A gentle introduction into Deep Learning: concepts and implementation

Christian Wolf
INRIA-Chroma, CITI, LIRIS, INSA-Lyon
Gestures

Activities

Articulated pose

Semantic segmentation
Learning to predict

We would like to predict a value $t$ from an observed input

$$y = h(x, \theta)$$

Parameters $\theta$ are learned from training data.
The standard toolbox

Deep neural networks

Convolutions introduce an inductive bias for imaging/vision applications.

Recurrent networks allow to model sequences.

[LeCun et al., 1998]  [Hocheiter and Schmidhuber, 1997]
Deep neural networks

\[ x \xrightarrow{w^{(1)}} \cdots \xrightarrow{w^{(2)}} y \xrightarrow{\mathcal{L}} y^* \]

Input layer \quad Hidden layer \quad Output layer
Learning by gradient descent

Iterative minimisation through gradient descent:

\[ \theta^{[t+1]} = \theta^{[t]} + \nu \nabla \mathcal{L}(h(x, \theta), y^*) \]

Learning rate

Can be blocked in a local minimum (not that it matters much …)

[Figure: C. Bishop, 2006]
The standard toolbox

Deep neural networks

Convolutions introduce an inductive bias for imaging/vision applications.

Recurrent networks allow to model sequences

Input  Hidden  Output

[LeCun et al., 1998]  [Hocheiter and Schmidhuber, 1997]
« LeNet »

[LeCun et al., 1998]
A linear and shift invariant operator is equivalent to a convolution with the impulse response of the operator!

\[
[\phi(f)](x, y) = \phi \left( \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} f(m, n) \phi^{(m,n)}(p) \right) (x, y)
\]

\[
= \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} f(m, n) \phi^{(m,n)} S(\phi^{(0,0)}) (x, y)
\]

\[
= \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} f(m, n) \phi^{(m,n)} S(h) (x, y)
\]

\[
= \sum_{m=-M/2}^{M/2} \sum_{n=-N/2}^{N/2} f(m, n) h(x - m, y - n)
\]

\[
= \sum_{m'=-M/2}^{M/2} \sum_{n'=-N/2}^{N/2} f(x - m', y - n') h(m', n')
\]

\[
(m' = x - m \ n' = y - n)
\]

[Jähne, 1997]
The standard toolbox

Deep neural networks

Convolutions introduce an inductive bias for imaging/vision applications.

Recurrent networks allow to model sequences

[LeCun et al., 1998]  [Hocheiter and Schmidhuber, 1997]
Hidden Markov Models

You might be familiar with Hidden Markov Models (HMMs):

- Dependency graph
- Transition graphs
Recurrent neural networks (RNNs)

Prediction: feed-forward computation in a DAG. No optimization is needed.
Graphical models vs. Neural networks

Graphical models:
- state is stochastic
- make it easier for experienced practitioners to model known relationships between data.
- optimization required for prediction
- specific structures allow to obtain global optima with message passing (chains and trees) or graph cuts (submodular potentials) etc.

Neural networks:
- state is deterministic
- complex models with componential hidden states
- higher order interactions are easier to handle
- no optimization during prediction

Is it better to get a global min/max of a simple model or a feed-forward prediction for a high-capacity model trained on a large amount of data?
From RNNs to LSTMs

Objectives:
- handle long and short term transitions
- Deal with vanishing/exploding gradients

Figures: Chris Olah

RNN

LSTM

[Hocheiter and Schmidhuber, 1997]
CNNs vs. RNNs … wait a minute #1

RNNs are not restricted to 1D and convolutions not to 2D. We can create recurrent connections on a 2D grid over images.

[Moysset, Kermorvant, Wolf, (under review)]
CNNs vs. RNNs … wait a minute #2

We can perform convolutions on temporal data. The different convolutional layers correspond to features extracted over different temporal distances.

[Baradel, Wolf, Mille, arxiv 2017]
Semantic Segmentation

Residual Conv-Deconv Grid Network for Semantic Segmentation

Damien Fourure, Rémi Emonet, Elisa Fromont, Damien Muselet, Alain Trémeau & Christian Wolf

Fourure, Emonet, Fromont, Muselet, Tremeau, Wolf, BMVC 2017
Resolution preserving prediction

Reducing spatial resolution assures that the receptive fields of fields increase in higher layers (gain in abstraction).

Challenge in cases where decisions need to be made in high resolution (segmentation, super-resolution etc.)

[Fourure et al., BMVC 2017]
Layers, scales and spatial resolutions

Conv-Deconv networks: downsampling followed by upsampling to regain the lost resolution

[Noah et al., 2015]
Grid Networks

[Fourure et al., BMVC 2017]
Functional mappings & Tensor flows

Inputs, outputs and parameters of functional mappings are usually tensors of multiple dimensions.

Example for input tensors in image processing: 4D tensors [index-in-batch, x-spatial, y-spatial, color channel]
We can define a symbolic (!) computation graph:

```python
1 # The input tensors
2 x = tf.placeholder(tf.float32, [None, 224, 224, 1])
3 # The target labels
4 y_gt = tf.placeholder(tf.float32, [None, NO_CLASSES])
5 # Layer 1: convolutional
6 W_conv1 = weight_variable([5, 5, 1, 32])
7 b_conv1 = bias_variable([32])
8 h_conv1 = tf.nn.relu(tf.nn.conv2d(x, W_conv1, strides=[1, 1, 1, 1], padding='SAME') + b_conv1)
9 h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
10 # Layer 2 (output): fully connected
11 W_fc2 = weight_variable([NO_CLASSES, NO_CLASSES])
12 b_fc2 = bias_variable([NO_CLASSES])
13 y = tf.matmul(h_pool1, W_fc2) + b_fc2
14 # The loss function
15 loss = tf.reduce_sum(tf.square(y - y_gt))
```

We can calculate derivatives of tensors with respect to other tensors:

```python
22 # Define gradients of loss with respect to some weights
23 grads_input = tf.gradients(loss, W_conv1)[0]
24 # Alternatively, define gradients of loss with respect to the input
26 grads_input = tf.gradients(loss, x)[0]
```
PyTorch (Facebook) : example

Calculation is imperative, not symbolic. All calculations are carried out immediately. Computation traces are stored to be able to backpropagate later. Easy debugging: no inference engine needed to show values.

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, 5, padding=2)
        self.conv2 = nn.Conv2d(64, 64, 5, padding=2)
        self.conv3 = nn.Conv2d(64, 64, 5, padding=2)
        self.conv4 = nn.Conv2d(64, 64, 5, padding=2)
        self.conv5 = nn.Conv2d(64, 64, 5, padding=2)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(7*7*64, 1024)
        self.d = nn.Dropout(p=0.5)
        self.fc2 = nn.Linear(1024, 10)

    def forward(self, x):
        # Input images are of size 224x224x1
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = self.pool(F.relu(self.conv5(x)))
        # Feature maps are now of size 7x7x64
        x = x.view(-1, 7*7*64)
        x = F.relu(self.fc1(x))
        x = self.d(x)
        x = F.relu(self.fc2(x))
        return x

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 64, 5, padding=2)
        self.conv2 = nn.Conv2d(64, 64, 5, padding=2)
        self.conv3 = nn.Conv2d(64, 64, 5, padding=2)
        self.conv4 = nn.Conv2d(64, 64, 5, padding=2)
        self.conv5 = nn.Conv2d(64, 64, 5, padding=2)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(7*7*64, 1024)
        self.d = nn.Dropout(p=0.5)
        self.fc2 = nn.Linear(1024, 10)

    def forward(self, x):
        # Input images are of size 224x224x1
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = self.pool(F.relu(self.conv5(x)))
        # Feature maps are now of size 7x7x64
        x = x.view(-1, 7*7*64)
        x = F.relu(self.fc1(x))
        x = self.d(x)
        x = F.relu(self.fc2(x))
        return x

output = net.forward(inp)
loss = crossentropy(output, lab_gtv)
loss.backward()
optimizer.step()
May the loss go down
Some gentle words on learning theory

\[ \mathcal{L}_{\mathcal{D}, f}(h) \overset{\text{def}}{=} \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f(x)] \]

Expected error over (unknown) data distribution \( \mathcal{D} \) and (groundtruth) labeling function \( f \)

\[ \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f(x)] \]

model

Labeling function (ground truth)

[Shai Shalev-Shwartz et al. 2014]
Empirical Risk Minimization

\[ L_S(h) \overset{\text{def}}{=} \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m}, \]

Empirical error over Samples \( y_i \) from training set \( S \)

How different is the empirical error from the expected error?

[Shai Shalev-Shwartz et al. 2014]
Sources of error

\[ L_{\mathcal{D},f}(h) \overset{\text{def}}{=} \mathbb{P}_{x \sim \mathcal{D}}[h(x) \neq f(x)] \]

1. Lack of generalization (shift between empirical error and expected error on the target domain)

2. Optimization problem (the solution of the ERM problem is not optimal)
Illustration: model fitting and generalization

How do we choose model complexity?

**Underfitting**

![Plot](image1)

**Overfitting**

![Plot](image2)

Underfitting:

- $M = 0$
- $M = 1$
- $M = 3$

Overfitting:

- $M = 9$

[RMS] error defined by

$$E_{RMS} = \sqrt{\frac{1}{N} E(w^*)}$$

in which the division by $N$ allows us to compare different sizes of data sets on an equal footing, and the square root ensures that $E_{RMS}$ is measured on the same scale (and in the same units) as the target variable $t$.

Graphs of the training and test set RMS errors are shown, for various values of $M$, in Figure 1.5. The test set error is a measure of how well we are doing in predicting the values of $t$ for new data observations of $x$. We note from Figure 1.5 that small values of $M$ give relatively large values of the test set error, and this can be attributed to the fact that the corresponding polynomials are rather inflexible and are incapable of capturing the oscillations in the function $\sin(2\pi x)$.

Values of $M$ in the range $3 \leq M \leq 8$ give small values for the test set error, and these also give reasonable representations of the generating function $\sin(2\pi x)$, as can be seen, for the case of $M = 3$, from Figure 1.4.

[C. Bishop, Pattern recognition and Machine learning, 2006]
Overfitting decreases with increasing amount of data

Figure 1.6 Plots of the solutions obtained by minimizing the sum-of-squares error function using the $M=9$ polynomial for $N=15$ data points (left plot) and $N=100$ data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.

It is also interesting to examine the behaviour of a given model as the size of the data set is varied, as shown in Figure 1.6. We see that, for a given model complexity, the over-fitting problem become less severe as the size of the data set increases. Another way to say this is that the larger the data set, the more complex (in other words more flexible) the model that we can afford to fit to the data. One rough heuristic that is sometimes advocated is that the number of data points should be no less than some multiple (say 5 or 10) of the number of adaptive parameters in the model. However, as we shall see in Chapter 3, the number of parameters is not necessarily the most appropriate measure of model complexity.

Also, there is something rather unsatisfying about having to limit the number of parameters in a model according to the size of the available training set. It would seem more reasonable to choose the complexity of the model according to the complexity of the problem being solved. We shall see that the least squares approach to finding the model parameters represents a specific case of maximum likelihood (discussed in Section 1.2.5), and that the over-fitting problem can be understood as a general property of maximum likelihood. By adopting a Bayesian approach, the over-fitting problem can be avoided. We shall see that there is no difficulty from a Bayesian perspective in employing models for which the number of parameters greatly exceeds the number of data points. Indeed, in a Bayesian model the effective number of parameters adapts automatically to the size of the data set.

For the moment, however, it is instructive to continue with the current approach and to consider how in practice we can apply it to data sets of limited size where we...
Visualization of learned network features

- Select the strongest activations in the feature map
- Backproject them into the lower layers

[Zeiler and Fergus, ECCV 2014]
Visualization of learned network features

Layer 3

[Zeiler and Fergus, ECCV 2014]
Visualization of learned network features

Layer 2
Layer 1
Layer 3
Layer 4
Layer 5

Zeiler and Fergus, ECCV 2014
Feature evolution during training

A single strongest activation in the feature map, downprojected
Epochs 1, 2, 5, 10, 20, 30, 40, 64

[Zeiler and Fergus, ECCV 2014]
[Yosinski, Clune, Bengio, Lipson, "How transferable are features in deep neural networks?", 2014]
[Yosinski, Clune, Bengio, Lipson, "How transferable are features in deep neural networks?", 2014]
Difficult problems

Example: Visual Question Answering

“What is the moustache made of?”

Huge Models
How can we still train all this?

- Regularization, normalization, tricks
- Data augmentation
- Throw large/insane amounts of data at the problem
  - Simulation, rendering
  - Complex and tiring acquisitions
- Increase the amount of information used from the data
  - Weakly supervised learning
  - Semi supervised learning
  - Unsupervised learning
  - Reinforcement learning
- Create « smarter » models (inductive bias)
- Learn to focus on the relevant parts the data
  - Attention mechanisms
- Regularization: weight decay

\[ \tilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} \{ y(x_n, w) - t_n \}^2 + \frac{\lambda}{2} \| w \|^2 \]

- Dropout

- Batch normalization

- Residual connections

- Fancy optimisation: SGD, momentum, Adam

Figure 1.7 plots of polynomials fitted to the data set shown in Figure 1.2 using the regularized error function (1.4) for two values of the regularization parameter $\lambda$ corresponding to $\ln \lambda = -18$ and $\ln \lambda = 0$. The case of no regularizer, i.e., $\lambda = 0$, corresponding to $\ln \lambda = -\infty$, is shown at the bottom right of Figure 1.4.

We see that, for a value of $\ln \lambda = -18$, the over-fitting has been suppressed and we now obtain a much closer representation of the underlying function $\sin(2\pi x)$. If, however, we use too large a value for $\lambda$ then we again obtain a poor fit, as shown in Figure 1.7 for $\ln \lambda = 0$. The corresponding coefficients from the fitted polynomials are given in Table 1.2, showing that regularization has the desired effect of reducing the coefficients.
How can we still train all this?

- Regularization, normalization, tricks
- Data augmentation
- Throw large/insane amounts of data at the problem
  - Simulation, rendering
  - Complex and tiring acquisitions
- Increase the amount of information used from the data
  - Weakly supervised learning
  - Semi supervised learning
  - Unsupervised learning
  - Reinforcement learning
- Create « smarter » models (inductive bias)
- Learn to focus on the relevant parts the data
  - Attention mechanisms
Throw insane amounts of data at the problem

CMU Panoptical dataset
How can we still train all this?

- Regularization, normalization
- Data augmentation
- Throw large/insane amounts of data at the problem
  - Simulation, rendering
  - Complex and tiring acquisitions
- Increase the amount of information used from the data
  - Weakly supervised learning
  - Semi supervised learning
  - Unsupervised learning
  - Reinforcement learning
- Create « smarter » models (inductive bias)
- Learn to focus on the relevant parts the data
  - Attention mechanisms
“Pure” Reinforcement Learning (cherry)
- The machine predicts a scalar reward given once in a while.
- A few bits for some samples

Supervised Learning (icing)
- The machine predicts a category or a few numbers for each input
- Predicting human-supplied data
- 10→10,000 bits per sample

Unsupervised/Predictive Learning (cake)
- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- Millions of bits per sample

Slide: Y. LeCun
Unsupervised Learning

« The brain has about $10^{14}$ synapses and we only live for about $10^9$ seconds. So we have more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input is the only place we can get $10^5$ dimensions of constraint per second. »

Geoffrey Hinton
Combining real and simulated data

Joint positions (NYU Dataset)        Synthetic data (part segmentation)

Work of Natalia Neverova
Phd @ LIRIS, Now at Facebook AI

Florian Nebout

With Graham W. Taylor,
University of Guelph, Canada
How can we still train all this?

- Regularization, normalization
- Data augmentation
- Throw large/insane amounts of data at the problem
  - Simulation, rendering
  - Complex and tiring acquisitions
- Increase the amount of information used from the data
  - Weakly supervised learning
  - Semi supervised learning
  - Unsupervised learning
  - Reinforcement learning
- Create « smarter » models (inductive bias)
- Learn to focus on the relevant parts the data
  - Attention mechanisms
Attention in language (google translate)

Figures: Stephen Merity

[Wu et al., arxiv 2016]
Human activity recognition
Pose stream: state of the art

\[ h_0 \rightarrow A \rightarrow A \rightarrow A \rightarrow h_t \]

\[ X_0 \rightarrow X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_t \]

\[ t \quad t+1 \quad t+2 \]
Articulated pose alone is not sufficient

Same class?!
Articulated pose alone is not sufficient

RGB is helpful...

Reading

Writing
Attention on relevant parts

Joint important for activity high attention

Joint wrongly located low attention

Work of Fabien Baradel, Phd @ LIRIS

With Julien Mille (INSA Val de Loire)
RGB raw sequence

Visualization of the attention process of our model

‘Handshaking’

[Baradel, Wolf, Mille, arxiv 2017]
Human attention: gaze patterns

[Johansson, Holsanova, Dewhurst, Holmqvist, 2012]
Recurrent models of visual attention


[Mnih et al., NIPS 2015]
Dynamic visual attention

1. Learn where to attend
2. Learn how to track attended glimpse points (assign glimpses to semantic entities)
3. Learn how to recognize activities from a collection of tracked semantic entities

[Baradel, Wolf, Mille, Taylor (under review)]
Results

<table>
<thead>
<tr>
<th>Methods</th>
<th>Pose</th>
<th>RGB</th>
<th>CS</th>
<th>CV</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lie Group [40]</td>
<td>✓</td>
<td>-</td>
<td>50.1</td>
<td>52.8</td>
<td>51.5</td>
</tr>
<tr>
<td>Skeleton Quads [10]</td>
<td>✓</td>
<td>-</td>
<td>38.6</td>
<td>41.4</td>
<td>40.0</td>
</tr>
<tr>
<td>Dynamic Skeletons [14]</td>
<td>✓</td>
<td>-</td>
<td>60.2</td>
<td>65.2</td>
<td>62.7</td>
</tr>
<tr>
<td>HBRNN [9]</td>
<td>✓</td>
<td>-</td>
<td>59.1</td>
<td>64.0</td>
<td>61.6</td>
</tr>
<tr>
<td>Deep LSTM [32]</td>
<td>✓</td>
<td>-</td>
<td>60.7</td>
<td>67.3</td>
<td>64.0</td>
</tr>
<tr>
<td>Part-aware LSTM [32]</td>
<td>✓</td>
<td>-</td>
<td>62.9</td>
<td>70.3</td>
<td>66.6</td>
</tr>
<tr>
<td>ST-LSTM + TrustG. [26]</td>
<td>✓</td>
<td>-</td>
<td>69.2</td>
<td>77.7</td>
<td>73.5</td>
</tr>
<tr>
<td>STA-LSTM [35]</td>
<td>✓</td>
<td>-</td>
<td>73.2</td>
<td>81.2</td>
<td>77.2</td>
</tr>
<tr>
<td>Ensemble TS-LSTM [24]</td>
<td>✓</td>
<td>-</td>
<td>74.6</td>
<td>81.3</td>
<td>78.0</td>
</tr>
<tr>
<td>GCA-LSTM [27]</td>
<td>✓</td>
<td>-</td>
<td>74.4</td>
<td>82.8</td>
<td>78.6</td>
</tr>
<tr>
<td>JTM [41]</td>
<td>✓</td>
<td>-</td>
<td>76.3</td>
<td>81.1</td>
<td>78.7</td>
</tr>
<tr>
<td>MTLN [18]</td>
<td>✓</td>
<td>-</td>
<td>79.6</td>
<td>84.8</td>
<td>82.2</td>
</tr>
<tr>
<td>VA-LSTM [47]</td>
<td>✓</td>
<td>-</td>
<td>79.4</td>
<td>87.6</td>
<td>83.5</td>
</tr>
<tr>
<td>View-invariant [28]</td>
<td>✓</td>
<td>-</td>
<td>80.0</td>
<td>87.2</td>
<td>83.6</td>
</tr>
<tr>
<td>DSSCA - SSLM [33]</td>
<td>✓</td>
<td>✓</td>
<td>74.9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hands Attention [5]</td>
<td>✓</td>
<td>✓</td>
<td>84.8</td>
<td>90.6</td>
<td>87.7</td>
</tr>
<tr>
<td>C3D†</td>
<td>-</td>
<td>✓</td>
<td>63.5</td>
<td>70.3</td>
<td>66.9</td>
</tr>
<tr>
<td>Resnet50+LSTM†</td>
<td>-</td>
<td>✓</td>
<td>71.3</td>
<td>80.2</td>
<td>75.8</td>
</tr>
<tr>
<td><strong>Ours</strong></td>
<td>-</td>
<td>✓</td>
<td><strong>86.6</strong></td>
<td><strong>93.2</strong></td>
<td><strong>89.9</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Glimpse</th>
<th>Type of attention</th>
<th>CS</th>
<th>CV</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D tubes</td>
<td>Attention</td>
<td>85.8</td>
<td>92.7</td>
<td>89.2</td>
</tr>
<tr>
<td>Seq. 2D</td>
<td>Random sampling</td>
<td>80.3</td>
<td>87.8</td>
<td>84.0</td>
</tr>
<tr>
<td>Seq. 2D</td>
<td>Saliency</td>
<td>86.2</td>
<td>92.9</td>
<td>89.5</td>
</tr>
<tr>
<td>Seq. 2D</td>
<td>Attention</td>
<td><strong>86.6</strong></td>
<td><strong>93.2</strong></td>
<td><strong>89.9</strong></td>
</tr>
</tbody>
</table>

[Baradel, Wolf, Mille, Taylor (under review)]
Some peculiar properties of deep models
Model fitting and generalization

How do we choose model complexity?

1.1. Example: Polynomial Curve Fitting

![Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown in Figure 1.2.](image)

Figure 1.4

[RMS] error defined by

\[
E_{\text{RMS}} = \sqrt{\frac{2}{E}(w^{\star})}/N
\]

in which the division by \(N\) allows us to compare different sizes of data sets on an equal footing, and the square root ensures that \(E_{\text{RMS}}\) is measured on the same scale (and in the same units) as the target variable \(t\). Graphs of the training and test set RMS errors are shown, for various values of \(M\), in Figure 1.5. The test set error is a measure of how well we are doing in predicting the values of \(t\) for new data observations of \(x\). We note from Figure 1.5 that small values of \(M\) give relatively large values of the test set error, and this can be attributed to the fact that the corresponding polynomials are rather inflexible and are incapable of capturing the oscillations in the function \(\sin(2\pi x)\). Values of \(M\) in the range \(3 \leq M \leq 8\) give small values for the test set error, and these also give reasonable representations of the generating function \(\sin(2\pi x)\), as can be seen, for the case of \(M = 3\), from Figure 1.4.

[C. Bishop, Pattern recognition and Machine learning, 2006]
Big Data!

Overfitting decreases with increasing amount of data

Figure 1.6: Plots of the solutions obtained by minimizing the sum-of-squares error function using the $M=9$ polynomial for $N=15$ data points (left plot) and $N=100$ data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.

Intuitively, what is happening is that the more flexible polynomials with larger values of $M$ are becoming increasingly tuned to the random noise on the target values.

It is also interesting to examine the behaviour of a given model as the size of the data set is varied, as shown in Figure 1.6. We see that, for a given model complexity, the over-fitting problem become less severe as the size of the data set increases. Another way to say this is that the larger the data set, the more complex (in other words more flexible) the model that we can afford to fit to the data. One rough heuristic that is sometimes advocated is that the number of data points should be no less than some multiple (say 5 or 10) of the number of adaptive parameters in the model. However, as we shall see in Chapter 3, the number of parameters is not necessarily the most appropriate measure of model complexity.

Also, there is something rather unsatisfying about having to limit the number of parameters in a model according to the size of the available training set. It would seem more reasonable to choose the complexity of the model according to the complexity of the problem being solved. We shall see that the least squares approach to finding the model parameters represents a specific case of maximum likelihood (discussed in Section 1.2.5), and that the over-fitting problem can be understood as a general property of maximum likelihood. By adopting a Bayesian approach, the over-fitting problem can be avoided. We shall see that there is no difficulty from a Bayesian perspective in employing models for which the number of parameters greatly exceeds the number of data points. Indeed, in a Bayesian model the effective number of parameters adapts automatically to the size of the data set.

For the moment, however, it is instructive to continue with the current approach and to consider how in practice we can apply it to data sets of limited size where we

$M=9$

[C. Bishop, Pattern recognition and Machine learning, 2006]
Regularisation

Restrict the expressive power of the model.

\[
\tilde{E}(w) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2 + \frac{\lambda}{2} \|w\|^2
\]

\(\ln \lambda = -18\)

\(\ln \lambda = 0\)

\(M=9\)

[C. Bishop, Pattern recognition and Machine learning, 2006]
Learning and generalization

Can we measure the complexity of a network architecture (a hypothesis class)?

Rademacher complexity of a class $\mathcal{H}$ with respect to samples $x$ and a distribution of random labels $\sigma_i$:

$$\mathcal{R}(\mathcal{H}) = \mathbb{E}_\sigma \left[ \sup_{h \in \mathcal{H}} \frac{1}{n} \sum_i \mathcal{L}(f(x_i, \theta), \sigma_i) \right]$$
However …

- Classical theory seems to fail with DL
- Deep models generalize very well, indicating model complexity fitted to the problem
- Recent work: deep models can learn random labels (!) on very large problems (ImageNet!) with zero error
- « *The effective capacity of a neural network is large enough for brute-force memorization of the entire data set* »

[Zhang, Bengio, Hardt, Recht, Vinyals, ICLR 2017]
Conclusion: current tendencies

- Learning where to look (attention mechanisms)

- Learning what to store where (e.g. external memories, Neural Turing machines)

- Learning how to generate data (e.g. adversarial networks) and through it, representations.

- Learning to go deeper (e.g. residual networks, highway networks)

- Combine Vision and Language

- Deep Reinforcement Learning