Precision Medicine: Lecture 12
Deep Learning

Michael R. Kosorok,
Nikki L. B. Freeman and Owen E. Leete

Department of Biostatistics
Gillings School of Global Public Health
University of North Carolina at Chapel Hill

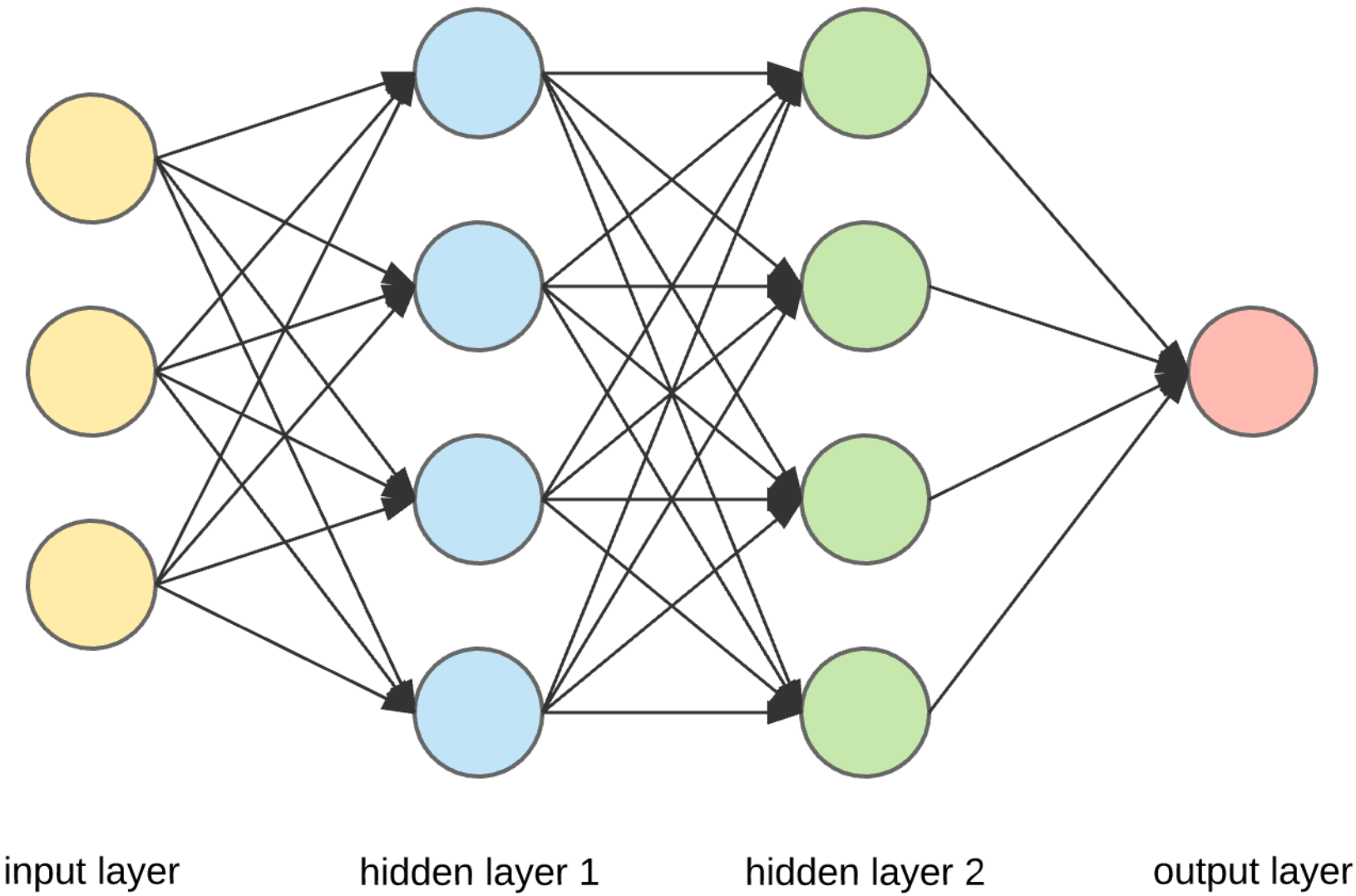
Fall, 2019

Outline

Artificial Neural Networks

- Artificial neural networks (ANN) are machine learning methods inspired by how neurons work in the brain
- ANNs are based on a collection of connected units or nodes called artificial neurons
- ANNs are mathematical functions of varying complexity that map a set of input values to output values
- ANNs are flexible models that can be used with many different types of input and output values
- By connecting the artificial neurons in different ways ANNs have been adapted to a wide variety of tasks

Artificial Neural Networks



Deep Learning

- Deep learning is a class of methods based on artificial neural networks
- The "deep" in deep learning refers to the number of hidden layers in an ANN
- A larger number of hidden layers allows deep neural networks to produce extremely intricate functions of its inputs
- Deep learning models can be simultaneously sensitive to minute details, but insensitive to large irrelevant changes

Feature Engineering

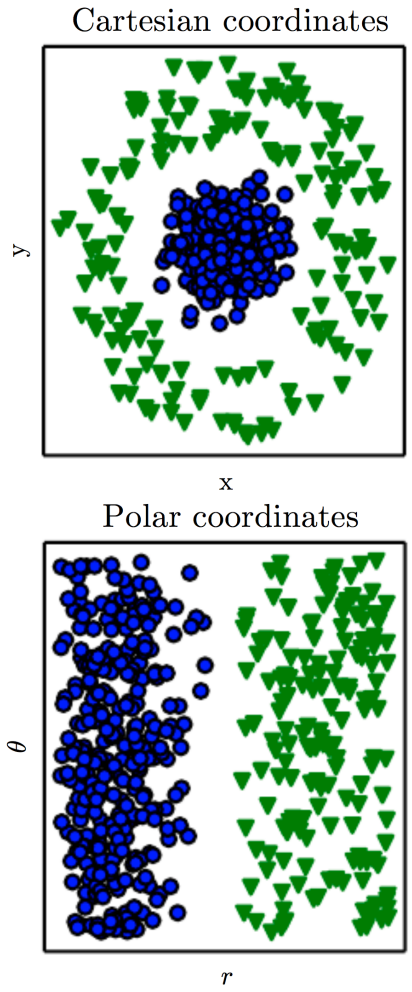
- Pattern-recognition and machine-learning systems have historically relied on carefully engineered features to extract useful representations from the raw data
- Engineered features are common in many applications
  - Example: BMI $=$ (weight in kg)/(height in m)^2
- In 2013, Andrew Ng said:

  Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

- Deep learning essentially automates the feature engineering process

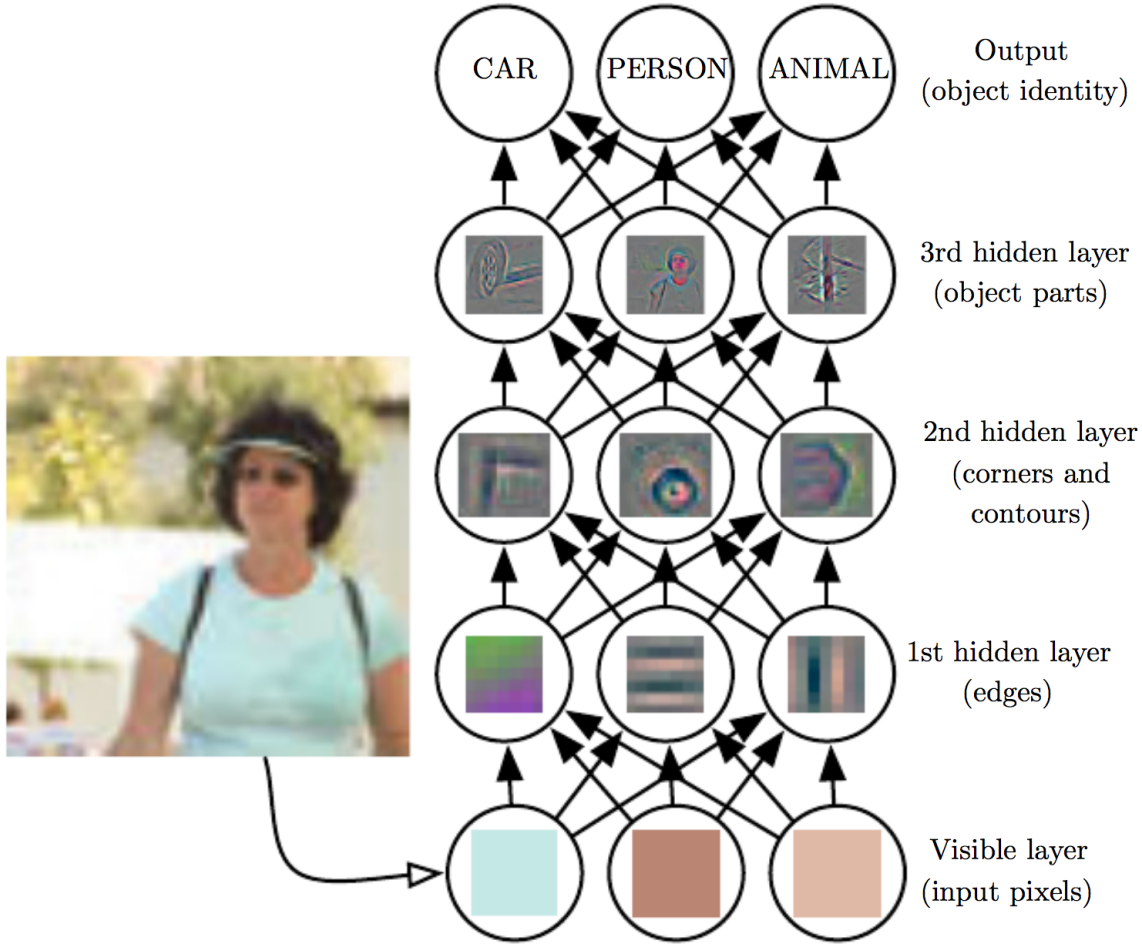
Representation learning

- Representation learning is a set of methods that allow ML algorithms to automatically discover representations of the data that make detection and classification easier
- Deep learning methods develop multiple levels of representation by compositing several simple non-linear transformations



Source: Goodfellow et al, 2016

Representation learning



ANN Origins — Perceptrons

- In 1958 Frank Rosenblatt described a binary classifier called the perceptron algorithm
- Given a d-dimensional vector of covariates $\mathbf{x}_i$, the class of the observation is predicted according to the function

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{d} w_i x_i + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

  where $\mathbf{w}$ is a vector of real-valued weights
- Perceptrons are an early form of linear classification
- ANNs are sometimes referred to as multi-layer perceptrons

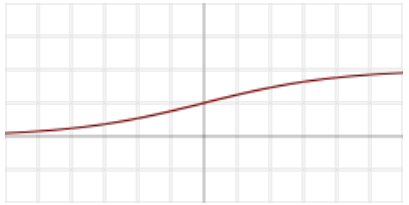
Activation Functions

- Each layer in an ANN is composed of a linear combination of the node values from the previous layer
- Applying a non-linear activation function to the linear combinations allows successive layers to learn increasingly complex features
- While selecting a model, it is common to test many different activation functions and find that several perform comparably
- There are some situations where the choice of activation functions can greatly impact the performance of ANNs

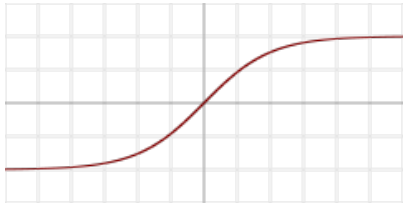
Activation Functions

- Several activation functions have been published, but it is likely that most remain unpublished
- Some of the most common activation functions are:

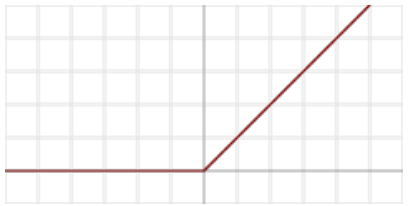
Logistic $$g(x) = \frac{1}{1 + e^{-x}}$$



TanH $$g(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



ReLU $$g(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$



Architecture Design

- A key design consideration for neural networks is determining the architecture
- Architecture refers to the overall structure of the network
  - How many layers
  - How many units in each layer
  - How should these units be connected to each other
  - Which activation functions to use
- Many ANNs use a chain based architecture
  - The first layer is given by

$$\mathbf{h}^{(1)} = g^{(1)}\left(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

  - Subsequent layers are given by

$$\mathbf{h}^{(j)} = g^{(j)}\left(\mathbf{W}^{(j)T}\mathbf{h}^{(j-1)} + \mathbf{b}^{(1)}\right)$$

Output Units

- ANNs can be used for a variety of different learning tasks by changing the output units
- Let $\mathbf{h}$ be the features from the final hidden layer
- Linear Units for Continuous Output Distributions
  - The output units produces a vector $\hat{\mathbf{y}} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$
  - Linear output layers are often used to produce the mean of a conditional Gaussian distribution:

$$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y}; \hat{\mathbf{y}}, \mathrm{I})$$

- Sigmoid Units for Bernoulli Output Distributions

$$\hat{y} = \frac{\exp\{\mathbf{w}^T\mathbf{h} + b\}}{1 + \exp\{\mathbf{w}^T\mathbf{h} + b\}}$$

Output Units, cont.

- Softmax Units for Multinoulli Output Distributions
  - A linear layer predicts unnormalized log (relative) probabilities

$$\mathbf{z} = \mathbf{W}^T \mathbf{h} + \mathbf{b}$$

    where $z_i = \log P(y = i \mid \mathbf{x})$
  - The softmax function can normalize z to obtain the desired $\hat{y}$

$$\text{softmax}(\mathbf{z}) = \frac{\exp\{z_i\}}{\sum_j \exp\{z_j\}}$$

- There are many other output units that can return images, sound, video, etc.

Training via Backpropagation

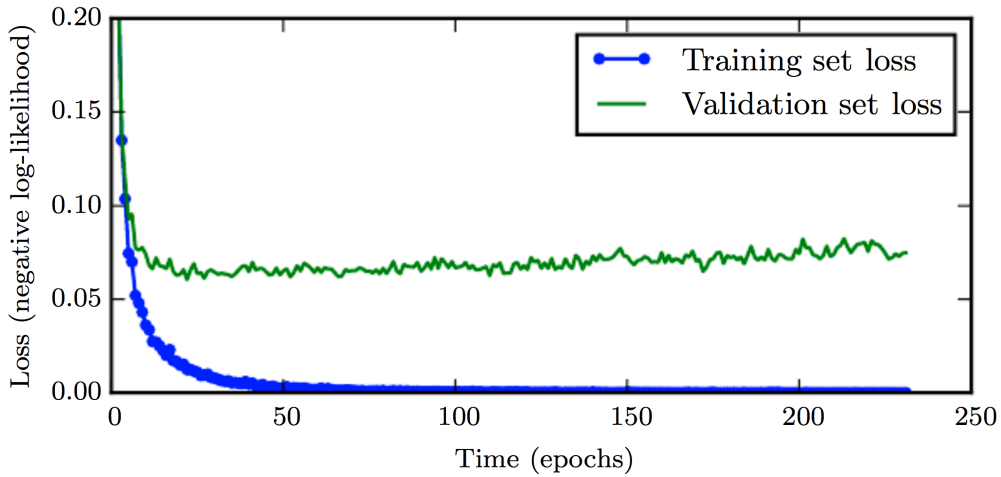
- Multi-layer architectures can be trained by gradient descent
- If the nodes are relatively smooth functions of the inputs, the gradients can be calculated using the backpropagation procedure
- For a given loss function we can determine how the weights in the final layer need to change to lower the loss
- Repeated application of the chain rule allows us to determine how weights in previous layers need to change
- Some activation functions are not differentiable at all points (e.g. ReLU), but they can still be used with gradient-based learning algorithms at all input points.

Regularization

- DL models typically have a large number of parameters, sometimes more parameters than training examples
- Regularization methods are required to prevent overfitting
- L^1 and L^2 norms can be applied to the weights for each node, but this is uncommon in DL
- Ensembles of neural networks with different model configurations are known to reduce overfitting
  - It is impractical to have an ensemble of multiple large neural networks
  - A single model can be used to simulate having a large number of different network architectures by randomly dropping out nodes during training
  - Dropout is a computationally efficient and remarkably effective method to approximate an ensemble approach

Regularization

- One of the most common regularization methods used for ANNs is early stopping
- The training error almost always decreases, but validation error tends to increases with excessive training
- A model with small validation error can be found buy stopping the training process early



Adversarial Examples

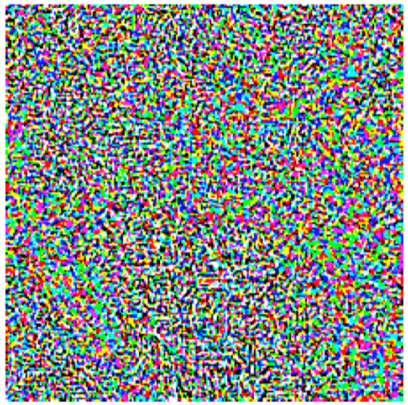
- Adversarial examples are samples of input data which are designed/selected to cause a machine learning classifier to misclassify it
- Adversarial examples can be used while training to make a DL model more robust
  - Samples with noise added can make the predictions less sensitive to small differences
  - Exposing a model to samples known to lie close to the decision boundary can improve performance
- Adversarial examples have important implications for the safety of certain applications (e.g. self driving cars)

Adversarial examples

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

- By adding a imperceptible amount of noise, the classification of the image can be changed

Adversarial examples

These examples are likely close to the decision boundary

Mop or Puli

Muffin or Chihuahua

Outline

Convolutional Neural Networks

- Convolutional Neural Networks (CNNs) are designed to process data that come in the form of multiple arrays
- CNNs are used in many applications such as: image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing
- The few layers of a typical CNN is composed of two types of layers
  - Convolutional layers
  - Pooling layers

Convolution

- A convolution is an operation on two functions of a real-valued argument
- Convolutions are used to look at localized areas of an array

$$s(t) = \int x(a)w(t-a)\,da$$

- The convolution operation is typically denoted with an asterisk

$$s(t) = (x * w)(t)$$

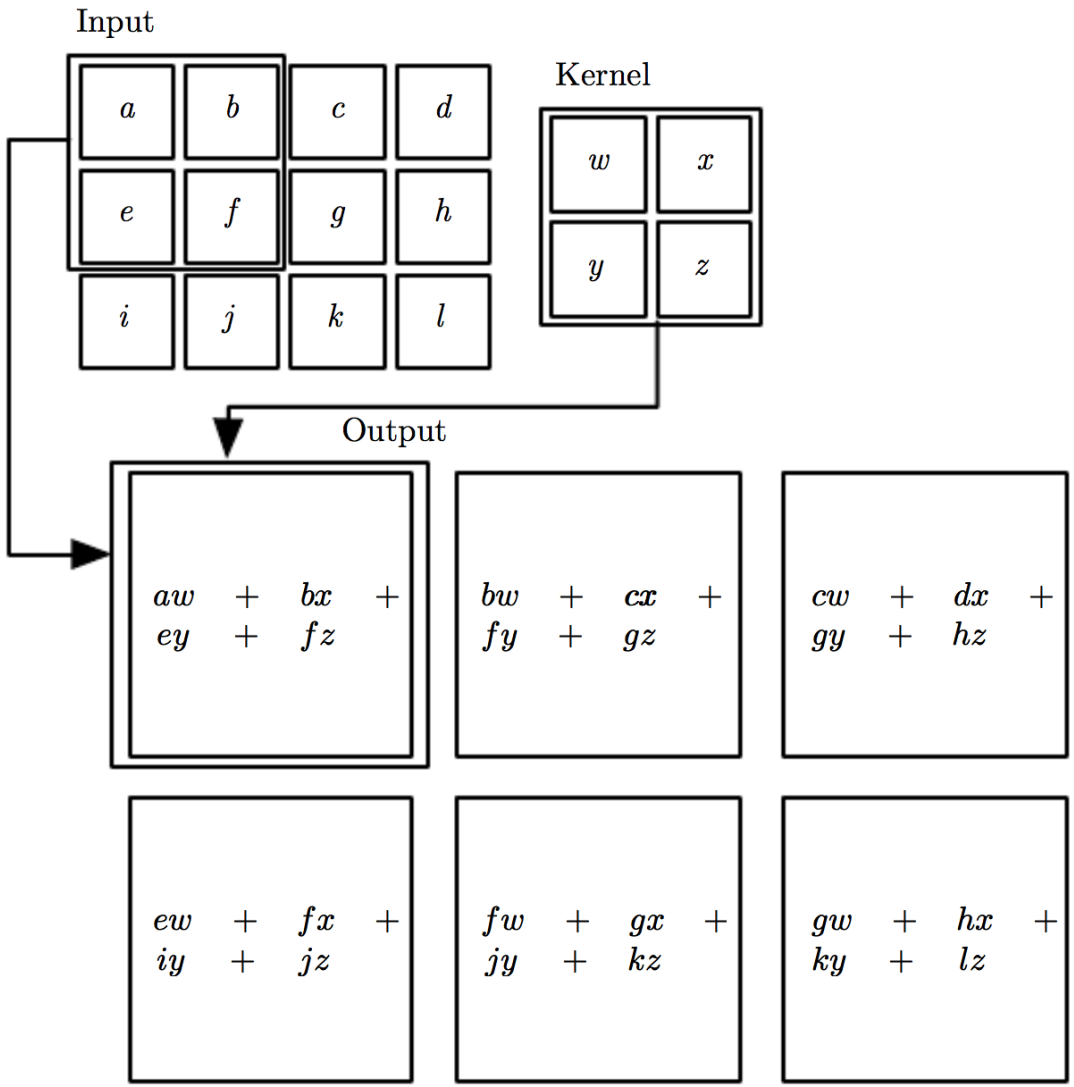
Convolution

- Convolutions are often used over more than one axis at a time
- For a d-dimensional input, convolutions can be calculated with a d-dimensional kernel K
- For an $m \times n$ image as input, we can write the convolution as

$$S(i,j) = (X * K)(i,j) = \sum_m \sum_n X(m,n)K(i-m, j-n)$$

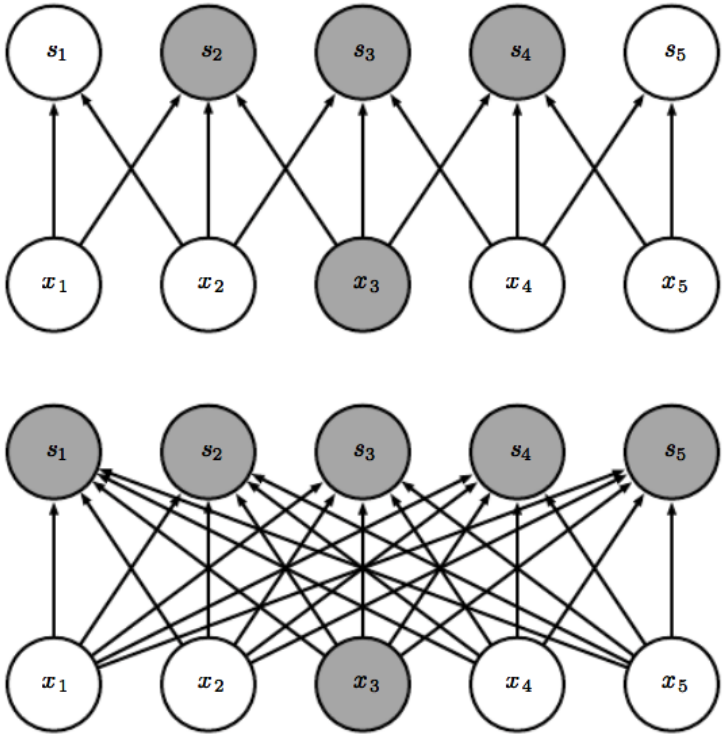
- Discrete convolution can be viewed as multiplication by a matrix, where the matrix has several entries constrained to be equal

Convolution Layer



Source: Goodfellow et al, 2016

Local Connectivity



Source: Goodfellow et al, 2016

- Unlike other ANNs, CNNs have layers that are not fully connected
- Convolutional layers have local connections
- For example, an input image might have thousands or millions of pixels, but meaningful features usually occupy only tens or hundreds of pixels

Parameter Sharing

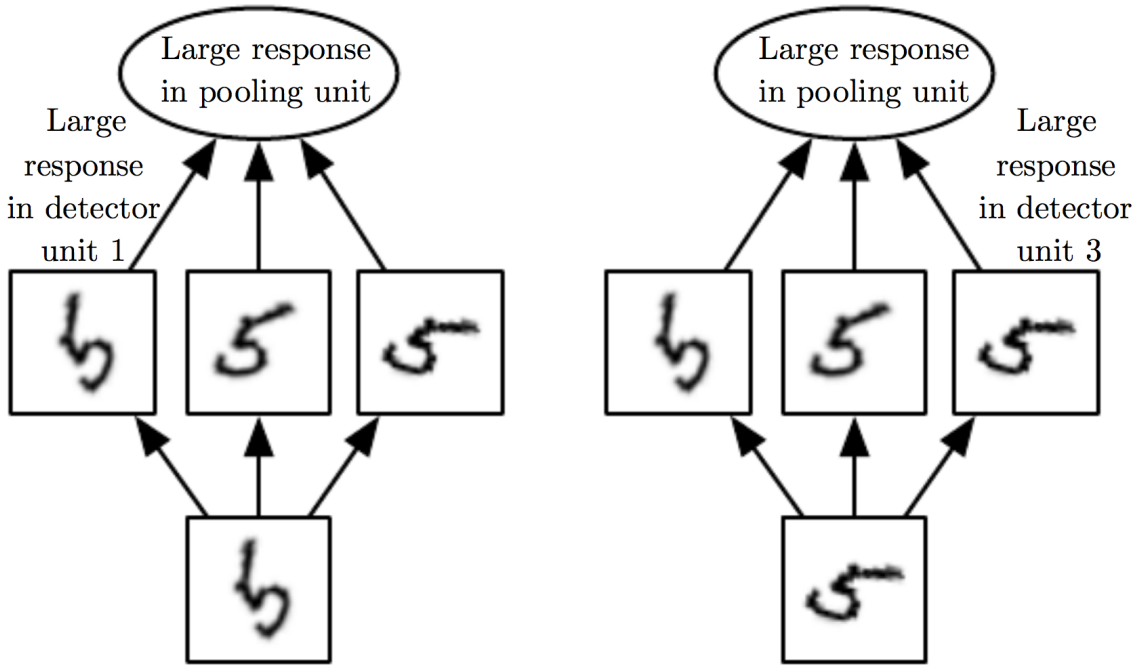
- In a convolutional neural net, each member of the kernel is used at every position of the input
- The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set
- Parameter sharing causes a layer to have a property called equivariance to translation
  - Features can be identified regardless of where they occur in an image
- Both local connectivity and parameter sharing can greatly reduce the number of parameters needed compared to a similarly sized traditional neural network

Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs
  - Example: Max pooling operation reports the maximum output within a rectangular neighborhood
- Pooling over spatial regions can help to make the representation approximately invariant to small translations of the input
- The feature generation process can learn which transformations to become invariant to by pooling over the outputs of a range of parameterized convolutions

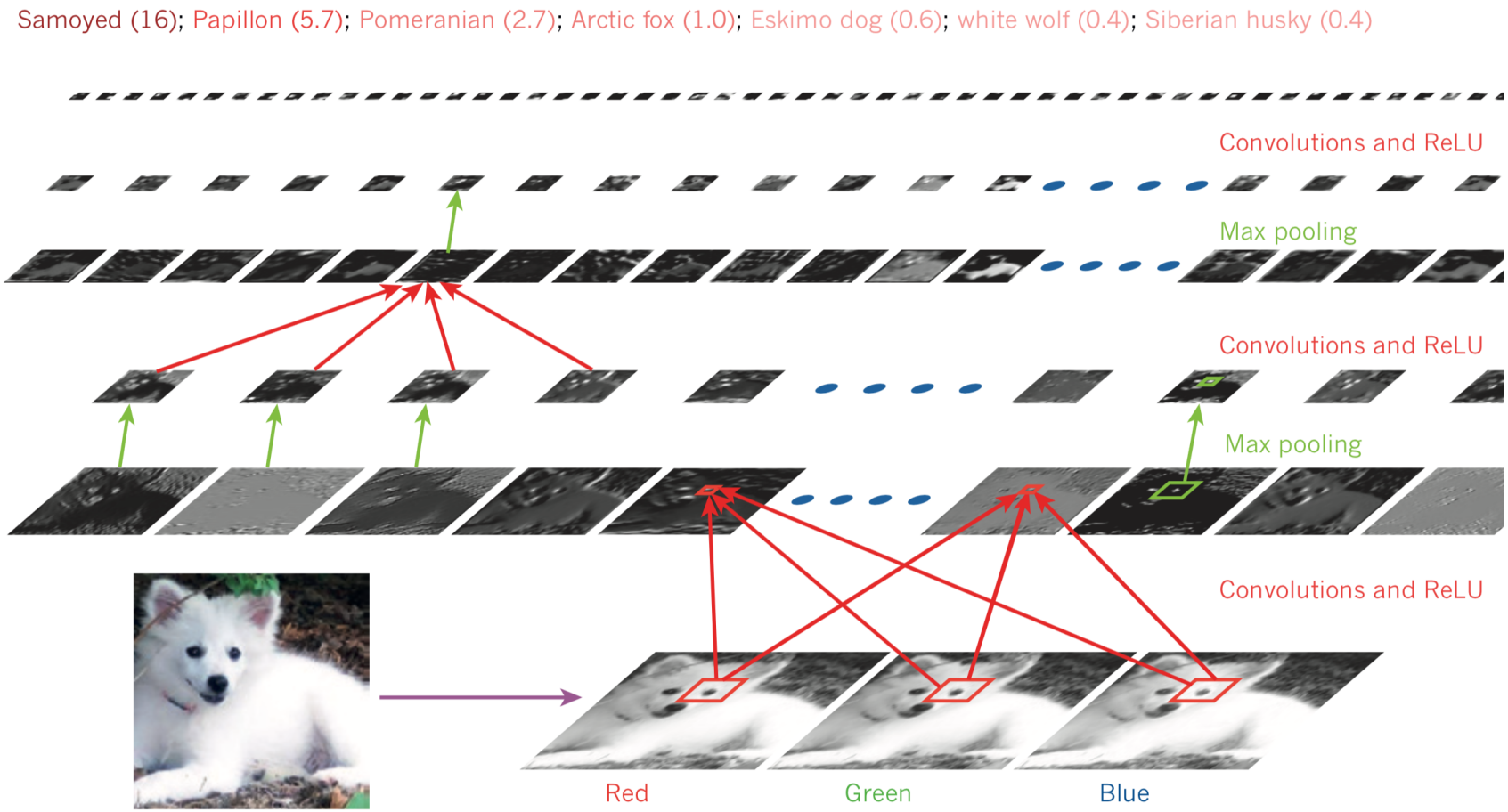
Pooling

- Example: All three filters are intended to detect a hand written 5
- Each filter attempts to match a slightly different orientation of the 5



Source: Goodfellow et al, 2016

Example of CNN Architecture



Outline

Recurrent Neural Networks

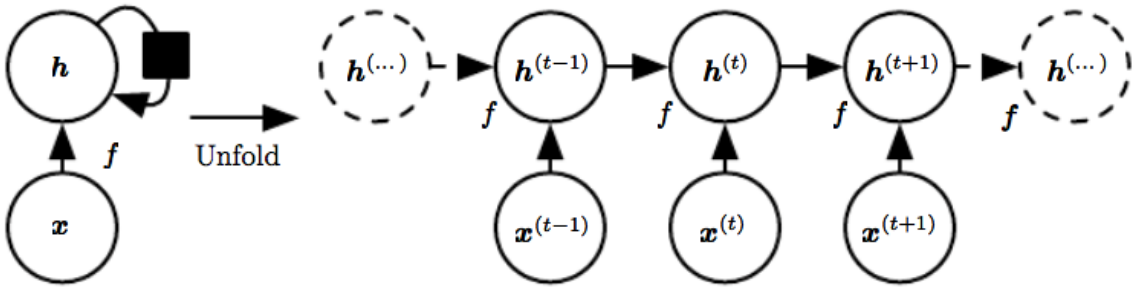
- Recurrent neural networks (RNNs) are a family of neural networks for processing sequential data
- RNNs process an input sequence one element at a time, maintaining in their hidden units a 'state vector' that contains information about the history of the sequence
- Most RNNs can process sequences of variable length, and can scale to much longer sequences than would be practical for networks without sequence-based specialization
  - Both of these qualities are largely due to parameter sharing

Unfolding Computational Graphs

- A computational graph is a way to formalize the structure of a set of computations
- Consider a dynamical system where the state at time t is $\mathbf{h}^{(t)}$. The system depends on a function f, parameters θ, and is driven by an external signal $\mathbf{x}^{(t)}$

$$\begin{aligned}\mathbf{h}^{(t+1)} &= f(\mathbf{h}^{(t)}, x^{(t)}; \theta) \\ &= f(f(\ldots f(\mathbf{h}^{(1)}, x^{(1)}; \theta), \ldots, x^{(t-1)}; \theta), x^{(t)}; \theta)\end{aligned}$$

- This system can be represented using the graphical model



Source: Goodfellow et al, 2016

Unfolding Computational Graphs

- RNNs can be described as a computational graph that has a recurrent structure
- A recurrent computational graph can be unfolded to a computational graph with a repetitive structure
- Complex models can be succinctly represented with a recurrent graph
- The unfolded graph provides an explicit description of which computations to perform

Recurrent Neural Networks

- RNNs learn a single shared model and apply the same set of computations at each time step
- A shared model allows generalization to sequence lengths that did not appear in the training set, and needs far fewer training examples than would be required without parameter sharing
- RNNs can output a result at each time step (stock market predictions) or read an entire sequence before outputting a result (meaning of a sentence)

Bidirectional RNNs

- RNNs need not have a causal structure. In many applications we want to output a prediction that may depend on the whole input sequence
- For example, in natural language processing, the meaning of a word might require the context of nearby words in both directions
- Bidirectional RNNs are composed of two RNNs: one that moves forward through time from the start of the sequence, and another that moves backward through time from the end of the sequence

The Challenge of Long-Term Dependencies

- Long-Term dependencies are difficult to model because gradients propagated over many stages tend to either vanish or explode
- There have been attempts to avoid the problem by staying in a region of the parameter space where the gradients do not vanish or explode
- Unfortunately, in order to store memories in a way that is robust to small perturbations, the RNN must enter a region of parameter space where gradients vanish
- Even if the parameters are such that the recurrent network is stable, long-term interactions have exponentially smaller weights compared to short-term interactions

Skip Connections and Leaky Units

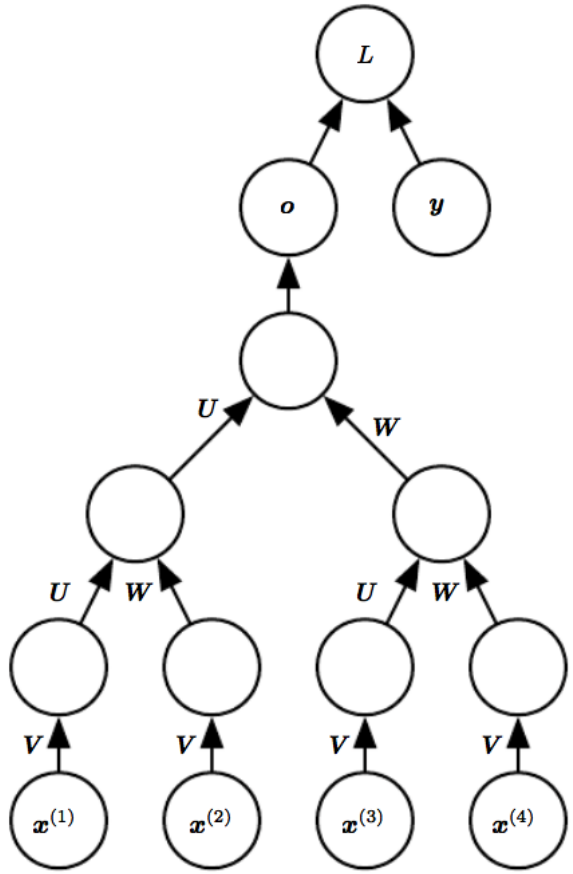
- Skip connections obtain coarse time scales by adding direct connections from variables in the distant past to variables in the present
  - In ordinary recurrent networks, a recurrent connection goes from a unit at time t to a unit at time $t+1$, but longer connections are possible $(t+d)$
  - For τ time steps, gradients now diminish exponentially as a function of τ/d rather than τ
- Leaky Units have linear self-connections that "remember" past values
  - Leaky units accumulate a running average $\mu^{(t)}$ of some value $v^{(t)}$ by applying the update $\mu^{(t)} = \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)}$
  - When α is near one, the leaky unit remembers information about the past for a long time, and when α is near zero, information about the past is rapidly discarded

Long Short-Term Memory Nodes

- Leaky units use self-connections to accumulate information, but there is no mechanism to "forget" old information even when it would be beneficial to do so
- Long Short-Term Memory units have several "gates" to control how the unit behaves at each time step
  - Input gate: Controls when the node gets updated
  - Forget gate: Controls how long information is retained
  - output gate: Controls when the node has an output value
- Each gate has parameters controlling its behavior allowing the model to learn when each behavior is beneficial

Recursive Neural Networks

- Recursive neural networks are a generalization of recurrent networks, with a computational graph which is structured as a tree
- For a sequence of the same length, the number of compositions of nonlinear operations is smaller for recursive neural networks than RNNs which might help deal with long-term dependencies



Source: Goodfellow et al, 2016

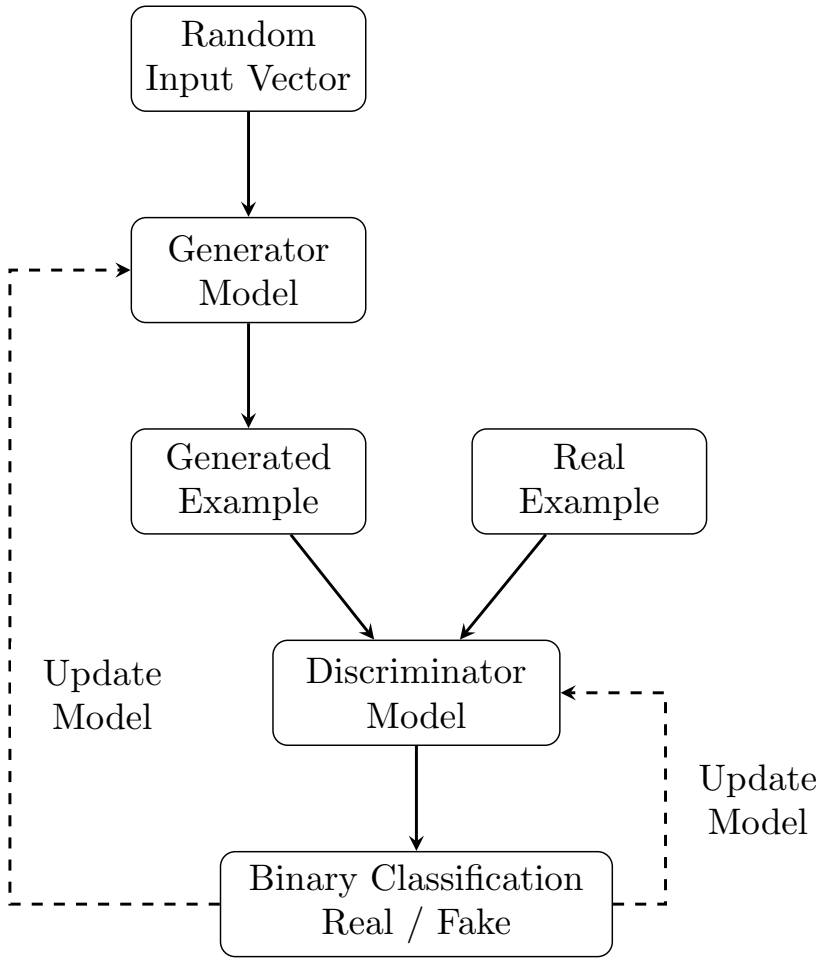
Outline

Generative Modeling

- Generative modeling is an unsupervised learning task
- A generative model is used to generate new examples that could have been drawn from the original data distribution
- Generative adversarial networks (GANs) are a way of training a generative model by framing it as a supervised learning problem with two sub-models
  - A generative network which learns to map from a latent space to a data distribution of interest
  - A discriminative network which distinguishes candidates produced by the generator from the true data distribution

Generative Adversarial Networks

- The generator model "learns" the data distribution by competing with the discriminator model
- Both the generator and discriminator models are updated to improve their performance
- Training continues until the discriminator is consistently "fooled" 50% of the time



GAN Progress

- GANs have made considerable progress in recent years
- Image generators can fool both discriminator networks and human observers, which misclassify up to 40 percent of generated images

GAN Applications

- GANs are useful for their ability to represent high-dimensional probability distributions
- There are many potential applications of GANs
  - Generation of images, video, etc.
  - Data augmentation
  - Missing Data imputation
  - Semi-supervised learning
  - Reinforcement learning
- If carefully constructed, GANs can be used to learn more about the underlying data distributions

Outline

Motivation

- The gold standard for discovering causal relationships is experiments
- Experiments can be prohibitively expensive, unethical, or impossible, so there is a need for observational causal discovery
- Causal generative neural networks (CGNNs) learn functional causal models by fitting a generative neural networks that minimizes the maximum mean discrepancy
- Using deep neural networks allows CGNNs to learn more complex causal relationships than other approaches

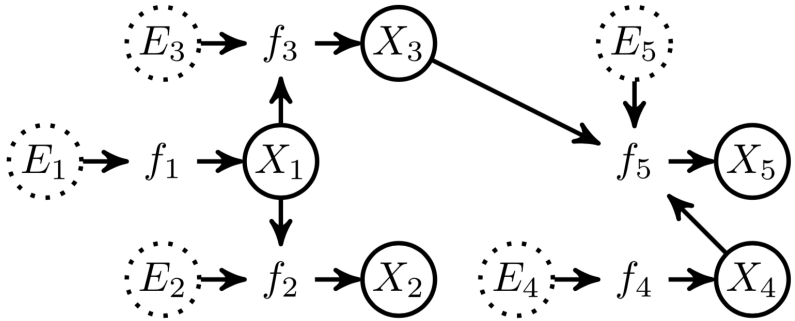
Functional Causal Models

- A functional causal model (FCM) on a vector of random variables $\mathbf{X} = (X_1, X_2, \ldots, X_d)$ is a triplet $C = (\mathcal{G}, f, \mathcal{E})$, where:
  - $\mathcal{G}$ is a graph
  - f characterizes the relationships between X's
  - $\mathcal{E}$ is an error distribution
- FCMs can be represented by a set of equations

$$X_i \leftarrow f_i(X_{Pa(i,\mathcal{G})}, E_i),\ E_i \sim \mathcal{E},\ \text{for } i = 1, \ldots, d$$

  where $X_{Pa(i;\mathcal{G})}$ are the "parents" of X_i in graph $\mathcal{G}$
- For notational simplicity X_i interchangeably denotes an observed variable and a node in the graph $\mathcal{G}$

Functional Causal Models



$$\begin{cases} E_i & \sim \mathcal{E} \\ X_1 & = f_1(E_1) \\ X_2 & = f_2(X_1, E_2) \\ X_3 & = f_3(X_1, E_3) \\ X_4 & = f_4(E_4) \\ X_5 & = f_5(X_3, X_4, E_5) \end{cases}$$

Source: Goudet et al., 2018

- FCMs can be represented as a directed acyclic graph (DAG) as in the example above
- There exists a direct causal relation from X_j to X_i iff there exists a directed edge X_j to X_i in $\mathcal{G}$

Causal Generative Neural Networks

- Let $\mathbf{X} = (X_1, \ldots, X_d)$ denote a set of continuous random variables with joint distribution P
- If the joint density function associated with P is continuous and strictly positive on a compact subset of $\mathbb{R}^d$ and zero elsewhere, it can be shown that there is a CGNN that approximates P with arbitrary accuracy
- Rather than use a discriminator model to evaluate the generator, CGNNs train the generator to minimize the maximum mean discrepancy (MMD) between the real and generated data

Maximum Mean Discrepancy

- MMD measures whether two distributions are the same
- Let $\mathcal{F}$ be a class of functions $f : X \to \mathbb{R}$ and let p, q be distributions

$$\mathrm{MMD}(\mathcal{F}, p, q) = \sup_{f \in \mathcal{F}} \left(\mathbf{E}_{x \sim p}[f(x)] - \mathbf{E}_{y \sim q}[f(y)] \right)$$

- For samples $X \sim p$ of size m and $Y \sim q$ of size n then the estimate of the MMD is

$$\widehat{\mathrm{MMD}}(\mathcal{F}, X, Y) = \sup_{f \in \mathcal{F}} \left(\frac{1}{m} \sum_{i=1}^{m} f(X_i) - \frac{1}{n} \sum_{i=1}^{n} f(Y_i) \right)$$

- Under certain conditions $\mathrm{MMD}(\mathcal{F}, p, q) = 0$ iff $p = q$

Scoring Metric

- The maximum over $\mathcal{F}$ is made tractable by assuming that $\mathcal{F}$ is the unit ball of a RKHS with kernel k
- For an estimated distribution $\widehat{P}$ we want to know if it is close to the true distribution P
- The estimated MMD between the n-sample observational data D, and an n-sample $\widehat{D}$ from $\widehat{P}$ is

$$\widehat{\mathrm{MMD}}_k(\mathcal{D}, \widehat{\mathcal{D}}) = \frac{1}{n^2}\sum_{i,j=1}^{n} k(x_i, x_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(\hat{x}_i, \hat{x}_j) - \frac{2}{n^2}\sum_{i,j=1}^{n} k(x_i, \hat{x}_j)$$

- The estimated FCM $\widehat{C}$ is trained by maximizing

$$S(\widehat{\mathcal{G}}, \mathcal{D}) = -\widehat{\mathrm{MMD}}_k(\mathcal{D}, \widehat{\mathcal{D}}) - \lambda|\widehat{\mathcal{G}}|$$

Searching Causal Graphs

- An exhaustive explorations of all DAGs with d variables using brute force search is infeasible for moderate d
- To solve this issue the authors assume that the skeleton of the graph $\mathcal{G}$ is obtainable from domain knowledge
- The CGNN follows a greedy procedure to find $\mathcal{G}$ and f_i:
  - Orient each $X_i - X_j$ as $X_i \to X_j$ or $X_j \to X_i$ by selecting the 2-variable CGNN with the best score
  - Follow paths from a random set of nodes until all nodes are reached and no cycles are present
  - For a number of iterations, reverse the edge that leads to the maximum improvement of the score $S(\mathcal{G}, \mathcal{D})$ over a d-variable CGNN, without creating a cycle
  - At the end of this process, we evaluate a confidence score for any edge $X_i \to X_j$ as

$$V_{X_i \to X_j} = S(\mathcal{G}, \mathcal{D}) - S(\mathcal{G} - \{X_i \to X_j\}, \mathcal{D})$$

Dealing with Hidden Confounders

- The search method relies on the no unmeasured confounders assumption
- If this assumption is violated, we know that each edge $X_i - X_j$ in the skeleton is due to one out of three possibilities
  - $X_i \rightarrow X_j$
  - $X_i \leftarrow X_j$
  - $X_i \leftarrow E_{i,j} \rightarrow X_j$ for some unobserved variable $E_{i,j}$
- The search method can be modified to allow for confounders as follows:
  - Each equation in the FCM is extended to:

$$X_i \leftarrow f_i(X_{Pa(i,\mathcal{G})}, E_{i,Ne(i,\mathcal{S})}, E_i)$$

    where $Ne(i, \mathcal{S})$ is the set of indicies of variables adjacent to X_i in the skeleton
  - In this case, regularization by $\lambda|\widehat{\mathcal{G}}|$ promotes simple graphs

Discovering v-structures

- Consider the random variables (A, B, C) with skeleton $A - B - C$, four causal structures are possible
  - $A \rightarrow B \rightarrow C$
  - $A \leftarrow B \leftarrow C$
  - $A \leftarrow B \rightarrow C$
  - $A \rightarrow B \leftarrow C$
- All four structures are Markov equivalent, and therefore indistinguishable from each other using statistics alone
- Previous methods have had difficulty identifying the correct structure
- CGNNs can accurately discriminate between the v-structures using the MMD criteria

Conclusion

- CGNNs are a new framework to learn functional causal models from observational data
- CGNNs combine the power of deep learning and the interpretability of causal models
- CGNNs are better able to identify the causal structure of relationships compared to other methods
- There is still a need to characterize the sufficient identifiability conditions for this approach