Machine Learning, Computer Vision & Open Source

Andreas Müller
Columbia University, scikit-learn

Alfred P. Sloan
FOUNDATION

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

What is machine learning?

Types of machine learning:

- supervised
- unsupervised
- reinforcement

Supervised Learning

$$(x_i, y_i) \propto p(x, y) \quad \text{i.i.d.}$$

$$x_i \in \mathbb{R}^n$$

$$y_i \in \mathbb{R}$$

$$f(x_i) \approx y_i$$

Examples of Supervised Learning

Classification and Regression

Classification:

- y discrete

Regression:

- y continuous

Generalization

Not only $f(x_i) \approx y_i$

Also for new data: $f(x) \approx y$

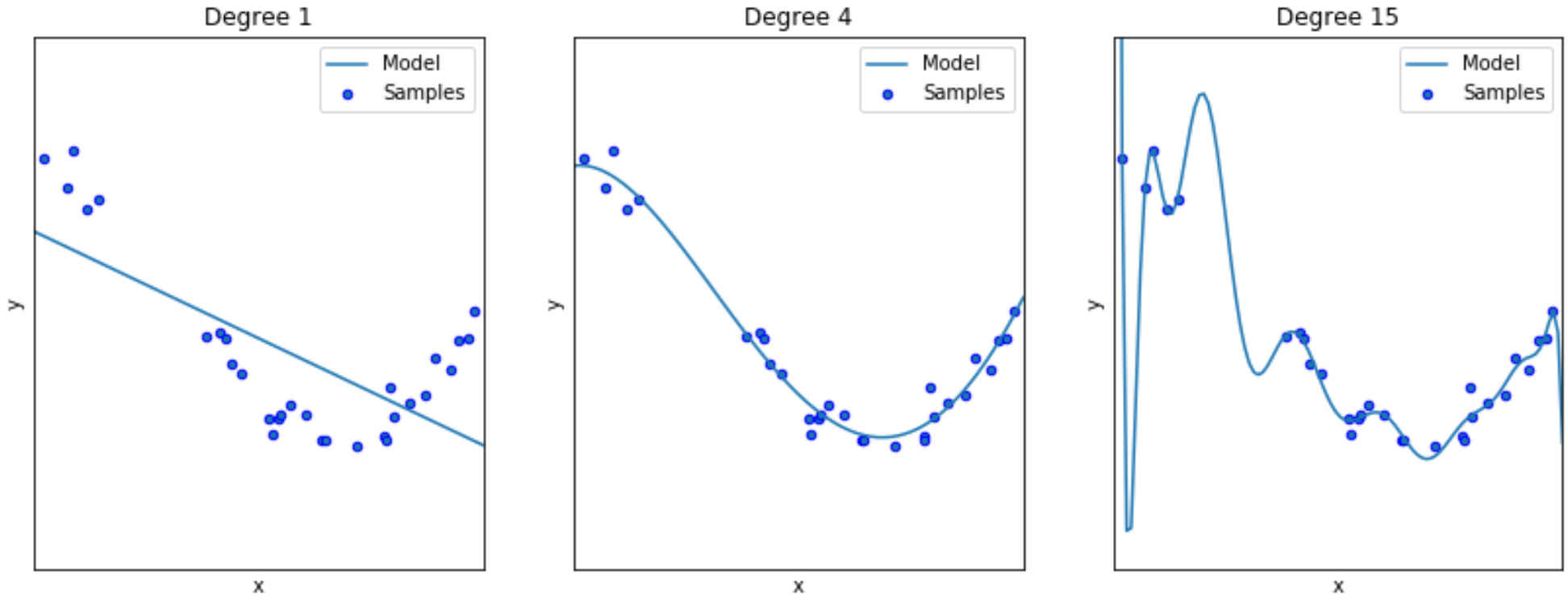
(regularized) Empirical Risk Minimization

Goal: $$\min_{f \in F} \mathbb{E}_{x,y} \ell(f(x), y)$$

Empirical Risk Minimization

Goal: $$\min_{f \in F} \mathbb{E}_{x,y} \ell(f(x), y)$$

solvable: $$\min_{f \in F} \sum_i \hat{\ell}(f(x_i), y_i)$$


Degree 1
Model
Samples
y
x
Degree 4
Model
Samples
y
x
Degree 15
Model
Samples
y
x

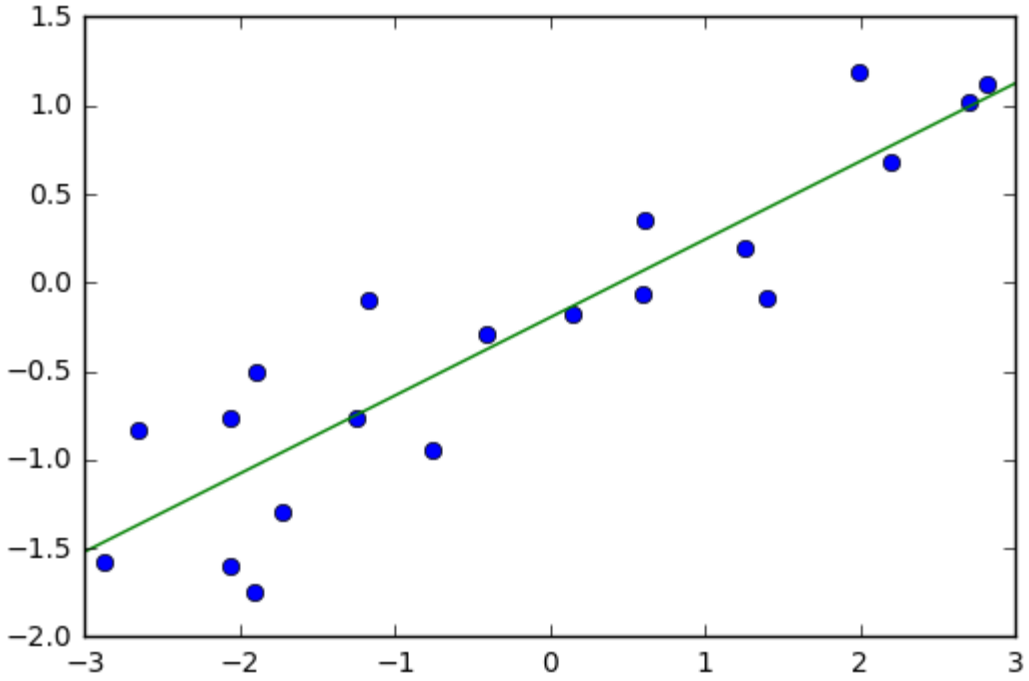
Regularized Empirical Risk Minimization

Goal: $$\min_{f \in F} \mathbb{E}_{x,y} \ell(f(x), y)$$

solvable: $$\min_{f \in F} \sum_i \hat{\ell}(f(x_i), y_i)$$

better: $$\min_{f \in F} \sum_i \hat{\ell}(f(x_i), y_i) + \lambda R(f)$$

Linear Regression as ERM



$$\min_{w \in \mathbb{R}^n} \sum_i ||w^T x_i - y_i||^2$$

Regularized Linear Regression: Ridge

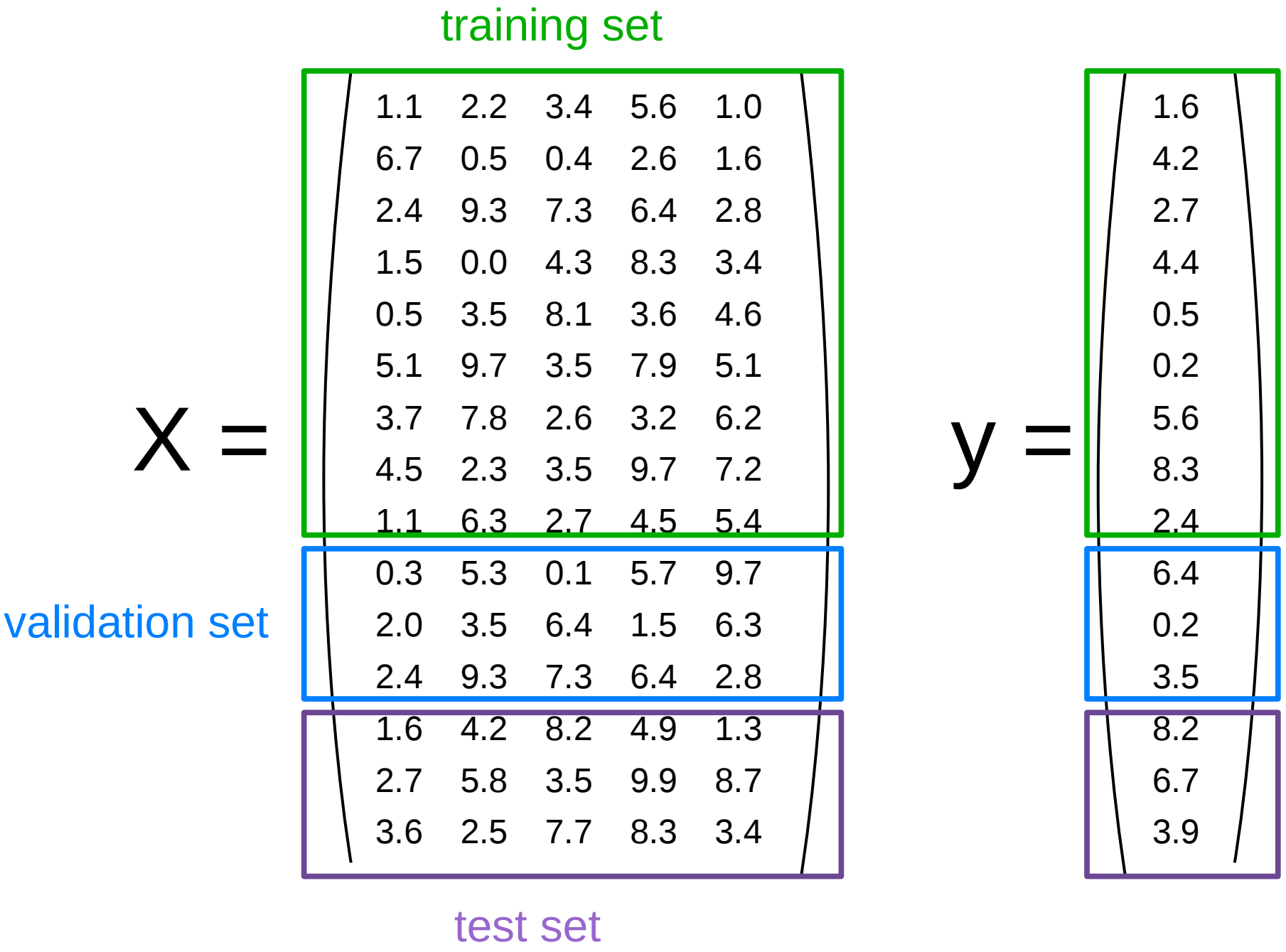
$$\min_{w \in \mathbb{R}^n} \sum_i ||w^T x_i - y_i||^2 + \lambda ||w||^2$$

training set

$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.2 \\ 4.5 & 2.3 & 3.5 & 9.7 & 7.2 \\ 1.1 & 6.3 & 2.7 & 4.5 & 5.4 \\ 0.3 & 5.3 & 0.1 & 5.7 & 9.7 \\ 2.0 & 3.5 & 6.4 & 1.5 & 6.3 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.6 & 4.2 & 8.2 & 4.9 & 1.3 \\ 2.7 & 5.8 & 3.5 & 9.9 & 8.7 \\ 3.6 & 2.5 & 7.7 & 8.3 & 3.4 \end{pmatrix}$$

$$y = \begin{pmatrix} 1.6 \\ 4.2 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 8.3 \\ 2.4 \\ 6.4 \\ 0.2 \\ 3.5 \\ 8.2 \\ 6.7 \\ 3.9 \end{pmatrix}$$

test set


training set
X =
1.1 2.2 3.4 5.6 1.0
6.7 0.5 0.4 2.6 1.6
2.4 9.3 7.3 6.4 2.8
1.5 0.0 4.3 8.3 3.4
0.5 3.5 8.1 3.6 4.6
5.1 9.7 3.5 7.9 5.1
3.7 7.8 2.6 3.2 6.2
4.5 2.3 3.5 9.7 7.2
1.1 6.3 2.7 4.5 5.4
validation set
0.3 5.3 0.1 5.7 9.7
2.0 3.5 6.4 1.5 6.3
2.4 9.3 7.3 6.4 2.8
test set
1.6 4.2 8.2 4.9 1.3
2.7 5.8 3.5 9.9 8.7
3.6 2.5 7.7 8.3 3.4
y =
1.6
4.2
2.7
4.4
0.5
0.2
5.6
8.3
2.4
6.4
0.2
3.5
8.2
6.7
3.9

Neural Networks

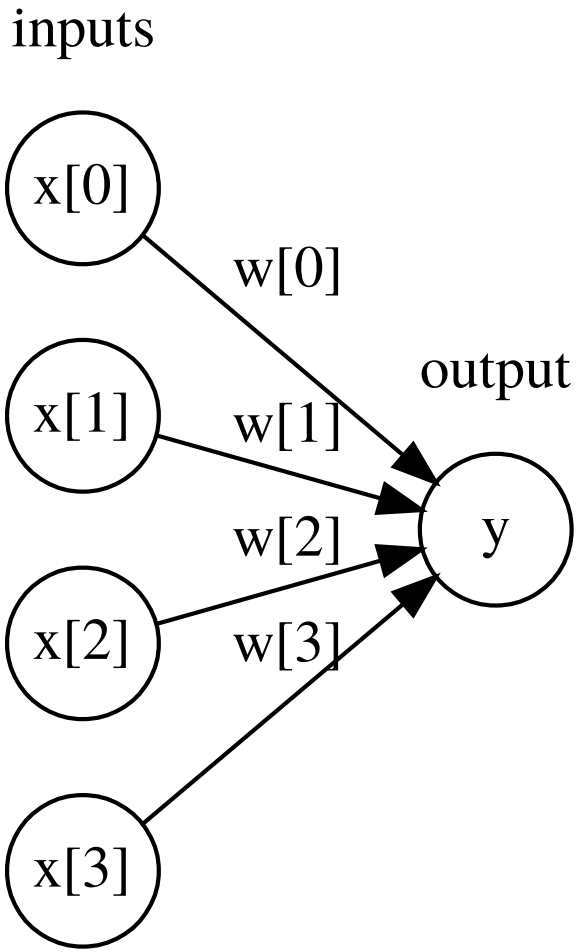
History

- Nearly everything we talk about today existed ~1990
- What changed?
  - More data
  - Faster computers (GPUs)
  - Some improvements in algorithms.

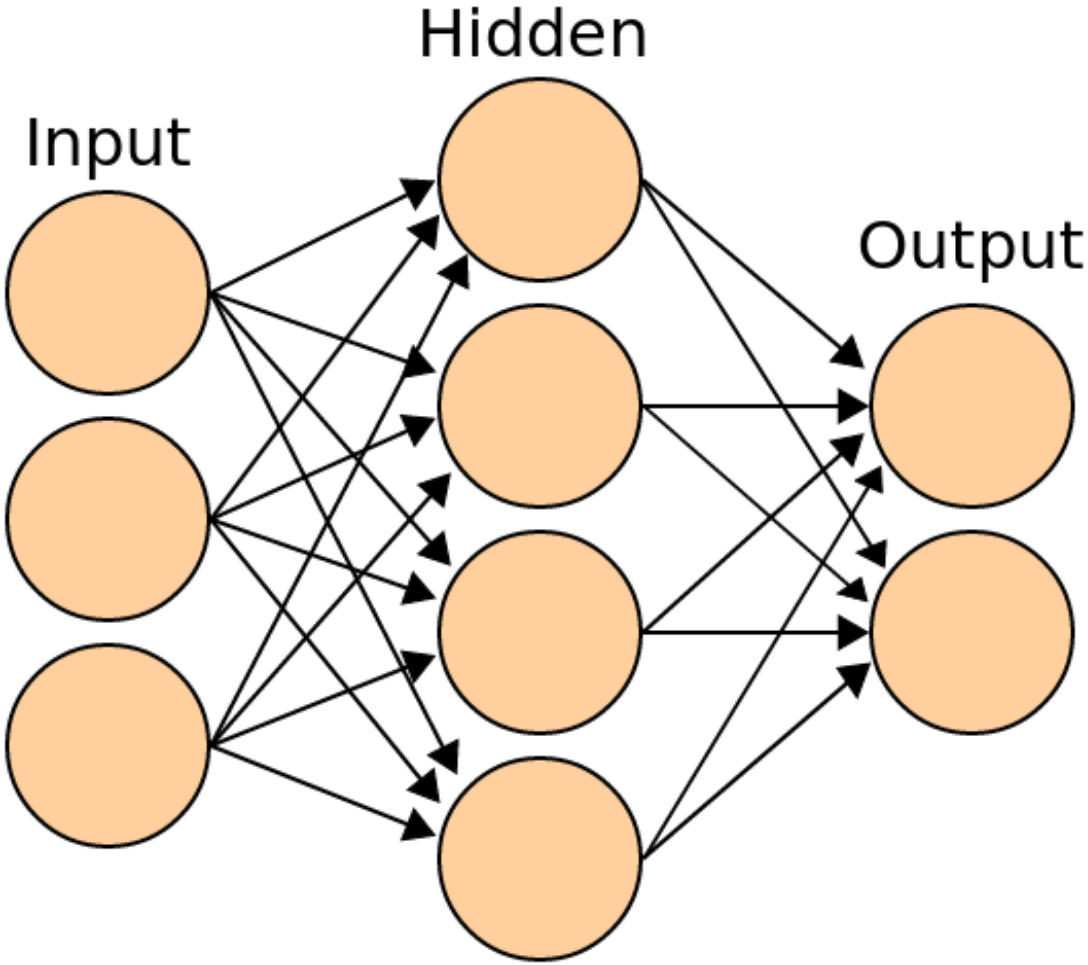
Supervised Neural Networks

- Non-linear models for classification and regression
- Work well for very large datasets
- Non-convex optimization
- Notoriously slow to train – need for GPUs
- Use dot products etc → require preprocessing, similar to SVM or linear models, unlike trees
- MANY variants (Convolutional nets, Gated Recurrent neural networks, Long-Short-Term Memory, recursive neural networks, variational autoencoders, general adverserial networks, neural turing machines...)

Linear regression drawn as neural net

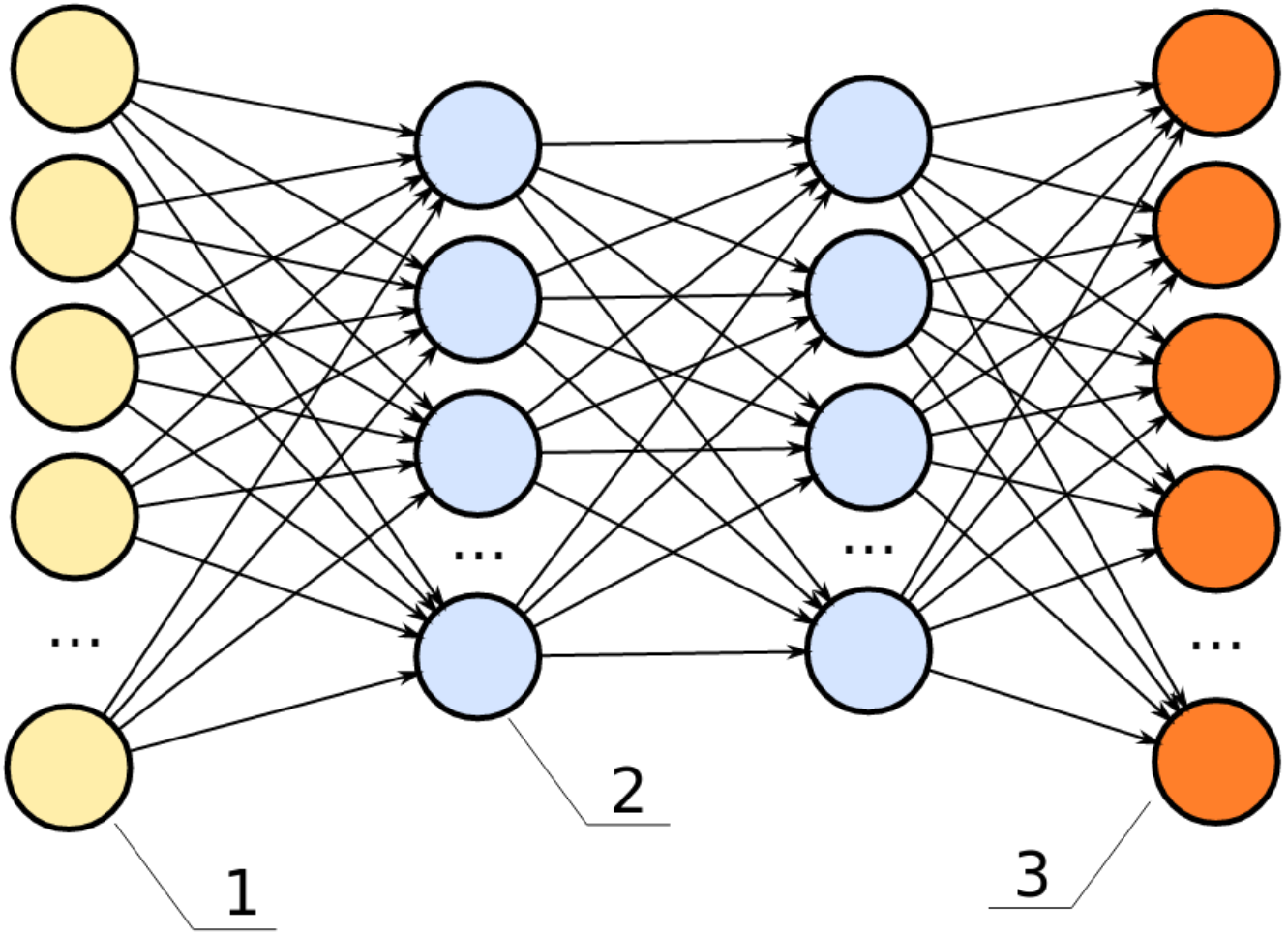


Basic Architecture (for making predictions)

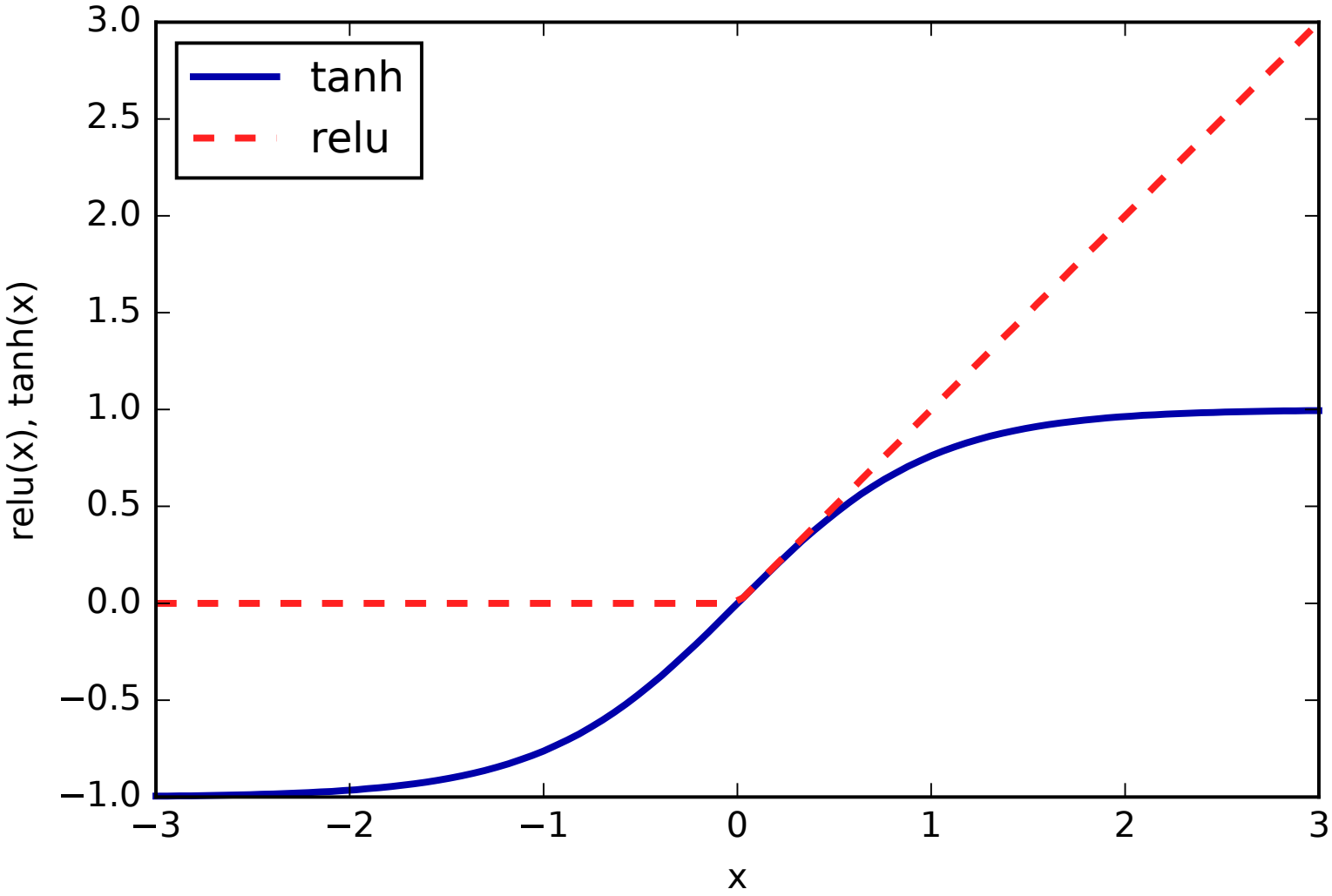


x $\quad$ $h(x) = f(Wx + b)$ $\quad$ $o(x) = g(W'h(x) + b')$

Can have arbitrary many layers



Nonlinear activation function



Training objective

$$h(\mathbf{x}) = f(W_1\mathbf{x} + \mathbf{b}_1)$$

$$o(\mathbf{x}) = g(W_2 h(\mathbf{x}) + \mathbf{b}_2) = g(W_2 f(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)$$

$$\min_{W_1, W_2, \mathbf{b}_1, \mathbf{b}_2} \sum_{i=1}^{N} \ell(y_i, o(\mathbf{x}_i))$$

Could add regularization

$$= \min_{W_1, W_2, \mathbf{b}_1, \mathbf{b}_2} \sum_{i=1}^{N} \ell(y_i, g(W_2 f(W_1\mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_2))$$

ℓ squared loss for regression
cross-entropy loss (multi-class log-loss) for classification

Backpropagation

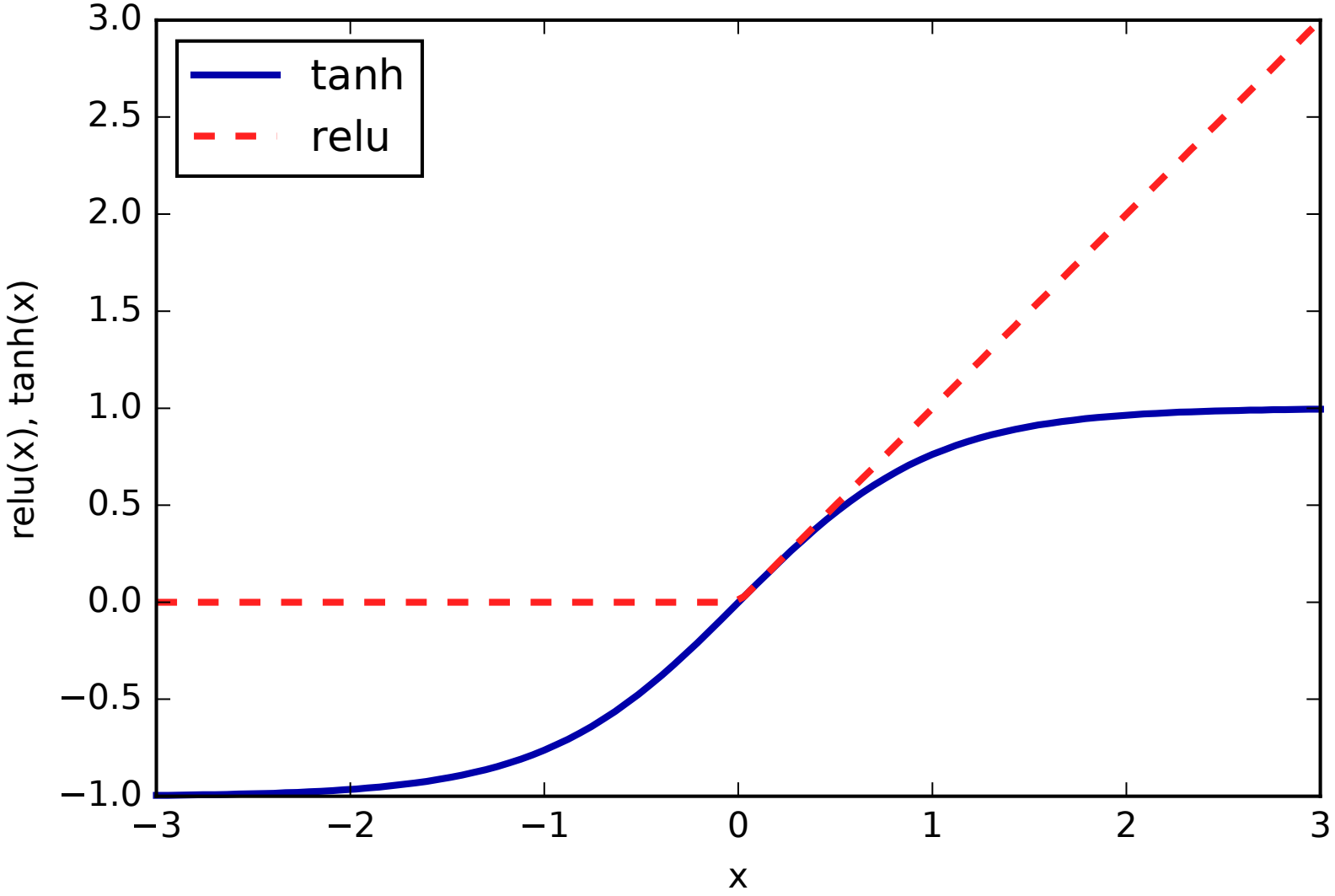
- For gradient based-method need $\frac{\partial o(\mathbf{x})}{\partial W_i}$ $\frac{\partial o(\mathbf{x})}{\partial \mathbf{b}_i}$

$$\text{net}(\mathbf{x}) := W_1\mathbf{x} + b_1$$

$$\frac{\partial o(\mathbf{x})}{\partial W_1} = \underbrace{\frac{\partial o(\mathbf{x})}{\partial h(\mathbf{x})}}_{\text{backpropagation of gradient of layer above.}} \underbrace{\frac{\partial h(\mathbf{x})}{\partial \text{net}(\mathbf{x})}}_{\text{Gradient of Non-linearity f}} \underbrace{\frac{\partial \text{net}(\mathbf{x})}{\partial W_1}}_{\text{Input to 1st layer x}}$$

Backpropagation = Chain Rule + Dynamic Programming

But wait!



Optimizing W, b

$$W_i \leftarrow W_i - \eta \sum_{j=1}^{n} \frac{\ell(\mathbf{x}_j, y_j)}{W_i}$$ batch

$$W_i \leftarrow W_i - \eta \sum_{j=k}^{k+m} \frac{\ell(\mathbf{x}_j, y_j)}{W_i}$$ minibatch

$$W_i \leftarrow W_i - \eta \frac{\ell(\mathbf{x}_j, y_j)}{W_i}$$ Online / stochastic

Learning Heuristics

- Constant η not good
- Can decrease η
- Better: adaptive η for each entry if Wi
- State-of-the-art: adam (with magic numbers)

https://arxiv.org/pdf/1412.6980.pdf

http://sebastianruder.com/optimizing-gradient-descent/

Open Questions in Neural Networks

- Why are they so effective?
- When are they effective?
- What are good learning methods and architectures?
- What role does optimization play?
- How can we implement memory or algorithms in neural networks?

Convolutional Neural Networks

Idea

- Translation invariance
- Weight sharing

Definition of Convolution

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]\, g[n-m]$$

$$= \sum_{m=-\infty}^{\infty} f[n-m]\, g[m].$$

f 0 1 2 0 1 1 0 0 1 0

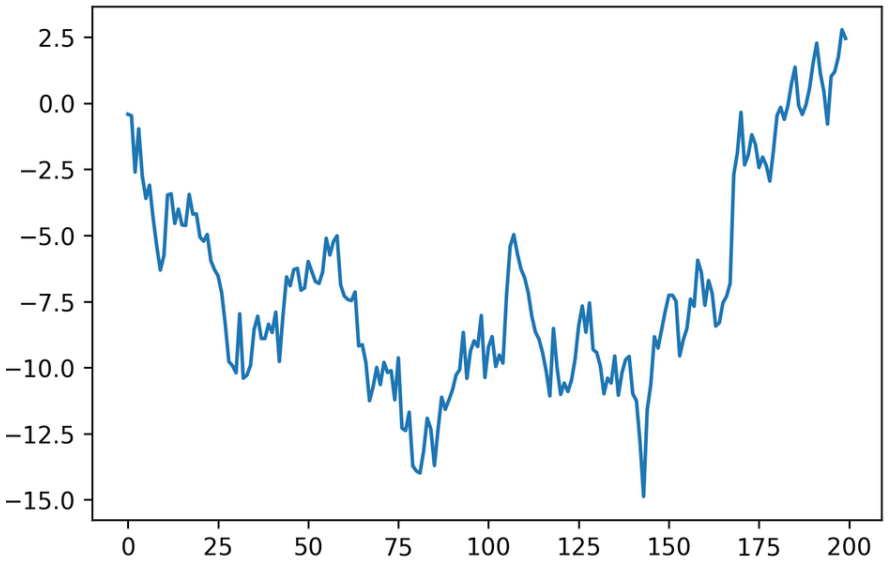
g -1 2 -1

-1 2 -1

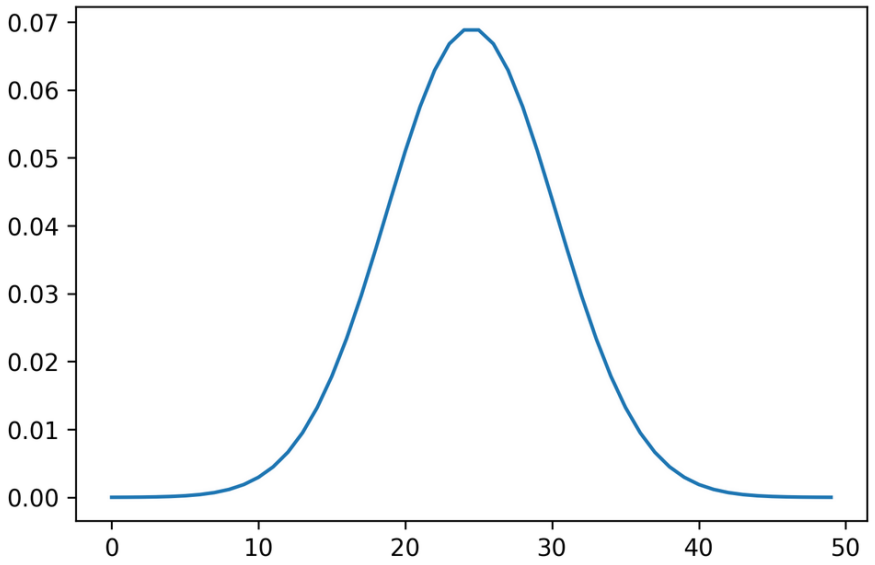
-1 2 -1

$f * g$ 0 3 -3 ...

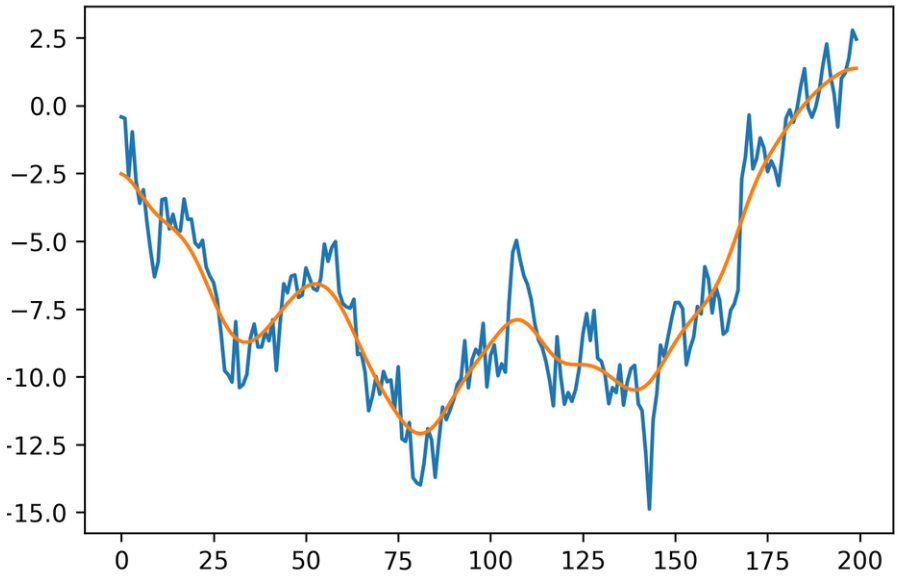
1d example: Gaussian smoothing



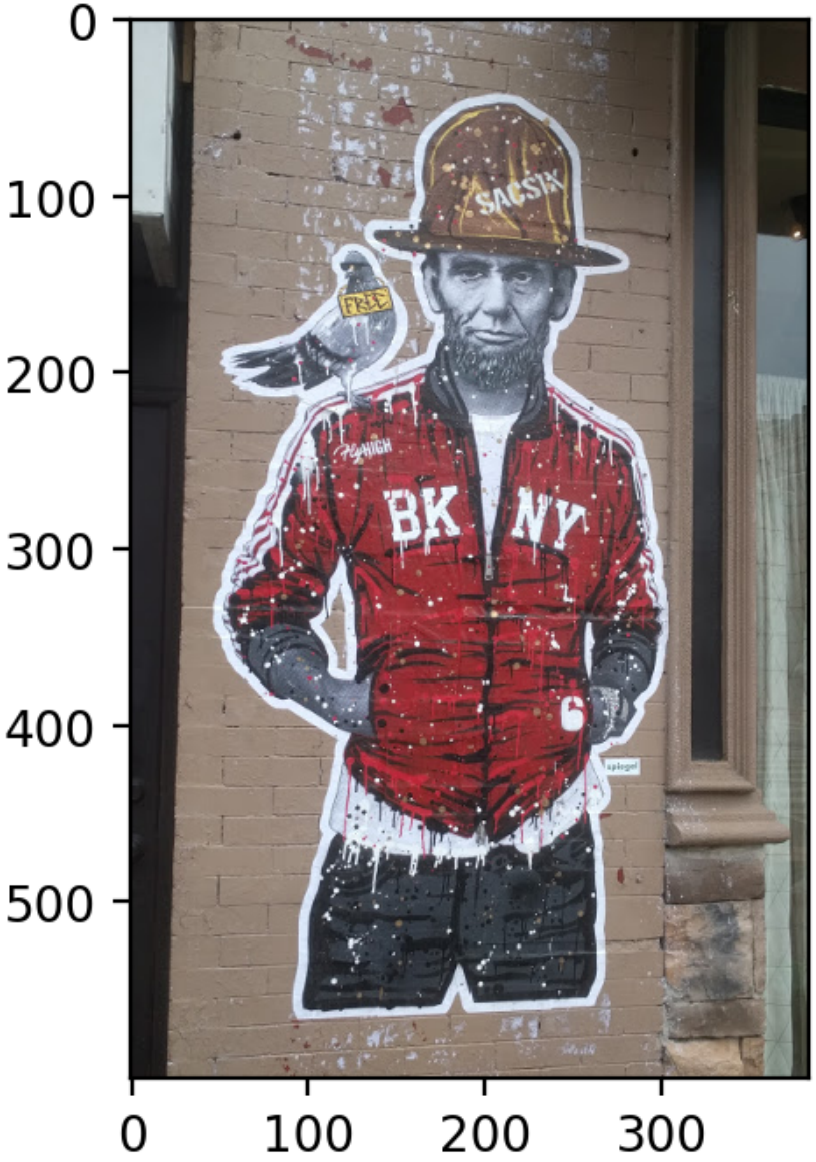
*

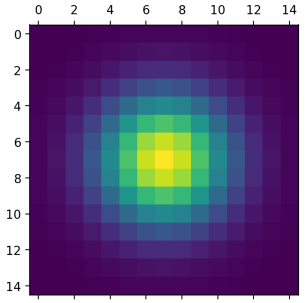


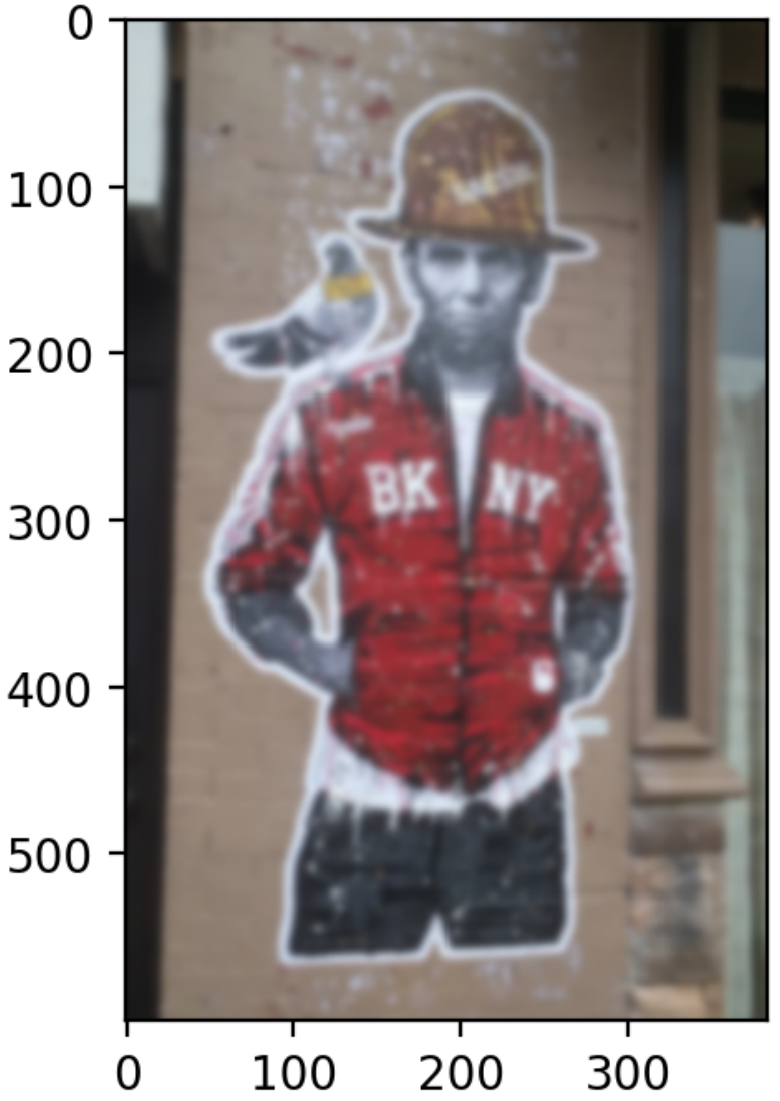
=



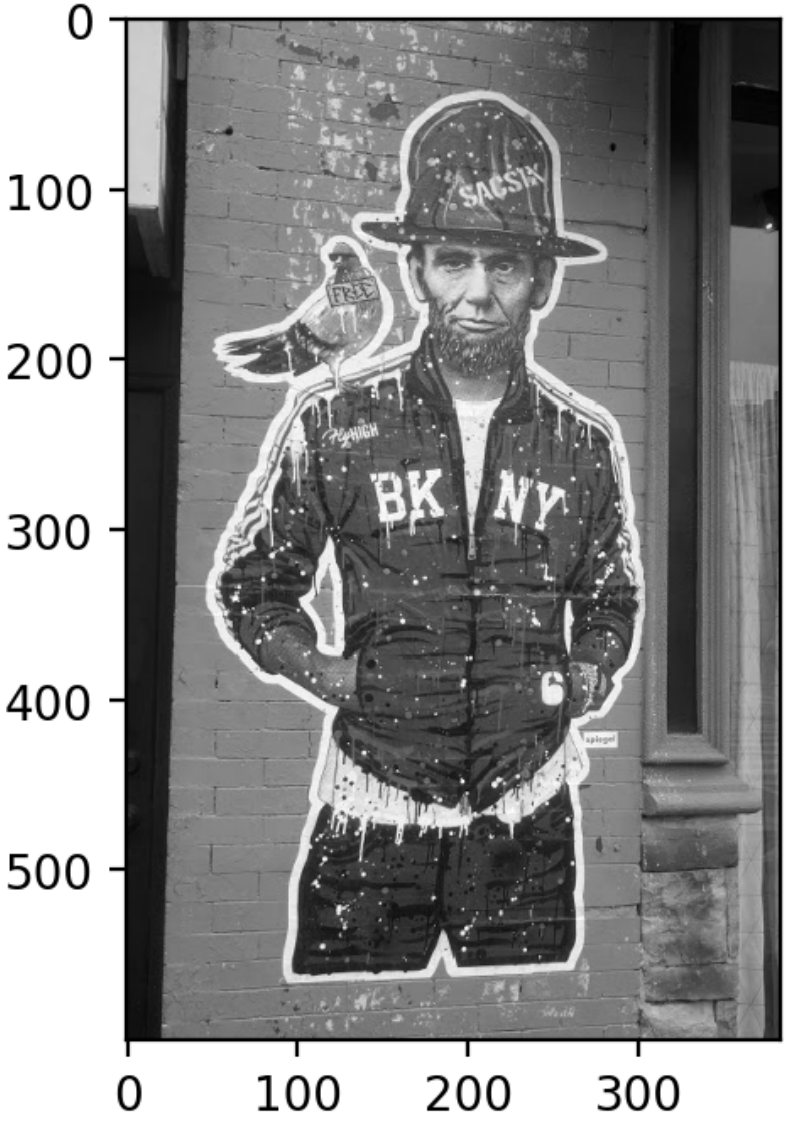
2d smoothing

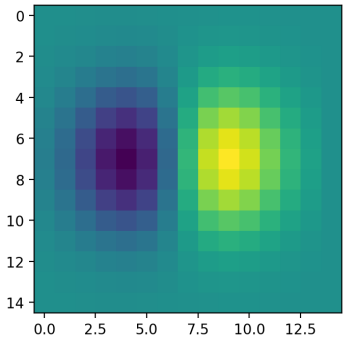


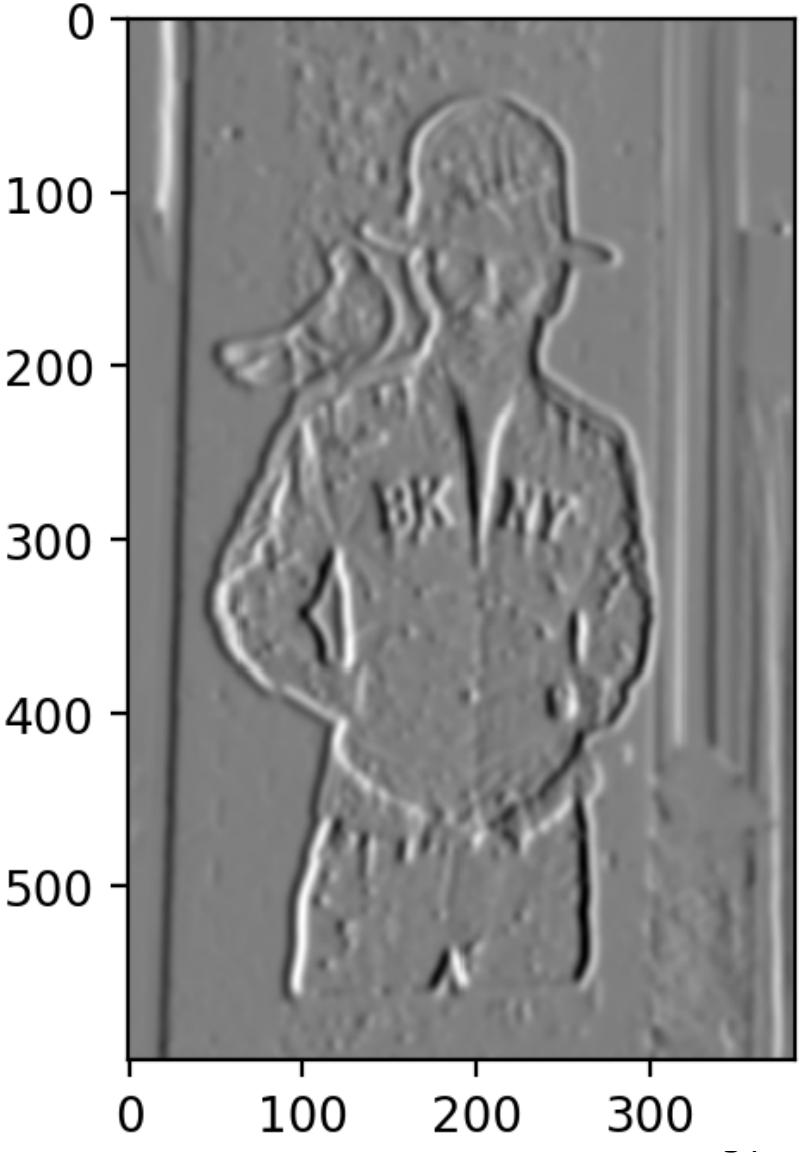




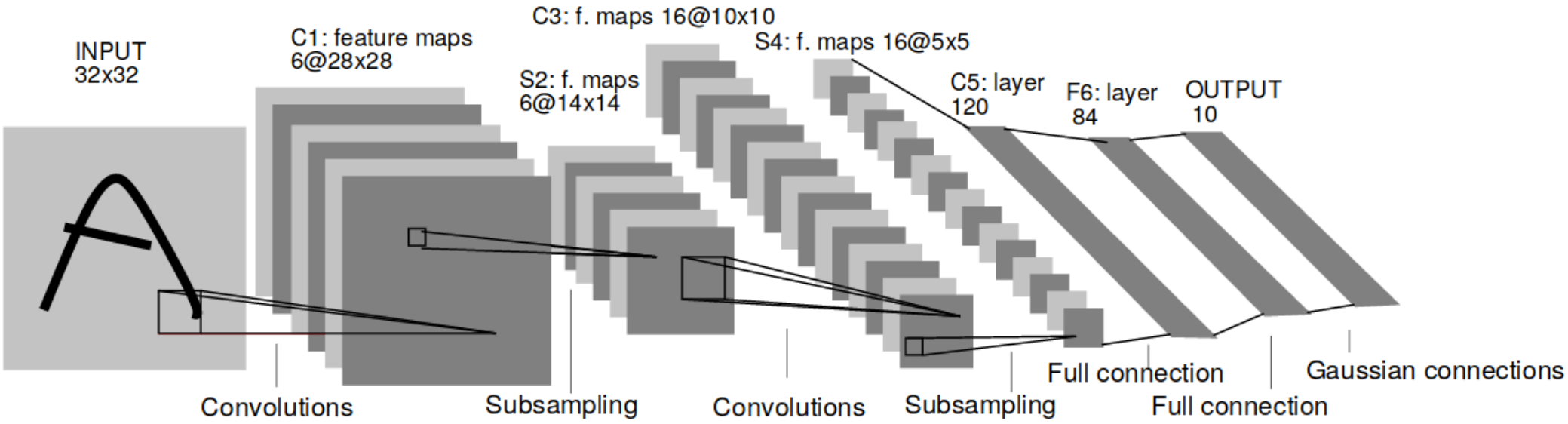
2d Gradients







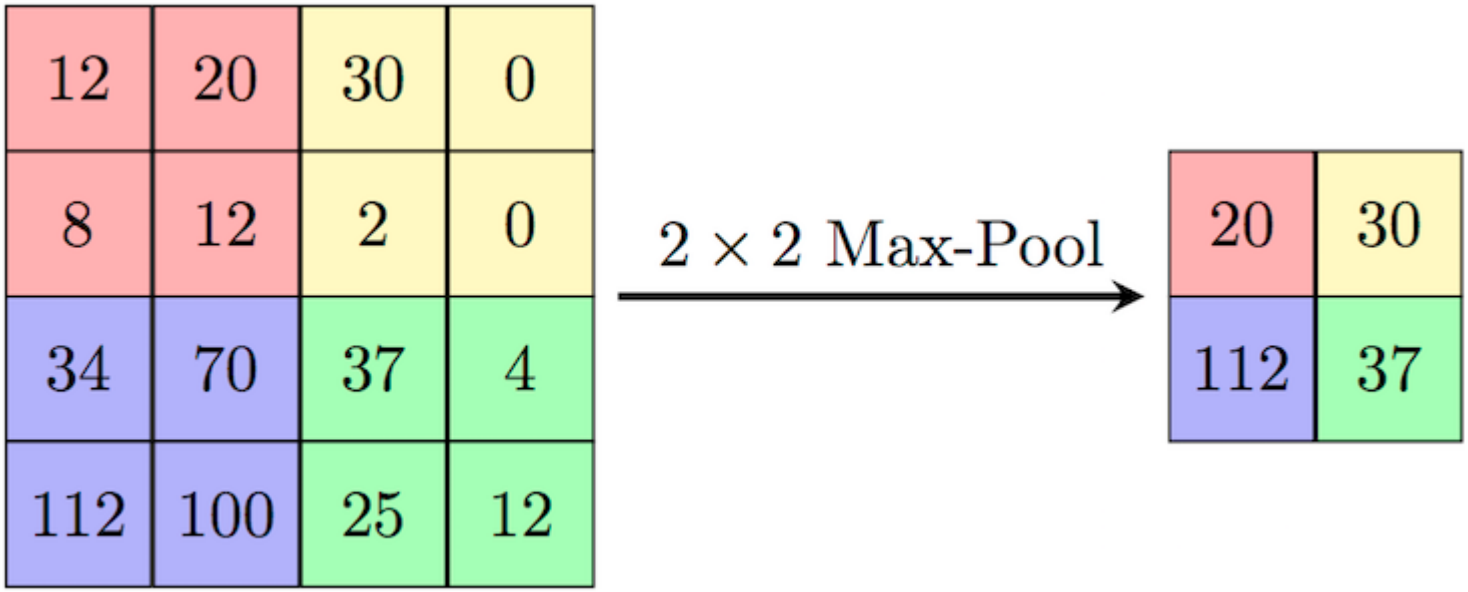
Convolutional Neural Networks



Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner.

Gradient-based learning applied to document recognition

Max pooling

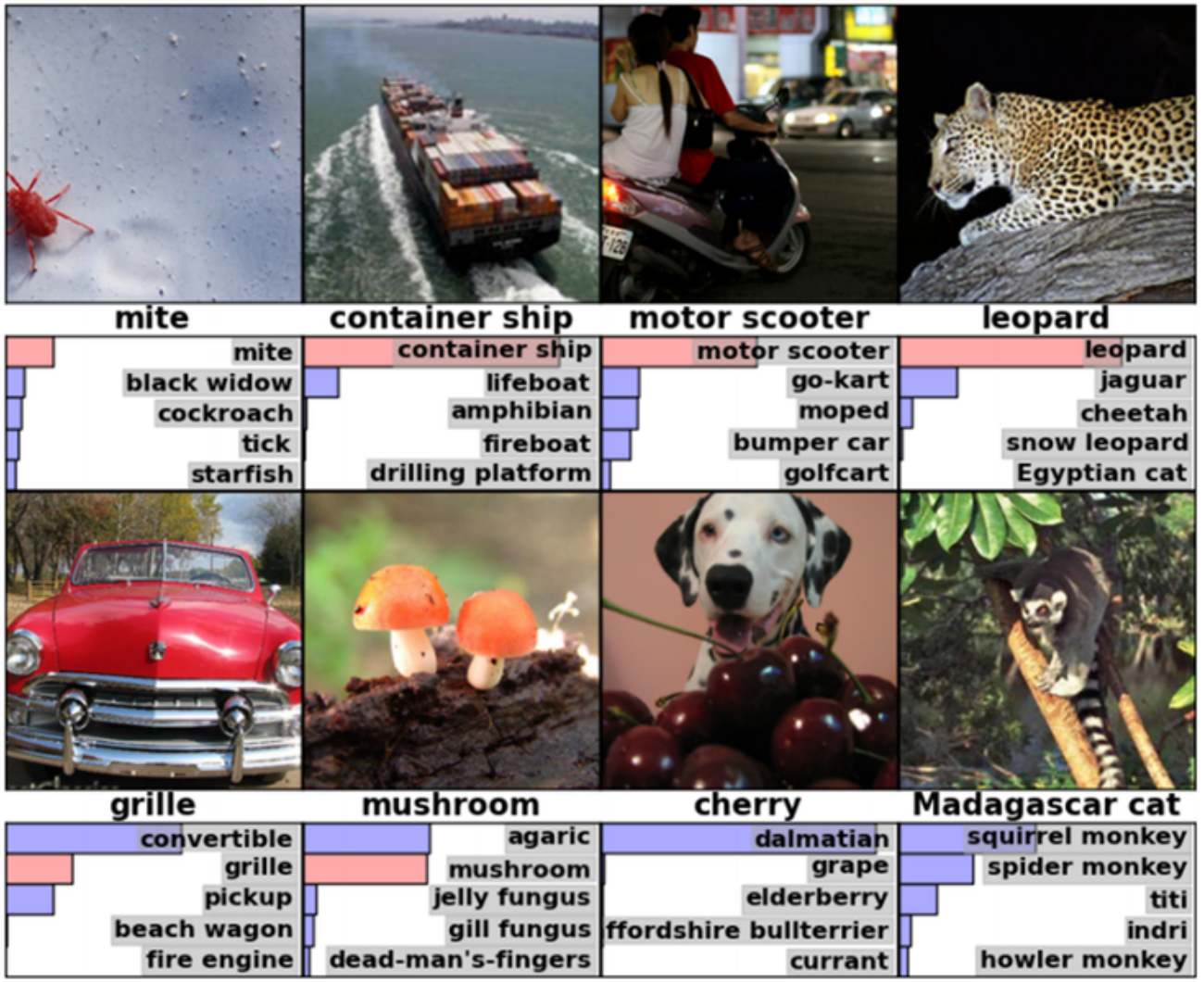


Need to remember position of maximum for back-propagation.
Again not differentiable → subgradient descent

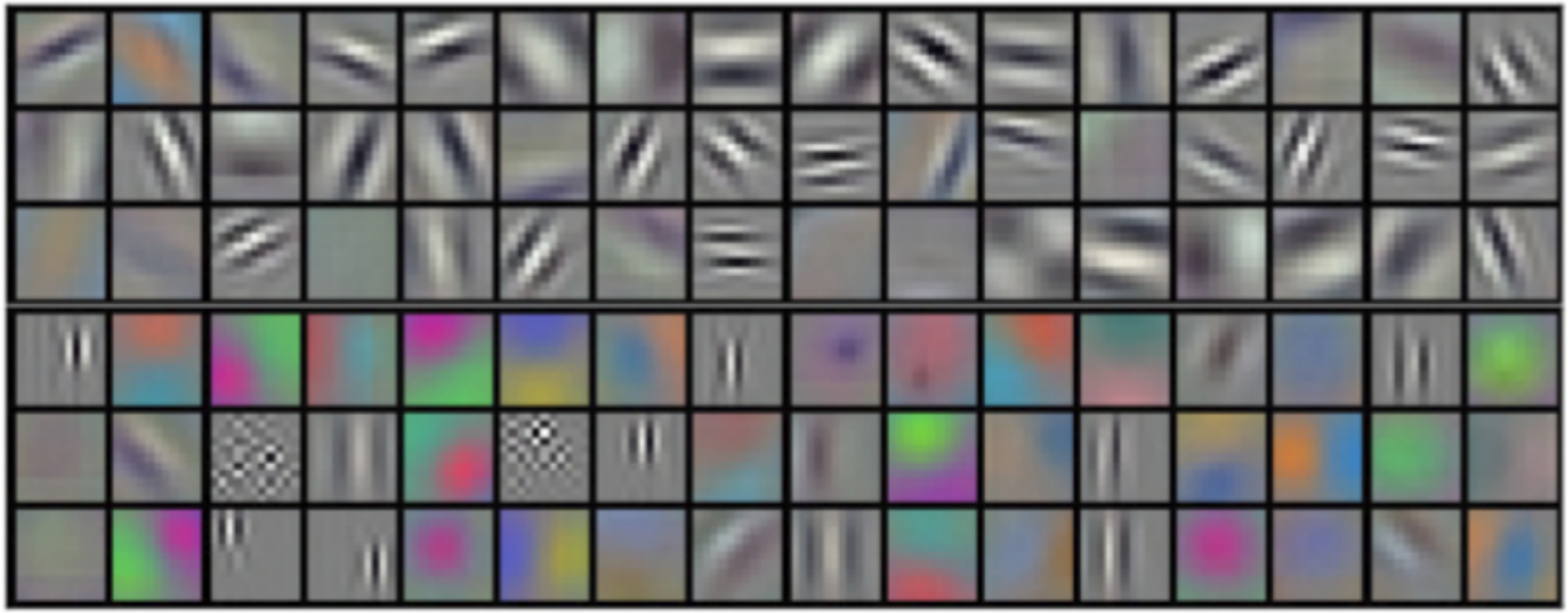
ImageNet Challenge

IMAGENET

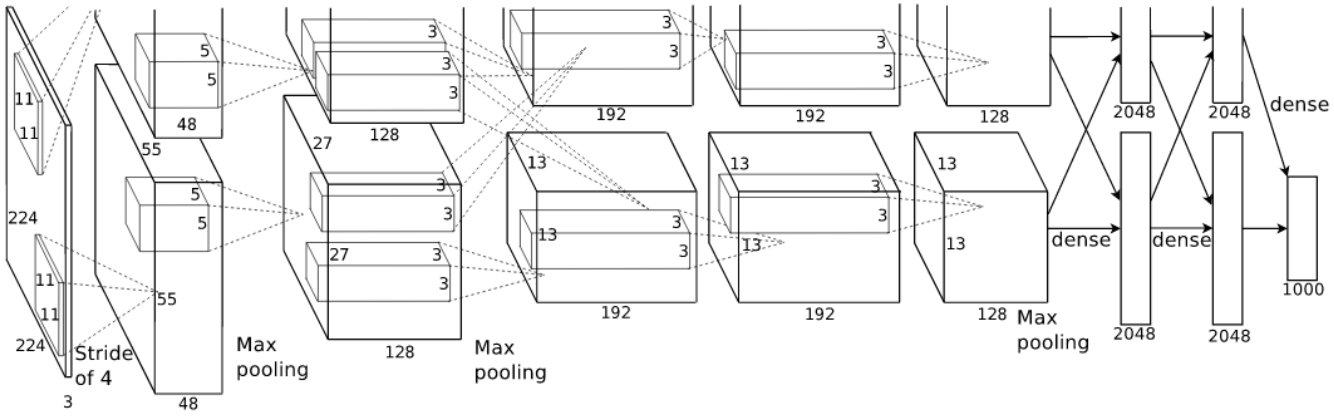
- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

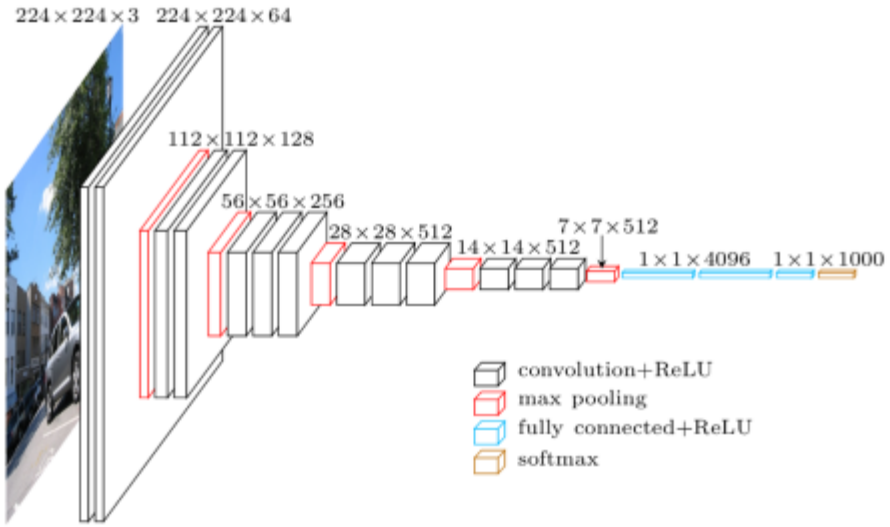


Typical first layer filters



Alex Net





VGG 16

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

conv <receptive field size>-<number of channels>

Open Questions in Convolutional Neural Networks

- What are good learning methods and architectures?
- Where else could they be applied?
- When to use CNN vs Recurrent networks?

Structured Prediction

Structured Prediction

$$y = (y_1, y_2, \cdots y_{n_k})$$

Applications: Multi-Label Classification

	Politics	Sports	Finance	Domestic	Religion
News Story1	1	0	0	1	1
News Story2	0	1	0	1	0
News Story3	0	0	1	0	0

Applications: Multi-Label Classification

	Politics	Sports	Finance	Domestic	Religion
News Story1	1	0	0	1	1
News Story2	0	1	0	1	0
News Story3	0	0	1	0	0

	Owns Car	Smokes	Married	Self-Employed	Has Kids
Customer1	1	0	1	0	1
Customer2	1	1	0	1	0
Customer3	0	1	1	0	0

Applications: Sequence Tagging





Applications: Sequence Tagging

Stroke cat.

Stroke cat.

Stroke cat.

Open trash can.

Put cat in trash can.

Applications: Image Segmentation

The Essence of Structured Prediction

$$f(x, w) := \arg\max_{y \in \mathcal{Y}} g(x, y, w)$$

The Essence of Structured Prediction

$$f(x, w) := \arg\max_{y \in \mathcal{Y}} g(x, y, w)$$

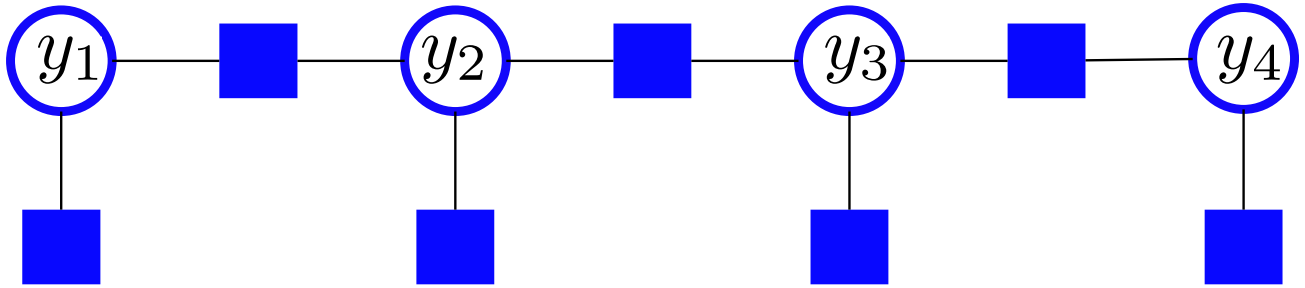
If you like:

$$\arg\max_{y \in \mathcal{Y}} p(y|x, w)$$

Pairwise Structured Models

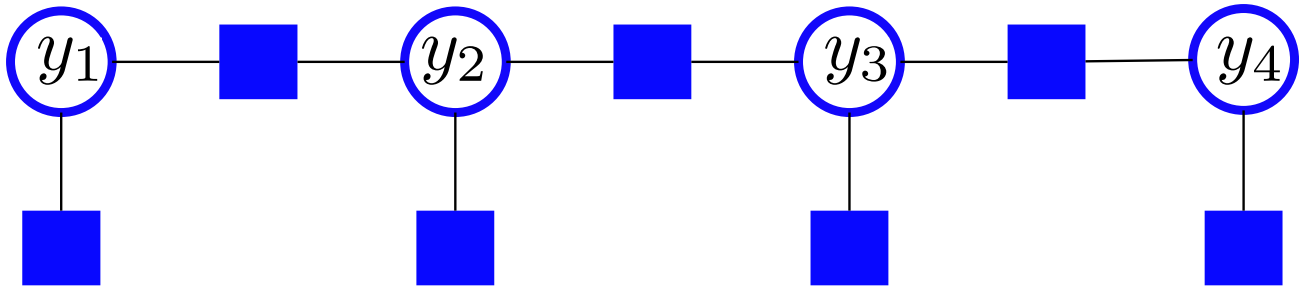
$$\arg\max_{y_1, y_2, \ldots, y_n} w^T \psi(x, y)$$

$$= \arg\max_{y_1, y_2, \ldots, y_n} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j) \in E} w_{i,j}^T \psi(x, y_i, y_j)$$



The Devil is in the Inference

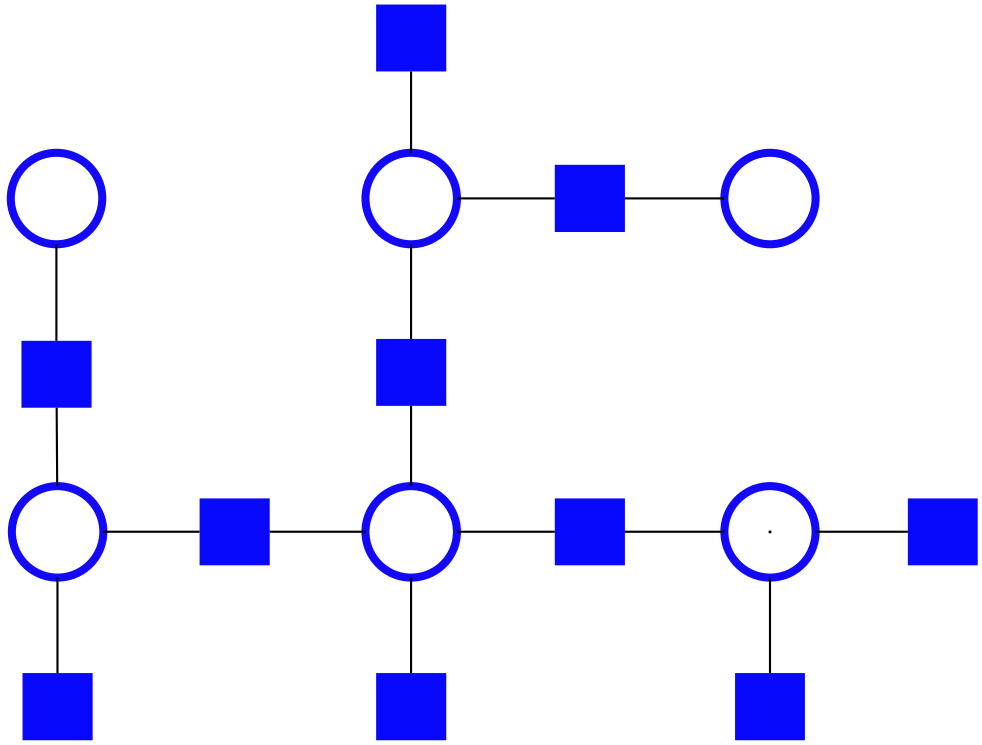
$$\arg\max_{y_1,y_2,\ldots,y_n} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j)\in E} w_{i,j}^T \psi(x, y_i, y_j)$$



Easy: Dynamic Programming

The Devil is in the Inference

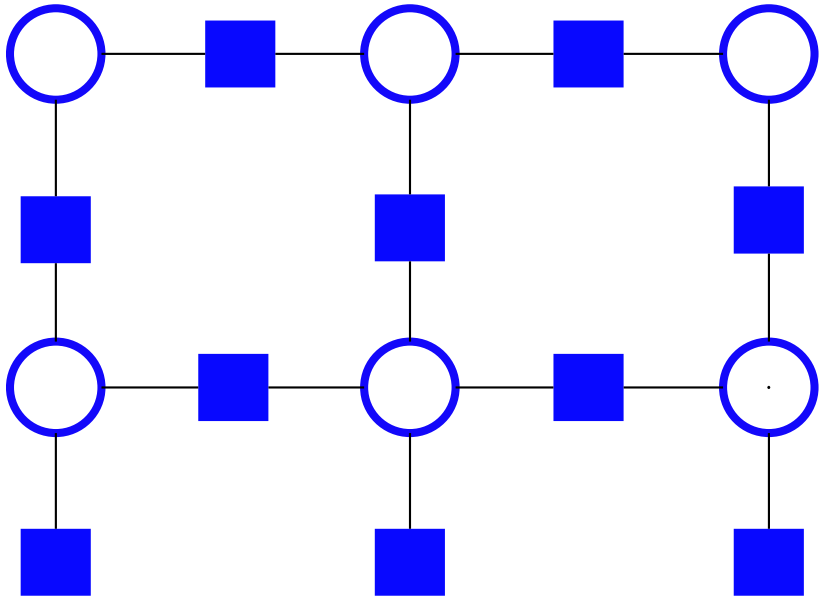
$$\arg\max_{y_1,y_2,\ldots,y_n} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j)\in E} w_{i,j}^T \psi(x, y_i, y_j)$$



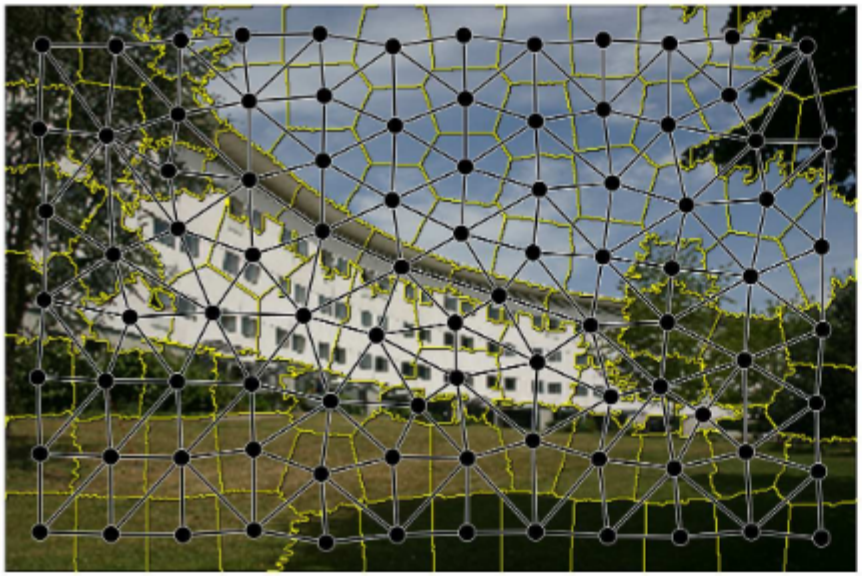
Easy: Dynamic Programming

The Devil is in the Inference

$$\underset{y_1,y_2,\ldots,y_n}{\arg\max} \sum_I w_i^T \psi(x, y_i) + \sum_{(i,j)\in E} w_{i,j}^T \psi(x, y_i, y_j)$$



HARD!
AD3, QPBO, LP, Loopy BP, ….



Open Questions in Structured Prediction

- How to learn more efficiently?
- How to best integrate with deep learning?
- How to do inference efficiently in more complex models?

The Open Source Idea

Open exchange

We can learn more from each other when information is open. A free exchange of ideas is critical to creating an environment where people are allowed to learn and use existing information toward creating new ideas.

Participation

When we are free to collaborate, we create. We can solve problems that no one person may be able to solve on their own.

Rapid prototyping

Rapid prototypes can lead to rapid failures, but that leads to better solutions found faster. When you're free to experiment, you can look at problems in new ways and look for answers in new places. You can learn by doing.

Meritocracy

In a meritocracy, the best ideas win. In a meritocracy, everyone has access to the same information. Successful work determines which projects rise and gather effort from the community.

New BSD License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

a. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
b. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
c. Neither the name of the Scikit-learn Developers nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Permissive vs Share-alike

BSD / MIT / Apache

Hadoop

Python (+libraries)

Apache web server

Do whatever you want
= Industry friendly

GPL / LGPL

Linux

Most linux tools (GNU)

Derived software must be licensed the same way.
Derived web-services must be open source.

No “free riders”


scikit
learn

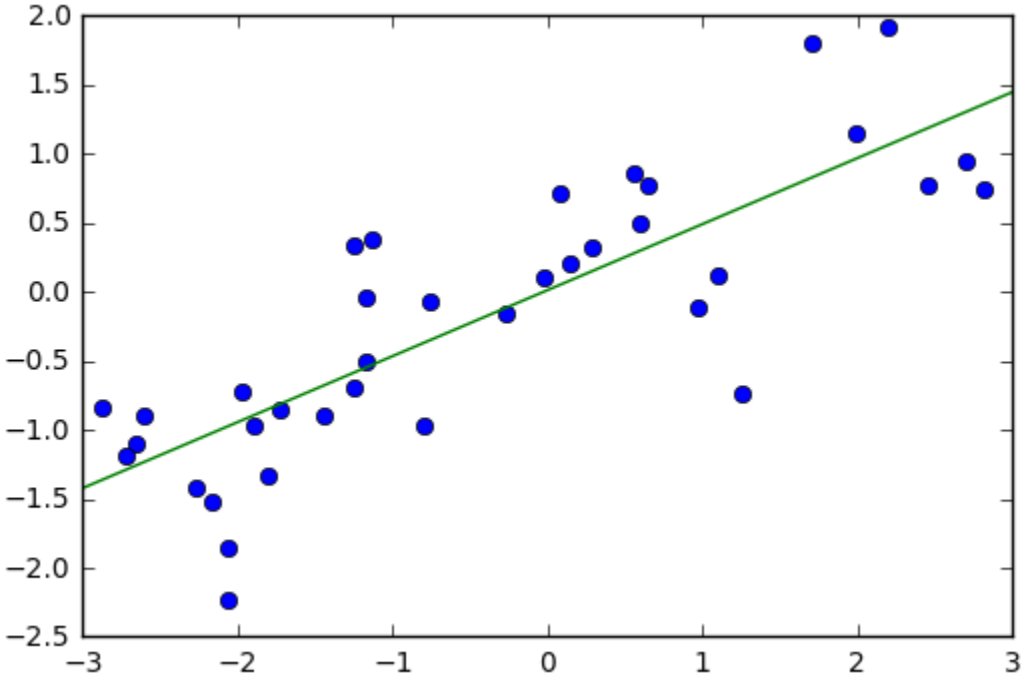
Classification
Regression
Clustering
Semi-Supervised Learning
Feature Selection
Feature Extraction
Manifold Learning
Dimensionality Reduction
Kernel Approximation
Hyperparameter Optimization
Evaluation Metrics
Out-of-core learning
……

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train)

print(lr.predict(X_test))
print(lr.score(X_test, y_test))
```

```
[-1.07669396 -0.94489099  0.29582604  1.32379186 -1.38209551 -1.42220547
  0.86691958 -0.63314097  0.79944587  1.28044585  0.64491641  0.23934466
 -0.73008772]
0.465661173258
```




Alexandre Gramfort
agramfort


Alexander Fabisch
AlexanderFabisch


Alexandre Passos
alextp


Andreas Mueller
amueller


Arnaud Joly
arjoly


Brian Holt
bdholt1

bthirion
bthirion


Chris Filo Gorgolewski
chrisfilo


David Cournapeau
cournape


Duchesnay
duchesnay


David Warde-Farley
dwf


Fabian Pedregosa
fabianp


Gael Varoquaux
GaelVaroquaux


Gilles Louppe
glouppe


Jake Vanderplas
jakevdp


Jaques Grobler
jaquesgrobler


Jan Hendrik Metzen
jmetzen


Jacob Schreiber
jmschrei


Joel Nothman
jnothman


Kyle Kastner
kastnerkyle


Lars
larsmans


Loïc Estève
lesteve


Shiqiao Du
lucidfrontier45


Mathieu Blondel
mblondel


Manoj Kumar
MechCoder


Noel Dawe
ndawe


Nelle Varoquaux
NelleV


Olivier Grisel
ogrisel


Paolo Losi
paolo-losi


Peter Prettenhofer
pprett


(Venkat) Raghav (Rajagopalan)
raghavrv


Robert Layton
robertlayton


Ron Weiss
ronw


Satrajit Ghosh
satra


sklearn-ci


sklearn-wheels


Tom Dupré la Tour
TomDLT


Vlad Niculae
vene


Virgile Fritsch
VirgileFritsch


Vincent Michel
vmichel

Wei Li
weilinear

Yaroslav Halchenko
yarikoptic

scikit-learn / **scikit-learn**

Unwatch 1,841 | Unstar 21,553 | Fork 11,545

Code | Issues 905 | Pull requests 582 | Projects 5 | Wiki | Settings | Insights

Filters | is:pr is:open | Labels | Milestones | New pull request

582 Open ✓ 4,913 Closed | Author | Labels | Projects | Milestones | Reviews | Assignee | Sort

classification_report: raise error if labels is None and target_names size is not equal to present classes (Fixes #9842) ✓
#9867 opened 10 hours ago by reiinakano

retrieve embedded neighbor indexes using KNN search ✓ 6
#9861 opened a day ago by PGryllos • Approved

Update performance.rst ✗ 1
#9859 opened 2 days ago by nbmorgan

[MRG] Add tol parameter and deprecate reorder parameter in auc ✓ 8
#9851 opened 5 days ago by qinhanmin2014

FIX Improve check to ensure that ROC curve starts at (0,0) point ✗ 7
#9850 opened 5 days ago by alexryndin

[MRG] speedup confusion_matrix ✓ 14
#9843 opened 6 days ago by Erotemic 5 of 5

[RFC/MRG] MAINT Use magic to list documentation versions ✓ 5
#9841 opened 6 days ago by jnothman 0.19.1

[MRG] Remove nose from CIs and documentation ✗ 19
#9840 opened 6 days ago by lesteve 0.20

[MRG] MAINT pytest test discovery ✓ 3
#9839 opened 6 days ago by rth

```
-    y_pred = y_pred[ind]
277  -    y_true = y_true[ind]
278  -    # also eliminate weights of eliminated items
279  -    sample_weight = sample_weight[ind]
     269 +    # If labels are not consecitive integers starting from zero, then
     270 +    # yt, yp must be converted into index form
     271 +    need_index_conversion = not (
```

jnothman 5 days ago **Owner**

Surely `np.all(labels == np.arange(len(labels))` is just as fast for a reasonable number of classes and much more readable?

Erotemic 3 days ago • edited Contributor

It is both faster and more readable, when I originally wrote this I didn't consider that the ordering of the labels was significant, so the `np.diff` was a quick fix after realizing this. Your solution is simpler and better.

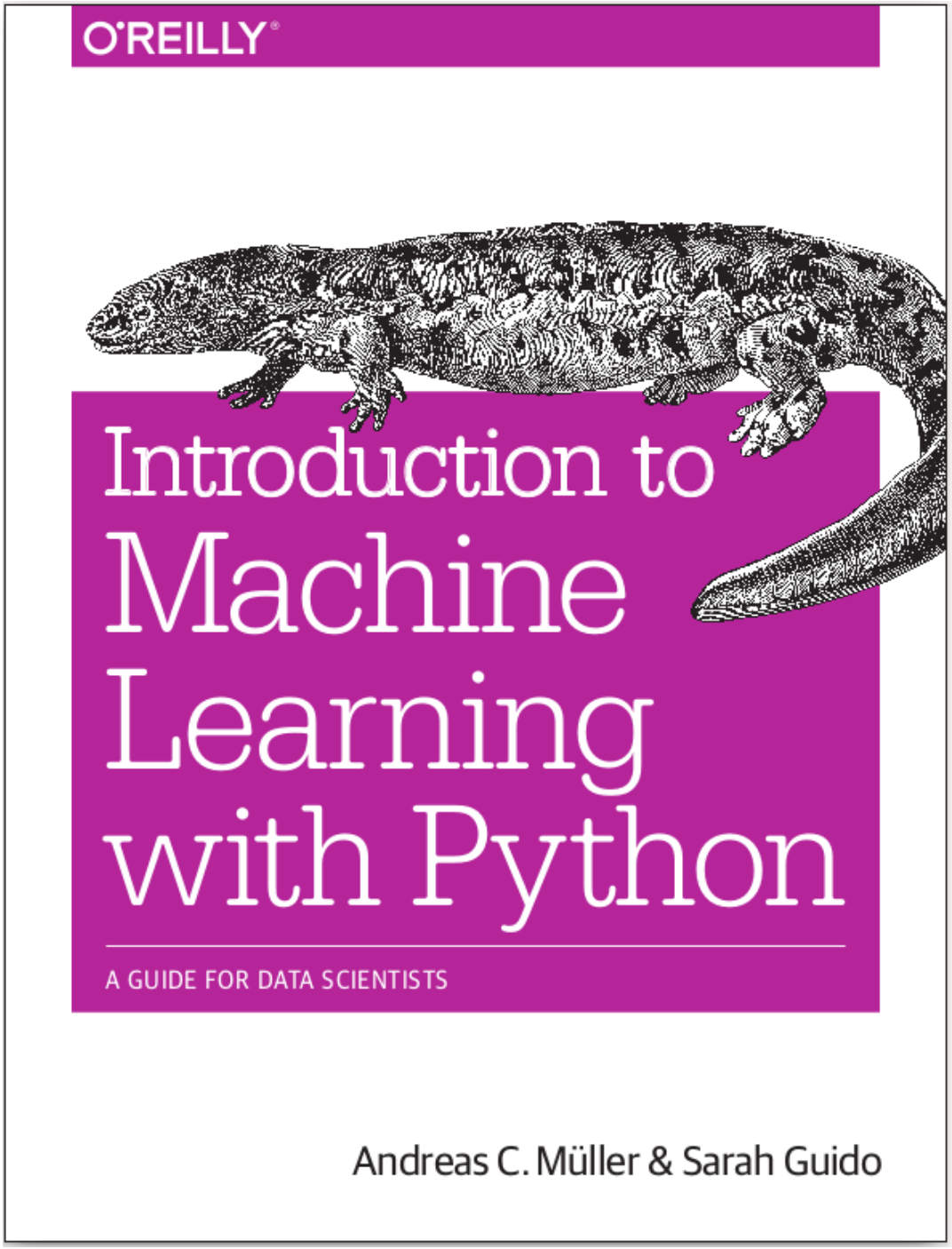
```
In [1]: labels = np.arange(100)

In [2]: %timeit labels.min() == 0 and np.all(np.diff(labels) == 1)
8.3 µs ± 162 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

In [3]: %timeit np.all(np.arange(len(labels)) == labels)
3.6 µs ± 47.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Reply...

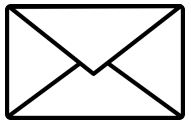
Start a new conversation



amueller.github.io

@amuellerml

@amueller



andreas.mueller
@columbia.edu

https://github.com/amueller/talks_odt/