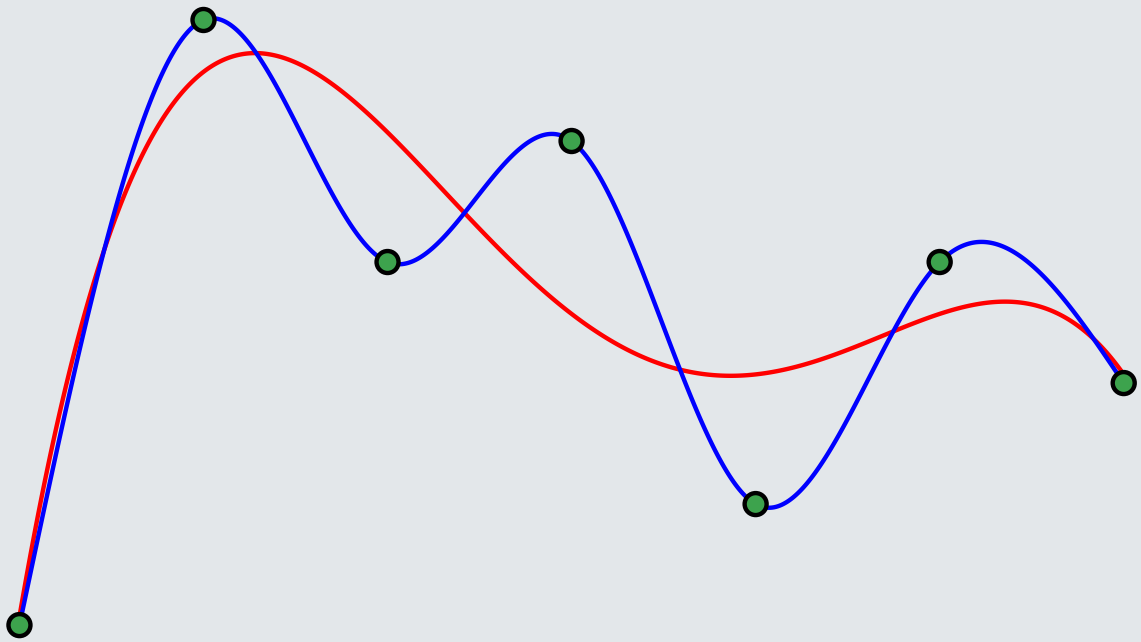
Neural Networks

Laurent Hoeltgen

14th June 2019

What's the Task?

- We are given ℓ correspondences $X \in \mathbb{R}^n \leftrightarrow Y \in \mathbb{R}^k$
- We want to find a correspondence for an arbitrary $X \in \mathbb{R}^n$
- Sounds like interpolation ... or approximation?



Neural Network with 3 Layers

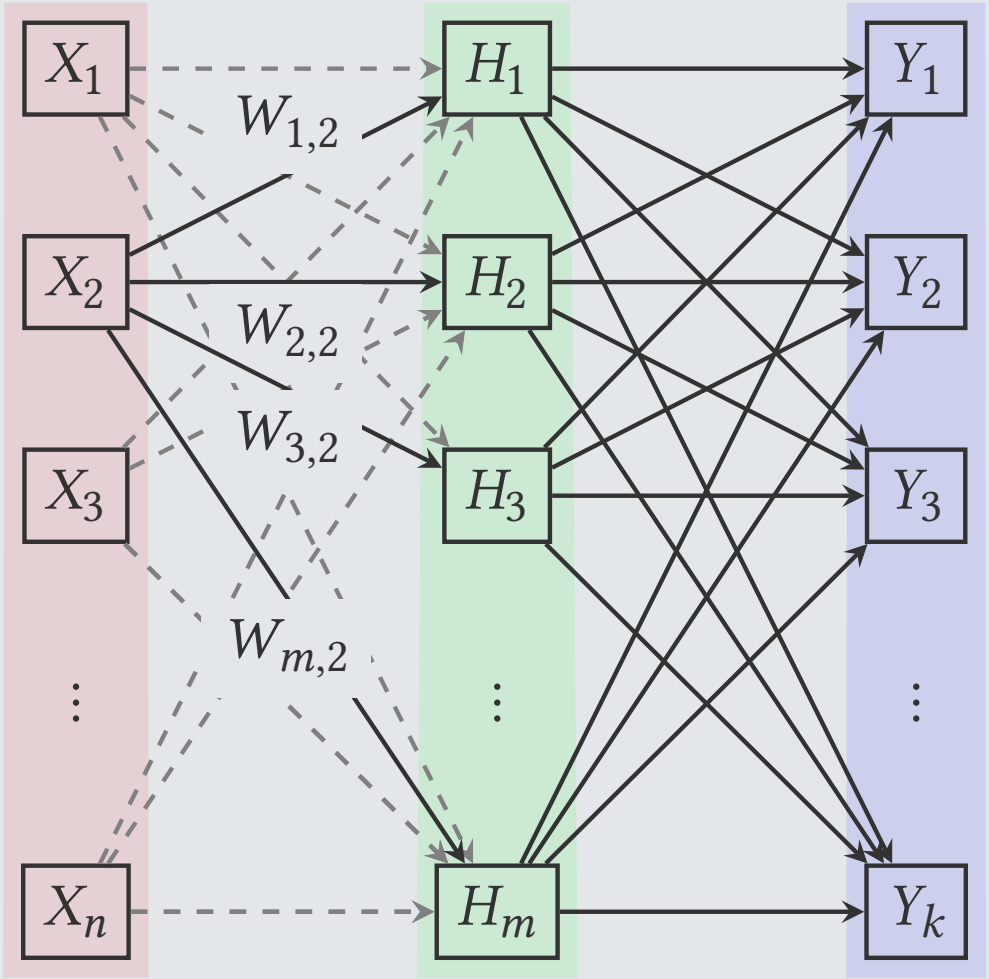


Figure: 3 Layer Neural Network

- Hidden layer evaluates $H = f_1(WX)$, with $W \in \mathbb{R}^{m,n}$, activation function $f_1 : \mathbb{R}^m \longrightarrow \mathbb{R}^m$ and input $X \in \mathbb{R}^n$
- Ouptut layer evaluates $Y = f_2(VH)$, with $V \in \mathbb{R}^{k,m}$, activation function $f_2 : \mathbb{R}^k \longrightarrow \mathbb{R}^k$ and input $H \in \mathbb{R}^m$
- Neural Network corresponds to the function

$$Y = f_2\left(V f_1\left(WX\right)\right)$$

- Matrices W and V are free parameters

Activation Functions

There's a whole zoo: SQNL, ReLU, APL, ...[1]
Nice to haves:

Nonlinear Otherwise the networks collapses to 1 layer

Suitable Range Improves learning

Smoothness Necessary for optimisation

Monotonicity Yields convex models (sometimes)

Approximates Identity Makes initialisation easier

For simplicity we use a sigmoid.

1. https://en.wikipedia.org/wiki/Activation_function

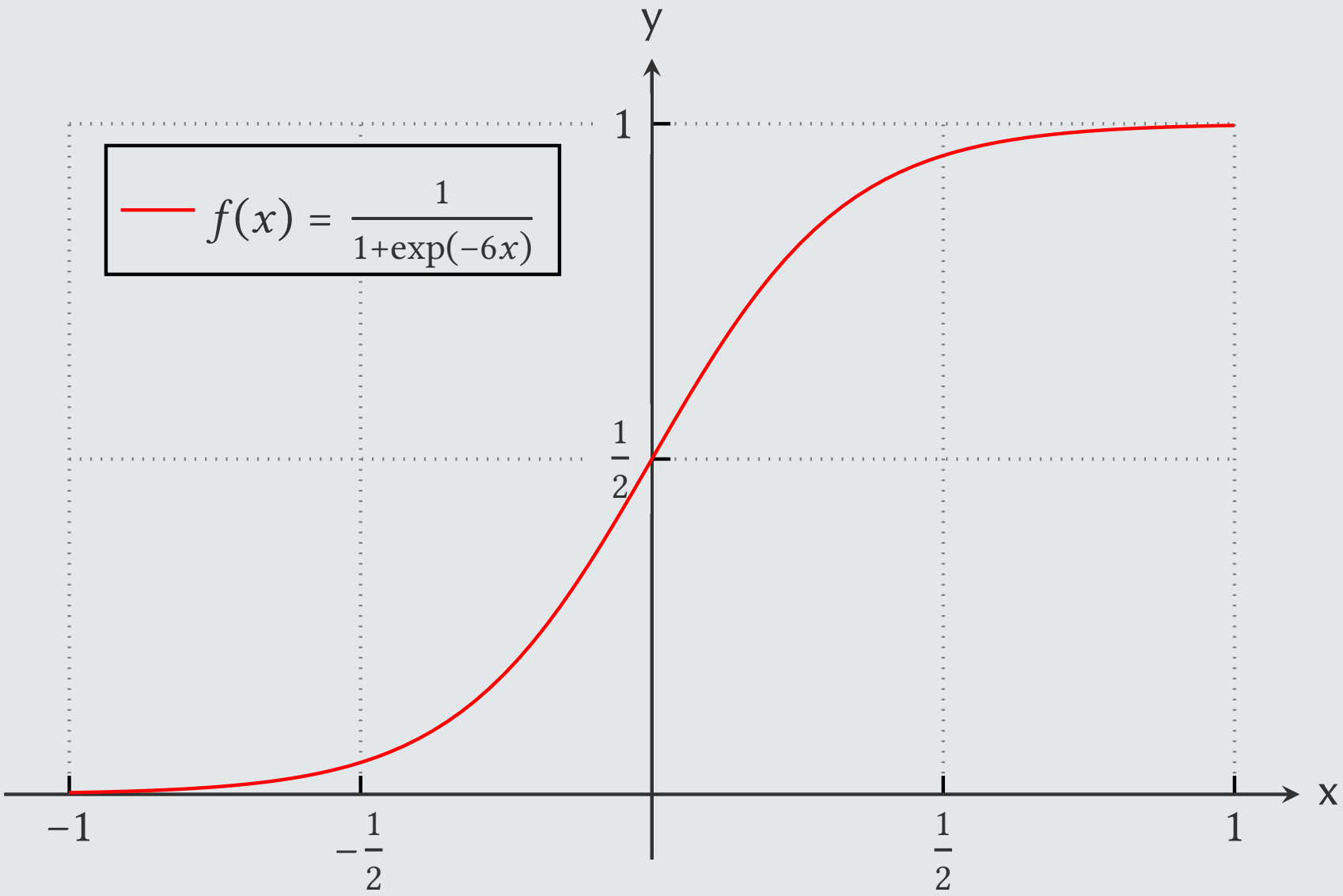


Figure: Sigmoid function

What do we learn?

A query of the Network is just an evaluation of $Y = f_2\left(V f_1\left(WX\right)\right)$

Training is done by finding matrices W and V that solve

$$\arg\min_{V,W}\left\{\sum_{i=1}^{\ell}\left\|Y_i - f_2\left(V f_1\left(WX_i\right)\right)\right\|^2\right\}$$

Simplest idea: gradient descent (per layer/per correspondence).

Moving forward from one layer to the next:

$$H_i = \sum_j W_{i,j} X_j, \quad (H = WX)$$

Moving backward fron one layer to the previous:

$$X_j = \sum_i W_{i,j} H_i, \quad (X = W^\top H)$$

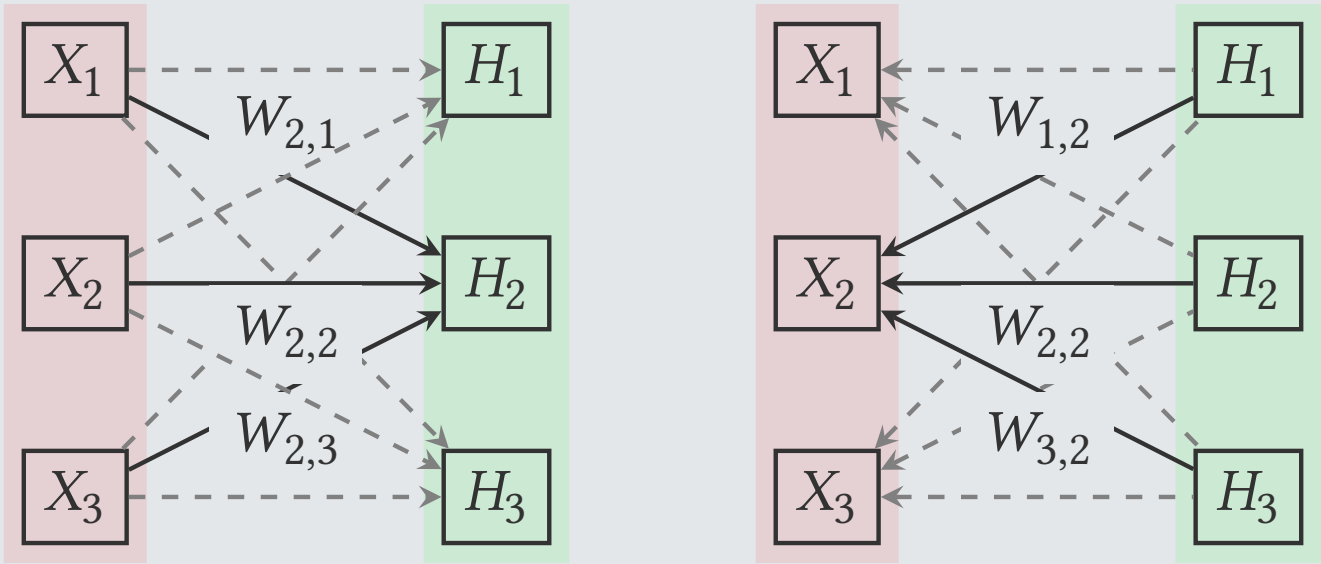


Figure: Moving between Layers

Gradient descent needs the derivatives w.r.t. V and W

$$\frac{\partial}{\partial V}\frac{1}{2}\left\|Y_i - f_2\left(V f_1\left(W X_i\right)\right)\right\|^2$$

$$= -\left(Y_i - f_2\left(V f_1\left(W X_i\right)\right)\right)^\top D[f_2]\left(V f_1\left(W X_i\right)\right)\left(\left(f_1\left(W X_i\right)\right)^\top \otimes \mathrm{I}\right)$$

$$\frac{\partial}{\partial W}\frac{1}{2}\left\|H_i - f_1\left(W X_i\right)\right\|^2$$

$$= -\left(H_i - f_1\left(W X_i\right)\right)^\top D[f_1]\left(W X_i\right)\left(X_i^\top \otimes \mathrm{I}\right)$$

Full training approach

Algorithm 1: Training a Neural Network (1 Epoch)

```
1 Evaluate f_2(V f_1(W X_i)) for all samples X_i
2 Compute Errors E_i := Y_i − f_2(V f_1(W X_i))
3 foreach error E_i do
4 |   foreach Layer do
5 |   |   Perform gradient descent step to update weights
6 |   |   Push error to previous layer
7 |   end foreach
8 end foreach
```

Literatur I

Magnus, Jan R., and Heinz Neudecker. 2007. **Matrix Differential Calculus with Applications in Statistics and Econometrics.** 3rd ed. Wiley Series in Probability and Statistics. John Wiley & Sons. ISBN: 0-471-98632-1.

Pollock, D. S. G. 1985. 'Tensor Products and Matrix Differential Calculus' [in English]. **Linear Algebra and Its Applications** 67:169–193.

Rashid, Tariq. 2016. **Neuronale Netze selbst programmieren: Ein verständlicher Einstieg mit Python** [in Deutsch]. Translated English by Frank Langenau. O'Reilly. ISBN: 978-3-96009-043-4.

———. 2018. 'Code for the Make Your Own Neural Network book'. Accessed 10 June 2019. https://github.com/makeyourownneuralnetwork/makeyourownneuralnetwork.