Deep Learning

Neural networks became popular in the 1980s.
Lots of successes, hype, and great conferences: NeurIPS, Snowbird.

Then along came SVMs, Random Forests and Boosting in the 1990s, and Neural Networks took a back seat.

Re-emerged around 2010 as *Deep Learning.*
By 2020s very dominant and successful.

Part of success due to vast improvements in computing power, larger training sets, and software: Tensorflow and PyTorch.

Deep Learning

Neural networks became popular in the 1980s.
Lots of successes, hype, and great conferences: NeurIPS, Snowbird.

Then along came SVMs, Random Forests and Boosting in the 1990s, and Neural Networks took a back seat.

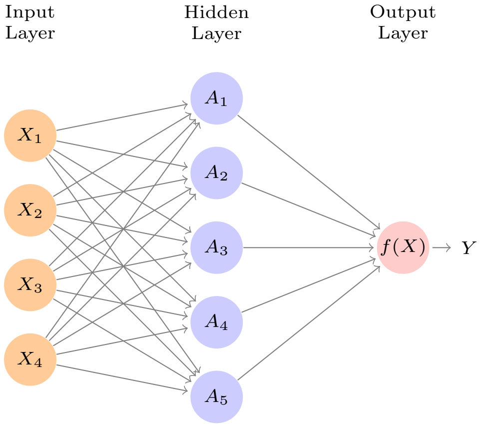
Re-emerged around 2010 as *Deep Learning.*
By 2020s very dominant and successful.

Part of success due to vast improvements in computing power, larger training sets, and software: Tensorflow and PyTorch.

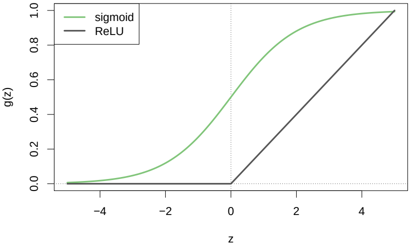
Much of the credit goes to three pioneers and their students: Yann LeCun, Geoffrey Hinton and Yoshua Bengio, who received the 2019 ACM Turing Award for their work in Neural Networks.

Single Layer Neural Network

$$
\begin{aligned}
f(X) &= \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) \\
&= \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j).
\end{aligned}
$$

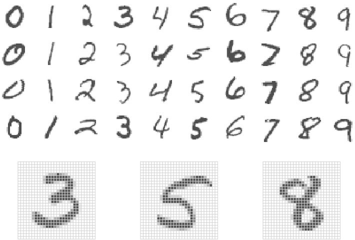


Details



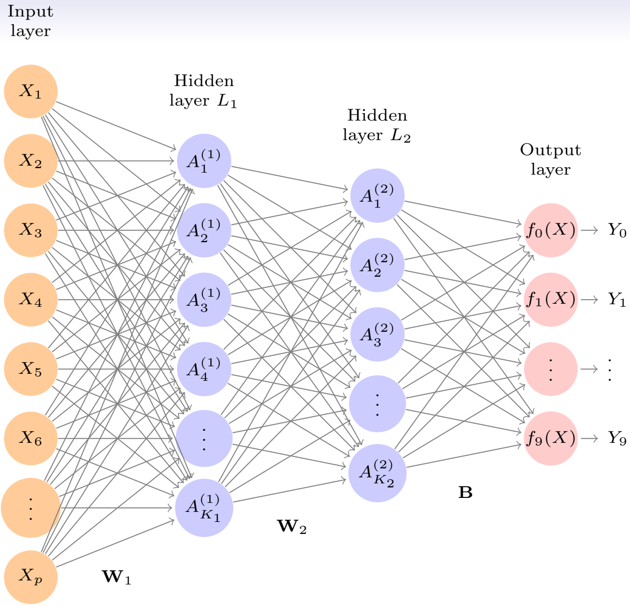
- $A_k = h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$ are called the *activations* in the *hidden layer*.
- $g(z)$ is called the *activation function*. Popular are the *sigmoid* and *rectified linear*, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model.
- So the activations are like derived features — nonlinear transformations of linear combinations of the features.
- The model is fit by minimizing $\sum_{i=1}^{n} (y_i - f(x_i))^2$ (e.g. for regression).

Example: MNIST Digits



Handwritten digits
28×28 grayscale images
$60K$ train, $10K$ test images
Features are the 784 pixel grayscale values $\in (0, 255)$
Labels are the digit class 0–9

- Goal: build a classifier to predict the image class.
- We build a two-layer network with 256 units at first layer, 128 units at second layer, and 10 units at output layer.
- Along with intercepts (called *biases*) there are 235,146 parameters (referred to as *weights*)


Input layer
Hidden layer L_1
Hidden layer L_2
Output layer
X_1
X_2
X_3
X_4
X_5
X_6
$\vdots$
X_p
$A_1^{(1)}$
$A_2^{(1)}$
$A_3^{(1)}$
$A_4^{(1)}$
$\vdots$
$A_{K_1}^{(1)}$
$A_1^{(2)}$
$A_2^{(2)}$
$A_3^{(2)}$
$\vdots$
$A_{K_2}^{(2)}$
$f_0(X)$
$f_1(X)$
$\vdots$
$f_9(X)$
Y_0
Y_1
$\vdots$
Y_9
$\mathbf{W}_1$
$\mathbf{W}_2$
$\mathbf{B}$

Details of Output Layer

- Let $Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)},\ m = 0, 1, \ldots, 9$ be 10 linear combinations of activations at second layer.
- Output activation function encodes the *softmax* function

$$f_m(X) = \Pr(Y = m|X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}}.$$

- We fit the model by minimizing the negative multinomial log-likelihood (or cross-entropy):

$$-\sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} \log(f_m(x_i)).$$

- y_{im} is 1 if true class for observation i is m, else 0 — i.e. *one-hot encoded*.

Results

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

- Early success for neural networks in the 1990s.
- With so many parameters, regularization is essential.
- Some details of regularization and fitting will come later.
- Very overworked problem — best reported rates are $< 0.5\%$!
- Human error rate is reported to be around 0.2%, or 20 of the 10K test images.