Deep Learning

Teaching Deep Learners to Generalize

Tiago Vieira

Institute of Computing
Universidade Federal de Alagoas

Summary

What is Model Generalization?

- In a machine learning problem, we try to generalize the known dependent variable on seen instances to unseen instances.
  - Unseen $\Rightarrow$ The model did not see it during training.
  - Given training images with seen labels, try to label an unseen image.
  - Given training emails labeled as spam or nonspam, try to label an unseen email.
- The classification accuracy on instances used to train a model is usually higher than on unseen instances.
  - We only care about the accuracy on unseen data.

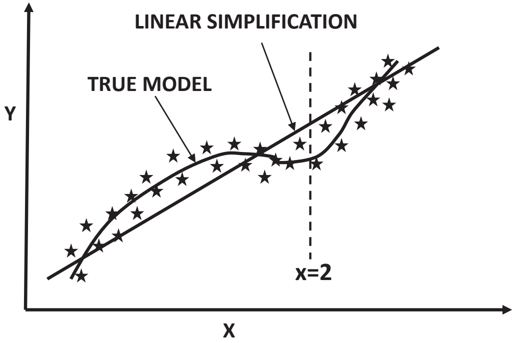
Memorization vs Generalization

- Why is the accuracy on seen data higher?
  - Trained model remembers some of the irrelevant nuances.
  - "Free Money" present on *spam* emails.
- When is the gap between seen and unseen accuracy likely to be high?
  - When the amount of data is limited. Suppose the model is trained on only two email messages.
  - When the model is complex (which has higher *capacity* to remember nuances).
  - The combination of the two is a deadly cocktail.
- A high accuracy gap between the predictions on seen and unseen data is referred to as *overfitting*.

Overfitting

Depends both on the complexity of the model and on the amount of data available.

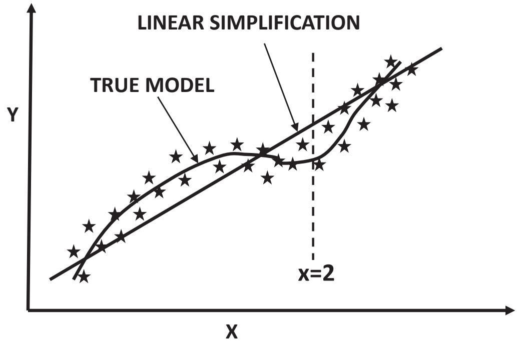
Example: Predict y from x



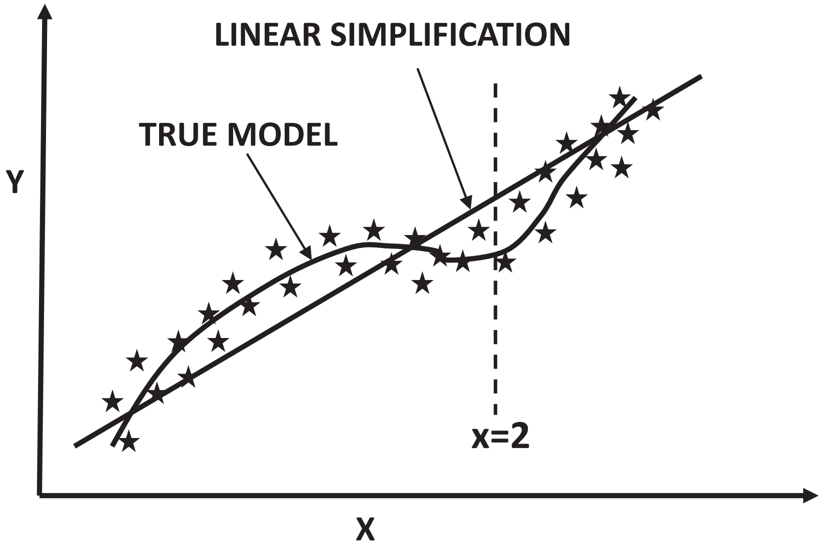
$$\hat{y} = \sum_{i=0}^{d} w_i x^i \tag{1}$$

$$\mathcal{L} = (y - \hat{y})^2$$

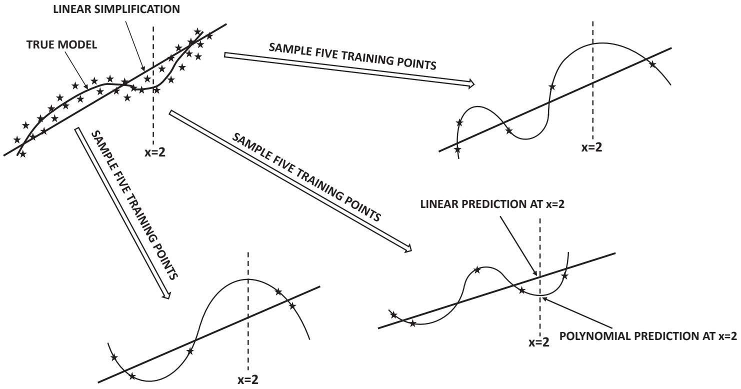
Example: Predict y from x



- **First impression:** Polynomial model such as $\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$ is "better" than linear model $\hat{y} = w_0 + w_1 x$.
  - Bias-variance trade-off says: "Not necessarily! How much data do you have?"


LINEAR SIMPLIFICATION
TRUE MODEL
Y
x=2
X

Different Training Data Sets with Five Points


LINEAR SIMPLIFICATION
TRUE MODEL
x=2
SAMPLE FIVE TRAINING POINTS
SAMPLE FIVE TRAINING POINTS
SAMPLE FIVE TRAINING POINTS
x=2
LINEAR PREDICTION AT x=2
POLYNOMIAL PREDICTION AT x=2
x=2
x=2

- Zero error on training data but wildly varying predictions of $x = 2$

Observations

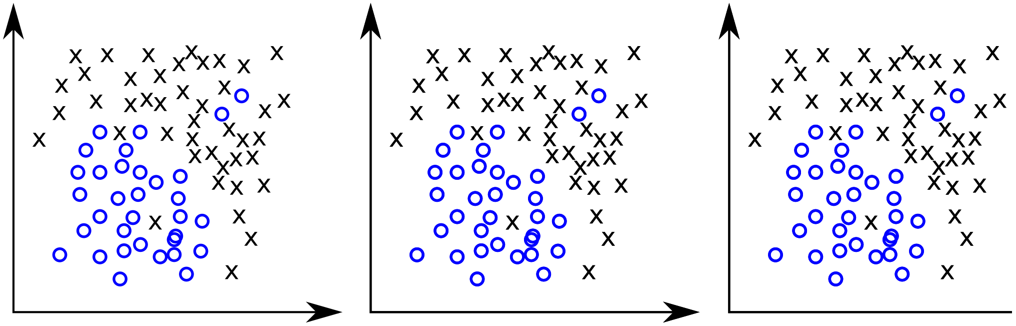
- Linear model does not change much with the training data, whereas the polynomial model changes drastically.
- The higher-order model is more complex than the linear model and has less *bias*.
  - But it has more parameters.
  - For a small training data set, the learned parameters will be more sensitive to the nuances of that data set.
  - Different training data sets will provide different predictions for y at a particular x.
  - This variation is referred to as model *variance*.
- Neural networks are inherently low-bias and high-variance learners $\Rightarrow$ Need ways of handling complexity.

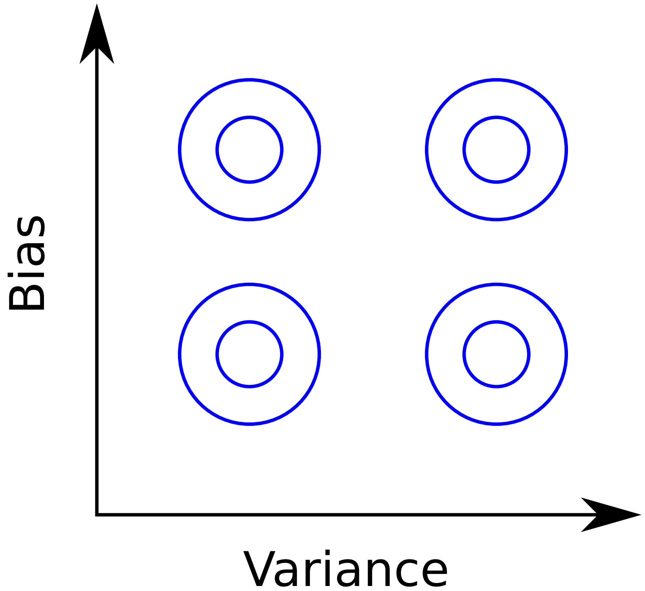
Noise Component

- Unlike bias and variance, noise is a property of the *data* rather than the model.
- Noise refers to unexplained variations ϵ_i of data from true model $y_i = f(x_i) + \epsilon_i$.
- Real-world examples:
  - Human mislabeling of test instance $\Rightarrow$ Ideal model will never predict it accurately.
  - Error during collection of temperature due to sensor malfunctioning.
- Cannot do anything about it even if seeded with knowledge about true model.

Tell-tale signs of overfitting

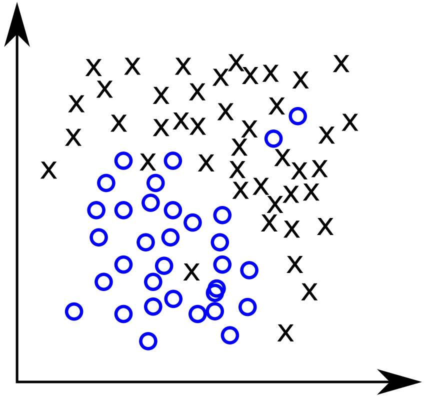
1. Same test instance obtains very different predictions $\rightarrow$ Indication that the training process is memorizing the nuances of the specific training data set, rather than learning patterns that generalize to unseen test instances.
2. The gap between the error of predicting training instances and unseen test instances is rather large.




Bias
Variance

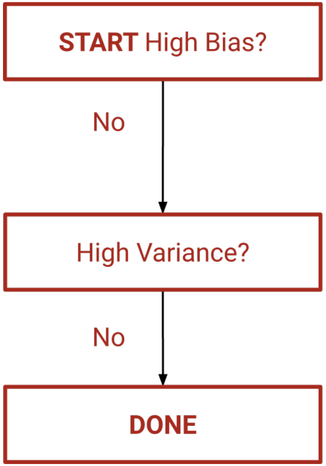
Train set error:	Dev Set Error

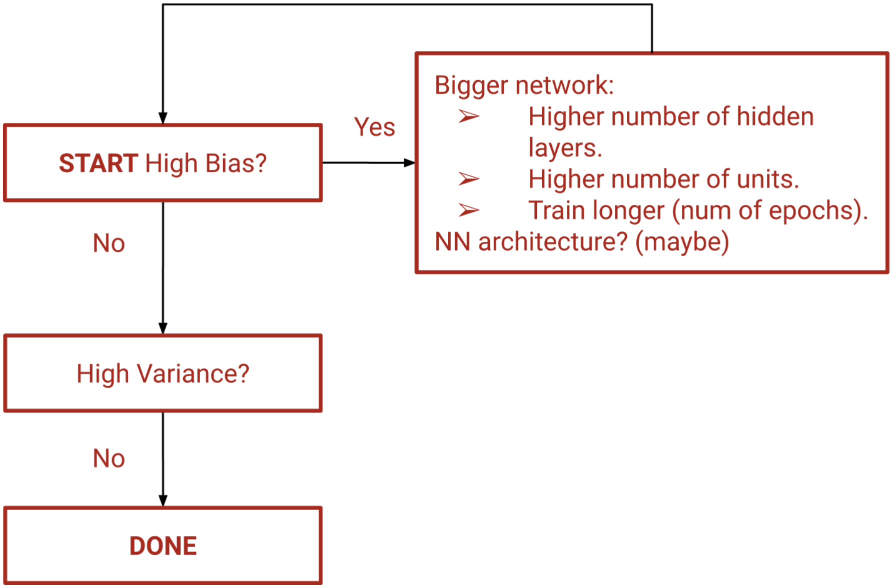
The worst of both worlds (high bias and high variance)

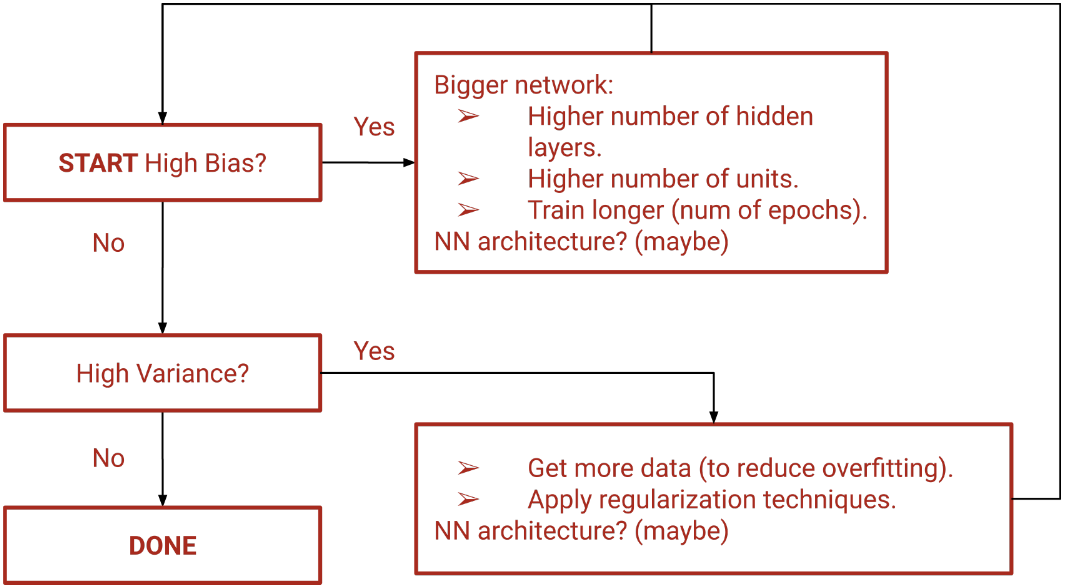


Key Takeaway of Bias-Variance Trade-Off

- A model with greater complexity might be *theoretically* more accurate (i.e., low bias).
  - But you have less control on what it might predict on a tiny training data set.
  - Different training data sets will result in widely *varying* predictions of same test instance.
  - Some of these must be wrong $\Rightarrow$ Contribution of model variance.
- *A more accurate model for infinite data is not a more accurate model for finite data.*
  - Do not use a sledgehammer to swat a fly!


START High Bias?
No
High Variance?
No
DONE


START High Bias?
Yes
Bigger network:
➢ Higher number of hidden layers.
➢ Higher number of units.
➢ Train longer (num of epochs).
NN architecture? (maybe)
No
High Variance?
No
DONE


START High Bias?
Yes
Bigger network:
➢ Higher number of hidden layers.
➢ Higher number of units.
➢ Train longer (num of epochs).
NN architecture? (maybe)
No
High Variance?
Yes
➢ Get more data (to reduce overfitting).
➢ Apply regularization techniques.
NN architecture? (maybe)
No
DONE

In "classical ML"(pre-DL):

- Tools capable of reducing one (say, bias) would generally increase the other (variance).

In Deep Learning:

- Bigger network: Lower bias keeping variance reasonably constant (with appropriate regularization).
- Get more data. Keeps bias reasonably constant whilst reducing variance.

The Bias-Variance Trade-Off

Assume

- x - Independent variable.
- y - Dependent variable.
- $f(x)$ - Describes the true underlying dependence of y on x.
- $y = f(x) + \epsilon$ - The result of $f(x)$ and random noise.
- ϵ - Random variable representing noise.
  - $\mathbb{E}[\epsilon] = 0$
  - $\text{var}[\epsilon] = \mathbb{E}[\epsilon^2] = \sigma_\epsilon^2$

Recall

Variance:

$$\mathrm{var}(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$$

$$= \mathbb{E}\left[X^2 - 2X\mathbb{E}[X] + \mathbb{E}^2[X]\right]$$

$$= \mathbb{E}\left[X^2\right] - 2\mathbb{E}^2[X] + \mathbb{E}^2[X]$$

$$= \mathbb{E}\left[X^2\right] - \mathbb{E}^2[X]$$

And also, since $\mathbb{E}[\epsilon] = 0$, we have:

$$\mathrm{var}(\epsilon) = \mathbb{E}\left[\epsilon^2\right] - \mathbb{E}^2\left[\epsilon\right]$$

$$\mathrm{var}(\epsilon) = \mathbb{E}\left[\epsilon^2\right] = \sigma^2$$

Goal

- Model the underlying real-life problem.
- I. e., find $\hat{f}$ s.t. $\hat{f}(x) \approx f(x)$ by reducing MSE $= \mathbb{E}\left[\left(Y - \hat{f}(X)\right)^2\right]$

Bias

- Difference of average value of prediction realizations (over different realizations of training data) to the true underlying function $f(x)$ for a given *unseen* test point, i.e.:
$$\text{bias}\left[\hat{f}(x)\right] = \mathbb{E}\left[\hat{f}(x)\right] - f(x)$$
- Notice that $\hat{f}(x)$ is a random variable affected by the randomness in which we obtain training data.

Variance

- Mean Squared Deviation of $\hat{f}(x)$ from its expected value $\mathbb{E}\left[\hat{f}(x)\right]$ over different realizations of training data, i. e.:

$$\text{var}\left(\hat{f}(x)\right) = \mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right]$$

Summary

- MSE $= \mathbb{E}_x \left[\left(y - \hat{f}(x) \right)^2 \right]$.
- bias $\left[\hat{f}(x) \right] = \mathbb{E} \left[\hat{f}(x) \right] - f(x)$.
- var $\left(\hat{f}(x) \right) = \mathbb{E} \left[\left(\hat{f}(x) - \mathbb{E} \left[\hat{f}(x) \right] \right)^2 \right]$

From the Mean Squared Error (MSE):

$$\mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}\left[\left(f(x) + \epsilon - \hat{f}(x)\right)^2\right]$$

$$= \mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2\right] + 2\mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)\epsilon\right] + \mathbb{E}\left[\epsilon^2\right]$$

Since expectation is linear and ϵ and $\hat{f}$ are independent;

$$= \mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2\right] + 2\mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)\right]\underbrace{\mathbb{E}\left[\epsilon\right]}_{=0} + \sigma_\epsilon^2$$

$$\text{MSE} = \mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2\right] + \sigma_\epsilon^2$$

$$\mathbb{E}\left[\left(f(x)-\hat{f}(x)\right)^2\right]=\mathbb{E}\left[\left(f(x)-\mathbb{E}\left[\hat{f}(x)\right]+\mathbb{E}\left[\hat{f}(x)\right]-\hat{f}(x)\right)^2\right]$$

$$=\mathbb{E}\left[\left(\left(f(x)-\mathbb{E}\left[\hat{f}(x)\right]\right)-\left(\hat{f}(x)-\mathbb{E}\left[\hat{f}(x)\right]\right)\right)^2\right]=$$

$$=\mathbb{E}\left[\underbrace{\left(\mathbb{E}\left[\hat{f}(x)\right]-f(x)\right)^2}_{\text{squared bias} = \text{cte}}+\left(\hat{f}(x)-\mathbb{E}\left[\hat{f}(x)\right]\right)^2-2\underbrace{\left(f(x)-\mathbb{E}\left[\hat{f}(x)\right]\right)}_{\text{cte}}\left(\hat{f}(x)-\mathbb{E}\left[\hat{f}(x)\right]\right)\right]$$

$$= \mathbb{E}\left[\underbrace{\left(\mathbb{E}\left[\hat{f}(x)\right] - f(x)\right)^2}_{\text{squared bias} = \text{cte}} + \left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2 - 2\underbrace{\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)}_{\text{cte}}\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\right]$$

$$= \underbrace{\left(\mathbb{E}\left[\hat{f}(x)\right] - f(x)\right)^2}_{\text{bias}^2[\hat{f}(x)]} + \underbrace{\mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)^2\right]}_{\text{var}[\hat{f}(x)]} - 2\left(f(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\underbrace{\mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}\left[\hat{f}(x)\right]\right)\right]}_{=\mathbb{E}[\hat{f}(x)]-\mathbb{E}[\hat{f}(x)]=0}$$

$$\boxed{\mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2\right] = \text{bias}^2\left[\hat{f}(x)\right] + \text{var}\left(\hat{f}(x)\right)}$$

Since

$$\mathsf{MSE} = \mathbb{E}\left[\left(y - \hat{f}(x)\right)^2\right] = \mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2\right] + \sigma_\epsilon^2$$

and we've just shown that

$$\mathbb{E}\left[\left(f(x) - \hat{f}(x)\right)^2\right] = \mathsf{bias}^2\left[\hat{f}(x)\right] + \mathsf{var}\left(\hat{f}(x)\right)$$

hence, we have

$$\boxed{\mathsf{MSE} = \mathsf{bias}^2\left[\hat{f}(x)\right] + \mathsf{var}\left(\hat{f}(x)\right) + \sigma_\epsilon^2}$$

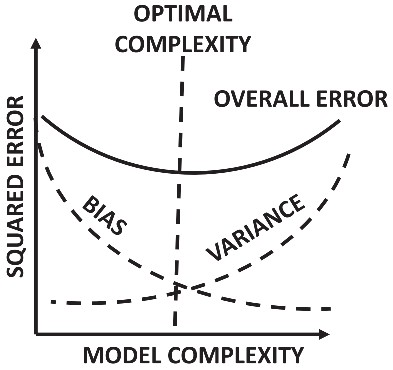
Bias-Variance Equation

- Let $E[MSE]$ be the expected mean-squared error of the fixed set of test instances over different samples of training data sets.

$$E[MSE] = \text{Bias}^2 + \text{Variance} + \text{Noise} \tag{2}$$

  - In linear models, the bias component will contribute more to $E[MSE]$.
  - In polynomial models, the variance component will contribute more to $E[MSE]$.
- We have a trade-off, when it comes to choosing model complexity!

The Bias-Variance Trade-Off



- Optimal point of model complexity is somewhere in middle.

Bias-Variance Trade-off: Setup

- Imagine you are given the true distribution $\mathcal{B}$ of training data (including labels).
- You have a principled way of sampling data sets $\mathcal{D} \sim \mathcal{B}$ from the training distribution.
- Imagine you create an infinite number of training data sets (and trained models) by repeated sampling.
- You have a *fixed* set $\mathcal{T}$ of unlabeled test instances.
  - The test set $\mathcal{T}$ does not change over different training data sets.
  - Compute prediction of each instance in $\mathcal{T}$ for each trained model.

Informal Definition of Bias

- Compute averaged prediction of each test instance x over different training models $g(x, \mathcal{D})$.
- Averaged prediction of test instance will be different from true (unknown) model $f(x)$.
- Difference between (averaged) $g(x, \mathcal{D})$ and $f(x)$ caused by erroneous assumptions/simplifications in modeling $\Rightarrow$ Bias
  - **Example:** Linear simplification to polynomial model causes bias.
  - If the true (unknown) model $f(x)$ were an order-4 polynomial, and we used any polynomial of order-4 or greater in $g(x, \mathcal{D})$, bias would be 0.

Informal Definition of Variance

- The value $g(x, \mathcal{D})$ will vary with $\mathcal{D}$ for fixed x.
  - The prediction of the same test instance will be different over different trained models.
- All these predictions cannot be simultaneously correct $\Rightarrow$ Variation contributes to error
- Variance of $g(x, \mathcal{D})$ over different training data sets $\Rightarrow$ Model Variance
  - **Example:** Linear model will have low variance.
  - Higher-order model will have high variance.

Bias-Variance Equation

- Let $E[MSE]$ be the expected mean-squared error of the fixed set of test instances over different samples of training data sets.

$$E[MSE] = \text{Bias}^2 + \text{Variance} + \text{Noise} \tag{3}$$

  - In linear models, the bias component will contribute more to $E[MSE]$.
  - In polynomial models, the variance component will contribute more to $E[MSE]$.
- We have a trade-off, when it comes to choosing model complexity!

Cross-Fold Validation

Model Generalization in Neural Networks

- The recent success of neural networks is made possible by increased data.
  - Large data sets help in generalization.
- In a neural network, increasing the number of hidden units in intermediate layers tends to increase complexity.
- Increasing depth often helps in reducing the number of units in hidden layers.
- Proper design choices can reduce overfitting in complex models $\Rightarrow$ Better to use complex models with appropriate design choices

How to Detect Overfitting

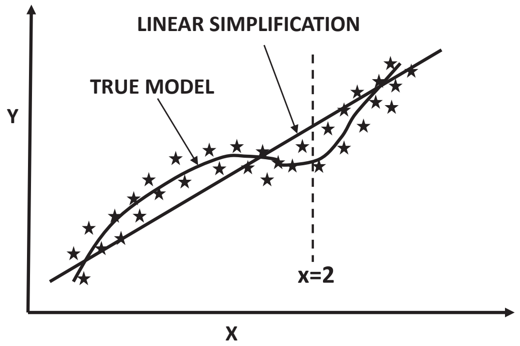
- The error on test data might be caused by several reasons.
  - Other reasons might be bias (underfitting), noise, and poor convergence.
- Overfitting shows up as a large gap between in-sample and out-of-sample accuracy.
- First solution is to collect more data.
  - More data might not always be available!

Penalty-Based Regularization

- Key techniques to improve generalization in NNs:
  - Penalty-based regularization.
  - Constraints like shared parameters.
  - Using ensemble methods like *Dropout*.
  - Adding noise and stochasticity to input or hidden units.

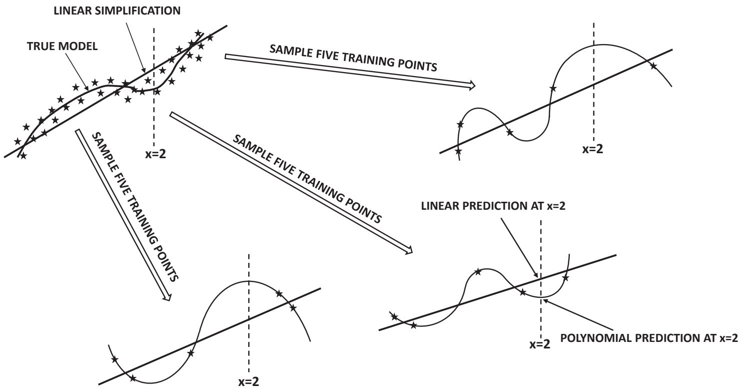
Penalty-Based Regularization

Revisiting Example: Predict y from x



- **First impression:** Polynomial model such as $y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4$ is "better" than linear model $y = w_0 + w_1 x$.
  - However, with less data, using the linear model is better.

Penalty-Based Regularization


LINEAR SIMPLIFICATION
TRUE MODEL
x=2
SAMPLE FIVE TRAINING POINTS
x=2
SAMPLE FIVE TRAINING POINTS
SAMPLE FIVE TRAINING POINTS
LINEAR PREDICTION AT x=2
x=2
POLYNOMIAL PREDICTION AT x=2
x=2

- Zero error on training data but wildly varying predictions of $x = 2$

Economy in Parameters

- A lower-order model has economy in parameters.
  - A linear model uses two parameters, whereas an order-4 model uses five parameters.
  - Economy in parameters discourages overfitting.
- Choosing a neural network with fewer units per layer enforces economy.
- Reducing the number of parameters is a *hard* penalty.
- We can also penalize parameters in a *soft* way.

Summary

L_2 Regularization and Soft Economy vs Hard Economy

- Fixing the architecture up front is an inflexible solution.
- A softer solution uses a larger model but imposes a (tunable) penalty on parameter use.

$$\hat{y} = \sum_{i=0}^{d} w_i x^i \tag{4}$$

- Loss function: $L = \sum_{(x,y)\in\mathcal{D}} (y - \hat{y})^2 + \underbrace{\lambda \cdot \sum_{i=0}^{d} w_i^2}_{L_2-\text{Regularization}}$
- The (tuned) value of λ decides the level of regularization.
- Softer approach with a complex model performs better!

Effect on Updates

- For learning rate α, effect on update is to multiply parameter with $(1 - \alpha\lambda) \in (0, 1)$.

$$w_i \Leftarrow w_i(1 - \alpha\lambda) - \alpha\frac{\partial L}{\partial w_i}$$

  - **Interpretation:** Decay-based forgetting!
- Unless a parameter is important, it will have small absolute value.
  - Model decides what is important.
  - Better than inflexibly deciding up front.
- A forgetting mechanism prevents a model from *memorizing* the training data, because only significant and repeated updates will be reflected in the weights.

Summary

L_1-Regularization

- In L_1-regularization, an L_1-penalty is imposed on the loss function.

$$L = \sum_{(x,y)\in\mathcal{D}} (y-\hat{y})^2 + \lambda \cdot \sum_{i=0}^{d} |w_i|_1$$

- Update has slightly different form (define the update equation at least for the case when $w_i \neq 0$:

$$w_i \Leftarrow w_i - \alpha\lambda s_i - \alpha\frac{\partial L}{\partial w_i}$$

- The value of s_i is the partial derivative of $|w_i|$ w.r.t. w_i:

$$s_i = \begin{cases} -1 & w_i < 0 \\ +1 & w_i > 0 \end{cases}$$

Weight space loss term $\mathcal{L}(y, \hat{y})$ and L_1- and L_2- regularization terms

L_1- or L_2-Regularization?

- L_1-regularization leads to sparse parameter learning.
  - Zero values of w_i can be dropped. Therefore, (slightly, in practice)less memory is required to store the model.
  - Equivalent to dropping edges from neural network.
- L_2-regularization generally provides better performance.
- L_2 is differentiable and can be used in different techniques.

Summary

Connections with Noise Injection

- *L_2-regularization with parameter λ is equivalent to adding Gaussian noise with variance λ to input.*
  - **Intuition:** Bad effect of noise will be minimized with simpler models (smaller parameters).
- Result is only true for single layer network (linear regression).
  - Main value of result is in providing general intuition.

Summary

Penalizing Hidden Units

- One can also penalize hidden units (*activations*).
- Applying L_1-penalty leads to sparse activations.
- Straightforward modification of backpropagation.
  - Penalty contributions from hidden units are picked up in backward phase.

$$L' = L + \lambda \sum_{i=1}^{M} |h_i| \tag{5}$$

Where:

- M is the total number of **units** in the network.
- h_i is the value of the ith hidden unit.
- λ is the regularization parameter.

Ensemble Methods

- Inspired in Bias-Variance trade-off.
- Try to reduce either the bias or the variance without affecting the other component.
- Ensemble methods are commonly used in Machine Learning.
- Two examples:
  - *Bagging* – Variance reduction.
  - *Boosting* – Bias reduction.

Ensemble Methods

- Most ensemble methods in NNs are focused on variance reduction.
- This is because neural networks are valued for their ability to build arbitrarily complex models with relatively low bias.
- But arbitrarily complex models lead to high variance:

OVERFITTING .

Therefore, the goal of most ensemble methods in NN is variance reduction.

Summary

Bagging and Sub-sampling

- If a sufficient number of samples is available, after all, the variance of most types of statistical estimates can be asymptotically reduced to zero.
- One approach: Predict one instance repeatedly using different training data sets.
- With a sufficient large number of training data sets is used, the variance can be reduced to zero (infinite source of data).
- Although we don't have infinite number of instances, an imperfect simulation of the aforementioned methodology has still better variance characteristics than a single execution of the model on the entire training data set.

Bagging and Sub-sampling

- The predictions on a particular test instance, which are obtained from the models built with different training sets, are then averaged to create the final prediction.
- One can average either the real-valued prediction (e.g., probability estimates of class labels) or the discrete predictions.
- In the case of real-valued predictions, better results are sometimes obtained by using the median of the values.

Summary

Parametric Model Selection and Averaging

- Problems:
  - Large number of hyper-parameters and configurations.
  - Sensitivity to some choices such as activation functions.
- Strategy: Hold out a portion of the training data.
- Select the model out of the pool providing highest performance.

Summary

Basic Dropout Training Procedure

- For each training instance do:
  - Sample each node in the network in each layer (except output layer) with probability p.
  - Keep only edges for which both ends are included in network.
  - Perform forward propagation and backpropagation only on sampled network.
- Note that weights are shared between different sampled networks.
- A different neural network is used for every small mini-batch of training examples.
- The number of NNs is rather large in Dropout.

Basic Dropout Testing Procedures

- First procedure:
  - Perform repeated sampling (like training) and average results.
  - Geometric averaging for probabilistic outputs (averaging log-likelihood)
- Second procedure with *weight scaling inference rule* (more common):
  - Multiply weight of each outgoing edge of a sampled node i with its sampling probability p_i.
  - Perform single inference on full network with down-scaled weights.

Why Does Dropout Help?

- By dropping nodes, we are forcing the network to learn without the presence of some inputs (in each layer).
- Will resist co-adaptation, unless the features are truly synergistic.
- Will create many (smaller) groups of self-sufficient predictors.
- Many groups of self-sufficient predictors will have a model-averaging effect.

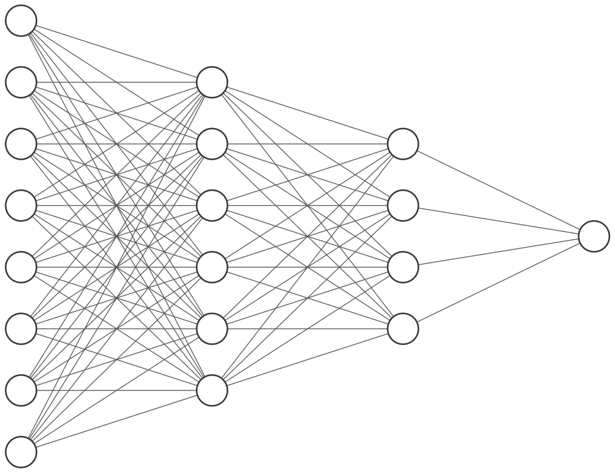
The Regularization Perspective

- One can view the dropping of a node as the same process as adding masking noise.
  - Noise is added to both input and hidden layers.
- Adding noise is equivalent to regularization.
- Forces the weights to become more spread out.
  - Updates are distributed across weights based on sampling.

Practical Aspects of Dropout

- Typical dropout rate (i.e., probability of exclusion) is somewhere between 20% to 50%.
- Better to use a larger network with Dropout to enable learning of independent representations.
- Dropout is applied to both input layers and hidden layers.
- Large learning rate with decay and large momentum.
- Impose a max-norm constraint on the size of network weights.
  - Norm of input weights to a node upper bounded by constant c.

Feature Co-Adaptation



One-Way Adaptation

- Consider a single-hidden layer neural network.
  - All edges into and out of half the hidden nodes are fixed to random values.
  - Only the other half are updated during backpropagation.
- Half the features will adapt to the other half (random features).
- Feature co-adaptation is natural in neural networks where rate of training varies across different parts of network over time.
  - Partially a manifestation of training inefficiency (over and above true synergy).

Why is Feature Co-Adaptation Bad?

- We want features working together only when essential for prediction.
  - We do not want features adjusting to each other because of inefficiencies in training.
  - Does not generalize well to new test data.
- We want *many* groups of minimally essential features for robust prediction $\Rightarrow$ Better redundancies.
- We do not want a *few* large and inefficiently created groups of co-adapted features.

Feature Co-Adaptation

- The process of training a neural network often leads to a high level of dependence among features.
- Different parts of the network train at different rates:
  - Causes some parts of the network to adapt to others.
- This is referred to as feature co-adaptation.
- Uninformative dependencies are sensitive to nuances of specific training data $\Rightarrow$ **OVERFITTING** .

Summary

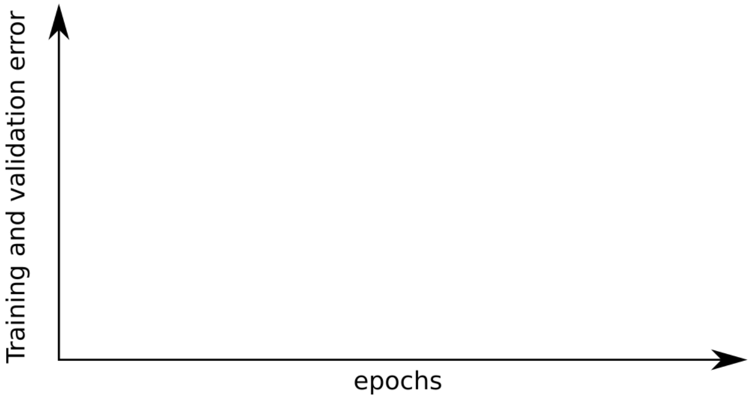
Data Perturbation Ensembles

- Most of the ensemble techniques discussed so far are either sampling-based ensembles of model-centric ensembles.
- *Dropout* can be considered an ensemble that adds noise to the data in an indirect way.
- Simplest case $\rightarrow$ noise is added to input data and weights are trained on the disturbed scenario.
- Repeat the process and average results.
- This is the *generci ensemble method*, not specific to neural networks.
- If one wants to add noise to hidden-layers, it must be carefully calibrated.
- Dropout *indirectly* adds noise to hidden layers by randomly dropping nodes.

Data Perturbation Ensembles

- Data augmentation can often greatly improve the accuracy of a learner by increasing its generalization power.
- But they are not perturbation schemes because the augmented examples are created with a calibrated procedure and understanding of the domain at hand.

Early Stopping

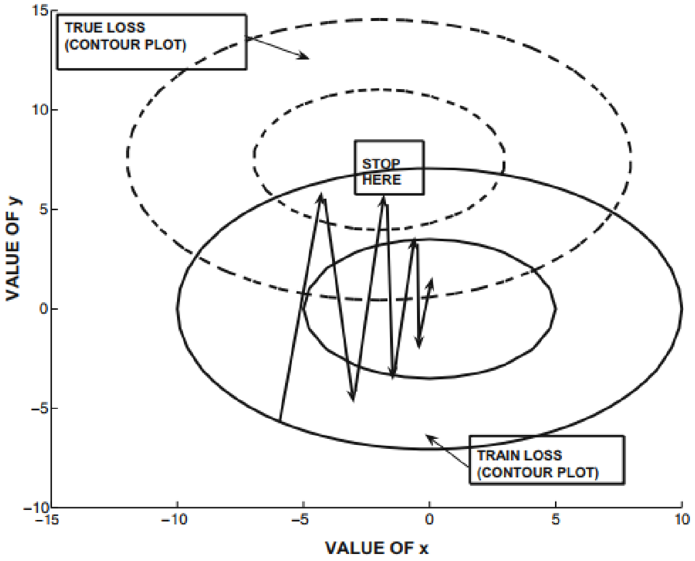


Early Stopping

Motivation:

- We use optimization to train NNs until convergence.
- Optimizes the loss on training data.
- Not necessarily on the out-of-sample test data.
- Final steps cause overfitting and generalization problems.
- Almost always used.

Early Stopping



Orthogonalization

Two tasks:

1. Optimize objective function.
   - Gradient descent, ...
2. Don't overfit.
   - Regularization, more samples, etc. . . .

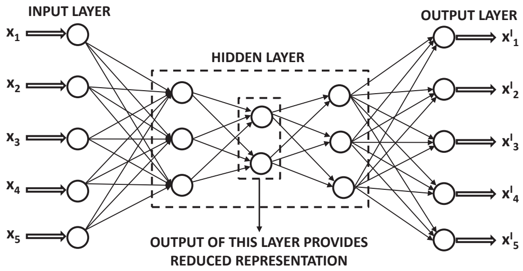
Early stopping

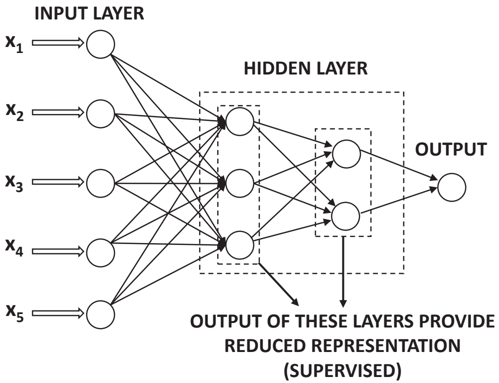
Couples the two tasks such that one can't work on each one separately.

Unsupervised Pre-training – Importance of Initialization

- Bad initializations can lead to unstable convergence.
- Typical approach is to initialize to a Gaussian with variance $1/r$, where r is the indegree of the neuron.
  - Xavier initialization uses both indegree and outdegree.
- Pretraining goes beyond these simple initializations *by using the training data*.

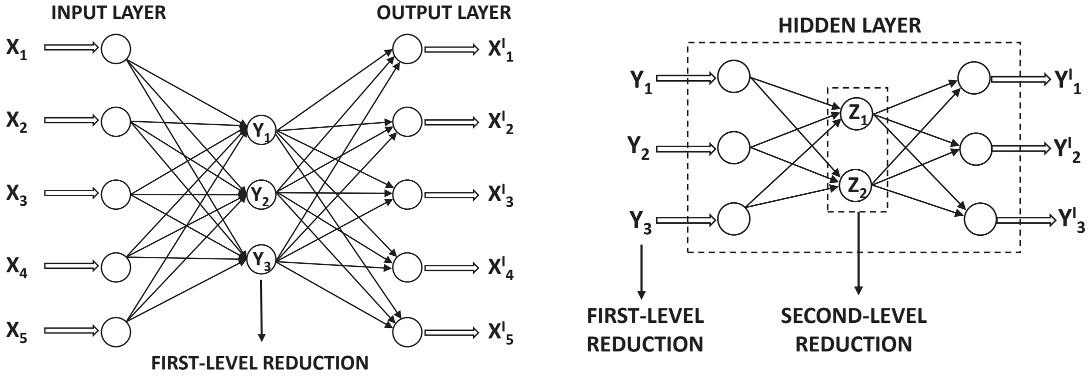
Types of Base Applications




INPUT LAYER
x_1
x_2
x_3
x_4
x_5
HIDDEN LAYER
OUTPUT
OUTPUT OF THESE LAYERS PROVIDE REDUCED REPRESENTATION (SUPERVISED)

- Both the two neural architectures use almost the same pretraining procedure

Layer-Wise Pretraining a Deep Autoencoder



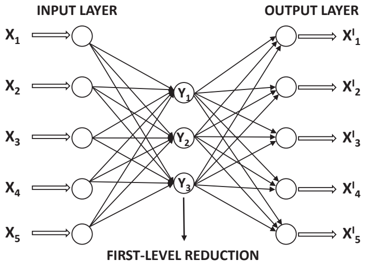
- Pretraining deep autoencoder helps in convergence issues

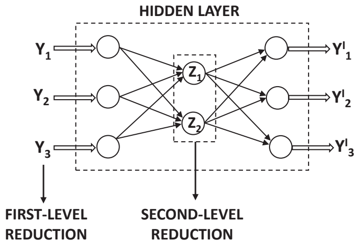
Types of Pretraining

- *Unsupervised pretraining:* Use training data **without** labels for initialization.
  - Improves convergence behavior.
  - Regularization effect.
- *Supervised pretraining:* Use training data **with** labels for initialization.
  - Improves convergence but might overfit.
- Focus on unsupervised pretraining.

Pretraining a Supervised Learner

- For a supervised learner with k hidden layers:
  - Remove output layer and create an autoencoder with $(2k-1)$ hidden layers.
  - Pretrain autoencoder as discussed in previous slide.
  - Keep only weights from encoder portion and cap with output layer.
  - Pretrain only output layer.
  - Fine-tune all layers.



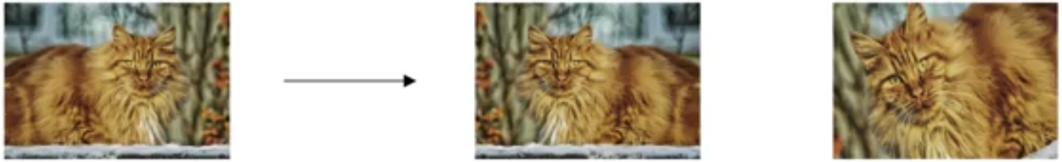


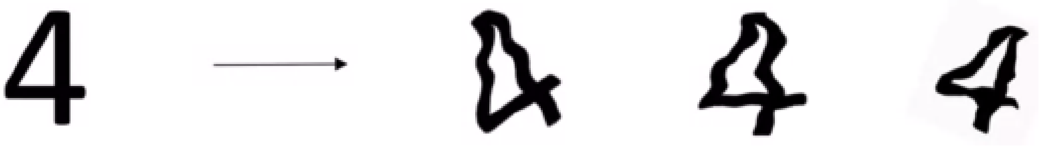
Why Does Pretraining Work?

- Pretraining already brings the activations of the neural network to the manifold of the data distribution.
- Features correspond to repeated patterns in the data.
- Fine-tuning learns to combine/modify relevant ones for inference.
  - Pretraining initializes the problem closer to the basin of global optima.
  - Hinton: "*To recognize shapes, first learn to generate images.*"

Data Augmentation

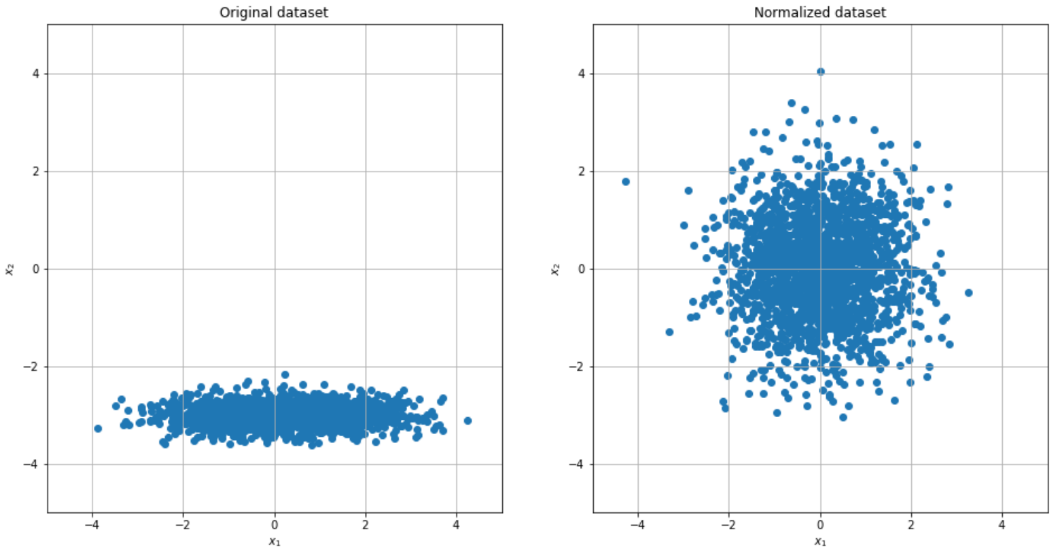
Data Augmentation

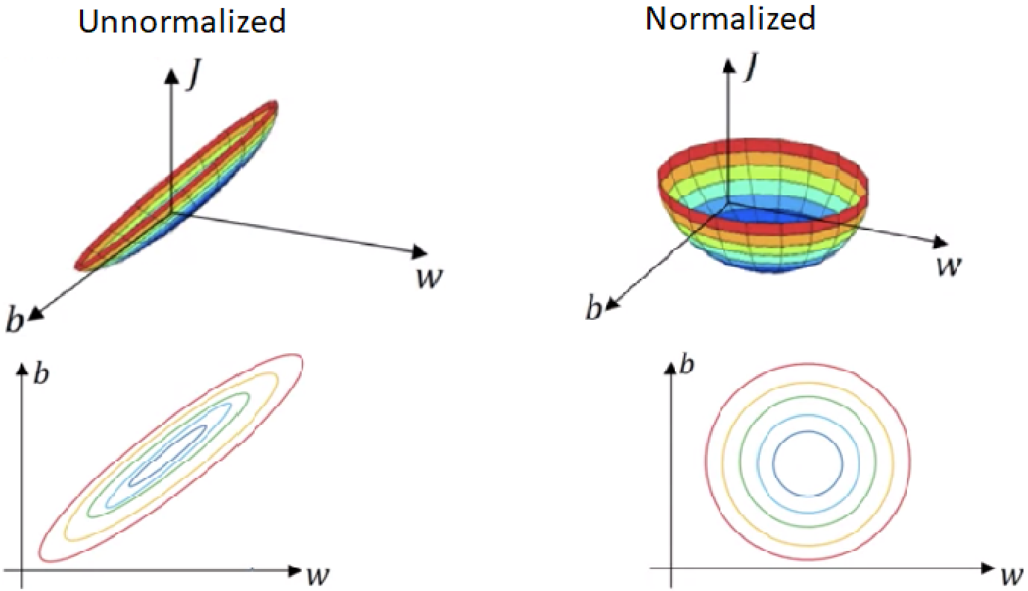




Maybe not flipping vertically.

Normalizing Input Features


Original dataset
Normalized dataset
x_1
x_2
−4
−2
0
2
4


Unnormalized
Normalized
J
w
b
b
w
J
w
b
b
w

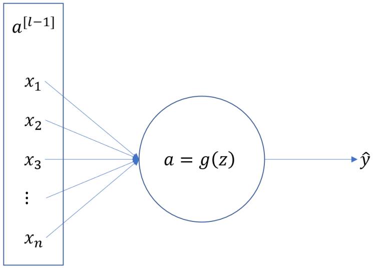
Vanishing/ Exploding Gradients

When training a very deep networks, your derivatives (or slopes) can sometimes get either;

- Very big, or;
- Very small (exponentially),

which makes training difficult.

One way to tackle exploding and vanishing Gradient Descent is to perform an adequate weight initialization.



$$z = \sum_{i=1}^{n} w_i x_i$$

For a large n one wants a smaller w. Set var$[w_i] = 1/n$.

Other variants

- For ReLU activation function:

$$w = \text{randn}(\text{shape}) \times \sqrt{\frac{2}{n^{[l-1]}}}$$

- He initialization: For TANH activation function, use

$$w = \text{randn}(\text{shape}) \times \text{tanh}\sqrt{\frac{1}{n^{[l-1]}}}$$

- Xavier initialization:

$$w = \text{randn}(\text{shape}) \times \sqrt{\frac{2}{n^{[l+1]} + n^{l-1}}}$$

Thank you!

tvieira@ic.ufal.br