6 DeepLearning

Our treatment of neural networks has been basically the state of the art for at least 20 or 30 years, though this area is now receiving considerable attention under the name of deep learning. Quite simply, deep learning refers to neural networks with many layers, which will of course lead to more complex models and hence it is likely that they can outperform simpler models for complex tasks. While this was already clear in the 1990s, we have only been able to train such networks in more complex tasks in the last few years. This advancement is due to several factors, but specifically though the availability of large supervised datasets, the enormous increase of computational power in particular though GPUs, and some more sophisticated applications of regularization techniques. We will briefly outline some of these issues here while basically following some of the tutorials in Tensorflow.

6.1 Convolution

Most of the success in deep neural networks have been demonstrated with image processing and involve convolutional neural networks, and this this section we will just review basic convolutions. A convolution in one dimension is defined as

$$f * g = \int_{-\infty}^{\infty} g(t')f(t-t')dt' \tag{6.1}$$

6.2 CNN

The idea of using convolutions in neural network have been first used by Fukushima in 1980. Fukushima worked at this time for NHK (the Japanese public broadcaster) together with physiologists as NHK was interested to understand the mechanisms of human vision. It was well known since the early 1960s form the experiments by Hubel and Wiesel that some neurons in the primary visual cortex (the first stage of visual processing in the cortex) are edge detectors.

Edge detectors are also the workhorse of computer vision, and we discussed in the first section how such filters are implemented with convolutions. The neural networks that we discussed before had to learn individual weights to each pixel location. Even if this network would learn to represent an edge detector, such detectors have to be learned for each location in an image since edges could usually appear in all locations. So another way of thinking about convolution is that a neuron (specific filter) is applied to every possible location in the image. This leads us to a **convolutional neural network (CNN)**.

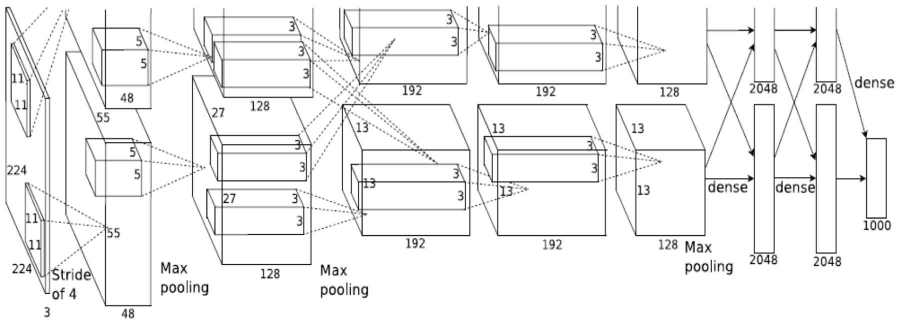

Fig. 6.1 A famous implementation (so called AlexNet) of a convolutional neural network for image classification Representation (Krizhevsky, Sutskever, Hinton, 2012).

A famous example called AlexNet is shown that was used to classify a big database of images from many different classes is shown in Fig. 6.1. This network takes three dimensional images (e.g. RGB values of pixels) and applies a layer of filters onto it. There are actually several layers of filters (so called filter banks) applied to the input image. The network consists actually os several layers of such convolutional computations. Also, to help with the computational coast of the operations we are sometimes not just shifting the filter by one pixel but might shift it by s pixels. This is called a **stride**.

If we would only apply convolutions on convolutions, then we would end with filtered images of filtered images. However, what we really hope to achieve are high level representation such as nodes that represent class labels. Such labels are likely not to depend on individual pixels and rather represent a highly compressed summary of an image. To help with this we add **pooling layers** after each convolution layer. A pooling operation is usually just taking the average or the maximum of the responses in a certain area of the filtered image, thus compressing the images down by only considering the average or maximal feature represented by the filter in this area. At the end we use a regular network, now often called a **fully connected layer**, to gather all the information and make the final classification based on the features extracted by the network.

While we have just outlined the operation of this network, an important part of using this network is of course the training. Indeed, for our further discussion it is good to realize that the filters are not chosen by hand but are learned from examples. This training was trained with the back propagation algorithm. This is fairly straight forward except when back propagating though the pooling layers. One approach is thereby to give all credit for the error (average pooling), or juts change the winning unit (max pooling).

6.3 Tensorflow

To implement deep networks we use the Google's Tensorflow package. Please make sure it is implemented. Please follow the first tutorials

`https://www.tensorflow.org/versions/r0.11/tutorials/mnist`
followed by
`/beginners/index.html#mnist-for-ml-beginners`
for the beginner tutorial, and by
`/pros/index.html#deep-mnist-for-experts` for the expert one.

6.4 Representational learning and compression

Above we pointed out that we used back-propagation (gradient descent learning) to learn all the filters in the machines and hence the specific representations in each of the layers of the deep network. This is a great advantage over more traditional system in which filters had to be designed carefully by hand, which is not only time consuming, but also might be less optimal than learned filters. This evolution in our field is outlined in Fig. 6.2.

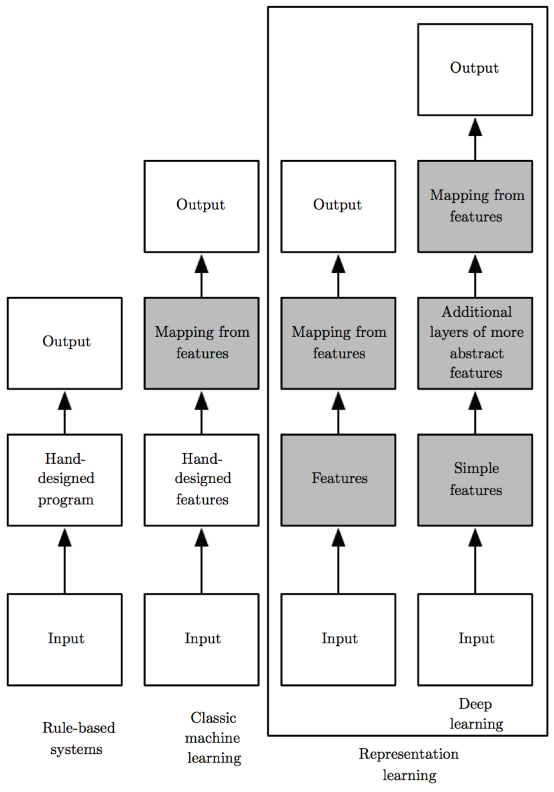

Fig. 6.2 Evolution of machine learning system (from Goodfellow, Bengio, Courville 2015).

To illustrate again the re-reprentation of a signal with filters, consider basic signal analysis. We have time varying signal that is represented with floating point values for each time step. Say we are sampling with 500HZ, typical for EEG, that is, we have one data point every 0.002 second. If we assume that a floating point is typically represented by computer word of 64bits, then a 10 minute length would take over 2MB of storage.

An example signal,$x(t)$ is illustrate in Fig. 6.3. While this is a relative complicated signal, we have also created two template signals, which we could also call a **basis function**, with which we can reconstruct the signal as

$$x(t) = a_1 y_1(t) + a_2 y_2(t) \tag{6.2}$$

Representing the original signal with these basis functions has a big advantage. For example, if we all know about the basis functions, than I could transmit the signal with only two computer words for the coefficients $a_1 = 2$ and $a_2 = 3$ (arbitrarily chosen numbers), which takes 16Bytes. Of course, we can not expect real signals to be made up of only these two basis functions. So in practice we want to create a large list of "appropriate" basis functions. The filters in deep neural networks represent such basis functions, and the appropriateness should come from the fact that they are learned from the examples.

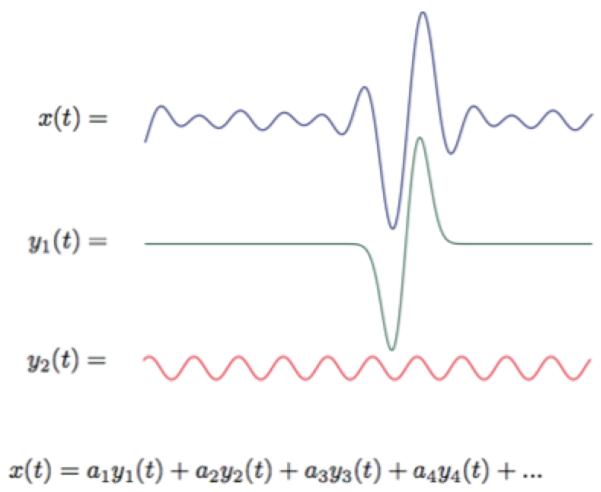
$x(t) =$
$y_1(t) =$
$y_2(t) =$
$x(t) = a_1y_1(t) + a_2y_2(t) + a_3y_3(t) + a_4y_4(t) + ...$

Fig. 6.3 Illustration of signal representation with templates.

Now if we have a large list of basis functions, then we still might need to submit a large number of coefficients. Indeed, if we make the basis function a value at each time step, then we would just end up with the same representation as before. However, a very important insight to make efficient use of resources while extracting the "essence of signals" is to try and find sparse representations. A **sparse representation** is a representation where I might have a large number of basis functions in my dictionary (nodes in a deep network) but are only using a relatively small number of active nodes to represent each example. In our example this might correspond to

$$x(t) = a_1 y_1(t) + a_2 y_2(t) + a_3 y_3(t) + a_4 y_4(t) + a_5 y_5(t) + a_6 y_6(t) + ... \tag{6.3}$$

with a coefficient vector

$$\mathbf{a} = (2, 3, 0, 0, 0, 0, 0, ...). \tag{6.4}$$

Sparse representations do lead to considerable compressions, and I believe that compression like $\mathbf{a} = (2, 3)$, and particular sparse compression, is an essential ingredients of machines to gain some "semantic knowledge" of the world. For example, there are many instantiations of a tables, but my ability to characterize them with one word is equivalent to semantic knowledge and a form of sparse representation if we take as our communication dictionary a large list of names for furnitures.

6.5 Denoising autoencoders

If compressed (or sparse) representations are so useful, can we force such representations in our deep networks? There are different techniques that will indeed force some compressed representations including the architecture outlined here as well as more general regularization methods discussed in the next section.

A simple example of an **autoencoder** is shown in Fig. 6.4. In this network we start with an input layer that is connected to a smaller hidden layer and then to an output layer that is the same size as the input layer. The reason for choosing an output layer that has the same size as the input layer is that we want to build a mapping function that maps inputs to the same output. This is actually an example of **unsupervised learning** as we do not require labels just raw data such as pictures.

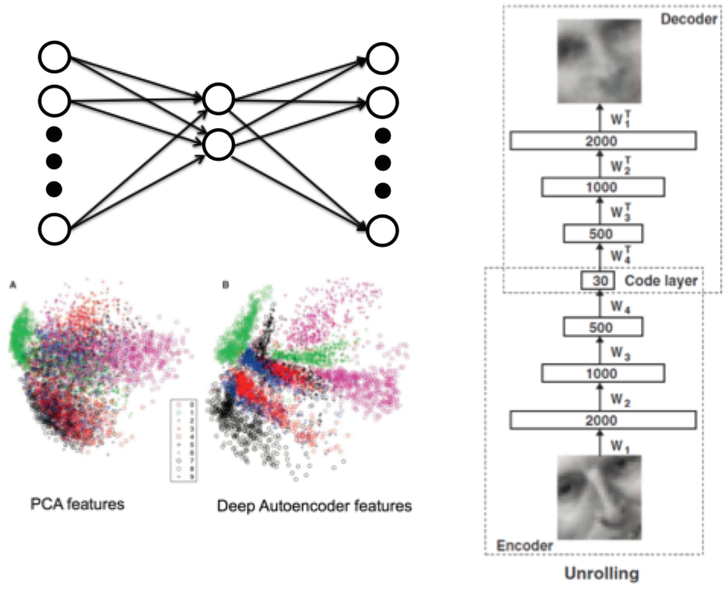

Fig. 6.4 Examples of autoencoders with the basic idea on top, the deep autoecncoder that kicked of deep learning by Hinton and Salakhutdinov 2006, and some comparisons of auto encoders to PCA.

Why would we want to build such functions and isn't this simply the identity function? It is not as we channel the input through a small layer that forces some compression. In this sense we try to extract useful filters, and some people have described this as **semantic hashing**. Most of the implementation use a version were the inputs are somewhat corrupted by noise and the labels are the noiseless pattern, which are called **denoisong autoencoders**. This will also help to force the solution away from the identity function and is an example of noise induction as a regularization technique as discussed further below.

6.6 Regularization

We have discussed before the bias-variance tradeoff in modelling. That is, if the model complexity is not sufficiently high to describe the degrees of freedom of the system that we want to model, than we can not expect good results. Indeed, we expect some form of bias or underfitting in our function fits. On the other hand, if the complexity of out model is too high, the we expect overfitting or a large variance of the accuracy of predictions. Deep learned represent a form of machine learning in which we build very large models with the hope that they will to some extend include the target model or at least a sufficient approximation. However, preventing these models form overfitting has therefor been a major challenge and the techniques discussed here have been a crucial part in making deep learning work.

For the most part we will loosely equate the number of parameters with the complexity of the model. Of course, the specific architecture such as a hierarchical versus flat representations is of course also part of the complexity equation, we think here more about a given architecture and ask how we can sufficiently constrain the parameters. One solution to the problem is to use lots of data compared to the number of parameters that will constrain the parameters sufficiently. Indeed, large data collections such as ImageNet have therefore been essential in demonstrating the abilities of deep networks.

However, we are nearly always in the situation that we do not have enough training data, and **data augmentation** is a common an essential technique even in the case of the ImageNet competition. Images are actually a good example to see where certain transformation are good candidates for good data augmentation. Shifting and image should not alter the content, and rotation and some form of stretching does also not alter essential features for classifying the content of the figure. Indeed, even changing individual pixels in a high resolution image do not have a drastic impact on recognition abilities. So generating more training images with these transformation from the labeled training set is a good way to increase the training data set. Injecting noise either at an input level or at an output level is another example of such data transformations as mentioned above for denoting autoencoders. The data augmentation technique is actually a good example where we use some expert knowledge augment the training data set. But we should also stress a warning, such transformation are not necessarily good in every situation. Indeed, to some extend we must already know the data distribution to generate proper examples, but this is usually what we are trying to do with machine learning. So data augmentation must clearly be seen as inducing some expert knowledge to restrict the space of all possible data transformations to a subset that are consistent with the world model.

Bagging (bootstrap aggregating) is a technique that uses the average of several trained models to prevent overfitting. Such models are often discussed under the topic of ensemble machines. A random forrest is a popular example of an ensemble method with decision trees. Decision trees are prone to overfitting and combining them with model averaging is therefor essential for good performance.

A common cause for overfitting in neural networks is that with a lot of neurone relative to the training examples we could dedicate individual neurone (grandmother cells) or a small number of them to the memory of individual training points. **early stopping** of the training procedure was hence a common techniques in the 1990s. A more recent technique is called **dropout** in which individual neurones will for some random invocations not contribute to the output. Hence, this forces that other neurone must be able to contribute to the explanation of a data point which should prevent grandmother cells.

Finally, a more systematic way to approach regularization is though priors and parameter constraints. In a Bayesian sense we are looking for a MAP estimate of the parameters which equates to a likelihood with a prior,

$$p(\mathbf{w}|\mathbf{x}, y) = p(\mathbf{x}, y|\mathbf{w})p(\mathbf{w}) \tag{6.5}$$

It is possible to show that for a Gaussian prior that is equivalent to minimizing a the regular loss function with a quadratic constraint on the parameters

$$\tilde{L}(\mathbf{w}, \mathbf{x}, y) = L(\mathbf{w}, \mathbf{x}, y) - \gamma||\mathbf{w}||^2, \tag{6.6}$$

where $L(\mathbf{w}, \mathbf{x}, y)$ is our regular loss function such as the cross entropy or MSE. We included here a hyper-parameter γ which allows us to vary how strongly we should take this constraint into account. The gradient descent of this loss function is

$$\Delta\mathbf{w} = -\alpha\frac{\mathrm{dL}(\mathbf{w}, \mathrm{x}, \mathrm{y})}{\mathrm{d}\mathbf{w}} - 2\gamma\mathbf{w}. \tag{6.7}$$

which we can also write in the form

$$\mathbf{w} \leftarrow (1 - 2\alpha\gamma)\mathbf{w} - \alpha\frac{\mathrm{dL}(\mathbf{w}, \mathrm{x}, \mathrm{y})}{\mathrm{d}\mathbf{w}}. \tag{6.8}$$

This corresponds to an exponential decay of the weights when the gradient is zero. Or in other words, this puts pressure on the weights to decay unless they are reinforced by the gradient, and it is therefore often called **weight decay**. The specific case of this quadratic penalty term (also called L^2 norm) is also called **ridge regression** or **Tikhonov regularization**.

Weight decay can be formulated in a more general way with other functions of the weights that relate to a prior. For example, a common choice is the L^1 norm,

$$\tilde{L}(\mathbf{w}, \mathbf{x}, y) = L(\mathbf{w}, \mathbf{x}, y) - \gamma||\mathbf{w}||, \tag{6.9}$$

which is related to a technique called **LASSO** (least absolute shrinkage and selection operator). This form of regularization is related to a prior is an isotropic Laplace distribution and leads to a constant weight decay,

$$\Delta \mathbf{w} = -2\gamma \mathrm{sign}(\mathbf{w}) - \alpha \frac{\mathrm{dL}(\mathbf{w}, \mathbf{x}, \mathrm{y})}{\mathrm{d}\mathbf{w}}. \tag{6.10}$$

There is some argument that LASSO leads to a more sparse representation than ridge regression, and we have argued that this regularization (with slight modifications) is very good in situations where a few relevant features are embedded in a large and noise feature vector with irrelevant features.

There are a many other techniques with the effect of guiding the search in a specific subspace of the parameter space. Unsupervised pre-training or semi-supervised methods can be placed in this category, but even adversarial training or a good discussion can be found in the recent MIT book *Deep Learning* by Ian Goodfellow, Yoshua Bengio and Aaron Courville.