INTRODUCTION TO DEEP LEARNING FOR ECONOMICS

LECTURE 0: INTRODUCTION AND GENERAL IDEAS

Michal W. Urdanivia*

*Université de Grenoble Alpes, Faculté d'Économie, GAEL,
e-mail: michal.wong-urdanivia@univ-grenoble-alpes.fr

January 7, 2024

Outline

1. Introduction

General references

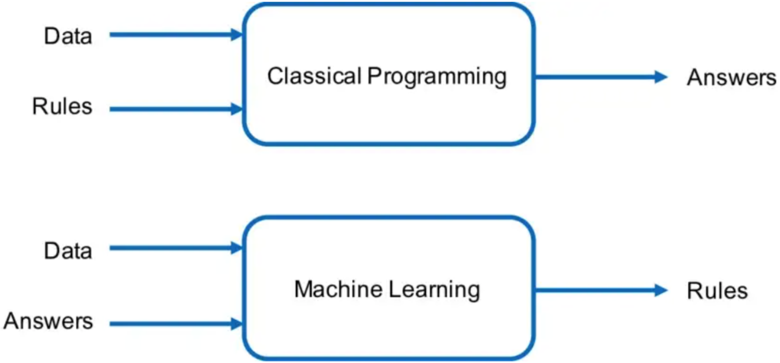
- Goodfellow, Bengio, and Courville (2016) : Deep Learning,
- Zhang et al. (2021) : Dive into Deep Learning,
- Bartlett, Montanari, and Rakhlin (2021) : Deep learning: a statistical viewpoint,
- Bottou, Curtis, and Nocedal (2016): Optimization Methods for Large-Scale Machine Learning.

Machine Learning

- Wide set of algorithms to detect and learn from patterns in the data (observed or simulated):
  - for decision making,
  - and/or forecast future realizations of random variables.
- Focus on recursive processing of information to improve performance over time. item In fact, this is clearer to see in its name in other languages
  - French : **Apprentissage automatique**,
  - Spanish : **aprendizaje automático**,
  - even in English: **Statistical learning**.
- More formally: we use rich datasets to select appropriate functions in a dense functional space.

Introduction

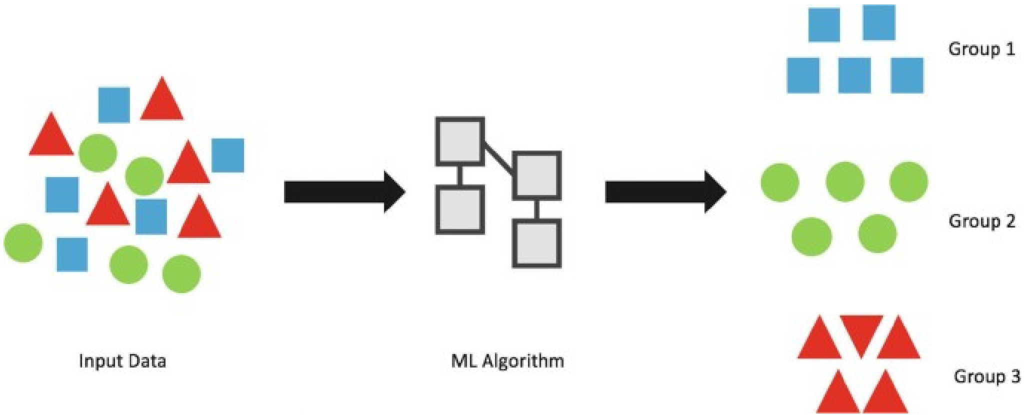
ML and "Classical" programming



Classical Programming vs. Machine Learning

Introduction

Unsupervised ML

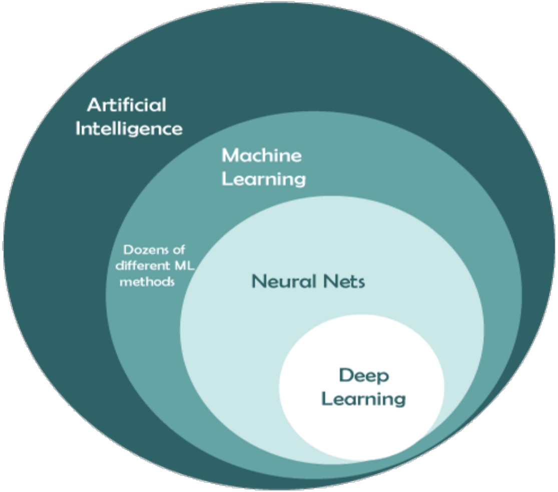


Introduction

AI, ML, Deep Learning

Introduction

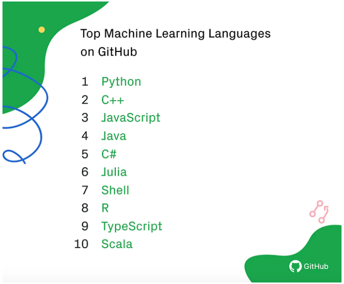
AI, ML, Deep Learning

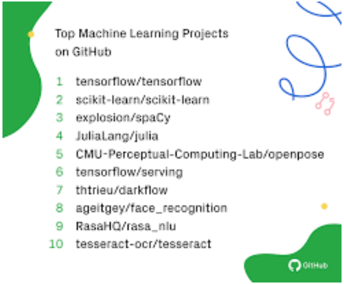


Why now?

- Many of the ideas of machine learning (e.g., basic neural network by (Mcculloch and Pitts, 1943), and perceptron by (Rosenblatt, 1958)) are decades old.
- Previous waves of excitement followed by backlashes.
- Four forces behind the revival;
- Big data.
- Long tails.
- Cheap computational power.
- Algorithmic advances.
- Likely that these four forces will become stronger over time.
- Exponential growth in industry,
    - $\Rightarrow$ plenty of libraries for Python, R, and other languages

Popular Languages(2018)





Some DL libraries

TensorFlow

PyTorch

flux

Caffe

Microsoft Cognitive Toolkit

ML in economics(some examples)

- "Classical" ML:
  - Applied Micro / Microeconometrics(mostly for policy evaluation): Mullainathan and Spiess (2017), Athey and Imbens (2019), Gentzkow, Shapiro, and Taddy (2019), . . .
  - Theoretical econometrics: there are many papers working on good inference practices for method using ML tools, e.g., Chernozhukov et al. (2017), this is an active area of research.
  - Macro : Goulet Coulombe et al. (2022), . . .
- DL:
  - New solution methods for economic models: Azinovic, Gaegauf, and Scheidegger (2022), . . .
  - Alternative to older bounded rationality models: reinforcement learning(e.g., (Finan and Pouzo, 2021)),. . .
  - Alternative empirical models for micro policy evaluation(aka., Treatment Effects): Chernozhukov et al. (2021), Hartford et al. (2017), Farrell, Liang, and Misra (2021), . . .

Outline

2. Formal introduction and general ideas

The problem

All exact science is dominated by the idea of approximation,

Betrand Russell.

- Let us suppose we want to approximate (“learn”) an unknown function

$$y = f(\mathbf{x})$$

  where y is a scalar and $\mathbf{x} := (x_0 = 1, x_1, x_2, ..., x_p)^\top$ a vector (why a constant?).
- We care about the case when$p$ is large (possibly in the thousands!).
- Easy to extend to the case where y is a vector (e.g., a probability distribution), but notation becomes cumbersome.
- In economics, $f(\mathbf{x})$ can be a value function, a policy function, a pricing kernel, a conditional expectation(e.g., a regression function), a classifier, ...

Neural network

- A neural network is an approximation to $f(\cdot)$ of the form:

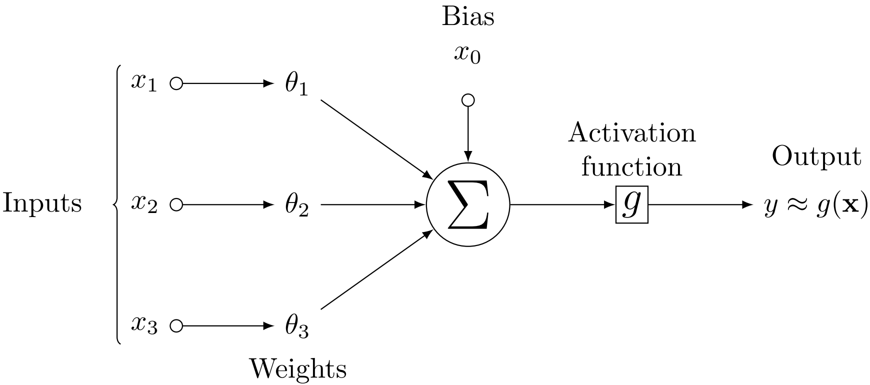
$$y \approx g^{NN}(\mathbf{x}; \boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{k} \theta_j \phi(z_j),$$

  where $\phi(\cdot)$ is an **activation function** and:

$$z_j = \sum_{l=0}^{p} \theta_{l,j} x_l.$$

- The x_l's are known as the features of the data, which belong to a feature space $\mathcal{X}$.
- The $\phi(z_j)$'s are known as the representation of the data.
- k is known as the width of the model (wide vs. thin networks).
- "Training" the network: selecting $\boldsymbol{\theta}$ such that $g^{NN}(\mathbf{x}; \boldsymbol{\theta})$ is as close to $f(\mathbf{x})$ as possible given some relevant metric (e.g., the $\mathcal{L}^2$ norm).

- **Flow representation:**



Comparison with other approximations

- Compare:

$$y \approx g^{NN}(\mathbf{x};\boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{k} \theta_j \phi\left(\sum_{l=0}^{p} \theta_{l,j} x_l\right),$$

  with a standard projection:

$$y \approx g^{CP}(\mathbf{x};\boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{k} \theta_j \phi_j(\mathbf{x}),$$

  where $\phi_j(\cdot)$ is, for example, a Chebyshev polynomial.
- Note:
  - We exchange the rich parameterization of coefficients for the parsimony of basis functions.
  - In a next course, I will explain why this is often a good idea. Suffice it to say now that evaluating a neural network is straightforward.
  - How we determine the coefficients is also different, but this is less important.

Deep learning

- A deep learning (neural) network is a **multilayer** composition of $M > 1$ neural networks:

$$z_j^0 = \theta_{0,j}^0 + \sum_{l=1}^{p} \theta_{l,j}^0 x_l,$$

and

$$z_j^1 = \theta_{0,j}^1 + \sum_{j=1}^{k_1} \theta_j^1 \phi^1(z_j^0)$$

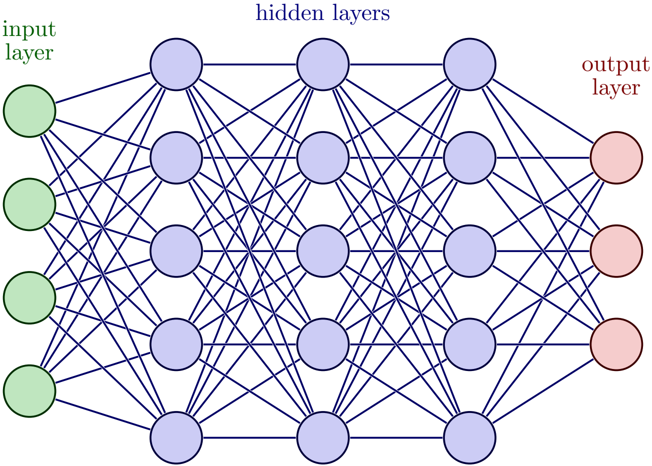
$$\vdots$$

$$y \approx g^{DL}(\mathbf{x}; \boldsymbol{\theta}) = \theta_0^M + \sum_{j=1}^{k_M} \theta_j^M \phi^M(z_j^{M-1}),$$

where the k_1, k_2,..., and $\phi^1(\cdot)$, $\phi_2(\cdot)$, ... are possibly different across each layer of the network.

Deep learning

- **Flow representation:**



Deep learning

- M is known as the depth of the network (deep vs. shallow networks). The case $M = 1$ is the neural network we saw before.
- From now on, we will refer to neural networks as including both single and multilayer networks.
- As before, we select $\boldsymbol{\theta}$ such that $g^{DL}(\mathbf{x}; \boldsymbol{\theta})$ approximates a target function $f(\mathbf{x})$ as closely as possible under some relevant metric.
- We can also add multidimensional outputs.
- Or even to produce a probability distribution as output, for example, using a **softmax layer**:

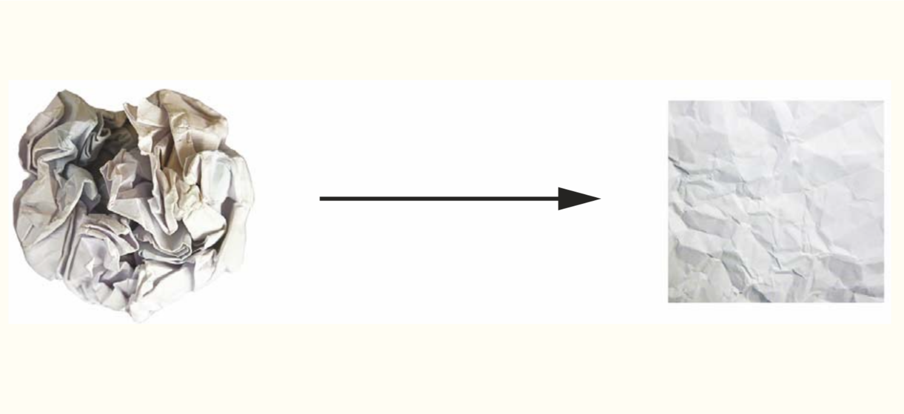
$$y_j = \frac{exp(z_j^{M-1})}{\sum_{j=1}^{k} \exp(z_j^{M-1})}.$$

- All other aspects (selecting $\phi(\cdot)$, M, k, $\ldots$) are known as the network architecture. We will discuss extensively further in the course how to determine them.

Why do neural networks “work”?

- Neural networks consist entirely of chains of tensor operations: we take $\mathbf{x}$, we perform affine transformations, and apply an activation function.
- Thus, these tensor operations are geometric transformations of $\mathbf{x}$.
- In other words: a neural network is a complex geometric transformation in a high-dimensional space.
- Deep neural networks look for convenient geometrical representations of high-dimensional manifolds.
- The success of any functional approximation problem is to search for the right geometric space in which to perform it, not to search for a “better” basis function.

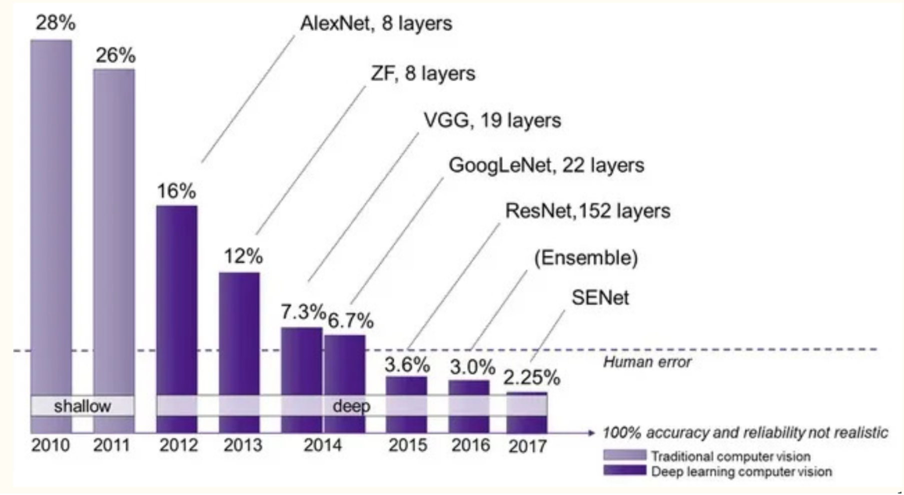
Why do neural networks “work”?



Why do deep neural networks “work” better?

- Why do we want to introduce hidden layers?
  1. It works! Evolution of ImageNet winners.
  2. The number of representations increases exponentially with the number of hidden layers while computational cost grows linearly.
  3. Intuition: hidden layers induce highly nonlinear behavior in the joint creation of representations without the need to have domain knowledge (used, in other algorithms, in some form of greedy pre-processing).

Why do deep neural networks "work" better?



Some consequences

- Because of the previous arguments, neural networks can efficiently approximate extremely complex functions.
- In particular, under certain (relatively weak) conditions:
    1. Neural networks are universal approximators.
    2. Neural networks break the “curse of dimensionality.”
- Furthermore, neural networks are easy to code, stable, and scalable for multiprocessing (neural networks are built around tensors).
- The richness of an ecosystem is key for its long-run success.

Outline

3. Introduction to deep learning for econometrics

Takeaways

- Deep learning is regression with complicated functional forms.
- Design considerations in feedforward networks include depth, width, and the connections between layers.
- Optimization is difficult in deep learning because of
  1. lots of data
  2. and even more parameters
  3. in a highly non-linear model.
- $\Rightarrow$ Specially developed optimization methods.
- Cross-validation for penalization is computationally costly, as well.
- A popular alternative is sample-splitting and early stopping.

Outline

4. Setup: what are neural nets?

Deep Neural Nets

Setup

- Deep learning is (regularized) maximum likelihood, for regressions with complicated functional forms.
- We want, for instance, to find $\boldsymbol{\theta}$ to minimize

$$\mathrm{E}\left[((Y - f(X, \boldsymbol{\theta}))^2\right],$$

for continuous outcomes Y, or to maximize

$$\mathrm{E}\left[\sum_y \mathbf{1}(Y = y) \cdot \log\left(f(X, \boldsymbol{\theta})\right)^2\right]$$

for discrete outcomes Y.

What's deep about that?

Feedforward nets

- Functions f used for deep (feedforward) nets can be written as

$$f(x,\boldsymbol{\theta}) = f^k\left(f^{k-1}\left(\dots f^1(\mathbf{x},\boldsymbol{\theta}^1),\boldsymbol{\theta}^2\right),\dots,\boldsymbol{\theta}^k\right).$$

- Biological analogy:
  - Each value of a component of f j corresponds to the "activation" of a "neuron."
  - Each f^j corresponds to a layer of the net: Many layers $\Rightarrow$ "deep" neural net.
  - The layer-structure and the parameters $\boldsymbol{\theta}$ determine how these neurons are connected.
- Inspired by biology, but practice moved away from biological models.
- Best to think of as a class of nonlinear functions for regression.

So what's new?

- Very non-linear functional forms f. Crucial when
  - mapping pixel colors into an answer to "Is this a cat?,"
  - or when mapping English sentences to Mandarin sentences.
  - Probably less relevant when running Mincer-regressions.
- Often more parameters than observations.
  - Not identified in the usual sense. But we care about predictions, not parameters.
  - Overparametrization helps optimization: Less likely to get stuck in local minima.
- Lots of computational challenges.
  1. Calculating gradients: Backpropagation, stochastic gradient descent.
  2. Searching for optima.
  3. Tuning: Penalization, early stopping.

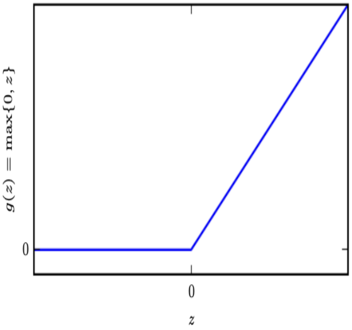
Outline

5. Network design

Network design

Activation functions

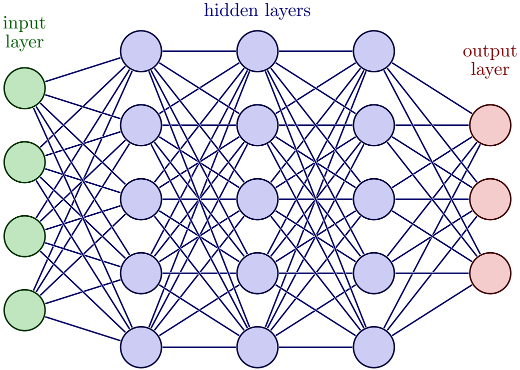
- Basic unit of a net: a neuron i in layer j.
- Receives input vector x_j^i(output of other neurons).
- Produces output $g(x_i^j, \boldsymbol{\theta}_i^j + \boldsymbol{\eta}_i^j)$.
- Activation function $g(\cdot)$:
  - Older nets: Sigmoid function (biologically inspired).
  - Modern nets: “Rectified linear units:” $g\left(\max(0, z)\right)$: More convenient for getting gradients.



Network design

Architecture

- These neurons are connected, usually structured by layers.
  Number of layers: Depth. Number of neurons in a layer: Width.
- Input layer: Regressors.
- Output layer: Outcome variables.
- A typical example:



Network design

Architecture

- Suppose each layer is fully connected to the next, and we are using RELU activation functions.
- Then we can write in matrix notation (using componentwise max):

$$\mathbf{x}^j = f^j\left(\mathbf{x}^{j-1}, \boldsymbol{\theta}^j\right) = \max\left(0, \mathbf{x}^{j-1} \cdot \boldsymbol{\theta}^j + \boldsymbol{\eta}_j\right),$$

- Matrix $\boldsymbol{\theta}^j$:
  - Number of rows: Width of layer $j-1$.
  - Number of columns: Width of layer j.
- Vector $\mathbf{x}^j$:
  - Number of entries: Width of layer j.
- Vector η_j:
  - Number of entries: Width of layer j.
  - Intercepts. Confusingly called “bias” in machine learning.

Network design

Output layer

- Last layer is special: Maps into predictions.
- Leading cases:
  1. Linear predictions for continuous outcome variables,
  $$f^k(\mathbf{x}^{k-1}, \boldsymbol{\theta}^k) = \mathbf{x}^{k-1} \cdot \boldsymbol{\theta}^k.$$
  2. Multinomial logit (aka “softmax”) predictions for discrete variables,
  $$f^{k,y_j}(\mathbf{x}^{k-1}, \boldsymbol{\theta}^k) = \frac{\exp(\mathbf{x}_j^{k-1} \cdot \boldsymbol{\theta}_j^k)}{\sum_{j'} \exp(\mathbf{x}_{j'}^{k-1} \cdot \boldsymbol{\theta}_{j'}^k)}$$
- Network with only output layer: Just run OLS / multinomial logit.

Outline

6. Backpropagation

Backpropagation

The chain rule

- In order to maximize the (penalized) likelihood, we need its gradient.
- Recall that

$$f(x,\boldsymbol{\theta}) = f^k\left(f^{k-1}\left(\dots f^1(\mathbf{x},\boldsymbol{\theta}^1),\boldsymbol{\theta}^2\right),\dots,\boldsymbol{\theta}^k\right).$$

- By the chain rule:

$$\frac{\partial f(\mathbf{x},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i^j} = \left(\prod_{j'=j+1}^{k} \frac{\partial f^{j'}(\mathbf{x}^{j'},\boldsymbol{\theta}^{j})}{\partial \mathbf{x}^{j'-1}}\right) \cdot \frac{\partial f^j(\mathbf{x}^{j-1},\boldsymbol{\theta}^j)}{\partial \boldsymbol{\theta}_i^j}.$$

- A lot of the same terms show up in derivatives w.r.t different $\boldsymbol{\theta}_i^j$:
  - $\mathbf{x}_i^{j'}$ (values of layer j').
  - $\frac{\partial f^{j'}(\mathbf{x}^{j'},\boldsymbol{\theta}^{j})}{\partial \mathbf{x}^{j'-1}}$ (intermediate layer derivatives w.r.t. $\mathbf{x}^{j'-1}$).

Backpropagation

The chain rule

- Denote $\mathbf{z}^j = \mathbf{x}^{j-1}\boldsymbol{\theta}^j + \eta^j$. Recall $\mathbf{x}^j = \max(0, \mathbf{z}^j)$.
- Note $\frac{\partial \mathbf{x}^j}{\partial \mathbf{z}^j} = \mathbf{1}\left(\mathbf{z}^j \geq 0\right)$(componentwise), and $\frac{\partial \mathbf{z}^j}{\partial \boldsymbol{\theta}^j} = \mathbf{x}^{j-1}$.
- First, **forward propagation**: Calculate all the $\mathbf{z}^j$ and $\mathbf{x}^j$ starting at $j = 1$
- Then **backpropagation**: Iterate backward, starting at $j = k$;
  1. Calculate and store
  $$\frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}^{j-1}} = \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}^j} \cdot \mathbf{1}(\mathbf{z}^j \geq 0)\boldsymbol{\theta}^{j\prime}.$$
  2. Calculate
  $$\frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}^j} = \frac{\partial f(\mathbf{x}, \boldsymbol{\theta})}{\partial \mathbf{x}^j} \cdot \mathbf{1}(\mathbf{z}^j \geq 0)\mathbf{x}^{j-1}.$$

Backpropagation

Advantages

- Backpropagation improves efficiency by **storing** intermediate derivatives, **rather than recomputing** them.
- Number of computations grows only linearly in number of parameters.
- The algorithm is easily generalized to more complicated network architectures and activation functions.
- Parallelizable across observations in the data (one gradient for each observation!).

References

Athey, Susan and Guido W. Imbens. 2019. "Machine Learning Methods That Economists Should Know About." *Annual Review of Economics* 11 (1):685–725.

Azinovic, Marlon, Luca Gaegauf, and Simon Scheidegger. 2022. "DEEP EQUILIBRIUM NETS." *International Economic Review* 63 (4):1471–1525.

Bartlett, Peter L., Andrea Montanari, and Alexander Rakhlin. 2021. "Deep learning: a statistical viewpoint." URL `https://arxiv.org/abs/2103.09177`.

Bottou, Léon, Frank E. Curtis, and Jorge Nocedal. 2016. "Optimization Methods for Large-Scale Machine Learning." URL `https://arxiv.org/abs/1606.04838`.

Chernozhukov, Victor, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, and Whitney Newey. 2017. "Double/Debiased/Neyman Machine Learning of Treatment Effects." *American Economic Review* 107 (5):261–65.

Chernozhukov, Victor, Whitney K. Newey, Victor Quintas-Martinez, and Vasilis Syrgkanis. 2021. "RieszNet and ForestRiesz: Automatic Debiased Machine Learning with Neural Nets and Random Forests."

References

Farrell, Max H., Tengyuan Liang, and Sanjog Misra. 2021. “Deep Neural Networks for Estimation and Inference.” *Econometrica* 89 (1):181–213. URL https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA16901.

Finan, Frederico and Demian Pouzo. 2021. “Reinforcing RCTs with Multiple Priors while Learning about External Validity.”

Gentzkow, Matthew, Jesse M. Shapiro, and Matt Taddy. 2019. “Measuring Group Differences in High-Dimensional Choices: Method and Application to Congressional Speech.” *Econometrica* 87 (4):1307–1340.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Goulet Coulombe, Philippe, Maxime Leroux, Dalibor Stevanovic, and Stéphane Surprenant. 2022. “How is machine learning useful for macroeconomic forecasting?” *Journal of Applied Econometrics* 37 (5):920–964.

Hartford, Jason, Greg Lewis, Kevin Leyton-Brown, and Matt Taddy. 2017. “Deep IV: A Flexible Approach for Counterfactual Prediction.” In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML’17. JMLR.org, 1414–1423.

References

Mcculloch, Warren and Walter Pitts. 1943. "A Logical Calculus of Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics* 5:127–147.

Mullainathan, Sendhil and Jann Spiess. 2017. "Machine Learning: An Applied Econometric Approach." *Journal of Economic Perspectives* 31 (2):87–106.

Rosenblatt, Frank. 1958. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65 6:386–408.

Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola. 2021. "Dive into Deep Learning." URL `https://arxiv.org/abs/2106.11342`.