

Art and Science of ML:

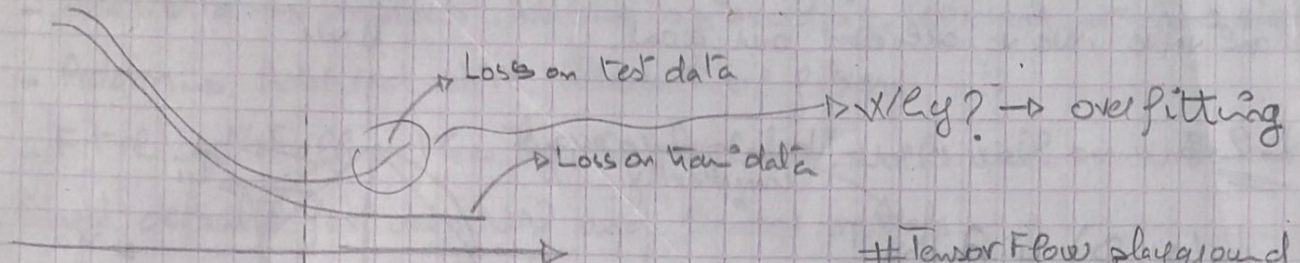
I. intro:

→ google cloud

→ How google does ML

II. The Art of ML

* Regularization:



TensorFlow playground

(→ visualizing how NN learn)

Early stopping!
→ It's an optima (a solution) but it can't help us in a case of complex data.

• oversimplified models

are useless so:

loss model complexity
 balance

→ model complexity

→ regularization

→ penalize model complexity

⇒ Minimize: loss + complexity

aim for low training error

but balance against complexity

We need to find the right balance between

simplicity and accurate fitting of training data.

• What problem regularization is solving for us?

→ regularization refers to any technique that helps generalize a model. a generalized model performs well not just on train data but also on never seen test data.

How to measure model complexity? \rightarrow regularization methods represent model complexity and try to keep it in check

\hookrightarrow as the magnitude of the weight vector

L1 norm: $\|w\|_1 = |w_0| + |w_1| + \dots$

L2 norm: $\|w\|_2 = (w_0^2 + w_1^2 + \dots)^{1/2}$

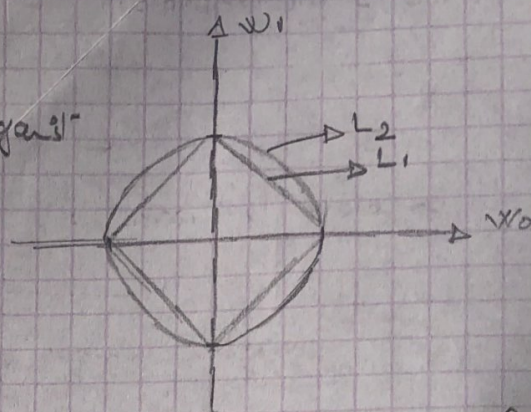
* if we keep the magnitude of our weight vector smaller than certain value, we've achieved our goal.

L2:

$$L(w, D) + \lambda \|w\|_2^2$$

\nearrow
aim for low training error

\nwarrow but balance against complexity



L1:

$$L(w, D) + \lambda \|w\|_1$$

* When applying L1 regularization the optimal value of certain weights can end up being zero!

\hookrightarrow Feature selection

• Complex models are bad, one of the ways to keep our model simple is by applying regularization.

* Learning rate and Batch size:

• Learning rate: controls the size of the step in the weight space.

• Batch size: controls the number of samples that the gradient is calculated on.

Lr:

• if too small, training will take a long time.

• if too large, training will bounce around.

Batch size:

• if too small, training will bounce around

• if too large, training will take a very long time

Optimization : it refers to the task of either maximizing or minimizing some function.

- Gradient Descent, SGD, \rightarrow find the min Loss
 - Momentum \rightarrow reduces Lr when gradient values are small
 - AdaGrad \rightarrow give frequently occurring features low Lr
 - AdaDelta \rightarrow improves AdaGrad by avoiding reducing Lr to zero
 - Adam \rightarrow AdaGrad with a bunch of fixes.
 - FTRL \rightarrow "Follow the regularized leader", works well on wide models.
- \hookrightarrow Good defaults for deep NN and linear models.

III - Hyperparameter Tuning :

parameters:

changed during mode

tuning (weights, bias, ...)

\hookrightarrow adjusted by tuning model

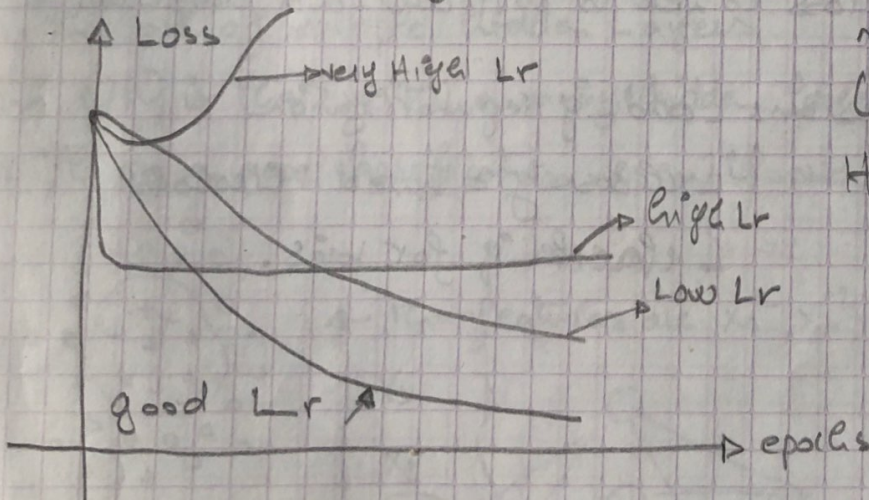
ML-model

Hyper-parameters :

set before tuning

(Lr, batch size, regularization rate, number of hidden layers, number of neurons, ...)

\hookrightarrow our job is to set the Hyperparameters



- Low Lr \rightarrow improvement is linear but you don't get the best possible performance.
- High Lr \rightarrow exponential improvement at first but you don't get it.
- very high Lr \rightarrow completely lost

ML Engine (MLE) for Hyperparameters Tuning

4/ a pinch of Science

- L_2 regularization only makes weights small, not zero. \rightarrow large and complex model!
- L_1 regularization tends to force the weights of not very predictive features to zero. \Rightarrow feature selector (Killing bad features and keep only strongest one in the model) \Rightarrow Small model = speed training \Rightarrow important for embedded models.

\rightarrow With L_1 , you can end up with a smaller model but it may be less predictive \Rightarrow The elastic net is a linear combination of L_1 and L_2 regularization penalties

$$L(w, D) + \lambda_1 \sum |w| + \lambda_2 \sum w^2$$

\rightarrow regularization techniques = adding a penalty term to the Loss function.

Why Regularization is important in logistic regression?

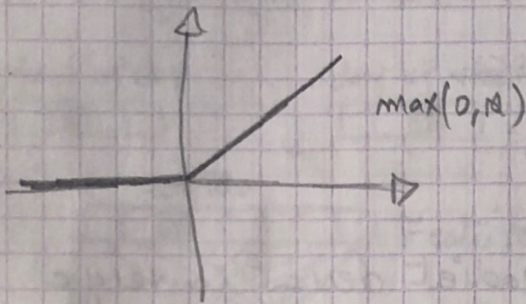
\hookrightarrow because driving the loss to zero is difficult and dangerous

- performing logistic regression:
 - adding regularization
 - choosing a tuned threshold
 - checking for bias.

5/ The Science of Neural Networks

- non-linear activation functions help create interesting transformations through our data, but it allows for deep compositional functions. (complex model)
- linear activation functions \rightarrow you end up with slower model with more computation but with all of your functional complexity reduced.
- vanishing gradient problem \rightarrow ^{where} gradient = 0 so the model weights don't update and training fails.

\hookrightarrow example: ReLU (non-linear activation)



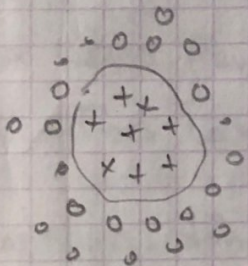
With input $< 0 \rightarrow$ the next activation layer will be 0 \rightarrow backpropagation updating \propto (error \times activation) \rightarrow we end up with gradient = 0 \rightarrow we don't change \rightarrow the training fails for that layer.

Solutions

different ReLU functions: ReLU, ELU

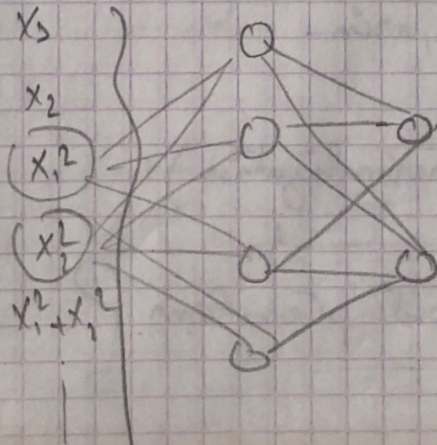
The art of multiple hidden layers

in NN is that with a many hidden layer our model will do a good classification even if we don't know "the features"



\hookrightarrow the features are x_1^2, x_2^2

We know the features so no need to multiple hidden layers!

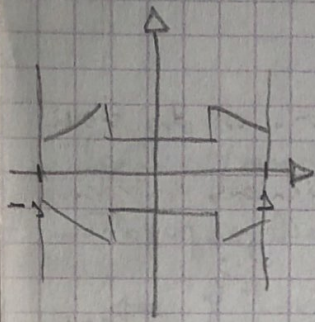


* so even if we don't know the best features for our classification, a multiple hidden layer NN can do a great classification.

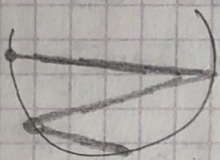
* Common failure modes for Gradient Descent:

- Gradients can vanish $\xrightarrow{\text{(solve)}}$ use ReLU instead of sigmoid can help.
- Gradients can explode $\xrightarrow{\text{ }}$ batch normalization can help.
- ReLU layers can die $\xrightarrow{\text{ }}$ Lower your learning rates.

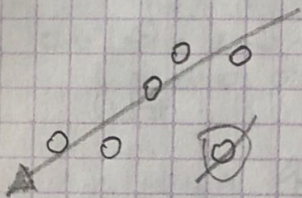
* There are benefits if feature values are small numbers:



Roughly zero-centered, $[-1, 1]$ range often works well



Small magnitudes help gradient descent converge and avoid NaN trap.



Avoiding outlier values helps with generalization

* We can use standard methods to make feature values scale to small numbers:

- linear scaling
- Hard cap (clipping) to max, min
- Log scaling
- standardization (batch normalization)

Multiclass NN output problem?

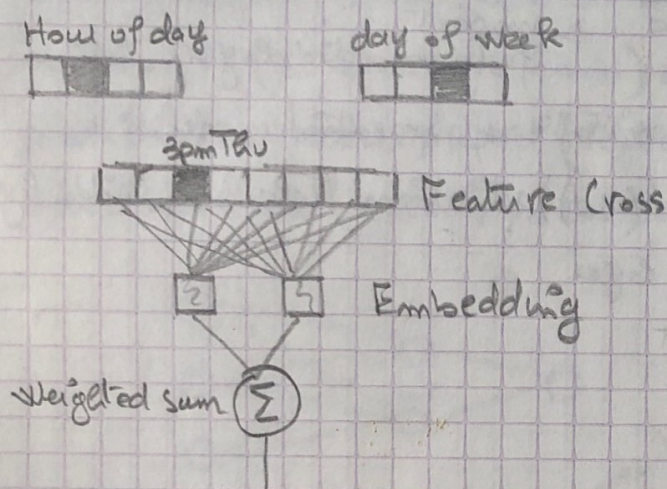
6/ Embeddings : used to → manage sparse data.

- a way to do dimensionality reduction
- increase model generalization
- cluster observations

} Embedding is a way to make the problem easier.

→ Feature crosses, structured data

Creating an embedding column from a feature cross :



} the feature cross of day hour has a big dimensional values, but we are forcing it to be represented with just 2 real value numbers (13 3) → so the model learns how to embed the feature cross in lower dimensional space.

7/ Custom estimators :

Keras : is a high level NN API written in Python but which supports TensorFlow as a backend, in other words, when you call a Keras function it turns around and calls a set of TensorFlow functions to implement that functionality.

Keras.estimator

—————