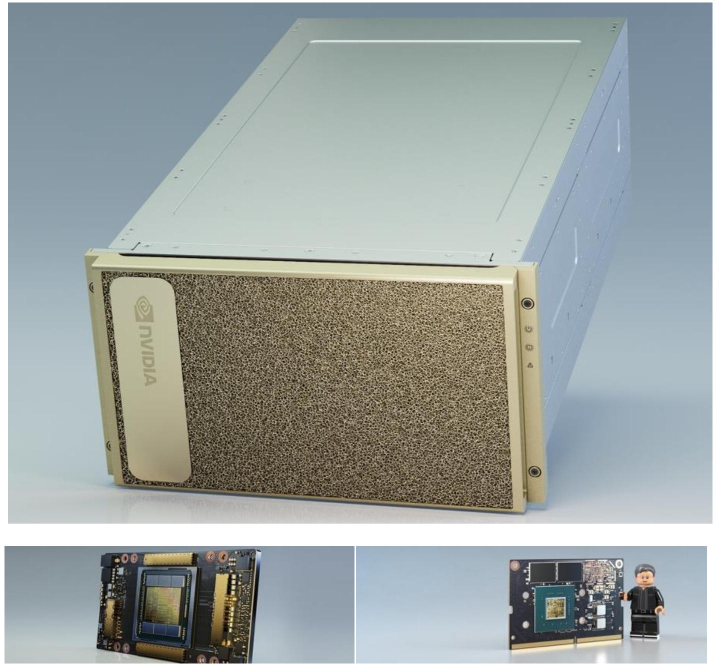
FIRST DEEP NEURAL NETWORK(DNN) USING KERAS

Kristopher Keipert

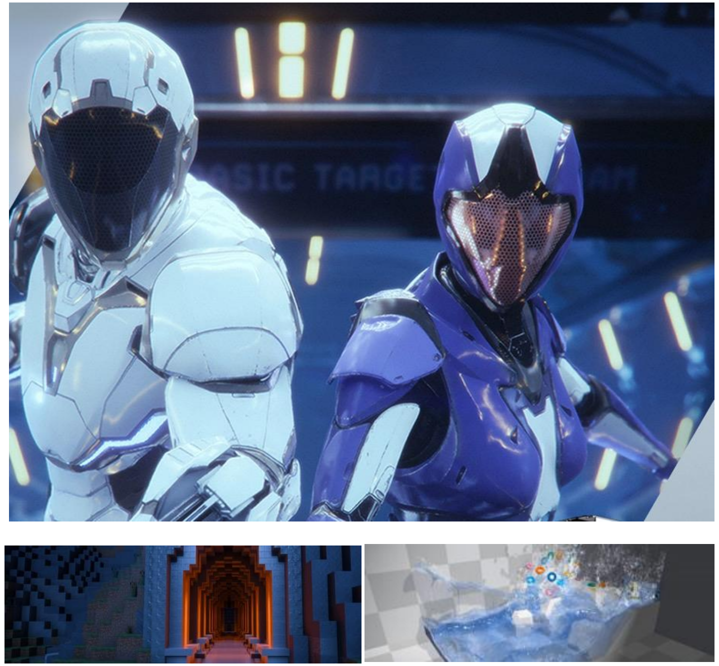
Kristopher Keipert
Solutions Architect
kkeipert@nvidia.com

Marc West
Account Manager
marcw@nvidia.com

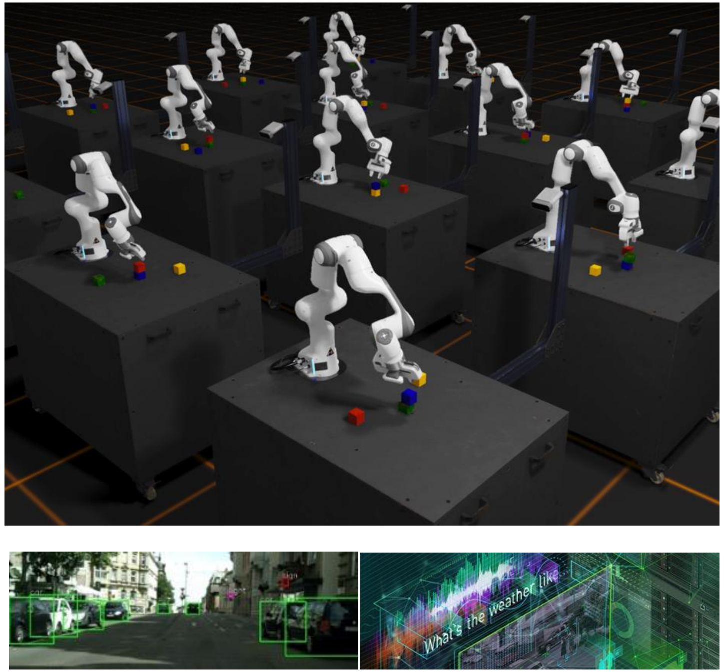
NVIDIA DELIVERS END-TO-END ACCELERATION



GPU Computing



Computer Graphics



Artificial Intelligence

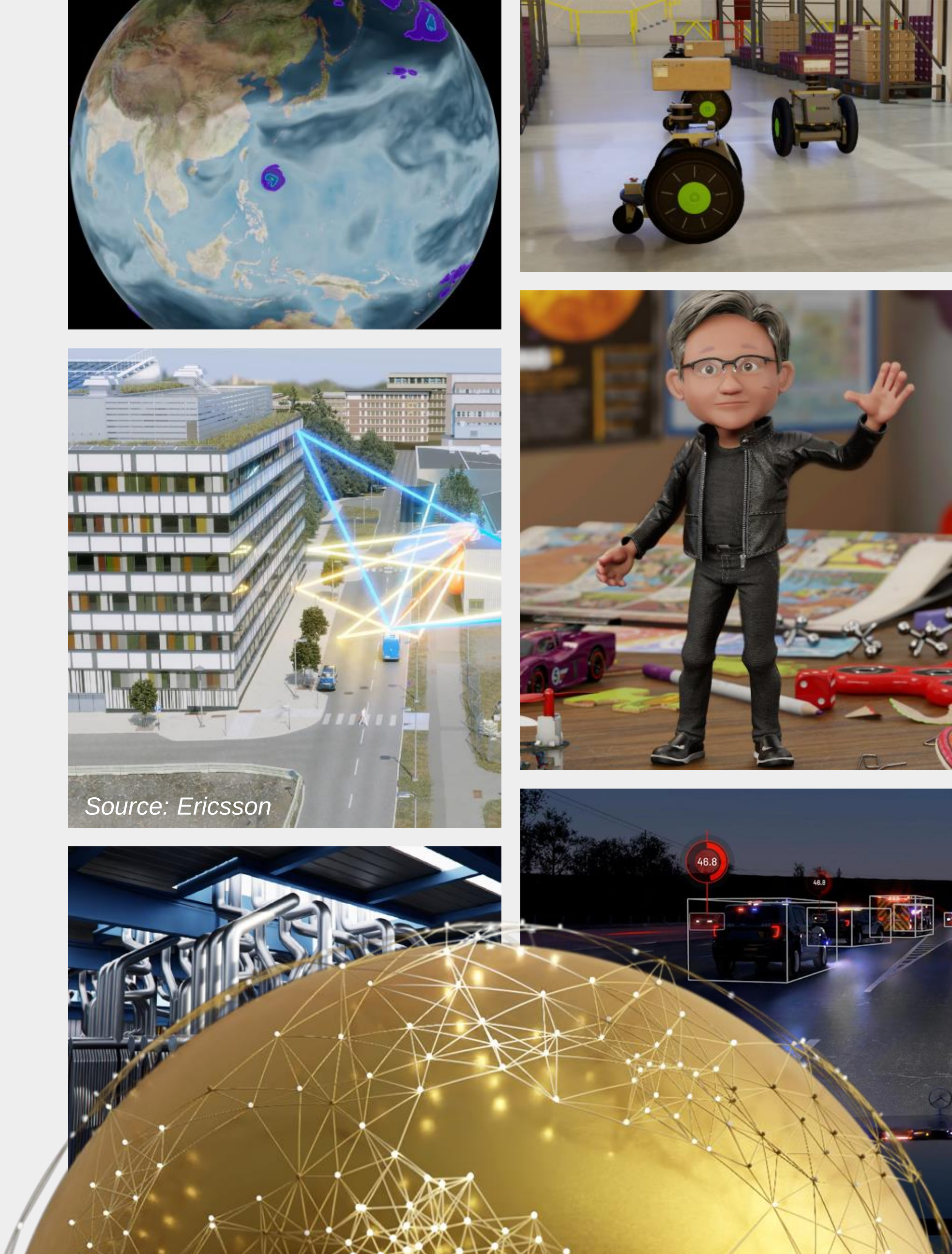
NVIDIA GTC

YOUR MOST BRILLIANT WORK STARTS HERE

The Developer Conference for the Era of AI

NVIDIA GTC is more than a game-changing AI developer conference. It's a global community committed to decoding the world's greatest challenges, transforming every major industry workflow, and exploring tomorrow's next big ideas—together. Join us this March and discover how to accelerate your life's work.

MARCH 21—24, 2022 | www.nvidia.com/GTC



WHAT TO EXPECT AT GTC 2022

500+ SESSIONS

•Live sessions, on-demand presentations, interactive panels, beginners content, and more.

AMAZING SPEAKERS

Jensen Huang (GTC Keynote) and thought leaders from every industry.

CONNECT WITH THE EXPERTS

Opportunities to connect with subject-matter experts from NVIDIA to get your pressing questions answered.

TRAINING

Workshops from the NVIDIA Deep Learning Institute (DLI) and NVIDIA Academy with courses for AI, accelerated computing, data science, and more.

STARTUPS

NVIDIA's startup ecosystem and industry execs sharing what it takes to succeed as a startup working in AI, data science, or HPC.

DEMOS

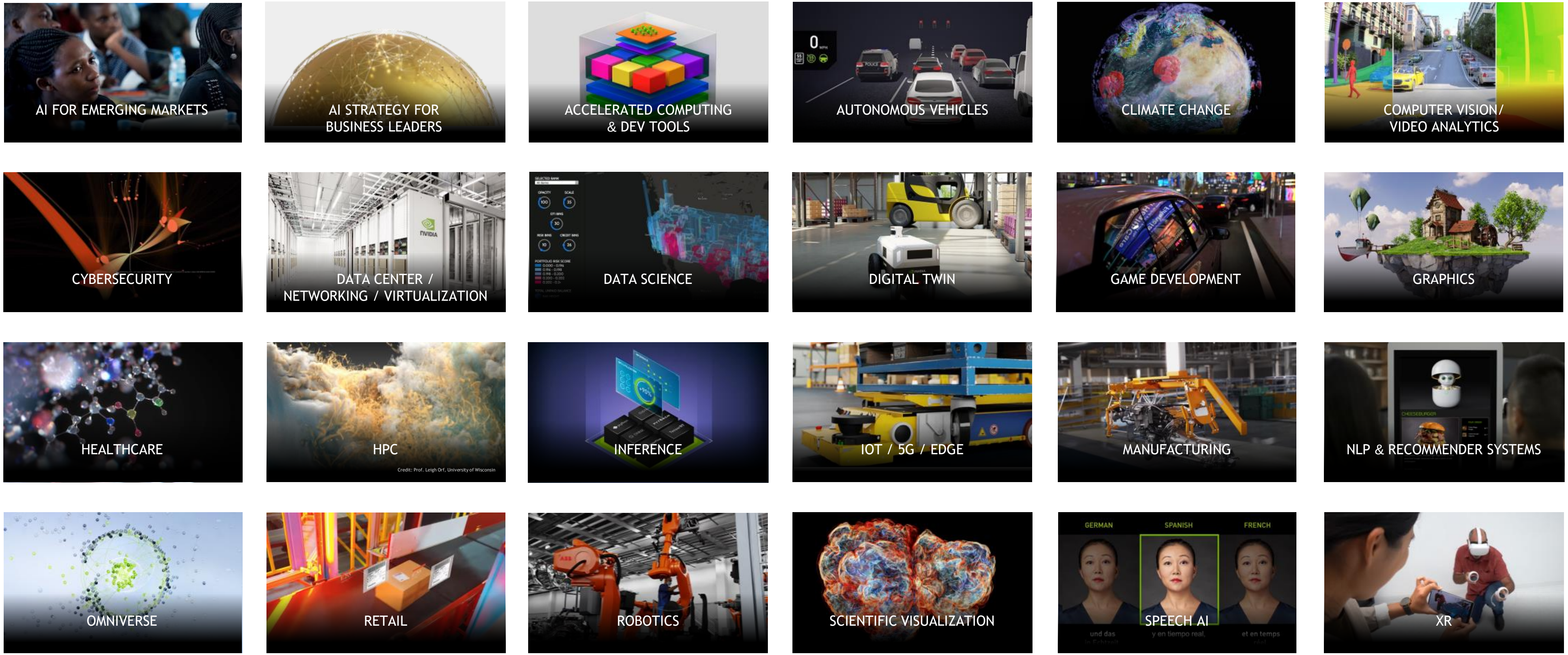
Solutions for workloads by subject-matter experts on the latest NVIDIA innovations and partner applications.

Conference and Training: March 21-24, 2022

Keynote: March 22, 2022

GTC sessions will run live in local time zones across NALA, EMEA, and APAC

GTC 2022 KEY TOPICS



NVIDIA GTC

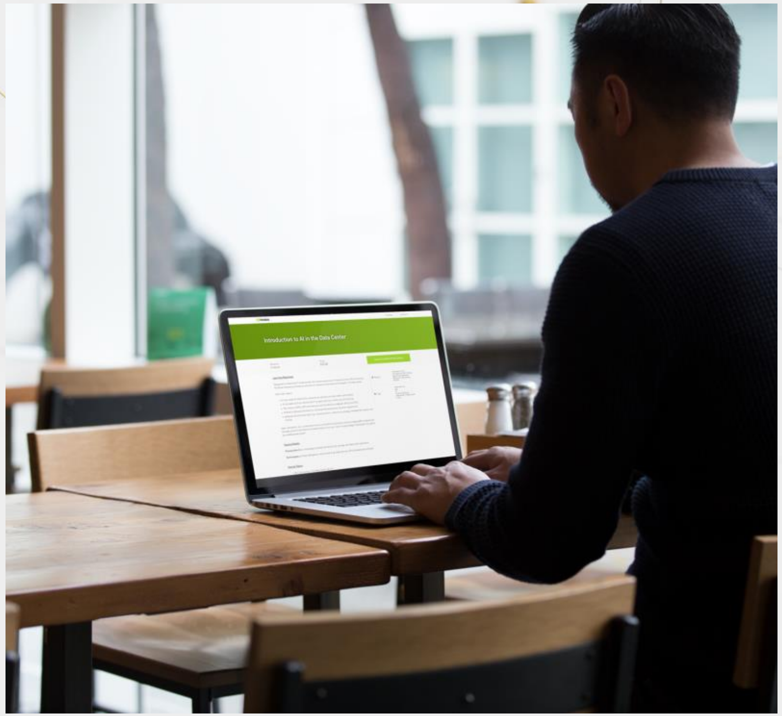
GET HANDS-ON INSTRUCTOR-LED TRAINING ONLINE AT GTC 2022

The NVIDIA Deep Learning Institute (DLI) offers in-depth workshops taught by subject matter experts to help you solve your most challenging problems in AI, accelerated computing, data science, graphics and simulation, and more.

Register for GTC and get free access to two-hour workshops. Or add a full-day workshop to your conference pass for a nominal fee. Visit the GTC training page for more details.

Spots are limited and registration is on a first come first serve basis. DLI workshops at NVIDIA GTC start Monday, March 21, 2022.

MARCH 21—24, 2022 | www.nvidia.com/GTC



NVIDIA | DEEP LEARNING INSTITUTE

GPU HACKATHONS

Events Partners Attendees Mentors About Log In

Search...

Expand Your Skills

Our events help you to hone your skills by connecting you with the latest technologies, robust tool sets, and collective expertise.

gpuhackathons.org

ALL HACKATHONS BOOTCAMPS Location Filter Select a location

C-DAC India GPU Bootcamp

Date(s):Feb 17, 2021 - Feb 18, 2021
Event Focus: HPC
Digital Event

Applications Open

Simon Fraser University GPU Hackathon

Date(s):Feb 21, 2021 - Mar 3, 2021
Event Focus: HPC+AI
Digital Event

Applications Closed

NVIDIA/ENCCS AI for Science Bootcamp

Date(s):Mar 8, 2021 - Mar 9, 2021
Event Focus: AI
Digital Event

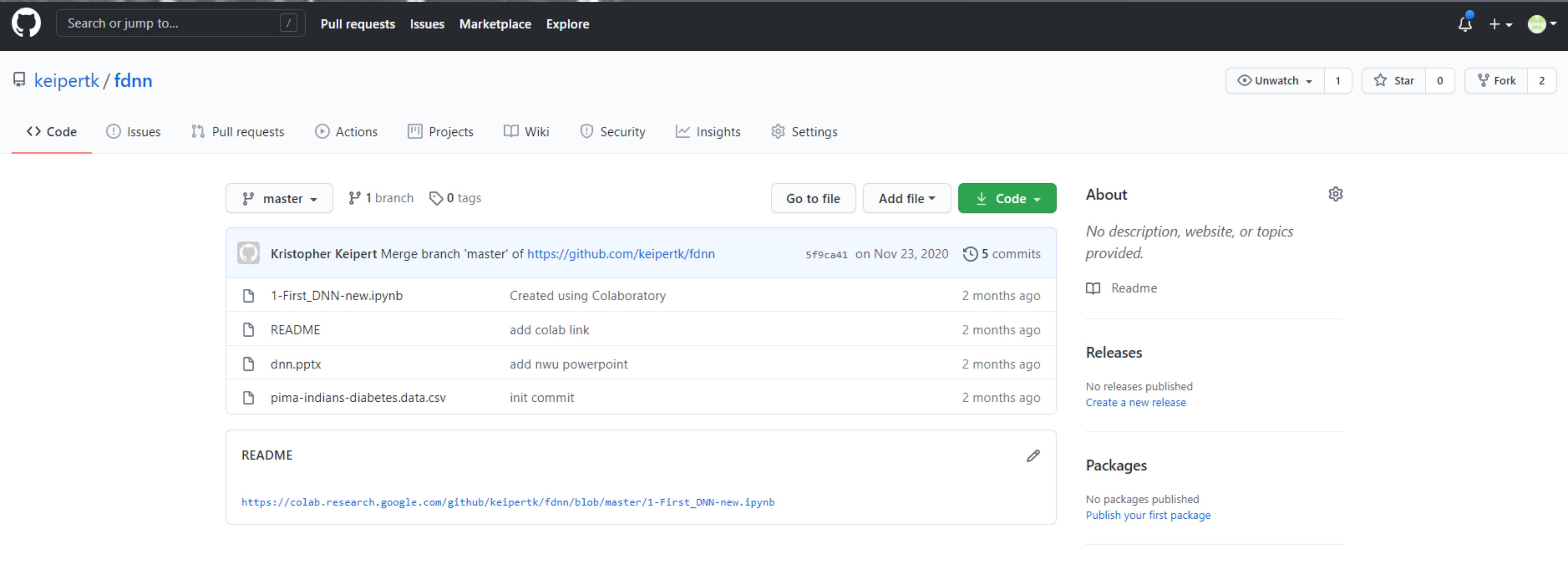
Applications Open

NVIDIA: Convolution Neural Network Models

Event Type	Conference/Workshop
Sponsor	Center for Artificial Intelligence Innovation
Virtual	
Date	Mar 9, 2022 3:00 - 5:00 pm
Speaker	Jeff Layton, NVIDIA
Views	25
Originating Calendar	Center for Artificial Intelligence Innovation

Join the Center for Artificial Intelligence Innovation at NCSA for a joint training with NCSA User-Services and NVIDIA on **Wednesday, March 9, from 3-5pm** via Zoom for an online training session NVIDIA: Convolution Neural Network Models

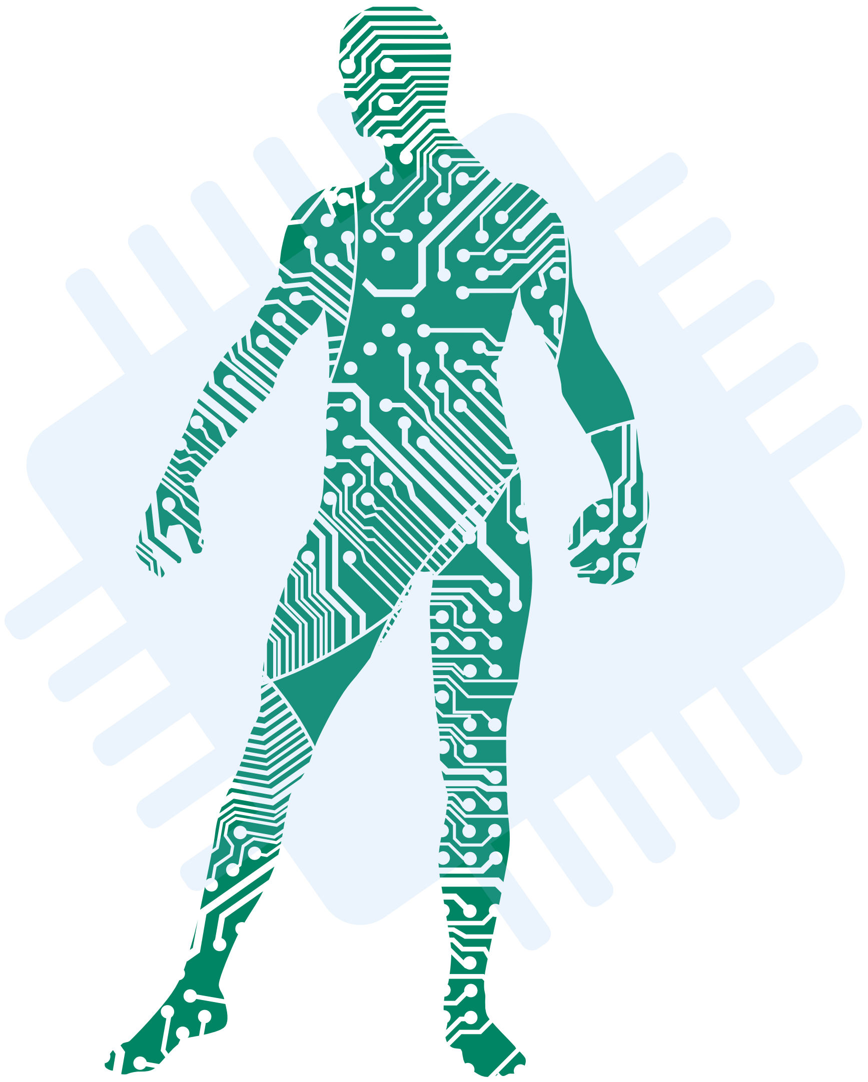
Register for the Zoom session here: https://go.ncsa.illinois.edu/CAIIHALTraining



github.com/keipertk/fdnn

Agenda

1. Deep Learning Approach
2. Seven NN Stages (Hands-on)
3. Basic Visualization +
   Model Summary

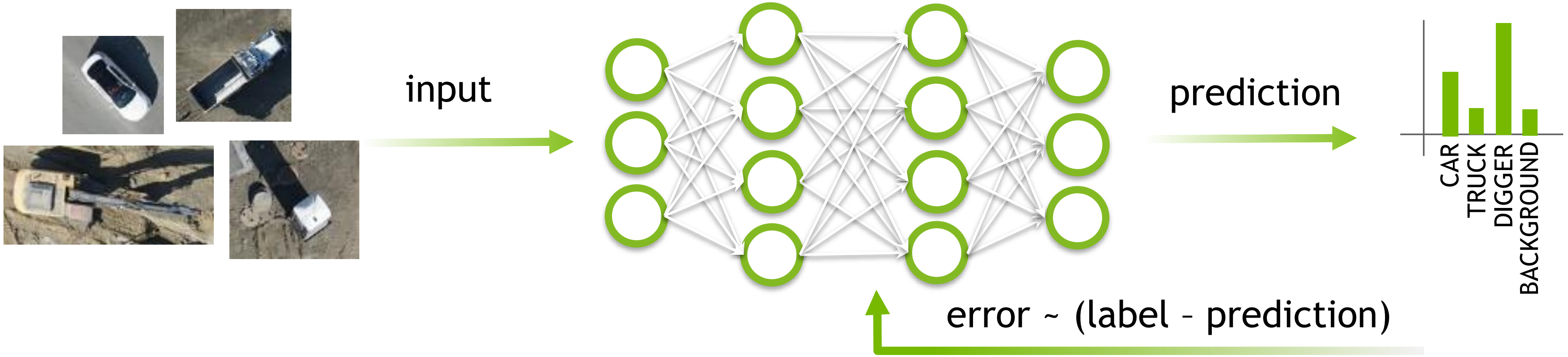


DEEP LEARNING APPROACH

LABELLED TRAINING DATA

DEEP NEURAL NETWORK "MODEL"

OBJECT CLASS PREDICTIONS



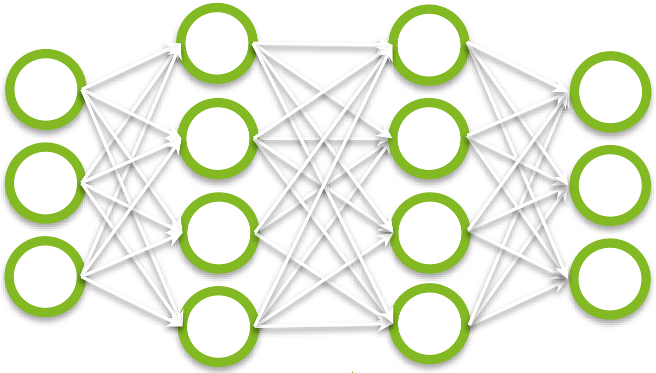
Back-propagate errors for parameter update

DEEP LEARNING APPROACH

LABELLED TRAINING DATA

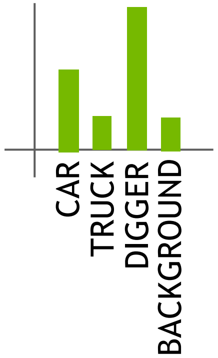
input

DEEP NEURAL NETWORK "MODEL"



prediction

OBJECT CLASS PREDICTIONS



error ~ (label – prediction)

Back-propagate errors for parameter update

- Need LOTS of data
  - Millions+ of images
- Divide Data into 2 Groups:
  - Training Data (67-80%)
  - Validation Data (20-33%)
- Have test data available

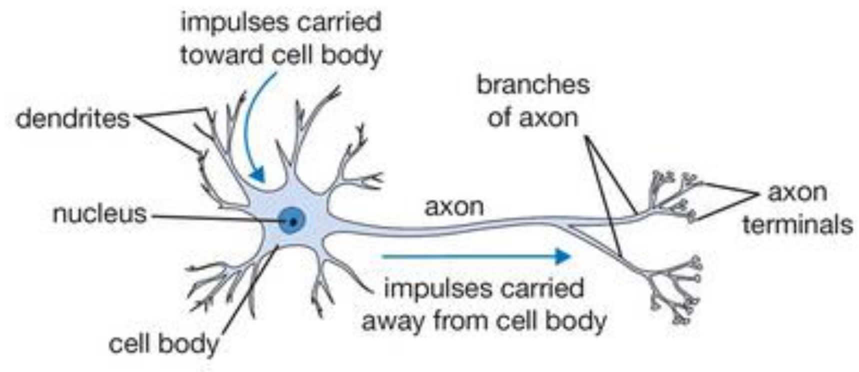
PERCEPTRON

SINGLE PERCEPTRON

- Fundamental building block for many Neural Networks or parts of networks
  - Using several perceptrons can create MLP – Multi-Layer Perceptron
  - Better known as a Neural Network
- Dates back to the 1950's
  - Great for binary classification (is it a cat or not?)
  - Also works for multi-class classification (is it a cat, dog, chipmunk, bicycle?)
- Idealized, simple model of neuron (artificial neuron)

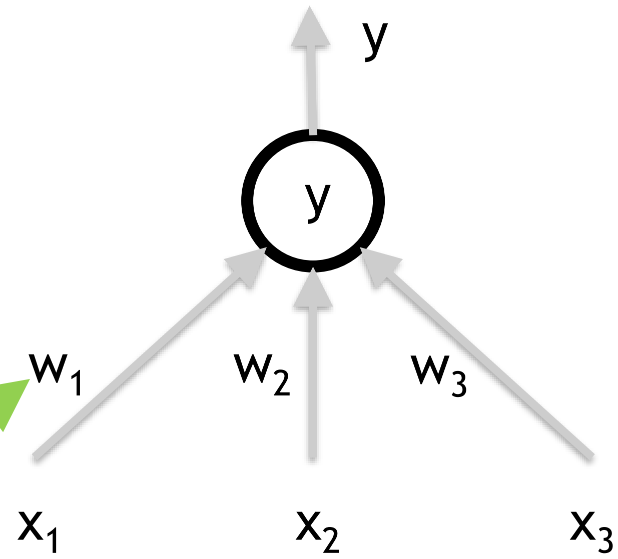
ARTIFICIAL NEURONS

Biological neuron



From Stanford cs231n lecture notes

Artificial neuron



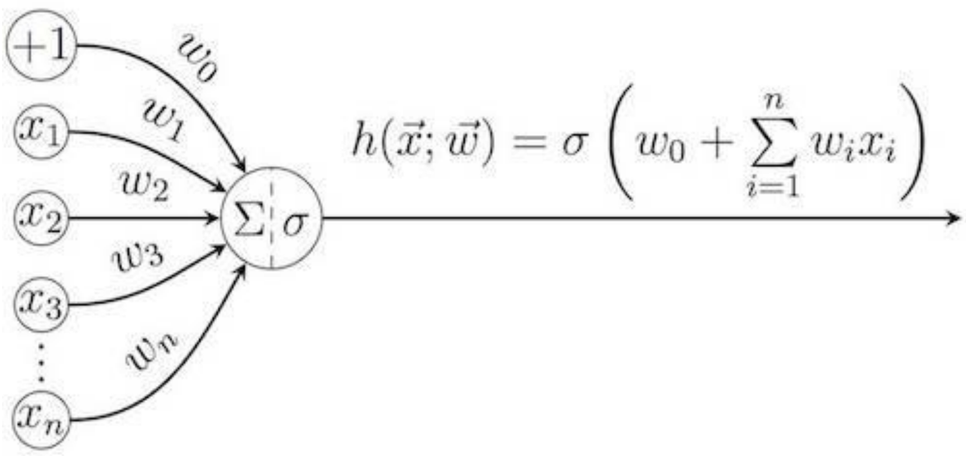
Weights (W_n)
= parameters

$$y=F(w_1x_1+w_2x_2+w_3x_3)$$

SINGLE PERCEPTRON

- Very simple model:
  - N input signals (x_1 .. x_n)
  - N weights (w_1 .. w_n)
  - 1 Bias (w_0)
- σ is activation function
  - Relu, sigmoid, heavyside
- N weights (w_1 .. w_n) and bias, w_0, are "Design variables"
  - These are "discovered" by training
  - No Human intervention or coding

Single Perceptron

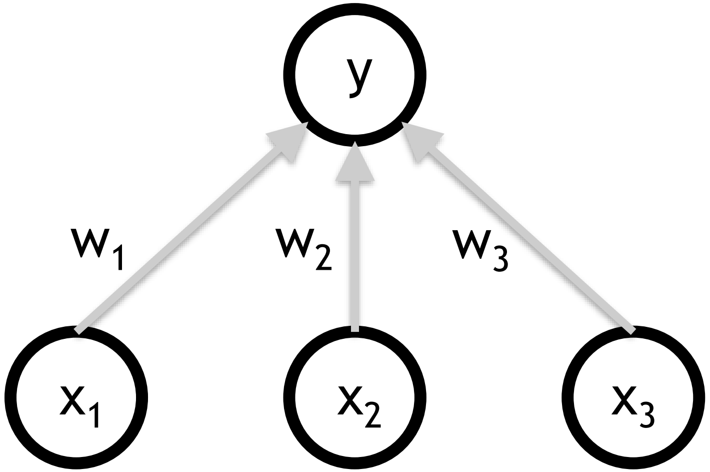


$$Y = \Sigma\,(weight * input) + bias$$

CRITICAL MODEL ITEMS

- Need to define the number of inputs
- The layer needs to be initialized (usually random numbers)
  - Weights and bias
- Specify the activation function (F)
  - ReLU
  - Sigmoid
  - Softmax
  - Tanh
  - Linear

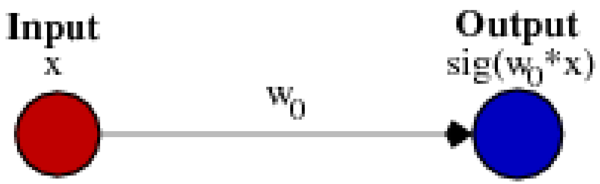
Artificial neuron



$$y=F(w_1x_1+w_2x_2+w_3x_3)$$

ROLE OF BIAS

- Biases are almost always helpful
- In effect, **a bias value allows you to shift the activation function to the left or right**, which may be critical for successful learning
- The NN “discovers” the bias value
    - Typically one per neuron
- Weights determine “steepness” of curve
- Bias shifts entire curve left or right



ACTIVATION FUNCTIONS

- These are very key to training!
  - They can greatly improve learning or just make it a flop
- In these slides, the focus is on numerical data
  - These activations functions are VERY useful for classification
  - No images – No Convolutions
    - See next lecture

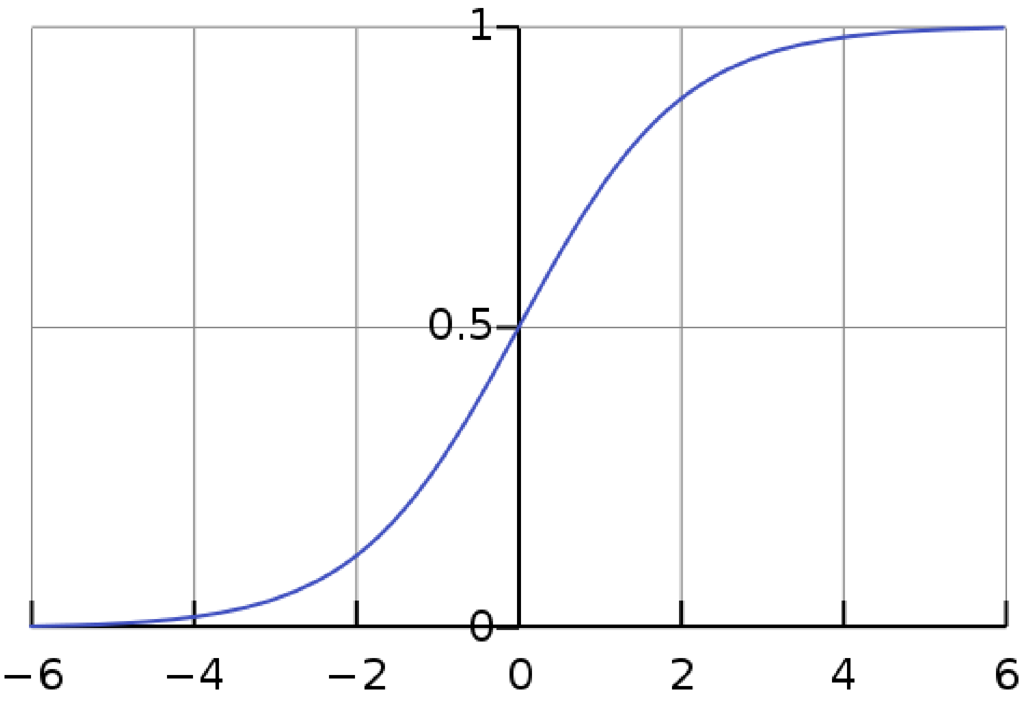
ACTIVATION FUNCTION

Sigmoid

- The sigmoid function in the output layer
  - 'sigmoid' in Keras

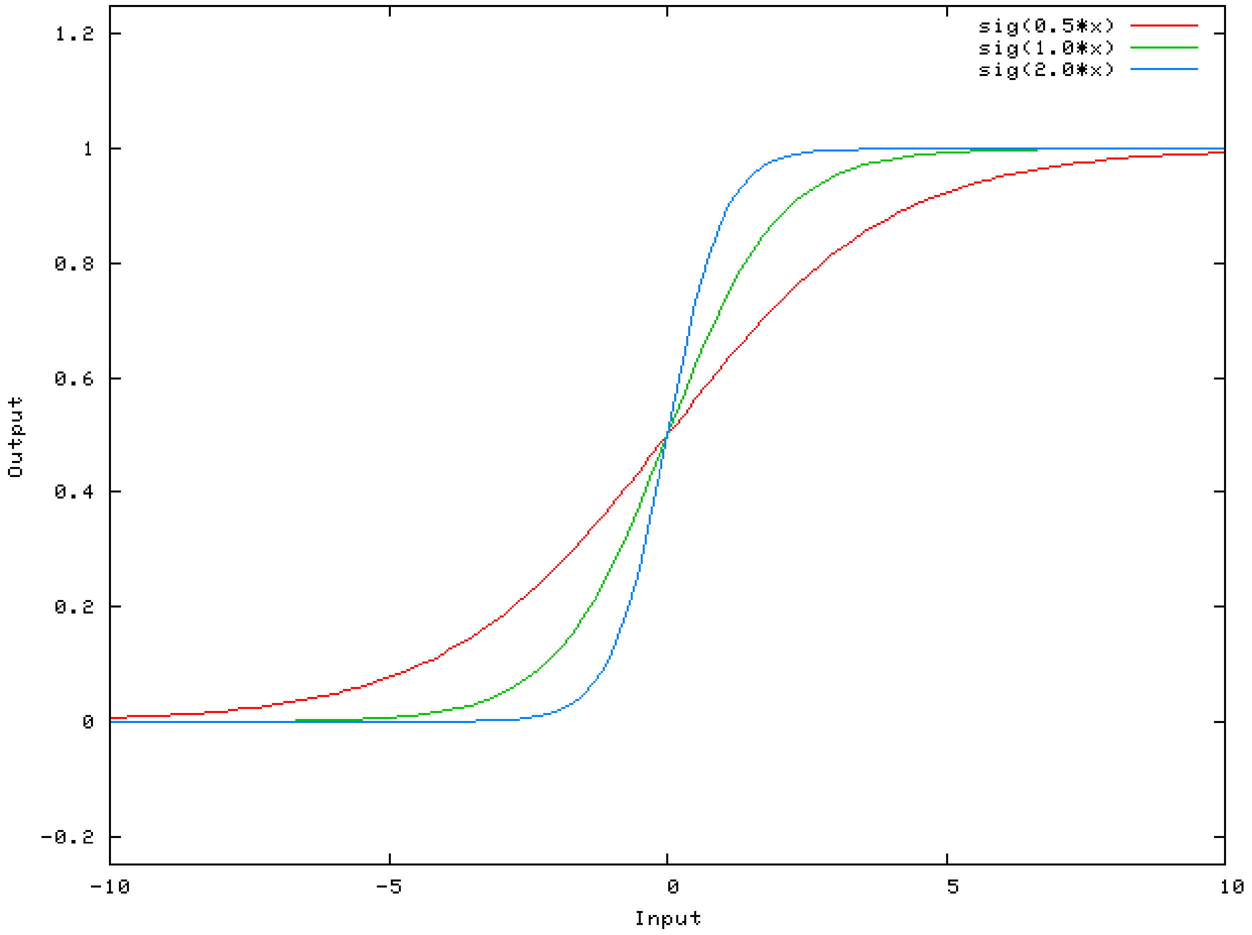
$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

- Special case of a **logistic function**
- Differentiable
- Using a sigmoid on the output layer ensures our network output is between 0 and 1
  - **Easier to map to a probability**
- `keras.activations.sigmoid(x)`



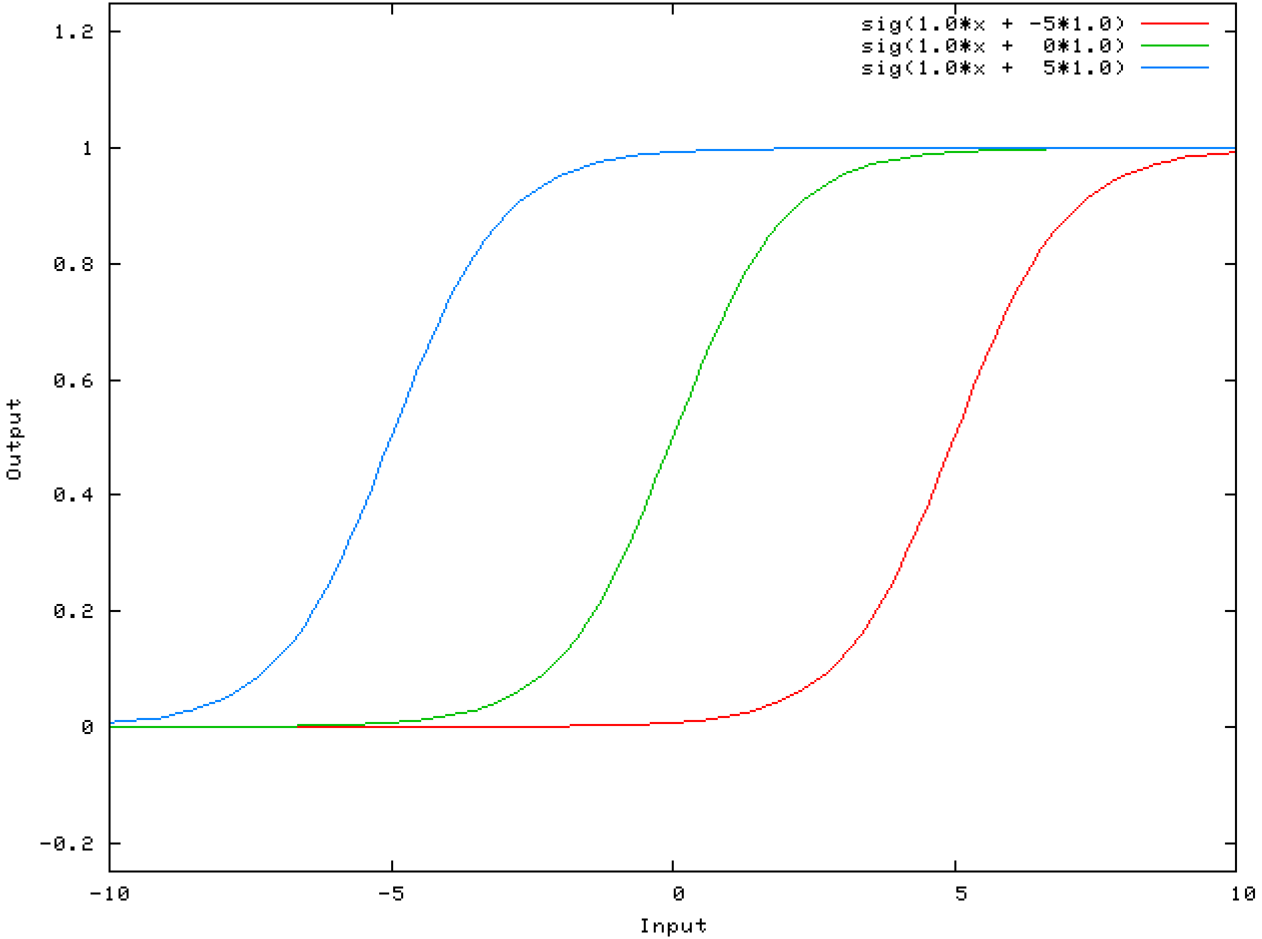
IMPACT OF WEIGHTS ON ACTIVATION FUNCTION

Sigmoid

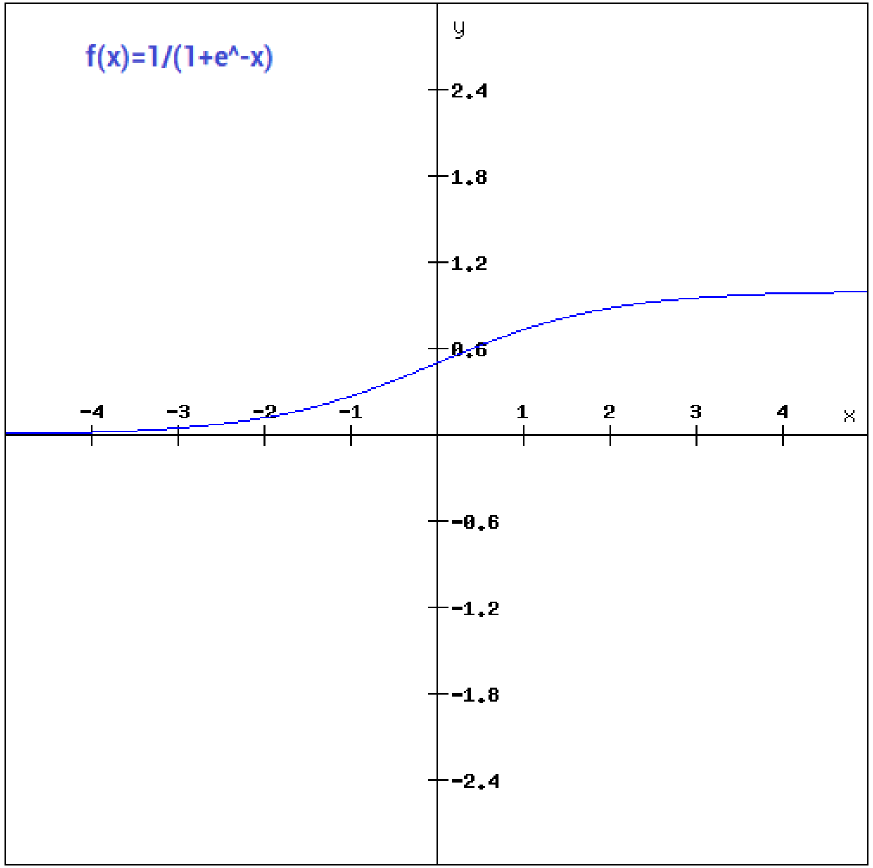


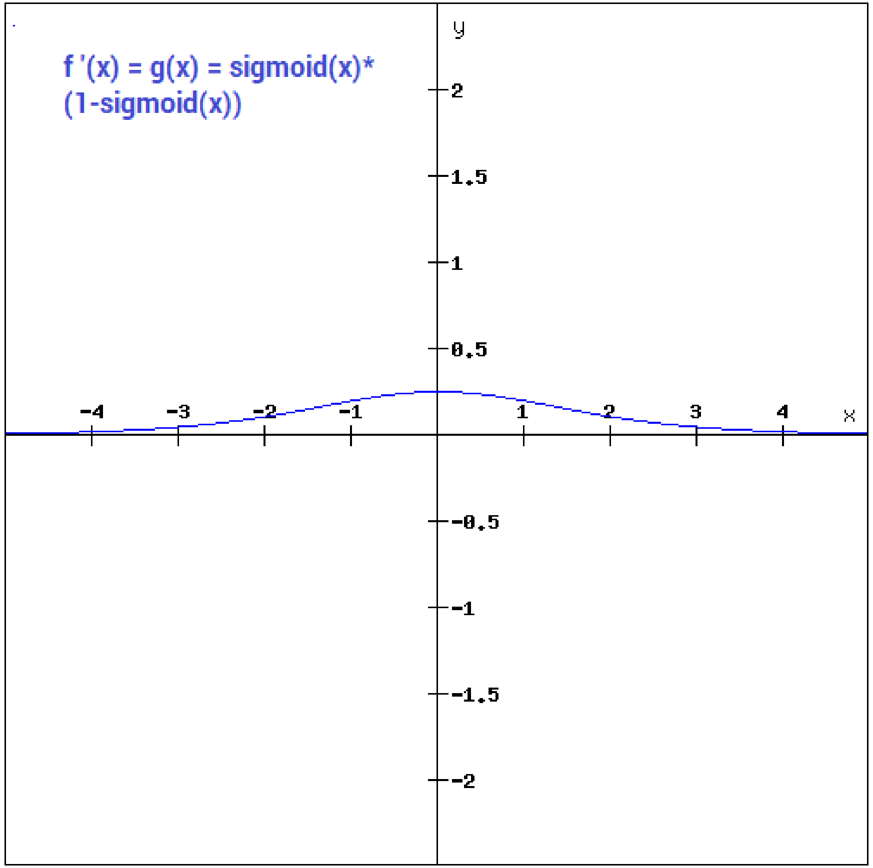
IMPACT OF BIAS ON ACTIVATION FUNCTION

Sigmoid



GRADIENT OF SIGMOID





<u>How much should I change the curve (slope)?</u>
<u>In what direction?</u>

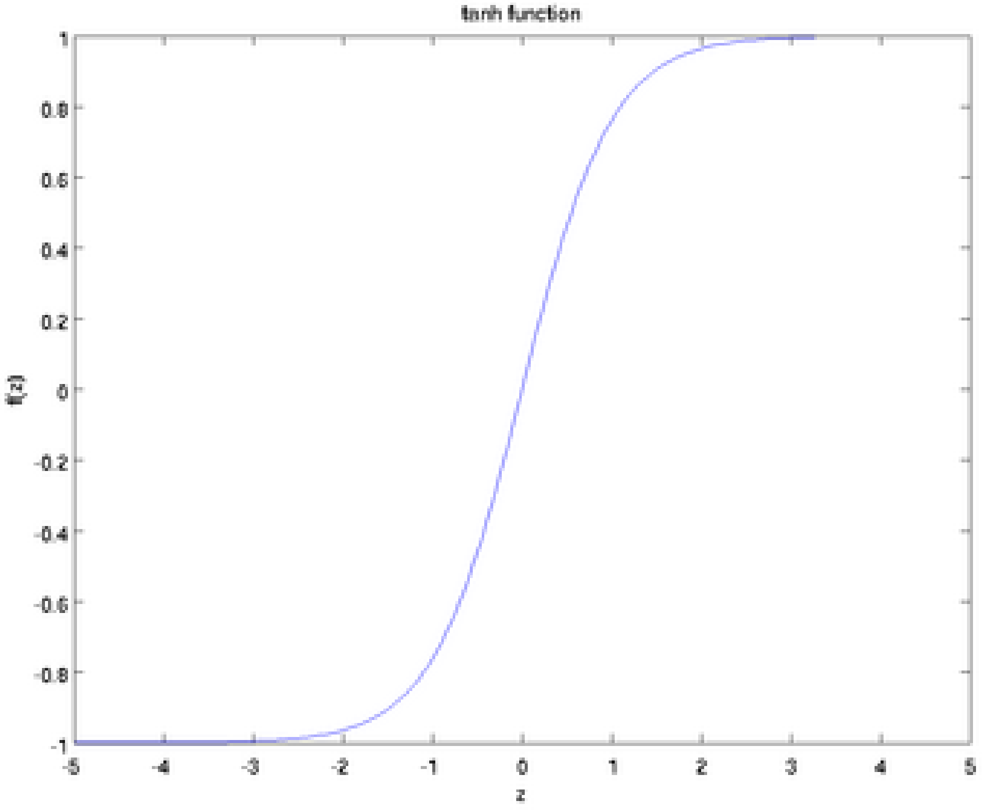
ACTIVATION FUNCTION

Tanh

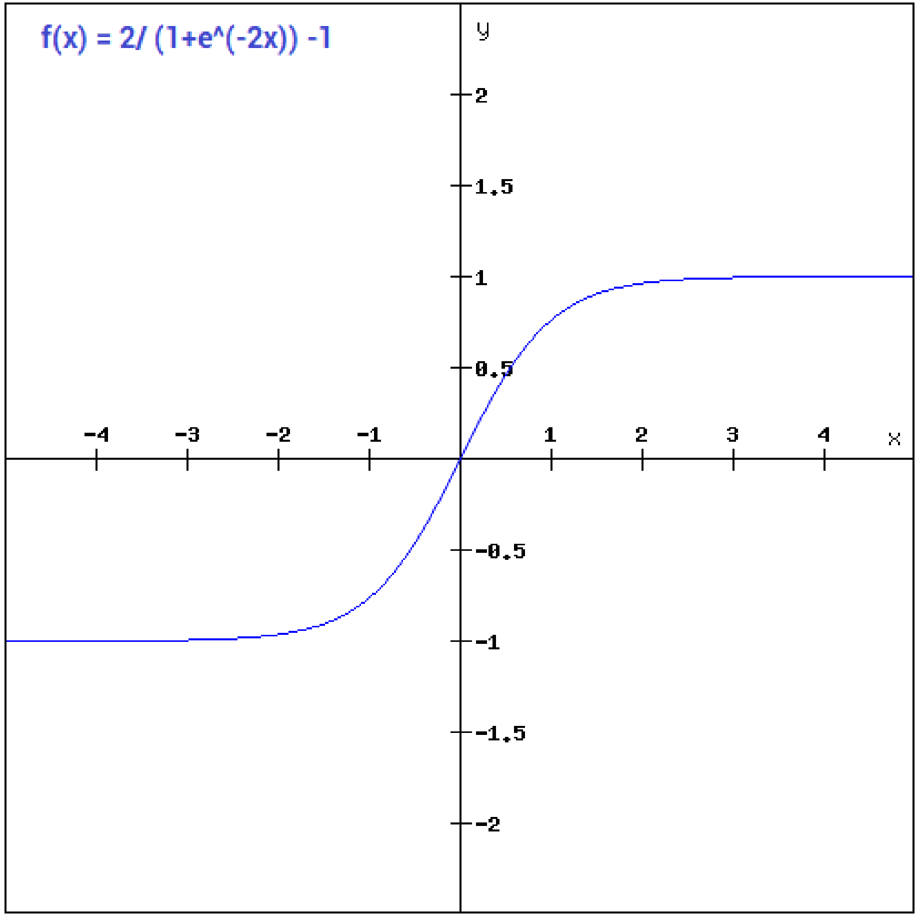
- Hyperbolic Tangent (In Keras, ‘tanh’)
- Values between -1 and 1
- Differentiable
- Scaled Sigmoid function

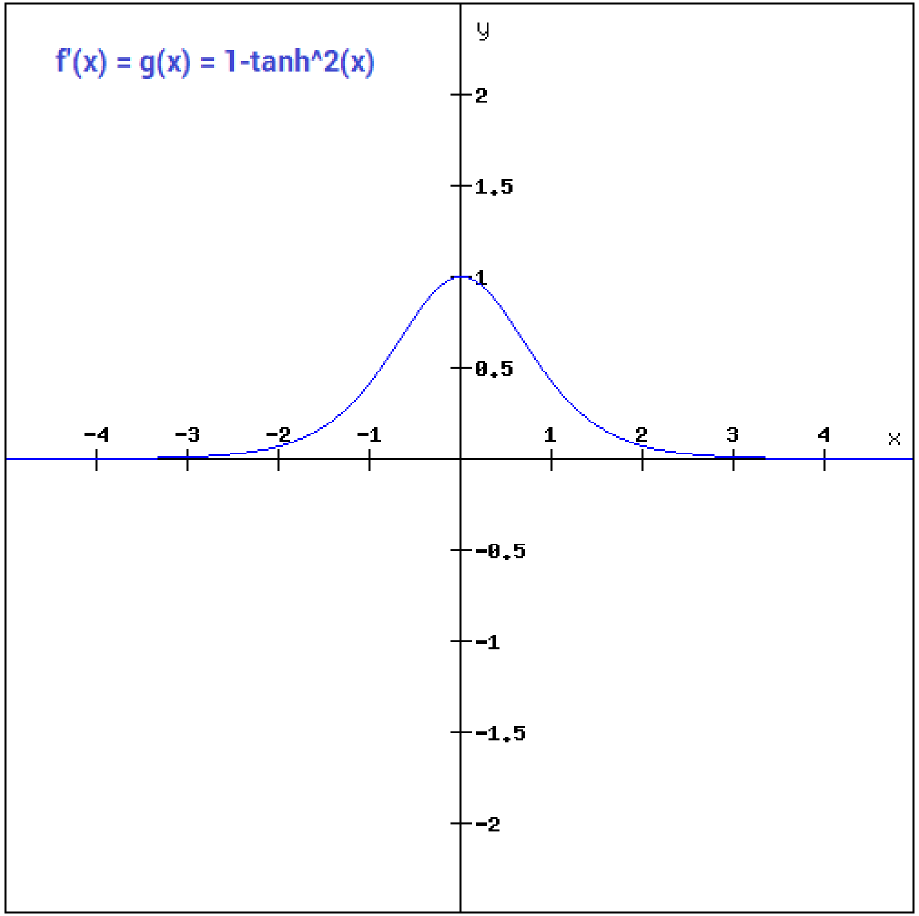
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} =$$

$$= \frac{e^{2x} - 1}{e^{2x} + 1} = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$



TANH - GRADIENT





ACTIVATION FUNCTION

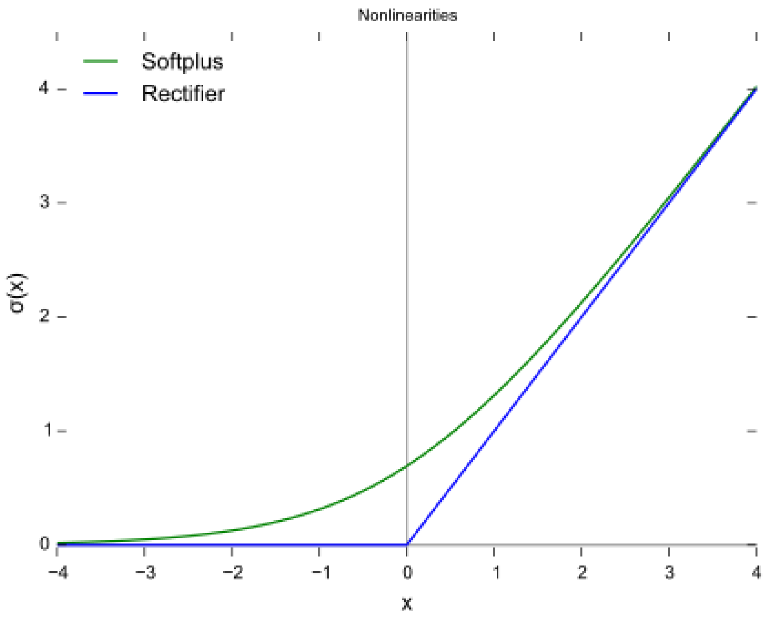
ReLu

- Rectified linear unit activation function. In Keras ('relu')
- $f(x) = \max(0, x)$
- Better training performance is usually achieved using relu
    - Analytical derivative wrt x, but not continuous
    - Most popular activation function right now, all CNNs
- **Negative values snap to 0 and give no information**

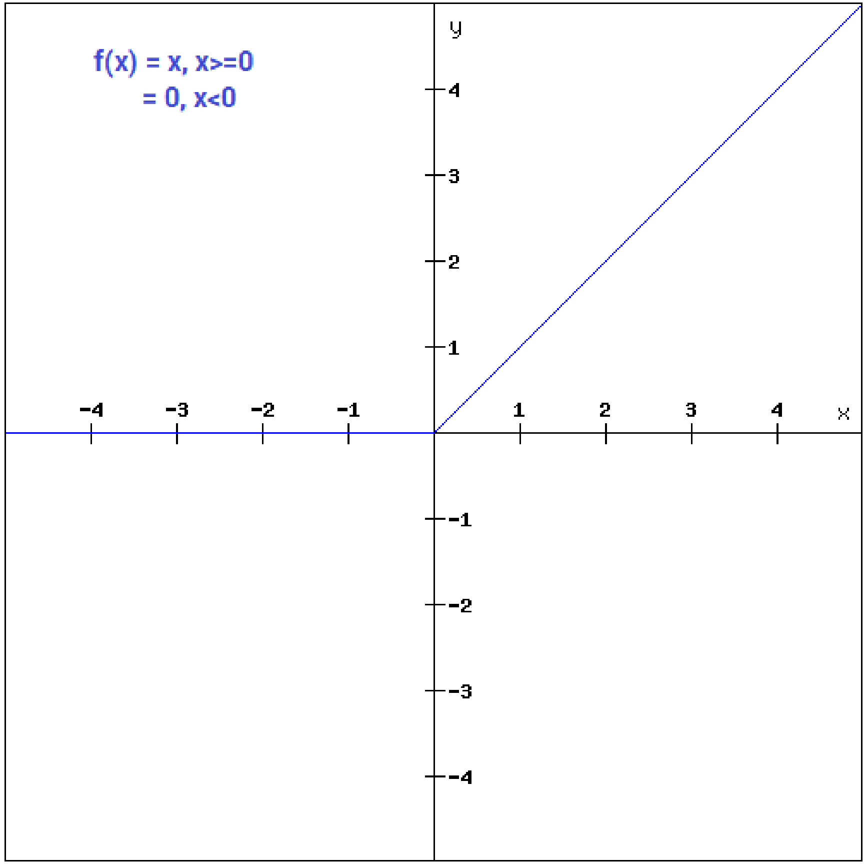
  Smooth approximation:

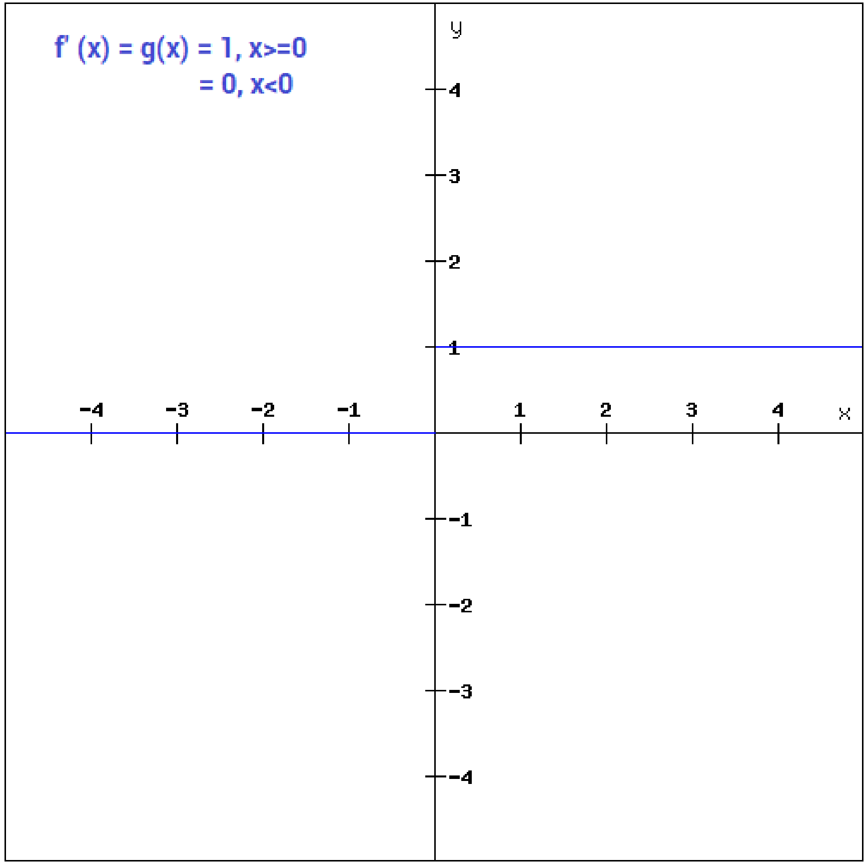
  $f(x) = \log(1 + \exp(x))$

    - Also see Leaky ReLU
- <u>Excellent initial activation function</u>



RELU GRADIENT



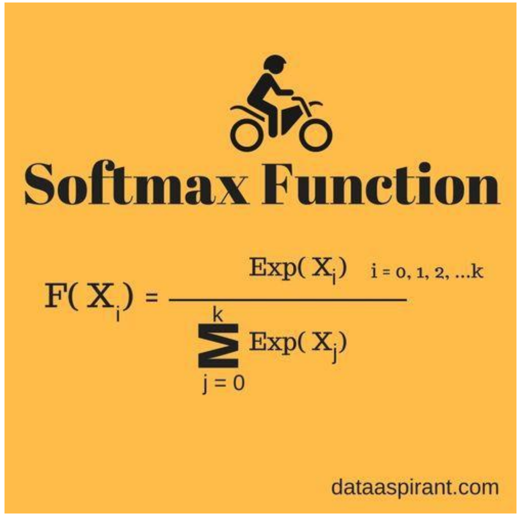


ACTIVATION FUNCTION - SOFTMAX

$$\sigma : \mathbb{R}^K \to (0,1)^K$$

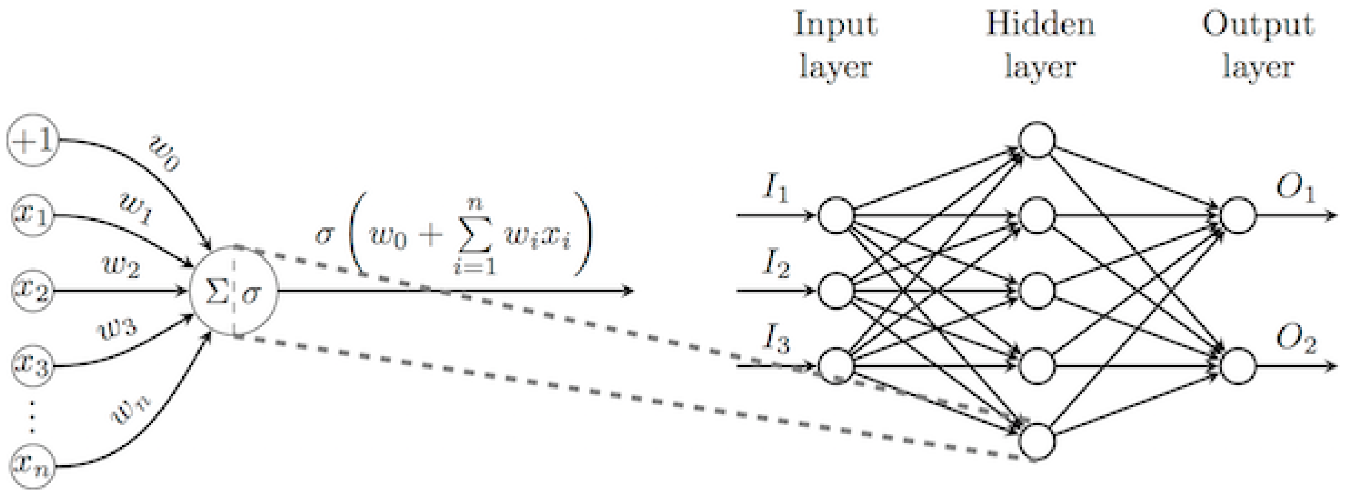
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- In mathematics, the softmax function, or normalized exponential function, is a generalization of the logistic function (In Keras, 'softmax')
- Neural Networks are commonly trained under a log loss (or cross-entropy) regime
- Probabilities will be in range of 0 to 1
  - Sum of all probabilities is 1
  - Often used in last layer



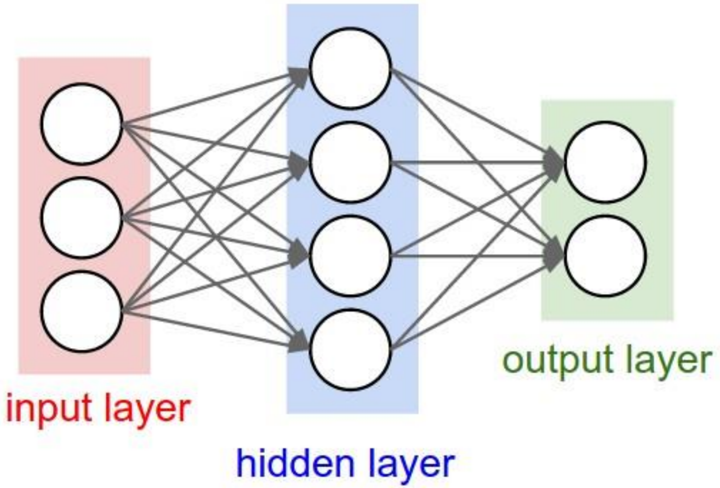
MULTI-LAYER PERCEPTRON

- Multi-layer Perceptron – several perceptrons
- Includes bias (w_0) per neuron
- Cybenko's universal approximation theorem, a (wide enough) MLP with a single hidden layer of sigmoid neurons is capable of approximating *any* continuous real function on a bounded interval
- The proof of this theorem is not constructive, and therefore does not offer an efficient training algorithm for learning such structures



MULTI-LAYER PERCEPTRON

- Conceptually simple by defining layers
- For many years 0-1 “hidden layers” were used
    - “hidden” means they are not the input or output
- Deep Neural Network can have tens, hundreds, or thousands of hidden layers
- Many neurons in layers
- All neurons in one layer are connected to neurons in previous layer and in next layer (left to right)
    - FCN = Fully Connected/convolutional Network



MULTI-LAYER PERCEPTRON - PARAMETERS

- There are unknown parameters that are found via training
  - Weights (1 per connection from layer to layer)
  - Bias (1 per neuron)
- They are found by “reducing” the error (optimization problem)
  - Error = Correct output – Computed output
  - We are optimizing the loss function
- You do not have to program these parameters nor a way to discover them
  - The NN does this for you (“Software writing software”)

MULTI-LAYER PERCEPTRON - PARAMETERS

- During training, the Neural Network determines the weights and biases (design parameters)
  - Can result in a HUGE number of parameters
  - GPT-2 8B
    - Transformer-based language model for generative language modeling
    - Designed to predict and generate text (e.g.- write the next sentence in a document given the initial paragraph)
    - 8.3B parameters
  - GPT-3 175B
  - Megatron-Turing NLP Model (NVIDIA, Microsoft) – 530B
- We have no real control of the design parameters during training
  - Many other design parameters
  - Physics-informed neural networks

MULTI-LAYER PERCEPTRON - PARAMETERS

- Sometimes interesting things happen during training and none of it is under our control
  - The NN discovers relationships between inputs and the outputs WITHOUT human intervention
  - Sometimes these relationships are new (previously not known)
    - Gives rise to the concept of “artificial intelligence”

FIRST NEURAL NETWORK

WHICH FRAMEWORK SHOULD I PICK?

Loads of possibilities

- Tensorflow
- Caffe2, NVCaffe
- PyTorch
- MXNet
- Microsoft Cognitive Toolkit
- Paddle, Paddle
- Chainer
- PlaidML
- Theano
- Torch

- Which one should you pick?
  - Frameworks change quickly
  - Lots of interface options:
    - Python, C/C++, R, Julia, Matlab, Perl, Wolfram Language, Java (There is a Fortran framework!)
- Pick a framework to get started
  - Make sure it supports your interface language
  - You can always change later
- Or - Pick a higher-level tool such as Keras

SWISH PYTHON CODE IN KERAS

```
from keras.backend import sigmoid

def swish(x, beta = 1):
    return (x * sigmoid(beta * x))
```

```
from keras.utils.generic_utils import
get_custom_objects
from keras.layers import Activation

get_custom_objects().update({'swish':
Activation(swish)})
```

```
model.add(Flatten())
model.add(Dense(256, activation = "swish"))
model.add(Dense(100, activation = "swish"))
model.add(BatchNormalization())
```

- Create your own "swish" function for Keras

KERAS- WHAT IS KERAS?

- High-level Neural Network API (Python, R)
- Written in **Python**
- Started out as interface capable of running on top of TensorFlow, CNTK, Theano, MXNet, PlaidML
- As of Keras 2.8.3, it is frozen for all but TensorFlow
  - Moving forward it will be developed only in TensorFlow (it's the main Python coding interface)
- Supports Fully Connected networks (FCN), convolutional networks (CNN), and recurrent networks (RNN) (also LSTM's)
- Runs on CPU and GPU
- **Horovod** integration (Distributed training framework)

KERAS INSTALLATION – PYTHON 3

- Install Keras:
  - `pip install keras`
  - `conda install keras`
    - Also installs tensorflow (backends)
    - Can install GPU version (`tensorflow-gpu`)
- Check:
  - `python -c "import keras; print keras.__version__”`
  - `python -c "from keras import backend; print backend._BACKEND“`
  - `KERAS_BACKEND=theano`
    - `python -c "from keras import backend; print(backend._BACKEND)"`

FIRST NEURAL NETWORK

Classifier/Predictor

- Pima Indian Nation - onset of diabetes dataset
    - Originally from National Institute of Diabetes and Digestive and Kidney Diseases
        - Objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset
        - Predictor variables:
            - **Pregnancies**: Number of times pregnant
            - **Glucose**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
            - **BloodPressure**: Diastolic blood pressure (mm Hg)
            - **SkinThickness**: Triceps skin fold thickness (mm)
            - **Insulin**: 2-Hour serum insulin (mu U/ml)
            - **BMI**: Body mass index (weight in kg/(height in m)^2)
            - **DiabetesPedigreeFunction**: Likelihood of diabetes w.r.t. family history
            - **Age**: Age (years)
            - **Outcome**: Class variable (0 or 1): 0 = no diabetes, 1 = diabetic

FIRST NEURAL NETWORK

- The data is in csv format

https://raw.githubusercontent.com/keipertk/fdnn/master/pima-indians-diabetes.data.csv

- Goal:
  - Develop model that can predict if a patient is diabetic or not, based on test data

FIRST NEURAL NETWORK

Data Exploration

Number of pregnancies	Glucose	Pressure	Skin Thickness	Insulin	BMI	DPF	Age	Diabetic Yes = 1 No = 0
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0

Label (Ground Truth)

Categories

NN TRAINING STAGES

1. Create data
2. Load data
3. Define Model
4. Compile model
5. Fit model (train!)
6. Evaluate Model
7. Predictions (Inference)

1. CREATE DATA

- Perhaps the most difficult step (can be over 50% of your time and as much as 80%)
- Gather data, look for outliers, filter it, examine data (plot it, statistical analysis)
  - Descriptive Statistics
    - Data range, median, mean, deviations, moments, etc. Logistic regression analysis.
- Plot the data!!!
- Create labels
- If you think you have enough data, you need more ☺
  - Can derive new data from existing data (image data)
  - Need lots of variety (if possible)
- May need to normalize, scale, perhaps even rotate data
- Label data!!

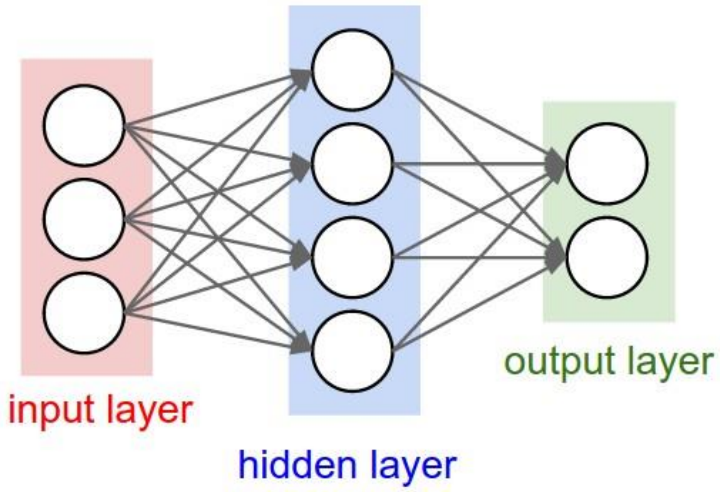
2. LOAD DATA – LOAD DATA INTO PYTHON

```
from keras.models import Sequential
from keras.layers import Dense
import numpy

# fix random seed for reproducibility
numpy.random.seed(7)

# load pima indians dataset
dataset = numpy.loadtxt(
   "pima-indians-diabetes.data.csv",
   delimiter=",")
# split into input (X) and
#    output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
```

- Pretty straightforward
  - In our case, csv data (use numpy)
  - Can be database (useful for images)
- X is “input” or predictor values – 8
  - Numpy list (array)
- Y is output (output) – 1
  - Numpy list (array)

3. DEFINE MODEL

- In Keras, the model is defined as a sequence of layers
- Create a Sequential model and add layers one at a time
- In this example, we will use a fully-connected network structure with three layers
  - FCN means every neuron in one layer is connected to every neuron in the next layer
  - Classic Multi-Level Perceptron
  - Designing layers is something of an art
    - Experiment
    - Steal!!! (transfer learning)
- Fully connected layers are defined using the *Dense* class in Keras



3. DEFINE MODEL

- Initialize the network weights to a small random number generated from a uniform distribution (‘uniform‘)
    - Between 0 and 0.05 (Keras default uniform weight initialization)
        - Another traditional alternative would be ‘normal’ for small random numbers generated from a Gaussian distribution
- Don’t use zero initialization – can lead to problems
- Random values for biases as well
- Kernel_initializer=`uniform` in layer definition.

3. DEFINE MODEL

```
# create model
model = Sequential()
model.add(Dense(12, input_dim=8,
   activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

- First hidden layer has 12 nodes and expects 8 input variables and uses the relu activation function
- Second hidden layer has 8 outputs and also uses the relu activation function
  - Keras determines that it has 12 inputs from previous layer
- Output layer has 1 output to predict the class (onset of diabetes or not)
  - Sigmoid activation function
  - Keras determines it has 8 inputs

3. PRINT MODEL SUMMARY

Great Information about Your Model

```
model.summary()
```

```
In [4]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 12)                108
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 104
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 221
Trainable params: 221
Non-trainable params: 0
_________________________________________________________________
```

4. COMPILE YOUR MODEL

- Compiling the model uses the most efficient numerical libraries in the backend
    - The backend automatically chooses the best representation of the network for training and making predictions for your hardware, such as CPU or GPU, or even distributed
- We must specify some additional properties required for training
    - Training a network means finding the best set of weights to make predictions for the problem
        - Optimization problem

4. COMPILE MODEL

- What do we optimize?
  - Need the function to optimize (minimize)
  - For classification problems it is typically the loss function [loss = true - predicted]
  - Variables are the different weights and biases in the model (network) - synapses
- For this problem the logarithmic loss function is the “objective function”
  - Defined in Keras as “binary_crossentropy“

$$\mathcal{L}(\hat{y}, y) = -\frac{1}{N}\sum_{i}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$$

4. COMPILE MODEL

- Define Optimization algorithm:
  - Gradient descent algorithm “adam” (Adaptive Moment Estimation)
  - Good place to start (very common optimizer for learning)

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

$$w := w - \eta\nabla Q(w) = w - \eta\sum_{i=1}^{n} \nabla Q_i(w)/n,$$

- In adam, the learning rate h, is reduced over time (also called the step size)
  - Reduce the step size as we approach the optimal solution

4. COMPILE MODEL

- In addition to an objective, we will collect other “metrics” to monitor
- By default Keras reports loss
    - There’s no one-size-fits-all loss function to algorithms in machine learning
    - Since we minimizing, we want the loss function to go down
- We will also report the classification accuracy as a metric
    - Accuracy is the percentage of “correct” answers

4. COMPILE MODEL

```
# Compile model
model.compile(loss='binary_crossentropy',
   optimizer='adam', metrics=['accuracy'])
```

- “metrics” is a list of additional things that we want to track (in addition to loss)
  - Accuracy is almost always tracked
- Available metrics:
  - binary_accuracy
  - categorical_accuracy
  - sparse_categorical_accuracy
  - top_k_categorical_accuracy
  - sparse_top_k_categorical_accuracy
  - Custom metrics

5. FIT MODEL (TRAIN!)

- We train or fit our model on the loaded data by calling the fit() function on the model
- Training process will run for a fixed number of iterations through the dataset
  - One pass through all of the data is called an epoch
  - We must specify using the number of epochs (nepochs)
- We can also set the number of data points that are evaluated before a weight update in the network is performed
  - "Batch size" using the batch_size argument
- Accumulates the error during the batch
  - Averages error over the batch
  - One back propagation for entire batch

5. FIT MODEL (TRAIN!)

```
# Fit the model
history = model.fit(X, Y, epochs=150,
                    batch_size=10)
```

- For our problem, we will use 150 epochs (relatively small)
  - Batch size of 10
  - These can be chosen experimentally by trial and error
- Batch size can help performance a great deal
  - It can impact how quickly the model converges
  - Overall, use a larger batch size, particularly for large data sets (runs faster)
    - For GPUs, recommended to make batch size to fit as much data in GPU memory as possible

5. FIT MODEL (TRAIN!)

- A “batch” is the number of training examples used in one iteration, before updating model parameters
    - The error using backpropagation is “averaged” over the number pieces of data
    - This can slow learning a bit, but greatly improves epoch time
- As optimizer works through data (epochs) it will randomize the order of the training data
    - Varies with epoch
    - Create more robust trained model since the error will vary
- Prevents “overfitting”
    - Overfitting is where the model works VERY well on training data but terrible on test data
        - It “memorizes” the training data set

5. FIT MODEL (TRAIN!)

- For this problem, the model is trained on entire dataset (small dataset)
  - Not really a problem since we are just learning
- We can evaluate the performance of the network on the same dataset
  - Give us an idea of how well we have modeled the dataset (e.g. training accuracy), but no idea of how well the algorithm might perform on new data
- You could separate your data into training and test datasets
  - Training dataset for training model
  - Testing dataset is used to evaluate model on “unknown” data (after training)
  - Evaluating the model using the training data is something of a pointless exercise

6. EVALUATE MODEL

```
# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%"
    %(model.metrics_names[1], scores[1]*100))
```

- Evaluation = input data into trained model
- You can evaluate your model on your training dataset using the `evaluate()` function on your model and pass it the same input and output used to train the model
    - We are not training the model but “testing it”
- Evaluating the test data may be somewhat pointless but the point is to understand how to evaluate new data using the trained model.

TRAINED

- Congratulations – you just trained your first Deep Learning Model!!!
    - Pick up your laminated certificate upon leaving ☺
- Seriously – congratulations!!!

VISUALIZING THE HISTORY

- It would be nice to visualize loss function vs. epochs (progress of training)
  - Accuracy vs. epochs, loss vs. epochs
- In Keras, “history” callback records all training metrics for each epoch
  - Includes the loss and the accuracy
  - Returned from calls to the fit() function
- Metrics are stored in a dictionary in the history of the object

VISUALIZE TRAINING HISTORY

```
…

# Fit the model
history = model.fit(X, Y, epochs=150,
batch_size=10)

# list all data in history
print(history.history.keys())
…
dict_keys(['loss', 'acc'])
```

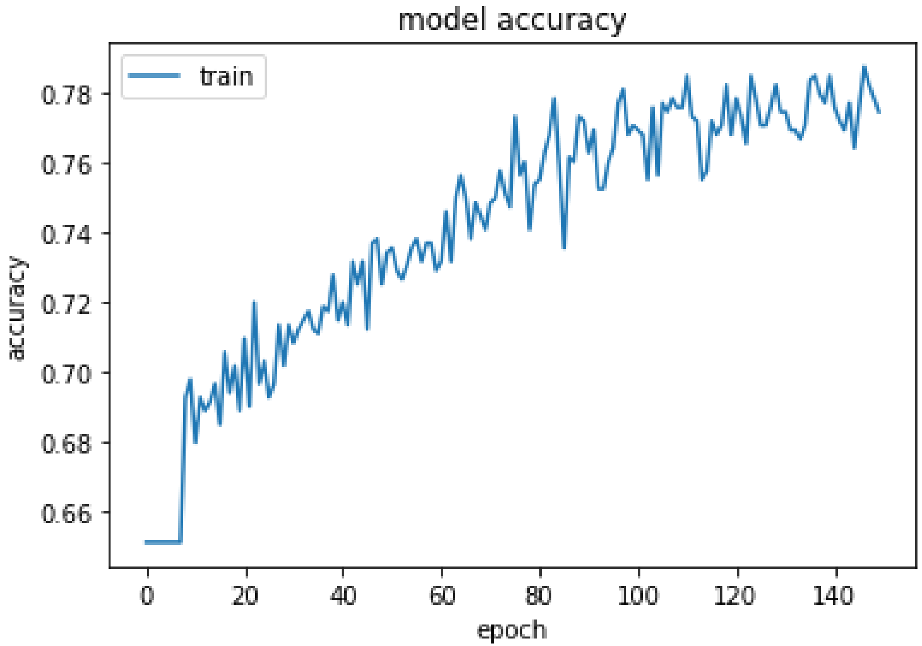
- “Output” is in history object (dictionary)
- Take a look at what “keys” are in the dictionary
- Metrics (loss and accuracy)
    - This is for the training dataset – we don’t have a test or validation set (yet)

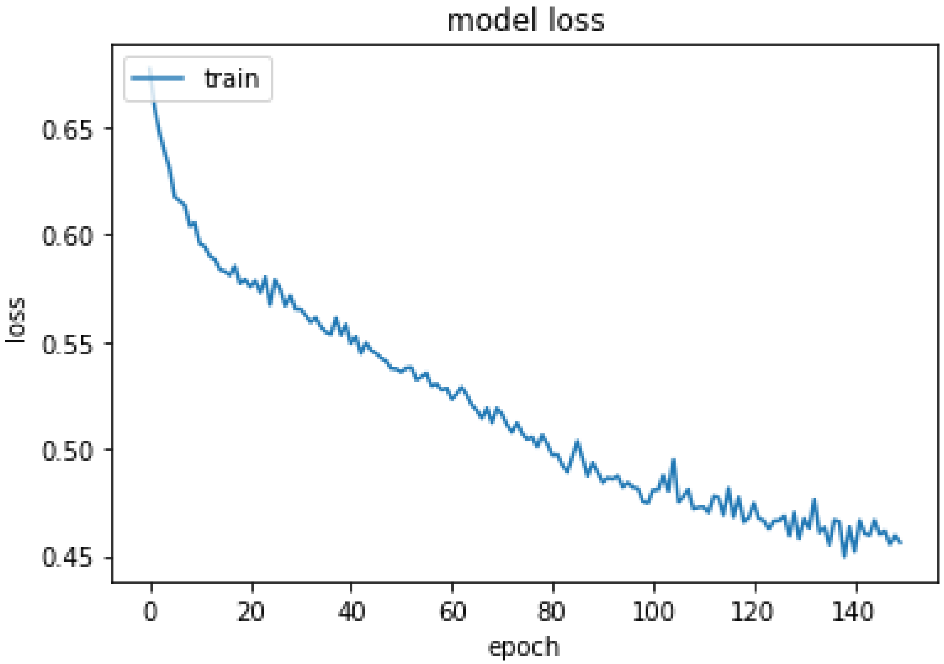
VISUALIZE TRAINING HISTORY

```
import matplotlib.pyplot as plt

# summarize history for accuracy
plt.plot(history.history['acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
```

- Create plots using matplotlib
- Plot accuracy history
- Plot loss history

VISUALIZE TRAINING HISTORY





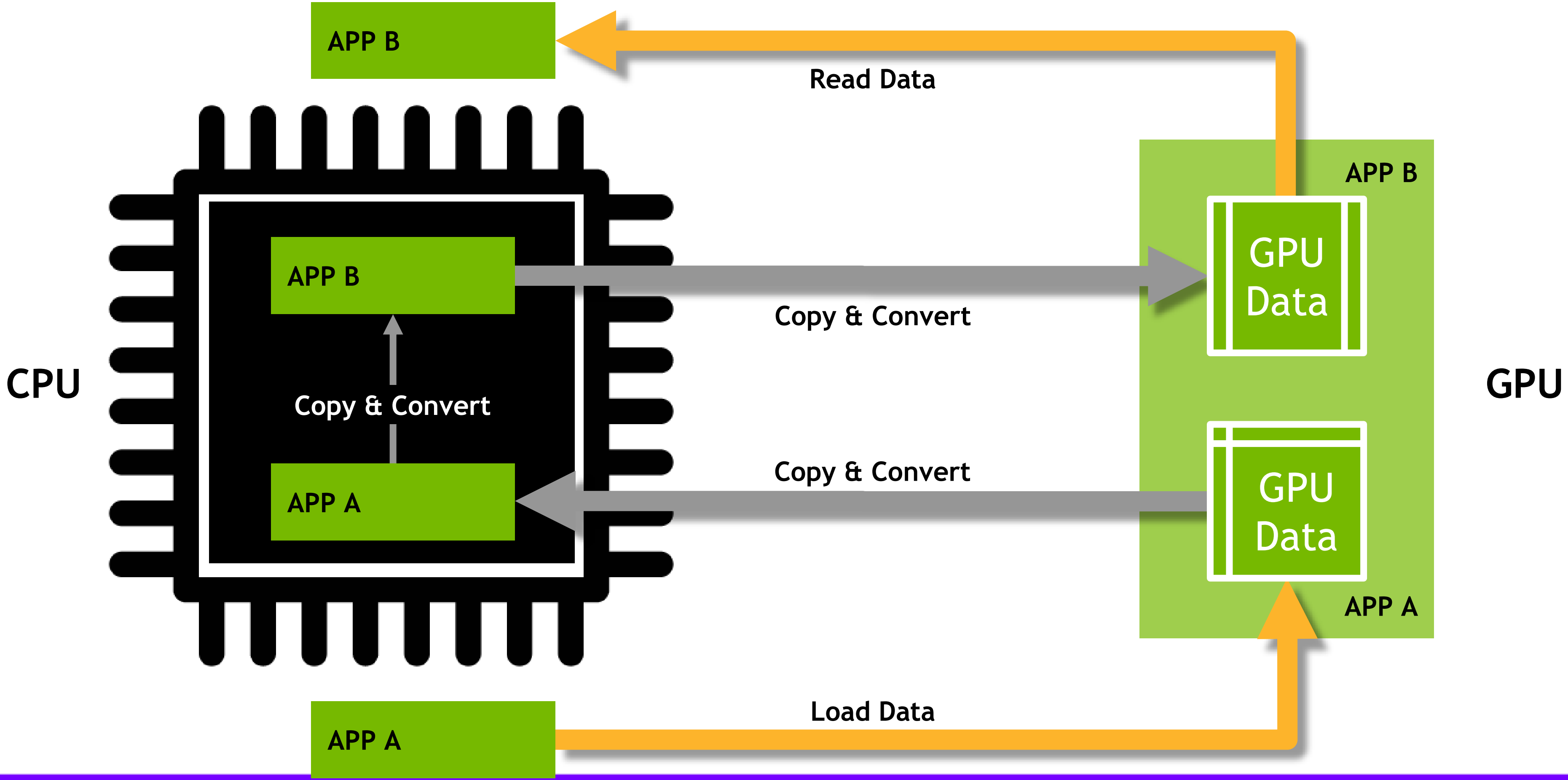
Don’t worry about the results for now

SUMMARY

- Keras is a great way to get started with Deep Learning
- Fairly easy to create a Fully Connected Network model (FCN)
    - Just a few lines of Python
    - Several steps
- Various tools/options to help train the model and visualize history
- Next steps – Advanced Options

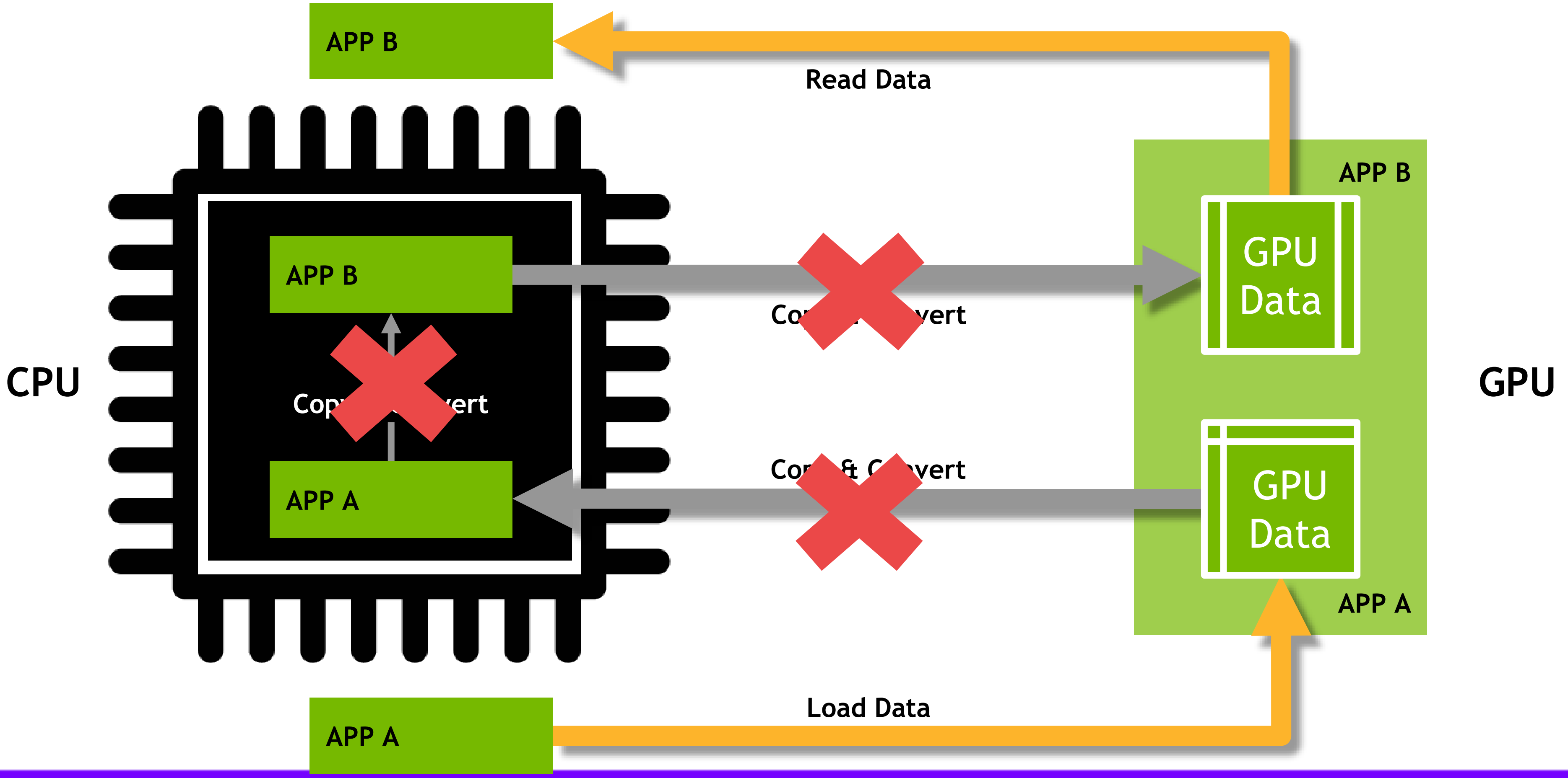
Data Movement and Transformation

The bane of productivity and performance



Data Movement and Transformation

What if we could keep data on the GPU?



What is RAPIDS?

End-to-End GPU Accelerated Data Science

Data Preparation/ETL → Analytics/ML/Graph → Visualization

cuDF

- GPU-accelerated ETL functions
- Tracks pandas and other common PyData APIs
- Dask + UCX integration for scaling

cuML

- GPU-native ML library, including XGBoost, FIL, HPO, and more

cuGraph

- GPU graph analytics, including TSP, PageRank, and more

cuxfilter

- GPU-accelerated cross-filtering

pyViz integration

- Plotly Dash, Bokeh, Datashader, HoloViews, hvPlot

Domain-Specific Libraries

CLX + Morpheus
Cyber log processing + anomaly detection

cuSignal
Signals processing

cuSpatial
Spatial analytics

cuStreamz
Streaming analytics

cuCIM
Computer vision & image processing primitives

node-RAPIDS
Bindings for node.js

...and more!

USING RAPIDS

- Only runs on Linux
- Available via pip and conda
- Use RAPIDS for heavy computational components
  - Modeling algorithms such as
  - Then use Pandas for visualization
- Can do:

```
import cudf as pd
```

```
import cudf as pd
import numpy as np
from time import time

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

wine_set = pd.read_csv("data/winequality-red.csv", sep=';')

wine_set.head(n=5)
wine_set.tail(n=5)
```

NVIDIA: Convolution Neural Network Models

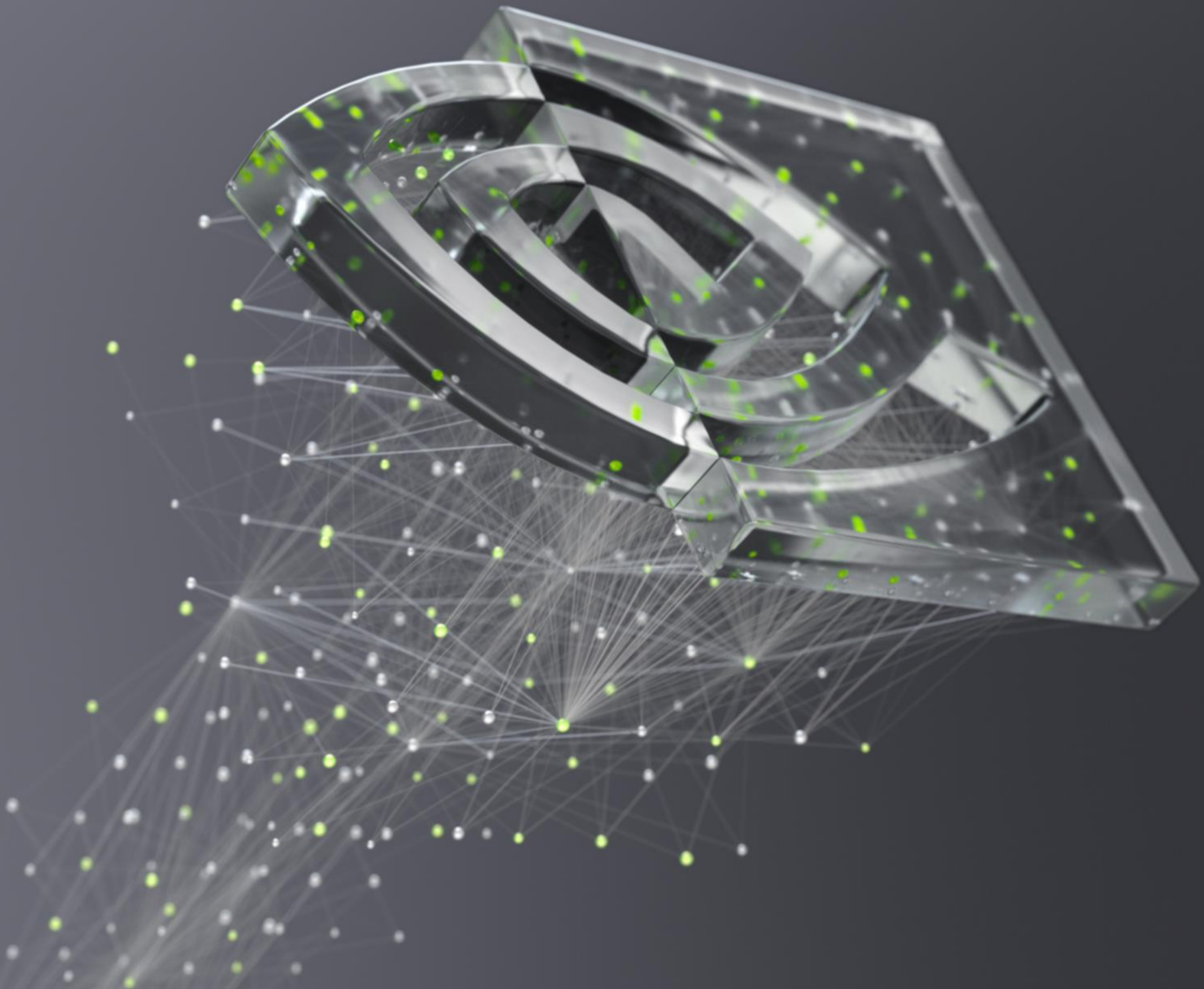
Event Type	Conference/Workshop
Sponsor	Center for Artificial Intelligence Innovation
Virtual	
Date	Mar 9, 2022 3:00 - 5:00 pm
Speaker	Jeff Layton, NVIDIA
Views	25
Originating Calendar	Center for Artificial Intelligence Innovation

Join the Center for Artificial Intelligence Innovation at NCSA for a joint training with NCSA User-Services and NVIDIA on **Wednesday, March 9, from 3-5pm** via Zoom for an online training session NVIDIA: Convolution Neural Network Models

Register for the Zoom session here: https://go.ncsa.illinois.edu/CAIIHALTraining

Kristopher Keipert
Solutions Architect
kkeipert@nvidia.com

Marc West
Account Manager
marcw@nvidia.com



NVIDIA