TensorFlow tutorial

Adam Elwood

1st Terascale School of Machine Learning, Oct 2018

adam.elwood@desy.de

1

Introduction and outline

- This is a tutorial designed to give a very basic introduction to the main concepts behind TensorFlow, consisting of:

  - A very brief overview of neural networks from a TensorFlow standpoint

  - TensorFlow as a computational graph framework

  - How Automatic Differentiation works in the context of TensorFlow

  - A simple example of classifying two different Gaussians with a neural network

- Following this there is an exercise to practice writing your own simple network with TensorFlow in a HEP context

- Thanks to Stefan Wunsch who provided the inspiration and lots of material for this tutorial in the IML workshop in April

  - https://indico.cern.ch/event/668017/contributions/2947042/

  - https://github.com/stwunsch/iml_tensorflow_keras_workshop

Introduction to TensorFlow

- TensorFlow is a low-level implementation of operations needed to implement neural networks in multi-threaded CPU and multi GPU environments

- Differences with **PyTorch** (Facebook vs Google)

  - TensorFlow designed to use static graph definition, you have to build the graph then compile it before running

  - PyTorch allows the graph to be built and executed dynamically

  - TensorFlow is currently the most widely used but PyTorch is gaining popularity

  - Generally speaking: TensorFlow better for production code and PyTorch better for research

  - More info: https://towardsdatascience.com/pytorch-vs-tensorflow-spotting-the-difference-25c75777377b

  - **Disclaimer**: these frameworks are developing so quickly this information may not stay relevant

- **Keras** provides high-level convenience wrapper for backend libraries, predominantly TensorFlow, to implement neural network models

  - Will be covered in much more detail by Christian later

vs

Source for code

- Github repository containing the slides and examples:

  - https://github.com/aelwood/iml_tensorflow_keras_workshop

- Notebooks relevant to this tutorial in 'tensorflow' folder

- Setup instructions (see backup for more specific instructions):

```
# get the repository
git clone https://github.com/aelwood/iml_tensorflow_keras_workshop.git
cd iml_tensorflow_keras_workshop

# install necessary software, this may be already available, to check:
python testSoftware.py
# setup on AWS
source activate tensorflow_p27
# setup with conda (locally)
source init_virtualenv_conda.sh
#or pip (locally)
source init_virtualenv.sh

cd tensorflow
jupyter notebook gaussian.ipynb #for example
```

A simple neural network
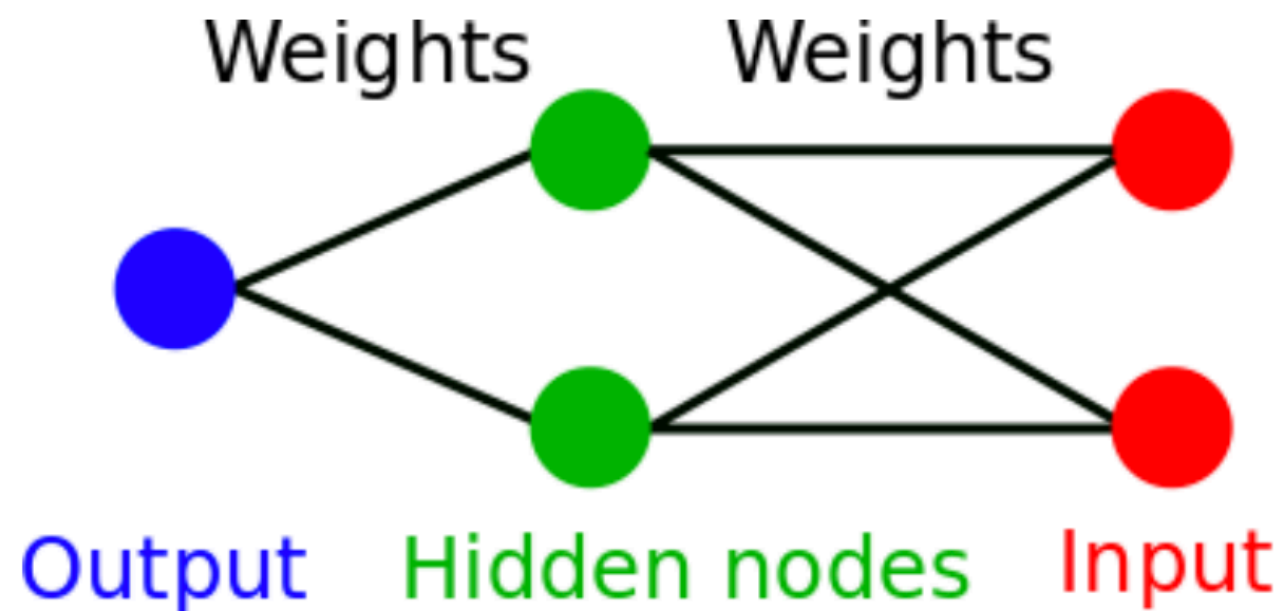
- ▶ **Important:** A neural network is only a mathematical function. No magic involved!
- ▶ **Training:** Finding the best function for a given task, e.g., separation of signal and background.

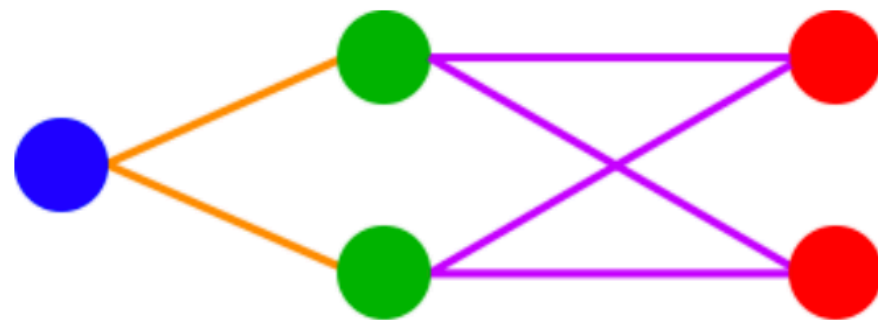# Mathematical representation

▶ **Why do we need to know this?**
→ TensorFlow implements these mathematical operations explicitly.
→ Basic knowledge to understand Keras' high-level layers.
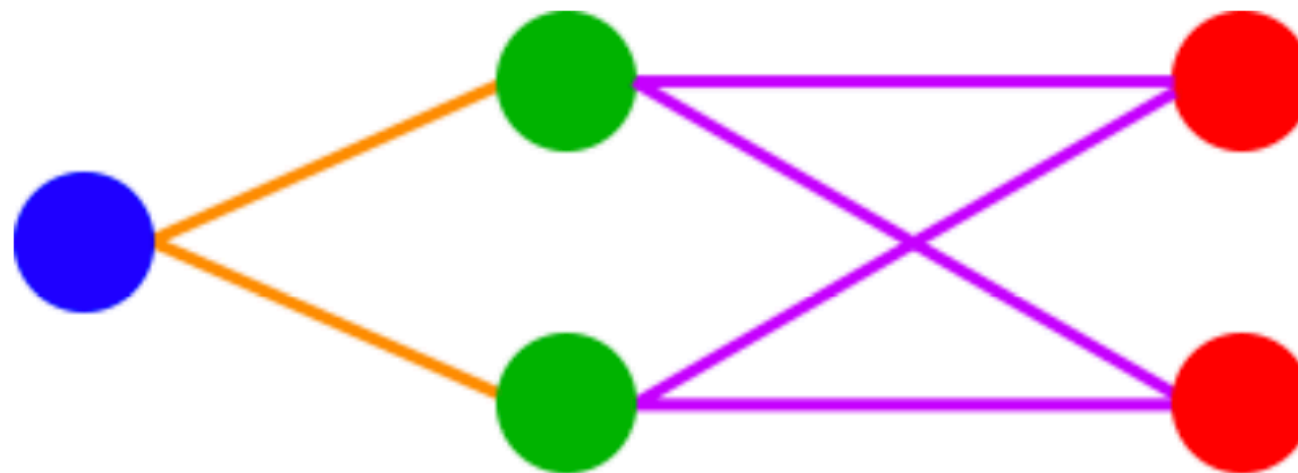
$$f_{NN} = \sigma(b_2 + W_2\sigma(b_1 + W_1 x))$$

Mathematical representation (2)

$f_{NN} = \sigma(b_2 + W_2\sigma(b_1 + W_1 x))$

Input : $x = \begin{bmatrix} x_{1,1} \\ x_{2,1} \end{bmatrix}$

Weight : $W_1 = \begin{bmatrix} W_{1,1} & W_{1,2} \\ W_{2,1} & W_{2,2} \end{bmatrix}$

Bias : $b_1 = \begin{bmatrix} b_{1,1} \\ b_{2,1} \end{bmatrix}$

Activation : $\sigma(x) = \tanh(x)$ (as example)

Activation is applied elementwise!

The "simple" neural network written as full equation:

$$f_{NN} = \sigma_2\left(\begin{bmatrix} b_{1,1}^2 \end{bmatrix} + \begin{bmatrix} W_{1,1}^2 & W_{1,2}^2 \end{bmatrix} \sigma_1\left(\begin{bmatrix} b_{1,1}^1 \\ b_{2,1}^1 \end{bmatrix} + \begin{bmatrix} W_{1,1}^1 & W_{1,2}^1 \\ W_{2,1}^1 & W_{2,2}^1 \end{bmatrix} \begin{bmatrix} x_{1,1} \\ x_{2,1} \end{bmatrix} \right) \right)$$

What is TensorFlow?

TensorFlow is an open source software library for *numerical computation using data flow graphs*. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them.
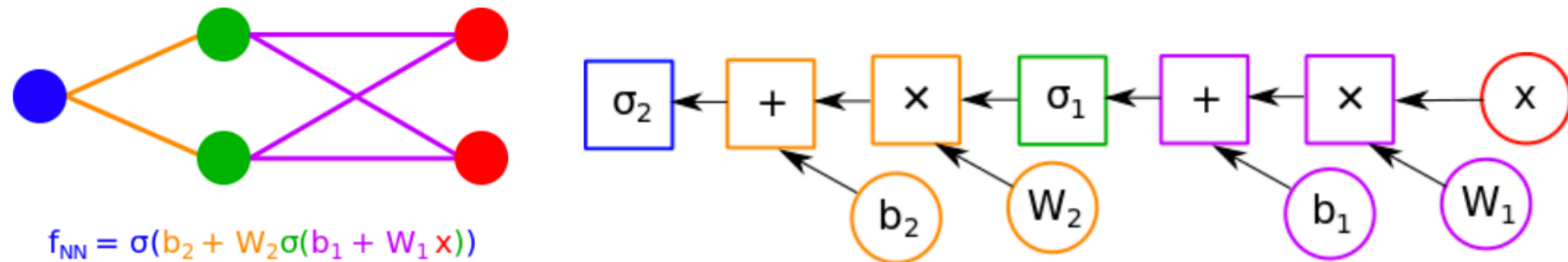
- ▶ **In first place:** TensorFlow is not about neural networks.
- ▶ But it is a **perfect match** to implement neural networks efficiently!
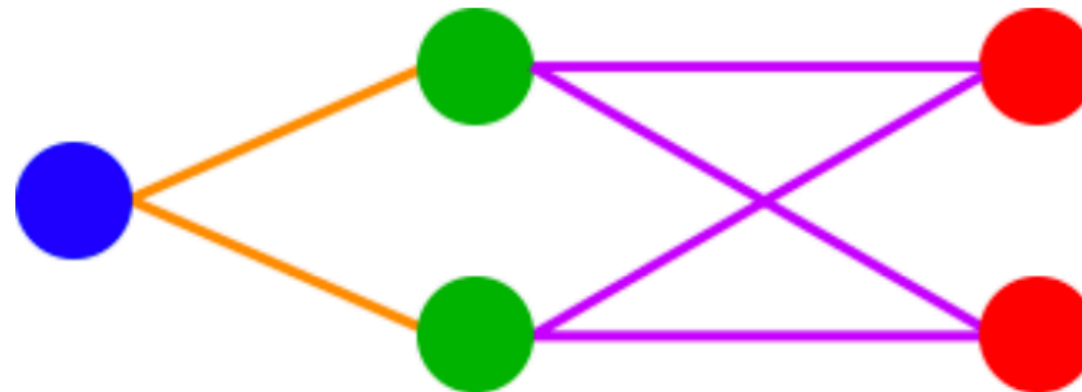
$$f_{NN} = \sigma(b_2 + W_2\sigma(b_1 + W_1 x))$$

Computational graphs

$$f_{NN} = \sigma(b_2 + W_2\sigma(b_1 + W_1 x))$$

**Example neural network** $\rightarrow$ **According computational graph**

- ▶ TensorFlow implements all needed **mathematical operations for multi-threaded CPU and multi GPU** environments.
- ▶ Computation of neural networks using data flow graphs is a perfect match!

  TensorFlow is an open source software library for numerical computation using data flow graphs. *Nodes* in the graph *represent mathematical operations*, while the *graph edges represent the multidimensional data arrays (tensors)* communicated between them.
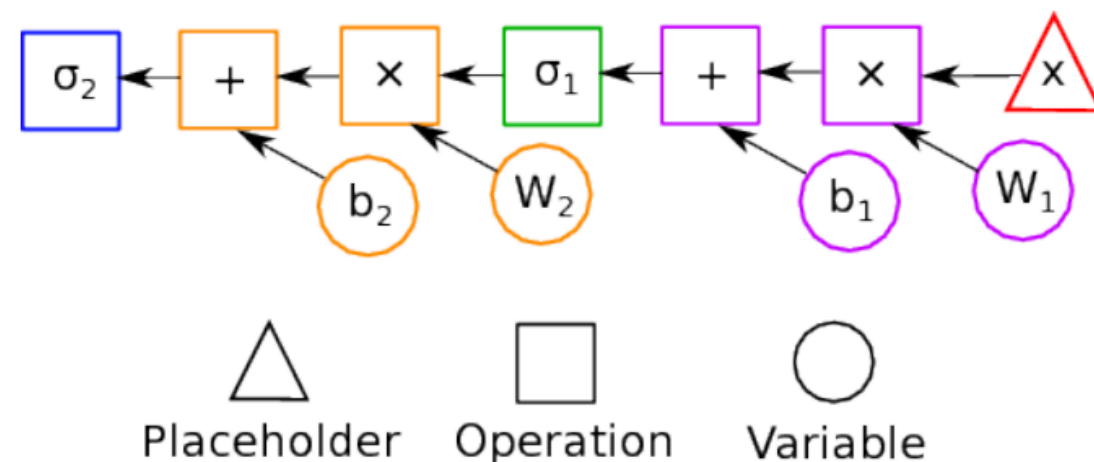
Basic blocks to build graphs in TensorFlow

▶ **Basic blocks:**

   ▶ **Placeholders:** Used for injecting data into the graph, e.g., the inputs x of the neural network
   ▶ **Variables:** Free parameters of the graph, e.g., the weight matrices W of the neural network
   ▶ **Operations:** Functions that operate on data in the graph, e.g., the matrix multiplication of W_1 and x
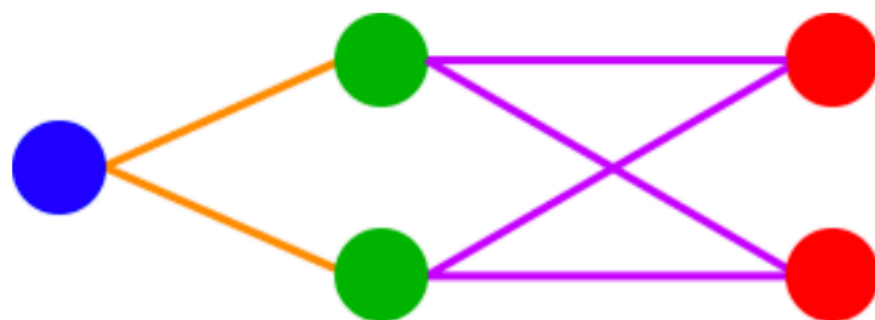
$$f_{NN} = \sigma(b_2 + W_2\sigma(b_1 + W_1 x))$$

Run the graph in a TensorFlow session

▶ A **graph** in TensorFlow can be run inside a **session**.
▶ Following example calculates $y = W \cdot x$ using TensorFlow:

Computational graph:

$$y = W \cdot x = \begin{pmatrix} 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 4 \end{pmatrix} = 11$$

TensorFlow code:

See example:
tensorflow/
xor.ipynb

```python
import tensorflow as tf
import numpy as np

# Build the graph y = W * x
x = tf.placeholder(tf.float32) # A placeholder
W = tf.get_variable("W", initializer=[[1.0, 2.0]]) # A variable
y = tf.matmul(W, x) # An operation

with tf.Session() as sess: # The session
    sess.run(tf.global_variables_initializer()) # Initialize variables
    result = sess.run(y, feed_dict={x: [[3.0], [4.0]]}) # Run graph
```

Automatic differentiation

- During training of neural networks make an excessive use of gradients during optimisation

- (Almost) each operation in TensorFlow is shipped with an inbuilt gradient

- Computation of full gradient using the chain-rule of derivatives through the computational graph

- Explicit TensorFlow call: tensorflow.gradients(z, x)

See example: tensorflow/automatic_differentiation.ipynb
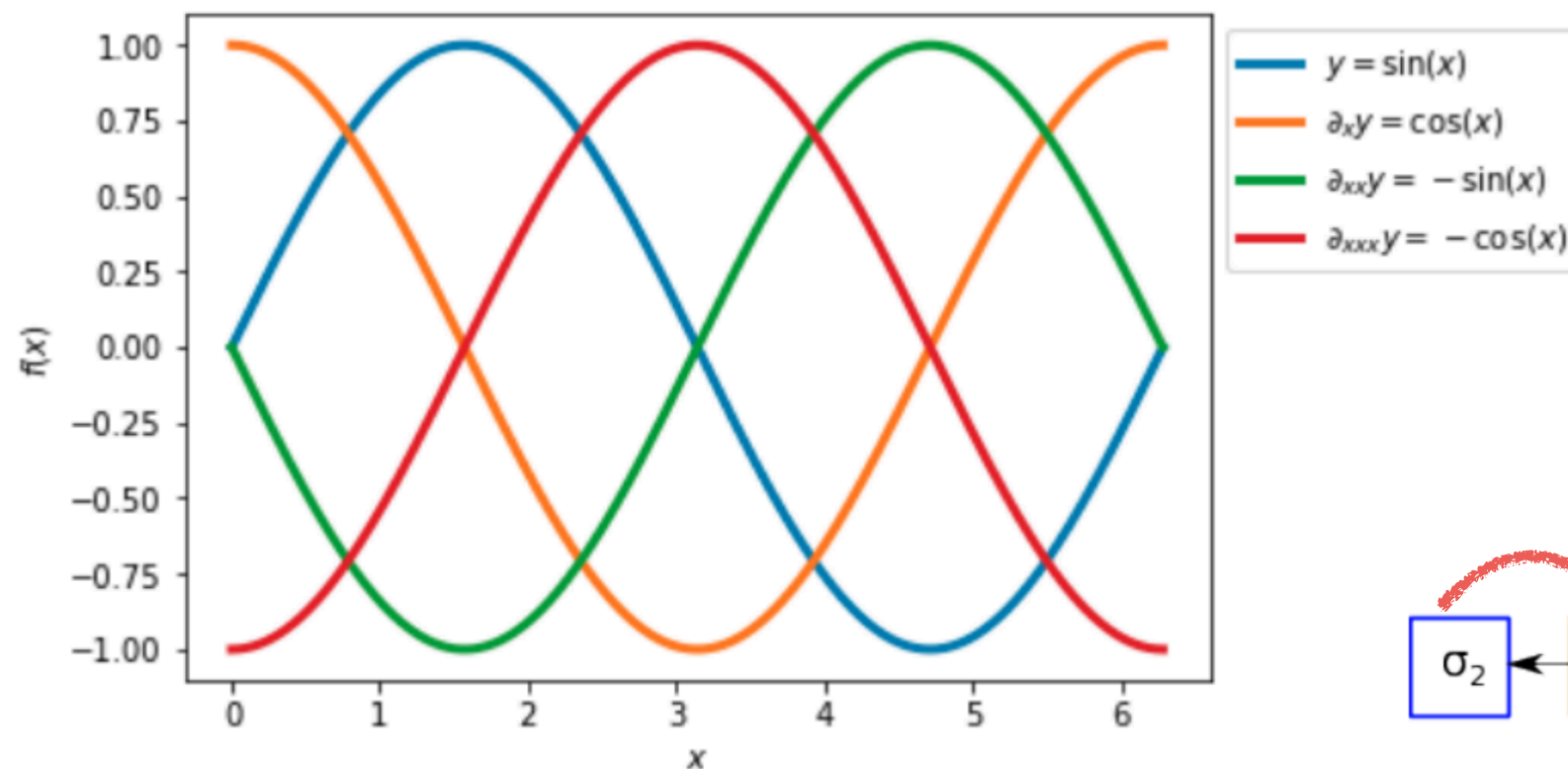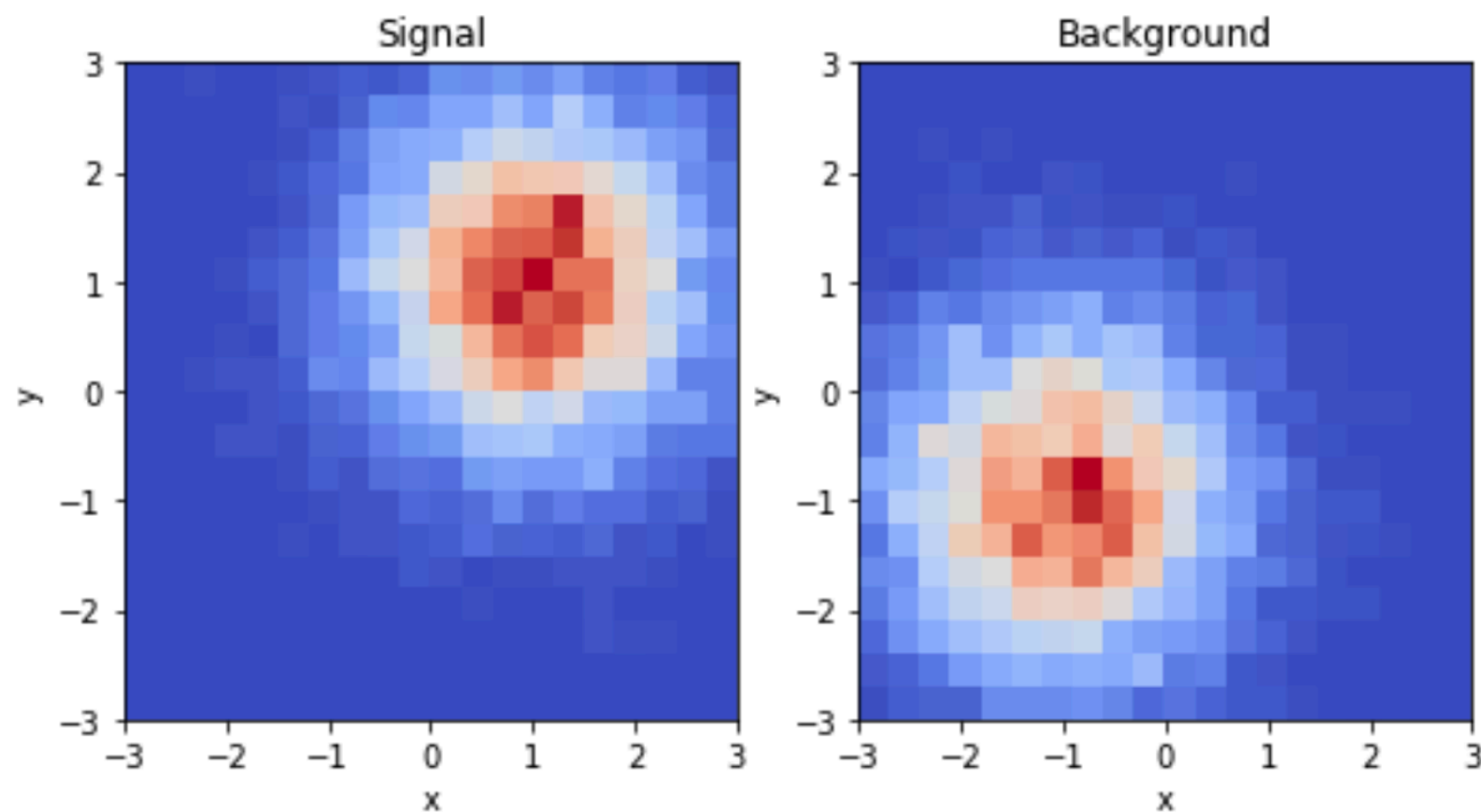

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$
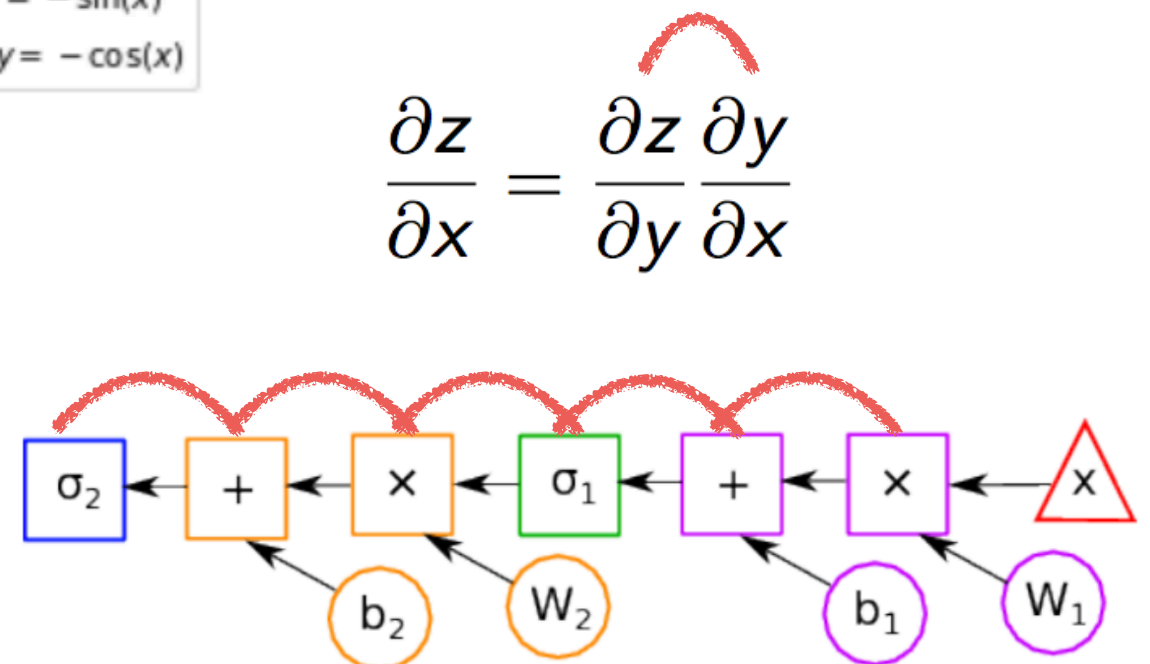
Concrete TensorFlow example: classification of toy Gaussian models

- Carry out a basic classification task separating signal from background

- We will build a single hidden layer neural network with tensor flow and optimise with the Adam gradient descent algorithm


See example: tensorflow/gaussian.ipynb

Define the network architecture

```
In [4]: def model(x):
            with tf.variable_scope("model") as scope:
                w1 = tf.get_variable('w1', shape=(2, 100), dtype=tf.float64,
                        initializer=tf.random_normal_initializer(stddev=0.1))
                b1 = tf.get_variable('b1', shape=(100), dtype=tf.float64,
                        initializer=tf.constant_initializer(0.1))
                w2 = tf.get_variable('w2', shape=(100, 1), dtype=tf.float64,
                        initializer=tf.random_normal_initializer(stddev=0.1))
                b2 = tf.get_variable('b2', shape=(1), dtype=tf.float64,
                        initializer=tf.constant_initializer(0.1))

            l1 = tf.nn.relu(tf.add(b1, tf.matmul(x, w1)))
            logits = tf.add(b2, tf.matmul(l1, w2))
            return logits, tf.sigmoid(logits)

        x = tf.placeholder(tf.float64, shape=[None, 2])
        logits, f = model(x)
```
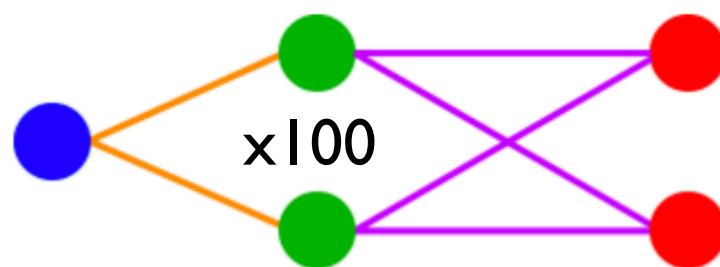
$f_{NN} = \sigma(b_2 + W_2\sigma(b_1 + W_1 x))$
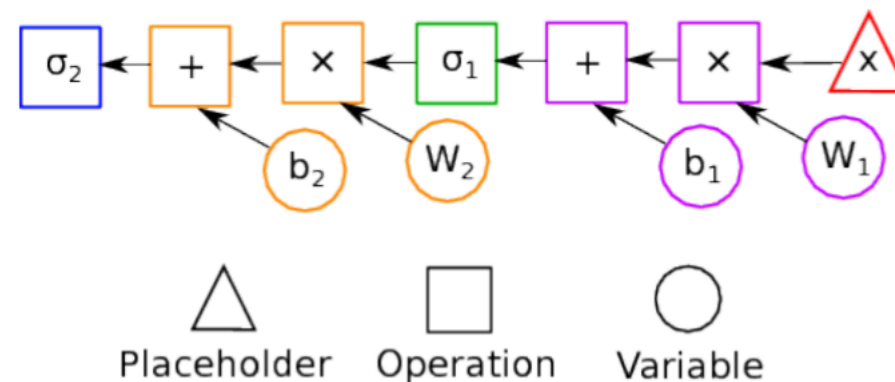
Define the training

- Train using the binary cross entropy calculated on logits

$$C = -\frac{1}{n}\sum_x [y\ln a + (1-y)\ln(1-a)],$$

  - Cross entropy usually best choice for classification*

    - Equivalent to minimising the negative log likelihood

- Output with a sigmoid activation function

  - Sigmoid trained to 1 for signal and 0 for background

- Use the Adam optimiser: gradient descent with momentum and path correction
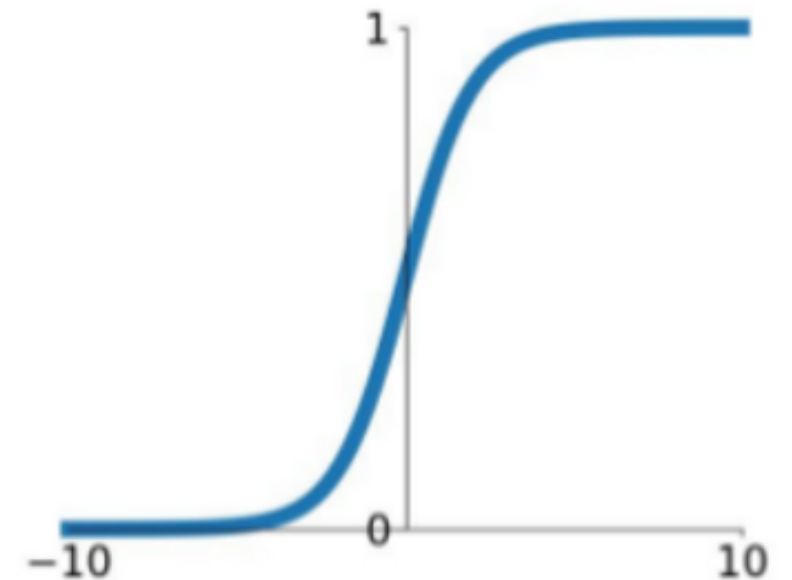
```
In [5]:  labels = tf.placeholder(tf.float64, shape=[None, 1])
         loss = tf.reduce_mean(
             tf.nn.sigmoid_cross_entropy_with_logits(labels=labels, logits=logits))
         minimize_loss = tf.train.AdamOptimizer().minimize(loss)
```

Compile and run the model

- Everything is done within a TensorFlow session

  - Define this session and call a global variable initialisation

- Pass the predefined loss and loss minimisation procedure to the session, broadcasting the input variables

- Additionally calculate the loss on the test data

```
In [6]: sess = tf.Session()
        sess.run(tf.global_variables_initializer())

        loss_train = []
        loss_val = []
        for i_step in range(100):
            loss_, _ = sess.run([loss, minimize_loss],
                                feed_dict={x: data_train, labels: labels_train})
            loss_train.append(loss_)

            loss_ = sess.run(loss, feed_dict={x: data_val, labels: labels_val})
            loss_val.append(loss_)
```

Extra TensorFlow features

- TensorFlow is designed to perform highly-efficient computations and ships many useful features

  - **Data-loading** often bottleneck if not all data fits in memory (very common for image processing!)

  - TensorFlow provides input pipelines directly inbuilt in the graph

  - Full utilisation of CPU/GPU by loading data form disk in queues in memory concurrently

  > See example: tensorflow/queues.ipynb

- '**TensorBoard**' can be used for visualisation of graphs

- '**Eager execution**' recently added to emulate PyTorch style

  - https://www.tensorflow.org/guide/eager

- **Keras** provides an excellent high layer wrapper, making development very convenient

- Many more features not mentioned here…

Further reading

- Stanford course on TensorFlow

    - Very well done and highly entertaining course!

    - Lecturer working in the field (OpenAI, DeepMind, Google, ...)

    - Small Keras part held by Francois Chollet (author of Keras!)

    - Link: https://web.stanford.edu/class/cs20si/syllabus.html

- Free textbook written by Ian Goodfellow, Yoshua Bengio and Aaron Courville:

    - Leading figures in current machine learning research

    - Covers much of what you could want to know

    - Link: http://www.deeplearningbook.org/

Practice with a physics example

- Now you can try to write your own TensorFlow code in a toy physics scenario

- A background vs signal classification task for a stop SUSY model (with a top anti-top standard model background)

m_{stop}= 900 GeV,
m_{LSP} = 100 GeV
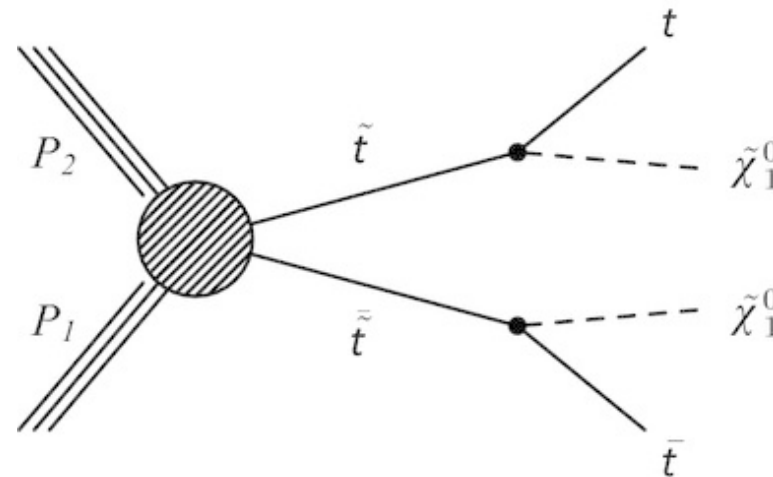
- Try to build a two layer network and train with mini-batch optimisation

- Instructions and skeleton code in tensorflow/physicsExample.ipynb

  - A solution available if you are stuck tensorflow/physicsExample_solution.ipynb

- After this task, have a go at doing a regression task, also in the notebook

Backup

Instructions to get going with AWS

#log in to your server, e.g. (you need to change the server name)
ssh -i A_DESY.pem -S ./tmp -L 1080:127.0.0.1:8888 ubuntu@yourserver.amazonaws.com

if it is not there (it should be) get the repository by uncommenting:
git clone https://github.com/aelwood/iml_tensorflow_keras_workshop.git
cd iml_tensorflow_keras_workshop

#source the software
source activate tensorflow_p27
cd tensorflow

#Run the notebook
jupyter notebook physicsExamples.ipynb --no-browser

#Then take the URL given by the notebook and put it in your browser with 8888 replaced by 1080, e.g.
http://localhost:1080/?token=db64b6e2d5a391dc8c1cdfae697e9de05f7fa0fae8f6fb44&token=db64b6e2d5a391dc8c1cdfae697e9de05f7fa0fae8f6fb44

Instructions with a DESY school machine

#log in to the server, e.g.
ssh schoolXX@naf-school0[1-6].desy.de

get the repository
mkdir NAME
cd NAME
git clone https://github.com/aelwood/iml_tensorflow_keras_workshop.git
cd iml_tensorflow_keras_workshop

#source the software
module load anaconda/2
cd tensorflow

#Run the notebook
jupyter notebook --no-browser #if you need to allow the cookies and quit with 'q'
#Now check the port opened by jupyter NNNN (after localhost:, e.g. 8888)
#open a new tunnel to connect to the notebook (making sure the naf-school N is the same):
ssh -L 1080:127.0.0.1:NNNN schoolXX@naf-school0[1-6].desy.de

#Then take the URL given by the notebook and put it in your browser with NNNN replaced by 1080, e.g.
http://localhost:1080/?token=997de4e92fc210f6777f4508487802b13a30c7d7c232107d