Neural Network

Remote Sensing Data Analysis (ASEN 6337)

Enrico Camporeale

CIRES, CU Boulder
NOAA / Space Weather Prediction Center

Contact: Enrico.camporeale@noaa.gov

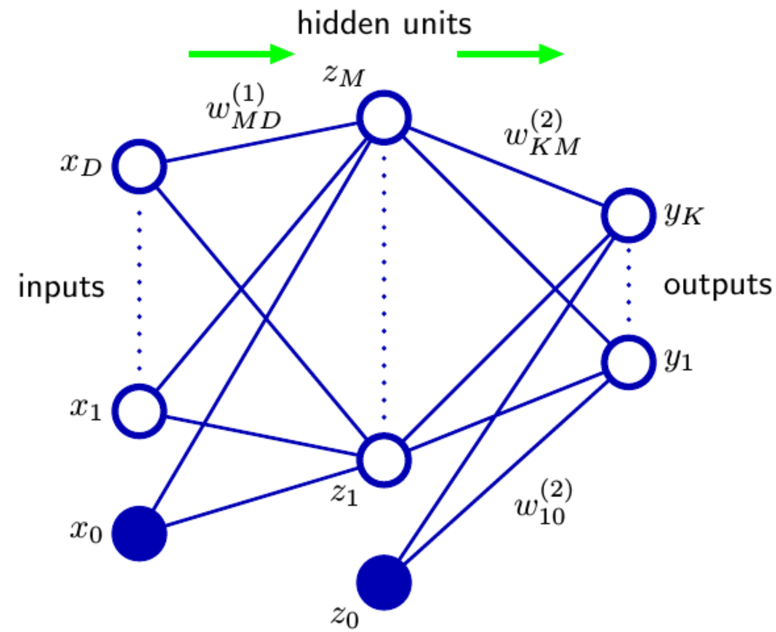
Slides and codes available on
https://github.com/ecamporeale/teaching-ASEN6337

Contents

Recap: Neural Networks

Neural Networks are universal function approximators

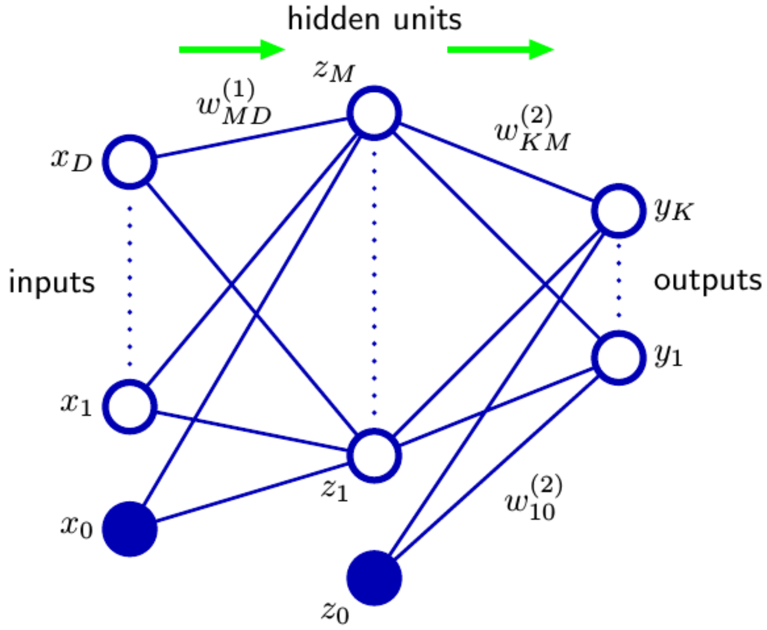
- Given enough complexity (number of neurons/depth) they can approximate any function with arbitray accuracy



$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{5.7}$$

Neural Networks are universal function approximators

- Given enough complexity (number of neurons/depth) they can approximate any function with arbitray accuracy
- However, NN is a parametric method, meaning it requires a (large) number of parameters and hyperparameters to work well:
  - Parameters are learned: weights and biases
  - Hyperparameters are chosen: # of neurons, layers, activation function, loss function, etc.

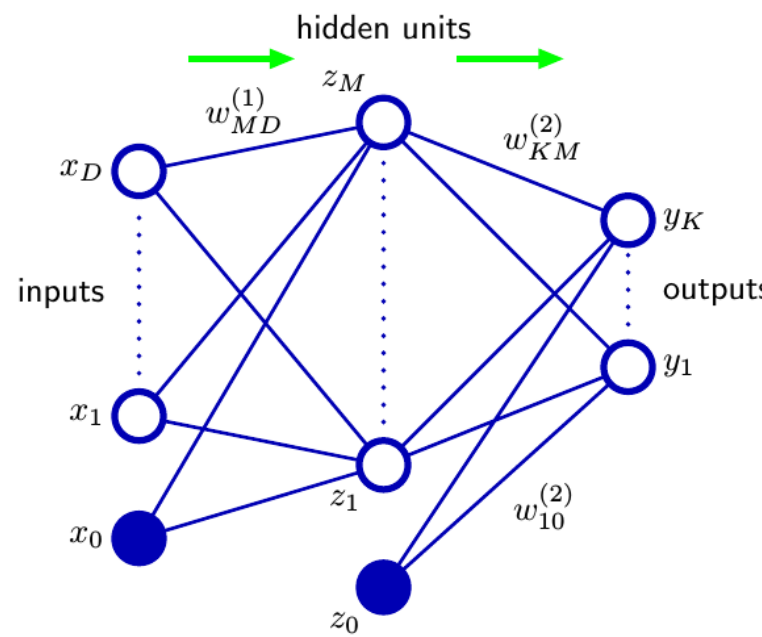


$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{5.7}$$

5

Neural Networks are universal function approximators

- Given enough complexity (number of neurons/depth) they can approximate any function with arbitray accuracy
- However, NN is a parametric method, meaning it requires a (large) number of parameters and hyperparameters to work well:
  - Parameters are learned: weights and biases
  - Hyperparameters are chosen: # of neurons, layers, activation function, loss function, etc.
- The training phase aims to minimize a chosen loss/cost function, e.g. MSE:



$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2.$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w^{(2)}_{kj}\, h\left(\sum_{i=1}^{D} w^{(1)}_{ji} x_i + w^{(1)}_{j0}\right) + w^{(2)}_{k0}\right) \quad (5.7)$$

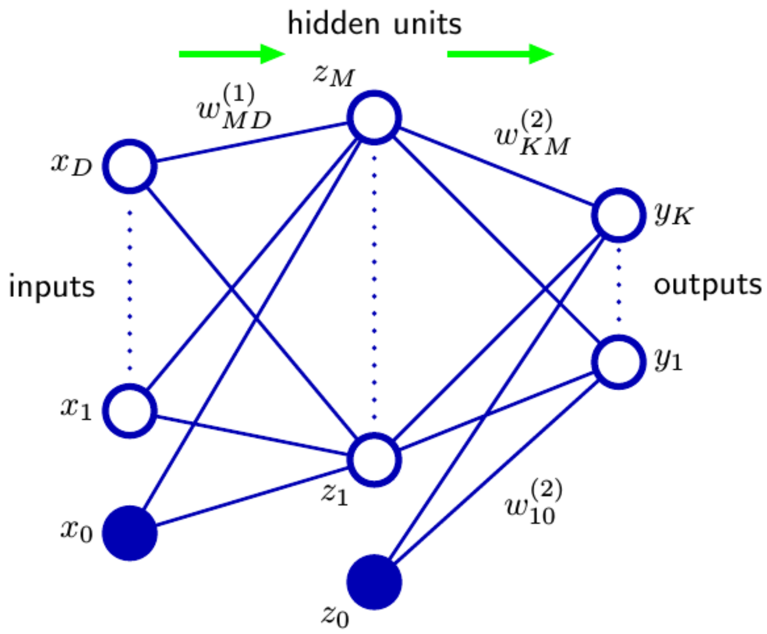
6

Neural Networks are universal function approximators

- Given enough complexity (number of neurons/depth) they can approximate any function with arbitray accuracy
- However, NN is a parametric method, meaning it requires a (large) number of parameters and hyperparameters to work well:
  - Parameters are learned: weights and biases
  - Hyperparameters are chosen: # of neurons, layers, activation function, loss function, etc.
- The training phase aims to minimize a chosen loss/cost function, e.g. MSE:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2.$$

- Training is usually performed through **backpropagation**

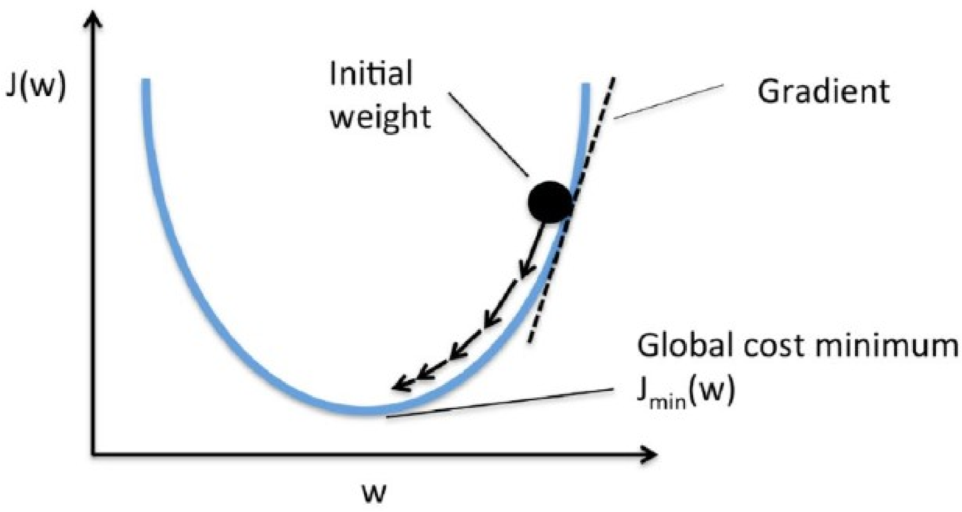


$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

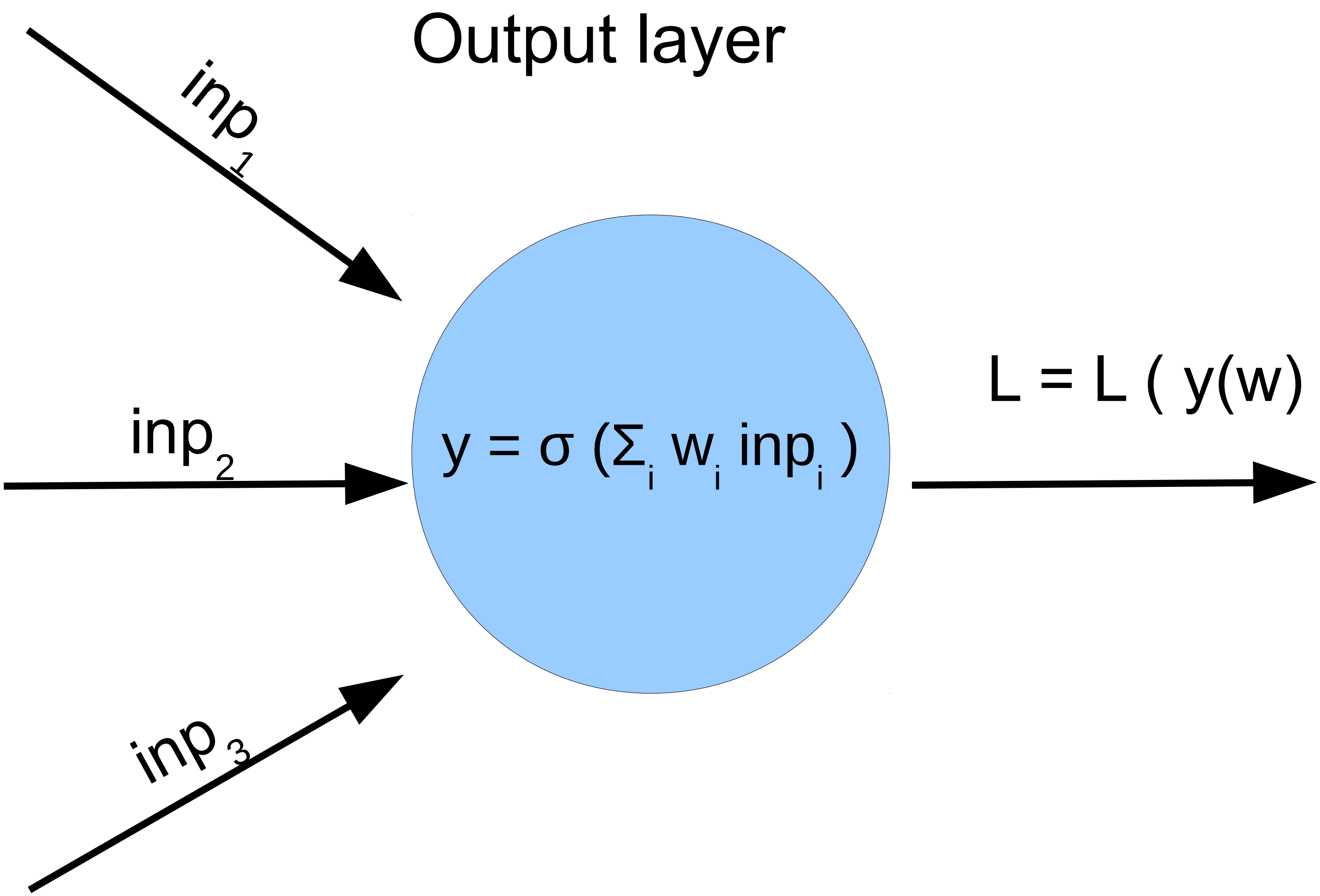
Backpropagation

- The cost function depends on the weights:
- We need to compute the derivative of the loss with respect to theweights, for instance gradient descent updates weights as:

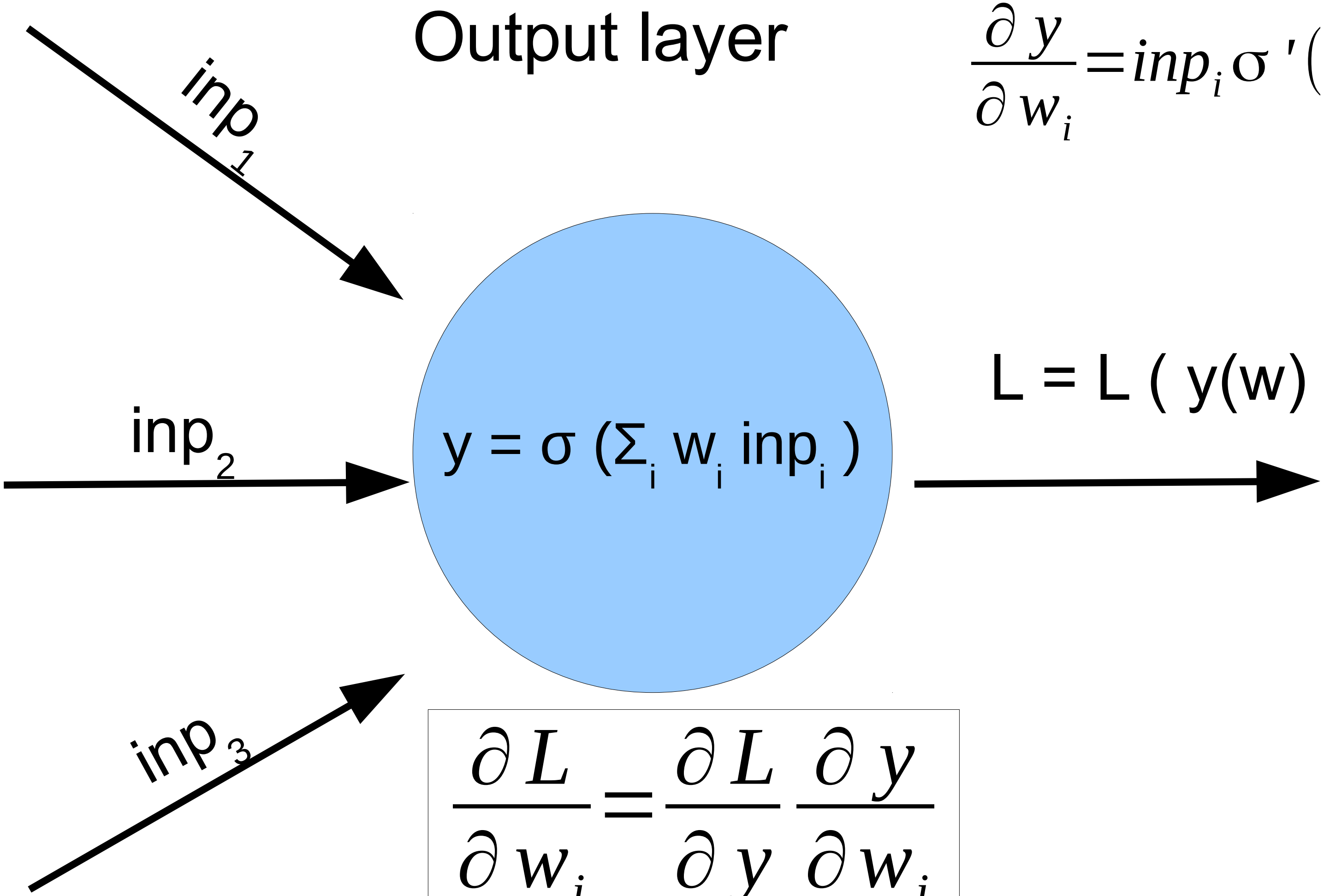
$$W_1 = W_1 - \alpha \frac{\partial J}{\partial W_1}$$



Backpropagation



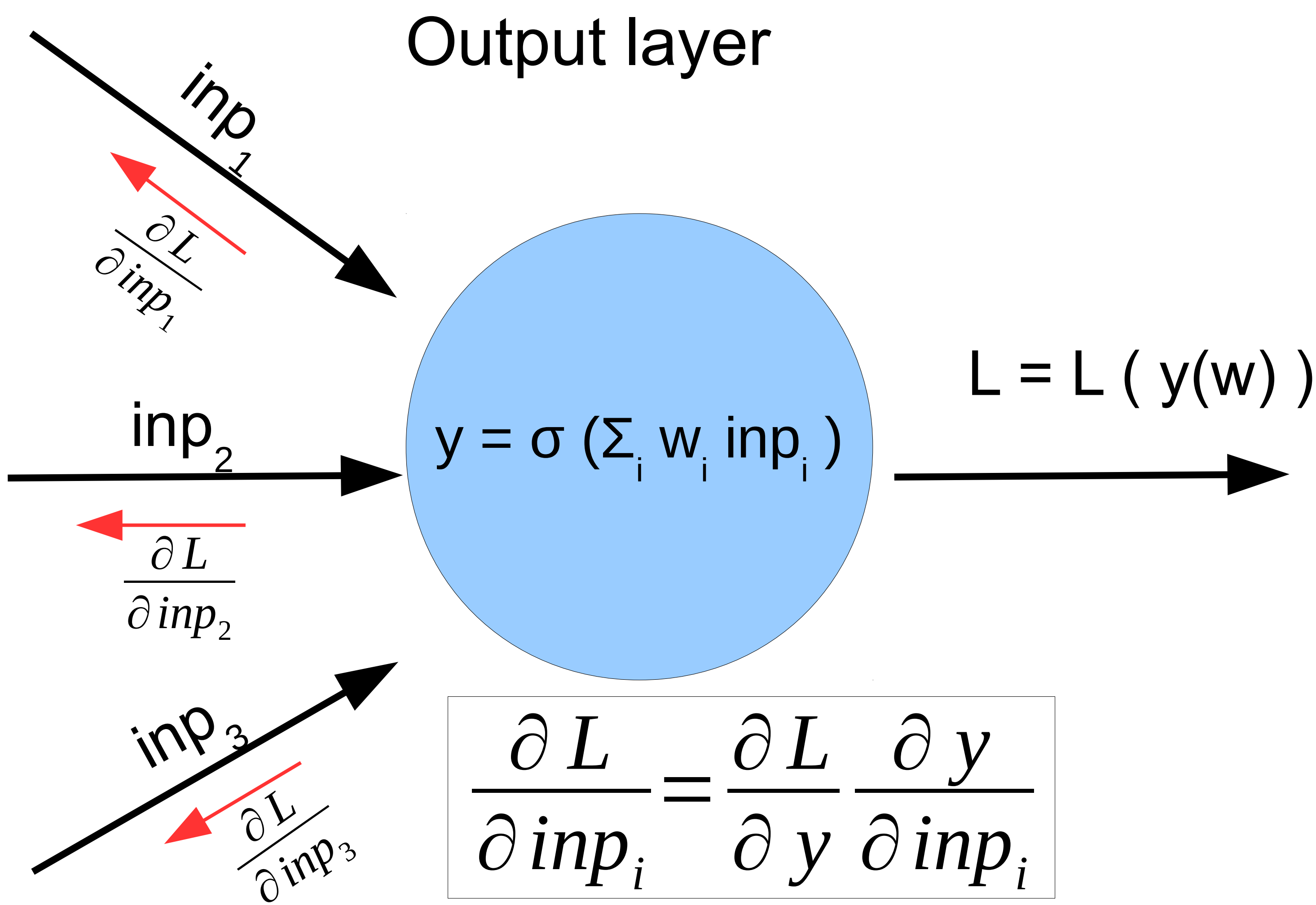
Backpropagation



$$\frac{\partial y}{\partial w_i} = inp_i \sigma'(w_i inp_i)$$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_i}$$

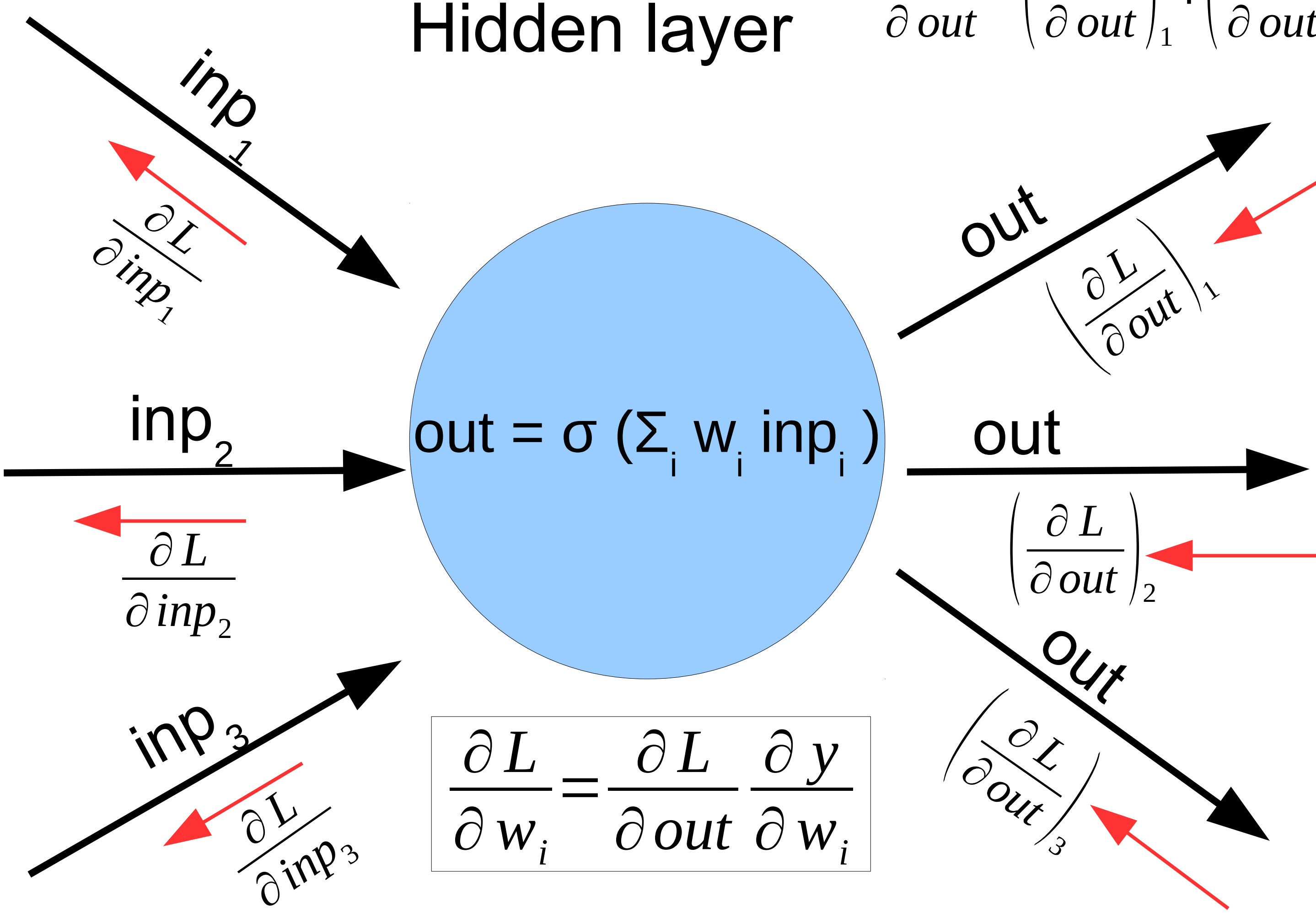
Backpropagation



$$\frac{\partial L}{\partial inp_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial inp_i}$$

Backpropagation

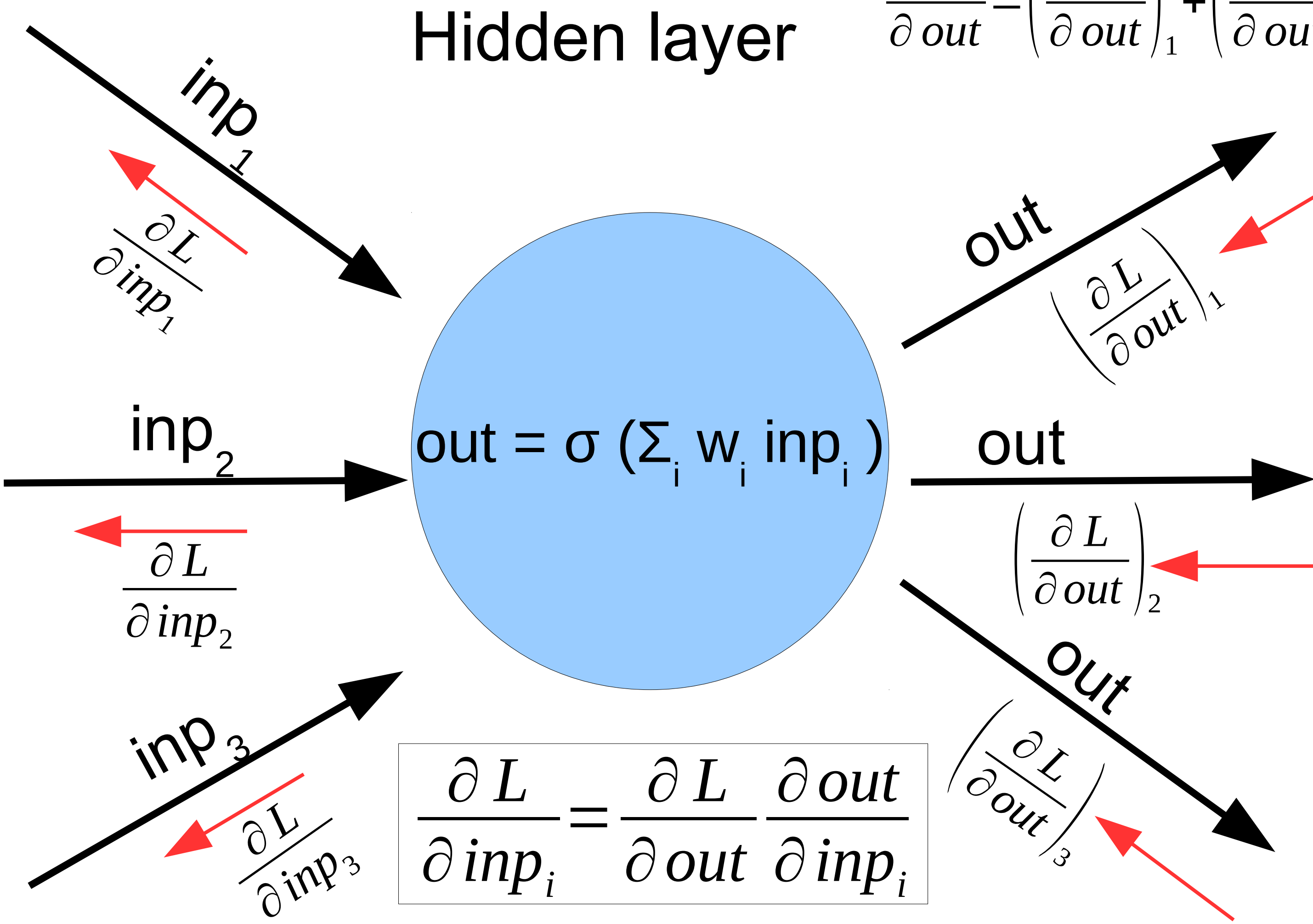
$$\frac{\partial L}{\partial out} = \left(\frac{\partial L}{\partial out}\right)_1 + \left(\frac{\partial L}{\partial out}\right)_2 + \left(\frac{\partial L}{\partial out}\right)_3$$



$$\boxed{\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial out}\frac{\partial y}{\partial w_i}}$$

Backpropagation

$$\frac{\partial L}{\partial out} = \left(\frac{\partial L}{\partial out}\right)_1 + \left(\frac{\partial L}{\partial out}\right)_2 + \left(\frac{\partial L}{\partial out}\right)_3$$



$$\frac{\partial L}{\partial inp_1} \qquad \frac{\partial L}{\partial inp_2} \qquad \frac{\partial L}{\partial inp_3}$$

$$\left(\frac{\partial L}{\partial out}\right)_1 \qquad \left(\frac{\partial L}{\partial out}\right)_2 \qquad \left(\frac{\partial L}{\partial out}\right)_3$$

$$\boxed{\frac{\partial L}{\partial inp_i} = \frac{\partial L}{\partial out}\frac{\partial out}{\partial inp_i}}$$

NN examples

https://github.com/ecamporeale/teaching-ASEN6337

- Five simple examples:

  1) Understand the role of the learning rate in a simple steepest descent backpropagation algorithm
  2) Play with different activation functions
  3) Convince ourself that a NN is an universal approximator
  4) Understand the role of the optimizer
  5) Convince ourself that a NN is not good at extrapolation
  6) Understand overfitting and how to avoid it
  7) Understand train, validation and test set

- A real-world example of geomagnetic index prediction

NN_ex1.m

% This is the simplest implementation of a Neural Network to learn its fundamental concepts
% It is a feed-forward NN with a single hidden layer
% with 1-dimensional input x and 1-dimensional output y
% The output is

% y = sum_i w2_i * f(w1_i*x+b1_i) + b2; with f the nonlinear activation function

% Purpose of exercise 1:
% 1) understand the role of the learning rate in a simple steepest descent backpropagation algorithm
% 2) Play with different activation functions

Take home message:

- Gradient descent is slow: more advanced optimizers
- Do not build your NN from scratch! Use libraries

NN_ex2.m

% 1) Convince ourself that a NN is an universal approximator
% 2) Understand the role of the optimizer

Compare two optimizers.
Optimizers available in matlab: type `help nntrain`

Take home message:

- Choose wisely your optimizer
- Catch: Levenberg-Marquardt works only with MSE and does not support GPU (in matlab). Other optimzers available in python (Adam, RMSprop)

NN_ex3.m

% 1) Convince ourself that a NN is not good at extrapolation

When the NN is trained only on portion of the domain, what happens to the function approximation in the 'unseen' portion of the domain?

Take home message:

- A NN is a function approximator. It can learn only what it sees (i.e. what it is trained on)

Overfit & Regularize

NN_ex4.m

% 1) Understand overfitting and how to avoid it

Experiment #1:
Why the NN approximation is so bad, yet the error is so small?

Take home message:

- A NN can construct any complex nonlinear function that fits the data, and it will do so, if it is not constrained to 'simpler' functions

NN_ex4.m

% 1) Understand overfitting and how to avoid it

Experiment #1:
Why the NN approximation is so bad, yet the error is so small?

Experiment #2:
Try a regularization term to constrain the complexity f the function

Take home message:

- A NN can construct any complex nonlinear function that fits the data, and it will do so, if it is not constrained to 'simpler' functions

NN_ex5.m

% 1) Understand train, validation and test set

Experiment: use early stop on validation set

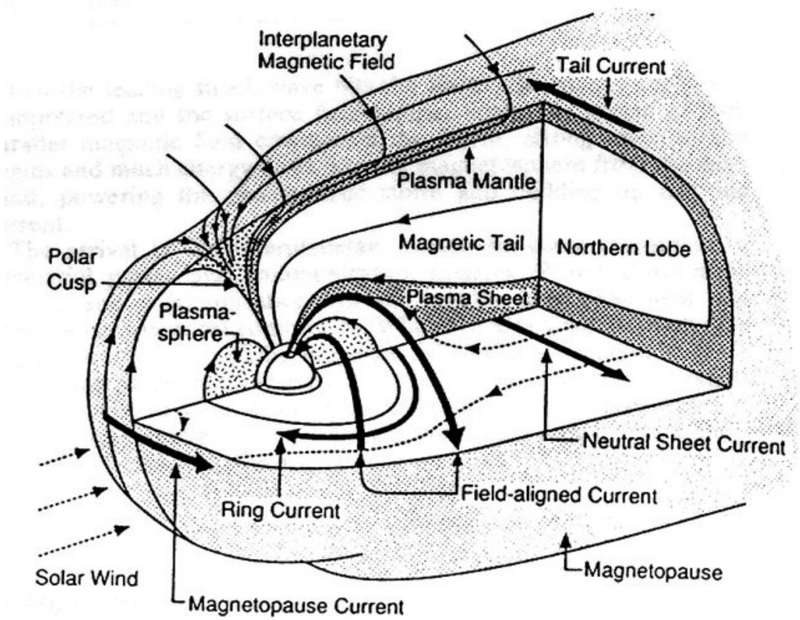
Take home message:

- A NN has no notion of noise vs signal. It will always overfit (as much as it can) the training set (i.e. fit the noise other than the signal) for a too large number of iterations

Space Physics Example

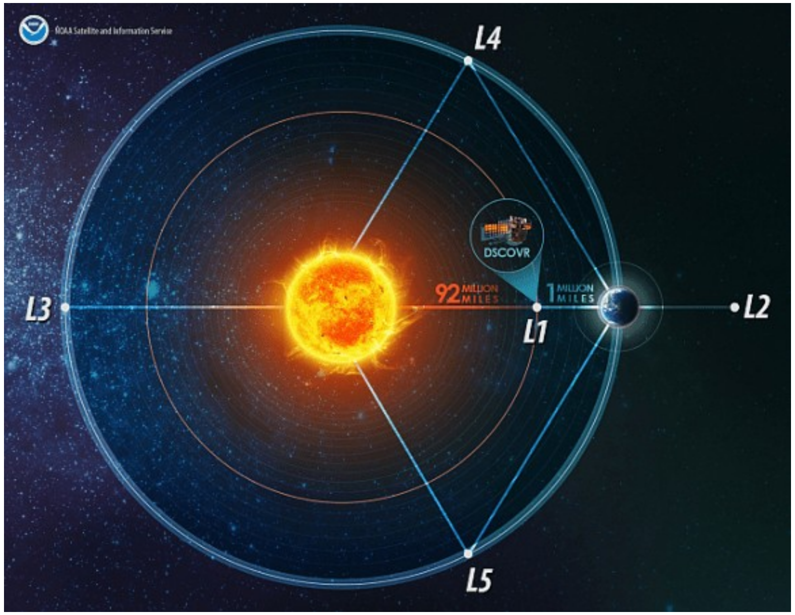
The Disturbance Storm Time (DST) index

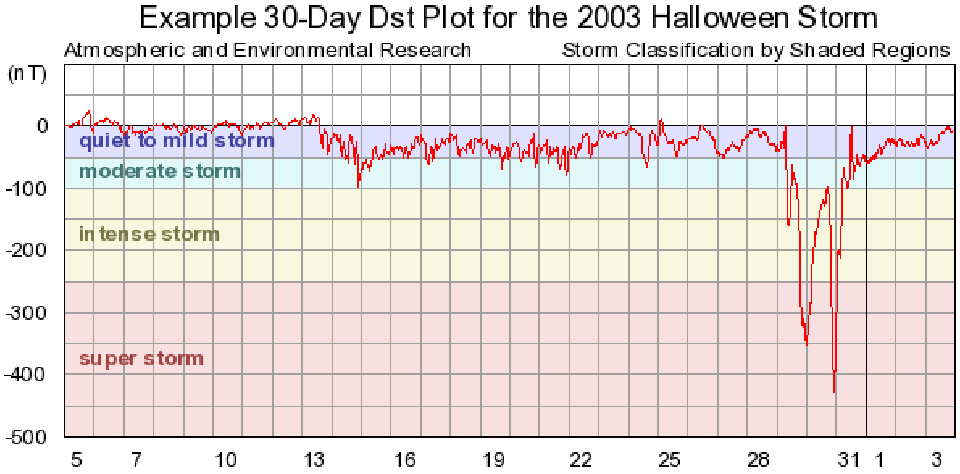
- DST is an index of geomagnetic activity that measures the intensity of the ring current
- DST is measured by a network of ground-based magnetometers at equatorial latitude
- DST has been archived for a few decades
  For instance on the OMNI dataset:
  https://omniweb.gsfc.nasa.gov/



DST prediction using OMNI data

- The task is to predict DST (a certain number of hours in advance), using measurements recorded from spacecraft at the 1st Lagrangian point





NN_predict_DST.m

- Load the OMNI dataset (1967 – 2007)
- Decide what quantities to use as inputs
- Decide the timelag for prediction
- Prepare the input/output data
- Decide the NN architecture
- Train
- Evaluate results

Homeworks !

- Rewrite the codes in Python (the only way to learn a code...write it yourself)

- Simple problems: experiment different architectures, different optimizers, different hyperparameters

- DST problem:
  - Does it always work so well (quite-time vs storm-time)?
  - What is the most important input (and how do you figure that)?
  - Can you think of other ways to evaluate the accuracy?
  - How many hours in advance can you predict?
  - Any strategy to further improve the prediction (modern architectures, LSTM, Dropouts, etc.)

Homeworks !

- Rewrite the codes in Python (the only way to learn a code...write it yourself)

- Simple problems: experiment different architectures, different optimizers, different hyperparameters

- DST problem:
  - Does it always work so well (quite-time vs storm-time)?
  - What is the most important input (and how do you figure that)?
  - Can you think of other ways to evaluate the accuracy?
  - How many hours in advance can you predict?
  - Any strategy to further improve the prediction (modern architectures, LSTM, Dropouts, etc.)

Send me the python code (if it works!)
Challenge your colleagues to get the best DST prediction (and send it to me!)