Neural Networks
Lecture Notes
Deep learning

Learning Goals

After studying the lecture material you should:

1. be able to motivate deep learning from a biological and theoretical point of view
2. know why supervised deep learning is hard (underfitting versus overfitting)
3. be able to provide solutions for underfitting and overfitting
4. be able to give an example of data augmentation
5. know the mathematical form of 2-D convolution
6. know what is meant by pooling and why it is useful
7. know what is meant by residual learning
8. know what batch normalization is used for
9. be able to reproduce the mathematical form for the ReLU and explain why it is beneficial
10. be able to explain how dropout works
11. understand what is meant by deconvolution

Notes

We now consider deep convolutional neural networks (DNNs) that are trained in a supervised manner. I.e. we deal with feedforward neural networks consisting of artificial neurons that are organized in terms of multiple hidden layers. Each layer of the DNN incorporates particular filtering steps. The goal is again to learn a mapping between an input (e.g. image) and an output (e.g. object categories) by adjusting synaptic weights between artificial neurons. Interestingly, representations in hidden layers capture increasingly complex characteristics of their input. These networks have a biological motivation since the brain's visual system can also be seen as a hierarchical feed-forward model using feature maps for representations. DNNs have a long history, such as Selridge's pandemonium, Fukushima's neocognitron and Riesenhuber and Poggio's HMAX model. Convolutional neural networks were first introduced by LeCun. From a theoretical point of view, DNNs can be shown to compute particular functions using much fewer nodes/weights than one would need using more shallow networks. Recall though that in principle, networks with one hidden layer can approximate any well-behaved function.

Training of deep neural networks is hard for two reasons. On the one hand, they can suffer from *underfitting*. That is, optimization becomes much harder due to vanishing gradient problem.

On the other hand, DNNs suffer from *overfitting*. Deep nets usually have lots of parameters that need to be estimated from finite data. This can results in a high variance / low bias situation in which the flexibility of DNNs completely overfits on the training data. The *curse of dimensionality* is a related issue, where the size of a dataset required to learn a mapping grows exponentially with the size of the the dataset's parameter space (the one that needs to be covered by the training data). Underfitting can be resolved by using better/more efficient optimization techniques and training for a long time using GPU architectures, or by the use of activation functions with more desirable properties. Overfitting can be resolved by using much larger datasets (applying techniques such as data augmentation), reducing the size of the input (dimensionality reduction), reducing the number of free parameters (weight sharing), or enforcing good generalisation (dropout training).

Important ingredients

We now list a number of ingredients that are used to make training of DNNs possible.

Data augmentation

Data augmentation reduces overfitting on image data by artificially enlarging the dataset using label-preserving transformations.

1. Generate image translations and horizontal reflections.
2. Alter the intensities of the RGB channels in training images.
3. Altering object backgrounds if they are no part of the object identity.

This scheme approximately captures an important property of natural images, namely, that object identity is invariant to changes in the intensity and color of the illumination. Without this scheme, the DNN suffers from substantial overfitting, which would force us to use much smaller networks.

Centering

Centering of the data is also important. This boils down to subtracting the mean image from all images before presenting them to the network. Basically, it's only deviations from the mean that then need to be captured.

Initialization

If the weights in a network start too small, then the signal shrinks as it passes through each layer until it's too tiny to be useful. If the weights in a network start too large, then the signal grows as it passes through each layer until it's too massive to be useful. An often used solution is Xavier initialization. I.e., draw weights coming into a node from a normal distribution with zero mean and variance

$$\sigma^2 = \frac{2}{n_{in} + n_{out}}$$

where n_{in} is the number of inputs and n_{out} is the number of outputs to a node.

Rectified linear units

Sigmoidal activation functions such as the logistic function and the tanh function can suffer from the vanishing gradient problem [BSF94]. Vanishing gradients occur when lower layers of a DNN have gradients of nearly 0 because higher layer units are nearly saturated at extreme values. Vanishing gradients cause slow optimization convergence, and in some cases the final trained

network converges to a poor local minimum. Rectified linear units (ReLU, [GBB11]) use the following activation function:

$$f(x) = \max(0, x)$$

They do not show the vanishing gradient problem and as a result they show better convergence. Various modifications have been developed: Leaky ReLUs, Randomized Leaky ReLUs, Parametric ReLUs, Exponential LUs.

Convolution

A subset of the hidden layers implement convolution. Two-dimensional convolution of a signal f with an $M \times M$ filter g is given by:

$$(f * g)[x, y] = \sum_{u=-M}^{M} \sum_{v=-M}^{M} f([x - u, y - v]g[u, v])$$

In such a layer, depth determines the number of stimulus features in a layer whereas height and width determine the size of a *feature map*. The feature map represents a particular feature relatively to different parts of the input space. Convolution allows for features to be detected regardless of their position in the visual field. Massive weight sharing greatly reduces the number of free parameters. They give much better generalization performance in e.g. vision problems and consecutive layers have increasingly global receptive fields.

Pooling

A subset of the hidden layers implement pooling. Each feature map is subsampled by taking the mean (mean pooling) or maximum value (max pooling) over $p \times p$ contiguous regions. This leads to translation invariance (tolerance to small displacements in the input).

Residual learning

Training very deep architectures does not necessarily give better performance. They should be able to perform as well as shallow networks. I.e. the deeper layers can just implement an identity mapping. Residual networks implement this idea by adding the identity mapping to the output [HZRS15].

Batch normalization

Minibatch learning can change the input statistics during each training iteration. This is known as covariate shift. Can be circumvented by normalizing the input to each layer in each iteration. Batch normalization has proven to be very effective [IS15].

Dropout

With *Dropout* ([HSK+12], [BF13]), every time an input is presented, the neural network samples a different architecture. It forces the artificial neurons to become more robust feature detectors. At test time, we use all the neurons but multiply their outputs by 0.5. This substantially reduces overfitting.

Dense layers

After the convolutional layers there may be any number of fully connected layers. The densely connected layers are identical to the layers in a standard multilayer neural network.

Softmax outputs

The final layer contains (e.g.) the image categories. The output for the j-th category is given by a softmax function of the form:

$$\sigma(\mathbf{a}) = \frac{\exp(a_j)}{\sum_k \exp(a_k)}$$

where $\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}$ are the input activations of all output units.

Adaptive gradient descent

The learning rate is an important parameter to ensure convergence. We should start with high learning rates and then gradually reduce them This can be achieved by globally annealing the learning rate. Adaptive learning rate for each parameter works much better. To this end various approaches have been developed:[1]

- Adagrad
- RProp
- RMSProp
- Adam

Adam is quite a popular default choice [KB14].

Looking at internal representations

An important question for understanding what these neural networks learn is how can we infer what stimulus feature an internal unit is sensitive to. One example is to perform gradient descent in image space to maximize a unit's activation. Another example is *deconvolution* (approximately invert the operations of a convolutional neural network).

In deconvolution [ZF12], we pass each image through the convolutional neural network, fix the feature map of interest and set the rest to zero, and finally propagate backwards through the layers using the deconvolutional neural network. We can then show the reconstructions of the features and the corresponding images that gave largest activations of that unit.

Examples

There are various examples of the use of neural networks next to its use in standard object recognition. To name a few:

- Semantic segmentation [FCNL12]
- Mimicking artistic style [GEB15]
- Convolutional sketch inversion [GGvLvG16]
- Personality trait prediction [GGvGvL16]
- Deep dreaming: http://deepdreamgenerator.com/

There are also so-called *adversarial examples*, which show how a neural network can fail using slight changes in the input [SZS+13].

Reading material

An overview of deep learning is provided in [LBH15].

[1] See http://cs231n.github.io/neural-networks-3/ for all kinds of other tricks.

References

[BF13] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Adv. Neural Inf. Process. Syst.*, pages 3084–3092, 2013.

[BSF94] Y Bengio, P Simard, and P Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Trans. Neural Networks*, 5(2):157–166, 1994.

[FCNL12] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers. *arXiv Prepr. arXiv ...*, page 9, 2012.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier networks. In *Proc. 14th Int. Conf. Artif. Intell. Stat. JMLR W&CP Vol.*, volume 15, pages 315–323, 2011.

[GEB15] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A Neural Algorithm of Artistic Style. *arXiv Prepr.*, pages 3–7, 2015.

[GGvGvL16] Yamur Güçlütürk, Umut Güçlü, Marcel A. J. van Gerven, and Rob van Lier. Deep Impression: Audiovisual deep residual networks for multimodal apparent personality trait recognition. In *ECCV*, 2016.

[GGvLvG16] Yamur Güçlütürk, Umut Güçlü, Rob van Lier, and Marcel A. J. van Gerven. Convolutional Sketch Inversion. pages 1–15, 2016.

[HSK+12] G E Hinton, N Srivastava, A Krizhevsky, I Sutskever, and R R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv Prepr. arXiv1207.0580*, 2012.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *Arxiv.Org*, 7(3):171–180, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, pages 1–11, 2015.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *Int. Conf. Learn. Represent.*, pages 1–13, 2014.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv Prepr. arXiv ...*, pages 1–10, 2013.

[ZF12] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *arXiv Prepr. arXiv1311.2901*, 2012.