1 New Problem Space

Deep Learning models non-linear patterns of data. This opens up a new range of problems to solve.

2 The steps

The most important pieces, mathematically, are *forward* and *backward* propagation. They consist in calculating the cost and updating parameters, respectively.

- Forward: Compute Loss and Cost.
- Backward: Minimize cost, we need to derive it, and find a way to move towards the minimum. We use *Gradient Descent*.

3 Example: Multivariable Linear Regression

A linear regression model for 2 features:

$$\begin{aligned} y &= w_1 \cdot x_1 + w_2 \cdot x_2 + b \\ &= \vec{w} \cdot \vec{x} + b \end{aligned}$$

w_1,w_2 control the slopes of this plane, b translates it up and down. $\vec{x}$ is for one sample. Many samples are represented as a matrix:

$$[y1, y2] = [w1, w2] \cdot \begin{bmatrix} x_1 & x_1 \\ x_2 & x_2 \end{bmatrix} + b \tag{1}$$

y_i is the result for each sample (columns in the matrix).

We need to do forward and backward propagation.

3.1 Forward Propagation

The Loss $= Loss(w, b)$ in linear regression is the *Square Error*:

$$\begin{aligned} L_i(\vec{w}, b) &= (y_i - \hat{y})^2 \\ &= (y_i - \vec{w} \cdot \vec{x}_j - b)^2 \end{aligned}$$

In Linear Regression, the cost is the averaged sum of L_i, and it's the *Mean Square Error*:

$$\begin{aligned} C(w_1, w_2, b) &= \frac{1}{2} \sum_{i=0}^{i=2} L_i(w_1, w_2, b) && (2) \\ &= \frac{1}{2}([y_1, y_2] - [\hat{y}_1, \hat{y}_2]) \cdot ([y_1, y_2] - [\hat{y}_1, \hat{y}_2]) && (3) \\ &= \frac{1}{2}([y_1, y_2] - [w_1, w_2]\mathbf{X} - b) \cdot ([y_1, y_2] - [w_1, w_2]\mathbf{X} - b) && (4) \end{aligned}$$

$\frac{1}{2}$ is to average over examples. Equation 4 is what we implement into code. Outliers may be critical on the effect of weights and biases.

In Python it'd look like:

```
Y' = np.dot(w,X) - b
cost = 1/2*np.pow(2,(Y-Y'))
```

3.2 Backward Propagation

We need the Cost's derivative to find better weights and bias. The complicated thing is keeping track of each step.

$$\begin{aligned} dC(\vec{w}, b) &= \frac{dC}{d\vec{w}} + \frac{dC}{db} \\ &= d(\sum_{i=0}^{i=2} L_i) = \sum_{i=0}^{i=2} dL_i(\vec{w}, b) && (5) \\ &= \sum_{i=0}^{i=2} \frac{dL_i}{d\vec{w}} + \frac{dL}{db} \end{aligned}$$

Equation 5 makes use of linearity property of diff. We sum over columns (each sample). Now, pick up a column/sample k:

$$\begin{aligned} L_k(\vec{w}, b) &= (y_k - \hat{y_k})^2 \\ L_k &= A_k(w_1, w_2, b)^2 \\ dL_k &= 2 \cdot A_k(w1, w2, b) \cdot dA_k \end{aligned}$$

We named A_k to the difference $y_i - \hat{y}_i$. So dC was reduced to:

$$dC = \frac{1}{2}\sum_{i=0}^{i=2} dL_i \tag{6}$$

$$= \frac{1}{2} \times 2 \cdot A_i(w_1, w_2, b) \cdot dA_i \tag{7}$$

Where A_i will be broadcasted, the shape is $1 \times m$ and the sape of dA_i is $n \times m$. Right now, $m = 2$. The next step will be to sum over colums. And although A_i will be calculated $(y_i - \hat{y}_i)$ in the code, we need it now to find dA_k:

$$\begin{aligned}
dL_k &= 2A_k \cdot (\frac{dA_k}{dw_1} + \frac{dA_k}{dw_2} + \frac{dA}{db}) \\
A_k &= A_k(w_1, w_2, b) \\
&= y_k - \vec{w} \cdot \vec{x_k} - b \\
&= y_k - w_1 \cdot x_{1k} - w_2 \cdot x_{2k} - b \\
dA_k &= -x_{1k} - x_{2k} - 1
\end{aligned}$$

where

$$\begin{aligned}
\frac{dA_k}{dw_1} &= -x_{1k} \\
\frac{dA_k}{dw_2} &= -x_{2k} \\
\frac{dA_k}{db} &= -1
\end{aligned}$$

This can be expressed in compact form:

$$dC = \frac{1}{2} \times sum(2A \times (-\mathbf{X} - 1)) \tag{8}$$

Where "sum" implies we sum all values from the resulting array, and b will be broadcast to the shape of $\mathbf{X}$.

We also found each partial derivative:

$$dC = \frac{dC}{dw} + \frac{dC}{db} \tag{9}$$

$$dC = -sum(2A \times \mathbf{X}) - sum(2A \times 1) \tag{10}$$

To update the parameters, *Gradient Descent* method is used. We update the vector $\vec{w}$ as follows:

$$\vec{w} = \vec{w} - \frac{dC}{dw} \cdot \alpha \tag{11}$$

$$\vec{b} = \vec{b} - \frac{dC}{db} \cdot \alpha \tag{12}$$

α is called *learning rate* and we use to tune the derivation. We go against the gradient so the sign is changed to $-$ (it points to max incresing direction otherwise).

3.3 General Derivation

The problem is to find w_i, b such that the multidimensional "plane" has small error respect to each datapoint. Then for a new datapoint we will have a trained predictor.

In linear regression, the model is:

$$\begin{aligned} y &= w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n \\ &= \sum_i^n w_i \cdot x_i + b \\ &= \vec{w} \cdot \vec{x} + b \end{aligned}$$

Here $\vec{x}$ is for one sample. For n samples, it becomes a matrix, we write $\vec{y} = \vec{w} \cdot \mathbf{X} + \vec{b}$. This is represented:

$$[y_1 y_2 \ldots y_n] = [w_1 w_2 \ldots w_n] \cdot \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1m} \\ x_{21} & x_{22} & \ldots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{22} & \ldots & x_{nm} \end{bmatrix} + [b_1 b_2 \ldots b_n] \tag{13}$$

There are m examples-columns with n features-rows. Hence $[\mathbf{X}] = m \times n$

The Loss $= L(w, b)$ in linear regression is the *Square Error*:

$$\begin{aligned} L(\vec{w}, b) &= (y - \hat{y})^2 \\ &= (y_k - \vec{w}_j \cdot \vec{x}_j - b)^2 \end{aligned}$$

The cost in any method/model measures how well it's doing with he current parameters. In Linear Regression, it is the averaged sum of L_k, and it's the *Mean Square Error*:

$$C(\vec{w}, \vec{b}) = \frac{1}{m} \sum_{i=0}^{m} L_i(\vec{w}, b) \tag{14}$$

$$= \frac{1}{m} \sum_{i=0}^{m} (\vec{y}_i - \hat{\vec{y}}_i) \cdot (\vec{y}_i - \hat{\vec{y}}_i) \tag{15}$$

$$= \frac{1}{m} (\vec{y} - \vec{w}\mathbf{X} - \vec{b}) \cdot (\vec{y} - \vec{w}\mathbf{X} - \vec{b}) \tag{16}$$

$C(\vec{w})$ is a way to denote C depends on all the variables in $\vec{w}$. *m* is the number of samples.

The cost is a "bowl", we will reach the global minimum (or close). We use the Cost derivative to find the better weight and bias.

$$\begin{aligned}
dC(\vec{w},\vec{b}) &= \frac{dC}{dw} + \frac{dC}{db} \\
&= \frac{1}{m} d(\sum_{0}^{m} L_i) \\
&= \frac{1}{m} \sum_{i=0}^{m} dL_i(\vec{w}, b_i) \\
&= \frac{1}{m} \sum_{i=0}^{m} \frac{dL_i}{d\vec{w}} + \frac{dL_i}{db} \\
&= \frac{1}{m} \sum_{i=0}^{m} \frac{dL_i}{dw_1} + \ldots + \frac{dL_i}{dw_n} + \frac{dL}{db}
\end{aligned}$$

Take a particular column k:

$$\begin{aligned}
dL_k &= 2A * (\frac{dA}{dw_1} + \frac{dA}{dw_2} + \frac{dA}{db}) \\
A_k &= A(w_1, \ldots, w_n, b) \\
&= y_k - \vec{w} \cdot \vec{x}_k - b \\
&= y_k - w_1 \cdot x_{1k} - \ldots - w_n \cdot x_{nk} - b \\
dA_k &= -x_{1k} - x_{2k} - x_{nk} - 1
\end{aligned}$$

There is no graphical justification as to why we update w and b like that, but we are moving w against (minus) the gradient multiplied by a constant (alpha) called *learning rate*.

The minus sign is because the gradient always points away from the minimum and we want towards it (in one dimension there are only 2 directions).

The process is called *Gradient Descent*.

Because the $y_d - y_p$ is squared, MSE is a parabola for b and w then it makes sense: the farther away we are from the minimum the larger the gradient, and the more we want to update w and b.

4 Summary

- Model/Architecture*: equation to fit the data,
- FordP: Calculate the Cost, Evaluates the error,
- BackP: Training. Use cost, update w, b.

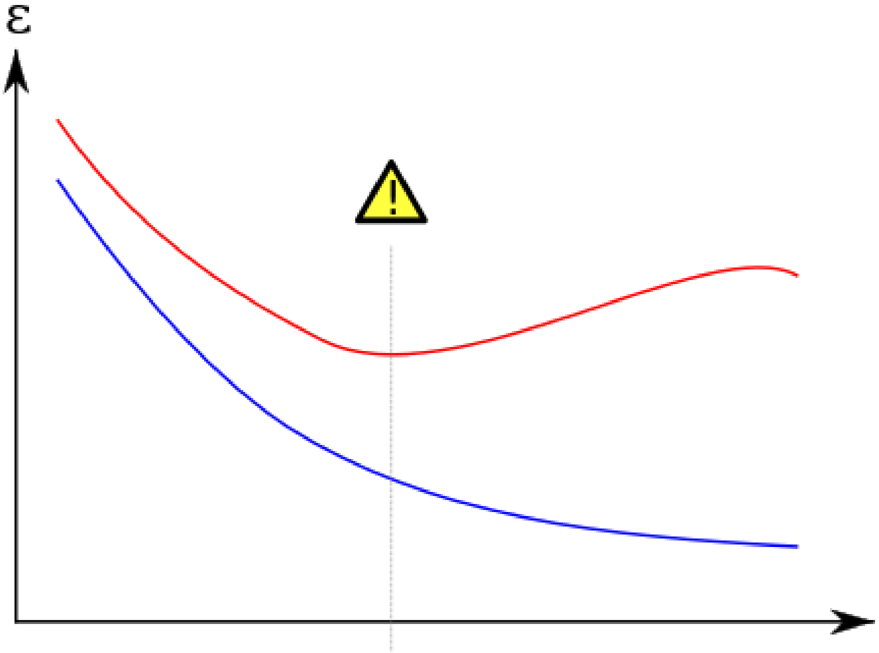
* Technically, model is only once the parameters are defined (after training).

Model = Architecture + parameters (fixed ones).

FP means from X to C; BP from L to z, in *terms of the chain rule*.

5 Other Concepts

Overfitting: Occurs when the cost in the training dataset decreases but it increases on the test dataset. The model starts to *memorize* data, and does not *generalize*.



Training error: blue; validation error: red, both as a function of the number of training cycles. The best predictive and fitted model would be where the validation error has its global minimum.

Underfitting: It occurs when the model or algorithm does not fit the data enough. It could be a bad model (too simple, or just not the right fit), or a lack of training, etc.

Classification v Regression: A classification model is one which attempts to predict a class, or category. That is, it's predicting from a number of discrete possibilities, such as "dog" or "cat." A regression model is one which attempts to predict one or more numeric quantities, such as a temperature or a location. Which one we use depends on the nature of the variables.

Cross Validation: The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias, and to give an insight on how the model will generalize to an independent dataset.

Why a CNN? It's the current state-of-the-art approach to creating computer vision models.

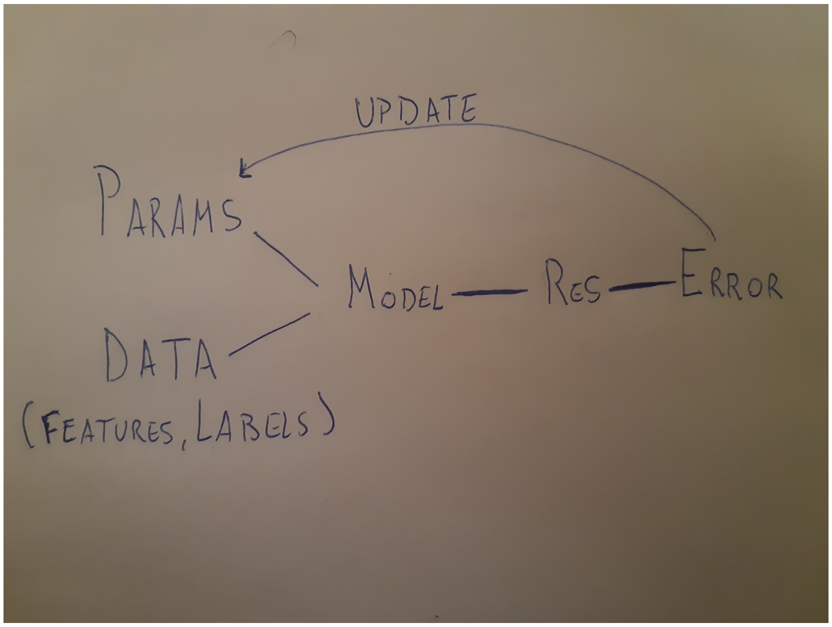


Figure 1: Machine Learning Process

6 High Level View

Deep Learning is a Machine Learning area. The latter can be associated with the image, where update maps to learning in human terms, and more data maps to more experience (in human life). A program that learns from experience.

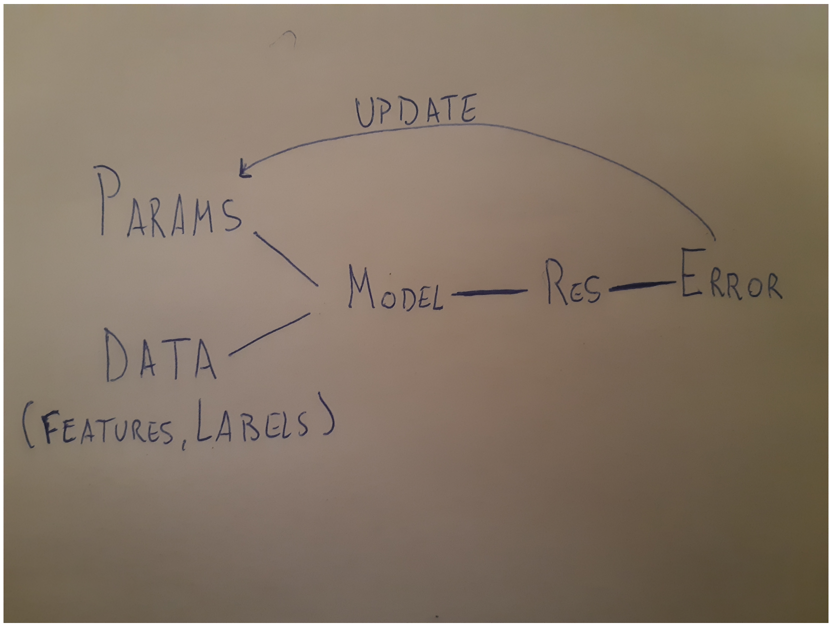


Figure 2: Two Layer NN