Deep Learning Models for Multimodal Sensing and Processing

Prepared by:
Farnaz Abtahi

Advisor:
Prof. Zhigang Zhu

Submitted in Partial Fulfillment of the Requirements for the Computer Science PhD Program at CUNY Graduate Center
Table of Contents

1. Introduction .................................................................................................................. 4

2. Support Vector Machines and Linear Discriminant Analysis ................................. 5
   2.1. SVM and LDA: Basic Models ................................................................................. 5
   2.2. SVM and LDA for Multimodal Data ..................................................................... 9
   2.3. Comparisons and Discussions .............................................................................. 12

3. Restricted Boltzmann Machines and Deep Learning Models ............................... 14
   3.1. RBM: Basic Model ............................................................................................... 14
   3.2. RBM-based Deep Learning Models ...................................................................... 19
      3.2.1. Deep Belief Networks .................................................................................. 19
      3.2.2. Deep Boltzmann Machines ........................................................................... 21
      3.2.3. Deep Autoencoders ..................................................................................... 23
   3.3. Multimodal Deep Learning using RBM-based Models ........................................ 24
   3.4. Comparisons and Discussions ............................................................................. 36

4. Convolutional Neural Networks ................................................................................. 39
   4.1. CNN: Basic Model ............................................................................................... 39
   4.2. Multimodal Deep Learning using CNNs ............................................................... 45
   4.3. Comparisons and Discussions ............................................................................. 48

5. Conclusions and Discussions ...................................................................................... 50

References .......................................................................................................................... 58
1. Introduction

Multimodal sensing and processing have shown promising results in detection, recognition and identification in various applications, such as human-computer interaction, surveillance, medical diagnosis, biometrics, etc. There are many ways to generate multiple modalities; one is via sensor diversity (specially in our everyday life tasks) and the other is via feature diversity (using engineered and/or learned features).

In the last few decades, many machine learning models have been proposed to deal with multimodal data. Hidden Markov Models (HMMs) have been very popular in dealing with dynamic data such as audio/visual speech [1], Gaussian Mixture Models (GMMs) for spatial-temporal data [2], Support Vector Machines (SVMs) and Linear Discriminant Analysis (LDA) for many tasks involving multimodal classification and recognition, etc [3,4].

In this survey, we will mostly focus on deep learning models for multimodal sensing and processing. The first group of multimodal deep learning approaches that we study are all based on Restricted Boltzmann Machines (RBMs) [5]. These methods include Deep Belief Networks (DBNs) [6], Deep Boltzmann Machines (DBNs) [7] and Deep Autoencoders [8]. Not only the building blocks of all these models are RBMs, their training algorithms are also very similar. In the literature we have surveyed, these methods have been compared to more traditional classification approaches such as SVMs and LDA. For that reason, before getting into deep learning models, we first briefly introduce SVM and LDA and mention a few of their applications in processing and classifying multimodal data.

Another successful deep learning model that has been widely used in the past few years is Convolutional Neural Network (CNN) [9]. Due to its nature, CNN is more promising when applied to 2D data, or images in particular. After RBM-based approaches that are introduced in section 3, we will introduce CNNs and their application as a multimodal deep model. The last section will summarize the survey and list our findings and future work.
2. Support Vector Machines and Linear Discriminant Analysis

2.1. SVM and LDA: Basic Models

SVMs were introduced in 1992 by Boser, Guyon and Vapnik [10] and have shown good empirical performance in many applications (bioinformatics, text, image recognition, etc). Figure 1 shows the basic idea of SVM for the simple case of 2D 2-class classification problem. Assuming that we represent the input/output sets as $X$ and $Y$, the goal is to learn the function $y = f(x, \alpha)$, where $\alpha$ are the parameters of the function. In the example of Figure 1, $f$ can be defined as $f(x, \{w, b\}) = \text{sign}(w \cdot x + b)$. So the goal is to find the best set of parameters $w$ and $b$ so that the margin is maximized.

![Figure 1. SVM on 2D 2-class data.](image)

For inseparable classes, the function $f$ is nonlinear and hard to find. In this case, the trick is to map data into a richer feature space including nonlinear features and then construct a hyperplane in that space to separate the classes in a linear way. This is shown in Figure 2. Formally, we need to preprocess the data with $x \rightarrow \Phi(x)$, and then learn the map from $\Phi(x)$ to $y$: $f(x) = w \cdot \Phi(x) + b$. 
Figure 2. Mapping inseparable 2D data to a separable feature space.

The problem here is that the dimensionality of $\Phi(x)$ can be very large, making $w$ hard to represent explicitly in memory, and hard for the QP to solve. For SVMs, for some variables $\alpha$,

$$w = \sum_{i=1}^{m} \alpha_i \Phi(x_i)$$

Now instead of optimizing $w$ directly, we can thus optimize $\alpha$ and the decision rule becomes:

$$f(x) = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \cdot \Phi(x) + b$$

We call $K(x_i, x) = \Phi(x_i) \cdot \Phi(x)$ the "kernel function". After applying the kernel, the decision function becomes:

$$f(x) = \sum_{i} \alpha_i \Phi(x_i) \cdot \Phi(x) + b$$

$$= \sum_{i} \alpha_i K(x_i, x) + b$$

and the goal becomes to minimize $P(w, b)$, where:

$$\min P(w, b) = \frac{1}{2} \left\| \sum_{i=1}^{m} \alpha_i \Phi(x_i) \right\|^2 + \sum_{i} C[H_1[y_i f(x_i)]]$$

(maximize margin) (minimize training error)
LDA [11, 12] is another commonly used techniques for data classification and dimensionality reduction. LDA easily handles the case where the within-class frequencies are unequal. This method maximizes the ratio of between-class variance to the within-class variance in any particular data set thereby guaranteeing maximal separability. LDA functions similar to Principal Component Analysis (PCA) [12]. The prime difference between LDA and PCA is that PCA does more of feature classification and LDA does data classification. In PCA, the shape and location of the original data sets changes when transformed to a different space whereas LDA does not change the location but only tries to provide more class separability and draw a decision region between the given classes. This method also helps to better understand the distribution of the feature data. The example in Figure 3 illustrates the theory of LDA.

Data sets can be transformed and test vectors can be classified in the transformed space by two different approaches [13]:

1. Class-dependent transformation: This type of approach involves maximizing the ratio of between class variance to within class variance. The main objective is to maximize this ratio so that adequate class separability is obtained. The class-specific type approach involves using two optimizing criteria for transforming the data sets independently.

2. Class-independent transformation: This approach involves maximizing the ratio of overall variance to within class variance. This approach uses only one optimizing criterion to transform the data sets and hence all data points irrespective of their class identity are transformed using this transform. In this type of LDA, each class is considered as a separate class against all other classes.
Figure 3. Data sets in original space and transformed space along with the transformation axis for class dependent LDA of a 2-class problem.

Based on the above two approaches to LDA, the LDA algorithm for a 2D class for 100 data points in each class can be divided into three main steps:

1. Formulate the data sets and the test sets, which are to be classified in the original space. For ease of understanding we represent the data sets as a matrix consisting of features in the form given below:

   \[
   \begin{align*}
   \text{set1} &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ \vdots & \vdots \\ a_{m1} & a_{m2} \end{bmatrix}, & \text{set2} &= \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ \vdots & \vdots \\ b_{m1} & b_{m2} \end{bmatrix}.
   \end{align*}
   \]

2. Compute the mean of each data set and mean of entire data set.

   \[
   \mu_3 = p_1 \times \mu_1 + p_2 \times \mu_2,
   \]

   where \( p_1 \) and \( p_2 \) are apriori probabilities of classes, in this case 0.5 and 0.5.
3. In LDA, within-class and between-class scatters are used to formulate the criteria for class separability. Within-class scatter is the expected covariance of each of the classes:

\[ S_w = \sum_j p_j \times (cov_j) \]

Therefore, for the two-class problem,\n
\[ S_w = 0.5 \times cov_1 + 0.5 \times cov_2 \]

The optimizing factors in case of class dependent type are computed as:

\[ criterion_j = inv(cov_j) \times S_b \]

For the class independent transform, the optimizing criterion is computed as:

\[ criterion = inv(S_w) \times S_b \]

where in both cases:

\[ cov_j = (x_j - \mu_j)(x_j - \mu_j)^T \]

\[ S_b = \sum_j (\mu_j - \mu_3) \times (\mu_j - \mu_3)^T \]

### 2.2. SVM and LDA for Multimodal Data

Several groups of researchers have proposed multimodal classification and data fusion SVM and LDA-based approaches. The authors of [14] claim that existing multi-biometric fusion techniques face a number of limitations since they are based on the assumptions that each biometric modality is local, complete, and static. These limitations are particularly pronounced when considered in the context of biometric identification, as opposed to verification. Key limitations include:

1. Each registered person must be entered into every modality. This may not be plausible and is very restrictive. Moreover, this makes adding additional modalities to an existing system difficult or impossible.

2. All of the classifiers must always be available. This will not be the case if the modalities are part of a distributed system, such as when a multi-biometric
system is composed of traditional biometric systems that are maintained by different groups.

3. No support for “offline” biometrics. Offline biometrics (such as DNA profiles) require laboratory processing to register individuals into the biometric system; the associated time and cost exacerbates limitations 1 and 2 listed above, and makes the utilization of offline biometrics impossible in existing biometric fusion systems.

4. Registration changes may decrease system accuracy. If learning is only performed when initially creating the multi-biometric system, the accuracy of the biometric fusion may degrade as individuals are later added to or removed from the system.

5. Limited to verification. Due to the other limitations listed above, most existing fusion techniques are explicitly designed for verification only – identification is not supported.

They propose a novel multi-biometric fusion technique that addresses the issues listed above and is suitable for both identification and verification. A mediator agent controls the fusion of the individual biometric match scores, using a “bank” of SVMs that cover all possible subsets of the biometric modalities being considered. This agent selects an appropriate SVM for fusion, based on which modality classifiers are currently available and have sensor data for the identity in question. This fusion technique differs from a traditional SVM ensemble – rather than combining the output of all of the SVMs, they apply only the SVM that best corresponds to the available modalities. The mediator agent also controls the learning of new SVMs when modalities are added to the system or sufficient changes have been made to the data in existing modalities.

The experiments utilize the following biometric modalities: face, fingerprint, and DNA profile data. They empirically show that their multiple SVM technique produces more accurate results than the traditional single SVM approach. The pipeline of this approach is shown in Figure 4.
In another work [15], LDA is used to build a multimodal biometric identity system that automatically recognizes or identifies the user based on voice and facial information. The system is demonstrated in Figure 5.

The system takes in the inputs and passes through two modules built in it namely visual recognition system and audio recognition system. Visual Recognition System makes an attempt to match the facial features of a user to its template in the database. It uses Principal Component Analysis, Linear Discriminant Analysis and K-nearest neighbor.
The Principal Component Analysis reduces the large dimensionality of the data space (observed variables) to the smaller intrinsic dimensionality of feature space (independent variables), which is needed to describe the data economically. Unlike PCA, LDA explicitly attempts to model the difference between the classes of data and factor analysis builds the feature combinations based on differences rather than similarities. LDA is also different from factor analysis in that it is not an interdependence technique: a distinction between independent variables and dependent variables (also called criterion variables) must be made. Therefore LDA significantly reduces the dimensionality of the data obtained after PCA and with the KNN classifier produces significant improvement in performance.

Audio Recognition System takes into account the fact that performance of a speech recognition system is affected considerably by the choice of features for most of the applications. Raw data obtained from speech recording cannot be used to directly train the recognizer, as for the same phonemes do not necessarily have same sample values.

2.3. Comparisons and Discussions

Both LDA and SVM compute hyperplanes that are optimal with respect to their individual objectives. However, there can be vast differences in performance between the two techniques depending on the extent to which their respective
assumptions agree with problems at hand. In [16] the authors compare the two techniques analytically and experimentally using a number of datasets. For analytical comparison purposes, a unified representation is developed and a metric of optimality is proposed.

It’s true that LDA and linear SVM share much in common, both draw a line, the technical differences between these two is significant. As a very informal explanation, LDA draws lines, while SVM can be nonlinear and draw curves instead. Also, as the linear SVM is a super-class (or generalization of) LDA, it is generally the better or more sophisticated approach.
3. Restricted Boltzmann Machines and Deep Learning Models

3.1. RBM: Basic Model

RBMs [5] are a group of non-directed probabilistic energy-based graphical models that assign a scalar energy value to each variable configuration. These models are trained in a way that the plausible configurations are associated with lower energies (higher probabilities). An RBM has three components: visible layer, hidden layer, and a weight matrix containing the weights of the connections between visible and hidden units. There are no connections between the visible units or between the hidden units. That is the reason why this model is called “restricted”. Figure 6 shows a sample RBM.

As mentioned above, every RBM tries to optimize its energy function in order to maximize the probability of the training data. The energy function has the following form.

\[
E(x, h) = -h'Wx - c'x - b'h = - \sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j
\]
The goal is for the model to represent the probability distribution of the training data, \( x \), using a layer of binary hidden units, \( h \). This probability distribution is defined as follows.

\[
p(x, h) = \frac{exp(-E(x, h))}{Z}
\]

Where \( Z \) is called the normalization factor, or the partition function. In practice, computing the partition function and hence the joint probability distribution \( p(x,h) \) and the probability distribution of the input \( p(x) \) is intractable. So we need a solution to this problem.

Before getting into a detailed description of how to compute these probabilities, we should note that because of the structure of the model and its connections, the two probabilities \( p(x|h) \) and \( p(h|x) \) have an interesting form. We will later use these probabilities for inference and drawing samples from the RBM.

\[
p(h|x) = \prod_j p(h_j|x)
\]

\[
p(x|h) = \prod_k p(x_k|h)
\]

Then the probability of a visible or a hidden unit being “on” will be as follows.

\[
p(h_j = 1|x) = \frac{1}{1 + \exp\left(-\left(b_j + W_{j}.x\right)\right)} = \text{sigm}(b_j + W_{j}.x)
\]

\[
p(x_k = 1|h) = \frac{1}{1 + \exp\left(-\left(c_k + h'.W_{k}\right)\right)} = \text{sigm}(c_k + h'.W_{k})
\]

An efficient algorithm for training RBMs is called Contrastive Divergence (CD) [17]. We know that to train an RBM, the goal is to maximize the probability of the training data. Thus we can set our loss function to the average negative log-likelihood of the training data:
Loss function: \[ \frac{1}{T} \sum_{t} -\log p(x^t) \]

Where \( x^t \) is the \( t \)th observation or training sample and \( T \) is the total number of training samples. We proceed by using stochastic gradient descent for optimizing the loss function and that means we need every partial derivative of the loss function with respect to any parameters \( \theta \) of the model.

\[
\frac{\partial -\log p(x^t)}{\partial \theta} = E_h \left[ \frac{\partial E(x^t, h)}{\partial \theta} \bigg| x^t \right] - E_{x,h} \left[ \frac{\partial E(x^t, h)}{\partial \theta} \right]
\]

Positive phase  Negative phase

As we can see, the computation has two parts. The first part is called the positive phase and depends on the observation whereas the second part, namely the negative phase, only depends only on the model. The expectations are performed because we can never know what the values of hidden parameters are. So an averaging would be necessary.

The problem is that the negative phase is still hard to compute because we have to make an exponential sum over both \( x \) and \( h \). But having a value for the observation \( x \), summing over all possible values of \( h \) is actually achievable. So we have to approximate the negative part in order to perform the stochastic descent efficiently.

The idea of CD is summarized below:

- Replace the expectation by a point estimate at \( x \).
- Obtain the point \( x \) by Gibbs sampling [18].
- Start sampling the chain at \( x^t \).

The sampling process is shown in Figure 7. We start by setting the value of the visible units to an observation. Then using this value, we obtain a sample of the hidden units by sampling from \( p(h|x) \). This sample is then used to generate a new
sample from the visible units using \( p(x|h) \), and these steps are repeated. In theory, we need to repeat the sampling process infinite number of times, but in practice, sampling for only a few iterations would suffice. The very last sample of \( x \) in the sequence of samplings is used as a negative sample for training the model. Intuitively, the reason is that this sample is what the model “believes” the input should look like. So we would like the model to forget it and learn to generate a better approximation of the true sample instead.

![Figure 7. Gibbs sampling for training the RBM.](image)

Now that we understand the intuition behind the CD algorithm, we will look into the update rule for the parameters of the model. The goal is the derivation of

\[
\frac{\partial E(x,h)}{\partial \theta}
\]

with respect to every parameter of the model. Assume we are interested in the partial derivative for \( \theta = W_{jk} \).

\[
\frac{\partial E(x,h)}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}} \left( - \sum_{jk} W_{jk} h_j x_k - \sum_{k} c_k x_k - \sum_{j} b_j h_j \right) \\
= - \frac{\partial}{\partial W_{jk}} \sum_{jk} W_{jk} h_j x_k \\
= - h_j x_k
\]

So the result in matrix form is:
Now the expectation with respect to \( h \) conditioned on any value of the vector \( x \) will be computed as follows.

\[
E_h \left[ \frac{\partial E(x, h)}{\partial W_{jk}} \right] = E_h \left[ -h_j x_k \right] = \sum_{h_j \in \{0, 1\}} -h_j x_k p(h_j | x) = -x_k p(h_j = 1 | x)
\]

\[
E_h [\nabla_W E(x, h) | x] = -h(x) x'
\]

Where,

\[
h(x) = \begin{pmatrix} p(h_1 = 1 | x) \\ \vdots \\ p(h_H = 1 | x) \end{pmatrix} = \text{sigm}(b + Wx)
\]

Now, given a training sample \( x^t \) and a negative sample \( \bar{x} \) obtained by Gibbs sampling from the RBM, the learning rule for \( \theta = W \) becomes:

\[
W \leftarrow W - \alpha (\nabla_W - \log p(x^t))
\]

\[
\leftarrow W - \alpha \left( E_h [\nabla_W E(x^t, h) | x^t] - E_{x,h} [\nabla_W E (x, h)] \right)
\]

\[
\leftarrow W - \alpha \left( E_h [\nabla_W E(x^t, h) | x^t] - E_h [\nabla_W E (\bar{x}, h) | \bar{x}] \right)
\]

\[
\leftarrow W + \alpha \left( h(x^t) x^t' - h(\bar{x}) \bar{x}' \right)
\]

So putting everything together, the CD algorithm works as follows.

1. For each training example \( x^t \)
   a. Generate a negative sample \( \bar{x} \) using \( k \) steps of Gibbs sampling, starting at \( x^t \).
   b. Update parameters:
      \[
      W \leftarrow W + \alpha (h(x^t) x^t' - h(\bar{x}) \bar{x}')
      \]
      \[
      b \leftarrow b + \alpha (h(x^t) - h(\bar{x}))
      \]
      \[
      c \leftarrow c + \alpha (x^t - \bar{x})
      \]
2. Go back to 1 until stopping criteria.

The CD algorithm is also represented by CD-k, where k is the number of iterations of the Gibbs sampling. In general, the greater k is, the less biased the estimate of the gradient will be. In practice though, k=1 works well enough.

3.2. **RBM-based Deep Learning Models**

3.2.1. **Deep Belief Networks**

DBNs [6,19] are probabilistic graphical models built by stacking up RBMs. Connections between layers are bi-directional and symmetric, which means both directions share the same weights and information flows in both directions. Figure 8 (left) shows an RBM with 4 visible and 3 hidden units. A DBN is illustrated in Figure 8 (right).

![RBM](image1)

**Figure 8.** Left: An RBM with 4 visible (input) and 3 hidden units. Right: A DBN with the same number of units in all layers.

The top two layers is an RBM with a probability distribution expressed as \( p(h^1, h^2) \). The other layers form a Bayesian network for which the conditional probabilities of the hidden layers or visible layers are defined as:
The full distribution of the DBN is as follows

\[ p(x, h^1, h^2, h^3) = p(h^2) p(h^1 | h^2) p(h^3 | h^2) p(x | h^1) \]

Where,

\[ p(h^2, h^3) = \exp\left( h^2 W^3 h^3 + b^2 h^2 + b^3 h^3 \right) / Z \]
\[ p(h^1 | h^2) = \prod_j p(h^1_j | h^2) \]
\[ p(x | h^2) = \prod_i p(x_i | h^2) \]

DBNs can be trained using the CD algorithm to extract a deep hierarchical representation of the training data [19]. During the learning process, the DBN is first trained one layer at a time, in a greedy unsupervised manner, by treating the values of hidden units in each layer as the training data for the next layer (except for the first layer, which is fed with the raw input data). This learning procedure, called pre-training, finds a set of weights that determine how the variables in one layer depend on the variables in the layer above. These parameters capture the structural properties of the training data. If the network is to be used for a classification task, then a supervised discriminative fine-tuning is performed by adding an extra layer of output units and back-propagating the error derivatives (using some form of stochastic gradient descent, or SGD [20]).

To generate a sample from the DBN, we need to perform Gibbs sampling for a long time between the top two layers \( h^1 \) and \( h^2 \) until we converge to a sample of the \( h^2 \) layer, then traverse the rest of the DBN in a top-down manner using the
conditional probability distributions to generate the desired sample at the visible layer [21].

The work presented in [22] studies the reasons why pre-trained deep networks work much better than traditional neural networks and proposes several possible explanations. One possible explanation is that pre-training initializes the parameters of the network in an area of parameter space where optimization is easier and better local optima is found. This is equivalent to penalizing solutions that are outside a particular region of the solution space. Another explanation is that pre-training acts as a kind of regularizer that minimizes the variance and introduces a bias towards configurations of the parameters that stochastic gradient descent can explore during the supervised learning phase, by defining a data-dependent prior on the parameters obtained through the unsupervised learning. In other words, pre-training implicitly imposes constraints on the parameters of the network to specify which minimum out of all local minima of the objective function is desired. The effect of pre-training relies on the assumption that the true target conditional distribution $p(y|x)$ shares structure with the input distribution $p(x)$.

### 3.2.2. Deep Boltzmann Machines

The next model that we introduce is the Deep Boltzmann Machine (DBM) [7]. DBMs are diagrammatically very similar to DBNs, but they are qualitatively very different since DBNs are directed while DBMs are undirected graphical models (Figure 9, left). As a result, unlike DBNs, the approximate inference procedure in DBMs, in addition to an initial bottom-up pass, can incorporate top-down feedback, allowing DBMs to better propagate uncertainty about, and hence deal more robustly with, ambiguous inputs.

We can apply approximate maximum likelihood to determine the parameters of a DBM, but it is rather slow. Instead, we also use a greedy layer-wise pre-training method to initialize the model parameters to good values (Figure 9, middle and right) [23].
Figure 9. Left: A three-layer DBN and a three-layer DBM. Right: Pre-training consists of learning a stack of modified RBM’s, that are then composed to create a DBM. [23]

The greedy layer-by-layer pre-training algorithm relies on learning a stack of RBM’s with a small modification. The key intuition is that for the lower-level RBM to compensate for the lack of top-down input into h1, the input must be doubled, with the copies of the visible-to-hidden connections tied. Conversely, for the top-level RBM to compensate for the lack of bottom-up input into h2, the number of hidden units is doubled. For the intermediate layers, the RBM weights are simply doubled. The stack of RBMs can then be trained in a greedy layer-by-layer fashion using the CD algorithm. When these three modules are composed to form a single model, the layer copies are removed and the total inputs coming into the first and second hidden layers are halved. For the intermediate RBM, the weights are halved in both directions. The algorithm is summarized in Algorithm 1 [23]. Greedily pre-training the weights of a DBM initializes the weights to reasonable values, which facilitates the subsequent joint learning of all layers.

**Algorithm 1** Greedy Pretraining Algorithm for a Deep Boltzmann Machine with 3-layers.

1: Make two copies of the visible vector and tie the visible-to-hidden weights $W^1$. Fit $W^1$ of the 1st layer RBM to data.
2: Freeze $W^1$ that defines the 1st layer of features, and use samples $h^1$ from $P(h^1|v, 2W^1)$ as the data for training the next layer RBM with weight vector $2W^2$.
3: Freeze $W^2$ that defines the 2nd layer of features and use the samples $h^2$ from $P(h^2|h^1, 2W^2)$ as the data for training the 3rd layer RBM with weight vector $2W^3$.
4: When learning the top-level RBM, double the number of hidden units and tie the visible-to-hidden weights $W^3$.
5: Use the weights $\{W^1, W^2, W^3\}$ to compose a Deep Boltzmann Machine.
3.2.3. Deep Autoencoders

Another model that is also similar to the models we have introduced in this section so far is the Autoencoder model [24] and one of its extensions - Denoising Autoencoder [25, 26]. We begin with a short discussion on Autoencoders. A deep autoencoder is composed of two, symmetrical deep belief networks that typically have four or five shallow layers representing the encoding half of the net, and second set of four or five layers that make up the decoding half.

The layers are restricted Boltzmann machines, the building blocks of deep-belief networks. Figure 10 shows a simplified schema of a deep autoencoder’s structure.

![Deep Autoencoder Diagram](image)

Figure 10. An example of Autoencoder.

The goal is to optimize the weights in both blocks in order to minimize the reconstruction error. The reconstruction error can be measured in many ways, depending on the appropriate distributional assumptions on the input. One of the options is the traditional squared error. If the input is interpreted as either bit vectors or vectors of bit probabilities, cross-entropy of the reconstruction can be used.

The denoising Autoencoder (dA) is an extension of a classical autoencoder which was introduced as a building block for deep networks in [25]. The idea behind denoising autoencoders is that in order to force the hidden layer to discover more
robust features and prevent it from simply learning the identity, we train the autoencoder to reconstruct the input from a corrupted version of it.

The denoising autoencoders is a stochastic version of the autoencoder. Intuitively, a denoising autoencoder does two things: try to encode the input (preserve the information about the input), and try to undo the effect of a corruption process stochastically applied to the input of the autoencoder. The latter can only be done by capturing the statistical dependencies between the inputs. The denoising autoencoder can be understood from different perspectives (the manifold learning perspective, stochastic operator perspective, bottom-up – information theoretic perspective, top-down – generative model perspective), all of which are explained in [25].

In [25], the stochastic corruption process randomly sets some of the inputs (as many as half of them) to zero. Hence the denoising autoencoder is trying to predict the corrupted (i.e. missing) values from the uncorrupted (i.e., non-missing) values, for randomly selected subsets of missing patterns. Note how being able to predict any subset of variables from the rest is a sufficient condition for completely capturing the joint distribution between a set of variables (this is how Gibbs sampling works).

To convert the autoencoder class into a denoising autoencoder class, all we need to do is to add a stochastic corruption step operating on the input. The input can be corrupted in many ways, one of which would be the original corruption mechanism of randomly masking entries of the input by making them zero.

### 3.3. Multimodal Deep Learning using RBM-based Models

DBNs, DBMs and deep autoencoders have recently been used in a number of multimodal applications. Srivastava and Salakhutdinov [27] proposed DBNs for learning the joint representation of data. The model was used on two modalities: text and image. According to the study, this model could deal with missing modalities and could be used for both image retrieval and image annotation. Achieved joint representation using the DBN showed superior results compared to SVM and LDA. As
a running example, multimodal DBN was constructed using an image-text bi-modal DBN. Each data modality was constructed using a separate two-layer DBN. The image pathway consists of a Gaussian RBM with 3857 visible and 1000 hidden units, followed by another layer of 1000 hidden units. The text pathway consists of a Replicated Softmax Model with 2000 visible and 1000 hidden units followed by another layer of 1000 hidden units. The joint layer also contains 1000 hidden units. The model was not found to be very sensitive to the choice of these hyperparameters.

![Multimodal DBN Diagram](image)

Figure 11. Left: Image-specific two-layer DBN that uses a Gaussian model to model the distribution over real-valued image features. Middle: Text-specific two-layer DBN that uses a Replicated Softmax model to model its distribution over the word count vector. Right: A multimodal DBN that models the joint distribution over image and text inputs. [27]

The probability that each DBN model assigns to a visible vector is:

\[
P(v_m) = \sum_{h^{(1)}, h^{(2)}} P(h^{(2)}, h^{(1)}) P(v_m | h^{(1)})
\]  
\[
P(v_t) = \sum_{h^{(1)}, h^{(2)}} P(h^{(2)}, h^{(1)}) P(v_t | h^{(1)})
\]  

Where \(v_m \in \mathbb{R}^D\) denotes an image and \(v_t \in \mathbb{N}^K\) denotes a text input.

The image-specific DBN uses Gaussian RBM to model the distribution over real-valued image features, whereas text-specific DBN uses Replicated Softmaxes to model the distribution over word count vectors. The conditional probabilities of the visible given hidden units used in equations 1 and 2 are as shown in equations 3 and 4 respectively:
To form a multimodal DBN, the two models are combined by learning a joint RBM on top of them. The joint distribution is:

\[
P(v_k = 1|\mathbf{h}; \theta) = \frac{\exp(-b_k + \sum_{j=1}^{J} W_{kj} h_j)}{\sum_{k'=1}^{K} \exp(-b_{k'} + \sum_{j=1}^{J} W_{k'j} h_j)}
\]

The parameters of this multimodal DBN can be learned approximately by greedy layer-wise training using CD.

In case of missing values of one of the modalities, or generating missing values of one modality when given the values of the other, at first the values of the hidden variable of the given modality are inferred all the way to the last hidden layer. At the top level RBM, alternating Gibbs sampling is performed using the following conditional distribution:

\[
P(\mathbf{v}_m, \mathbf{v}_t) = \sum_{\mathbf{h}_m^{(2)}, \mathbf{h}_t^{(2)}, \mathbf{h}^{(3)}} P(\mathbf{h}_m^{(2)}, \mathbf{h}_t^{(2)}, \mathbf{h}^{(3)}) \times \sum_{\mathbf{h}_m^{(1)}} P(\mathbf{v}_m|\mathbf{h}_m^{(1)}) P(\mathbf{h}_m^{(1)}|\mathbf{h}_m^{(2)}) \times \sum_{\mathbf{h}_t^{(1)}} P(\mathbf{v}_t|\mathbf{h}_t^{(1)}) P(\mathbf{h}_t^{(1)}|\mathbf{h}_t^{(2)}).
\]

The nearest neighbors to these features are located and the corresponding
inputs are then retrieved. L2 distance between the feature vectors was used to find nearest neighbors (all features were normalized to have zero mean and unit variance). An example of how the DBN performs on the MIR Flicker Dataset is shown in Figures 12 and 13.

Presented model was used for classification tasks by adding a simple logic classifier to do 1-vs-all classification on top of the multimodal DBN and then fine-tuning it by using stochastic gradient descent.

Experiments were conducted using the MIR Flickr Data set which consisted of 1 million images among which 25,000 have been annotated for 24 categories such as object categories: birds, tree, and scene categories: sky, night; totaling to 38 classes allowing the same image to be a part of several classes. The unlabeled 975,000 images were used only for pre-training the DBN, 15,000 images were used for
training and 10,000 for testing. Mean Average Precision (MAP) was used to judge the performance. Average number of tags associated with a single image was approximately 5.15 with the SD of 5.13. About 18% of the labeled data did not have any tag. Word counts were replaced with $d \log(1 + w)e$. Pyramid Histogram of Words (PHOW) features [28], Gist [29] and MPEG-7 descriptors [30] (EHD, HTD, CSD, CLD, SCD) were concatenated to get a 3857 dimensional representation of images. PHOW features are bags of image words obtained by extracting dense SIFT features [31] over multiple scales and clustering them. Each dimension was mean-centered. PHOW features are bags of image words obtained by extracting dense SIFT features over multiple scales and clustering them.

The results of the experiments (in discriminative aspect) show that DBN-Lab (variation of the model without the SIFT-based features) achieves a MAP (over 38 classes) of 0.503, compared to 0.475 and 0.492 achieved by SVM and LDA models. DBN-Unlab model (variation of the model that was pre-trained using the unlabeled data) significantly improved upon DBN-Lab almost across all classes, achieving a MAP of 0.532. Next variation of the model included SIFT-based features along with unlabeled data used for pre-training achieved a MAP of 0.563. Model was also compared to an autoencoder that was initialized with the DBN weights and fine-tuned as in [32]. Autoencoder performs much better than SVM and LDA getting a MAP of 0.547. Autoencoder does better than the DBN model on some categories, however, on average it does not do as well.
In another work [33], the same group investigates the use of the Deep Boltzmann Machines for extraction of the unified representation of the diverse input modalities, which is useful for classification and information retrieval tasks even with missing modalities. Experiments of this study showed that bi-modal representation of image and text significantly outperformed SVM and LDA on discriminative task and had gains over autoencoder and DBNs as well.

Similar to their previous work, the multimodal DBM was constructed using an image-text bi-modal DBM. The image pathway consists of a Gaussian RBM with 3857
visible units followed by 2 layers of 1024 hidden units. The text pathway consists of a Replicated Softmax Model with 2000 visible units followed by 2 layers of 1024 hidden units. The joint layer contains 2048 hidden units.

Each modality contained a set of visible units $v \in \{0, 1\}^D$, and a sequence of layers of hidden units $h^{(1)} \in \{0, 1\}^{F_1}, h^{(2)} \in \{0, 1\}^{F_2}, \ldots, h^{(L)} \in \{0, 1\}^{F_L}$. The visible-hidden interaction is achieved using the Gaussian RBM, and hidden–hidden was achieved using Binary RBM (Figure 14).

The image-specific two-layer DBM assigns probability to $v_m$ that is given by (ignoring bias terms on the hidden units for clarity):

$$P(v_m; \theta) = \sum_{h^{(1)}, h^{(2)}} P(v_m, h^{(1)}, h^{(2)}; \theta) =$$

$$= \frac{1}{Z(\theta)} \sum_{h^{(1)}, h^{(2)}} \exp \left( -\sum_{i=1}^{D} \frac{(v_{mi} - b_i)^2}{2\sigma_i^2} + \sum_{i=1}^{D} \sum_{j=1}^{F_1} \frac{v_{mi} - b_i}{\sigma_i} W_{ij} h^{(1)}_{j} + \sum_{j=1}^{F_1} \sum_{l=1}^{F_2} h^{(1)}_{j} W_{jl} h^{(2)}_{l} \right)$$

Figure 14. Left: Image-specific two-layer DBM that uses a Gaussian model to model the distribution over real valued image features. Middle: Text-specific two-layer DBM that uses a Replicated Softmax model to model its distribution over the word count vector. Right: A multimodal DBM that models the joint distribution over image and text inputs. [33]

To form a multimodal DBM, the two models were combined by adding an additional layer of binary hidden units on top of them. The joint distribution over multi-modal input can be written as:
The efficient approximate learning was performed using mean-field inference to estimate data-dependent expectations, and an MCMC based stochastic approximation procedure to approximate the model’s expected sufficient statistics. During the inference step, the true posterior \( P(h|v; \theta) \), where \( v = \{v_m, v_t\} \), with a fully factorized approximating distribution over the five sets of hidden units \( \{h^{(1)}_m, h^{(2)}_m, h^{(1)}_t, h^{(2)}_t, h^{(3)}\} \) was approximated:

\[
P(v^m, v^t; \theta) = \sum_{h^{(2)}_m, h^{(2)}_t, h^{(3)}} P(h^{(2)}_m, h^{(2)}_t, h^{(3)}) \left( \sum_{h^{(1)}_m} P(v^m, h^{(1)}_m, h^{(2)}_m) \right) \left( \sum_{h^{(1)}_t} P(v^t, h^{(1)}_t, h^{(2)}_t) \right)
\]

Each layer of hidden units in the DBM contributes a small part to the overall task of modeling the distribution over \( v_m \) and \( v_t \). In the process, each layer learns successively higher-level representations and removes modality-specific correlations. Therefore, the middle layer in the network can be seen as a (relatively) “modality-free” representation of the input as opposed to the input layers which were “modality-full” as in the case of the DBN models. In a DBN model the responsibility of the multimodal modeling falls entirely on the joint layer. In the DBM, on the other hand, this responsibility is spread out over the entire network.

The Multimodal DBM can be used to generate missing data modalities by clamping the observed modalities at the inputs and sampling the hidden modalities from the conditional distribution by running the standard alternating Gibbs sampler (Figure 15).
The MIR Flickr Data set was used in the experiments. The data set consisted of one million images along with their user assigned tags. The unlabeled 975,000 images were used only for pre-training, 15,000 images were used for training and 10,000 for testing. MAP was used as the performance metric. Each text input was represented using a vocabulary of the 2000 most frequent tags. The average number of tags associated with an image is 5.15 with a standard deviation of 5.13. About 18% of the labeled data had images but was missing text. Images were represented by 3857-dimensional features that were extracted by concatenating Pyramid Histogram of Words (PHOW) features, Gist and MPEG-7 descriptors (EHD, HTD, CSD, CLD, SCD). Each dimension was mean-centered and normalized to unit variance. The results of the experiments show that DMB-lab model (excluded SIFT-based features) outperformed SVM and LDA models by achieving MAP of 0.526, compared to 0.475 and 0.492, achieved by SVM and LDA models. DBM-unlab (model trained using unlabeled data during its pre-training stage) significantly improved upon DBM-Lab achieving a MAP of 0.585. DBM model (was trained using additional SIFT-based features) improved the MAP to 0.609. This model was compared to the DBN and a
deep Autoencoder. The DBN achieved a MAP of 0.599 and the autoencoder got 0.600, which is slightly worse than that of the DBM. In terms of precision@50 (precision at top 50 predictions), the autoencoder performed marginally better than the rest. In the experiment focused on the retrieval tasks the DBM model performed the best among the compared models achieving a MAP of 0.622. The autoencoder and DBN models performed worse with a MAP of 0.612 and 0.609 respectively.

In a study conducted by the Stanford AI Lab [32] which is led by Andrew Ng, multimodal learning and different settings for employing architecture of deep autoencoders is investigated to learn the multimodal data representation.

The study focuses on two modalities: speech audio and the video of the lips. The tasks are divided into three phases - feature learning, supervised training, and testing. Three learning settings are considered - multimodal fusion, cross modality learning, and shared representation learning. For the multimodal fusion setting, data from all modalities is available at all phases; in cross modality learning, one has access to data from multiple modalities only during feature learning. During the supervised training and testing phase, only data from a single modality is provided. These settings are explained in Figure 16.

<table>
<thead>
<tr>
<th></th>
<th>Feature Learning</th>
<th>Supervised Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic Deep Learning</td>
<td>Audio</td>
<td>Audio</td>
<td>Audio</td>
</tr>
<tr>
<td></td>
<td>Video</td>
<td>Video</td>
<td>Video</td>
</tr>
<tr>
<td>Multimodal Fusion</td>
<td>Audio + Video</td>
<td>Audio + Video</td>
<td>Audio + Video</td>
</tr>
<tr>
<td>Cross Modality Learning</td>
<td>Audio + Video</td>
<td>Audio</td>
<td>Audio</td>
</tr>
<tr>
<td></td>
<td>Audio + Video</td>
<td>Video</td>
<td>Video</td>
</tr>
<tr>
<td>Shared Representation Learning</td>
<td>Audio + Video</td>
<td>Audio</td>
<td>Video</td>
</tr>
<tr>
<td></td>
<td>Audio + Video</td>
<td>Video</td>
<td>Audio</td>
</tr>
</tbody>
</table>

Figure 16. Multimodal learning settings [32].

Bimodal deep autoencoder was trained in a denoising fashion, using an augmented dataset with examples that require the network to reconstruct both modalities given only one (Figure 17). The first layer features for video and audio
were extracted through 1536 and 1500 units respectively. The second layers have 4554 units. Both models are pre-trained using sparse RBMs (was used as a layer-wise building block for the model). This configuration makes it easy to compute the conditional probability distributions, when \( \mathbf{v} \) or \( \mathbf{h} \) is fixed (equation 5):

\[
-\log P(\mathbf{v}, \mathbf{h}) \propto E(\mathbf{v}, \mathbf{h}) = \frac{1}{2\sigma^2} \sum_i v_i^2 - \frac{1}{\sigma^2} \left( \sum_i c_i v_i + \sum_j b_j h_j + \sum_{i,j} v_i h_j w_{i,j} \right)
\]

\[
p(h_j | \mathbf{v}) = \text{sigmoid}(\frac{1}{\sigma^2}(b_j + \mathbf{w}_j^T \mathbf{v}))
\]

(5)

Gaussian visible units were used for the RBM that is connected to the input data. When training the deeper layers, binary visible units were used. The parameters of the model \((w_{ij}, b_j, c_i)\) were learned using CD. To regularize the model for sparsity, each hidden unit was encouraged to have a pre-determined expected activation using a regularization penalty of the form

\[
\lambda \sum_j (\rho - \frac{1}{m} (\sum_{k=1}^m \mathbb{E}[h_j | v^k]))^2,
\]

where \(\{v^1, ..., v^m\}\) is the training set and \(\rho\) determines the sparseness of the hidden units.

Figure 17. Deep autoencoder models. A “video-only” model is shown in (a) where the model learns to reconstruct both modalities given only video as the input. A similar model can be drawn for the “audio-only” setting. (b) shows a bimodal deep autoencoder trained in a denoising fashion. Using an augmented dataset with examples that require the network to reconstruct both modalities given only one. [32]

For the experiments, the video was preprocessed, so that the frames included only the region of interest - the mouth, which was rescaled to 60X80 pixels and
further reduced to 32 dimensions using PCA whitening. Temporal derivatives were computed over the reduced vector. The audio signal was preprocessed using its spectrogram with temporal derivatives resulting in a 483 dimension vector which was reduced to 100 dimensions using PCA whitening. For both modalities feature mean normalization over time was performed. Since only unlabeled data was required for unsupervised feature learning, diverse datasets were combined to learn features. All the datasets (AVLetters, AVLetters2, Stanford Dataset, TIMIT, CUAVE) were used for feature learning. AVLetters and CUAVE were further used for supervised classification. In the cross modality learning the deep autoencoder model performed the best, obtaining the classification score of 65.8% compared to RBM Video of 53.1%, bi-modal deep autoencoder of 59.2%. On the CUAVE dataset, the experiment showed an improvement by learning video features with both video audio compared to learning features with only video data. The deep autoencoder models performs the best, obtaining a classification score of 69.7% compared to RBM Video of 65.5% and Bi-modal Deep autoencoder of 66.7%. However, cross modality model did not help to learn better audio features. In the multimodal fusion settings the combination of the best audio features with the multimodal features of the autoencoder performed better than the simple concatenation of the audio and visual features, which gave results of 90.0% compared to 94.4%. In shared representation learning setting the ability to form a “shared” representation was tested by providing the algorithm data solely from one modality during the supervised training and testing later only on the other modality. Trained on audio and tested on video came to 29.4% and trained on video and tested on audio, 27.5% which shows that learned representation of the model has some invariance to the presented modality (Figure 18).
3.4. Comparisons and Discussions

A naive approach for multimodal deep learning is to concatenate the data descriptors from different input sources to construct a single high-dimensional feature vector and use it to solve a unimodal representation-learning problem. However, the correlation between features in each data modality is much stronger than that between data modalities. As a result, the learning algorithms are easily tempted to learn dominant patterns in each data modality separately while giving up learning patterns that occur simultaneously in multiple data modalities. To resolve this issue, deep learning methods, such as deep autoencoders or DBMs, have been adapted, where the common strategy is to learn joint representations that are shared across multiple modalities at the higher layer of the deep network, after learning layers of modality-specific networks. The rationale is that the learned features may have less within-modality correlation than raw features, and this makes it easier to capture patterns across data modalities. This has shown promise, but there still remains the challenging question of how to learn associations between multiple heterogeneous data modalities so that we can effectively deal with missing data modalities at testing time.
One necessary condition for a good generative model of multimodal data is the ability to predict reasons about missing data modalities given partial observation. Honglak Lee’s research group at the University of Michigan have proposed a new approach to satisfy this condition and improve multimodal deep learning [34]. There emphasis is on efficiently learning associations between heterogeneous data modalities. According to their study, the data from multiple sources are semantically correlated and provide complementary information about each other and a good multimodal model must be able to generate a missing data modality given the rest of the modalities. They propose a novel leaning framework that explicitly aims at this goal by training the model to minimize the Variation of Information (VI) instead of maximizing the likelihood.

The key idea behind the variation of information criteria is to minimize the information distance between modalities through the shared hidden representations, in other words, to learn to maximize the amount of information each modality has about the others. VI quantifies this amount of information about different modalities and is defined as follows:

\[ \text{VI}_Q(X,Y) = -\mathbb{E}_{Q(X,Y)}[\log Q(X|Y) + \log Q(Y|X)] \]

Where \( Q(X,Y) = P_{\theta}(X,Y) \) is any joint distribution on joint variables \((X,Y)\) parameterized by \( \theta \). Informally, VI is small when the conditional likelihood \( Q(X|Y) \) and \( Q(Y|X) \) are peaked, meaning that \( X \) has low entropy conditioned on \( Y \) and vice versa. Based on VI, the multimodal learning criteria proposed in this study, namely Minimum Variation of Information (MinVI) is defined as follows:

\[ \text{MinVI}: \quad \min_\theta \mathcal{L}^{VI}(\theta), \quad \mathcal{L}^{VI}(\theta) = -\mathbb{E}_{P_D(X,Y)}[\log P_{\theta}(X|Y) + \log P_{\theta}(Y|X)] \]

They apply this learning objective to the top shared layer of the deep network.

A good comparison between the three models we introduced in this section, plus the models used as their building blocks, is presented in Table 1 [35]. For each
model, the most important characteristics are explained, which makes it easier to compare and choose the most suitable model for a specific task.

Table 1. Comparison of basic deep learning models [35].

<table>
<thead>
<tr>
<th>Model</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep belief network (DBN)</td>
<td>Probabilistic generative models composed of multiple layers of stochastic, hidden variables. The top two layers have undirected, symmetric connections between them. The lower layers receive top-down, directed connections from the layer above.</td>
</tr>
<tr>
<td>Boltzmann machine (BM)</td>
<td>A network of symmetrically connected, neuron-like units that make stochastic decisions about whether to be on or off.</td>
</tr>
<tr>
<td>Restricted Boltzmann machine (RBM)</td>
<td>A special BM consisting of a layer of visible units and a layer of hidden units with no visible-visible or hidden-hidden connections.</td>
</tr>
<tr>
<td>Deep Boltzmann machine (DBM)</td>
<td>A special BM where the hidden units are organized in a deep layered manner, only adjacent layers are connected, and there are no visible-visible or hidden-hidden connections within the same layer.</td>
</tr>
<tr>
<td>Deep neural network (DNN)</td>
<td>A multilayer network with many hidden layers, whose weights are fully connected and are often initialized (pre-trained) using stacked RBMs or DBN. (In the literature, DBN is sometimes used to mean DNN)</td>
</tr>
<tr>
<td>Deep autoencoder</td>
<td>A DNN whose output target is the data input itself, often pre-trained with DBN or using distorted training data to regularize the learning.</td>
</tr>
</tbody>
</table>
4. Convolutional Neural Networks

4.1. CNN: Basic Model

CNNs [9] are biologically-inspired multi-layer neural networks specifically adapted for computer vision problems and visual object recognition. The idea of CNNs is similar to the mechanism of the visual cortex. We know that the visual cortex contains a complex arrangement of cells, which act as local filters and are sensitive to small sub-regions of the visual field, called the receptive field. The sub-regions are tiled to cover the entire image.

CNNs are characterized by three properties that define and highlight the advantages of these models. We will discuss these characteristics in more details shortly.

1. Local connectivity of the hidden units, meaning that each hidden unit is only connected to a small local region of the input image.
2. Parameter sharing, which means a lot of the hidden units will share parameters with each other.
3. The use of pooling and subsampling operations between the hidden layers.

Figure 19 shows the idea of local connectivity in CNNs. Each unit in a hidden layer is connected only to a small number of units from the previous layer. In the example shown in the figure, the previous layer that acts as the input to the hidden layer is the input image, so each hidden unit is connected to a small local area of image.

Each hidden unit is connected to all the channels in the local area of the image. The number of channels is one if the image is grayscale, and three if it is an RGB color image. This is shown in Figure 20.
The local connectivity property solves the following two problems:

1. Fully connected hidden layers would have an unmanageable number of parameters.
2. Computing the linear activations of the hidden units would be very expensive.

The idea of parameter sharing is that certain units of a CNN can share the matrix of weighted connections. The hidden units in a hidden layer are organized into a set of “feature maps” and the units within a feature map share the exact same parameters. These units cover different areas in the image (Figure 21).
Feature maps add two more benefits to the CNN model:

1. They reduce the number of parameters even more.
2. The units in a feature map extract the same features at every position in the input to which they are connected.

We can think of a feature map as a filter that detects a certain group of equivariant features everywhere in the input image. These features can be edges, corners, or more complex patterns.

The computation of the feature maps is equivalent to discrete convolution (*) of a kernel matrix with a receptive field from the previous layer. This is demonstrated in Figure 22.

Assuming:

- \( x_i \) is the \( i^{th} \) channel of the input,
- \( k_{ij} \) is the convolution kernel,
• \( g_j \) is a learned scaling factor,
• and \( y_j \) is the hidden layer,
then the activations of the hidden units in \( y_j \) is computed as follows:

\[
y_j = g_j \tanh \left( \sum_i k_{ij} \ast x_i \right)
\]

The kernel matrix \( k_{ij} \) is basically the hidden weights matrix \( W_{ij} \) with its rows and columns flipped. The tanh function is added for nonlinearity and can be replaced by any other nonlinear function such as sigm. Also, the constant \( g_j \) is not necessary in the definition of the CNN model, but has been used in the literature. We can also add a bias to the computation of \( y_j \), which will be unique shared by all the units across a feature map.

The third idea that characterized CNNs is the pooling and subsampling hidden units. Pooling basically takes a set of hidden units in a feature map and somehow aggregates the activations to obtain a single number. For instance, in max pooling, the aggregation will be computing the maximum among the activations on a certain neighborhood or receptive field. This is shown in Figure 23 and the following computations.

![Pooling / Subsampling](image)

**Figure 23.** Pooling and subsampling in CNN

\[
y_{ijk} = \max_{p,q} x_{i,j+p,k+q}
\]
Where

- $x_{i,j,k}$ is the value of the $i^{th}$ feature map at position $j,k$,
- $p$ is the vertical index in the local neighborhood,
- $q$ is the horizontal index in the local neighborhood,
- and $y_{i,j,k}$ is the pooled and subsampled layer.

Another alternative to max pooling is “average” pooling where $y_{i,j,k}$ is computed as follows:

$$y_{i,j,k} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q}$$

Where $m$ is the neighborhood height/weight.

Pooling and subsampling has two main benefits:

1. They introduce invariance to local translations,
2. and reduce the number of hidden units in hidden layers.

Now we can put together all the pieces and build a complete CNN. A CNN alternates between the convolutional and pooling layers (Figure 24). Note that the pooling windows could overlap, or be completely separate but adjacent. The output layer is a fully connected layer with softmax non-linearity and its size depends on the number of classes in the classification problem the CNN is supposed to solve. So the output vector will be an estimate of the conditional probability of each class given the input image.
The CNN is trained by stochastic gradient descent. Back-propagation is used similarly as in a fully connected neural network. The gradients are easily passed through the element-wise activation function, but we also need to pass gradients through the convolution and pooling/subsampling operation.

First, we compute the gradient of the convolution layer. Let $l$ be the loss function. For the convolution operation $y_j = x_i * k_{ij}$, the gradient for $x_i$ is:

$$\nabla_x l = \sum_j (\nabla_{y_j} l) * (W_{ij})$$

And the gradient for $W_{ij}$ is:

$$\nabla_{W_{ij}} l = (\nabla_{y_j} l) * (x^T_i)$$

Where $*$ is the convolution with zero padding and $x^T_i$ is the row/column flipped version of $x_i$.

The gradient for the pooling/subsampling layer is computed as follows. For max pooling operation $y_{ijk} = \max_{p,q} x_{i,j+p,k+q}$, the gradient for $x_i$ is $\nabla_{x_{ijk}} l = 0$, except for $\nabla_{x_{i,j+p,k+q}} l = \nabla_{y_{ijk}} l$, where $(p', q') = \text{argmax} x_{i,j+p,k+q}$. In other words, only the “winning” units in layer $x$ get the gradient from the pooled layer.
For average pooling operation \( y_{ijk} = \frac{1}{m^2} \sum_{p,q} x_{i,j+p,k+q} \), the gradient for \( x_i \) is
\[ \nabla x_{ijk} l = \frac{1}{m^2} \text{upsample}(\nabla y l), \]
where “upsample” inverts subsampling (copies the same value to every cell in the neighborhood).

### 4.2. Multimodal Deep Learning using CNNs

The various approaches that have been proposed for RGB-D object recognition can be divided into two categories from the perspective of feature generation: methods with hand-crafted features and methods with learned features. The handcrafted or learned features are typically fed into classifiers such as linear SVMs and random forests for the final classification.

In the first category, hand-crafted features such as SIFT, texton or color histogram are used to describe color and texture information of color images and 3D geometry information of depth images. The problem with hand-crafted features is that they do not readily extend to different datasets or other modalities, since they are often manually tuned for the conditions encountered in the datasets in mind. In addition, handcrafted features can only capture a subset of the cues that are useful for recognition.

In the second category, features are learned from raw data for RGB-D object recognition. Representative methods include convolutional-recursive deep learning, hierarchical matching pursuit, convolutional k-means descriptors, hierarchical sparse coding and local coordinate coding. Most of these methods either learn the features for individual modalities independently, or treat RGB-D simply as undifferentiated four-channel data, which cannot adequately exploit the complementary relationship between the two modalities.

To address the above issues, a general CNN based multimodal learning method for RGB-D object recognition is proposed in [36]. The basic idea of the proposed approach is illustrated in Figure 25. In particular, the authors build deep CNNs to learn feature representations for color and depth separately, which are then connected with a final multimodal layer. This layer is designed to not only discover
the most discriminative features for each modality, but also harness the complementary relationship between the two modalities. The results of the multimodal layer are back-propagated to update the parameters of CNNs and the multimodal feature learning and the back-propagation are iteratively performed until convergence. The accuracy of the model is approximately $86.7 \pm 2.7\%$.

Figure 25. Overview of the proposed general CNN-based multimodal learning framework in [36].

The major contribution of this work lies in the proposed multi-modal deep learning framework, which exploits the complementary information between different modalities, instead of just treating them as multi-channel input data or concatenating features independently learned from them. The proposed method is a general framework, which could be used for other multi-modal applications.

A few CNN-based multimodal approaches have been proposed in the emotion recognition in the wild (EmotiW) challenge during the past two years. The task in EmotiW is to assign one of seven emotions to short video clips extracted from Hollywood style movies. The videos depict acted-out emotions under realistic conditions with a large degree of variation in attributes such as pose and illumination.

The work presented in [37], which was a participant in EmotiW 2014, present a new approach to learning several specialist models using deep learning techniques, each focusing on one modality. Among these are a convolutional neural network, focusing on capturing visual information in detected faces, a deep belief net focusing
on the representation of the audio stream, a K-Means based “bag-of-months” model, which extracts visual features around the mouth region and a relational autoencoder, which addresses spatio-temporal aspects of videos. The pipeline of the method proposed in this work and the architecture of their ConvNet is depicted in Figure 26 and 27 respectively.

![Figure 26. The pipeline of the method proposed in [37]](image)

![Figure 27. The architecture of the ConvNet proposed in [37].](image)

Because of the small number of training samples, their initial experiments with ConvNets showed severe overfitting on the training set. For this reason, the authors decided to train on a separate dataset, which they refer to as “extra data”. Their proposed approach is divided into four stages:

1. Training the ConvNet on faces from extra data.
2. Extraction of 7-class probabilities for each frame of the facetubes (bounding boxes of successive frames for each subject in the video).
3. Aggregation of single frame probabilities into fixed-length video descriptors for each video in the competition dataset by expansion or contraction.
4. Classification of all video clips using a support vector machine (SVM) trained on video descriptors of the competition training set.
The best results they could achieve was 47.67% accuracy.

In another work presented at EmotiW 2015 [38], the authors follow a similar idea and combine learned and engineered features. Multimodal features including video (spatial-temporal), audio (temporal), and image (spatial) deep features are integrated with multi-kernel learning. The deep features are extracted from a pre-trained CNN that is fine-tuned using a dataset of extra imaged obtained from the web. The overall emotion classification accuracy is then improved by optimizing the multi-class SVM decision rules using a “prior” based on the number of samples in each class. More specifically, the SVMs that are trained on fewer samples are given higher scores, as they are “underrepresented”. The best performance they could achieve was 50.2%.

4.3. Comparisons and Discussions

Apart from tolerance to translation of the input images, one major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces required memory size and improves performance. Compared to other image classification algorithms, CNNs use relatively little pre-processing. This means that the network is responsible for learning the filters that in traditional algorithms were hand-engineered. The lack of a dependence on prior-knowledge and the existence of difficult to design hand-engineered features is a major advantage for CNNs.

Another benefit of CNNs is that features extracted in different layers of the network can be used as the input to other classification and recognition methods. These learned features are much more suitable for a specific task than classic engineered computer vision features, since they are adapted to the application as much as possible and are extracted automatically without any manual work.

All the above characteristics and benefits of CNNs have made them a powerful image processing/classification/recognition tool in recent years.

CNNs are somewhat similar to the three deep models introduced in previous section, but instead of learning single global weight matrix between two layers, they
aim to find a set of locally connected neurons. CNNs are mostly used in image recognition. Their name comes from "convolution" operator or simply "filter". In short, filters are an easy way to perform complex operation by means of simple change of a convolution kernel. Apply Gaussian blur kernel and you'll get it smoothed. Apply Canny kernel and you'll see all edges. Apply Gabor kernel to get gradient features.

The goal of CNNs is not to use one of predefined kernels, but instead to learn data-specific kernels. The idea is the same as with autoencoders or RBMs - translate many low-level features (e.g. image pixels) to the compressed high-level representation (e.g. film genres or edges) - but now weights are learned only from neurons that are spatially close to each other.
5. Conclusions and Discussions

In this survey, we summarized some of the most commonly used traditional shallow machine learning methods, which were often used as the baseline for comparison to recently developed deep architectures. Then, we thoroughly explained two important categories of deep learning models. The first group includes DBNs, DBMS and deep autoencoders, which are all build using RBMs as building blocks and are trained in a similar way. Secondly, we explained CNNs and their training methods. Here we offer some discussions in using deep learning models: the advantages and challenges, the problems in practice, the use of multimodalities, and the selection of software packages.

5.1. The Rise and Challenges of Deep Learning Models

Until a few years ago (around 2006), most machine learning and signal processing techniques were based on shallow-structured architectures. These architectures typically contain a single layer of nonlinear feature transformations and they lack multiple layers of adaptive non-linear features. Examples of the shallow architectures are conventional, commonly used GMMs, HMMs, LDA and SVMs. A property common to these shallow learning models is the relatively simple architecture that consists of only one layer responsible for transforming the raw input signals or features into a problem-specific feature space, which may be unobservable.

The features used as the input to these methods are generated based on traditional engineered feature extraction methods and are manually obtained from the raw data. This would make shallow architectures effective in solving many simple or well-constrained problems. But, their limited modeling and representational power can cause difficulties when dealing with more complicated real-world applications involving natural signals such as human speech, natural sound and language, and natural images and visual scenes. In such cases, methods
that are able to automatically extract task-specific features form the data are much more desirable. Deep learning models are one of the solutions.

The presence of local optima in the non-convex objective function of the deep networks is the main source of difficulties in the learning. Back-propagation, which is used in almost all deep learning models during the training phase, is based on local gradient descent and usually gets trapped in poor local optima, and the severity increases significantly as the depth of the networks increases. This difficulty is partially responsible for steering away most of the machine learning and signal processing research from deep networks to shallow models that have convex loss functions for which global optimum can be efficiently obtained at the cost of less powerful models. [39]

The optimization difficulty associated with the deep models was empirically alleviated when a reasonably efficient, unsupervised learning algorithm was introduced in [6,7], in which a class of deep generative models was introduced, namely DBN, which is composed of a stack of RBMs. A core component of the DBN is a greedy, layer-by-layer learning algorithm, which optimizes DBN weights at time complexity linear to the size and depth of the networks. Similar models have also been proposed, which include DBMs and deep autoencoders. All these models can be used as generative models, feature extractors, or classifiers/regressors.

Another class of deep models that are specifically developed for 2-D data, or essentially images, are CNNs. These models are also able to extract features, but are different with previous deep learning models in the sense that they function based on the convolution operator and are thus suitable for finding similarities between local, spatially independent areas of the images. Since CNNs and all other deep learning models are powerful feature extractors that do not need any hand-crafting except for adjusting the parameters, they can be combined with traditional classifiers in a way that the features extracted by those deep learning methods be the input to the shallow classifiers.
5.2. The Problems in Using Deep Learning Models

LDA and SVM have nice properties that have made them widely used in several classification applications. The advantages of SVM as an example of more traditional classification methods include high accuracy, nice theoretical guarantees regarding overfitting, and ability to classify non-linearly separable data with an appropriate kernel. Unfortunately, these algorithms are memory-intensive and the outputs are incomprehensible. Picking the right kernel could also be a challenge. Another issue is that these classifiers are essentially binary, meaning that they have no standardized way for dealing with multi-class problems. These issues lead us toward using deep learning methods since most of these issues are addressed in the implementation of deep learning algorithms. However, with all the capabilities and advantages of deep learning models over traditional machine learning methods, they have some shortcomings and limitations that need to take special care for successful applications:

1. **Overfitting:** The naïve approach when training a neural network is to present it with a huge dataset and train repetitively until the error drops below a certain threshold. Unfortunately, this is not the best way, as backpropagation is susceptible to overfitting the training data, becoming more used to the training data and losing accuracy on unseen examples (generalization). If not properly accounted for, overfitting can result in future iterations of training causing the network to lose accuracy, contradictory to the expected result of increased accuracy.

   One of the best methods for overcoming overfitting is simply to supply the network with a set of validation data that is disjoint from the training set [49]. The algorithm can then check against the validation data on every iteration and measure the error variance. The number of iterations that produces the lowest error rate is then used for the network weights. However, one must be careful with this approach though, as errors can fluctuate on earlier iterations.
A different way to fix this is to use a technique known as weight decay [50]. Weight decay works by decreasing each weight by a small factor on every iteration, i.e. introducing a penalty term to the error of the network. This works by keeping weight values small, so that the network is biased towards learning complex decision surfaces. The decay is biased against large weights - i.e. large weights are penalized more than small.

Another method to combat overfitting is to probabilistically update the weights on each layer with the effect that only a certain percentage of the nodes on each layer get updated. This method is known as dropout [40] and it results in weights being learned in a slower, more controlled manner and that objective function learned ends up being smoother than it otherwise would have been.

Finally, it's possible to augment the dataset by adding noise or rotating, mirroring and scaling [41] the source images for CNNs. This makes the network more robust to noise. Noise that it's likely to meet in the real world from imperfect test examples.

2. **Small datasets:** When dealing with a dataset of limited size, there are a few methods available to re-sample for a better average. One method is to partition the set into k disjoint, equal sized subsets [51]. From here, each example is used once in a test set, and k-1 times in a training set.

   Another method to help eliminate general error is to randomly choose a test set of at least 30 examples (the lower bound for reliable testing) for validation [49], and use the remaining set members for training, and repeat several times. From this, we can then calculate average values and shrink the confidence interval to the desired width.

3. **Network sizes:** In order to perform well on complex tasks with a DNN, we often have to have networks that have a large number of parameters e.g. Le et al. [42] which had ~1 billion. This not only causes problems with overfitting, but also introduces a time penalty for both training and classifying. Whilst overfitting can be tackled using the techniques
mentioned above, we still remain with no reduction in the number of parameters we have to keep track of. This means that we require large server clusters and many gigabytes (if not terabytes) of RAM to keep track of these parameters. Although it’s possible to train a neural network using distributed computing [43], ideally we want to keep the number of servers used small so that we don't have to deal with the risk of a computer on the network failing and causing us to possibly lose data, which would be disastrous on a training algorithm that is running for weeks. The result here is that we must invest in large servers in order to train our networks reliably, hence more motivation for reducing the sizes of networks so that we don't need as much computing power to train or use them.

We should always be looking for ways to reduce the sizes of the networks we create. Early work from LeCun et al. [44] showed that we could selectively delete nodes from a trained network to reduce the size of the network whilst retaining, or even improving, its accuracy and error rate. More recently, dropout [40] has been used in a similar manner to create multiple thinned networks during training, which can then be combined together to increase the performance of the network. Romero et al. [45] created FitNets, a way to reduce the size of the final network by making a large, wide DNN train a DNN with fewer nodes layer by layer.

Smaller networks can be made that perform just as well if not better than larger nets on the same problem. However, these approaches incur a time penalty during training and require thinning down from a larger network in the first instance.

4. **They can be fooled:** Despite the state of the art results we have been granted by using DNNs, they do have a major flaw. A recent paper [46] detailed how DNNs can be fooled by using an evolutionary algorithm tested against a well-trained DNN at each stage. This process was then continued until the image became unrecognizable to a human.
On the other side, Szegedy et al. [47] showed that barely perceptible changes to the image could cause a DNN to completely misclassify an image. This is not just a random error caused by the particular data that was used by the DNN during training. The same change to the same image caused a similar misclassification by a DNN trained on a different subset of the data.

5.3. Multimodalities in Deep Learning Models

In most real-world applications, dealing with multimodal data is inevitable due to the nature of the task. This requires machine learning methods that are capable of efficiently combining their learned knowledge from multiple modalities. In traditional machine learning methods such as SVM, multimodal learning is performed by training a separate SVM on each individual modality and combining the results by voting, weighted average or other probabilistic methods.

A very important aspect of multimodal learning that is missed in these approaches is the ability of learning the association between different modalities. This can be easily achieved by utilizing deep learning methods, as they are capable of extracting task-specific features from the data and learning the relationship between modalities through a shared representation. This shared representation of the data which reveals the association between different modalities makes the trained structure a generative model. That is, the model would be able to handle missing modalities as long as the relationship between the absent modality is efficiently learned by the model. This happened during the unsupervised training phase, in which unlabeled multimodal data is used to learn a joint “modality-free” representation of the data. The model will then be able to Infer missing modalities given the observed ones. The high-level method for filling in the missing modalities has three main steps:

1. Building a joint density on all modalities,
2. using states of top-level hidden units as joint representation, and
3. sampling from the conditional density to fill in the missing modality.
We also reviewed the literature on the application of multimodal learning models including both traditional and deep learning methods. Due to their advantages, deep multimodal models outperform traditional models in tasks where the data has hidden structural characteristics that need to be extracted in order to efficiently model the data.

5.4. Software Packages of Deep Learning Models

Many deep learning software packages have been developed by different groups [48]. In 2015 KDnuggets Software Poll, a new category for Deep Learning Tools was added, with most popular tools in that poll listed below:

1. **Theano and Pylearn2**: Theano and Pylearn2 are both developed at University of Montreal with most developers in the LISA group led by Yoshua Bengio. Theano is a Python library, and you can also consider it as a mathematical expression compiler. It is good for making algorithms from scratch.

   If we want to use standard algorithms, we can write Pylearn2 plugins as Theano expressions, and Theano will optimize and stabilize the expressions. It includes all things needed for multilayer perceptron/RBM/Stacked Denoting Autoencoder/ConvNets (CNNs).

2. **Caffe**: Caffe is developed by the Berkeley Vision and Learning Center, created by Yangqing Jia and led by Evan Shelhamer. It is a fast and readable implementation of ConvNets in C++. As shown on its official page, Caffe can process over 60M images per day with a single NVIDIA K40 GPU with AlexNet. It can be used like a toolkit for image classification, while not for other deep learning application such as text or speech.

3. **Torch and OverFeat**: Torch is written in Lua, and used at NYU, Facebook AI lab and Google DeepMind. It claims to provide a MATLAB-like environment for machine learning algorithms. OverFeat is a feature
extractor trained on the ImageNet dataset with Torch7 and also easy to start with.

4. **Cuda:** There is no doubt that GPU accelerates deep learning researches these days. News about GPU especially Nvidia Cuda is all over the Internet. Cuda-convnet/CuDNN supports all the mainstream softwares such as Caffe, Torch and Theano and is very easy to enable.

5. **Deeplearning4j:** Unlike the above packages, Deeplearning4j is designed to be used in business environments, rather than as a research tool. As on its introduction, DL4J is a “Java-based, industry-focused, commercially supported, distributed deep-learning framework.”
References

artificial intelligence and statistics. NP: Society for Artificial Intelligence and Statistics, 2005.