

# Deep Learning on SpiNNaker

Jonas Fassbender

*[jonas@fassbender.dev](mailto:jonas@fassbender.dev)*



THE UNIVERSITY *of* EDINBURGH

Supervisors:

Alan Stokes, Kevin Stratford, Caoimhín Laoide-Kemp

# Agenda



## 1 SpiNNaker

## 2 Prototype

## 3 Problems

## 4 Next Steps

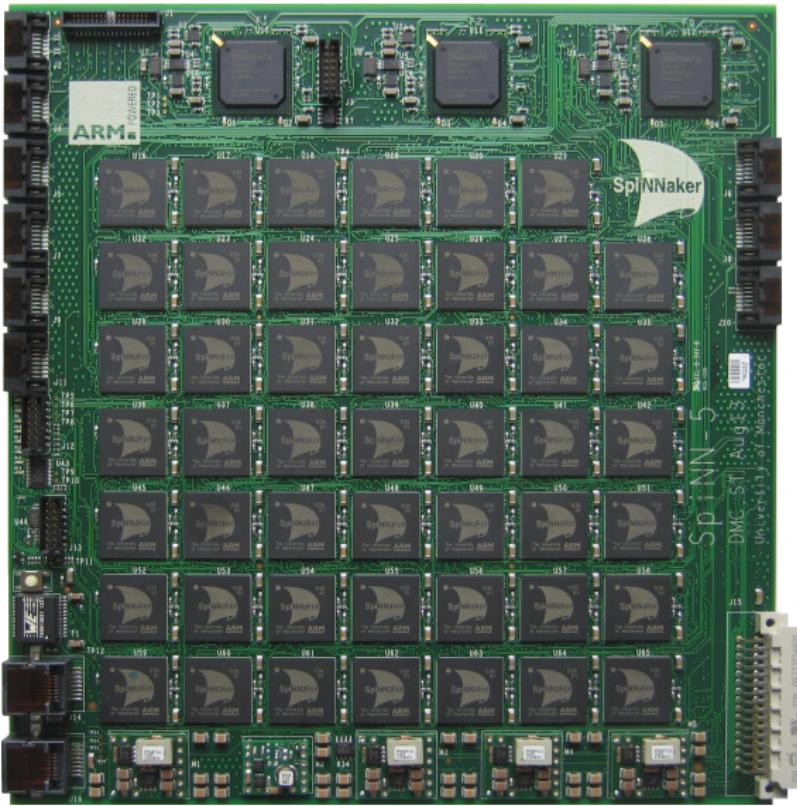
# SpiNNaker

# SpiNNaker



[SpiNNaker is] a platform for high-performance massively parallel [and energy efficient] processing appropriate for the simulation of large-scale [spiking] neural networks [2]

# SpiNNaker



# Prototype

# User API



```
import numpy as np
import tensorflow as tf

# generate random data set
X = np.random.rand(500, 10)

# Keras MLP
kmodel = tf.keras.models.Sequential()
kmodel.add(tf.keras.layers.Dense(
    50, activation="relu",
    input_shape=(10,)))
kmodel.add(tf.keras.layers.Dense(
    50, activation="softmax"))
kmodel.add(tf.keras.layers.Dense(
    300, activation="tanh"))
kmodel.add(tf.keras.layers.Dense(
    50, activation="sigmoid"))
kmodel.add(tf.keras.layers.Dense(25))

# predict labels from random data
# with random weights
p_ = kmodel.predict(X)
```

```
from spiDNN import Model
from spiDNN.layers import Input, Dense

# equivalent model for SpiNNaker
model=Model()
model.add(Input(10))
model.add(Dense(50, activation="relu"))
model.add(Dense(50, activation="softmax"))
model.add(Dense(300, activation="tanh"))
model.add(Dense(50, activation="sigmoid"))
model.add(Dense(25))

# make sure both models have same parameters
model.set_weights(kmodel.get_weights())

p = model.predict(X)

# floating point discrepancies may occur
assert np.amax(np.absolute(p - p_)) < 1e-4
```



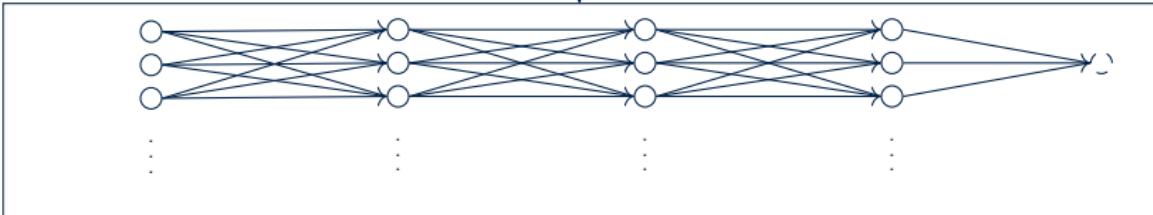
# Prototype Architecture

model



prototype

SpiNNaker toolchain

`.fit, .predict`

machine graph

# Features



## Supported Features

- Dense layers
- 1D convolutional layers
- (Flatten layers)
- Most common activation functions
- Most common loss functions
- SGD optimizer with constant learning rate

# Features



## Supported Features

- Dense layers
- 1D convolutional layers
- (Flatten layers)
- Most common activation functions
- Most common loss functions
- SGD optimizer with constant learning rate

## Missing Features

- 2D convolutional layers
- Pooling layers
- Shortcut connections [1]
- (Batch normalization)

# Problems

# Problems



- TIME!

# Problems



- TIME!
- Dropped packets

# Problems



- TIME!
- Dropped packets
- Convolutional layer as neurons is a complex abstraction

## Next Steps

# Overcoming Dropped Packets in the Future



SpiNNaker APIs we were not able to try:

# Overcoming Dropped Packets in the Future



SpiNNaker APIs we were not able to try:

- Application graph instead of machine graph (less packets sent)

# Overcoming Dropped Packets in the Future



SpiNNaker APIs we were not able to try:

- Application graph instead of machine graph (less packets sent)
- P2P communications instead of multicast packets

# Overcoming Dropped Packets in the Future



SpiNNaker APIs we were not able to try:

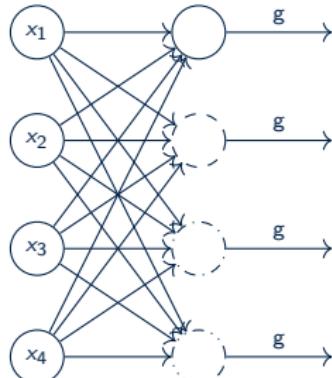
- Application graph instead of machine graph (less packets sent)
- P2P communications instead of multicast packets
- Time-division multiple access (TDMA) system (sends packets at the right time to avoid dropping them)

# Overcoming Dropped Packets in the Future



Better domain decomposition:

$$\left( \begin{array}{|c|c|c|c|} \hline w_{11} & w_{12} & w_{13} & w_{14} \\ \hline w_{21} & w_{22} & w_{23} & w_{24} \\ \hline w_{31} & w_{32} & w_{33} & w_{34} \\ \hline w_{41} & w_{42} & w_{43} & w_{44} \\ \hline \end{array} \right)^T \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$



number of packets sent in each direction:

16  
→

64  
←

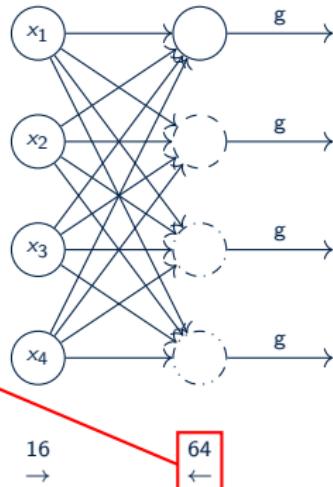
**Figure:** How the computational graph of a dense layer is decomposed into neurons (implemented by the prototype).

# Overcoming Dropped Packets in the Future



Better domain decomposition:

$$\left( \begin{array}{cccc} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{array} \right)^T \left( \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \right) + \left( \begin{array}{c} b_1 \\ b_2 \\ b_3 \\ b_4 \end{array} \right)$$



Number of vertices · number of edges · number of sends per vertex

number of packets sent in each direction:

16  
→

64  
←

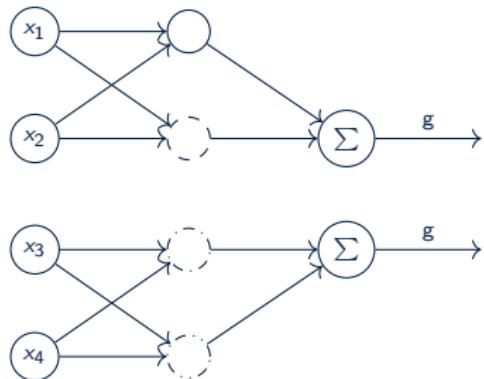
**Figure:** How the computational graph of a dense layer is decomposed into neurons (implemented by the prototype).

# Overcoming Dropped Packets in the Future



Better domain decomposition:

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix}^\top \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$



number of packets sent in each direction:

8 →	32 ← →	8 ←
--------	-----------	--------

**Figure:** How the computational graph of a dense layer could be decomposed in order to reduce the peak and overall number of packets by sparsifying the graph.

# References



- [1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [2] SpiNNaker. SpiNNaker Project. <http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/>, 2020.