Machine learning

Compositional non-linearity

Joshua Loftus

Mathematical context

Non-linear function approximation

Neural networks

Universal approximators / building blocks

Deep learning

A revolution(?) in modeling, optimization, and data

Function approximation of $\mathbb{E}[\mathbf{y}|\mathbf{x}]$

- Linear models
- Linear models with global transformations of predictors
  - $\mathbf{x}'_j = h_j(\mathbf{X}_j)$
- Additive non-linear models
  - Kernel methods $\hat{f}(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T\mathbf{y}$
  - GAMs, similar to global with supervised learning of $\hat{h}_j$
  - Tree-based methods
- **Compositional non-linear models**

$$\hat{f}(\mathbf{x}) = \hat{f}_L \circ \cdots \circ \hat{f}_2 \circ \hat{f}_1(\mathbf{x}) = \hat{f}_L(\cdots(\hat{f}_2(\hat{f}_1(\mathbf{x}))\cdots)$$

Example: you already know GLMs!

$$\mathbb{E}[\mathbf{y}|\mathbf{x}] = g^{-1}(\mathbf{x}^T\beta)$$

Neural networks

Multilayer perceptron (MLP), feedforward network. ESL:

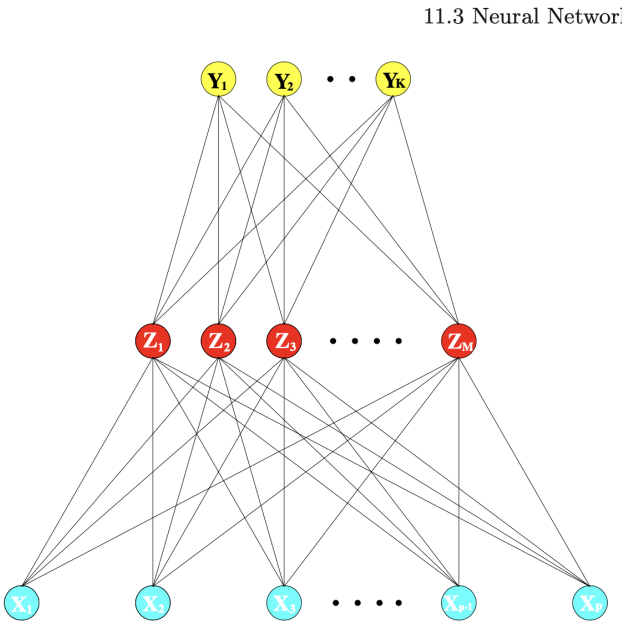


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

One "hidden layer" and other terminology

Inputs: $\mathbf{X} = [\mathbf{X}_1, \ldots, \mathbf{X}_p]^T$

"Hidden" layer: $\mathbf{Z} = \sigma(\beta_0^{(1)} + \mathbf{X}\beta^{(1)}) = [\mathbf{Z}_1, \ldots, \mathbf{Z}_M]^T$

Outputs: $\mathbf{Y} = g(\beta_0^{(2)} + \mathbf{Z}\beta^{(2)}) = [\mathbf{Y}_1, \ldots, \mathbf{Y}_K]^T$

Units: (`keras` terminology) outputs of a given layer, e.g. M (hidden units) for the input layer or K (outputs/classes) for the hidden layer

Activation (non-linear) functions: σ and g in this example

Parameters: weights β and biases β_0 (yes, really...) 🙄🤦

Learning/optimization: fixed activations, fit parameters

MNIST example with ESL 11.2 notation p, M, K

```
library(keras); p <- c(28, 28); M <- 128; K <- 10
model <- keras_model_sequential() %>%
  layer_flatten(input_shape = p) %>%
  layer_dense(units = K, activation = "relu") %>%
  layer_dense(10, activation = "softmax")
```


History / connection to "neurons"

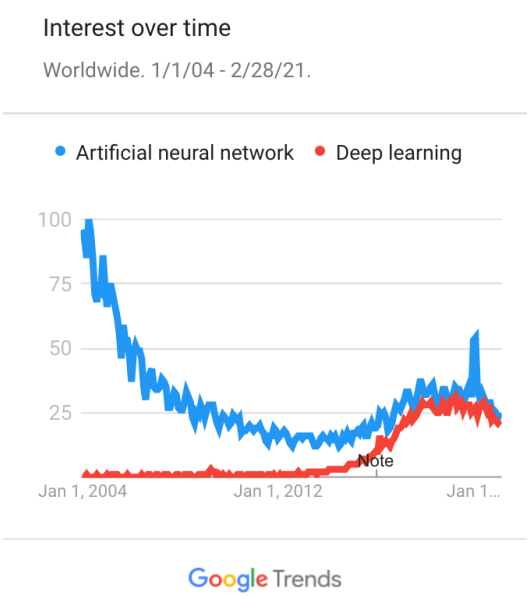
Early work (1943) motivated by analogy to neurons, "activation" of a neuron cell's action potential which sends an electric signal to connected neurons

Equivalence between a mathematical model of brain activity and boolean logic

Universal approximation means this class of models can approximate any continuous function. First shown in 1989 for softmax activation, later extended to other activations

Useful analogy, can inspire research ideas (remember it's just an analogy, it doesn't mean actual brain biology works this way or that neural networks are like thinking brains...)

Second wave of neural network research



Combination of algorithms, software, and **datasets** allow training much larger and more complex models

NNs with multiple hidden layers start beating SoTA on image classification tasks

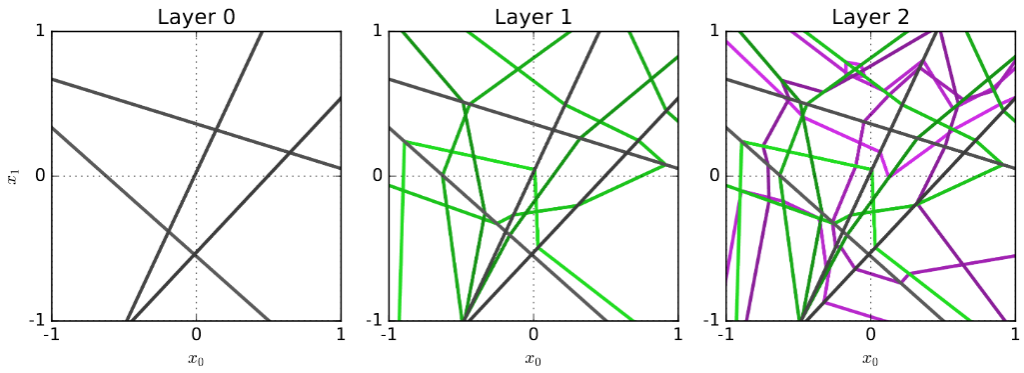
Driven by empirical performance on prediction/classification, very little theory or interpretability

Deep = multiple hidden layers (CASI 18.3)

Benefits of deepness

Some functions require exponentially many parameters to represent with a 2-layer network, but grow polynomially if depth of network increases

Expressivity depends more strongly on depth, e.g. trajectory length of a curve in input space grows exponentially in depth (Figure 1 from previous link). (Recall "real data" manifold idea)



What's special about deep learning?

Faster optimizers and hardware, and larger datasets, also allow other methods to scale up number of parameters. So why did deep learning become SoTA?

Flexibility, expressivity, "capacity", *degrees of freedom is not just about counting parameters*

1. Smaller layers limit capacity (bottleneck effect)

2. But shallow networks may not learn structure

$\rightarrow$ Want more layers (2), but to avoid making them small (1) need enough data to estimate many more parameters

Specialized modeling choices

For deep networks, choose number L of (hidden) layers, and *for every layer* choose

- Layer type (dense, convolutional, pooling, ...) (matrix/tensor multiplication with some kind of structure)
- Number/shape (e.g. vector/matrix/tensor) of outputs
- Activation function (ReLU, sigmoid, linear, tanh, ...)
- Regularizer (L1 or L2 norm penalties, dropout, ...)

Compare to e.g. regularized kernel regression: choose kernel, regularizer, bandwidth. Not that many choices.

Network architecture = combinatorial explosion of choices

Modeling side of the DL revolution

Specialized optimization

After network architecture is fixed, how do we train the network / estimate the weights and biases?

To use gradient descent, need to compute gradient of

$$\sigma_L(\cdots \sigma_2(\beta_0^{(2)} + \sigma_1(\beta_0^{(1)} + \mathbf{X}\beta^{(1)})\beta^{(2)})\cdots)$$

with respect to all the β's

Chain rule, automatic differentiation, efficient implementations, handling numerical issues, etc

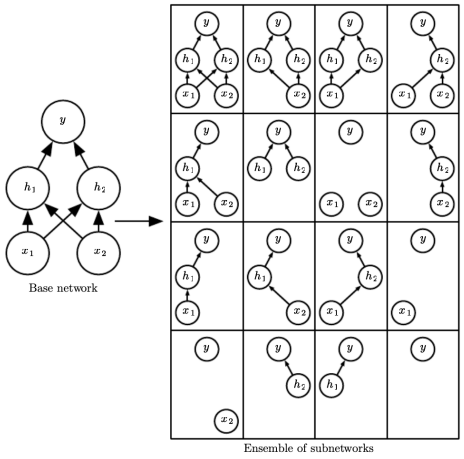
Epoch: one gradient descent pass over entire training data

Algorithm and software side of the DL revolution

Specialized regularization

- **Dropout** is like bagging
- During training: randomly drop units, iteration of gradient descent uses subnetwork
- During testing: use full network

Averaging over many models represented by all possible subnetworks



(Figure 7.6 from DLbook)

Specialized (or same old?) statistical problems

Decisions about network architecture made by experimenting, checking error on test data, and trying again...

Recall Duda and Hart:

> A [...] problem arises *when a classifier undergoes a long series of refinements guided by the results of repeated testing on the same data*. This form of "**training on the testing data**" often escapes attention until new test samples are obtained.

Crisis: adversarial examples, robustness, OOD generalization

Crisis: breakdown of the Common Task Framework? (David Donoho attributes the success of machine learning to the CTF)

Questions/practical lessons

Why does it "work" (beat SoTA)?

Short answer: a lot of resources (expertise, time, data, money) went into making it work! e.g. MLstory **SGD Quick Start Guide**:

1. Pick as large a minibatch size as you can given your computer's RAM.
2. Set your momentum parameter to either 0 or 0.9. Your call!
3. Find the largest constant stepsize such that SGD doesn't diverge. This takes some trial and error, but you only need to be accurate to within a factor of 10 here.
4. Run SGD with this constant stepsize until the empirical risk plateaus.
5. Reduce the stepsize by a constant factor (say, 10)
6. Repeat steps 4 and 5 until you converge.

Questions/mathematical lessons

Non-convex, but doesn't seem to suffer local minima problems

Overparametrized so global optimum is 0 training loss. Real question is why do these solutions generalize, aren't overfit?

- Representer theorem heuristic - p dimensional problem reduces to n dimensional problem
- "Implicit regularization" - SGD converges to minimum norm solutions
- Remember, optimization and generalization are still not equivalent! No guarantee that minimum norm solution is a good one, but maybe if there's signal in the data [can always find a minimum norm solution to predict y sampled independently from `rnorm(n)`]

Two point statistical summary of DL

- Overparametrization apparently has a lot of benefits, at the cost of interpretability
- But don't forget researcher degrees of freedom

Deep = problematic?

Where are we going?

Don't usually talk about this in our courses...

But professional ethics are an *increasingly* important, central part of our discipline

- This tech is going to be used for everything
- It's "data hungry" -- implications for everyone/everything
- Large organizations, mostly unaccountable, are currently designing a very different future
- These already make up what we can think of as a distributed system optimizing for something...

References

- ESL Chapter 11 (pre-deep, solid foundation)
- CASI Chapter 18
- MLstory chapters on optimization, generalization, deep learning, and datasets
- DLbook, especially part II and chapters 6-9
- Jared Tanner's course on theories of DL
- New paper surveying applications to causality
- Beyond feedforward networks: architectures like RNN, GAN, LSTM, transformer (lecture 10)