Econ 2148, spring 2019
Deep Neural Nets

Maximilian Kasy

Department of Economics, Harvard University

Agenda

- What are neural nets?
- Network design:
  - Activation functions,
  - network architecture,
  - output layers.
- Calculating gradients for optimization:
  - Backpropagation,
  - stochastic gradient descent.
- Regularization using early stopping.

Takeaways for this part of class

- 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.

Deep Neural Nets

Setup

- Deep learning is (regularized) maximum likelihood, for regressions with complicated functional forms.
- We want, for instance, to find θ to minimize

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

for continuous outcomes Y, or to maximize

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

for discrete outcomes Y.

What's deep about that?

Feedforward nets

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

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

- 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 θ 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.

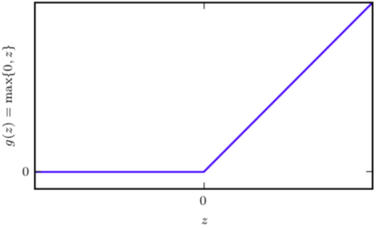
So what's new?

- 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.

Network design

Activation functions

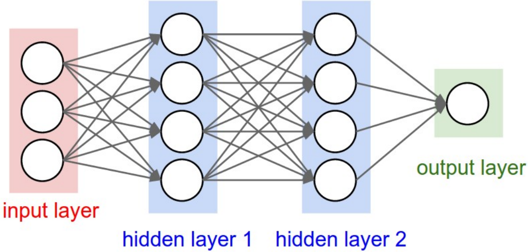
- Basic unit of a net: a neuron i in layer j.
- Receives input vector x_i^j (output of other neurons).
- Produces output $g(x_i^j \theta_i^j + \eta_i^j)$.
- Activation function $g(\cdot)$:
  - Older nets: Sigmoid function (biologically inspired).
  - Modern nets: "Rectified linear units:" $g(z) = \max(0, z)$. 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):

$$\boldsymbol{x}^j = f^j(\boldsymbol{x}^{j-1}, \theta^j) = \max(0, \boldsymbol{x}^{j-1} \cdot \theta^j + \eta_j)$$

- Matrix θ^j:
  - Number of rows: Width of layer $j-1$.
  - Number of columns: Width of layer j.
- Vector $\boldsymbol{x}^j$:
  - 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,

$$f^k(x^{k-1}, \theta^k) = x^{k-1} \cdot \theta^k.$$

  2. Multinomial logit (aka "softmax") predictions for discrete variables,

$$f^{ky_j}(x^{k-1}, \theta^k) = \frac{\exp(x_j^{k-1} \cdot \theta_j^k)}{\sum_{j'} \exp(x_{j'}^{k-1} \cdot \theta_{j'}^k)}$$

- Network with only output layer: Just run OLS / multinomial logit.

The gradient of the likelihood

Practice problem

Consider a fully connected feedforward net with rectified linear unit activation functions.

1. Write out the derivative of its likelihood, for *n* observations, with respect to any parameter.
2. Are there terms that show up repeatedly, for different parameters?
3. In what sequence would you calculate the derivatives, in order to minimize repeat calculations?
4. Could you parallelize the calculation of derivatives?

Backpropagation

The chain rule

- In order to maximize the (penalized) likelihood, we need its gradient.
- Recall $f(\boldsymbol{x}, \theta) = f^k(f^{k-1}(\ldots f^1(\boldsymbol{x}, \theta^1), \theta^2), \ldots, \theta^k)$.
- By the **chain rule**:

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

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

Backpropagation

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

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!).

Stochastic gradient descent

- Gradient descent updates parameter estimates in the direction of steepest descent:

$$g_t = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta m(X_i, Y_i, \theta)$$
$$\theta_{t+1} = \theta_t - \varepsilon_t g_t.$$

- Stochastic gradient descent (SGD) does the same, but instead uses just a random subsample $B_t = \{i_1^t, \ldots, i_b^t\}$ (changing across t) of the data:

$$\hat{g}_t = \frac{1}{b} \sum_{i \in B_t} \nabla_\theta m(X_i, Y_i, \theta)$$
$$\theta_{t+1} = \theta_t - \varepsilon_t \hat{g}_t.$$

Stochastic gradient descent

- We can do this because the full gradient is a sum of gradients for each observation.
- Typically, the batches B_t cycle through the full dataset.
- If the learning rate ε_t is chosen well, some convergence guarantees exist.
- The built-in randomness might help avoiding local minima.
- Extension: SGD with **momentum**,

$$\begin{aligned} v_t &= \alpha v_{t-1} - \varepsilon_t \hat{g}_t, \\ \theta_{t+1} &= \theta_t + v_t. \end{aligned}$$

- Initialization matters. Often start from previously trained networks.

Regularization for neural nets

- To get good predictive performance, neural nets need to be regularized.
- As before, this can be done using **penalties** such as $\lambda\|\theta\|_2^2$ ("Ridge") or $\lambda\|\theta\|_1$ ("Lasso").
- Problem: Tuning using cross-validation is often computationally too costly for deep nets.
- An alternative regularization method is **early stopping**:
  - Split the data into a training and a validation sample.
  - Run gradient-based optimization method on the training sample.
  - At each iteration, calculate prediction loss in the validation sample.
  - Stop optimization algorithm when this prediction loss starts increasing.

References

- *Goodfellow, I., Bengio, Y., and Courville, A. (2016).* Deep learning. *MIT Press.*

These slides are mostly based on chapters 6-8 of this book.