Title: Normalization (machine learning)

URL: https://en.wikipedia.org/wiki/Normalization_(machine_learning)

PageID: 77557393

Categories: Category:Deep learning, Category:Machine learning, Category:Neural networks, Category:Statistical data transformation

-----

Supervised learning

Unsupervised learning

Semi-supervised learning

Self-supervised learning

Reinforcement learning

Meta-learning

Online learning

Batch learning

Curriculum learning

Rule-based learning

Neuro-symbolic AI

Neuromorphic engineering

Quantum machine learning

Classification

Generative modeling

Regression

Clustering

Dimensionality reduction

Density estimation

Anomaly detection

Data cleaning

AutoML

Association rules

Semantic analysis

Structured prediction

Feature engineering

Feature learning

Learning to rank

Grammar induction

Ontology learning

Multimodal learning

Apprenticeship learning

Decision trees

Ensembles Bagging Boosting Random forest

Bagging

Boosting

Random forest

k -NN

Linear regression

Naive Bayes

Artificial neural networks

Logistic regression

Perceptron

Relevance vector machine (RVM)

Support vector machine (SVM)

BIRCH

CURE

Hierarchical

k -means

Fuzzy

Expectation–maximization (EM)

DBSCAN

OPTICS

Mean shift

Factor analysis

CCA

ICA

LDA

NMF

PCA

PGD

t-SNE

SDL

Graphical models Bayes net Conditional random field Hidden Markov

Bayes net

Conditional random field

Hidden Markov

RANSAC

k -NN

Local outlier factor

Isolation forest

Autoencoder

Deep learning

Feedforward neural network

Recurrent neural network LSTM GRU ESN reservoir computing

LSTM

GRU

ESN

reservoir computing

Boltzmann machine Restricted

Restricted

GAN

Diffusion model

SOM

Convolutional neural network U-Net LeNet AlexNet DeepDream

U-Net

LeNet

AlexNet

DeepDream

Neural field Neural radiance field Physics-informed neural networks

Neural radiance field

Physics-informed neural networks

Transformer Vision

Vision

Mamba

Spiking neural network

Memtransistor

Electrochemical RAM (ECRAM)

Q-learning

Policy gradient

SARSA

Temporal difference (TD)

Multi-agent Self-play

Self-play

Active learning

Crowdsourcing

Human-in-the-loop

Mechanistic interpretability

RLHF

Coefficient of determination

Confusion matrix

Learning curve

ROC curve

Kernel machines

Bias–variance tradeoff

Computational learning theory

Empirical risk minimization

Occam learning

PAC learning

Statistical learning

VC theory

Topological deep learning

AAAI

ECML PKDD

NeurIPS

ICML

ICLR

IJCAI

ML

JMLR

Glossary of artificial intelligence

List of datasets for machine-learning research List of datasets in computer vision and image processing

List of datasets in computer vision and image processing

Outline of machine learning

v

t

e

In machine learning , normalization is a statistical technique with various applications. There are two main forms of normalization, namely data normalization and activation normalization . Data normalization (or feature scaling) includes methods that rescale input data so that the features have the same range, mean, variance, or other statistical properties. For instance, a popular choice of feature scaling method is min-max normalization , where each feature is transformed to have the same range (typically [0 , 1] {\displaystyle [0,1]} or [− 1 , 1] {\displaystyle [-1,1]}). This solves the problem of different features having vastly different scales, for example if one feature is measured in kilometers and another in nanometers.

Activation normalization, on the other hand, is specific to deep learning , and includes methods that rescale the activation of hidden neurons inside neural networks .

Normalization is often used to:

increase the speed of training convergence,

reduce sensitivity to variations and feature scales in input data,

reduce overfitting ,

and produce better model generalization to unseen data.

Normalization techniques are often theoretically justified as reducing covariance shift, smoothing optimization landscapes, and increasing regularization , though they are mainly justified by empirical success. [1]

Batch normalization

Batch normalization (BatchNorm) [2] operates on the activations of a layer for each mini-batch.

Consider a simple feedforward network, defined by chaining together modules:

x (0) ■ x (1) ■ x (2) ■ ■ {\displaystyle x^{(0)}\mapsto x^{(1)}\mapsto x^{(2)}\mapsto \cdots }

where each network module can be a linear transform, a nonlinear activation function, a convolution, etc. x (0) {\displaystyle x^{(0)}} is the input vector, x (1) {\displaystyle x^{(1)}} is the output vector from the first module, etc.

BatchNorm is a module that can be inserted at any point in the feedforward network. For example, suppose it is inserted just after x (l) {\displaystyle x^{(l)}} , then the network would operate accordingly:

■ ■ x (l) ■ B N (x (l)) ■ x (l + 1) ■ ■ {\displaystyle \cdots \mapsto x^{(l)}\mapsto \mathrm {BN} (x^{(l)})\mapsto x^{(l+1)}\mapsto \cdots }

The BatchNorm module does not operate over individual inputs. Instead, it must operate over one batch of inputs at a time.

Concretely, suppose we have a batch of inputs x (1) (0) , x (2) (0) , … , x (B) (0) {\displaystyle x_{(1)}^{(0)},x_{(2)}^{(0)},\dots ,x_{(B)}^{(0)}} , fed all at once into the network. We would obtain in the middle of the network some vectors:

x (1) (l) , x (2) (l) , … , x (B) (l) {\displaystyle x_{(1)}^{(l)},x_{(2)}^{(l)},\dots ,x_{(B)}^{(l)}}

The BatchNorm module computes the coordinate-wise mean and variance of these vectors:

μ i (l) = 1 B ∑ b = 1 B x (b) , i (l) (σ i (l)) 2 = 1 B ∑ b = 1 B (x (b) , i (l) − μ i (l)) 2 {\displaystyle {\begin{aligned}\mu _{i}^{(l)}&={\frac {1}{B}}\sum _{b=1}^{B}x_{(b),i}^{(l)}\\(\sigma _{i}^{(l)})^{2}&={\frac {1}{B}}\sum _{b=1}^{B}(x_{(b),i}^{(l)}-\mu _{i}^{(l)})^{2}\end{aligned}}}

where i {\displaystyle i} indexes the coordinates of the vectors, and b {\displaystyle b} indexes the elements of the batch. In other words, we are considering the i {\displaystyle i} -th coordinate of each vector in the batch, and computing the mean and variance of these numbers.

It then normalizes each coordinate to have zero mean and unit variance:

x ^ (b) , i (l) = x (b) , i (l) − μ i (l) (σ i (l)) 2 + ■ {\displaystyle {\hat {x}}_{(b),i}^{(l)}={\frac {x_{(b),i}^{(l)}-\mu _{i}^{(l)}}{\sqrt {(\sigma _{i}^{(l)})^{2}+\epsilon }}}}

The ■ {\displaystyle \epsilon } is a small positive constant such as 10 − 9 {\displaystyle 10^{-9}} added to the variance for numerical stability, to avoid division by zero .

Finally, it applies a linear transformation:

y (b) , i (l) = γ i x ^ (b) , i (l) + β i {\displaystyle y_{(b),i}^{(l)}=\gamma _{i}{\hat {x}}_{(b),i}^{(l)}+\beta _{i}}

Here, γ {\displaystyle \gamma } and β {\displaystyle \beta } are parameters inside the BatchNorm module. They are learnable parameters, typically trained by gradient descent .

The following is a Python implementation of BatchNorm:

Interpretation

γ {\displaystyle \gamma } and β {\displaystyle \beta } allow the network to learn to undo the normalization, if this is beneficial. [3] BatchNorm can be interpreted as removing the purely linear transformations, so that its layers focus solely on modelling the nonlinear aspects of data, which may be beneficial, as a neural network can always be augmented with a linear transformation layer on top. [4] [3]

It is claimed in the original publication that BatchNorm works by reducing internal covariance shift, though the claim has both supporters [5] [6] and detractors. [7] [8]

Special cases

The original paper [2] recommended to only use BatchNorms after a linear transform, not after a nonlinear activation. That is, φ (B N (W x + b)) {\displaystyle \phi (\mathrm {BN} (Wx+b))} , not B N (φ (W x + b)) {\displaystyle \mathrm {BN} (\phi (Wx+b))} . Also, the bias b {\displaystyle b} does not matter, since it would be canceled by the subsequent mean subtraction, so it is of the form B N (W x) {\displaystyle \mathrm {BN} (Wx)} . That is, if a BatchNorm is preceded by a linear transform, then that linear transform's bias term is set to zero. [2]

For convolutional neural networks (CNNs), BatchNorm must preserve the translation-invariance of these models, meaning that it must treat all outputs of the same kernel as if they are different data points within a batch. [2] This is sometimes called Spatial BatchNorm, or BatchNorm2D, or per-channel BatchNorm. [9] [10]

Concretely, suppose we have a 2-dimensional convolutional layer defined by:

x h , w , c (l) = ∑ h ′ , w ′ , c ′ K h ′ − h , w ′ − w , c , c ′ (l) x h ′ , w ′ , c ′ (l − 1) + b c (l) {\displaystyle x_{h,w,c}^{(l)}=\sum _{h',w',c'}K_{h'-h,w'-w,c,c'}^{(l)}x_{h',w',c'}^{(l-1)}+b_{c}^{(l)}}

where:

x h , w , c (l) {\displaystyle x_{h,w,c}^{(l)}} is the activation of the neuron at position (h , w) {\displaystyle (h,w)} in the c {\displaystyle c} -th channel of the l {\displaystyle l} -th layer.

K Δ h , Δ w , c , c ′ (l) {\displaystyle K_{\Delta h,\Delta w,c,c'}^{(l)}} is a kernel tensor. Each channel c {\displaystyle c} corresponds to a kernel K h ′ − h , w ′ − w , c , c ′ (l) {\displaystyle K_{h'-h,w'-w,c,c'}^{(l)}} , with indices Δ h , Δ w , c ′ {\displaystyle \Delta h,\Delta w,c'} .

b c (l) {\displaystyle b_{c}^{(l)}} is the bias term for the c {\displaystyle c} -th channel of the l {\displaystyle l} -th layer.

In order to preserve the translational invariance, BatchNorm treats all outputs from the same kernel in the same batch as more data in a batch. That is, it is applied once per kernel c {\displaystyle c} (equivalently, once per channel c {\displaystyle c}), not per activation x h , w , c (l + 1) {\displaystyle x_{h,w,c}^{(l+1)}} :

μ c (l) = 1 B H W ∑ b = 1 B ∑ h = 1 H ∑ w = 1 W x (b) , h , w , c (l) (σ c (l)) 2 = 1 B H W ∑ b = 1 B ∑ h = 1 H ∑ w = 1 W (x (b) , h , w , c (l) − μ c (l)) 2 {\displaystyle {\begin{aligned}\mu _{c}^{(l)}&={\frac {1}{BHW}}\sum _{b=1}^{B}\sum _{h=1}^{H}\sum _{w=1}^{W}x_{(b),h,w,c}^{(l)}\\(\sigma _{c}^{(l)})^{2}&={\frac {1}{BHW}}\sum _{b=1}^{B}\sum _{h=1}^{H}\sum _{w=1}^{W}(x_{(b),h,w,c}^{(l)}-\mu _{c}^{(l)})^{2}\end{aligned}}}

where B {\displaystyle B} is the batch size, H {\displaystyle H} is the height of the feature map, and W {\displaystyle W} is the width of the feature map.

That is, even though there are only B {\displaystyle B} data points in a batch, all B H W {\displaystyle BHW} outputs from the kernel in this batch are treated equally. [2]

Subsequently, normalization and the linear transform is also done per kernel:

x ^ (b) , h , w , c (l) = x (b) , h , w , c (l) − μ c (l) (σ c (l)) 2 + ■ y (b) , h , w , c (l) = γ c x ^ (b) , h , w , c (l) + β c {\displaystyle {\begin{aligned}{\hat {x}}_{(b),h,w,c}^{(l)}&={\frac {x_{(b),h,w,c}^{(l)}-\mu _{c}^{(l)}}{\sqrt {(\sigma _{c}^{(l)})^{2}+\epsilon }}}\\y_{(b),h,w,c}^{(l)}&=\gamma _{c}{\hat {x}}_{(b),h,w,c}^{(l)}+\beta _{c}\end{aligned}}}

Similar considerations apply for BatchNorm for n -dimensional convolutions.

The following is a Python implementation of BatchNorm for 2D convolutions:

For multilayered recurrent neural networks (RNN), BatchNorm is usually applied only for the input-to-hidden part, not the hidden-to-hidden part. [11] Let the hidden state of the l {\displaystyle l} -th layer at time t {\displaystyle t} be h t (l) {\displaystyle h_{t}^{(l)}} . The standard RNN, without normalization, satisfies h t (l) = φ (W (l) h t l − 1 + U (l) h t − 1 l + b (l)) {\displaystyle h_{t}^{(l)}=\phi (W^{(l)}h_{t}^{l-1}+U^{(l)}h_{t-1}^{l}+b^{(l)})} where W (l) , U (l) , b (l) {\displaystyle W^{(l)},U^{(l)},b^{(l)}} are weights and biases, and φ {\displaystyle \phi } is the activation function. Applying BatchNorm, this becomes h t (l) = φ (B N (W (l) h t l − 1) + U (l) h t − 1 l) {\displaystyle h_{t}^{(l)}=\phi (\mathrm {BN} (W^{(l)}h_{t}^{l-1})+U^{(l)}h_{t-1}^{l})} There are two possible ways to define what a "batch" is in BatchNorm for RNNs: frame-wise and sequence-wise . Concretely, consider applying an RNN to process a batch of sentences. Let h b , t (l) {\displaystyle h_{b,t}^{(l)}} be the hidden state of the l {\displaystyle l} -th layer for the t {\displaystyle t} -th token of the b {\displaystyle b} -th input sentence. Then frame-wise BatchNorm means normalizing over b {\displaystyle b} : μ t (l) = 1 B ∑ b = 1 B h i , t (l) (σ t (l)) 2 = 1 B ∑ b = 1 B (h t (l) − μ t (l)) 2 {\displaystyle {\begin{aligned}\mu _{t}^{(l)}&={\frac {1}{B}}\sum _{b=1}^{B}h_{i,t}^{(l)}\\(\sigma _{t}^{(l)})^{2}&={\frac {1}{B}}\sum _{b=1}^{B}(h_{t}^{(l)}-\mu _{t}^{(l)})^{2}\end{aligned}}} and sequence-wise means normalizing over (b , t) {\displaystyle (b,t)} : μ (l) = 1 B T ∑ b = 1 B ∑ t = 1 T h i , t (l) (σ (l)) 2 = 1 B T ∑ b = 1 B ∑ t = 1 T (h t (l) − μ (l)) 2 {\displaystyle {\begin{aligned}\mu ^{(l)}&={\frac {1}{BT}}\sum _{b=1}^{B}\sum _{t=1}^{T}h_{i,t}^{(l)}\\(\sigma ^{(l)})^{2}&={\frac {1}{BT}}\sum _{b=1}^{B}\sum _{t=1}^{T}(h_{t}^{(l)}-\mu ^{(l)})^{2}\end{aligned}}} Frame-wise BatchNorm is suited for causal tasks such as next-character prediction, where future frames are unavailable, forcing normalization per frame. Sequence-wise BatchNorm is suited for tasks such as speech recognition, where the entire sequences are available, but with variable lengths. In a batch, the smaller sequences are padded with zeroes to match the size of the longest sequence of the batch. In such setups, frame-wise is not recommended, because the number of unpadded frames decreases along the time axis, leading to increasingly poorer statistics estimates. [11]

It is also possible to apply BatchNorm to LSTMs . [12]

Improvements

BatchNorm has been very popular and there were many attempted improvements. Some examples include: [13]

ghost batching: randomly partition a batch into sub-batches and perform BatchNorm separately on each;

weight decay on γ {\displaystyle \gamma } and β {\displaystyle \beta } ;

and combining BatchNorm with GroupNorm.

A particular problem with BatchNorm is that during training, the mean and variance are calculated on the fly for each batch (usually as an exponential moving average), but during inference, the mean and variance were frozen from those calculated during training. This train-test disparity degrades performance. The disparity can be decreased by simulating the moving average during inference: [13] : Eq. 3

μ = α E [x] + (1 − α) μ x , train σ 2 = (α E [x] 2 + (1 − α) μ x 2 , train) − μ 2 {\displaystyle {\begin{aligned}\mu &=\alpha E[x]+(1-\alpha)\mu _{x,{\text{ train}}}\\\sigma ^{2}&=(\alpha E[x]^{2}+(1-\alpha)\mu _{x^{2},{\text{ train}}})-\mu ^{2}\end{aligned}}}

where α {\displaystyle \alpha } is a hyperparameter to be optimized on a validation set.

Other works attempt to eliminate BatchNorm, such as the Normalizer-Free ResNet. [14]

Layer normalization

Layer normalization (LayerNorm) [15] is a popular alternative to BatchNorm. Unlike BatchNorm, which normalizes activations across the batch dimension for a given feature, LayerNorm normalizes across all the features within a single data sample. Compared to BatchNorm, LayerNorm's performance is not affected by batch size. It is a key component of transformer models.

For a given data input and layer, LayerNorm computes the mean μ {\displaystyle \mu } and variance σ 2 {\displaystyle \sigma ^{2}} over all the neurons in the layer. Similar to BatchNorm, learnable parameters γ {\displaystyle \gamma } (scale) and β {\displaystyle \beta } (shift) are applied. It is defined by:

x i ^ = x i − μ σ 2 + ■ , y i = γ i x i ^ + β i {\displaystyle {\hat {x_{i}}}={\frac {x_{i}-\mu }{\sqrt {\sigma ^{2}+\epsilon }}},\quad y_{i}=\gamma _{i}{\hat {x_{i}}}+\beta _{i}}

where:

μ = 1 D ∑ i = 1 D x i , σ 2 = 1 D ∑ i = 1 D (x i − μ) 2 {\displaystyle \mu ={\frac {1}{D}}\sum _{i=1}^{D}x_{i},\quad \sigma ^{2}={\frac {1}{D}}\sum _{i=1}^{D}(x_{i}-\mu)^{2}}

and the index i {\displaystyle i} ranges over the neurons in that layer.

Examples

For example, in CNN, a LayerNorm applies to all activations in a layer. In the previous notation, we have:

μ (l) = 1 H W C ∑ h = 1 H ∑ w = 1 W ∑ c = 1 C x h , w , c (l) (σ (l)) 2 = 1 H W C ∑ h = 1 H ∑ w = 1 W ∑ c = 1 C (x h , w , c (l) − μ (l)) 2 x ^ h , w , c (l) = x ^ h , w , c (l) − μ (l) (σ (l)) 2 + ■ y h , w , c (l) = γ (l) x ^ h , w , c (l) + β (l) {\displaystyle {\begin{aligned}\mu ^{(l)}&={\frac {1}{HWC}}\sum _{h=1}^{H}\sum _{w=1}^{W}\sum _{c=1}^{C}x_{h,w,c}^{(l)}\\(\sigma ^{(l)})^{2}&={\frac {1}{HWC}}\sum _{h=1}^{H}\sum _{w=1}^{W}\sum _{c=1}^{C}(x_{h,w,c}^{(l)}-\mu ^{(l)})^{2}\\{\hat {x}}_{h,w,c}^{(l)}&={\frac {{\hat {x}}_{h,w,c}^{(l)}-\mu ^{(l)}}{\sqrt {(\sigma ^{(l)})^{2}+\epsilon }}}\\y_{h,w,c}^{(l)}&=\gamma ^{(l)}{\hat {x}}_{h,w,c}^{(l)}+\beta ^{(l)}\end{aligned}}}

Notice that the batch index b {\displaystyle b} is removed, while the channel index c {\displaystyle c} is added.

In recurrent neural networks [15] and transformers , [16] LayerNorm is applied individually to each timestep. For example, if the hidden vector in an RNN at timestep t {\displaystyle t} is x (t) ∈ R D {\displaystyle x^{(t)}\in \mathbb {R} ^{D}} , where D {\displaystyle D} is the dimension of the hidden vector, then LayerNorm will be applied with:

x i ^ (t) = x i (t) − μ (t) (σ (t)) 2 + ■ , y i (t) = γ i x i ^ (t) + β i {\displaystyle {\hat {x_{i}}}^{(t)}={\frac {x_{i}^{(t)}-\mu ^{(t)}}{\sqrt {(\sigma ^{(t)})^{2}+\epsilon }}},\quad y_{i}^{(t)}=\gamma _{i}{\hat {x_{i}}}^{(t)}+\beta _{i}}

where:

μ (t) = 1 D ∑ i = 1 D x i (t) , (σ (t)) 2 = 1 D ∑ i = 1 D (x i (t) − μ (t)) 2 {\displaystyle \mu ^{(t)}={\frac {1}{D}}\sum _{i=1}^{D}x_{i}^{(t)},\quad (\sigma ^{(t)})^{2}={\frac {1}{D}}\sum _{i=1}^{D}(x_{i}^{(t)}-\mu ^{(t)})^{2}}

Root mean square layer normalization

Root mean square layer normalization (RMSNorm): [17]

x i ^ = x i 1 D ∑ i = 1 D x i 2 , y i = γ x i ^ + β {\displaystyle {\hat {x_{i}}}={\frac {x_{i}}{\sqrt {{\frac {1}{D}}\sum _{i=1}^{D}x_{i}^{2}}}},\quad y_{i}=\gamma {\hat {x_{i}}}+\beta }

Essentially, it is LayerNorm where we enforce μ , ■ = 0 {\displaystyle \mu ,\epsilon =0} . It is also called L2 normalization . It is a special case of Lp normalization , or power normalization : x i ^ = x i (1 D ∑ i = 1 D | x i | p) 1 / p , y i = γ x i ^ + β {\displaystyle {\hat {x_{i}}}={\frac {x_{i}}{\left({\frac {1}{D}}\sum _{i=1}^{D}|x_{i}|^{p}\right)^{1/p}}},\quad y_{i}=\gamma {\hat {x_{i}}}+\beta } where p > 0 {\displaystyle p>0} is a constant.

Adaptive

Adaptive layer norm (adaLN) computes the γ , β {\displaystyle \gamma ,\beta } in a LayerNorm not from the layer activation itself, but from other data. It was first proposed for CNNs, [18] and has been used effectively in diffusion transformers (DiTs). [19] For example, in a DiT, the conditioning information (such as a text encoding vector) is processed by a multilayer perceptron into γ , β {\displaystyle \gamma ,\beta } , which is then applied in the LayerNorm module of a transformer.

Weight normalization

Weight normalization (WeightNorm) [20] is a technique inspired by BatchNorm that normalizes weight matrices in a neural network, rather than its activations.

One example is spectral normalization , which divides weight matrices by their spectral norm . The spectral normalization is used in generative adversarial networks (GANs) such as the Wasserstein GAN . [21] The spectral radius can be efficiently computed by the following algorithm:

INPUT matrix W {\displaystyle W} and initial guess x {\displaystyle x}

Iterate x ■ 1 ■ W x ■ 2 W x {\displaystyle x\mapsto {\frac {1}{\|Wx\|_{2}}}Wx} to convergence x ∗ {\displaystyle x^{*}} . This is the eigenvector of W {\displaystyle W} with eigenvalue ■ W ■ s {\displaystyle \|W\|_{s}} .

RETURN x ∗ , ■ W x ∗ ■ 2 {\displaystyle x^{*},\|Wx^{*}\|_{2}}

By reassigning W i ← W i ■ W i ■ s {\displaystyle W_{i}\leftarrow {\frac {W_{i}}{\|W_{i}\|_{s}}}} after each update of the discriminator, we can upper-bound ■ W i ■ s ≤ 1 {\displaystyle \|W_{i}\|_{s}\leq 1} , and thus upper-bound ■ D ■ L {\displaystyle \|D\|_{L}} .

The algorithm can be further accelerated by memoization : at step t {\displaystyle t} , store x i ∗ (t) {\displaystyle x_{i}^{*}(t)} . Then, at step t + 1 {\displaystyle t+1} , use x i ∗ (t) {\displaystyle x_{i}^{*}(t)} as the initial guess for the algorithm. Since W i (t + 1) {\displaystyle W_{i}(t+1)} is very close to W i (t) {\displaystyle W_{i}(t)} , so is x i ∗ (t) {\displaystyle x_{i}^{*}(t)} to x i ∗ (t + 1) {\displaystyle x_{i}^{*}(t+1)} , thus allowing rapid convergence.

CNN-specific normalization

There are some activation normalization techniques that are only used for CNNs.

Response normalization

Local response normalization [22] was used in AlexNet . It was applied in a convolutional layer, just after a nonlinear activation function. It was defined by:

b x , y i = a x , y i (k + α ∑ j = max (0 , i − n / 2) min (N − 1 , i + n / 2) (a x , y j) 2) β {\displaystyle b_{x,y}^{i}={\frac {a_{x,y}^{i}}{\left(k+\alpha \sum _{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)}\left(a_{x,y}^{j}\right)^{2}\right)^{\beta }}}}

where a x , y i {\displaystyle a_{x,y}^{i}} is the activation of the neuron at location (x , y) {\displaystyle (x,y)} and channel i {\displaystyle i} . I.e., each pixel in a channel is suppressed by the activations of the same pixel in its adjacent channels.

k , n , α , β {\displaystyle k,n,\alpha ,\beta } are hyperparameters picked by using a validation set.

It was a variant of the earlier local contrast normalization . [23]

b x , y i = a x , y i (k + α ∑ j = max (0 , i − n / 2) min (N − 1 , i + n / 2) (a x , y j − a ¯ x , y j) 2) β {\displaystyle b_{x,y}^{i}={\frac {a_{x,y}^{i}}{\left(k+\alpha \sum _{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)}\left(a_{x,y}^{j}-{\bar {a}}_{x,y}^{j}\right)^{2}\right)^{\beta }}}}

where a ¯ x , y j {\displaystyle {\bar {a}}_{x,y}^{j}} is the average activation in a small window centered on location (x , y) {\displaystyle (x,y)} and channel i {\displaystyle i} . The hyperparameters k , n , α , β {\displaystyle k,n,\alpha ,\beta } , and the size of the small window, are picked by using a validation set.

Similar methods were called divisive normalization , as they divide activations by a number depending on the activations. They were originally inspired by biology, where it was used to explain nonlinear responses of cortical neurons and nonlinear masking in visual perception. [24]

Both kinds of local normalization were obviated by batch normalization, which is a more global form of normalization. [25]

Response normalization reappeared in ConvNeXT-2 as global response normalization . [26]

Group normalization

Group normalization (GroupNorm) [27] is a technique also solely used for CNNs. It can be understood as the LayerNorm for CNN applied once per channel group.

Suppose at a layer l {\displaystyle l} , there are channels 1 , 2 , … , C {\displaystyle 1,2,\dots ,C} , then it is partitioned into groups g 1 , g 2 , … , g G {\displaystyle g_{1},g_{2},\dots ,g_{G}} . Then, LayerNorm is applied to each group.

Instance normalization

Instance normalization (InstanceNorm), or contrast normalization , is a technique first developed for neural style transfer , and is also only used for CNNs. [28] It can be understood as the LayerNorm for CNN applied once per channel, or equivalently, as group normalization where each group consists of a single channel:

μ c (l) = 1 H W ∑ h = 1 H ∑ w = 1 W x h , w , c (l) (σ c (l)) 2 = 1 H W ∑ h = 1 H ∑ w = 1 W (x h , w , c (l) − μ c (l)) 2 x ^ h , w , c (l) = x ^ h , w , c (l) − μ c (l) (σ c (l)) 2 + ■ y h , w , c (l) = γ c (l) x ^ h , w , c (l) + β c (l) {\displaystyle {\begin{aligned}\mu _{c}^{(l)}&={\frac {1}{HW}}\sum _{h=1}^{H}\sum _{w=1}^{W}x_{h,w,c}^{(l)}\\(\sigma _{c}^{(l)})^{2}&={\frac {1}{HW}}\sum _{h=1}^{H}\sum _{w=1}^{W}(x_{h,w,c}^{(l)}-\mu _{c}^{(l)})^{2}\\{\hat {x}}_{h,w,c}^{(l)}&={\frac {{\hat {x}}_{h,w,c}^{(l)}-\mu _{c}^{(l)}}{\sqrt {(\sigma _{c}^{(l)})^{2}+\epsilon }}}\\y_{h,w,c}^{(l)}&=\gamma _{c}^{(l)}{\hat {x}}_{h,w,c}^{(l)}+\beta _{c}^{(l)}\end{aligned}}}

Adaptive instance normalization

Adaptive instance normalization (AdaIN) is a variant of instance normalization, designed specifically for neural style transfer with CNNs, rather than just CNNs in general. [29]

In the AdaIN method of style transfer, we take a CNN and two input images, one for content and one for style . Each image is processed through the same CNN, and at a certain layer l {\displaystyle l} , AdaIn is applied.

Let x (l) , content {\displaystyle x^{(l),{\text{ content}}}} be the activation in the content image, and x (l) , style {\displaystyle x^{(l),{\text{ style}}}} be the activation in the style image. Then, AdaIn first computes the mean and variance of the activations of the content image x ′ (l) {\displaystyle x'^{(l)}} , then uses those as the γ , β {\displaystyle \gamma ,\beta } for InstanceNorm on x (l) , content {\displaystyle x^{(l),{\text{ content}}}} . Note that x (l) , style {\displaystyle x^{(l),{\text{ style}}}} itself remains unchanged. Explicitly, we have:

y h , w , c (l) , content = σ c (l) , style (x h , w , c (l) , content − μ c (l) , content (σ c (l) , content) 2 + ■) + μ c (l) , style {\displaystyle {\begin{aligned}y_{h,w,c}^{(l),{\text{ content}}}&=\sigma _{c}^{(l),{\text{ style}}}\left({\frac {x_{h,w,c}^{(l),{\text{ content}}}-\mu _{c}^{(l),{\text{ content}}}}{\sqrt {(\sigma _{c}^{(l),{\text{ content}}})^{2}+\epsilon }}}\right)+\mu _{c}^{(l),{\text{ style}}}\end{aligned}}}

Transformers

Some normalization methods were designed for use in transformers .

The original 2017 transformer used the "post-LN" configuration for its LayerNorms. It was difficult to train, and required careful hyperparameter tuning and a "warm-up" in learning rate , where it starts small and gradually increases. The pre-LN convention, proposed several times in 2018, [30] was found to be easier to train, requiring no warm-up, leading to faster convergence. [31]

FixNorm [32] and ScaleNorm [33] both normalize activation vectors in a transformer. The FixNorm method divides the output vectors from a transformer by their L2 norms, then multiplies by a learned parameter g {\displaystyle g} . The ScaleNorm replaces all LayerNorms inside a transformer by division with L2 norm, then multiplying by a learned parameter g ′ {\displaystyle g'} (shared by all ScaleNorm modules of a transformer). Query-Key normalization (QKNorm) [34] normalizes query and key vectors to have unit L2 norm.

In nGPT , many vectors are normalized to have unit L2 norm: [35] hidden state vectors, input and output embedding vectors, weight matrix columns, and query and key vectors.

Miscellaneous

Gradient normalization (GradNorm) [36] normalizes gradient vectors during backpropagation.

See also

Data preprocessing

Feature scaling

References

Further reading

"Normalization Layers" . labml.ai Deep Learning Paper Implementations . Retrieved 2024-08-07 .

v

t

e

History timeline

timeline

Companies

Projects

Parameter Hyperparameter

Hyperparameter

Loss functions

Regression Bias–variance tradeoff Double descent Overfitting

Bias–variance tradeoff

Double descent

Overfitting

Clustering

Gradient descent SGD Quasi-Newton method Conjugate gradient method

SGD

Quasi-Newton method

Conjugate gradient method

Backpropagation

Attention

Convolution

Normalization Batchnorm

Batchnorm

Activation Softmax Sigmoid Rectifier

Softmax

Sigmoid

Rectifier

Gating

Weight initialization

Regularization

Datasets Augmentation

Augmentation

Prompt engineering

Reinforcement learning Q-learning SARSA Imitation Policy gradient

Q-learning

SARSA

Imitation

Policy gradient

Diffusion

Latent diffusion model

Autoregression

Adversary

RAG

Uncanny valley

RLHF

Self-supervised learning

Reflection

Recursive self-improvement

Hallucination

Word embedding

Vibe coding

Machine learning In-context learning

In-context learning

Artificial neural network Deep learning

Deep learning

Language model Large language model NMT

Large language model

NMT

Reasoning language model

Model Context Protocol

Intelligent agent

Artificial human companion

Humanity's Last Exam

Artificial general intelligence (AGI)

AlexNet

WaveNet

Human image synthesis

HWR

OCR

Computer vision

Speech synthesis 15.ai ElevenLabs

15.ai

ElevenLabs

Speech recognition Whisper

Whisper

Facial recognition

AlphaFold

Text-to-image models Aurora DALL-E Firefly Flux Ideogram Imagen Midjourney Recraft Stable Diffusion

Aurora

DALL-E

Firefly

Flux

Ideogram

Imagen

Midjourney

Recraft

Stable Diffusion

Text-to-video models Dream Machine Runway Gen Hailuo AI Kling Sora Veo

Dream Machine

Runway Gen

Hailuo AI

Kling

Sora

Veo

Music generation Riffusion Suno AI Udio

Riffusion

Suno AI

Udio

Word2vec

Seq2seq

GloVe

BERT

T5

Llama

Chinchilla AI

PaLM

GPT 1 2 3 J ChatGPT 4 4o o1 o3 4.5 4.1 o4-mini 5

1

2

3

J

ChatGPT

4

4o

o1

o3

4.5

4.1

o4-mini

5

Claude

Gemini Gemini (language model) Gemma

Gemini (language model)

Gemma

Grok

LaMDA

BLOOM

DBRX

Project Debater

IBM Watson

IBM Watsonx

Granite

PanGu-Σ

DeepSeek

Qwen

AlphaGo

AlphaZero

OpenAI Five

Self-driving car

MuZero

Action selection AutoGPT

AutoGPT

Robot control

Alan Turing

Warren Sturgis McCulloch

Walter Pitts

John von Neumann

Claude Shannon

Shun'ichi Amari

Kunihiko Fukushima

Takeo Kanade

Marvin Minsky

John McCarthy

Nathaniel Rochester

Allen Newell

Cliff Shaw

Herbert A. Simon

Oliver Selfridge

Frank Rosenblatt

Bernard Widrow

Joseph Weizenbaum

Seymour Papert

Seppo Linnainmaa

Paul Werbos

Geoffrey Hinton

John Hopfield

Jürgen Schmidhuber

Yann LeCun

Yoshua Bengio

Lotfi A. Zadeh

Stephen Grossberg

Alex Graves

James Goodnight

Andrew Ng

Fei-Fei Li

Alex Krizhevsky

Ilya Sutskever

Oriol Vinyals

Quoc V. Le

Ian Goodfellow

Demis Hassabis

David Silver

Andrej Karpathy

Ashish Vaswani

Noam Shazeer

Aidan Gomez

John Schulman

Mustafa Suleyman

Jan Leike

Daniel Kokotajlo

François Chollet

Neural Turing machine

Differentiable neural computer

Transformer Vision transformer (ViT)

Vision transformer (ViT)

Recurrent neural network (RNN)

Long short-term memory (LSTM)

Gated recurrent unit (GRU)

Echo state network

Multilayer perceptron (MLP)

Convolutional neural network (CNN)

Residual neural network (RNN)

Highway network

Mamba

Autoencoder

Variational autoencoder (VAE)

Generative adversarial network (GAN)

Graph neural network (GNN)

Category