Machine Learning (and Deep Learning) in just over 200 minutes with R

From basic linear models to neural networks

Presented at the 2021 SAGI Symposium via Zoom

Presented by Yoni Nazarathy

https://yoninazarathy.com/

R Material created by Ajay Hemanth

https://www.linkedin.com/in/ajayhemanth/

These slides are available at: https://yoninazarathy.com/talks/ML225MinutesWithR.pdf
R source code is available at: https://github.com/ajayhemanth/Machine-Learning-Workshop

THE UNIVERSITY OF QUEENSLAND AUSTRALIA

On the menu

1. The state of the art and the ML world
2. The ML workflow and common “practice” data sets
3. Some tools you know - used for ML
4. Getting practical…

Who are you?
Biometricians, using R, using specialized software.
Strong understanding of statistics.

Practical Activities A - E

A) Linear Classifiers

B) Tensorflow playground

C) Dense Neural Networks

D) Random Forests

E) Convolutional Neural Networks

The speaker…

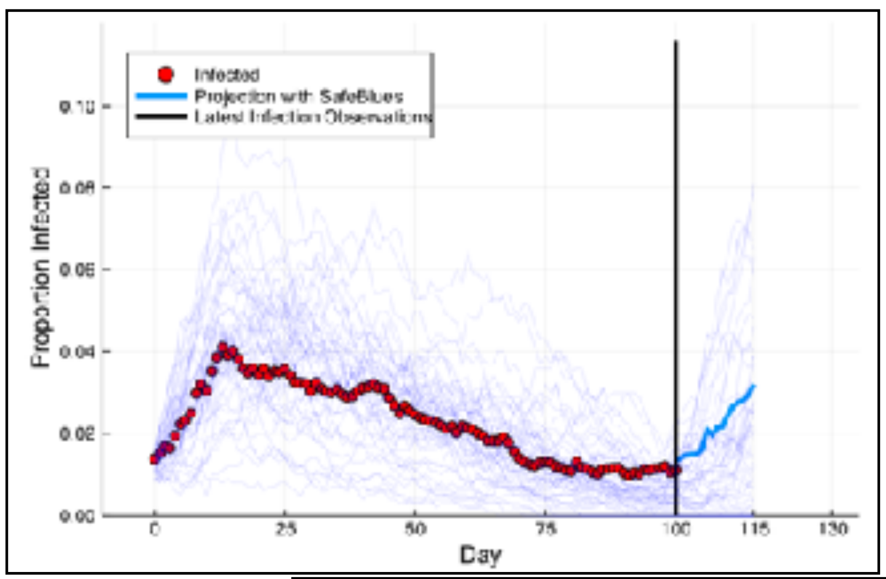
https://yoninazarathy.com/

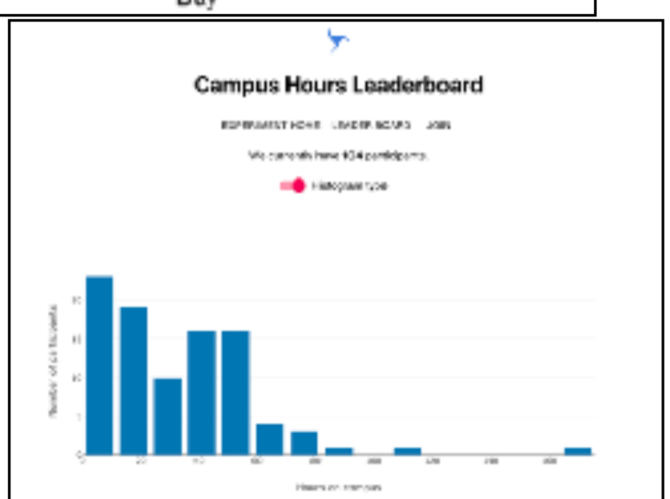
I am an Associate Professor at the School of Mathematics and Physics of The University of Queensland.

My expertise is in Machine Learning, Applied Probability, Statistics, Operations Research, Simulation, Scientific Computing, Control Theory, Queueing Theory, Scheduling, and Mathematical Education.

Application areas of my work include epidemics, wireless communication networks, bio-statistics, agriculture, power systems, software and hardware design, manufacturing logistics, healthcare logistics, and road traffic networks.

Current "Main" Project: AI4PAN and Safe Blues





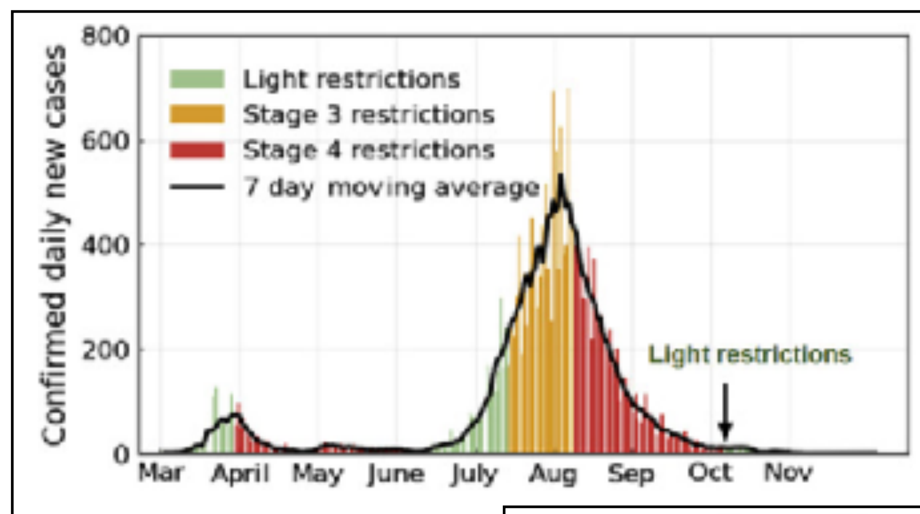
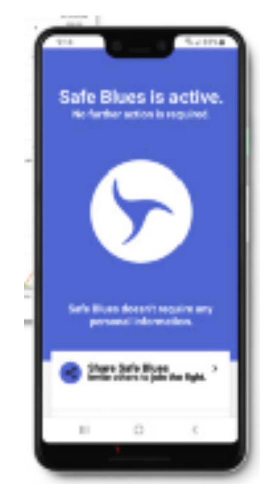
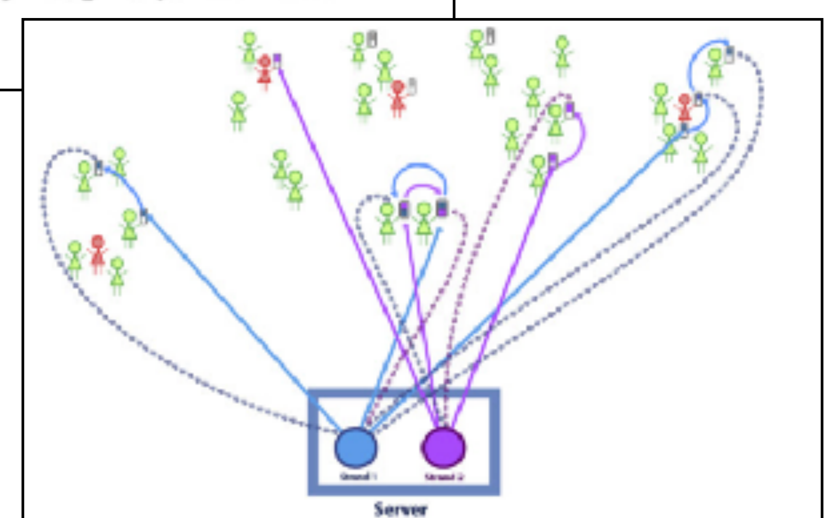
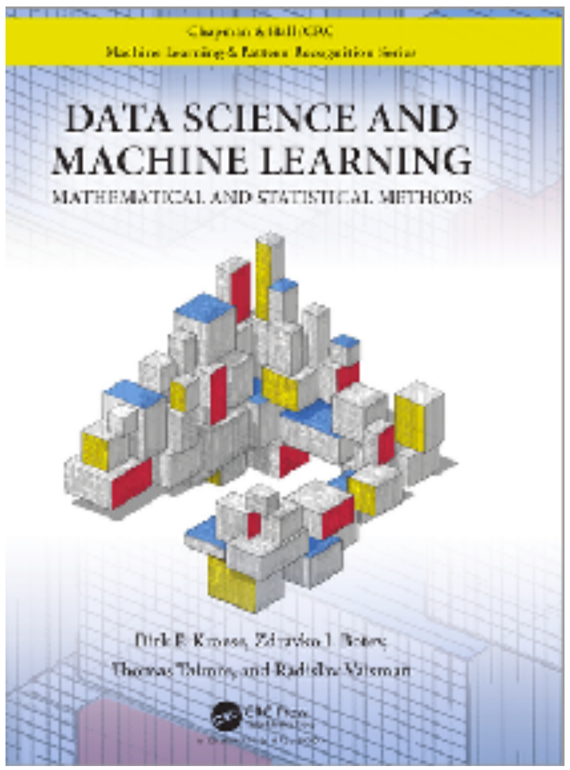


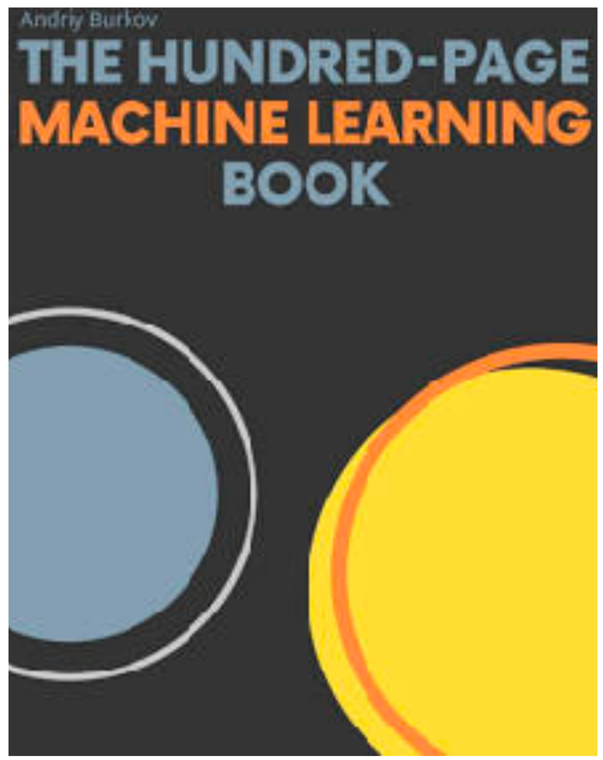
Figure 1 The 2020 outbreak in Victoria

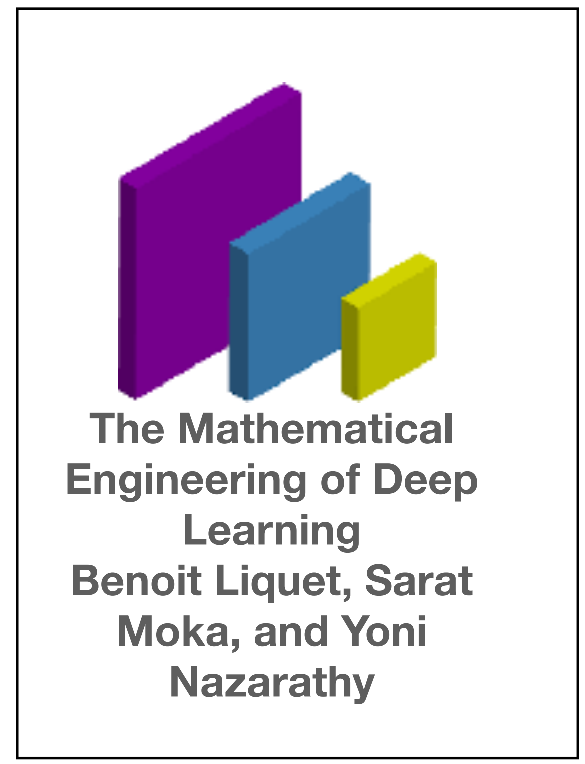


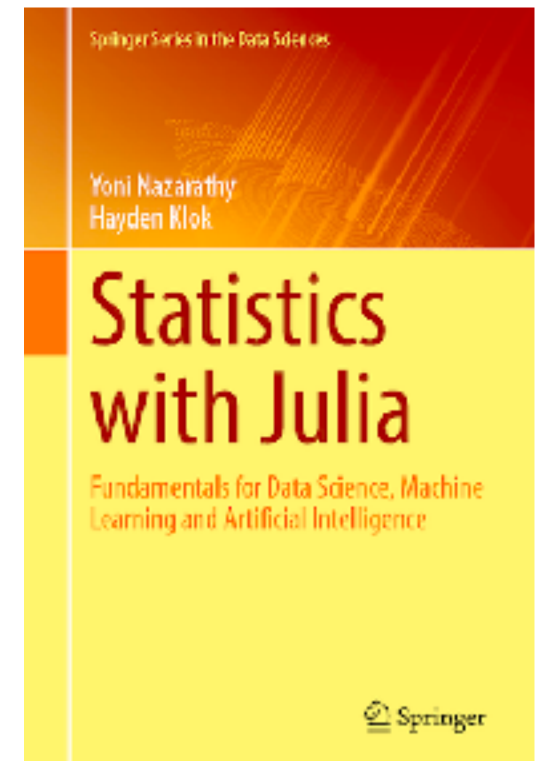


Some resources









https://people.smp.uq.edu.au/DirkKroese/DSML/

http://themlbook.com/

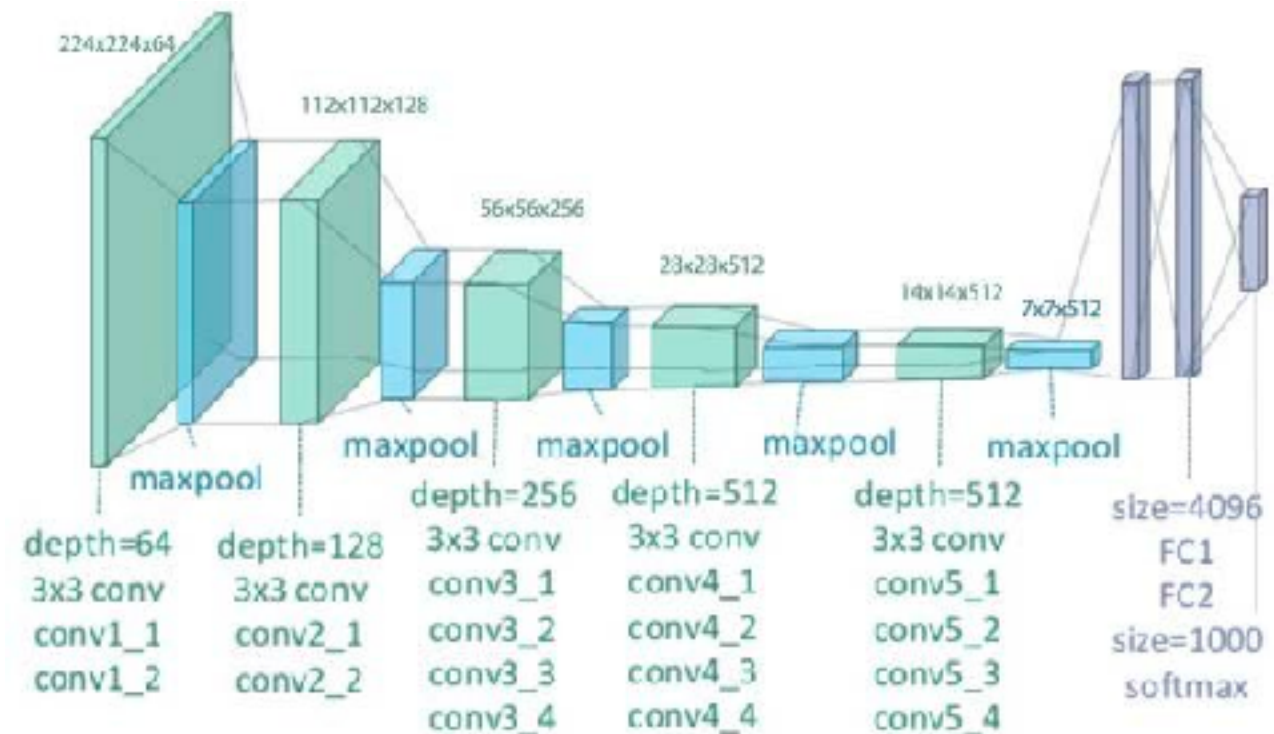
https://deeplearningmath.org/

https://statisticswithjulia.org/

Part 1: The state of the art and the ML world

```julia
using Metalhead
vgg = VGG19() #downloads about 0.5Gb of a pretrained neural network from the web
for (i,l) in enumerate(vgg.layers)
    println(i,": ", l)
end
```

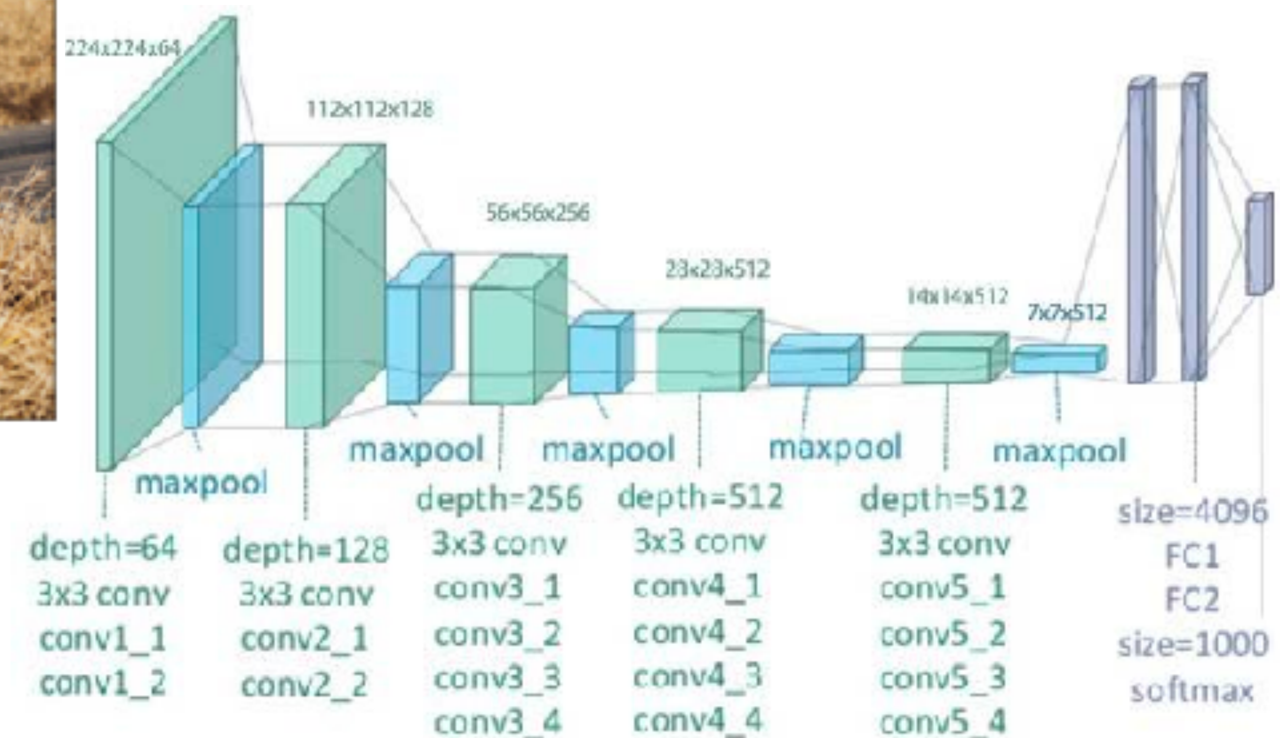
```
1: Conv((3, 3), 3=>64, relu)
2: Conv((3, 3), 64=>64, relu)
3: MaxPool((2, 2), pad = (0, 0, 0, 0), stride = (2, 2))
4: Conv((3, 3), 64=>128, relu)
5: Conv((3, 3), 128=>128, relu)
6: MaxPool((2, 2), pad = (0, 0, 0, 0), stride = (2, 2))
7: Conv((3, 3), 128=>256, relu)
8: Conv((3, 3), 256=>256, relu)
9: Conv((3, 3), 256=>256, relu)
10: Conv((3, 3), 256=>256, relu)
11: MaxPool((2, 2), pad = (0, 0, 0, 0), stride = (2, 2))
12: Conv((3, 3), 256=>512, relu)
13: Conv((3, 3), 512=>512, relu)
14: Conv((3, 3), 512=>512, relu)
15: Conv((3, 3), 512=>512, relu)
16: MaxPool((2, 2), pad = (0, 0, 0, 0), stride = (2, 2))
17: Conv((3, 3), 512=>512, relu)
18: Conv((3, 3), 512=>512, relu)
19: Conv((3, 3), 512=>512, relu)
20: Conv((3, 3), 512=>512, relu)
21: MaxPool((2, 2), pad = (0, 0, 0, 0), stride = (2, 2))
22: #44
23: Dense(25088, 4096, relu)
24: Dropout(0.5)
25: Dense(4096, 4096, relu)
26: Dropout(0.5)
27: Dense(4096, 1000)
28: softmax
```



The VGG19 Deep Neural network
(Image courtesy of Clifford K. Yang)

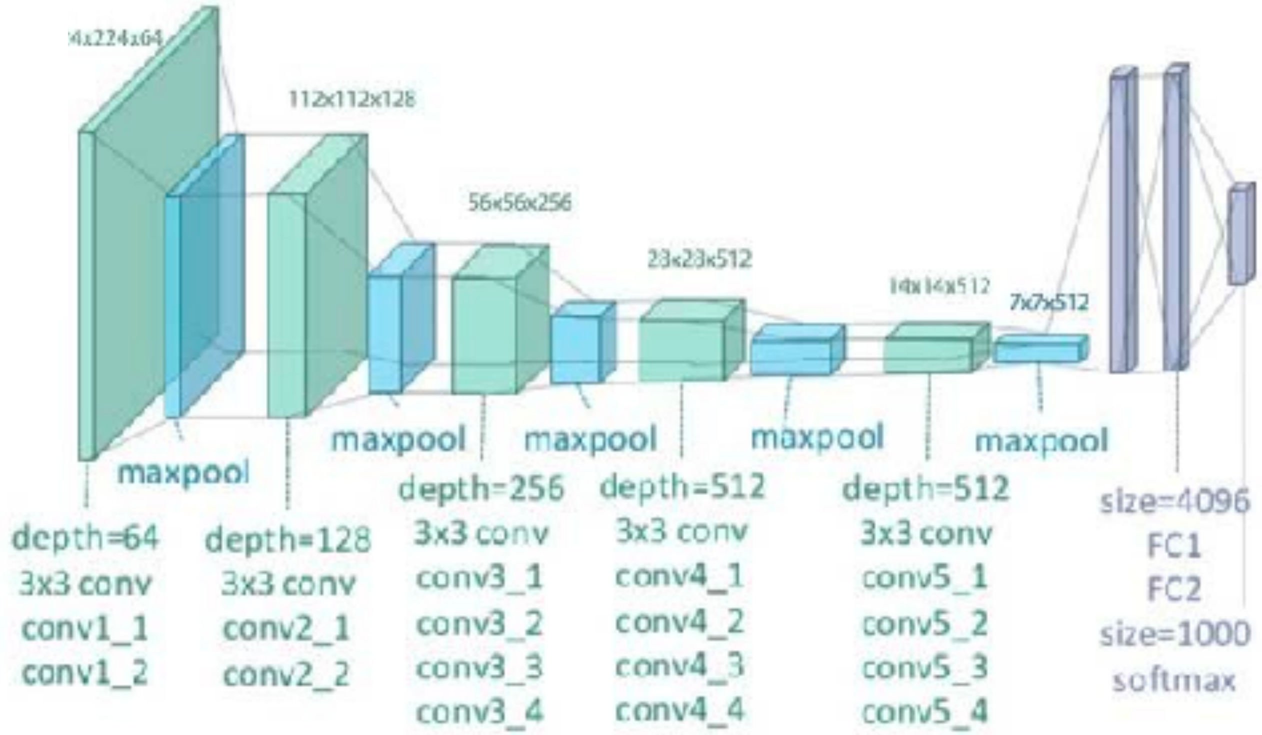
```
img = load("bicycle.jpg")
@show classify(vgg,img)
img
```

```
classify(vgg, img) = "mountain bike, all-terrain bike, off-roader"
```



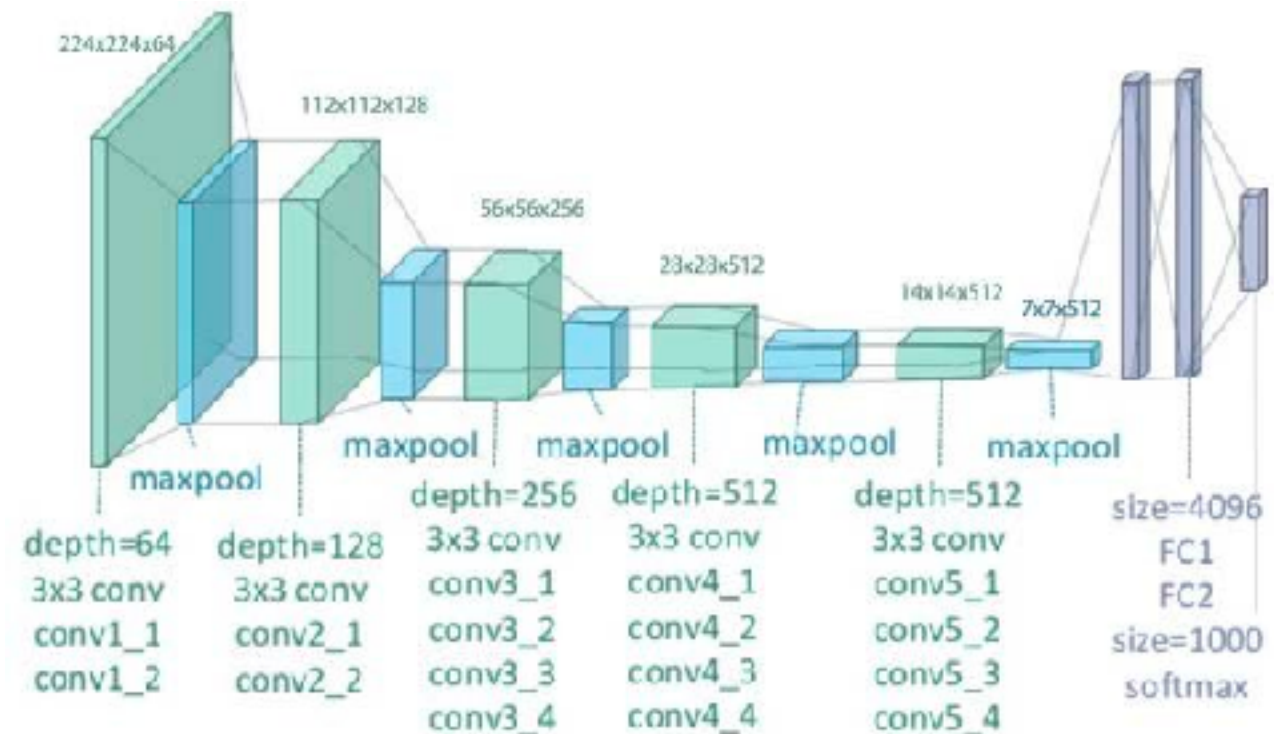
```
img = load("appleFruit.jpg")
@show classify(vgg,img)
img
```

```
classify(vgg, img) = "Granny Smith"
```



```
img = load("baby.jpg")
@show classify(vgg,img)
img
```

```
classify(vgg, img) = "diaper, nappy, napkin"
```



Data Science

Artificial Intelligence

Machine Learning

Deep Learning

Computer Science

Statistics

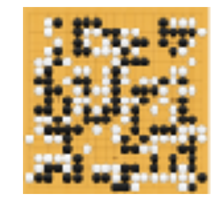
Statistical Learning

Neuroscience

Data Mining

Timeline of weak (narrow) AI

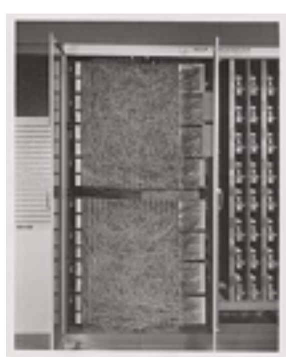
Weak AI `Explosion’ and Deep Learning



~2010 Machine Learning Renaissance begins

~2000 Revival of machine learning

1980’s Reinforcement Learning



1970 Reverse Mode Automatic Differentiation

1960’s Neural Networks

1950 Turing Test Proposed

Timeline of strong AI (Artificial General Intelligence)

2010+ Weak AI Hype…

1970’s-80’s Expert Systems

1970's Thoughts and definitions about AGI

1950 Turing Test Proposed

Timeline of Deep Learning (prehistory)

"Deep Learning" is a thing!
Deep Learning = AI?

2012 (**AlexNet**): ImageNet Classification with Deep Convolutional Neural Networks, by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton.

1998: (**Convolutional nets**): Gradient based learning applied to document recognition, by Yann Lecun, Leon Bottou, Yoshua Bengio and Patrick Haffne.

1982 (**Hopfield nets**): Neural networks and physical systems with emergent collective computational abilities, by John Hopfield.

1958 (**Perceptron**): The perceptron: a probabilistic model for information storage and organization in the brain, by Frank Rosenblatt.

Current Deep Learning

2017 (**Transformers**): Attention is all you need, by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser and Illia Polosukhin.

2016 (**Deep RL vs. Go**): Mastering the game of Go with deep neural networks and tree search, by David Silver, Aja Huang, Chris Maddison and others.

2015 (**ResNet**): Deep residual learning for image recognition, by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun.

2015 (**Inception**): Going deeper with convolutions, by Christian Szegedy et. al.

2015 (**VGG**): Very deep convolutional networks for large-scale image recognition, by Karen Simonyan and Andrew Zisserman.

2014: (**GAN**): Generative adversarial nets, by Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville and Yoshua Bengio.

2014: (**ADAM**): Adam: A method for stochastic optimization, by Diederik Kingma and Jimmy Ba.

2013: (**Deep RL**): Playing Atari with Deep Reinforcement Learning by Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller.

2013: (**Inner layers**): Visualizing and Understanding Convolutional Networks, by Matthew Zeiler and Rob Fergus

2012 (**AlexNet**): ImageNet Classification with Deep Convolutional Neural Networks, by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton.

The main activities of machine learning

Regression

- Supervised learning (Y = labels, X = features)

Classification

- Unsupervised learning (only X)
- Semi-supervised learning (hybrid)
- Reinforcement Learning (a time component)
- Generative modeling (generating new X's)
- Dealing with sequence data

Part 2: The ML workflow and common “practice” data sets

The MNIST digits dataset

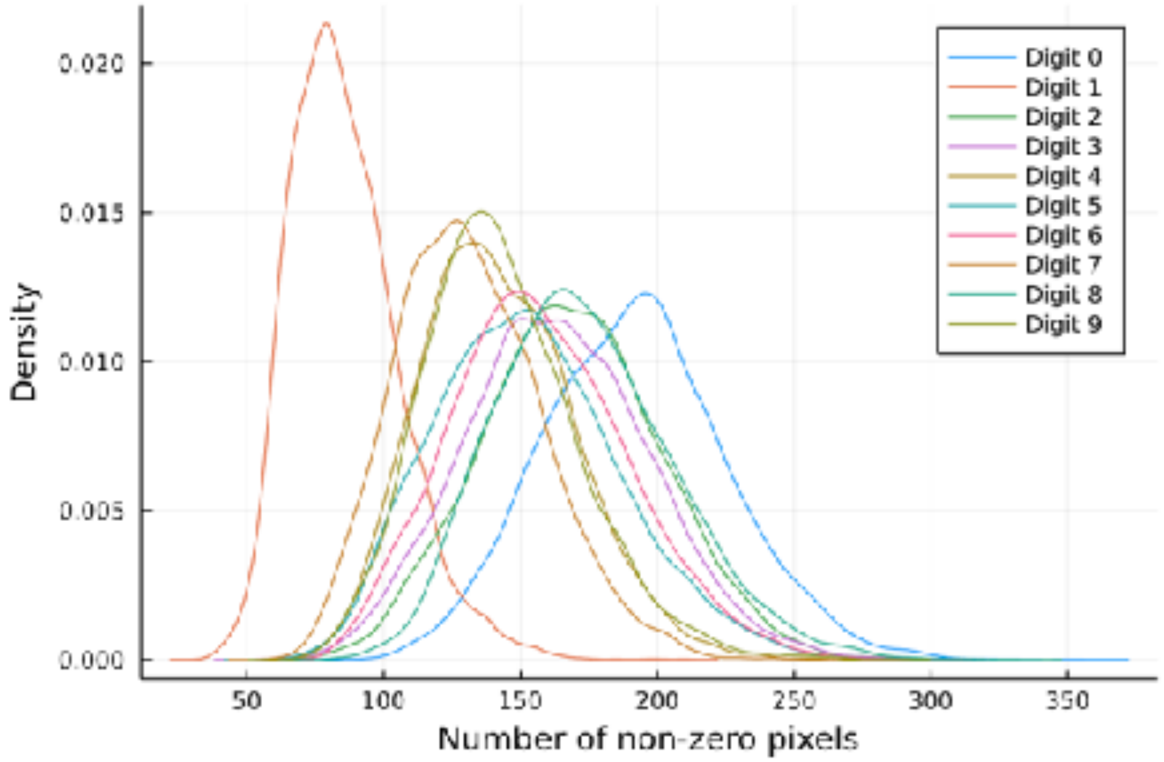
```
using Flux
imgs, labels = Flux.Data.MNIST.images(), Flux.Data.MNIST.labels();
@show length(imgs)
@show size(imgs[1])
@show labels[1:8]
imgs[1:8]
```

```
length(imgs) = 60000
size(imgs[1]) = (28, 28)
labels[1:8] = [5, 0, 4, 1, 9, 2, 1, 3]
```

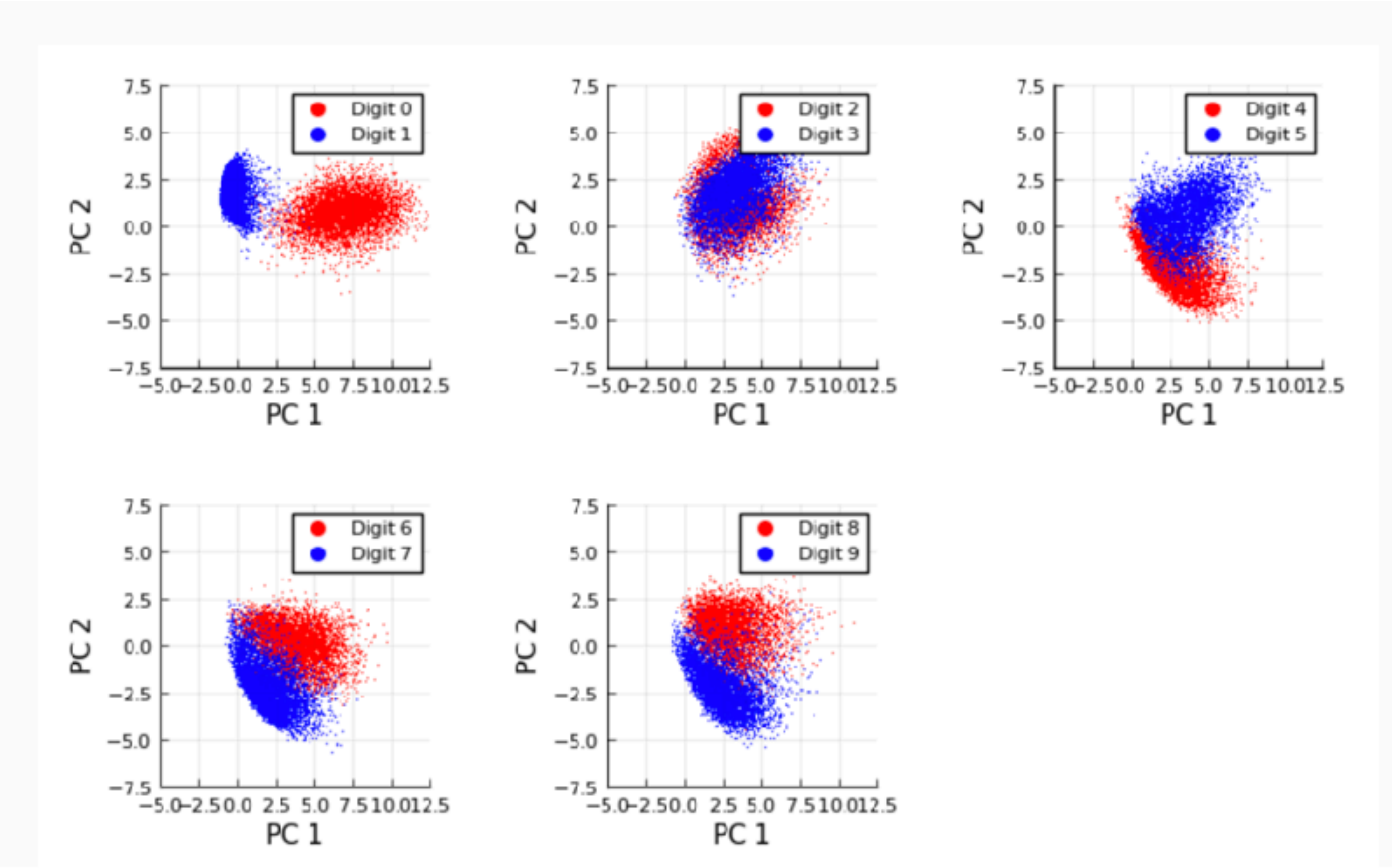
Basic EDA for MNIST

```
#Basic Exploratory Data Analysis (EDA) for MNIST
using Statistics, StatsPlots, Plots

x = hcat([vcat(float.(im)...) for im in imgs]...)

d, n = size(x)
@show (d,n)

onMeanIntensity = mean(filter((u)->u>0,x))
@show onMeanIntensity

prop0 = sum(x .== 0)/(d*n)
@show prop0

prop1 = sum(x .== 1)/(d*n)
@show prop1

print("Label counts: ", [sum(labels .== k) for k in 0:9])

p = plot()
for k in 0:9
    onPixels = [ sum(x[:,i] .> 0) for i in (1:n)[labels .== k] ]
    p = density!(onPixels, label = "Digit $(k)")
end
plot(p,xlabel="Number of non-zero pixels", ylabel = "Density")
```

```
(d, n) = (784, 60000)
onMeanIntensity = Gray{Float32}(0.6833625f0)
prop0 = 0.8087977040816327
prop1 = 0.006681164965986395
Label counts: [5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949]
```

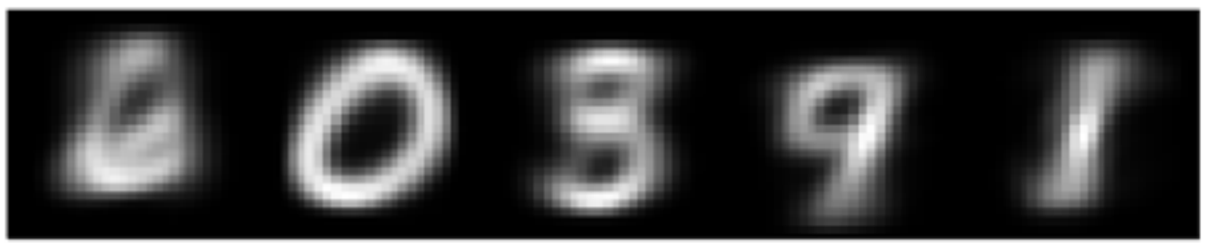


Breaking MNIST up via PCA (2 components)



The centroids of k-means

k=3



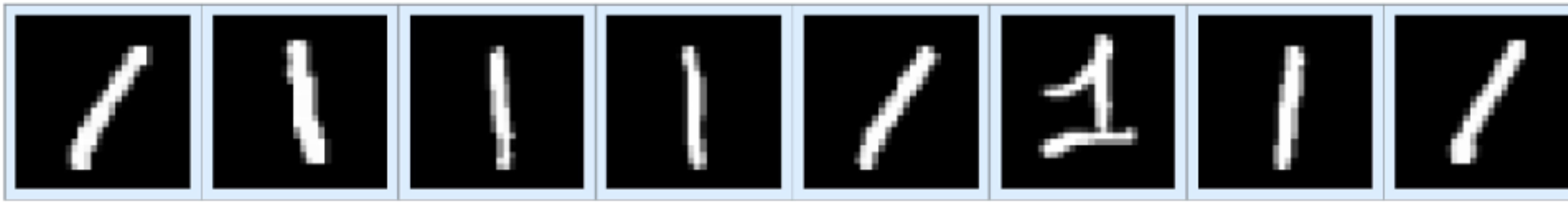
k=5

k=10

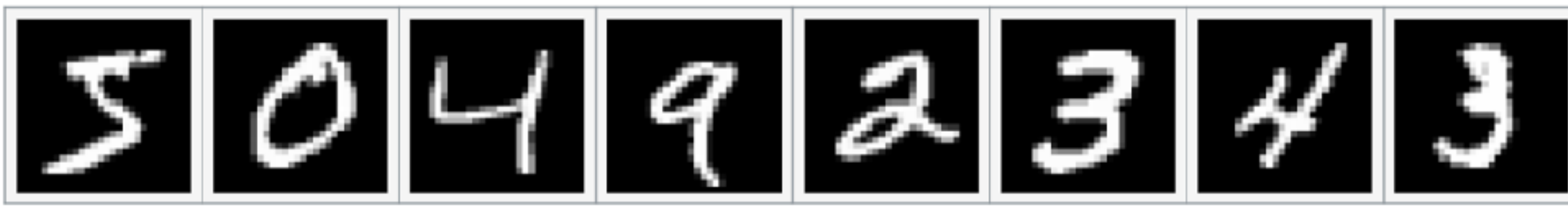
k=15

A toy binary classifier digit 1, or not

```
imgs[labels .== 1][1:8]
```



```
imgs[labels .!= 1][1:8]
```



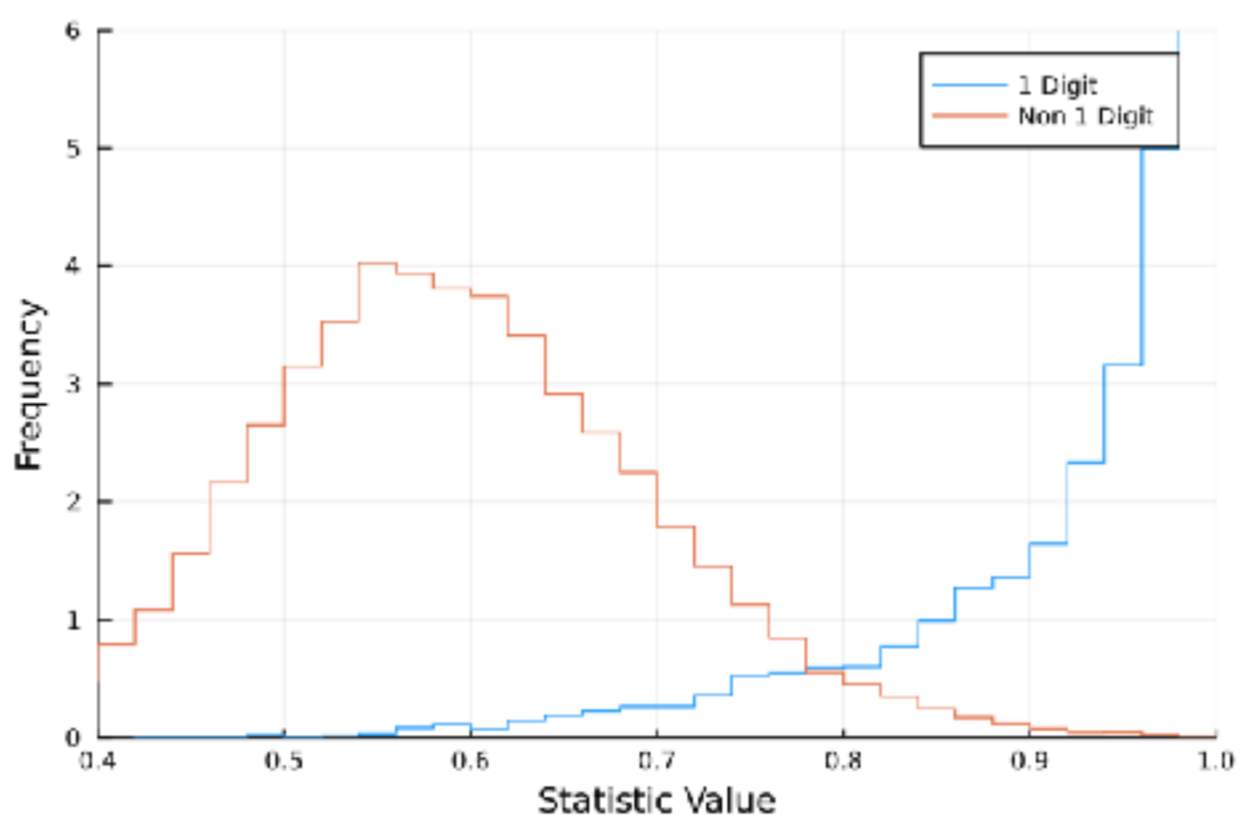
Define a statistic (specific to the digit 1) : Count the proportion of intensity around the peak per row:

$$\chi(x) = \frac{\sum_{i=1}^{28} \sum_{k=-2}^{2} x_{i,m(x,i)+k}}{\sum_{i=1}^{28} \sum_{j=1}^{28} x_{i,j}}, \quad \text{where} \quad m(x,i) = \operatorname{argmax}_{j=1,\ldots,28} x_{i,j}$$

A classifier:
$$\hat{f}_\theta(x) = \begin{cases} -1 & \chi(x) \le \theta, \\ +1 & \chi(x) > \theta\,. \end{cases}$$

A toy binary classifier digit 1, or not

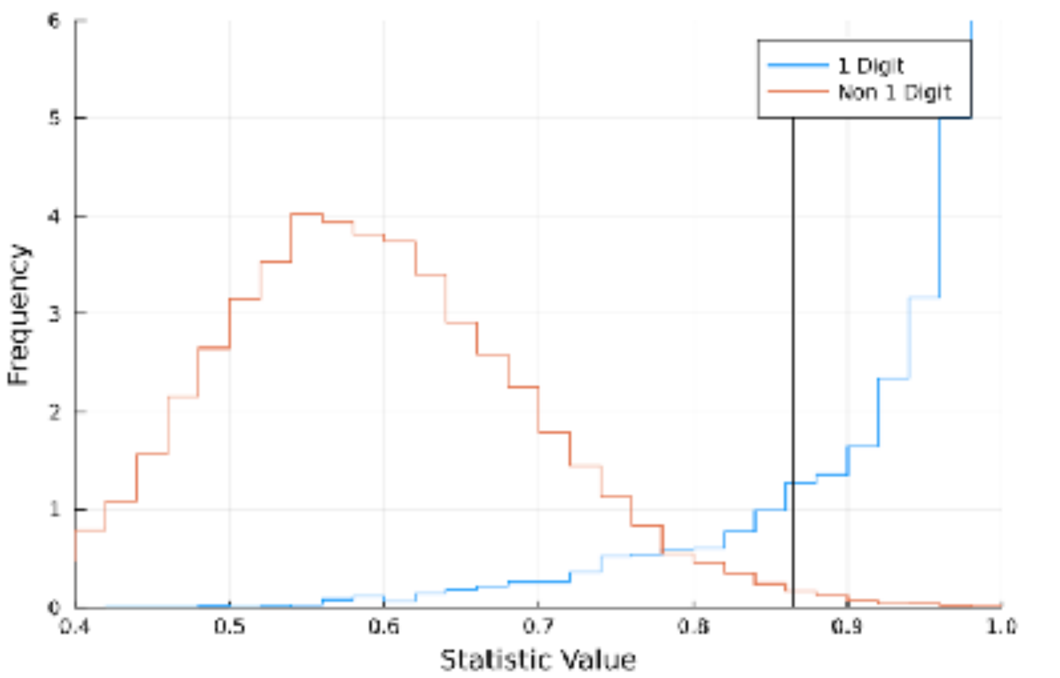
$$\chi(x) = \frac{\sum_{i=1}^{28} \sum_{k=-2}^{2} x_{i,m(x,i)+k}}{\sum_{i=1}^{28} \sum_{j=1}^{28} x_{i,j}}, \quad \text{where} \quad m(x,i) = \operatorname{argmax}_{j=1,\ldots,28} x_{i,j}$$



$$\hat{f}_\theta(x) = \begin{cases} -1 & \chi(x) \le \theta, \\ +1 & \chi(x) > \theta. \end{cases}$$

How to choose the threshold θ?

Say we chose it at $\theta = 0.865$



		Decision	
		Decide -1 (8,916)	Decide $+1$ (1,084)
Reality	Label is -1 (8,865)	True negative (8,781)	False positive (84)
	Label is $+1$ (1,035)	False negative (135)	True Positive (1000)

Say we chose it at $\theta = 0.865$

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Decision</td></tr>
<tr><td>Decide -1
(8,916)</td><td>Decide $+1$
(1,084)</td></tr>
<tr><td rowspan="2">Reality</td><td>Label is -1
(8,865)</td><td>True negative
(8,781)</td><td>False positive
(84)</td></tr>
<tr><td>Label is $+1$
(1,035)</td><td>False negative
(135)</td><td>True Positive
(1000)</td></tr>
</table>

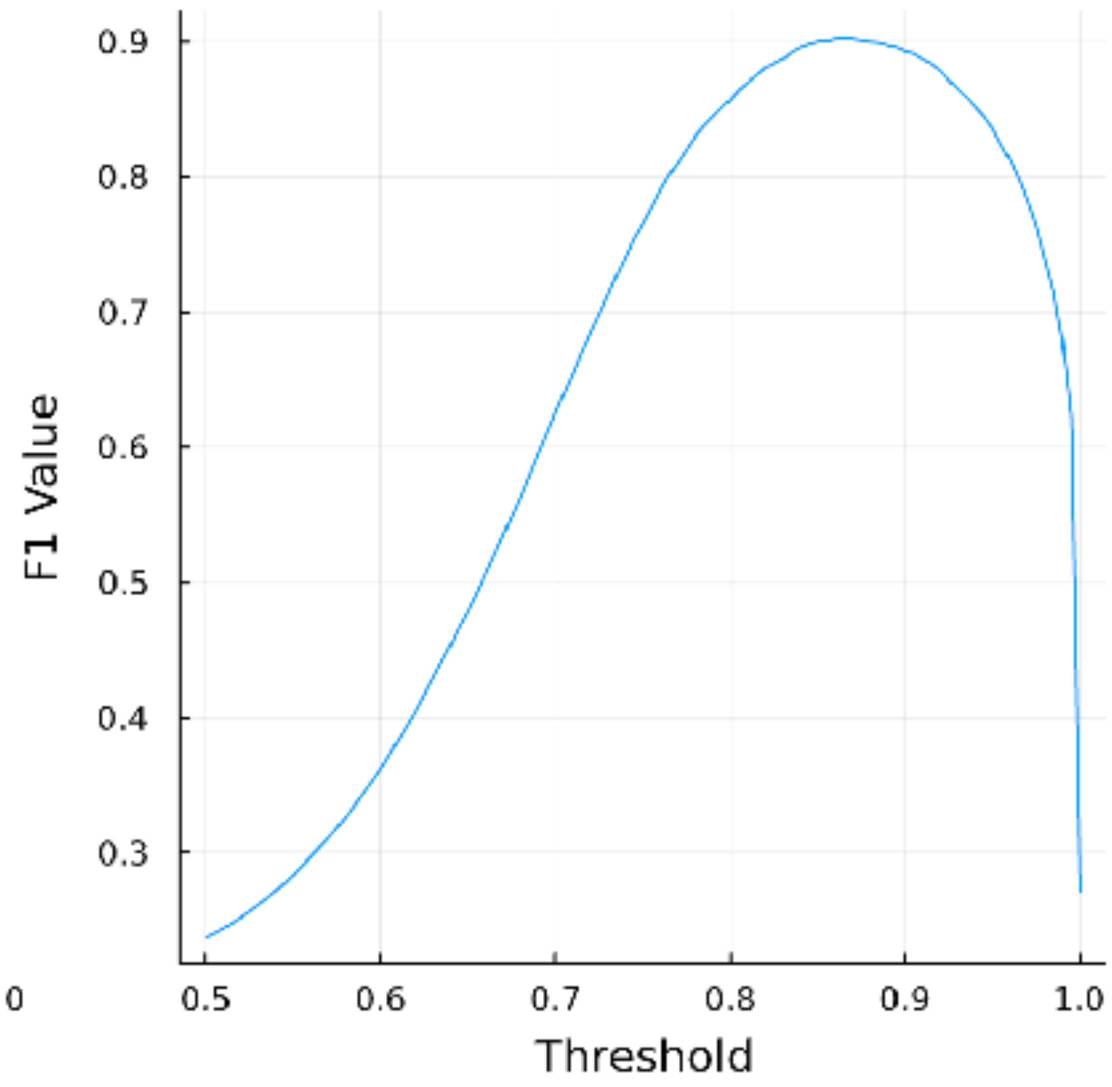
$$\text{Precision} = \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false positive}|}, \qquad \text{Recall} = \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false negative}|}.$$

$$\text{Precision} = \frac{1000}{1000 + 84} = 92.25\%, \qquad \text{Recall} = \frac{1000}{1000 + 135} = 88.11\%.$$

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 90.13\,\%$$

How we chose $\theta = 0.865$

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = 2\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 90.13\,\%$$



Why not just accuracy?

$$\text{accuracy} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}\left\{\hat{f}\left(x^{(i)}\right) = y_i\right\}$$

What can be a problem?

Recap:

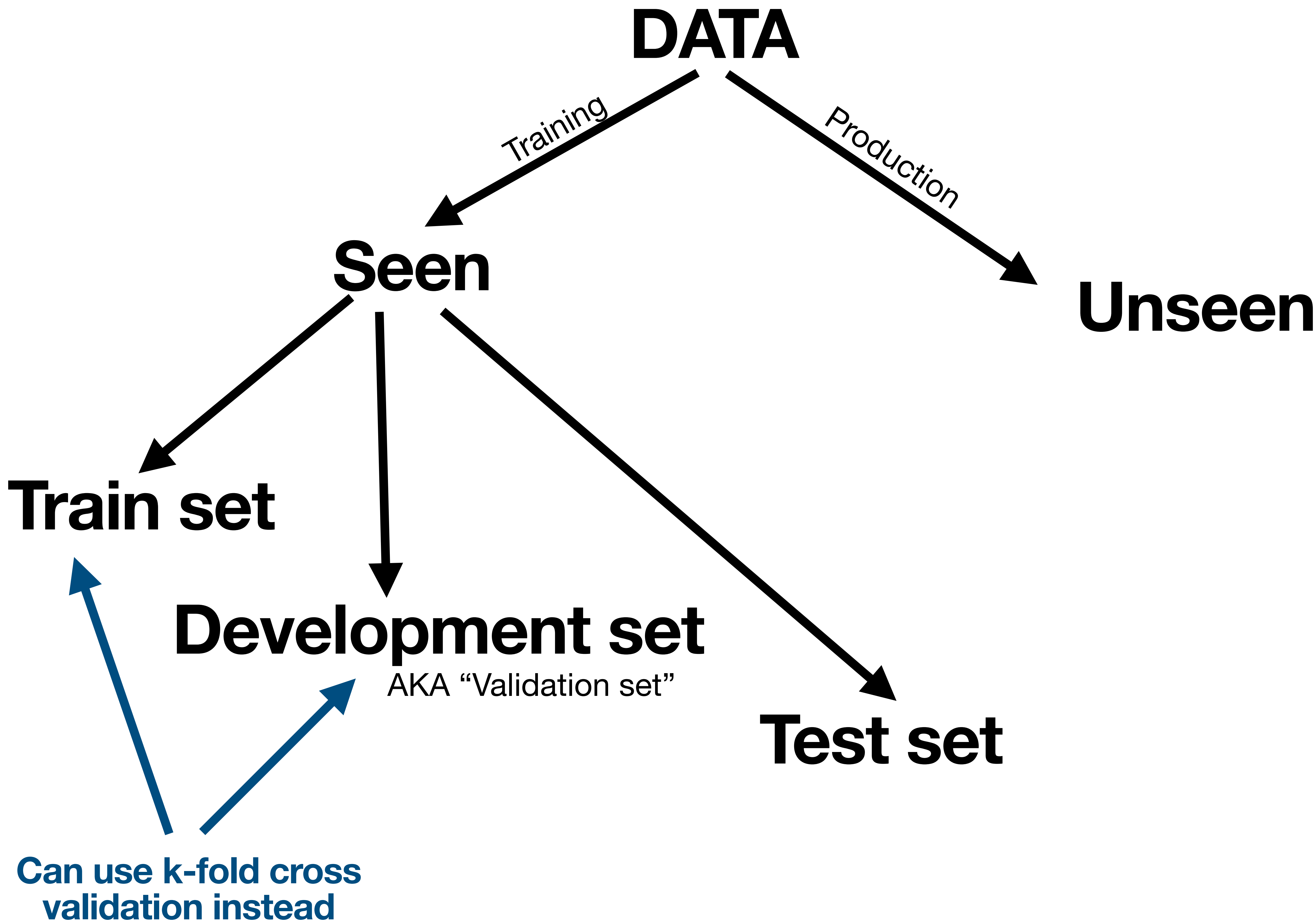
We “learned” the parameter θ

We had some hyper-parameters too:

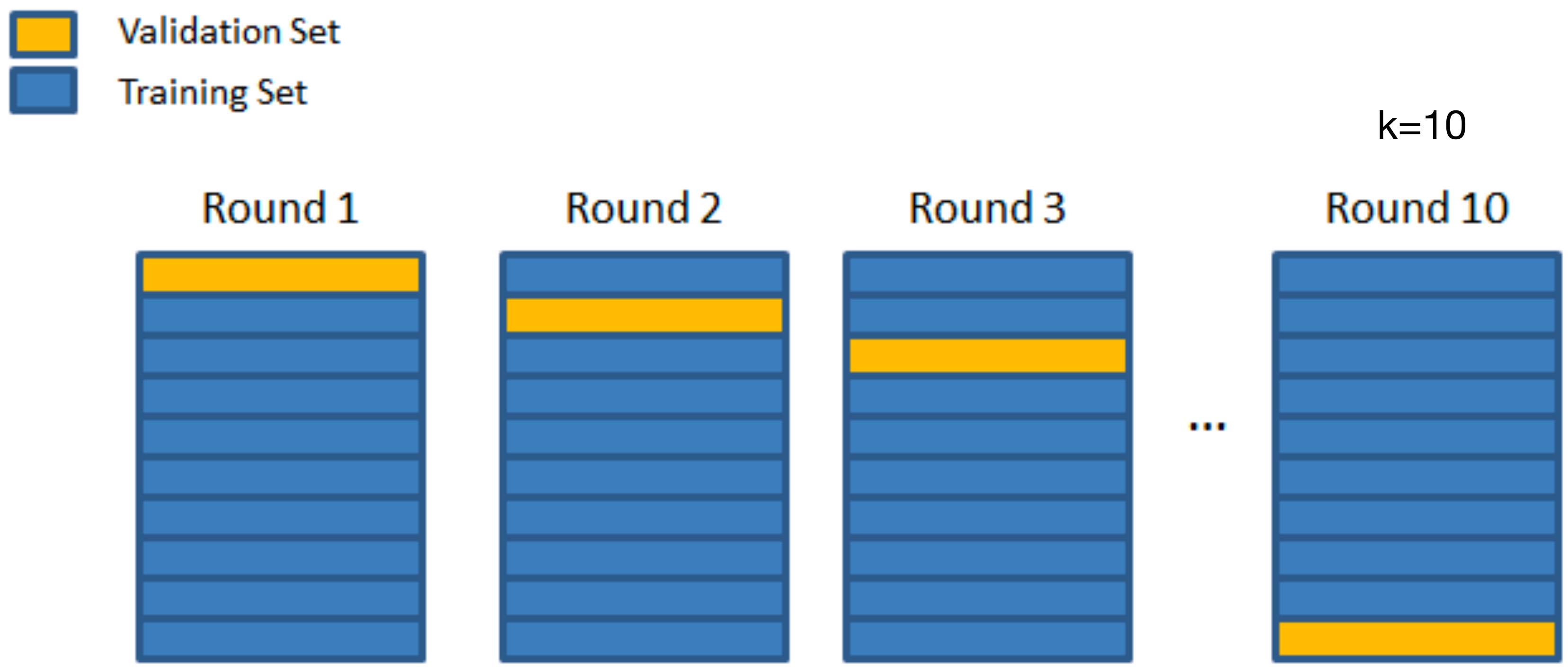
$$\chi(x) = \frac{\sum_{i=1}^{28} \sum_{k=-2}^{2} x_{i,m(x,i)+k}}{\sum_{i=1}^{28} \sum_{j=1}^{28} x_{i,j}}, \quad \text{where} \quad m(x,i) = \operatorname{argmax}_{j=1,\ldots,28} x_{i,j}$$

How do we choose hyper-parameters?

How do we test our classifier?

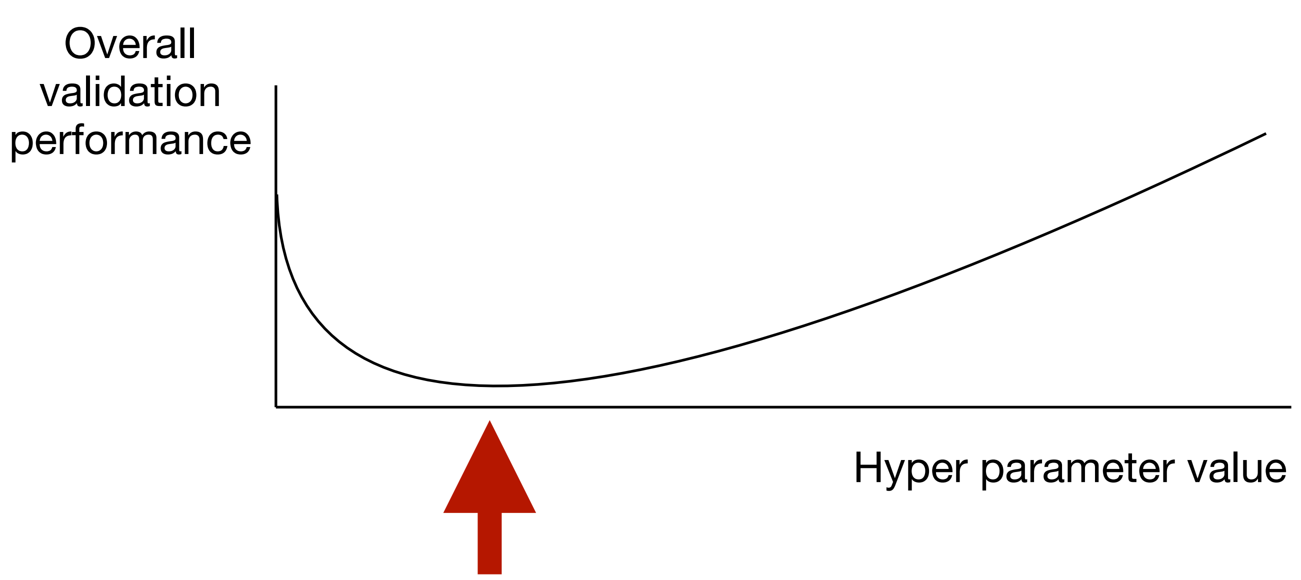

DATA
Training
Production
Seen
Unseen
Train set
Development set
AKA “Validation set”
Test set
Can use k-fold cross
validation instead

K-fold cross validation



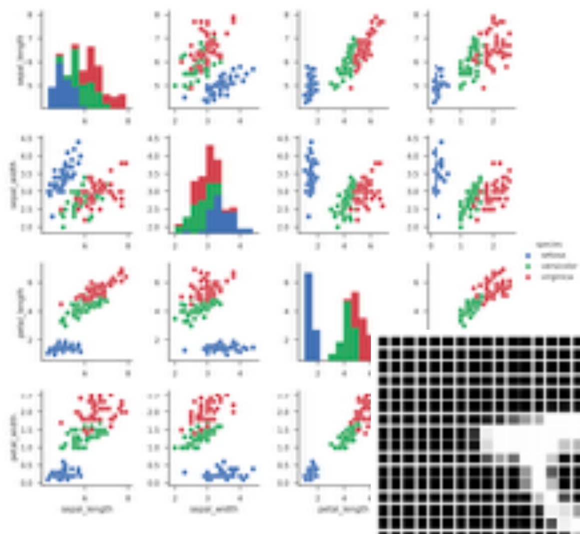
Validation performance: 87.3% 92.5% 91.2% 89.6%

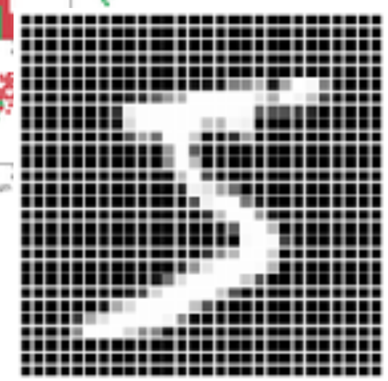
Overall validation performance: mean(87.3, 92.5, 91.2, …. ,89.6)

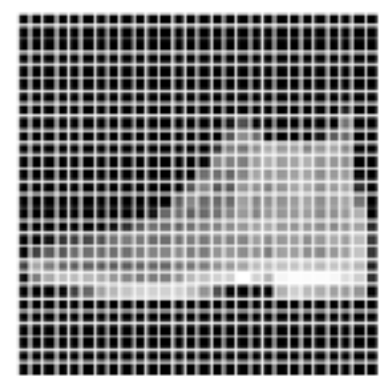


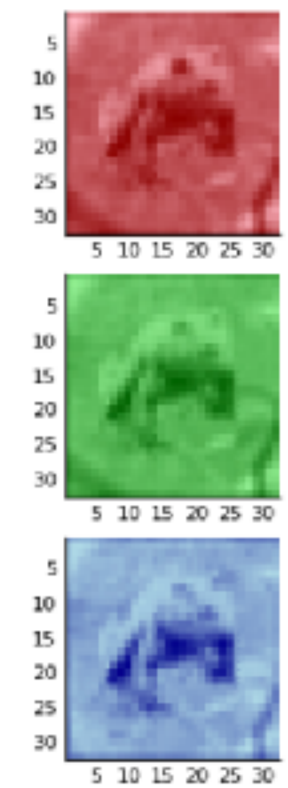
Some more popular "toy" datasets

1. Iris Dataset
2. MNIST
3. Fashion MNIST
4. CIFAR-10
5. ImageNet
6. Twitter Sentiment Analysis







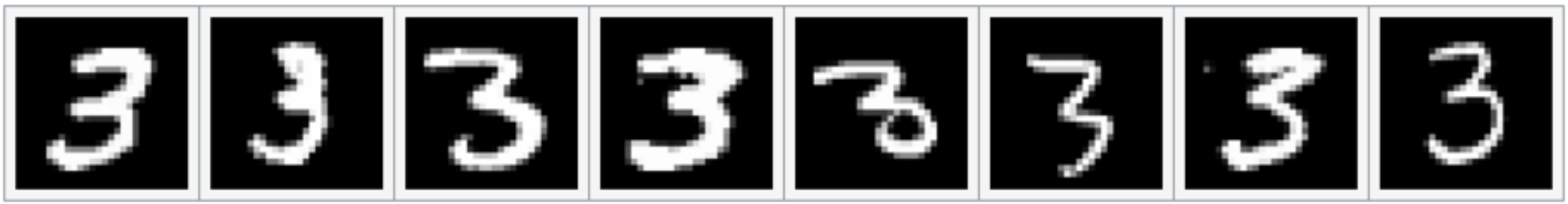


Part 3: Some tools you know - used for ML

A linear (binary) classifier

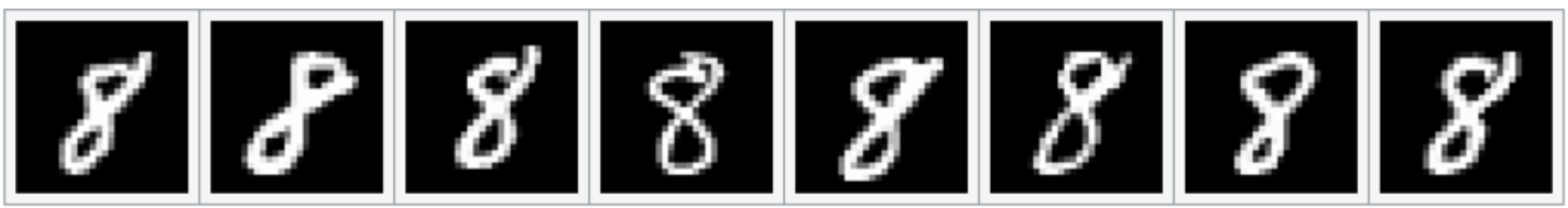
Positive (+1)

```
imgs[labels .== 3][1:8]
```



Negative (-1)

```
imgs[labels .== 8][1:8]
```



A linear (binary) classifier

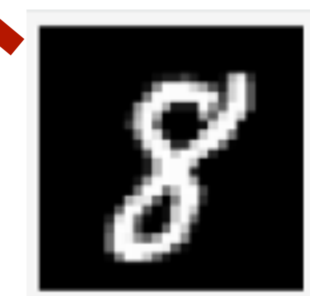
$+1$

Vectorize

$$y = \beta_0 + \beta^T x$$

Vectorize

-1



Minimize: $\text{Loss}(x, y) = \sum_{i=1}^{m} (y_i - \beta_0 - \beta^T x_i)^2$

Classifier: $\hat{f}(x) = \text{sign}(\hat{\beta}_0 + \hat{\beta}^T x)$

A linear (binary) classifier

Minimize: $\text{Loss}(x, y) = \sum_{i=1}^{m} (y_i - \beta_0 - \beta^T x_i)^2 = ||y - A\beta||^2$

Sometimes

Option 1: $\hat{\beta} = A^\dagger y$ $\quad A^\dagger = (A^T A)^{-1} A^T$

Option 2: $\hat{\beta}(0), \hat{\beta}(1), \hat{\beta}(2), \hat{\beta}(3), \ldots$ With (some form of) gradient descent

$$\hat{\beta}(t+1) = \hat{\beta}(t) - \eta \nabla L\left(\hat{\beta}(t)\right)$$

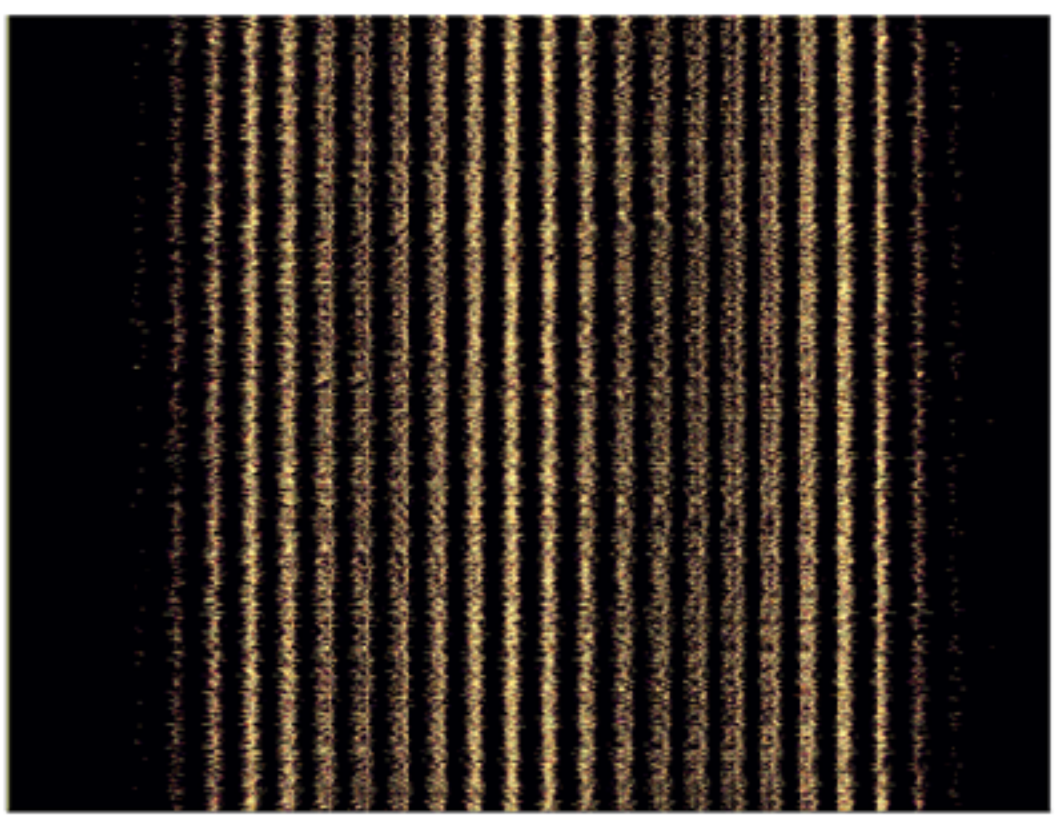
$$\nabla L\left(\beta\right) = 2A^T(A\beta - y)$$

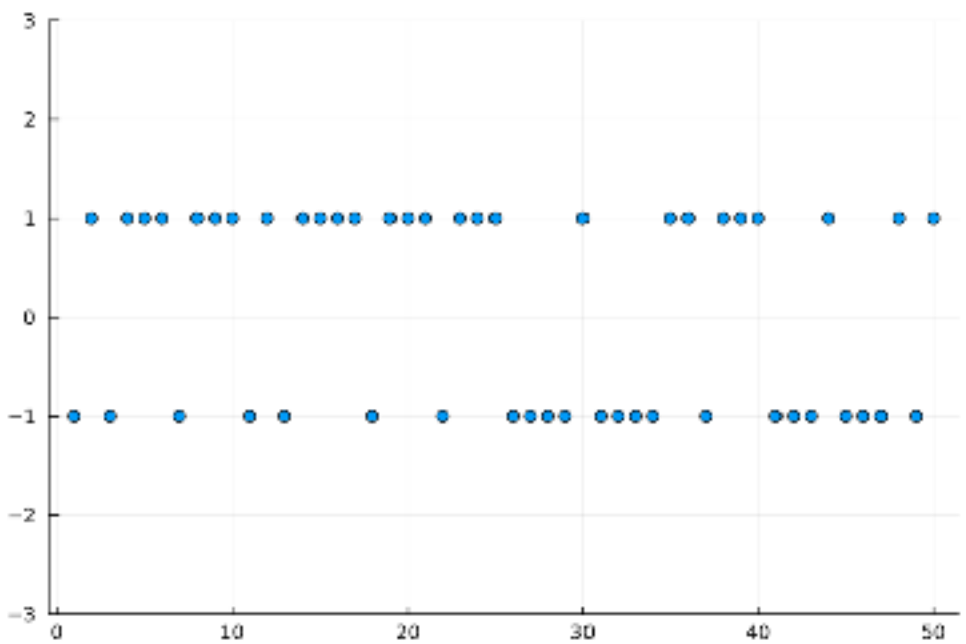
Let's do it!

Sometimes

Option 1: $\hat{\beta} = A^{\dagger} y$ $\qquad A^{\dagger} = (A^T A)^{-1} A^T$

Option 2: $\hat{\beta}(t+1) = \hat{\beta}(t) - \eta 2 A^T (A\hat{\beta}(t) - y)$

$A =$ 

$y =$ 

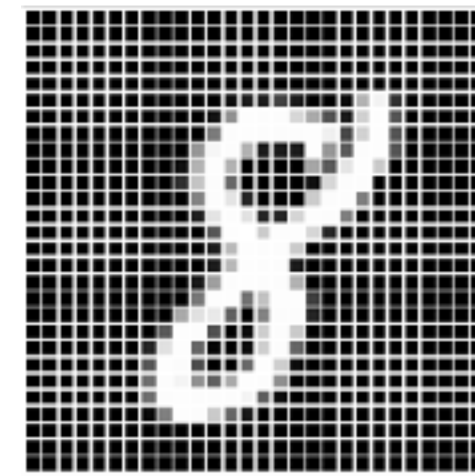
$\ldots$

Activity A: MNIST digit classification with least squares

https://github.com/ajayhemanth/Machine-Learning-Workshop/blob/main/Activity_A_Linear_Classifier.ipynb

Multi-class: One vs. rest

```julia
using Flux.Data.MNIST, PyPlot, LinearAlgebra
using Flux: onehotbatch

imgs   = MNIST.images()
labels = MNIST.labels()
nTrain = length(imgs)

trainData = vcat([hcat(float.(imgs[i])...) for i in 1:nTrain]...);
trainLabels = labels[1:nTrain];

testImgs = MNIST.images(:test)
testLabels = MNIST.labels(:test)
nTest = length(testImgs)

testData = vcat([hcat(float.(testImgs[i])...) for i in 1:nTest]...);

A = [ones(nTrain) trainData];
Adag = pinv(A);
tfPM(x) = x ? +1 : -1
yDat(k) = tfPM.(onehotbatch(trainLabels,0:9)'[:,k+1])
bets = [Adag*yDat(k) for k in 0:9];

classify(input) = findmax([([1 ; input])'*bets[k] for k in 1:10])[2]-1

predictions = [classify(testData[k,:]) for k in 1:nTest]
confusionMatrix = [sum((predictions .== i) .& (testLabels .== j))
                        for i in 0:9, j in 0:9]
accuracy = 100*sum(diag(confusionMatrix))/nTest

println("Accuracy: ", accuracy,"%")
confusionMatrix
```



```
Accuracy: 86.03%

10×10 Array{Int64,2}:
 944     0   18    4    0   23   18    5   14   15
   0  1107   54   17   22   18   10   40   46   11
   1     2  813   23    6    3    9   16   11    2
   2     2   26  880    1   72    0    6   30   17
   2     3   15    5  881   24   22   26   27   80
   7     1    0   17    5  659   17    0   40    1
  14     5   42    9   10   23  875    1   15    1
   2     1   22   21    2   14    0  884   12   77
   7    14   37   22   11   39    7    0  759    4
   1     0    5   12   44   17    0   50   20  801
```

Basic statistics
(least squares/regression)

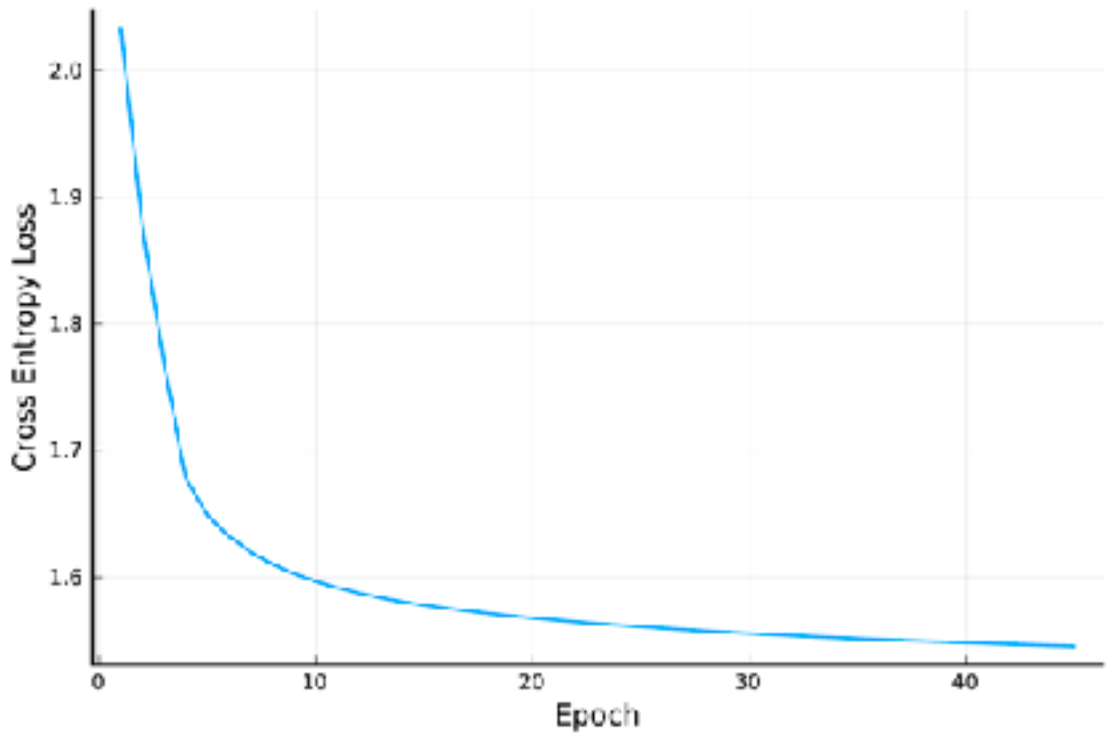
Logistic softmax regression

$$\sigma(u) = \frac{1}{1+e^{-u}} = \frac{e^u}{e^u+1}$$

$$\hat{y}(\tilde{x}) = \text{argmax}_{\ell=0,\ldots,9}\ \sigma(w^{(\ell)} \cdot \tilde{x} + b^{(\ell)}).$$

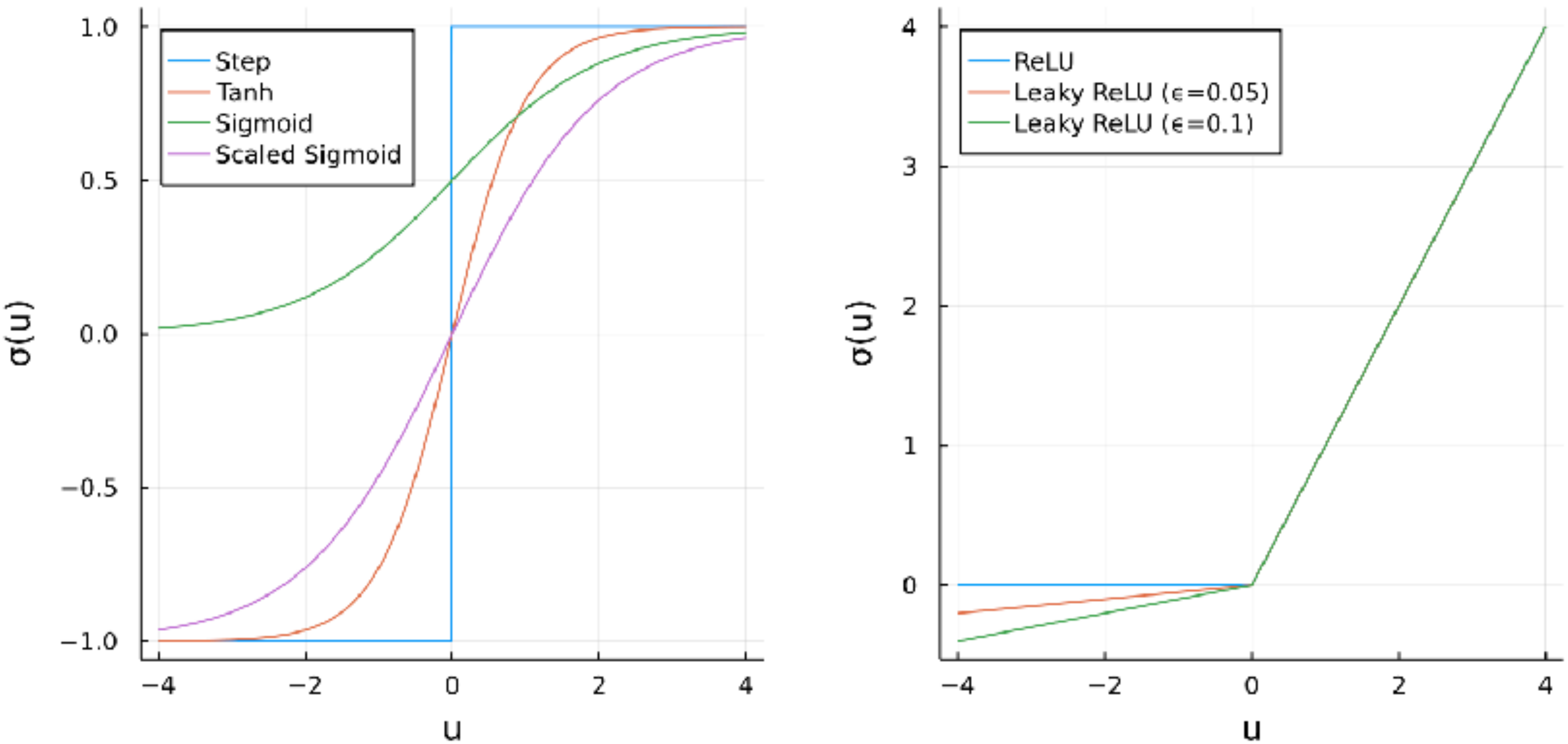
$$s(z) = \frac{1}{\sum_{j=1}^{K} e^{z_j}} \begin{bmatrix} e^{z_1} & e^{z_2} & \ldots & e^{z_K} \end{bmatrix}^T$$

$$\hat{y}(\tilde{x}) = \text{argmax}_{\ell=0,\ldots,9}\ s_\ell(W\tilde{x} + b)$$

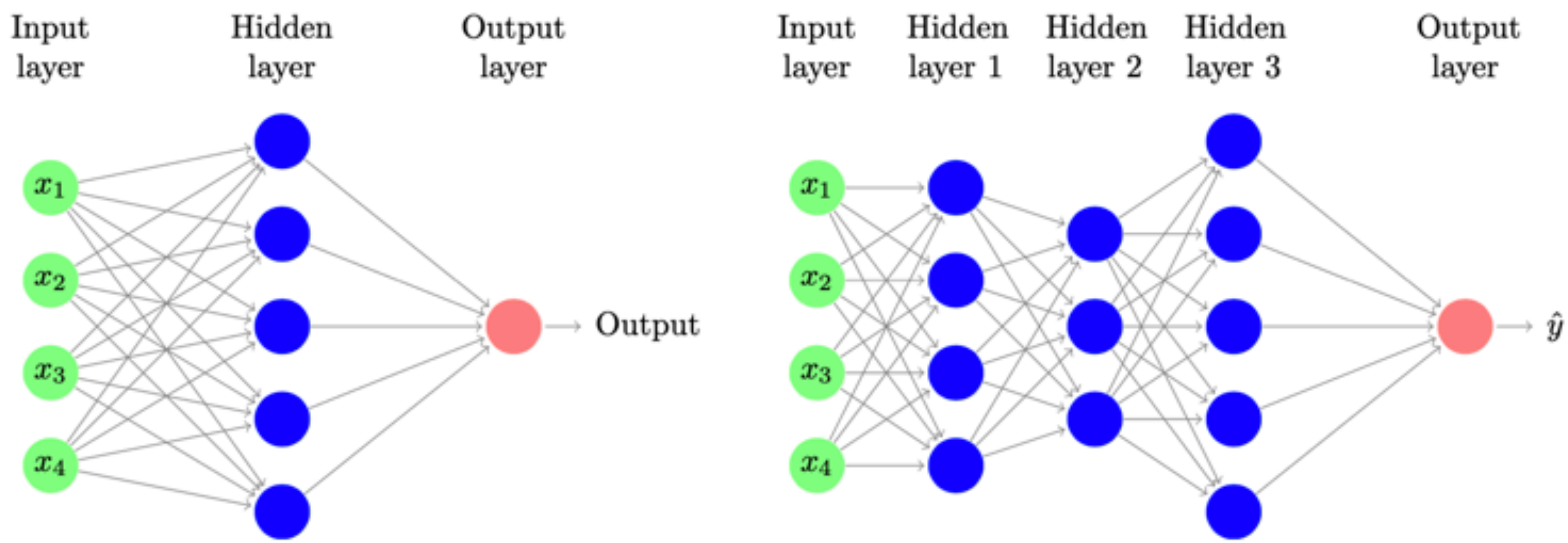


Cross Entropy Loss: $-\sum_{i=1}^{n} \log(\hat{\mathcal{Y}}_{y_i+1})$

Some common activation functions



Adding activations and layers



(a) One-layer hidden Network　　(b) Deep Neural Networks

$$f_\theta(x) = f^{[L]}_{\theta^{[L]}}(f^{[L-1]}_{\theta^{[L-1]}}(\ldots(f^{[1]}_{\theta^{[1]}}(x))\ldots))$$

$$a^{[\ell-1]} \xrightarrow{\text{Affine Transformation}} z^{[\ell]} := W^{[\ell]}a^{[\ell-1]} + b^{[\ell]} \xrightarrow{\text{Activation}} a^{[\ell]} := S^{[\ell]}(z^{[\ell]})$$

$f^{(\ell)}_{\theta^{[\ell]}}$

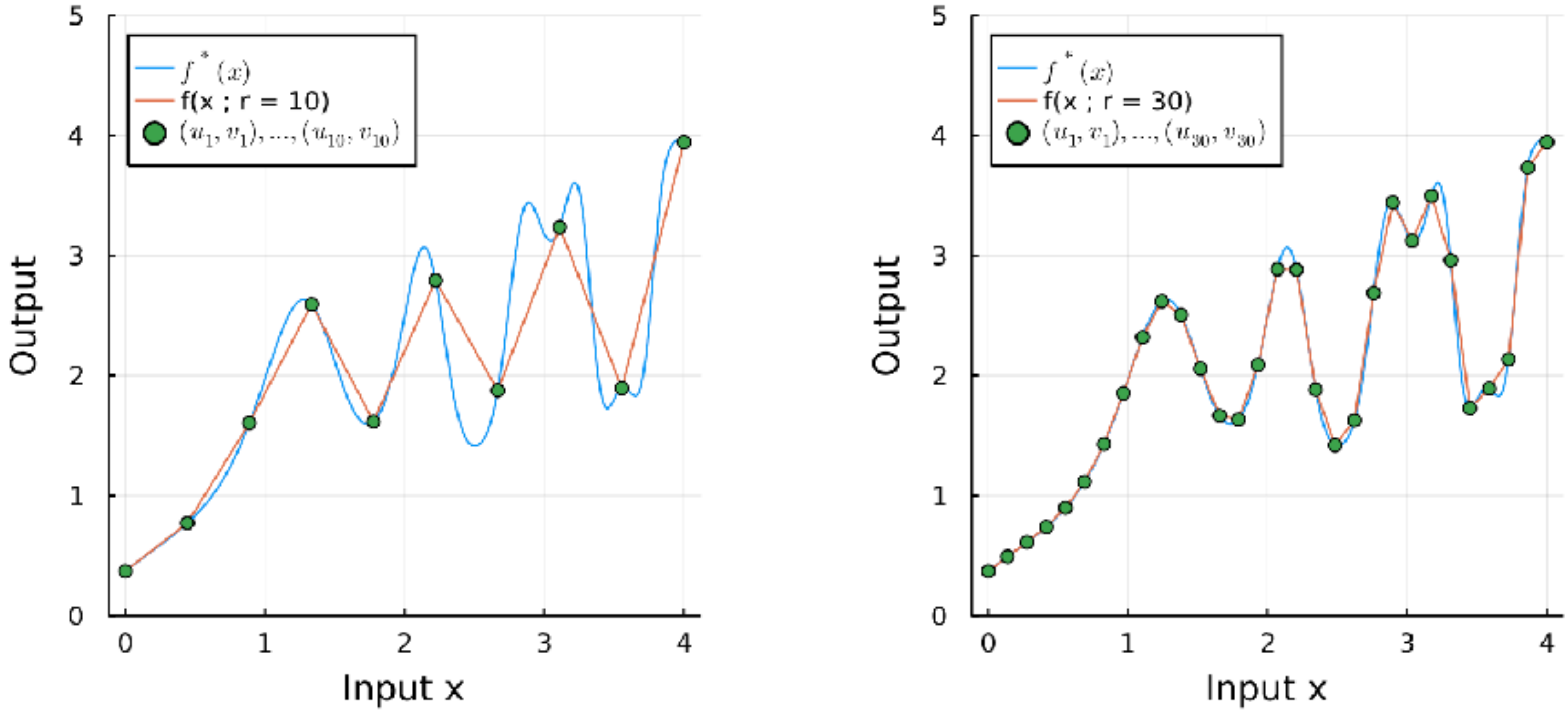
Representing a neural network with equations

$$S^{[\ell]}(z) = \left[\sigma^{[\ell]}\left(z_1\right), \ldots, \sigma^{[\ell]}\left(z_{N_\ell}\right)\right]^T$$

$$f_\theta(x) = \underbrace{S^{[2]}\left(\overbrace{W^{[2]}\underbrace{S^{[1]}(\overbrace{W^{[1]}x + b^{[1]}}^{z^{[1]}})}_{a^{[1]}} + b^{[2]}}^{z^{[2]}}\right)}_{a^{[2]}},$$

$$\begin{array}{c}\text{Affine} \\ \text{Transformation}\end{array} : \begin{cases} z_1^{[\ell]} & = & w_1^{[\ell]\top} a^{[\ell-1]} + b_1^{[\ell]} \\ z_2^{[\ell]} & = & w_2^{[\ell]\top} a^{[\ell-1]} + b_2^{[\ell]} \\ & \vdots & \\ z_{N_\ell}^{[\ell]} & = & w_{N_\ell}^{[\ell]\top} a^{[\ell-1]} + b_{N_\ell}^{[\ell]} \end{cases} \Rightarrow \begin{array}{c}\text{Activation} \\ \text{Step}\end{array} : \begin{cases} a_1^{[\ell]} & = & \sigma\left(z_1^{[\ell]}\right) \\ a_2^{[\ell]} & = & \sigma\left(z_2^{[\ell]}\right) \\ & \vdots & \\ a_{N_\ell}^{[\ell]} & = & \sigma\left(z_{N_\ell}^{[\ell]}\right) \end{cases}$$

Neural Networks are Expressive



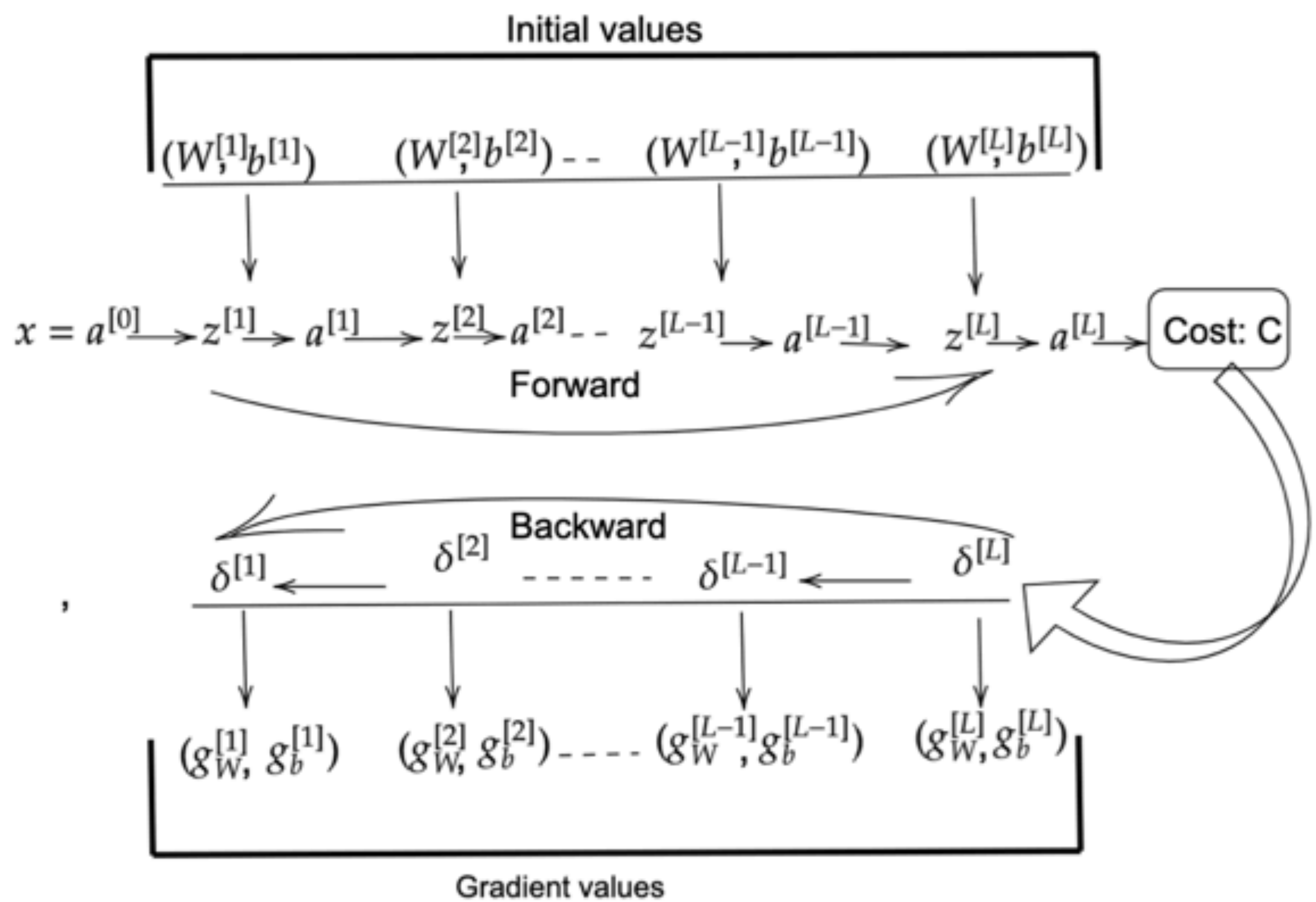
Function approximations with a neural network with one hidden layer

Activity B: Tensorflow Playground

https://playground.tensorflow.org/

The Devil is in the details: Backpropagation

$$\delta^{[\ell]} := \frac{\partial C(a^{[L]}, y\,;\,\theta)}{\partial z^{[\ell]}}, \qquad \ell = 1, \ldots, L,$$



Activity C: Dense Neural Nets

https://github.com/ajayhemanth/Machine-Learning-Workshop/blob/main/Activity_C_Dense_Neural_Networks.ipynb

K Keras

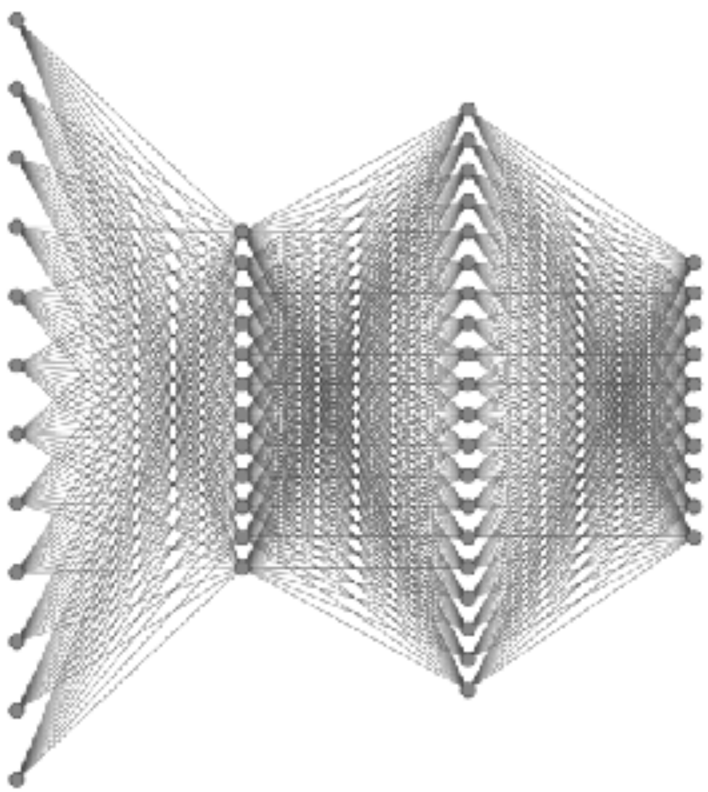
Simple. Flexible. Powerful.

A walk in the random forest

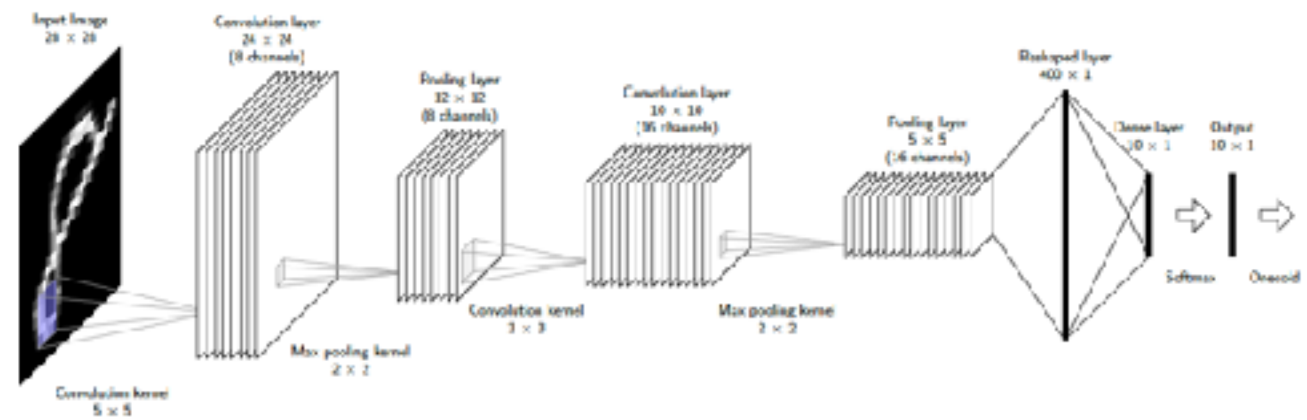
Activity D: Random Forests

https://github.com/ajayhemanth/Machine-Learning-Workshop/blob/main/Activity_D_RandomForests_with_H2O.ai.ipynb

Going Convolutional



Dense



Convolutional

Activity E: Conv Nets

https://github.com/ajayhemanth/Machine-Learning-Workshop/blob/main/Activity_E_ConvolutionalNets.ipynb

Closing