E. Alpaydın (2018). "Classifiying Multimodal Data," In S, Oviatt, B. Schuller, P. Cohen, D. Sonntag, G. Potamianos, and A. Krueger, editors, The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition. Chapter 2, pp. 49-69, Morgan & Claypool Publishers, San Rafael, CA

2 Classifying Multimodal Data

Ethem Alpaydin

2.1 Introduction

Multimodal data contains information from different sources/sensors that may carry complementary information and, as such, combining these different modalities intelligently improves accuracy in classification tasks. We are going to discuss three approaches: (1) in ***early integration***, features from all modalities are concatenated as one long input and a single classifier is trained; (2) in ***late recognition***, a separate classifier is trained with each modality, which independently makes a decision and then the decisions are combined, and (3) ***intermediate integration*** is between these two extremes—there is a single classifier but it is trained with some suitably processed, a more abstract version of the input from each modality. We consider two possibilities for this: one uses the ***multiple kernel learning*** framework where each modality has its own specific kernel and the other is multimodal ***deep neural networks*** where processing of different modalities are separate in early layers but are combined later on. For each approach, we will discuss the pros and cons in a comparative manner. We conclude that in building classifiers that combine modalities, there is no single best method and one needs to think about the level of abstraction where correlation is expected to occur between features and choose a combiner accordingly.

2.2 Classifying Multimodal Data

The idea of having an ***ensemble*** of ***learners*** and combining their predictions is not a new idea in pattern recognition and ***machine learning***, and there is a variety of methods; see Kuncheva [2004] for a comprehensive review. See the Glossary

for key terminology related to ***machine learning*** and ***ensemble*** models. For such a combination to be useful, two questions become important [Alpaydın 2014]:

1. Since it does not make sense to have multiple learners that make the same mistakes, how can we generate learners that are diverse/independent/uncorrelated so that they make errors on different cases and complement each other?
2. When we have these multiple learners each making a prediction on a test instance, how can we combine their predictions to calculate the overall output for highest possible accuracy?

These two questions are not necessarily orthogonal but it may be useful to answer them one by one. For the first question of *how to generate different learners*, there have been a number of alternatives.

- The most popular approach is to use different learning algorithms or models. Each learning algorithm or model makes a different set of assumptions about the data and picking one algorithm corresponds to one set of assumptions. Learning is an ill-posed problem, that is, the data by itself is not sufficient to get a unique model. The set of assumptions we need to make to get a unique solution is called the *inductive bias* for the model or the algorithm. For example, the linear discriminant assumes that the classes are linearly separable in the input space and the k-nearest neighbor assumes that nearby instances are likely to have the same label. Hence, to make a safer bet and not "put all our eggs in the same basket," we choose a number of algorithms/models that we believe are likely to perform well and combine their predictions. From the perspective of experiment design, we can view the learning algorithm as a (controllable) factor that affects the accuracy of the final classifier and using many algorithms corresponds to averaging over the many levels of this factor. Averaging over different models also averages out the effect of random (uncontrollable) factors; for example, neural networks are trained with gradient-descent that is randomly initialized and averaging over multiple neural networks decreases dependence on initialization.
- One can use the same learning algorithm or model, but with different hyper-parameter settings. Each model has a hyper-parameter that allows its complexity to be adjusted to the task; for example, with multi-layer perceptrons, it is the structure of the network, i.e., the number of hidden layers and the number of hidden units; with the k-nearest neighbor classifier, it is k, that is,

Glossary

In **machine learning**, the **learner** is a model that takes an input x and learns to give out the correct output y. In **pattern recognition**, typically we have a classification task where y is a class code; for example in face recognition, x is the face image and y is the index of the person whose face it is we are classifying.

In building a learner, we start from a data set $\mathcal{X} = \{x^t, r^t\}, t = 1, \ldots, N$ that contains training pairs of instances x^t and the desired output values r^t (e.g., class labels) for them. We assume that there is a dependency between x and r but that it is unknown—If it were known, there would be no need to do any learning and we would just write down the code for the mapping.

Typically, x^t is not enough to uniquely identify r^t; we call x^t the observables and there may also be unobservables that affect r^t and we model their effect as noise. This implies that each training pair gives us only a limited amount of information. Another related problem is that in most applications, x has a very high dimensionality and our training set samples this high dimensional space very sparsely.

Our prediction is given by our predictor $g(x^t|\theta)$ where $g()$ is the model and θ is its set of parameters. Learning corresponds to finding that best θ^* that makes our predictions as close as possible to the desired values on the training set:

$$\theta^* = \arg\min_{\theta} \sum_{t=1}^{N} L(r^t, g(x^t|\theta))$$

$L()$ is the **loss function** that measures how far the prediction $g(x^t|\theta)$ is from the desired value r^t. The complexity of this optimization problem depends on the particular $g()$ and $L()$. Different learning algorithms in the machine learning literature differ either in the model they use, the loss function they employ, or the how the optimization problem is solved.

This step above optimizes the parameters given a model. Each model has an inductive bias that is, it comes with a set of assumptions about the data and the model is accurate if its assumptions match the characteristics of the data. This implies that we also need a process of *model selection* where we optimize the model structure. This model structure depends on dimensions such as (i) the learning algorithm, (ii) the hyper-parameters of the model (that define model complexity), and (iii) the input features and representation, or modality. Each model corresponds to one particular combination of these dimensions.

An **ensemble** is a set of models and we want the models in the set to differ in their predictions so that they make different errors. If we consider the space defined by the three dimensions that define a model as we defined above, the idea is to sample smartly from that space of learners. We want the individual models to be as accurate as possible individually, and at the same time, we want them to complement each other. How these two criteria affect the accuracy of the ensemble depends on the way we do the combination.

Glossary *(continued)*

From another perspective, we can view each particular model as one noisy estimate to the real (unknown) underlying problem. For example, in a classification task, each base classifier, depending on its model, hyper-parameters, and input features, learns one noisy estimator to the real discriminant. In such a perspective, the ensemble approach corresponds to constructing a final estimator from these noisy estimators—for example, voting corresponds to averaging them.

When the different models use inputs in different modalities, there are three ways in which the predictions of models can be combined, namely, early, late, and intermediate combination/integration/fusion.

In **early combination**, the inputs from all the different modalities are concatenated and fed to a single model. In *late combination*, for each modality there is a separate model that makes a prediction based on its modality, and these model predictions are later fused by a combining model.

In **intermediate combination**, each modality is first processed to get a more abstract representation and then all such representations from different modalities are fed together to a single model. This processing can be in the form of a **kernel function**, which is a measure of similarity, and such an approach is called **multiple kernel learning**. Or the intermediate processing may be done by one or more layers of a neural network, and such an approach corresponds to a **deep neural network**.

The level of combination depends on the level we expect to see a **dependency** between the inputs in different modalities. Early combination assumes a dependency at the lowest level of input features; intermediate combination assumes a dependency at a more abstract or semantic level that is extracted after some processing of the raw input; late combination assumes no dependency in the input but only at the level of decisions.

the number of nearest neighbors taken into account, and so on. Using multiple copies of the same model but with different hyper-parameter values—for example, combining three multi-layer perceptrons one with 20, one with 30, and one with 40 hidden units, again corresponds to averaging over this factor of the hyper-parameter.

- Another approach to generate learners that differ in their decisions is by training them on different training data. This can be done by sampling from the same training set and in bagging [Breiman 1996] we use bootstrap which is sampling at random with replacement; in adaboost [Freund and Schapire 1996], sampling is done sequentially and is driven by error where the next

learner is trained on instances misclassified by the one before. In the mixture of experts model [Jacobs et al. 1991], there is a gating network that divides the input space into regions and each learner (expert) is trained only with the data falling in its region of expertise.

- Yet another approach is to train different models on different random feature subsets [Ho 1998]. In the random forest, for example, each decision tree sees only a random subset of the original set of features—the different subsets can overlap. Different learners see slightly different versions of the same problem; some of the features may be noisy and some may be redundant, and combining over multiple learners averages out the effect of this factor of the feature set.
- But the most promising approach, and it is the topic of this chapter, seems to be training the different learners/models using data coming from different modalities. Such data from different sensor sources provide different representations of the same object or event to be classified, and hence can carry information that has the highest chance of being diverse or complementary. In machine learning literature, this is also known as *multi-modal, multi-view, multi-representation,* or *multi-source* learning.

  The earliest example is in speech recognition, where the first modality is the acoustic signal captured by a microphone and the second modality is the visual image of the speaker's lips as they speak. Two utterances of the same word may differ a lot in terms of the acoustics (e.g., when the speaker is a man or a woman), but we expect their lips to move similarly; so accuracy can be improved by taking this second visual modality into account as well. Incorporating this new type of sensor, here a camera for visual input, provides a completely new source of information, and this is the power of multimodality—adding the visual source to acoustics can improve the accuracy much more than what we would get if we combined multiple learners all looking at slightly different versions of the same acoustic data; see Chapter 1 of this volume [Baltrusaitis et al. 2018] for various examples of multimodal settings.

  When there are multiple modalities, the immediate approach would be to concatenate features of different modalities to form one long vector and use a single learner to classify it (early integration) but, as we will see shortly, feeding different modalities to different learners (late integration) or feeding them to a single learner after some preprocessing (intermediate integration) may work better.

When it comes to the second question of *how to combine/integrate/fuse the predictions of multiple learners,* again, there are different possibilities.

- The most intuitive, and the most frequently used approach is *voting*: Each learner looking at its input votes for one class, we sum up the votes for all classes and choose the class that receives the highest vote. In the simplest case, the votes are binary 0/1: a classifier votes 1 for the class that it believes is most likely, and majority voting chooses the class that gets the highest number of votes.

  Let us say $g_{ij}(x) \in \{0, 1\}$ is the output of model $i = 1, \ldots, m$ for class $j = 1, \ldots, k$: $g_{ij}[x)$ is 1 if model i votes for class j and is 0 otherwise. The total vote for class j is

$$y_j = \sum_{i=1}^{m} g_{ij}(x) \tag{2.1}$$

  and we choose class l if

$$y_l = \max_{j=1}^{k} y_j \tag{2.2}$$

  This is known as *majority voting*.

  Certain classifiers can generate outputs that indicate their belief in classes; for example, some classifiers estimate class posterior probabilities, and in such a case, a classifier gives soft votes (e.g., in $[0, 1]$) for all classes, indicating the strength of the vote. Frequently, these soft votes are nonnegative and sum up to 1 (as we have with posterior probabilities) or are normalized to be so before any combination. In such a case, using Equations (2.1) and (2.2) implies the *sum rule* where we choose the class that gets the maximum total soft vote:

$$l = \arg\max_{j} \sum_{i} g_{ij}(x). \tag{2.3}$$

  This is the most straightforward rule for combination and an equivalent is the *average* rule where we choose the class that gets the highest average vote. Other possibilities are the *median, minimum, maximum,* and *product* rules each of which has its use in particular circumstances [Kittler et al. 1998]. For example, the median rule

$$l = \arg\max_{j} \operatorname{median}_{i} g_{ij}(x) \tag{2.4}$$

makes more sense than the average rule when we have a large number of possibly unreliable voters whose votes may be noisy outliers.

Regardless of whether the votes themselves are binary or continuous, one can use simple or weighted voting:

$$l = \arg\max_j \sum_i w_i g_{ij}(x). \tag{2.5}$$

In the simplest case when we have no a priori reason to favor one learner over another, we use simple voting where all learners have the same weight: $w_i = 1/m$. We use weighted voting when for some reason or another "some are more equal than others." For example, one learner may have higher accuracy on a left-out validation set and hence we may want to give it more weight. In this case, with the sum rule, we calculate the weighted sum and then choose the class with the maximum total weighted vote. We generally require these weights to be non-negative and sum up to 1: $w_i \geq 0$, $\sum_i w_i = 1$.

We can also interpret weighted voting from a Bayesian perspective. We can write:

$$P(C_j|x) = \sum_{i=1}^{m} P(M_i)P(C_j|M_i, x). \tag{2.6}$$

Here, $P(C_j|M_i, x)$ is the estimate of the posterior probability for class C_j by model M_i given input x. We cannot integrate over the whole space of possible models, so we sample m likely models and take the average of their predictions weighted by the model probabilities $P(M_i)$.

- In *stacking*, this task of combination is viewed as another learning task [Wolpert 1992]. The learners to be combined are named the L_0 learners and we have the L_1 learner whose task is to predict the correct class given the predictions of L_0 learners as its input:

$$y_j = f(g_{1j}(x), \ldots, g_{mj}(x)|\psi). \tag{2.7}$$

Here, $g_{ij}(x)$ are the L_0 base learners and $f()$ denotes the combining L_1 learner with its own parameters ψ. Note that L_1 does not see the original input x, it only learns to correctly combine the predictions of L_0 learners. Typically, L_0 learners and the L_1 learner are trained on different data sets because L_1 needs to learn when, and how, L_0 learners succeed or fail in predicting the correct class.

When L_1 is a linear model, stacking works just like weighted voting except that weights are learned; they need not be positive nor sum up to 1 (although we can constrain them to do so if we want). But the nice property of stacking is that L_1 can be *any* model—for example, L_1 can be a multi-layer perceptron or a decision tree—thereby allowing a nonlinear combination of learner outputs implying a much more powerful combination than voting.

- The *mixture of experts*, which we mentioned above, can also be seen as a variant of weighted voting where the weight of a learner is dynamic. There is a gating model that also sees the input and its output are the combination weights—the gating model works like a probabilistic classifier and its task is to assign the input to the expert that it believes is the right model to make decision for it [Jacobs et al. 1991]. Learners are given different weights by the gating model depending on the input; a learner is given the highest weight in its region of expertise.

  We generalize Equation (2.5) as

$$l = \arg\max_j \sum_i w_i(x) g_{ij}(x), \tag{2.8}$$

  where $w_i(x), i = 1, \ldots, m$ are the outputs of the gating model calculated for input x.

  In the hierarchical mixture of experts, this partitioning of the input space among experts is done hierarchically [Jordan and Jacobs 1994]. One can view this as a soft tree where gating models act as decision nodes and experts are the leaves, so the final output is again calculated as a weighted sum but propagated from the leaves to the root level by level.

- *Cascading* differs from the approaches above in the sense that the learners do not work in parallel but in series [Alpaydın and Kaynak 1998, Kaynak and Alpaydın 2000]. They are ordered and the input is first given to the first classifier, it makes a prediction for a class, and if it is confident in its output (e.g., if the highest posterior is greater than a certain threshold) we use that output, otherwise the input is fed to the second classifier, which in turn makes a prediction and we check if it is confident, and so on. The classifiers are ordered in terms of complexity so stopping early decreases the overall complexity. In a multimodal setting, the classifiers may be ordered according to the cost of sensing these different modalities, so we do not pay for a costly modality if the earlier cheaper ones suffice for confident classification.

$$l = \begin{cases} \arg\max_j g_{ij}(x) & \text{if } g_1(x) \text{ is confident} \\ \arg\max_j g_{2j}(x) & \text{if } g_2(x) \text{ is confident} \\ \vdots & \\ \arg\max_j g_{mj}(x) & \text{otherwise} \end{cases} \tag{2.9}$$

These approaches for model combination, namely voting, stacking, and so on, are typically used for combining learners that use different algorithms, hyperparameters, and/or training data. In the rest of this chapter, we will discuss how these model combination approaches are applied when we have multimodal data.

2.3 Early, Late, and Intermediate Integration

Let us say we have input from different modalities and we want to make a decision using information coming from all the modalities. We denote the input in modality i as the $d^{(i)}$ dimensional vector $x^{(i)}$ where $i = 1, \ldots, m$ and m is the number of modalities. Here, we assume that each $x^{(i)}$ is available at once; there are also integration approaches where inputs are sequences; see Chapter 3 of this volume [Panagakis et al. 2018].

The most straightforward approach is *early integration* where we concatenate all these vectors to form a single $x = (x^{(1)}, x^{(2)}, \ldots, x^{(m)})$ which is a $d = \sum_{i=1}^{m} d^{(i)}$ dimensional vector. We train a single classifier with this concatenated x (see Figure 2.1(a)):

$$y = g(x^{(1)}, x^{(2)}, \ldots, x^{(m)}|\theta). \tag{2.10}$$

Here, y is the output and $g()$ is the model defined up to a set of parameters θ. The advantages are that we train and use a single learner, and after concatenation we can use any learning algorithm to learn the classifier.

But there are also two risks here. First, the concatenated input dimensionality may be very high and this causes two type of problems. (i) With more inputs, the classifier becomes more complex, in terms of space (more memory) and time (more computation), and hence higher input dimensionality implies higher implementation costs. (ii) Because the model gets more complex with more parameters, we also need more training data, otherwise there is a higher risk of overfitting; this is called the curse of dimensionality.

The second and more important risk associated with early integration is that these features from different modalities come from different sources, their units and scales are different; they are like apples and oranges. Hence the joint space define by their concatenation can be a very strange space and the class distributions

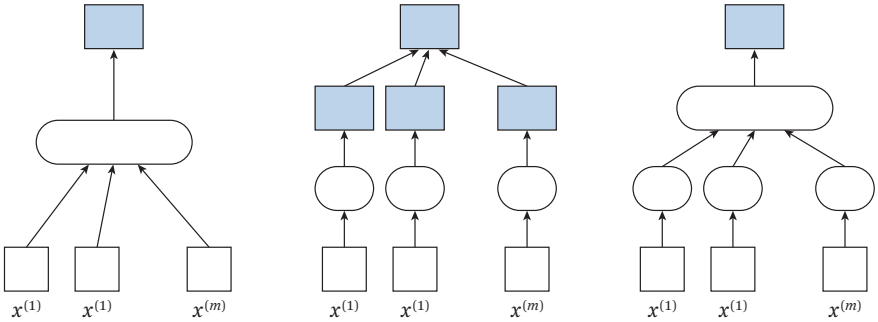

Figure 2.1 (a) Early, (b) late, and (c) intermediate (deep) integration drawn as multi-layer networks. Squares are inputs in different modalities (each of which may be a vector), ovals are extracted (hidden) features and shaded rectangles are predicted outputs. Each oval can be one or more hidden layers.

in there can be very difficult to model. For example, consider a scenario where we have image and speech; in such a case, some of the dimensions are pixels and some are frequency coefficients; using dot product or Euclidean distance to compare their concatenations does not make much sense.

Early integration may also lead to a problem of alignment—when we have a sequence of observations in each modality, it may be tricky to know which ones should be used together; see Chapter 3 of this volume [Baltrusaitis et al. 2018].

One clear application of early integration is when if for each modality we have a representation with very few features that provide only limited information. For example, when we are doing credit scoring, we have age, gender, profession, salary, and so on; those are actually different modalities, but each by itself is not sufficient to make a prediction with, so we concatenate them and consider the whole as a four-dimensional vector. Concatenating and feeding them together may also allow finding correlations between them—typically age and salary are positively correlated.

But if for each modality, we have a representation that is long and detailed enough, giving us enough information for prediction and we do not expect to see much correlation between features of different modalities, we prefer to use *late integration* where we have a separate learner for each modality that is trained with the input in its corresponding representation. For example, for user authentication, given the face image and speech, we have one classifier that looks at the image to

make a decision and another that looks at the speech to make a decision and the outputs of both classifiers are in the same scale and mean the same, e.g., both may be class posteriors, and hence fusing them makes sense.

Given a test instance in m modalities, each learner, independently and in parallel, makes its prediction using its own modality and then we combine their decisions (see Figure 2.1(b)):

$$y = f\left(g_1(x^{(1)}|\theta_1), g_2(x^{(2)}|\theta_2), \ldots, g_m(x^{(m)}|\theta_m) \mid \psi\right), \tag{2.11}$$

where $g_i(x^{(i)}|\theta_i)$ is model i with its parameter θ_i taking input in modality i. As the combining model $f()$, one can use any of the combining methods discussed before, i.e., voting, stacking, mixture of experts, or cascading, and ψ are the parameters, e.g., weights in voting, the parameters of the L_1 model in stacking, and so on, also trained on some data.

In combining such separately trained models, in practice, we see that no matter how much we play with the learning algorithms, hyper-parameters, or any other factor that affects the trained model, classifiers turn out to be positively correlated—they tend to make similar decisions. Depending on how the modalities are defined, such a correlation may also occur when we have multimodal combination. This has two consequences. First, if we have two models that are highly correlated, we do not need both; we can just keep one and discard the other. The second consequence is that when we have correlated models, having more models actually decreases accuracy; a larger ensemble is more accurate only if the voting models are independent. Both of these consequences indicate the need for post-processing an ensemble to reduce correlations.

One approach is subset selection [Ulaş et al. 2009]: If we have m models, we want to choose a smaller subset of size $k < m$ without losing from accuracy. The algorithms proposed for this are basically the same as the ones we use for feature extraction: if m is small, we can do an exhaustive search of all possible subsets, otherwise we perform a greedy search with hill-climbing where we start with the empty set and we add one learner at a time, adding the one that increases the accuracy the most, until no further addition improves accuracy any further. We can also do a backward search where we start with all models and remove one at a time until one more removal drastically worsens performance, or we can do a floating search that allows both additions and removals. When we use a subset instead of all, we save the space/time complexity of the pruned learners and in case where they use inputs from different modalities with associated costs, we also save the cost of sensing the modalities that turn out to be unnecessary.

The other approach is to post-process outputs to remove correlations. This is reminiscent of feature extraction algorithms, such as principal components analysis, where we define new features in terms of the original features, e.g., by linear projection. Indeed, in the approach of eigenclassifiers, we define new features taking into account the correlations between classifiers [Ulaş et al. 2012].

Easy integration combines at the lowest level of input features and late integration combines at the highest level of output predictions. *Intermediate integration*, as its name suggests, is between these two extremes of early and late integration. First, for each modality, there is some processing done to convert the raw input to some more abstract representation and then these are fed together to a classifier. That is, there is a single learner but it is trained with some abstract version of the input from each modality (see Figure 2.1(c)):

$$y = g(z^{(1)}, z^{(2)}, \ldots, z^{(m)}|\theta), \tag{2.12}$$

where $z^{(i)}$ is a processed version of $x^{(i)}$. We discuss below two variants, one using multiple kernels and the other one using deep neural networks.

2.4 Multiple Kernel Learning

In a kernel machine, such as the support vector machine, we write the class discriminant in terms of ***kernel functions*** [Cortes and Vapnik 1995]. A kernel function is a measure of similarity between two vectors, one of which is the input and the other is a training instance (on or inside the margin, or on the wrong side of the discriminant), named a support vector in the support vector machine algorithm. The kernel function implicitly defines a basis space that these vectors are mapped to and are compared in: $K(x, y) = \phi(x)^T \phi(y)$. That is, $K()$ returns a value which is equal to the dot product of the images of the two vectors in the space of the basis function $\phi(\cdot)$. Every valid kernel corresponds to a different basis space.

In kernel machines, the most critical model selection problem is the choice of the appropriate kernel. The good kernel calculates a good similarity measure between instances so that, for example in classification, the similarity between two instances of the same class is larger than the similarity between instances of different classes. In the typical case where instances are represented as vectors, kernels typically use the dot product or its variants, such as the polynomial kernel, or the Euclidean distance or its variants, such as the Gaussian kernel.

But one of the attractive properties of the kernel approach is that we do not need to have our inputs represented vectorially. We can define kernels starting directly with similarities. That is, if we have some application-specific similarity measure

that we can apply to pairs of instances, we can define a kernel in terms of it. So if we have some complex data structure such as a graph or a document, we do not need to worry about how to represent it in terms of a vector, as long as we can come up with some similarity measure to compare two graphs or documents. For documents, for example, the need for a vectorial representation led to the bag of words representation which has various disadvantages; it may be easier to directly define a function to compare two documents for similarity. This advantage also holds for the multimodality case: it may be easier to define a similarity measure for a modality instead of generating a vectorial representation and then using a kernel in terms of such vectors.

The analog of multiple learners in kernel machines is multiple kernels: just like we have different learning algorithms to choose from, in kernel machines we have different kernels available. Typically, we do not know beforehand which one is the most suitable and the typical approach is to try different kernel functions and choose the best (e.g., by checking accuracy on a left-out validation data set), treating kernel selection as a model selection problem. The other possibility is to combine those kernels; this is called *multiple kernel learning* [Gönen and Alpaydın 2011]. The idea is that each kernel is a different measure of similarity and we use a set of candidate measures and write the discriminant as a combination of such similarities, again averaging out the effect of this factor.

This multiple kernel learning framework can easily be applied to the multimodal setting where we have one kernel for each modality. The simplest and most frequently used approach is a linear combination:

$$K(x, y) = \sum_{i=1}^{m} w_i K_i(x^{(i)}, y^{(i)}), \tag{2.13}$$

where $x^{(i)}, y^{(i)}$ are the representations of two instances x, y in modality i and $K_i(x^{(i)}, y^{(i)})$ is the kernel measuring their similarity by kernel i according to that modality.

The weights $w_i, i = 1, \ldots, m$ are trained on labelled data [Lanckriet et al. 2004, Sonnenburg et al. 2006]. Frequently, they are constrained to be nonnegative, and sometimes also to sum to 1. This helps interpretation—a higher w_i implies a more important kernel and hence a more important modality. If the kernel weights are nonnegative, such a combination corresponds to scaling and concatenation of the underlying feature representations $\phi_i(x)$—this implies a combination similar to early integration but in the space of the basis functions.

Various variants of the multiple kernel learning framework have been proposed (Gönen and Alpaydın [2011] is a survey), including also nonlinear combinations. One variant is local combination where w_i are not constant but are a function of the input, effectively working as a gating model choosing between kernels depending on the input [Gönen and Alpaydın 2008]; such an approach can also be viewed as the kernelized version of the mixture of experts.

2.5 Multimodal Deep Learning

Recently, deep neural networks have become highly popular in a variety of applications. A neural network is composed of layers of processing units where each unit takes input from units in the preceding layer through weighted connections. The unit then calculates its value after this weighted sum is passed through a nonlinear activation function. Given an input, the processing proceeds as the units calculate their values layer by layer until we get to the final output layer.

The network structure, i.e., the number of layers, the number of units in each layer, and the way the units are interconnected, define the model and the weights of the connections between the units are the parameters. Given a training set of pairs of input and the desired output, the weights are updated iteratively to make the actual outputs in the output layer as close as possible to the desired outputs.

If there is no a priori information, layers are fully connected among them. In applications where the input has locality, connectivity is restricted to reflect dependencies; for example, in vision applications, the input is a two-dimensional image and in a *convolutional* layer, a hidden unit sees only a small local patch of the inputs. With data where there is temporal ***dependency***, a *recurrent* connection allows a hidden unit to take into account not only the current input but also its value in the previous time step. In certain network structures, there are *gating* units, as we have in the mixture of experts, that allow or not the value of a unit to propagate through. A judicious use of all these type of units and connectivity makes neural networks quite powerful in a variety of applications.

The idea in a neural network is that each hidden layer after the input layer learns to be a feature detector by responding to a certain combination of values in its preceding layer, and when we have a network with many such layers, i.e., in a deep neural network, successive layers learn feature detectors of increasing abstraction. A deep learning model in its many layers extract increasingly higher-level and abstract set of features and this allows a better representation of the task and hence improves accuracy [Bengio 2009, Goodfellow et al. 2016].

The most important advantage of neural networks is that such a model makes very simple assumptions about the data and because it is a universal approximator, it can learn any mapping. The disadvantage is that because the model is general, we need large amount of data to constrain it and make sure that it learns the correct task and generalizes well to data outside of the training set.

Another advantage of neural networks is that calculations are local in hidden units and parallel architectures such as GPUs can be efficiently programmed to handle the computation in a neural network with significant speed-up.

We can view each hidden layer of neural network as learning a kernel and when we have many such hidden layers in a deep network, it is as if we are learning increasingly abstract kernels calculated in terms of simpler kernels. The advantage is that the kernels are not defined a priori but are learned from data; the disadvantage is that the optimization problem is non-convex and we need to resort to stochastic gradient-descent with all its concomitant problems.

In learning the weights of the feature-detecting hidden units, the earlier approach was to use the autoencoder model [Cottrell et al. 1987] where the output is set to be equal to the input and the hidden layer in between has fewer hidden units. The hidden layer hence acts as a bottleneck and learns a compressed and abstract representation with minimum reconstruction error. The autoencoder model can also be trained to be robust to missing inputs [Vincent et al. 2008]. Roughly speaking, we can view the hidden representation learned in the autoencoder as the $\phi_i(\cdot)$ basis of kernel $K_i()$ in kernel machines. We can then stack such autoencoders to generate a deep neural network with multiple hidden layers. The autoencoder model has the advantages that first, it can be trained with unlabeled data, and, second, learning is fast because we train one layer at a time.

Deep architectures have also been used to combine multiple modalities. The idea is to first train separate autoencoders for each modality and then learn to combine them across modalities by training a supervised layer on top (see Figure 2.1(c)). This is an example of intermediate combination defined in Equation (2.12) where the earlier modality-specific layers learn to generate the $z^{(i)}$ which are then fused in the later layer(s) (denoted by $g()$ with its weights ψ).

Nowadays with large labeled data sets and processing power available, end-to-end deep neural networks are trained directly in a supervised manner, bypassing the training of autoencoders altogether. Because the whole training is supervised and all the parameters are trained together, we can achieve higher accuracy, but the disadvantage is that training multiple layers using stochastic gradient-descent is slow and one needs to use regularization methods such as dropout to make sure that the large network does not overfit.

If we use early combination and just concatenate features from different modalities and feed them to a single network, feature-extracting (hidden) units have strong connections to a single modality with few units connecting across modalities; it is not possible to correlate basic features from one modality with basic features of another modality. But if both are separately processed using separate hidden units (trained either as an autoencoder or end-to-end) to get a higher-level and more abstract representation, these extracted features can be combined to learn a shared representation and a classifier that uses such a cross-modality representation has higher accuracy. This has been shown to be true in combining audio and lip image features for speech recognition [Ngiam et al. 2011].

A similar approach and result also holds for image retrieval where in addition to image data, there are also text tags [Srivastava and Salakhutdinov 2012, Srivastava and Salakhutdinov 2014]. For each modality, there is a separate deep network whose hidden units learn the abstract features that are modality specific; then the two such abstract representations can be combined in a set of features and we can for example use such a network to map one modality into another, so that, for example, given an input image, the network can generate a set of candidate tags, or given a set of tags, the network can find the best matching image.

In training the shared features that we mention above, that combine raw features from different modalities, different unsupervised criteria can also be used. Additional to minimization of the reconstruction error, one can also use variation of information [Sohn et al. 2014] or canonical correlation analysis [Andrew et al. 2013, Wang et al. 2015].

Multimodal deep networks can also be trained to combine similarities. Separate autoencoders learn modality-specific representations and instead of using them as vectors, a similarity measure is applied to each and their weighted sum is calculated to get an overall similarity [Wu et al. 2013]. This approach is very similar to multiple kernel learning where we take a weighted sum of kernels (which are also measures of similarity); see also McFee and Lanckriet [2011]. See Keren et al. [2018] for an extensive survey of deep learning methods for multi-sensorial and multimodal interaction.

2.6 Conclusions and Future Work

Combining multiple models to improve accuracy is an approach frequently used in pattern recognition and machine learning. Mostly it is used to average out the effect of factors such as the learning algorithm, hyper-parameters, or randomness in the data or in initialization. Combining multiple modalities promises to bring

significant improvement in accuracy because data from different modalities have the highest chance of providing complementary information about the object or event to be classified.

To combine multiple modalities, there are different possibilities that we have outlined above, and the right one, that is, the level where combination is to be done, depends on the level of abstraction where correlation is expected to occur between the different modalities.

If features in different modalities are correlated at the feature level, one can use early combination by just concatenating all the modalities and feeding it to a single classifier. This is the easiest and fastest approach and works well if its assumptions hold and if there are not many features in total.

But if the data in different modalities are correlated at a more abstract semantic level, one can use intermediate integration. For example, if we have an image and a set of tags, no individual image pixel is correlated with any tag, but the existence of a lot of blue patches in the upper half of the image may be correlated with the tag "sky." To find such an abstract representation in each modality, one can use modality-specific hidden layers to extract it from data. By suitably combining and stacking such representations, a deep neural network can be trained. In some applications, we may know a good measure of similarity in advance for each modality which we can write down as a kernel, and a smart combination of kernels is another way to combine modalities. One big advantage of kernels is that one can define a similarity between instances without necessarily generating a vectorial representation and using a vectorial kernel.

Late combination is used when there is no correlation at any level in the input, neither at the lowest level of input features nor after any amount of feature extraction. For example, in biometrics, if we have face image and fingerprint image, there is no correlation between the pixels of the two images, nor can we extract any correlation between any higher-level features extracted separately from these images. The only correlation is at the level of labels—whether they belong to the same person or not. In such a case, we can only do late integration where we classify the two images separately and combine their decisions.

The take away message of this chapter should be that in building classifiers that combine modalities, there is no single best method and one needs to think about the level where correlation is expected to occur and choose a combiner accordingly.

Multimodal classification and learning is a relatively recent idea but we expect to see it applied more in the future with a wider availability of divers sensors in many modalities. Mobile devices, smart objects, and wearable sensors detect and record data in different modalities [Neff and Nafus 2016]. Each such device

or sensor provides a partial clue from a different modality, but combining them higher precision may be attained. With advances in digital technology and all types of smart online objects with their sensors—the Internet of Things [Greengard 2015]—appearing in the market, multimodal combination will only become more important in the future.

Acknowledgments

This work is partially supported by Boğaziçi University Research Funds with Grant Number 14A01P4.

Focus Questions

2.1. Consider the level of dependency between modalities in example applications, and for each, which combination—early, intermediate, or late is appropriate.

2.2. Consider the average of identically distributed g_i:

$$y = \sum_{i=1}^{m} g_i / m.$$

Show that (a) the variance of y is minimized when g_i are independent, and (b) the variance of y increases when g_i are positively correlated.

2.3. In some studies on multimodal deep learning, researchers split each digit image into two, as left or right halves, or top and bottom halves, and process them as if they are two different modalities. Discuss the suitability of this approach for testing intermediate integration.

2.4. A kernel machine uses fixed kernels but defines a convex problem, which we can solve optimally. A multi-layer perceptron is trained using stochastic gradient-descent that converges to the nearest local minimum but its hidden units can be trained. Discuss the pros and cons of the two in the context of multimodal classification.

2.5. Some researchers have proposed methods for learning good kernel functions from data. Discuss how such a method can be used in multiple kernel learning in the context of multimodal classification.

2.6. In a multimodal deep learner, some layers learn features that are specific to a modality, and some learn features across modalities. How can we decide how many layers to have for each in a deep neural network?

2.7. In training a deep neural network with many hidden layers, the earlier approach was to train autoencoders one layer at a time and then stack them; nowadays, however researchers prefer to train the whole network end-to-end. Discuss the advantage and disadvantages of the two approaches.

2.8. In this chapter, we discussed methods for multimodal classification. Discuss how these can be adapted for multimodal regression and multimodal clustering.

References

E. Alpaydın. 2014. *Introduction to Machine Learning*. 3rd edition The MIT Press. 50

E. Alpaydın and C. Kaynak. 1998. Cascading classifiers. *Kybernetika*, 34(3): 369–374. 56

G. Andrew, R. Arora, J. Bilmes, and K. Livescu. 2013. Deep canonical correlation analysis. In *International Conference on Machine Learning*, pp. 1247–1255. 64

T. Baltrusaitis, C. Ahuja, and L.-Ph. Morency. 2018. Multimodal machine learning. In S. Oviatt, B. Schuller, P. Cohen, D. Sonntag, G. Potamianos, and A. Krueger, editors, *The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition*. Chapter 1, Morgan & Claypool Publishers, San Rafael, CA. 53, 58

Y. Bengio. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2: 1–127. DOI: 10.1561/2200000006. 62

L. Breiman. 1996. Bagging predictors. *Machine Learning*, 26: 123–140. DOI: 10.1023/A:1018054314350. 52

C. Cortes and V. Vapnik. 1995. Support vector networks. *Machine Learning*, 20: 273–297, 1995. DOI: 10.1023/A:1022627411411. 60

G. W. Cottrell, P. Munro, and D. Zipser. 1987. Learning internal representations from gray-scale images: An example of extensional programming. In *Ninth Annual Conference of the Cognitive Science Society*, pp. 462–473. 63

Y. Freund and R. E. Schapire. 1996. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pp. 148–156. 52

M. Gönen and E. Alpaydın. 2008. Localized multiple kernel learning. In *International Conference on Machine Learning*, pp. 352–359. DOI: 10.1145/1390156.1390201. 62

M. Gönen and E. Alpaydın. 2011. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12: 2211–2268. 61, 62

I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*. MIT Press. 62

S. Greengard. 2015. *The Internet of Things*. Essential Knowledge Series. MIT Press. 66

T. K. Ho. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20: 832–844. DOI: 10.1109/34.709601. 53

R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. 1991. Adaptive mixtures of local experts. *Neural Computation*, 3: 79–87. DOI: 10.1162/neco.1991.3.1.79. 53, 56

M. I. Jordan and R. A. Jacobs. 1994. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2): 181–214. DOI: 10.1162/neco.1994.6.2.181. 56

C. Kaynak and E. Alpaydın. 2000. Multistage cascading of multiple classifiers: One man's noise is another man's data. In *International Conference on Machine Learning*, pp. 455–462. 56

G. Keren, A. E. Mousa, and B. Schuller. 2018. Deep learning for multi-sensorial and multi-modal interaction. In S, Oviatt, B. Schuller, P. Cohen, D. Sonntag, G. Potamianos, and A. Krueger, editors, *The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition*. Chapter 4, Morgan & Claypool Publishers, San Rafael, CA. 64

J. Kittler, M. Hatef, R. P. W. Duin, and J. Matas. 1998. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20: 226–239. DOI: 10.1109/34.667881. 54

L. I. Kuncheva. 2004. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley. 49

G. R. Lanckriet, G. N. Cristianini, P. Bartlett, L. ElGhaoui, and M. I. Jordan. 2004. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5: 27–72. 61

B. McFee and G. Lanckriet. 2011. Learning multi-modal similarity. *Journal of Machine Learning Research*, 12: 491–523. 64

G. Neff and D. Nafus. 2016. *Self-Tracking*. Essential Knowledge Series. MIT Press. 65

J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng. 2011. Multimodal deep learning. In *International Conference on Machine Learning*, pp. 689–696. 64

Y. Panagakis, O. Rudovic, and M. Pantic. 2018. Learning for multi-modal and context-sensitive interfaces. In S, Oviatt, B. Schuller, P. Cohen, D. Sonntag, G. Potamianos, and A. Krueger, editors, *The Handbook of Multimodal-Multisensor Interfaces, Volume 2: Signal Processing, Architectures, and Detection of Emotion and Cognition*. Chapter 3, Morgan & Claypool Publishers, San Rafael, CA. 57

K. Sohn, W. Shang, and H. Lee. 2014. Improved multimodal deep learning with variation of information. In *Advances in Neural Information Processing Systems 27*, pp. 2141–2149. 64

S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. 2016. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7: 1531–1565. 61

N. Srivastava and R. Salakhutdinov. 2012. Multimodal learning with deep Boltzmann machines. In *Advances in Neural Information Processing Systems 25*, pp. 2222–2230. 64

N. Srivastava and R. Salakhutdinov. 2014. Multimodal learning with deep Boltzmann machines. *Journal of Machine Learning Research*, 15: 2949–2980. 64

A. Ulaş, M. Semerci, O. T Yıldız, and E. Alpaydın. Incremental construction of classifier and discriminant ensembles. *Information Sciences*, 179: 1298–1318, 2009. DOI: 10.1016/j.ins.2008.12.024. 59

A. Ulaş, O. T. Yıldız, and E. Alpaydın. 2012. Eigenclassifiers for combining correlated classifiers. *Information Sciences*, 187: 109–120. DOI: 10.1016/j.ins.2011.10.024. 60

P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*, pp. 1096–1103. DOI: 10.1145/1390156.1390294. 63

W. Wang, R. Arora, K. Livescu, and J. Bilmes. 2015. On deep multi-view representation learning. In *International Conference on Machine Learning*, pp. 1083–1092. 64

D. H. Wolpert. 1992. Stacked generalization. *Neural Networks*, 5: 241–259. DOI: 10.1016/S0893-6080(05)80023-1. 55

P. Wu, S. C. H. Hoi, H. Xia, P. Zhao, D. Wang, and C. Miao. 2013. Online multimodal deep similarity learning with application to image retrieval. In *ACM International Conference on Multimedia*, pp. 153–162. DOI: 10.1145/2502081.2502112. 64