Fine-tuning Neural Network Hyperparameters

( tinh chỉnh tham số)

*Tính linh hoạt của mạng NN cx là 1 trong những nhược điểm chính*

*Vì có nhiều hyperparament để tinh chỉnh bởi bạn có thể sử dụng bất kì kiến trúc mạng nào và còn có thể thay đổi đc nhiều thức như number of layer , number of neuron , activation function , …………….*

* *Vậy làm sao để biết cái hyperparament bạn chọn là tốt nhất ???*

a\ 1 cách là thử kết hợp nhiều hyperparament và xem cái nào hoạt động tốt nhất trên valdation set (hoặc sử dụng K-fold cross-validation) với cách này thì đơn giản là sử dụng GridSearchCV or RandomizedSearchCV (chap 2)

* Tuy nhiên cách này đa số chỉ có tác dụng với những bài toán đơn giản còn khi gặp các bài slow training ( large dataset ) 🡪 nó sẽ chỉ tiếp cận 1 phần nhỏ của không gian siêu tham số.

Bạn có thể giảm bớt phần nào bằng cách thủ công

b\ Tuy nhiên may mắn có nhiều khám phá không gian tìm kiếm hiệu quả hơn nhiều là random. Ý tưởng cốt lõi là : khi 1 vùng không gian chả về tốt thì nó nên được khám phá nhiều hơn. Quan tâm vào “zooming” chương trình cho bạn và dẫn bạn đến nhiều giải pháp tốt hơn trong thời gian ngắn.

Có 1 số thư viện có thể giúp bạn :

-Hyperopt

-Hyperas , kopt or Talos :

-scikit – optimize

-Spearmint

-Sklearn- Deap:

…………….

Note : fine-tuniniing hyperparament là vấn đề vẫn đang được nghiên cứu

Có thể đọc qua bài báo : **DeepMind’s excellent 2017 paper**

**Google** cũng sử dụng phương pháp tiếp cận tiến hóa ko chỉ để tìm kiếm hyperpatament mà còn để tìm ra thiết kế 1 mạng neuron tốt nhất cho vấn đề .

Những mạng có sẵn này gọi là **AutoML** được cung cấp bới các **Cloud server.**

* Thiết kế mạng thủ công có lẽ sẽ sớm kết thúc ( nên đọc bài của GG về vấn đề này [Google AI Blog: Using Evolutionary AutoML to Discover Neural Network Architectures (googleblog.com)](https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html)
* Các evolutionary algorithms cx được dùng một cách thành công , dần tháy thế Gradient Descent . (tham khảo bài báo của Uber về thuật toán của họ *Deep Neuroevolution technique)* . [Welcoming the Era of Deep Neuroevolution - Uber Engineering Blog](https://eng.uber.com/deep-neuroevolution/)

Bất chấp những công nghệ trên thì có 1 cách dưới đây sẽ cho bạn ý tưởng về giá trị nào là hợp lí cho mỗi siêu tham số để bạn có thể xây dựng nhanh và hạn chế không gian .

1 . Number of Hidden Layer ?????

* Đối với nhiều vấn đề thì bạn có thể bắt đầu với 1 lớp ẩn duy nhất và bạn sẽ nhận đc kết quả hợp lí miễn là đủ neurons.
* Tuy nhiên với các mạng sâu thì cho chúng ta các tham số có hiệu suất tốt hơn
* Giái thích : hiểu đơn giản như bạn sẽ 1 cái cây mà ko được sử dụng copy/paste thì bạn phải vẽ tất cả mọi thứ nhưng với mạng sâu hơn bạn có thể tận dụng những thứ trc đó như khi bạn vẽ 1 cái lá thì chỉ cần copy paste cái lá cho tất cả còn lại là xong 🡪 nhanh
* Không chỉ giups hội tụ nhanh hôn , mạng sâu hơn còn giúp ta cải thiện khả năng tổng quát và cho ra bộ dữ liệu mới

-Giải thích : bạn có thể sử dụng các lớp thấp hơn để đào tạo một mạng mới 🡪 đỡ phải để mạng mới khởi tạo lại trọng số từ đầu

* Transfer learning

Tóm lại : nhiều vấn đề chỉ cần 1-2 layer là giải quyết được rồi

Với những vấn đề phức tạp hơn thì có thể tăng dần dần số layer cho với khi overrfiting

Còn với những vấn đề phức tạp hơn nữa cần cả trăm, triệu layer thì như CV hay NLP thì bạn ít khi phải đào tạo từ đầu mà có thể tận dụng các mạng trước đó (transfer learning) .

1. Number of Neuron per hidden layer

* Lớp input và output thì tùy thuộc vào dữ liệu đầu vào và vấn đề cần giải quyết
* Với lớp hidden 🡪 càng ngày càng ít neuron tại mỗi lớp , lý do vì đối tượng cấp thấp kết hợp thành ít đối tượng cấp cao hơn.
* Nói chung việc xác định xem bao nhiêu neuron hay layer vẫn là vấn đề còn đ nghiên cứu .

Có hai cách tiếp cận : tăng dần cho đến khi overfitting

Khởi tạo thật nhiều và dùng dropout để dừng lại khi có overfitting

1. Learning rate, Batch Size , ………..

* Learing rate được cho là siêu tham số quan trọng nhất . Nó tối ưu là khoảng 1 nửa MAX\_learning rate

Chọn 1 learning rate thật lớn -> cho model phân kì

Sau đó chia cho 3 🡪 try again

…………

* Optimizer : Chọn 1 thuật toán tối ưu tốt hơn Mini GD là vô cùng quan trọng
* Batch\_size : kích thước của 1 batch ko được quá nhỏ cx khoogn được quá lớn …có người đã tìm được nó tối ưu trong khoảng 20<batch\_size<32
* Activation : Relu là lựa chọn tốt cho mọi hidden layer
* Khuyến nghị nên đọc [[1206.5533] Practical recommendations for gradient-based training of deep architectures (arxiv.org)](https://arxiv.org/abs/1206.5533)