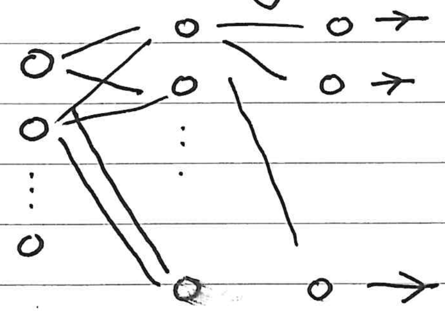
Artificial
# Neural Networks (ANN)

$\boxed{\text{TIF285}}$

201013 (1)

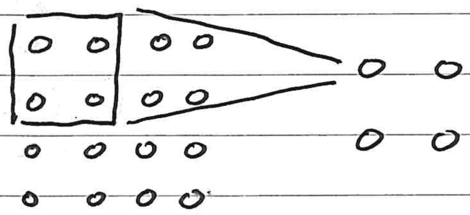
Various types of ANNs are used for many different ML tasks:

- Feed-forward neural networks (FFNN) (a.k.a multilayer perceptron although not necessarily step-function activation)



~~Exp.~~

- Convolutional NN (CNN)



 - not fully connected
 - spatial structure of input
 - reduced # weights e.g. image recognition

- Recurrent NN (RNN)
 - not just feed-forward
 - output depends on previous cycles ⇒ sequential information
 - used for e.g. text and speech recognition

- Other types; also for unsupervised learning. Deep Boltzmann Machines

## FFNN

Can be described by its

1. Architechure
2. Activation rule(s)
3. Learning algorithm

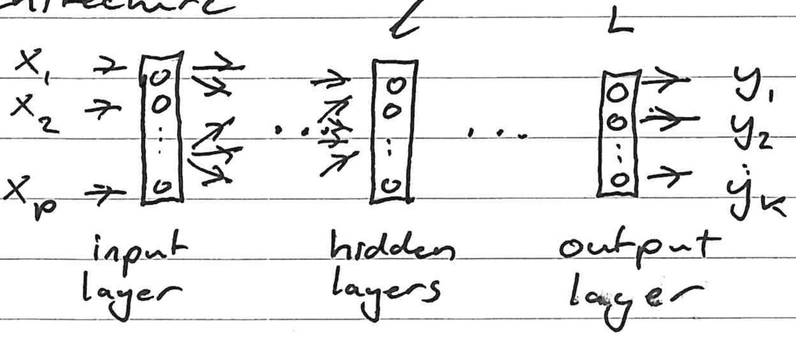
Universal approximation theorems imply that ANNs ~~kap~~ are able to represent various functional relationships.

---

E.g. a FFNN with a single hidden layer and a finite # of neurons can approximate a continuous multidim. function to arbitrary accuracy assuming that the activation function is non-constant, bounded, monotonically increasing, continuous.
( ⇒ non-linear )

---

1/ Architechure

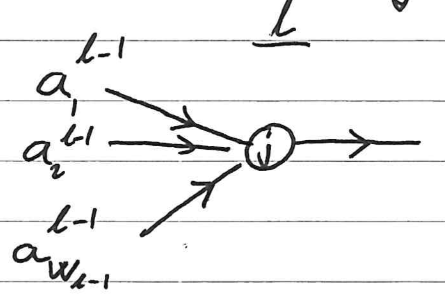



to be compared with targets $t$

Each layer has a number of nodes/units/neurons, each with its set of weights and bias.
The width $W_\ell$ of layer $\ell$

2/ Activation rule

consider a single neuron $j$
in hidden layer $l$




$a_j^l = f(z_j^l)$

↑
outputs from $W_{l-1}$ nodes in layer $l-1$

Note that the output $a_j^l$ becomes an input ~~for~~ to all nodes in layer $(l+1)$

The activation $z_j^{(l)} = \sum_i w_{ij}^{(l)} a_i^{l-1} + b_j^l$

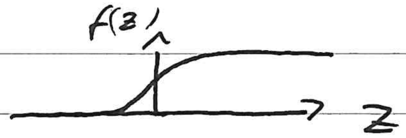
or in matrix form

$$\underset{[W_l \times 1]}{z^l} = \underbrace{[\mathbb{W}^l]^T}_{W_{l-1} \times W_l} \underbrace{a^{l-1}}_{W_{l-1} \times 1} + \underset{\text{vector } [W_l \times 1]}{\mathbb{b}^l}$$

Several different activation functions can be considered

- Sigmoid (logit) $f(z) = \sigma(z) \equiv \frac{e^z}{1+e^z}$

  · suffers from vanishing gradients ⇒ slow learning



- Hyperbolic tangent

$$f(z) = \tanh(z) = 2\sigma(2z) - 1$$

Both of these imply that signals will be non-zero everywhere ⇒ inefficient.

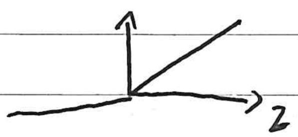
- Rectified linear unit (ReLU)

$$f(z) = \max(0, z)$$

Many neurons will be quiet. Good!
But $\frac{\partial f}{\partial w} = 0$ if $z < 0$

⇒ dying ReLUs

⇒ Leaky ReLU



$$f(z) = \max(0, z) + \alpha \min(0, z)$$

with $\alpha \sim 0.01$

or Exponential Linear Unit (ELU)

$$f(z) = \begin{cases} z & \text{for } z \geq 0 \\ \alpha e^{z} - 1 & \text{for } z \leq 0 \end{cases}$$

Note that the output layer often has a linear activation function to give a continuous output, or softmax to give classification probabilities.
The propagation of signals through the ANN is known as <u>feed-forward</u>

3) Learning algorithm

- Model (cost function)
  MSE, MAE, cross-entropy
  i.e. how to compare output with targets. (Physics knowledge can be incorporated)
- Regularization
- Back-propagation (using the chain rule to obtain the gradients to adjust the parameters)
- Gradient descent
  - standard SGD
  - momentum -11-
  - AdaGrad
  - RMS prop
  - Adam

- Data
  - training data
  - ~~test data~~
  - validation data (adjust hyperpars)
  - test data (final test of performance. <u>cannot</u> be used for tuning)

Since learning is stochastic we typically feed the training data many times (feed-forward + backprop). One complete pass is known as an <u>epoch</u>.

The data is often split into <u>batches</u>: doing the gradient for a batch of data.

The validation score is evaluated for each ~~bat.~~ epoch.

Tune # of epochs (eventually overfitting)
batch size
learning rate (monitor both ~~training~~ and val. scores)

Conclusion

- Scary many options...
- A physicist also wants to learn about the model itself!

# Backpropagation algorithm

Backprop: Using the chain rule to obtain the gradient of the cost function w.r.t weights and biases.

Learning: E.g. with SGD; training of weights and biases

## Regression example

The ANN gives $n$ output $\{y^{(i)}\}_{i=1}^{n}$ to be compared with targets $\{t^{(i)}\}_{i=1}^{n}$.

The output is the output of the last layer ($L$): $y^{(i)} = a_i^L$

And the cost function

$$C(\Theta) = \frac{1}{2}\sum_{i=1}^{n}\left(a_i^L - t^{(i)}\right)^2$$

depends on all weights and biases of the network.

Consider node $j$ in hidden layer $l$

$$z_j^l = \sum_i w_{ij}^l a_i^{l-1} + b_j^l \quad \text{(activation)}$$

$$a_j^l = f(z_j^l) = \frac{1}{1+e^{-z_j^l}} \quad \text{(output; logistic activation)}$$

Chain rule

$$\frac{\partial z_j^l}{\partial w_{ij}^l} = a_i^{l-1} \quad ; \quad \frac{\partial z_j^l}{\partial a_i^{l-1}} = w_{ij}^l$$

$$\boxed{\frac{\partial a_j^l}{\partial z_j^l}} = \frac{\partial f(z_j^l)}{\partial z_j^l} = \begin{Bmatrix}\text{Logit} \\ \text{function}\end{Bmatrix} = -\frac{-e^{-z_j^l}}{(1+e^{-z_j^l})^2}$$

$$= \frac{(1+e^{-z_j^l})-1}{(1+e^{-z_j^l})^2} = \boxed{a_j^l(1-a_j^l)}$$

Derivatives of the cost function

$$\frac{\partial C}{\partial w_{jk}^L} = (a_j^L - t^{(j)})\frac{\partial a_j^L}{\partial w_{jk}^L}$$

$$= \frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial w_{jk}^L}$$

$$= a_j^L(1-a_j^L)\, a_k^{L-1}$$

Let us define

$$\delta_j^L \equiv a_j^L(1-a_j^L)\left[a_j^L - t^{(j)}\right]$$

$$= f'(z_j^L)\frac{\partial C}{\partial a_j^L} = \frac{\partial C}{\partial z_j^L}$$

which gives

$$\boxed{\frac{\partial C}{\partial w_{jk}^{L}} = \delta_j^{L} \cdot a_k^{L-1}}$$

Also

$$\boxed{\frac{\partial C}{\partial b_j^{L}}} = \left[a_j^{L} - t^{(j)}\right] \underbrace{\frac{\partial a_j^{L}}{\partial b_j^{L}}}_{= \frac{\partial a_j^{L}}{\partial z_j^{L}} \underbrace{\frac{\partial z_j^{L}}{\partial b_j^{L}}}_{=1}}$$

$$= \left[a_j^{L} - t^{(j)}\right] a_j^{L}(1-a_j^{L}) = \boxed{\delta_j^{L}}$$

Now, consider Layer $l$

$$\delta_j^{l} = \frac{\partial C}{\partial z_j^{l}}$$

can be defined in terms of Layer $(l+1)$:

$$\delta_j^{l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^{l}}$$

$$= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^{l}}$$

where

$$z_j^{l+1} = \sum_{i=1}^{W_l} w_{ij}^{l+1} \underbrace{a_i^l}_{= f(z_j^l)} + b_j^{l+1}$$

such that

$$\delta_j^l = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l)$$

Finally, we can compute all the $\delta_j^l$ quantities from the current set of weights and outputs — starting from the last layer and propagating backwards!

The SGD learning (with rate $\eta$) then implies

$$\boxed{\begin{aligned} w_{jk}^l &\leftarrow w_{jk}^l - \eta \delta_j^l a_k^{l-1} \\ b_j^l &\leftarrow b_j^l - \eta \delta_j^l \end{aligned}}$$