Deep learning

2. Neural networks

Clément Gorin
clement.gorin@univ-paris1.fr

Harbin Institute of Technology, Shenzhen
August 2024

Introduction

Introduction

Neural networks are a family of flexible predictive statistical models with a network structure

- This lecture introduces the most fundamental model structure called the feed-forward network
- Specialised architectures for image (CNN), language (RNN), generative (GAN) modelling, Transformers, etc.
- State-of-the-art performance in problems involving with high-dimensional and unstructured data

Introduction

Networks are called “neural” because their basic structure and functioning were loosely inspired by neuroscience

- This analogy is no longer relevant as developments follows mathematical and engineering principles
- Early implementations in the 1980’s but networks have become popular over the past decade
- Computing power and algorithmic innovations made networks more capable and easier to estimate

Structure

Units

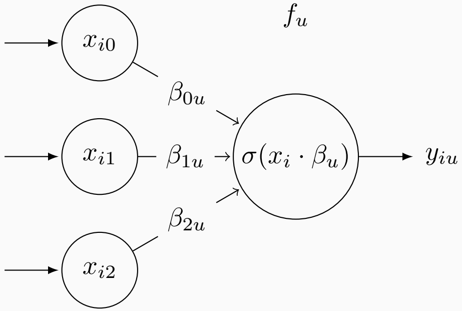
A network is composed of many simple non-linear functions called neurons or units denoted by u

$$y_{iu} = \sigma(x_{i \leftarrow x_0} \cdot \beta_u) \tag{1}$$

where $x_{i \leftarrow x_0}$ and β_u are vectors of length k_{+1}, $\leftarrow$ denotes concatenation and σ is a non-linear function called activation

- A unit performs a dot product between a vector of variables and a vector of parameters
- Early networks used a sigmoid activation function, so a unit can be seen as a logistic regression model

Network unit with two inputs and a constant



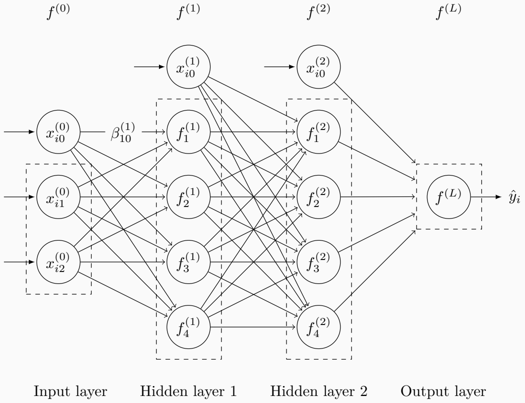
Networks can be represented as directed graphs. Circles are network units, simple arrows are parameters and plain arrows are the unit's inputs and output. For reasons that will become clear, input variables are represented as units despite the absence of computation i.e. a unit is a number.

Layers

The unit's output is not the model's final prediction but an intermediate transformation of the input

- Multiple transformations are computed using nested groups of units called layers $f^{(l)},\ l = 0, \ldots, L$
- Hidden layers $f^{(1)}, \ldots, f^{(L-1)}$ compute recursively intermediate transformations of the input
- The output layer $f^{(L)}$ produces the model prediction in the transformed inputs i.e. the output of $f^{(L-1)}$

Structure



Feed-forward network with three layers

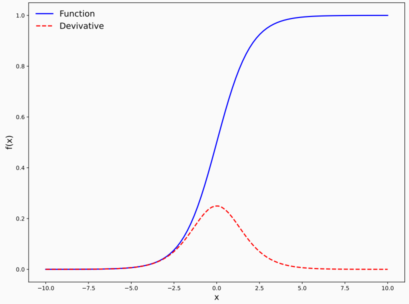
Dashed rectangles represent network layers. The model has two hidden layers with $k^{(1)} = k^{(2)} = 4$ hidden units. Each of the $k^{(l)}$ units of a layer transform the same inputs using different sets of parameters.

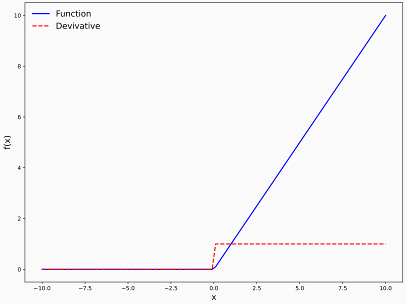
Hidden layers

Hidden units produce intermediate transformations by computing non-linear combinations of their input

- This allows the optimised model to detect non-linearities and interactions i.e. functional form
- The structure of the hidden layers, i.e. the number of layers and units, are tuning parameters
- The activation function is chosen for optimisation e.g. differentiable, monotonous, zero-centred

Common activations and their derivatives





Sigmoid (right) and Rectified Linear Units (left). The plain lines represent the activation functions and their derivatives are depicted by the dashed lines. With linear activation the network would simplify to a linear model. Many differentiable functions can be used in the hidden layers but some have computational advantages e.g. differentiable, monotonous, zero-centred.

Output layer

Output unit transform the output of the last hidden layer (i.e. transformed variables) into the estimated response

- The number of output units and their activation depends on the distribution of the response
- The activations are often the same as the GLM's link functions e.g. identity, logistic, poisson
- The output layer can be interpreted as a generalised linear model on the transformed variables

Learning

Learning

Why should would expect this layered model structure to increase predictive performance?

- The composition of numerous parametrised non-linear function gives the model considerable flexibility
- Unlike linear models, networks do not attempt to fit the entire data space in a single parametric equation
- They gradually approach a functional form by composing numerous simple non-linear transformations

Learning

Researchers have stressed the importance of distributed and hierarchical representations (Bengio et al. 2013)

- Network representations or transformed variables can be seen as both composite and latent variables
- A hidden unit may combine surface area and the number of bedrooms to determine the type of housing
- Another may combine the distance to school and the student-teacher ratio into measures of schooling quality
- The same applies to any feature present in the data, which the model identifies as relevant for prediction

Learning

The units in the second hidden layer may combine these transformed inputs into more abstract representations

- E.g. composite measures of house characteristics, local labour markets, local amenities or accessibility
- Hidden units are sub-models constructing increasingly abstract representations from the input
- Units encode multiple such concepts in a single numerical value so they are not readily interpretable

Generalisation

Generalisation

Equation (1) can generalised to multiple observation and units. Note that $X^{(l)} = y^{(l)}$ to simplify notation

$$X^{(l)} = y^{(l)} = \sigma^{(l)}\left(X^{(l-1)}_{\leftarrow x_0} \cdot \beta^{(l)}\right) \tag{2}$$

Where $X^{(l)}$ and $\beta^{(l)}$ are matrices of concatenated observations (as rows) and parameters (as columns)

$$X^{(l)} = \begin{bmatrix} - & x_1^{(l)} & - \\ & \vdots & \\ - & x_n^{(l)} & - \end{bmatrix}_{n \times k^{(l)}} \qquad \beta^{(l)} = \begin{bmatrix} | & & | \\ \beta_0^{(l)} & \dots & \beta_{k^{(l)}}^{(l)} \\ | & & | \end{bmatrix}_{k^{(l-1)}_{+1} \times k^{(l)}}$$

▸ Matrix product

Generalisation

Forward propagation

		Operations	Dimensions
$f^{(1)}$	$X^{(1)}$	$= \sigma^{(1)}\left(X^{(0)}_{\leftarrow x_0} \cdot \beta^{(1)}\right)$	$n \times 4 = [n \times 2_{+1}] \cdot [3 \times 4]$
$f^{(2)}$	$X^{(2)}$	$= \sigma^{(2)}\left(X^{(1)}_{\leftarrow x_0} \cdot \beta^{(2)}\right)$	$n \times 4 = [n \times 4_{+1}] \cdot [5 \times 4]$
$f^{(L)}$	$\hat{y} = X^{(L)}$	$= \sigma^{(L)}\left(X^{(2)}_{\leftarrow x_0} \cdot \beta^{(L)}\right)$	$n \times 1 = [n \times 4_{+1}] \cdot [5 \times 1]$

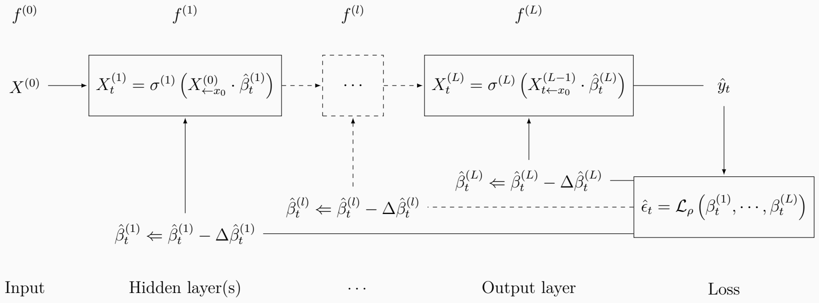
This table details the forward propagation operations for the network illustrated in figure 2, along with the matrix dimensions. More generally, $X^{(l)}$ has dimensions $n \times k^{(l)}$ while $\beta^{(l)}$ has dimensions $k^{(l-1)}_{+1} \times k^{(l)}$.

Estimation

Since hidden units have no target output, the estimated error is evaluated at the end of the forward pass

- The training error is computed using the loss function corresponding to the response distribution
- Regularisation ρ may sum the parameters, either squared or expressed as an absolute value (i.e. L_1, L_2)
- There are many other ways to regularise the model (e.g. early stopping, dropout, normalisation, adversarial)

Abstract feed-forward network


$f^{(0)}$
$f^{(1)}$
$f^{(l)}$
$f^{(L)}$
$X^{(0)}$
$X_t^{(1)} = \sigma^{(1)}\left(X^{(0)}_{\leftarrow x_0} \cdot \hat{\beta}_t^{(1)}\right)$
$\cdots$
$X_t^{(L)} = \sigma^{(L)}\left(X^{(L-1)}_{t \leftarrow x_0} \cdot \hat{\beta}_t^{(L)}\right)$
$\hat{y}_t$
$\hat{\beta}_t^{(L)} \Leftarrow \hat{\beta}_t^{(L)} - \Delta\hat{\beta}_t^{(L)}$
$\hat{\beta}_t^{(l)} \Leftarrow \hat{\beta}_t^{(l)} - \Delta\hat{\beta}_t^{(l)}$
$\hat{\epsilon}_t = \mathcal{L}_\rho\left(\beta_t^{(1)}, \cdots, \beta_t^{(L)}\right)$
$\hat{\beta}_t^{(1)} \Leftarrow \hat{\beta}_t^{(1)} - \Delta\hat{\beta}_t^{(1)}$
Input
Hidden layer(s)
$\cdots$
Output layer
Loss

The subscript t indicates the quantities that change with each iteration of the optimisation routine, $\Leftarrow$ is the update operator and $\Delta\hat{\beta}_t^{(l)}$ is the amount by which the parameters are updated. The loss is computed at the end of the forward pass. The backward pass updates the parameters iteratively to decrease the training error.

Within this parameter space, the optimisation searches the $\hat{\beta}$ that minimise the (penalised) training error

- The loss function is never strictly convex and the optimisation problem has no closed-form solution
- Gradient-based optimisation updates the parameters iteratively to converge toward a local minimum
- Gradients are computed for each training observation and averaged before updating the parameters

Update

For simplicity consider a single training observation i and a single parameter of unit u indexed j. The update rule is

$$\beta_j \Leftarrow \beta_j - \eta \frac{\partial \mathcal{L}_i}{\partial \beta_j} \tag{3}$$

where $\Leftarrow$ is the update operator and η is a tuning parameter called the learning rate

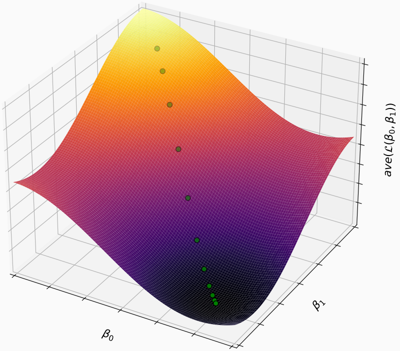
- Parameters are updated according to their relative contribution to the training error
- This is measured the partial derivatives of the loss function with respect to the parameters i.e. gradient

Optimisation path

The closest local approximation of a function is the tangent, whose slope is the partial derivative

- The vector of partial derivatives with respect to every function parameter is called the gradient
- The gradient points to the direction where the function is increasing by the largest amount
- The update is proportional to the partial derivative but smaller in magnitude i.e. local approximation

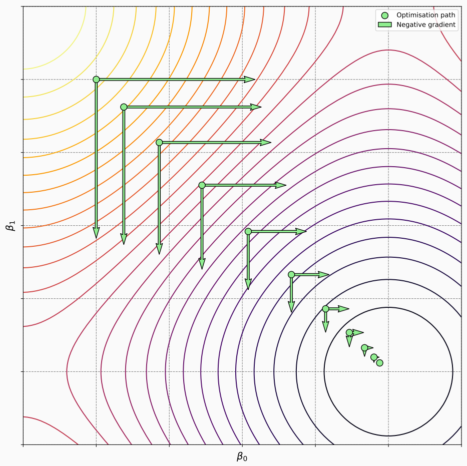
Optimisation path



Loss function with two parameters

Illustrative function with two parameters, but the intuition applies to higher dimensions. The loss function averaged across training observations can be represented as a hilly landscape in the multi-dimensional space of parameter values. Note the presence of local minima and saddle points.

Optimisation path



Loss function with two parameters

In the vector space, the gradient points to the direction where the loss function increases by the largest amount. The steps represent the negative gradient, weighted by the learning rate. The colour denotes the loss averaged over the training batch.

Generalisation

Generalising to multiple training observations, n partial derivatives are computed for each parameter

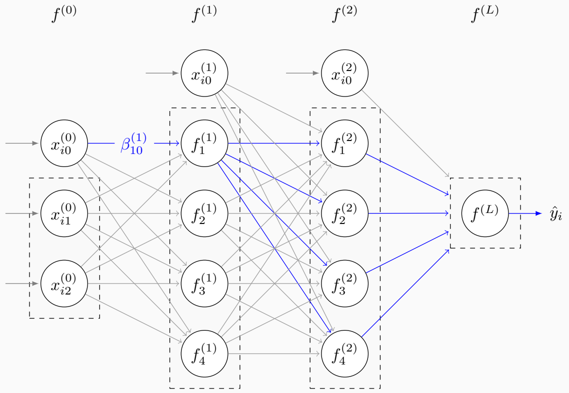
- A change in a single parameter affects the training error for every observation, either positively or negatively
- Partial derivatives are averaged out to compute the update that reduces the average training error
- The size of the training data and the number of parameters affect the computational cost of the update

Backpropagation

Intuition

Computing the gradients is demanding given the number of parameters and the nested nature of the model

- Backpropagation (Rumelhart et al. 1986) addresses these issues using the chain rule of differentiation
- The complex gradient expressions can be expressed as the product of simple partial derivatives
- Duplicate calculations are avoided by using quantities that are computed during the forward pass



Backpropagation

A small change to $\beta_{10}^{(1)}$ alters the result of the dot product $z_1^{(1)}$ and the unit's output $x_1^{(1)}$. This modifies the output of the four units in the second hidden layer, the predicted response and training error. To compute the contribution of a parameter, we take into account its impact on all units in the next layers.

Chain rule

Consider the function composition

$$y = f(x)$$
$$z = g(y) = (g \circ f)(x)$$

The chain rule states that

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

The complex gradient expressions can be expressed as the product of simple partial derivatives

Impact of changing $\beta_{10}^{(1)}$ on network

$f_1^{(1)}$ $f^{(2)}$ $f^{(L)}$

$$\begin{tikzcd}
 & & & \boxed{z_{i1}^{(2)}} \arrow[r] & \boxed{x_{i1}^{(2)}} \arrow[ddr, bend left] & & & \\
 & & & \boxed{z_{i2}^{(2)}} \arrow[r] & \boxed{x_{i2}^{(2)}} \arrow[dr, bend left] & & & \\
\boxed{\beta_{10}^{(1)}} \arrow[r] & \boxed{z_{i1}^{(1)}} \arrow[r] & \boxed{x_{i1}^{(1)}} \arrow[uur, bend left] \arrow[ur, bend left] \arrow[dr, bend right] \arrow[ddr, bend right] & & & \boxed{z_i^{(L)}} \arrow[r] & \boxed{x_i^{(L)}} \arrow[r] & \boxed{\mathcal{L}_i} \\
 & & & \boxed{z_{i3}^{(2)}} \arrow[r] & \boxed{x_{i3}^{(2)}} \arrow[ur, bend right] & & & \\
 & & & \boxed{z_{i4}^{(2)}} \arrow[r] & \boxed{x_{i4}^{(2)}} \arrow[uur, bend right] & & &
\end{tikzcd}$$

Hidden layer 1 (unit 1) Hidden layer 2 Output layer Loss

Detailed backpropagation operations

$f_1^{(1)}$ $f^{(2)}$ $f^{(L)}$

$$\begin{tikzcd}
 & & & z_{i1}^{(2)} \arrow[dddll, bend right, "\frac{\partial z_{i1}^{(2)}}{\partial x_{i1}^{(1)}}"'] & x_{i1}^{(2)} \arrow[l, "\frac{\partial x_{i1}^{(2)}}{\partial z_{i1}^{(2)}}"', "\delta_{i1}^{(2)}"] & & & \\
 & & & z_{i2}^{(2)} \arrow[ddll, bend right, "\frac{\partial z_{i2}^{(2)}}{\partial x_{i1}^{(1)}}"'] & x_{i2}^{(2)} \arrow[l, "\frac{\partial x_{i2}^{(2)}}{\partial z_{i2}^{(2)}}"', "\delta_{i2}^{(2)}"] & & & \\
\beta_{10}^{(1)} & z_{i1}^{(1)} \arrow[l, "\frac{\partial z_{i1}^{(1)}}{\partial \beta_{10}^{(1)}}"'] & x_{i1}^{(1)} \arrow[l, "\frac{\partial x_{i1}^{(1)}}{\partial z_{i1}^{(1)}}"', "\delta_{i1}^{(1)}"] & & & z_i^{(L)} \arrow[uuul, bend right, "\frac{\partial z_i^{(L)}}{\partial x_{i1}^{(2)}}"'] \arrow[uul, bend right, "\frac{\partial z_i^{(L)}}{\partial x_{i2}^{(2)}}"'] \arrow[dl, bend left, "\frac{\partial z_i^{(L)}}{\partial x_{i3}^{(2)}}"] \arrow[ddl, bend left, "\frac{\partial z_i^{(L)}}{\partial x_{i4}^{(2)}}"] & x_i^{(L)} \arrow[l, "\frac{\partial x_i^{(L)}}{\partial z_i^{(L)}}"', "\delta_i^{(L)}"] & \mathcal{L}_i \arrow[l, "\frac{\partial \mathcal{L}_i}{\partial x_i^{(L)}}"'] \\
 & & & z_{i3}^{(2)} \arrow[ull, bend left, "\frac{\partial z_{i3}^{(2)}}{\partial x_{i1}^{(1)}}"] & x_{i3}^{(2)} \arrow[l, "\delta_{i3}^{(2)}"', "\frac{\partial x_{i3}^{(2)}}{\partial z_{i3}^{(2)}}"] & & & \\
 & & & z_{i4}^{(2)} \arrow[uull, bend left, "\frac{\partial z_{i4}^{(2)}}{\partial x_{i4}^{(1)}}"] & x_{i4}^{(2)} \arrow[l, "\delta_{i4}^{(2)}"', "\frac{\partial x_{i4}^{(2)}}{\partial z_{i4}^{(2)}}"] & & &
\end{tikzcd}$$

Hidden layer 1 (unit 1) Hidden layer 2 Output layer Loss

Chain rule

The partial derivative of the loss with respect to $\beta_{01}^{(1)}$ for observation i can be expressed as

$$\begin{aligned}\frac{\partial \mathcal{L}_i}{\partial \beta_{10}^{(l)}} = & \underbrace{\frac{\partial \mathcal{L}_i}{\partial x_i^{(L)}} \frac{\partial x_i^{(L)}}{\partial z_i^{(L)}}}_{\delta_i^{(L)}} \\ & \underbrace{\frac{\partial z_i^{(L)}}{\partial x_{i1}^{(2)}} \frac{\partial x_{i1}^{(2)}}{\partial z_{i1}^{(2)}}}_{\prod \cdots = \delta_{i1}^{(2)}} \underbrace{\frac{\partial z_i^{(L)}}{\partial x_{i2}^{(2)}} \frac{\partial x_{i2}^{(2)}}{\partial z_{i2}^{(2)}}}_{\prod \cdots = \delta_{i2}^{(2)}} \underbrace{\frac{\partial z_i^{(L)}}{\partial x_{i3}^{(2)}} \frac{\partial x_{i3}^{(2)}}{\partial z_{i3}^{(2)}}}_{\prod \cdots = \delta_{i3}^{(2)}} \underbrace{\frac{\partial z_i^{(L)}}{\partial x_{i4}^{(2)}} \frac{\partial x_{i4}^{(2)}}{\partial z_{i4}^{(2)}}}_{\prod \cdots = \delta_{i4}^{(2)}} \\ & \underbrace{\frac{\partial z_{i1}^{(2)}}{\partial x_{i1}^{(1)}} \frac{\partial z_{i2}^{(2)}}{\partial x_{i1}^{(1)}} \frac{\partial z_{i3}^{(2)}}{\partial x_{i1}^{(1)}} \frac{\partial z_{i4}^{(2)}}{\partial x_{i1}^{(1)}} \frac{\partial x_{i1}^{(1)}}{\partial z_{i1}^{(1)}}}_{\ldots \delta_{i1}^{(1)}} \frac{\partial z_{i1}^{(1)}}{\partial \beta_{10}^{(1)}} \end{aligned} \tag{4}$$

where $\prod \ldots$ denotes the product of the bracket with the lines above and $\delta_{iu}^{(l)}$ is the "error signal" defined as $\partial \mathcal{L}_i / \partial z_{iu}^{(l)}$

More generally, the partial derivative of the loss with respect to $\beta_{ju}^{(l)}$ for observation i can be expressed as

$$\frac{\partial \mathcal{L}_i}{\partial \beta_{ju}^{(l)}} = \frac{\partial \mathcal{L}_i}{\partial z_{iu}^{(l)}} \frac{\partial z_{iu}^{(l)}}{\partial \beta_{ju}^{(l)}} = \delta_{iu}^{(l)} \frac{\partial z_{iu}^{(l)}}{\partial \beta_{ju}^{(l)}} \tag{5}$$

where the “error signal” $\delta_{iu}^{(l)}$ measures the contribution of $z_{iu}^{(l)}$ to the training error

- Backpropagation computes efficiently the error signals $\delta_{iu}^{(l)}$ for each unit and observation
- The error signals can be easily related to the partial derivative with respect to each parameter

To computes the error signals efficiently, backpropagation avoids duplicated computations

- The error signal of a layer l can be expressed as a function of the error signal in next layer $l + 1$
- These error signals are computed starting from the output layer all the way to the first hidden layer
- For increased efficiency, the relevant calculations can be stored in memory during the forward pass

Forward and backward passes

	Forward propagation			Backpropagation		
	$\downarrow$	$X^{(0)}$	$= X^{(0)}_{\leftarrow x_0}$	$\leftarrow$		
$f^{(1)}$	$\downarrow$	$Z^{(1)}$	$= X^{(0)} \cdot \beta^{(1)}$	$\uparrow$	$\partial Z^{(1)} / \partial \beta^{(1)}$	$= X^{(0)T}$
	$\downarrow$	$X^{(1)}$	$= \sigma(Z^{(1)})$	$\delta(1)$	$\partial X^{(1)} / \partial Z^{(1)}$	$= \sigma'(Z^{(1)})$
	$\downarrow$	$X^{(1)}$	$= X^{(1)}_{\leftarrow x_0}$	$\uparrow$	$\partial Z^{(2)} / \partial X^{(1)}$	$= \beta^{(2)T}$
	$\vdots$			$\vdots$		
$f^{(l)}$	$\downarrow$	$Z^{(l)}$	$= X^{(l-1)} . \beta^{(l)}$	$\uparrow$	$\partial Z^{(l)} / \partial \beta^{(l)}$	$= X^{(l-1)T}$
	$\downarrow$	$X^{(l)}$	$= \sigma(Z^{(l)})$	$\delta^{(l)}$	$\partial X^{(l)} / \partial Z^{(l)}$	$= \sigma'(Z^{(l)})$
	$\downarrow$	$X^{(l)}$	$= X^{(l)}_{\leftarrow x_0}$	$\uparrow$	$\partial Z^{(l+1)} / \partial X^{(l)}$	$= \beta^{(l+1)T}$
	$\vdots$			$\vdots$		
$f^{(L)}$	$\downarrow$	$Z^{(L)}$	$= X^{(L-1)} . \beta^{(L)}$	$\uparrow$	$\partial Z^{(L)} / \partial \beta^{(L)}$	$= X^{(L-1)T}$
	$\downarrow$	$X^{(L)}$	$= \sigma(Z^{(L)})$	$\delta^{(L)}$	$\partial X^{(L)} / \partial Z^{(L)}$	$= \sigma'(Z^{(L)})$
	$\rightarrow$			$\uparrow$	$\partial \mathcal{L}(\beta) / \partial X^{(L)}$	$= \mathcal{L}'(X^{(L)})$

Most of the quantities required to compute the gradients, such as $X^{(l)}$, $\beta^{(l)}$ and $Z^{(l)}$ are stored during forward propagation to avoid duplicated computations.

The matrix $\delta^{(l)}$ has dimensions $n \times k^{(l)}$ and contains the error signals for every unit and observations

$$\begin{aligned}
\delta^{(L)} &= \frac{\partial \mathcal{L}(\beta)}{\partial X^{(L)}} \frac{\partial X^{(L)}}{\partial Z^{(L)}} \\
&= \mathcal{L}'(X^{(L)}) \odot \sigma'^{(L)}(Z^{(L)}) \qquad (6)
\end{aligned}$$

$$\begin{aligned}
\delta^{(l)} &= \delta^{(l+1)} \frac{\partial Z^{(l+1)}}{\partial X^{(l)}} \frac{\partial X^{(l)}}{\partial Z^{(l)}} \\
&= \delta^{(l+1)} \cdot \beta^{(l+1)T} \odot \sigma'^{(l)}(Z^{(l)}_{\leftarrow x_0}) \qquad (7)
\end{aligned}$$

where $\odot$ denotes the Hadamard product and the gradient is represented as a column vector (i.e. denominator layout)

Using the expression for the error signal, the matrix of partial derivatives with respect to each parameter is computed as

$$\frac{\partial \mathcal{L}(\beta)}{\partial \beta^{(l)}} = X^{(l-1)T}_{\leftarrow x_0} \cdot \delta^{(l)}_{\rightarrow x_0} \tag{8}$$

For each parameter, this formulation sums the gradients across training observations. The update rule becomes

$$\beta^{(l)} \Leftarrow \beta^{(l)} - \frac{\eta}{n} \frac{\partial \mathcal{L}(\beta)}{\partial \beta^{(l)}} \tag{9}$$

In practice, each gradient descent iteration is performed using a partition of the training observations called “batch”

Backpropagation

	Operations		Dimensions
$f^{(L)}$	$\delta^{(L)}$	$= \mathcal{L}'(X^{(L)}) \odot \sigma^{(L)\prime}(Z^{(L)})$	$n \times 1 = [n \times 1] \odot [n \times 1]$
	$\dfrac{\partial \mathcal{L}(\beta)}{\partial \beta^{(L)}}$	$= X^{(2)T}_{\leftarrow x_0} \cdot \delta^{(L)}$	$5 \times 1 = [4_{+1} \times n] \cdot [n \times 1]$
$f^{(2)}$	$\delta^{(2)}$	$= \delta^{(L)} \cdot \beta^{(L)T} \odot \sigma^{(2)\prime}(Z^{(2)}_{\leftarrow x_0})$	$n \times 5 = [n \times 1] \cdot [1 \times 5] \odot [n \times 4_{+1}]$
	$\dfrac{\partial \mathcal{L}(\beta)}{\partial \beta^{(2)}}$	$= X^{(1)T}_{\leftarrow x_0} \cdot \delta^{(2)}_{\rightarrow x_0}$	$5 \times 4 = [4_{+1} \times n] \cdot [n \times 5_{-1}]$
$f^{(1)}$	$\delta^{(1)}$	$= \delta^{(2)}_{\rightarrow x_0} \cdot \beta^{(2)T} \odot \sigma^{(1)\prime}(Z^{(1)}_{\leftarrow x_0})$	$n \times 5 = [n \times 5_{-1}] \cdot [4 \times 5] \odot [n \times 4_{+1}]$
	$\dfrac{\partial \mathcal{L}(\beta)}{\partial \beta^{(1)}}$	$= X^{(0)T}_{\leftarrow x_0} \cdot \delta^{(1)}_{\rightarrow x_0}$	$3 \times 4 = [2_{+1} \times n] \odot [n \times 5_{-1}]$

The denominator layout implies the inversion of the two terms and the transpose for the matrix multiplications. Note that $\delta^{(l+1)} \cdot \beta^{(l+1)T}$ contains the bias while $\sigma'(Z^{(l)})$ does not. To allow for element-wise multiplication, a column of ones is added to $Z^{(l)}$. When we back-propagate to the previous layer, $\delta^{(l)}$ contains the error linked to the bias, which is removed because it is not connected to the previous layer.

Gradient checking

You can check your implementation of backpropagation against a numerical approximation of the gradient

$$\frac{\partial \mathcal{L}(\beta)}{\partial \beta} \approx \frac{\mathcal{L}(\beta + \epsilon) - \mathcal{L}(\beta - \epsilon)}{2\varepsilon}$$

where ε is a small random value. In the case where β is a vector of values, we can compute for each $\beta_j,\ j = 1, \ldots, k$

$$\frac{\partial \mathcal{L}(\beta_)}{\partial \beta_0} \approx \frac{\mathcal{L}(\beta_0 + \epsilon, \beta_1, \ldots, \beta_k) - \mathcal{L}(\beta_0 - \epsilon, \beta_1, \ldots, \beta_k)}{2\varepsilon}$$

$$\frac{\partial \mathcal{L}(\beta_)}{\partial \beta_1} \approx \frac{\mathcal{L}(\beta_0, \beta_1 + \epsilon, \ldots, \beta_k) - \mathcal{L}(\beta_0, \beta_1 - \epsilon, \ldots, \beta_k)}{2\varepsilon}$$

$$\ldots$$

Application

Hedonic pricing

This network can be used as drop-in replacement of linear models when interpretability is not required

- This is illustrated on a hedonic price problem on the Boston dataset (Harrison and Rubinfeld 1978)
- The response is the median value of owner-occupied houses for 506 census tracts in Boston in 1970
- The input contains 13 variables including measures of air quality, schooling quality among others

Comparison of predictive models

Model	R^2 training		R^2 test	
Log-linear regression	0.7416	(0.0091)	0.7144	(0.0743)
Feed-forward network	0.9908	(0.0015)	0.8664	(0.0434)
– regularised	0.9502	(0.0023)	0.9016	(0.0498)

Statistics are produced using 10-fold cross-validation. Standard deviations are in parenthesis. The network has 2 hidden layers with 64 units and ReLU activation. The output layer has a single unit with sigmoid activation. Note that the (irreducible) error is substantial in social science data.

Summary

Summary

The processing of high-dimensional and unstructured data offer strong potential for original economic analysis

- Traditional models perform poorly due to the dimensionality and the absence of theory
- Neural networks are powerful and flexible predictive models designed to solve these problem
- Specialised networks offer state-of-the art performance on image an language data among other sources

Summary

On the downside networks trade most interpretability for flexibility and are computationally heavier

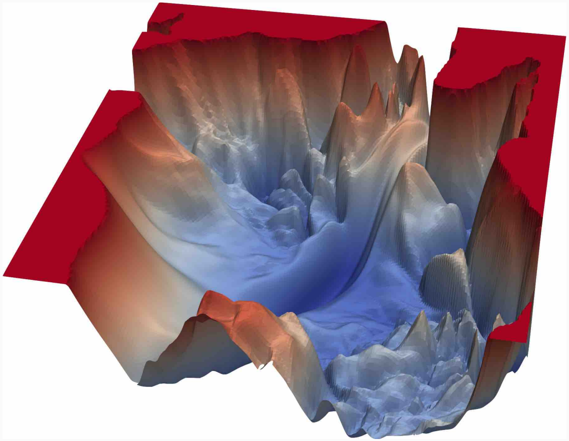
- Approximating even simple functions involve many parameters and requires more observations
- Many tuning parameters e.g. layers, units, activation, loss, optimiser, regularisation, learning rate
- Network structures and implementation details and are important in this more empirical literature

Summary

Computing the gradients is demanding given the number of parameters and the nested nature of the model

- The backpropagation algorithm addresses these issues by using the chain rule of differentiation
- The complex gradient expressions can be expressed as a product of simple partial derivatives
- It avoids duplicate calculations by re-using quantities that are computed and stored during the forward pass

Summary



Illustrative loss landscape

Three dimensional representation of a VGG-56 network loss landscape (Li et al. 2018). Complex loss landscapes have many local minima and saddle points where the optimisation could get trapped and converge to sub-optimal values.

Thank you for your attention!

References

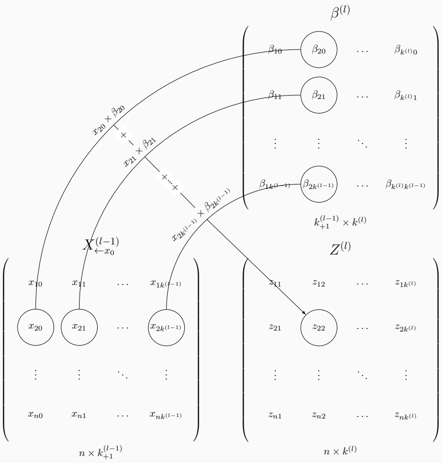
References i

- Harrison, David J. and Daniel L. Rubinfeld (1978). **“Hedonic housing prices and the demand for clean air”.** In: *Journal of Environmental Economics and Management* 5.1, pp. 81–102 (cit. on p. 44).
- Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). **“Learning representations by back-propagating errors”.** In: *Nature* 323.6088, pp. 533–536 (cit. on p. 30).
- Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). **“Representation learning: A review and new perspectives”.** In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8, pp. 1798–1828 (cit. on p. 15).

References ii

- Nielsen, Michael A. (2015). ***Neural networks and deep learning*.** Determination Press.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). ***Deep Learning*.** MIT Press.
- Li, Hao et al. (2018). **"Visualizing the loss landscape of neural nets".** In: *Advances in Neural Information Processing Systems*. Vol. 31 (cit. on p. 50).

Appendix



Matrix product

$X_{\leftarrow x_0}^{(l-1)}$ contains n observations (rows) and $k_{+1}^{(l-1)}$ variables (columns), including the constant. $\beta^{(l)}$ contains the corresponding $k_{+1}^{(l-1)}$ parameters for each of the $k^{(l)}$ units (columns). The product of these two matrices is denoted $Z^{(l)}$ with n observations, where each $k^{(l)}$ variable represents a combination of the same input with different parameters. The activation function $\sigma^{(l)}$ is applied element-wise to produce the matrix $X^{(l)}$.

Presentation