Introduction to Deep Learning with Keras

INFIERI19

Lisa BENATO, Patrick L.S. CONNOR, Dirk KRÜCKER, Mareike MEYER, Teddy BEAR
Wuhan, May 2019

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

DESY.

Introduction to Deep Learning with Keras

Outline

00 Introduction (15 mins)

- Machine learning in a nutshell
- Scope of the lab
- Setting up environment

01 Regression (75 mins)

- References
- What is machine learning
- Build a neural network
- Activation function
- TensorFlow & Keras
- Building a network with Keras
- Loss function
- Example
- Optimiser & compilation
- Training & learning curve
- Predicting

Break (5 mins)

02 Classification (75 mins)

- Classification (vs. regression)
- Loss function: *cross-entropy*
- Example
- Validation sample
- Learning curves & accuracy
- Multi-class classification
- Fashion-MNIST
- Methods to prevent over-training

03 Summary & Conclusions (10 mins)

- What we have (not) covered
- Plans for the second lab

04 Back-up

Introduction

Introduction to Deep Learning with Keras

The concept of Machine Learning in a nutshell

Typical problems in science & technology

Modelling, diagnosis/classification, pattern recognition, task automation, ...

Example: Modelling a physics phenomenon

1. Write a model with limited number of parameters
2. Measure data
3. Fit parameters

$$\hat{y} = f(x|p)$$

$$x_i, y_i$$

Examples

- Predict the temperature based on yesterday's weather
- Classify astrophysical objects
- Identifty particles at colliders
- Diagnose diseases
- Speech and face recognition
- ...

Advantages
Better understanding and control.
Parameters may have physical meaning.

Drawbacks
You need to have an idea for a model.
Difficult when many parameters are involved.

Introduction to Deep Learning with Keras

The concept of Machine Learning in a nutshell

Typical problems in science & technology

Modelling, diagnosis/classification, pattern recognition, task automation, ...

Example: Modelling a physics phenomenon

1. *Construct a network with a large number of parameters (or weights)*
2. Measure data
3. Fit parameters

$$\hat{y} = f(x|p)$$

$$x_i, y_i$$

Machine learning means that you don't need to come up with a model: the machine will learn how to interpret the data for you

Supervised learning means that you have a measurement to test your predictions

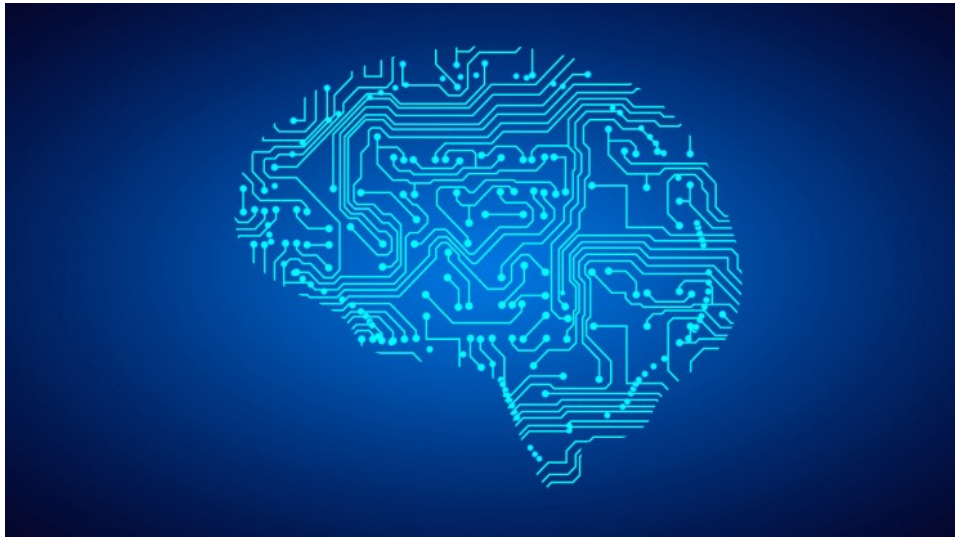
Advantages
Catch effects difficult to model.
Come up without any precise idea.

Drawbacks
Fit is not straightforward.
No direct interpretation.

Introduction to Deep Learning with Keras

Scope of the lab

- (Supervised) Deep learning
- Regression & classification
- Loss function, optimiser, learning curve, learning rate, ...
- TensorFlow & Keras
- ...



We here emphasise on practical aspects of machine learning

Introduction to Deep Learning with Keras

Setting up the environment

The present slides are synchronised with the notebooks

jupyter Intro_DL_with_Keras Last Checkpoint: Last Tuesday at 10:41 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 2

Run Markdown

Introduction to Deep Learning with Keras

This is a dense introduction to Deep Learning!

For an in-depth introduction, there are a several free online books:

- http://neuralnetworksanddeeplearning.com/index.html by Michael Nielsen, a concise introduction.
- http://www.deeplearningbook.org/ Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, provides a detailed introduction to the theoretical background.
- https://torres.ai/first-contact-deep-learning-practical-introduction-keras/ by Jordi Torres, an introduction similar to this tutorial by the use of Keras.
- https://d2l.ai/ by A. Zang et al., another excellent resource but with code examples in Gluon/MXNet (https://gluon.mxnet.io/) instead of Keras.

What is a Neural Network?

What is Machine Learning?

In Machine Learning (ML) we try to adopt(=**learn**) a certain model to some data.

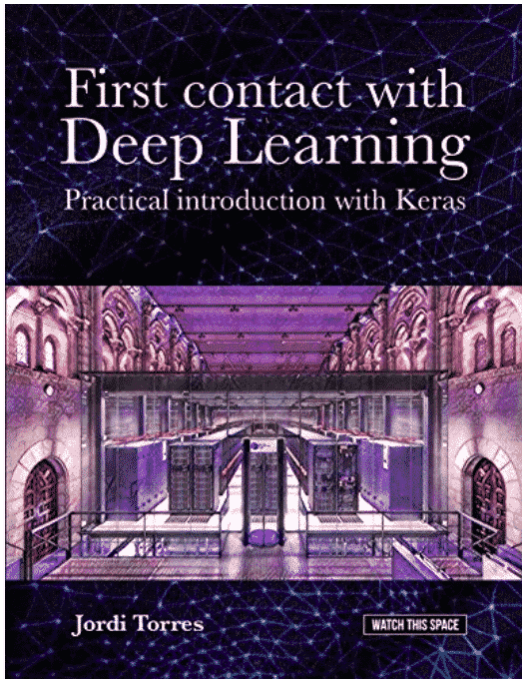
Linear regression is the easiest example. Some data is given and a regression line (predicted value $\hat{y}$) is fitted to the data:

Part One
Regression

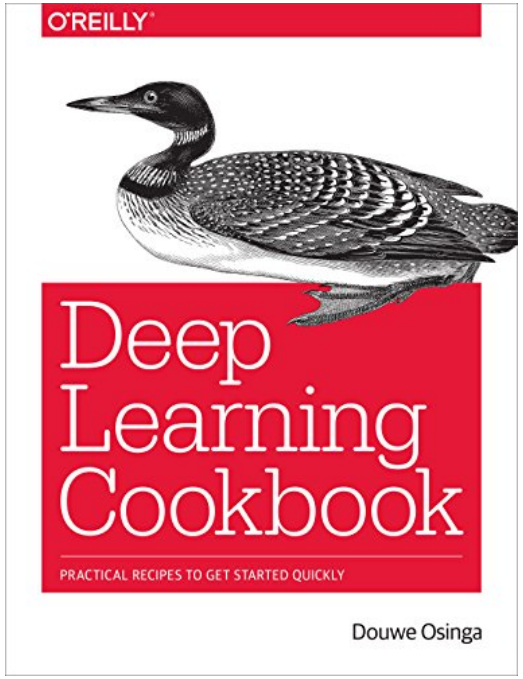
Introduction to Deep Learning with Keras

References

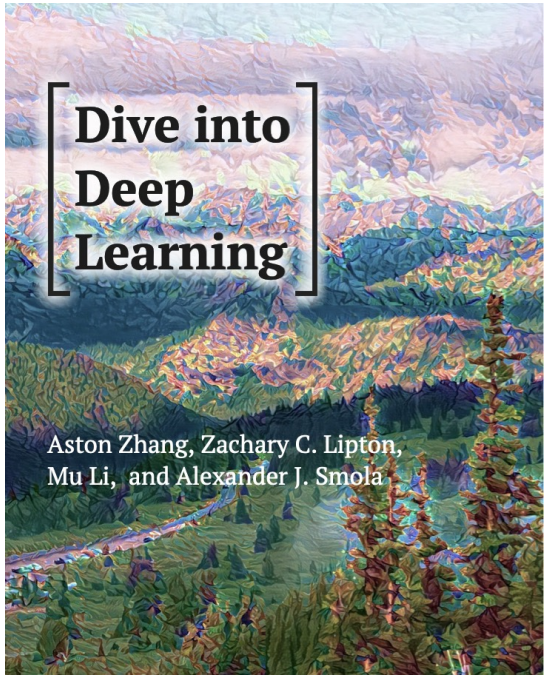
Good for a quick intro



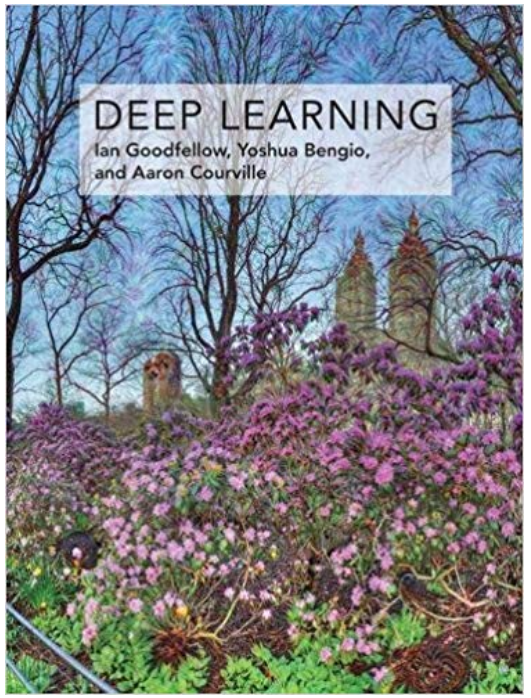
Similar style, but many more details



Amazon reference



***THE* standard reference**



… and many more, also for free & online!

Introduction to Deep Learning with Keras

What is machine learning?

$$f(x) = \sum a_n x^n$$

$$f(x) = \sum \left(A_n \cos nx + B_n \sin nx \right)$$

Machine learning means that you don't need to come up with a model: the machine will learn how to interpret the data for you

Introduction to Deep Learning with Keras

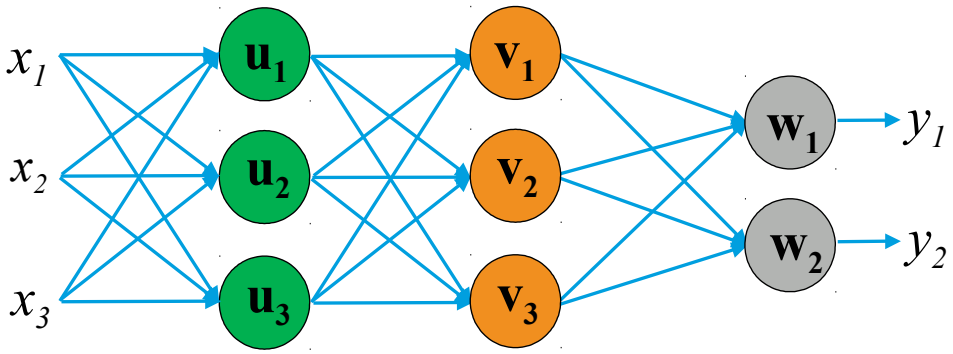
What is machine learning?

Machine learning means that you don't need to come up with a model: the machine will learn how to interpret the data for you

$$f(x) = \sum a_n x^n$$

$$f(x) = \sum \left(A_n \cos nx + B_n \sin nx \right)$$

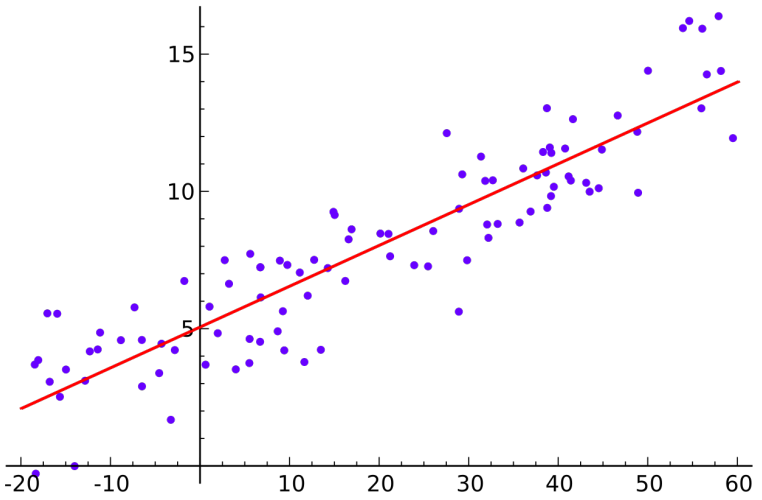
$$f(x) =$$



Neural network = Connected neurons/nodes

Introduction to Deep Learning with Keras

What is machine learning?



Starting point: *Linear Regression*

Data points $x^i, y^i \in \mathbb{R}, i = 1, ..., N_{\text{data}}$

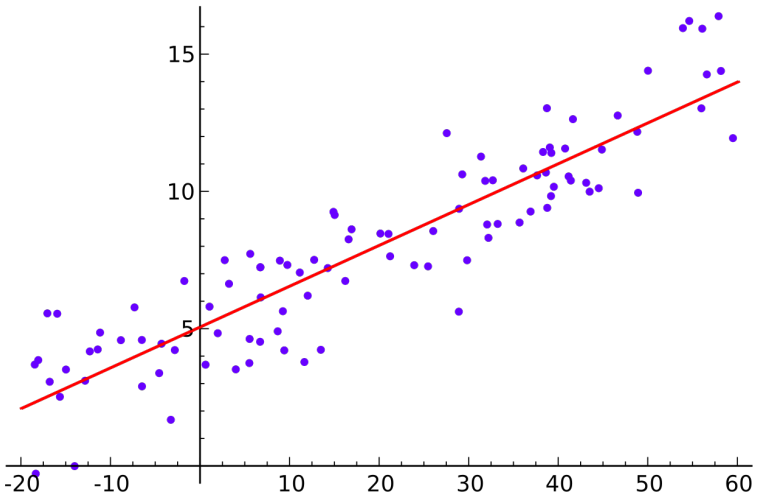
Model $\hat{y} = ax + b$

→ Minimise *loss function* $\min_{a,b} \sum_{i=0}^{N_{\text{data}}} \left(\hat{y}(x_i) - y_i\right)^2$

Introduction to Deep Learning with Keras

What is machine learning?

Starting point: *Linear Regression*



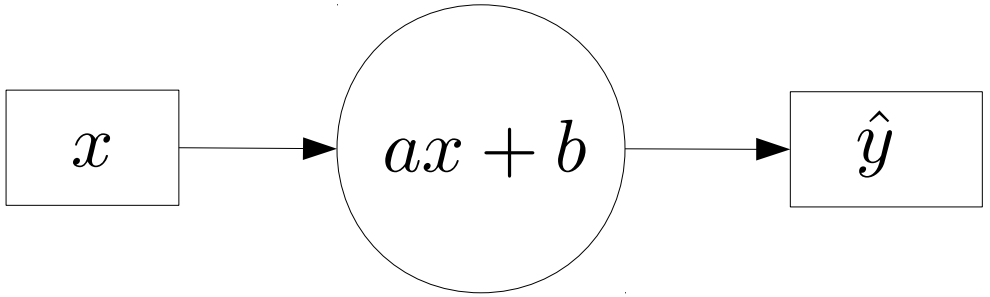
Data points $x^i, y^i \in \mathbb{R}, i = 1, ..., N_{\text{data}}$

Model $\hat{y} = ax + b$

→ Minimise *loss function* $\min_{a,b} \sum_{i=0}^{N_{\text{data}}} \left(\hat{y}(x_i) - y_i\right)^2$

In Machine Learning, we have **many (many) more parameters...**

Our first node/neuron:
(or almost)



Introduction to Deep Learning with Keras

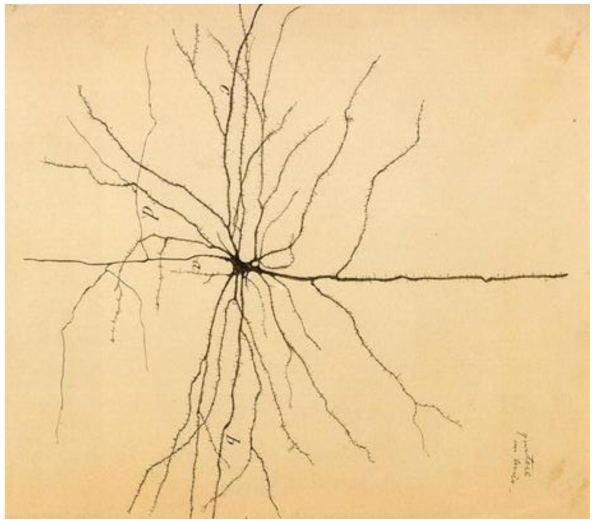
What is machine learning?

Neuron = *Linear Regression* *+ Activation function*

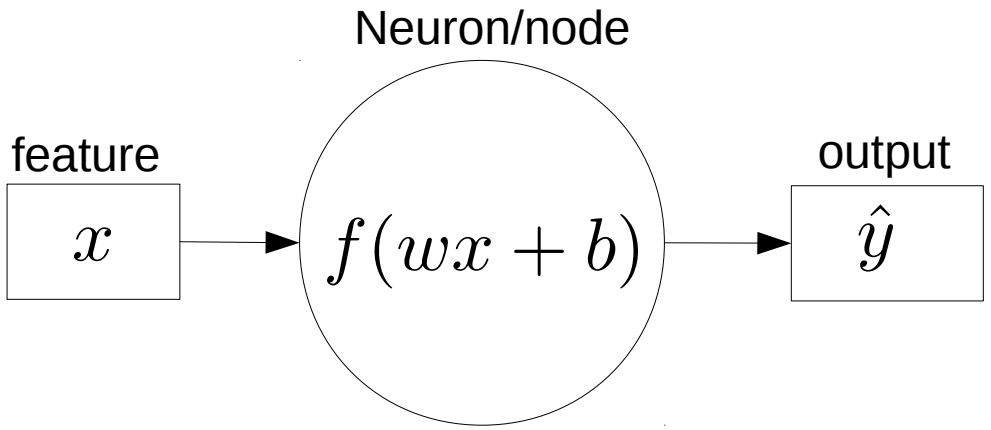
- **Generalisation** from scalar to **vectors** is straightforward
- **Neurons/nodes** are **connected** into a neural network
- The **activation function** introduces a **non-linearity**

Brain = neural network*

*simplistic model for real neuron

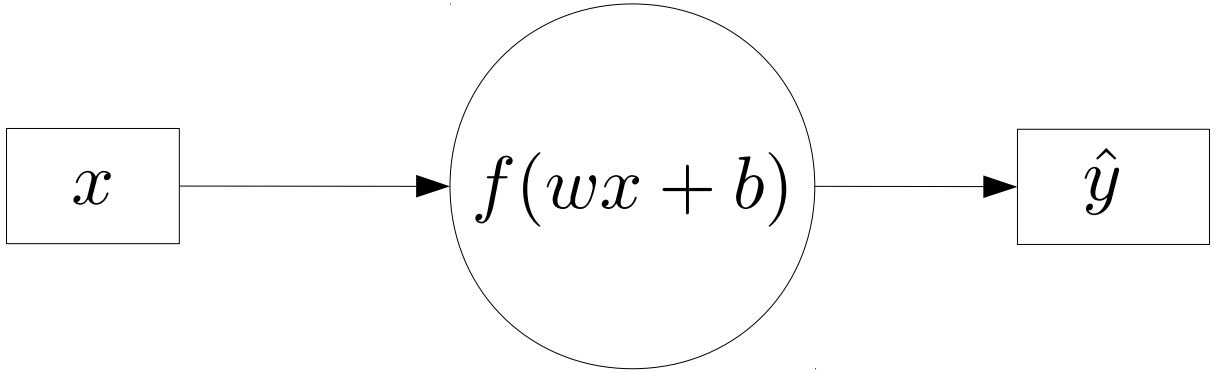


Intelligence arises from the connections



Introduction to Deep Learning with Keras

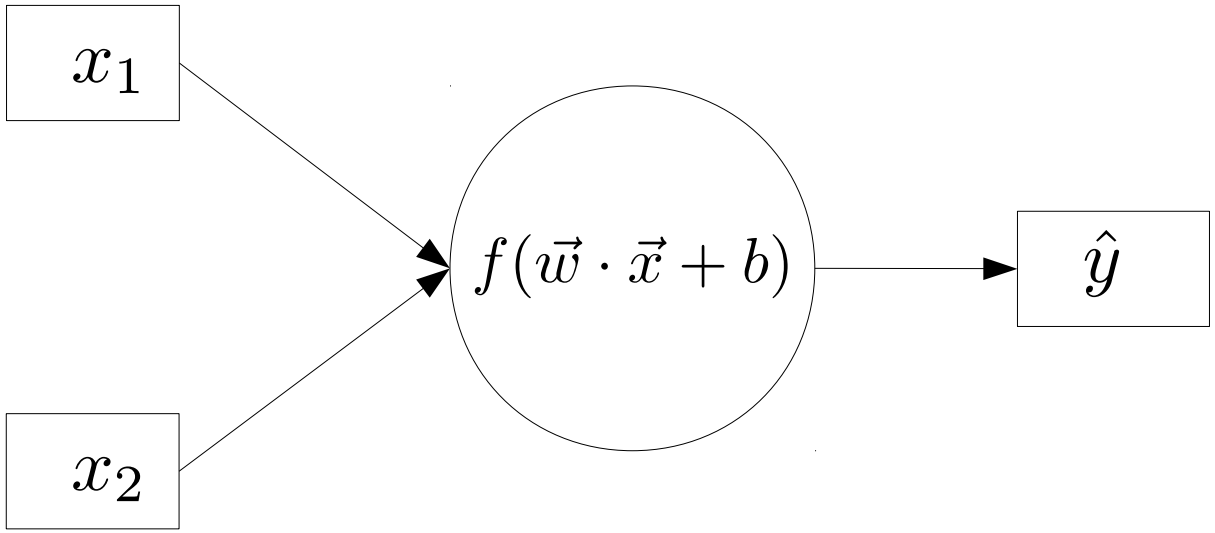
Build a neural network



$$\hat{y} = f(wx + b)$$

Introduction to Deep Learning with Keras

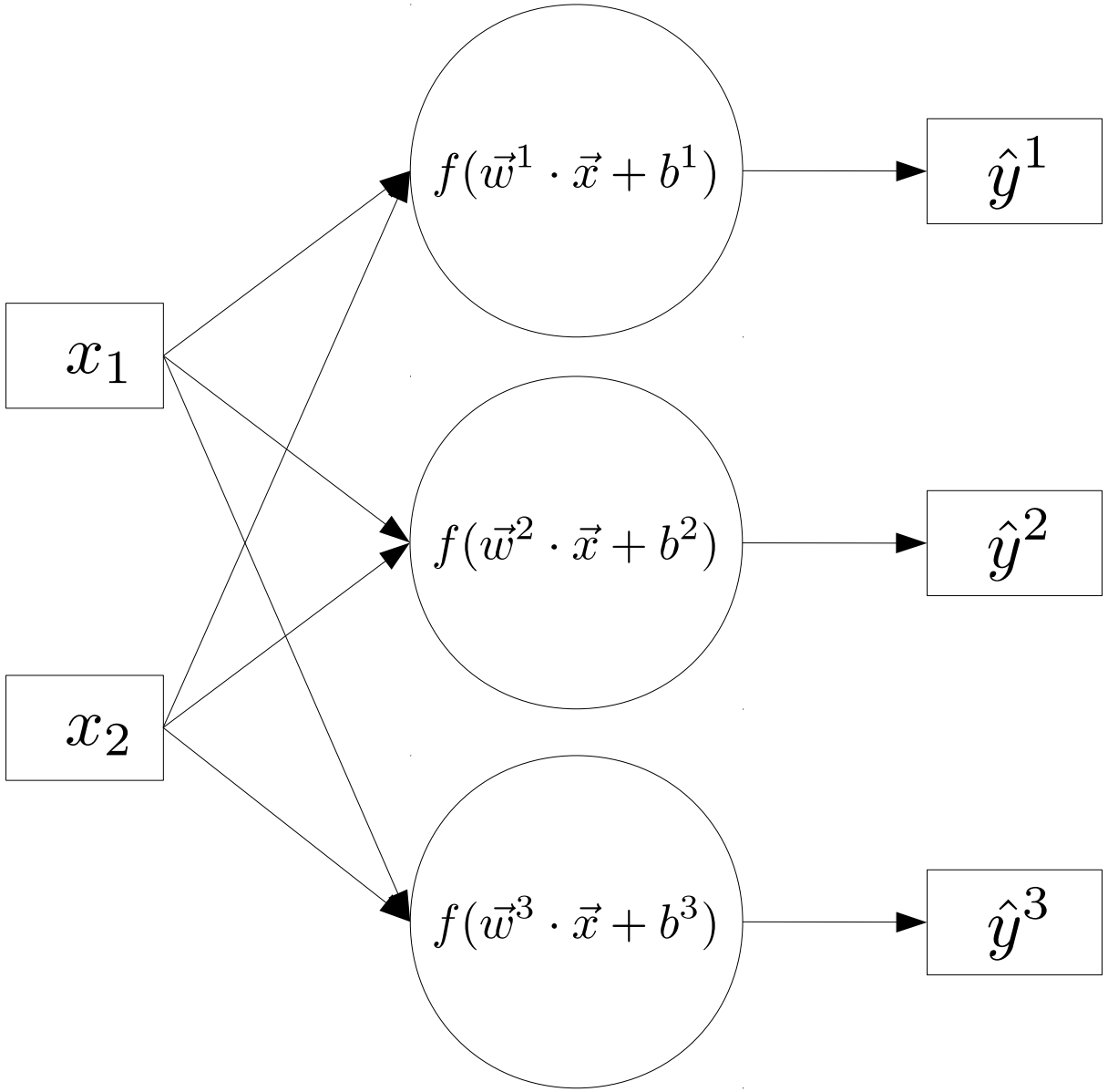
Build a neural network



$$\hat{y} = f\left(\sum_{k=1}^{N} w_k x_k + b\right)$$

Introduction to Deep Learning with Keras

Build a neural network



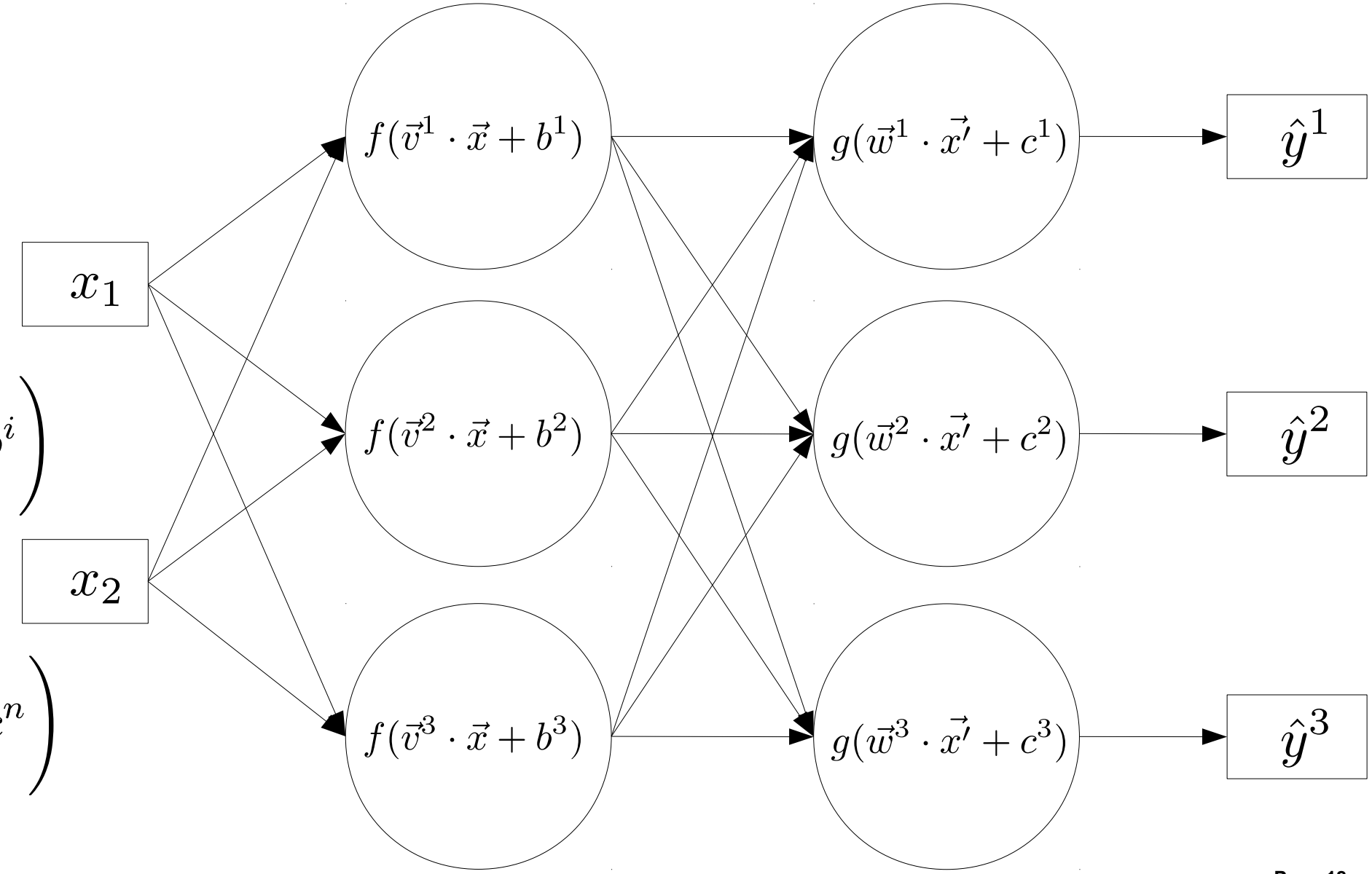
$$\hat{y}^i = f\left(\sum_{k=1}^{N} w_k^i x_k + b^i\right)$$

Introduction to Deep Learning with Keras

Build a neural network

$$x'^i = f\left(\sum_{k=1}^{N} v_k^i x_k + b^i\right)$$

$$\hat{y}^n = g\left(\sum_{i=1}^{M} w_i^n x_i' + c^n\right)$$

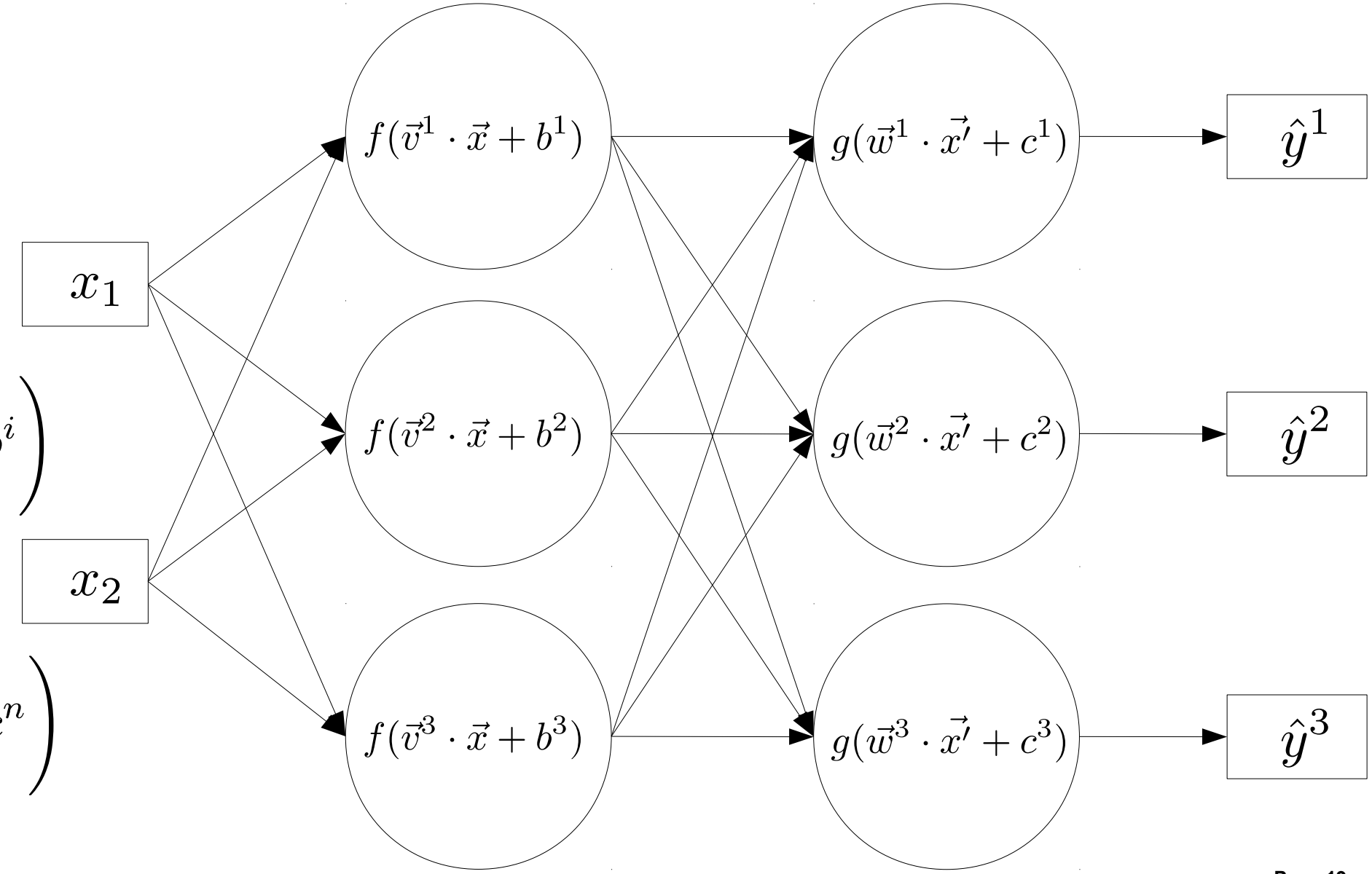


Introduction to Deep Learning with Keras

Build a neural network

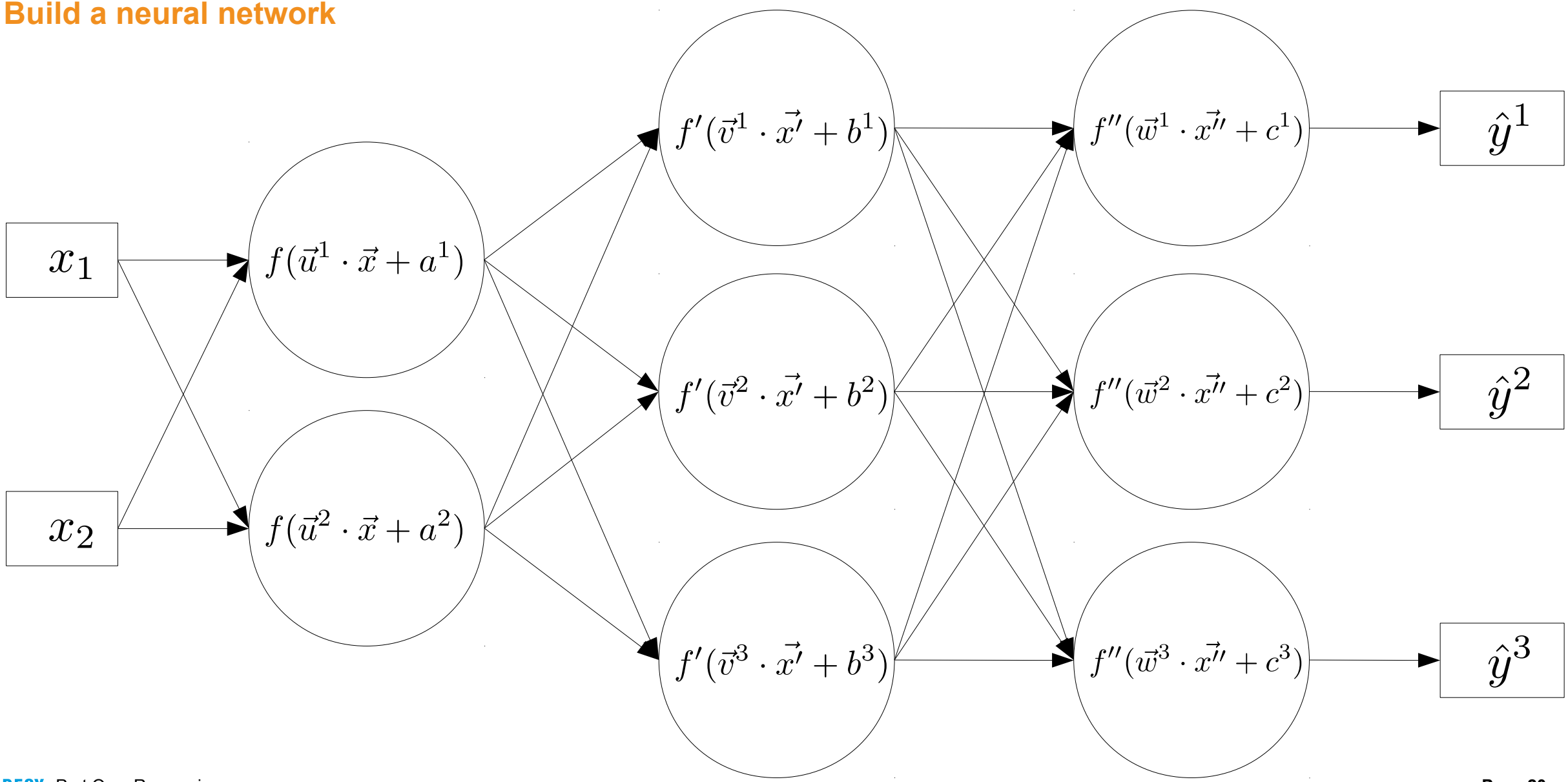
$$x'^i = f\left(\sum_{k=1}^{N} v_k^i x_k + b^i\right)$$

$$\hat{y}^n = g\left(\sum_{i=1}^{M} w_i^n x'_i + c^n\right)$$



Introduction to Deep Learning with Keras

Build a neural network

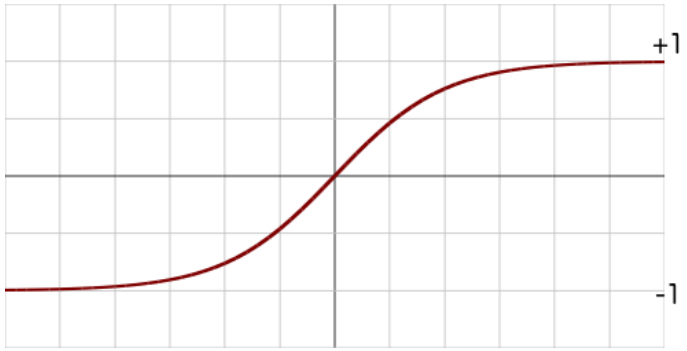


Introduction to Deep Learning with Keras

Activation function

The activation function introduces a *non-linearity* at each layer

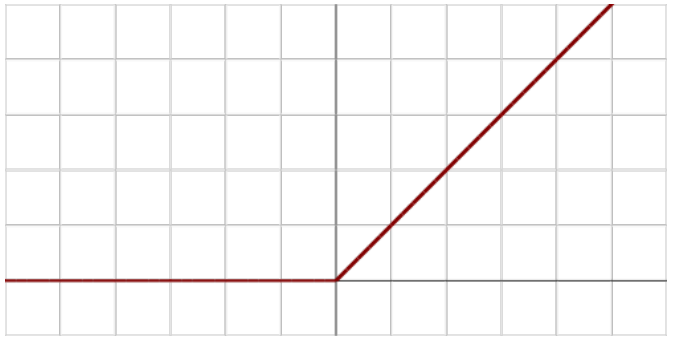
Hyperbolic tangent



$$\hat{y} = \tanh x$$

Used to be popular

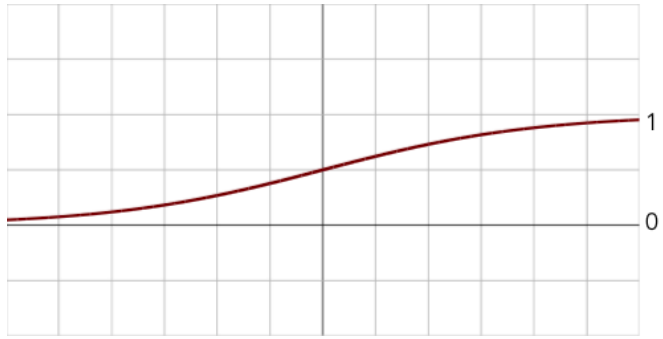
Rectified **L**inear **U**nit (ReLU)



$$\hat{y} = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

Currently most popular for regression

Logistic function



$$\hat{y} = \frac{1}{1 + \exp(-x)}$$

Important in classification (see Part Two)

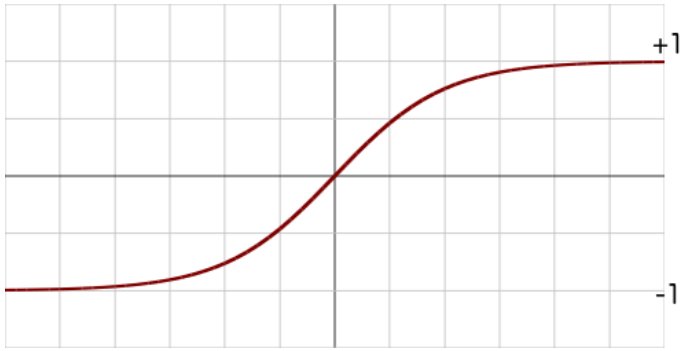
Introduction to Deep Learning with Keras

Activation function

The activation function introduces a *non-linearity* at each layer

Q: what would happen if no activation function is used between two consecutive layers?

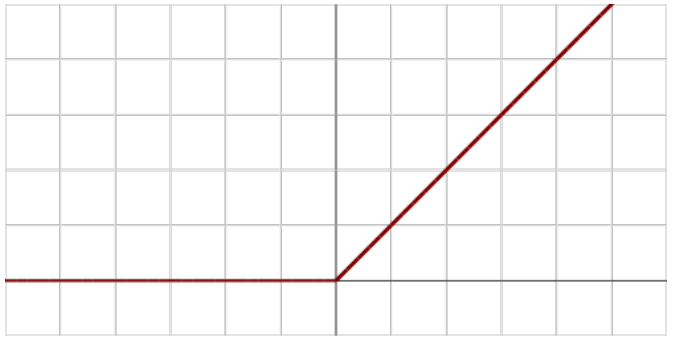
Hyperbolic tangent



$$\hat{y} = \tanh x$$

Used to be popular

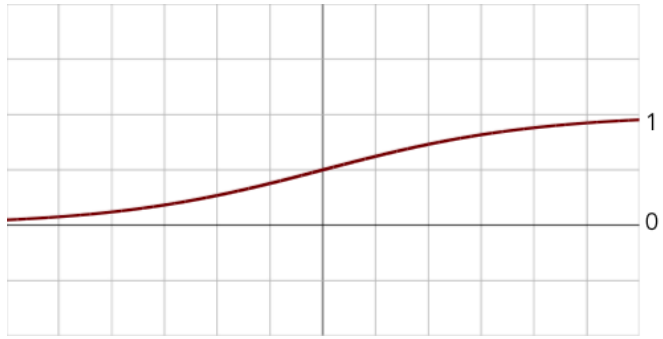
Rectified **L**inear **U**nit (ReLU)



$$\hat{y} = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

Currently most popular for regression

Logistic function



$$\hat{y} = \frac{1}{1 + \exp(-x)}$$

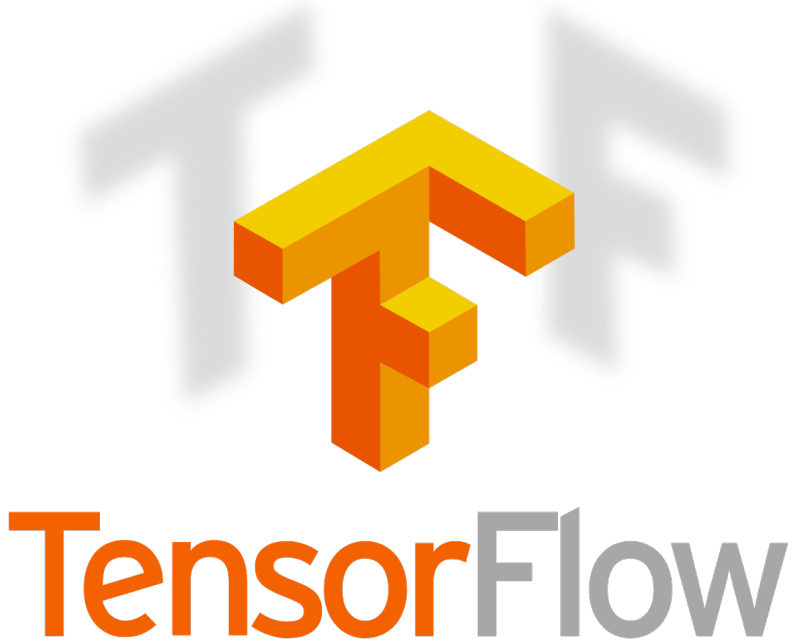
Important in classification (see Part Two)

Introduction to Deep Learning with Keras

TensorFlow & Keras

TensorFlow (or TensorFly)

- One standard library
- Powerful (*i.e.* complicate) language
- Other examples: PyTorch (Facebook), MXNet (Apache), Theano, ...



Keras

- Wrapper for TensorFlow in Python
- Similar language to NumPy
- Originally independent, now also integrated to TensorFlow package

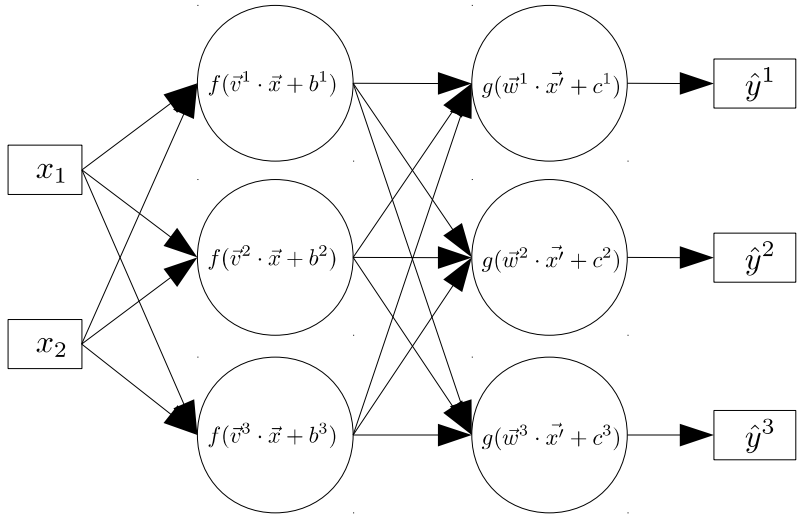
```
# We take the keras implementation from tensorflow
from tensorflow import keras
from tensorflow.keras import layers,models
```

**In this tutorial,
we will use the version of Keras
included in TensorFlow**

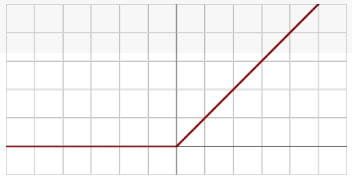
Introduction to Deep Learning with Keras

Building a network with Keras

$$\text{model} = \text{ReLU}\Big[W_2 \cdot \text{ReLU}\Big[W_1 \cdot x + b_1\Big] + b_2\Big]$$



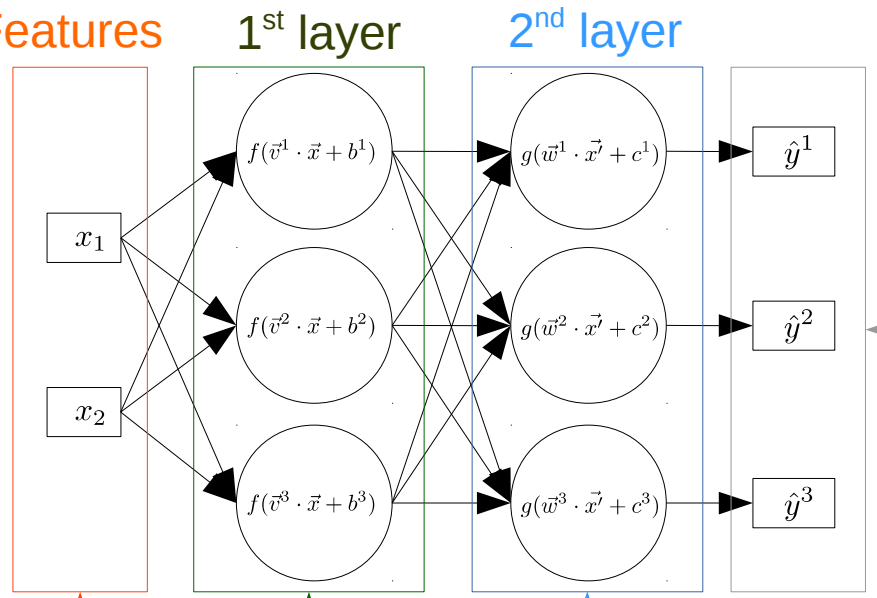

```
model = models.Sequential([
    layers.Dense(20, activation='relu', input_dim=10),
    layers.Dense(30, activation='relu')]
)
```



"*Dense*" means
that all nodes of a layer
are connected
to all nodes of the next layer

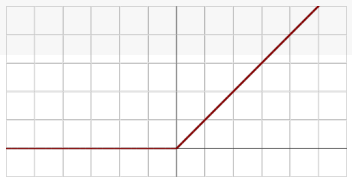
Introduction to Deep Learning with Keras

Building a network with Keras



$$\text{model} = \text{ReLU}\left[W_2 \cdot \text{ReLU}\left[W_1 \cdot x + b_1\right] + b_2\right]$$

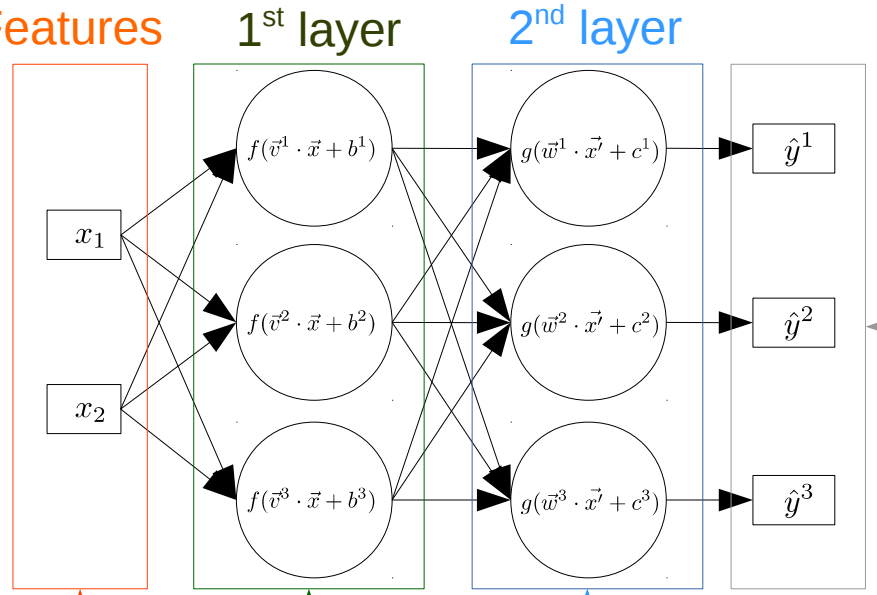
```
model = models.Sequential([
    layers.Dense(20, activation='relu', input_dim=10),
    layers.Dense(30, activation='relu')]
)
```



"*Dense*" means
that all nodes of a layer
are connected
to all nodes of the next layer

Introduction to Deep Learning with Keras

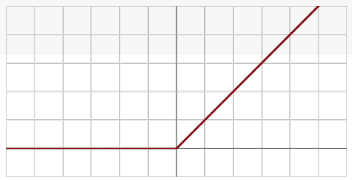
Building a network with Keras



$$\text{model} = \text{ReLU}\left[W_2 \cdot \text{ReLU}\left[W_1 \cdot x + b_1\right] + b_2\right]$$

- 10 features
- 20 nodes in the first layer
- 30 nodes in the second layer

```
model = models.Sequential([
    layers.Dense(20, activation='relu', input_dim=10),
    layers.Dense(30, activation='relu')]
)
```



"*Dense*" means
that all nodes of a layer
are connected
to all nodes of the next layer

Q: how to count the number of parameters?

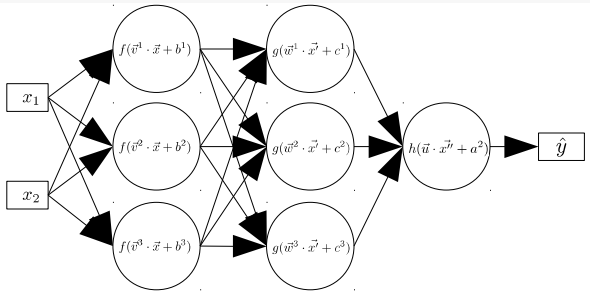
Introduction to Deep Learning with Keras

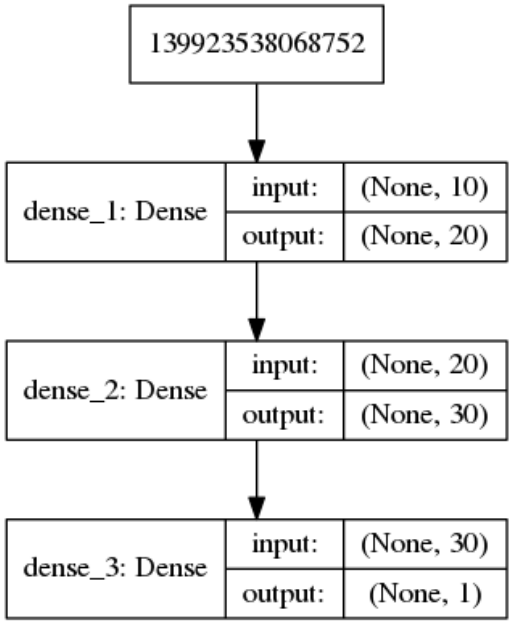
Building a network (cont'd)

$$\text{model} = \text{ReLU}\left[W_2 \cdot \text{ReLU}\left[W_1 \cdot x + b_1\right] + b_2\right]$$

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                220
_________________________________________________________________
dense_2 (Dense)              (None, 30)                630
=================================================================
Total params: 850
Trainable params: 850
Non-trainable params: 0
_________________________________________________________________
```

```
model = models.Sequential([
    layers.Dense(20, activation='relu', input_dim=10),
    layers.Dense(30, activation='relu')]
)
```





Introduction to Deep Learning with Keras

Building a network (cont'd)

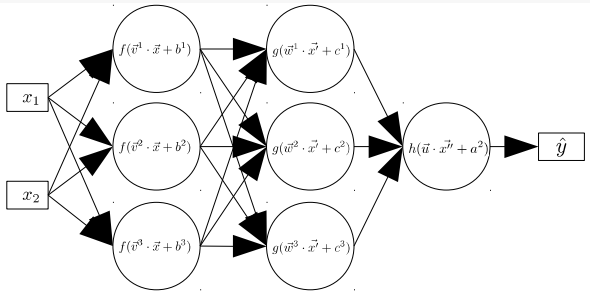
$$\text{model} = \text{ReLU}\left[W_2 \cdot \text{ReLU}\left[W_1 \cdot x + b_1\right] + b_2\right]$$

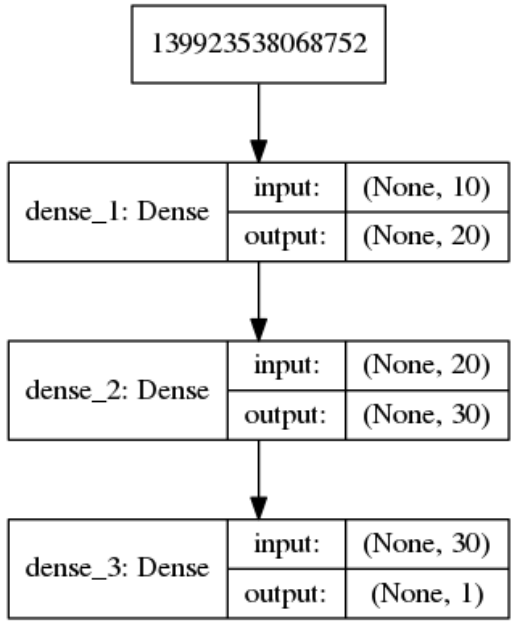
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                220
_________________________________________________________________
dense_2 (Dense)              (None, 30)                630
=================================================================
Total params: 850
Trainable params: 850
Non-trainable params: 0
_________________________________________________________________
```

$(10 \text{ features} + 1 \text{ bias}) \times 20 \text{ nodes}$

$(20 \text{ features} + 1 \text{ bias}) \times 30 \text{ nodes}$

```
model = models.Sequential([
    layers.Dense(20, activation='relu', input_dim=10),
    layers.Dense(30, activation='relu')]
)
```





Introduction to Deep Learning with Keras

Building a network (cont'd)

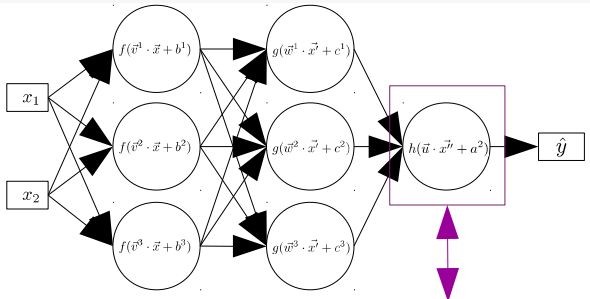
$$\text{model} = \text{ReLU}\left[W_2 \cdot \text{ReLU}\left[W_1 \cdot x + b_1\right] + b_2\right]$$

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                220
_________________________________________________________________
dense_2 (Dense)              (None, 30)                630
=================================================================
Total params: 850
Trainable params: 850
Non-trainable params: 0
_________________________________________________________________
```

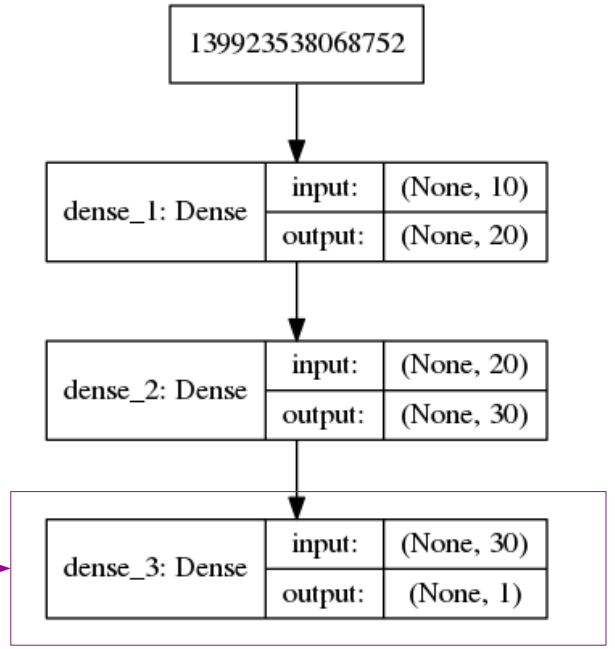
$(10 \text{ features} + 1 \text{ bias}) \times 20 \text{ nodes}$

$(20 \text{ features} + 1 \text{ bias}) \times 30 \text{ nodes}$

```
model = models.Sequential([
    layers.Dense(20, activation='relu', input_dim=10),
    layers.Dense(30, activation='relu')]
)
```




```
model.add(layers.Dense(1))
```



Introduction to Deep Learning with Keras

Building a network (cont'd)

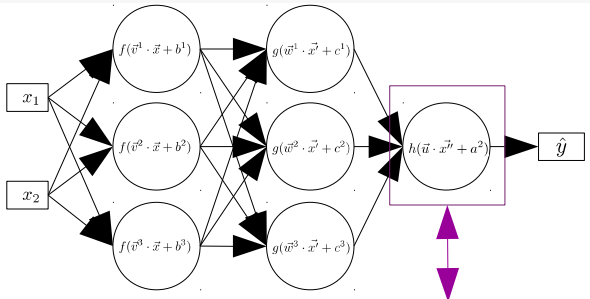
$$\text{model} = \text{ReLU}\left[W_2 \cdot \text{ReLU}\left[W_1 \cdot x + b_1\right] + b_2\right]$$

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 20)                220
_________________________________________________________________
dense_2 (Dense)              (None, 30)                630
=================================================================
Total params: 850
Trainable params: 850
Non-trainable params: 0
_________________________________________________________________
```

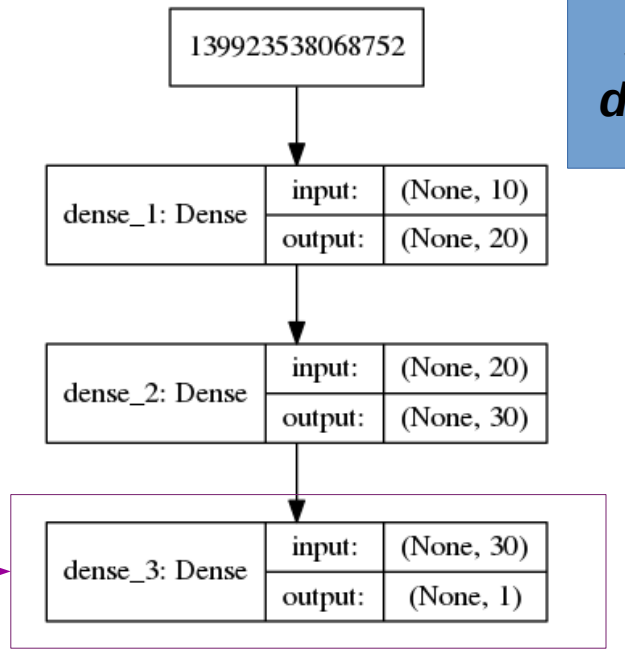
$(10 \text{ features} + 1 \text{ bias}) \times 20 \text{ nodes}$

$(20 \text{ features} + 1 \text{ bias}) \times 30 \text{ nodes}$

```
model = models.Sequential([
    layers.Dense(20, activation='relu', input_dim=10),
    layers.Dense(30, activation='relu')]
)
```




```
model.add(layers.Dense(1))
```



So far, we have just *defined* the network...

Q: how can we fit the parameters?

Introduction to Deep Learning with Keras

Loss function

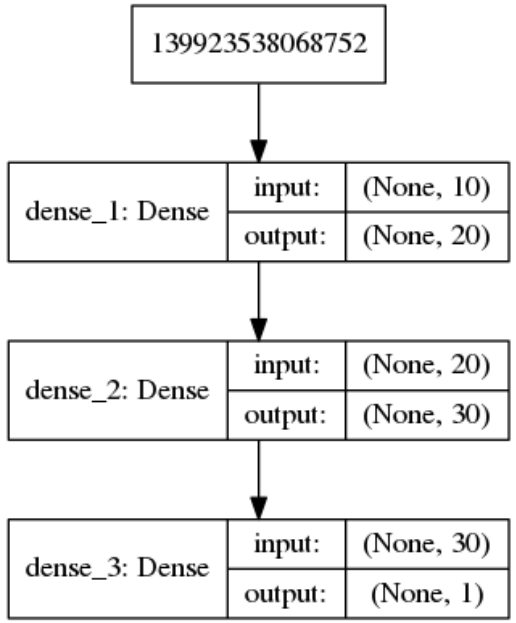
Reminder:
we are doing
supervised learning

Compare output of the model
with real (measured) values
and **minimise loss function**

Example: mean-square error (MSE)

$$\min_{w_i} \sum_{i=0}^{N_{\text{data}}} \left(\hat{y}(x_i) - y_i\right)^2$$

$$\text{model} = \text{ReLU}\Big[W_2 \cdot \text{ReLU}\Big[W_1 \cdot x + b_1\Big] + b_2\Big]$$



Introduction to Deep Learning with Keras

Loss function

Reminder:
we are doing
supervised learning

↓

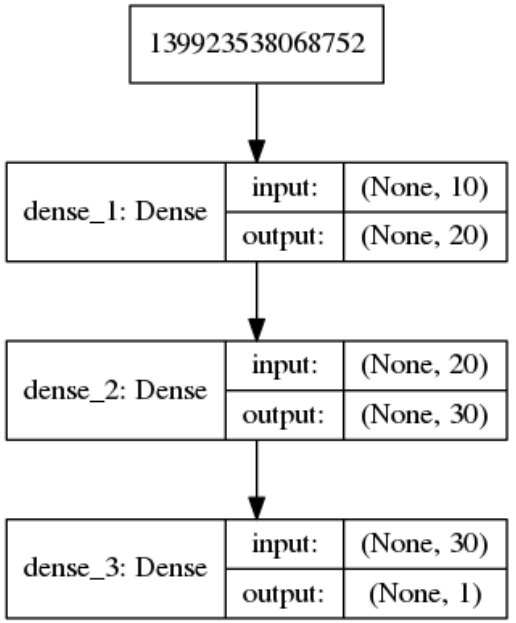
Compare output of the model
with real (measured) values
and **minimise loss function**

Example: mean-square error (MSE)

$$\min_{w_i} \sum_{i=0}^{N_{\text{data}}} \left(\hat{y}(x_i) - y_i\right)^2$$

$$\text{model} = \text{ReLU}\left[W_2 \cdot \text{ReLU}\left[W_1 \cdot x + b_1\right] + b_2\right]$$

- ✓ We can **construct a network**
- ✓ We know in principle **how to fit the weights**
- ✗ We need **some data points** (next slide)
- ✗ We need a strategy to fit this **many weights**



Introduction to Deep Learning with Keras

Just a toy!

Example

Generate randomly 10k events of 10-tuples, such that $y = \sin \sum_{k=1}^{10} x_k^2$

$x_k \in [0, 1)$

```
import numpy as np
# We create 10000 random vectors each 10-dim
N_samples=10000
N_in=10
# A matrix N_samplesxN_in, uniform in [0,1)
x_train=np.random.rand(N_samples,N_in)
# Sum of squares along N_in
z = np.sum( np.square(x_train),axis=1)
y_train = np.sin(z)
```

Introduction to Deep Learning with Keras

Example

Generate randomly 10k events of 10-tuples, such that

$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1)$$

```
import numpy as np
# We create 10000 random vectors each 10-dim
N_samples=10000
N_in=10
# A matrix N_samplesxN_in, uniform in [0,1)
x_train=np.random.rand(N_samples,N_in)
# Sum of squares along N_in
z = np.sum( np.square(x_train),axis=1)
y_train = np.sin(z)
```

Introduction to Deep Learning with Keras

Example

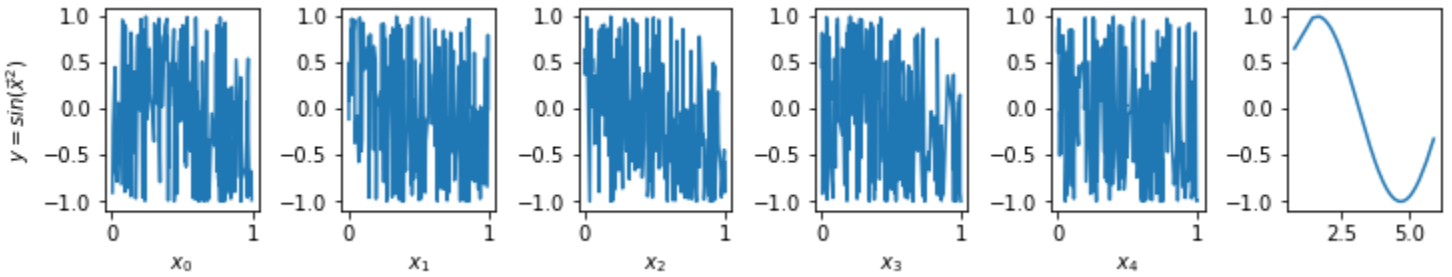
Generate randomly 10k events of 10-tuples, such that

$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1)$$

```
import numpy as np
# We create 10000 random vectors each 10-dim
N_samples=10000
N_in=10
# A matrix N_samplesxN_in, uniform in [0,1)
x_train=np.random.rand(N_samples,N_in)
# Sum of squares along N_in
z = np.sum( np.square(x_train),axis=1)
y_train = np.sin(z)
```

Looks quite random: **hidden structure**



Introduction to Deep Learning with Keras

Example

Generate randomly 10k events of 10-tuples, such that

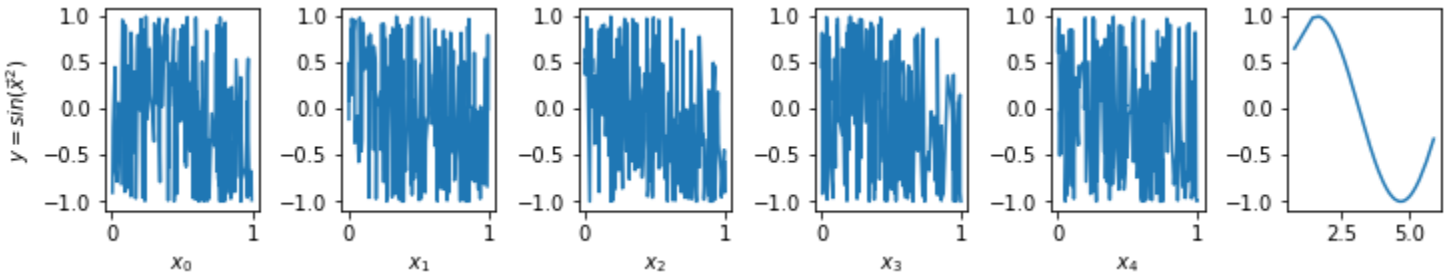
$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1)$$

```
import numpy as np
# We create 10000 random vectors each 10-dim
N_samples=10000
N_in=10
# A matrix N_samplesxN_in, uniform in [0,1)
x_train=np.random.rand(N_samples,N_in)
# Sum of squares along N_in
z = np.sum( np.square(x_train),axis=1)
y_train = np.sin(z)
```

Now we need to define a strategy to fit a large amount of parameters

Looks quite random: **hidden structure**



Introduction to Deep Learning with Keras

Network optimiser & compilation

```
from tensorflow.keras import models, layers, losses, optimizers
from time import time
model = models.Sequential(
    [
        layers.Dense(20, activation='relu', input_dim=10),
        layers.Dense(30, activation='relu'),
        layers.Dense(1)
    ]
)
optimizer = optimizers.SGD(lr=0.01)

model.compile(optimizer=optimizer,loss='mse')
```

$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1]$$

Introduction to Deep Learning with Keras

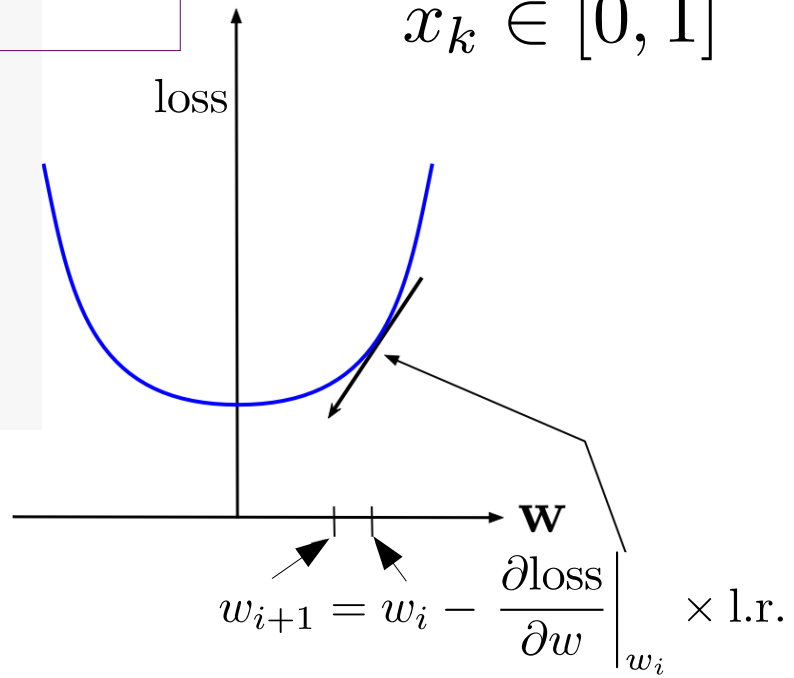
Network optimiser & compilation

```
from tensorflow.keras import models, layers, losses, optimizers
from time import time
model = models.Sequential(
    [
        layers.Dense(20, activation='relu', input_dim=10),
        layers.Dense(30, activation='relu'),
        layers.Dense(1)
    ]
)
optimizer = optimizers.SGD(lr=0.01)

model.compile(optimizer=optimizer,loss='mse')
```

Optimiser:
Stochastic
Gradient
Descent

$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1]$$



- The *minimisation* of so many parameters is performed *iteratively* ("**descent**")
- Use the **gradient** of the loss function w.r.t. NN parameters
- **Stochastic** means that only a subsample of the data is used at each iteration

Introduction to Deep Learning with Keras

Network optimiser & compilation

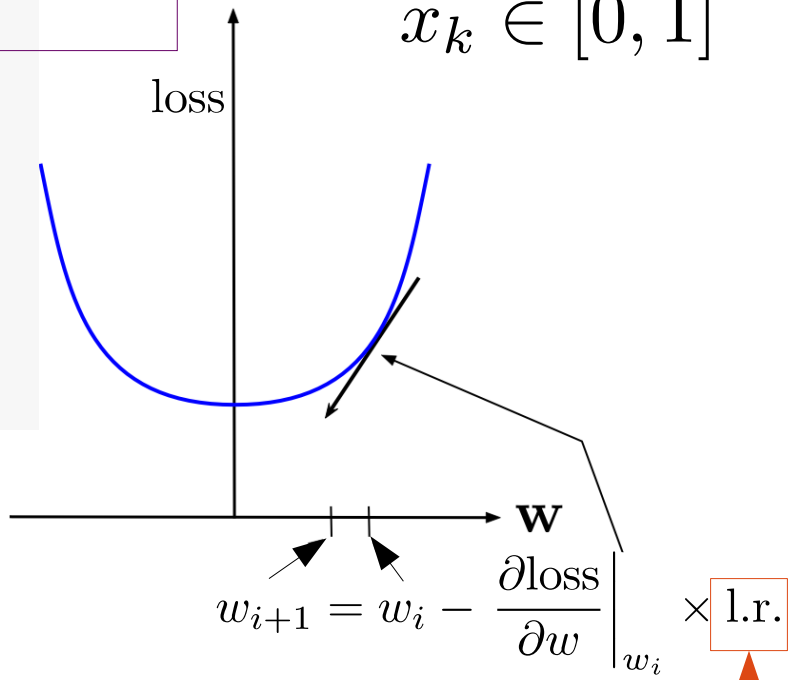
```
from tensorflow.keras import models, layers, losses, optimizers
from time import time
model = models.Sequential(
    [
        layers.Dense(20, activation='relu', input_dim=10),
        layers.Dense(30, activation='relu'),
        layers.Dense(1)
    ]
)
optimizer = optimizers.SGD(lr=0.01)

model.compile(optimizer=optimizer,loss='mse')
```

Optimiser:
Stochastic
Gradient
Descent

Learning rate

$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1]$$



- The *minimisation* of so many parameters is performed *iteratively* ("**descent**")
- Use the **gradient** of the loss function w.r.t. NN parameters
- **Stochastic** means that only a subsample of the data is used at each iteration

Introduction to Deep Learning with Keras

Network optimiser & compilation

```
from tensorflow.keras import models, layers, losses, optimizers
from time import time
model = models.Sequential(
    [
        layers.Dense(20, activation='relu', input_dim=10),
        layers.Dense(30, activation='relu'),
        layers.Dense(1)
    ]
)
optimizer = optimizers.SGD(lr=0.01)

model.compile(optimizer=optimizer,loss='mse')
```

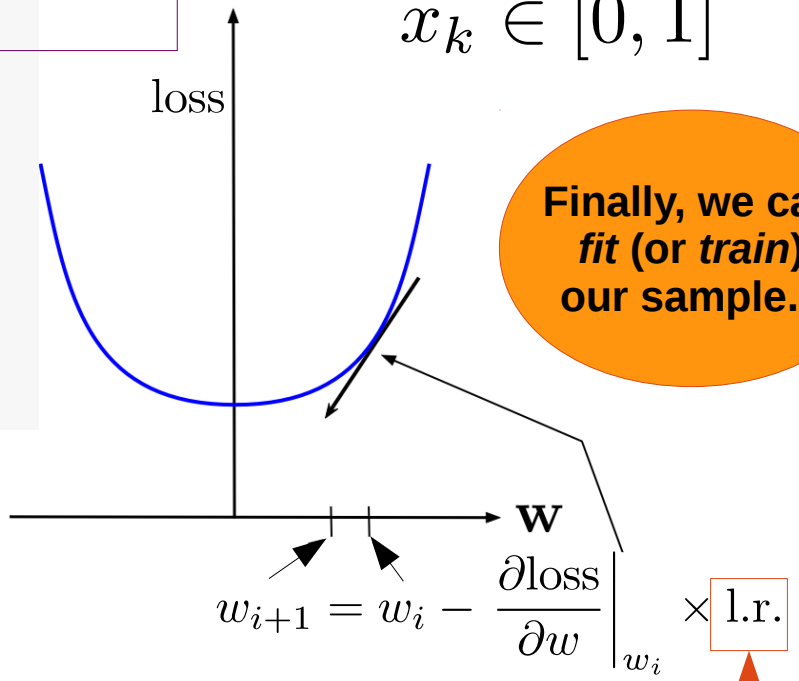
Optimiser: **S**tochastic **G**radient **D**escent

Compile the network before fitting anything

Learning rate

$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1]$$



Finally, we can *fit* (or *train*) our sample...

- The *minimisation* of so many parameters is performed *iteratively* ("**descent**")
- Use the **gradient** of the loss function w.r.t. NN parameters
- **Stochastic** means that only a subsample of the data is used at each iteration

Introduction to Deep Learning with Keras

Training & Learning curve

$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1)$$

```
histObj = model.fit(x_train, y_train, batch_size=32, epochs=10)
```

Batch size

Size of the sample used by the optimiser

Number of epochs

Number of times that the whole sample will be used

Monitor the training

Learning rate

Internal parameter

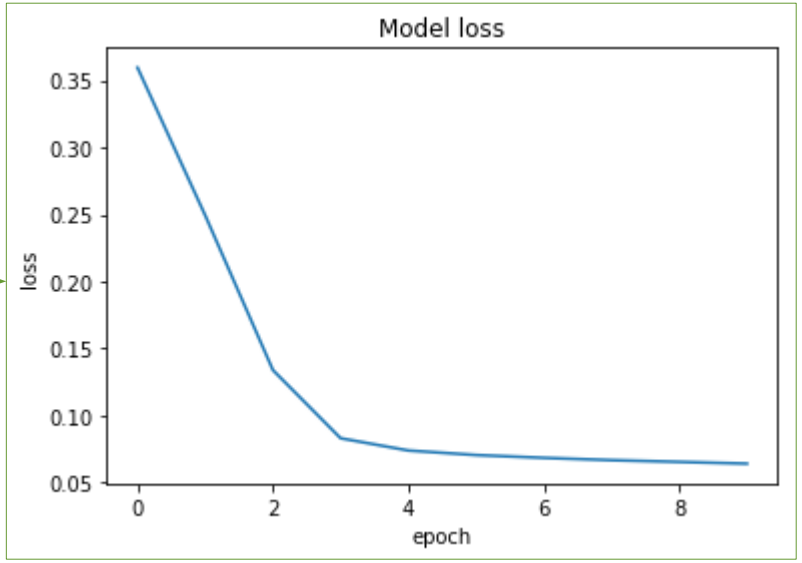
- Too small = slow convergence
- Too high = may diverge

```
optimizer = optimizers.SGD(lr=0.01)
```

Learning Curve

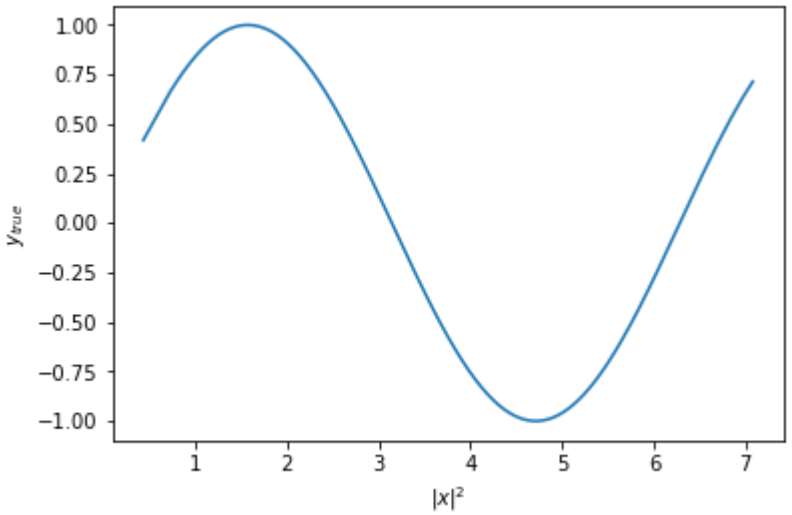
Loss value along epochs

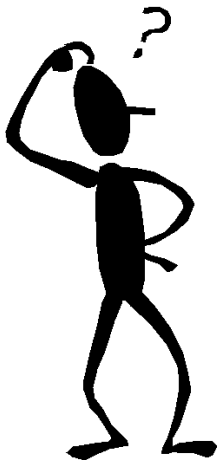
(see also Part Two!)

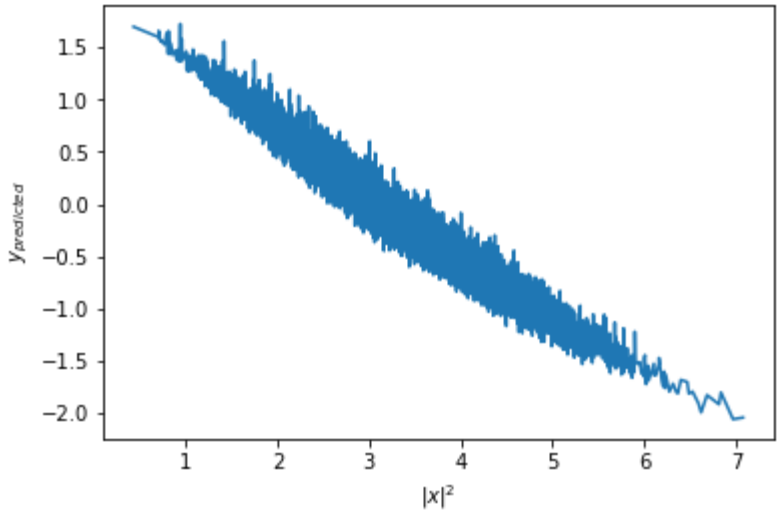


Introduction to Deep Learning with Keras

Predicting







$$y = \sin \sum_{k=1}^{10} x_k^2$$

$$x_k \in [0, 1)$$

Deep Neural Network

- Number of layers
- Number of nodes for each layer
- Activation function(s)

Training

- Sample size
- Batch size
- Optimiser
- Number of epochs

Play with the parameters...

Try to use the Adam optimiser...

Break!

Part Two
Classification

Introduction to Deep Learning with Keras

Classification (vs. regression)

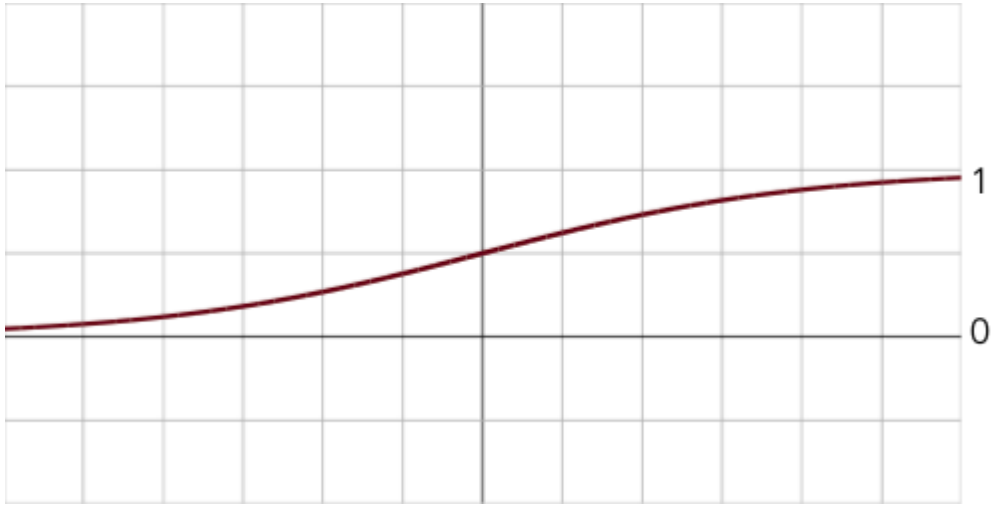
1. Discrete class labels $y_k = 0, 1$
2. Probabilist prediction $\hat{y} \in [0, 1]$

Activation function
For last node/layer:
Logistic function

$$f(x) = \frac{1}{1 + \exp(-x)}$$

Binary* classification:

Last layer must be made of a single node with output between 0 and 1



* we discuss later on multi-classification

Introduction to Deep Learning with Keras

Classification (vs. regression)

1. Discrete class labels $y_k = 0, 1$
2. Probabilist prediction $\hat{y} \in [0, 1]$

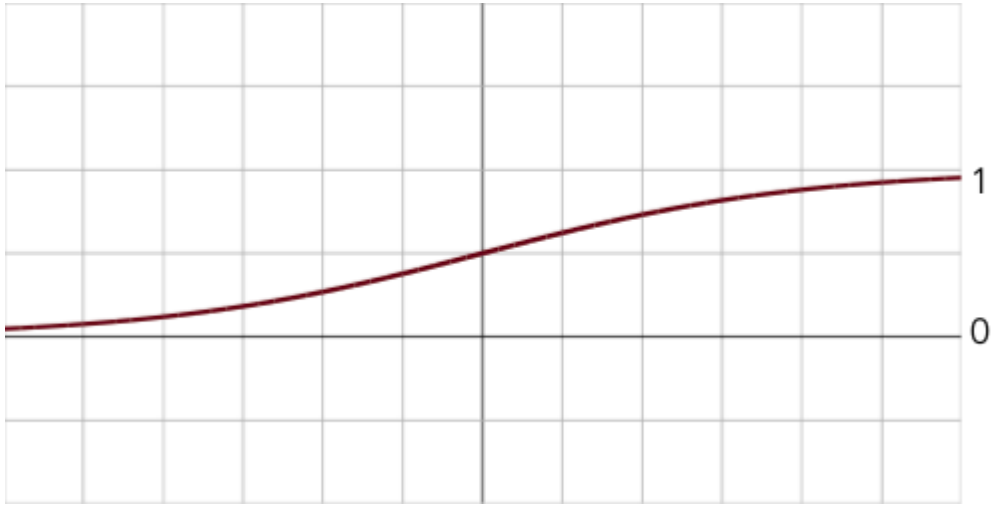
Activation function
For last node/layer:
Logistic function

$$f(x) = \frac{1}{1 + \exp(-x)}$$

We need to adapt the network w.r.t. regression

Binary* classification:

Last layer must be made of a single node with output between 0 and 1



* we discuss later on multi-classification

Introduction to Deep Learning with Keras

Loss function: *cross-entropy*

1. Discrete class labels $y_k = 0, 1$
2. Probabilist prediction $\hat{y} \in [0, 1]$

Bernoulli trial
→ two possible outcomes $p, q \,|\, p + q = 1$

$$\mathcal{L} = \prod_{k \text{ in class } 1}^{n_{\text{batch}}} \hat{y}(x_k) \cdot \prod_{k \text{ in class } 0}^{n_{\text{batch}}} \left(1 - \hat{y}(x_k)\right)$$

→ *Cross entropy* is found from this likelihood

Jacob Bernoulli 1654-1705

Reminder

The batch corresponds to the data subsample used at each iteration to minimise the loss function

Introduction to Deep Learning with Keras

Loss function: *cross-entropy*

Jacob Bernoulli 1654-1705

1. Discrete class labels $y_k = 0, 1$
2. Probabilist prediction $\hat{y} \in [0, 1]$

Bernoulli trial
→ two possible outcomes $p, q \,|\, p + q = 1$

Reminder

The batch corresponds to the data subsample used at each iteration to minimise the loss function

$$\mathcal{L} = \prod_{k}^{n_{\text{batch}}} \left(y_k \cdot \hat{y}(x_k) + (1 - y_k) \cdot (1 - \hat{y}(x_k)) \right)$$

normalisation

Cross-entropy

$$\rightarrow -\frac{1}{n_{\text{batch}}} \log \mathcal{L} = -\frac{1}{n_{\text{batch}}} \sum_{k=1}^{n_{\text{batch}}} \left(y_k \log \left(\hat{y}(x_k) \right) + (1 - y_k) \log \left(1 - \hat{y}(x_k) \right) \right)$$

Introduction to Deep Learning with Keras

Loss function: *cross-entropy*

$$\text{entropy} = \int p(x) \log\big(p(x)\big) dx$$

$$\text{cross-entropy} = \int q(x) \log\big(p(x)\big) dx$$

$$-\frac{1}{n_{\text{batch}}} \log \mathcal{L} = -\frac{1}{n_{\text{batch}}} \sum_{k=1}^{n_{\text{batch}}} \Big(y_k \log\big(\hat{y}(x_k)\big) + \big(1 - y_k\big) \log\big(1 - \hat{y}(x_k)\big) \Big)$$

Introduction to Deep Learning with Keras

Loss function: *cross-entropy*

$$\text{entropy} = \int p(x) \log\big(p(x)\big) dx$$

$$\text{cross-entropy} = \int q(x) \log\big(p(x)\big) dx$$

data

prediction

$$-\frac{1}{n_{\text{batch}}} \log \mathcal{L} = -\frac{1}{n_{\text{batch}}} \sum_{k=1}^{n_{\text{batch}}} \Big(y_k \log\big(\hat{y}(x_k)\big) + \big(1 - y_k\big) \log\big(1 - \hat{y}(x_k)\big) \Big)$$

Introduction to Deep Learning with Keras

Loss function: *cross-entropy*

$$\text{entropy} = \int p(x) \log\big(p(x)\big) dx$$

$$\text{cross-entropy} = \int q(x) \log\big(p(x)\big) dx$$

Gibbs inequality

Cross-entropy is minimal if

$p = q$

data

prediction

Hence the name...

$$-\frac{1}{n_{\text{batch}}} \log \mathcal{L} = -\frac{1}{n_{\text{batch}}} \sum_{k=1}^{n_{\text{batch}}} \Big(y_k \log\big(\hat{y}(x_k)\big) + \big(1 - y_k\big) \log\big(1 - \hat{y}(x_k)\big) \Big)$$

Introduction to Deep Learning with Keras

Example

```
import numpy as np
# We create 10000 random vectors each 10-dim
N_samples=10000
N_in=10
# A matrix N_samplesxN_in, uniform in [0,1)
x_train=np.random.rand(N_samples,N_in)
# Sum of squares along N_in
z = np.sum( np.square(x_train),axis=1)
y_train = (np.sin(z) >= 0).astype(np.int32)
```

$$\hat{y} = \begin{cases} 1 & \text{if } \sin \sum_{k=1}^{10} x_k^2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Recycle the example from Part One

$$-\frac{1}{n_{\text{batch}}} \log \mathcal{L} = -\frac{1}{n_{\text{batch}}} \sum_{k=1}^{n_{\text{batch}}} \left(y_k \log \left(\hat{y}(x_k) \right) + \left(1 - y_k \right) \log \left(1 - \hat{y}(x_k) \right) \right)$$

Introduction to Deep Learning with Keras

Example

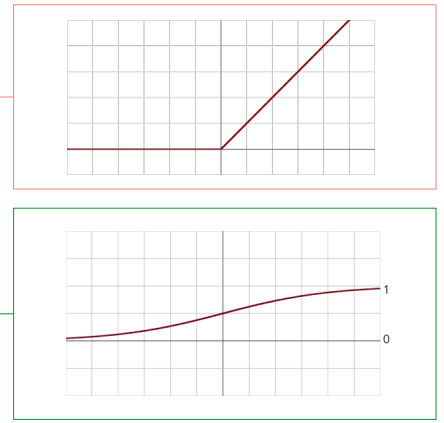
```
import numpy as np
# We create 10000 random vectors each 10-dim
N_samples=10000
N_in=10
# A matrix N_samplesxN_in, uniform in [0,1)
x_train=np.random.rand(N_samples,N_in)
# Sum of squares along N_in
z = np.sum( np.square(x_train),axis=1)
y_train = (np.sin(z) >= 0).astype(np.int32)
```

$$\hat{y} = \begin{cases} 1 & \text{if } \sin \sum_{k=1}^{10} x_k^2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Recycle the example from Part One

```
model = models.Sequential(
    [
        layers.Dense(100, activation='relu', input_dim=10),
        layers.Dense(100, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ]
)
optimizer = optimizers.Adam()
model.compile(optimizer=optimizer,loss='binary_crossentropy', metrics=['acc'])
```

Accuracy is used to Monitor the training



$$-\frac{1}{n_{\text{batch}}} \log \mathcal{L} = -\frac{1}{n_{\text{batch}}} \sum_{k=1}^{n_{\text{batch}}} \left(y_k \log \left(\hat{y}(x_k) \right) + \left(1 - y_k \right) \log \left(1 - \hat{y}(x_k) \right) \right)$$

Introduction to Deep Learning with Keras

Validation sample

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 100)               1100
_________________________________________________________________
dense_1 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 11,301
Trainable params: 11,301
Non-trainable params: 0
```

$$\hat{y} = \begin{cases} 1 & \text{if } \sin \sum_{k=1}^{10} x_k^2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

```
model.fit(x_train, y_train, batch_size=256, epochs=25, validation_split=0.2)
```

Validation sample
Keep 20% of the data to control that the machine has modelled what it was supposed to model

Control **overfitting / overtraining**

Introduction to Deep Learning with Keras

Learning curves & accuracy

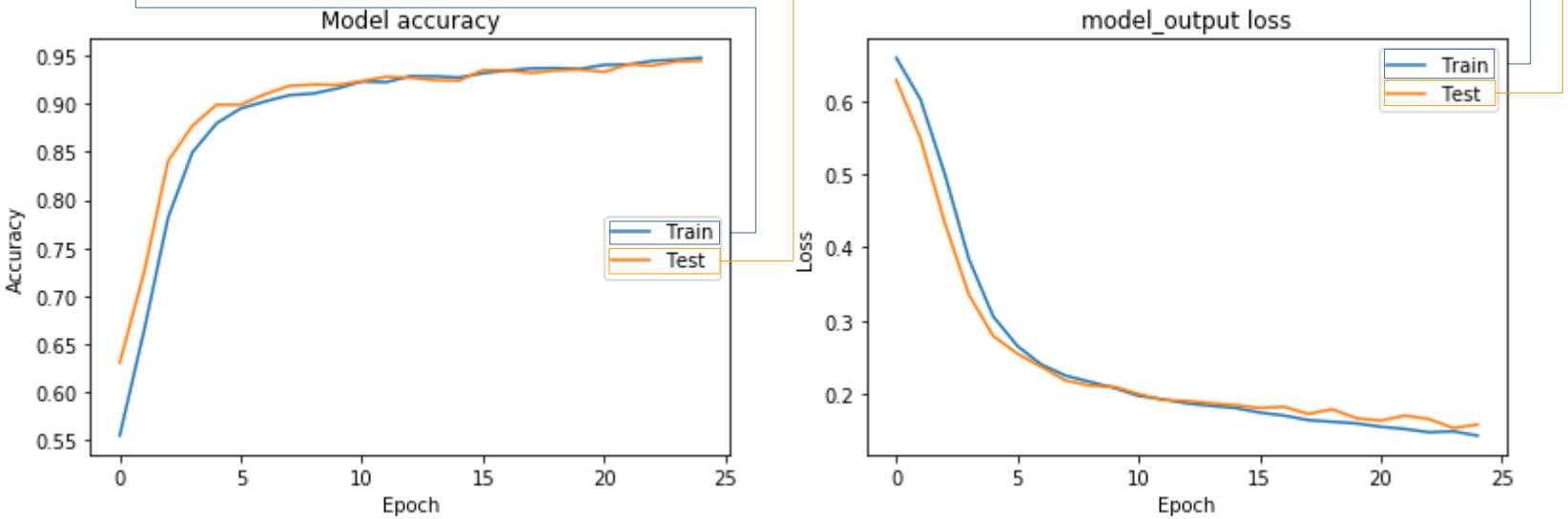
$$\text{accuracy} = \frac{\text{correctly classified data}}{\text{all data}}$$

$$\hat{y} = \begin{cases} 1 & \text{if } \sin \sum_{k=1}^{10} x_k^2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

```
histObj.history.keys()
```

```
['acc', 'loss', 'val_acc', 'val_loss']
```

- With **too few** parameters, the NN may not be able to model the data.
- With **too many** parameters, the NN may start *memorising* the training data.



Introduction to Deep Learning with Keras

Learning curves & accuracy

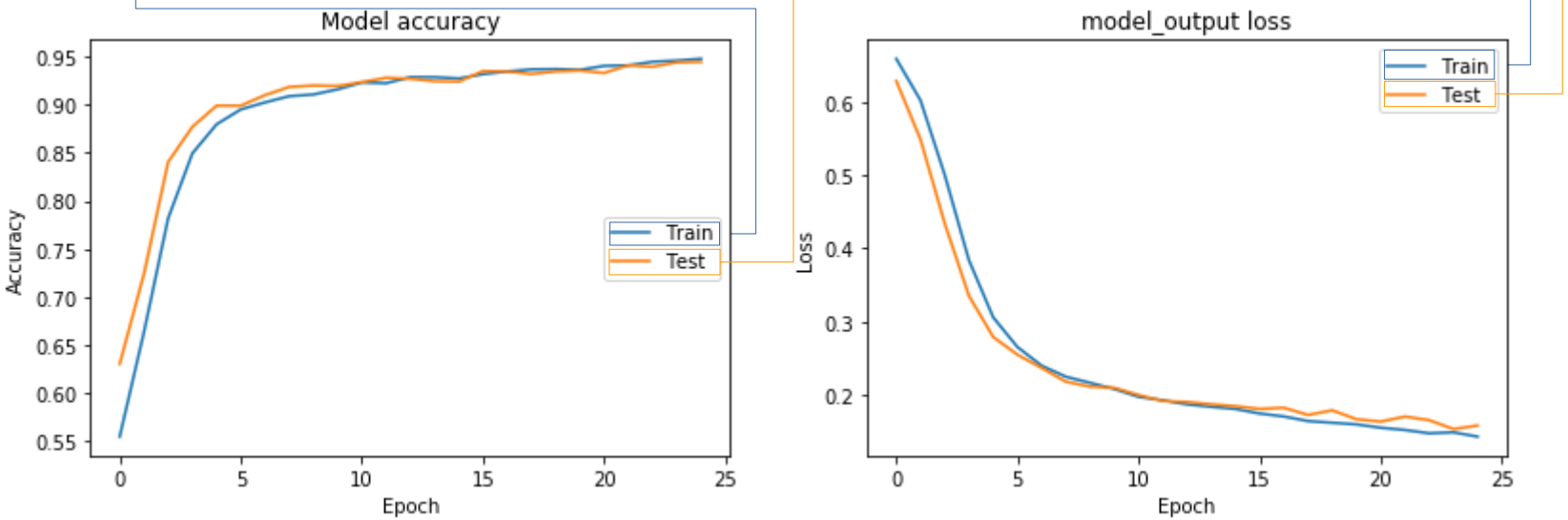
$$\text{accuracy} = \frac{\text{correctly classified data}}{\text{all data}}$$

$$\hat{y} = \begin{cases} 1 & \text{if } \sin \sum_{k=1}^{10} x_k^2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

```
histObj.history.keys()
```

```
['acc', 'loss', 'val_acc', 'val_loss']
```

- With **too few** parameters, the NN may not be able to model the data.
- With **too many** parameters, the NN may start *memorising* the training data.



Q: when should one interrupt training?

Introduction to Deep Learning with Keras

Learning curves & accuracy

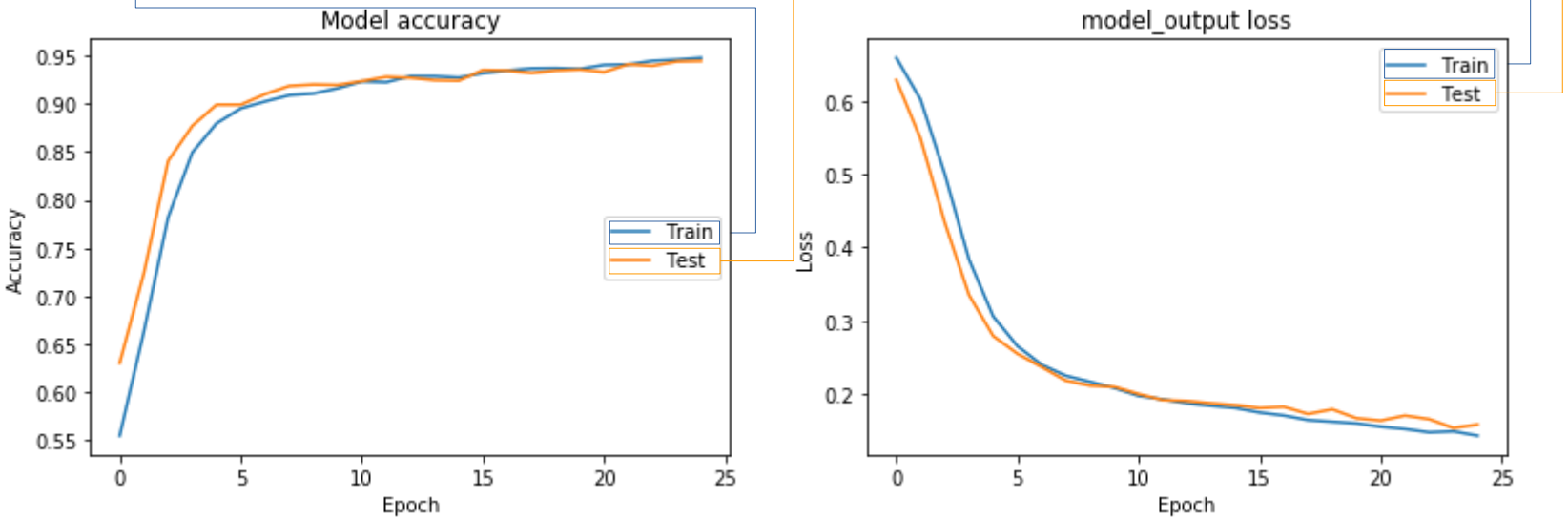
$$\text{accuracy} = \frac{\text{correctly classified data}}{\text{all data}}$$

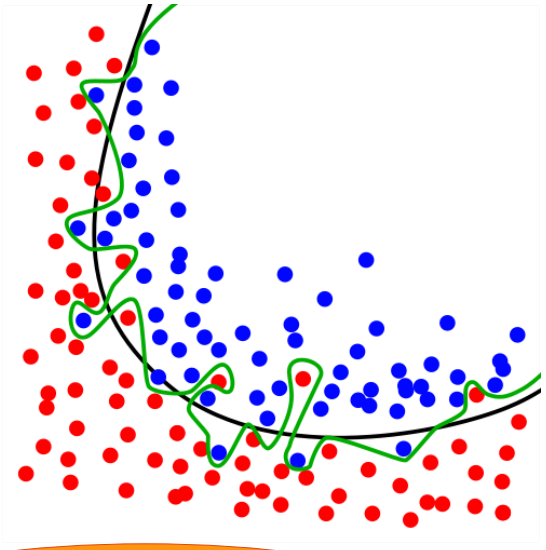
$$\hat{y} = \begin{cases} 1 & \text{if } \sin \sum_{k=1}^{10} x_k^2 \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

```
histObj.history.keys()
```

```
['acc', 'loss', 'val_acc', 'val_loss']
```

- With **too few** parameters, the NN may not be able to model the data.
- With **too many** parameters, the NN may start *memorising* the training data.





Q: when should one interrupt training?

Introduction to Deep Learning with Keras

Multi-class classification

If more than one output, we need to extend some of the concepts given above

Pedagogical examples

- Identify hand-written numbers
- Recognise clothes from catalogue

Introduction to Deep Learning with Keras

Multi-class classification

If more than one output, we need to extend some of the concepts given above

Pedagogical examples

- Identify hand-written numbers
- Recognise clothes from catalogue

Categorical cross-entropy* (CCE)

Just generalisation to multi-nominal case

*we skip the mathematical details

Softmax activation function

$$f_k(\vec{z}) = \frac{\exp z_k}{\sum_{k=1}^{K} \exp z_k}$$

Introduction to Deep Learning with Keras

Fashion-MNIST

All pictures are **prepared** in a comparable and simple format

28 pixels

28 pixels

60k pictures

Grayscale 0-255

```
import tensorflow as tf
dataset = tf.keras.datasets.fashion_mnist
```

```
(x_train, y_train),(x_test, y_test) = dataset.load_data()
print(x_train.shape)
print(x_train.dtype)
```

→ `(60000, 28, 28)`

→ `uint8`

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Introduction to Deep Learning with Keras

Fashion-MNIST

All pictures are **prepared** in a comparable and simple format

28 pixels

28 pixels

60k pictures

Grayscale 0-255

```
import tensorflow as tf
dataset = tf.keras.datasets.fashion_mnist
```

```
(x_train, y_train),(x_test, y_test) = dataset.load_data()
print(x_train.shape)
print(x_train.dtype)
```

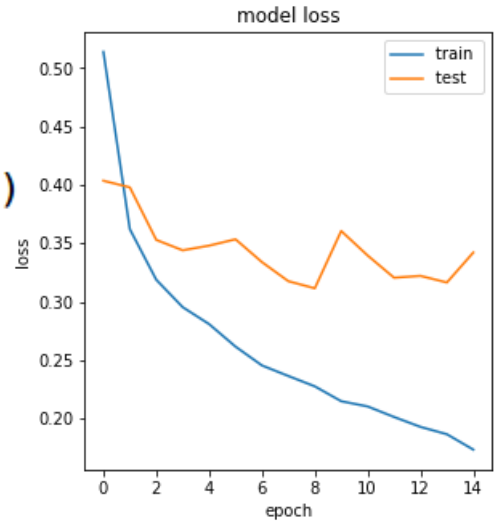
→ (60000, 28, 28)

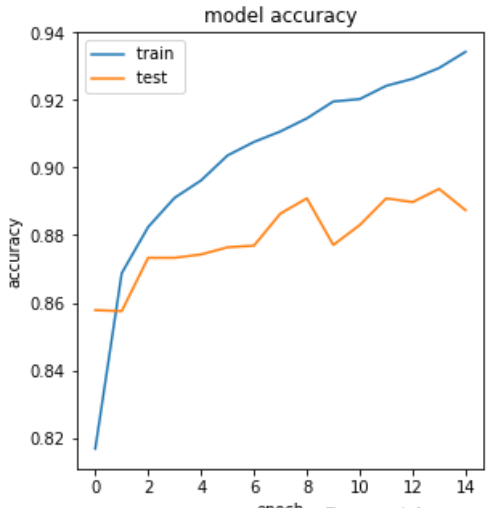
→ uint8

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```


```
model1 = tf.keras.models.Sequential([
  layers.Flatten(input_shape=(28, 28)),
  layers.Dense(512, activation='relu'),
  layers.Dense(512, activation='relu'),
  layers.Dense(10, activation='softmax')
])
model1.summary()

model1.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['acc'])
```

Improve it!





Introduction to Deep Learning with Keras

Methods to prevent over-training

Reminder

- With **too few** parameters, the NN may not be able to model the data.
- With **too many** parameters, the NN may start *memorising* the training data.

```
from tensorflow.keras import models, layers, losses, optimizers, regularizers
```

Weight regularisation

Constrain weights in successive layers with penalty term in the loss

```
kernel_regularizer=
        =regularizers.l2(0.0001)
```

Batch normalisation

Normalise the data between the layers and for each batch

```
layers.BatchNormalization()
```

Dropout

Remove randomly a certain number of nodes from iteration to iteration

```
layers.Dropout(rate=0.3)
```

Play with them!

Summary & Conclusion

Introduction to Deep Learning with Keras

What we have (not) covered & plans for tomorrow

Covered

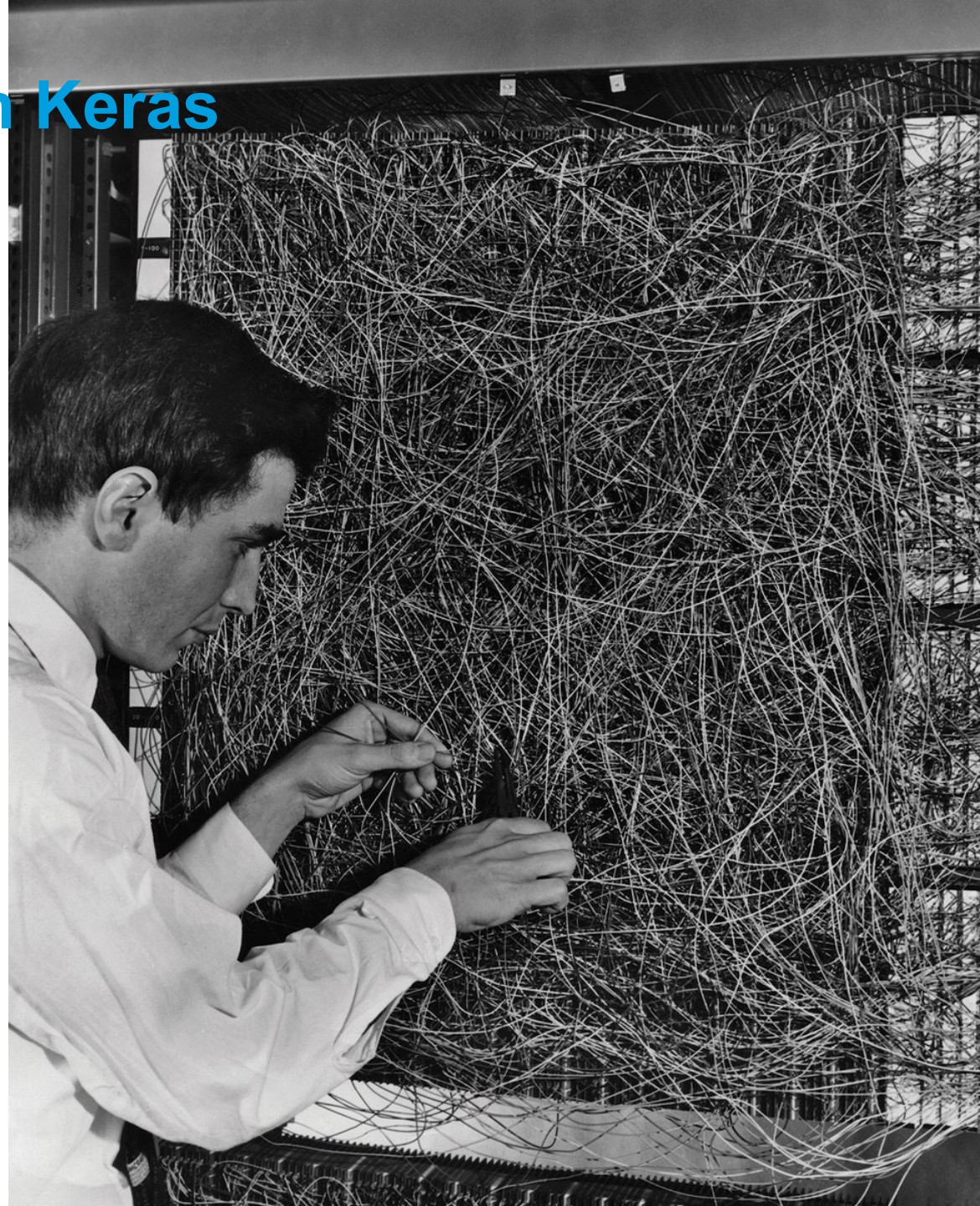
- Supervised learning
- Dense neural network
  - Activation function
    - ReLU
    - Softmax
  - Node/neuron
  - Layer
- Optimiser
  - SGD
  - Adam
- Monitoring
  - Learning curve
  - Accuracy
- Control sample

***Not* covered**

- Convolutional NN (CNN)
- Initial value for the weights
- Callbacks
- Unsupervised learning
- Auto-encoders
- Alternative libraries
- ...

Plan for the second lab

- CNN
- Use DL in context of high-energy physics
- Identify presence of top quark in *pp* collisions
- Small challenge for best performance



非常感谢！

DESY.

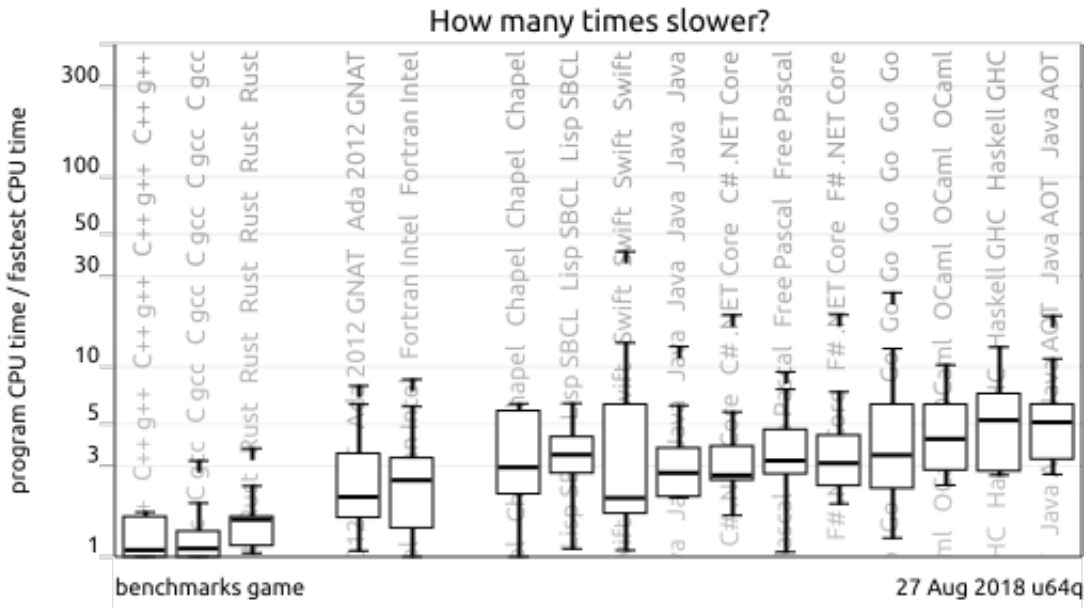
Back-up

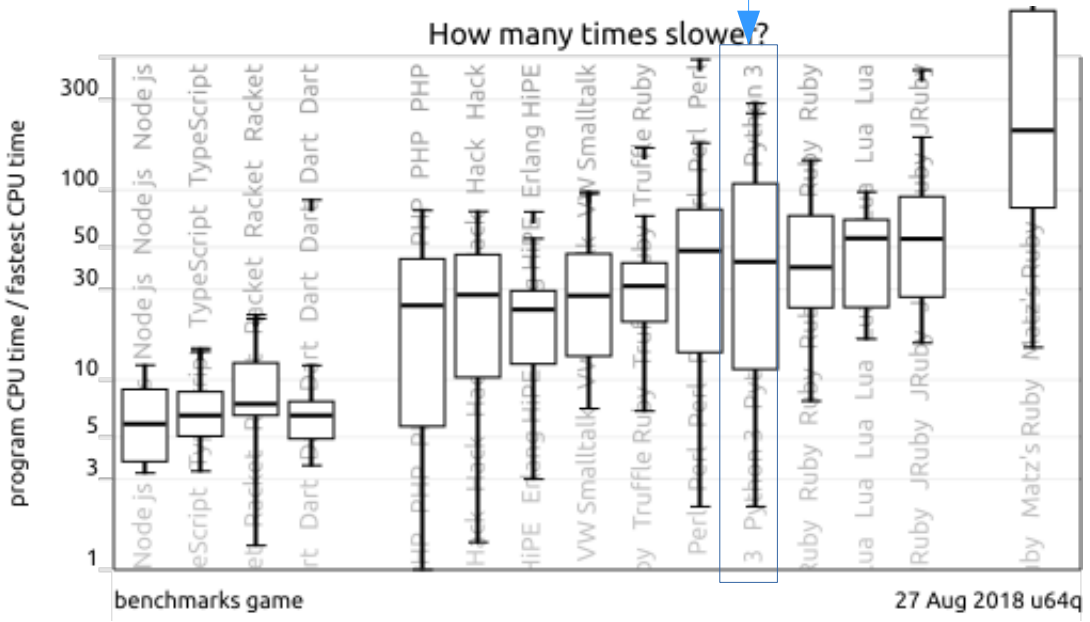
Introduction to Deep Learning with Keras

Why Python?

But don't use Python blindly!
In DL, it is "just"
a powerful interface.

- Not always the fastest language itself
- But Simple programming syntax (close to human language)
- No need to worry about pointers, references, etc.
- Powerful libraries and interfaces



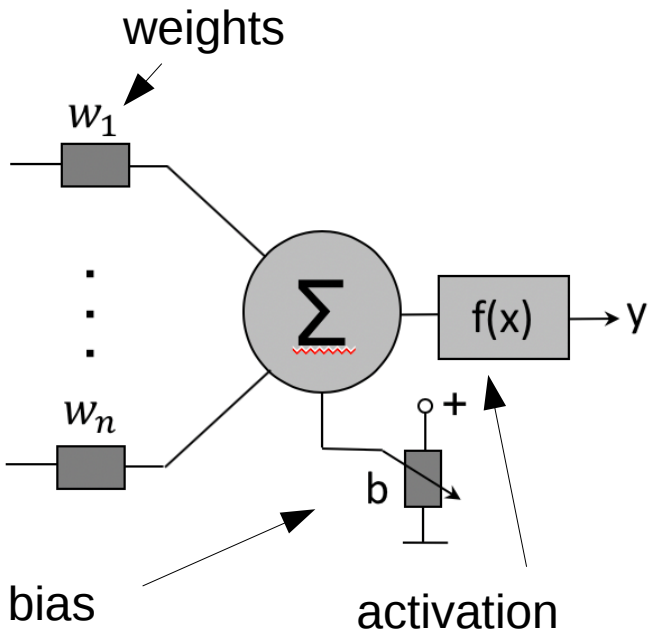


Introduction to Deep Learning with Keras

Build a neural network

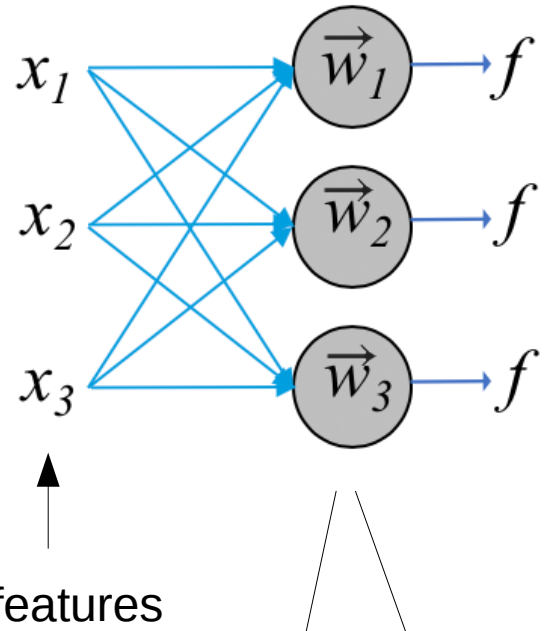
Deep Learning means many hidden layers

Node or neuron



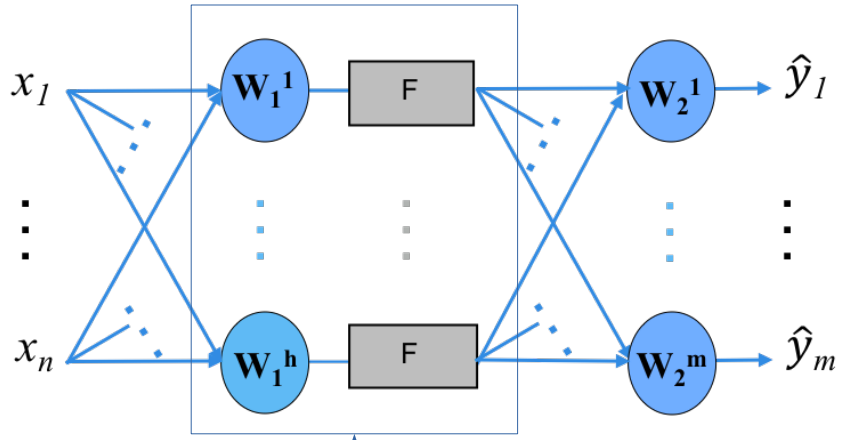
$$\hat{y} = f\left(\sum_{k=1}^{N} w^k x_k + b\right)$$

Layer



$$\hat{y}_i = f_i\left(\sum_{k=0}^{N} w_i^k x_k + b_i\right)$$

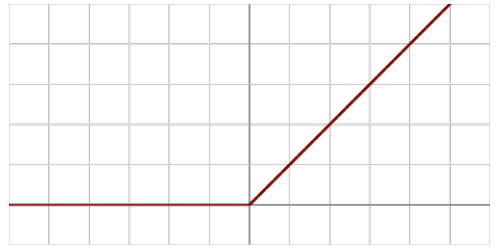
Network



$$\hat{y}_i = f_i^2\left(\sum_{k_2=1}^{N_2} w_i^{k_2} f_{k_2}^1\left(\sum_{k_1=1}^{N_1} w_i^{k_1} x_{k_1} + b_i^1\right) + b_i^2\right)$$

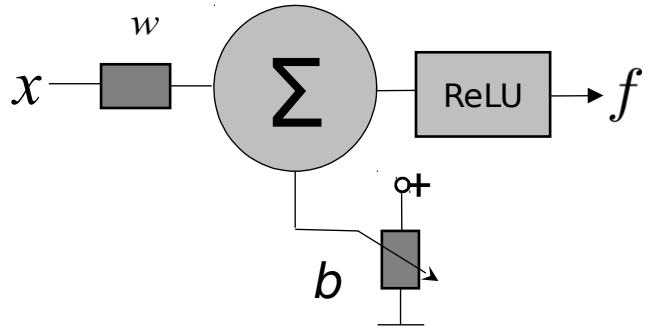
Introduction to Deep Learning with Keras

Bias: the case of ReLU



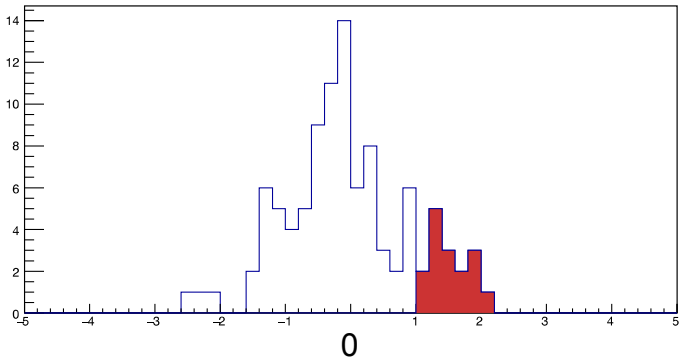
- The condition defines a cut on the input variable x

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

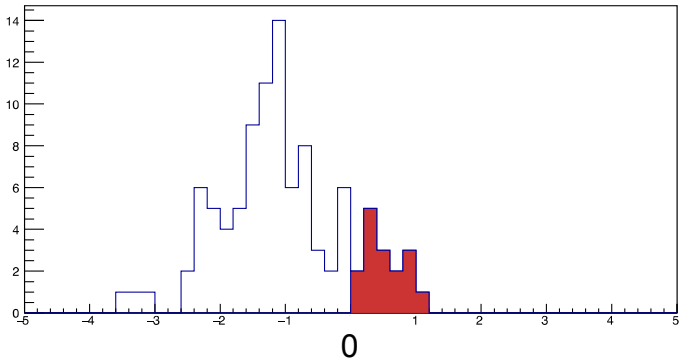


- The cut on *x* is defined by the weight *w* and the bias *b*
- Instead of cutting on $x_{in}>x_{cut}$ we scale x_{in} by *w* and shift the input distribution by *b*:

$$f(wx_{\text{in}} + b)$$

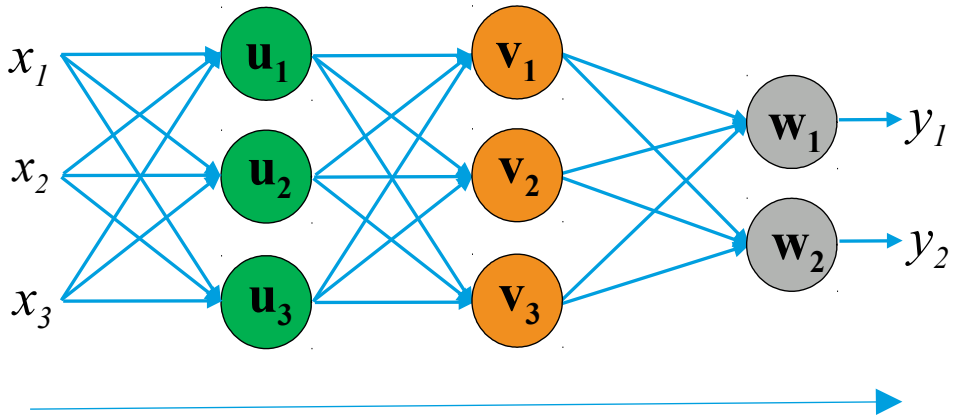


- *"We move the distribution to the cut"*
- In a ReLU layer the **bias defines the scale**



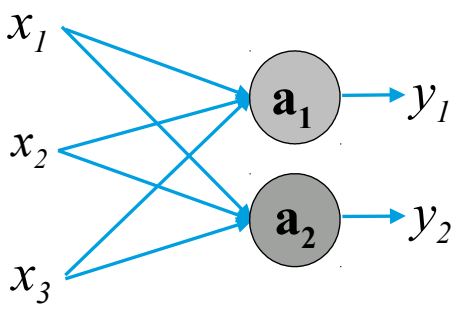
Introduction to Deep Learning with Keras

Activation function



$$\underbrace{\vec{\mathbf{y}}}_{2\times1} = \overbrace{\mathbf{W}}^{2\times3}\ \underbrace{\mathbf{V}}_{3\times3}\ \overbrace{\mathbf{U}}^{3\times3}\ \underbrace{\vec{\mathbf{x}}}_{3\times1}$$

If you remove the activation function, layers just contract...

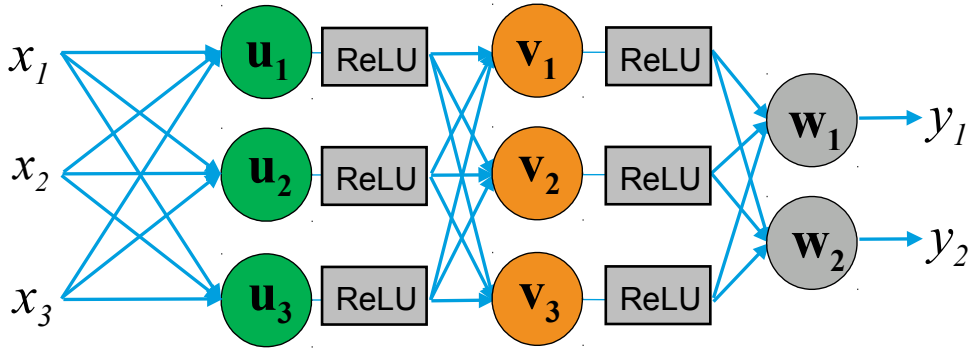


$$\underbrace{\vec{\mathbf{y}}}_{2\times1} = \overbrace{\mathbf{A}}^{2\times3}\ \underbrace{\vec{\mathbf{x}}}_{3\times1}$$

The activation function is a <u>fundamental</u> component of a Neural Network

Introduction to Deep Learning with Keras

Activation function



$$\underbrace{\vec{\mathbf{y}}}_{2\times1} = \overbrace{\mathbf{W}}^{2\times3} F\Big(\underbrace{\mathbf{V}}_{3\times3} \underbrace{F(\mathbf{U}\vec{\mathbf{x}})}_{3\times1}\Big)$$

with $F(\vec{\mathbf{x}}) := \big[F(x_i)\big]$

i.e. F applied to each component of $\vec{\mathbf{x}}$

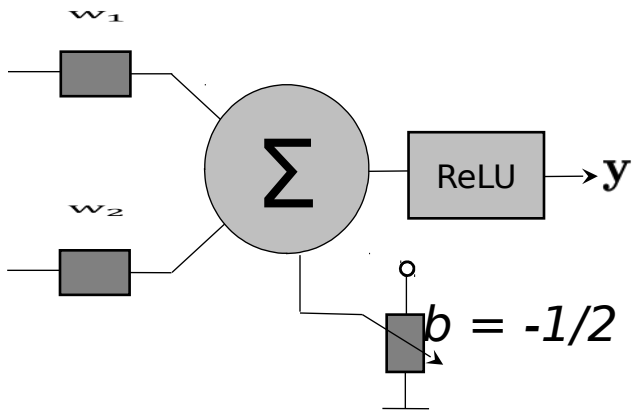
and F=ReLU

If you remove the activation function, layers just contract...

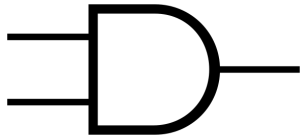
The activation function is a <u>fundamental</u> component of a Neural Network

Introduction to Deep Learning with Keras

ReLU as a logic gate

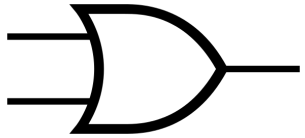


$w_1 = w_2 = 1/2$



x	y	Σ	y
0	0	0	0
1	0	1/2	0
0	1	1/2	0
1	1	1	1

$w_1 = w_2 = 1$



x	y	Σ	y
0	0	0	0
1	0	1	1
0	1	1	1
1	1	2	1

All kinds of can data manipulations can be realize by ReLU networks

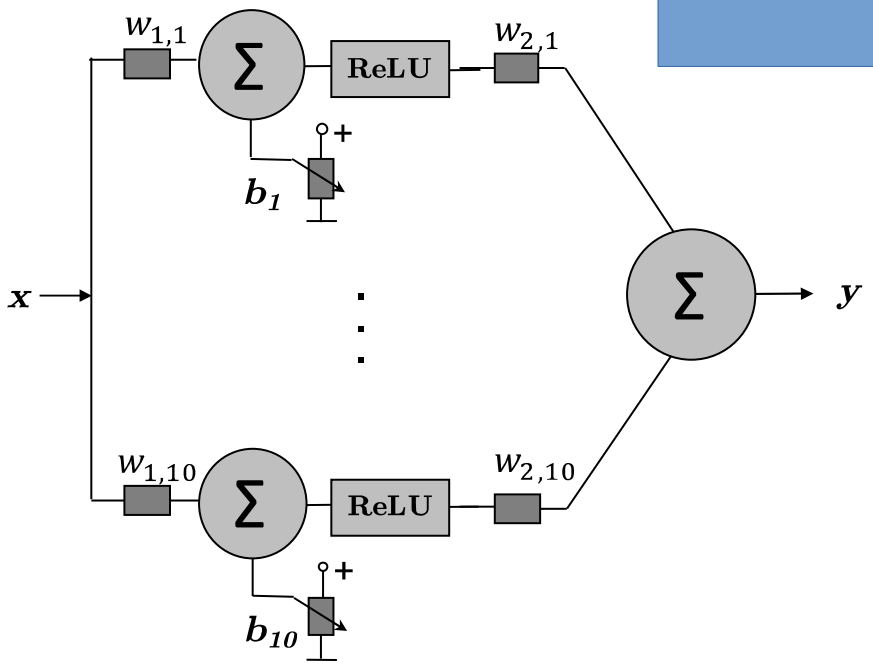
negative weight ⟶ logic not

Introduction to Deep Learning with Keras

Universal approximation

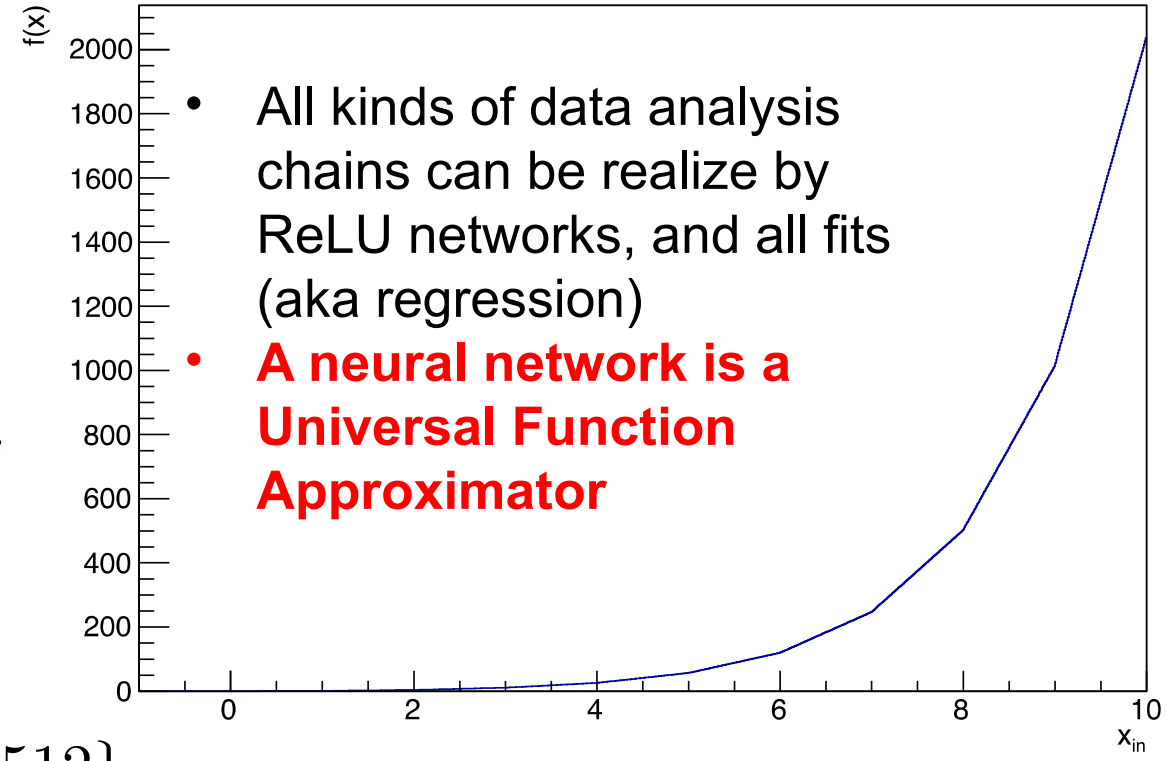
Any funtion f can be approximated with a layers FA

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)| \, \mathrm{d}x < \epsilon$$



Example

$$f(x_{\text{in}}) \approx 2^x$$



- All kinds of data analysis chains can be realize by ReLU networks, and all fits (aka regression)
- **A neural network is a Universal Function Approximator**

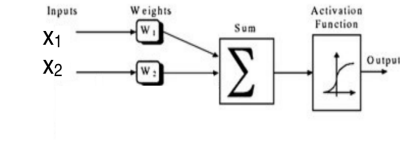
$$\{w_{1,1}, \ldots, w_{1,10}\} = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$$
$$\{w_{2,1}, \ldots, w_{2,10}\} = \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$$
$$\{b_1, \ldots, b_{10}\} = \{0, -1, -2, -3, -4, -5, -6, -7, -8, -9\}$$

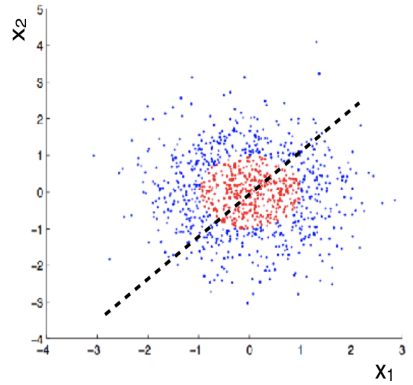
Introduction to Deep Learning with Keras

From classical ML to DL

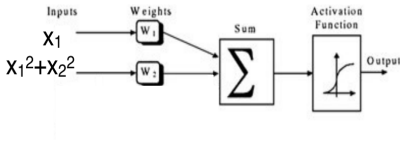
A common answer

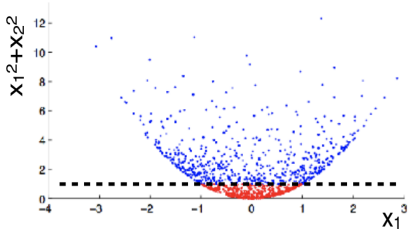
- In classical Machine Learning we had been interested in **feature** engineering, finding the best variable to solve the problem easily
- In a deep NN the first layer(s) can construct such high level features themselves from low level features





useless?





easy!

Introduction to Deep Learning with Keras

Loss minimisation

$$loss(\mathbf{y}, \hat{\mathbf{y}}) = (\hat{\mathbf{y}} - \mathbf{y})^2 \quad \text{with}$$

$$\underset{m\times 1}{\hat{\mathbf{y}}} = \underset{m\times h}{\mathbf{W}_2}\, F(\underset{h\times n}{\mathbf{W}_1}\, \underset{n\times 1}{\mathbf{x}})$$

Back-propagation

- To minimize the loss we take the derivative wrt. to $\boldsymbol{w}$

$$\frac{\partial loss}{\partial w_i} = 0 \Rightarrow \frac{\partial (\hat{\mathbf{y}} - \mathbf{y})^2}{\partial \mathbf{W}_i} = 2(\hat{\mathbf{y}} - \mathbf{y}) \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_i} = 0$$

E.g. for W_2 -
Similar for W_2

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_1} = \mathbf{W}_2 \frac{\partial F(z)}{\partial z} \mathbf{x}$$

- Equation to be solved for $W_{1,2}$
- **Chain rule – from the end to the beginning**

Introduction to Deep Learning with Keras

Loss minimisation

- The previous chain rule calculation is known as **Backpropagation**
- The deviation between true and estimated y, i.e. the loss, is back-propagated to a linear change of the weights.
  - Invented by different people in the 1960/70s
- **NB: The chain rule creates a chain of factors that can be evaluated numerically**

$$\Delta loss(\hat{\mathbf{y}}) \approx \frac{\partial loss}{\partial \mathbf{W}} \Delta \mathbf{W}$$

We calculate a **gradient** with respect to the weights/biases

$$\frac{\partial loss(\hat{\mathbf{y}}, \mathbf{y})}{\partial \mathbf{W}_i} = \frac{\partial loss(\hat{\mathbf{y}}, \mathbf{y})}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial F} \frac{\partial F(z)}{\partial z} \frac{\partial z}{\partial \mathbf{W}_i}$$

- These are vector and matrix multiplication. If you need a Matrix calculus primer or work it out with tensor indices but in reality …

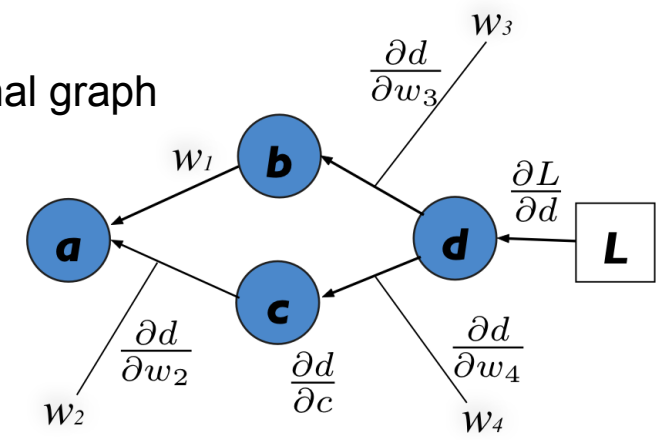
Introduction to Deep Learning with Keras

Automatic Differentiation

- Deep Learning libraries are able to calculate the derivative of a piece of code
  - This is **not** a numerical approximation (no small epsilon)
  - This **not** symbolic differentiation (not as in e.g. Mathematica)
  - Think about it as **a derivative of your python code**

```
b = w1 * a
c = w2 * a
d = (w3 * b) + (w4 * c)
L = f(d)
```

computational graph



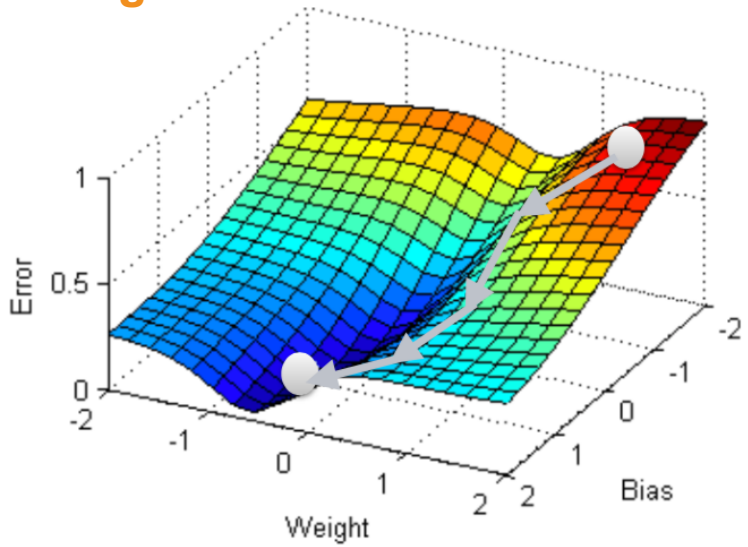
- It records your calculations (Forward step)
- It then calculates by applying the chain rule (Backward step) the gradients at the same numerical value
- The DL library keeps track of **hundreds of thousands of weights and more**
- There are different approaches to automatic differentiation

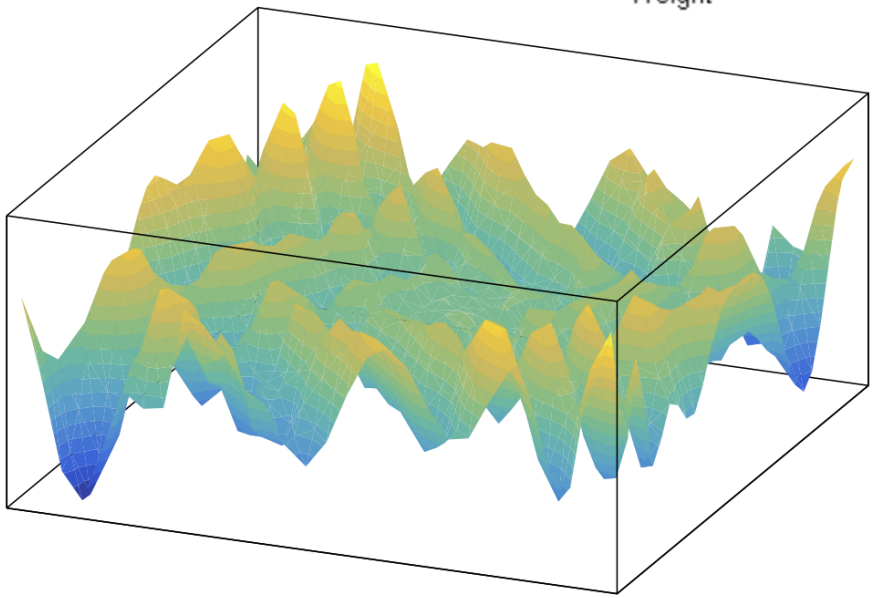
  The different libraries e.g. Tensorflow, Pytorch etc. follow slightly different concepts

  Important to understand if you want to define your own special layers and loss function

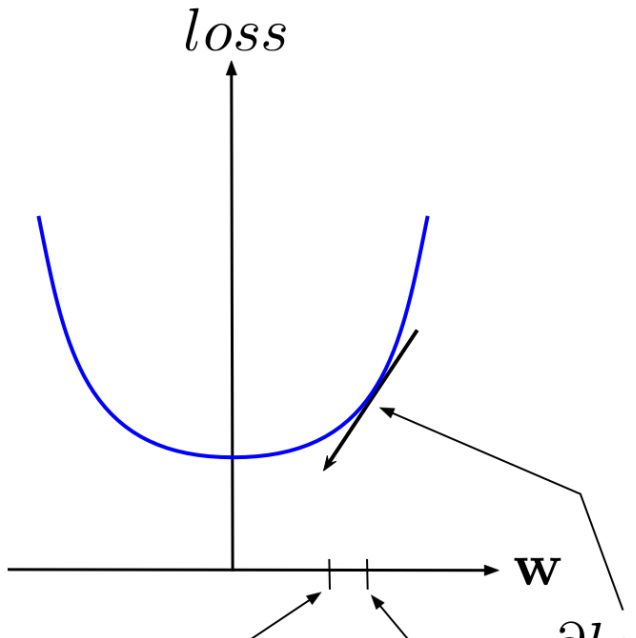
Introduction to Deep Learning with Keras

Gradient descent & learning rate





$$\Delta loss(\hat{\mathbf{y}}) \approx \frac{\partial loss}{\partial \mathbf{W}} \Delta \mathbf{W}$$



$$\mathbf{w_{i+1}} = \mathbf{w_i} - \left.\frac{\partial loss}{\partial \mathbf{w}}\right|_{\mathbf{w_i}} \times learning\ rate$$

Introduction to Deep Learning with Keras

Adam

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Overview: https://arxiv.org/abs/1609.04747
Seminal paper by LeCun (1998)

Introduction to Deep Learning with Keras

GPU usage

Graphical
Processor
Unit

GPUs for video games can be used for DL

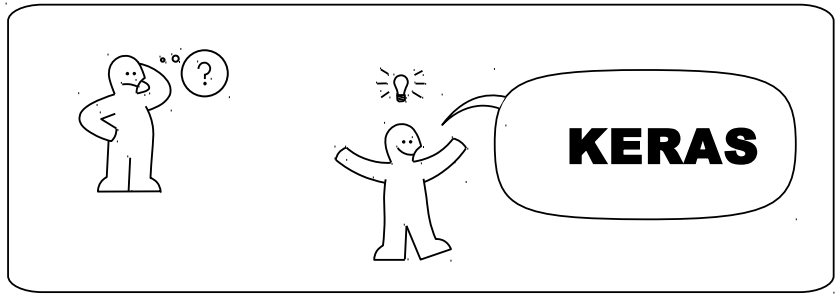
Applications

- 3D graphics
- Cryptocurrencies
- Vectorial calculation
- *Deep learning*

```
from tensorflow.python.client import device_lib
import tensorflow as tf
print(tf.__version__)
print(device_lib.list_local_devices())
```

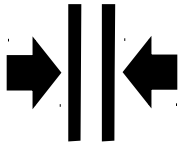
NEURALA NÄTVERK

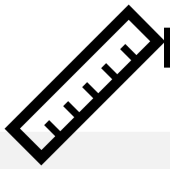
Assembly Instruction

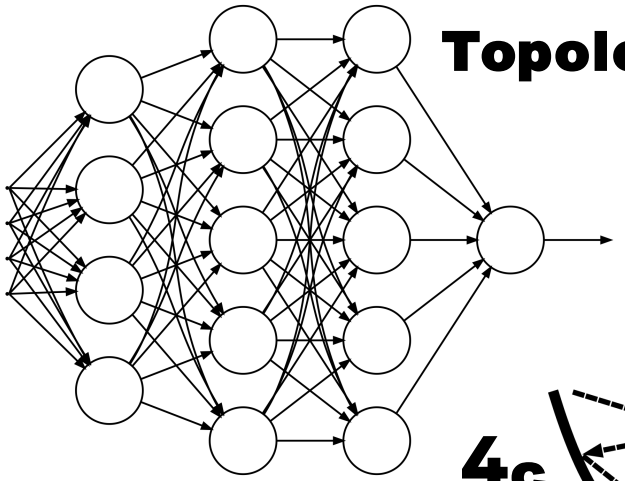


1  Features

2a Train  Test

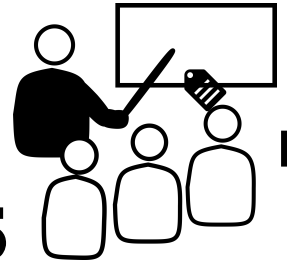
2b  Normalize

3  Metric

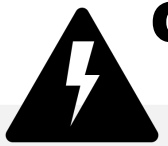
4a  Topology

4c  Optimizer

4b  Loss

 Fit

5

6  Overtraining

7 Apply!

NEURALA NÄTVERK

1x python

1x K + TensorFlow

1x

2x **ReLU**

1x **Flatten**

1x **Softmax**

1x **X-entropy**

1x **Adam**

Assembly Instruction

```
import tensorflow as tf
mnist = tf.keras.datasets.fashion_mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0 - 0.5, x_test / 255.0 - 0.5

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(512, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
=model.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
history = model.fit(x_train, y_train,
           validation_data=(x_test,y_test),
           batch_size=32, epochs=5 )
model.evaluate(x_test, y_test)
```

```
import numpy as np
fiveImages = x_test[0:5]
predictions = model.predict(fiveImages)
predictions = np.argmax(predictions,axis=1)
import matplotlib.pyplot as plt
class_names = ['T-shirt/top', 'Trouser', 'Pullover',
               'Dress', 'Coat', 'Sandal', 'Shirt',
               'Sneaker', 'Bag', 'Ankle boot']
plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(fiveImages[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[predictions[i]])
plt.show()
```