Deep Neural Networks

# Mandag 23.08 – Introduction

**Information**

Neural networks and deep learning combined into one course title, but these are subtly different.

New starting time: 09:15 each Monday xD

Normal: 2-3 hours of lecture +exercise

Weekly, mandatory exercises: 25% of grade

* Can submit at end of semester

Exam is a project, 1-3 group members

**Inspiration behind DNN:**

* Brain has neurons
  + Either on or off
  + Propagation of information
  + No use to replicate human brain exactly – we want better

**How to tell if an algorithm is good?**

* What does it mean if something works well?
* Mathematical approach
* Overfitting: Algorithm “memorizes” the training data
* Time data: Mean squared error (How well does the model fit reality for each case?)
* Classification: Accuracy – Correct predictions out of total number of predictions
  + Precision: TP/TP+FP – when you make a prediction, how good is it?
    - Stock market
  + Recall: TP/TP+FN – how much of the real life system can we find
    - Finding sick people
* F1 score = 2\* (precision \* recall) /( precision + recall)

Training phase:

Data set 🡪Feature extraction🡪 Machine learning model

Prediction phase: V

New Data🡪Feature extraction 🡪 Classify 🡪Output

**Supervised learning**

Define groups in data (any animal darker than X, longer than Y is a dog, etc.)

* Getting a loan at the bank
* Predicting diseases

**Unsupervised learning**

Algorithm finds grouping in data, without “supervision” – user definition

* Networking; whoever talks to each other often should have priority
* Mail: Which letters should go with the same mail van etc.

Overlearning example: Learning a subject

Good :learn the subject 🡪 do well on trial exam 🡪 do well on real exam

Overfitting: memorize answers on trial exam (AAABCCDEEBACD) 🡪 fail real exam

**Exercise: Classifying data**

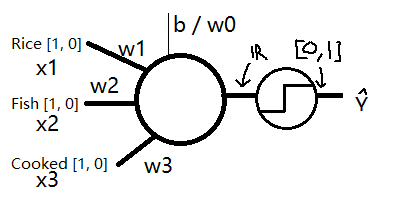
K nearest neighbours

Grouped data -> classify

New data -> compute distance to existing data, give prediction of class

# Mandag 30.08 – Neural Networks and Perceptron

* Tightly connected to next week’s lecture
* Today’s assignment: Make your first neural network
* Y is the common symbol for true value, y^ is a prediction
* Sushi example: Neuron that uses three values to predict if something is sushi:



* + x1, x2, x3 are different input values
  + w1, w2, w3 are different weights – added significance to certain inputs
    - input + weight is called a synapse
  + Sushi would produce the following x-values: [1,1,0]
    - Spaghetti: [0,0,1] (True negative)
    - California rolls: [1,1,0] (false positive?)
  + Y^’ is calculated as follows: sum(1,3)xi\*wi
    - Multiply each input by its weight, sum the products
    - Is Y^’ above a threshold 🡪 sushi
      * Else: Not sushi
    - Example weights for sushi: [0.9, 0.5, -0.2}
  + A network of such neurons is called a neural network
* More notation:
  + X’ = [x1, x2, x3]T
  + W’ = [w1, w2, w3]
  + Is W’X’ > b
    - b is threshold value
    - aka bias
    - sometimes called w0
  + Y^ = W’X’ -b > 0
    - Also called an activation function
    - In image: a step function (not most common)
    - g(z) = 1 if z>0, else 0
    - Y^ = g(wx)
  + X = [x1, x2, x3, 1]T
  + W = [w1, w2, w3, b]
  + WX = sum(xi\*wi +b\*1)>0
* Example calculations: Fried rice
  + Rice: yes
  + Fish: no
  + Cooked: yes
  + X = [1 0 1 1]T
  + W = [0,9 0.1 0.2 -0.8]
  + WX = (1\*0.9 + 0\*0.1 + 1\*0.2 + 1\*-0.8) = 0.3
    - Is 0.3 above 0?
      * Y^ = 1
      * Indicates sushi, but since fried rice isn’t sushi, weights are not perfect
* Step function: has a “limbo” value at threshold value
* Instead: Sigmoid function (S-shaped curve): mathematical value
  + T(z) = 1/1+e-z
  + Same in practice, but more correct
  + PyTorch only includes sigmoid; step is not default
* Example 2: Is an object a rock or biological material? – Code on canvas
  + X1 (sensor 1, weight w1 = 0.2)
  + X2 (sensor 2, weight w2 = -0.2)
  + Bias w0 = -0.1
  + W = [-0.1 0.2 -0.2] (Note: Bias at start, not end)
  + X = [1 0 1] (Sensor 1 off, sensor 2 on, and 1 to pair with bias)
  + WXT = (-0.1\*1) + (0\*0.2) + (1\*-0.2) = - 0.3
  + Activation: Y^ = g(wx) = g(-0.3) = 0 (value not above 0)
* Example 3: House prices in area; multi-layered NN, code on canvas:
  + X1 and X2 are closely related, as are X3 and X4.
  + X1 (Crime rates, weight w1: )
  + X2 (Tax, weight w2: )
  + X3 (Age weight w4: )
  + X4 (Pupils/Teacher, weight w5:)
  + One neuron for X1+X2, one for X3, X4
  + Neurons 1 and 2 form X’s for Neuron 3
    - Neuron 1 has bias w0
    - N1 represents X1 and X2, has weight w7
    - Neuron 2 has bias w3
    - N2 represents X3 and X4, has weight w8
    - Neuron 3 represents house prices, has bias w6
  + N1 and N2 are considered **hidden layers**, Network takes X1,X2,X3,X4, returns 1 value, no need to “know” the inner workings to use.
  + If all inputs are connected to the subsequent layer, it is a “dense network”
  + Weights are expressed per layer:
    - W[1] or W[2]
    - W1 = [w0 w1 w2 0 0 0 |  
       | 0 0 0 w3 w4 w5]

# Mandag 06.09.21 – Stochastic Gradient Descent through Backpropagation

* **Wn+ = Wn – lambda dE/dWn**
* New weight = old weight – learning factor (lambda) times influence of error
* Consider linear regression; finding a line that fits multiple points.
  + Has equation y=ax+b
  + But we choose a random b, measure sum of squared diff
    - A is typically known
    - F(x) gives value, compare to “real” point, measure square of difference
      * (Yn-Y^n)2 with Y^ being prediction, F(x)
    - Sum these differences
    - AKA a loss function
  + Plotting b against error should show which b value gives lowest error
  + The option is gradient descent, which is the topic for today
* Sum of squared errors, SS1
  + Derivative of SS1, dSS1/db, gives us the direction
    - Each squared error is derived
      * (Yn-Y^n)2 becomes -2(Yn-Y^n)
      * Becomes sum of derivatives
  + New B = old B - step size
  + Step size = slope (derivative above) \* learning rate (lambda)
    - Lambda is a hyperparameter (we choose it)
      * Higher lambda means effect of results is larger, but learning is faster
      * Typical values are 0.1/0.01/0.001
    - Because derivative gets sign based on slope, this b will converge to local minimum error
* For neural networks, instead of finding b, we find weights.
  + 1. Guess b (in this case, weights) – randomize between 0 and 1
  + 2. Compute outputs and check with data how bad our guess is, MSE
  + 3. Change weights depending on how they impact the error
    - dEtotal / dwn
    - Partial derivative of total error with respect to a given weight
    - Same as:
    - dEtotal/dAn \*dAn/dOk \* dOk/dWn
      * Product of:
      * partial derivative of total error with respect to neuron with weight we guessed
      * partial derivative of neuron with neuron it influences
      * partial derivative of output with respect to weight we guessed
    - For each, setup formula for calculation and derivate, usually fine
    - **Wn+ = Wn – lambda dE/dWn**
      * New weight = old weight – lambda \* this derivative
        + Lambda = learning factor = we choose
* This approach is called gradient descent
* Stochastic involves looking at a random selection of data
  + Can optimize for different groupings of data, never getting full picture
* In practice: Minibatch GD
  + Taken N samples
    - Idea: N should be larger than largest grouping of data if possible, to force mix
* Task: Heartbeats into normal/abnormal

# Mandag 20-09-21 – Convolutional Neural Networks

* Classify pictures
* Ex: Erna Solberg posted picture of Syrian girl during bombing, got banned for “nudity”.
* Ex 2: Mars Rover detects rocks so it doesn’t crash 😊
* How to tell if an image contains an X or an O?
  + Check for patterns of pixels, detect enough and the program “votes” for X or O
    - Also known as filters
    - Each pass of filters is known as a convolution
* Assume image of an X – pixels in a matrix [], 1 for white, -1 for black
  + Introduce a filter, which is a matrix of the pattern we’re looking for
    - They are learned with stochastic gradient descent
  + When applying the filter, multiply the filter values with the image values
    - Output is a new matrix where the new pixel value is the average of the products
      * Full hit (100% match) will have 1
      * Full miss (0% match) will have -1
    - Output matrix of a 4x4 image with 2x2 filter is 3x3 – how many positions can the filter have within the image?
      * 1 filter, 1 stride (distance moved per calculation), 0 padding (no added 0’s so output is smaller than input)
  + Output is passed to activation function (some variant of Relu)
    - Relu = 0 if x below 0, f(x) if x above 0
  + Activation becomes list of filter “Hits”.
  + Activation is sent to pooling – taking max value from region of image, then reduce image size
    - Keep only relevant information to send to further layers
  + Max values are flattened (put into a single row)
  + Flattened values are linearly added to a dense layer, followed by an activation function
  + Essentially, output from filter is used as activation of neurons in first layer, modified by learned weights to second layer (X or O), which in turn give a verdict
  + Cool activation function: Softmax – gives values between 0 and 1 that sum to 1
    - Each value is divided by the total of all values
    - “How much of the total is represented by each value?”
* RGB image: Three matrices, how do we process it?
  + Run it three times, once per color layer

# Mandag 27-09

* More CNNs
* Bigger networks intuitively should have better performance
  + Actually not the case; more params don’t improve performance
  + Image detection/big data: Something called a vanishing gradient
    - Something that is learned disappears
    - Bigger networks increase the chance of this
    - “Koker bort I kålen”
* Solution: Resnets (Residual net)
  + Residual layer
    - Data from prev layer X
    - Dense layer
    - Relu activation
    - Another dense layer
    - Output
    - Nothing new yet
    - New: X is added at output as an “Identity”
    - Whole layer is f(x), and the output is f(x)+x
  + Key idea: Keep information from before a convolution to avoid it getting lost.
* Different solution: Inception
  + Inspired by “We need to go deeper”
  + Even bigger networks!
  + Instead of doing a 1x1, 3x3, 5x5, 3x3 pooling, etc. convolution, we do all at once
  + Whatever comes out of the convolutions, we’ll let the network choose what is important later.
  + Can stack “towers” – sets of parallel convolutions
  + Can extract with softmax after any convolution
    - Errors are only propagated back to the last point of extraction
    - Can run different parts at different times; does not require single run
* Dense network
  + Same idea as resnet; push data from start with each step
  + Can get messy!
* Squeeze and excitation
  + Fire modules
  + Significantly less params
  + Loses a tiny bit of accuracy
  + Can run each block separate
  + Good for speed
  + Downsample late in the network
* Object detection:
  + Landmark detection:
    - A person has multiple facial features; eyes, nose, mouth, etc.
    - A Neural network is trained for location of eyes – gets points around the eyes (Inner/outer, upper/lower, etc)
    - Output is points instead of presence of classes (from classification)
    - Can also do classification with location:
      * Gives probability of the presence of a class, as well as position of landmark points, height, width, of the thing.
      * Output is vector with p, x, y, h, w, c1, c2, c3, c4
        + Cn represent different classes, but p is probability of a given thing
    - Loss is usually calculated with MSE of each vector point
      * Makes no sense for things not in the scope
      * If P is above a threshold value, do MSE as normal
      * If not, only do MSE on first value (probability)
    - Limited to having only one object in the image
      * Breaks down with multiple cars for example.
    - Might detect same object multiple times
      * Filter out boxes with large overlap

# Mandag 04.10 – Autumn Break

# Mandag 11.10 – Recurrent Neural networks

* From before: feed-forwards from inputs to the output
* Whenever something changes over time (conversation, stocks, music) – Recurrent networks are best
* Essentially, there are “sideways” connections between hidden layer neurons in some way
* Assume a dinner plan:
  + Pizza – Taco – Sushi – Pizza – Taco – Y^
  + What comes next? – Sushi
  + How did we know? We saw a pattern over time
  + Two uses: Classification (What is this thing about) and Generation (What comes next)
  + How do we represent these things to the neural network?
    - Numbers? Pizza 1, Taco 2, Sushi 3 is bad, implies pizza + taco = sushi
    - Bitwise: Pizza 001, Taco 010, Sushi 100 allows to represent each in a separate category.
      * Problem 1: Really long lists of items is problematic due to the size (a thousand 1’s and 0’s)
      * Problem 2: Does not account for order where important
        + I like pizza not taco
        + I like taco not pizza
        + Same words, opposite meanings.
    - Embedding:
      * Send input to a NN
      * Reduce layer size down to a small number (6 down to for ex. 2)
      * Then increase size again
      * If input (X) is the same as the output Y^, then we know that on the central step of size 2 “knows” the input
      * The small step is called an embedding
      * Strong representation of the data
      * Embedding comes out as a vector
        + Can do math on embedding vectors
        + Example:
        + Embedding vector for King – embedding vector of man + embedding vector of woman = Queen
        + Tokyo – Japan + France = Paris
        + Chicken + Sweet = Kitten (What is “sweet” than a chicken?)
        + MD – man + woman = Nurse
      * In pyTorch: Can implement on your own
        + In practice: Just add an embedding layer
* Some small network
  + Previous information: ST-1
  + ST-1 is sent into a network, combined with an input, outputs ST
    - Example: Send the word “Luke” to a network, output should be “I”
    - “I” is sent in, output should be “am”
    - Etc.
* Code example will have structure:
  + Input
  + Embedding
  + Recurring Neural Network RNN
  + Output

Bigger example

* Pizza [0 0 1]
* Taco [0 1 0]
* Sushi [1 0 0]
* Matrix A = Three foods stacked vertically
* A\* PizzaT \* A[1 0 0] = [0 1 0] = TacoT
* If sunny: Same food as Tuesday  
  Else: next food
* Food from last time as prev information ST-1, weather as input.