The importance of representations

- The choice of representation is key to solving a given problem correctly and efficiently

- A few examples to motivate this (from the Deep Learning book):

    1. Long division:

        "Divide 210 by 6"

The importance of representations

- The choice of representation is key to solving a given problem correctly and efficiently

- A few examples to motivate this (from the Deep Learning book):

    1. Long division:

        "Divide 210 by 6" vs. "Divide CCX by VI"

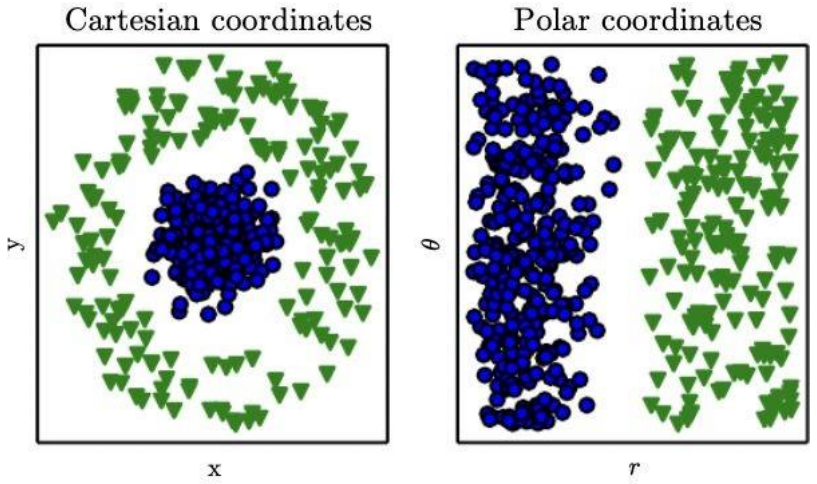
The importance of representations

- A few examples to motivate this (from the Deep Learning book):

1. Long division:

   "Divide 210 by 6" vs. "Divide CCX by VI"

2. Binary Classification using a Linear Classifier



The importance of representations

- A few examples to motivate this (from the Deep Learning book):

3. Insert a number into a sorted list
   a. Linked List -> $O(n)$
   b. Balanced Tree -> $O(\log n)$

Representation Learning

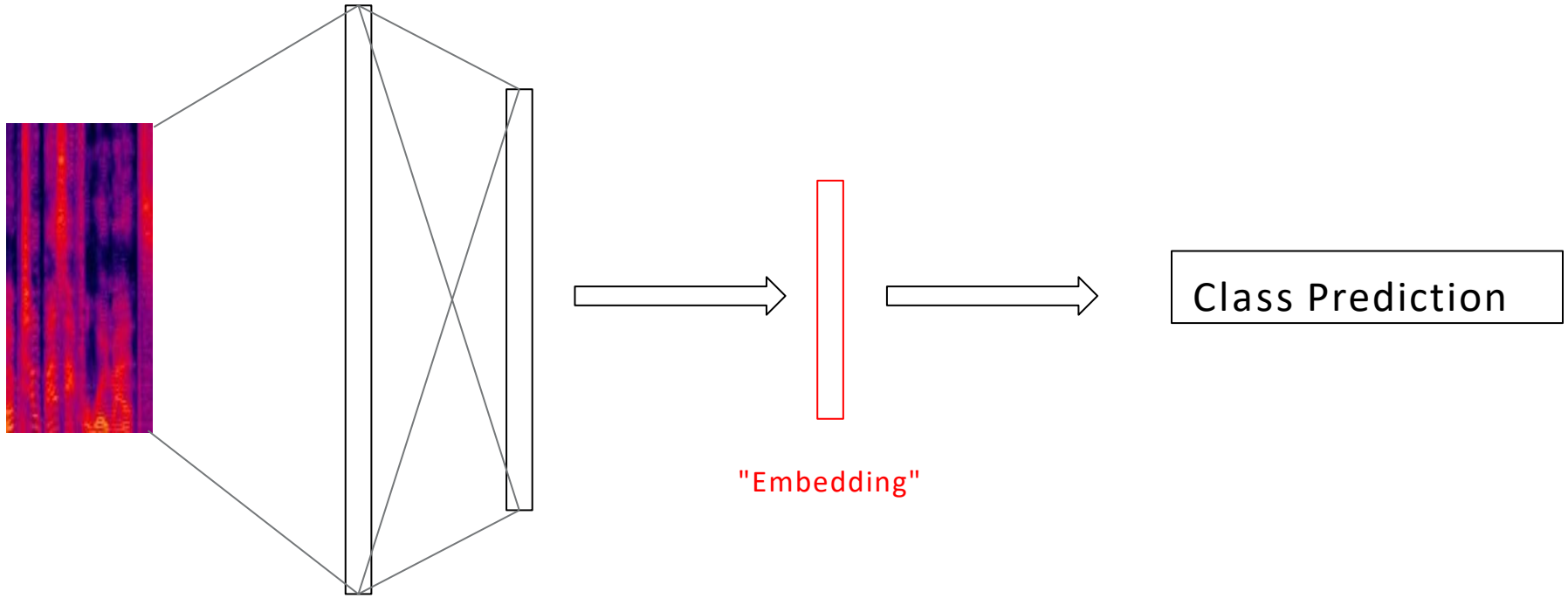
- The task of "*learning* representations of the data that make it easier to extract useful information when building classifiers or other predictors" and has become "a field in itself"*
- A shift from hand-crafting complex features from data to being able to learn these features (with a neural network)
- Learned representations often desired to satisfy certain properties:
  - Useful in downstream tasks through **transfer learning**,
  - Interpretability**

* Bengio, Yoshua et al. “Representation Learning: A Review and New Perspectives.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (2013): 1798-1828.

What is a representation?

We have seen several instances of "representations" in the course already:

- HW1P2: Phoneme classification



What is a representation?

- HW2P2: Face classification

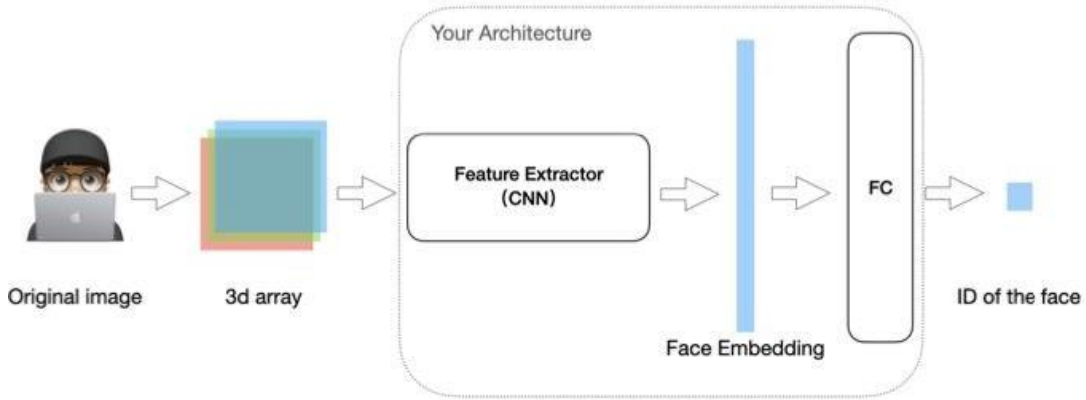
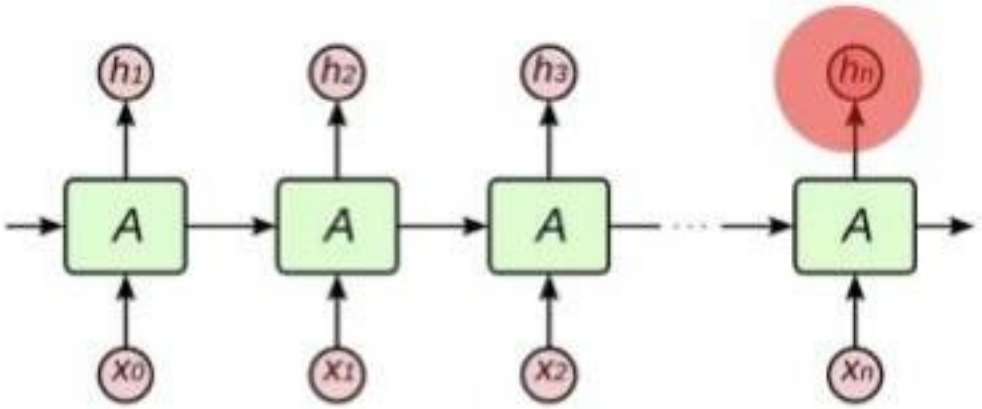


Figure 1: A typical face classification architecture

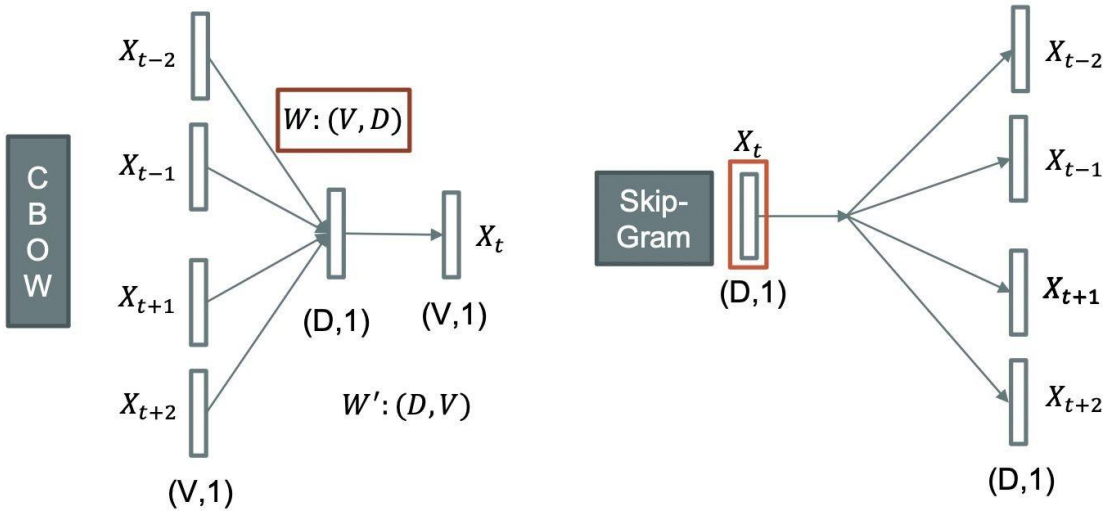
What is a representation?

- Recurrent Neural Networks (from Attention Recitation):



Applications

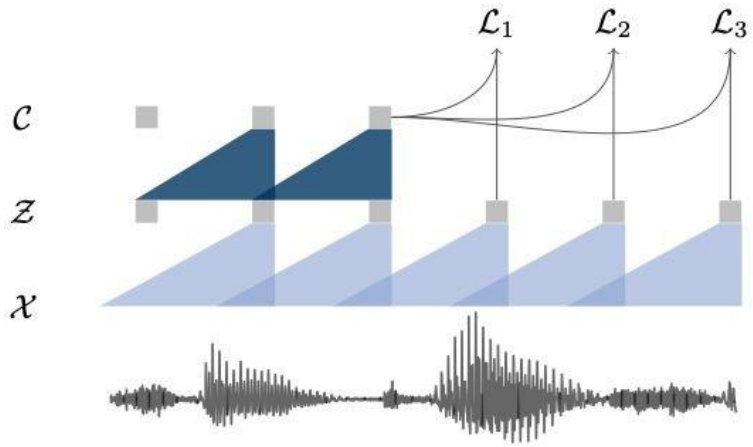
- Word Embeddings (word2vec*):
  - Make several downstream tasks easier - sentiment classification, question answering, etc.



*Mikolov, Tomas et al. “Distributed Representations of Words and Phrases and their Compositionality.” *NIPS* (2013).

Applications

- Speech Utterance Embeddings (wav2vec*):
  - Useful for word/letter classification



*Schneider, Steffen et al. “wav2vec: Unsupervised Pre-training for Speech Recognition.” *INTERSPEECH* (2019).

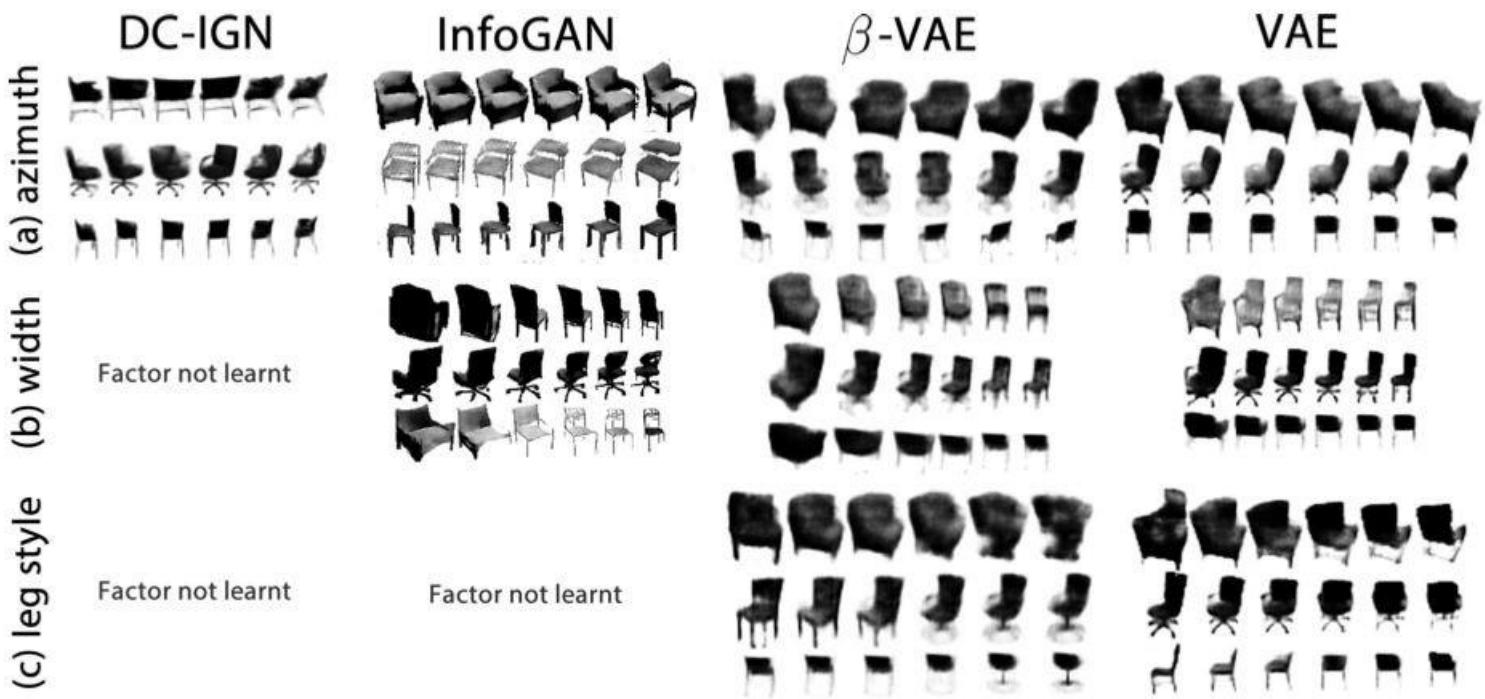
Applications

- A plethora of following work that:
  - **improves performance** on given task,
  - demonstrates usefulness of learned representations on **many other tasks** (ex. BERT, Mockingjay)

- A trend in learning representations for many problems - "anything2vec":
  - speech2vec: Chung, Yu-An and James R. Glass. “Speech2Vec: A Sequence-to-Sequence Framework for Learning Word Embeddings from Speech.” *INTERSPEECH* (2018).
  - node2vec: Grover, Aditya and J. Leskovec. “node2vec: Scalable Feature Learning for Networks.” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016): n. pag.
  - (batter|pitcher)2vec: Alcorn, Michael A.. “(batter|pitcher)2vec: Statistic-Free Talent Modeling With Neural Player Embeddings.” (2018).

Applications

- Recent trends:
  - Interpretability of learned representations



Applications

- Recent trends:
  - Fairness of learned representations - recidivism, health insurance, etc
    - Important that the learned representations do not encode biases from demographic features,
    - Learning Fair Representations: ***Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, Cynthia Dwork*** ; Proceedings of the 30th International Conference on Machine Learning, PMLR 28(3):325-333, 2013.
    - Learning Adversarially Fair and Transferrable Representations: ***David Madras, Elliot Creager, Toniann Pitassi, Richard Zemel*** *Proceedings of the 35th International Conference on Machine Learning*, PMLR 80:3384-3393, 2018.

Deep Learning

DR. PRAVEEN BLESSINGTON T
Professor
Email: praveentblessington@gmail.com
Mobile: 9730562120

Representation Learning

- Representation learning is a technique that involves automatically learning and extracting features or representations from raw data, which can then be used for a variety of downstream tasks such as classification, clustering, and prediction.
- It involves transforming raw data into a more meaningful and structured format. This allows the model to capture relevant patterns and features.
- It enables the model to extract features from the data rather than relying on hand-crafted features.

Continues....

- There are several techniques for representation learning in deep learning, including:
  - **Autoencoders:** Neural networks that learn to encode and decode data.
  - **Convolutional Neural Networks (CNNs):** Especially effective for image data.
  - **Recurrent Neural Networks (RNNs):** Suitable for sequential data.
  - **Word Embeddings:** Techniques like Word2Vec and GloVe for natural language processing.
  - **Self-Attention Mechanisms:** Used in transformers for tasks like language translation.

Greedy Layer wise Pre-training

- Greedy Layer wise Pre-training is a technique used in deep learning where a deep neural network is trained layer by layer in a greedy manner.
- Each layer is pre-trained as an unsupervised autoencoder and then fine-tuned using supervised learning.
- The greedy layer wise unsupervised pre training algorithm is a neural network training technique where the layers of a neural network are trained one at a time in an unsupervised manner, with the output of each layer being used as input for the next layer.
- This technique is used to improve the performance of deep neural networks.

Continues....

- The most well-known and widely used algorithm for greedy layer-wise unsupervised pretraining is the Restricted Boltzmann Machine (RBM) and Autoencoders.
- These algorithms allow each layer of a neural network to learn useful representations of the data progressively, starting from the input layer and moving deeper into the network.
- This process can help the network learn more abstract and hierarchical features of the data.
- Once each layer is pretrained, the entire network is typically fine-tuned using supervised learning, such as backpropagation, to learn the specific task or classification problem at hand.

Continues....

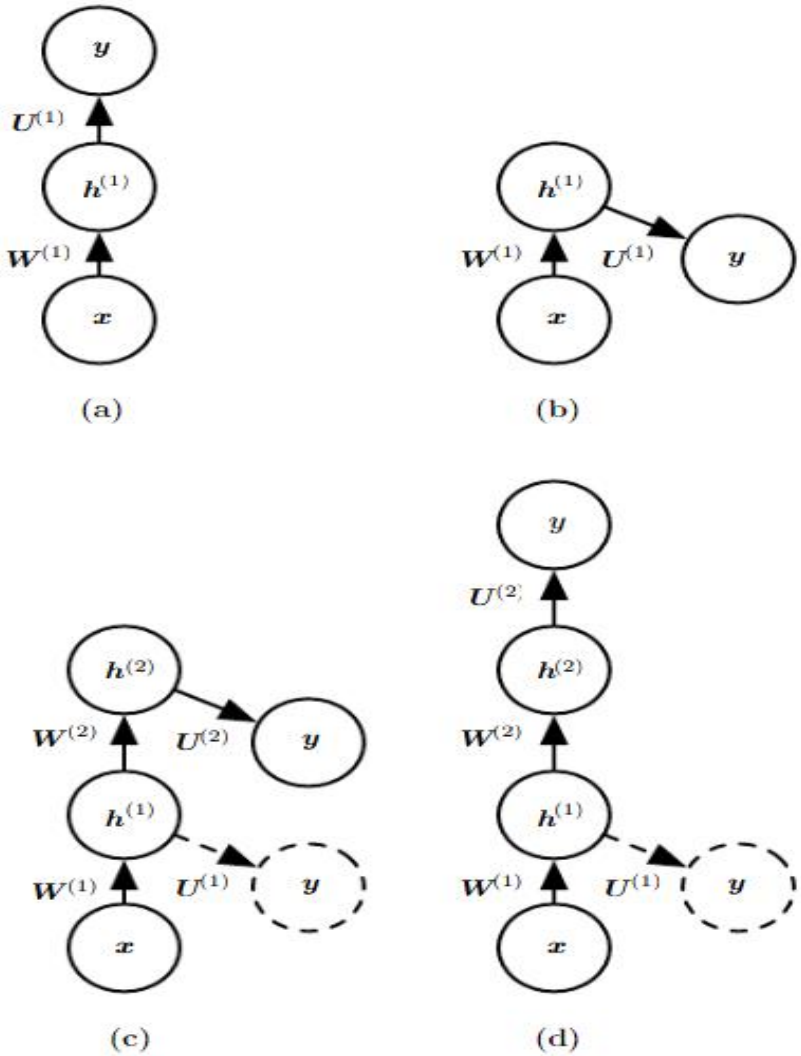
- The Greedy Layer-Wise Unsupervised Pre-Training algorithm differs from other pre-training techniques as it trains one layer at a time, starting from the input layer and moving towards the output layer.
- In each step, the algorithm trains a new layer by using the output of the previous layer as input. This process continues until all layers have been trained.
- The advantage of this approach is that it allows for better initialization of the weights in each layer, which can lead to improved performance in the final model. Additionally, it can be used to pre-train models on large amounts of unlabeled data, which can be particularly useful when labeled data is scarce.
- Other pre-training techniques, such as pre-training on autoencoders or using transfer learning, may use different approaches to initializing the weights in a deep neural network.

Continues....

Given the following: Unsupervised feature learning algorithm $\mathcal{L}$, which takes a training set of examples and returns an encoder or feature function f. The raw input data is $\boldsymbol{X}$, with one row per example, and $f^{(1)}(\boldsymbol{X})$ is the output of the first stage encoder on $\boldsymbol{X}$. In the case where fine-tuning is performed, we use a learner $\mathcal{T}$, which takes an initial function f, input examples $\boldsymbol{X}$ (and in the supervised fine-tuning case, associated targets $\boldsymbol{Y}$), and returns a tuned function. The number of stages is m.

$f \leftarrow$ Identity function
$\tilde{\boldsymbol{X}} = \boldsymbol{X}$
for $k = 1, \ldots, m$ **do**
  $f^{(k)} = \mathcal{L}(\tilde{\boldsymbol{X}})$
  $f \leftarrow f^{(k)} \circ f$
  $\tilde{\boldsymbol{X}} \leftarrow f^{(k)}(\tilde{\boldsymbol{X}})$
end for
if *fine-tuning* ***then***
  $f \leftarrow \mathcal{T}(f, \boldsymbol{X}, \boldsymbol{Y})$
end if
Return f

Continues....


y
U(1)
h(1)
W(1)
x
(a)
h(1)
W(1)
U(1)
y
x
(b)
h(2)
W(2)
U(2)
y
h(1)
W(1)
U(1)
y
x
(c)
y
U(2)
h(2)
W(2)
h(1)
W(1)
U(1)
y
x
(d)

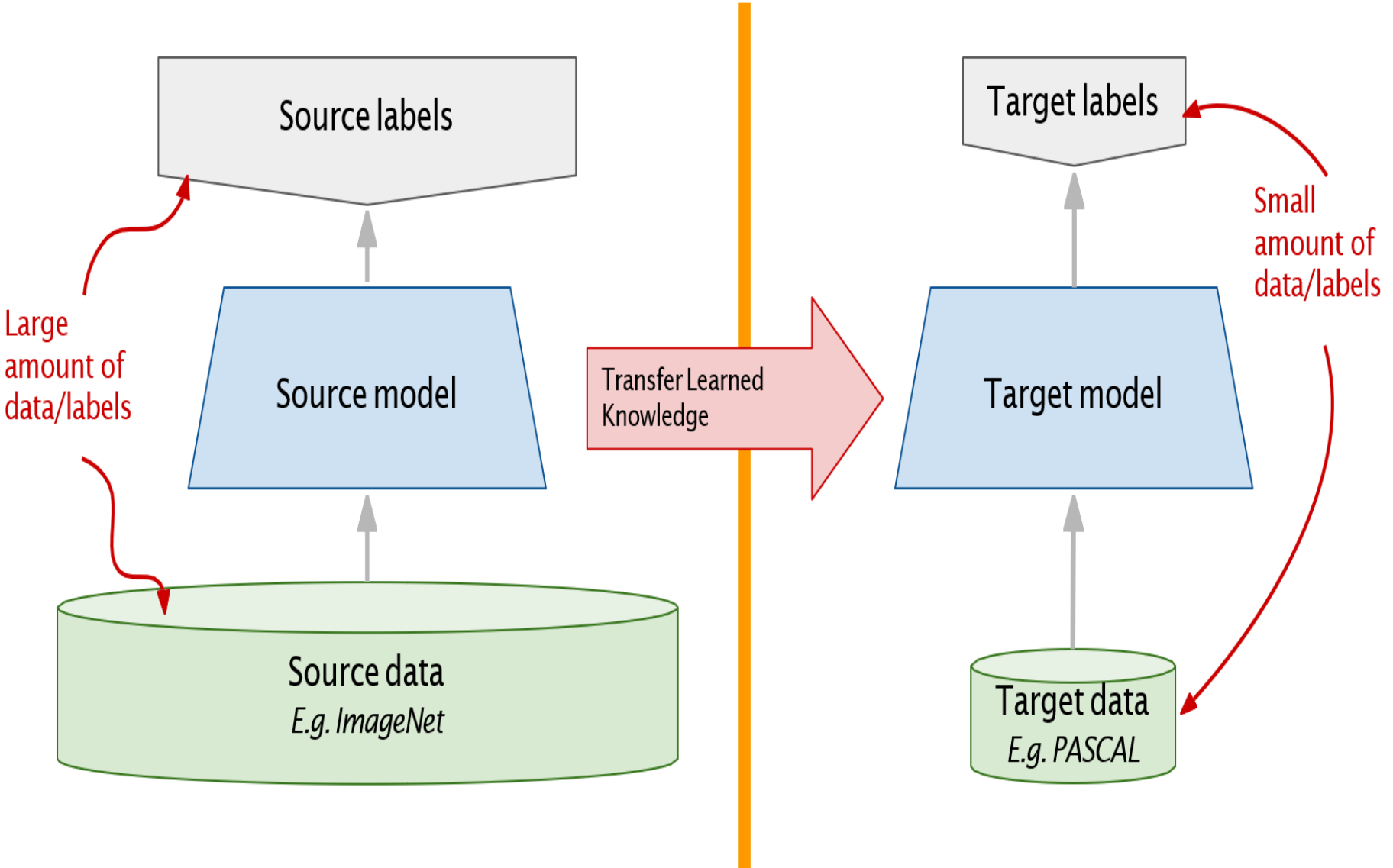
Transfer Learning and Domain Adaptation

- **Transfer learning** is a technique where a pre-trained model, which has learned from a large dataset, is used as a starting point for a new task or domain. It involves leveraging the knowledge gained from one task to improve performance on another related task.
- **Domain adaptation**, is a specific form of transfer learning where the source and target domains are different but related. It aims to adapt the pre-trained model to perform well on the target domain by reducing the distribution shift between the two domains.

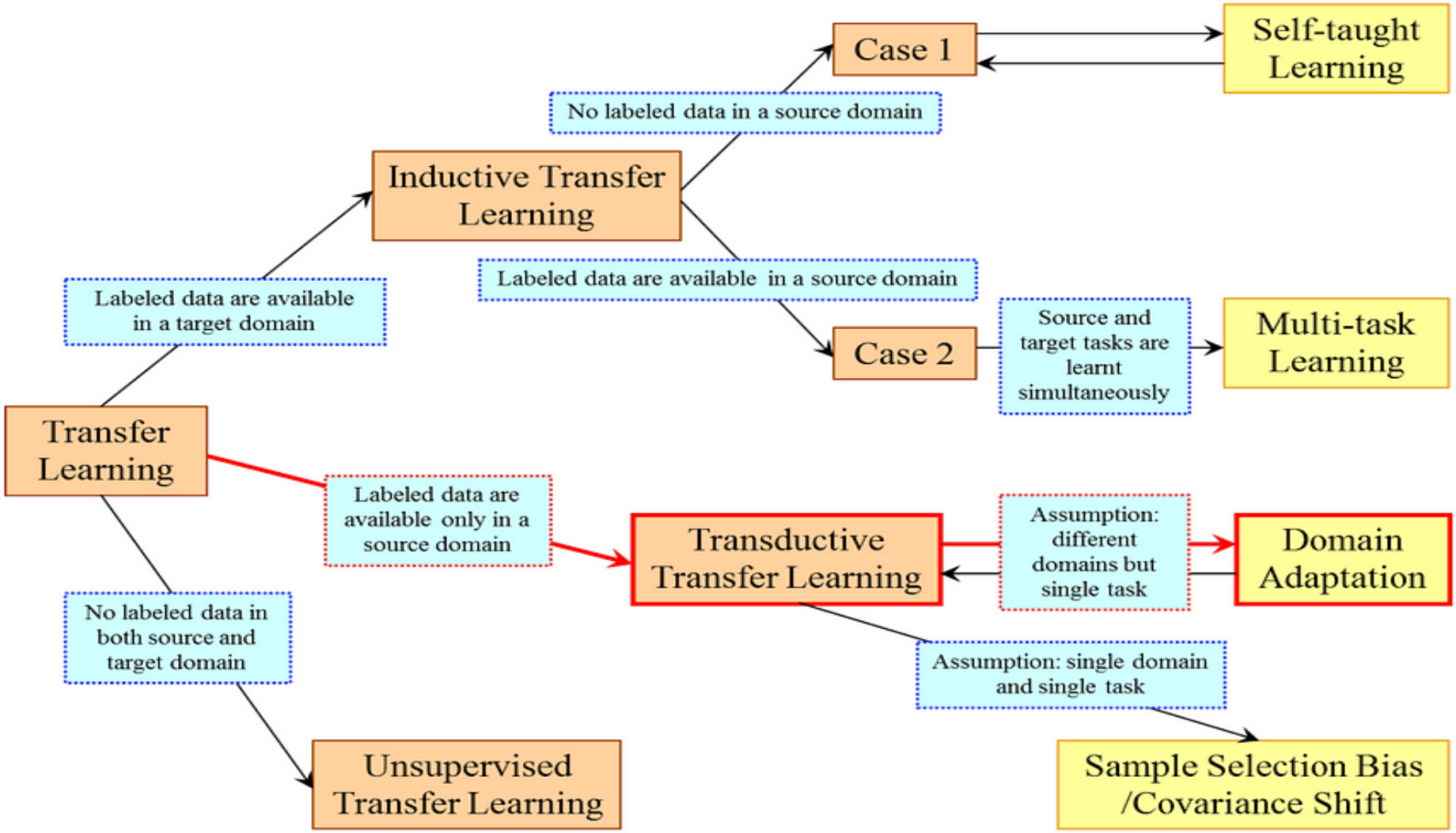
Transfer Learning

- **Myth:** you can't do deep learning unless you have a million labelled examples for your problem.
- **Reality:** You can learn useful representations from unlabeled data
  - You can transfer learned representations from a related task
  - You can train on a nearby surrogate objective for which it is easy to generate labels
- Instead of training a deep network from scratch for your task:
  - Take a network trained on a different domain for a different source task
  - Adapt it for your domain and your target task

Continues....


Source labels
Large amount of data/labels
Source model
Source data
E.g. ImageNet
Transfer Learned Knowledge
Target labels
Small amount of data/labels
Target model
Target data
E.g. PASCAL

Continues....


Transfer Learning
Labeled data are available in a target domain
Inductive Transfer Learning
No labeled data in a source domain
Case 1
Self-taught Learning
Labeled data are available in a source domain
Case 2
Source and target tasks are learnt simultaneously
Multi-task Learning
Labeled data are available only in a source domain
Transductive Transfer Learning
Assumption: different domains but single task
Domain Adaptation
Assumption: single domain and single task
Sample Selection Bias /Covariance Shift
No labeled data in both source and target domain
Unsupervised Transfer Learning

Domain Adaptation:

- Domain adaptation is a subset of transfer learning that deals with scenarios where the source and target domains are related but not identical. The goal is to adapt the knowledge from the source domain to improve performance in the target domain. There are two primary types of domain adaptation:
  - **Domain Shift:** This occurs when there's a difference between the data distribution in the source and target domains. Domain adaptation methods aim to reduce this distribution discrepancy.
  - **Unsupervised Domain Adaptation (UDA):** In UDA, you adapt a model from a labeled source domain to an unlabeled target domain. This can be challenging but is valuable when labeled data in the target domain is scarce.

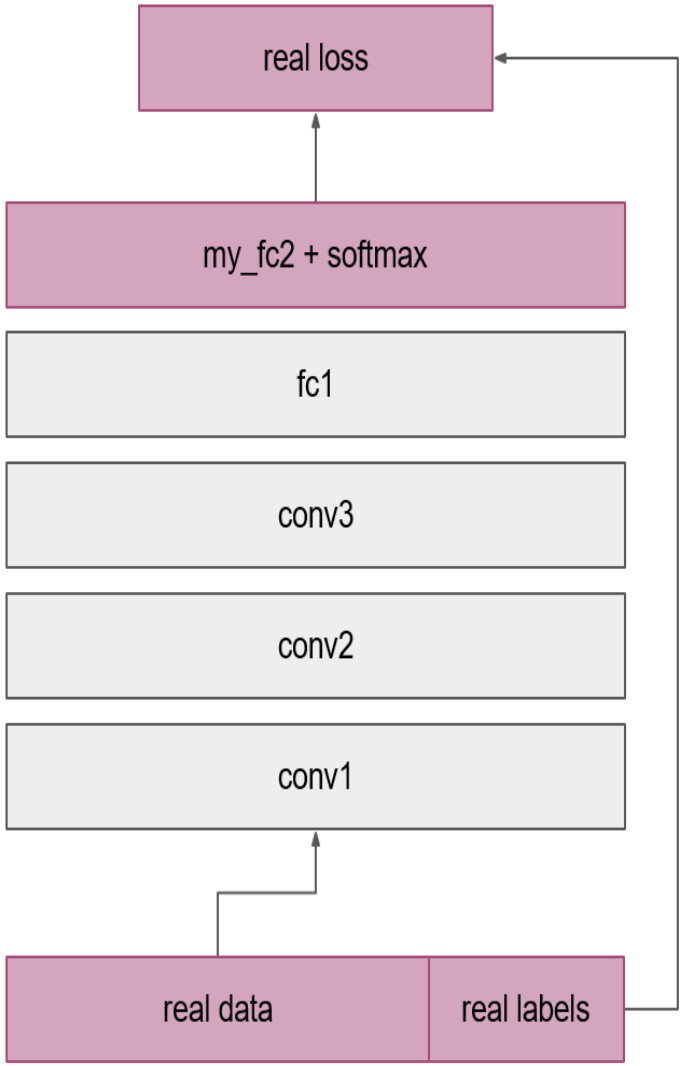
Continues….

Train deep net on “nearby” task for which it is easy to get labels using standard backprop

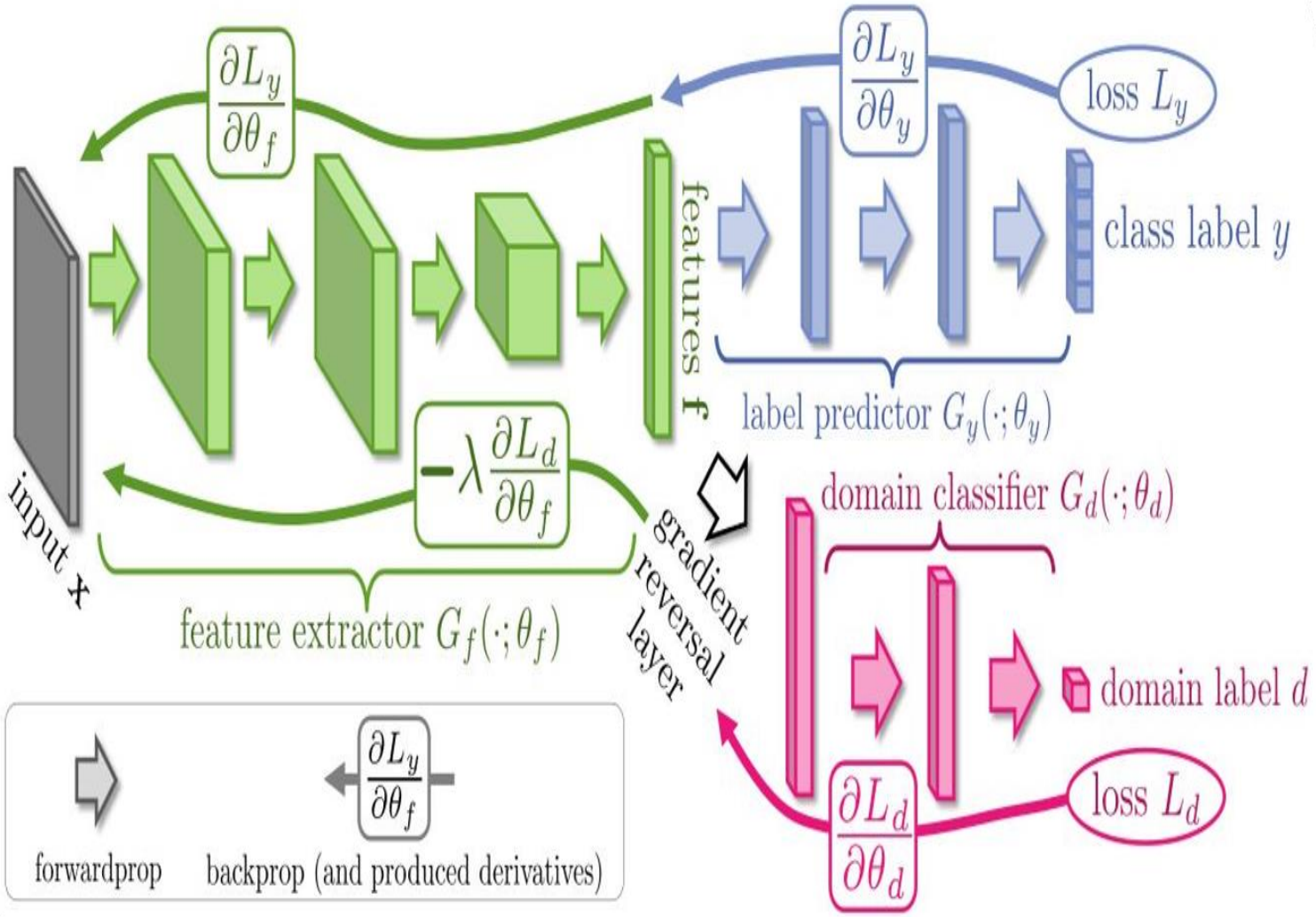
- E.g. ImageNet classification
- Pseudo classes from augmented data
- Slow feature learning, ego-motion

Cut off top layer(s) of network and replace with supervised objective for target domain

Fine-tune network using backprop with labels for target domain until validation loss starts to increase



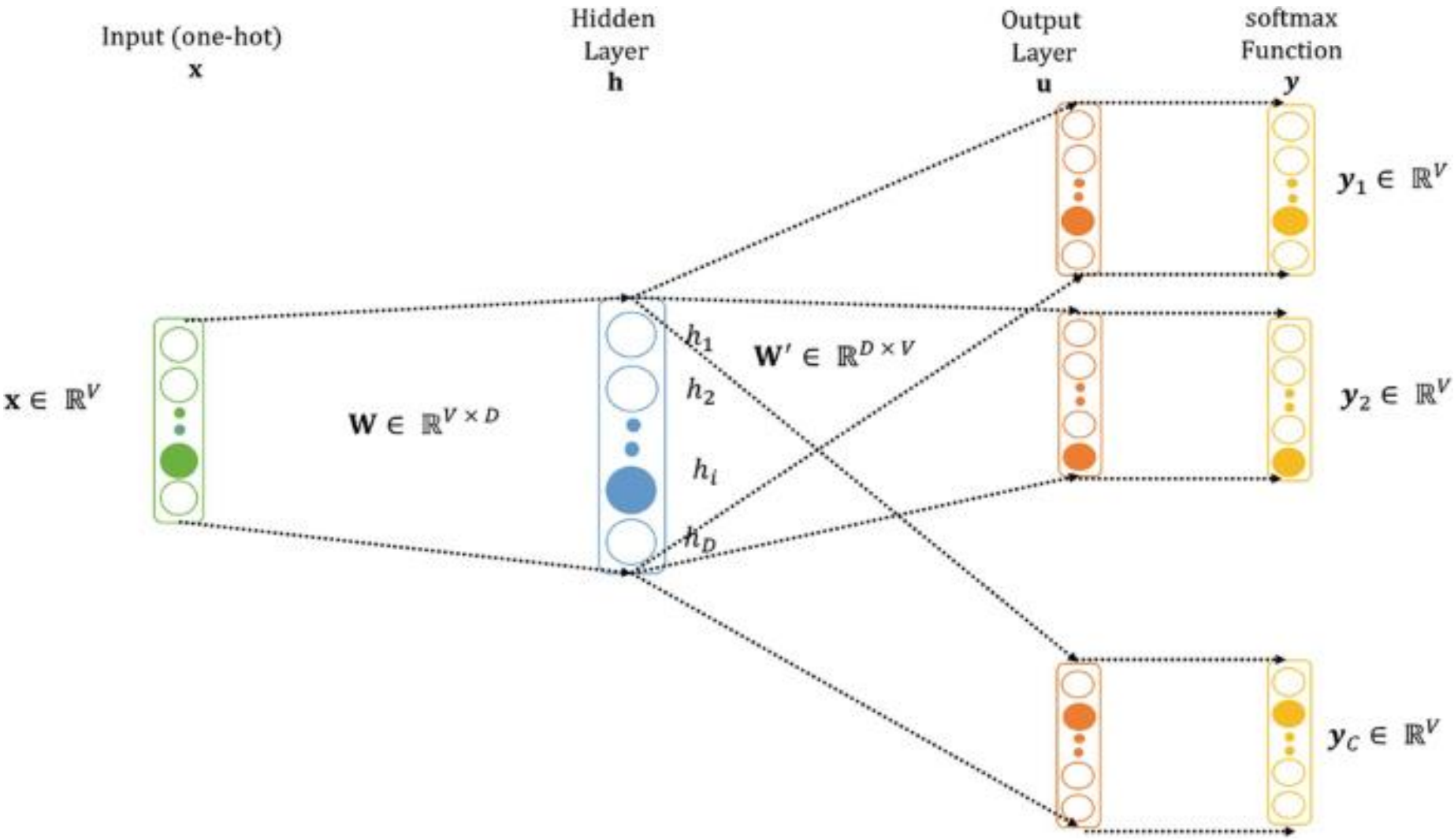
Continues....


$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
features $\mathbf{f}$
class label y
label predictor $G_y(\cdot;\theta_y)$
$-\lambda\frac{\partial L_d}{\partial \theta_f}$
input $\mathbf{x}$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
feature extractor $G_f(\cdot;\theta_f)$
domain label d
$\frac{\partial L_d}{\partial \theta_d}$
loss L_d
forwardprop
$\frac{\partial L_y}{\partial \theta_f}$
backprop (and produced derivatives)

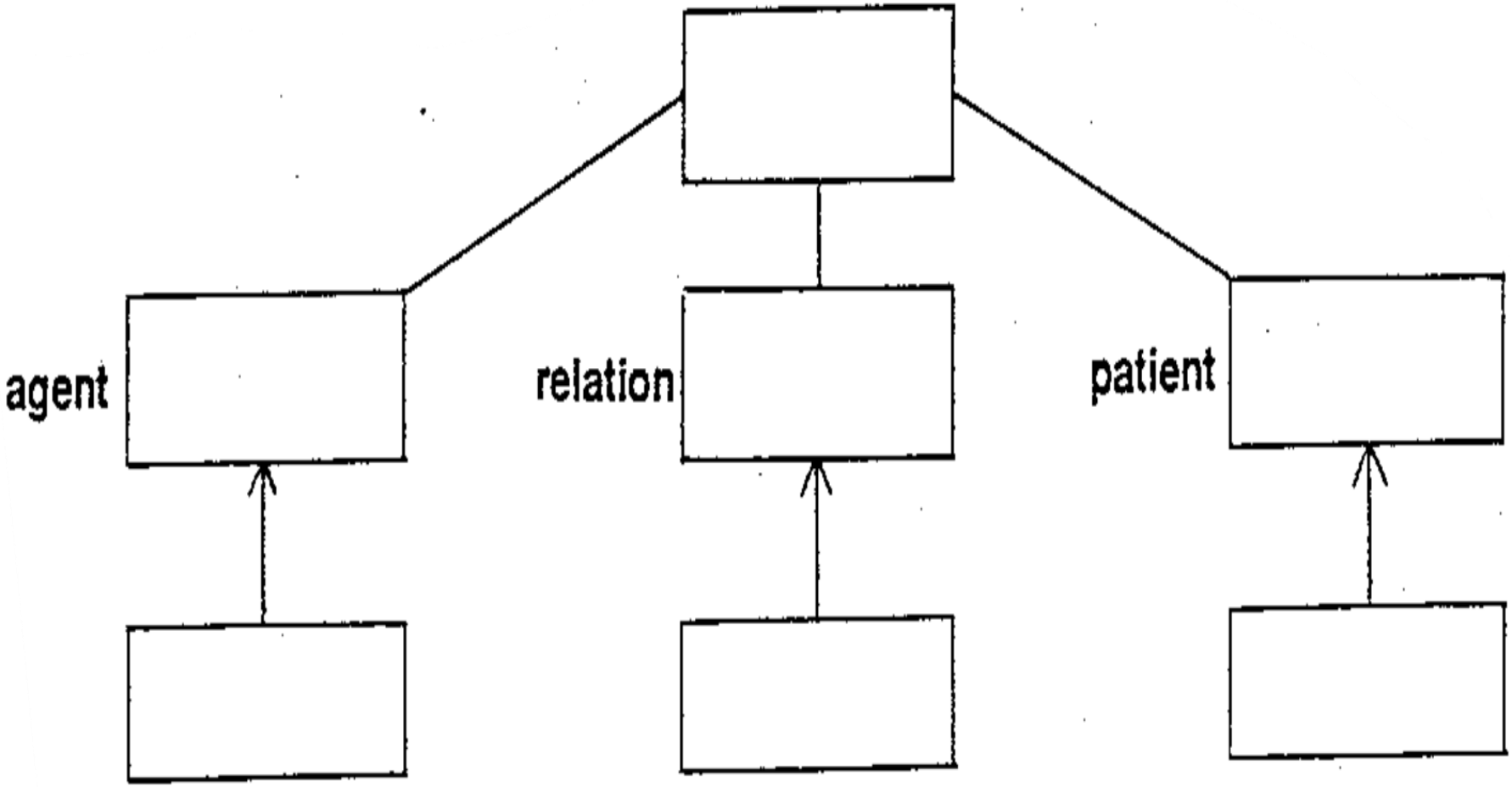
Distributed Representation:

- Distributed Representation is a concept in deep learning where each feature or attribute of an input data point is represented by multiple neurons in the neural network.
- This allows more efficient and effective learning as the network can identify relationships between different features.
- In deep learning, distributed representation is a way of representing information where each piece of the representation is spread across many features in a layer.
- The output of each node is interpreted as a feature, and the vector comprising these feature values provides information about the input patterns.

Continues....


Input (one-hot)
x
Hidden
Layer
h
Output
Layer
u
softmax
Function
y
$\mathbf{x} \in \mathbb{R}^V$
$\mathbf{W} \in \mathbb{R}^{V \times D}$
h_1
h_2
h_i
h_D
$\mathbf{W}' \in \mathbb{R}^{D \times V}$
$y_1 \in \mathbb{R}^V$
$y_2 \in \mathbb{R}^V$
$y_C \in \mathbb{R}^V$

Continues....



Variants of CNN: DenseNet

- Convolutional Neural Networks (CNNs) are a class of deep learning models primarily used for image and video analysis.
- DenseNet, short for Densely Connected Convolutional Networks, is a type of deep learning architecture for convolutional neural networks (CNNs) designed to improve training efficiency and model performance.
- DenseNet is a deep learning architecture that aims to improve the vanishing gradient problem in deep neural networks by connecting each layer to every other layer in a feed-forward fashion.

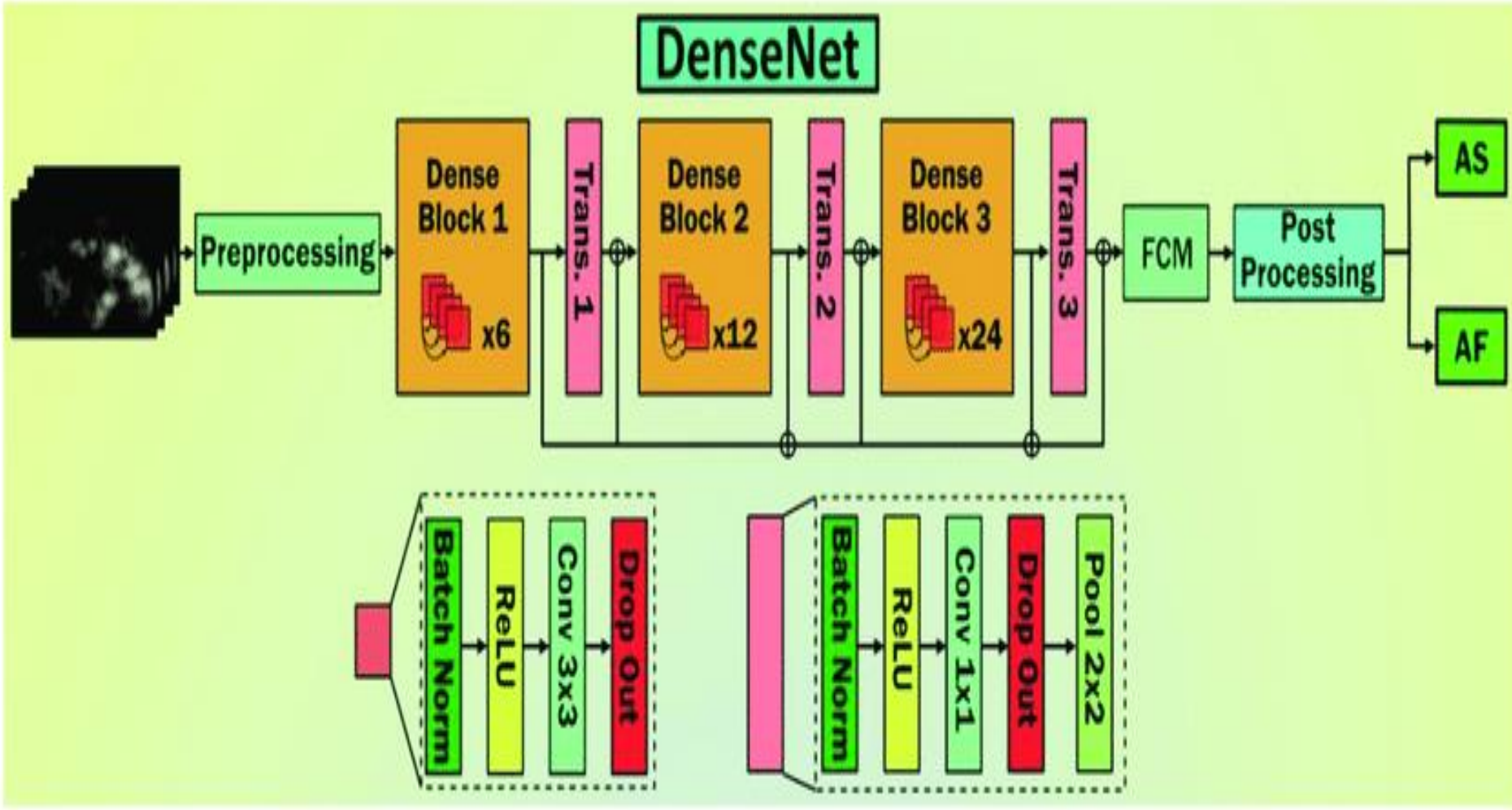
DenseNet Architecture

- In a traditional feed-forward Convolutional Neural Network (CNN), each convolutional layer except the first one (which takes in the input), receives the output of the previous convolutional layer and produces an output feature map that is then passed on to the next convolutional layer.
- Therefore, for 'L' layers, there are 'L' direct connections; one between each layer and the next layer.

Continues....

- However, as the number of layers in the CNN increase, i.e. as they get deeper, the 'vanishing gradient' problem arises.
- This means that as the path for information from the input to the output layers increases, it can cause certain information to 'vanish' or get lost which reduces the ability of the network to train effectively.
- DenseNets resolve this problem by modifying the standard CNN architecture and simplifying the connectivity pattern between layers.
- In a DenseNet architecture, each layer is connected directly with every other layer, hence the name Densely Connected Convolutional Network. For 'L' layers, there are L(L+1)/2 direct connections.

Continues....


DenseNet
Preprocessing
Dense Block 1
x6
Trans. 1
Dense Block 2
x12
Trans. 2
Dense Block 3
x24
Trans. 3
FCM
Post Processing
AS
AF
Batch Norm
ReLU
Conv 3x3
Drop Out
Batch Norm
ReLU
Conv 1x1
Drop Out
Pool 2x2

Continues....

- Components of DenseNet include:

1. Connectivity
2. DenseBlocks
3. Growth Rate
4. Bottleneck layers

1. Connectivity:

- In each layer, the feature maps of all the previous layers are not summed but concatenated and used as inputs. Consequently, DenseNets require fewer parameters than an equivalent traditional CNN, and this allows for feature reuse as redundant feature maps are discarded. So, the l^{th} layer receives the feature-maps of all preceding layers, $x_0,\ldots,x_{l-1}$, as input:

$$x_\ell = H_\ell([x_0, x_1, \ldots, x_{\ell-1}]),$$

2. DenseBlocks:

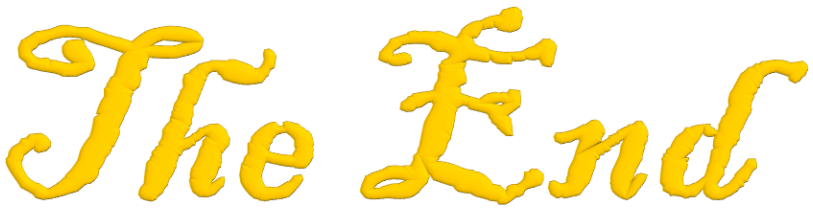
- The use of the concatenation operation is not feasible when the size of feature maps changes. However, an essential part of CNNs is the down-sampling of layers which reduces the size of feature-maps through dimensionality reduction to gain higher computation speeds.
- To enable this, DenseNets are divided into DenseBlocks, where the dimensions of the feature maps remains constant within a block, but the number of filters between them is changed. The layers between the blocks are called Transition Layers which reduce the the number of channels to half of that of the existing channels.
- For each layer, from the equation above, H_l is defined as a composite function which applies three consecutive operations: batch normalization (BN), a rectified linear unit (ReLU) and a convolution (Conv).

3. Growth Rate :

- One can think of the features as a global state of the network. The size of the feature map grows after a pass through each dense layer with each layer adding 'K' features on top of the global state (existing features). This parameter 'K' is referred to as the growth rate of the network, which regulates the amount of information added in each layer of the network. If each function H_l produces k feature maps, then the l^{th} layer has input feature-maps, where k_0 is the number of channels in the input layer. Unlike existing network architectures, DenseNets can have very narrow layers. $k_l = k_0 + k * (l - 1)$

4. Bottleneck layers:

- Although each layer only produces k output feature-maps, the number of inputs can be quite high, especially for further layers. Thus, a 1x1 convolution layer can be introduced as a bottleneck layer before each 3x3 convolution to improve the efficiency and speed of computations.


The End