Intro to Deep Learning

Mar. 8, 2022

V. Ashley Villar

Our Goals:

- A second look at machine learning, in the context of “deep learning”
- A look at multilayer perceptrons
- Limited extensions to neural network architectures

How to fit a model using MOO

1. **Model** to fit the data (e.g. physics).
2. **Objective Function** (or ‘loss/cost function’) which is a metric that you will choose to quantify how well the model fits the data (e.g. chi-squared).
3. **Optimization Method** which you will use to find the best model (e.g. gradient descent).

Let’s look at a simple linear model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + ... = \vec{\beta}\vec{X}$$

In our language:

Dependent variable/output

Independent variable(s)

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots = \vec{\beta}\vec{X}$$

Parameters

In ML language:

Output Features

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots = \vec{\beta}\vec{X}$$

Weights

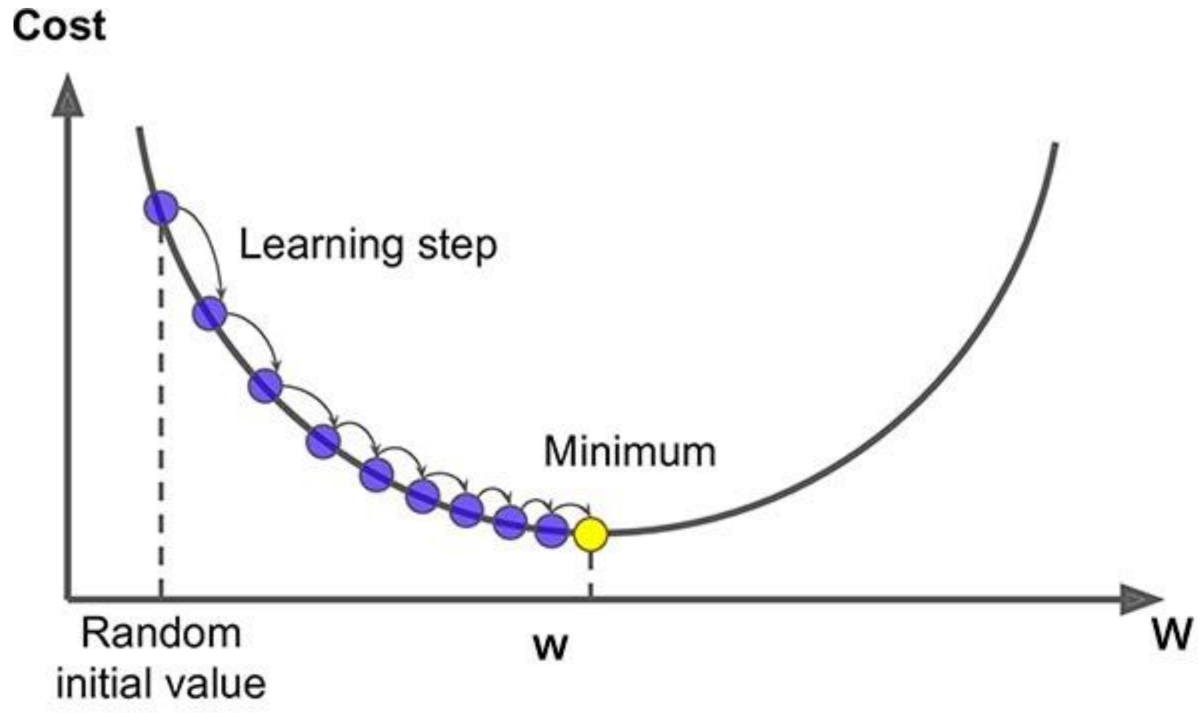
Objective function: A scalar to optimize

$$C = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

$$C = \sum_{i=1}^{n}(y_i - f(x_i))^2 + \underbrace{\boxed{\lambda \sum_{j=0}^{2} \beta_j^2}}_{\text{Regularization}}$$

Optimization:



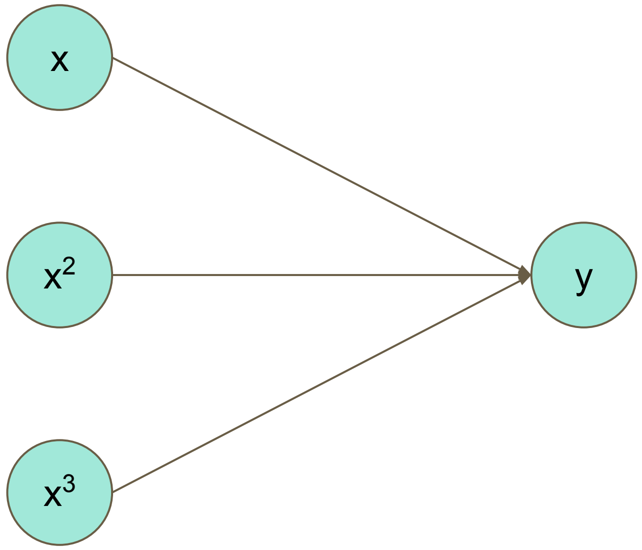
A Neural Network is a model

Typically use (stochastic) gradient descent to optimize it

What is a neural network?

A (barely) nonlinear combination of random variables which can simulate complex, nonlinear interactions among input variables.

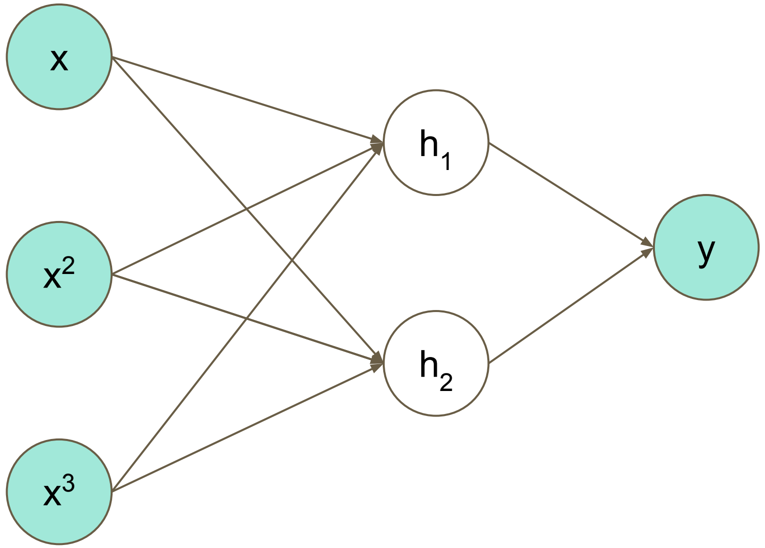
$$f(x) = \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$



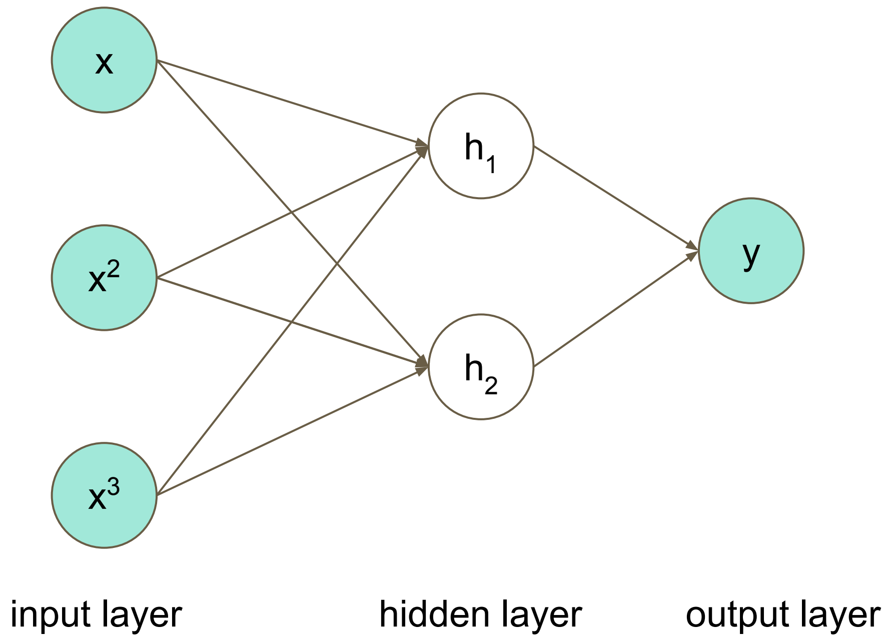
$$h_1(x, x^2, x^3) = \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

$$h_2(x, x^2, x^3) = \beta_4 x + \beta_5 x^2 + \beta_6 x^3$$

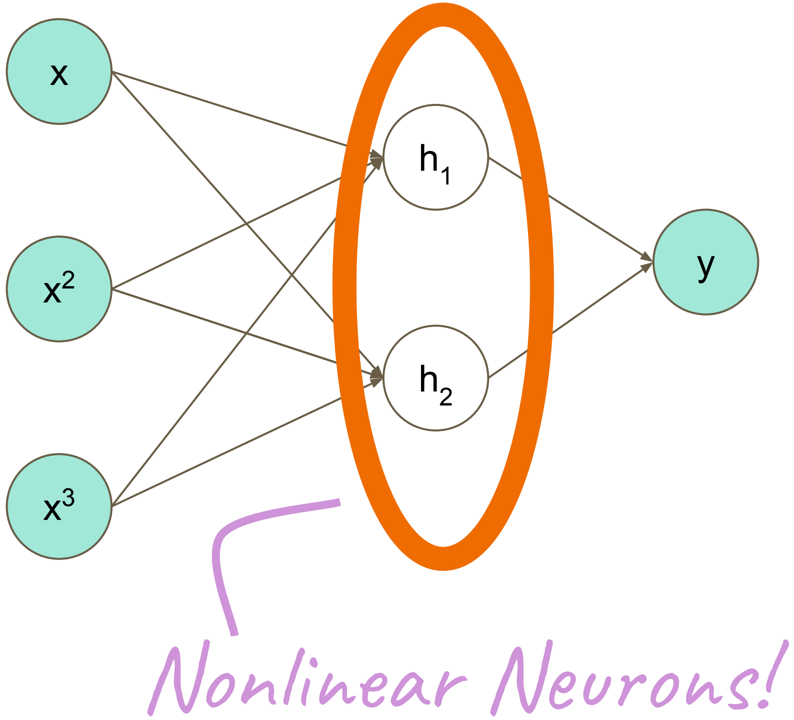
$$y(h_1, h_2) = \beta_7 h_1 + \beta_8 h_2$$



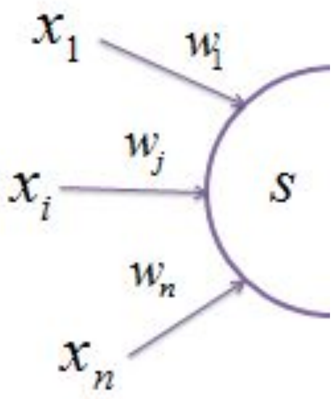
A Multi-layered Perceptron



A Multi-layered Perceptron



Inside a Neuron (Hidden Unit)

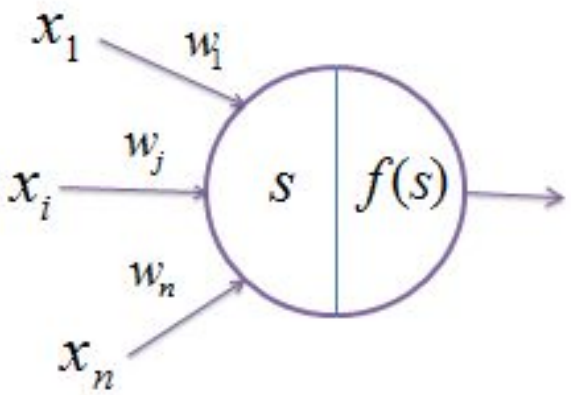


Summation

$$s = \sum w \cdot x$$

+b

Inside a Neuron (Hidden Unit)



Summation

$$s = \sum w \cdot x$$

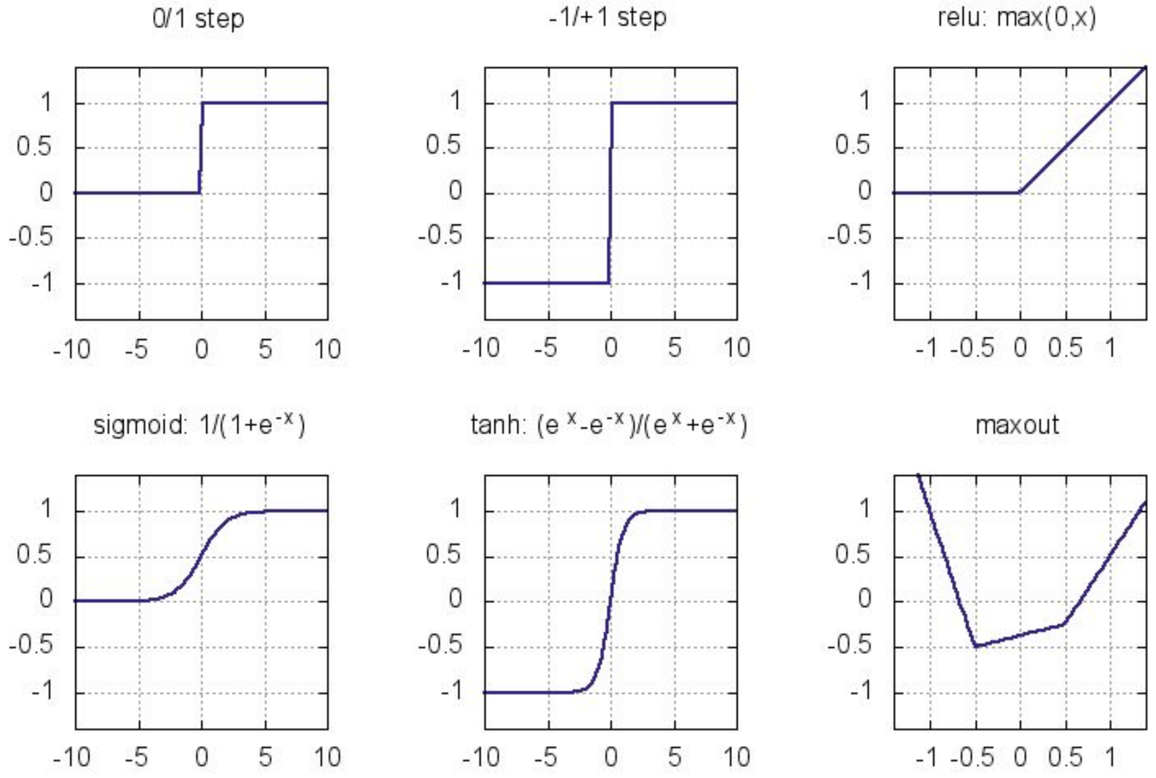
+b

Transformation

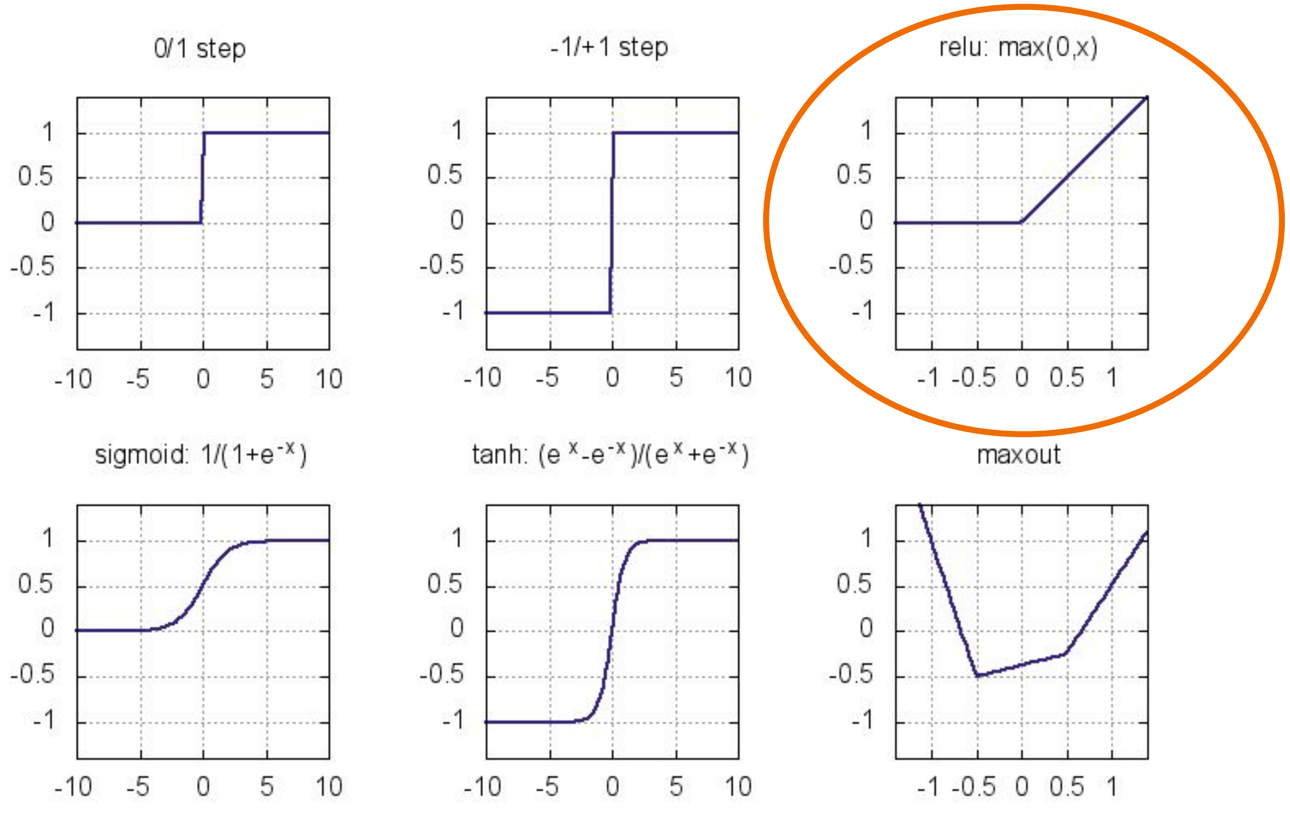
$$f(s) = \frac{1}{1+e^{-s}}$$

Activation Function

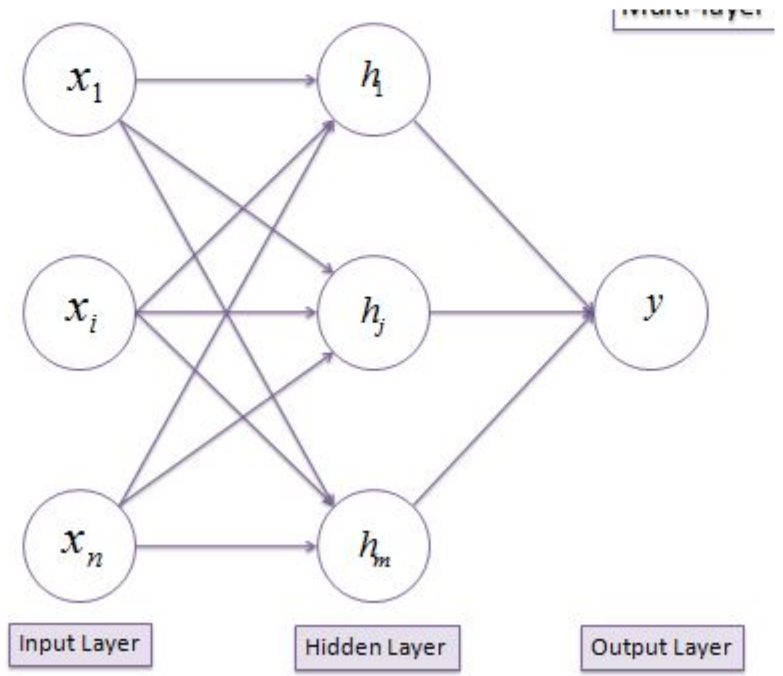
Your activation function can be ANY (nonlinear) function!



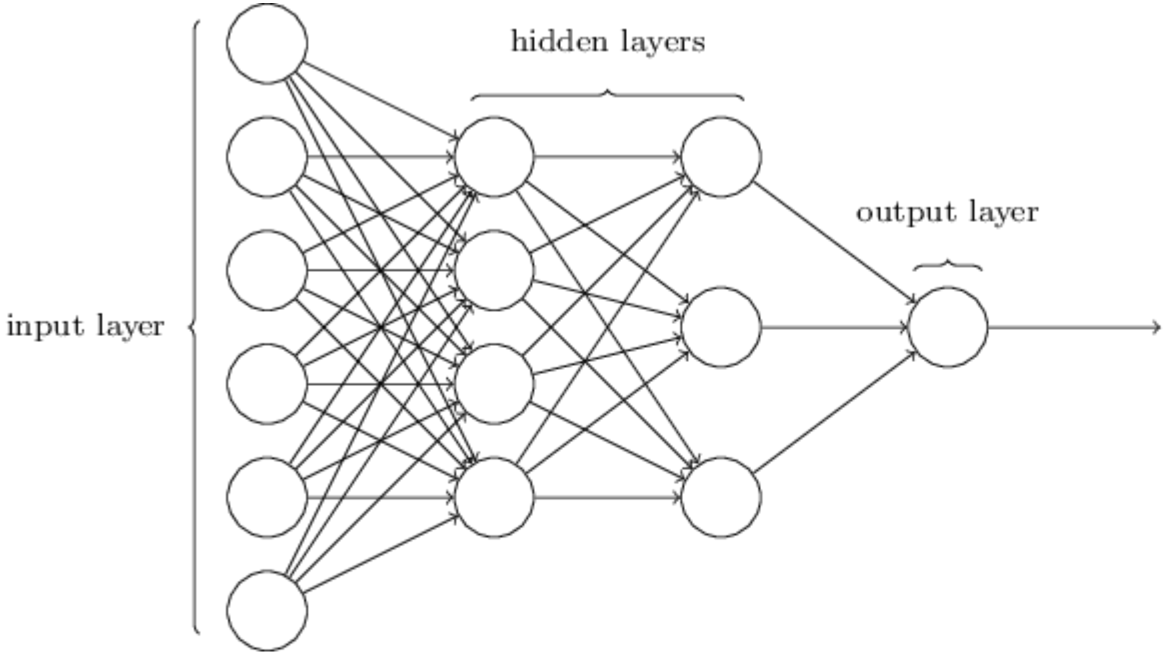
Your activation function can be ANY (nonlinear) function!



Pop quiz: How many free parameters?



Deeeeeeep Learning



In general, different ‘types’ of neural networks are different combinations of the following flavors:

1. Sizes and depth of the neural network
2. The input and outputs
3. The types of “neurons” (eg more than just a simple activation function)

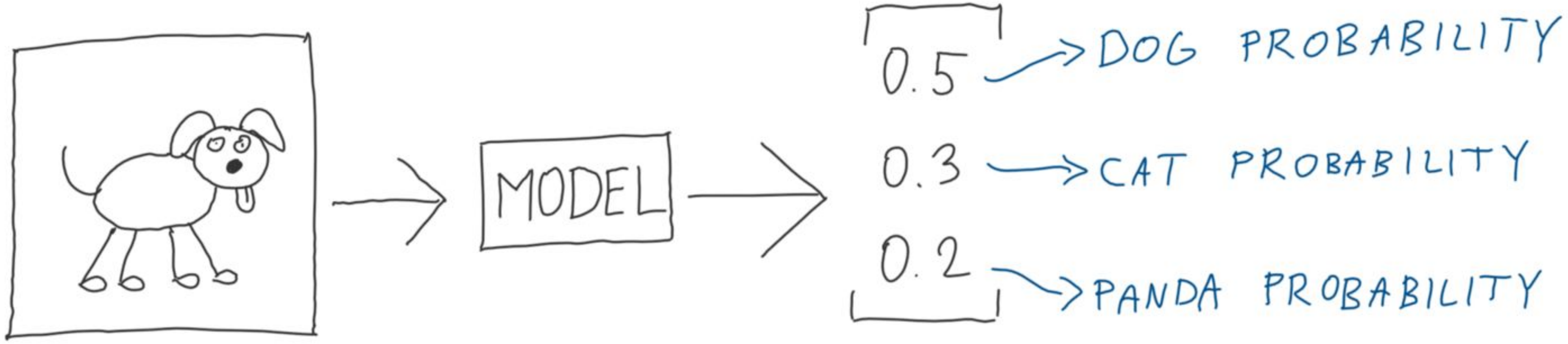
Example inputs

- Powers of 1-D data
- Raw spectral/image data
- One-hot vectors for categorical data

Now we need to “train” our neural work, which means we need to optimize the neuron weights

To train, we need to specify an objective function

Categorical cross entropy is a common classification objective function



From Vlastimil Martinek on towardsdatascience

Categorical cross entropy is a common classification objective function

$$\begin{bmatrix} 0.5 \\ 0.3 \\ 0.2 \end{bmatrix}$$

How do i guarantee that this is a “probability’ vector?
What are the properties of a probability vector?

From Vlastimil Martinek on towardsdatascience

Categorical cross entropy uses the *softmax* of the output

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

From Vlastimil Martinek on towardsdatascience

An example objective function for regression

$$C = \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \sum_{j=0}^{2} \beta_j^2$$

From Vlastimil Martinek on towardsdatascience

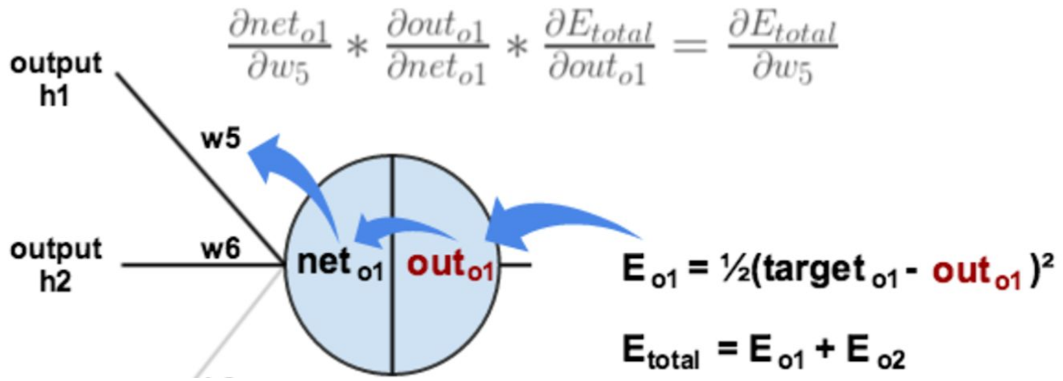
Now we must optimize our objective function

“Backpropagation” for gradient descent

By applying the chain rule we know that:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

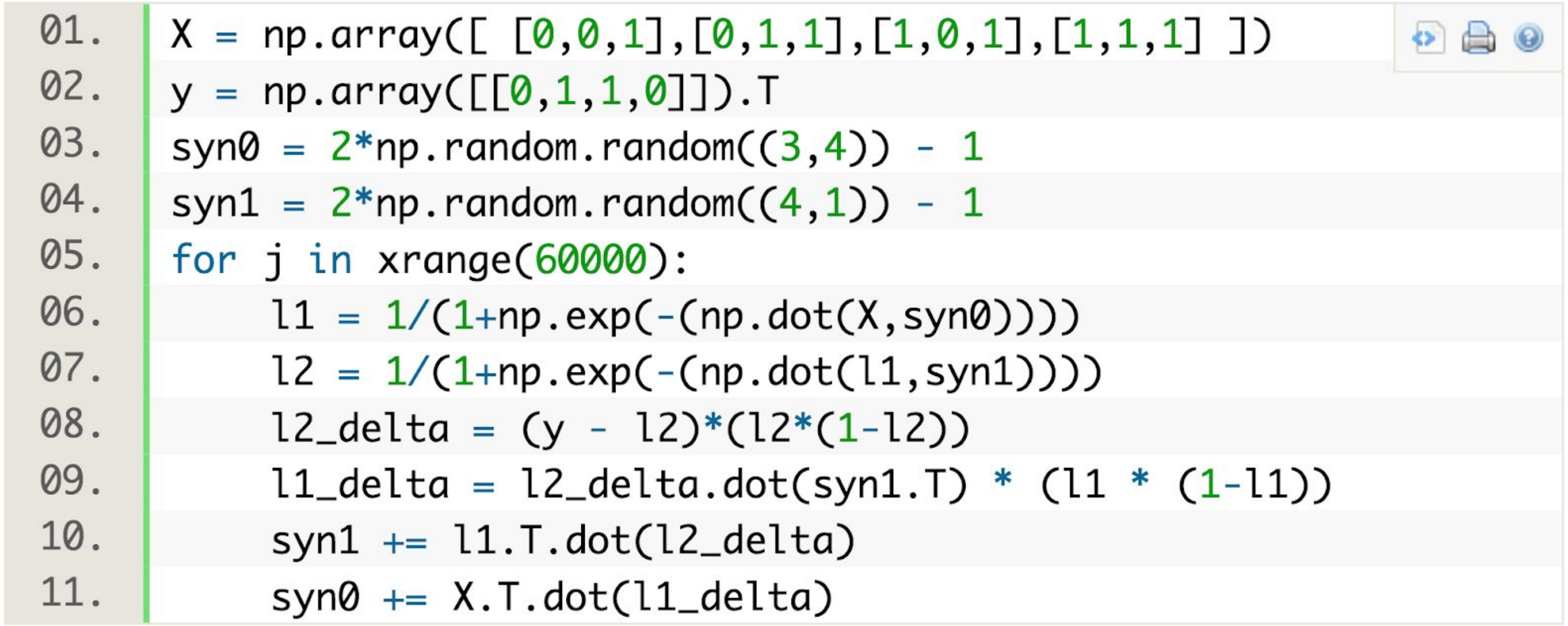
Visually, here’s what we’re doing:



$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$
output h1
w5
output h2
w6
net_{o1}
out_{o1}
$E_{o1} = ½(target_{o1} - out_{o1})^2$
$E_{total} = E_{o1} + E_{o2}$

2/3-layer NNs in Python

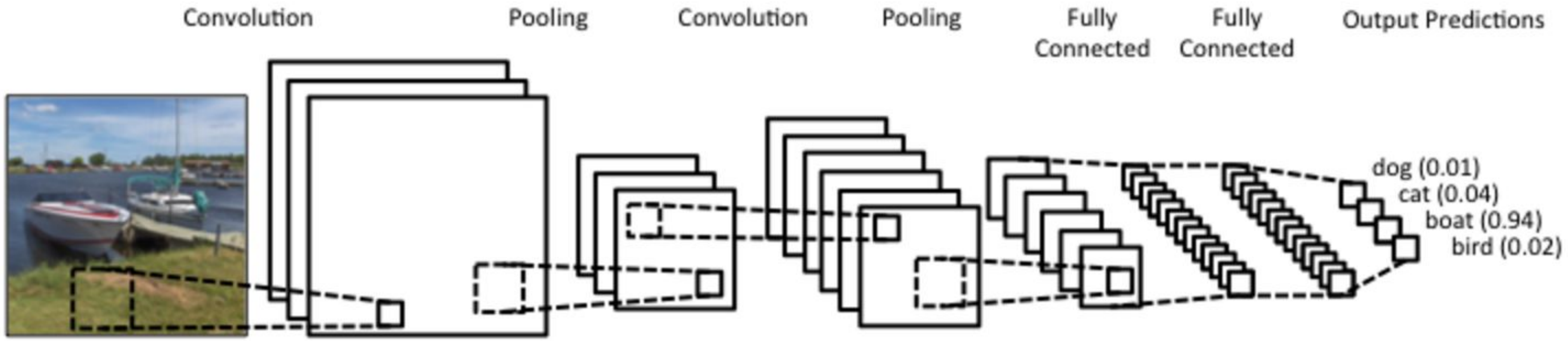
https://iamtrask.github.io/2015/07/12/basic-python-network/



```python
X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
y = np.array([[0,1,1,0]]).T
syn0 = 2*np.random.random((3,4)) - 1
syn1 = 2*np.random.random((4,1)) - 1
for j in xrange(60000):
    l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
    l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
    l2_delta = (y - l2)*(l2*(1-l2))
    l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
    syn1 += l1.T.dot(l2_delta)
    syn0 += X.T.dot(l1_delta)
```

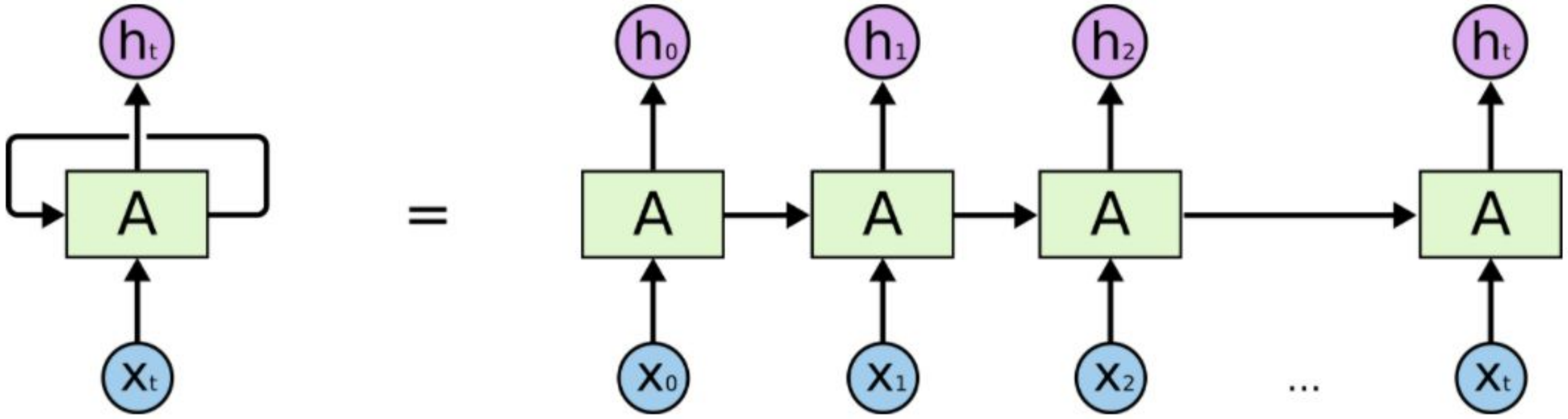
What about other NN architectures?

Convolutional Neural Network (CNN)



Typically used for images

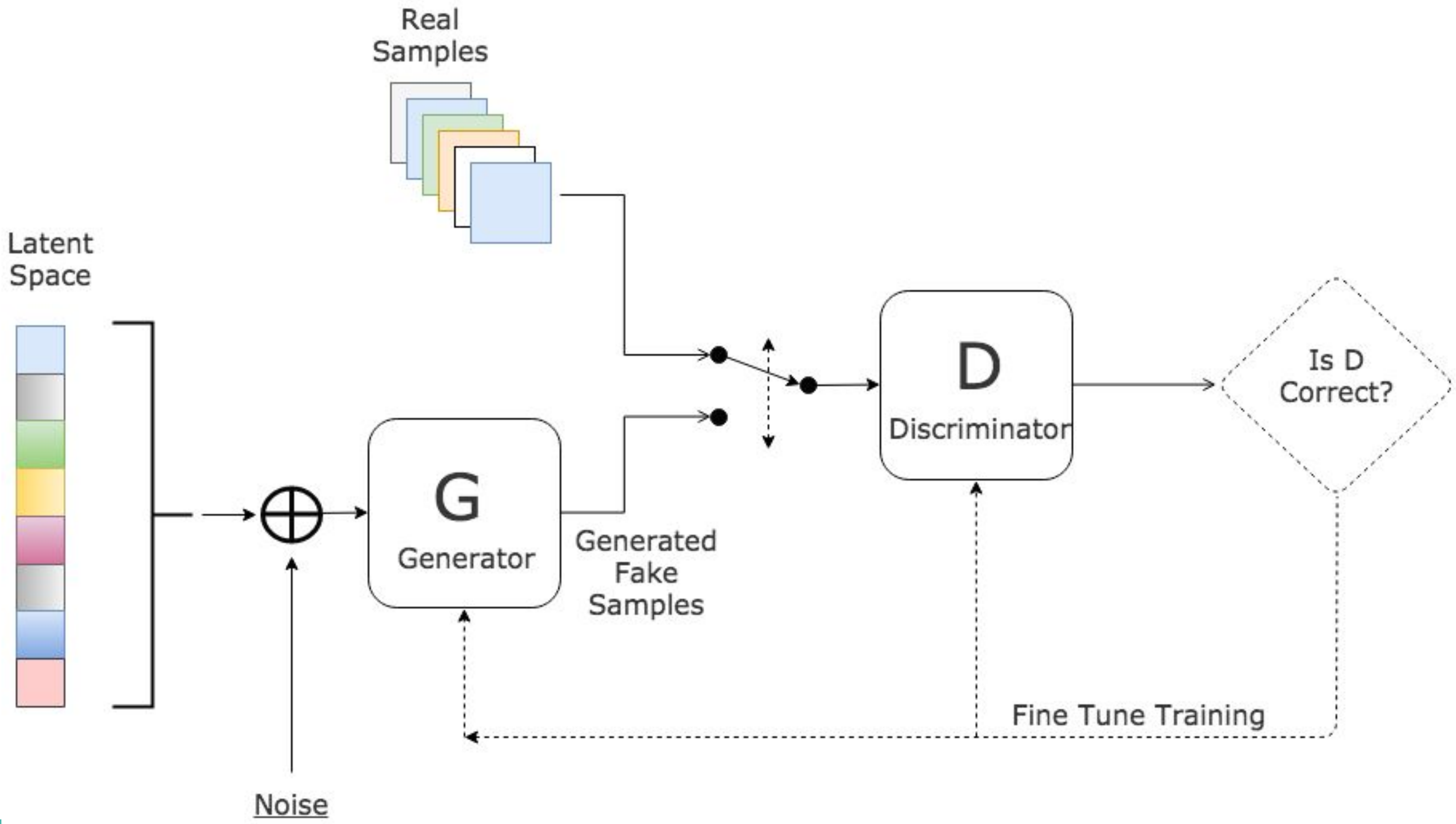
Recurrent Neural Network



An unrolled recurrent neural network.

Best for sequences of data (like predicting the next word in a sentence.)

Generative Adversarial Network



Some thoughts on coding NNs in Python

Just like “scipy” or “numpy” as the go-to backends for science/mathematical coding, deep learning also has some “go-to” modules…unfortunately, there is *not* a clear winner

Pytorch

This is what we’ll use. It is, currently, the most common deep learning module. The vast majority of ML papers make their code available with a Pytorch backend

From the Facebook AI team ¯_(ツ)_/¯

Works natively with the new M1 chip in MAC

Keras, with a tensorflow backend

Extremely easy to use. Ashley’s favorite, basic tutorials use Keras

Keras is an open source project, but it is built on tensorflow, which is developed originally by Google Brain

Losing steam, unfortunately. This is due to the somewhat haphazard nature of tensorflow development.

Currently a *PAIN* to use on M1 Mac

Probabilistic Programming!

This includes JAX (google), numpyro (opensource, built on JAX backend) and pyro (Uber, build on pytorch backend)

These more easily integrate probabilistic methods, which is really gaining traction in the community.

JAX is currently a pain to get working on the M1

To conclude:

ML still utilizes the “MOO” framework for fitting a model, yet capitalized on the universal approximation theorem

The “architecture” of a neural network is dependent on its structure and the type of neurons

There is a whole zoo of neural network architectures, so go explore!