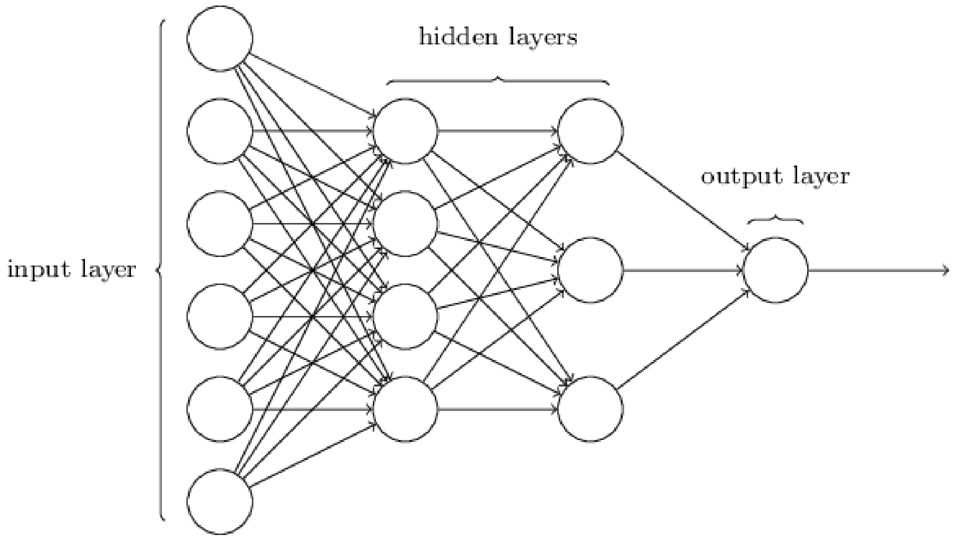
Lecture 3 – Machine Learning Fundamentals

Outline – Lecture 3

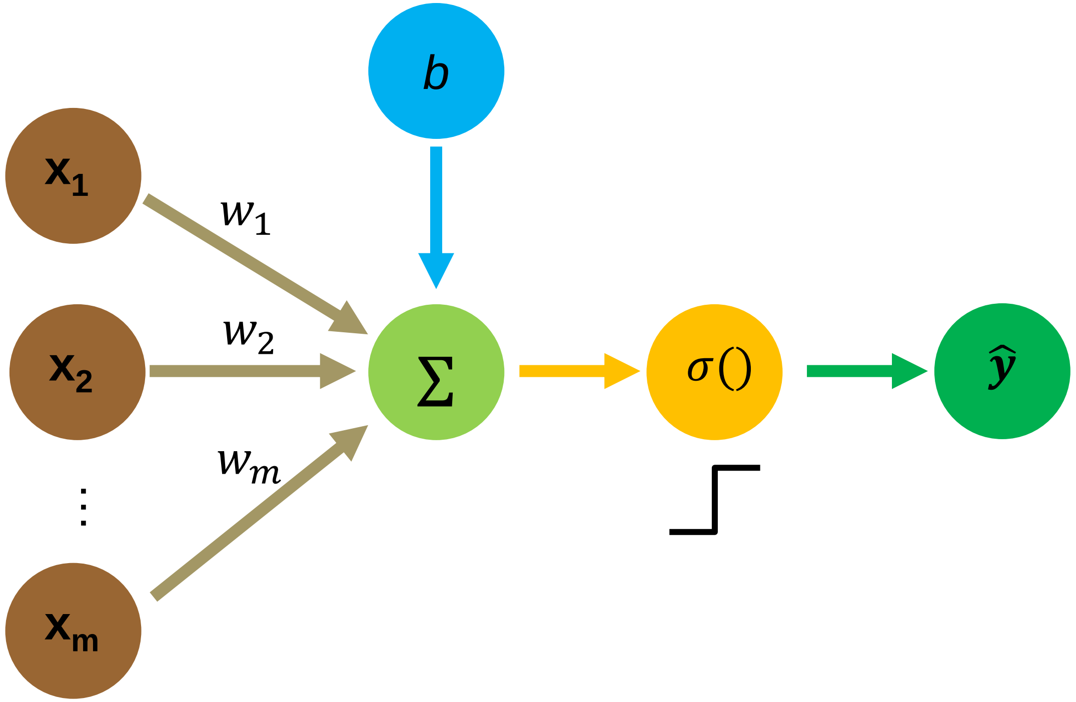
- Artificial Neural Networks for Deep Learning
- Neuronal activation functions
- Backpropagation and Gradient descent
- Deep Belief Nets



Motivation

traffic light
truck
car
person
car
car
bicycle
car
bicycle
car

The ”atom” of an Artificial Neural network



Output

Weighted sum of inputs

$$\hat{y} = \sigma\left(b + \sum_{i=1}^{m} x_i w_i\right)$$

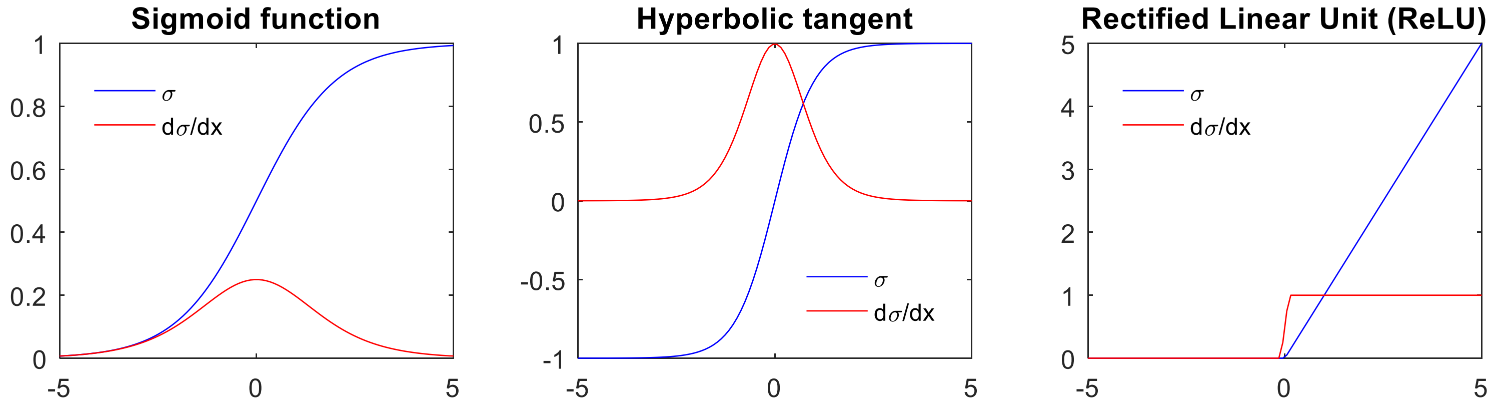
Non-linear
activation function

“Bias”

The Activation function, $\sigma(z)$

- **Q:** Why do we need an activation function? → **Nonlinearity** (allows modelling of nonlinear systems)

$$\hat{y} = \sigma\left(b + \sum_{i=1}^{m} x_i w_i\right)$$

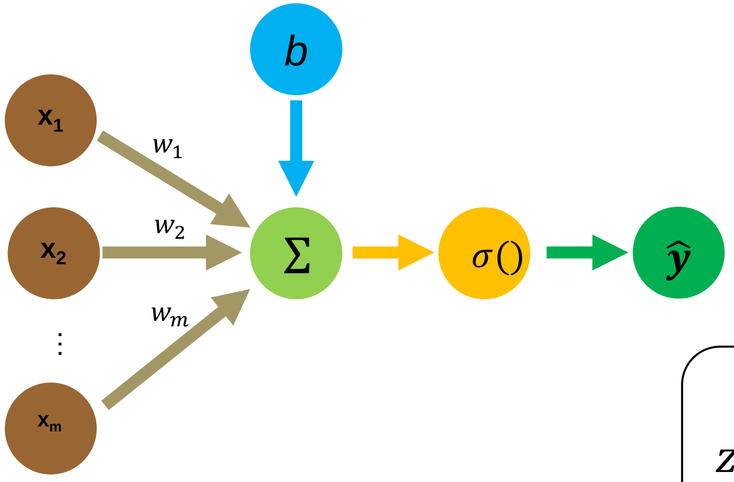


$$\sigma(z) = \frac{1}{1+e^{-z}}$$

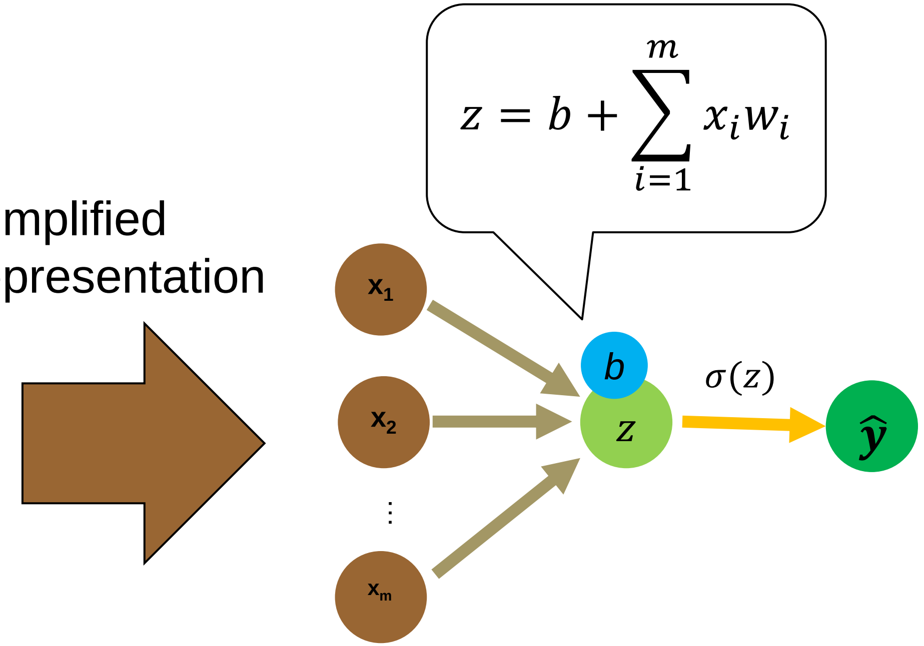
$$\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

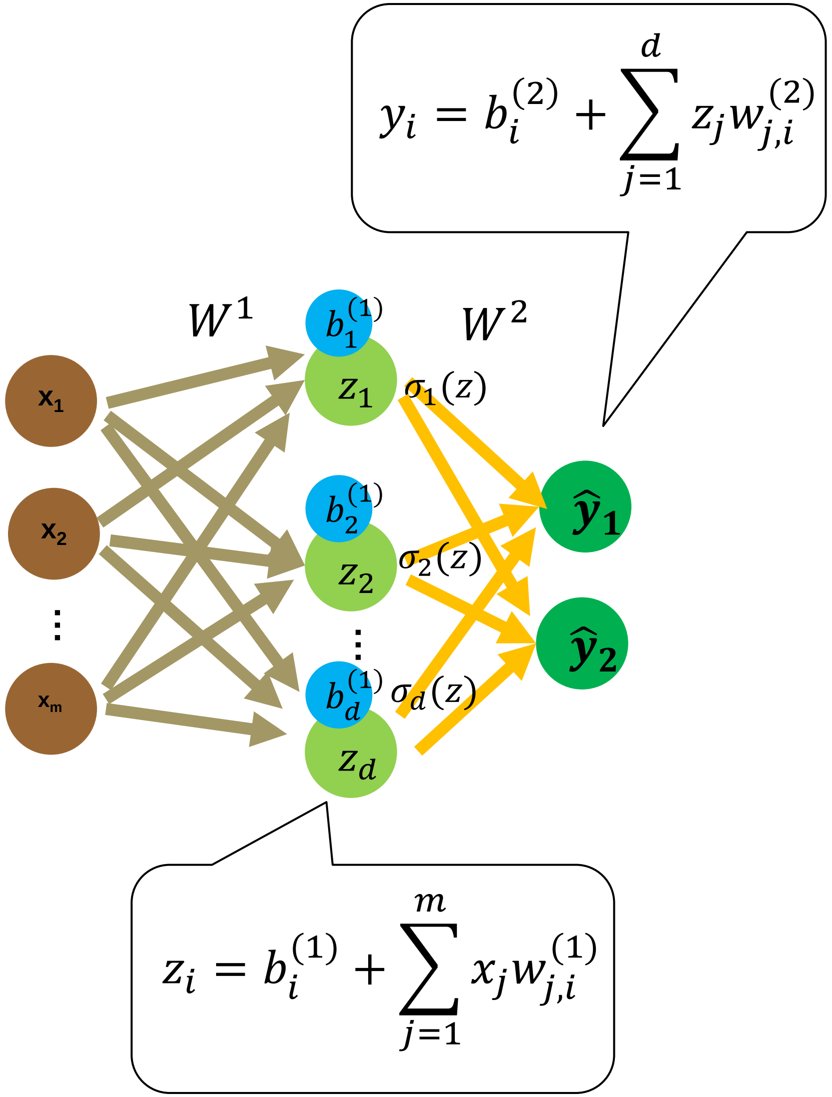
$$\sigma(z) = \max(0, z)$$

Single layer Neural network

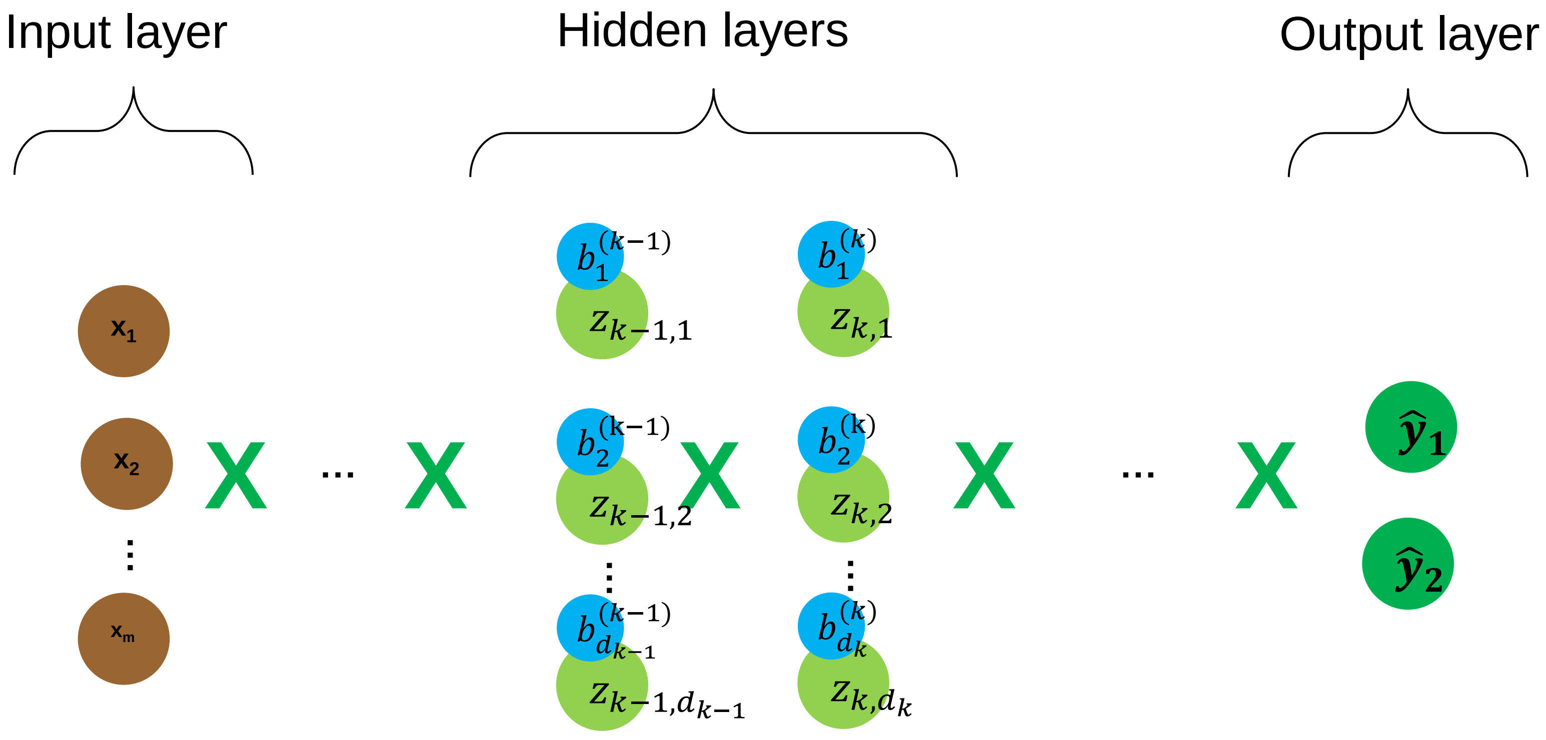


Simplified representation





Deep Neural Network

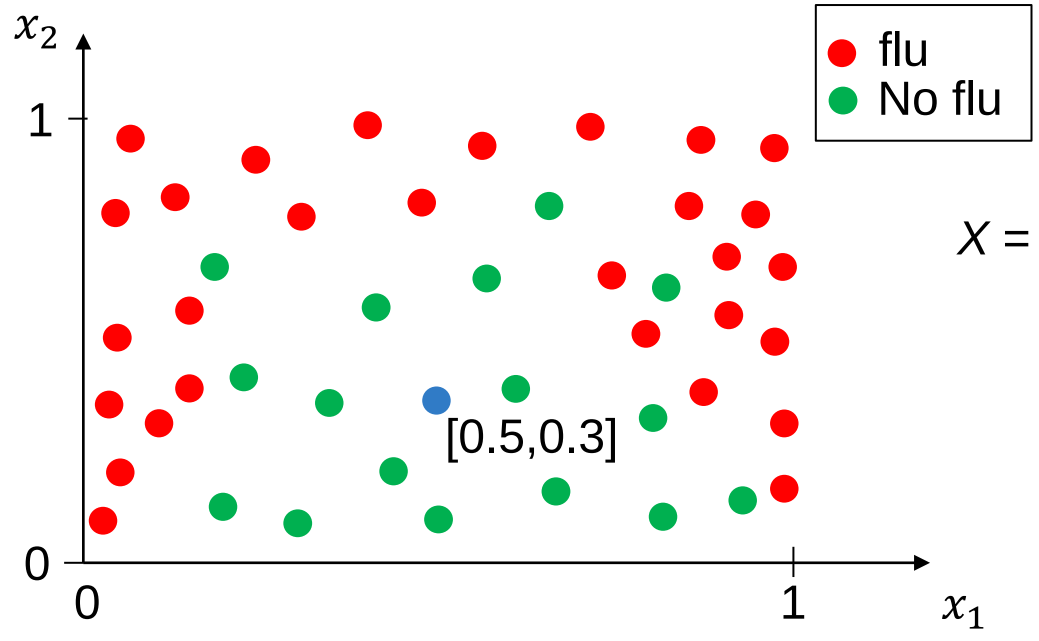


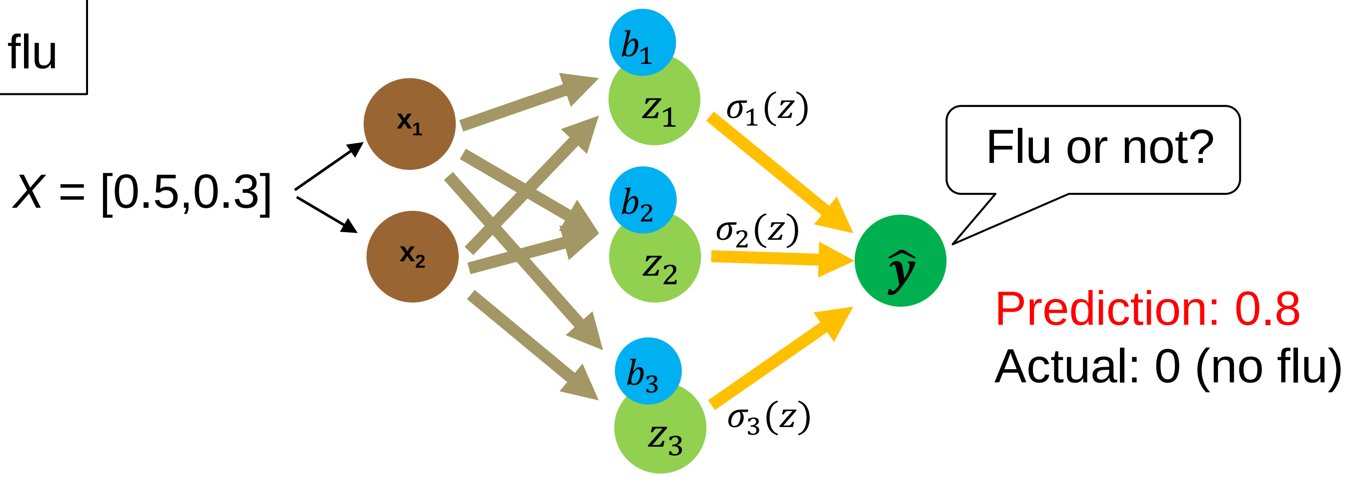
$$z_{k,i} = b_i^{(k)} + \sum_{j=1}^{d_{k-1}} \sigma(z_{k-1,j}) w_{j,i}^{(k)}$$

Q: What are the variable parameters of this network?

Example use case

- Task: Network to predict likelihood of getting the flu
- x_1: Time of the year (normalized) (i.e. 1st Jan = 0, 31th Dec = 1)
- x_2: Time spent with other people (0 = never, 1 = 24/7)

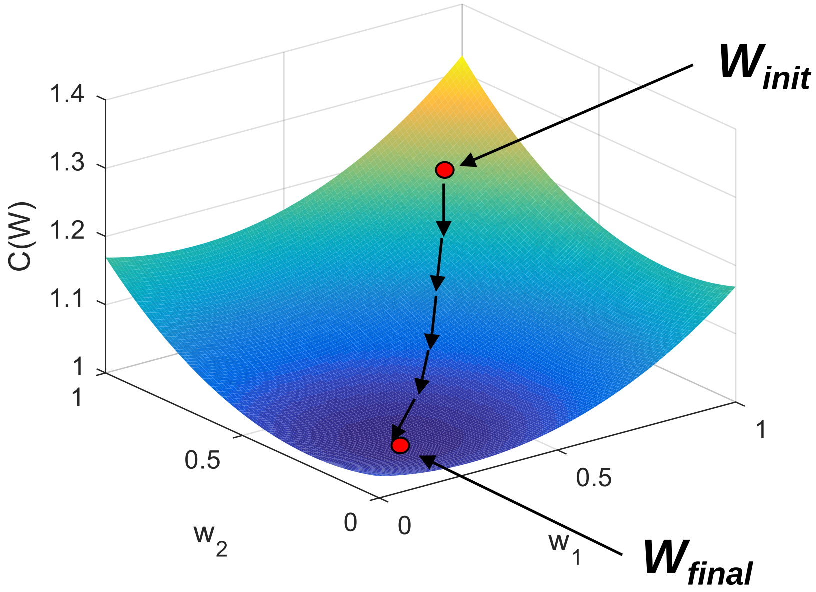




Cost functions and training principle

- Cross-entropy cost function: $C(W,b) = -\frac{1}{n}\sum_{i=1}^{n}(t \ln \hat{y} + (1-t)\ln(1-\hat{y}))$ For ***discrete*** output
- Mean Squared Error Cost function: $C(W,b) = \frac{1}{n}\sum_{i=1}^{n}\left(t^i - f\left(X^i, W, b\right)\right)^2$ For ***continuous*** output

- Training means finding the (W,b) that minimizes C for our dataset.



Learning rate

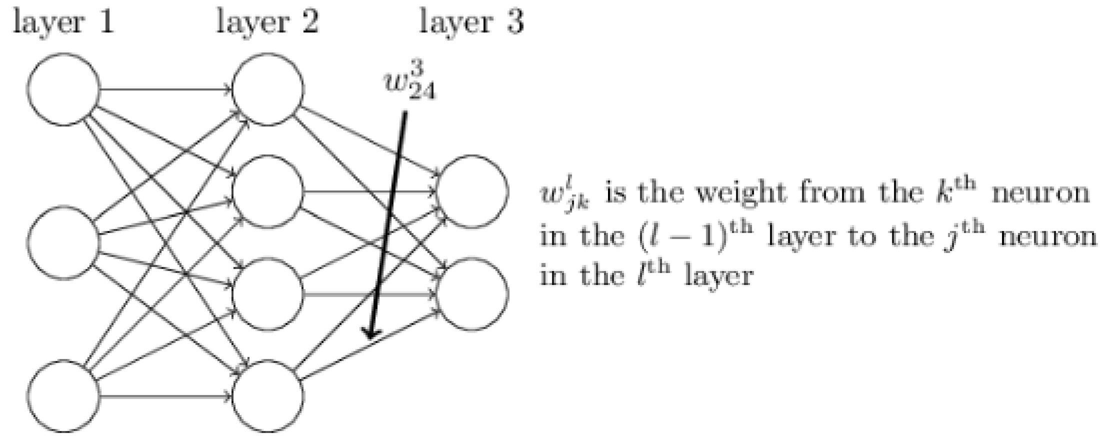
1. Initialize ***W*** and ***b***
2. *Calculate* $\nabla C(\boldsymbol{W}, \boldsymbol{b})$
3. *Update W and b according to* $W = W - \eta\nabla C$
4. *Repeat 2-3 until convergence*

Go in direction of negative gradient

So how to calculate ∇C?

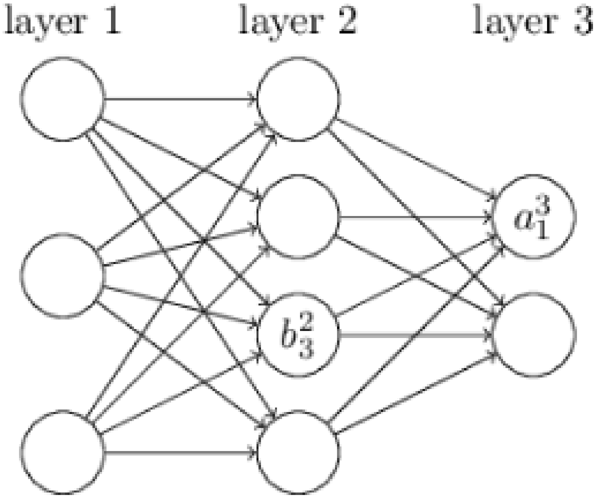
Matrix representation

$$W_l = \begin{bmatrix} w_{11} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{bmatrix}$$



$$A_l = \begin{bmatrix} a^l_1 \\ \vdots \\ a^l_m \end{bmatrix}$$

$$B_l = \begin{bmatrix} b^l_1 \\ \vdots \\ b^l_m \end{bmatrix}$$



The weighted input of the l^{th} layer:

$$Z_l = W_l A_{l-1} + B_l$$

The activation of the l^{th} layer is thus:

$$A_l = \sigma(Z_l)$$

Backpropagation algoritm

- A way to calculate ∇C

1. **Input** *X*: Set the activation $\boldsymbol{A}_1$ at the input layer
2. **Feed-forward**: for each *l* = *2,3,…,L* compute Z_l and A_l
3. **Output error:**

   Compute $C(W_L, B_L)$ and $\frac{\partial C}{\partial Z_L} = \frac{\partial C}{\partial A_L} .* \sigma'(Z_L)$ of the layer *L*

   → $\frac{\partial C}{\partial W_L} = \frac{\partial C}{\partial Z_L} A_{L-1}$
4. **Back-propagate:** For each layer $l < L$, calculate

$$\frac{\partial C}{\partial W_l} = \frac{\partial C}{\partial Z_l}\frac{\partial Z_l}{\partial W_l} = \underbrace{\left(W_{l+1}^T * \frac{\partial C}{\partial Z_{l+1}}\right)}_{\text{Looks forward}} .* \ \sigma'(Z_l) * \underbrace{A_{l-1}^T}_{\text{Looks backward}}$$

5. **Update weights according to gradient descent**

$$W_{new} = W - \eta \nabla C$$

Improving speed at calculating ∇C

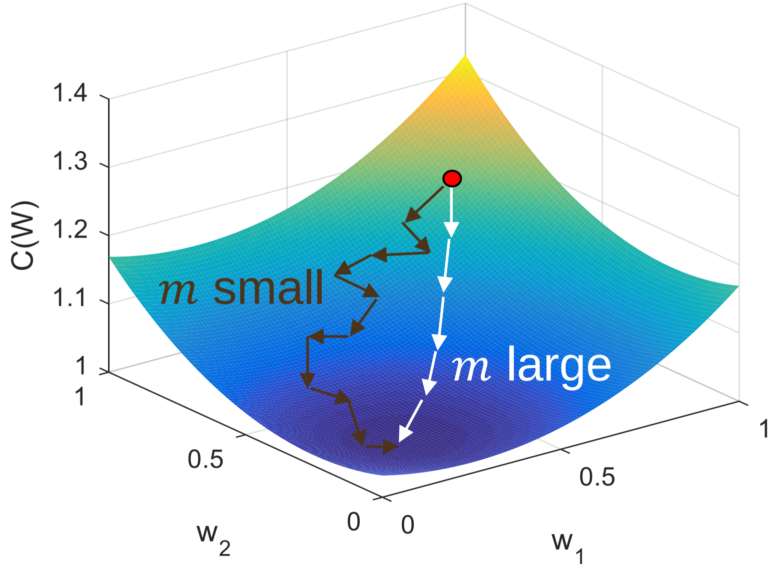
X: Complete data set (n ~ 10^5)

m	m	m	m	m	m
m	m	m	m	m	m
m	m	m	m	m	m
m	m	m	m	m	m
m	m	m	m	m	m

- Need to compute the gradient for each training example x and then average
  - $C = \frac{1}{n}\sum_x C_x \rightarrow \nabla C = \frac{1}{n}\sum_x \nabla C_X$
- With typical n > 10^4 this is extremely slow

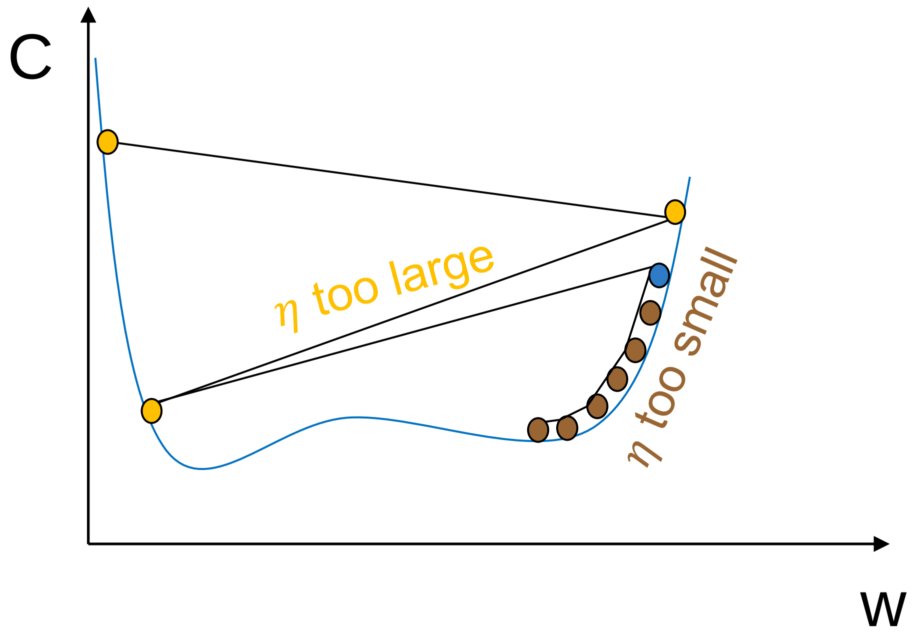
→ **Stoichastic gradient descent**

- Randomly choose sample m ~ 10^2 of X (mini-batch)
- $\nabla C \approx \frac{1}{m}\sum_{j=1}^{m} \nabla C_{x_j}$
- Train on this mini-batch, then choose a new batch until having trained with all (1 epoch)
- Then start over and redo until training is finished (100-1000 epochs)
- **Less straight path, but still can be more time-efficient!**



The learning rate

- Too small learning rate → can get caught in local minima
- Too large learning rate → Solution can diverge!
- Best solution: Adaptive learning rate based on
  - How large gradient is
  - How fast learning happens
  - Size of some weights
  - Momentum
  - …



Energy use of training by backpropagation

- On average 10 ms time for one layer operation, t_{op} (forward + backwards pass)
- Hardware NVIDIA Tesla V100 GPU: 250W
- Ex: L= 100, m = 100, N = 100 000, 100 epochs → $t_{exec} = t_{op} L \frac{N}{m} e = 100\,000\ s \sim 27h$.

→ 25 MJ energy consumption! (0.25kW * 27h = 7 kWh)

1kWh

100 days of LED light, (2.5W)

~300 full charges of your cell phone

Vanishing gradient problem

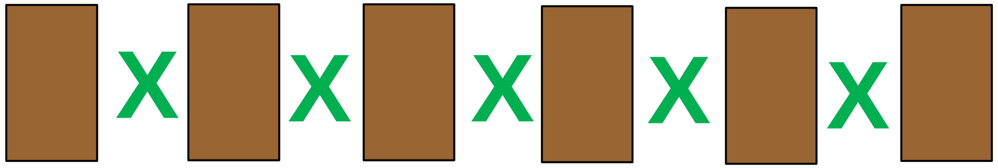
- In back-propagation:

$$\frac{\partial C}{\partial W_l} = \frac{\partial C}{\partial Z_l}\frac{\partial Z_l}{\partial W_l} = \left(W_{l+1}^T * \frac{\partial C}{\partial Z_{l+1}}\right) .* \underbrace{\sigma'(Z_l)}_{<1} * \underbrace{A_{l-1}^T}_{<1}$$

i.e. multiplication of numbers smaller than 1

→ Gradients vanish in deep networks

$$\frac{\partial C}{\partial W_i} = \quad 0.0001 \quad 0.003 \quad 0.035 \quad 0.25 \quad 0.55 \quad 0.8$$



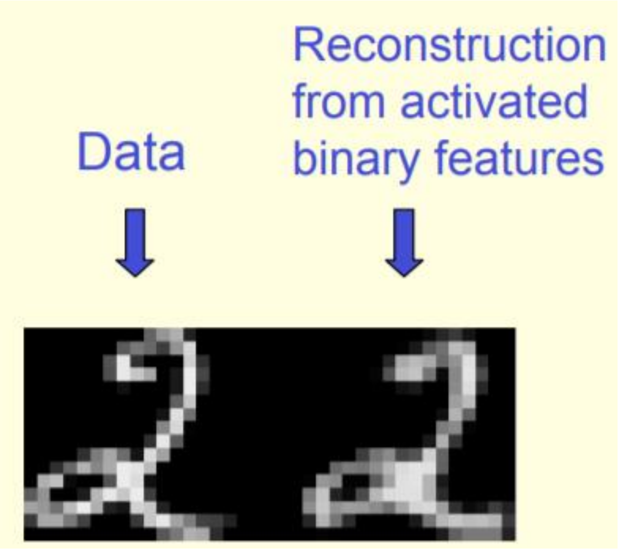
l=1 2 3 4 5 6

Deep Belief Nets

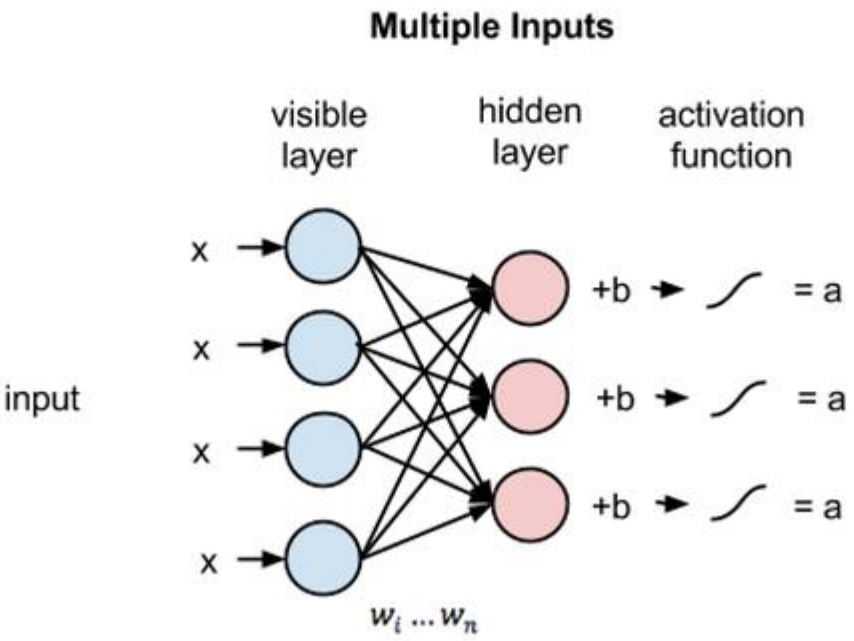
- A deep network that still avoids the vanishing gradient problem.
- NOT trained by backpropagation → unsupervised training
- Based on principles of Statistical Physics, Boltzmann statistics, probabilities, energies etc..

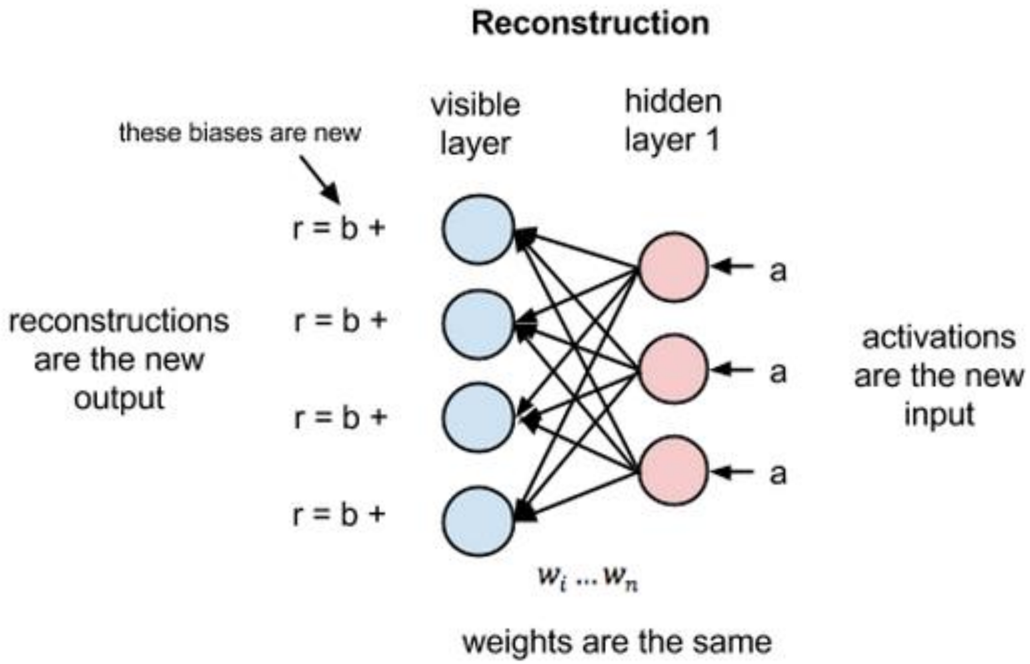
- Very brief introduction only…

Restricted Boltzmann Machines



- Two layer fully connected network with stochastic binary units
- Trained iteratively without labels by repeated (1) forward pass and (2) reconstruction pass
- Change weights as to minimize error between input and ”reconstructed” input





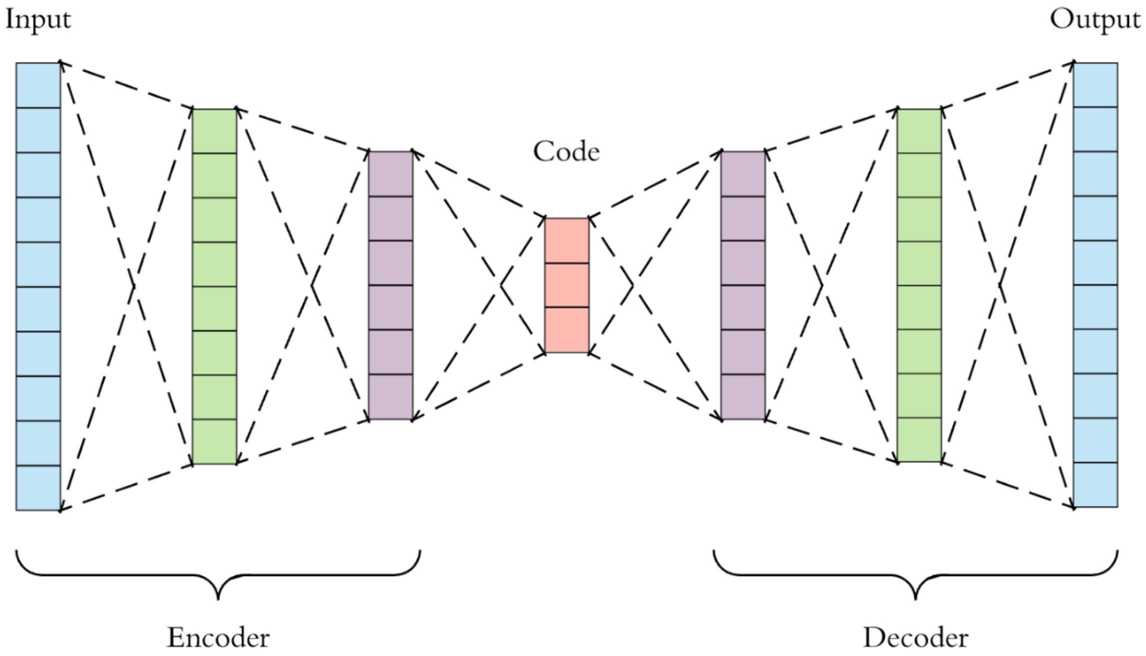
Training Deep Belief Net

- Each bilayer RBM is trained individually
- Each hidden layer of RBM is visible layer of the next
- Train each RBM separately in sequence until the whole network is trained.
  - Unsupervised learning! No labels needed
- Finally: Finetuning with small labeled dataset

http://www.cs.toronto.edu/~hinton/digitmovie1.gif

DBM as Autoencoder

- Stacking a ”mirrored” DBM on the output → generate new data!
- Trained in the same way
- Useful for tons of things:
  - Data denoising
  - Generate new similar images/video based on data set
  - Anomaly detection
  - DeepFake
  - Find similar documents (Document retrieval)



https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Summary

- Artificial neural networks – abstraction of the learning principles of the brain (matrix math)
- Free parameters: weights of connections and biases of ”neurons”
- Optimize parameter values to learn things
  - Backpropagation
  - Gradient descent
- Unsupervised learning → Deep Belief Nets
  - Generation of new data (autoencoder)

Exercise: Make your own Quiz

- Come up with a good quiz question based on the topics of L1-L3
- You have until the end of the lecture → Send it to mattias.borg@eit.lth.se
- Quiz will be posted on Canvas for you to practise on..

What is the chance that you win on the lottery?

1. Chance? I always win
2. 1 in 100 000
3. As good as dying in a plane crash
4. I will win when pigs can fly

Lecture 1 – Introduction
Memory cell

Lecture 2 – Current memories
DRAM
Flash
SRAM

Lecture 3 – Machine Learning Fundamentals
Backpropagation
Gradient descent
Restricted Boltzmann Machines
Deep Belief Networks