MO434 - Deep Learning
Art of Training Deep Neural Networks

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

Agenda

- Preparing data for training, validation, and test.
- Selecting a DNN architecture and its hyperparameters.
- Dealing with loss gradient instabilities.
- Further speeding up convergence.
- Scheduling the learning rate and avoiding under/overfitting
- Training and evaluating NNs in practice.

Data preparation

- A dataset $\mathcal{Z} = (X, Y)$ must be randomly divided into training (X_{tr}, Y_{tr}), validation (X_{vl}, Y_{vl}), and test (X_{ts}, Y_{ts}) sets by simulating the real distribution of samples.

- While (X_{tr}, Y_{tr}) is used to train the DNN, its validation on (X_{vl}, Y_{vl}) provides insights about training that guide the choice of the DNN's hyperparameters.

- $(X_{ts}, Y_{ts}$ is reserved to evaluate how well the DNN performs on unseen data.

- One may use a k-fold cross validation by dividing $(X_{tr}, Y_{tr}) \cup (X_{vl}, Y_{vl})$ into k parts, holding each part per time for (X_{vl}, Y_{vl}) and using the remaining samples for (X_{tr}, Y_{tr}). However, multiple splits of $\mathcal{Z}$ should be used to measure mean and standard deviation on the test sets.

Data preparation

- Since the network weights are updated during a backward pass after each forward pass (one epoch), it is crucial to set a suitable number of updates per epoch by dividing (X_{tr}, Y_{tr}) into batches and updating weights per batch.

Data preparation

- Since the network weights are updated during a backward pass after each forward pass (one epoch), it is crucial to set a suitable number of updates per epoch by dividing (X_{tr}, Y_{tr}) into batches and updating weights per batch.
- The batch size (e.g., $[2, 32]$) has great influence on the training process.

Data preparation

- Since the network weights are updated during a backward pass after each forward pass (one epoch), it is crucial to set a suitable number of updates per epoch by dividing (X_{tr}, Y_{tr}) into batches and updating weights per batch.

- The batch size (e.g., $[2, 32]$) has great influence on the training process.

- Data augmentation is another data preparation trick that speeds up convergence by coping with the lack of training samples.

Data preparation

- Since the network weights are updated during a backward pass after each forward pass (one epoch), it is crucial to set a suitable number of updates per epoch by dividing (X_{tr}, Y_{tr}) into batches and updating weights per batch.

- The batch size (e.g., $[2, 32]$) has great influence on the training process.

- Data augmentation is another data preparation trick that speeds up convergence by coping with the lack of training samples.

Let's see a simple example of data preparation ▸ DATA PREPARATION

Selecting a DNN architecture and its hyperparameters

- A shallow network with one hidden layer and one decision layer can already solve many problems.

Selecting a DNN architecture and its hyperparameters

- A shallow network with one hidden layer and one decision layer can already solve many problems.

- However, more complex problems require too many neurons per hidden layer to be solved by shallow networks.

Selecting a DNN architecture and its hyperparameters

- A shallow network with one hidden layer and one decision layer can already solve many problems.

- However, more complex problems require too many neurons per hidden layer to be solved by shallow networks.

- Deep neural networks can be created with three or more hidden layers, but those dense layers with too many weights are difficult to be optimized.

Selecting a DNN architecture and its hyperparameters

- A shallow network with one hidden layer and one decision layer can already solve many problems.
- However, more complex problems require too many neurons per hidden layer to be solved by shallow networks.
- Deep neural networks can be created with three or more hidden layers, but those dense layers with too many weights are difficult to be optimized.
- Convolutional layers have considerably reduced that number of weights, making viable and successful the use of DNNs.

Selecting a DNN architecture and its hyperparameters

- Classical DNNs may have tens to hundreds of convolutional layers followed by two or three dense layers - a Convolutional Neural Network (CNN).

Selecting a DNN architecture and its hyperparameters

- Classical DNNs may have tens to hundreds of convolutional layers followed by two or three dense layers - a Convolutional Neural Network (CNN).

- Heuristic optimization (*Network Architecture Search* - NAS) can be applied to determine the hyperparameters [1, 2].

Selecting a DNN architecture and its hyperparameters

- Classical DNNs may have tens to hundreds of convolutional layers followed by two or three dense layers - a Convolutional Neural Network (CNN).

- Heuristic optimization (*Network Architecture Search* - NAS) can be applied to determine the hyperparameters [1, 2].

- However, the number of hyperparameter combinations is high and NAS may not consider the entire search space.

Selecting a DNN architecture and its hyperparameters

- Classical DNNs may have tens to hundreds of convolutional layers followed by two or three dense layers - a Convolutional Neural Network (CNN).

- Heuristic optimization (*Network Architecture Search* - NAS) can be applied to determine the hyperparameters [1, 2].

- However, the number of hyperparameter combinations is high and NAS may not consider the entire search space.

- We have also seen the choice of suitable activation and loss functions for regression and classification.

Selecting a DNN architecture and its hyperparameters

- Classical DNNs may have tens to hundreds of convolutional layers followed by two or three dense layers - a Convolutional Neural Network (CNN).

- Heuristic optimization (*Network Architecture Search* - NAS) can be applied to determine the hyperparameters [1, 2].

- However, the number of hyperparameter combinations is high and NAS may not consider the entire search space.

- We have also seen the choice of suitable activation and loss functions for regression and classification.

Ironically, except for NAS, architecture learning is still a feature engineering task.

Selecting a DNN architecture and its hyperparameters

- The simplification of existing architectures, pretrained on the ImageNet, have been actively pursued [3, 4].

Selecting a DNN architecture and its hyperparameters

- The simplification of existing architectures, pretrained on the ImageNet, have been actively pursued [3, 4].

- Knowledge distillation can be used to train shallower networks from scratch using the “knowledge” of pre-trained deep networks [5].

Selecting a DNN architecture and its hyperparameters

- The simplification of existing architectures, pretrained on the ImageNet, have been actively pursued [3, 4].
- Knowledge distillation can be used to train shallower networks from scratch using the “knowledge” of pre-trained deep networks [5].
- However, efficiency is not the only problem.

Selecting a DNN architecture and its hyperparameters

- The simplification of existing architectures, pretrained on the ImageNet, have been actively pursued [3, 4].

- Knowledge distillation can be used to train shallower networks from scratch using the “knowledge” of pre-trained deep networks [5].

- However, efficiency is not the only problem.

- We need methods to build effective DNNs with minimum human effort in data annotation [6, 7, 8, 9] and, ideally,

Selecting a DNN architecture and its hyperparameters

- The simplification of existing architectures, pretrained on the ImageNet, have been actively pursued [3, 4].
- Knowledge distillation can be used to train shallower networks from scratch using the “knowledge” of pre-trained deep networks [5].
- However, efficiency is not the only problem.
- We need methods to build effective DNNs with minimum human effort in data annotation [6, 7, 8, 9] and, ideally,
- from scratch guided by data visualization, optimization criteria, and user intervention [10, 11, 12].

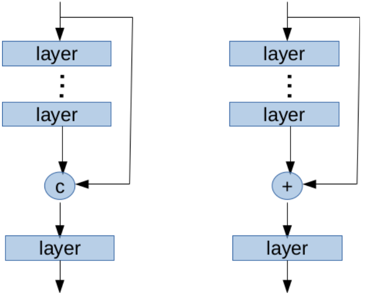
Dealing with loss gradient instabilities

- The loss gradients may either reduce to zero towards the initial layers (vanishing gradient) or increase too much towards the last layers (exploding gradient).

Dealing with loss gradient instabilities

- The loss gradients may either reduce to zero towards the initial layers (vanishing gradient) or increase too much towards the last layers (exploding gradient).

- Concatenation or addition of a previous layer input to the output of a subsequent layer is a way to avoid the vanishing gradient problem.



Dealing with loss gradient instabilities

- Indeed, a suitable choice of weight initialization and activation function can reduce gradient variations along the layers, amending both types of gradient problems.

Dealing with loss gradient instabilities

- Indeed, a suitable choice of weight initialization and activation function can reduce gradient variations along the layers, amending both types of gradient problems.

- The weights of each layer may be randomly generated by a normal distribution with mean 0 and variance σ^2 or by a uniform distribution from $-r$ to $+r$, with $r = \sqrt{3\sigma^2}$.

Initialization	activation function	σ^2
Xavier/Glorot	None, tanh, logistic, softmax	$1/fan_{avg}$
He	ReLU and variants	$2/fan_{in}$
LeCun	SELU	$1/fan_{in}$

where fan_{in} and fan_{out} are the numbers of input and output channels, $fan_{avg} = \frac{fan_{in}+fan_{out}}{2}$.

Dealing with loss gradient instabilities

- Batch normalization can additionally treat gradient instabilities during the training process, speeding up convergence.

Dealing with loss gradient instabilities

- Batch normalization can additionally treat gradient instabilities during the training process, speeding up convergence.
- Let $\boldsymbol{x}^{(i)}$ be the i-th feature vector of a batch $\mathcal{B} \subset X_{tr}$.

Dealing with loss gradient instabilities

- Batch normalization can additionally treat gradient instabilities during the training process, speeding up convergence.

- Let $\boldsymbol{x}^{(i)}$ be the i-th feature vector of a batch $\mathcal{B} \subset X_{tr}$.

- Batch normalization creates a new feature vector $\boldsymbol{z}^{(i)}$ as follows.

$$
\begin{aligned}
\boldsymbol{\mu}_{\mathcal{B}} &\leftarrow \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}\boldsymbol{x}^{(i)} \\
\boldsymbol{\sigma}^2_{\mathcal{B}} &\leftarrow \frac{1}{|\mathcal{B}|}\sum_{i=1}^{|\mathcal{B}|}\left(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\mathcal{B}}\right)^2
\end{aligned}
$$

Dealing with loss gradient instabilities

$$\boldsymbol{z}^{(i)} \quad \leftarrow \quad \boldsymbol{\gamma} \otimes \frac{\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\boldsymbol{\sigma}^2_{\mathcal{B}} + \epsilon}} + \boldsymbol{\beta}$$

where

- $\boldsymbol{\gamma}$ is a scale parameter vector for the layer, with one scale value per input.
- $\otimes$ is element-wise multiplication.
- $\boldsymbol{\beta}$ is a shift parameter vector for the layer, with one shift value per input.
- ϵ is typically 10^{-5} to avoid division by zero.

obs: all operations are element-wise from now on.

Dealing with loss gradient instabilities

- Gradient clipping is another possibility to maintain the values within a given interval (e.g., $[-1, 1]$).

Dealing with loss gradient instabilities

- Gradient clipping is another possibility to maintain the values within a given interval (e.g., $[-1, 1]$).

- Using pretrained layers (backbone) from another network to generate an input to the current network with the remaining layers for training is another way to treat gradient instabilities and speed up convergence (transfer learning).

Dealing with loss gradient instabilities

- Gradient clipping is another possibility to maintain the values within a given interval (e.g., $[-1, 1]$).

- Using pretrained layers (backbone) from another network to generate an input to the current network with the remaining layers for training is another way to treat gradient instabilities and speed up convergence (transfer learning).

- Finally, momentum and faster optimizers than SGD can be used to speed up convergence.

Further speeding up convergence

A momentum α (e.g., 0.9) can reduce oscillation in the loss values as follows.

$$\begin{aligned}\Delta \boldsymbol{w}_j^r(i) &\leftarrow \alpha \Delta \boldsymbol{w}_j^r(i-1) - \mu \sum_{s \in \mathcal{B} \subset \mathcal{Z}_{tr}} \delta_j^r(s) \boldsymbol{y}^{r-1}(s), \\ \boldsymbol{w}_j^r(i+1) &\leftarrow \boldsymbol{w}_j^r(i) + \Delta \boldsymbol{w}_j^r(i).\end{aligned}$$

For the sake of simplicity, these equations may be rewritten as

$$\begin{aligned}\boldsymbol{m} &\leftarrow \alpha \boldsymbol{m} - \mu \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{m}.\end{aligned}$$

where μ is the learning rate, $\boldsymbol{m}$, $\boldsymbol{\theta}$ and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ represent updates, weights and the gradient of the loss function during training.

Further speeding up convergence

Faster optimizers essentially estimate and correct the gradient vector towards a local minimum of the loss function.

- Nesterov Accelerated Gradient (NAG).

$$
\begin{aligned}
\boldsymbol{m} &\leftarrow \alpha\boldsymbol{m} - \mu\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \alpha\boldsymbol{m}), \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \boldsymbol{m}.
\end{aligned}
$$

- RMSProp.

$$
\begin{aligned}
\boldsymbol{s} &\leftarrow \rho\boldsymbol{s} + (1-\rho)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} - \mu\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\boldsymbol{s} + \epsilon}.
\end{aligned}
$$

where $\otimes$ and $\oslash$ are element-wise multiplication and division, respectively, $\rho = 0.9$, $\epsilon = 10^{-10}$, $\boldsymbol{m}$ and $\boldsymbol{s}$ initialized with zeros.

Further speeding up convergence

- Adam, Nadam, and AdaMax.

$$
\begin{aligned}
\boldsymbol{m} &\leftarrow \beta_1\boldsymbol{m} - (1-\beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}),\\
\boldsymbol{s} &\leftarrow \beta_2\boldsymbol{s} + (1-\beta_2)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}),\\
\hat{\boldsymbol{m}} &\leftarrow \frac{\boldsymbol{m}}{1-\beta_1^t},\\
\hat{\boldsymbol{s}} &\leftarrow \frac{\boldsymbol{s}}{1-\beta_2^t},\\
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \mu\hat{\boldsymbol{m}} \oslash \sqrt{\hat{\boldsymbol{s}} + \epsilon}.
\end{aligned}
$$

where $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$, $t \geq 1$ is the iteration, first $\boldsymbol{m}$ and second $\boldsymbol{s}$ momenta are initialized with zeros. Nadam is Adam with the Nesterov trick $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \beta_1\boldsymbol{m})$ and AdaMax replaces the second equation by $\boldsymbol{s} \leftarrow \max\{\beta_2\boldsymbol{s}, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})\}$.

Scheduling the learning rate and avoiding under/overfitting

- The learning rate μ (e.g., 10^{-3}) is the most important parameter to avoid underfitting and overfitting.

Scheduling the learning rate and avoiding under/overfitting

- The learning rate μ (e.g., 10^{-3}) is the most important parameter to avoid underfitting and overfitting.

- Strategies to change the learning rate during training are called learning schedules – e.g., $\mu^{(t)} = \mu^{(0)}0.1^{t//s}$ drops it by a factor of 10 at every s iterations.

Scheduling the learning rate and avoiding under/overfitting

- The learning rate μ (e.g., 10^{-3}) is the most important parameter to avoid underfitting and overfitting.

- Strategies to change the learning rate during training are called learning schedules – e.g., $\mu^{(t)} = \mu^{(0)}0.1^{t//s}$ drops it by a factor of 10 at every s iterations.

- By starting from a very low learning rate $\mu^{(0)}$ (e.g., low accuracy on $\mathcal{Z}_{tr}$ - underfitting) and increasing it at every s iterations until $\mu^{(t)}$, when we observe high accuracy on $\mathcal{Z}_{tr}$ and low accuracy on $\mathcal{Z}_{vl}$ (overfitting), we may estimate the best learning rate as $\mu^{(t-\delta)}$ (e.g., $\delta = 1$).

Scheduling the learning rate and avoiding under/overfitting

Recall our loss $J = \sum_{s \in \mathcal{B} \subset \mathcal{Z}_{tr}} \mathcal{E}(s)$ per batch. Overfitting can be avoided by regularization techniques.

- L1 and L2.

$$\begin{aligned}
\mathcal{E}(s) &= -\sum_{j=1}^{N_L} y_j(s) \log(y_j^L(s)) + 0.01 \sum_{r=1}^{L} \sum_{j=1}^{N_r} \|\mathbf{w}_j^r\|, \\
\mathcal{E}(s) &= -\sum_{j=1}^{N_L} y_j(s) \log(y_j^L(s)) + 0.0001 \sum_{r=1}^{L} \sum_{j=1}^{N_r} \|\mathbf{w}_j^r\|^2.
\end{aligned}$$

- Max-Norm, which rescales by $s \in (0, 1]$ (lower s, more regularization) the weights after each update – i.e., $\mathbf{w}_j^r \leftarrow s \frac{\mathbf{w}_j^r}{\|\mathbf{w}_j^r\|}$.
- Dropout, which before any layer $r \in [1, L]$ sets its weights $\mathbf{w}_j^r \leftarrow 0$ with probability $p \in [0.1, 0.5]$ at every training step.

Training and evaluating NNs in practice

To fix all we have learned, let's project and evaluate our first neural network with images. ▸ FirstDeepNeuralNetwork.

- We will understand the role of the validation set – it allows you to avoid underfitting and overfitting.
- The next step is an introduction to image analysis, so we can understand the success of DNNs in this field.
- We will then see that convolutional layers play a crucial role to make feasible the training of DNN models for image analysis.

[1] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter.
Neural architecture search: a survey.
J. Mach. Learn. Res., 20(1):1997–2017, January 2019.

[2] Jesús L. Llano García, Raúl Monroy, and Víctor Adrián Sosa Hernández.
Neural architecture search for image super-resolution: A review on the emerging state-of-the-art.
Neurocomputing, 610:128481, 2024.

[3] R. Interdonato, M. Magnani, D. Perna, A. Tagarelli, and D. Vega.
Multilayer network simplification: Approaches, models and methods.
Computer Science Review, 36:100246, 2020.

[4] D. Osaku, J.F. Gomes, and A.X. Falcão.
Convolutional neural network simplification with progressive retraining.
Pattern Recognition Letters, 150:235–241, 2021.

[5] Amir Moslemi, Anna Briskina, Zubeka Dang, and Jason Li.
A survey on knowledge distillation: Recent advancements.
Machine Learning with Applications, 18:100605, 2024.

[6] Bárbara Caroline Benato, Alexandru Cristian Telea, and Alexandre Xavier Falcão.
Semi-supervised learning with interactive label propagation guided by feature space projections.
In *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 392–399, 2018.

[7] Bárbara C. Benato, Jancarlo F. Gomes, Alexandru C. Telea, and Alexandre X. Falcão.
Semi-automatic data annotation guided by feature space projection.
Pattern Recognition, 109:107612, 2021.

[8] Bárbara C. Benato, Alexandru C. Telea, and Alexandre X. Falcão.
Deep feature annotation by iterative meta-pseudo-labeling on 2d projections.
Pattern Recognition, 141:109649, 2023.

[9] Bárbara C. Benato, Cristian Grosu, Alexandre X. Falcão, and Alexandru C. Telea.

Human-in-the-loop: Using classifier decision boundary maps to improve pseudo labels.

Computers & Graphics, 124:104062, 2024.

[10] R. Garcia, A.X. Falcão, A.C. Telea, B.C. da Silva, J. Tørresen, and J.L. Comba.

A methodology for neural network architectural tuning using activation occurrence maps.

In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2019.

[11] I.E. de Souza and A.X. Falcão.

Learning cnn filters from user-drawn image markers for coconut-tree image classification.

IEEE Geoscience and Remote Sensing Letters, pages 1–5, 2020.

[12] David Aparco-Cardenas, Jancarlo F. Gomes, Alexandre X. Falcão, and Pedro J. de Rezende.

Consensus-based iterative meta-pseudo-labeling for deep semi-supervised learning.

Information Sciences, 692:121671, 2025.