Boosting, Deep Learning and Neural Nets

American Economic Association,

Continuing Education Program 2018

Machine Learning and Econometrics, Lecture 7

Tuesday January 10th, 8.00-9.15am

Guido Imbens - Stanford University

Outline

1. Regression and Classification

2. Boosting

3. Deep Neural Networks

4. The Backpropagation Algorithm

5. Stochastic Gradient Descent

1. Regression and Classification

Scalar outcome y_i, p-dimensional vector of covariates/inputs/features x_i, l-th element equal to x_{il}.

We focus on two cases here:

- **(Regression)** Outcome y_i is ordered, possibly continuous, e.g., expenditure, earnings.

- **(Classification)** Outcome y_i is element of finite unordered set, e.g., choice of item.

I. In the **regression** problem we are interested in predicting the value of y_i, with a focus on out-of-sample prediction.

Given a data set, (x_i, y_i), $i = 1, \ldots, N$, we split it into two parts, a **training** sample, say the first N_{train} observations, and and a **test** sample, the last N_{test} observations, we are interested in using the training sample (but not the test sample) to choose a function $f : \mathbb{X} \mapsto \mathbb{R}$ that minimizes the average squared error in the test sample:

$$\frac{1}{N_{\text{test}}} \sum_{i=N_{\text{train}}+1}^{N} \Big(y_i - f(x_i)\Big)^2$$

• From a conventional econometric/statistical perspective, if we think that the original sample is a random sample from a large population, so we can think of the (x_i, y_i) as realizations of random variables (X_i, Y_i), then the function that minimizes the objective is the conditional expectation of the outcome given the covariates/features:

$$f(x) = \mathbb{E}[Y_i | X_i = x]$$

In that case we can think of the problem as just using the training sample to get the optimal (in terms of expected squared error) estimator for this conditional expectation.

There are two conceptual points to note with regard to that perspective:

• We do not care separately about bias and variance, just about expected squared error. In the same vein, we do not necessarily care about inference.

• We do not **need** to think about where the sample came from (e.g., the infinite superpopulation). The problem of finding a good predictor is well-defined irrespective of that.

II. In the **classification** problem we are interested in classifying observations on the basis of their covariate/feature values x_i in terms of the outcome y_i. Suppose $y_i \in \{1, \ldots, M\}$.

Again we wish to come up with a function $f : \mathbb{X} : \{1, \ldots, M\}$ that minimizes the mis-classification rate in the test sample:

$$\frac{1}{N_{\text{test}}} \sum_{i=N_{\text{train}}+1}^{N} \mathbf{1}_{y_i \neq f(x_i)}$$

Suppose again that the sample is a random sample from an infinite population, and that in the population the fraction of units with $Y_i = m$ in the subpopulation of units with $X_i = x$ is

$$p_m(x) = \text{pr}(Y_i | X_i = x)$$

In that case the solution is

$$f(x) = m \quad \text{iff } p_m(x) = \max_{k=1}^{M} p_k(x)$$

Note that we are **not** actually focused here on estimating $p_m(x)$, just the index of the choice with the maximum probability.

• Knowing $p_m(\cdot)$ solves the problem, but it is **not** necessary to estimate $p_m(\cdot)$.

Consider the classical example of the classification problem, the zipcode problem:

Given an image of a digit, with the accompanying label (from zero to nine), classify it as a 0-9.

• Thinking of this as a problem of estimating a conditional probability is not necessarily natural: it is not clear what the conditional probabilities mean. Although, from a Bayesian perspective one might improve things by taking into account the other digits in a handwritten zipcode.

• Lucas critique type issues: People may change their handwriting if they know delivery speed is dependent on the ability of the machine to be able to read the zipcode.

Suppose that $J = 2$, so we are in a setting with a binary outcome, where the regression and classification problem are the same in the standard econometric/statistics approach.

Suppose that the sample is a random sample from a large population, with the conditional probability that $Y_i = 2$ equal to $p_2(x) = 1 - p_1(x)$.

Viewing the problem as a regression problem the solution is

$$f(x) = 1 + p_2(x) \quad \text{with } f(x) \in [1, 2]$$

The solution to the classification problem is slightly different:

$$f(x) = 1 + \mathbf{1}_{p_2(x) > 1/2} \quad \text{with } f(x) \in \{1, 2\}$$

2. Back to the Regression Problem: Gradient Boosting

Initial estimate $\widehat{f}_0(x) = 0$.

First estimate $f(\cdot)$ using a very simple method. For example, a regression tree with a single split on $(y_i - \widehat{y}_0(x_i), x_i)$. Call this estimate $\widehat{g}_1(x)$, and define $\widehat{g}_1(x) = \widehat{g}_0(x) + \widehat{g}_1(x)$

Then calculate the residual $\widehat{\varepsilon}_{1i} = y_i - \widehat{f}_1(x_i)$.

Apply the same simply method again to $(\widehat{\varepsilon}_{1i}, x_i)$, with estimator $\widehat{g}_2(x)$. The estimator for $f(x)$ is now $\widehat{f}_2(x) = \widehat{f}_1(x) + \widehat{g}_2(x)$.

Apply the same simply method again to $\widehat{\varepsilon}_{2i} = y_i - \widehat{f}_2(X_i)$, with estimator $\widehat{g}_3(x)$. The estimator for $f(x)$ is now $\widehat{f}_3(x) = \widehat{f}_2(x) + \widehat{g}_3(x)$.

What does this do?

Each $\widehat{g}_k(x)$ depends only on a single element of x (single covariate/feature).

Hence $\widehat{f}(x)$ is always an additive function of $x_1, \ldots, x_p$.

What if we want the approximation to allow for some but not all higher order interactions?

If we want only first order interactions, we can use a **base learner** that allows for two splits. Then the approximation allows for the sum of general functions of two variables, but not more.

Boosting refers to the repeated use of a simple basic estimation method, repeatedly applied to the residuals.

Can use methods other than trees as base learners.

For each split, we can calculate the improvement in mean squared error, and assign that to the variable that we split on.

Sum this up over all splits, and over all trees.

This is informative about the importance of the different variables in the prediction.

Modification

Three tuning parameters: number of trees B, depth of trees d, and shrinkage factor $\varepsilon \in (0, 1]$.

Initial estimate $\widehat{f}_0(x) = 0$, for all x.

First grow tree of depth d on $(y_i - \widehat{f}_0(x_i), x_i)$, call this $\widehat{g}_1(x)$.

New estimate: $\widehat{f}_1(x) = \widehat{f}_0(x) + \eta \widehat{g}_1(x)$.

Next, grow tree of depth d on $(Y_i - \widehat{f}_b(X_i), X_i)$, call this $\widehat{g}_{b+1}(x)$.

$\eta = 1$ is regular boosting. $\eta < 1$ slows down learning, spreads importance around more variables.

Generalized Boosting

We can do this in more general settings. Suppose we are interested in estimating a binary response model, with a high-dimensional covariate. Start again with

$\widehat{f}_0(x) = 0$ specify parametric model $g(x;\gamma)$

Minimize over γ : $\sum_{i=1}^{N} L(Y_i, \widehat{f}_0(X_i) + g(X_i;\gamma))$

and update $\widehat{f}_k(x) = \widehat{f}_k(x) + \eta g(x;\widehat{\gamma})$

$L(\cdot)$ could be log likelihood with g log odds ratio:

$$L(y,g) = y\,(g - \ln(1+\exp(g))) - (1-y)\ln(1+\exp(g))$$

3. Deep Learning and Deep Neural Nets

Think back to Linear Model for $f(\cdot)$:

$$f(x) = \omega_0 + \sum_{j=1}^{p} \omega_j x_j$$

Minimize sum of squared deviations over "weights" (parameters) ω_j:

$$\min_{\omega_0,\ldots,\omega_p} \sum_{i=1}^{N_{\text{train}}} \left(Y_i - \omega_0 - \sum_{j=1}^{p} \omega_j X_{ij} \right)^2$$

This is not very flexible, but has well-understood properties.

Let's make this more flexible, short of being completely non-parametric and using kernel regression.

Single index model:

$$f(x) = g\left(\sum_{j=1}^{p} \omega_j x_j\right)$$

Estimate both weights ω_j and transformation $g(\cdot)$

Additive model:

$$f(x) = \sum_{j=1}^{p} g_j(x_j)$$

Estimate the p feature-specific transformations $g_j(\cdot)$.

Advantage over nonparametric kernel regression is the improved rate of convergence (rate of convergence does not depend on number of covariates, whereas for kernel regression the rate of convergence slows down the larger the dimension of x_i, p, is).

Projection Pursuit (hybrid of single index model and additive models):

$$f(x) = \sum_{l=1}^{L} g_l \left(\sum_{j=1}^{p} \omega_{lj} x_j \right)$$

Estimate both the L transformations $g_l(\cdot)$ and the $L \times p$ weights ω_{lj}.

Neural Net with single hidden layer:

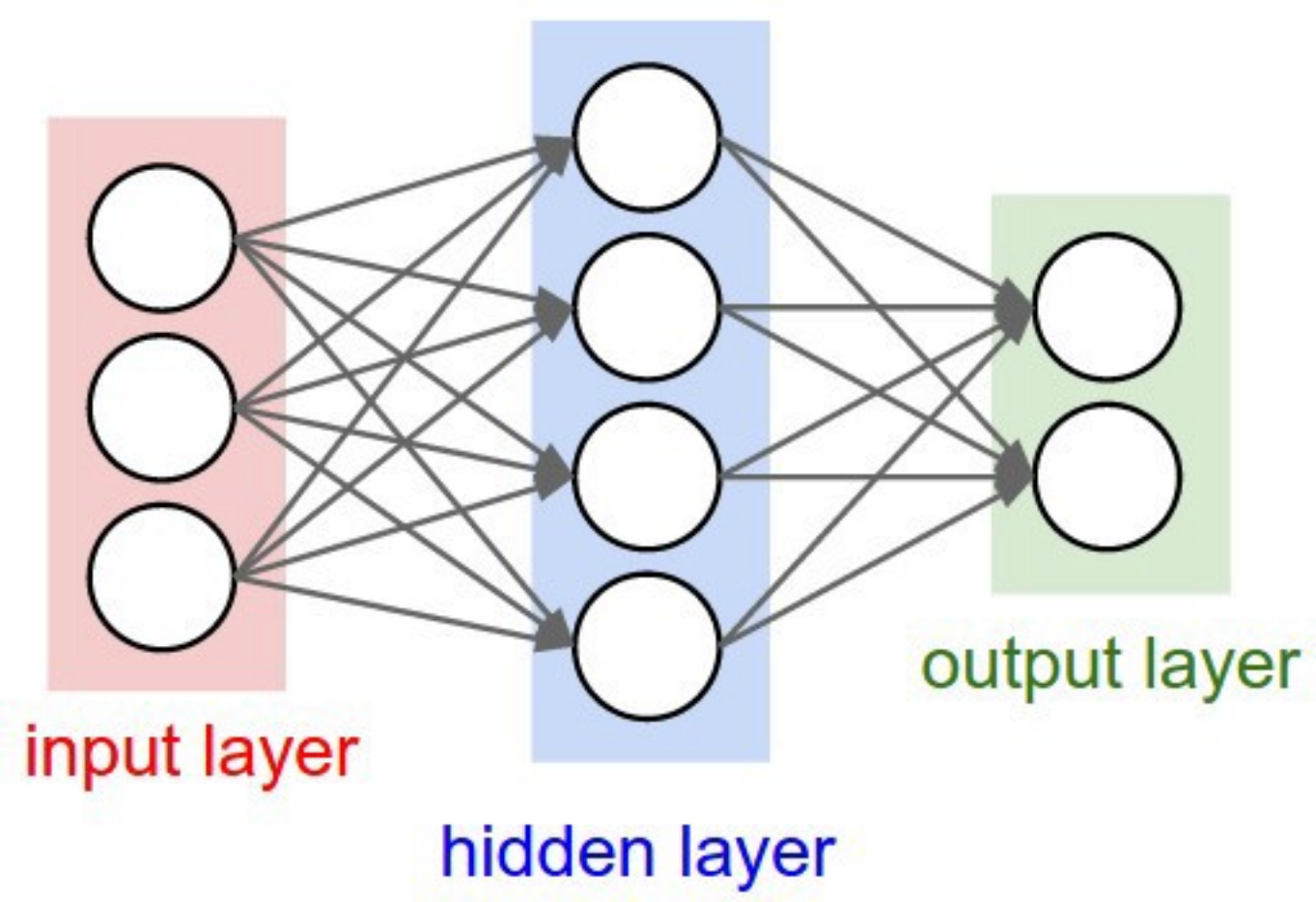
$$f(x) = \omega_0^{(2)} + \sum_{l=1}^{L} \omega_l^{(2)} g\left(\sum_{j=1}^{p} \omega_{lj}^{(1)} x_j\right)$$

Fix transformation $g(\cdot)$ and estimate only the weights $\omega_{lj}^{(1)}$ and $\omega_l^{(2)}$ (but in both layers).

• Increased flexibility in one place, at the expense of flexibility in other places.

• Computational advantages especially when generalized to multiple hidden layers.

• Parametric model, but may require messy normalizations for identification depending on choice for $g(\cdot)$.


input layer
hidden layer
output layer

General neural net with K hidden layers, one observed input layer, one observed output layer, $K+2$ layers total.

Need additional notation to keep things tractable. Superscripts in parentheses denote layers, from (1) to $(K+2)$.

$p_1 = p$ observed inputs (dimension of x). Define $\alpha_l^{(1)} = z_l^{(1)} = x_l$ for $l = 1, \ldots, p_1$.

First hidden layer: p_2 hidden elements $z_1^{(2)}, \ldots, z_{p_2}^{(2)}$

$$z_l^{(2)} = \omega_{l0}^{(1)} + \sum_{j=1}^{p_1} \omega_{lj}^{(1)} \alpha_j^{(1)} = \omega_{l0}^{(1)} + \sum_{j=1}^{p_1} \omega_{lj}^{(1)} x_j, \qquad l = 1, \ldots, p_2$$

$$\alpha_l^{(2)} = g\left(z_l^{(2)}\right) \qquad \text{(with transformation } g(\cdot) \text{ known)}$$

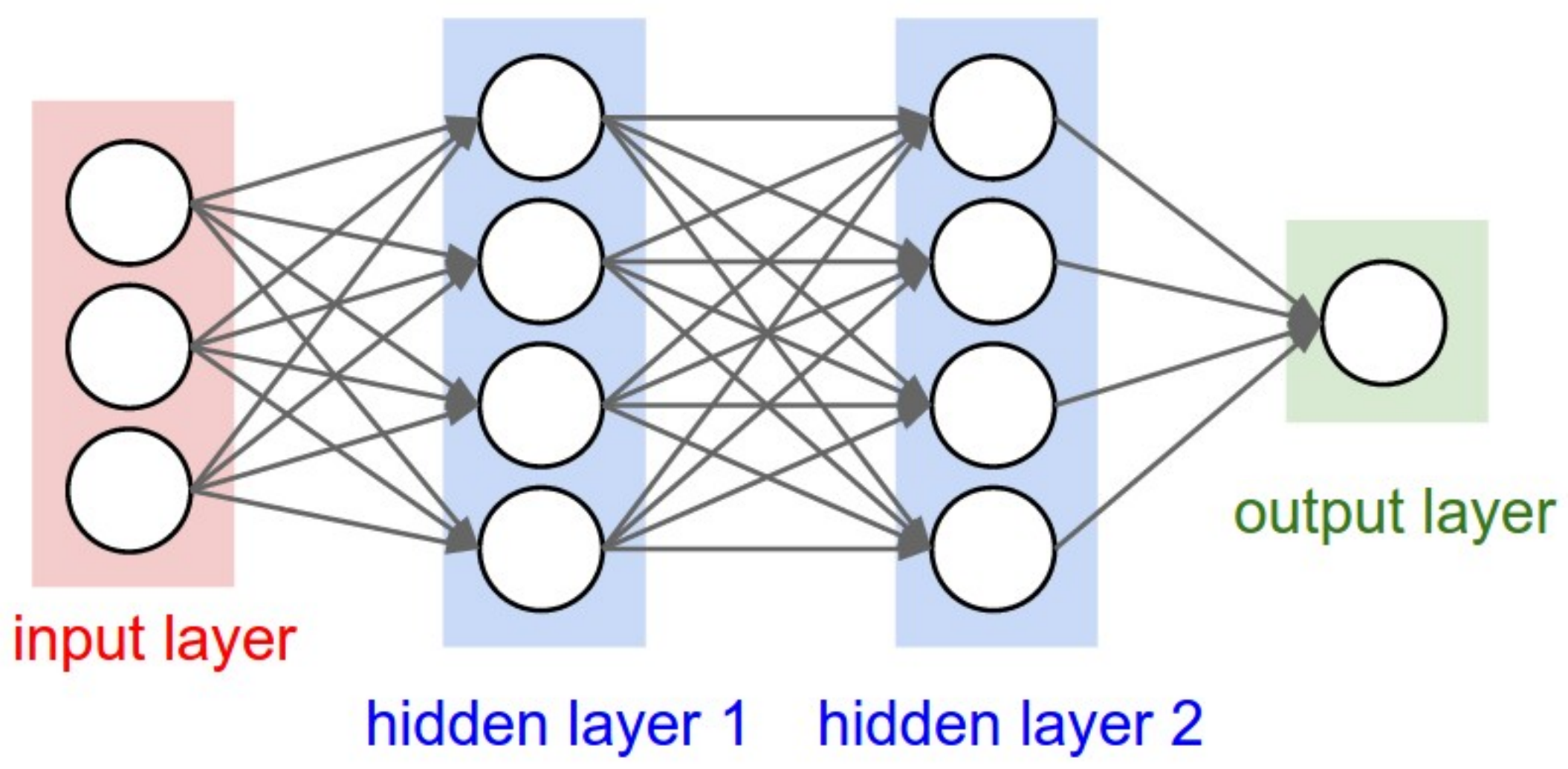
Hidden layer k, with p_k hidden elements $z_1^{(k)}, \ldots, z_{p_k}^{(k)}$, for $k = 2, \ldots, K+1$

$$z_l^{(k)} = \omega_{l0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} \omega_{lj}^{(k-1)} \alpha_j^{(k-1)}, \quad \alpha_l^{(k)} = g\left(z_l^{(k)}\right)$$

Final output layer (layer $K+2$) with $p_{K+2} = 1$:

$$z_1^{(K+2)} = \omega_{10}^{(K+1)} + \sum_{j=1}^{p_{K+1}} \omega_{1j}^{(K+1)} \alpha_j^{(K+1)},$$

$$f(x;\omega) = \alpha_1^{(K+2)} = g^{(K+2)}\left(z_1^{(K+2)}\right) = z_1^{(K+2)}$$


input layer
hidden layer 1
hidden layer 2
output layer

Choices for $g(\cdot)$ (pre-specified, **not** choosen by formal data-driven optimization):

1. sigmoid: $g(a) = (1 + \exp(-a))^{-1}$

2. tanh: $g(a) = (\exp(a) - \exp(-a))/(\exp(a) + \exp(-a))$

3. rectified linear $g(a) = a \times \mathbf{1}_{a>0}$

4. leaky rectified linear $g(a) = a \times \mathbf{1}_{a>0} + \gamma \times \mathbf{1}_{a<0}$

Lost of complexity allowed for, but comes with lots of choices.

Not easy to use out-of-the-box, but very succesful in complex settings.

Computationally tricky because of multi-modality.

• can approximate smooth functions accurately (universal approximator) with many layers and many hidden units.

• complex interplay between additional flexibility through more hidden layers or through more hidden elements for each layer.

• Structure can be imposed by having $z_l^{(k)}$ not depend on all $\alpha_j^{(k-1)}$, $j = 1, \ldots, p_{k=1}$.

Interpretation

We can think of the layers up to the last one as a data-driven way of constructing regressors: $z^{(K+1)} = h(\omega, x)$

Alternative is to directly choose functions of regressors, e.g., polynomials $z_{ij} = x_{i1} \times x_{i4} \times x_{i7}^2$.

In what sense is this better? Is this a statement about the type of functions we encounter?

Multiple layers versus multiple hidden units

"We observe that shallow models [models with few layers] in this context overfit at around 20 millions parameters while deep ones can benefit from having over 60 million. This suggests that using a deep model expresses a useful preference over the space of functions the model can learn." *Deep Learning*, Goodfellow, Bengio, and Courville, 2016, MIT Press, p. 197.

Link to standard parametric methods

The sequential set up implies a (complicated) par function

$$f(x;\omega)$$

We could simply take this as parametric model for $\mathbb{E}[Y_i|X_i = x]$.

• Then we can estimate ω by minimizing, over all $\omega_{lj}^{(k)}$, $j = 1,\ldots,p_k$, $l = 1,\ldots,p_{k+1}$, $k = 1,\ldots,K+1$, using nonlinear optimization methods (e.g., Newton-Raphson):

$$\sum_{i=1}^{N}\Big(y_i - f(x_i;\omega)\Big)^2$$

• Linearize to do standard inference for nonlinear least squares and get asymptotic normality for $\omega_{lj}^{(k)}$ and predicted values given model specification.

That is not what this literature is about:

• Computationally messy, multiple modes, too many parameters to calculate second derivatives.

• Too many parameters, with the model too non-linear, so that asymptotic normality is unlikely to be accurate approximation.

• Unlikely to be identified without restrictions, e.g., normalizations, that may be computationally messy.

Instead, first add regularization over parameter space:

Minimize, over all $\omega_{lj}^{(k)}$,
$j = 1, \ldots, p_k$, $l = 1, \ldots, p_{k+1}$, $k = 1, \ldots, K+1$:

$$\sum_{i=1}^{N} \Big(y_i - f(x_i; \omega) \Big)^2 + \text{penalty term}$$

where the penalty term is something like a LASSO penalty, or similar:

$$\lambda \times \sum_{k=1}^{K} \sum_{j=1}^{p_k} \sum_{l=1}^{p_{k-1}} \left(\omega_{jl}^{(k)} \right)^2$$

Choose λ through cross-validation methods.

4. Estimating the Parameters of a Neural Network: Back-propagation Algorithm

Define objective function (initially without regularization)

$$J_i(\omega, x, y) = \Big(y_i - f(x_i; \omega)\Big)^2$$

$$J(\omega, \mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} J_i(\omega, x_i, y_i) = \sum_{i=1}^{N} \Big(y_i - f(x_i; \omega)\Big)^2$$

We wish to minimize this over ω.

• How do we calculate first derivatives (at least approximately)?

• Forget about second cross derivatives (too many, too difficult)!

The **backpropagation** algorithm calculates the derivatives of $J_i(\omega, x_i, y_i)$ with respect to all $\omega_{li}^{(k)}$.

You start at the input layer and calculate the hidden elements $z_{lj}^{(k)}$, one layer at a time.

Then after calculating all the current values of the hidden elements, we calculate the derivatives, starting with the derivatives for the $\omega^{(k)}$ in the last, output layer, working backwards to the first layer.

Key is that the ω only enter in one layer each.

Recall layer k,

$$z_l^{(k)} = \omega_{l0}^{(k-1)} + \sum_{j=1}^{p_{k-1}} \omega_{lj}^{(k-1)} g^{(k)}\left(z_l^{(k-1)}\right)$$

or in vector notation:

$$\mathbf{z}^{(k)} = h^{(k)}\left(\omega^{(k-1)}, \mathbf{z}^{(k-1)}\right)$$

- This function $h^{(k)}(\cdot)$ does not depend on ω beyond $\omega^{(k-1)}$.

Hence we can write the function $f(\cdot)$ recursively:

$$f(x;\omega) = h^{(K+2)}\left(\omega^{(K+1)}, h^{(K+1)}\left(\omega^{(K)}, \ldots . h^{(2)}\left(\omega^{(1)}, \mathbf{x}\right)\ldots\right)\right)$$

Now start at the last layer where

$$h^{(K+2)}(z^{(K+2)}) = z^{(K+2)}.$$

Define for observation i:

$$\delta_i^{K+2} = \frac{\partial}{\partial z_i^{(K+2)}} \left(y_i - h^{(K+2)} \left(z_i^{(K+2)} \right) \right)^2$$

$$= -2 \left(y_i - h \left(z_i^{(K+2)} \right) \right) h^{(K+2)\prime} \left(z_i^{(K+2)} \right)$$

$$= -2 \left(y_i - z_i^{(K+2)} \right)$$

(this is just the scaled residual).

Down the layers:

Define $\delta_{li}^{(k)}$ in terms of $\delta^{(k+1)}$ and $z^{(k)}$ as a linear combination of the $\delta^{(k+1)}$:

$$\delta_{li}^{(k)} = \left(\sum_{j=1}^{p_k} \omega_{jl}^{(k)} \delta_{li}^{(k+1)} \right) g' \left(z_{li}^{(k)} \right)$$

Then the derivatives are

$$\frac{\partial}{\partial \omega_{lj}^{(k)}} J_i(\omega, x_i, y_i) = g \left(z_{li}^{(k)} \right) \delta_{li}^{(k+1)}$$

Given the derivatives, iterate over

$$\omega_{m+1} = \omega_m - \alpha \times \frac{\partial}{\partial \omega} J(\omega, \mathbf{x}, \mathbf{y})$$

α is the "learning rate" often set at 0.01.

This is still computationally (unnecessarily) demanding:

• given that we dont do optimal line search, we do not need the exact derivative.

5. stochastic gradient descent: Instead of calculating the actual derivative,

$$\frac{\partial}{\partial \omega} J(\omega, \mathbf{x}, \mathbf{y})$$

simplify the computations by **estimating** the derivative:

$$\sum_{i=1}^{N} R_i \frac{\partial}{\partial \omega} J(\omega, x_i, y_i) \Big/ \sum_{i=1}^{N} R_i$$

for a random selection of units ($R_i \in \{0, 1\}$) because it is faster.

Regularization to avoid overfitting:

- add penalty term, $\lambda \sum_{jlk} \left(\omega_{jl}^{(k)}\right)^2$, or $\lambda \sum_{jlk} \left|\omega_{jl}^{(k)}\right|$

- early stopping rule: monitor average error in test sample, and stop iterating when average test error deteriorates.

References

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

Hartford, J., G. Lewis, K. Leyton-Brown, & M. Taddy, (2016), "Counterfactual Prediction with Deep Instrumental Variables Networks", arXiv:1612.09596

LeCun, Y., Y. Bengio & G. Hinton, (2015) "Deep learning" *Nature*, Vol. 521(7553): 436-444.

Schapire, Robert E., and Yoav Freund (2012). *Boosting: Foundations and algorithm*, MIT press.