Reduce clinical image by deep learning

# Introduction

Image based clinical diagnosis is well practiced for decades, typical examples are chest radiographic for the lung cancer diagnosis and cerebral MRI for neurodegeneration and Alzheimer's disease. Despite being powerful confirmation tools for under-suspect disease which already exhibit clinical signs and symptoms, image reading is hardly applicable for mass screening and accurate prediction required by primary and secondary prevention (pre-onset and early intervention), because 1) image analysis relies little on objective measurement (e.g. systolic blood pressure, IgA counter, etc.) but mostly on physician’s subjective knowledge and experience; 2) subtle patterns, even if distinctly associated with disease of interest or its early stage of development, is hardly recognizable with naked eyes.

To solve the screening and prediction problem we must replace the subjective human labor with an objective measure (e.g. risk score or dichotomous prediction) derived from the image, which must be sufficiently accurate, also reasonably cheap and fast. Naturally this leads to machine learning – also aliased artificial intelligence (AI), expert system or decision making system. More specifically, we are mentioning **supervised** learning which takes in a series of feature-outcome pairs as training materials to learn a model best fit the training material. The model will later be asked to predict disease by giving it features from new subject of undiagnosed disease status.

# Supervised machine learning

Putting supervised machine learning in symbols, the goal is to find the best set of parameters ***θ*** for the disease predictor ***f***:

, that minimized the discrepancy between predicted ***y*i’** and the truth ***y*i**, over all subjects (***i***=***1***…***n***), measured by some prediction loss function ***L***(***y***i, ***y*i**').

. Here ***x*i** denotes observed features of the subject under question; ***y*i’** denotes predicted outcomes of the *i*th subject given its feature ***x*i**; ***y*i** denotes the true outcome of the same subject (also called label) determined by golden standards (e.g. disease onset or mortality); **θ'** is an approximation of the best **θ** we are looking for, which is done by optimization procedures (e.g. stochastic gradient descent (**SGD**)[\*]).

The choice of prediction loss measure L could be the traditional squared error ***L***(***y*i**, ***y*i’**)=|| ***y*i***-* ***y*i’**||2. If the outcome is binary probabilities of being a future case or control, cross-entropy of the predicted probability is frequently used, which resembles one half of the deviance of logistic regression model:

Popular forms of supervised machine learning are logistic regression, neuron-network and multiple layered perceptron (MLP), in the order of increasing complexity and non-linearity.

The trained model ***f*** that best fits the training tuples is not guaranteed to yield enough accuracy when faces with new data in the future. The extreme case is fitting a logistic model with the number of parameters exactly the same with the number of observations to reach perfect fit on the training set(***θ*'**= ***θ***) but close to random guess when new data is given. During regression analysis, statisticians usually prefer simpler models with fewer parameters for better stability. The same idea is applied to machine learning in a more generalized manner that the complexity of the trained model is measured by a regulator function R based on the overall magnitude of parameters in **θ**, the usual form is L2 norm:

. Here ***K*** is the number of parameters in the model. The rationale is that prediction outcome of a model with small parameters is not likely to be sensitive to trivial changes of the input due to noises, therefore, minimization of the regulator ***R*** is equivalent to the preservation of model stability and generalizability. More ever, a model with many of its parameter close to zero is even preferred because the projection of input on the latent coding layer would be sparsely distributed, which helps higher layers to pick robust features if we are to stack a deep autoencoder. Incorporate the regulator R into the search for the best ***θ*** is thus equivalent to solve the following arithmetic minimum:

, where ***λ*** is used to tune the balance between best fit of training data and model simplicity.

## Dimensionality challenge

Although supervised learning has produced disease diagnostic models of plausible sensitivity and specificity, the image data poses server dimensionality challenge. A radiograph or a single MRI slice usually comprises thousands to millions pixels, the raw features is far more numerous and redundant than a dozen of obvious clinical signs and symptoms or a few dozen questions in a scoring sheet. To even train a runnable predictor who takes image as input, we must first reduce the dimensionality of **x** by extracting a handful of important, non-redundant features from the raw data (pixels in a radiograph, voxels in a MRI cubic).

One well known method is principle component analyses (PCA), but it doesn’t take the non-linearly correlation between raw features into consideration. In this study we will use unsupervised machine learning, more specifically, to exploit the capability of denoising auto-encoders – one in the framework of the unsupervised learning, to compress the huge and redundant image information while preserving distinct features. In the near future, the compressed output will in turn serve as a low dimension input for the learning of mainstream models (e.g. logistic regression, MLP).

# Auto-encoders

An auto-encoder takes an input x in [0, 1]d and first maps it (with an encoder) to a hidden representation z in [0,1]d' through a deterministic mapping:

, where **s** is a non-linearity mapping, the usual choices are sigmoid or tanh (hyperbolic tangent). What’s interesting here is z, the latent representation of input feature x, whose dimensionality **d’** is supposedly of much lower than the input dimension **d** in order to achieve compression and feature extraction, but for a single layered auto-encoder this is usually not the case. To significantly compress the input, we need a deep auto-encoder formed by stacking multiple basic auto-encoders, in which the first few layers closer or directly linked to the raw feature **x** usually have dimensions higher than **d,** but sparsely distributed such that the correlated features gather and the independent ones separate in the space spanned by coding bits (e.g. the binary units in layer z) so upper layers can easily pick these major, correlated features. Although not explicitly mentioned, a practical auto-encoder must also make sure the latent representation y is decodable in a fashion similar to the encoding process. The decoding is done by mapping **z** back into a reconstructed **x’**:

(The prime symbol does not indicate matrix transpose) Optionally, the weight matrix **W’** of the reverse mapping may be constrained to be the transpose of the forward mapping: **W’**= **WT**. This is referred to as tied weights. The parameters of this model **θ**=(**W**, **b**, **b’,** **W’**) are iteratively optimized such that the average reconstruction error ***L***(see below) is minimized.

## Unsupervised VS supervised

Here we could relate unsupervised machine learning to the supervised by viewing the former as a special case of the later, where the model to be learned must contain at least one hidden layer(e.g. the compressed coding **z**), and more importantly, the truth component ***y*** of the supervised training tuples (**x**, **y**) is not specified (hence unsupervised), but replaced with the input feature **x** itself so the feature-truth tuples become (***x***, ***x***), and the feature-prediction tuples become repectively. Similar to how we access the prediction losses for supervised learning, we access the reconstruction loss of unsupervised learning by looking into the discrepancy between original feature ***x*** and the reconstructed ***x*’** with loss function ***L***.

Like supervised machine learning, the loss function ***L*** could come in a number of forms depending on the appropriate distributional assumptions on the input ***x*** given the code ***z***. The traditional squared error ***L***(***x***, ***x*’**)=|| ***x*** *-* ***x*’**||2 can be used. Nowadays however, cross-entropy of the reconstruction is more popular, if we interpret the reconstruction as element-wise binary prediction of the input feature:

. Here ***d*** is the dimension of both the input feature and the reconstruction.

## Rationale

The hope is that the latent coding **z** is a distributed representation that captures the coordinates along the main factors of variation in the data in a similar manner of projecting input features on the principal components to capture the main factors of variation in the data[\* citation]. As a matter of fact, if we force the hidden coding layer z to be linear, that is, we pick **s**(**u**)=**cu** where c is a fixed scalar, and use mean squared error to train the network, that is, we pick **L**(**x**, **x’**)=||**x** - **x’**||**2**, then the **d’** hidden units will learn to project input feature **x** in the span of the first **d’** principal components of the data. If the hidden layer is non-linear, the auto-encoder behaves differently from PCA, with the ability to capture multi-modal aspects of the input distribution. The departure from PCA becomes even more important when we consider stacking multiple encoders (and their corresponding decoders) when building a deep auto-encoder [Hinton06].

Because **y** is viewed as a loss-compression of sampled **x**, it cannot be a good (small-loss) compression for all input drawn from arbitrary data generating distribution. Optimization makes it a good compression for training examples, and hopefully for other inputs of similar distribution as well. That is the sense in which an auto-encoder generalizes: it gives low reconstruction error on test examples from the same distribution as the training examples, but generally high reconstruction error on samples randomly chosen from the input space.

# Denoising Autoencoder (DA)

As mentioned before, auto-encoders closer or directly linked to the raw data usually has its dimension **d’** higher than of the input. If no constraint besides minimizing the reconstruction error is specified, one might expect an auto-encoder with inputs of dimension **d** and an encoding of dimension **d**’ > **d** to learn the identity function, merely mapping an input to its copy. Such an auto-encoder would not differentiate training set from other input of different dimensionality and thus possess no generalizability at all.

Surprisingly, experiments reported in [Bengio07] suggest that, in practice, when trained with stochastic gradient descent (SGD), non-linear auto-encoders with more hidden units than inputs (overcomplete) yield useful representations in the coding layer **z**. A simple explanation is that both implicit regularization imposed by early stopping rule of SGD, and explicit regularization imposed by the term **R** prevented the parameters in W reach great values need for an identical mapping from **x** to **x’**.

Yet, if we wish to surely prevent an auto-encoder with more hidden units than the dimension of input feature(**d**’ > **d**) from simply approximating a trivial identity mapping, and also looking for better generalizability of the model, we could first randomly destroy part of the input then pump the corrupted version through a basic auto-encoder describe above [Vincent et al. 2008]. The whole process is symbolized into the following steps, the first step is to corrupt the input feature by means of a stochastic mapping **qd**:

(**corruption**)

. The usually behavior of **qd** is randomly choosing a pre-specified percentage from **d** elements and set them to 0. The rest of the steps resemble much the same with a basic auto-encoder, the next would be mapping the corrupted data to the latent coding space(the hidden layer **z**):

(**encoding**)

. The reconstruction of input and error assessment follows:

(**decoding**)

(**reconstruction** **loss**)

## Rationale

An intuitive way of understanding the benefit of partially destroying the input is that, only the strong, distinctive and consistent feature across the same class of subject will more likely to survive the random damage dealt by **qd**. Given one of the purposes of unsupervised learning being the extraction of major non-linear variations, the stochastic damage of the data actually helped reaching this goal by forcing the learning process to notice more robust feathers instead of falling for trivial details.

One experiment to demonstrate the effect of randomly alter the data is shown below, the original image was put through a filter randomly assign noises to the input, such damage actually act as an edge detector, since edges are much harder to be destroyed than trivial features.



Notice that a randomly assigned noise acts very much like an edge detector

The de-noising auto-encoder is a stochastic version of the auto-encoder. It tries to recover a set of damaged variable from the rest of undamaged ones, which can only be done by capturing the statistical dependencies between the input features. [Vincent08] [Bengio09].

# Stacked deep autoencoder

Denoising autoencoders can be stacked to form a deep network. Besides the lowest one directly taking raw features as input to its latent coding layer (z1), the coding layer of the jth encoder (**zj**) is fed with the coding output from the encoder found below (zj-1). The unsupervised training of such architecture is done one layer at a time. Each layer is trained as a denoising autoencoder by minimizing the reconstructing loss of its input - the latent representation of raw feature **x** given by encoders from below (or x itself if we are training the first encoder). In other words, to train the jth encoder, we must train the previous j-1 ones since we need the latent representation from the j-1 layers below.

The symbolic representation of the stacked denoising autoencoders is:

(**the last**, ***Jth* encoding**)

(**the 2nd last**, ***J-1th* encoding**)

… … (higher above)

(**j**th **encoding**)

… …

(2nd **encoding**)

(1st **encoding**)

(**raw** **feature**)

Here we represent the raw input ***x*** as trivial identity encoding ***z*0** of itself to unify the symbolic representations; ***J*** denotes the number of basic autoencoders to be stacked together, which also indices the top most encoder; ***zj*** denotes the latent representation of original ***x*** in the jth autoencoder, or the jth layer of the stack; **θj**= (**Wj**, **bj**) denote the weight and bias parameters used to transform the j-1th into the jth latent representation of **x**, which is to be trained by layer; denote the stochastic corruptor employed to partially destroy input of the ***k***th encoder(or output from the ***k***-1th layer), which takes in data of dimension ***d*k** to do some damage; ***d*k** denotes the dimension of the latent representation in the ***k***th autoencoder, when ***k***=0, this means the dimension of raw feature ***x***.

The training of the jth encoder is symbolized as the following:

(**jth encoding**)

(**jth decoding**)

(**jth loss**)

(**jth optimization**)

Here subscript ***i*** denotes the ***i***th training sample, ***k*** denote the ***k***th dimension within the latent representation computed in the ***jth*** encoder; ***θj'*** is the approximation of the best parameter set ***θj*** for the jth encoder that minimizing the weighted sum of ***jth*** reconstruction loss ***L***(***z***j-1,***z***’j-1) and the ***jth*** regulation penalty ***R***(***θj***).

A deep denoising autoencoder is capable of reducing the dimension of initial input x by gradually reduce the dimensionality of the later staked encoders, that is, by setting **d0** < **d1** <**d2** … **dk-1** > **dk** > **dk+1** … > **dJ**., and **d0** >> **dJ**., the deep encoder formed by **J** basic encoders has its dimensionality start to drop from the kth layer and above, and reach an finally representation of the initial input x with significantly lowered dimensionality [\*].

# Fine-tune deep autoencoder

Despite every basic autoencoder in the stack had approximated optimal configuration of its own set of parameters ***θj*=**(*Wj*, *Wj*’, *bj*, *bj*’), the deep encoder built by stacking these encoders is not at optimal configuration of all the parameters ***θ*=**(***θ1***, ***θ2***… ***θJ-1***, ***θJ***), where ***J*** is the number of encoders in the stack. Thus we must fine tune the whole stack for the best setup of all parameters **θ**. The fine turn can be done by both supervised and unsupervised learning.

## Pre-train deep autoencoder

In fact, we are free to build the stack and start training the whole structure even without separately training each layer, the problems is the difficulty of convergence due to the vast number of parameters in ***θ*** (the first few layers could even have more units than the dimension of raw input). Things become especially ugly if we attempt to directly tune the deep encoder by unsupervised learning on high dimensional materials (e.g. image data), because the reconstruction come out of the top must share the same matrices with the input at the bottom (figure \*), the **Jacobian** of reconstruction loss ***L*** with respect to input ***x*** will have ***d2*** elements, which is excessively complex and severely slowdown or even hamper the convergence. Therefore, before tuning the whole structure, it would be wise to optimize each single floor of autoencoders, which is called pre-training in [\* et al, 2006]. The purpose of pre-training is to setup initial values for all parameters that are closer to optimal; therefore much less time is need by SGD to achieve convergence.

## Supervised tuning

If the training data is labeled (i.e. the true outcome ***y*** is available), after structuring the deep encoder, we pipe the final, deep encoding (hopefully of much lower dimension then ***x***) from the top of the stack into a typical supervised learner. The supervised learner serves as upper structure, which could be a logistic regression model, or a more complex neuron network with hidden layer, or even a MLP as deep as the stacked encoder below. By symbolic representation, the final prediction *yi’* of the *i*th training tuple (*xi, yi*) will be:

(**Upper structure, supervised learning**)

(**Lower structure, deep encoding**)

(**Whole structure**)

, where ***θ*** denotes the pre-trained parameters of the deep encoder that is to be fine-tuned, and ***β*** denotes the parameters in the upper structure to be trained by the supervised learning; ***ziJ*** denote the deep encoding of the raw feathers of the *i*th subject, retrieved from the top of the deep autoencoder; ***J*** is the number of basic encoders stacked in the deep encoder, which also indices the top one.

To fine-turn ***θ*** and to learn ***β***, we need to define prediction loss and regulator penalty to be minimized. The definition is similar to typical supervised learning:

(**L2 norm regulator**)

(**binary** **prediction** **loss**)

(**optimization**)

## Unsupervised tuning

To fine-tune the deep autoencoder in unsupervised manner, we first build the upper structure by stacking all the decoders from last(***z*’J-1**) to first(***z*0’**) in reversed order, such that ***z*’J-1** goes to bottom of the upper structure and accept the deep encoding from the top most encoder,while ***z*0’** goes to the very top of everything and its output serves as final reconstruction ***x***’ (see Figure\*). ***x***’ is to be compared with the initial input ***x***(***z***0) located at the very bottom of everything to assess the overall reconstruction loss of the whole stack.

Use symbolic representation, the deep decoder(the upper structure) is stacked as the following:

(**1st decoding, reconstruction**)

(**2nd decoding**)

… …

(j**th decoding**)

… …

(**2nd last, *J*-1th decoding**)

(**last, *J*th decoding**)

. We denote the above chinned deep decoders with one mapping:

(**deepdecoding**)

As an analogy of supervised fine-tuning, ***β*** denote all parameters involved in the upper decoding structure:

(**deep decoder parameters**)

Together with parameters ***θ*** involved in the lower, deep encoder structure, both ***θ***, ***β*** are to be fine-tuned by minimizing reconstruction loss. We get the final unsupervised deep learner by piping the output of lower deep encoder into the upper deep decoder. The reconstruction loss and regulation penalty can be formulated similar to a basic denoising autoencoder:

(**Upper structure, deep decoding**)

(**Lower structure, deep encoding**)

(**L2 norm regulator**)

(**reconstruction** **loss**)

(**optimization**)

Here ***xi*** is the ***i***th training example(the outcome ***yi*** is not required); ***J*** is the number of basic autoencoders forming the deep structure; (***θ'***,***β'***) is an approximation of best parameter configuration.