Name:

CMSC 838B & 498Z: Differentiable Programming

Tues/Thur 12:30pm – 1:45pm
IRB 4105 (T) & IRB 5105 (R)
http://www.cs.umd.edu/class/fall2021/cmsc838b

Ming C. Lin

IRB 5162

lin@cs.umd.edu

http://www.cs.umd.edu/~lin

Office Hours: After Class or By Appointment

M. C. Lin

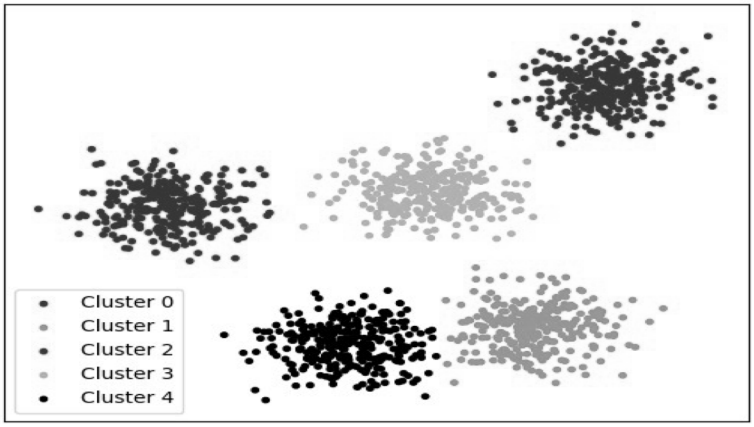
Introduction to Learning

- **Unsupervised Learning - discover a good internal representation of the input**
- **Supervised Learning - learn to predict an output when given an input vector**
- **Reinforcement Learning - learn to select an action to maximize the expectation of future rewards (payoff)**
- **Self-supervised Learning - learn with targets induced by a prior on the unlabelled training data**
- **Semi-supervised Learning - learn with few labelled examples and many unlabelled ones**

M. C. Lin

Name:

Unsupervised Learning



not provided with any pre-assigned labels or scores for training data

M. C. Lin

Supervised Learning

"*Stacked hourglass networks for human pose estimation*"
by Newell, Alejandro, Kaiyu Yang, and Jia Deng. ECCV 2016.

M. C. Lin

Name:

Types of Supervised Learning

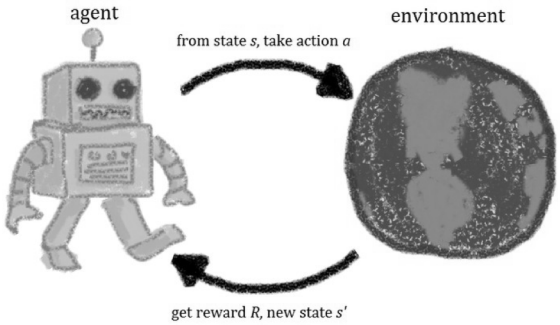
- ***Regression***: **The machine is asked to predict k numerical values given some input. The machine is a function**

  f: $R^n \rightarrow R^k$
- ***Classification***: **The machine is asked to specify which of k categories some input belongs to:**
  - **Multiclass classification - target is one of the k classes**
  - **Multilabel classification - target is some number of the k classes**
  - **In both cases, the machine is a function**

    f: $R^n \rightarrow \{1 \ldots k\}$. But, it is most common for the learning algorithm to actually learn f: $R^n \rightarrow R^k$

M. C. Lin

Reinforcement Learning

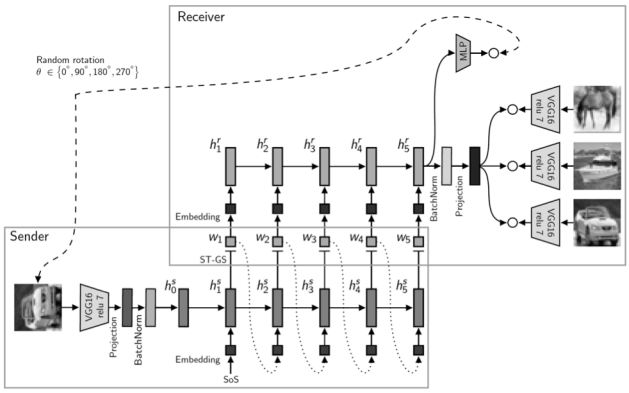


- a set of environment and agent states, *S*
- a set of actions, *A*, of the agent
- *Pa(s,s')* is probability of transition at time t from states *s* to *s'* under action *a*
- *Ra(s,s')* is the immediate reward after transition from *s* to *s*' with action *a*

M. C. Lin

Name:

Self-Supervised Learning



1. Pretext task solved based on pseudo-labels to initialize network weights
2. Complex downstream task (e.g. speech recognition) that computes the actual task to be performed with supervised or unsupervised learning

**Not necessarily require explicit data labeling/classified by human*

M. C. Lin

Semi-Supervised Learning

1. Start with unlabelled image data.

2. Use deep learning to automatically find structure in image.

3. Human adds labels to different clusters successfully classified, and distinguishes between images classified incorrectly.

4. Repeat stages 2&3 (re-train classifier using this additional info.

M. C. Lin

Name:

Generative Models

- Many unsupervised and self-supervised models can be classed as 'Generative Models'
- Given unlabelled data X, a unsupervised generative model learns P[X]:
  - Could be direct modelling of the data (e.g. Gaussian Mixture Models)
  - Could be indirect modelling by learning to map the data to a parametric distribution in a lower dimensional space (e.g. a VAE's Encoder) or by learning a mapping from a parameterized distribution to the real data space (e.g. a VAE Decoder or GAN)
- These are characterised by an ability to 'sample' the model to `create' new data

M. C. Lin

Generative vs. Discriminative Models

***For classification, use different statistical modeling*:**

- ***Discriminative*** **models learn the boundary between classes. A (probabilistic) discriminative model is a model of the conditional probability of the target Y given an observation X: P[Y|X]**
- ***Generative*** **models of labelled data model the distribution of individual classes. Given an observable variable X and a target variable Y , a generative model is a statistical model that tries to model P[X|Y] & P[Y] in order to model the joint probability distribution P[X,Y]**

M. C. Lin

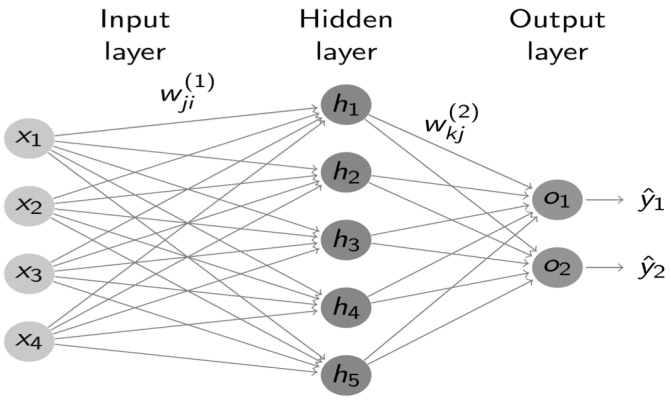
Name:

How Supervised Learning Often Works

- Start by choosing a model-class, $\hat{y} = f(x;W)$, where the model-class f is a way of using some numerical parameters, W, to map each input vector x to a predicted output $\hat{y}$
- Learning means adjusting the parameters to reduce the discrepancy between the true target output y on each training case and the output $\hat{y}$, predicted by the model

M. C. Lin

Artificial Neural Network



Without loss of generality, we can write the above as: $\hat{y} = g(\ f(x;W^{(1)});\ W^{(2)}) = g(W^{(2)}\ f(W^{(1)}x))$, where f and g are activation functions

M. C. Lin

Name:

Terminology

- ***Learning***: adaptation of the network to better handle a task by considering sample observations, e.g. adjusting the weights (and optional thresholds) of the network to improve the accuracy of the result (e.g. error rates → 0)
- ***Propagation function*** computes the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum. A *bias* term can be added to the result of the propagation.
- **Organization:** the neurons are typically organized into multiple layers, esp. in DNN
- **Hyperparameter**: a constant parameter whose value is set before the learning process begins. The values of parameters are derived via learning, e.g. learning rate (stepsize), the number of hidden layers, batch size, etc **M. C. Lin**

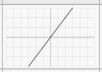
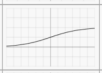
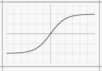
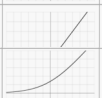
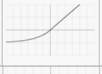
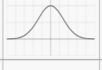
Activation Function

- To find the output of the neuron, first we take the weighted sum of all the inputs, weighted by the *weights* of the ***connections*** from the inputs to the neuron.
- We add a *bias* term to this sum. This weighted sum is sometimes called the ***activation***.
- This weighted sum is then passed through a (usually nonlinear) ***'activation function***' to produce the output.
- The initial *inputs* are external data, such as images and documents. The ultimate *outputs* accomplish the task, such as recognizing an object in an image.

M. C. Lin

Name:

Common Activation Functions

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0,1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$[1]	$f(x)(1-f(x))$	$(0,1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1,1)$	C^∞
Rectified linear unit (ReLU)[7]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}$	$[0, \infty)$	C^0
Softplus[8]		$\ln(1+e^x)$	$\frac{1}{1+e^{-x}}$	$(0, \infty)$	C^∞
Exponential linear unit (ELU)[9]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$
Gaussian		e^{-x^2}	$-2xe^{-x^2}$	$(0,1]$	C^∞
Growing Cosine Unit (GCU)[17]		$x\cos(x)$	$\cos(x) - x\sin(x)$	$(-\infty, \infty)$	C^∞

Final layer activations

- What form should the final layer function *g* take?
- It depends on the task (and on the chosen loss function)...
  - For regression it is typically linear (e.g. identity), but you might choose others if you say wanted to clamp the range of the network.
  - For binary classification (MLP has a single output), use Sigmoid
  - For multilabel classification, typically one would choose Sigmoid
  - For multiclass classification, typically use the Softmax function:

Name	Equation, $f_i(\vec{x})$	Derivatives, $\frac{\partial f_i(\vec{x})}{\partial x_j}$	Range	Order of continuity
Softmax	$\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$	$f_i(\vec{x})\left(\delta_{ij} - f_j(\vec{x})\right)$[3][4]	$(0,1)$	C^∞
Maxout[18]	$\max_i x_i$	$\begin{cases} 1 & \text{if } j = \operatorname{argmax}_i x_i \\ 0 & \text{if } j \neq \operatorname{argmax}_i x_i \end{cases}$	$(-\infty, \infty)$	C^0

M. C. Lin

Name:

Loss Functions

- The choice of loss function depends on the task (e.g. classification, regression, or something else)
- The choice also depends on the activation function of the last layer
  - For numerical reasons, often the activation is computed directly within the loss rather than being part of the model
  - Some classification losses require raw outputs (e.g. a linear layer) of the network as their input
    - These are often called unnormalized log probabilities or logits
    - An example would be hinge-loss used to create a Support Vector Machine that maximizes the margin -- e.g.: l_{hinge} (^y; y) = max(0; 1 - y * ^y) with a true label, y ∈ {-1; 1} for binary classification
- There are many different loss functions we might encounter (MSE, Cross-Entropy, KL-Divergence, huber, L1 (MAE), CTC, Triplet, ...) for different tasks

M. C. Lin

Cost Function (measure of discrepancy)

- Mean Squared Error (MSE) loss for a single data point (here assumed to be a vector, but equally applicable to a scalar):

$$\ell_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_i (\hat{y}_i - y_i)^2 = (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})$$

- l_{MSE} *(^y; y)* is the predominant choice for regression problems with linear activation in the last layer
- For a classification problem with Softmax or Sigmoidal (or really anything non-linear) activations, MSE can cause slow learning, especially if predictions are very far off the targets
  - Gradients of l_{MSE} are proportional to the difference in target and predicted multiplied by the gradient of the activation function
  - The Cross-Entropy loss function is generally a better choice here

M. C. Lin

Name:

Multiclass classification w/ Softmax Outputs

- Softmax can be thought of making the K outputs of the network mimic a probability distribution
- The target label y could also be represented as a distribution with a single 1 and zeros everywhere else. e.g. they are "one-hot encoded"
- In such a case, the obvious loss function is the negative log likelihood of the Categorical distribution (aka Multinoulli, Generalized Bernoulli, Multinomial with one sample):

$$\ell_{NLL} = -\sum_{k=1}^{K} y_k \log \hat{y}_k$$

M. C. Lin

Gradient Descent

- Define total loss as $\mathcal{L} = \sum_{(\boldsymbol{x},y)\in \boldsymbol{D}} \ell(g(\boldsymbol{x}, \boldsymbol{\theta}), y)$ for some loss function l, dataset D and model g with learnable parameters $\ominus$
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate n
- ***Gradient Descent*** updates the parameters $\ominus$ by moving them in the direction of the negative gradient with respect to the total loss $\mathcal{L}$ by learning rate n multiplied by the gradient:

for each Epoch:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$$

M. C. Lin

Name:

Stochastic Gradient Descent

- Define loss function l, dataset D and model g with learnable parameters $\ominus$
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η
- ***Stochastic Gradient Descent*** updates the parameters $\ominus$ by moving them in the direction of the negative gradient with respect to the loss of a single item l by the learning rate η multiplied by the gradient:

  for each Epoch:

$$\texttt{for each } (\boldsymbol{x}, y) \in \boldsymbol{D}\texttt{:}$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \ell$$

M. C. Lin