Predicting Financial Time Series using Deep Learning

Module3. Hyperparameter tuning, Regularization and Optimization

Jongho Kim

NICE Pricing & Information Inc.

Winter, 2018

Tuning Process

- One of the painful things about training deepness is the sheer number of hyperparameters you have to deal with
  - Learning rate: alpha
  - Momentum: beta (for Adam Optimization Algorithm beta one, two)
  - Learning rate decay (you don't just use a single learning rate alpha)
  - Number of layers
  - Number of hidden units for the different layers
  - Mini-batch size

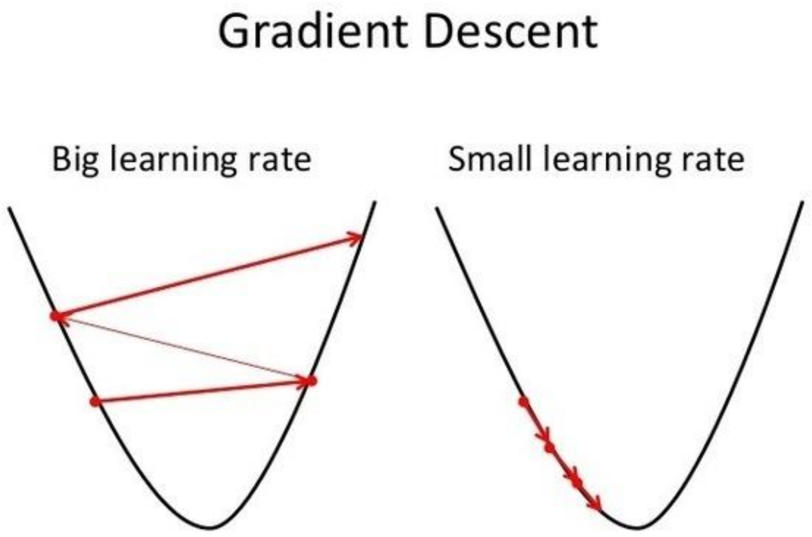
Tuning Process

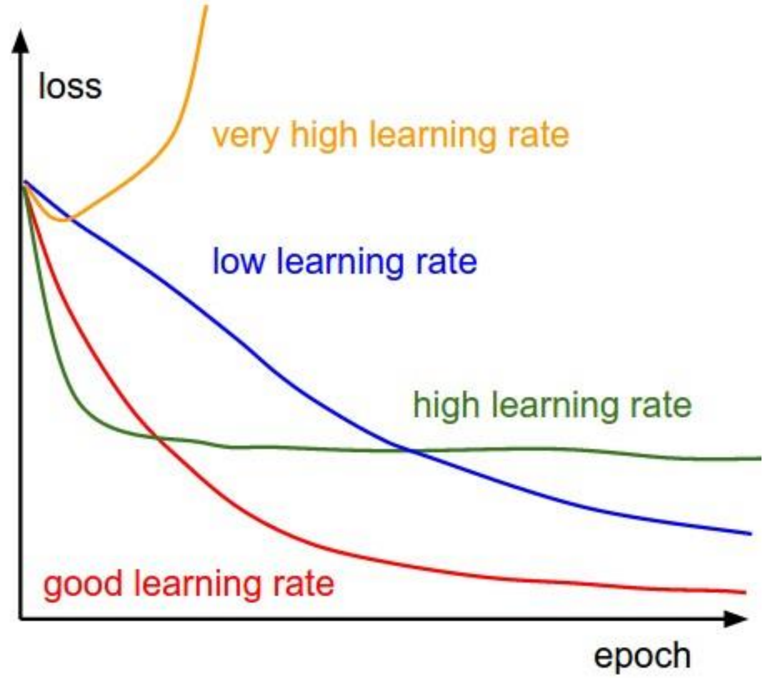
- One of the painful things about training deepness is the sheer number of hyperparameters you have to deal with
  - Learning rate: alpha ← The most important hyperparameter
  - Momentum: beta ← Second important
  - Learning rate decay ← Third important
  - Number of layers ← Third important
  - Number of hidden units for the different layers ← Second important
  - Mini-batch size ← Second important

Source: https://www.coursera.org/learn/deep-neural-network/lecture/DHNcc/hyperparameters-tuning-in-practice-pandas-vs-caviar

Learning Rates

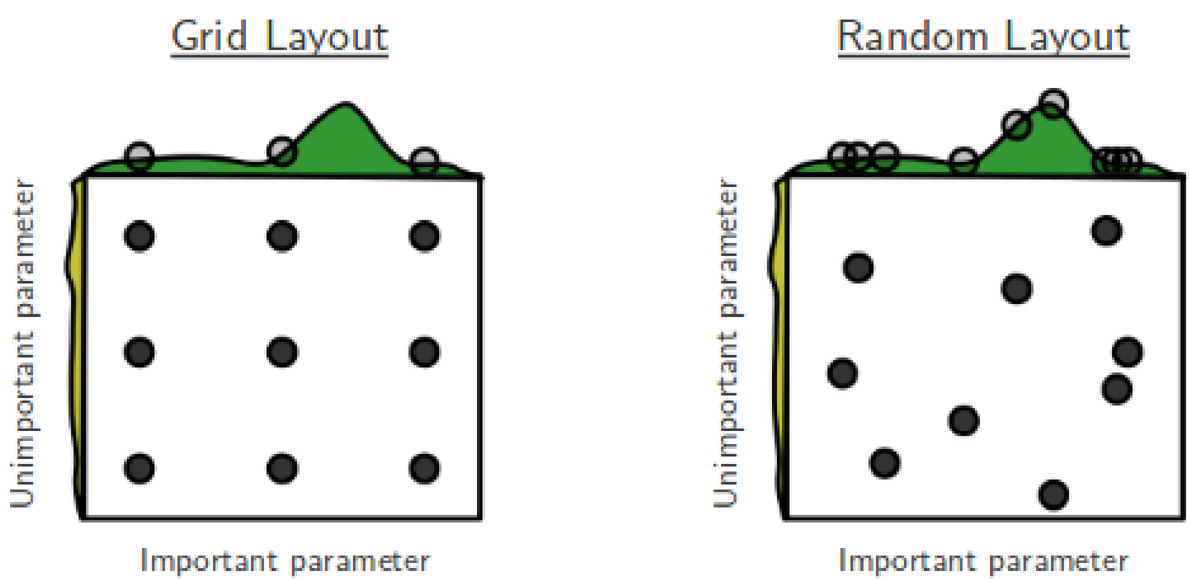
- If we choose small learning rate, our cost will be updated slowly with small steps
- However, if our learning rate is too high, we overshoot and go beyond the minimum and go further away from the minimum





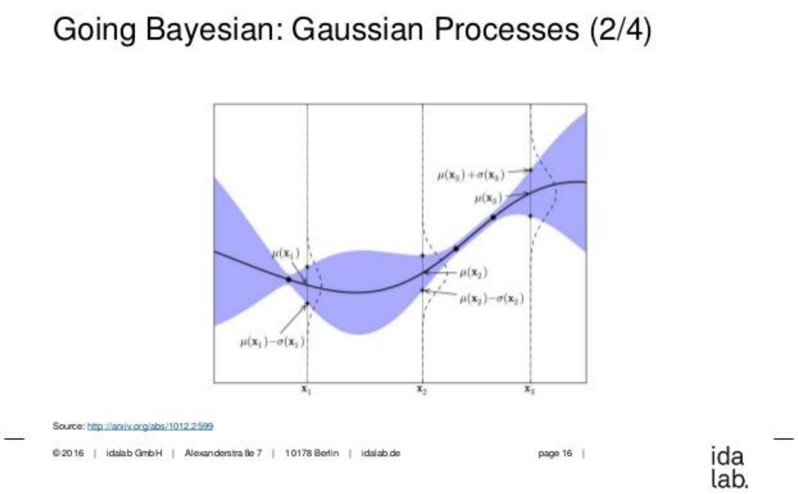
How to select hyperparameters

- Grid search
  - Selecting several points from ranges, then train your network using every combination of parameters and select the combination that performs best
- Random search
  - Instead of picking values from ranges in a methodical manner you instead

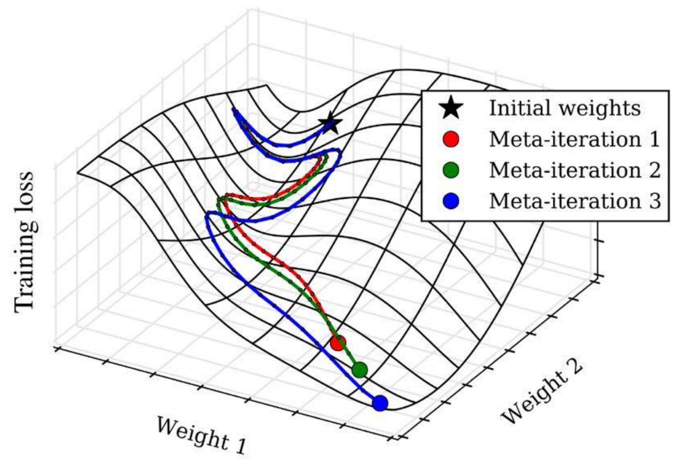


How to select hyperparameters

- Bayesian optimization
  - Using the information gained from any given experiment to decide how to adjust the hyper parameters

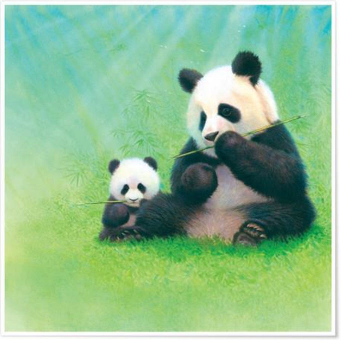


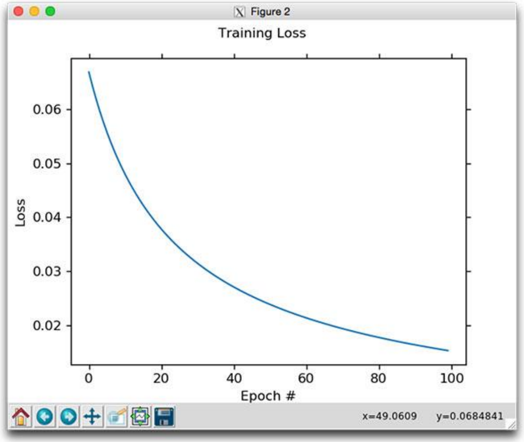
- Gradient based optimization
  - For specific learning algorithms, it is possible to compute the gradient with respect to hyperparameters

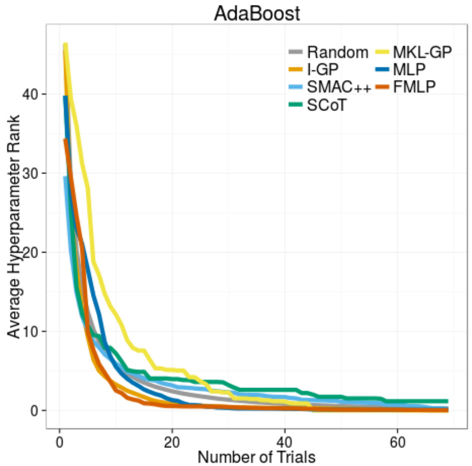


How to select hyperparameters

- Panda Approach vs. Caviar Approach
  - Take the caviar approach and try a lot of different hyperparameters and see what works
  - But in some application domains, there's just so much data and the models you want to train are so big that it's difficult to train a lot of models at the same time







Source: https://www.coursera.org/learn/deep-neural-network/lecture/DHNcc/hyperparameters-tuning-in-practice-pandas-vs-caviar

Initialization

Why Initialization is Important?

- Training your neural network requires specifying an initial value of the weights

- A well chosen initialization can:
  - Speed up the convergence of gradient descent
  - Increase the odds of gradient descent converging to a lower training (and generalization) error
- Poor initialization can lead to vanishing/exploding gradients, which also slows down the optimization algorithm.

- But how do you choose the initialization for a new neural network?

Types of Initialization

- Zero initialization (W: zeros)
  - In general, initializing all the weights to zero results in the network failing to break symmetry, every neuron in each layer will learn the same thing

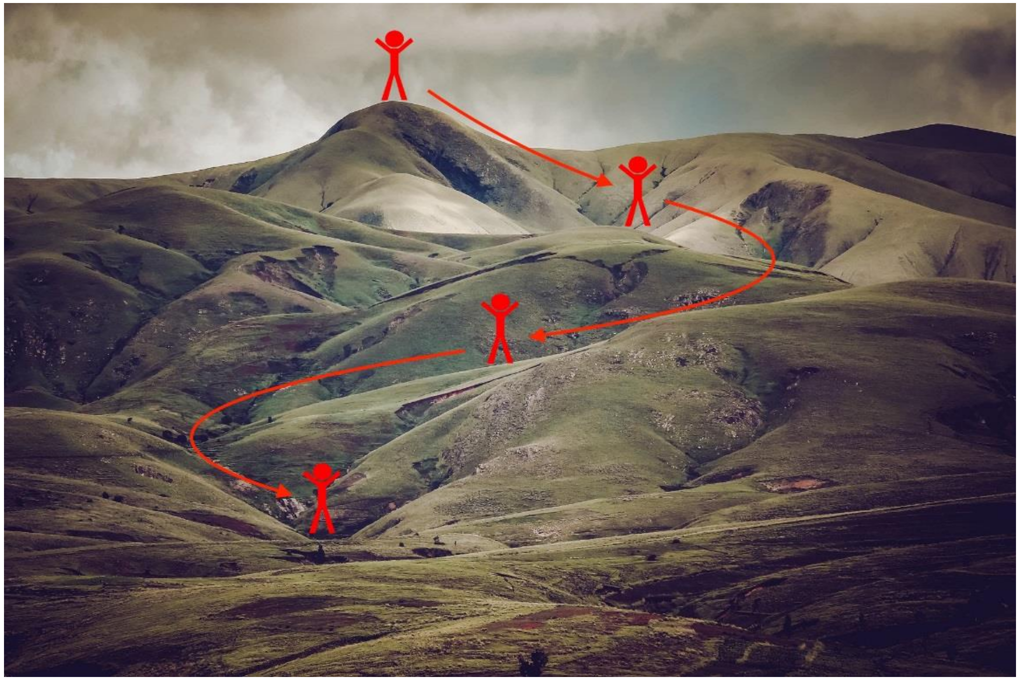
- Random initialization (W: random)
  - To break symmetry, we can intialize the weights randomly
  - However, if the weights are intialized randomly, but to very large values then vanishing/exploding gradients can happen

- He initialization (W: random + scale factor)
  - Thus we scale the vector with factor : sqrt(2./layers_dims[l-1]).)
  - This is similar to Xavier initialization: sqrt(1./layers_dims[l-1])
  - He initialization recommends for layers with a ReLU activation.

Optimization Algorithms

Optimization Methods

- Update the parameters to minimize the cost
- More advanced optimization methods can speed up learning and even get you to a better final value for the cost function



Minimizing the cost is like finding the lowest point in a hilly landscape

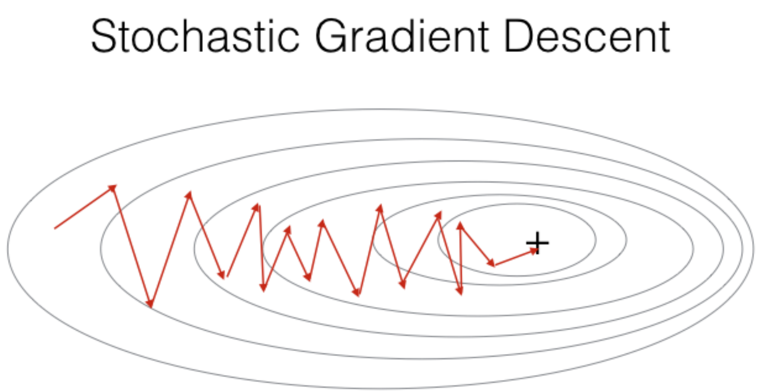
Gradient Descent

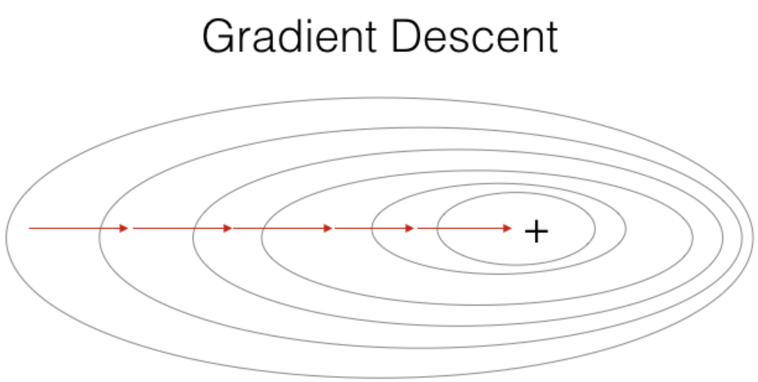
- (Batch) Gradient Descent
  - Single update occurs using all training data (1 epoch) => Slow estimation
- Stochastic Gradient Descent
  - Single training data point is used to estimate gradients => Very noise estimation

Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

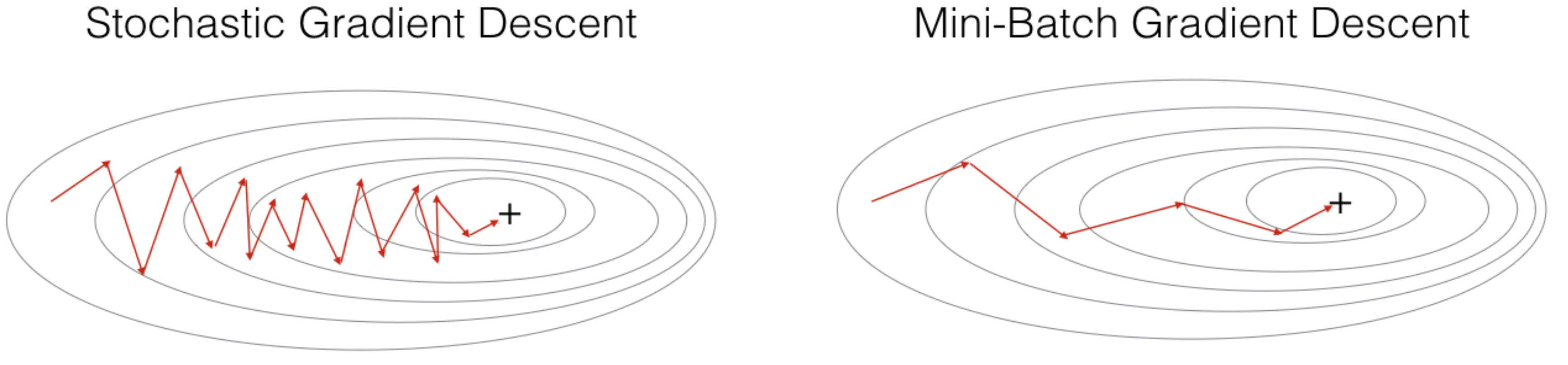
}





Gradient Descent

- (Mini Batch) Gradient Descent
    - Split epoch into small groups of data points
    - Compromise between SGD and (Batch) GD
    - Less noise and many updates
    - Ex) 16, 32, 64, 128, 256 etc



Momentum

- The key is update weights many times + less noise estimation
- To achieve this, SGD has been extended with momentum (or velocity)
  - Avoid jumping, have some trends in updating weights
  - SGD with momentum

Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boxed{\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

  - SGD with Nesterov mometum

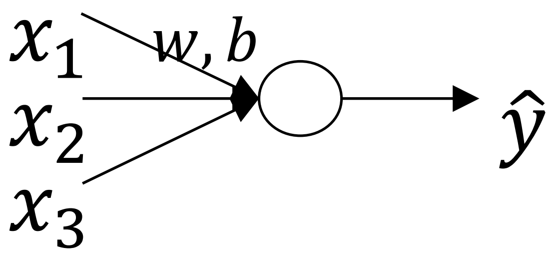
Apply interim update: $\boxed{\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}}$
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boxed{\tilde{\boldsymbol{\theta}}}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

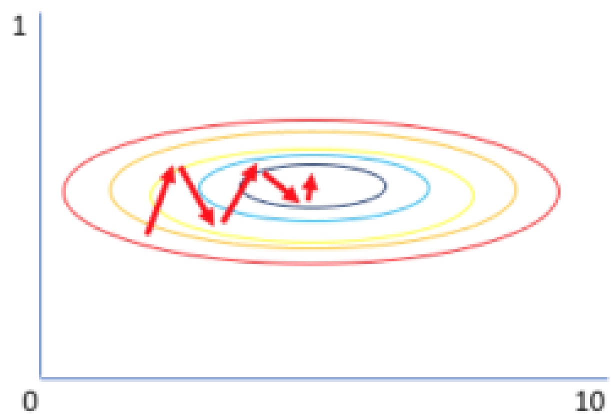
Batch Normalization

Normalizing inputs to speed up learning

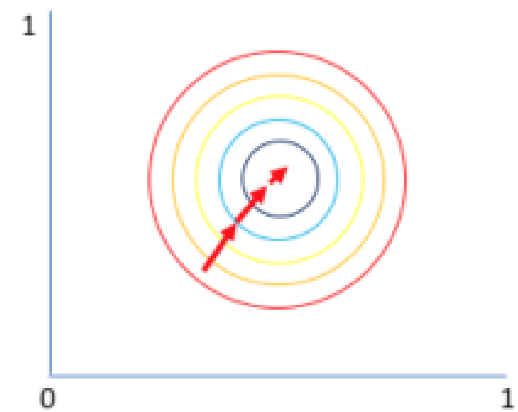
- **Unnormalized data** can lead toward an awkward loss function topology which places more emphasis on certain parameter gradients.

- By normalizing all of our inputs to a standard scale, we're allowing the network to *more quickly* learn the optimal parameters for each input node.





Gradient of larger parameter dominates the update

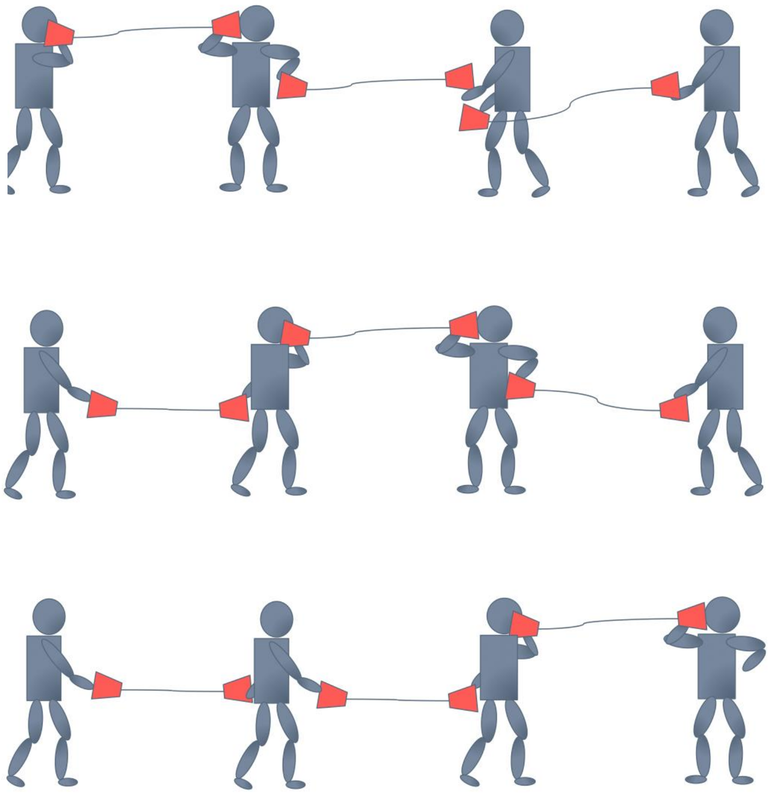


Both parameters can be updated in equal proportions

Source: https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b

How about deeper model?

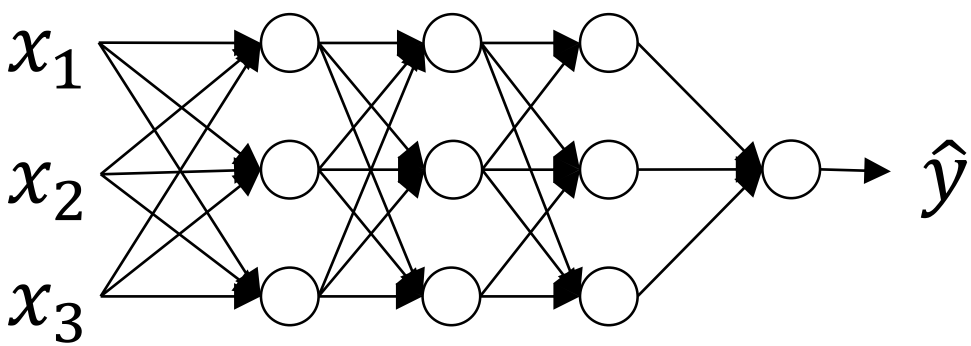
- Internal Covariance Shift Problem
  - Covariate shift: change in the distribution of a function's domain
  - The problem is, the output of the first layer feeds the second layer, the second feeds the third, and so on
  - Have you ever played the game telephone with cups and strings?
    - "go water the plants" => "got water in your pants" => "kite bang eat face monkey"
  - We can fix our cups so that we pass messages better.



Source: https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b

Normalizing deeper model by batch normalization

- Batch Normalization
  - Normalizing the input of your network is a well-established technique for improving the convergence properties of a network.
  - By extending the intuition normalizing these values on each layer will help the network more effectively learn the parameters



Normalizing deeper model by batch normalization

- Batch Normalization for Training
  - We can calculate mean, variance, z norm, Z tilder for each of mini batch
  - This is in fact, taking the batch statistics (not population statistics)

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

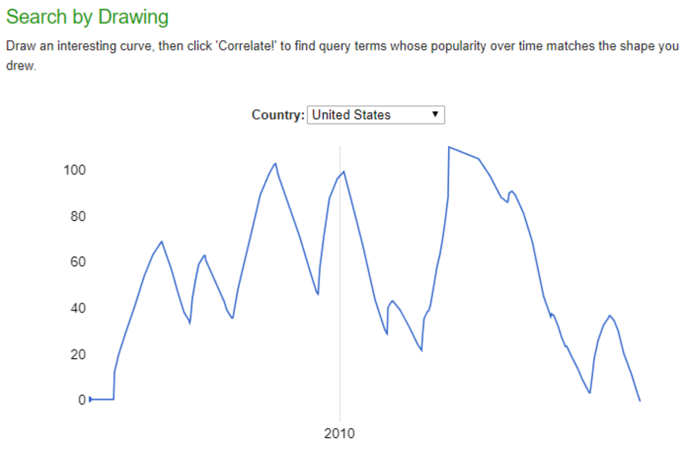
$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

- Batch Normalization for Inferring
  - For test time we use exponential weighted average across mini batch
  - Estimating the true mean and variance is calculated over the entire population

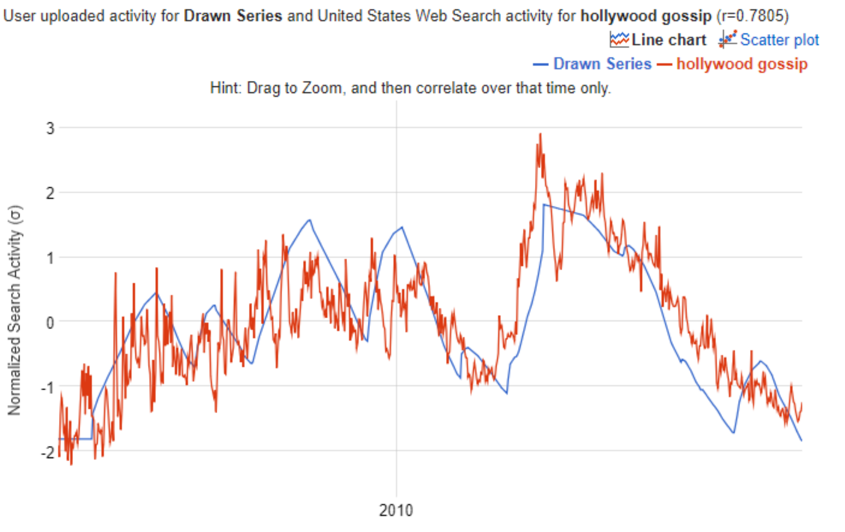
Overfitting Issues

Overfitting is Easy

- When I draw a random curve of Google search trends...



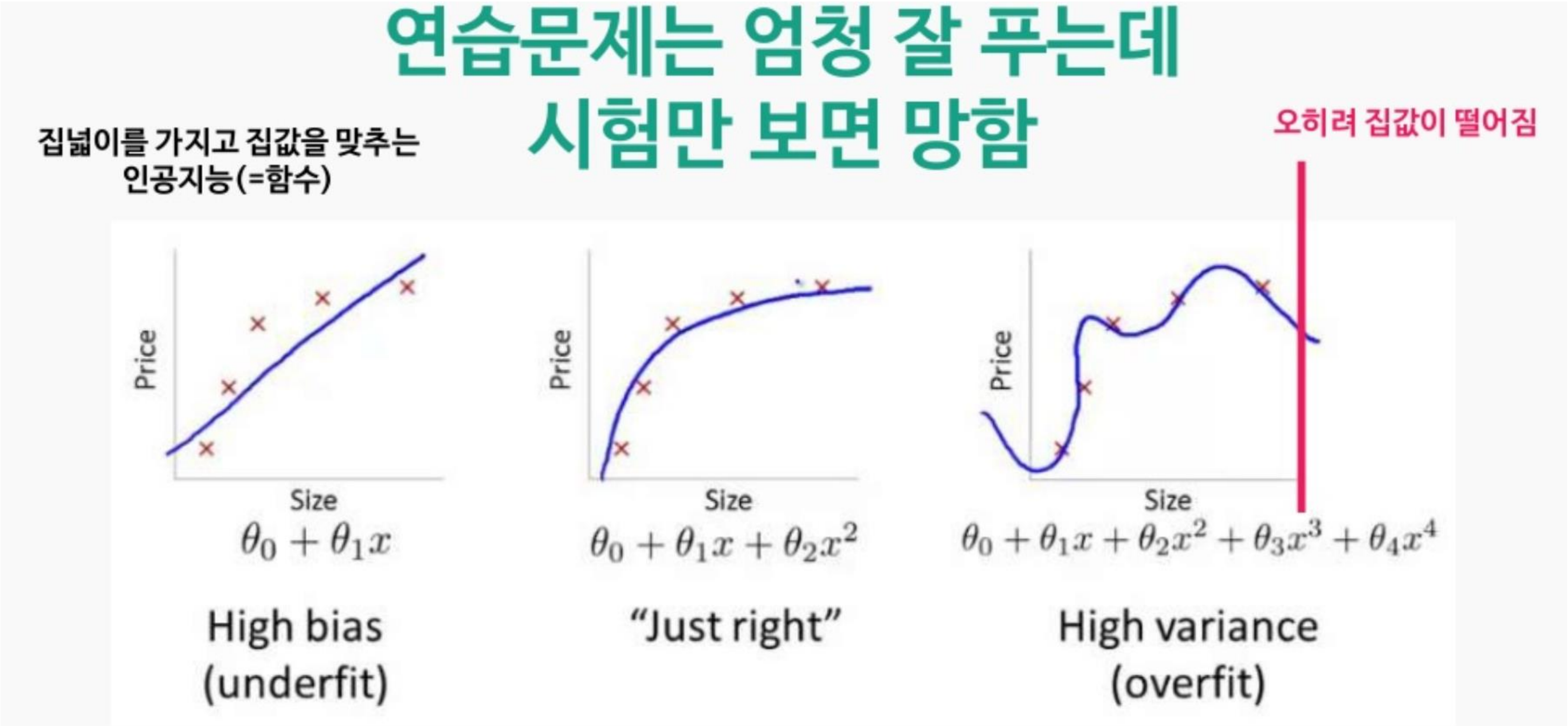
https://www.google.com/trends/correlate/draw



- “Eureka! I accidently succeed in predicting how people search about ‘Hollywood gossip’ on Google. Its correlation is 0.78!”
- … Really?

Overfitting is Common in Prediction

- Overfitting often results from (i) too complex models and (ii) too few data.



Source: https://www.slideshare.net/modulabs/2-cnn-rnn

Occam's Razor

- Occam's razor is still valid (that is, simple is best).
  - ➢ For out-of-sample predictions, simpler models are more likely to hold up on future observations than more complex ones, all else being equal (Dhar 2013).
- Sometimes, machine learning algorithms might have poorer performances than a simple logistic regression.
  - ➢ It may be especially when (1) there is relatively simple relationships, or (2) there are too few data to learn about the relationships.
  - ➢ Example: Predicting movie success (Lee et al. 2018)

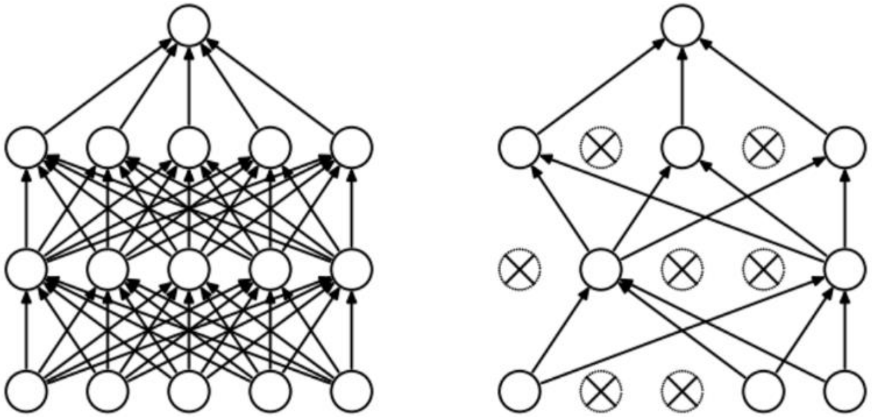
Logistic Regression		Deep NN		Random Forest		Support Vector Machine	
LR		*NN (MLP)*		*RF*		*SVC*	
Bingo	1-Away	Bingo	1-Away	Bingo	1-Away	Bingo	1-Away
36.0 %	85.3 %	48.0 %	86.7 %	46.7 %	84.0 %	26.7 %	64.0 %
45.3 %	93.3 %	40.0 %	88.0 %	56.0 %	90.7 %	30.7 %	62.7 %
61.3 %	88.0 %	38.7 %	84.0 %	53.3 %	90.7 %	26.7 %	56.0 %
54.7 %	86.7 %	50.7 %	88.0 %	56.0 %	86.7 %	26.7 %	48.0 %
56.0 %	90.7 %	42.7 %	81.3 %	62.7 %	89.3 %	26.7 %	61.3 %
54.7 %	89.3 %	36.0 %	82.7 %	49.3 %	88.0 %	41.3 %	74.7 %
50.7 %	90.7 %	49.3 %	85.3 %	57.3 %	81.3 %	29.3 %	44.0 %
45.3 %	86.7 %	34.7 %	75.7 %	49.3 %	86.7 %	28.0 %	65.3 %
42.7 %	89.3 %	45.3 %	86.7 %	50.7 %	90.7 %	25.3 %	53.3 %
50.7 %	82.7 %	38.7 %	81.3 %	49.3 %	76.0 %	25.3 %	60.0 %
49.7 %	**88.3 %**	**42.4 %**	**84.0 %**	**53.1 %**	**86.4 %**	**28.7 %**	**58.9 %**
7.5 %	3.1 %	5.7 %	3.9 %	4.9 %	4.8 %	4.8 %	8.9 %

Accuracy

Dhar, V., 2013. Data Science and Prediction. *Communications of the ACM*, 56(12), pp.64-73.
Lee, K., Park, J., Kim, I. and Choi, Y., 2018. Predicting Movie Success with Machine Learning Techniques: Ways to Improve Accuracy. *Information Systems Frontiers*, 20(3), pp.577-588.

How Does Deep Learning Overcome Overfitting?

- Regularization
  - Dropout regularization
  - Penalizing model complexity
    (e.g., ridge regression)



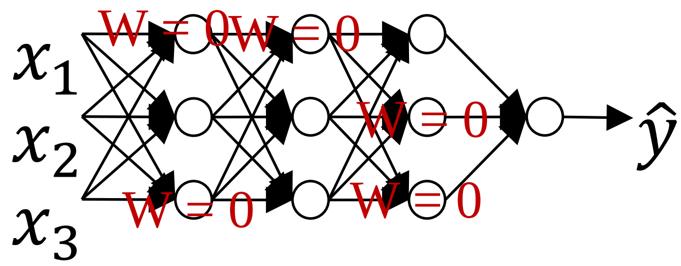
(a) Standard Neural Net (b) After applying dropout.

- Allowing white-noise or loose-fit
  - Data augmentation
  - Debiasing autoencoder / Pooling in CNN / Taking noise as input in GAN

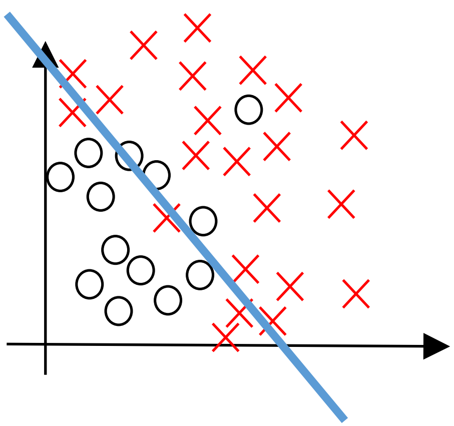
Regularization

Why Regularization Reduces Overfitting?

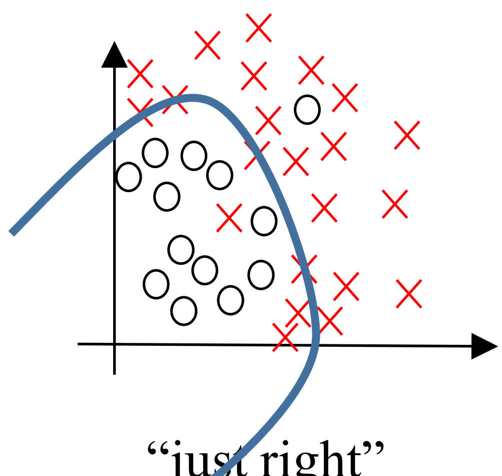
- L2 norm penalty makes weight vector to be sparse => Simpler model



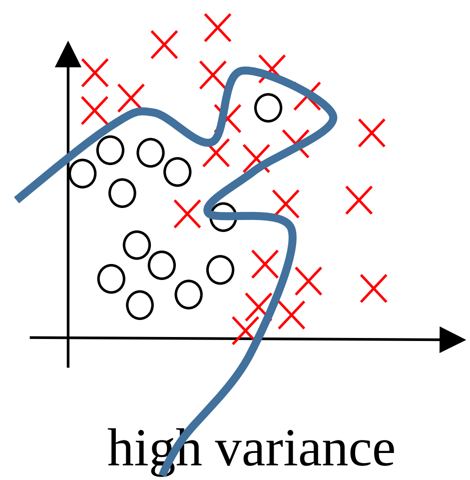
Overfitting



high bias



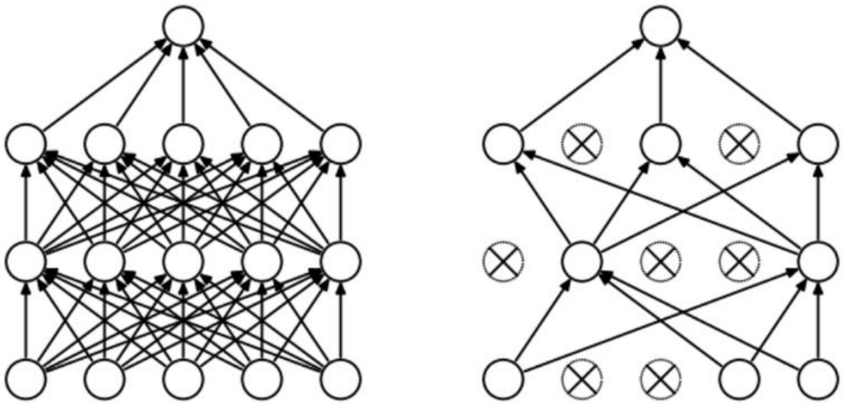
“just right”



high variance

Dropout Regularization

- Dropout regularization
  - During training, dropout samples from an exponential number of different “thinned” networks.
  - Much smaller networks are being trained



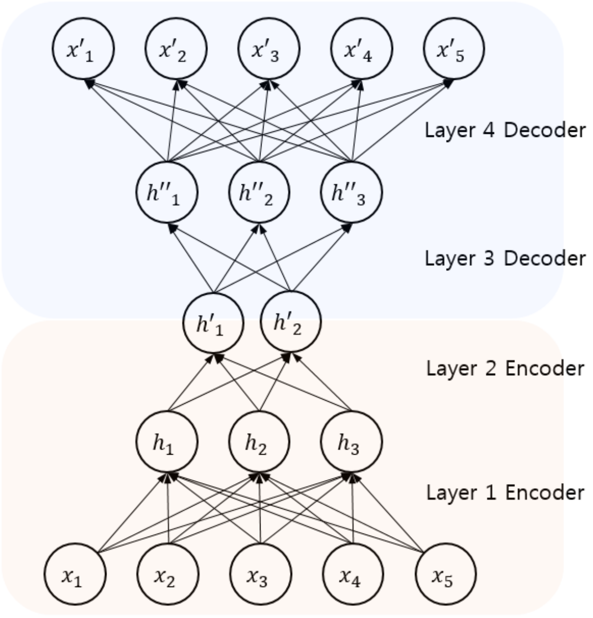
(a) Standard Neural Net (b) After applying dropout.

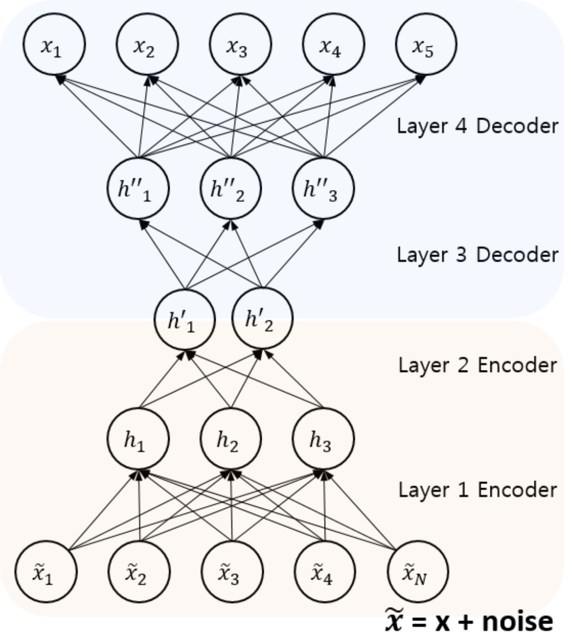
- Why does drop-out work?
  - Can’t rely on any one feature, so have to spread out weights
  - Very efficient way of performing model averaging
  - Preventing complex co-adaptations on training data

Denoising Techniques

Autoencoder

- Autoencoder is a variant of neural networks whose output is the input.
  - ➢ It is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.
  - ➢ Denoising autoencoder is to intentionally introduce noises into the input, allowing the model to encode features robust to noises.

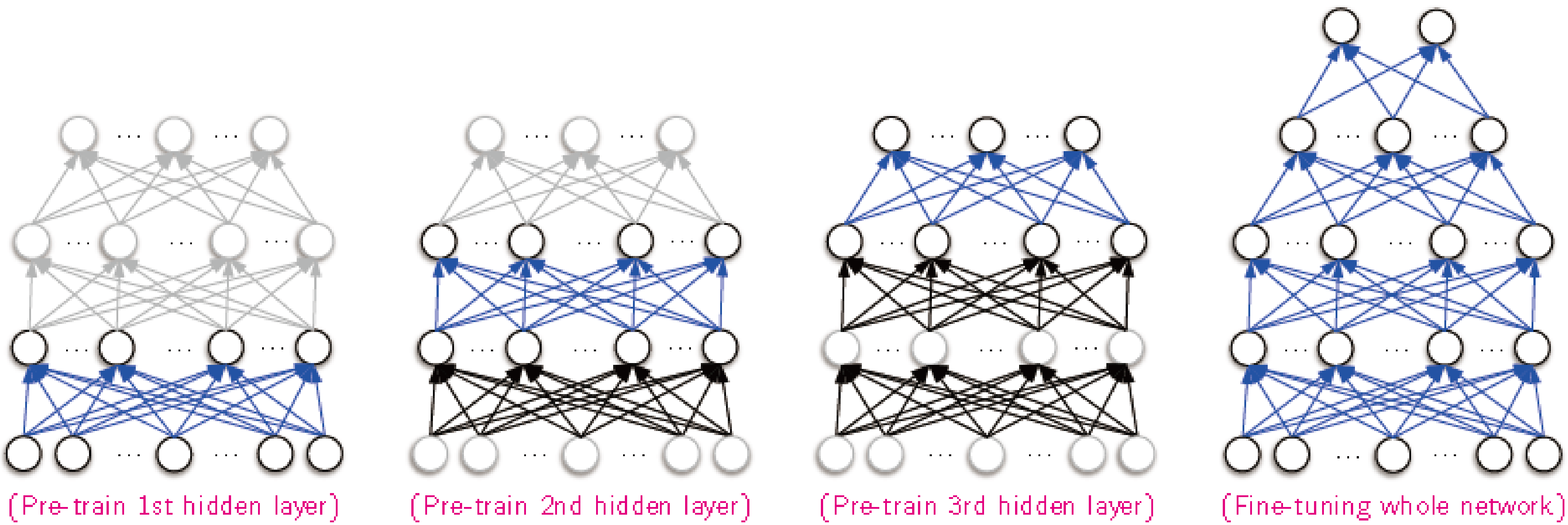




$\tilde{x}$ = x + noise

Pre-Training using Unsupervised Learning

- Greedy layer-wise training of deep networks (Hinton 2006; Bengio et al. 2007)
  - ➢ Feed-forward, pre-training one layer at a time in a unsupervised way
  - ➢ Fine-tuning whole networks using supervised back propagation

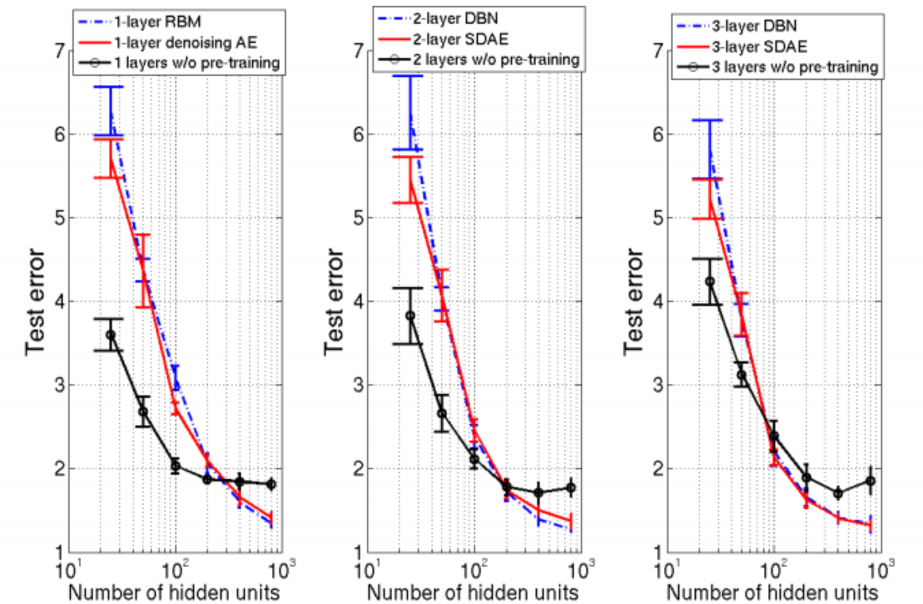


Hinton, G.E., Osindero, S. and Teh, Y.W., 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7), pp.1527-1554.

Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H., 2007. Greedy Layer-wise Training of Deep Networks. *In Advances in Neural Information Processing Systems (NIPS).*

Pre-Training using Unsupervised Learning

- Unsupervised pre-training helps supervised deep learning
  - ➢ Many studies suggest that denoising autoencoders performs generally well at pre-training. (Vincent et al. 2008)



DBN: Deep Belief Net
SDAE: Stacked Denoising Auto Eocoder

(Erhan et al. 2010)

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P. and Bengio, S., 2010. Why Does Unsupervised Pre-training Help Deep Learning?. *Journal of Machine Learning Research*, 11, pp.625-660.

Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.A., 2008. Extracting and Composing Robust Features with Denoising Autoencoders. In Proceedings of the 25th International Conference on Machine Learning (ICML)

Thank you ☺

Contact Info: quantic.jh@gmail.com

References

- How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
- Batch Normalization—What the hey? https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b
- Hyperparameter Optimization with Keras https://towardsdatascience.com/hyperparameter-optimization-with-keras-b82e6364ca53
- Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization https://www.coursera.org/learn/deep-neural-network/lecture/4ptp2/normalizing-activations-in-a-network
- Jiyong Park (2018), KAIST Summer Session, Retrieved from https://sites.google.com/view/kaist-mis-session2018/overview?authuser=0