

Art and Science of ML:

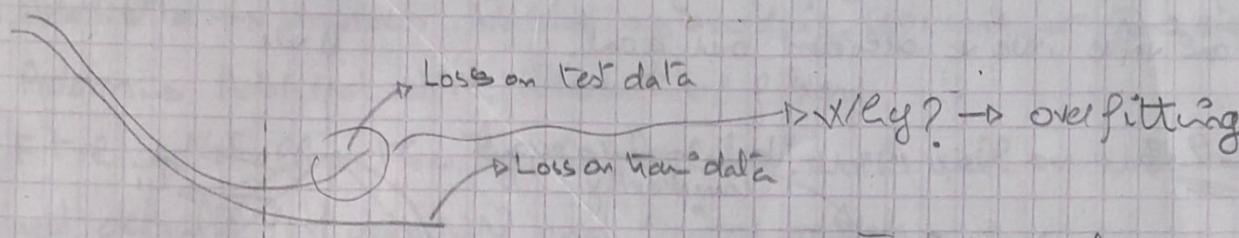
I - Intro:

→ google cloud

→ How google does ML

II - The Art of ML 8

* Regularization:



TensorFlow playground

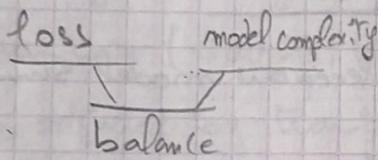
(visualizing how NN learn)

→ It's an option (a solution) but it can't help us

in a case of complex data.

oversimplified models

are useless so:



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.

→ model complexity

→ regularization

→ penalize model complexity

⇒ Minimize: loss + complexity

aim for low training error

but balance against complexity

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

↓

↑

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

as the magnitude of the weight vector

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

L2 norm: $\|w\|_2 = (\omega_0^2 + \omega_1^2 + \dots)^{1/2}$

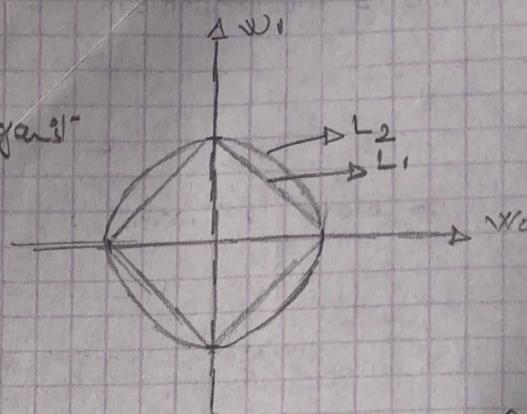
* if we keep the magnitude of our weight vector smaller than certain value, we've achieved our goal.

L2

but balance against complexity

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

aim for low training error



L1

* 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, → find the minⁿ Loss
- Momentum → reduces Lr when gradient values are small
- AdaGrad → give frequently occurring features low Lr
- AdaDelta → improves AdaGrad by avoiding reducing Lr to zero
- Adam → Adagrad with a bunch of fixes.

{ - FTRL → "Follow The regularized Leader", Wolfe's rule on wide models.
↳ Good defaults for deep NN and linear models.

III - Hyperparameter Tuning

parameters:

changed during model

training (weights, bias, ...)

(→ adjusted by training model)

↑ Loss

every High Lr

High Lr

Low Lr

good Lr

→ epochs

Hyperparameters:

set before training

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

(→ our job is to set the Hyperparameters)

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

ML Engine (MLE) for Hyperparameter Tuning

4/ a pinch of Science 8

- L₂ regularization only makes weights small, not zero. \rightarrow large and complex model!
- L₁ regularization tends to force the weights of not very predictive features to zero. \Rightarrow feature selector (Kipping bad features and keep only strongest one in the model) \Rightarrow Small model = Speed tiny \Rightarrow important for embedded models.

\rightarrow with L₁, you can end up with a smaller model but it may be less predictive \Rightarrow The elastic net is a linear combination of L₁ and L₂ regularization penalties

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

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

Why regularization is important in logistic regression?

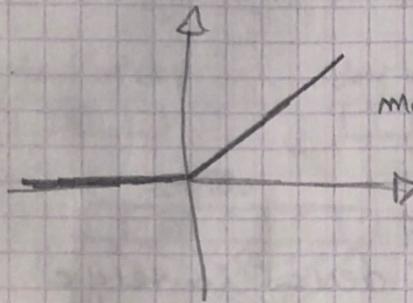
\hookrightarrow because driving the loss to zero is difficult and dangerous

- 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 → you end up with simple model with more computation but with all of your functional complexity reduced.
- Vanishing gradient problem → $\frac{d}{dx} \text{gradient} = 0$ so the model's weights don't update and training halts.

↳ example: ReLU (non-linear activation)



With input $< 0 \Rightarrow$ the next activation layer will be 0 → back propagation updating $\nabla (\text{error} \times \underbrace{\text{activation}}_{=0}) \Rightarrow$ we end up with gradient = 0 ⇒ ∇ don't change → the training fails for that layer.

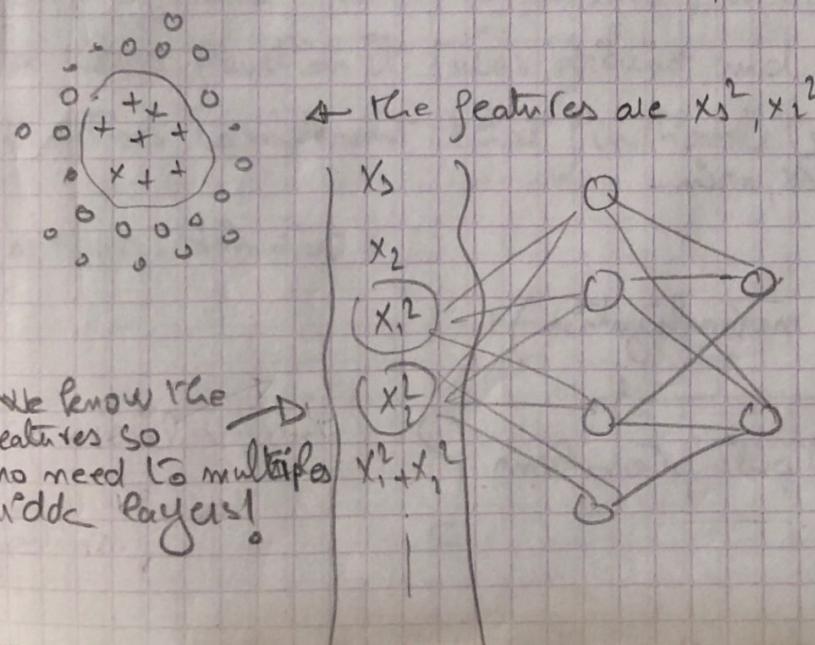
Solutions:

different ReLU patterns: ReLub

The art of multiple hidden layers

"NN is that" with a many hidden layer our model

will do a good classification even if we don't know "the features"



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

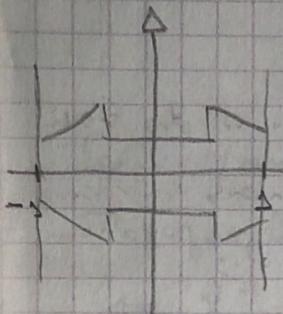
ELU

—

* Common failure modes for Gradient Descent:

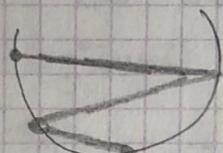
- Gradients can vanish \rightarrow use ReLU instead of sigmoid can help.
(solution)
- Gradients can explode \rightarrow batch normalization can help.
- ReLU layers can die \rightarrow Lower your learning rates.

* There are benefits if feature values are small numbers:

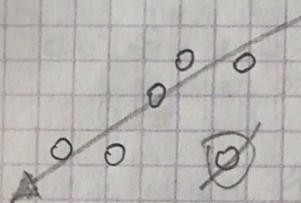


Roughly zero-centered, $[-5, 5]$ range of Len Wolfs

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:

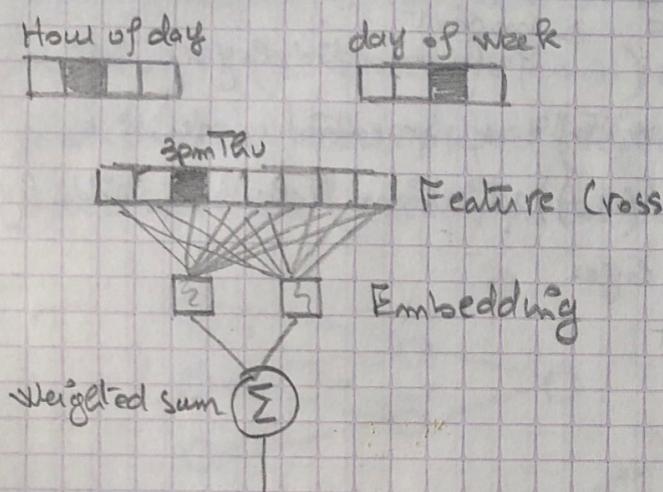
- Linear scaling
- Hard cap (clipping) to max, min
- Log scaling
- Standardization (batch normalization)

Multi-class NN output problem?

- 6/ Embeddings is 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 classes, 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 (3 5)
 → 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

