

Introduction to Deep Learning

Mnacho Echenim

Grenoble INP-Ensimag

2022-2023



Notes

Feature preprocessing

- The range of input parameters can be very diverse
 - ▶ One way of improving a neural network is to have a uniform representation of inputs
 - ▶ Force inputs to have comparable ranges
- Two main techniques
 - ▶ Data normalization, or min-max normalization
 - ★ Preprocess training data to collect min and max values along each dimension, obtain the vectors m and M of minima and maxima, respectively
 - ▶ **Data standardization**
 - ★ Impose the distribution of training data to have a mean of 0 and standard deviation of 1
 - ★ Compute the mean μ and standard deviation σ of the training data



Notes

An input standardizing layer

- Constructor parameters:
 - ▶ Mean of the training input
 - ▶ StdDev of the training input
 - ▶ Underlying layer
- Forward propagation
 - ▶ Standardize the input
 - ▶ Invoke forward propagation on the underlying layer with the standardized input
- Backpropagation
 - ▶ Deferred to the underlying layer
- Parameter update
 - ▶ Deferred to the underlying layer

Notes

Faster training with input standardization

- Compute mean μ and stddev σ of training data S
- Construct preprocessed training data $S' = \left\{ \frac{x - \mu}{\sigma} \mid x \in S \right\}$
- Construct input standardizing layer with two modes:
 - ▶ Training mode: input is fed to underlying layer with no modification \rightarrow less costly
 - ▶ Testing mode: input is standardized before being fed to underlying layer
- Train network with S'
- Set mode to Testing to evaluate network performance

Notes

Batch normalization: Presentation

- Proposed by Ioffe and Szegedy in 2016
- *"One of the most exciting recent innovations in optimizing deep neural networks"* (Deep Learning book)
- Original design: optimization of the training phase of a neural network
 - ▶ Especially for very deep neural networks
- Impressive results:
 - ▶ With the at the time best-performing ImageNet classification network
 - ▶ Matched its performance with 14 times less training steps

Notes

Features

- Larger learning rates can be applied
- More activation functions can be used
- The initialization of parameters is less of a problem
- **Batch normalization also improves generalization**
 - ▶ Regularization techniques such as Dropout are less necessary
- **But** more computations are required
 - ▶ Not trivial to implement efficiently

Notes

On Internal Covariate Shift

- Neural networks rely on the assumption that the distribution of data is the same during the training and testing phase
 - ▶ Optimal parameters for the training phase should work well on the testing phase

Question

How does this distribution of data evolve within hidden layers?

- Example: a neural network with 2 hidden layers
 - ▶ The output is $\alpha^2 = f_2(f_1(\alpha^0; \theta^1); \theta^2)$
 - ▶ α^0 has distribution \mathcal{D}_0 , α^1 has distribution $\mathcal{D}_1(\mathcal{D}_0, \theta^1)$
- Gradient descent:
 - ▶ $\theta^1 \leftarrow \theta^1 - \eta \nabla_{\theta^1} \mathcal{C}$: optimization w.r.t. distribution \mathcal{D}_0
 - ▶ $\theta^2 \leftarrow \theta^2 - \eta \nabla_{\theta^2} \mathcal{C}$: optimization w.r.t. distribution \mathcal{D}_1
- But after the update of θ^1 , the distribution for α^1 is $\mathcal{D}'_1 \neq \mathcal{D}_1$; the optimization of θ^2 may not be efficient

Notes

Principle of Batch Normalization

- Goal: reduce internal covariate shift
- How: by applying a technique similar to input standardization to hidden layers
 - ▶ The parameter updates of preceding layers then have less effect on a given layer

Params: γ, β

Input : $(\alpha_{(1)}, \dots, \alpha_{(M)})$, an input mini-batch

```
1  $\mu \leftarrow \frac{1}{M} \sum_{m=1}^M \alpha_{(m)}$ ;  
2  $\sigma \leftarrow \sqrt{\frac{1}{M} \sum_{m=1}^M (\alpha_{(m)} - \mu)^2}$ ;  
3 for  $m \leftarrow 1$  to  $M$  do  
4    $\widehat{\alpha}_{(m)} \leftarrow \frac{\alpha_{(m)} - \mu}{\sigma}$ ;  
5    $\chi_{(m)} \leftarrow \gamma \widehat{\alpha}_{(m)} + \beta$ ;  
6 end  
7 return  $(\chi_{(1)}, \dots, \chi_{(M)})$ 
```

Algorithm 1: Batch normalization algorithm

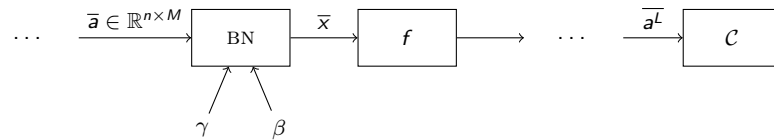
Notes

Remarks on batch normalization computations

- A batch normalization layer has a weights matrix $\Omega = \mathbf{Id}$ and an activation function $\Phi : \mathbb{R}^{n \times M} \rightarrow \mathbb{R}^{n \times M}$, for which **each column depends on the entire batch**
- At Line 4 of the algorithm, it is standard to replace σ by $\sqrt{\sigma^2 + \epsilon}$, with $\epsilon \approx 10^{-3}$
- All operations on vectors are applied **componentwise**
- γ and β are not hyperparameters, they are parameters **to be learned**
- Because $\Omega = \mathbf{Id}$, computations for backpropagation and gradient descent can be carried out componentwise

Notes

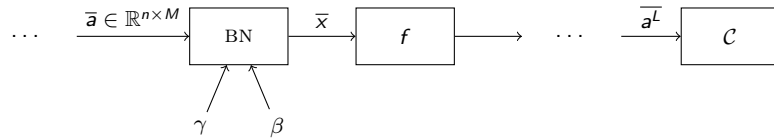
Backpropagation in batch normalization



- During backpropagation, the batch norm layer receives $[\nabla_{x_{(m)}} \mathcal{C}]_i$ as an input, where $m \in \llbracket 1, M \rrbracket$ and $i \in \llbracket 1, n \rrbracket$
- The quantities to compute are:
 - ▶ $\nabla_{\gamma} \mathcal{C}$, for the update of γ
 - ▶ $\nabla_{\beta} \mathcal{C}$, for the update of β
 - ▶ $\nabla_{a_{(j)}} \mathcal{C}$, where $j \in \llbracket 1, M \rrbracket$, for the backpropagation to the previous layer

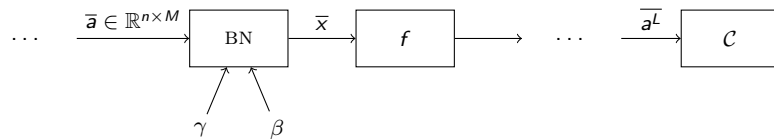
Notes

Backpropagation in batch normalization (2)



Notes

Backpropagation in batch normalization (3)



Notes

Where should batch normalization be applied?

- There are two possibilities:

- ▶ After the activation

$$[\alpha^j]' = \text{BN}(\alpha^j)$$

- ▶ Between the net input and the activation

$$[\alpha^j]' = \Phi(\text{BN}(\zeta^j))$$

- Original paper by Ioffe and Szegedy: apply between the net input and the activation

- ▶ But still some debate
- ▶ May depend on activation function
- ▶ It may be worthwhile to test both solutions

Notes

Layer normalization

- Proposed by Ba, Kiros and Hinton in 2016
- Designed to mitigate some issues with Batch Norm
 - ▶ Dependent on size of mini-batch
 - ▶ Unclear how to apply it to Recurrent Neural Networks
- Principle
 - ▶ Normalize a **single** input α at layer l into $\gamma \cdot \bar{\alpha} + \beta$
 - ▶ $\bar{\alpha} = \frac{\alpha - \mu'}{\sigma'}$
 - ▶ μ' and σ' are the mean and standard deviation over all features of layer l

Notes

Mini-batch size

- Trade-off between gradient stability and efficient computation
- Until recently sizes could be very large, up to a few thousand
- Masters, Luschi: *Revisiting Small Batch Training for Deep Neural Networks*, 2018
 - ▶ Better to choose sizes between 2 and 32

Notes

Adjusting hyperparameters

- Some of them have recommended values it is better to start with
 - ▶ Example: exponential decay rates in Adam: recommended values are $\rho_1 = 0.9$ and $\rho_2 = 0.999$
- Some common techniques
 - ▶ Manual search: can be more of an art than a science
 - ▶ Grid search: systematic search of different combinations of hyperparameters; select the combination that works best
 - ▶ Random search: randomly sample combinations of hyperparameters

Notes

Dropout recommendation

Dropout should be tuned at the same time as the size of hidden layers

- Turn dropout off ($p = 1$)
- Adjust layer sizes until the network fits the training data
- Keep the same layer sizes and retrain the network with dropout turned back on

Notes

Training, validation and test sets

Data is generally partitioned into 3 sets:

- Training set: used to optimize the parameters of the neural network
- Validation set: used to tune the hyperparameters
 - ▶ Learning rate
 - ▶ Decay rate
 - ▶ Momentum parameter
 - ▶ Architecture of the network
- Test set: used to forecast how well the network will perform on unknown data

Standard partitioning rule

Partition the data as follows:

- 70% for the training set
- 15% for the validation set
- 15% for the test set

Notes

On cross-validation

- Generally recommended when there is not much data available train and test a model
 - ▶ Principle: iteratively use every sample in the dataset at some point in the test phase
- Basic algorithm:
 - Input:** A network N , number of folds k and dataset \mathcal{S}
 - 1 create a partition (Π_1, \dots, Π_k) of \mathcal{S} ;
 - 2 **for** $i \leftarrow 1$ **to** k **do**
 - 3 | train N on $\mathcal{S} \setminus \Pi_i$;
 - 4 | $\mathcal{E}_i \leftarrow \text{validate}(N, \Pi_i)$;
 - 5 **end**
 - 6 **return** $\frac{\sum_{i=1}^k \mathcal{E}_i}{k}$
- Features:
 - ▶ Advantage: can reduce overfitting
 - ▶ Drawback: training can take much more time
 - ▶ Many existing variations (leave-one-out, nested, ...)

Notes

Open questions

- Why does gradient descent work so well on neural networks?
 - ▶ The error function is not convex
 - ▶ It is very unlikely that gradient descent will reach the global minimum
 - ▶ Yet, for an appropriate neural network architecture, the local minimum yields values that are quite close to the global one
- Why can complicated functions even be approximated?
 - ▶ Why are so many impressive results obtained on quantum mechanics simulations, image processing, speech recognition or games?
 - ▶ The same global algorithm permits to obtain these results; what similarities are there between these functions?
 - ▶ Why are there intuitively simple functions that are difficult to approximate?
 - ★ MLPs are not good at learning equality:
<https://arxiv.org/pdf/1812.01662.pdf>

Notes

Resources on deep learning techniques in finance

- B. Huge, A. Savine. *Differential Machine Learning*
 - ▶ <https://arxiv.org/pdf/2005.02347.pdf>
 - ▶ **Principle:** supervised learning or values and differentials
 - ▶ TensorFlow implementation available
- B. Lapeyre, J. Lelong. *Neural network regression for Bermudan option pricing*
 - ▶ <https://arxiv.org/pdf/1907.06474.pdf>
 - ▶ **Principle:** use a neural network to approximate the conditional expectations used to price Bermudan options
- S. Becker, P. Cheridito, A. Jentzen. *Deep optimal stopping*
 - ▶ <https://arxiv.org/pdf/1804.05394.pdf>
 - ▶ **Principle:** use Reinforcement Learning to compute optimal stopping times (used to price Bermudan options)

Notes



Other resources

- Practical advice for building and debugging a neural network:
 - ▶ <https://pcc.cs.byu.edu/2017/10/02/practical-advice-for-building-deep-neural-networks/>
 - ▶ <http://cs231n.github.io/neural-networks-3/>
- Seminar by Pierre Courtiol at Collège de France:
<https://www.college-de-france.fr/site/stephane-mallat/seminar-2018-02-21-11h15.htm>

Notes

