Big Data
Data Science
Deep Learning
for Statisticians

Ming Li @ Amazon
mli@alumni.iastate.edu

Hui Lin @ Netlify
hui@linhui.org

Links to Notebooks, Books and URLs

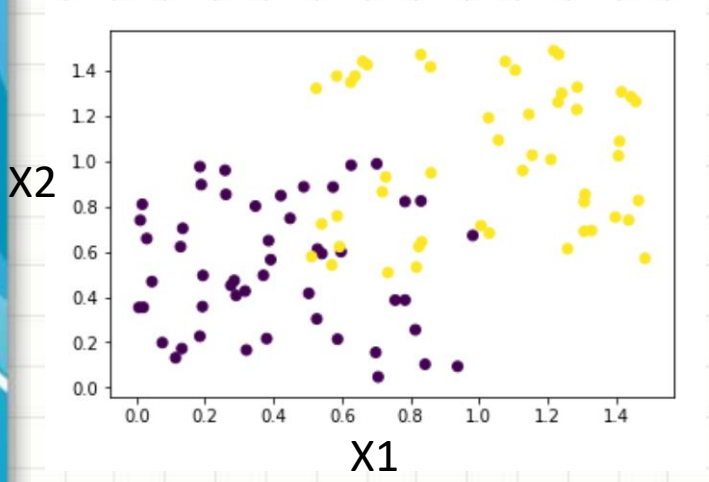
1. Course homepage: https://course2019.netlify.com/
2. Databrick free community edition account open: link
3. Perceptron notebook: link
4. Adaline notebook: link
5. Feedforward neural network notebook: link
6. Convolutional neural network notebook: link
7. Recurrent neural network notebook: link
8. Big Data Platform notebook: link
9. Data preprocessing notebook: link
10. Data wrangling notebook: link
11. Industry recommendations for academic data science programs: link
12. Deep Learning Using R, François Chollet with J. J. Allaire, ISBN 9781617295546 (2018)
13. Python Machine Learning by Sebastian Raschka, ISBN-13: 978-1787125933 (2018)
14. https://keras.rstudio.com/
15. http://spark.rstudio.com/
16. https://databricks.com/spark/about
17. https://github.com/onnx/onnx

Deep Learning

The New Tool in Data Scientist's Tool Box

A Little Bit of History – Perceptron

- Fun video: https://www.youtube.com/watch?v=cNxadbrN_aI
- Classification of N points into 2 classes: -1 and +1 (i.e. two different colors in the picture below)
- In this example below, only two features to use (X1 and X2)



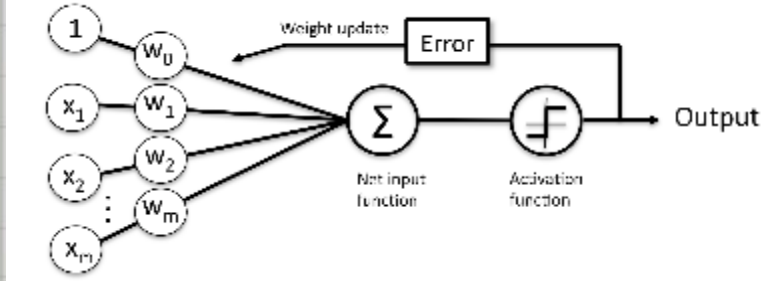
- **Linear functions to separate classes, to find (w_0, w_1, w_2) such that:**

$$\phi_j = w_0 + w_1 x_{1,j} + w_2 x_{2,j}$$

$$Pred_j = \begin{cases} 1, if\ \phi_j > 0 \\ -1, if\ \phi_j \leq 0 \end{cases}$$

- **How to find a good line? Perceptron algorithm:**
  - **Start with random weights**
  - **For each data point**
    1. **Predict class label**
    2. **Update weights when prediction is not correct using a preset learning rate and the value of features of that data point, for example for w_1:**

$$w_1 = w_1 + \eta\left(Actual_j - Pred_j\right)x_{1,j}$$



* From Sebastian *Python Machine Learning*

- **Perceptron algorithm is easy to be implemented in any modern program language.**
- **It is a linear classification function and the weight is updated after each data points are feed to the algorithm (concept similar to stochastic gradient descent).**
- **The algorithm continues to update when we feed the same dataset again and again (i.e. epochs)**
- **It is not going to converge for none-linearly spreadable problems.**

A Little Bit of History – Perceptron

For i in 1:M: **We set a maximum of epochs of M to run.**

For j in 1:N:

$$\phi_j = w_0 + w_1 x_{1,j} + w_2 x_{2,j}$$

$$Pred_j = \begin{cases} 1, if\ \phi_j > 0 \\ -1, if\ \phi_j \leq 0 \end{cases}$$

$$w_0 = w_0 + \eta\left(Actual_j - Pred_j\right)$$
$$w_1 = w_1 + \eta\left(Actual_j - Pred_j\right)x_{1,j}$$
$$w_2 = w_2 + \eta\left(Actual_j - Pred_j\right)x_{2,j}$$

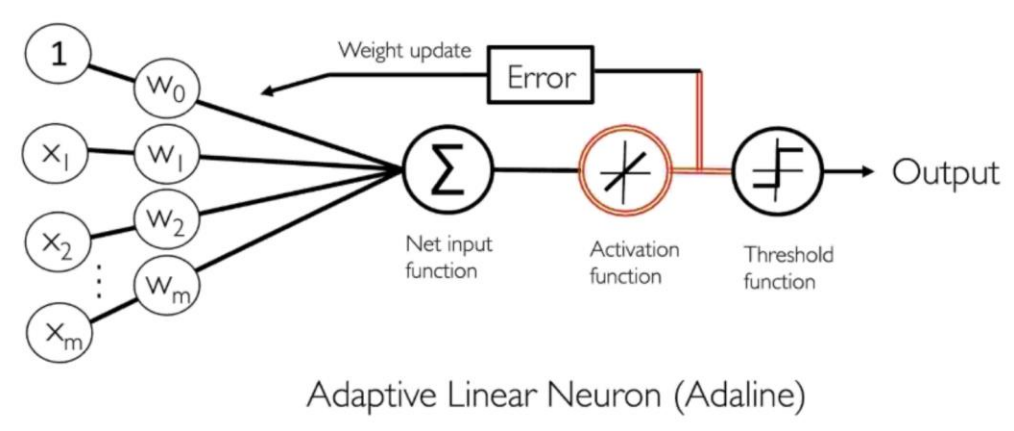
For every data point, we update the weight based on the prediction correctness, learning rate and feature values.

Calculate accuracy for the entire dataset to see whether the criteria has meet after each epoch.

For not linearly separable dataset, we need to use some accuracy threshold to stop the algorithm.

Perceptron R notebook: link

A Little Bit of History – Adaline



Adaptive Linear Neuron (Adaline)

* From Sebastian *Python Machine Learning*

Adaline R notebook: link

- We can use the error for the entire dataset as the loss function (i.e. SSE): $\sum_{j=1}^{N}(Actual_j -$

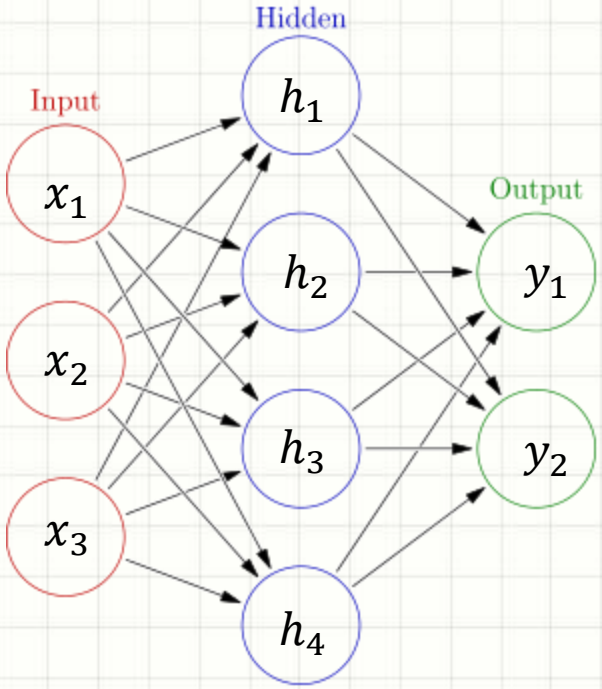
- Very similar to Perceptron and the only difference is that the error is calculated based on $\frac{1}{2}(Actual_j -$

$$w_0 = w_0 + \eta(Actual_j - \phi_j)$$
$$w_1 = w_1 + \eta(Actual_j - \phi_j)x_{1,j}$$
$$w_2 = w_2 + \eta(Actual_j - \phi_j)x_{2,j}$$

- When calculating prediction accuracy, we still based on whether ϕ_j is larger than zero or not with the final weight learned from the data.

Feed Forward Neural Network

Simple Feed Forward Neural Network (FFNN)



Concepts stated as early as 1940s!

$$h_j = f_1\left(b_{j0} + \sum_{i=1}^{3} b_{ji}x_i\right), j = 1,2,3,4$$

$$y_k = f_2\left(b_{k0} + \sum_{j=1}^{4} b_{kj}h_j\right), k = 1,2$$

$$\boldsymbol{y} = f_2(f_1(\boldsymbol{x}, \boldsymbol{b}_1), \boldsymbol{b}_2)$$ ***Vector representation***

$[x_1, x_2, x_3]$: Input features vector for one observation

$[y, y_2\ ]$: Actual output outcome for one observation

$f_1(\cdot), f_2(\cdot)$: Activation functions, usually non-linear

$L(y, \hat{y}(\boldsymbol{b}, \boldsymbol{x})\ )$: Loss function where $\hat{y}$ the model forecast responses and y actual observed responses

Total number of parameters (i.e. size of a NN): (3+1)*4 + (4+1)*2 = 26

Typical Loss Functions

- Two-class binary responses
  - ✓ Binary cross-entropy:

$$\sum_{i=1}^{N}\left(-y_i log(p_i) - (1-y_i) log(1-p_i)\right)$$

where y_i is actual value of 1 or 0, p_i is the predicted probability of being 1, and N is the total number of observations in the training data.

- Multiple-class responses
  - ✓ Categorical cross-entropy for M classes:

$$\sum_{i=1}^{N}\left(-\sum_{j=1}^{M} y_{i,j} log(p_{i,j})\right)$$

where $y_{i,j}$ is actual value of 1 or 0 for a class of j, $p_{i,j}$ is the predicted probability of being at class j and N is the total number of observations in the training data.

- Continuous responses
  - ✓ Mean square error
  - ✓ Mean absolute error
  - ✓ Mean absolute percentage error

From Slow Progress to Wide Adoption

1940s – 1980s, very slow progress due to:

- ❑ **Computation hardware capacity limitation**
- ❑ **Number of observations with labeled results in the dataset**
- ❑ **Lack efficient algorithm to estimate the parameters in the model**

1980s – 2010, a few applications in real word due to:

- ❑ **Moore's Law + GPU**
- ❑ **Internet generated large labeled dataset**
- ❑ **Efficient algorithm for optimization (i.e. SGD + Backpropagation)**
- ❑ **Better activation functions (i.e. Relu)**

2010-Now, very broad application in various areas:

- ❑ **Near-human-level image classification**
- ❑ **Near-human-level handwriting transcription**
- ❑ **Much improved speech recognition**
- ❑ **Much improved machine translation**

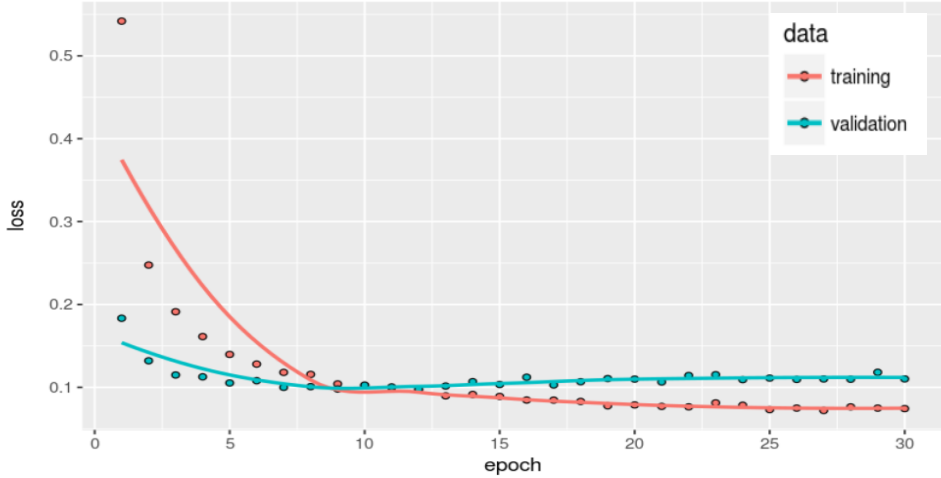
Now we know working neural network models usually contains many layers (i.e. the depth of the network is deep), and to achieve near-human-level accuracy the deep neural network need huge training dataset (for example millions of labeled pictures for image classification problem).

Optimization Methods

- Mini-batch Stochastic Gradient Descent (SGD)
  - Use **a small segment of data** (i.e. 128 or 256) to update the SGD parameters: $b = b - \alpha \nabla_b L(b, x^s, y^s)$ where α is the **learning rate** which is a hyper parameter that can be changed
  - Gradients are efficiently calculated using **backpropagation** method
  - When the entire dataset are used to updated the SGD, it is called one **epoch** and multiple epochs are needed to run to reach convergence
  - An updated version with 'momentum' for quick convergence:

$$v = \gamma v + \alpha \nabla_b L(b, x^s, y^s)$$
$$b = b - v$$

  - The optimal number for epoch is determined by when the model is not overfitted (i.e. validation accuracy reaches the best performance).



More Optimization Methods

- With Adaptive Learning Rates
  - **Adagrad**: Scale learning rate inversely proportional to the square root of the sum of all historical squared values of the gradient (stored for every weight)
  - **RMSProp**: Exponentially weighted moving average to accumulate gradients
  - **AdaDelta**: Uses sliding window to accumulate gradients
  - **Adam** (adaptive moments): momentum integrated, bias correction in decay
- A good summary: http://ruder.io/optimizing-gradient-descent/

Activation Functions

$$\boldsymbol{y} = f_n(f_{n-1}(f_{n-2}(\dots f_1(\boldsymbol{x}, \boldsymbol{b}_1) \dots, \boldsymbol{b}_{n-2}), \boldsymbol{b}_{n-1}), \boldsymbol{b}_n)$$

Gradient: $$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial f_n} \frac{\partial f_n}{\partial f_{n-1}} \frac{\partial f_{n-2}}{\partial f_{n-1}} \dots \frac{\partial f_1}{\partial \boldsymbol{x}}$$

- Intermediate layers
  - Relu (i.e. rectified linear unit) is usually a good choice which has the following good properties: (1) fast computation; (2) non-linear; (3) reduced likelihood of the gradient to vanish; (4) Unconstrained response
  - Sigmoid, studied in the past, not as good as Relu in deep learning, due to the gradient vanishing problem when there are many layers
- Last layer which connects to the output
  - Binary classification: sigmoid with binary cross entropy as loss function
  - Multiple class, single-label classification: softmax with categorical cross entropy for loss function
  - Continuous responses: identity function (i.e. y = x)

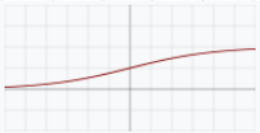
Rectified linear unit (ReLU)[10]

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

Softmax

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad \text{for } i = 1, \dots, J$$

Logistic (a.k.a. Sigmoid or Soft step)



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

Deal With Overfitting

- Huge number of parameters, even with large amount of training data, there is a potential of overfitting
  - Overfitting due to size of the NN (i.e. total number of parameters)
  - Overfitting due to using the training data for too many epochs
- Solution for overfitting due to NN size
  - Dropout: randomly dropout some proportion (such as 0.3 or 0.5) of nodes at each layer, which is similar to random forest concept
  - Using L1 or L2 regularization in the activation function at each layer
- Solution for overfitting due to using too many epochs
  - Run NN with large number of epochs to reach overfitting region through cross validation from training/validation vs. epoch curve
  - Choose the model with number of epochs that have the minimum validation accuracy as the final NN model

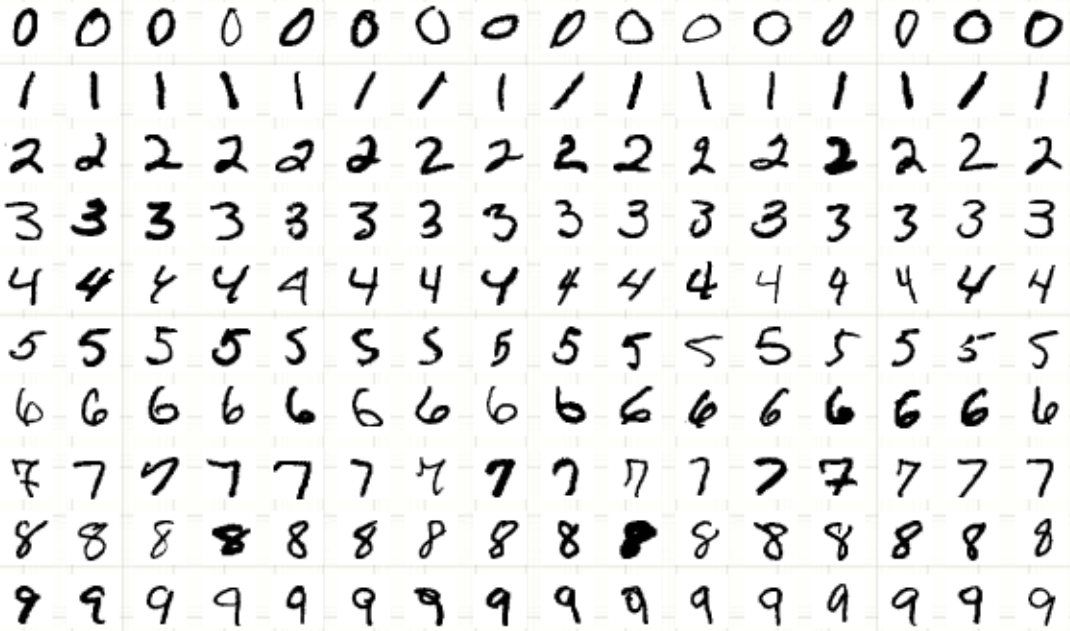
Recap of A Few Key Concepts

- **Data**: Require large well-labeled dataset
- **Computation**: intensive matrix-matrix operation
- **Structure** of fully connected feedforward NN
  - Size of the NN: total number of parameters
  - Depth: total number of layers (this is where deep learning comes from)
  - Width of a particular layer: number of nodes (i.e. neurons) in that layer
- **Activation function**
  - Intermediate layers
  - Last layer connecting to outputs
- **Loss function**
  - Classification (i.e. categorical response)
  - Regression (i.e. continuous response)
- **Optimization methods (SGD)**
  - Batch size
  - Learning rate
  - Epoch
- **Deal with overfitting**
  - Dropout
  - Regularization (L1 or L2)

The MNIST Dataset

MNIST Dataset

- Originally created by NIST, then modified for machine leaning training purpose
- Contains 70000 handwritten digit images and the label of the digit in the image where 60000 images are the training dataset and the rest 10000 images are the testing dataset.
- Census Bureau employees and American high school students wrote these digits
- Each image is 28x28 pixel in greyscale
- Yann LeCun used convolutional network LeNet to achieve < 1% error rate at 1990s



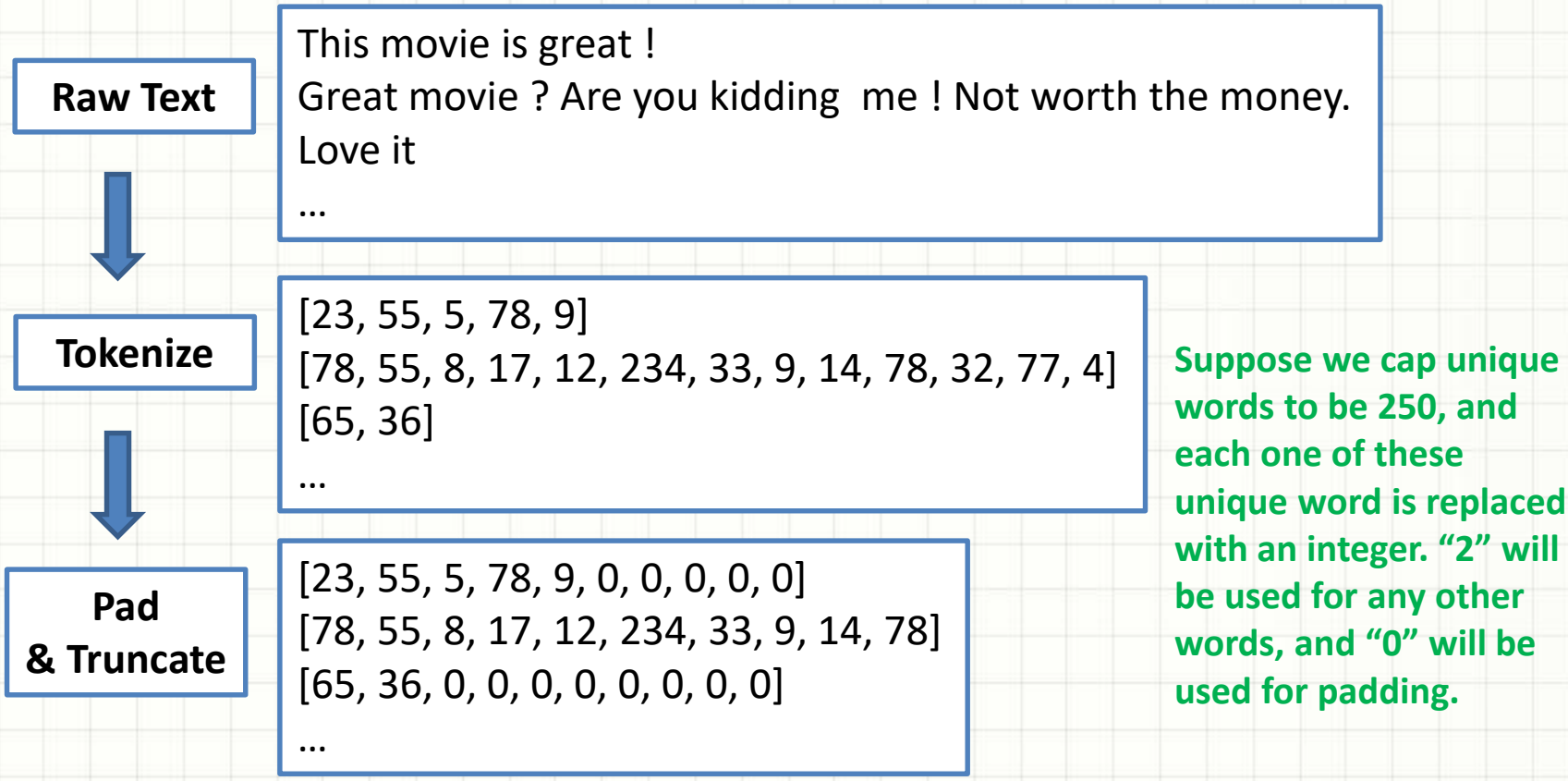
The IMDB Dataset

IMDB Dataset

- Raw data: 50,000 movie review text (X) and it's corresponding sentiment of positive (1, 50%) or negative (0, 50%) (Y).
- Included in Keras package, can be easily loaded and preprocessed
- Preprocessing includes:
  - Set size of the vocabulary (i.e. N most frequently occurring words)
  - Set length of the review by padding using '0' by default or truncating as we have to have same length for all reviews for modeling
  - Any words not in the chosen vocabulary replaced by '2' by default
  - Words are indexed by overall frequency in the chosen vocabulary
- Once the dataset is preprocessed, we can apply encoding or embedding and then feed the data to FFNN or RNN

IMDB Dataset - Tokenization

Algorithm cannot deal with raw text and we have to convert text into numbers for machine learning methods.

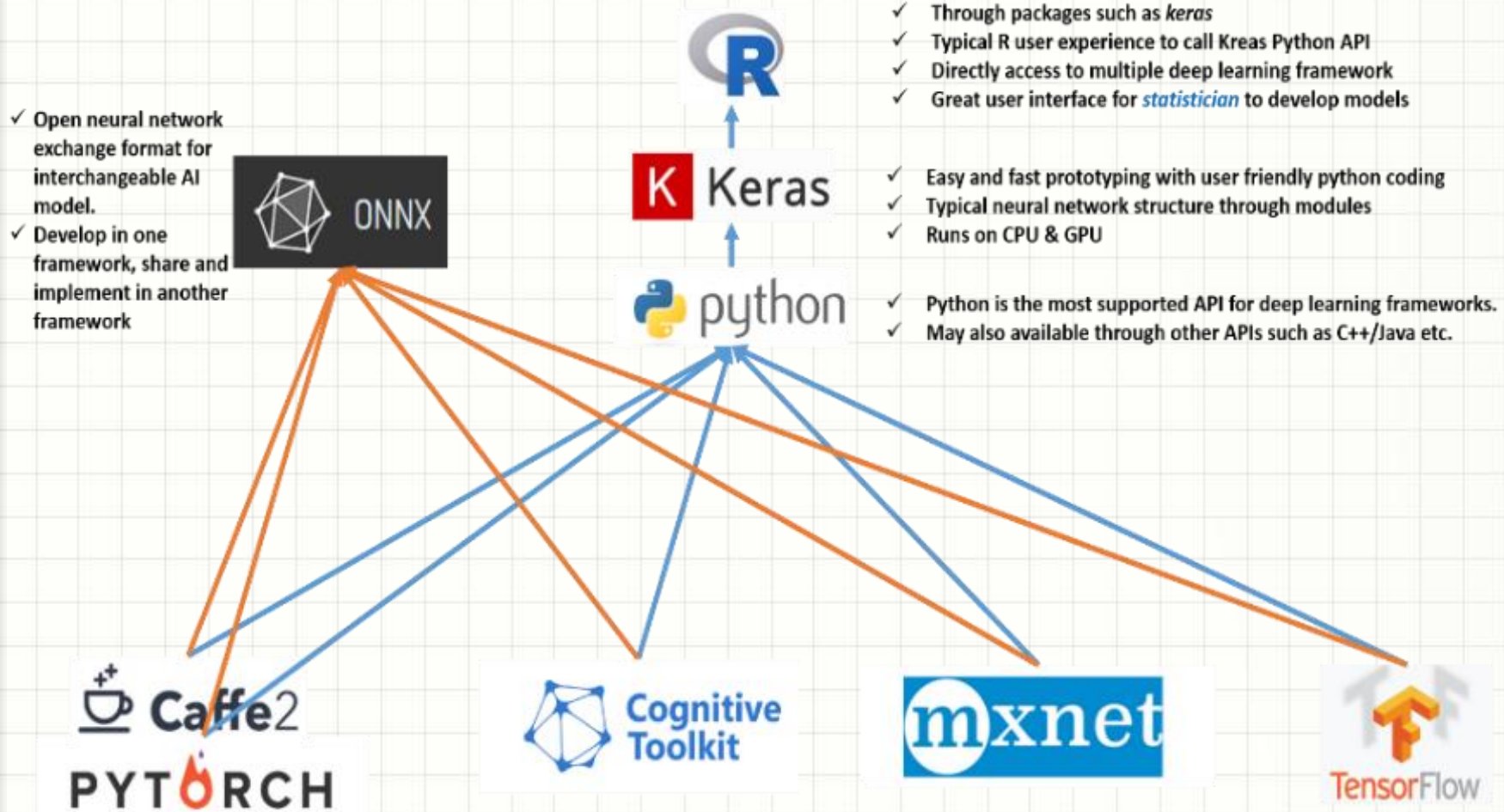


Suppose we cap unique words to be 250, and each one of these unique word is replaced with an integer. "2" will be used for any other words, and "0" will be used for padding.

Now we have a typical data frame, each row is an observation, and each column is a feature. Here we have 10 columns by designing after the padding and truncating stage. We have converted raw text into categorical integers.

Deep Learning Models Across Platforms

Open NN Exchange Format (ONNX)



https://github.com/onnx/onnx

THANK YOU!