Introduction to Neural Networks

RIT SCG Talk
October 21 2019

Marko Ristić

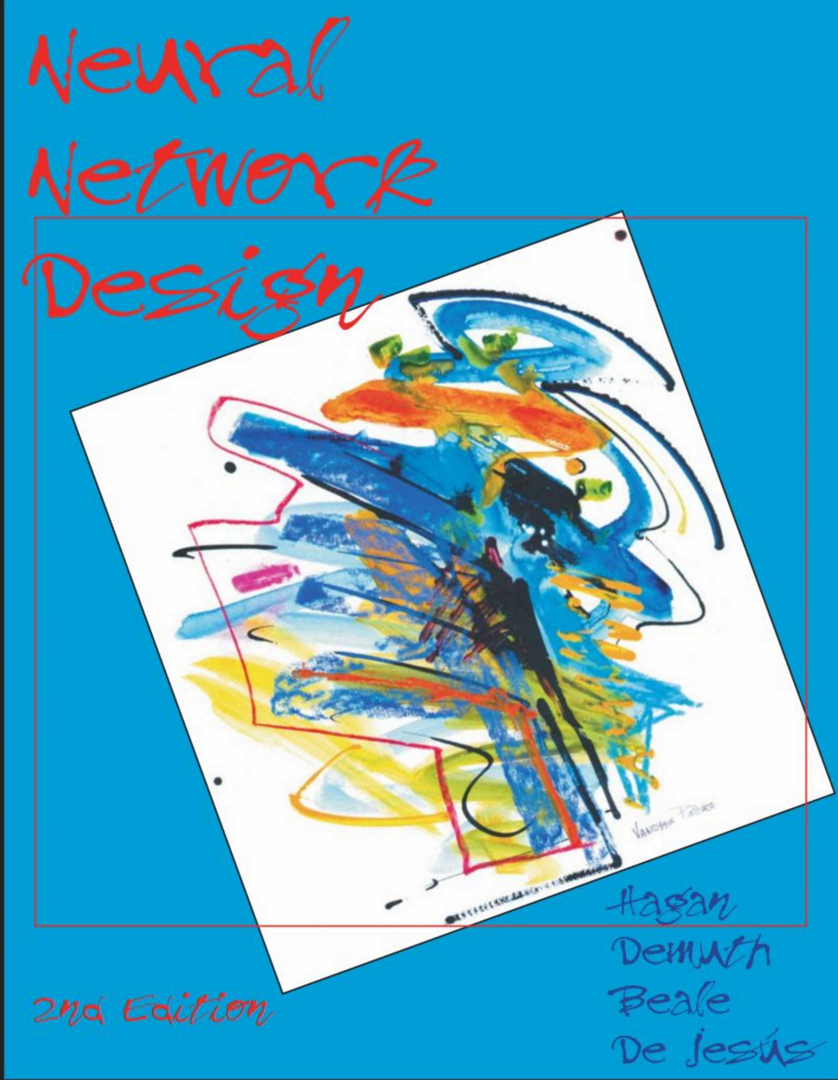
CS 231n:
Convolutional Neural Networks for Visual Recognition

cs231n.stanford.edu

Neural Network Design

2nd edition

Hagan et al.



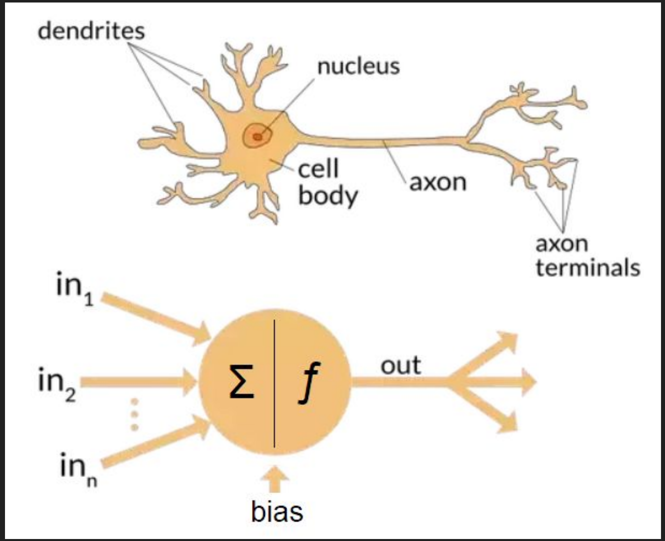
Idea credit:
Aleksander Obuchowski

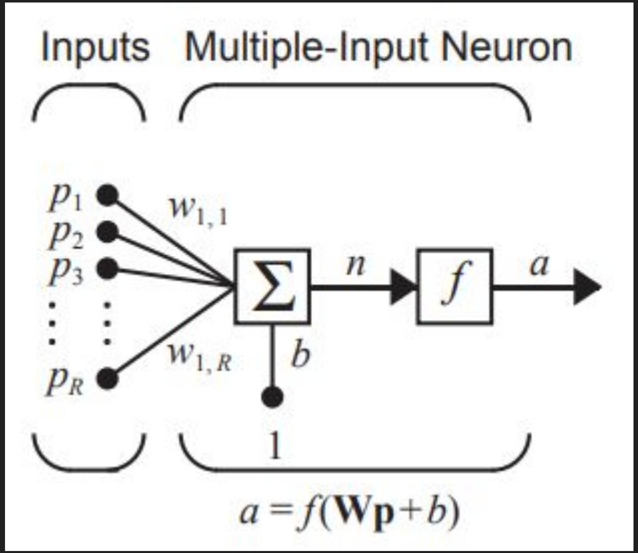
What is a neural network?

- Subset of machine learning algorithms based on biology, linear algebra, and statistics
- Searches for patterns and features in the data
- Compare to a provided ground truth (supervised)
- Alternatively, allow the network to make connections it thinks are logical (unsupervised)

Credit: XKCD

Single Artificial Neuron Diagram





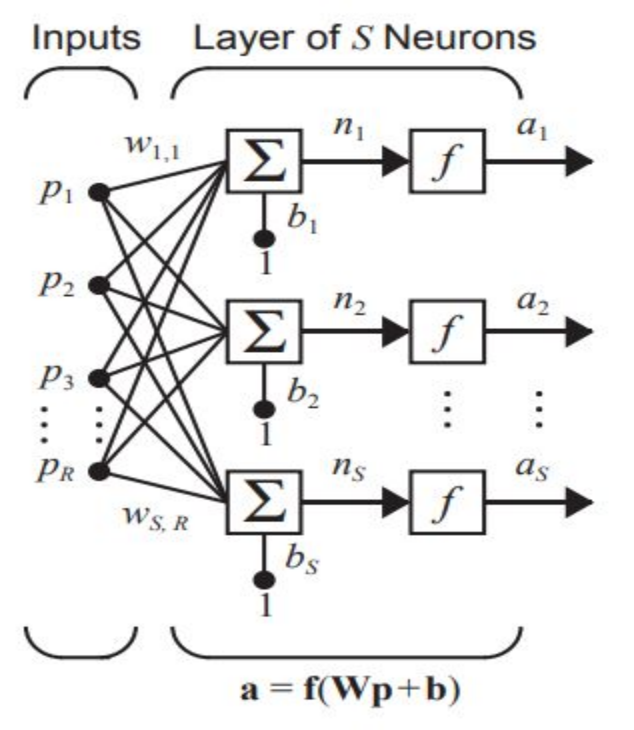
$$a = f(w_{1,1}p_1 + w_{1,2}p_2 + w_{1,3}p_3 + \ldots + w_{1,R}p_R + b)$$

Credit: Nagyfi Richard, Neural Network Design, Hagan et al., 2nd ed.

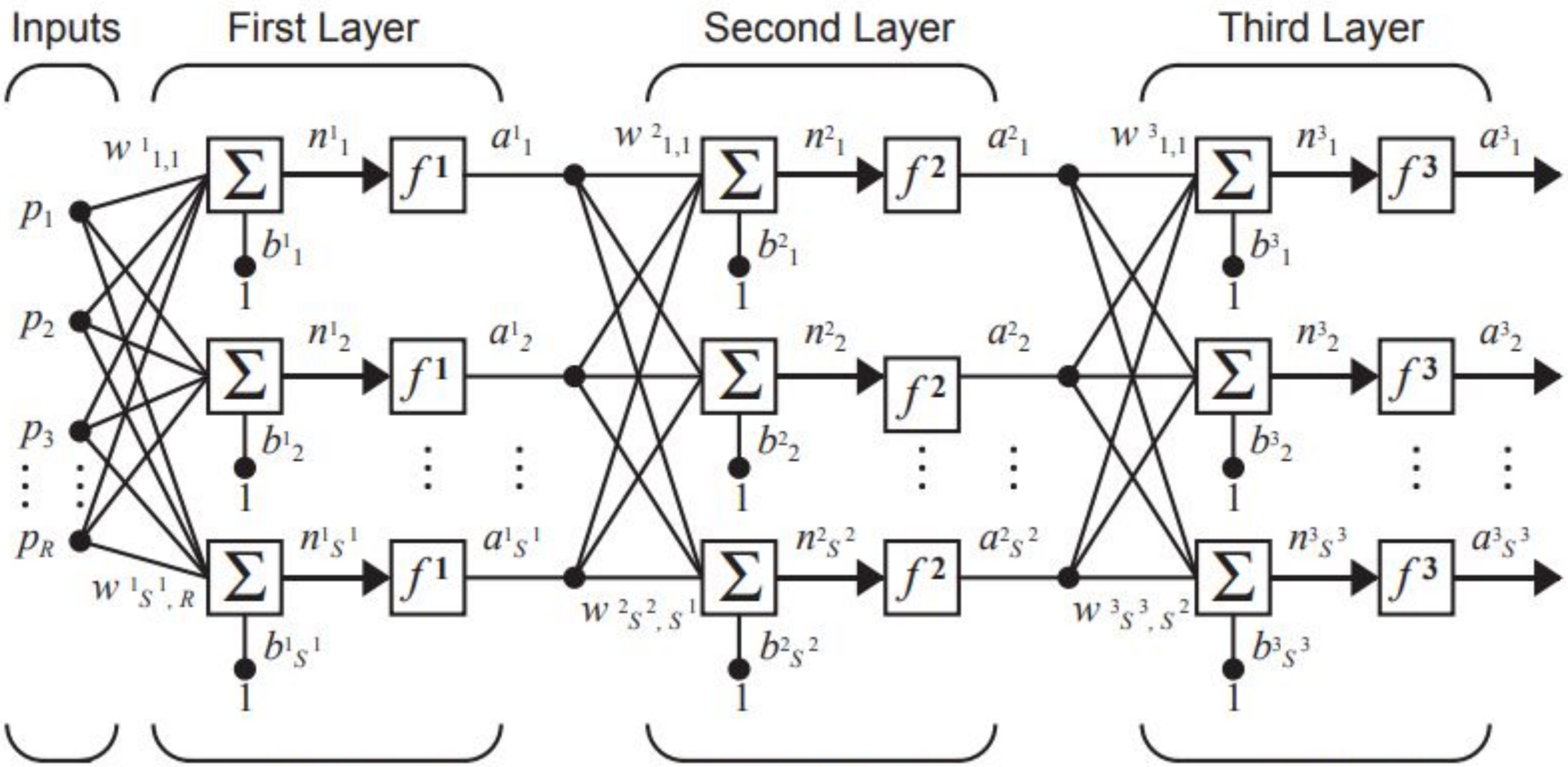
Multiple Neuron Diagram

- Every input is fed into each neuron which returns output a_i

- Number of outputs from a given layer depends on number of neurons in the layer which need not necessarily be constant between layers



Credit: Neural Network Design, Hagan et al., 2nd ed.



$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$$

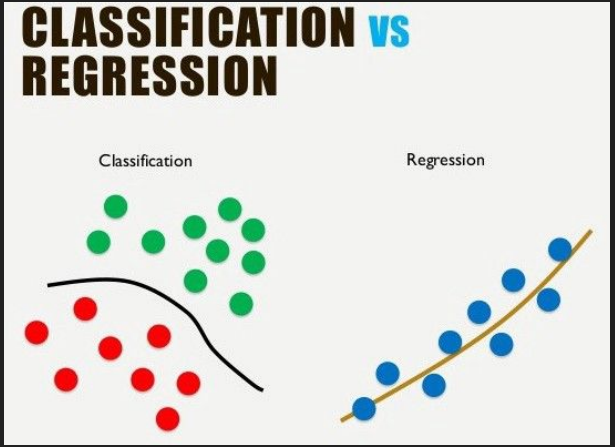
$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Credit: Neural Network Design, Hagan et al., 2nd ed.

Regression vs classification

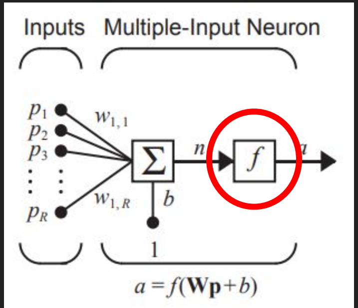
- Regression: outputs are real numbers ℝ which correspond to some normalized/pre-processed version of the dataset

- Classification: outputs are real numbers ℝ which represent the likelihood that a given input belongs to a certain class

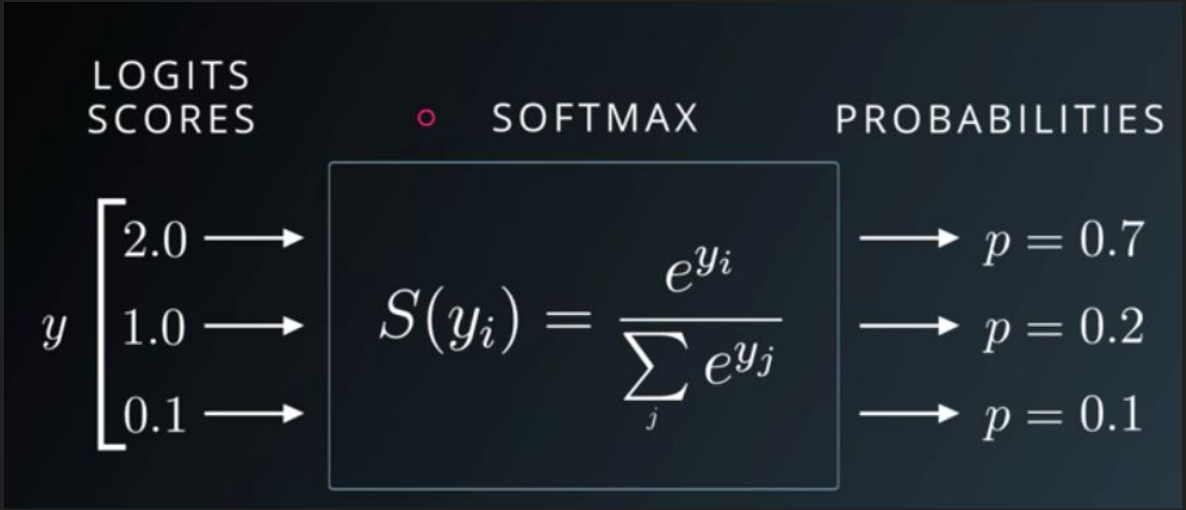


Credit: Codeing School

Activation function (classification)

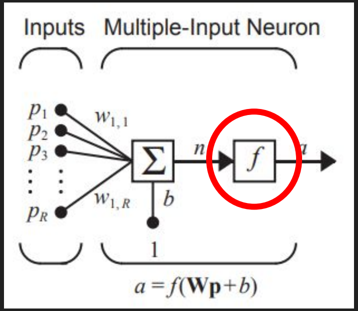


- Suppose you have 3 classes for images
  - Bird, deer, frog
- Activation function can normalize inputs to follow a probability distribution
- Based on current network parameters, predict most probable classification

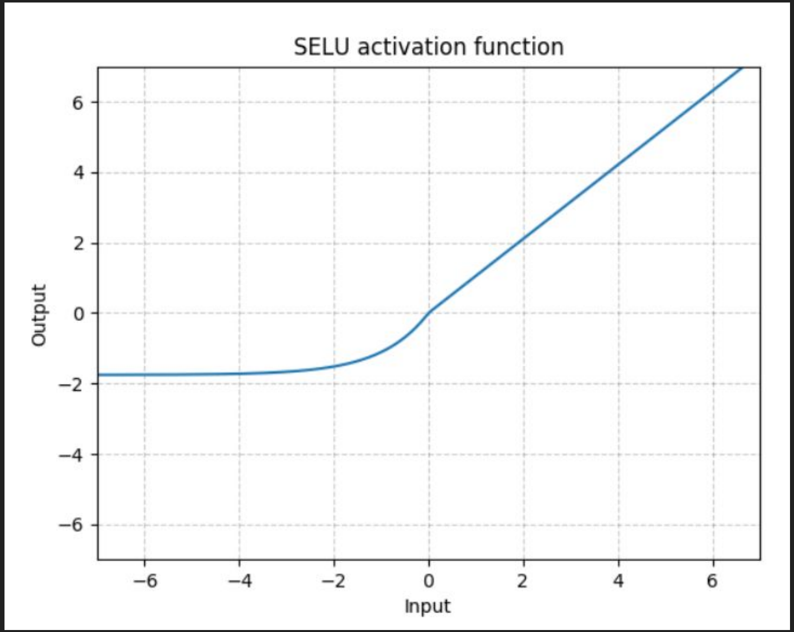


Credit: Udacity Deep Learning

Activation function (regression)



- Different type of problem requiring different function
- Function we use is scaled exponential linear unit (SELU)
- Decides which neurons are activated during a given iteration
- For regression, final layer has linear activation to allow for unbounded $\mathbb{R}$ values
- Data is usually normalized to keep behavior under control



Loss function (regression)

- Mean squared error
- Outputs some value in $\mathbb{R}$
- The error in the network's prediction
  is conveniently packaged into one number
- Minimize loss function for best results
- Classification uses different loss functions
  but the idea of minimizing still holds

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

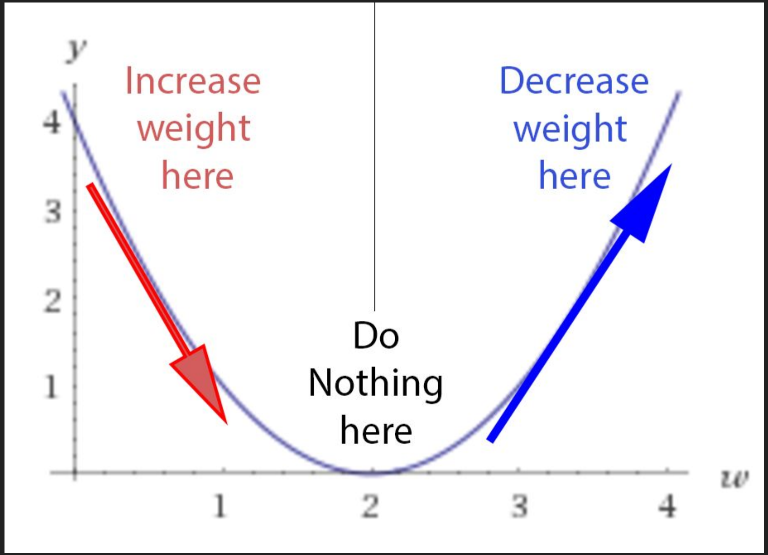
where N is the number of data points, f_i the value returned by the model and y_i the actual value for data point i.

Learning

- Adjust network based on previous loss value output such that following prediction is more accurate

- For one neuron, equivalent of testing whether increasing or decreasing the weight shows improvement

- Small adjustments of the weights across the network for all neurons

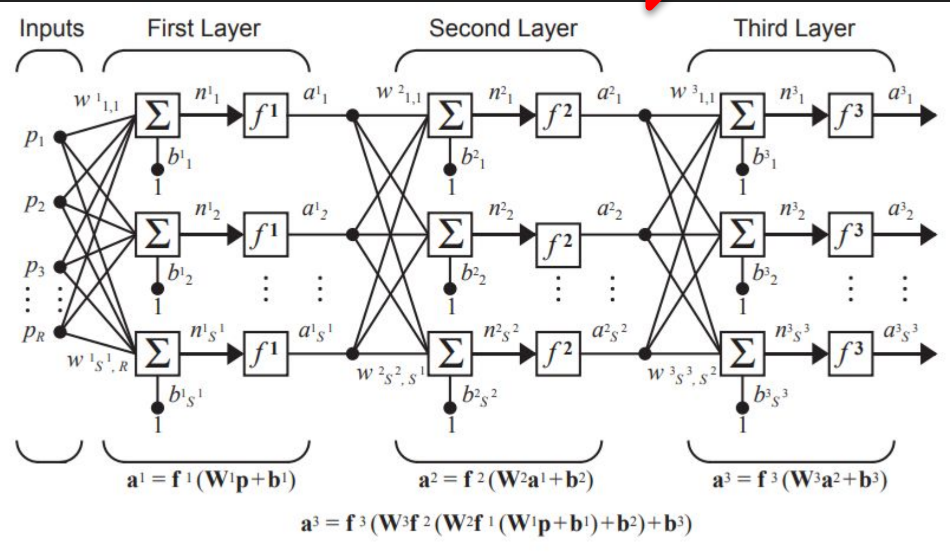


Credit: Assaad Moawad

Feed-forward Neural Networks

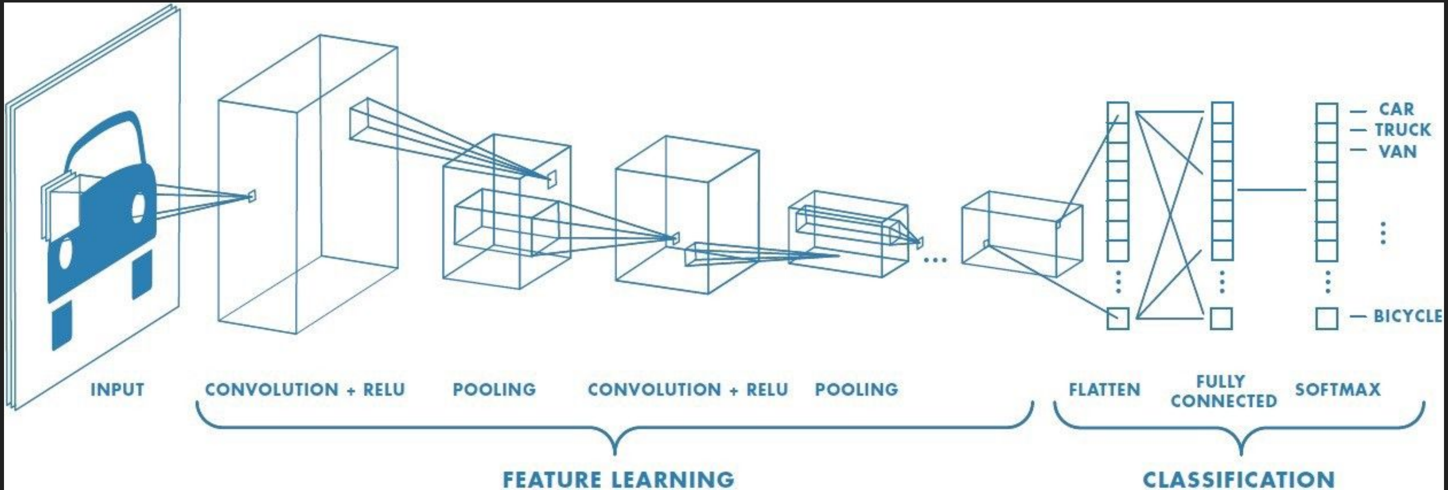
- Simplest category of NN
- Information flows in only one direction
- Want to approximate some function that maps the inputs to the outputs
- Regression ($\mathbb{R}^N$ to $\mathbb{R}^M$) or image classification

flow of information



Convolutional Neural Networks

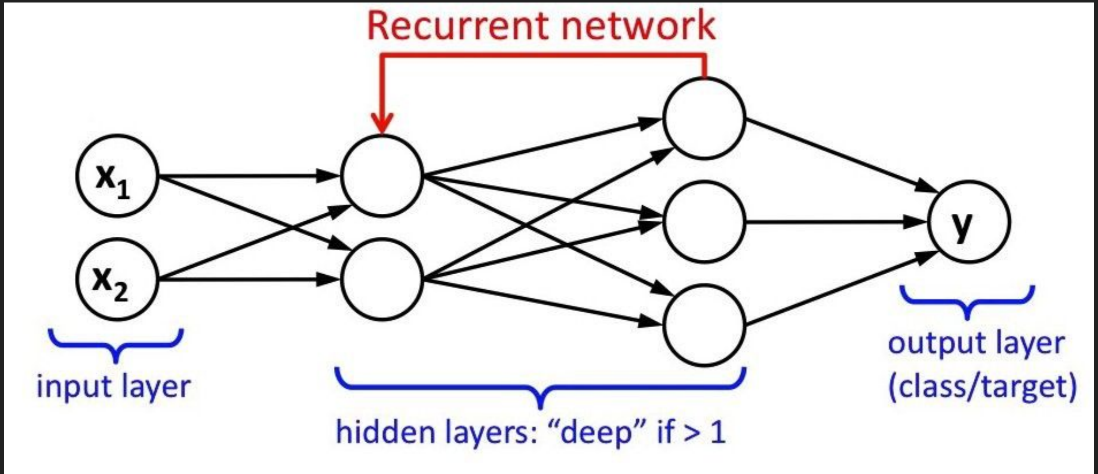
- Most commonly used in image classification
- Uses convolutions instead of direct weight multiplication
- Reduces image into a flattened vector of values representing learned features
  - Edges, shapes, etc.
- Softmax activation function determines class based on output probability



Credit: Sumit Saha

Recurrent Neural Networks

- Commonly used in unsegmented speech and handwriting recognition
- “Remember” what they learned from generating outputs based on prior Inputs
- As such, order of input processing determines possible predictions



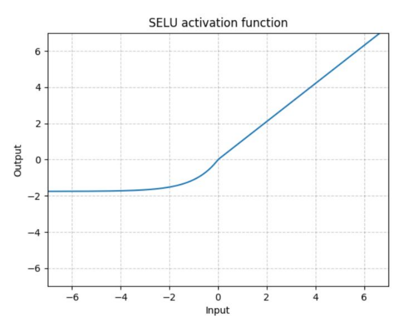
Credit: Chandra Churh Chatterjee

My Network

- Linear means fully-connected layers (as seen in diagrams)
- Final layer has no activation function
- Network doesn’t care what the data is, so this network should work for any 2D/4D dataset

```
def __init__(self, n_inputs, hlayer_size, n_outputs):
    super(Net, self).__init__()
    self.linear1 = torch.nn.Linear(n_inputs, hlayer_size)
    self.linear2 = torch.nn.Linear(hlayer_size, hlayer_size)
    self.linear3 = torch.nn.Linear(hlayer_size, hlayer_size)
    self.linear4 = torch.nn.Linear(hlayer_size, hlayer_size)
    self.linear5 = torch.nn.Linear(hlayer_size, hlayer_size)
    self.linear6 = torch.nn.Linear(hlayer_size, hlayer_size)
    self.linear7 = torch.nn.Linear(hlayer_size, hlayer_size)
    self.linear8 = torch.nn.Linear(hlayer_size, hlayer_size)
    self.linear9 = torch.nn.Linear(hlayer_size, n_outputs)

def forward(self, x):
    x = torch.selu(self.linear1(x))
    x = torch.selu(self.linear2(x))
    x = torch.selu(self.linear3(x))
    x = torch.selu(self.linear4(x))
    x = torch.selu(self.linear5(x))
    x = torch.selu(self.linear6(x))
    x = torch.selu(self.linear7(x))
    x = torch.selu(self.linear8(x))
    x = self.linear9(x)
    return x
```



Okay, but who cares?

- Want to interpolate the likelihood function for application in parameter estimation through Bayesian inference

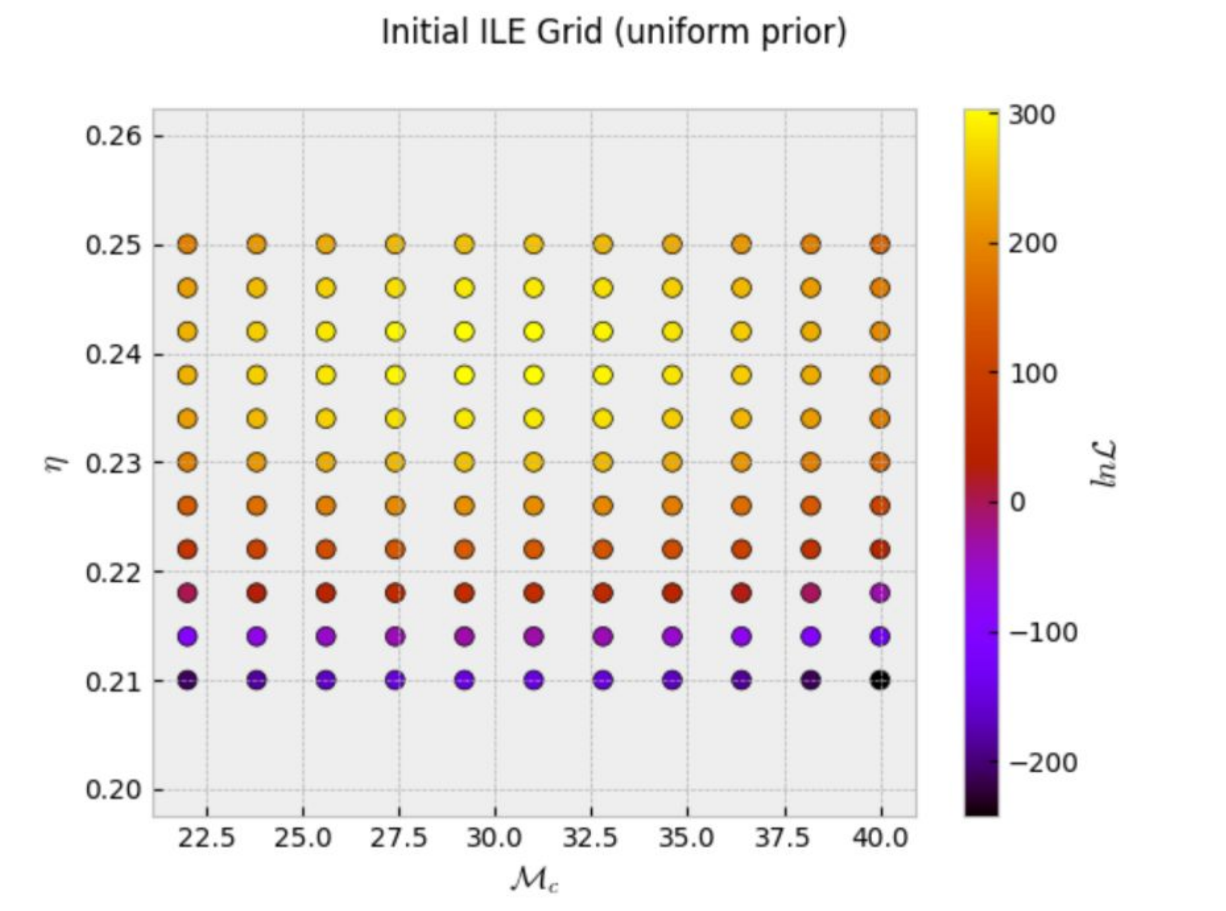
likelihood function

$$P(H|D,I) = \frac{\boxed{P(D|H,I)} \times P(H|I)}{P(D|I)}$$

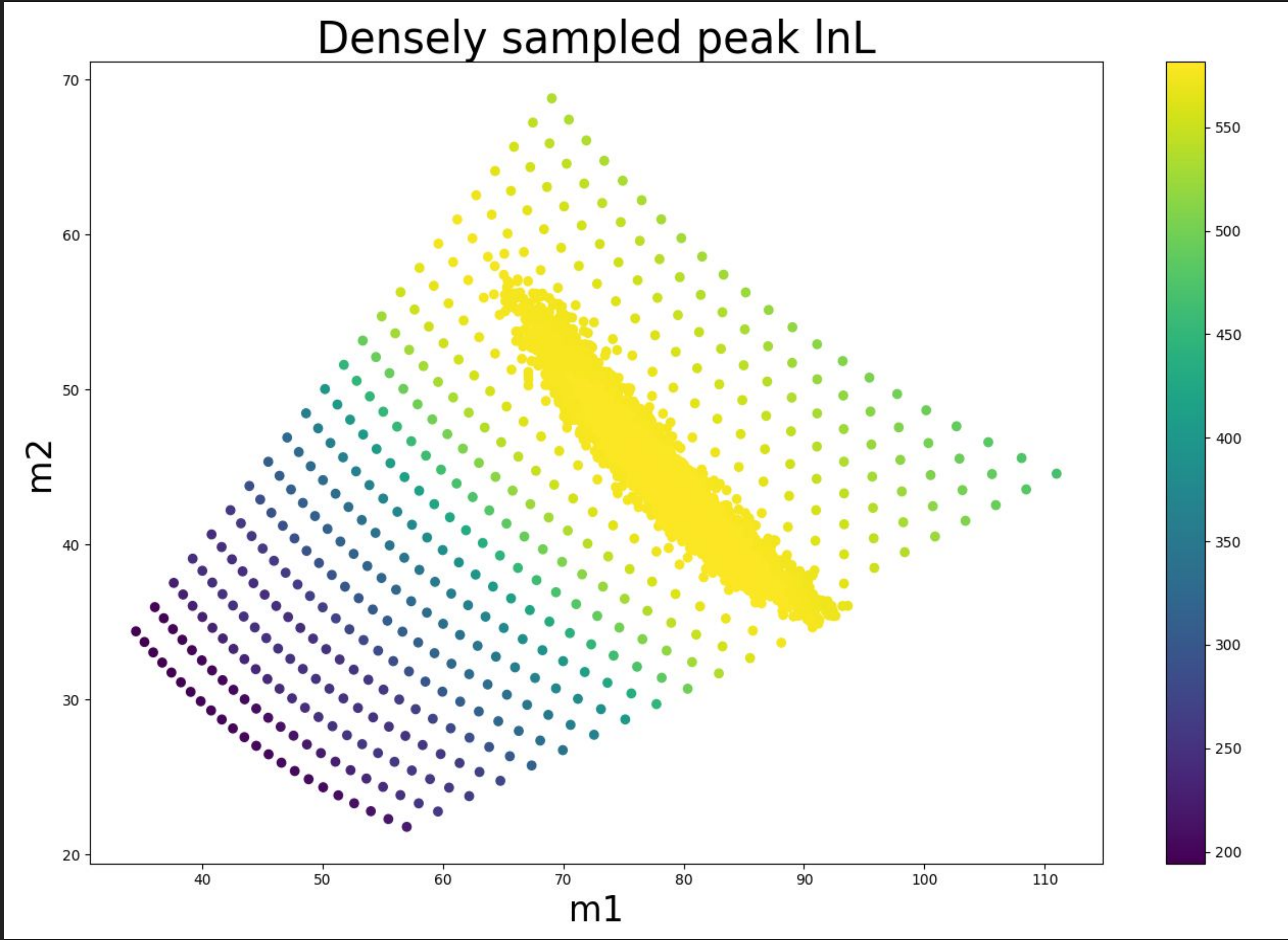
- H is the hypothesis, D is the observational data, I is some set of assumptions,
  P(H | D, I) is the posterior probability,
  P(D | H, I) is the likelihood function,
  P(H | I) is the prior probability,
  and P(D | I) can effectively be reduced to a weighting constant

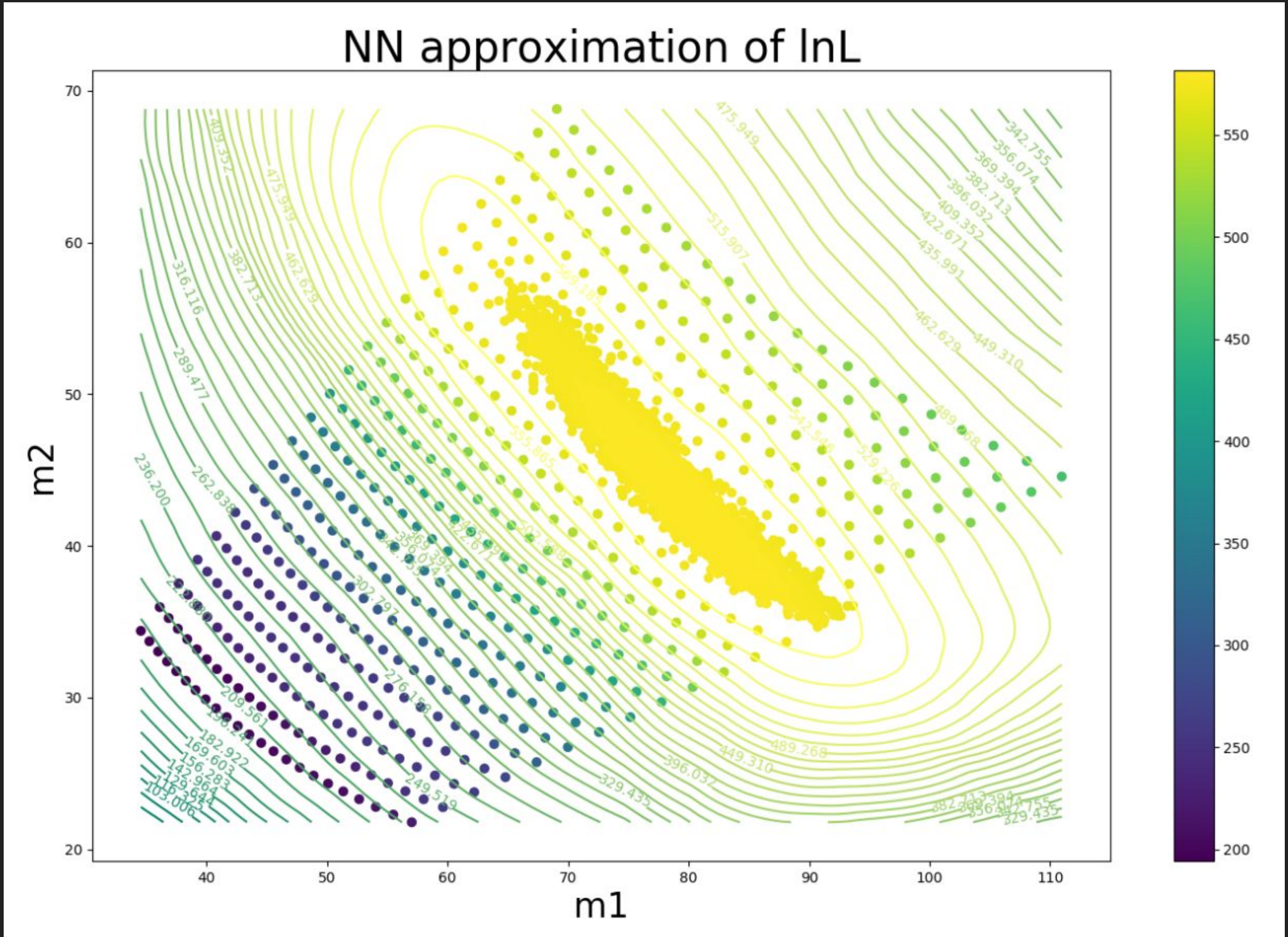
Likelihood grid

- Initially on an evenly-spaced grid
- Iterates several times and populates peak log-likelihood region with more samples
- Discrete representation of log-likelihood space



Credit: Vera Delfavero


Densely sampled peak lnL
m1
m2
20
30
40
50
60
70
40
50
60
70
80
90
100
110
200
250
300
350
400
450
500
550



NN approximation of lnL
m1
m2

Interpolation Improvement

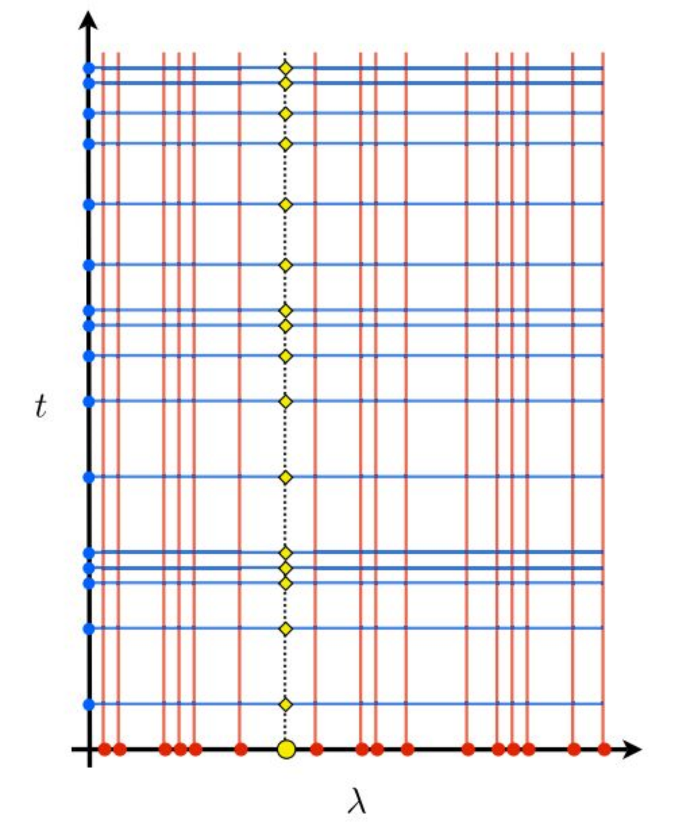
- Current method of Gaussian processes takes on the order of a day to interpolate the likelihood function, NN done on the order of minutes

- GPs scale on the order of n^3 which becomes problematic as the number of dimensions increases

- Neural networks have the advantage of benefiting from more data and also take on the order of minutes for a single interpolation

Future Application

- Interpolation of kilonova light curves
- Select a collection of models which can function as a basis for all kilonovae
- Do the same with points in time during the kilonova evolution
- Interpolate these to, ideally, create any possible kilonova model at any time in its evolution and recover the parameters



Credit: Field et al. 2014

Conclusion

- Neural networks learn patterns and features in data in different ways depending on the architecture of the network

- Our problem specifically involves approximating a continuous function based on a finite-sampled likelihood dataset in up to 4 dimensions

- Vast improvement over Gaussian processes with strong potential for direct application to kilonova interpolation