Lecture 14:

Deep Learning and Regularization

Week of February 27, 2023

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Spring 2023

Prof. Teresa Head-Gordon

Departments of Chemistry, Bioengineering, Chemical and Biomolecular Engineering

Summary Previous Lecture

Dimensionality reduction is the process of reducing the data by only keeping the most relevant features from the original dataset necessary for unsupervised learning, where the only information is the data itself (i.e. with no corresponding known outcome or labelled data).

Principal Component Analysis (PCA): It is a feature extraction method that combines the input variables to capture the greatest amount of variance in the data.

Multiple methods are used to determine the number of principle components to keep, viewed as an additional step of feature selection.

Self Organizing Maps (SOM): Is a competitive learning algorithm that activates neurons as centers for detecting similar data features and their relationships. Can be used for feature selections to eliminate feature correlations.

Deep Learning

Purpose of Today's Lecture:

What is meant by Deep learning? Deep learning extends the simple and multi-layer perceptron beyond shallow networks, i.e. well beyond no or one single hidden layer. By increasing the number of hidden layers it can predict outcomes for more difficult regression and classification problems, also using error correction methods such as back-propagation.

What is different about Deep learning in practice? Deep learning requires a careful consideration of many heuristics

- Loss Functions
- Capacity
- L1 and L2 Regularization
- Training and Data
- Hidden Transformations
- Distribution shifts
- Architectures
- Data shifts

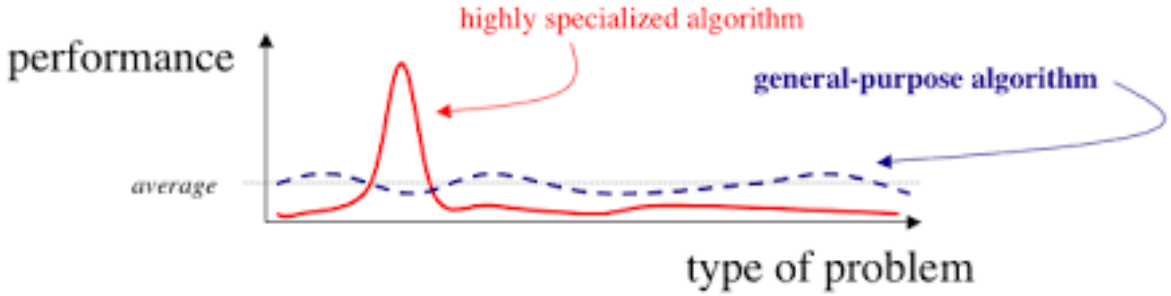
To address the bias-variance tradeoffs

No Free Lunch Theorem

There is a universal approximation theorem that states that a single hidden layer with many perceptrons and suitable activation functions can represent any function of $\{\boldsymbol{x}\}$ – no matter how difficult – to predict $\boldsymbol{f}(\boldsymbol{y}|\{\boldsymbol{x}\})$.

However, what is not guaranteed is a procedure for how to learn that representation and thus perform well on previously unobserved inputs. Good generalization tells us that our ANN has "understood" and can reproduce the mapping function, whereas poor generalization it has not "learnt" the representation.

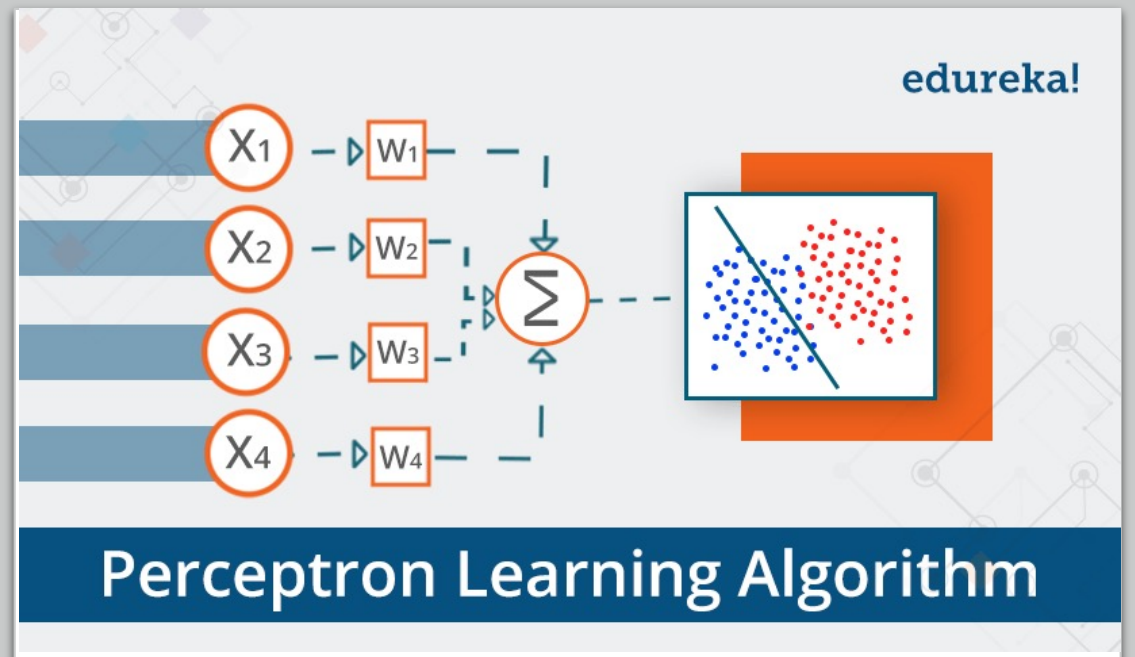
The No Free Lunch theorem states that there is nothing particularly magical about deep learning, but instead is a specific design of a meta-heuristic algorithm to perform well on a specific task or be more general purpose.

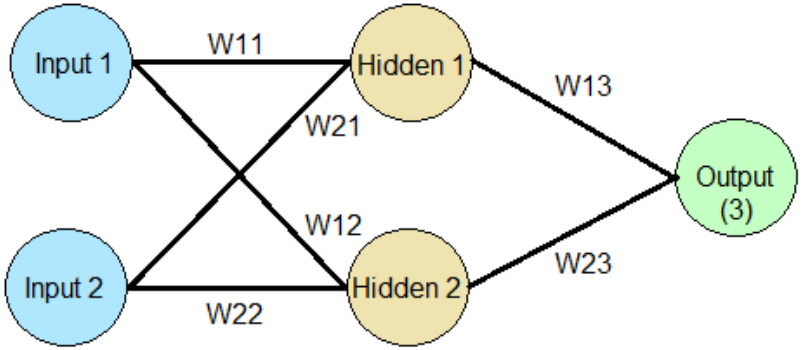


Perceptron vs. Deep Learning

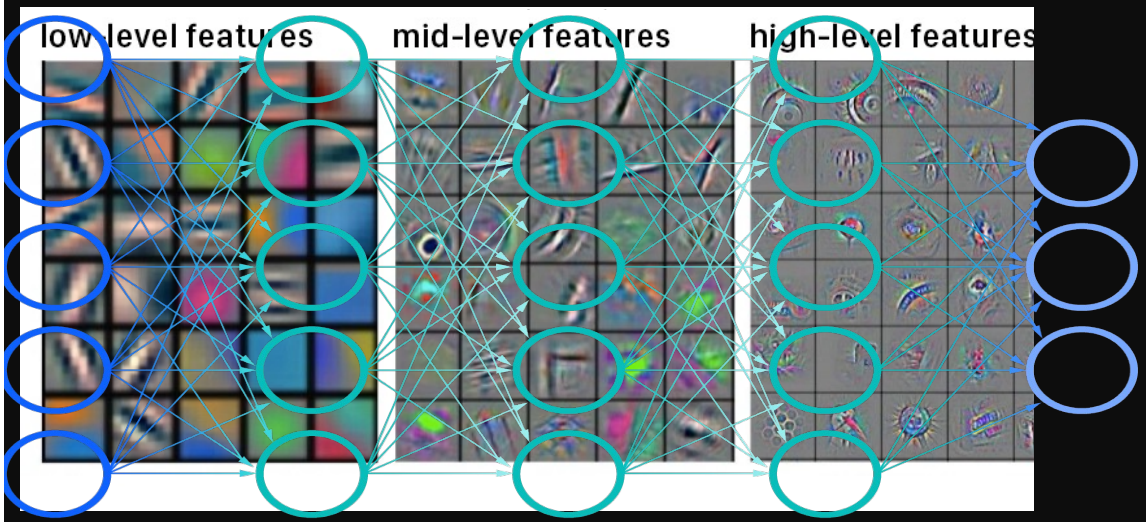
Remember that the simple perceptron is capable of doing relatively simple tasks such as linear regression or logistic regression for a single decision plane or linear separator.

We saw that the introduction of a hidden layer of neurons pre-processed or transformed the feature data through an intermediate step – albeit a very simple one – in order to "learn" more complicated tasks such as Boolean XOR





Deep Learning is Pattern Matching

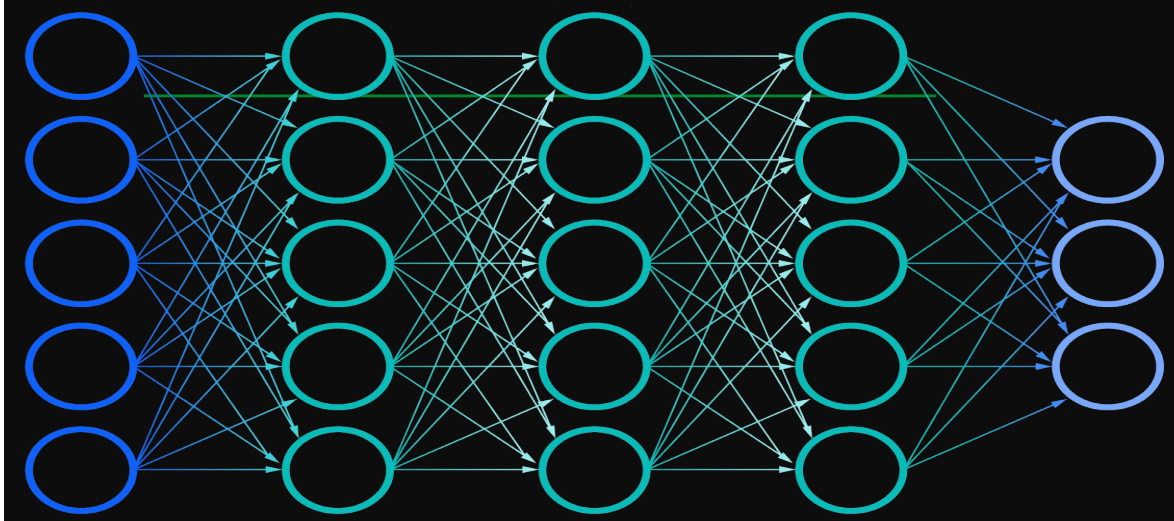


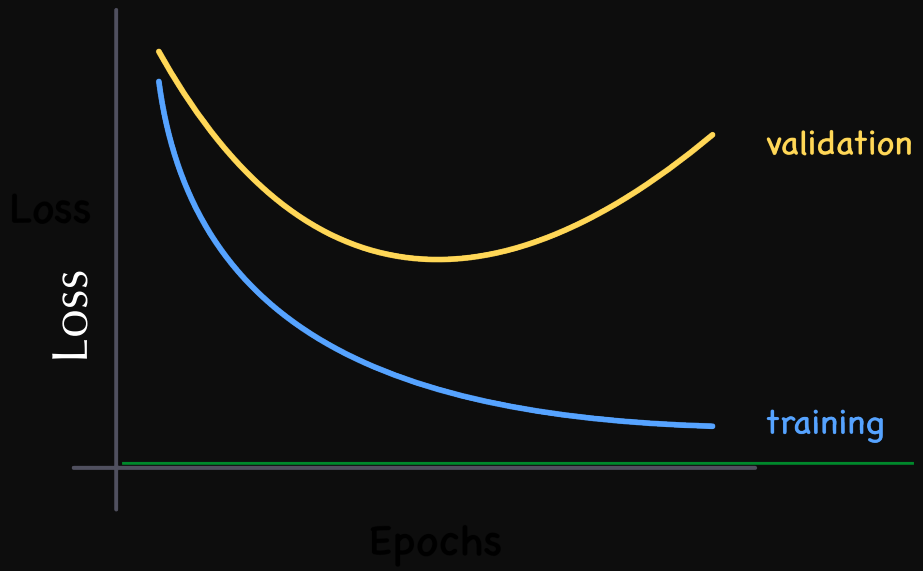
The input data are low-level features and as the data moves forward through the network, each subsequent layer extracts features at a higher level.

Over the course of training, the decision boundaries that it learns will try to adapt to the distribution of the training data and its transformations. I.e. deep learning transforms the input space such that the data are forced to be linearly separable at the end.

The goal of deep learning is to most reliably discover the function $\boldsymbol{f}(\boldsymbol{y}|\{\boldsymbol{x}\})$, or at least a good approximation to it, by building transformations of the input data $\{\boldsymbol{x}\}$ through multiple layers to discover patterns that determine $\boldsymbol{f}(\boldsymbol{y}|\{\boldsymbol{x}\})$.

Deep Learning and Generalization





"Any modification we make to the learning algorithm that is intended to reduce its test error but not its training error." - Goodfellow

"Any supplementary technique that aims at making the model... produce better results on the test set" - Kukacka et al.

Loss Functions

The learning process involves presenting the network with labeled data on which to train. We do this by minimizing error of a loss function. Loss functions are best determined by ML task: typically (non-linear) regression and (multi-) classification.

Regression: Maximum likelihood model for a Gaussian PDF is mean squared error – appropriate for regression problems and when labelled O is Gaussian distributed.

$$C = \frac{1}{N}\sum_{\mu}^{N}\sum_{j}^{J}\left[O_j^{\mu} - \hat{y}_j^{\mu}\right]^2$$

This loss function is very sensitive to outliers (squared quantity).

Can also try the Mean Absolute Error (MAE) which is more robust to outliers.

$$C = \frac{1}{N}\sum_{\mu}^{N}\sum_{j}^{J}\left|O_j^{\mu} - \hat{y}_j^{\mu}\right|$$

Loss Functions

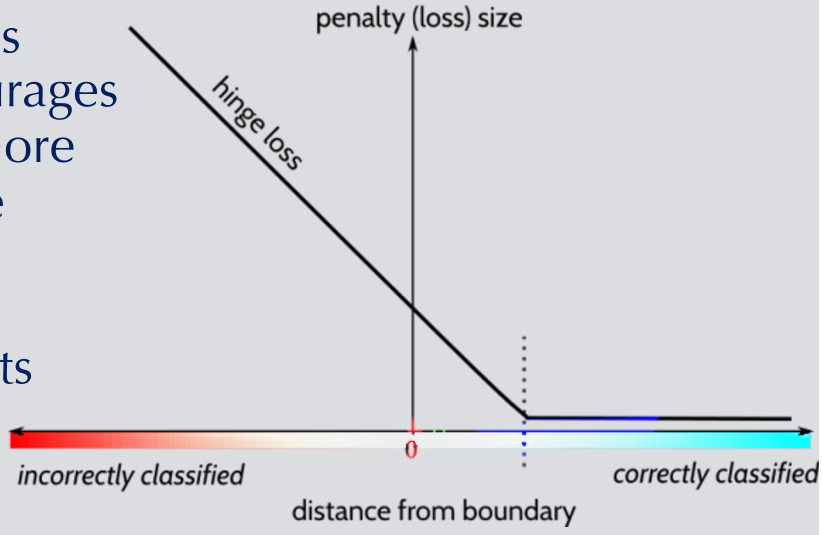
Classification: Maximum entropy or SoftMax takes only the additional information that is given to produce a probability distribution that is the most uncertain, i.e. constrained only on the information provided. Cross entropy (related to maximum entropy) cost function

$$C = -\sum_{x,y} p(O_k|x_i)\ log\ q(y_k|x_i)$$

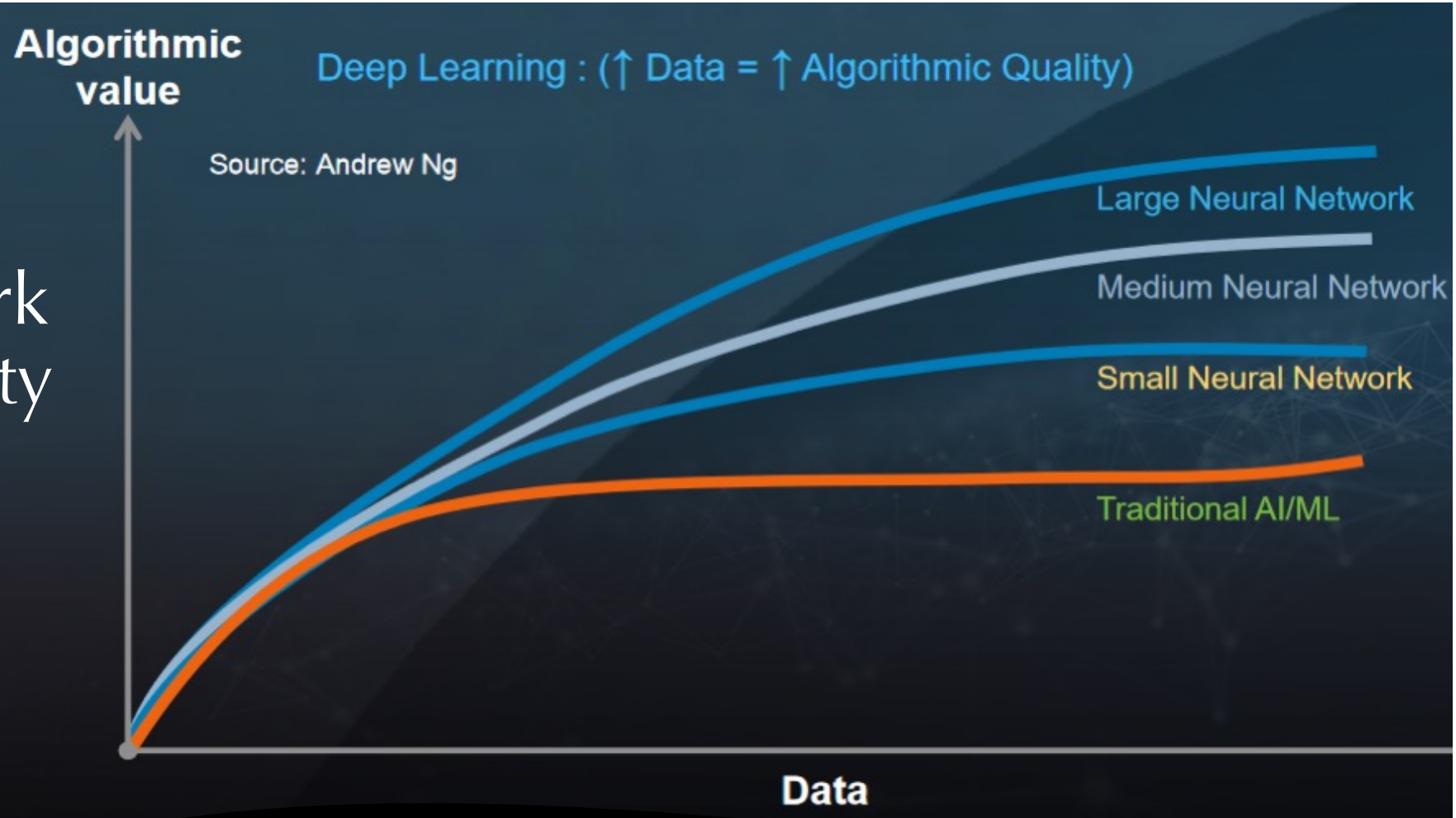
where $q(y_k|x_i)$ is our model guess on multi-category classification.

Hinge Loss: Primarily developed for binary target values that are in the set {-1, 1}. The hinge loss function encourages model predictions to have the correct sign, assigning more error when there is a difference in the sign between the actual and predicted class values.

Preferred if want to maximize the distance of data points and the decision margin boundary (no uncertainty!)



Network Capacity



Capacity is the ability of a deep learning model to fit a wide variety of functions. Often quantified informally by the number of free parameters or to take on more data.

Effective capacity is influenced by architecture, loss function, and numerous other factors.

Factors determining how well an ML algorithm will perform are its ability to:

1. Make the training error loss small
2. Make gap between training and test errors small

Bias-Variance Tradeoff

This can be analyzed from the LSE loss over our dataset that we derived from the maximum likelihood of a Gaussian PDF

$$LSE = \frac{1}{2}\sum_{\mu}^{N}\sum_{j}^{J}\left[O_j^{\mu} - \hat{y}_j^{\mu}\right]^2$$

where $O = f(\{x\}) + \epsilon$ and $\hat{y} = f'(\{x\})$. Thus LSE is actually measuring

$$LSE = (E[\hat{y}] - O)^2 + [E([\hat{y}] - \hat{y})]^2 + \sigma_{\epsilon}^2$$

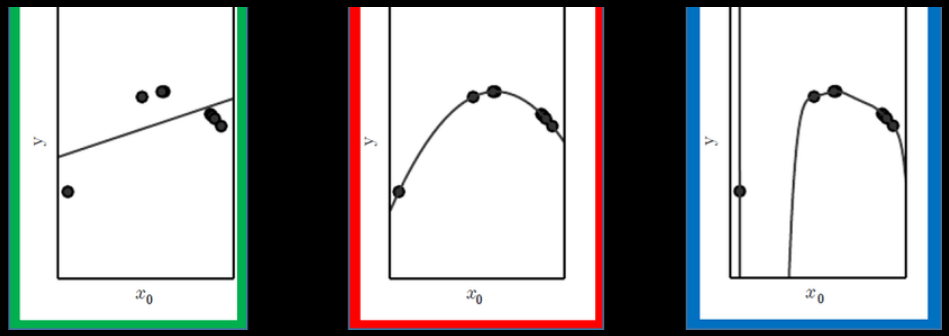
Bias variance intrinsic error

Where $E[\hat{y}]$ measures capacity of the model; all ML has intrinsic error!

To state the goal of generalization in a different way, we are trying to control the bias and variance tradeoff of our mapping function $\hat{y}$

Bias-Variance Tradeoff

Consider the data that is meant to represent a quadratic function



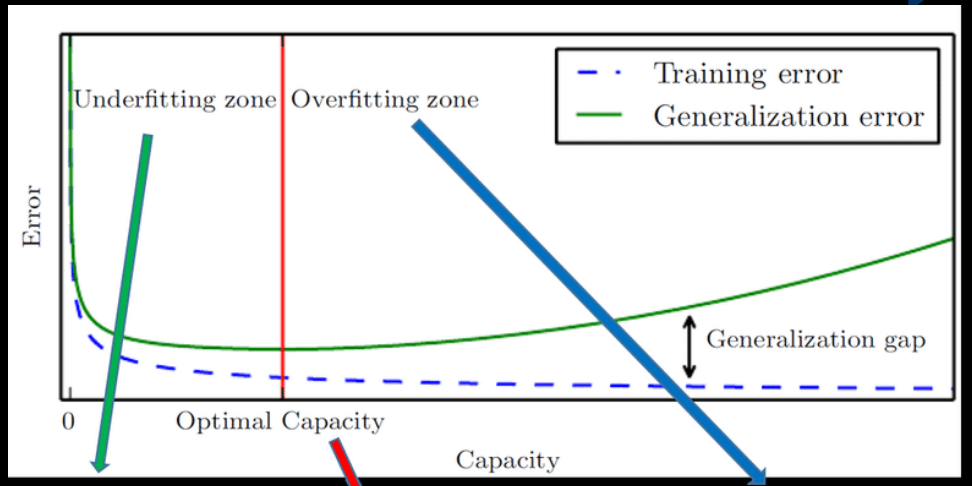
$$LSE = (E[\hat{y}] - O)^2 + [E([\hat{y}] - \hat{y})]^2 + \sigma_\epsilon^2$$

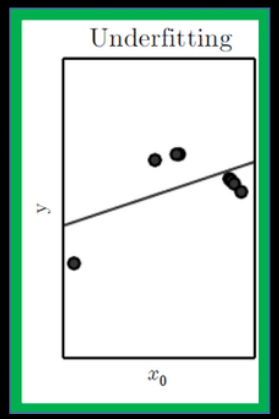
High bias. Network has a linear representation: low capacity and underfits. It will perform badly on training and testing

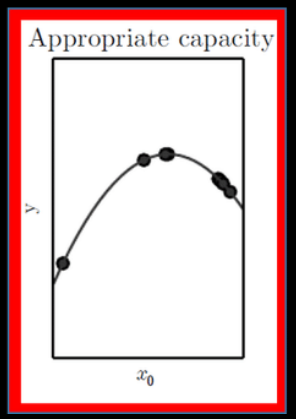
High variance. Network has higher n-order polynomial representation: network overfits and generalizes poorly

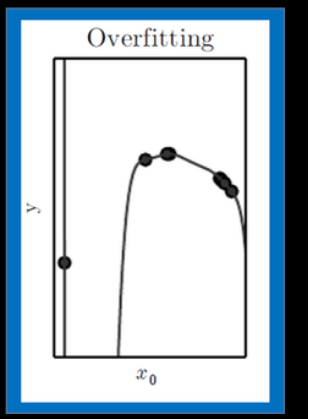
Bias-Variance Tradeoff

We can control whether a model is more likely to overfit or underfit by altering its capacity.









Learning Curves

Learning curves can be used to identify high variance from high bias problems.

Divide data into training, validation, and test set. While training the model, take the current set of weights and feed forward through the network to collect the validation or "generalization" error.

High bias models show large error in both training and validation sets and errors may approximate each other (since no overfitting). More data does not help, but add more capacity in the model.

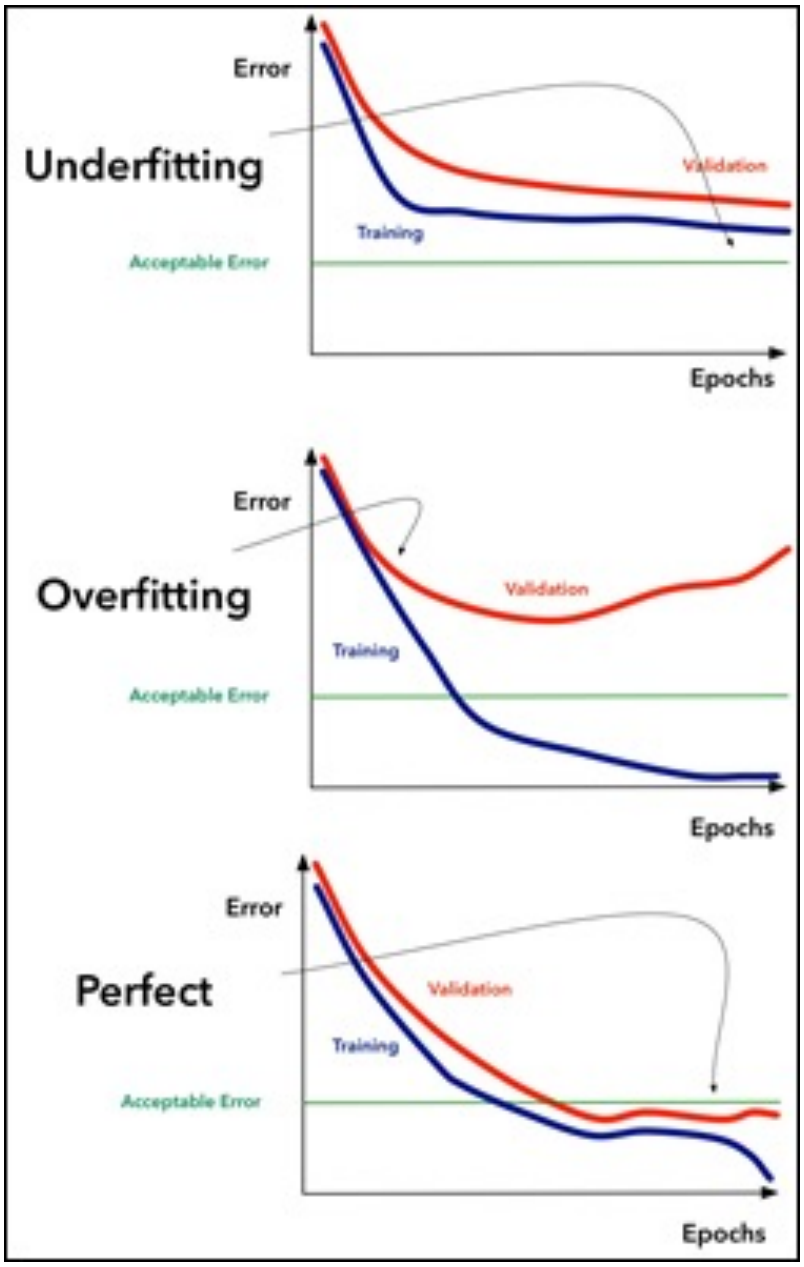
- Add more layers
- Try additional/completely new features
- Add feature transformations: hypothesis model

$$y = w^T x + b$$

implies hypothesis of linear functions of x_i vs

$$y = (w^T x)^2 + w^T x + b$$

hypothesis of quadratic functions of x_i



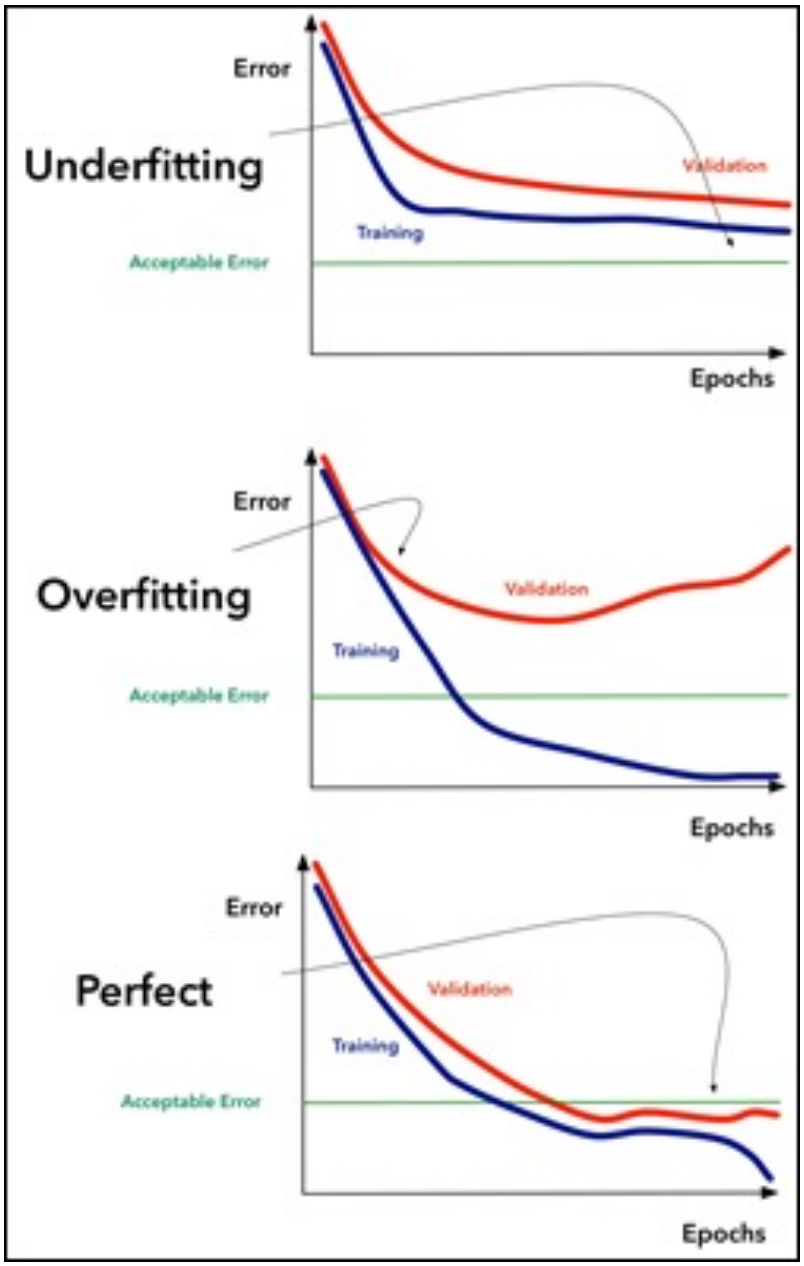
Learning Curves

Learning curves can be used to identify high variance from high bias problems.

Divide data into training, validation, and test set. While training the model, take the current set of weights and feed forward through the network to collect the validation or "generalization" error.

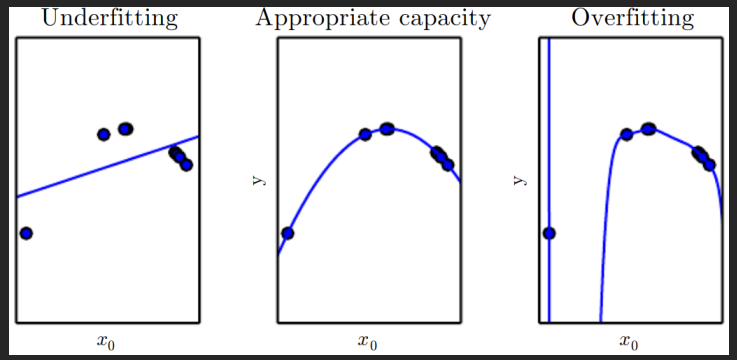
High variance models usually have low training error but shows a large gap with validation error (overfitting). More capacity does not help!

- Get more data to train on
- Try smaller sets of features
- Add feature transformations
- Benefits from hyperparameter tuning



L^2 and L^1 Regularization

Consider the data that is meant to represent a quadratic function



Can reduce overfitting in OLS by limiting the size of polynomial coefficients.

—

In ML, add a L regularization term to the loss function

L^2 regularization (ridge regression):

$$LSE = \frac{1}{2}\sum_{\mu}^{N}\sum_{j}^{J}\left[O_j^{\mu} - \hat{y}_j^{\mu}\right]^2 + \frac{\boldsymbol{\alpha}}{\boldsymbol{2}}\boldsymbol{w^T w}$$

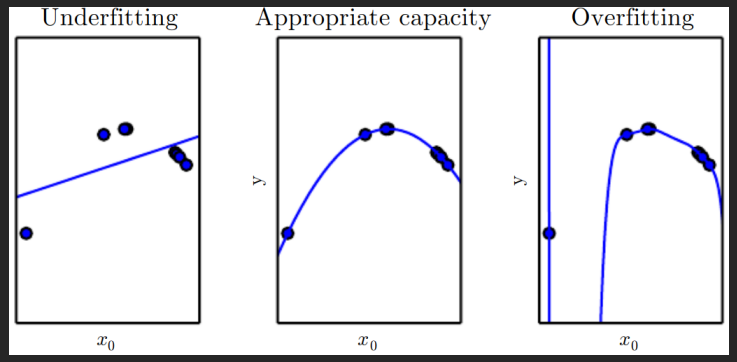
$$\vec{w}_{t+1} = \vec{w}_t - \eta[\alpha\vec{w}_t - \vec{\nabla} f(\vec{w}_t)]$$

$$\vec{w}_{t+1} = \vec{w}_t(1 - \alpha\eta) - \eta\vec{\nabla} f(\vec{w}_t)$$

i.e. weights decay during the training. Ridge regression prioritizes minimizing large weight parameters over small weight parameters.

L^2 and L^1 Regularization

Consider the data that is meant to represent a quadratic function



Can reduce overfitting in OLS by limiting the size of polynomial coefficients.

—

In ML, add a L regularization term to the loss function

L^1 regularization:

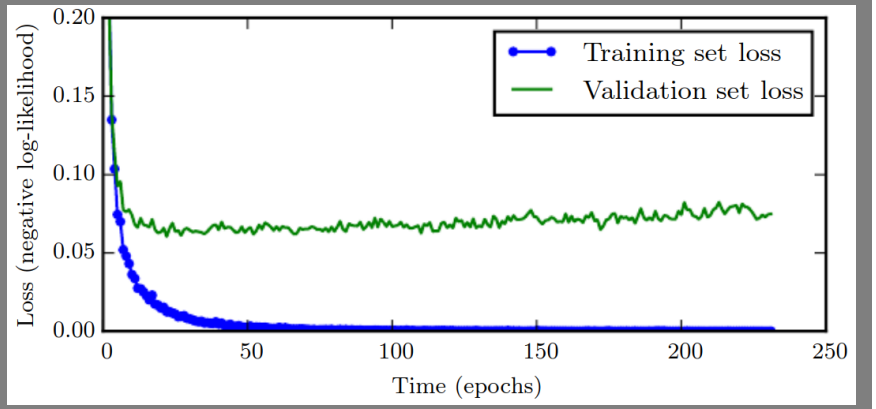
$$LSE = \frac{1}{2}\sum_{\mu}^{N}\sum_{j}^{J}\left[O_j^{\mu} - \hat{y}_j^{\mu}\right]^2 + \boldsymbol{\alpha}\|\boldsymbol{w}\|$$

Note that L^1 can set some weights to zero, while L^2 scaled the weights by a non-zero factor. But weight updates are more complicated (and will not be covered here)

L^1 regularization gives a more sparse solution and can be used as for feature selection (discard features for which the corresponding learned weights are zero)

Initialization: Generally draw elements of initial weight matrices from uniform or normal distribution with limits on variance given by $\sqrt{c/n}$ where c is a small scale constant and n is an integer (usually the number of input or output units to the layer or their sum/average). The idea here is to try and keep the variance of the inputs to each layer near to 1 (or at least not extremely big/small).

Early Stopping: Can monitor performance on a validation data set and stop training at appropriate time. Reduces model capacity by restricting model parameters to be reasonably near to their initialized values



Many problems have symmetries. For example, energies (forces) are invariant (equivariant) with respect to rotations of the coordinate system (chemical environment). Elastic deformations can be applied as well (deformed objects are often still recognizable).

Class imbalances: can make the model fine-tune too much on the most common class and can even guide the learning to trivial solutions (remember 2nd structure prediction). Can combat this by over-sampling the less-common classes during training

Other Regularization Techniques

Deep learning can handle lots of data so a good strategy is to add new examples based on transforming old ones. Can do this by adding noise to features from given examples

If inputs/outputs have inherent uncertainty, then injecting noise can bake the uncertainty into the model and prevent the model from learning the noise to give better generalization

A specific type of noise injection: Most common example is Dropout

- multiply hidden layers by binary noise during training phase and re-scale weights during test phase (set hidden units to 0 with probability p training and re-scale weights by p during test)

Benefits of Dropout

- Units in a given layer cannot depend on units in other layers so they must learn to be more self-reliant and generally useful.
- Applying noise to hidden units can destroy some information about the input without erasing all such information as would be done by adding noise directly to the input
- The model must then learn other hidden units that either redundantly encodes the same derived feature or some other that performs the task.

Dropout

In previous lectures we showed how to adjust our weight through back-propagation based on gradients of our loss function using optimization approaches that have been adapted for better test errors through modifications of standard 1st order methods, such as Stochastic Gradient Descent and Momentum SGD

- Nearly all of deep learning is powered by SGD and its better variants. However full gradients are prohibitively expensive for large and deep networks.

- The important recognition in ML is the fact that we only have an expectation that $\boldsymbol{f}'(\{\boldsymbol{x}\})$ is the mapping function, and hence the gradient itself $\boldsymbol{\nabla f}'(\{\boldsymbol{x}\})$ is only an expectation. Thus like any random variable, the expectation maybe approximately estimated using a small set of samples.

- For example, on each step of the learning, we can sample a mini-batch of examples drawn uniformly from the training set, typically chosen to be a relatively small number. We may fit a training set with billions of examples using updates computed on only hundreds of examples.

Mini-Batch and Batch Normalization

Mini-Batch and SGD

Mini-Batch learning examples makes sense for several reasons:

- Often data is redundant and gradient is not that different on one section of it vs another
- Weight update is much cheaper allowing for more time steps
- One can use efficient matrix routines which are not tractable over the whole data set but tractable in small chunks
- It is a form of regularization – we are not trying to minimize the loss function but instead want to generalize

Hence (Momentum) Stochastic GD (SGD) was developed on mini-batches and noise injection that creates noisy gradients.

SGD acts as implicit regularizer (short training time prevents over-fitting and stochastic batching helps escape saddle points and smooths surface)

Gradient can also be clipped or norm constrained (L2 regularization) as it should only provide local (directional) information anyway so large gradients are not necessarily reliable.

Mini-Batch and Batch Normalization

Recall from initialization that we would prefer variance ~1 for each hidden unit and that this motivated the choice of weights ~ $\sqrt{c/n}$

Can put in Batch Normalization layers to do something like this on a continuing basis during training

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

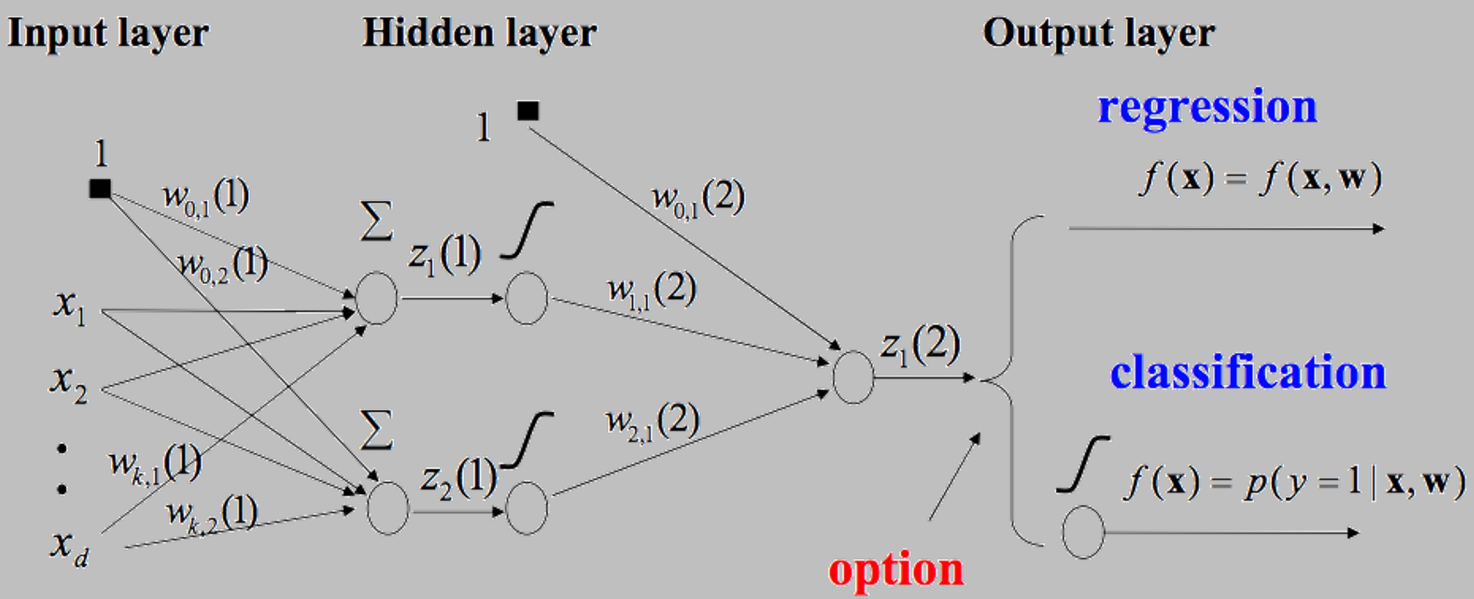
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Ioe and Szegedy, arXiv:1502.03167v3

A common principle throughout machine learning is that we can build complicated models from minimal components. We saw how we could build up non-linear models through combinations of activation functions that combine logistic regression (tanh, sigmoidal) and linear regression (linear).

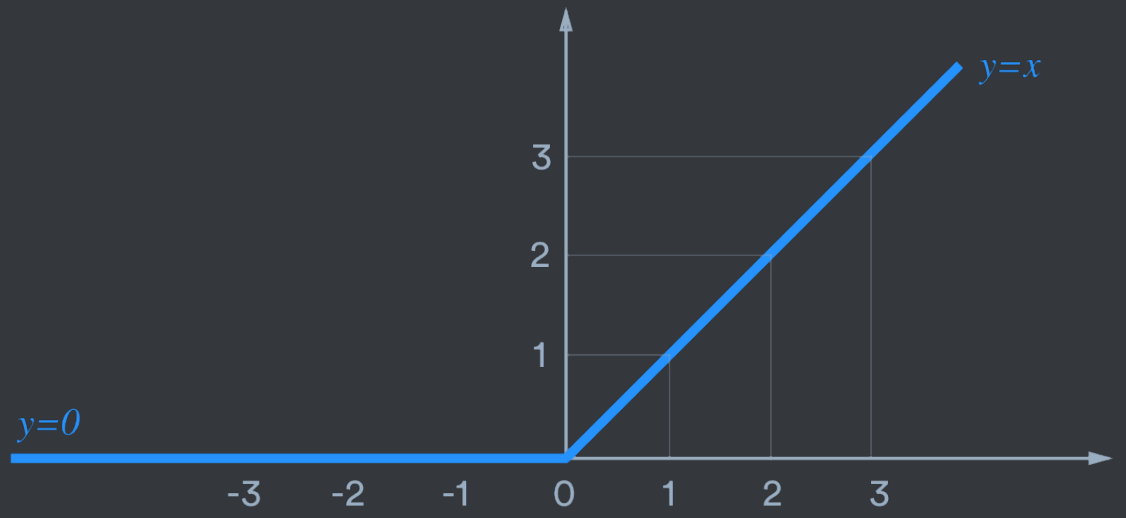


At present although artificial intelligence (mimicking brain function) is assumed to utilize sigmoidal activations, we can't utilize those effectively in learning representations.

Activation Functions

Activation Functions

The rectified linear activation function is recommended for use with most feedforward (and back-propagation) neural networks. Applying this function to the output of a linear transformation yields a nonlinear transformation. ReLU is linear (identity) for all positive values, and zero for all negative values.



- It's cheap to compute and the model takes less time to train or run.
- It converges faster. Linearity means that the slope doesn't plateau, or "saturate," when x gets large. It doesn't have vanishing gradient problem suffered by other activation functions like sig/tanh.
- It's sparsely activated. ReLU is zero for all negative inputs, so it's likely for any given unit to not activate at all. This is often desirable

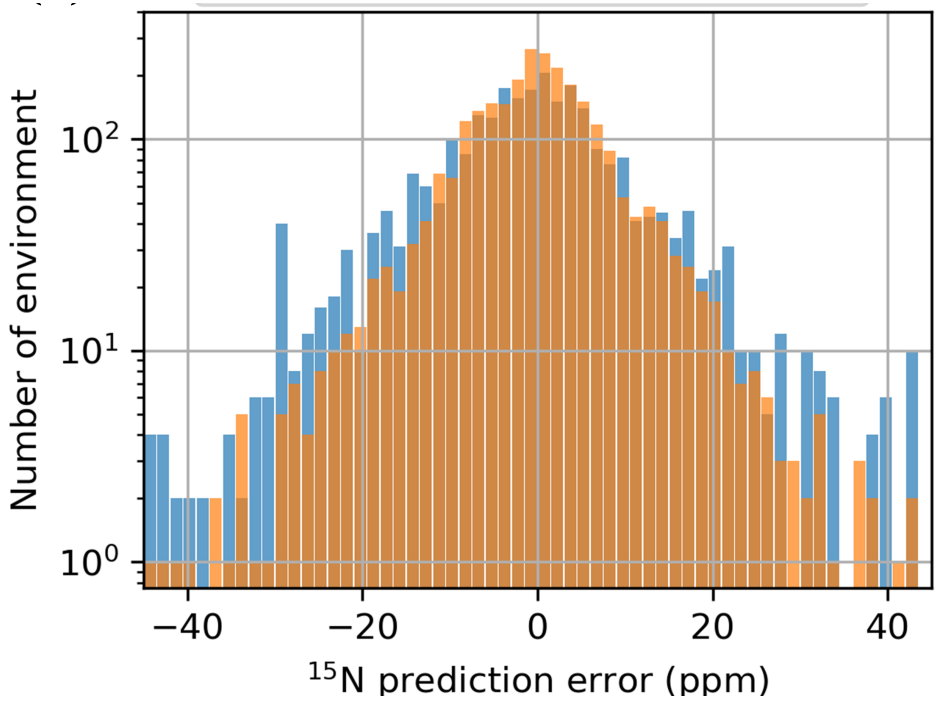
Although supervised learning is finding the conditional probability of y given x, the generalizability of the model depends on the distribution of x and y themselves.

Assumptions about training and test sets

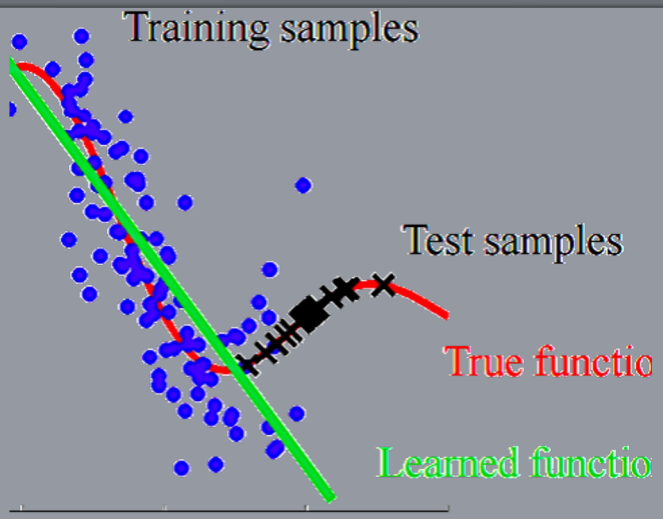
- Training/test data arise from same process
- Examples in each data set are independent
- Training set and testing set are identically distributed

We expect the model to work well for new examples, if the distributions of the new examples are not that different from what we used to train the model.

Distribution Shifts

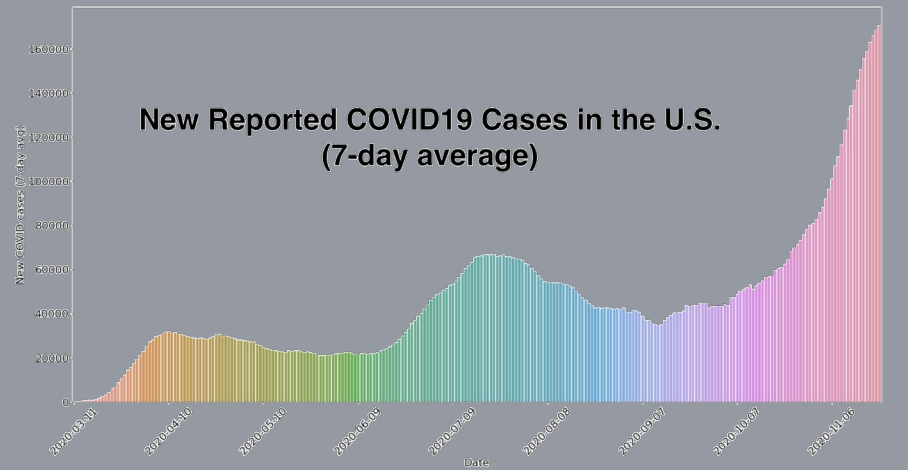


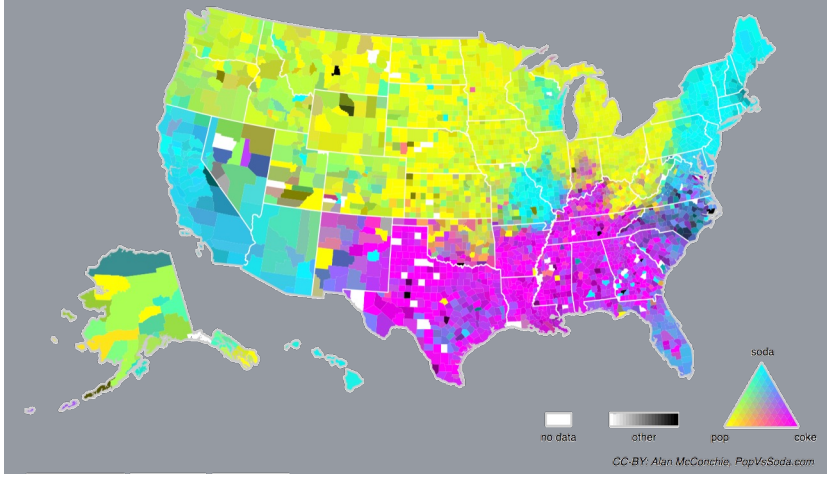
Distribution shifts and generalizability



Covariate shift: shift of input distributions. We want to learn this true function, but only have access to some of the training samples. If test data occurs in the same range, the model will work well. But if the test samples are outside of the range of the training samples, it will have very large generalization error.

Prior or Label shift: shift of output distributions between train and test. It is common in cases where we may have multiple different "targets" for the same input.





Concept shift: both input, output are same, but the concept they refer to might be different. Here is an example showing how the concept of "soft drink" refers to different drinks in different places in U.S.