Machine Learning and Deep Learning
Lecture-12

By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Overview

- Autoencoder
- Hyper-parameter Optimization (HPO)

Autoencoder

Autoencoder

- Autoencoders are:
  - unsupervised (self-supervised learning)
  - generative models: capable of randomly generating new data that looks very similar to training data.

- ANNs capable of learning dense representations of input data, called latent representations or codings.
- Codings typically have a much lower dimensionality than input data.
- Autoencoders useful for dimensionality reduction good for visualization purposes.

- Autoencoders simply learn to copy their inputs to their outputs.

Autoencoder

- Codings are byproducts of autoencoder learning identity function under some constraints.
- An autoencoder looks at inputs, converts them to an efficient latent representation, and then output something that (hopefully) looks very close to inputs.

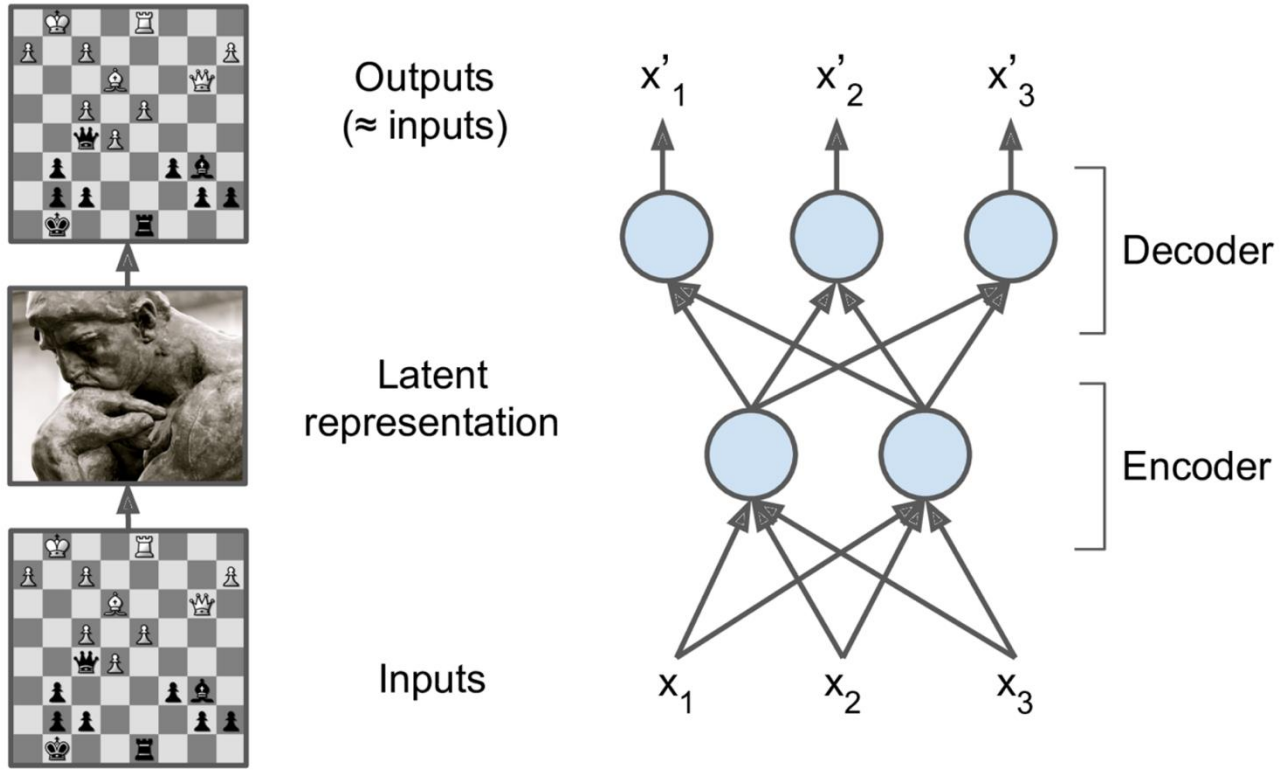
- E.g.,: you can add noise to inputs and train network to recover original inputs.

- Autoencoder composed of two parts:
  - An encoder (or recognition network) that converts inputs to a latent representation.
  - A decoder (or generative network) that converts internal representation to outputs.

Autoencoder

- Autoencoder has same architecture as MLP except number of neurons in output layer must be equal to number of inputs.

- Here is single hidden layer composed of two neurons (encoder).
- One output layer composed of three neurons (decoder).
- Outputs are often called reconstructions because autoencoder tries to reconstruct inputs



Autoencoder

- Cost function contains a reconstruction loss that penalizes model when reconstructions are different from inputs.

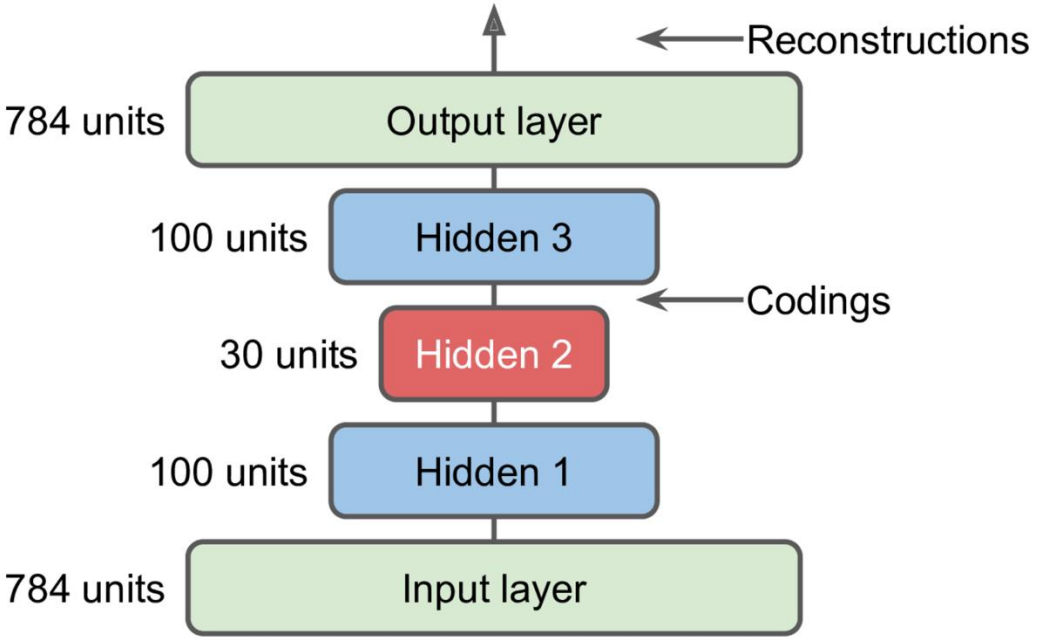
- Undercomplete autoencoder where internal representation has a lower dimensionality than input data.
  - An undercomplete autoencoder cannot trivially copy its inputs to codings, and forced to learn most important features in input data (and drop unimportant ones).

- If autoencoder uses only linear activations and cost function is mean squared error (MSE), then it ends up performing Principal Component Analysis (PCA).

Stacked Autoencoder

- Stacked autoencoders (or deep autoencoders): Autoencoders can have multiple hidden layers.
  - Advantage: Adding more layers helps autoencoder learn more complex codings.

- An autoencoder for MNIST have 784 inputs, followed by a hidden layer with 100 neurons.
- A central hidden (coding) layer of 30 neurons, then another hidden layer with 100 neurons, and an output layer with 784 neurons



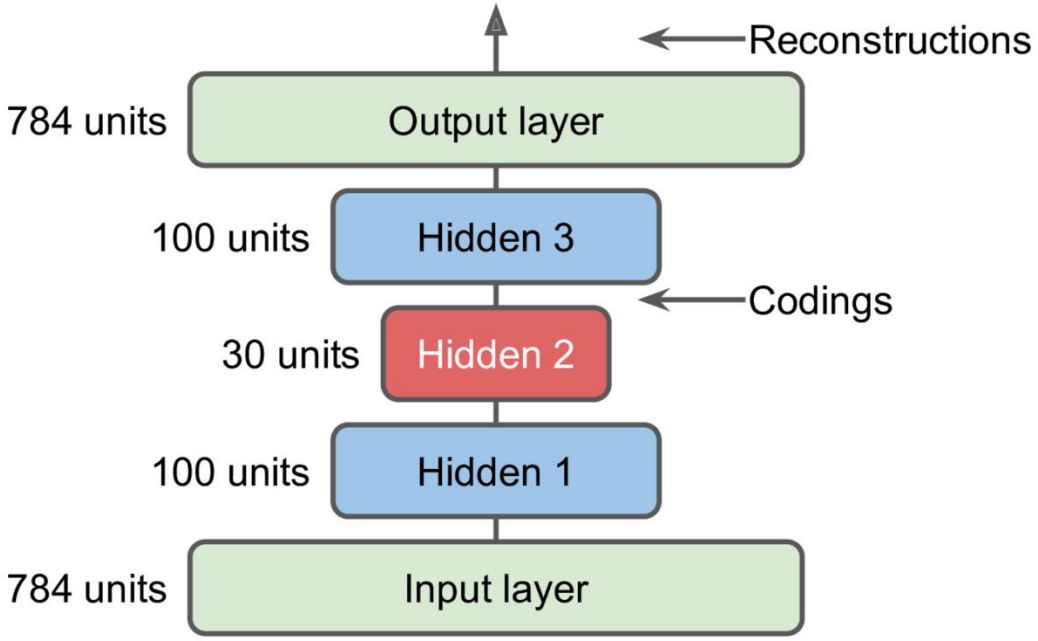
Stacked Autoencoder

- Stacked autoencoder very much like a regular deep MLP.

```
stacked_encoder = keras.models.Sequential([
keras.layers.Flatten(input_shape=[28, 28]),
keras.layers.Dense(100, activation="selu"),
keras.layers.Dense(30, activation="selu"),
])

stacked_decoder = keras.models.Sequential([
keras.layers.Dense(100, activation="selu", input_shape=[30]),
keras.layers.Dense(28 * 28, activation="sigmoid"),
keras.layers.Reshape([28, 28])
])

stacked_ae = keras.models.Sequential([stacked_encoder, stacked_decoder])
stacked_ae.compile(loss="binary_crossentropy",
optimizer=keras.optimizers.SGD(learning_rate=1.5))
history = stacked_ae.fit(X_train, X_train, epochs=10,
validation_data=[X_valid, X_valid])
```



Convolution Autoencoder

- If you want to build an autoencoder for images, you need to build a convolutional autoencoder.
  - Encoder is a regular CNN composed of convolutional layers and pooling layers.
  - It reduces spatial dimensionality of inputs (i.e., height and width) while increasing depth (i.e., number of feature maps).
  - Decoder must do reverse (upscale image and reduce its depth back to original dimensions), and for this use transpose convolutional layers.

Recurrent & Denoising Autoencoders

- If you want to build an autoencoder for sequences (such as time series or dimensionality reduction), then RNNs may be better suited than dense networks.
- Recurrent autoencoder:
  - Encoder is typically a sequence-to-vector RNN which compresses input sequence down to a single vector.
  - Decoder is a vector-to-sequence RNN that does reverse.

- Denoising Autoencoders: force autoencoder to learn useful features by adding noise to its inputs, training it to recover original (noise-free).
- Noise can be pure Gaussian noise added to inputs, or it can be randomly switched-off inputs (dropout).
- Autoencoders could also be used for feature extraction*.

*Pascal Vincent et al., "Extracting and Composing Robust Features with Denoising Autoencoders," Proceedings of the 25th International Conference on Machine Learning (2008): 1096–1103.

Recap

- Autoencoders learn to copy their inputs to their outputs by converting inputs into an efficient latent representation and then outputting something close to the original inputs.
  - Structure: An autoencoder is composed of two main parts:
    - Encoder (or recognition network): Converts inputs to a latent representation.
    - Decoder (or generative network): Converts the internal representation back to outputs.
- The cost function of an autoencoder contains a reconstruction loss that penalises the model when the reconstructions differ from the inputs.
- An autoencoder has the same architecture as a Multilayer Perceptron (MLP), except that the number of neurons in the output layer must be equal to the number of inputs.
- A recurrent autoencoder typically has an encoder that is a sequence-to-vector RNN compressing the input sequence into a single vector, and a decoder that is a vector-to-sequence RNN doing the reverse.
- Applications: Autoencoders are useful for dimensionality reduction (good for visualization)

Hyper-parameter Optimization (HPO)

Hyper-parameters

- NNs have several hyperparameters: learning rate, weight of regularization.
- “Hyperparameter” regulate design of model.
- Different from more fundamental parameters representing weights of connections in NN.
  - Primary model parameters weights are optimized with backpropagation only.
  - After fixing the hyperparameters either manually or with the use of a tuning phase.
- Hyperparameters should not be tuned using same data (SGD).
  - Portion of data is held out as validation data.
  - Performance of model is tested on validation set with various choices of hyperparameters.
- Ensures not overfit of training data set.

Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." *Neurocomputing* 415 (2020): 295-316.

Challenges Hyper-parameter Optimization

- Function evaluations can be extremely expensive for large models (e.g., in deep learning), complex machine learning pipelines, or large datasets.
- Configuration space is often complex (comprising a mix of continuous, categorical and conditional hyperparameters) and high-dimensional (not always clear which hyperparameters to optimize).
- No access to a gradient of hyperparameter loss function. Other properties of target function often used in classical optimization do not typically apply (such as convexity and smoothness).
- One cannot directly optimize for generalization performance as training datasets are of limited size.

F. Hutter et al. (eds.), Automated Machine Learning, The Springer Series on Challenges in Machine Learning, https://doi.org/10.1007/978-3-030-05318-5_1

Hyper-parameters

ML Algorithm	Main HPs	Optional HPs	HPO methods	Libraries
Linear regression	–	–	–	–
Ridge & lasso	alpha	–	BO-GP	Skpot
Logistic regression	penalty, c, solver	-	BO-TPE, SMAC	Hyperopt, SMAC
KNN	n_neighbors	weights, p, algorithm	BOs, Hyperband	Skpot, Hyperopt, SMAC, Hyperband
SVM	C, kernel, epsilon (for SVR)	gamma, coef0, degree	BO-TPE, SMAC, BOHB	Hyperopt, SMAC, BOHB
NB	alpha	–	BO-GP	Skpot
DT	criterion, max_depth, min_samples_split, min_samples_leaf, max_features	splitter, min_weight_fraction_leaf, max_leaf_nodes	GA, PSO, BO-TPE, SMAC, BOHB	TPOT, Optunity, SMAC, BOHB
RF & ET	n_estimators max_depth, criterion, min_samples_split, min_samples_leaf, max_features	splitter, min_weight_fraction_leaf, max_leaf_nodes	GA, PSO, BO-TPE, SMAC, BOHB	TPOT, Optunity, SMAC, BOHB
XGBoost	n_estimators, max_depth, learning_rate, subsample, colsample_bytree,	min_child_weight, gamma, alpha, lambda	GA, PSO, BO-TPE, SMAC, BOHB	TPOT, Optunity, SMAC, BOHB
Voting	estimators, voting	weights	GS	Sklearn
Bagging	base_estimator, n_estimators	max_samples, max_features	GS, BOs	sklearn, Skpot, Hyperopt, SMAC
AdaBoost	base_estimator, n_estimators, learning_rate	-	BO-TPE, SMAC	Hyperopt, SMAC
Deep learning	number of hidden layers, 'units' per layer, loss, optimizer, Activation, learning_rate, dropout rate, epochs, batch_size, early stop patience	number of frozen layers (if transfer learning is used)	PSO, BOHB	Optunity, BOHB
K-means	n_clusters	init, n_init, max_iter	BOs, Hyperband	Skpot, Hyperopt, SMAC, Hyperband
Hierarchical clustering	n_clusters, distance_threshold	linkage	BOs, Hyperband	Skpot, Hyperopt, SMAC, Hyperband
DBSCAN	eps, min_samples	-	BO-TPE, SMAC, BOHB	Hyperopt, SMAC, BOHB
Gaussian mixture	n_components	covariance_type, max_iter, tol	BO-GP	Skpot
PCA	n_components	svd_solver	BOs, Hyperband	Skpot, Hyperopt, SMAC, Hyperband
LDA	n_components	solver, shrinkage	BOs, Hyperband	Skpot, Hyperopt, SMAC, Hyperband

Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." *Neurocomputing* 415 (2020): 295-316.

Parameter Initialization

- Parameter initialization is crucial to NN performance:
  - Can't initialize weights in same layer to same value, or they will stay same.
  - Can't initialize weights too large, it will take too long to learn.
- A traditional random initialization:
  - Initialize bias variables to 0.
  - Sample from standard normal, divided by 10^5(0.00001*randn).
  - w = .00001*randn(k,1)
  - Performing multiple initializations does not seem to be important.
- Popular approach from 10 years ago:
  - Initialize with deep unsupervised model (like "autoencoders")

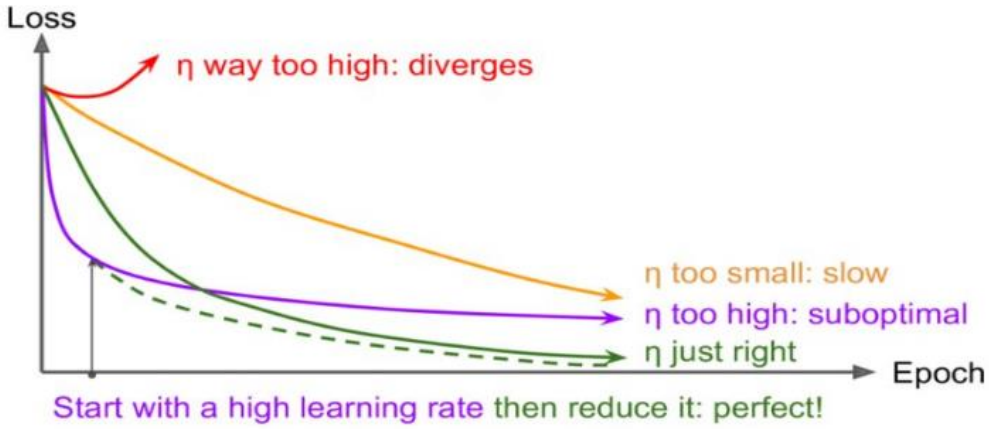
Learning Rate: Setting the Step-Size

- Stochastic gradient is very sensitive to step size in deep models.
- A common approach:
  - Run SG for a while with a fixed step-size.
  - Occasionally measure error and plot progress.
  - If error is not decreasing, decrease step-size.

- Bias step-size multiplier: use bigger step-size for bias variables.
- Momentum: Add term that moves in previous direction where $\beta^t=0.9$

Learning Rate Scheduling

- Good learning rate is very important.
  - If you set it much too high, training may diverge (“Gradient Descent”)
  - If you set it too low, training will eventually converge to optimum, at cost of very long time.



- Find a good learning rate by:
  - Training model for a few hundred iterations, exponentially increasing learning rate from a very small value to a very large value.
  - Next look at learning curve and pick a learning rate slightly lower than one at which learning curve starts shooting back up.
  - Reinitialize model and train it with that learning rate.

Learning Rate Scheduling

- Strategies to reduce learning rate during training called learning schedules.

1. Power scheduling: Set learning rate to a function of iteration number t: $\eta(t) = \eta_0 / (1+t/s)^c$. (initial learning rate η_0, power c (set to 1), steps s are hyperparameters).
2. Exponential scheduling: Set learning rate to $\eta(t) = \eta_0\ 0.1^{t/s}$. Learning rate will gradually drop by a factor of 10 every s steps.
3. Piecewise constant scheduling: Use a constant learning rate for a number of epochs (e.g., $\eta_0 = 0.1$ for 5 epochs), then smaller learning rate for another number of epochs (e.g., $\eta_1 = 0.001$ for 50 epochs), and so on.
4. Performance scheduling: Measure validation error every N steps (like early stopping) and reduce learning rate by a factor of λ when error stops dropping.

Hyper-parameters

- How should candidate hyperparameters be selected for testing? --> Grid search
  - All combinations of selected values of hyperparameters are tested to determine optimal choice.
  - Number of points in grid increases exponentially with number of hyperparameters.
  - Ex- 5 hyperparameters, and test 10 values for each hyperparameter, training procedure needs to be executed $10^5 = 100000$ times to test its accuracy.
- Grid-based hyperparameter exploration is NOT best choice.

Hyper-parameters

- Make random sample of hyperparameters uniformly within grid range.
- Logarithms of hyperparameters are sampled uniformly rather than hyperparameters themselves (regularization rate and learning rate).
- Ex: instead of sampling learning rate (alpha) between 0.1 and 0.001, first sample log(α) uniformly between −1 and −3, and then exponentiate it to power of 10.
- Common to search in logarithmic space

Hyper-parameter Optimization Techniques

- Babysitting or ‘Trial and Error’ or Grad student descent (GSD) method is implemented by 100% manual tuning and widely used.
  - Problems due to large number of hyper-parameters, complex models, time-consuming model evaluations, and non-linear hyper-parameter interactions.

- Grid search (GS) most commonly-used methods
  - An exhaustive search or a brute-force method
  - Works by evaluating the Cartesian product of user-specified finite set of values
  - Problem: Inefficiency for high-dimensionality hyper-parameter configuration space
    - Since number of evaluations increases exponentially to number of hyper-parameters growth

Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." *Neurocomputing* 415 (2020): 295-316.

Hyper-parameter Optimization Techniques

- Random search (RS): like GS.
  - Randomly selects a pre-defined number of samples between upper and lower bounds as candidate hyper-parameter values, and then trains these candidates until defined budget is exhausted.
  - With a limited budget, RS can explore a larger search space than GS.
  - Problem: unnecessary function evaluations since it does not exploit previously well-performing regions.

- Gradient-based optimization: traditional technique.
  - After randomly selecting a data point, it moves towards opposite direction of largest gradient to locate next data point.
    - Local optimum can be reached after convergence.
  - Gradient-based algorithms have a fast convergence speed to reach local optimum.
  - Can use to optimize learning rate in neural networks (NN).

Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." *Neurocomputing* 415 (2020): 295-316.

Hyper-parameter Optimization Techniques Comparisions

The comparison of common HPO algorithms (n is the number of hyper-parameter values and k is the number of hyper-parameters).

HPO Method	Strengths	Limitations	Time Complexity
GS	• Simple.	• Time-consuming • Only efficient with categorical HPs.	$O(n^k)$
RS	• More efficient than GS • Enable parallelization.	• Not consider previous results • Not efficient with conditional HPs.	$O(n)$
Gradient-based models	• Fast convergence speed for continuous HPs.	• Only support continuous HPs • May only detect local optimums.	$O(n^k)$
BO-GP	• Fast convergence speed for continuous HPs.	• Poor capacity for parallelization • Not efficient with conditional HPs.	$O(n^3)$
SMAC	• Efficient with all types of HPs.	• Poor capacity for parallelization.	$O(nlogn)$
BO-TPE	• Efficient with all types of HPs • Keep conditional dependencies.	• Poor capacity for parallelization.	$O(nlogn)$
Hyperband	• Enable parallelization.	• Not efficient with conditional HPs • Require subsets with small budgets to be representative.	$O(nlogn)$
BOHB	• Efficient with all types of HPs • Enable parallelization.	• Require subsets with small budgets to be representative.	$O(nlogn)$
GA	• Efficient with all types of HPs • Not require good initialization.	• Poor capacity for parallelization.	$O(n^2)$
PSO	• Efficient with all types of HPs • Enable parallelization.	• Require proper initialization.	$O(nlogn)$

Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." *Neurocomputing* 415 (2020): 295-316.

Hyper-parameter Optimization Frameworks

- Sklearn:
  - GridSearchCV can be implemented to detect optimal hyper-parameters using GS algorithm.
  - RandomizedSearchCV to implement a RS method.

- TensorFlow
  - Trieste: Bayesian optimization toolbox built on TensorFlow (https://github.com/secondmind-labs/trieste)

Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." *Neurocomputing* 415 (2020): 295-316.