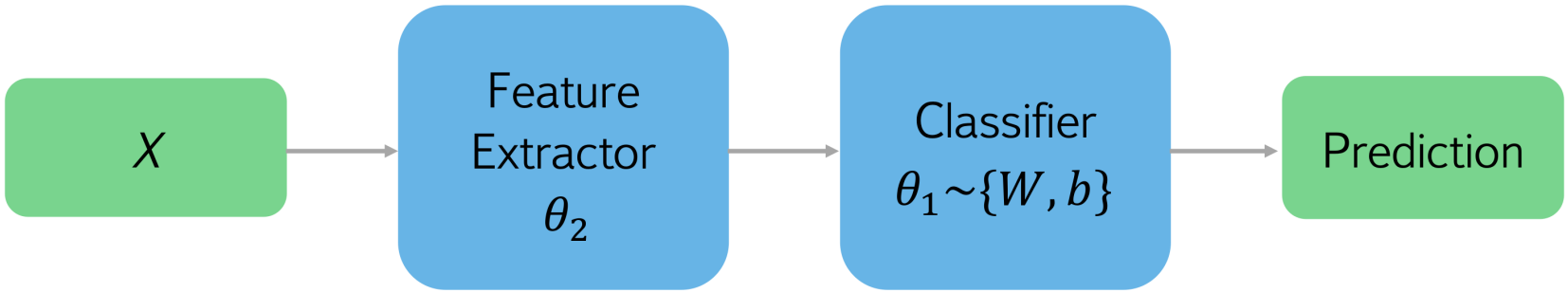
Feature extraction



- *Manually* extracted features
- Training is left with finding $\underset{\theta_1}{\operatorname{argmin}} L\big(y, P(y \mid x)\big)$

12

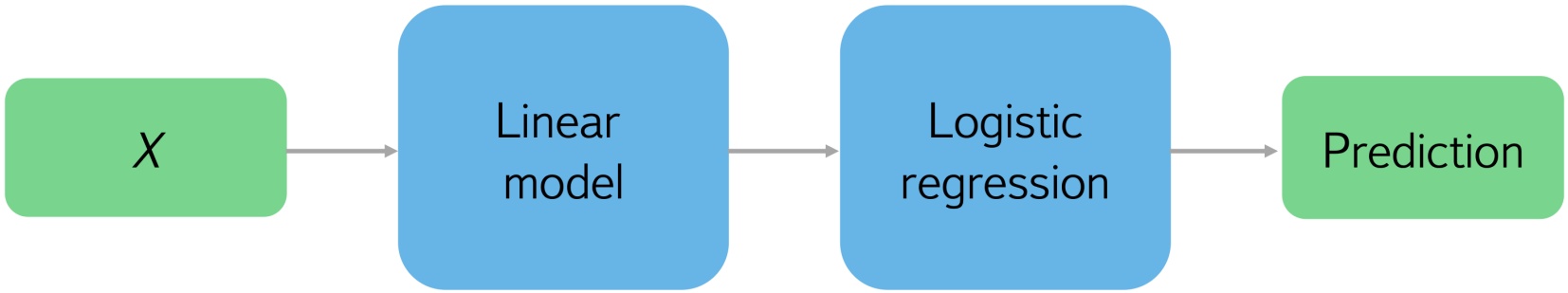
Can it be done automatically?



- *Automatically* extracted features
- Training still needs to find $\underset{\theta_1}{\operatorname{argmin}} L\big(y, P(y \mid x)\big)$
- Yet, we face a different challenge of finding θ_2

13

Try stacked linear models



- Compute features $h_j = \sum_i w_{ij}^h x_i + b_j^h \qquad j \in \{1,2,\ldots,n\}$
- Eventual output of the model $y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$
- Train by jointly minimizing loss to search for $\underset{w^h,w^0,b^h,b^0}{\operatorname{argmin}} L\big(y, P(y \mid x)\big)$

14

A question

- Will stacking linear functions improve quality?

15

Answer: **no**

- Why?
- A combination of linear models is a linear model:

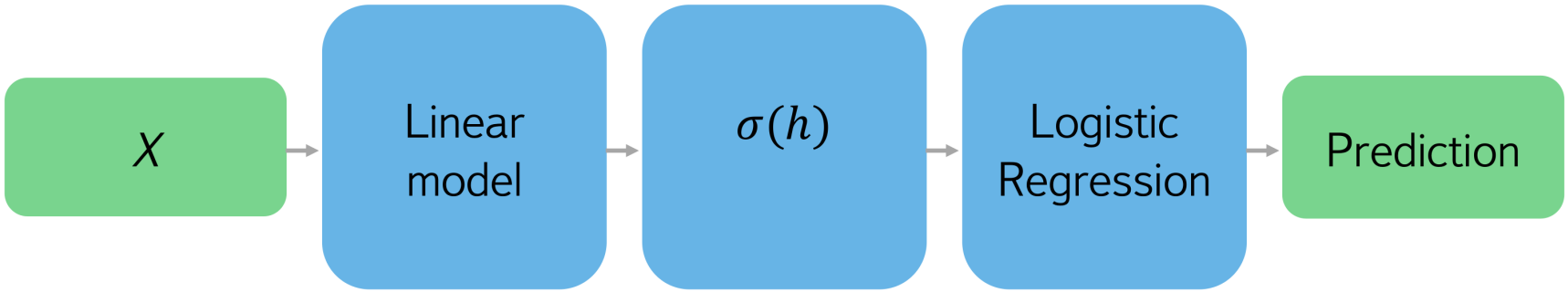
$$P(y \mid x) = \sigma\left(\sum_j w_j^o \left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$$

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

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

16

The nonlinearity



- Compute features $h_j = \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) \quad j \in \{1,2,\ldots,n\}$
- Eventual output of the model $y_{pred} = \sigma\left(\sum_j w_j^o h_j + b^o\right)$
- Compositionality: $P(y \mid x) = \sigma\left(\sum_j w_j^o \sigma\left(\sum_i w_{ij}^h x_i + b_j^h\right) + b^o\right)$

17

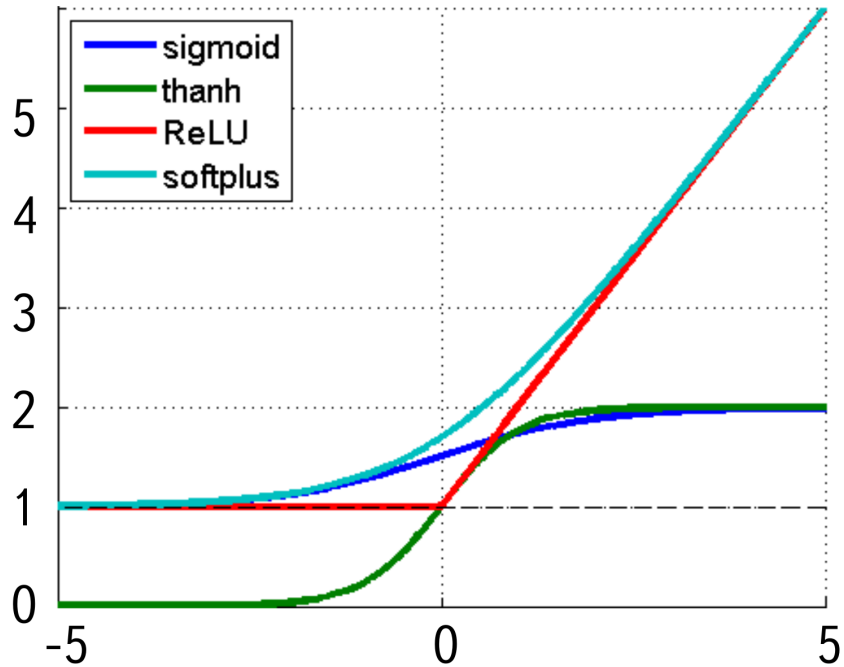
Types of nonlinearity

$$f(a) = \frac{1}{1 + e^{a}}$$

$$f(a) = \tanh(a)$$

$$f(a) = \max(0, a)$$

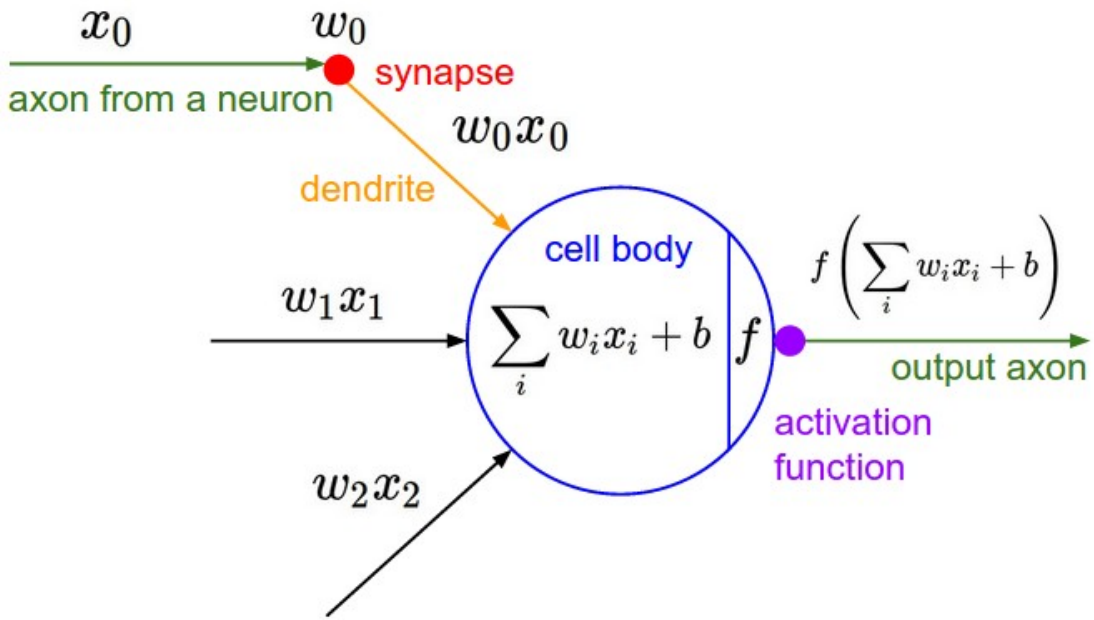
$$f(a) = \log(1 + e^{a})$$



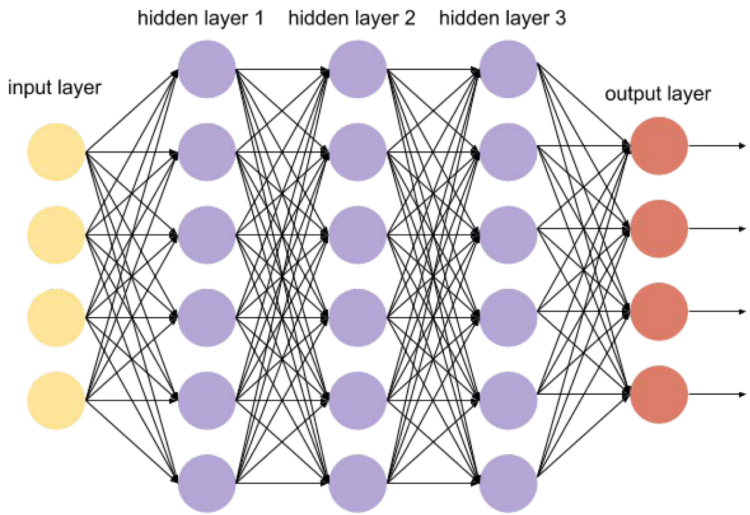
Wei Di, https://imiloainf.wordpress.com/2013/11/06/rectifier-nonlinearities/

19

Neuron


x_0
w_0
axon from a neuron
synapse
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation function
$w_2 x_2$

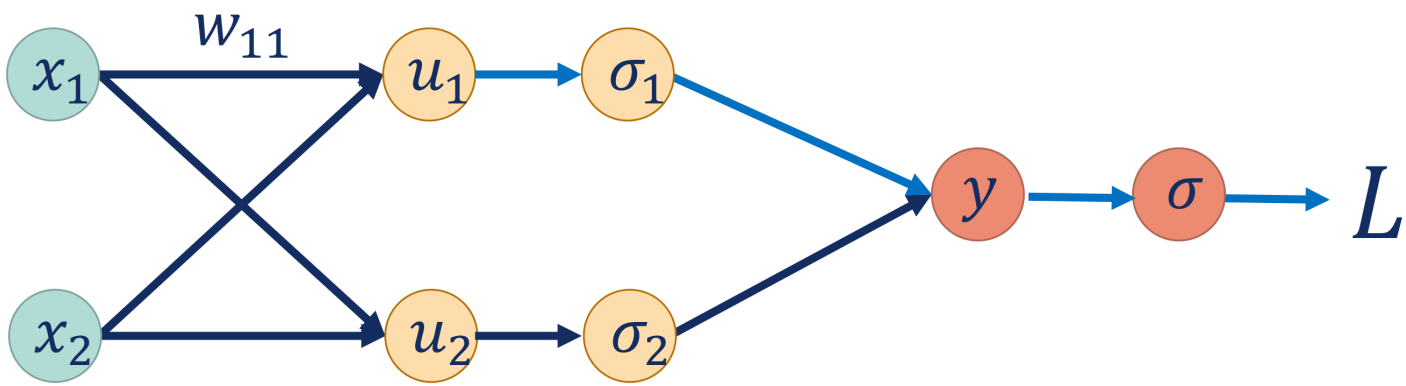
Neural network



$$h_t = f(Wh_{t-1} + b)$$

$$a(x) = f_3(W_3 f_2(W_2 f_1(W_1 f_0(W_0 x + b_0) + b_1) + b_2) + b_3)$$

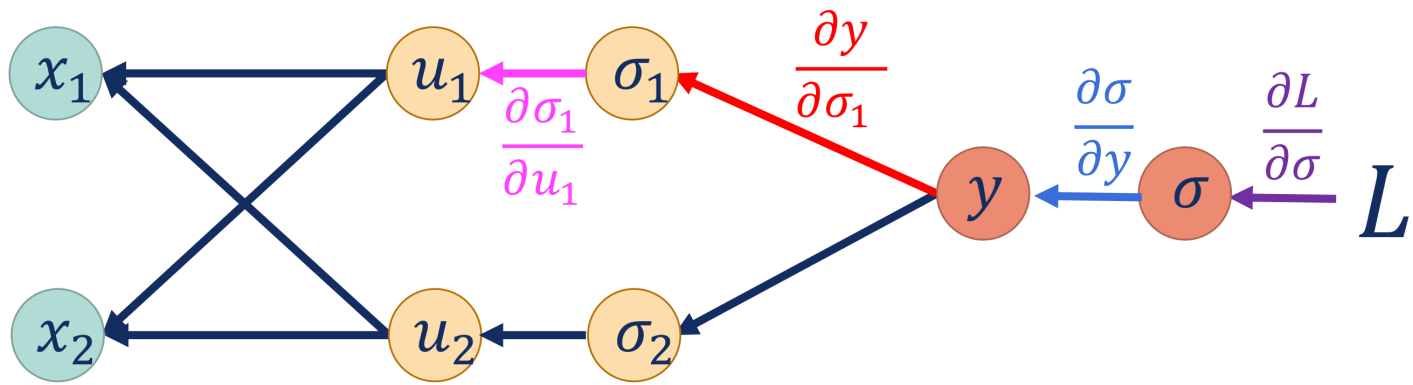
The backpropagation algorithm



$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial u_1}\frac{\partial u_1}{\partial w_{11}} = \frac{\partial L}{\partial u_1} x_1$$

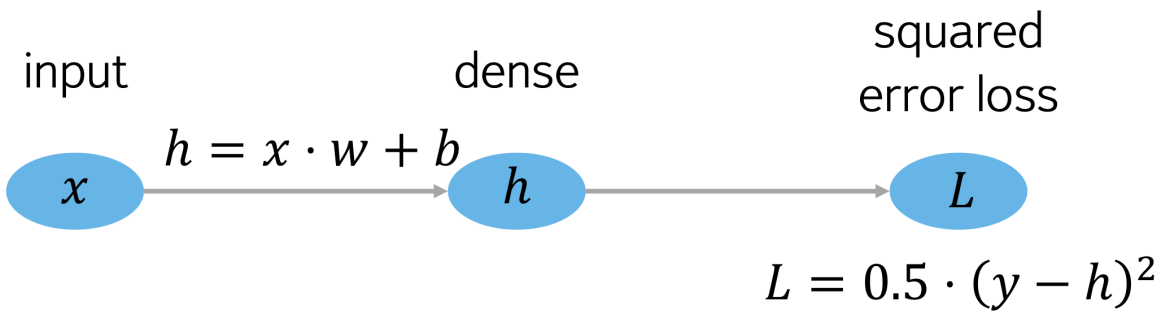
need to know gradients at each graph's node!



$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial y}\frac{\partial y}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial u_1} \leftarrow \text{chain rule}$$

Backprop in a nutshell: using the chain rule
propagate loss derivatives back going over each node step by step
starting from the NN's end

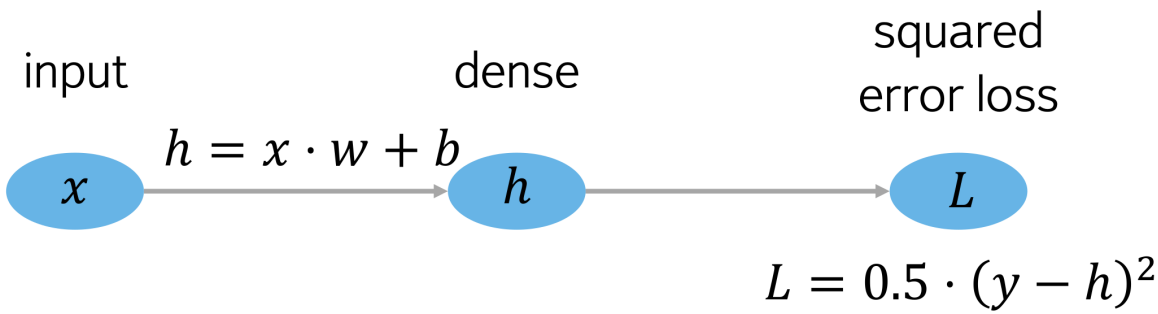
The simplest NN ever



- Parameters:
  - Weight w and bias b
- Input: x
- Target: y

Also known as
least squares linear regression

The simplest NN ever



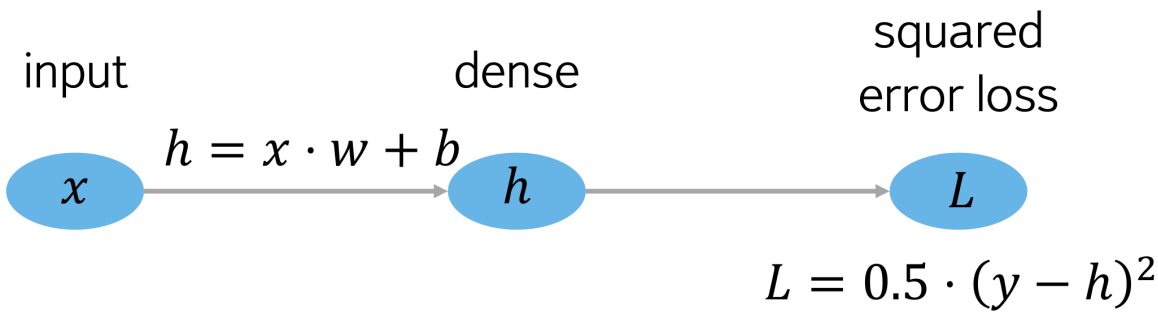
- Parameters:
  - Weight w and bias b
- Input: x
- Target: y

L is just a function of parameters, features and target:

$$L = f\big(y, g(x, w, b)\big)$$

24

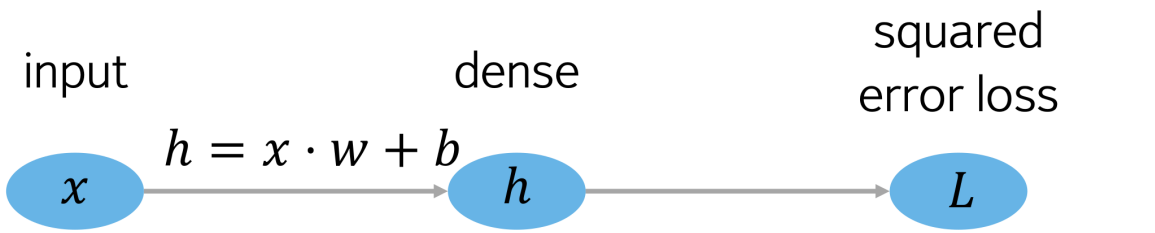
The simplest NN ever



- Gradient?
- $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b}$

25

The simplest NN ever

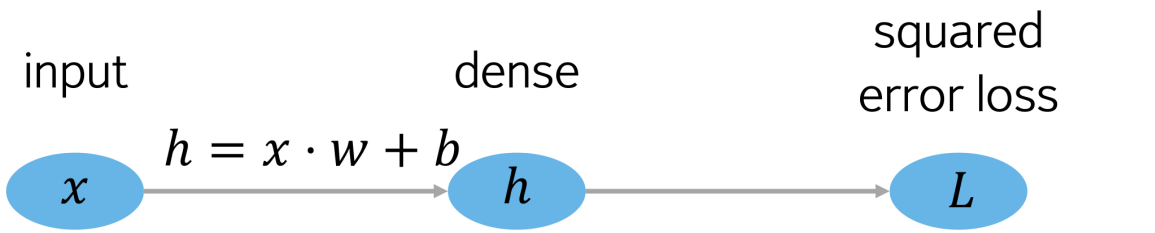


$$L = 0.5 \cdot (y - h)^2$$

- Let's fit
  - $y = 3, x = 1$
- Initial
  - $w = 0.1, b = 1$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b

Forward pass

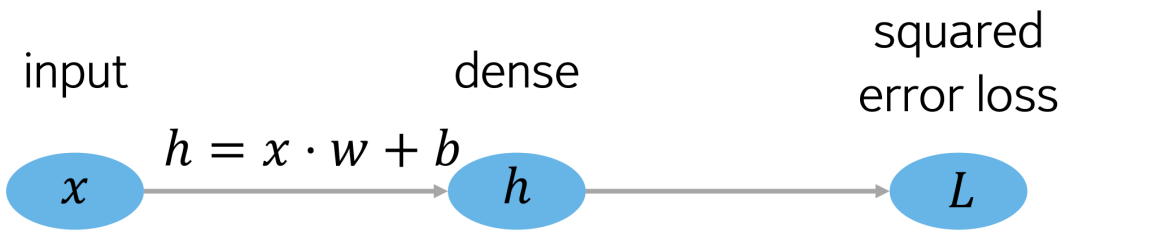


$$L = 0.5 \cdot (y - h)^2$$

- Let's fit
  - $y = 3, x = 1$
- Initial
  - $w = 0.1, b = 1$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80					

Backward pass

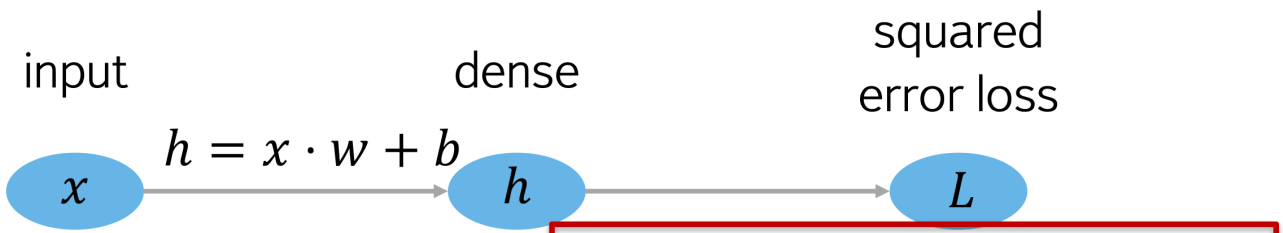


- Let's fit
  - $y = 3, x = 1$
- Initial
  - $w = 0.1, b = 1$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9				

28

Backward pass



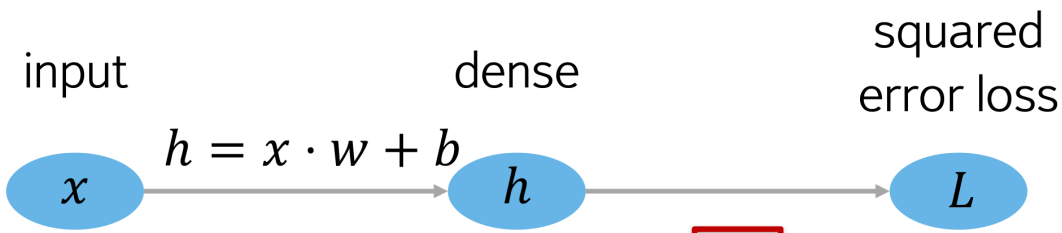
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial w} \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b}$$

- Let's fit
  - $y = 3, x = 1$
- Initial
  - $w = 0.1, b = 1$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9				

29

Backward pass



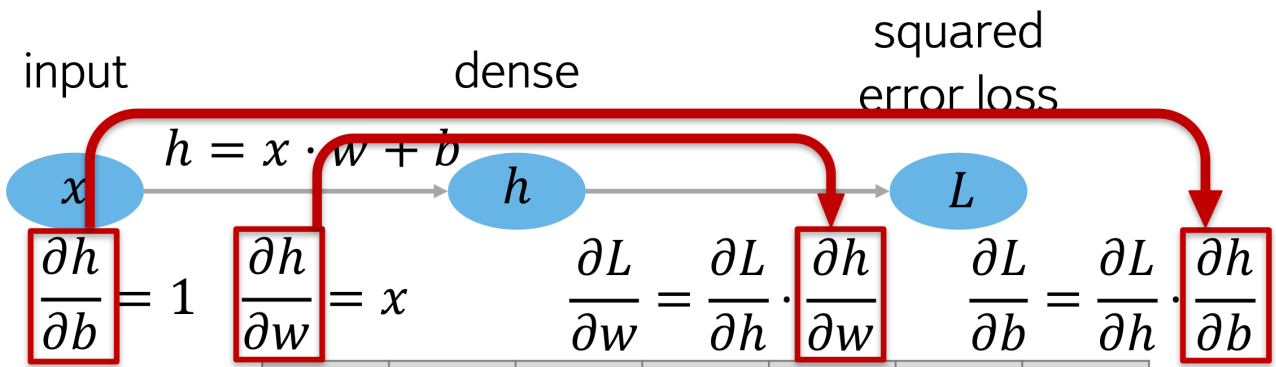
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial w} \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b}$$

- Let's fit
  - $y = 3, x = 1$
- Initial
  - $w = 0.1, b = 1$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9				

30

Backward pass



- Let's fit
  - $y = 3, x = 1$
- Initial
  - $w = 0.1, b = 1$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9				

31

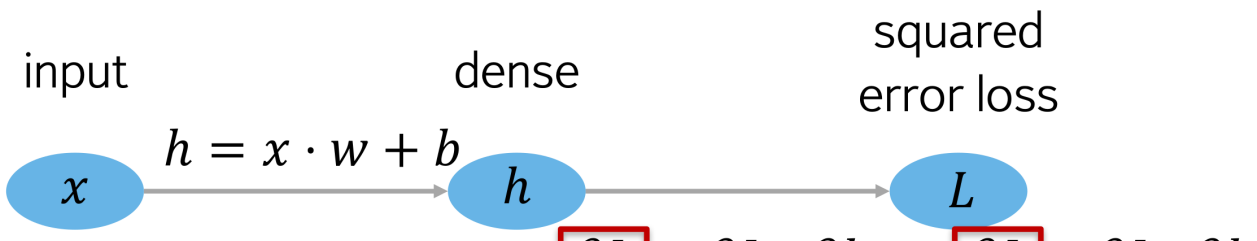
Backward pass

$$\frac{\partial h}{\partial b} = 1 \quad \frac{\partial h}{\partial w} = x \quad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial w} \quad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b}$$

- Let's fit
  - $y = 3, x = 1$
- Initial
  - $w = 0.1, b = 1$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9	-1.9	-1.9		

Backward pass



$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial w} \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b}$$

- Update parameters:

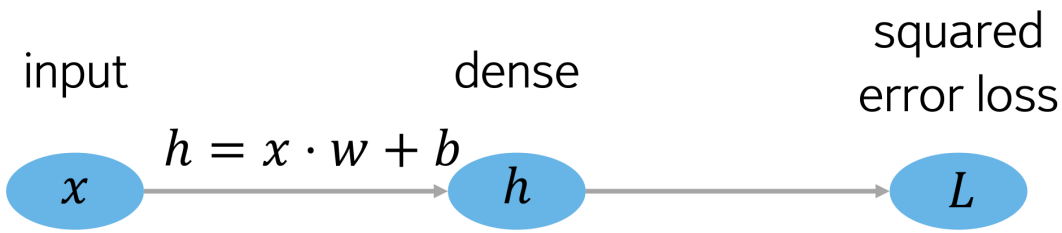
$$w \mathrel{-}= \eta \frac{\partial L}{\partial w}$$

$$b \mathrel{-}= \eta \frac{\partial L}{\partial b}$$

$$\eta = 0.2$$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9	-1.9	-1.9		

Backward pass



$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial w} \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b}$$

- Update parameters:

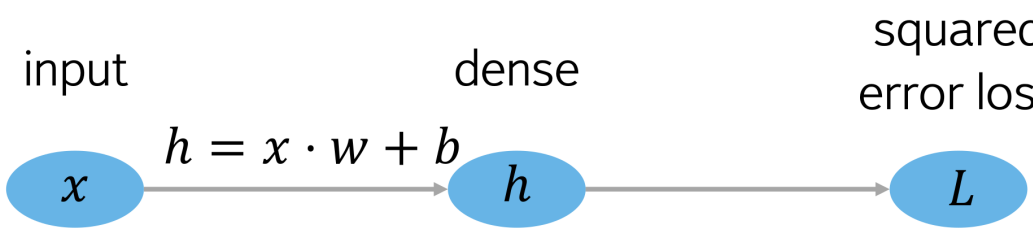
$$w \mathrel{-}= \eta \frac{\partial L}{\partial w}$$

$$b \mathrel{-}= \eta \frac{\partial L}{\partial b}$$

$$\eta = 0.2$$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9	-1.9	-1.9	0.48	1.38

After a few more updates...



$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial w} \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b}$$

- Update parameters:

$$w \mathrel{-}= \eta \frac{\partial L}{\partial w}$$

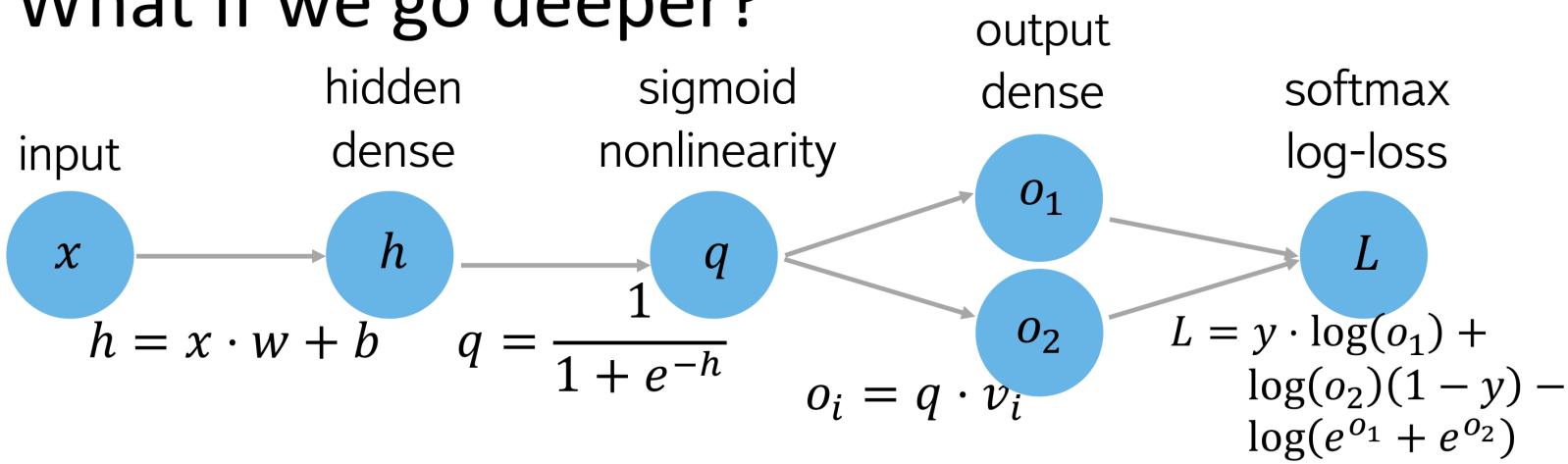
$$b \mathrel{-}= \eta \frac{\partial L}{\partial b}$$

$$\eta = 0.2$$

h	L	$\frac{\partial L}{\partial h}$	$\frac{\partial L}{\partial w}$	$\frac{\partial L}{\partial b}$	w	b
1.1	1.80	-1.9	-1.9	-1.9	0.48	1.38
1.86	0.65	-1.14	-1.14	-1.14	0.71	1.61
2.32	0.23	-0.68	-0.68	-0.68	0.84	1.75

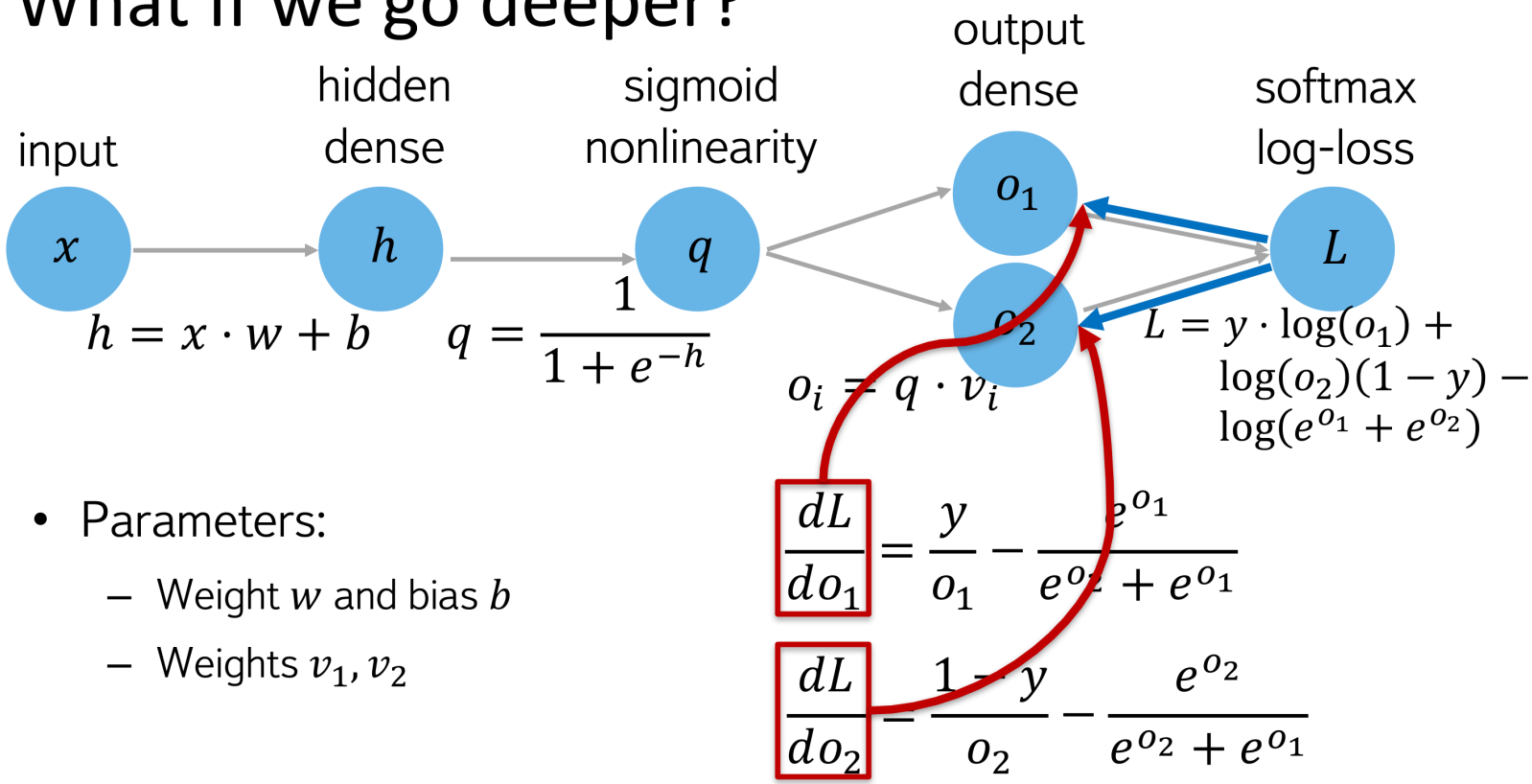
35

What if we go deeper?



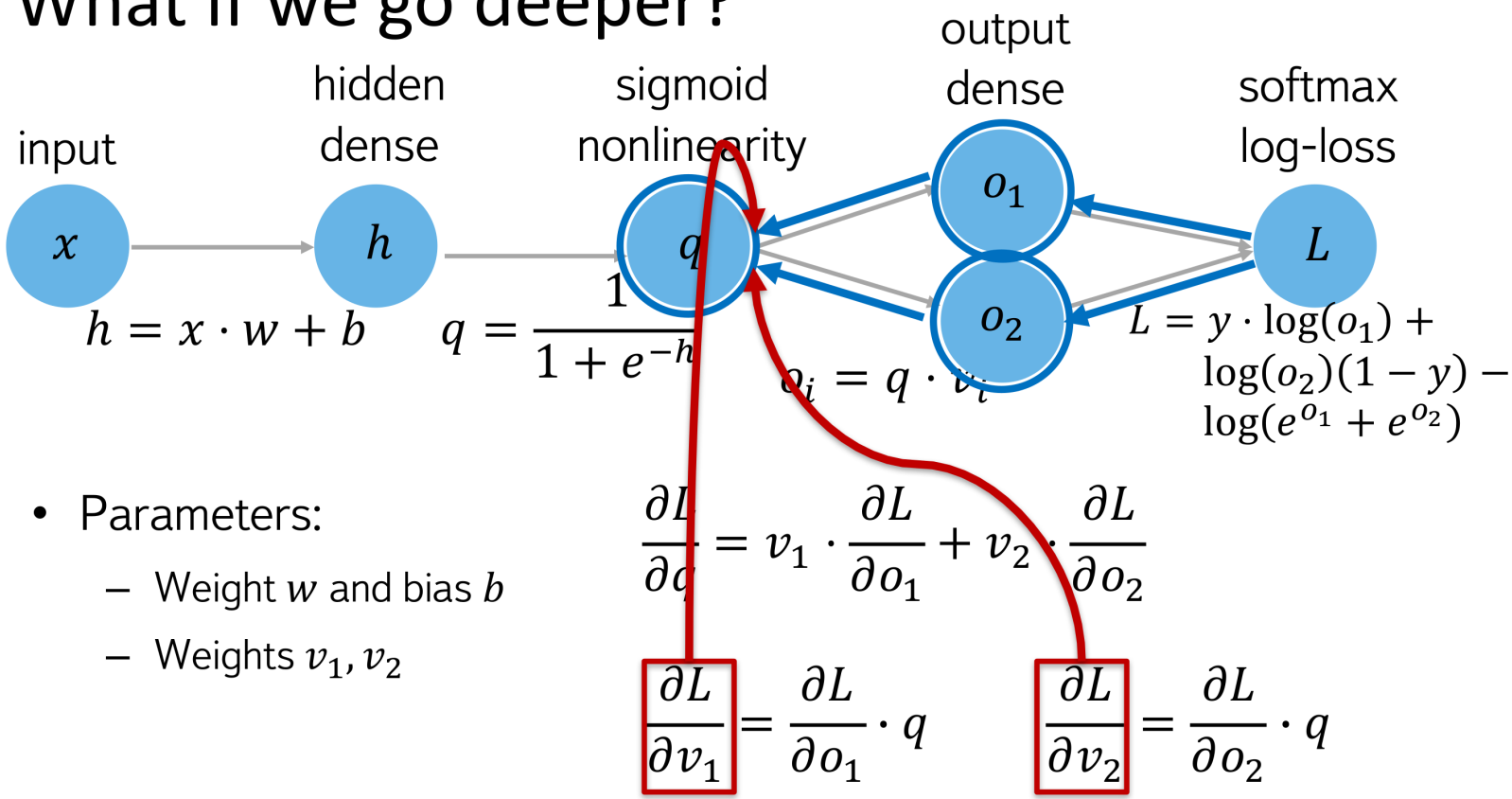
- Parameters:
  - Weight w and bias b
  - Weights v_1, v_2

What if we go deeper?



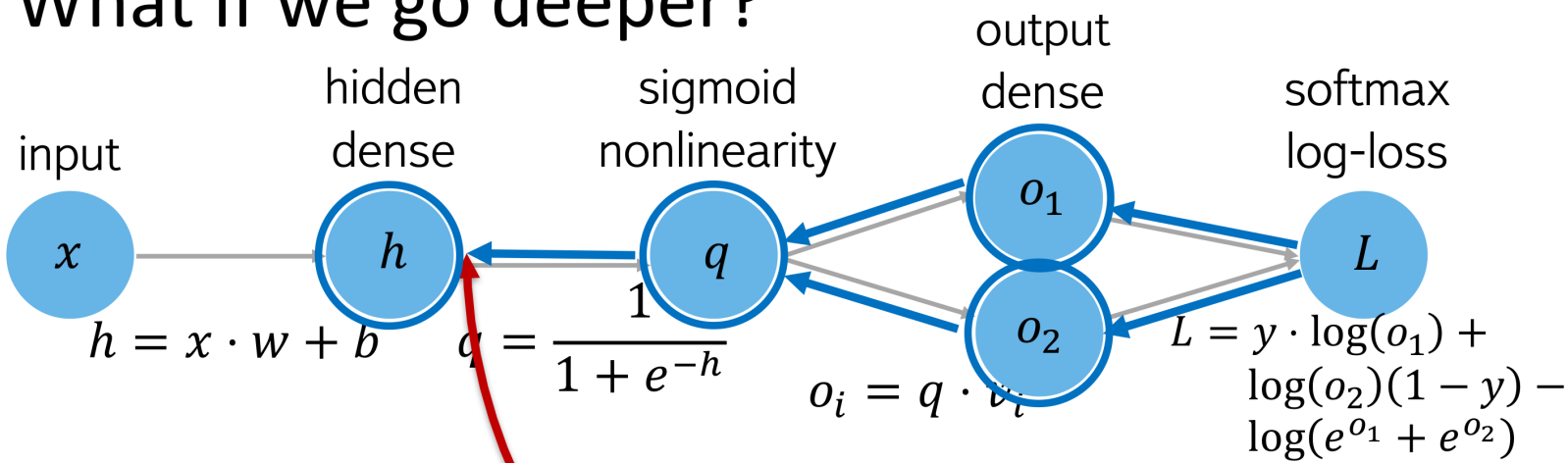
- Parameters:
  - Weight w and bias b
  - Weights v_1, v_2

What if we go deeper?



- Parameters:
  - Weight w and bias b
  - Weights v_1, v_2

What if we go deeper?



- Parameters:
  - Weight w and bias b
  - Weights v_1, v_2

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial q} \frac{e^{-q}}{(1 + e^{-q})^2}$$

39

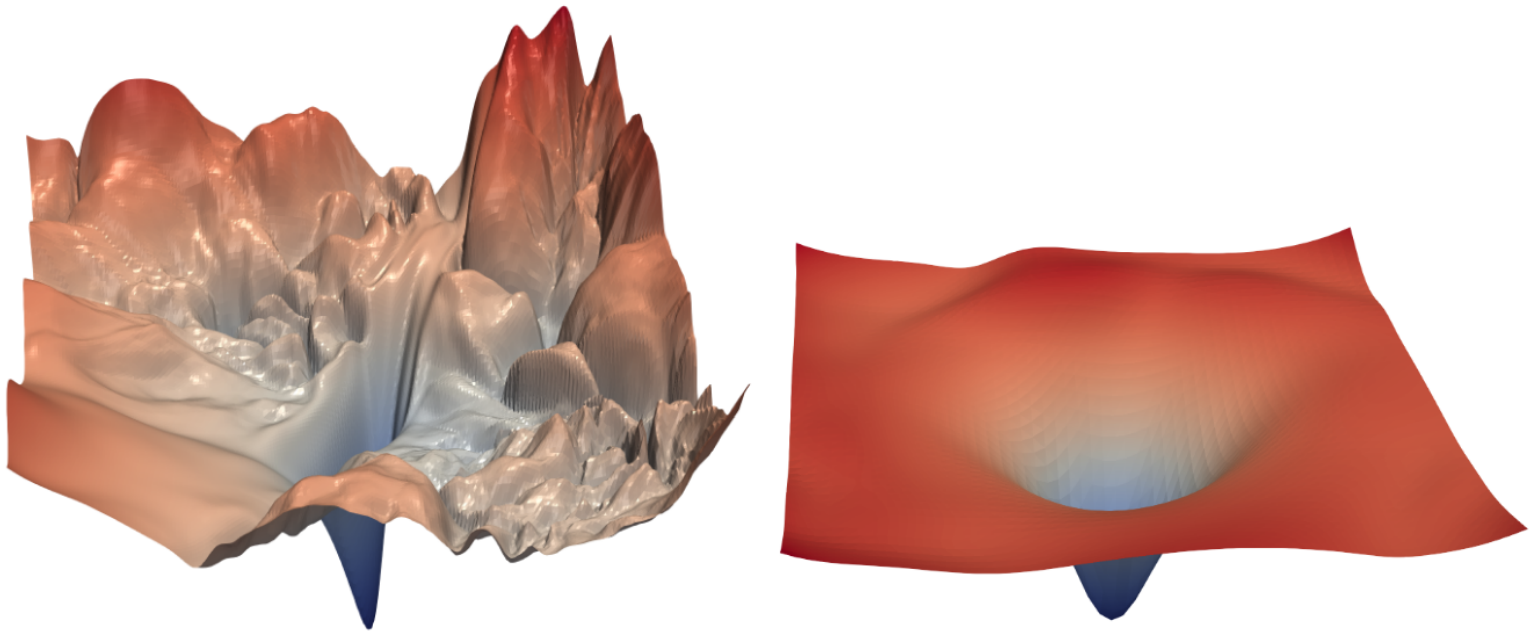
Backpropagation: the algorithm

- Chain rule can be evaluated numerically!
- Compute the network output and the loss value
- Compute "dLoss" /"dActivation_of_output_layer"
- For each layer, starting from the last:
  - Compute “dActivation” / “dLayer_parameters”, “dActivation” / “dLayer_input”
  - Multiply it by “dLoss” / “dActivation”, get “dLoss” /...
- Make optimization step for the parameters

40

from paper [link]

NN landscape



(a) without skip connections

(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Practical issues

- bad convergence
  - lots of local minima, can get stuck there
  - overfitting as a piece of cake
- gradient flow problems
  - explosion
  - vanishing

Practical issues

- bad convergence
  - lots of local minima, can get stuck there
  - overfitting as a piece of cake
- gradient flow problems
  - explosion
  - vanishing

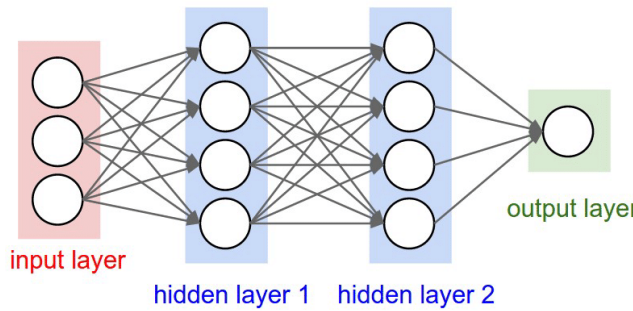
$\Rightarrow$ need addit. heuristics

awesome methods comparison
on Higgs Kaggle dataset [link]

Bells & Whistles

you should know about

- Optimization
  - grad. descent modifications (e.g., SGD, Adam, Nesterov momentum)
  - learning rate schedules (e.g., LR decay)
  - early stopping
- Architecture
  - activations (e.g., SELU, Leaky RELU, Swish)
  - dropout
  - batch normalization
  - weight initialization (why 0-init is bad?)
  - penalty loss terms (L1, L2)



TF vs PyTorch comparison [link]

Frameworks

you should know about

- TensorFlow
- Keras (TF & Theano back-end)
- PyTorch
- ? Theano

Baseline

you should start from

- normalize data
- train in batches
- RELU activation
- He et al. weight init
- Adam optimizer
- hyperparams with random search (not grid) or go bayesian
- try BatchNorm
- try Dropout (p=0.5) and L1/L2 regularization

Neural Nets

pros & cons

+ damn very good performance
+ general to adapt to various problem domains
+ modular and easy to construct
+ automatic feature engineering
+ breaks No Free Lunch theorem (almost) [1],[2]
- overfits
- requires careful baby-sitting
- you will probably need to get GPU
- "black box" (no)

Here goes fancy

Convolutional NN

Convolutional NN

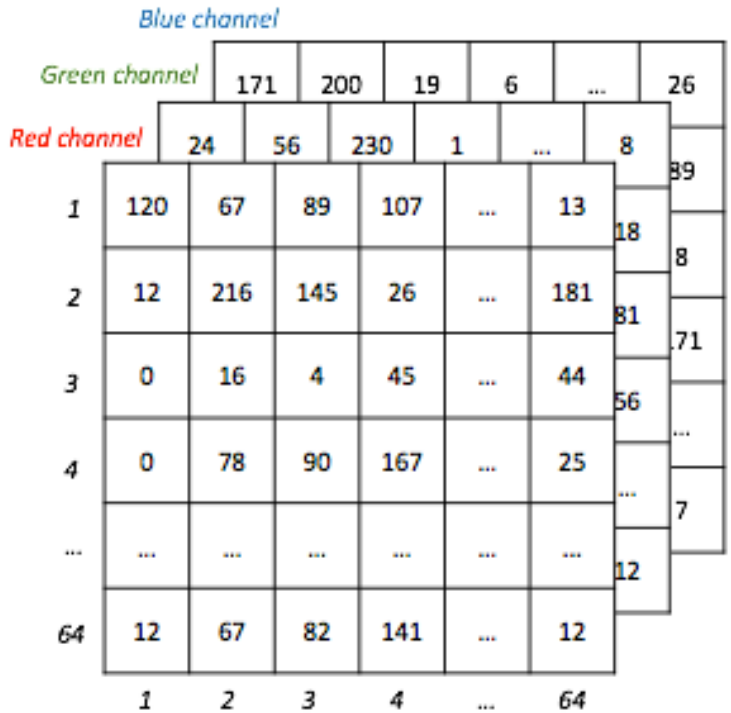
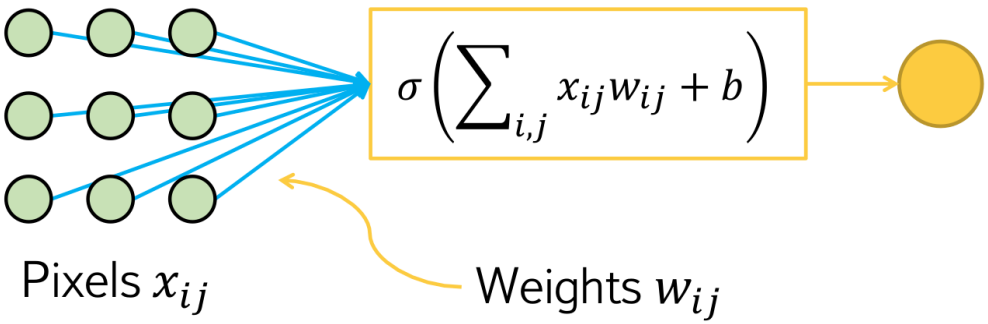
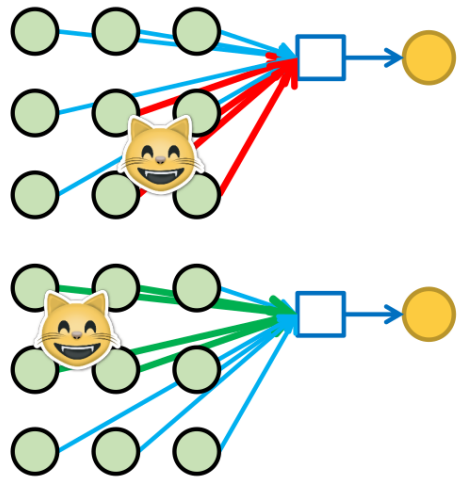


Image array: [64 x 64 x 3]

Convolutional NN

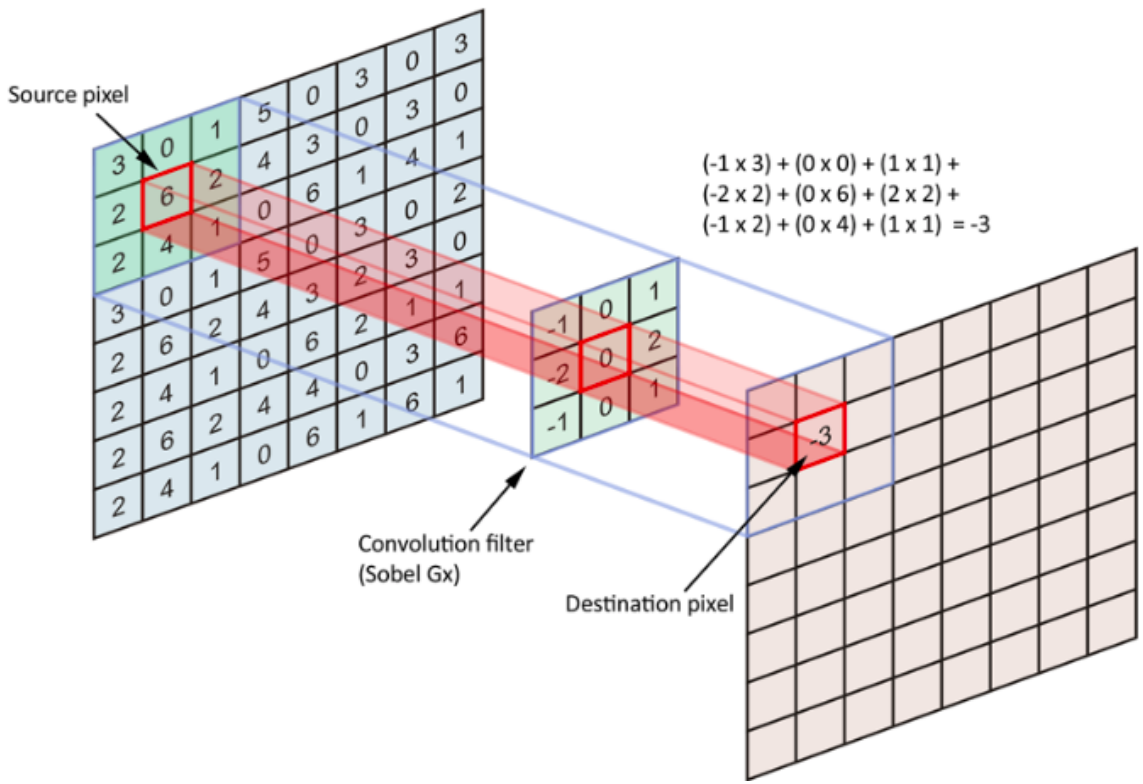
why not Dense?





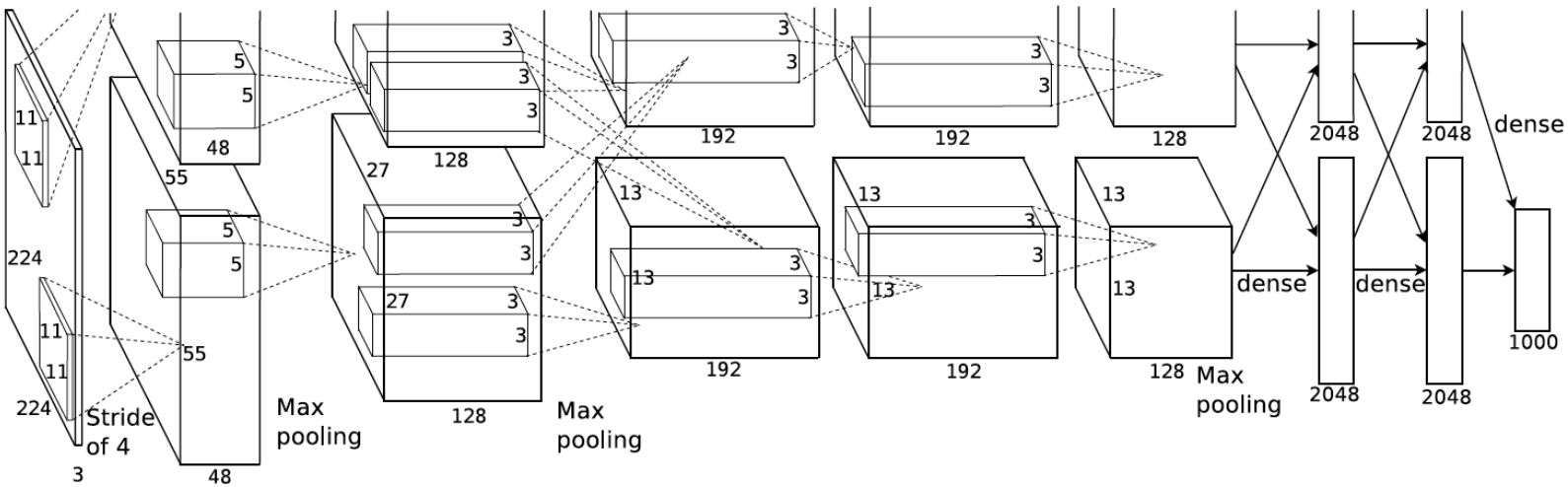
Convolutional NN

conv. layer



Convolutional NN

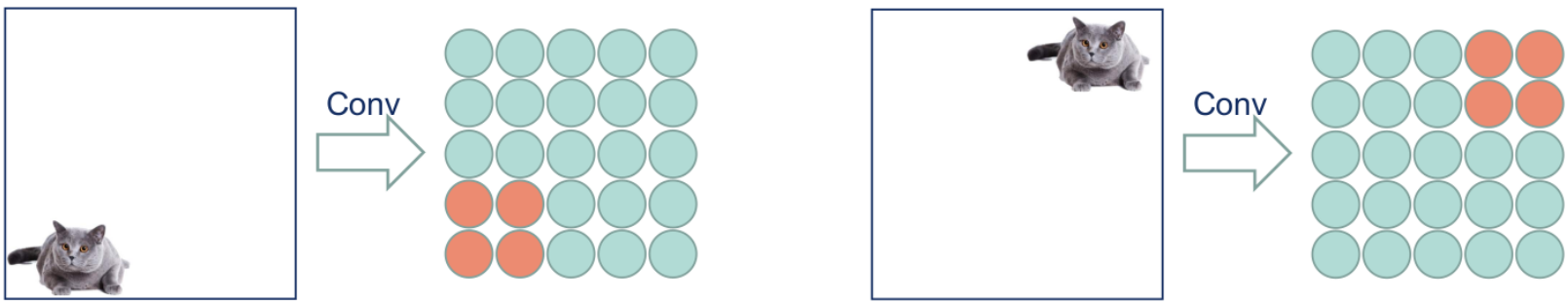
AlexNet



Convolutional NN

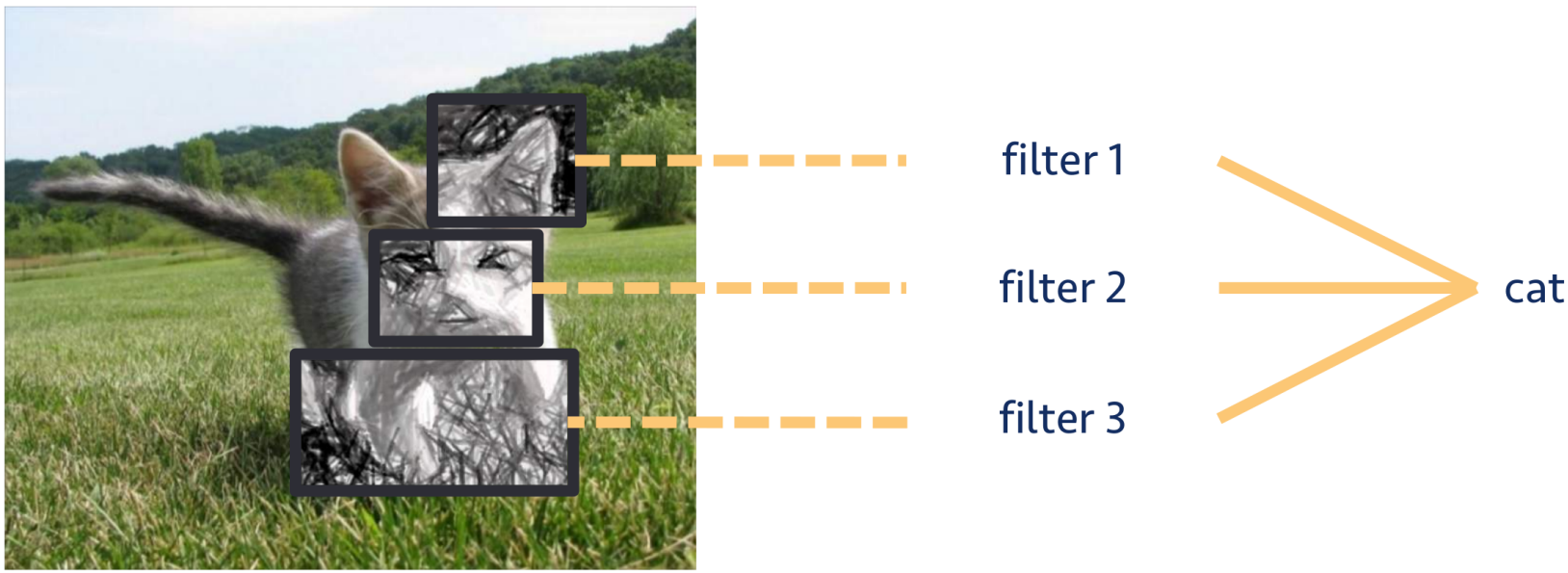
equivariance

$$f\big(T(x)\big) = T(f(x))$$



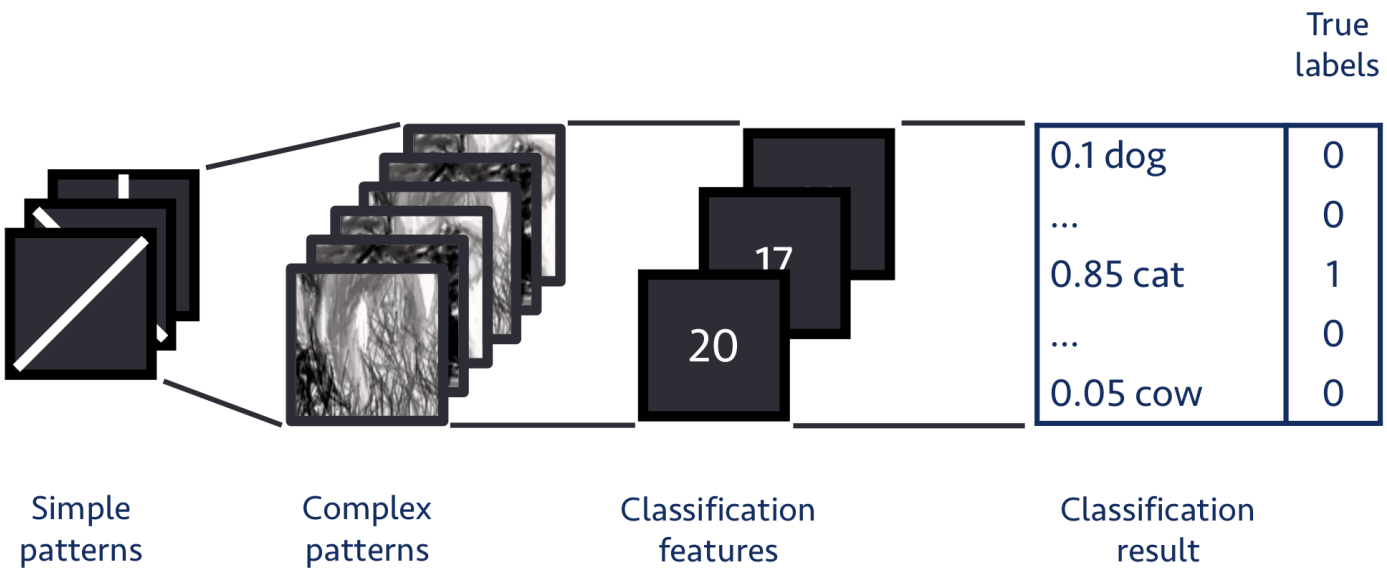
Convolutional NN

patterns



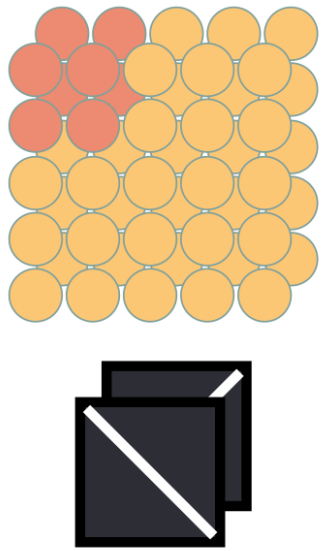
Convolutional NN

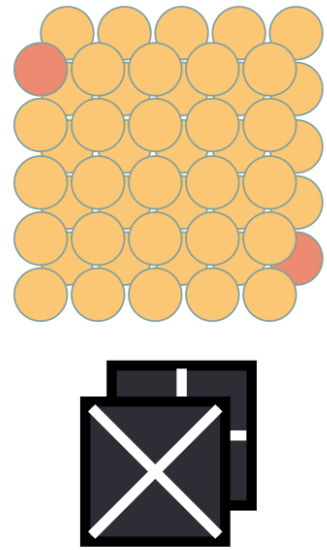
patterns



Convolutional NN

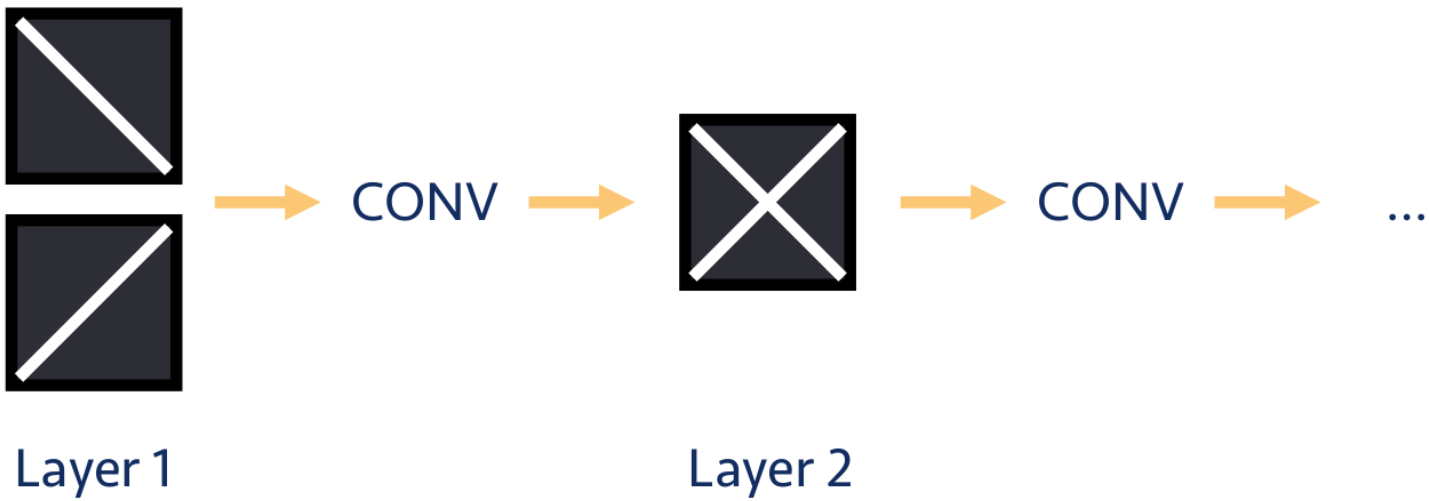
patterns





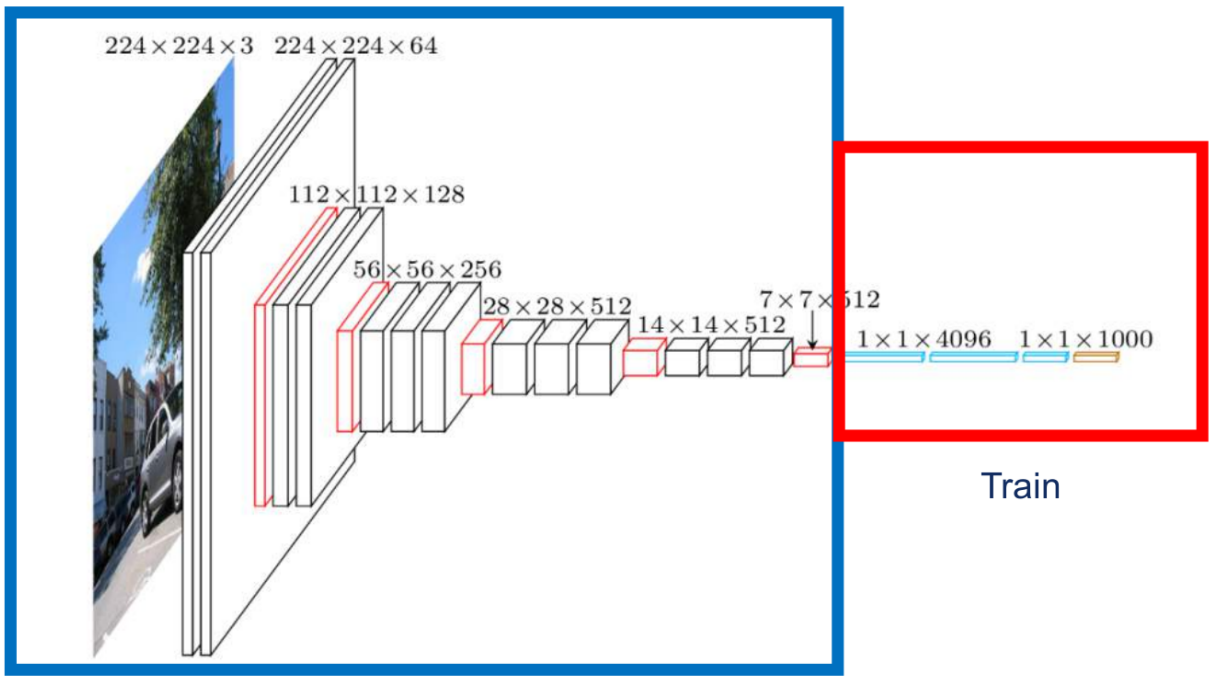
Convolutional NN

patterns



Convolutional NN

modularity



Recurrent NN

motivation

- sequence processing
  - sentence
  - song
  - jet
  - market stock
- problems
  - translation
  - speech recognition
  - jet grooming
  - price prediction
- why not Dense NN?

Recurrent NN

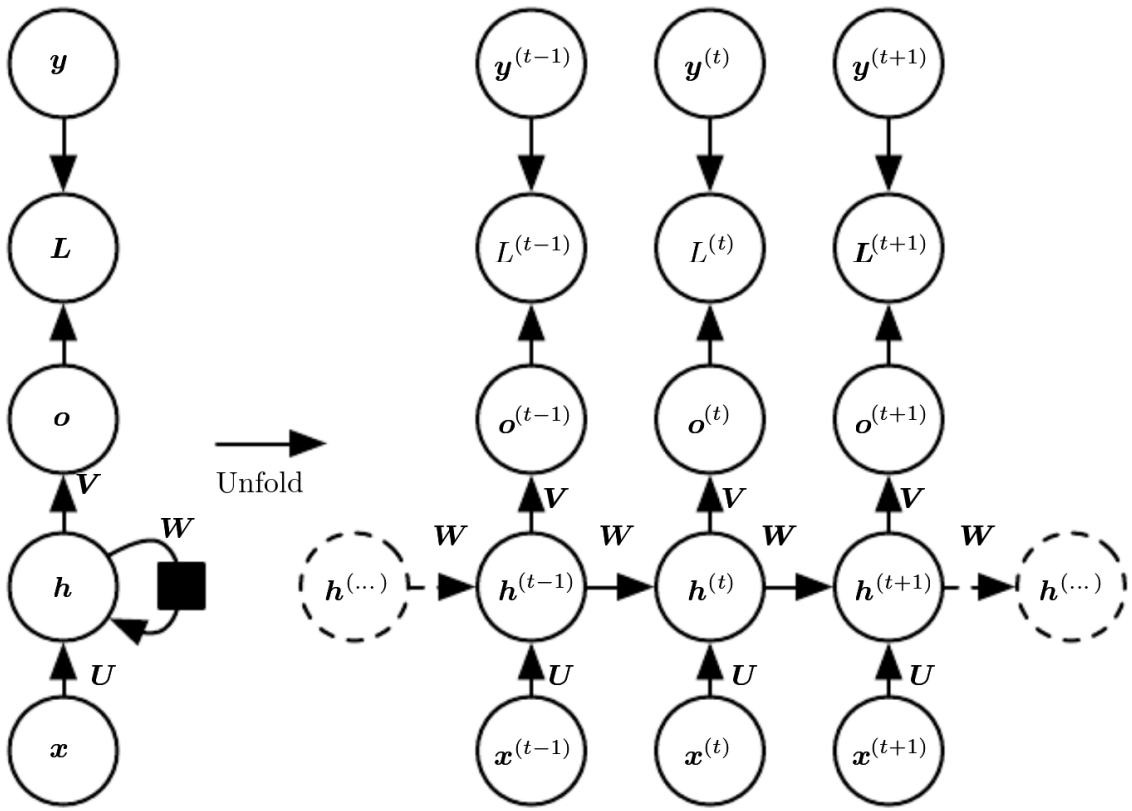
motivation

- sequence processing
  - sentence
  - song
  - jet
  - market stock
- problems
  - translation
  - speech recognition
  - jet grooming
  - price prediction
- why not Dense NN?
  - no time invariance
  - fixed sequence size

from Deep Learning book [link]

Recurrent NN

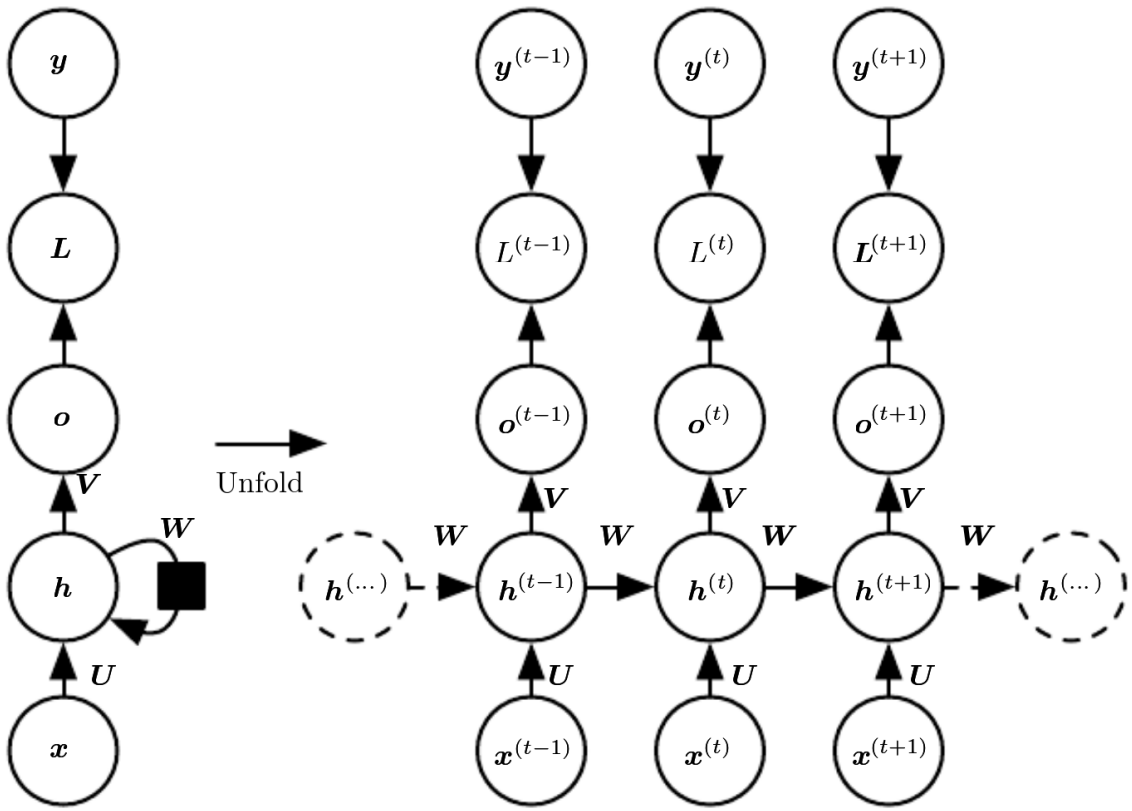
vanilla edition



from Deep Learning book [link]

Recurrent NN

vanilla edition

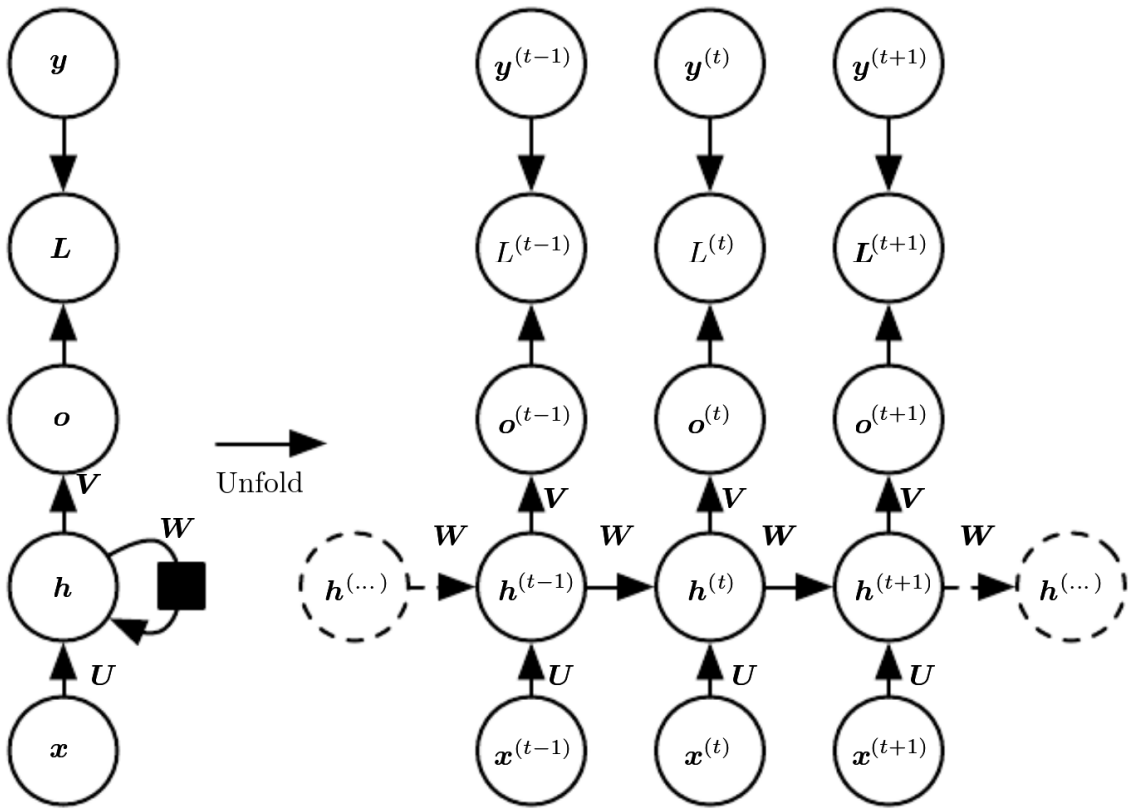


what is x?

from Deep Learning book [link]

Recurrent NN

vanilla edition



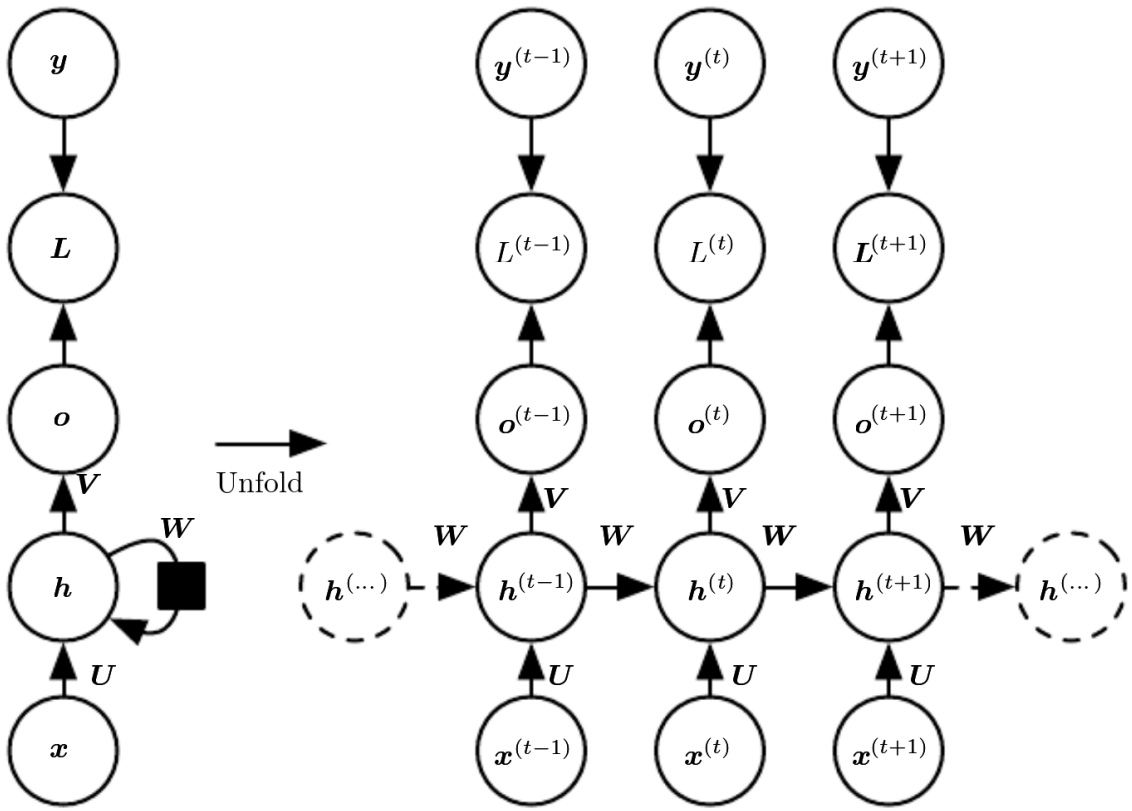
what is x? embeddings

from Deep Learning book [link]

Recurrent NN

vanilla edition

where are activations?



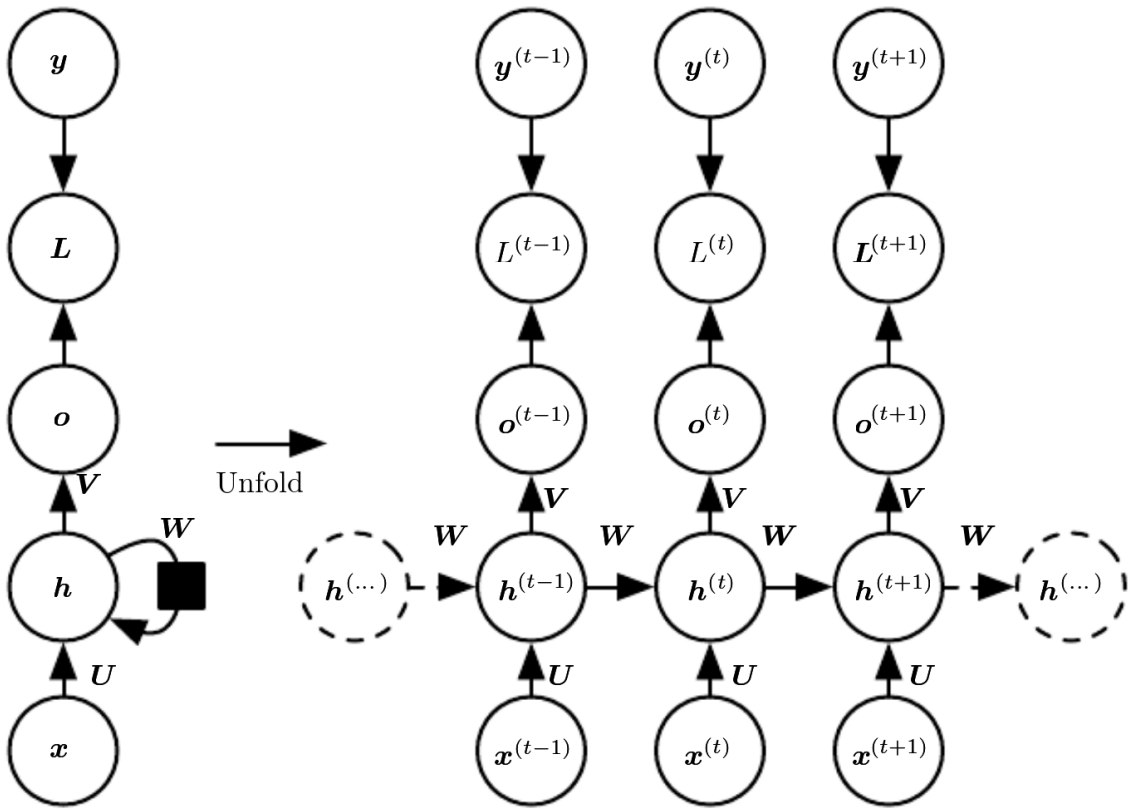
what is x? embeddings

from Deep Learning book [link]

Recurrent NN

vanilla edition

where are activations?



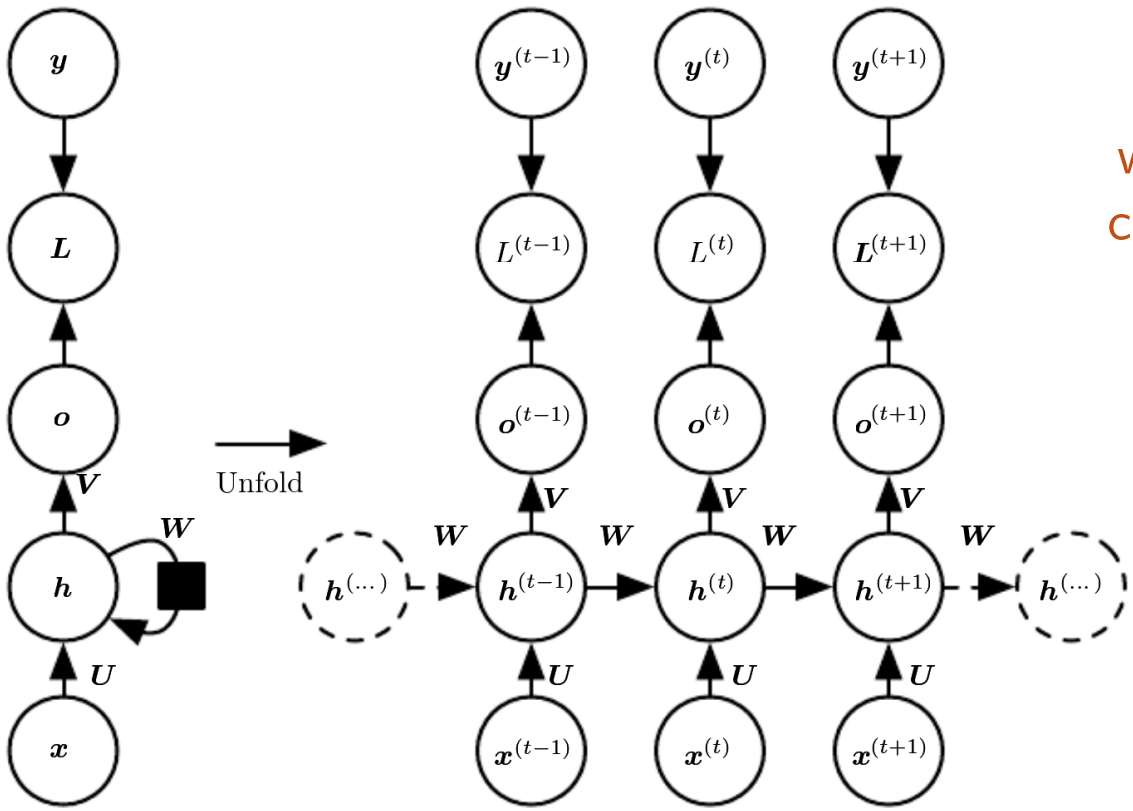
what is x? embeddings

from Deep Learning book [link]

Recurrent NN

vanilla edition

where are activations?



where this one can be useful?

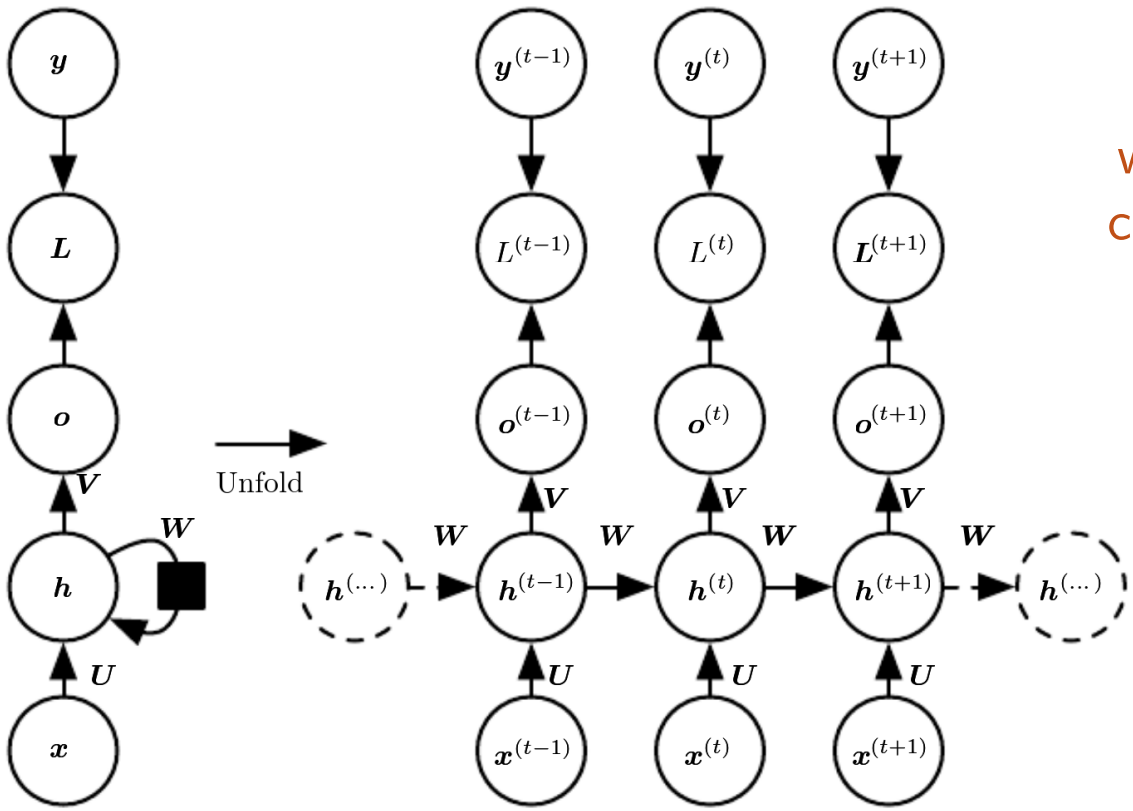
what is x? embeddings

from Deep Learning book [link]

Recurrent NN

vanilla edition

where are activations?



where this one can be useful?

seq labeling

what is x?

embeddings

Summary

- Ensembles
  - Motivation
  - Bootstrap & Bagging $\rightarrow$ Random Forest
  - Boosting $\rightarrow$ BDT
- Neural Nets
  - Motivation
  - Building
  - Training
  - Special tricks
- $*$ Bonus
  - Convolutional NN
  - Recurrent NN

next lecture

- Thrilling applications in HEP

Крафтер Колян
28 июн 2018

- Здравствуйте, Николай. Поздравляем с успешным выступлением в контесте "Обучение скорости обучения". Нам хотелось бы обсудить ваше решение.
- Хорошо, без проблем.

Через 20 минут в офисе Яндекса (Отдел исследований)

- Итак, по заданию надо было предсказать learning rate для разных выборок, чтобы алгоритм CatBoost давал максимальную точность. Как вы это делали?
- Ну, там данных немного, всего 80 примеров, я закинул их в Excel и решил посмотреть, а что собственно есть.
- Так, что дальше?
- Оставил только примеры с большим N, заметил, что у меня в обучающей выборке есть почти то же самое, что и для тестовой. Для них ответ сразу выписал.
- Вы нашли зависимость от N?
- Нет, этого же не было в задании.
- Вы использовали gradient boost или регрессию?
- Если честно, я ни слова не понимаю.
- Вы использовали какие-нибудь алгоритмы, математику?
- Нет, я просто искал для каждого теста похожий пример в обучающей выборке и давал такой же ответ.
- Вы хотя бы калькулятором пользовались?
- Нет.
- И у Вас лучшее решение?
- Да, и у него точность выше на 0.15%, чем у вашего решения, причём у всех остальных максимум на 0.04%.
- ...

[link]