National Institute of Allergy and Infectious Diseases

ACE Uganda – Makerere University

Intro to Deep Learning with Tensorflow and Keras

GitHub (Slides and Code): https://github.com/dan-veltri/ace-intro-to-deep-learning

December 10th, 2021

NIAID

National Institute of Allergy and Infectious Diseases

Daniel Veltri, Ph.D.

For Today's Training

***IF** using the HPC cluster and try to activate the conda environment (further instructions are on the GitHub page):*

```
conda activate /home/bcbb_teaching_files/intro_deep_learning/envs
```

Clone the repo on this GitHub to your local home directory to have local copies of the scripts to use: **https://github.com/dan-veltri/ace-intro-to-deep-learning**

If you have trouble pulling from GitHub, copy the files from here:
`/home/bcbb_teaching_files/intro_deep_learning/`

What Can Deep Learning Do?

- Biological Classification and Prediction
  - Biological sequences – predicting DNA motifs or protein classes
  - Protein folding [Google's Deep Mind is now entering this space!]
- Computer Vision
  - Facial recognition
  - Self-driving cars
- Natural Language Processing
  - Auto-translation
  - Text ←→ Speech Recognition [Alexa, Google Home, Siri, etc.]
  - Sentiment analysis
- Game Playing
  - Chess and Go [Humans no longer can compete with AI]

Image Source: https://www.nvidia.com/en-us/deep-learning-ai/industries/

Deep Neural Networks (DNN) in the News…

nature International weekly journal of science

Home | News & Comment | Research | Careers & Jobs | Current Issue | Archive | Audio & Video | Fo

Archive > Volume 550 > Issue 7676 > News & Views > Forum > Article

NATURE | NEWS & VIEWS | FORUM

日本語要約

Artificial intelligence: Learning to play Go from scratch

Satinder Singh, Andy Okun & Andrew Jackson

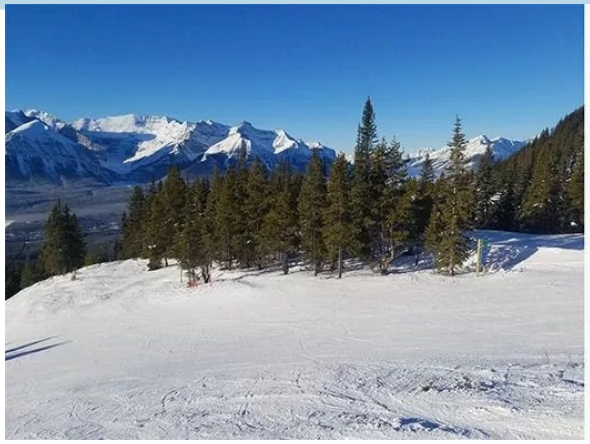
Affiliations | Corresponding authors

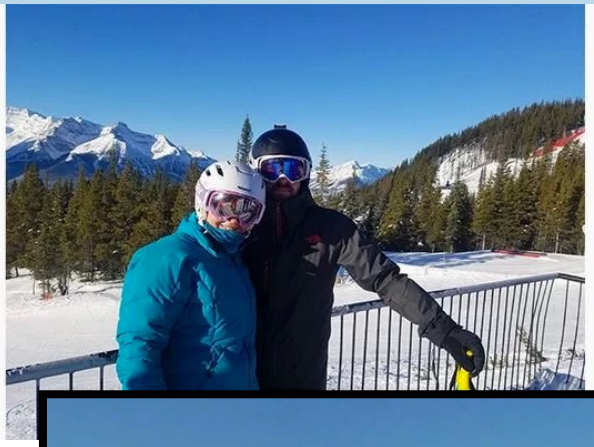
Nature **550**, 336–337 (19 October 2017) | doi:10.1038/550336a
Published online 18 October 2017

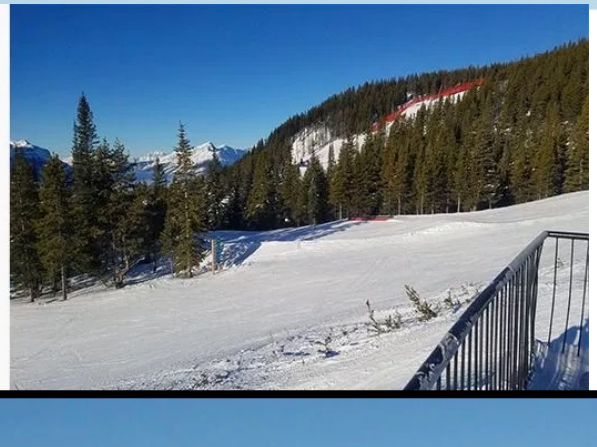
In March 2016, the artificial-intelligence program AlphaGo defeated a world Go champion, Lee Sedol.

Lee Jin-Man/AP/Rex/Shutterstock

Keep in Mind the *Limits* of Predictive Models…







Deep Learning is by no means perfect:
This creation by Google Photos' "Assistant" has recently been a running joke online.

Images by: Alex Harker

National Institute of Allergy and Infectious Diseases

NIAID

Keep in Mind the *Limits* of Predictive Models…

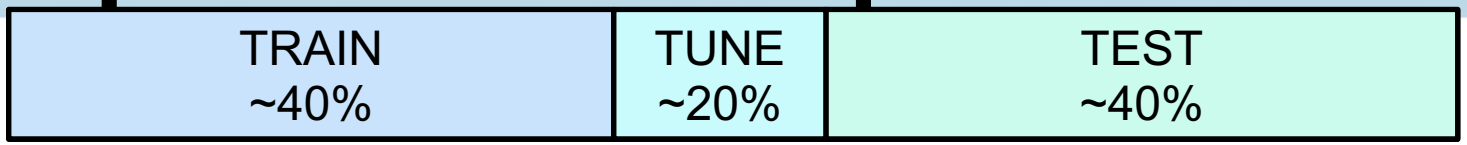
A recent major medical controversy for Deep Learning: ***CheXNet***

- A Stanford-released project claiming to be on par with human radiologists at identifying pneumonia. Caused a huge uproar amongst the radiology community
- Controversy stemmed much from *how the results were reported,* as well as the *quality of the data set and labeling*
- Things have calmed down thanks to the openness of the authors to address issues- *but the medical community is likely to be more skeptical about models that come out in the future*

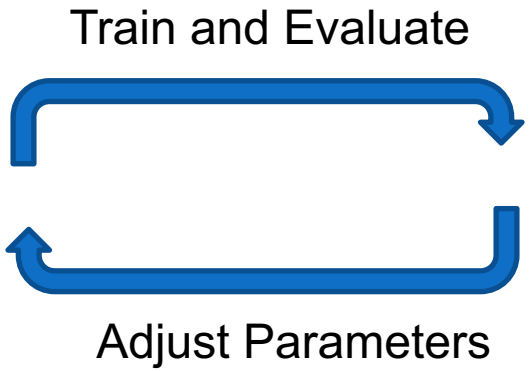
Are results too good to be true? Use your head.
Verify your work carefully!

Setting Up a Supervised ML Experiments

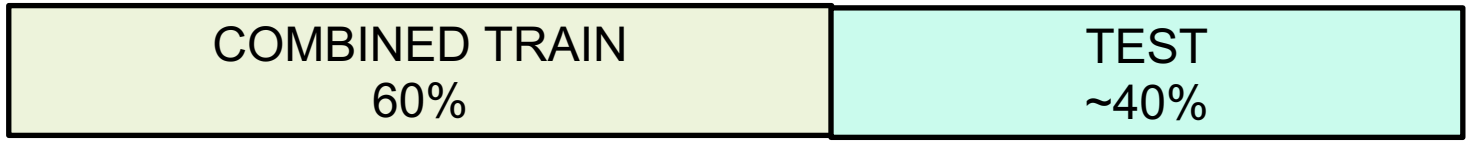
- Gather Representative Data



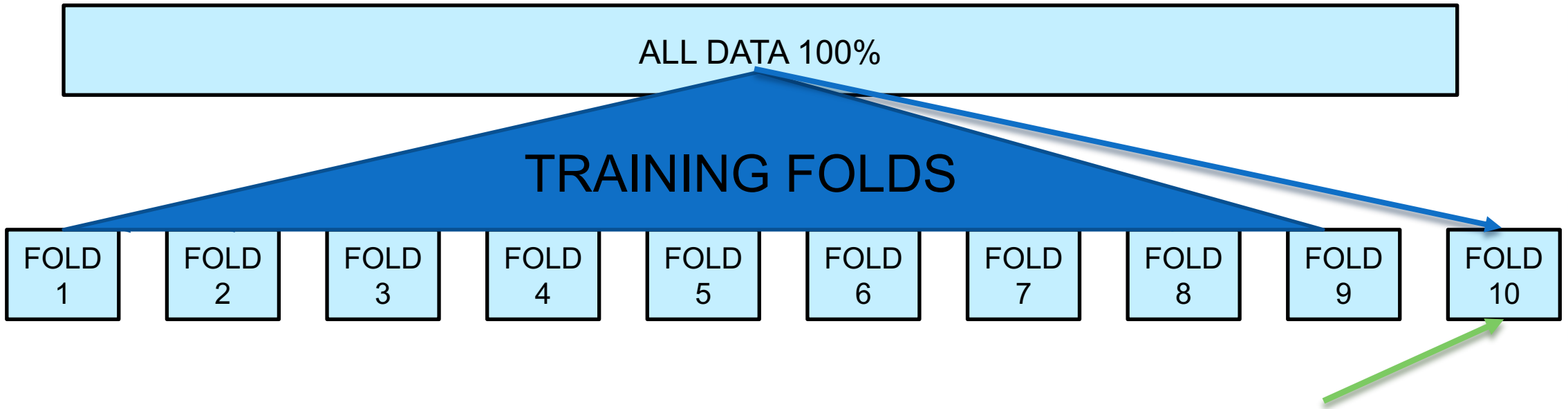
- Split into Training, Tuning and Testing Partitions



- Train Model and Use Tuning Partition like “Testing” while Adjusting Model parameters
- Merge Training + Tuning and Train Final Model
- Run Final Model on Testing Partition and Evaluate



K-Fold Cross Validation Experiments



- Split Data in *K* Folds (*K* is typically 3, 5, or 10)

- Use K-1 Folds for Training and 1 for Testing

- Rotate Until Each Fold is Used for Testing Once Report Results as the Average Over ALL Tests

So How Do We Evaluate ML Performance?

- For algorithms that produce real-valued numerical predictions we could calculate the difference between those and the the actual answers. For example, sum of squares: $\sum_{i=1}^{n}(y_i - \bar{y})^2$
- Today we will be dealing *mostly* with binary predictions for two classes: TRUE/FALSE, 0/1, A/B, etc.
- Common performance metrics for binary predictions are based on the number of true positives (**TP**), true negatives (**TN**), false positives (**FP**) and false negatives (**FN**):

Sensitivity (true positive rate): $^{TP}/_{TP+FN}$ **Specificity** (false positive rate): $^{TN}/_{TN+FP}$

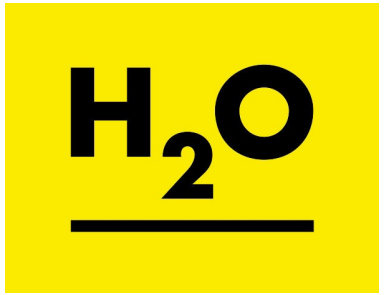
Accuracy: $^{TP+TN}/_{TP+TN+FP+FN}$ **F1-Score**: $^{2TP}/_{2TP+FP+FN}$

Matthews Correlation Coefficient: $$\frac{TP * TN - FP * FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

On to Deep Learning!

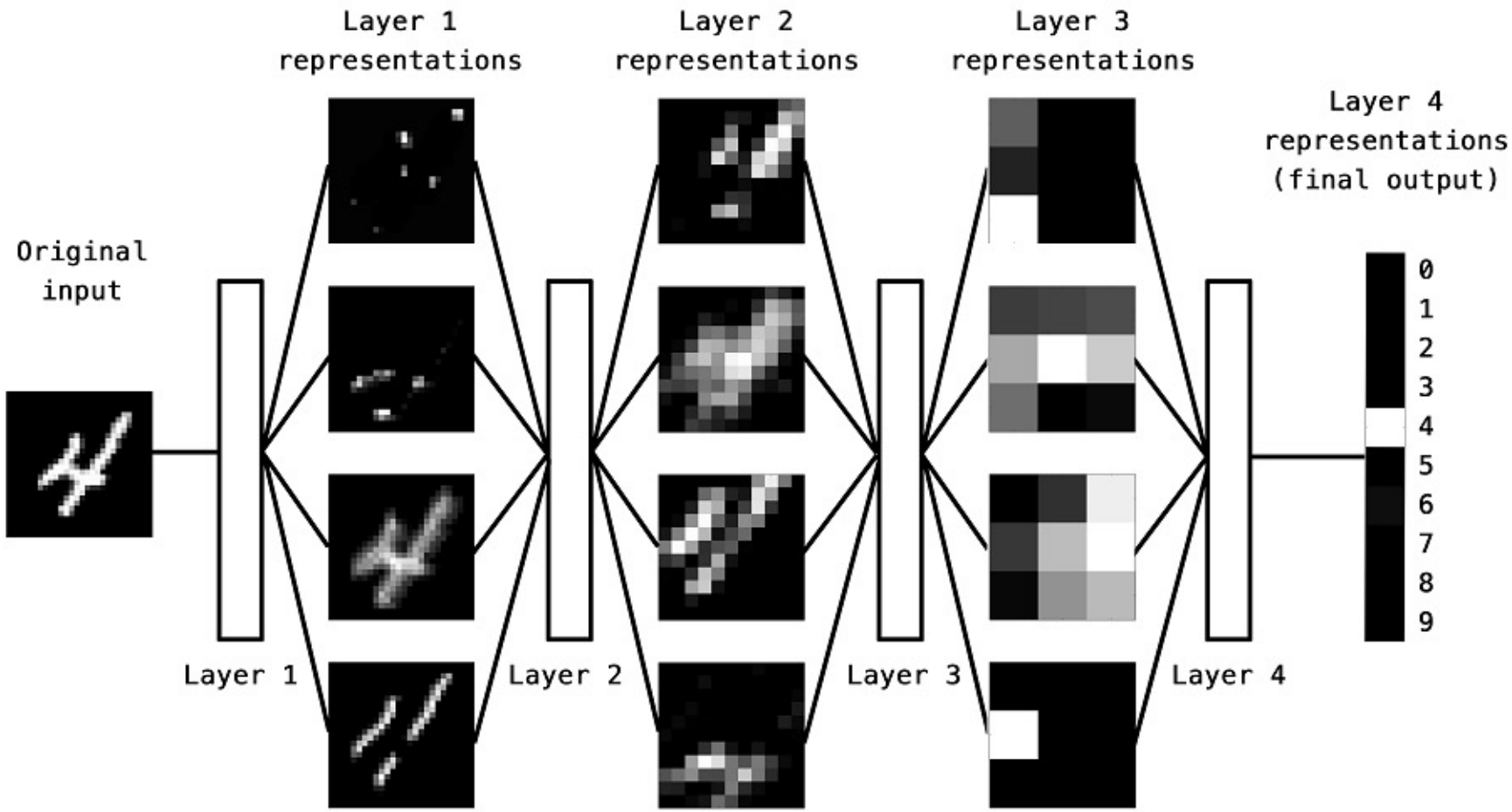
Caffe

theano



There are many frameworks to choose from!

“Deep” Neural Networks Have Multiple Layers

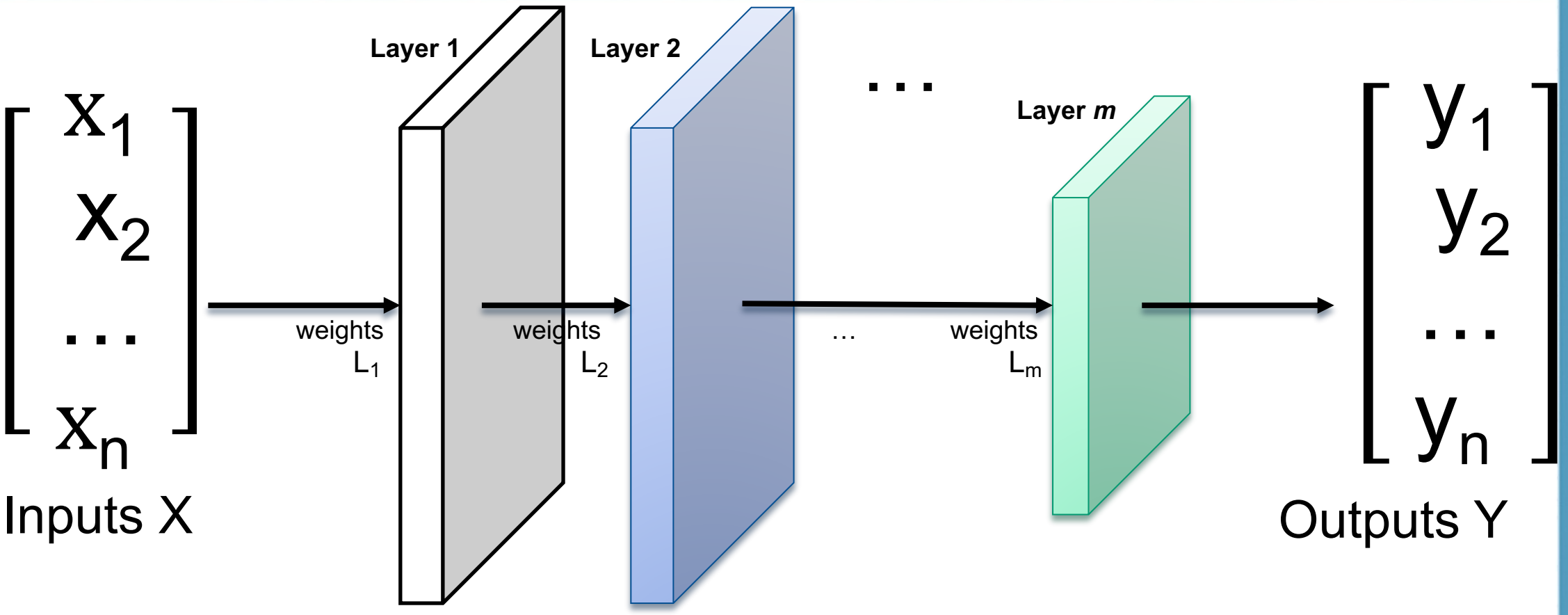


National Institute of Allergy and Infectious Diseases

Source: Chollet and Allaire “Deep Learning with R” pp.9, 2018.

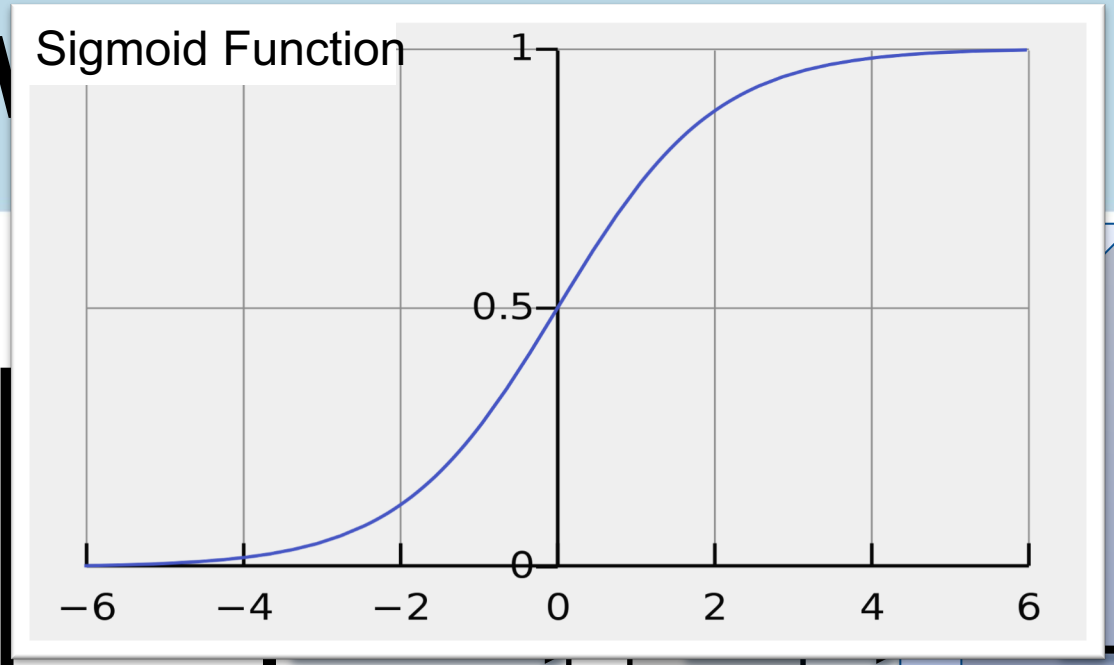
NIAID

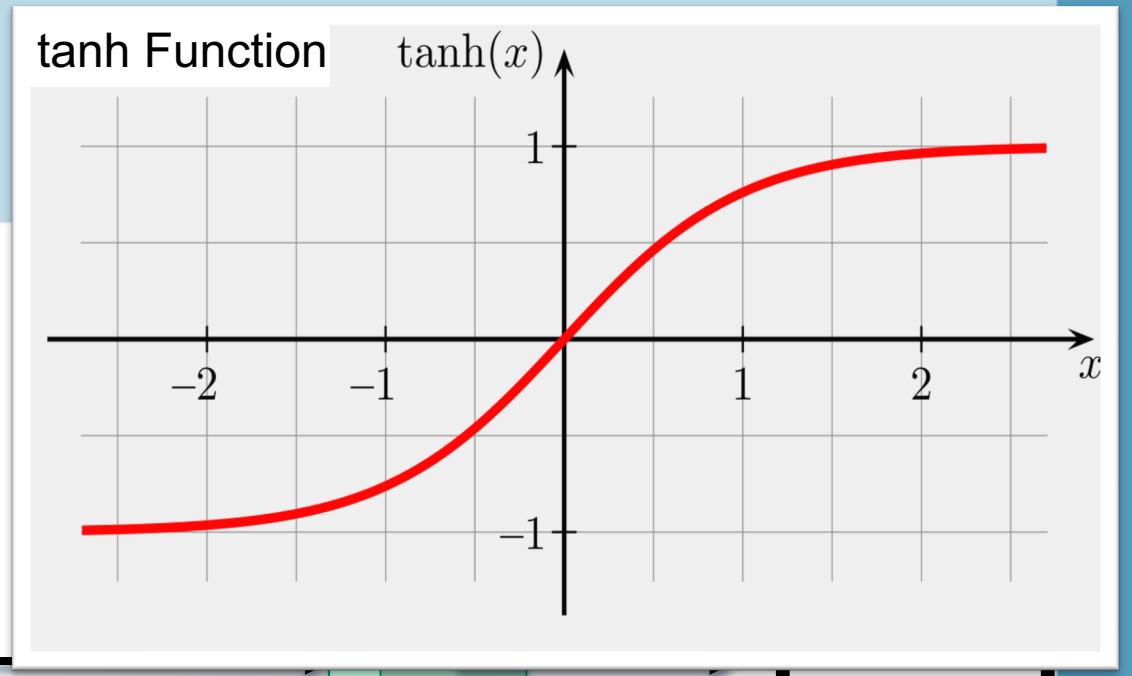
Weights and Layers



Often weights are randomly initialized and layer outputs are often "activated" using functions to force numbers in a certain range. Typical examples include: ***sigmoid***, ***tanh***, and **rectifier linear unit** (ReLU) functions.

W





x_n

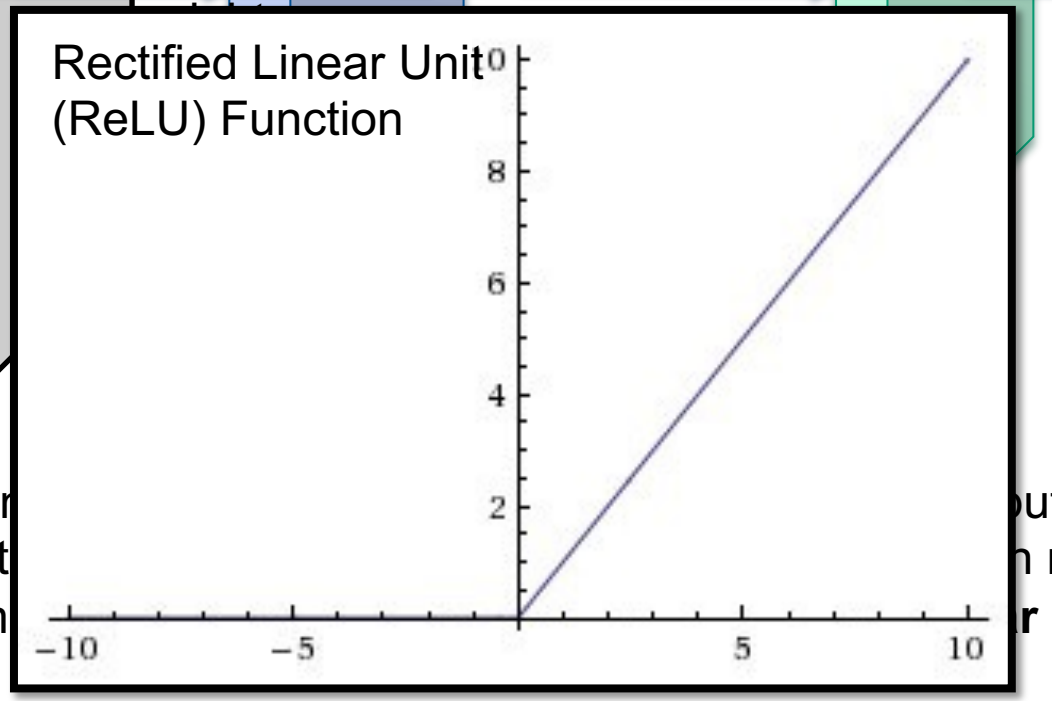
Inputs X

weights

weights

L_1

L_m



y_n

Outputs Y

Ofter
“activat
exam

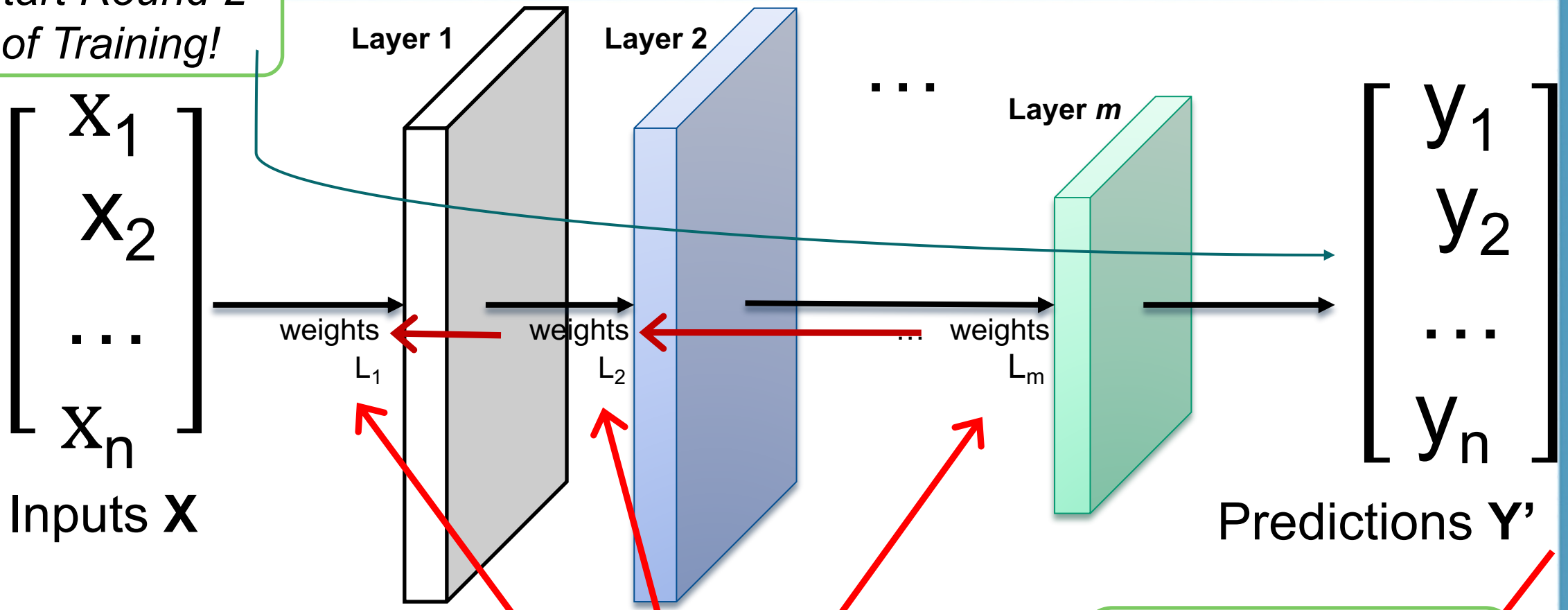
uts are often
n range. Typical
r unit (ReLU)

National Institute of Allergy and Infectious Diseases

NIAID

Weights and Layers – Optimizing the Network

Start Round 2 of Training!



Optimizer (adjust weights)

Apply scoring function using *actual* **Y** values

Optimizing via Backpropagation
The Secret Sauce of Deep Neural Networks (DNNs)

- How do DNNs learn so well? The key is they compute answers across layers in a ***forward pass*** and then to use a ***backwards pass*** to optimize the weights. This way **all** layers are updated each round (sometimes called an 'epoch') of training!

- How does this backward pass work? **The chain rule** (remember from calculus?) – where we can calculate the derivative (the slope or rate of change) from *two or more* functions.

You might have seen this written as: $(f \circ g)' = (f' \circ g) \cdot g'$

or maybe like this: $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$ or maybe like this: $F'(x) = f'\big(g(x)\big)g'(x)$

The takeaway: we can calculate the derivative using multiple functions at the same time!

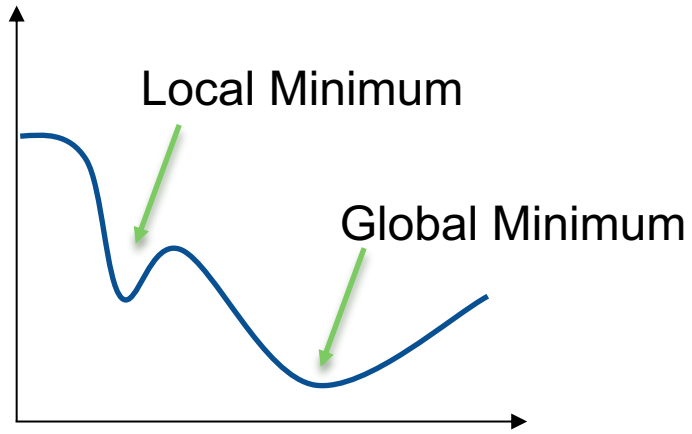
National Institute of Allergy and Infectious Diseases

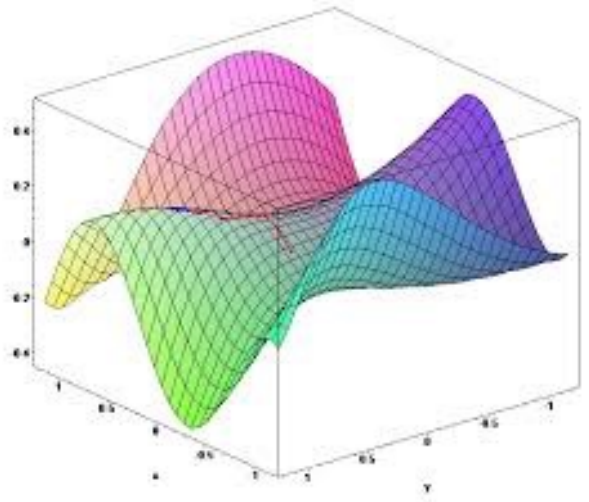
The following examples comes from here with more details available: http://cs231n.github.io/optimization-2/

NIAID

Optimizing via Backpropagation
The Secret Sauce of Deep Neural Networks (DNNs)

- For our DNNs we are calculating the ***gradient*** (a vector of derivatives) to account for the change across the network based on the forward pass results.
- Given a function $f(x)$ where x's are our **training** inputs- the gradient forms a vector: $\nabla f(x) = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [y, x]$





National Institute of Allergy and Infectious Diseases

NIAID

Some Backpropagation Intuition

Lets look at multiplication: $f(x, y) = xy \rightarrow \frac{\partial f}{\partial x} = y \quad \frac{\partial f}{\partial y} = x$

If x = 4 and y = −3 $f(x, y) = -12 \quad \frac{\partial f}{\partial x} = -3 \quad \frac{\partial f}{\partial y} = 4$

Lets look at basic addition: $f(x, y) = x + y \rightarrow$???

What happens to each function if we change x … or change y?

Some Backpropagation Intuition

Lets look at ***multiple*** functions:

$$f(x, y, z) = (x + y)\, z$$

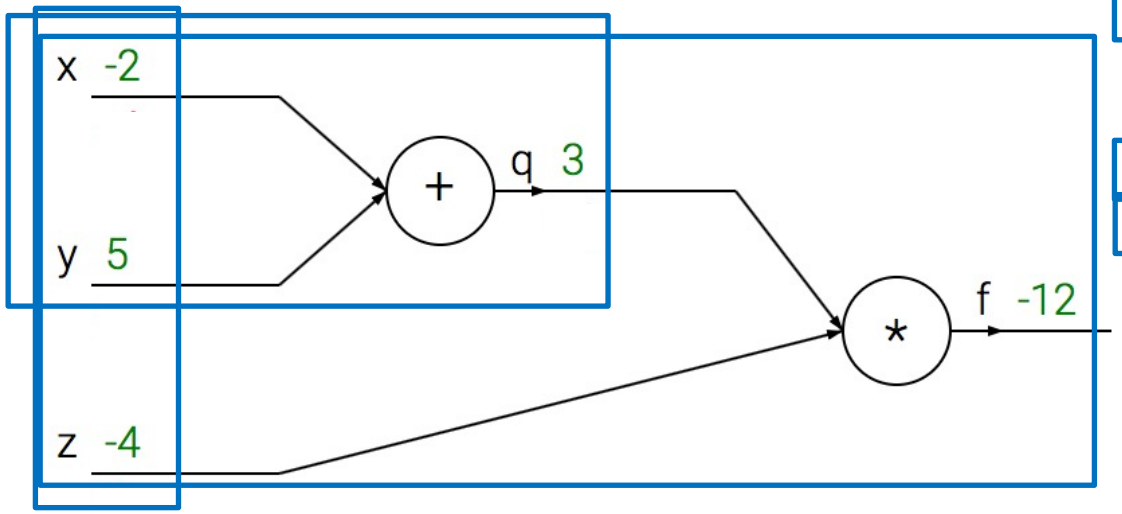
We can rewrite this as: $q = x + y \;\; and \; f = qz$

SO $\frac{\partial f}{\partial q}$ = **z** , $\frac{\partial f}{\partial z}$ = **q** … for $(x + y)$ as we saw before: $\frac{\partial f}{\partial x}$ = **1** , $\frac{\partial f}{\partial y}$ = **1**

The **chain rule** says **multiply:** $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x}$

Lets look at this with code and a visual representation!

Backpropagation Example




```
# set some inputs
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation)
# first backprop through f = q * z
dfdq = z # df/dq = z, so gradient on q becomes -4
dfdz = q # df/dz = q, so gradient on z becomes 3

# now backprop through q = x + y
dfdx = 1.0 * dfdq # dq/dx = 1  chain rule!
dfdy = 1.0 * dfdq # dq/dy = 1
```

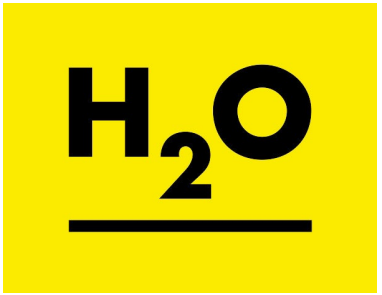
*Forward pass is **green**. Backward pass is **red**.*

The big takeaway: The final gradient $[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}]$ tells us how sensitive our function f is to the variables x, y, and z.

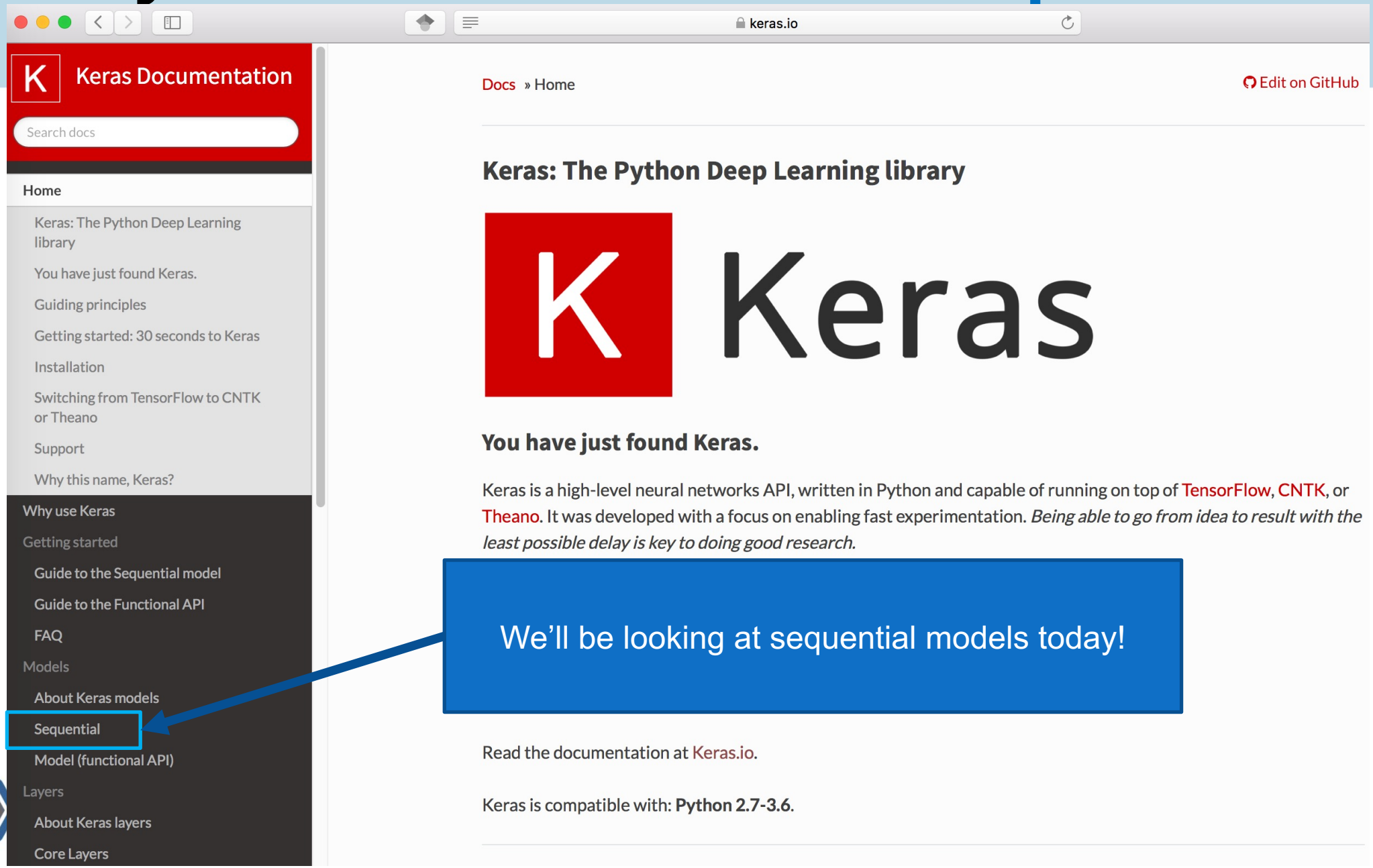
On to Deep Learning… really this time!

Caffe

theano



Keras Python Documentation: https://keras.io



We'll be looking at sequential models today!

NIH

NIAID

Keras R Documentation:
https://keras.rstudio.com

Reference version 2.1.3.1003

Keras Models

keras_model	Keras Model
keras_model_sequential	Keras Model composed of a linear stack of layers
multi_gpu_model	Replicates a model on different GPUs.
summary	Print a summary of a Keras model
compile	Configure a Keras model for training
fit	Train a Keras model
evaluate	Evaluate a Keras model
export_savedmodel	Export a Saved Model
fit_generator	Fits the model on data yielded batch-by-batch by a generator.
evaluate_generator	Evaluates the model on a data generator.
predict	Generate predictions from a Keras model
predict_proba predict_classes	Generates probability or class probability predictions for the input samples.
predict_on_batch	Returns predictions for a single batch of samples.
predict_generator	Generates predictions for the input samples from a data generator.
train_on_batch test_on_batch	Single gradient update or model evaluation over one batch of samples.

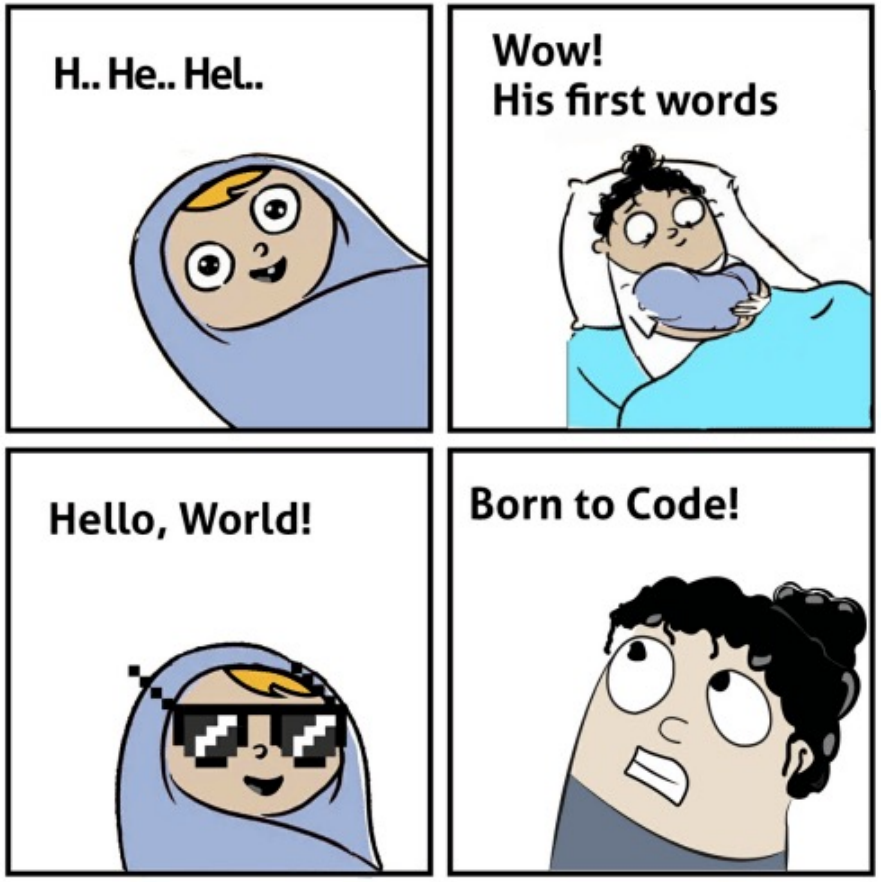
Contents

NIAID

Basic Setup to Make a DNN in Keras

1. Import the Keras library and any other supporting packages
2. Load in our data and pre-process if necessary (e.g. normalize images, pad sequences, etc.)
3. Setup our model structure – chose our data input/output shape, what layers to include, what loss and optimization function will we use?
4. Fit our model to the training (and validation) data
5. Visualize training and validation performance to see if we need to change model parameters
6. Make predictions on our testing data and evaluate performance

Starting a Model in Keras



Lets look at some code!

“Hello World” with Fisher/Anderson’s Iris data set

We’ll get started using a small data set of collected *Iris* data from Fisher’s famous 1936 LDA paper

Iris setosa

Iris versicolor

Iris virginica

Data set contains:

-50 samples each for three different *Iris* spp. (rows)- each class is given a number

-Four measurements/features (cols) per flower- width and length of sepals and petals

National Institute of Allergy and Infectious Diseases

Images from Wikimedia

NIAID

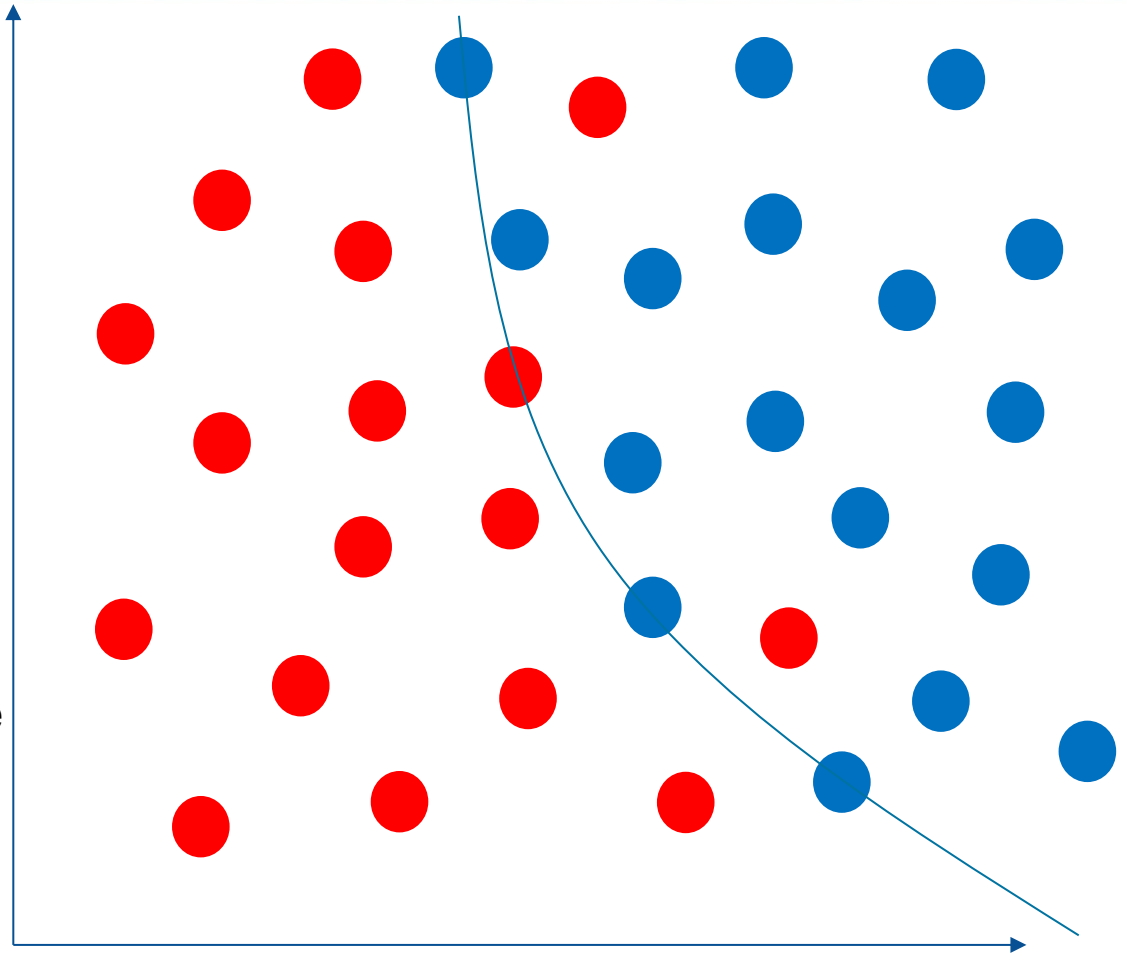
So how long do we perform training?

Visualizing Training Performance
Are we Over- or Underfitting our Data?

Overfitting: Training our model too much!
We are making it fit the training data so well that it does not *generalize* well to new unseen data.

Underfitting: Our model is missing key parameters needed to model the data- *typically this means you need to train more on the data*.

Regularization: Relaxing our requirements so that we penalize changes to the weights. This is typically done with “L1” or “L2” regularization in Keras but the specifics are beyond our talk today.



Plotting out Training and Validation Curves

We can learn a lot from looking at the history of our performance on the training and validation data!

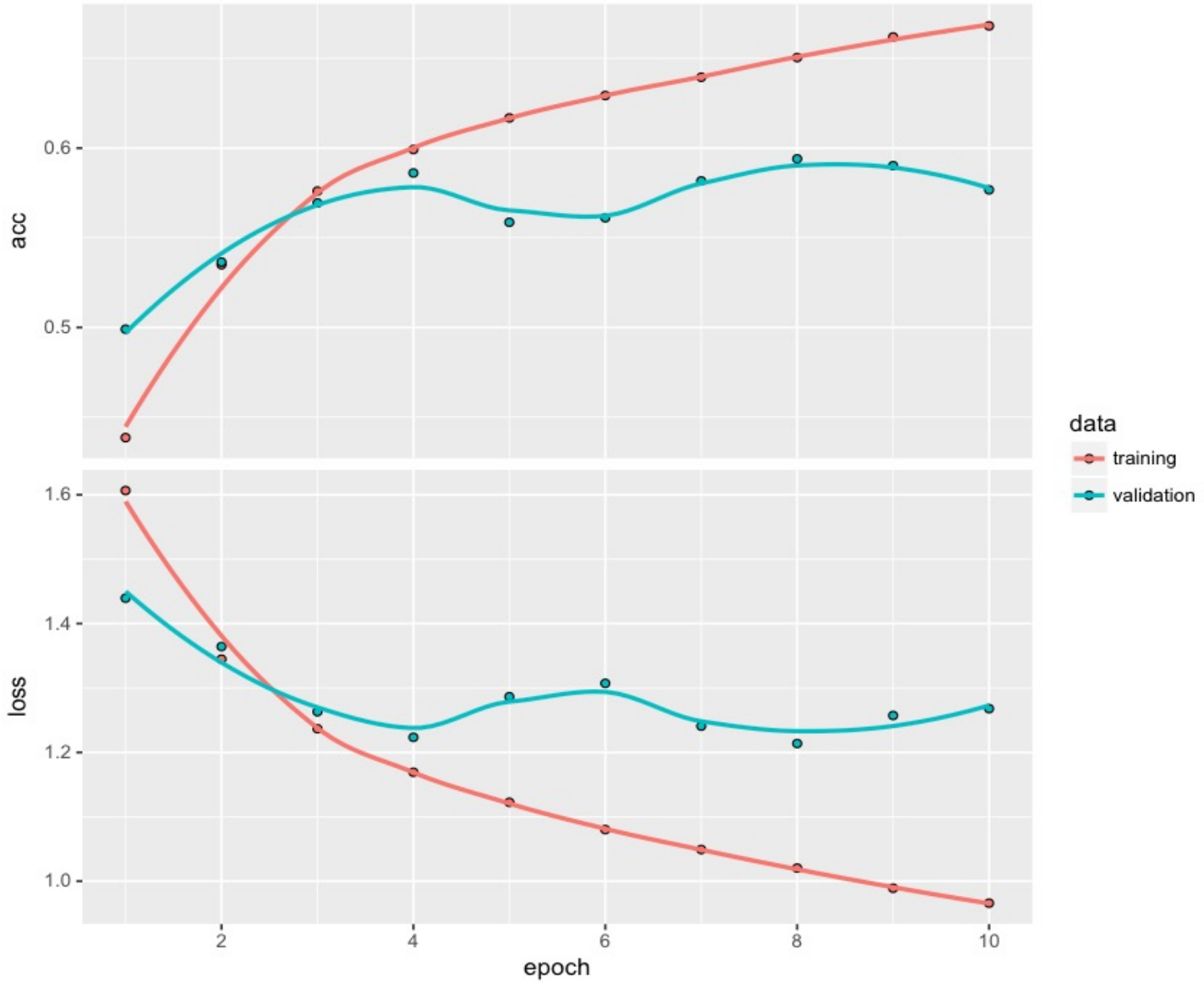
In Python:

```
model_info = model.fit( … )
model_info.history['acc'] # train ACC
model_info.history['val_acc'] # val ACC
model_info.history['loss'] # train loss
model_info.history['val_loss'] # val loss
```

… now put this in a plotting function

In R (*much easier!*):

```
history <- model %>% fit( … )
plot(history)
```



National Institute of Allergy and Infectious Diseases

NIAID

Plotting even more with TensorBoard

Tensorflow comes with TensorBoard, a real-time display of your model's training progress.

In Terminal (start up TensorBoard):

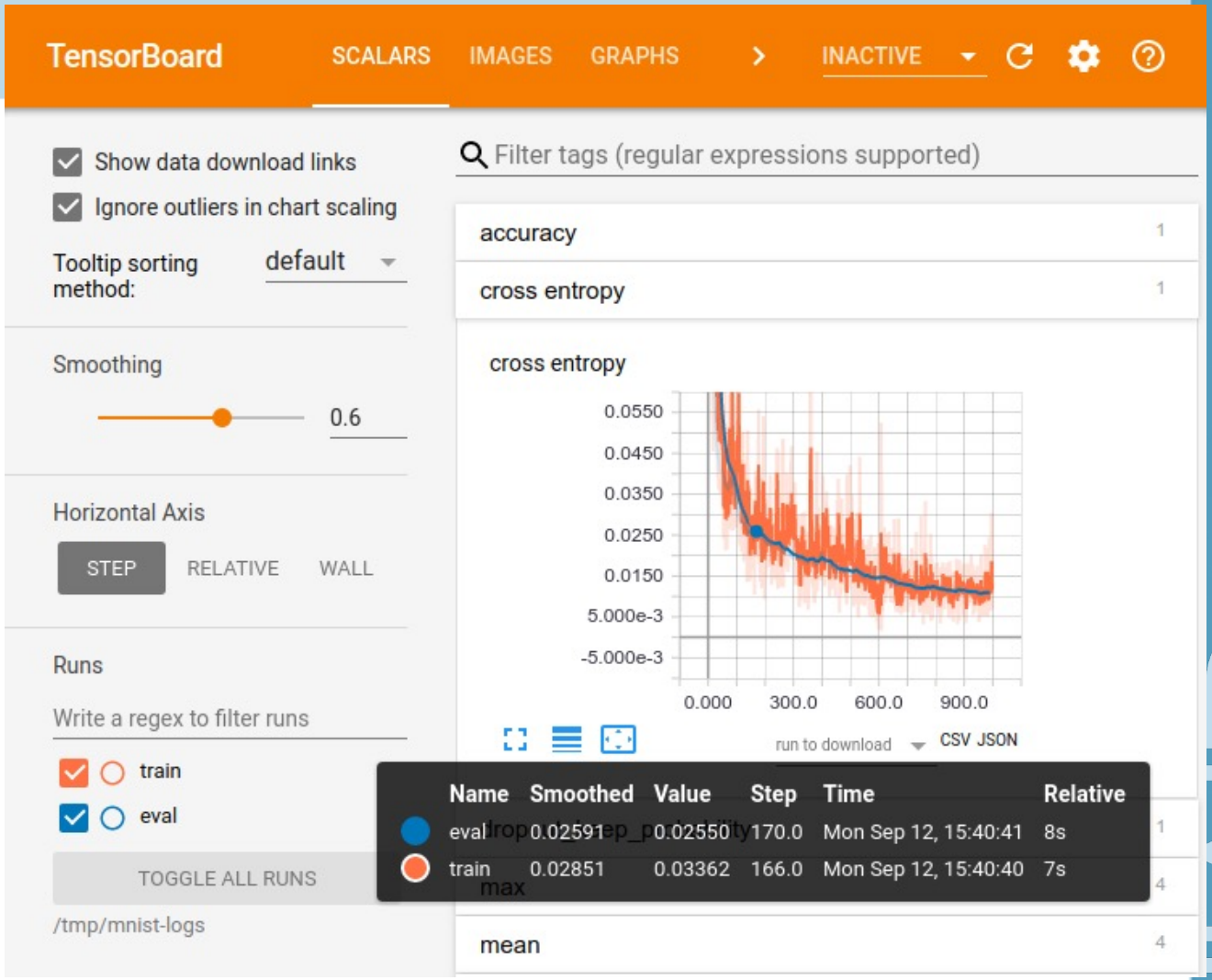
```
tensorboard --logdir=/path_to_logs
```

In Python:

```
from keras.callbacks import TensorBoard
tb = TensorBoard(log_dir='/path_to_logs',
                 histogram_freq=1,
                 embeddings_freq=1)
history = model.fit( …, callbacks=[tb])
```

In R:

```
Tensorboard('/path_to_logs')
tb = list(callback_tensorboard(log_dir='/path_to_logs',
                 histogram_freq=1,
                 embeddings_freq=1)
)
history <- model %>% fit( …, callbacks=tb)
```



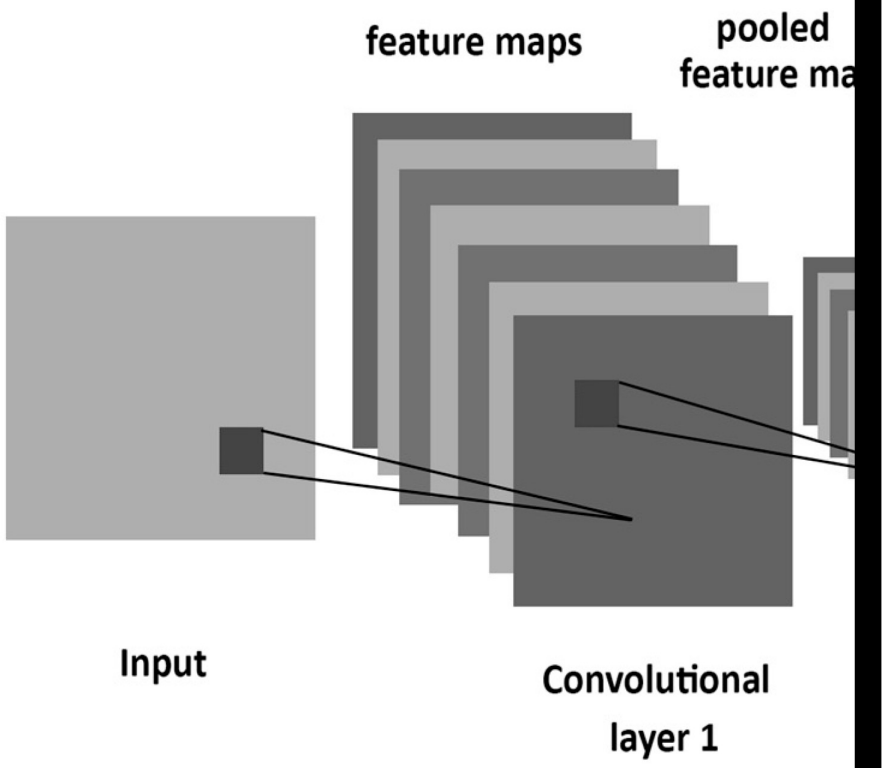
View with your Internet browser pointed (by default) to:
localhost:6006

National Institute of Allergy and Infectious Diseases

NIAID

So how do we learn 2D patterns like images?

Convolutional Layers



1	1	1 ×1	0 ×0	0 ×1
0	1	0 ×0	1 ×1	0 ×0
0	0	1 ×1	0 ×0	1 ×1
0	0	1	1	0
0	1	1	0	0

Image

4	3	4

Convolved
Feature

Convolutional Layer Defaults in Keras [Python]

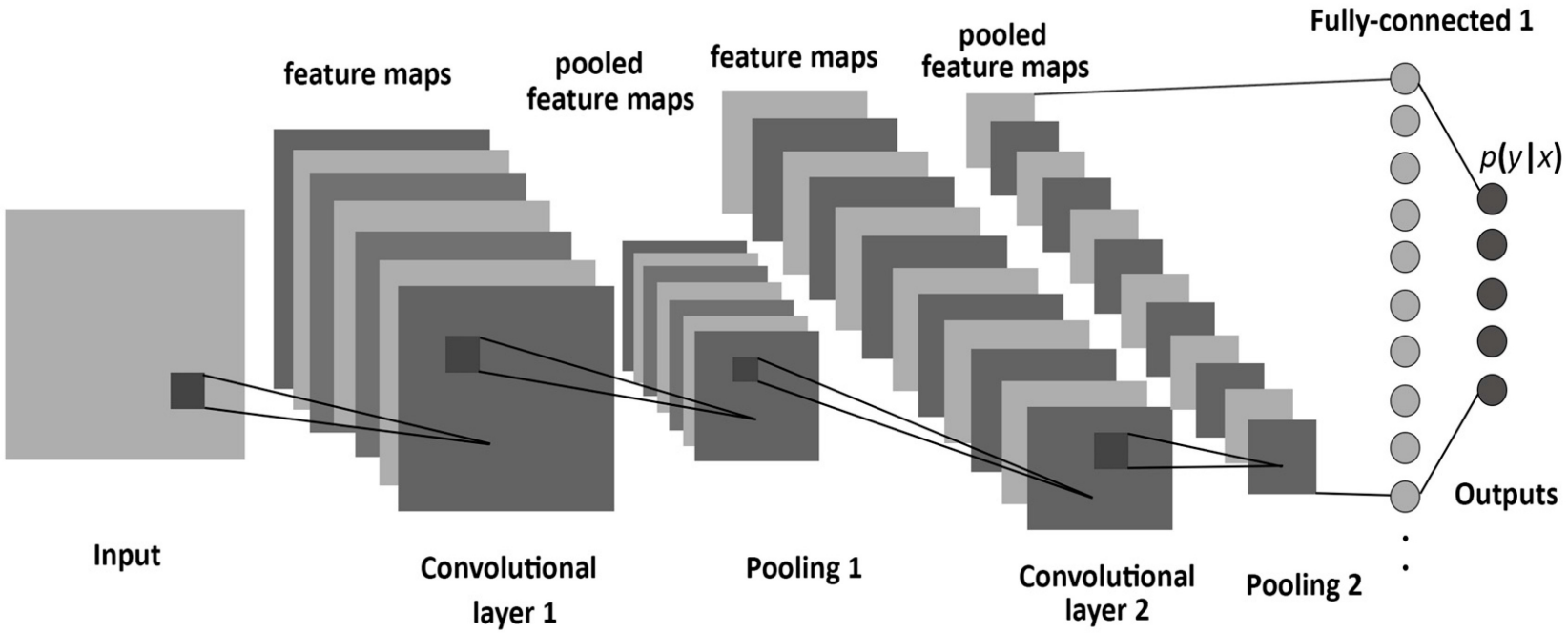
1D is typically for text – Input Shape: (batch_size, steps, input_dim)

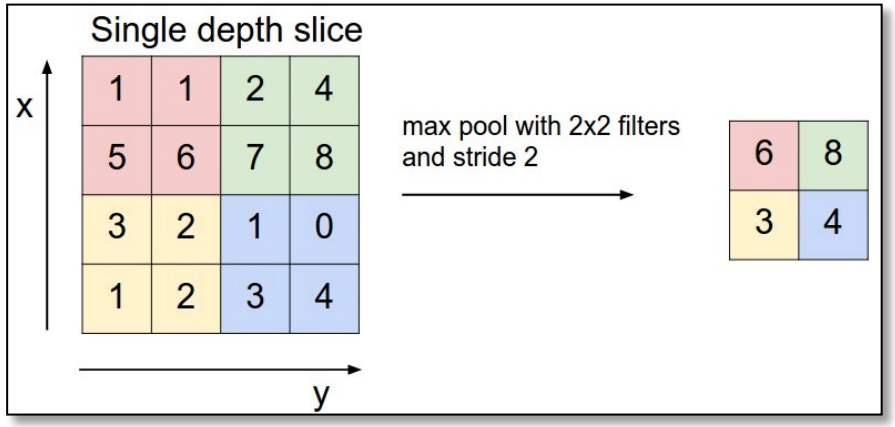
keras.layers.**Conv1D**(filters, kernel_size, strides=1, padding='valid', dilation_rate=1, activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

2D is typical for Images – Input Shape: (samples, channels, rows, cols)

keras.layers.**Conv2D**(filters, kernel_size, **strides=(1, 1)**, padding='valid', **data_format=None**, **dilation_rate=(1, 1)**, activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

Pooling Layers





National Institute of Allergy and Infectious Diseases

NIAID

Pooling Layer Defaults in Keras [Python]

1D is typically for text – Input Shape: (batch_size, steps, features)

keras.layers.**MaxPooling1D**(pool_size=2, strides=**None**, padding='valid')

2D is typical for Images – Input Shape: (batch_size, rows, cols, channels)

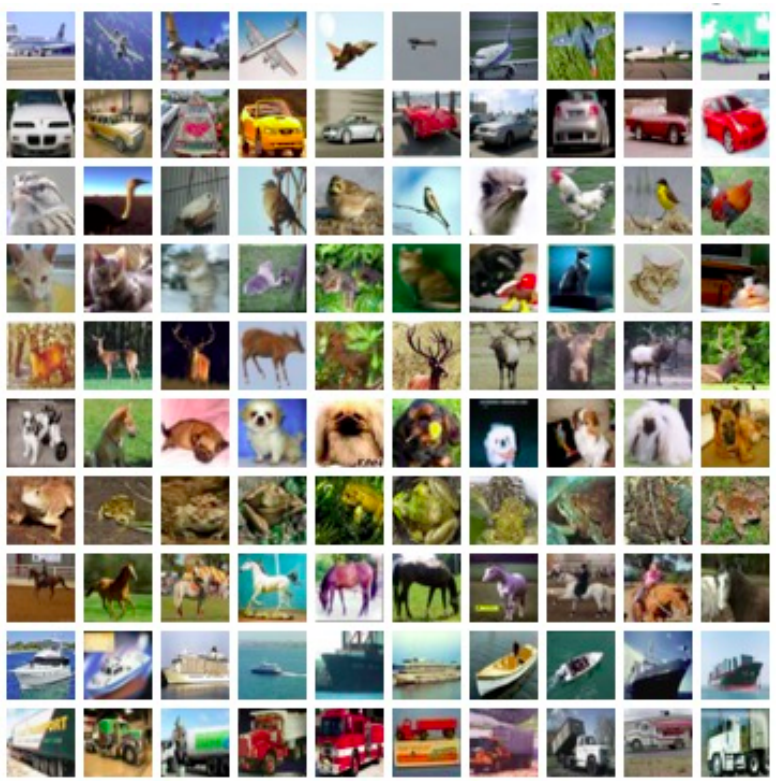
keras.layers.**MaxPooling2D**(pool_size=(2, 2), strides=**None**, padding='valid', data_format=**None**)

Let's Build a Model to Classify Images!

CIFAR10 small image classification set

The dataset is 60K 32x32 color images containing one of 10 numeric classes:

- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck



CIFAR10 is provided with Keras!

Let's Build a Model to Classify Images!

CIFAR10 small image classification set

Dataset of 50,000 32x32 **color (RGB)** training images, labeled over 10 categories, and 10,000 test images.

Usage:

```
from keras.datasets import cifar10 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Returns:

2 tuples:

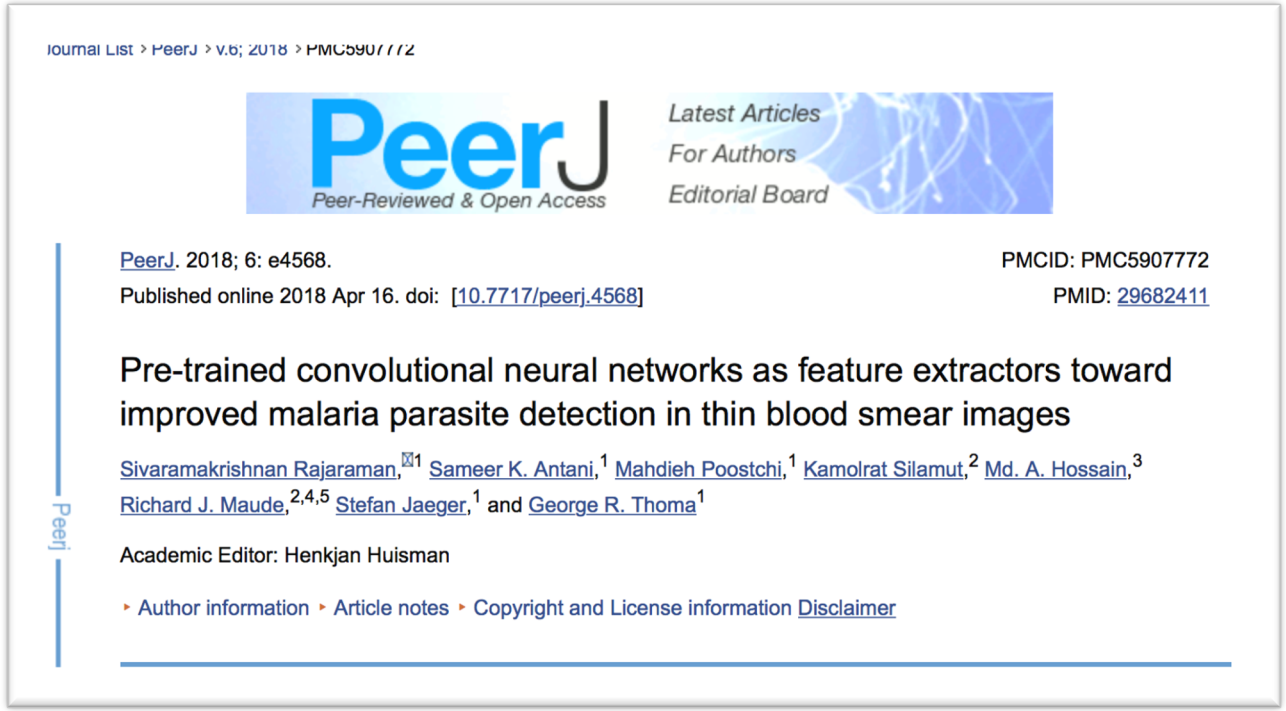
x_train, x_test: uint8 array of RGB image data with shape (num_samples, 3, 32, 32).

y_train, y_test: uint8 array of category labels (integers in range 0-9) with shape (num_samples,).

Transfer Learning: We don’t always need to learn everything from scratch!

Let’s predict if cells have malaria using a model built for cellphones!

- Rajaraman et al. recently used *pre-trained* neural networks to build a new predictor to classify if blood smear images show malaria presence with > 95% ACC. They built this using multiple pre-trained networks and a considerable amount of time and computing power.



Journal List › PeerJ › v.6; 2018 › PMC5907772

PeerJ
Peer-Reviewed & Open Access

Latest Articles
For Authors
Editorial Board

PeerJ. 2018; 6: e4568.
Published online 2018 Apr 16. doi: [10.7717/peerj.4568]

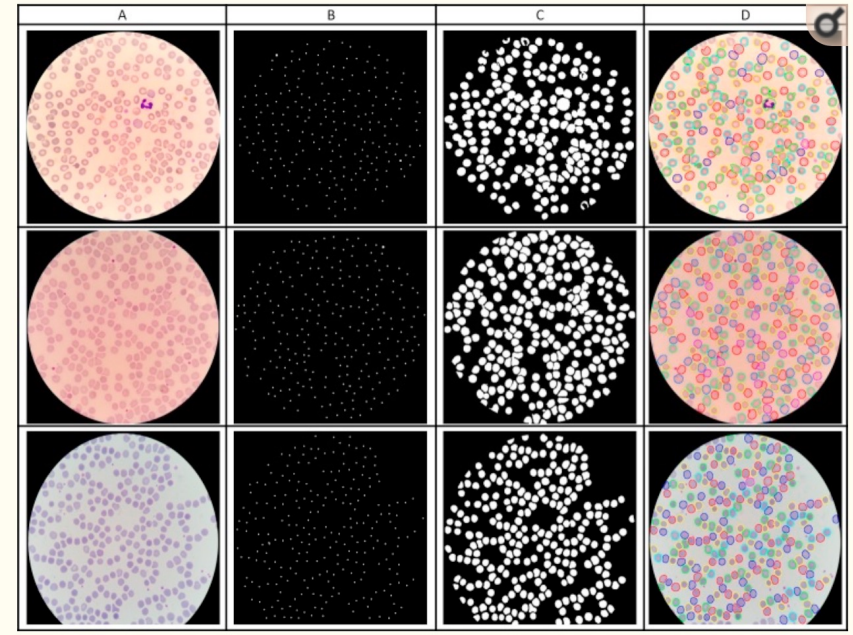
PMCID: PMC5907772
PMID: 29682411

Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images

Sivaramakrishnan Rajaraman,[1] Sameer K. Antani,[1] Mahdieh Poostchi,[1] Kamolrat Silamut,[2] Md. A. Hossain,[3] Richard J. Maude,[2,4,5] Stefan Jaeger,[1] and George R. Thoma[1]

Academic Editor: Henkjan Huisman

Author information · Article notes · Copyright and License information Disclaimer



Uninfected

Parasitized

National Institute of Allergy and Infectious Diseases

Plate images were segmented so the dataset is comprise of individual cells images. Those with malaria are stained **dark pink**.

NIAID

Lets predict if cells have malaria using a model built for cellphones!

- Howard et al. at Google built MobileNet to assist with cellphone vision applications like facial recognition

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard Menglong Zhu Bo Chen Dmitry Kalenichenko
Weijun Wang Tobias Weyand Marco Andreetto Hartwig Adam

Google Inc.

{howarda,menglong,bochen,dkalenichenko,weijunw,weyand,anm,hadam}@google.com

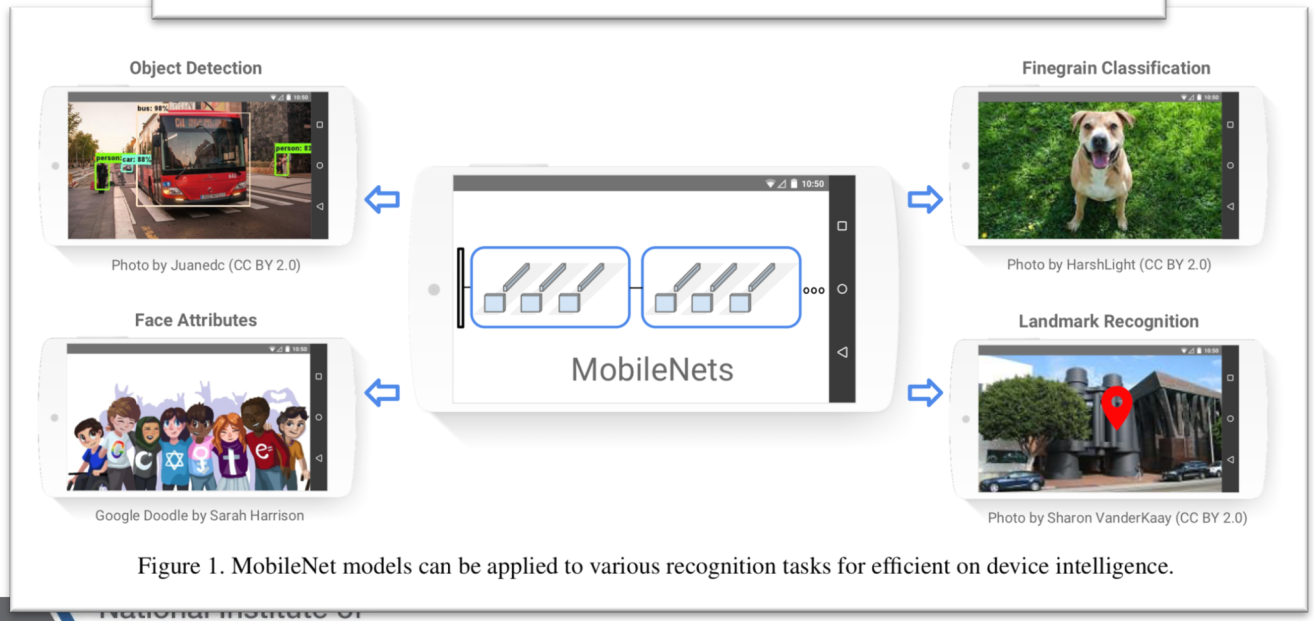


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

NIH National Institute of Allergy and Infectious Diseases

How might MobileNet help us detect malaria?

How do we handle variable-length inputs like text, DNA sequences, etc.???

What about Variable-Length Inputs?

- **PROBLEM:** We often (but not always) expect a fixed shape for our input – with text we often have words or sentences of variable size!
- **Solution 1:** Assign each "word" (sentence, nucleic acid, etc.) a number/index and use 0's to pad vectors until they are all the same size. The network will learn to ignore the 0's!

WORD	INDEX
the	1
cat	2
ran	3
to	4
park	5

The cat ran to the park
< 1, 2, 3, 4, 1, 5 >

The cat ran
< 0, 0 , 0, 1, 2, 3 >

You need a spot for every word in your corpus.
Often have very long "sparse" vectors filled with zeros!

What about Variable-Length Inputs?

- **PROBLEM:** We often (but not always) expect a fixed shape for our input – with text we often have words or sentences of variable size!
- **Solution 2:** Use "One-Hot" vectors - assign each "word" (sentence, nucleic acid, etc.) to a particular column, and mark with 0/1 if missing/present!

The cat ran to the park: < 1, 1, 1, 1, 0, … >

The cat ran: < 1, 1 , 1, 0, 0, … >

THE CAT RAN PARK DOG …

Again, you need a spot for every word in your corpus.
Often have very long "sparse" vectors filled with zeros!

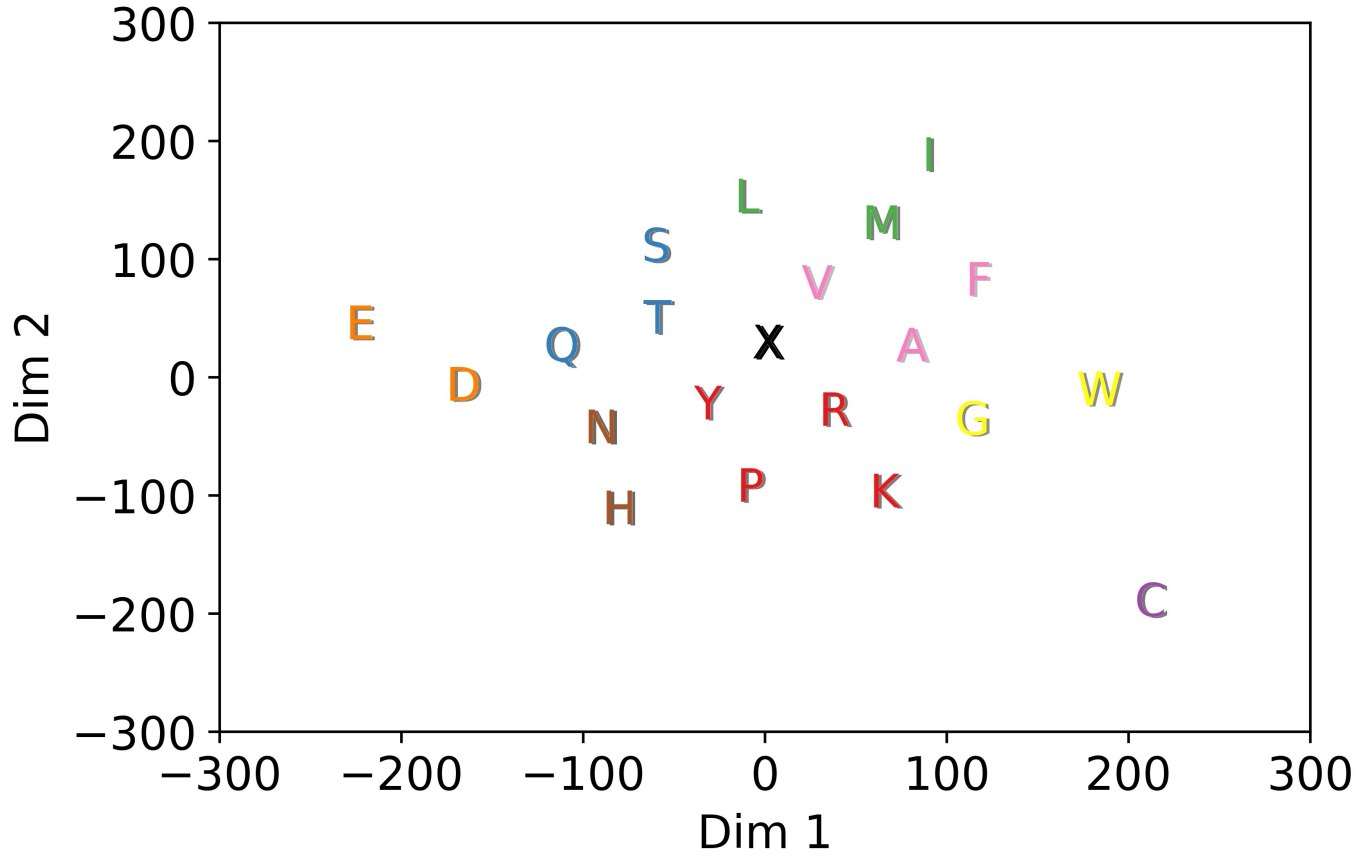
What about Variable-Length Inputs?

- **Embedding Layer:** We can also represent words by assigning them each to a vector of number (of a length we choose)
- We initially fill these with random weights (or selected from a distribution) and train them along with our model. This avoids having a bunch of unnecessary 0's.
- After training, the weights can be used to see how words are related to each other. More similar objects should have more similar vector representations.

keras.layers.Embedding(input_dim, output_dim, embeddings_initializer='uniform',
embeddings_regularizer=**None**, activity_regularizer=**None**,
embeddings_constraint=**None**, mask_zero=**False**, input_length=**None**)

If used, this needs to be the first layer in your model!

Example of 128-dim embedding vectors for amino acids in antimicrobial peptides (projected in 2D with t-SNE)



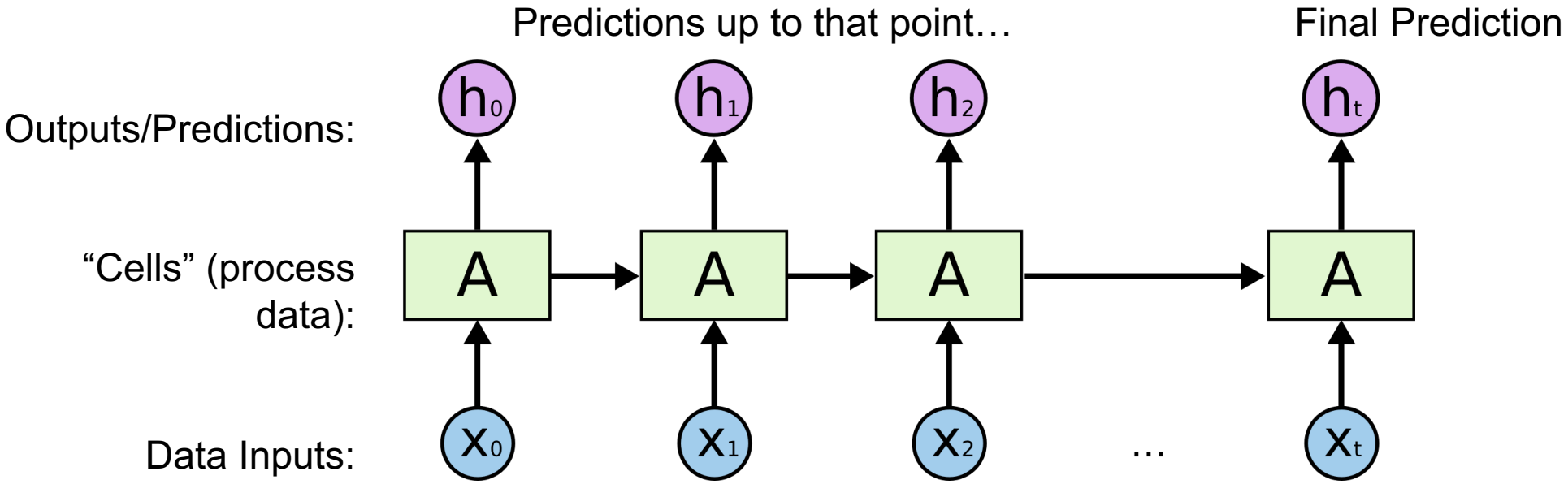
Colors were assigned by K-means clustering (K=9) - they *generally* group by known physicochemical properties (charge, size, etc.)

Veltri et al. (2018) *Bioinformatics*

National Institute of Allergy and Infectious Diseases

NIAID

Recurrent Networks

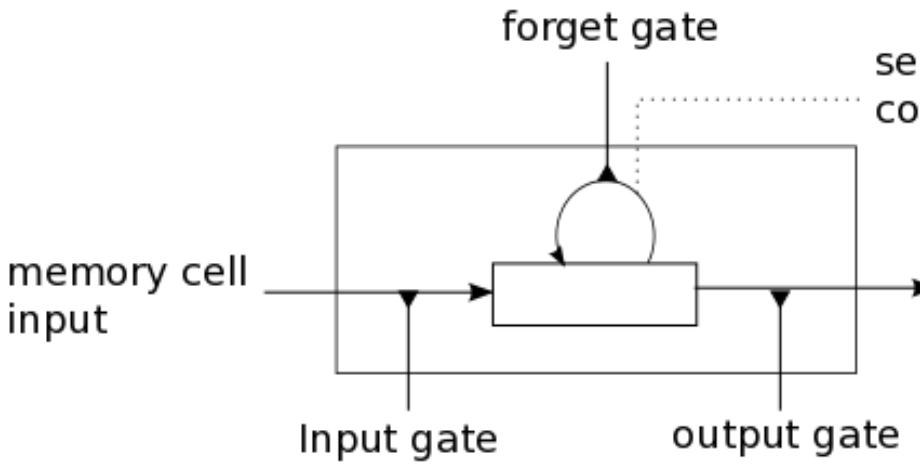


The Basic Idea: Slide along data inputs (letter, word, etc.) one step at a time and pass along info to the next step to help in making a prediction. Variable lengths are OK!

The Problem: Remembering everything quickly becomes intractable for long sequences!

Figure and great tutorial:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

One Solution: Long Short-Term Memory (LSTM)



Direction of Reading →

Ignore!

…TCCGCGATCGTTCGGGTGGCCTTTAATATTATGTGCGCGTTAGCTGCACGCG…

Recognize Pattern!

National Institute of Allergy and Infectious Diseases

Original LSTM Paper: Hochreiter and Schmidhuber (1997) Long short-term memory.

Figure: *deeplearning.net*

NIAID

Recurrent LSTM Layer Defaults in Keras [Python]

Always for sequential data – Input Shape: (batch_size, timesteps, input_dim)

keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid',
use_bias=**True**, kernel_initializer='glorot_uniform',
recurrent_initializer='orthogonal', bias_initializer='zeros', unit_forget_bias=**True**,
kernel_regularizer=**None**, recurrent_regularizer=**None**, bias_regularizer=**None**,
activity_regularizer=**None**, kernel_constraint=**None**, recurrent_constraint=**None**,
bias_constraint=**None**, dropout=0.0, recurrent_dropout=0.0, implementation=1,
return_sequences=**False**, return_state=**False**, go_backwards=**False**, stateful=**False**,
unroll=**False**))

We can have the LSTM check forward and backwards by wrapping it in a ***Bidirectional*** layer!

keras.layers.Bidirectional(layer, merge_mode='concat', weights=**None**)

For example: model.add(Bidirectional(LSTM(units=10)))

Let's Build a Model to Recognize Movie Reviews!

Internet Movie Database (IMDB) sentiment classification set

Dataset of 25,000 IMDB movie review, labeled positive or negative. Each review is encoded as word indexes (numbers that stand in for words). Only the top 10K words in the database are considered.

Usage:

```
from keras.datasets import imdb

(x_train, y_train), (x_test, y_test) = imdb.load_data(path="imdb.npz",
              num_words=None, skip_top=0, maxlen=None, seed=113, start_char=1,
              oov_char=2, index_from=3)
```

Returns:

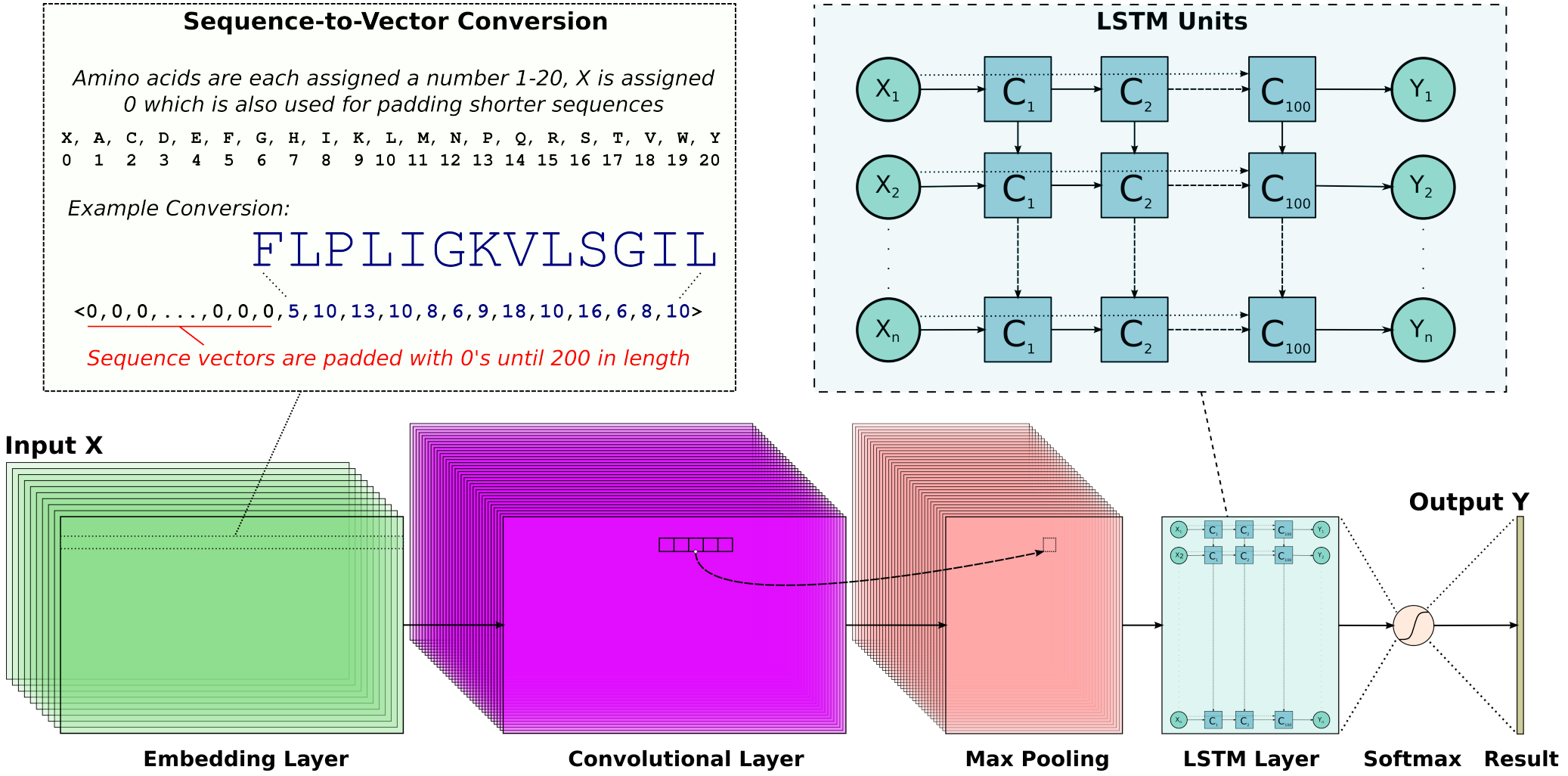
2 tuples:

x_train, x_test: list of sequences, which are lists of indexes (integers). If the num words argument was specific, the maximum possible index value is num_words-1. If the maxlen argument was specified, the largest possible sequence length is maxlen.

y_train, y_test: list of integer labels (1 or 0).

Combining Both Approaches

- CNN + LSTM has been a popular combination for voice recognition (Bahdanau et al. 2014; Vinyals et al. 2015)
- Here's an example for encoding short protein sequences using padding…



How can we improve our model's performance?

Typical Activation and Loss Settings

Problem	Final-Layer Activation	Loss Function
Binary Classification	sigmoid	binary_crossentropy
Multiple Classes but One Label	softmax	categorical_crossentropy
Multiple Classes and Multiple Labels	sigmoid	binary_crossentropy
Logistic Regression (arbitrary numbers)	NONE	Mean Square Error (mse)
Regression (numbers 0-1)	sigmoid	mse or binary_crossentropy

Source: Chollet and Allaire “Deep Learning with R” pp.105, 2018.

Three More Ways to (Possibly) Improve …

- **More Data:** Are we under fitting our data? If you have access to more observations use them! [Requires more training and memory]

- **Regularization:** Sometimes this can help your model avoid “oddball” examples in your training set and better generalize. You can add this as a parameter to most layers in Keras. [May require more training]

- **Dropout:** Randomly zero-out a percentage of layer- can help avoid outliers and improve generalization. Another parameter you can add to many Keras layers. [May require more training]

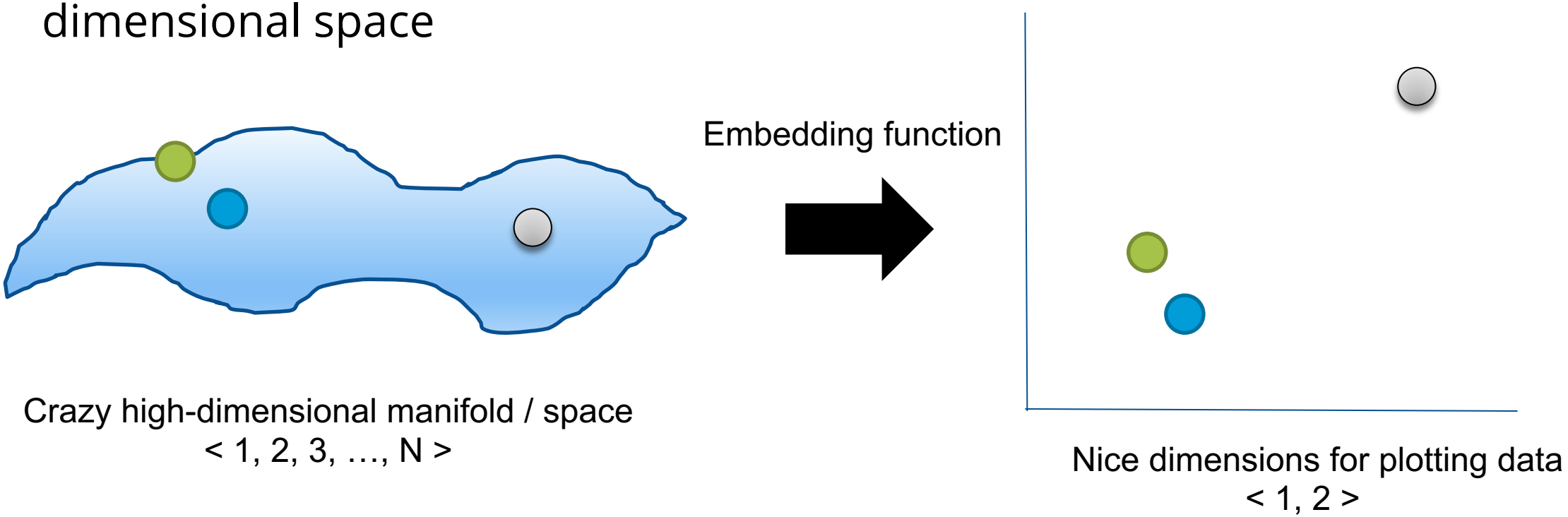
Resources to Continue Learning

- Books:
  - Deep Learning with R by Francois Chollet and J. J. Allaire (2018)
  - Deep Learning in Python by Francois Chollet (2017)
  - Neural Networks and Deep Learning (Free ebook: http://neuralnetworksanddeeplearning.com)
- Blogs:
  - Deep Learning Blog: http://deeplearning.net/blog/ (Lots of conference announcements)
  - Google's Deep Mind Blog: https://deepmind.com/blog/ (lots of research papers published)
  - Jason Brownlee's Site: https://machinelearningmastery.com (lots of great code examples)
- Online Competitions
  - Kaggle – You can see lots of Deep Learning (and other ML) approaches to a wide array of data sets: https://www.kaggle.com
- arXiv – This seems to be *the place* where many of the industry greats first publish
  - Check out publications by Yann LeCun, Yoshua Bengio, Geoffrey Hinton for some classic papers
  - Papers on "Adversarial Networks" are currently a hot topic

Bonus Topic: How do we visualize results?

What is dimensionality reduction (DR) again?

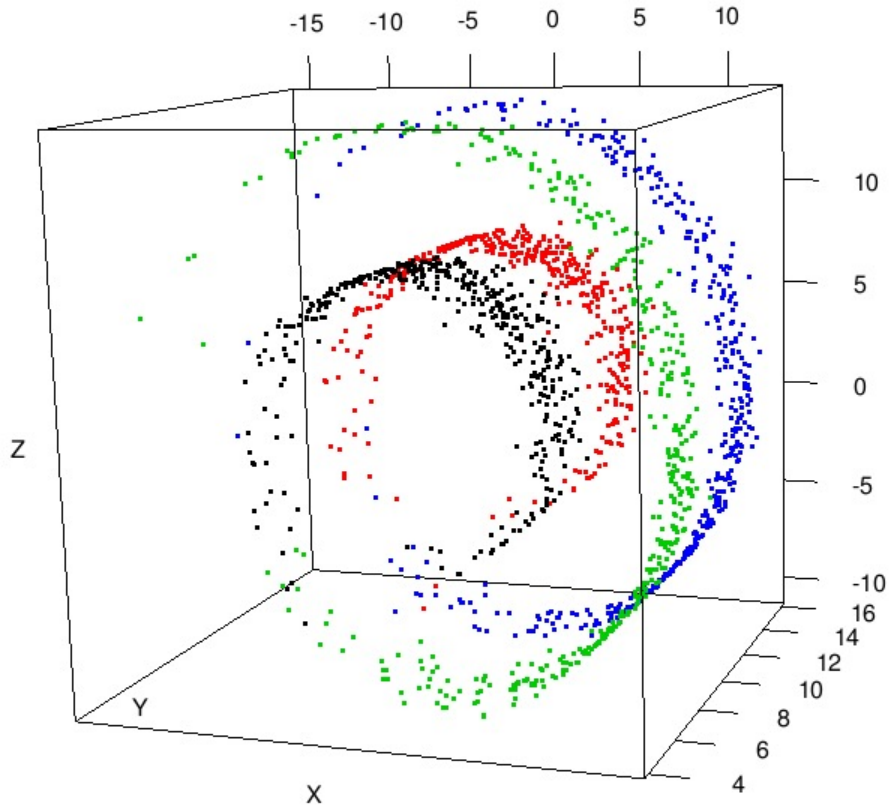
- The idea is we want to take “high dimensional” data (let us just assume this is $\mathbf{R}^d$ with d >> 3 … data that is hard to plot) and embed it in a lower dimensional space



So what's wrong with good old PCA?

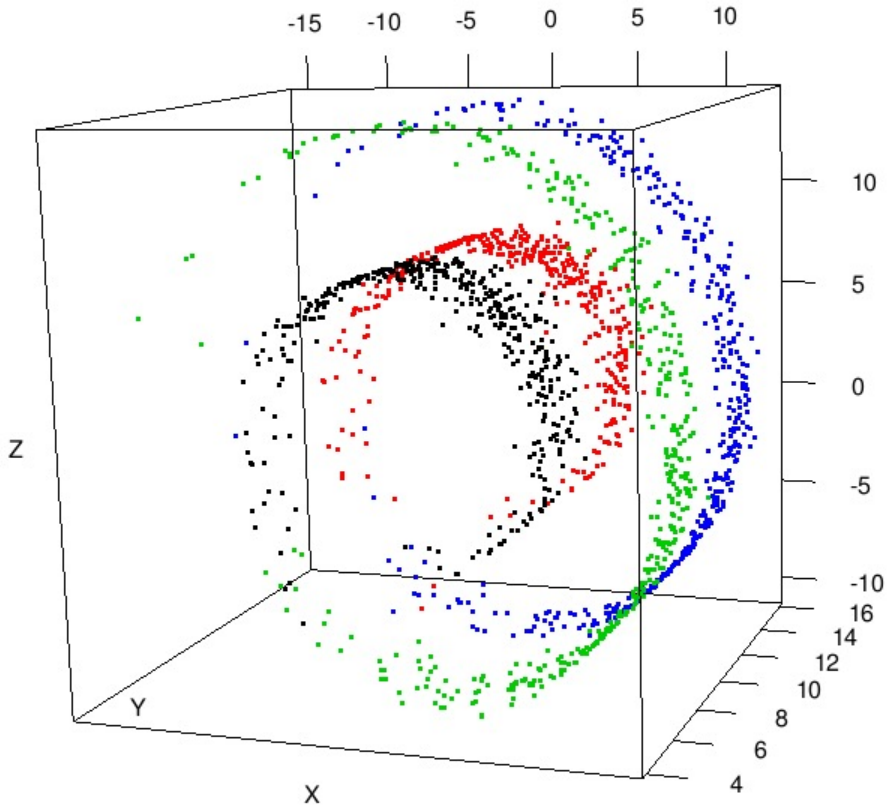
- Answer: *Nothing!* For many data sets it continues to work very well… probably still worth trying it first when you get new data
- PCA Benefits: Established track record, *allows conversion between original and PCA projections*, fast, many great software libraries available for it
- Drawbacks: It is a linear method with an objective to maximize *variance* by preserving the largest *pairwise* distances… can create ugly pictures for complex data

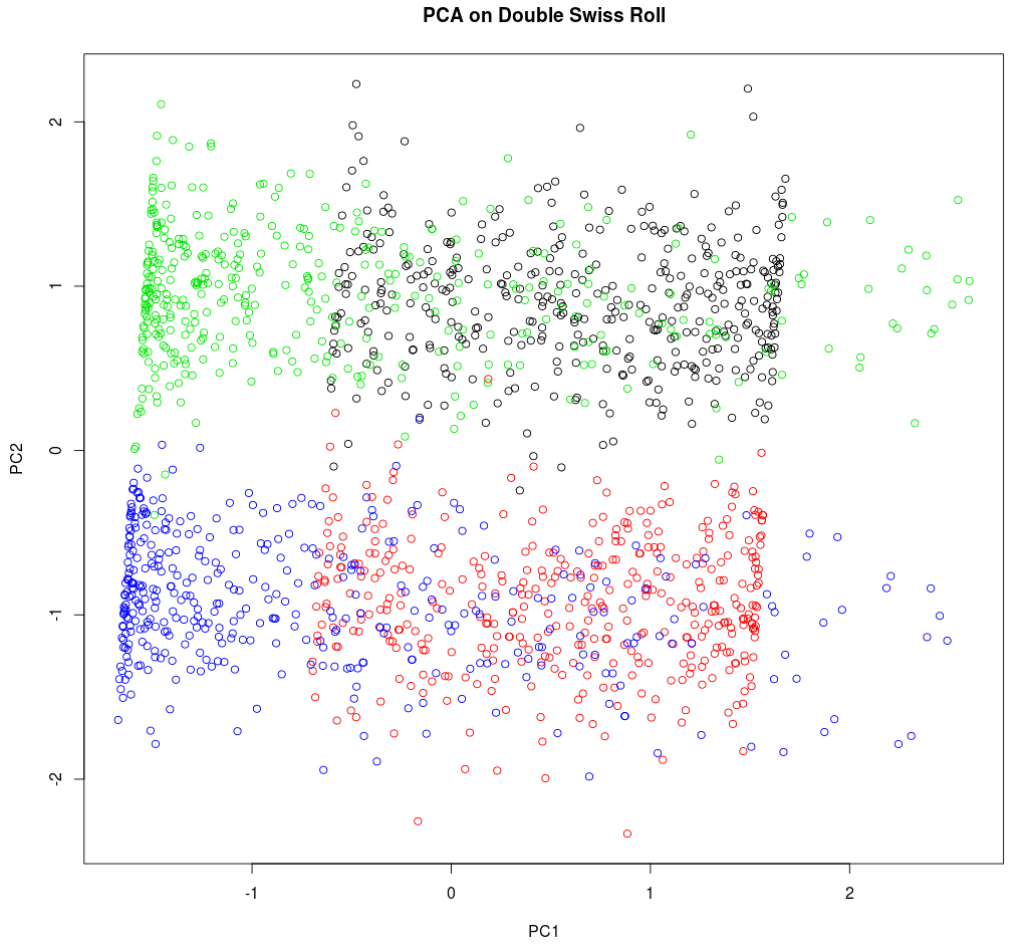
Where PCA has some trouble…



You can try this yourself! Sample swiss roll data at:
http://people.cs.uchicago.edu/~dinoj/manifold/swissroll.html

Where PCA has some trouble…





National Institute of Allergy and Infectious Diseases

NIAID

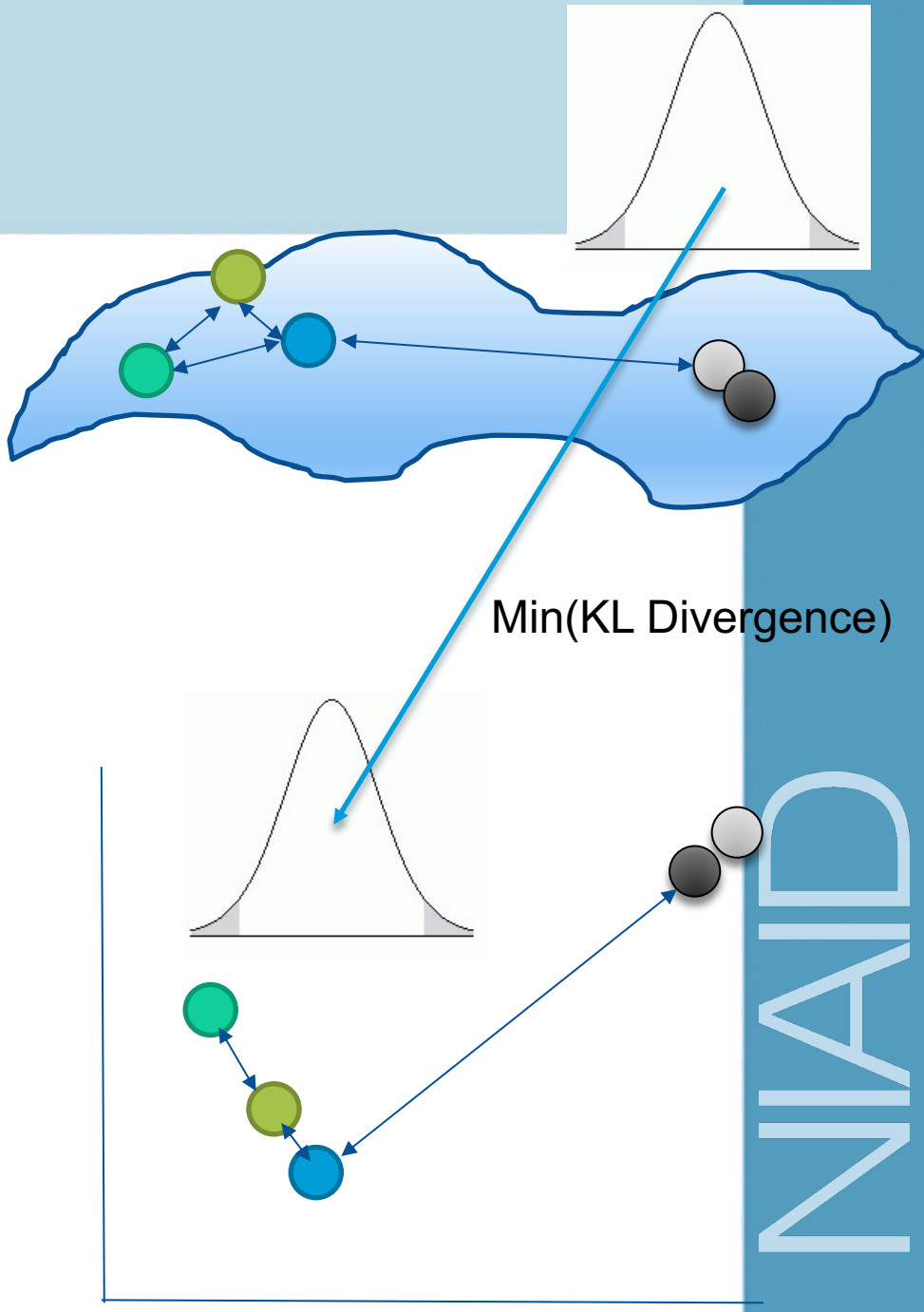
t-Distributed Stochastic Neighbor Embedding (t-SNE)

- A nonlinear embedding method that can capture *local* patterns in complex non-uniform data (the "crowding problem")
- Introduced in 2008 by Laurens van der Maaten (Formerly Delft Univ., now Facebook AI) and Geoffrey Hinton (Univ. Toronto and Google)
- Original Paper: van der Maaten and Hinton (2008) *Journal of Machine Learning Research* 9:2579-2605.
- Winner of the Merck Visualization Challenge Kaggle Competition
- Library Implementations: R, Python, Matlab, Java, CUDA, Torch, Javascript, etc.

Code available at: https://lvdmaaten.github.io/tsne/

Basic concept of t-SNE

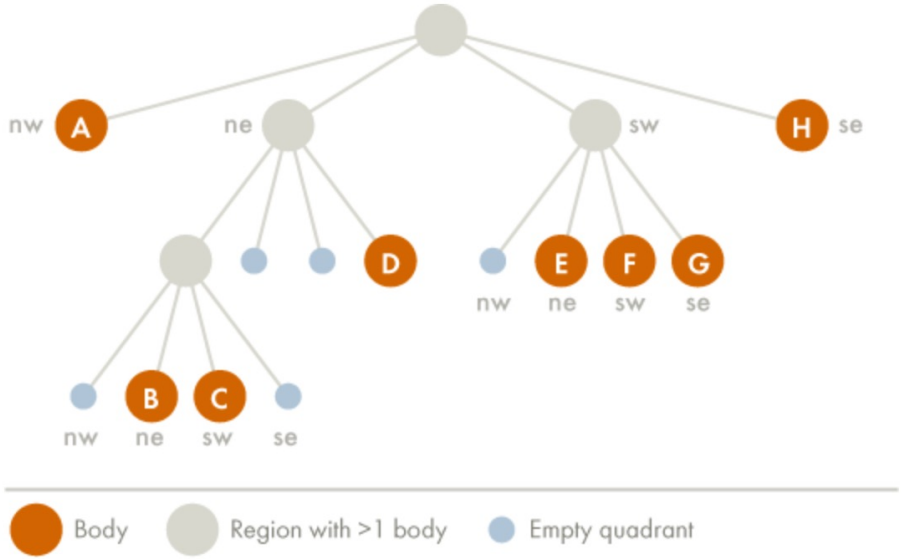
- First, identify local neighborhoods in the original space (think k-nearest neighbors) by building a *probability distribution* using a t-Student Gaussian to represent the pairwise distances of points
- The distribution identifies points that have a higher probability of being close to each other in the *original* space. Tails of t-Student distribution help where groups of data points have variable density
- To map to our new space, minimize the Kullback-Leibler (KL) divergence of the original and new embedded point distributions
- Mapping is stochastic, ***so the algorithm may produce different figures each run***, but the KL divergence should be consistent.
- Unlike PCA, ***t-SNE does not have a clear way to map new data to the embedding space***- van der Maaten recommends training a multivariate regressor to map new points.



Barnes-Hut approximation (BH-SNE)

- t-SNE is rather slow $O(N^2)$ and limited to smaller data sets (a few thousand points)
- Van der Maaten found a way to greatly speed up the algorithm using the “Barnes-Hut” transformation based on quad trees to get runtime down to O(N log N)
- Using this implementation we can handle tens of thousands of points, perhaps hundreds of thousands, but the algorithm is still limited depending on your concept of “big data”
- BH Implementation Paper: LJP van der Maaten (2014) *Journal of Machine Learning Research* 15(Oct):3221-3245.

Each external node represents a single body. Each internal node represents the group of bodies beneath it, and stores the center-of-mass and the total mass of all its children bodies. Here is an example with 8 bodies:



National Institute of Allergy and Infectious Diseases

How you initialize values is important!

- Important read:

nature biotechnology

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › nature biotechnology › matters arising › article

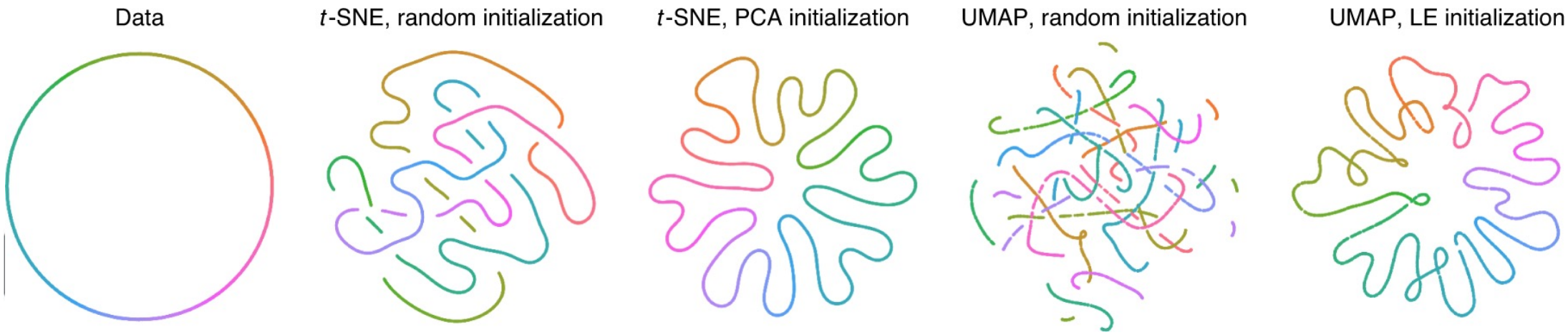
Matters Arising | Published: 01 February 2021

Initialization is critical for preserving global data structure in both *t*-SNE and UMAP

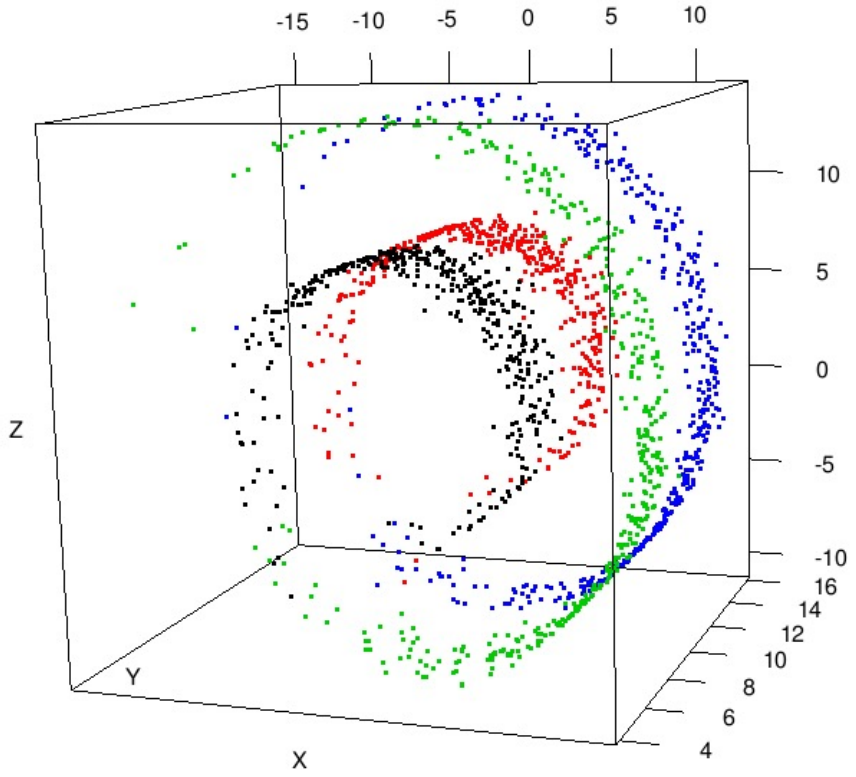
Dmitry Kobak ✉ & George C. Linderman ✉

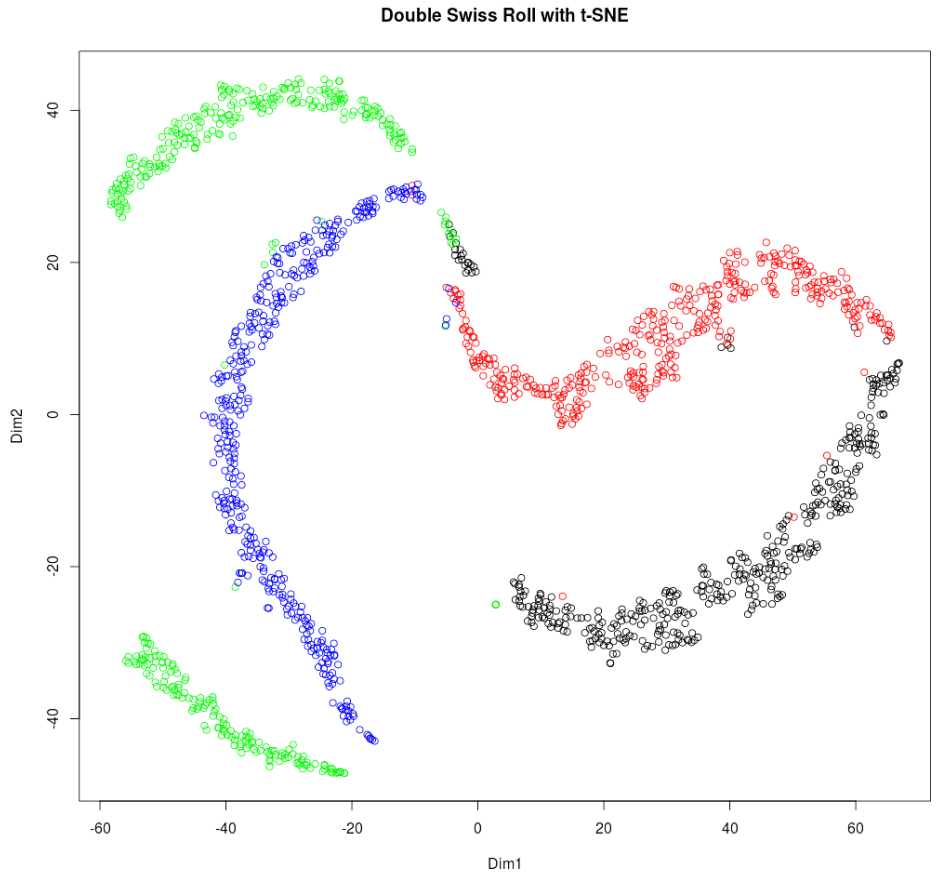
Nature Biotechnology **39**, 156–157 (2021) | Cite this article

11k Accesses | **13** Citations | **218** Altmetric | Metrics



t-SNE on the double swiss roll…

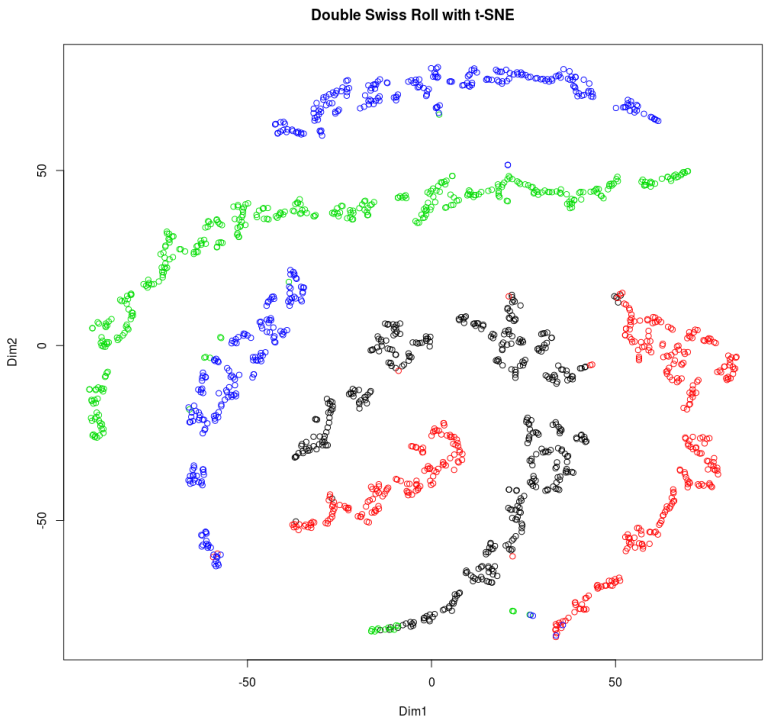




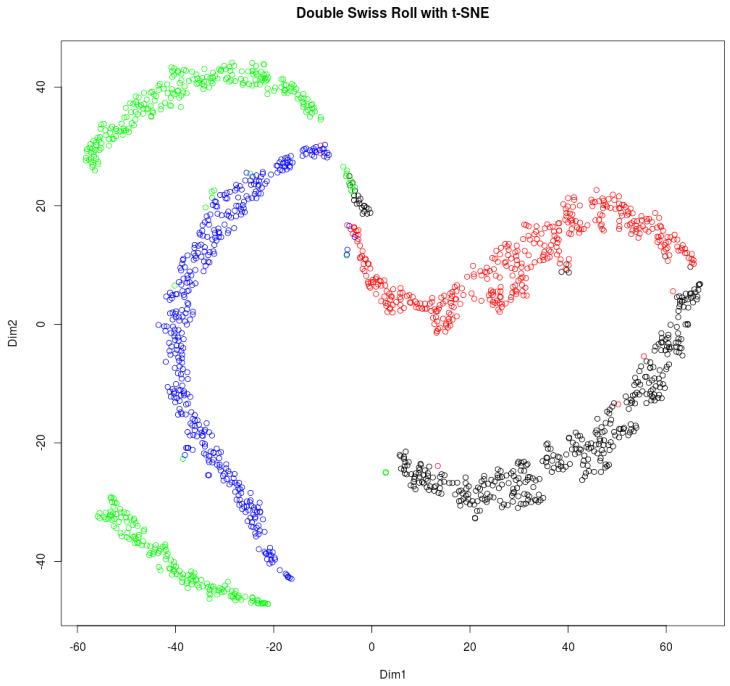
National Institute of Allergy and Infectious Diseases

NIAID

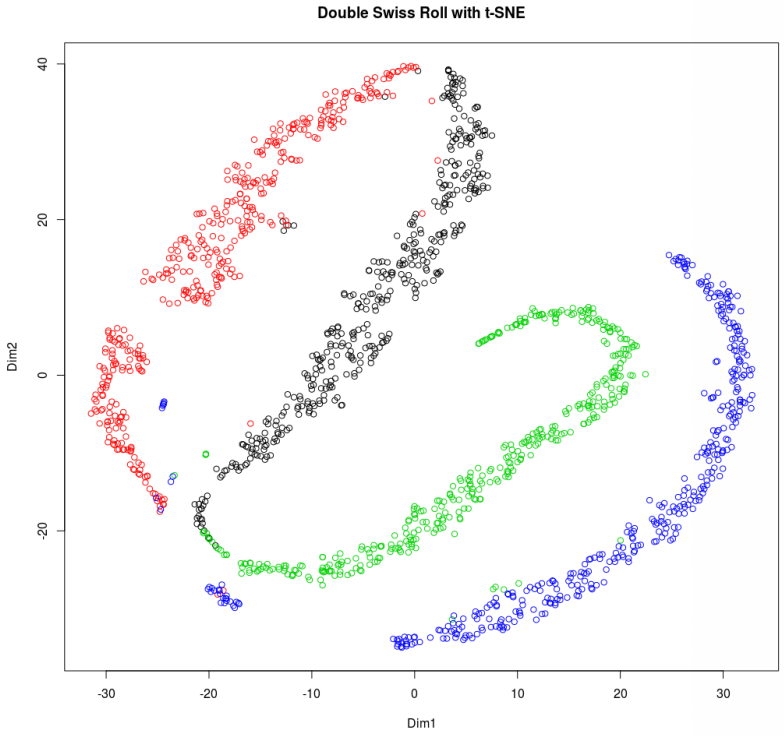
Trying some different parameters…



Theta = 0 (original t-SNE)
Perplexity = 10



Theta = 0 (original t-SNE)
Perplexity = 30 (default)



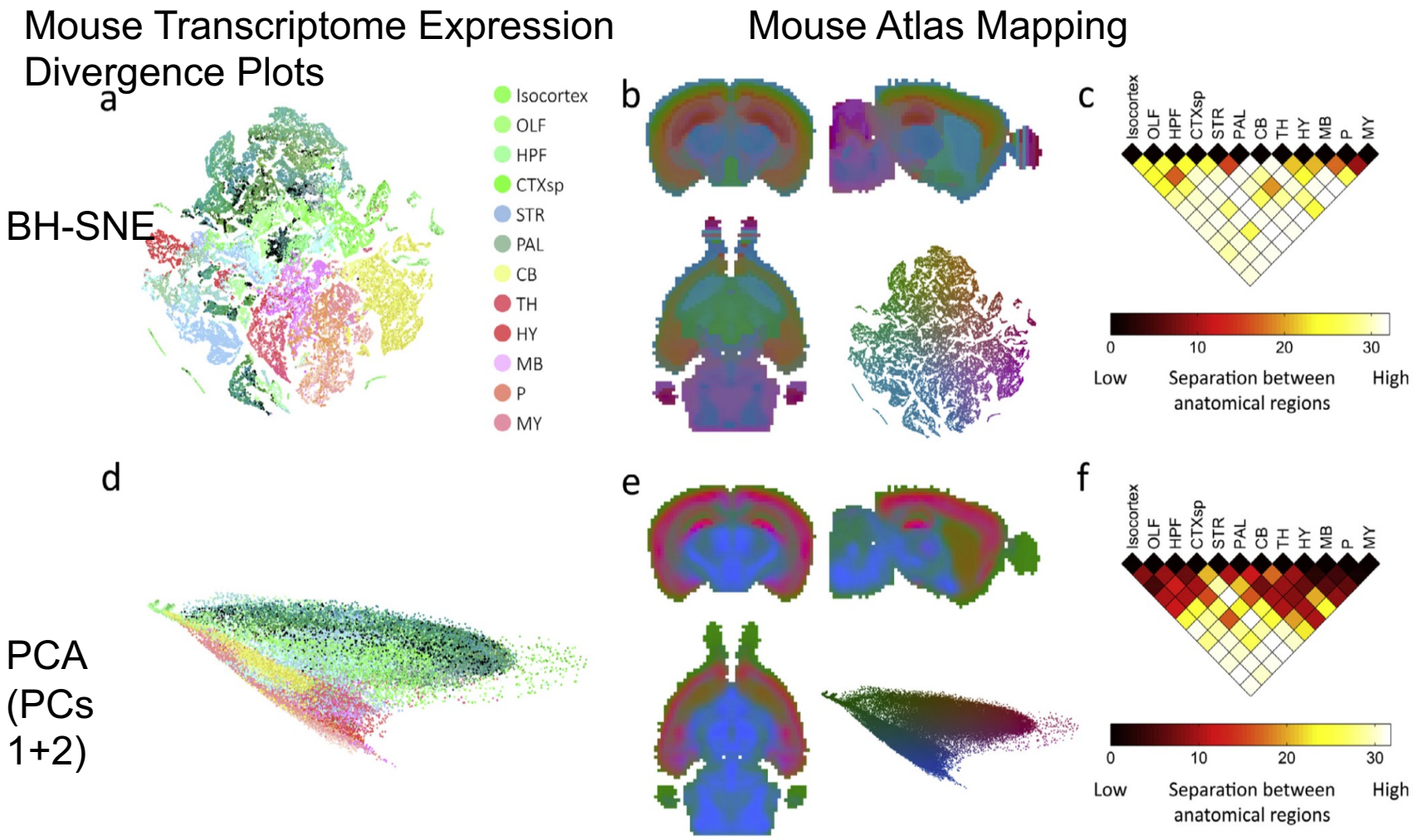
Theta = 0.5 (BH default)
Perplexity = 30 (default)

National Institute of Allergy and Infectious Diseases

NIAID

A look at some biological data from Mahfouz et al. 2015

- Mahfouz et al. compared linear and non-linear embedding methods to visualize brain transcriptome data
- They first considered data from the ALLEN Mouse Brain Atlas (~20K genes profiled)
- They also collected new data from 6 human brains and mapped to the ALLEN Human Brain Atlas (~29K genes profiled)

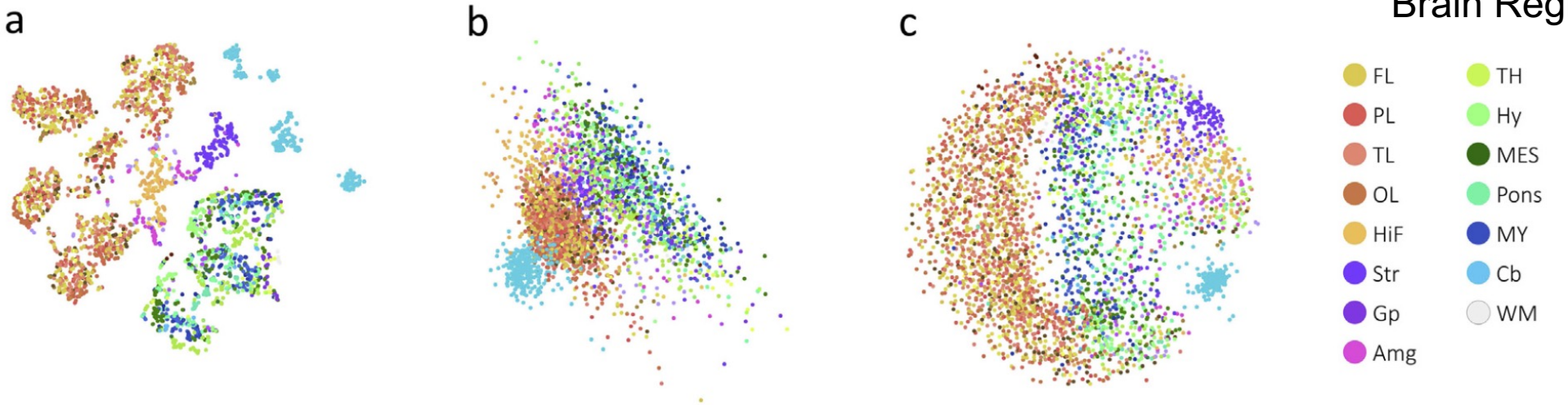


Mahfouz et al. (2015) *Methods* 73:79-89

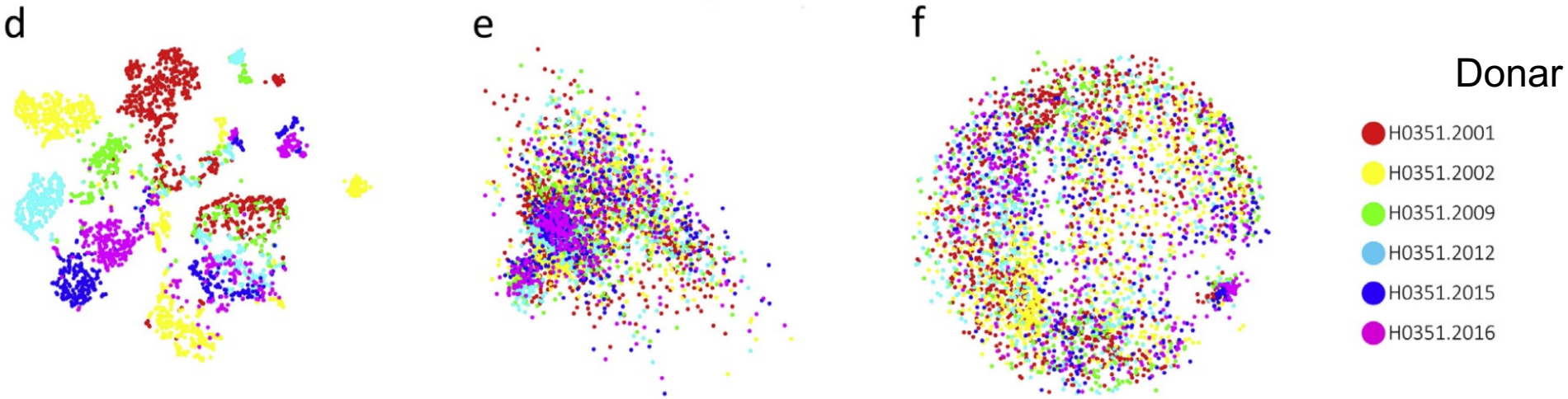
Embeddings for the human brain data

BH-SNE | PCA | Multidimensional Scaling

Aggregated gene expression levels colored by region:



Re-colored by brain donor:



NIH National Institute of Allergy and Infectious Diseases

Mahfouz et al. (2015) *Methods* 73:79-89

Running t-SNE in Code:

t-SNE in R

```
library(tsne)

X <- matrix(c(0,0,1,1,0,1,0,1,0,1,1,1),nrow=4,ncol=3)

dim(X) => [4,4]

X_2d <- tsne(X, k=2)

dim(X) => [4, 2]



# Get model layer weights for input

weights <- get_layer(model, layer_name)$output
```

t-SNE in Python

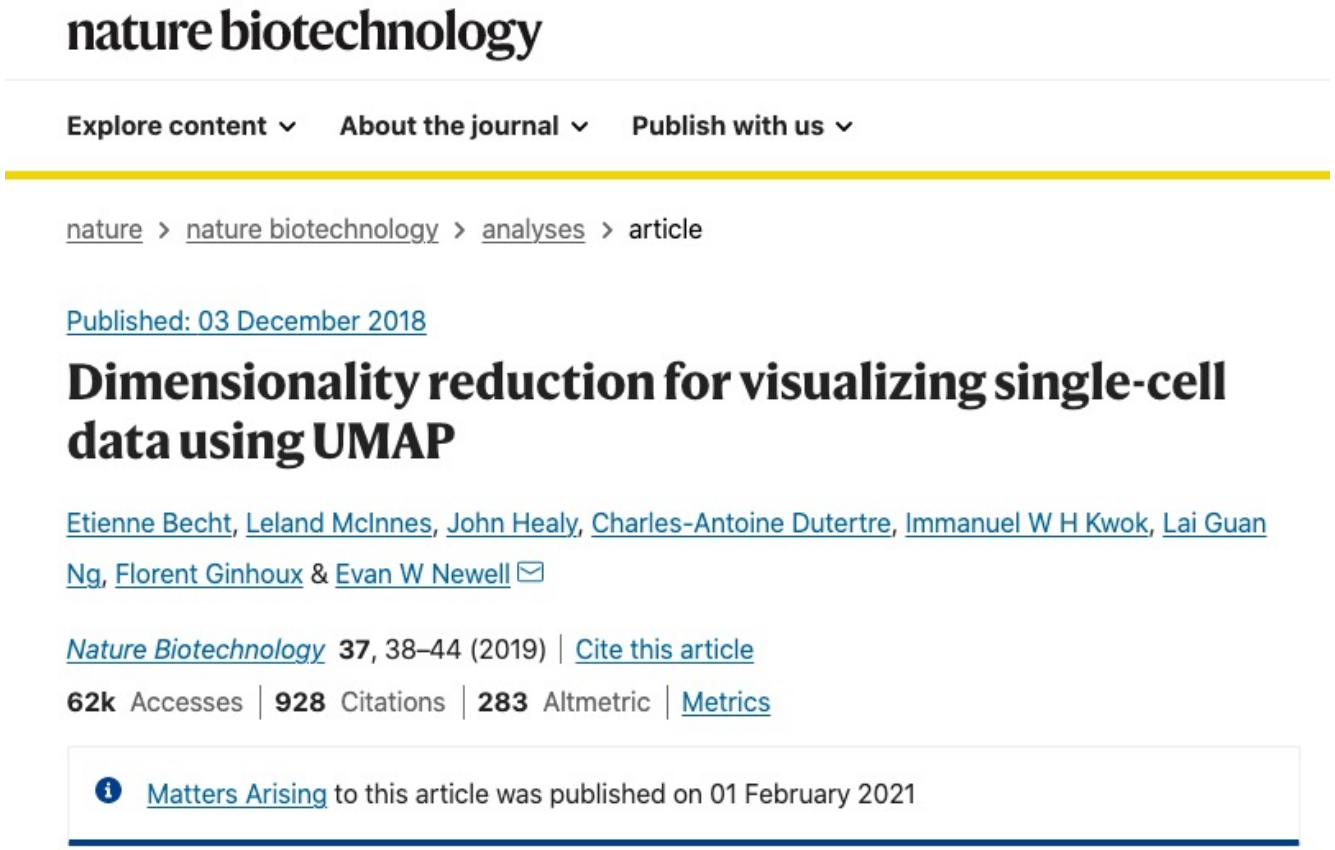
```
import numpy as np
from sklearn.manifold import TSNE
X = np.array([[0,0,0],[0,1,1],[1,0,1],[1,1,1]])
X.shape => (4,4)
X_2d=TSNE(n_components=2).fit_transform(X)
X_2d.shape => (4,2)

# Get model layer weights for input
weights = model.layers[0].get_weights()[0]
```

Some newer DR methods to check out…

- Uniform Manifold Approximation (UMAP)

Initialization is also important here like with t-SNE!



nature biotechnology

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › nature biotechnology › analyses › article

Published: 03 December 2018

Dimensionality reduction for visualizing single-cell data using UMAP

Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel W H Kwok, Lai Guan Ng, Florent Ginhoux & Evan W Newell ✉

Nature Biotechnology **37**, 38–44 (2019) | Cite this article

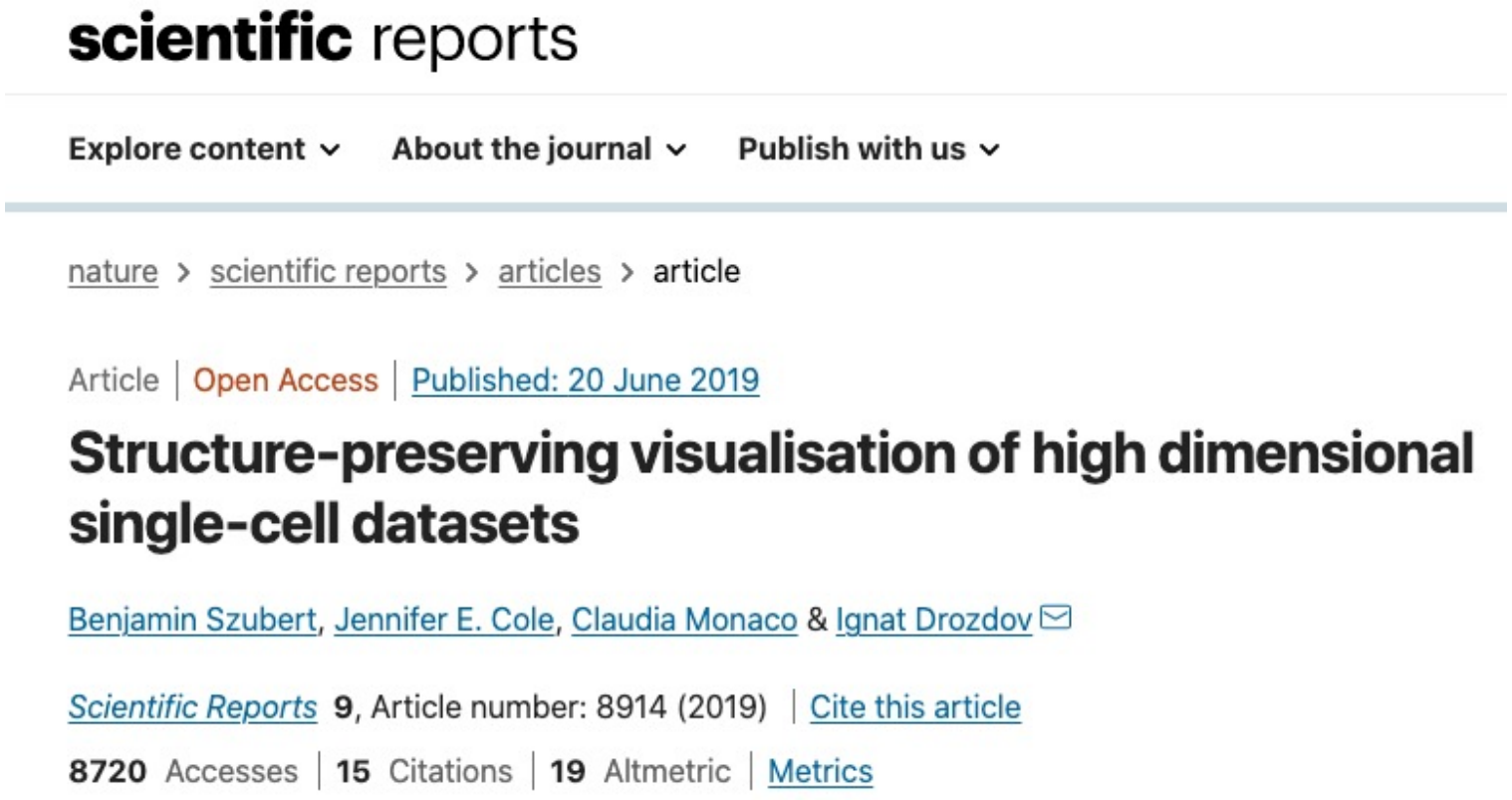
62k Accesses | **928** Citations | **283** Altmetric | Metrics

ⓘ Matters Arising to this article was published on 01 February 2021

Some newer DR methods to check out…

- Ivis algorithm

  Comparing ivis to UMAP: https://dpfoose.github.io/posts/2019/10/29/Ivis_vs_UMAP.html



scientific reports

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › scientific reports › articles › article

Article | Open Access | Published: 20 June 2019

Structure-preserving visualisation of high dimensional single-cell datasets

Benjamin Szubert, Jennifer E. Cole, Claudia Monaco & Ignat Drozdov ✉

Scientific Reports **9**, Article number: 8914 (2019) | Cite this article

8720 Accesses | **15** Citations | **19** Altmetric | Metrics