

**PhUSE US Connect 2020**  
**Paper ML12**  
**R for Deep Learning with Tensorflow™**  
**with Applications in Cancer Immunotherapy**

Phil Bowsher, RStudio; Boston, USA

## **Abstract**

You can access the examples below live with this link:

- <https://rstudio.cloud/project/1011147>

Examples are in the **immunotherapy folder**. The project above is based on Will Landau's interactive workshop on drake. It has most of the R packages (Tensorflow and Keras) we need for the immunotherapy analysis below in this paper. If you like, you can go to the drake workshop link below:

<https://rstudio.cloud/project/627076>

Then you can add the immunotherapy examples by downloading a zip folder from here and then adding uploading it to the drake session above:

<https://github.com/philbowsher/phuse-ML12-R-Tensorflow-Immunotherapy>

Interest in deep learning is increasing in the drug development space. Many clinical statistical programmers use R and or SAS. When Tensorflow (an open source library for numerical computation created by Google) was originally released, the bindings however were in python. In 2017, Tensorflow and Keras for R were released and opened the door into deep learning for many people that use R.

The PhUSE paper, Paper TT14 (R & Python for Clinical Workflows) provides valuable information for which this paper builds on.

The information below will focus on an application of deep learning in cancer immunotherapy. The community is full of deep learning stories in pharma and healthcare but understanding how it was administered can be difficult. This paper builds on the work of Dr. Leon Eyrich Jessen, a global leader in the application of deep learning within immunology. Tensorflow examples and plumber APIs were developed by Andrie de Vries, engineer at RStudio. This paper breaks down how deep learning can be applied for a clinical workflow and helps statistical programmers that are realizing the need of deep learning to stay abreast of these emerging technologies

## **Introduction**

This paper is for clinical statistical programmers that are new to deep learning (a subset of machine learning, and both part of artificial Intelligence) and have wondered how it can be applied for clinical work. For the focus of this paper, information will be provided about the use of R and Python on a desktop machine as well as a server.

Below will discuss the current landscape for deep learning for cancer immunotherapy. Below is not an introduction to deep learning in R or Python but will provide valuable information for clinical statistical programmers using R and looking to understand a deep learning example to their knowledge base.

## Getting Up and Running

Installing Tensorflow can be challenging, especially if you plan to use high computing power like GPUs, which are usually needed. In a corporate environment, you likely will never install Tensorflow and Keras as a statistical programmer. Usually there are dedicated IT teams that provision access to environments with these tools ready to go. Also, inside a company, installing software on your laptop machine can be challenging if not permitted at all.

You may be wondering then, how do I learn? How do I get my hands into the mix of deep learning? My advice, don't install all of this from scratch, find pre-built environments to use for learning. Your time is valuable and a precious asset. I want you to spend the greatest portion of your time on the actual problems you are working to solve and minimally with the setup.

I enjoy installing software locally and playing with technology though. Building your own deep learning workstation might be a great idea as it is nice to have a free environment but motivation is needed to get Tensorflow up and running. If you would like to set up this environment for this paper, you will minimally need to install these packages using the instructions on each page:

<https://rstudio.github.io/reticulate/index.html>

<https://tensorflow.rstudio.com/installation/>

<https://Keras.rstudio.com/>

I am more interested in learning the techniques and application of software than how to install it at this point in my life. For my own deep learning use, I would likely just use one of the ready to go images from a cloud provider. After my results are generated, the notebooks or R Markdown files would be exported and the trained model sent to a version control system, or I would even just deploy the model as a REST API to a separate place altogether.

There have been advancements in computing that makes replicating software environments easier than ever. The cloud, docker, Kubernetes all have made this world more imaginable than 5 years ago and opened up a wide array of opportunities. To bad these options were not as prevalent when Spark first came out, it would have saved me days getting Hadoop and Spark running. With these advancements, it does seem that future data engineering and data architectural work could be easier. For the time being though, below are some easy ways to give the examples in this paper a test drive. The options below are meant to enhance your learning. Please consult your internal IT on how best to access deep learning tooling internally to your organization. Please note too that not all of the options are free but are certainly easier to spin up than starting off from scratch either locally or on a server:

- RStudio Cloud: I setup a RStudio Cloud instance that has Tensorflow and Keras ready to go. All you have to do to access Tensorflow, is click the link below and wait for the environment to set up. You will need an account also. At the current moment, this option is free. I tested the example below and it works well. If you click the link below, it will spin up the same environment for you to explore. Please note that the containers are limited to 1 core, and 1GB of memory. RStudio Cloud has a CPU so keep that in mind. So if you want to explore your own Tensorflow applications, this likely will not work if you need more storage or compute: <https://rstudio.cloud/project/1009743>
- RStudio Server with Tensorflow-GPU for Amazon AWS EC2: This has a GPU which is great. Please note this option is not free: <https://aws.amazon.com/marketplace/pp/B0785SXYB2>  
<https://mraess.netlify.com/2018/06/setting-up-an-aws-ec2-instance-with-rstudio-server-and-Tensorflow-gpu/>
- RStudio GPU Workstations in the Cloud with Paperspace: Cloud server/desktops with various GPU configurations are available from Paperspace. These have a NVIDIA Quadro P4000 GPU which is great. Like AWS, this option is not free and is based on your monthly or hourly usage: [https://Tensorflow.rstudio.com/installation/gpu/cloud\\_desktop\\_gpu/](https://Tensorflow.rstudio.com/installation/gpu/cloud_desktop_gpu/)
- Docker is an exciting tool for deep learning. Rocker contains Dockerfiles for different Docker containers of interest to R users. The pre-built Tensorflow example I suspect could be used with any of the major cloud

(AWS, Google, Azure) providers. This might be a little harder to setup than the other options above:  
<https://hub.docker.com/r/rocker/Tensorflow>

## **Brief Review of Paper TT14 (R & Python for Clinical Workflows)**

### **Python**

Before we dive into deep learning, it is important to understand why python is important in this space. Python is often the main programming language for those coming from the computer science/data engineering fields as python is a general-purpose programming language. The challenge for the R community, was that Tensorflow and Keras were originally released for mainly Python programmers and the R users had limited ways to work with these tools. Many of the public Python clinical examples focus heavily on machine learning and deep learning. To solve this problem, there had to be a way to manage the interoperability between python and R with a translation layer.

Therefore it is important to understand that most of the deep learning work we will look at in this paper will be python, but accessed through R. Even though the analysis will use python, we will never interact with it directly. This is done often by writing R code that is translated to python etc. This is an important part of the workflow, as R users can use the language they are familiar with while tapping into the python ecosystem for deep learning.

### **Reticulate**

Keras and Tensorflow both depend on python. For the analysis below, we are never going to directly work with python. If you need to, you may want to review anaconda as a free and open-source distribution of Python that includes many of the core python packages. conda is the package manager used for these python packages. Miniconda is a free minimal installer for conda that includes only conda, Python, the packages they depend on, and a number of other packages.

### **Short History of Deep Learning**

The 1960s-70s were an exciting time for AI as Symbolic AI grew in popularity. As a side note, it is during this time that S was created at AT&T Bell Labs by John Chambers et al.. R, first developed in 1993 was based mainly on S. 4 years prior, in 1989, Yann LeCun et al. also at AT&T Bell Labs could successfully identify handwritten zip codes using backpropagation, an important moment for deep learning. Just as R was taking shape in the 1990s, so too was deep learning. After Expert Systems failed to deliver on the hype in the corporate space, much of the momentum around AI was slowing down. But in 1995, things started to pick up with Support Vector Machines (SVM) developed by Vladimir Vapnik and co-workers, again at AT&T Bell Labs! Into the 2000s, decision trees were picking up steam, especially Random Forest around 2010 and then gradient boosting in 2014. All of these applications were mainly shallow learning methods, those that are not deep and with one or two layers. This changed significantly in 2012 during the annual ImageNet project contest, when a convolutional neural network (CNN) called AlexNet dominated and set the way for years to come. In 2017, A CNN achieved 97.7% accuracy for the top-5, ending ImageNet as it was deemed solved.

What changed in 2012 to open the door for such growth in deep learning? At this time, there was a convergence of 3 important things - hardware (Nvidia), datasets and algorithms (gradient propagation). In 2015, much of this space was being programmed in python, fueled by investments from Google, Amazon etc. At the sametime, many researchers, students, academic professionals, scientists, were using R for statistics and machine learning. Much of the work in R was being led by Max Kuhn (Director of Nonclinical Statistics at Pfizer Global R&D) with numerous R packages for techniques in machine learning, most popular being the caret and parsnip packages. Because deep learning is little math theory, and engineering oriented, the computer science space gravitated to it and therefore much of the work being done was in python. But S was originally an interface into C and fortran libraries, could we use R to interface into this space as well? In 2017, that is what happened with the release of Tensorflow and Keras for R. Keras is a very important part of the deep learning story. Programming in C++ and CUDA is challenging for many, and Keras provided a user-friendly interface into the main deep learning computing environments (Tensorflow,

theano and CNTK). Into 2016, the funding into artificial intelligence poured in and now many organizations have data science teams deploying deep learning methods.

### **Tensorflow & Keras**

This paper will focus on the use of Tensorflow and Keras for R. Tensorflow was developed by the Google Brain team for internal Google use. It was publicly released under the Apache License 2.0 on November 9, 2015 and written in Python, C++ and CUDA. Working with Tensorflow directly was challenging and to interface with Tensorflow, most people use the user-friendly Keras package, developed by François Chollet (a Google engineer) and written in Python. Keras can run on top of many deep learning deep learning computing environments like Tensorflow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML.

The challenge for the R community, was that Tensorflow and Keras were originally released for Python programmers and the R user had limited ways to work with these tools. In 2017, Tensorflow™ 1.0 for R was released on Jan 9th, 2017 and Keras for R on Sep 3, 2017. These important packages made it possible for the first time for R users to use packages that were primarily written in Python and allowed R users to utilize state of the art tools in deep learning. This was possible as the authors of the packages wrote the bindings to Python from R. These packages laid the foundation for an important package to help further the Interoperability between R and Python... reticulate. In March of 2017, the reticulate 0.7 package was officially released. It provided a comprehensive set of tools for interoperability between Python and R. After the heavy lifting had been completed in 2017 to write the bindings to Python for Tensorflow and Keras, it was time to further support working with Python from R for other use cases besides deep learning workflows. reticulate helps to streamline a bilingual workflow by allowing a user to utilize Python directly in R.

It is worth mentioning PyTorch, an open source machine learning Python library based on the Torch library and developed by Facebook's AI research group. It has grown in popularity in the last few years. It is used for applications such as computer vision and natural language processing and worth a look as an alter. Many use it with or as an alternative to Tensorflow. People do not interface with it using Keras like Tensorflow, Theano, and CNTK.

### **Applications of Deep Learning in Healthcare and Pharma**

After you have researched the deep learning space for sometime, similar stories arise as use cases such as image classification, speech recognition and language translation. Below is a short sample of areas in pharma and healthcare where deep learning is of interest:

- Disease Prediction Models
- Cancer Immunotherapy
- PK/PD
- Breast Cancer Screening - <https://www.nature.com/articles/s41586-019-1799-6>
- Classifying cCellular Images into Phenotypes
- Estimating Individualized Optimal Combination Therapy
- Drug Repurposing Efforts
- R&D & Drug Discovery
- Medical Devices:  
<https://www.fda.gov/medical-devices/software-medical-device-samd/artificial-intelligence-and-machine-learning-software-medical-device>
- Clinical Operations
- Novel Drug Targets
- Population Health
- Disease Diagnosis - Stanford's - Dermatologist-level classification of skin cancer with deep learning
- Diagnostic Imaging
- FDA Cleared AI Algorithms: <https://www.acrdsi.org/DSI-Services/FDA-Cleared-AI-Algorithms>
- Medical Claims & Electronic Medical Records

- Deep Learning at Novartis  
[novartis.com/stories/discovery/machine-learning-poised-accelerate-drug-discovery](https://novartis.com/stories/discovery/machine-learning-poised-accelerate-drug-discovery)
- DSP-1181, Drug Designed by AI Entering Clinical Human Trials for Obsessive-Compulsive Disorder by Exscientia
- Diputi.com, Using Deep Learning with a Smartphone to Diagnose Urinary Tract Infections in UK
- Predicting Risk of Suicide Attempts Over Time Through Machine Learning  
<https://journals.sagepub.com/doi/abs/10.1177/2167702617691560?journalCode=cpxa>

## Cancer Immunotherapy

Much of the following work is based on Leon Eyrich Jessen's article "Deep Learning for Cancer Immunotherapy".

Cancer is horrible. It used to be that the average time between diagnosis and death was not long for many cancers. Standard treatments were chemotherapy and radiation. James P. Allison, was awarded the Nobel Prize for his discoveries about the immune system and how it can be used to attack cancers through immune checkpoint therapy, antibodies to help the immune system's fight cancer cells. Jimmy Carter used immunotherapy to fight his cancer. Now with drugs like Merck & Co.'s Keytruda and Bayer's Larotrectinib, there are new cancer options available in addition to the traditional treatments that target tumors based on their genetic characteristics. Work on immunotherapy has become a key focus area for many biotech organizations in the last decade. There are various types of cancer immunotherapies: Immune Checkpoint Modulators, Immune System Modulators, Therapeutic antibodies, Immune/Adoptive Cell Therapy, Cancer Vaccines, oncolytic virus therapies and Personalized Tumor-Specific Neoantigen-Targeted Vaccines or T-cell-based Therapy. These different cancer immunotherapies work on the immune system in unique ways. T-cell therapy can be tricky as tumours use checkpoint blockades (signatures on the cell's membrane that tells the immune system the cell is normal) to hide from the cancer killing T-cells. Another challenge is that approaches must be developed to identify which patients are likely to respond well.

On the surface of our cells is the MHC1, a signal of the health of cells. Peptides are displayed that signal the health of the cell. T-cells check these signals to see if it is cancer. Adoptive T-cell therapy isolates tumor infiltrating T-cells from a cancer patient, genetically engineer them and then reintroduce them to fight cancer. The T-cells are activated by being exposed to tumor peptides bound to MHC1 (pMHC1). Based on many factors, one can predict which pMHC1 have the highest chance of being present in the cancer tumor. This will help in knowing which pMHC1s to use in activating the T-cells.

Leon Eyrich Jessen tested 3 models, a deep feed forward fully connected ANN, a convolutional ANN (connected to a FFN), and a random forest (for comparison) to classify a peptide as a 'strong binder' SB, 'weak binder' WB or 'non-binder' NB. to MHC1. You can see a walk through of this analysis here:

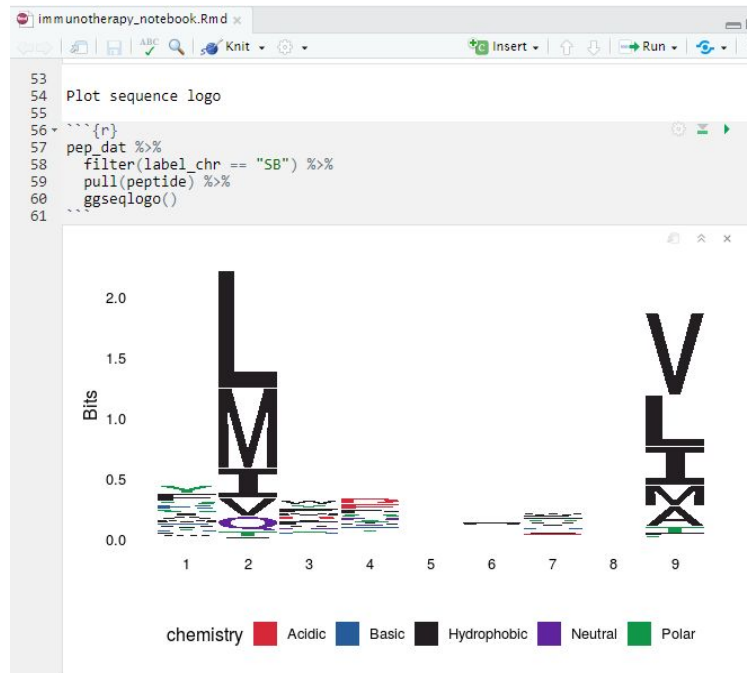
<https://blogs.rstudio.com/Tensorflow/posts/2018-01-29-dl-for-cancer-immunotherapy/>

You can run the code from the analysis here:

<https://rstudio.cloud/project/1011147>

immunotherapy\_notebook.Rmd introduces the topic, analyses and plots some peptide genetic sequences. You can see the live R Notebook here:

<https://colorado.rstudio.com/rsc/connect/#/apps/2339/access/2484>



## Tensorflow model

The deep learning model returns the class of a peptide. The input data is a flattened pep encoding of 180 values. After we have trained the model, we will deploy it so it can be consumed by other processes.

Below is a summary of the model we defined:

```

> dim(x_train)
[1] 21384    9    20
> x_train <- array_reshape(x_train, c(nrow(x_train), 9 * 20))
> x_test <- array_reshape(x_test, c(nrow(x_test), 9 * 20))
> y_train <- to_categorical(y_train, num_classes = 3)
> y_test <- to_categorical(y_test, num_classes = 3)
> model <- keras_model_sequential() %>%
+   layer_dense(units = 180, activation = "relu", input_shape = 180) %>%
+   layer_dropout(rate = 0.4) %>%
+   layer_dense(units = 90, activation = "relu") %>%
+   layer_dropout(rate = 0.3) %>%
+   layer_dense(units = 3, activation = "softmax")
> summary(model)

```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 180)	32580
dropout (Dropout)	(None, 180)	0
dense_1 (Dense)	(None, 90)	16290
dropout_1 (Dropout)	(None, 90)	0
dense_2 (Dense)	(None, 3)	273
Total params: 49,143		
Trainable params: 49,143		
Non-trainable params: 0		

**Files** | **Plots** | **Packages** | **Help** | **View**

New Folder
Upload
Delete

Cloud
project
BiolT2019-R-Tense

▲ Name

..

1\_train\_model.R

2\_publish\_model.R

4\_publish\_api.R

5\_consume\_api.R

immunotherapy.Rproj

immunotherapy\_notebook.Rmd

plumber

README.md

README.Rmd

saved\_models

Here are the live results while training the model:

```

# Define the model

model <- keras_model_sequential() %>%
  layer_dense(units = 180, activation = "relu", input_shape = 180) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 90, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 3, activation = "softmax")

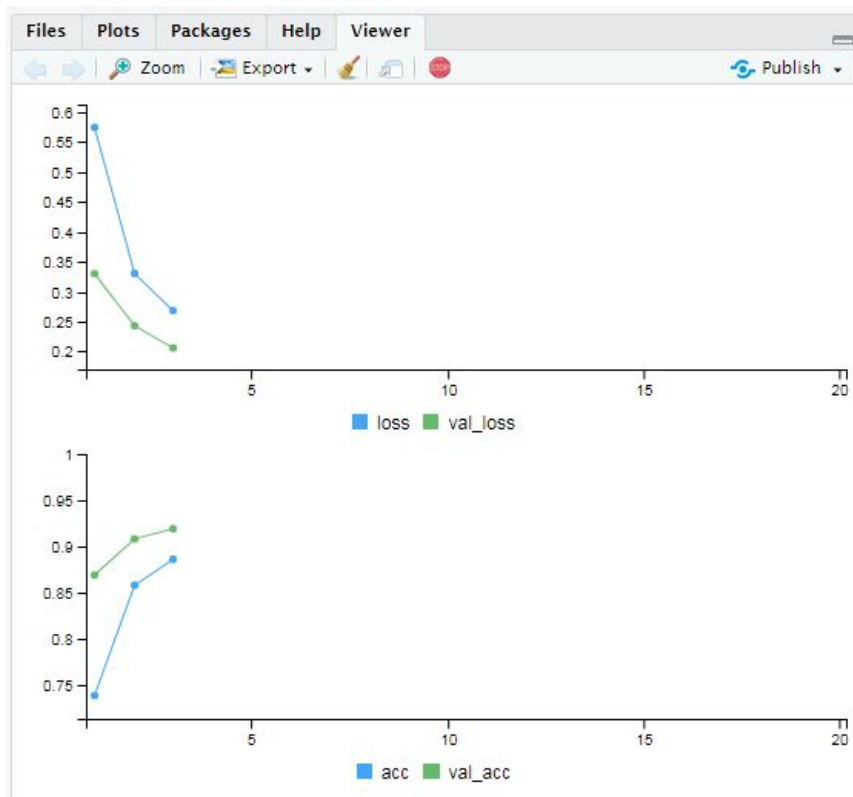
summary(model)

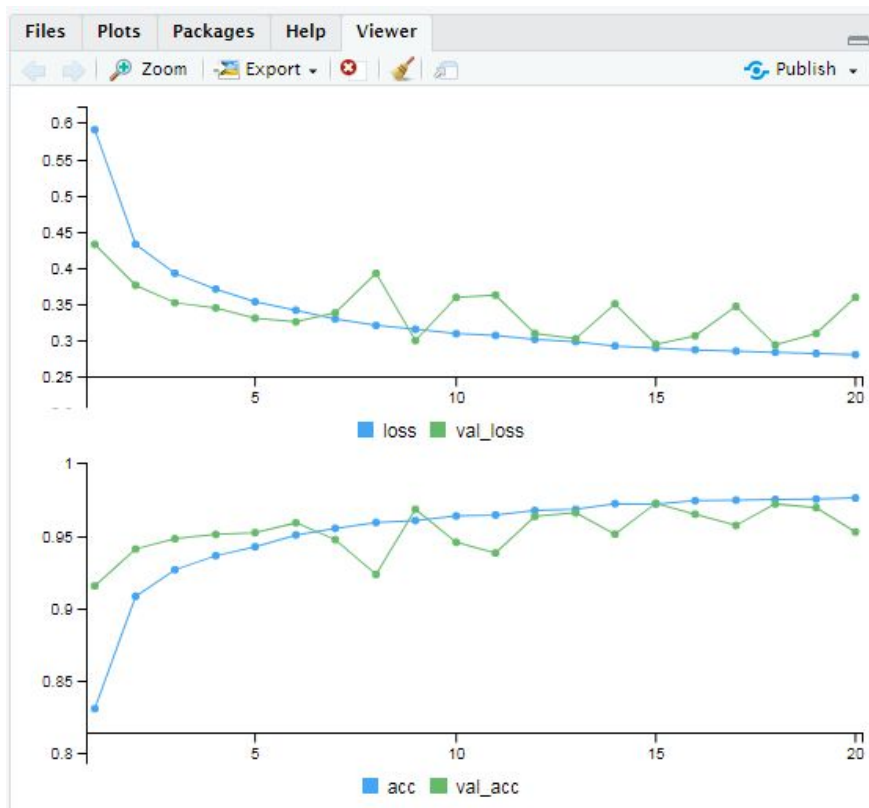
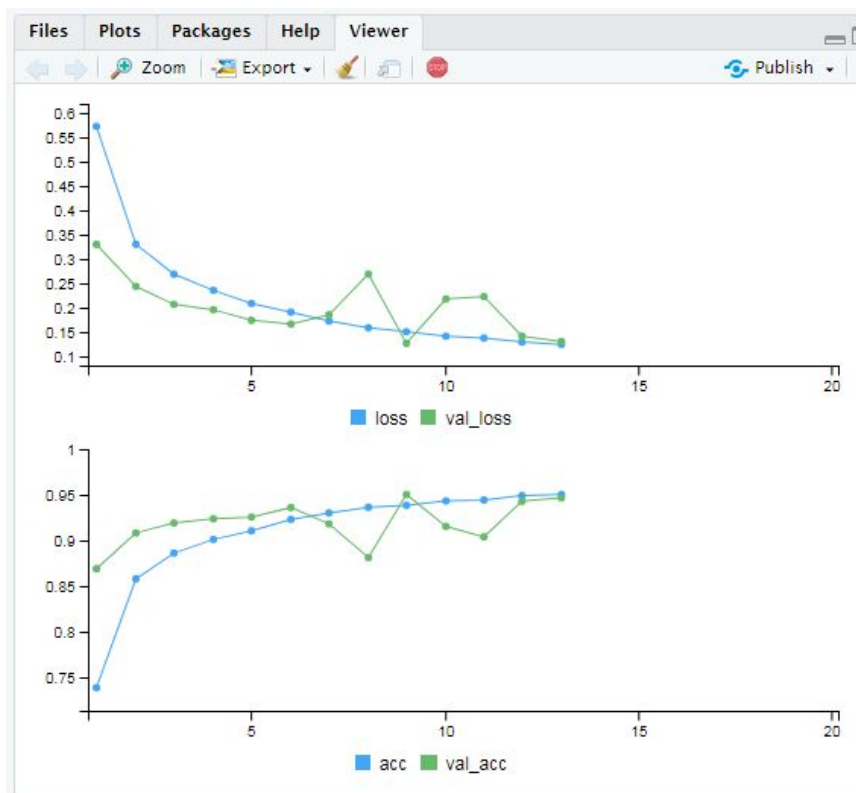
model %>%
  compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(),
    metrics = c("accuracy")
  )

# Train the model

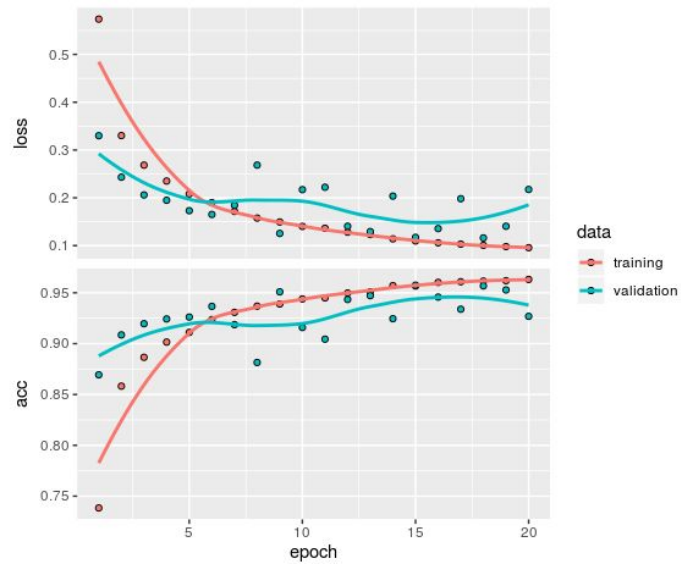
history <- model %>%
  fit(
    x_train, y_train,
    epochs = 20,
    batch_size = 64,
    validation_split = 0.2
  )

```









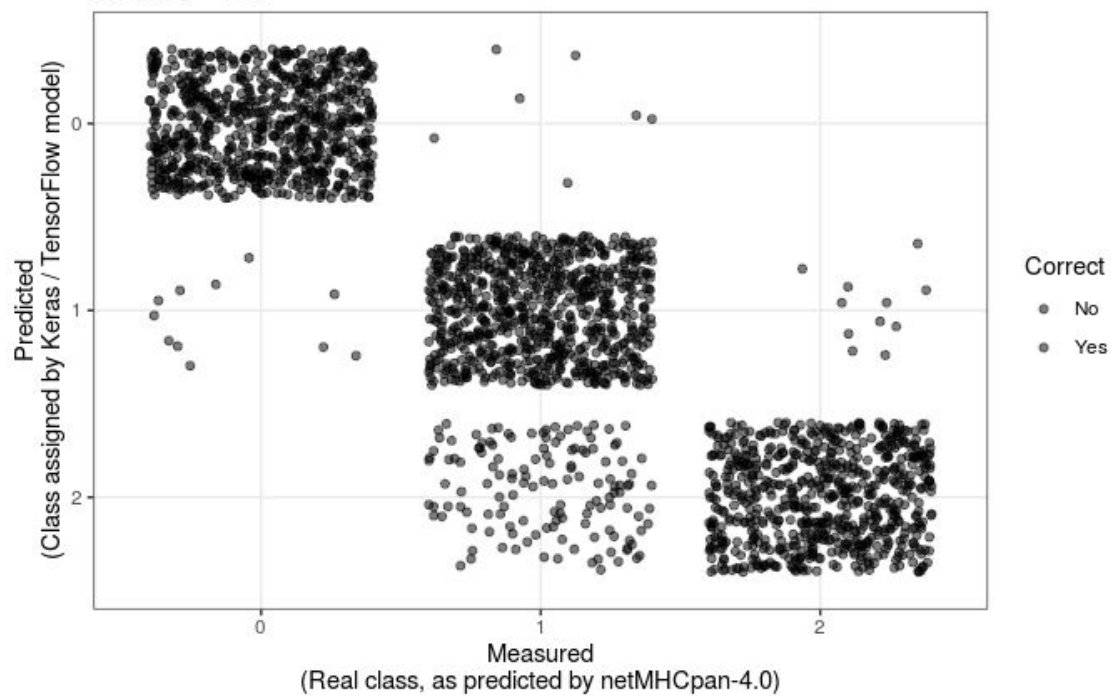
Now, we will review the model performance:

```
> perf
$loss
[1] 0.1847098

$acc
[1] 0.9297138
```

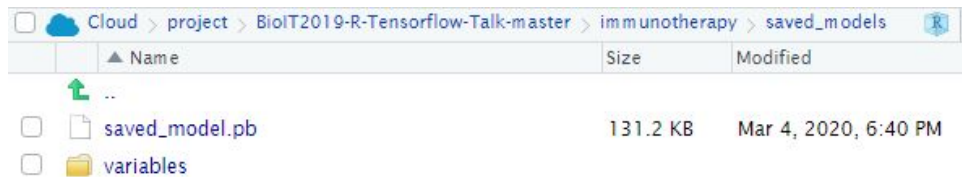
### Performance on 10% unseen data - Feed Forward Neural Network

Accuracy = 93%



After that we have our model, we will save it:

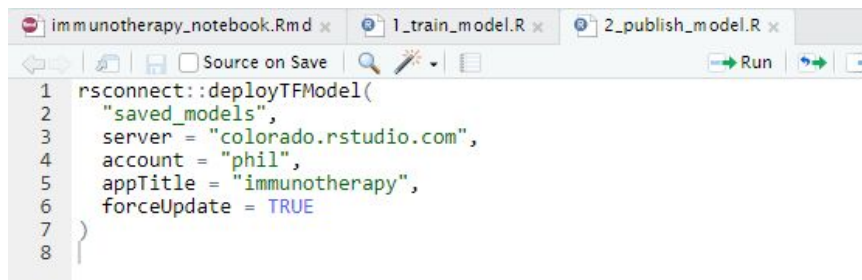
```
# save model for deployment -----  
model %>% tensorflow::export_savedmodel(export_dir_base = "saved_models")
```



Name	Size	Modified
..		
saved_model.pb	131.2 KB	Mar 4, 2020, 6:40 PM
variables		

This is where the above analysis ends via the article by Leon Eyrich Jessen. It is common though in the deep learning space to deploy your model. Below we will review how that is done in R:

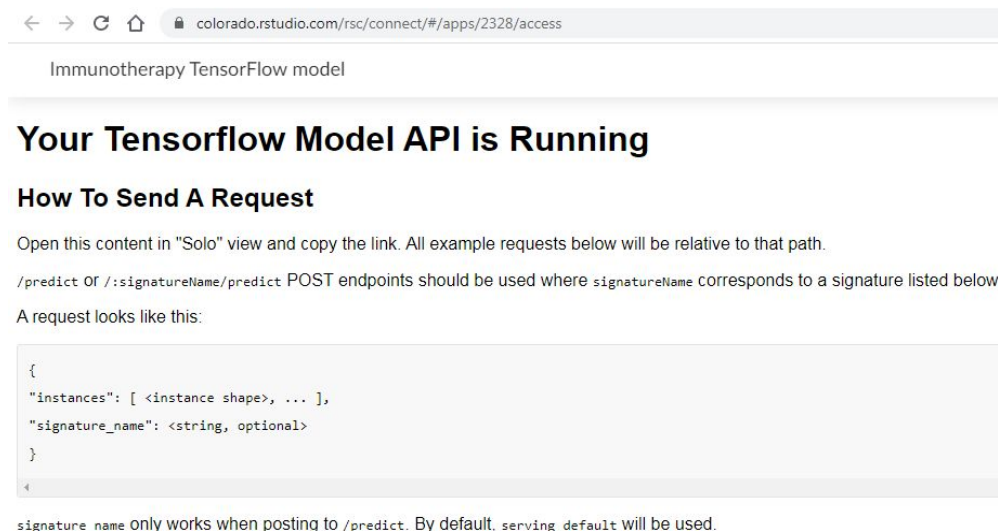
And finally will deploy it to our production model system:



```
1 rsconnect::deployTFModel(  
2   "saved_models",  
3   server = "colorado.rstudio.com",  
4   account = "phil",  
5   appTitle = "immunotherapy",  
6   forceUpdate = TRUE  
7 )  
8
```

You can see the live Tensorflow model here:

<https://colorado.rstudio.com/rsc/connect/#/apps/2328/access>



Immunotherapy TensorFlow model

## Your Tensorflow Model API is Running

### How To Send A Request

Open this content in "Solo" view and copy the link. All example requests below will be relative to that path.

/predict OF /:signatureName/predict POST endpoints should be used where signatureName corresponds to a signature listed below.

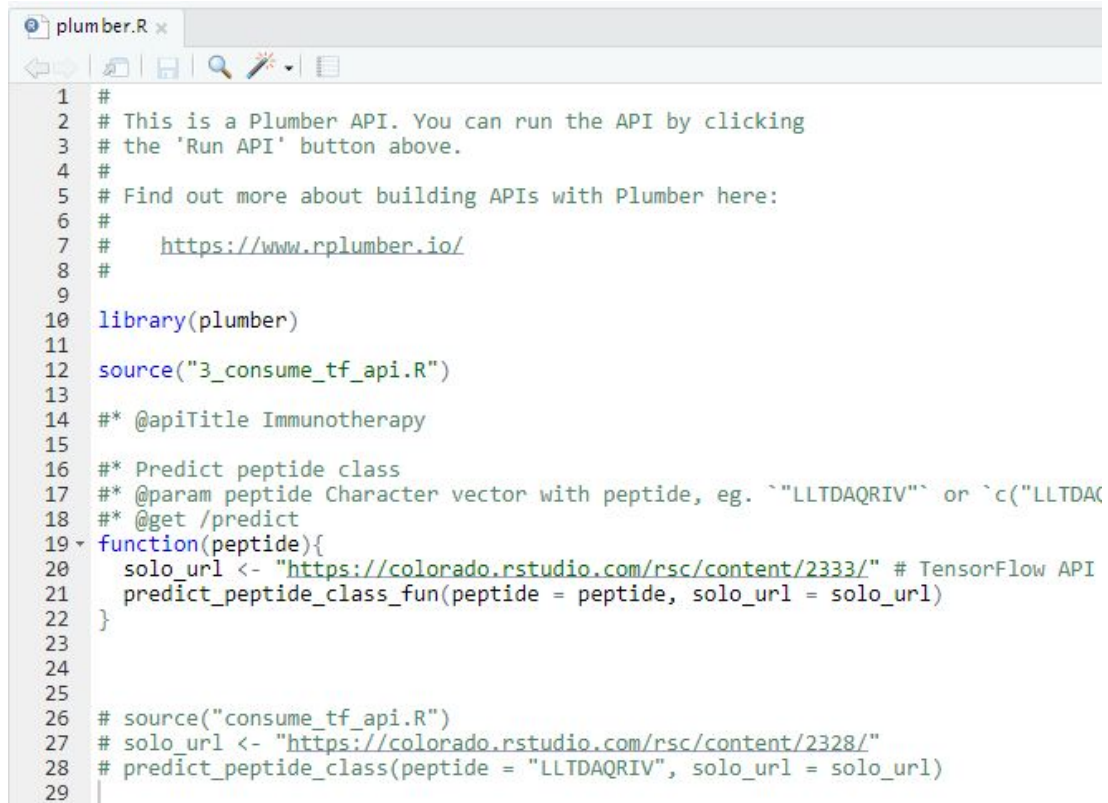
A request looks like this:

```
{  
  "instances": [ <instance shape>, ... ],  
  "signature_name": <string, optional>  
}
```

signature\_name only works when posting to /predict. By default, serving\_default will be used.

## Plumber API

Plumber is an R package that converts your existing R code to a web API. It is a popular R package for deploying machine learning models as RESTful APIs. Next, we will take our Tensorflow model and convert it to a plumber API:



```
1 #
2 # This is a Plumber API. You can run the API by clicking
3 # the 'Run API' button above.
4 #
5 # Find out more about building APIs with Plumber here:
6 #
7 # https://www.rplumber.io/
8 #
9
10 library(plumber)
11
12 source("3_consume_tf_api.R")
13
14 #* @apiTitle Immunotherapy
15
16 #* Predict peptide class
17 #* @param peptide Character vector with peptide, eg. `LLTDAQRIV` or `c("LLTDAQ
18 #* @get /predict
19 function(peptide){
20   solo_url <- "https://colorado.rstudio.com/rsc/content/2333/" # TensorFlow API
21   predict_peptide_class_fun(peptide = peptide, solo_url = solo_url)
22 }
23
24
25
26 # source("consume_tf_api.R")
27 # solo_url <- "https://colorado.rstudio.com/rsc/content/2328/"
28 # predict_peptide_class(peptide = "LLTDAQRIV", solo_url = solo_url)
29
```

The code in 3\_consume\_tf\_api.R will define a predict\_peptide\_class\_fun function that will be the heart of the plumber API above. This function will allow another person or system to classify a set of peptides as 'strong binder' SB, 'weak binder' WB or 'non-binder' NB to MHC1. One could imagine another pharma or hospital could use our model to test out new peptides.

To use the Tensorflow model, it is convenient to have another function that does the pre- and post-processing. Our plumber API returns a peptide class given a peptide string as input. Just as before, we will publish the plumber API to our production modeling/API environment:

```
therapy_notebook.Rmd x 1_train_model.R x 2_publish_model.R x 4_publish_api.R x
Source on Save Run So
1 library(rsconnect)
2
3
4 withr::with_dir("plumber", list.files())
5
6 withr::with_dir(
7   "plumber",
8   rsconnect::deployAPI(
9     ".",
10    server = "colorado.rstudio.com",
11    # account = "{account}", # <<- edit this line if necessary
12    appTitle = "immunotherapy_api",
13    forceUpdate = TRUE
14  )
15 )
16
17
```

You can see the live plumber API here:

<https://colorado.rstudio.com/rsc/connect/#/apps/2337/access>

By clicking try it out, you can pass in a peptide:

Name	Description
peptide	Character vector with peptide, eg. "LLTDAQRIV" or c("L
string	
(query)	<input type="text" value="LLTDAQRIV"/>
<button>Execute</button>	

Here we can see that the peptide tested has weak binding:

200

Undocumented

#### Response body

```
[
  {
    "peptide": "LLTDAQRIV",
    "NB": 0.0018,
    "WB": 0.8791,
    "SB": 0.1191
  }
]
```

#### Response headers

```
content-length: 61
content-type: application/json
date: Thu, 05 Mar 2020 00:34:35 GMT
server: nginx/1.14.0 (Ubuntu)
status: 200
x-content-type-options: nosniff
```

Alternatively, you can test out our Tensorflow model programmatically with 5\_consume\_api.R:



```
1 library(httr)
2
3 predict_peptide <- function(peptide){
4   solo_url = "https://colorado.rstudio.com/rsc/content/2341/"
5   if (substring(solo_url, nchar(solo_url)) != "/") solo_url <- paste0(solo_url, "/")
6   api_url <- paste0(solo_url, "/predict")
7
8   r <- GET(api_url, query = list(peptide = peptide), encode = "json", content_type_json())
9   if (httr::http_error(r)){
10     cat(http_status(r))
11     stop(http_status(r))
12   }
13   r %>%
14     content() %>%
15     as.data.frame()
16 }
17
18
19 predict_peptide(peptide = "LLTDAQRIV")
20 predict_peptide(c("LLTDAQRIV", "LMAFYLYEV", "VMSPITLPT", "SLHLTNCFV", "RQFTCMIAV"))
21
```

Results (same as above via the live API):

```

+   labels = c("NB", "WB", "SB"),
+   values = c("red", "blue")
+ ) +
+   theme_bw()
> library(httr)
> predict_peptide <- function(peptide){
+   solo_url = "https://colorado.rstudio.com/rsc/content/2337/"
+   if (substring(solo_url, nchar(solo_url)) != "/") solo_url <- paste0(solo_url, "/")
+   api_url <- paste0(solo_url, "/predict")
+
+   r <- GET(api_url, query = list(peptide = peptide), encode = "json", content_type_json())
+   if (httr::http_error(r)){
+     cat(http_status(r))
+     stop(http_status(r))
+   }
+   r %>%
+     content() %>%
+     as.data.frame()
+ }
> predict_peptide(peptide = "LLTDAQRIV")
  peptide    NB    WB    SB
1 LLTDAQRIV 0.0018 0.8791 0.1191
>

```

## Conclusion

This paper was to highlight how R and Python can come together via interoperability to help R statistical programmers see a use case for deep learning. As interest in deep learning increases by clinical statistical programmers, it is important to understand how this tooling can be used. Leon Eyrich Jessen's work provides R users insight into an exciting area of cancer research that is a hot topic for many pharmaceutical companies. The information outlined in this paper highlights the history of deep learning and how R and Python interoperability can help bring this tool to the clinical space.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Phil Bowsher

[phil@rstudio.com](mailto:phil@rstudio.com)