ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE

Paper ID #37016

Pacman Trainer: Classroom-Ready Deep Learning from Data to Deployment

Masao Kitamura (Loyola Marymount University)

Mandy Barrett Korpusik (Assistant Professor)

Dr. Korpusik is an Assistant Professor of Computer Science at Loyola Marymount University. She received her B.S. in Electrical and Computer Engineering from Franklin W. Olin College of Engineering and completed her S.M. and Ph.D. in Computer Science at MIT. Her primary research interests include natural language processing and spoken language understanding for dialogue systems. Dr. Korpusik used deep learning models to build the Coco Nutritionist application for iOS that allows obesity patients to more easily track the food they eat by speaking naturally. This system was patented, as well as her work at FXPAL using deep learning for purchase intent prediction.

Andrew Forney (Andrew Forney, Ph.D)

Pacman Trainer:
Classroom-Ready Deep Learning from Data to Deployment

Masao Kitamura, Mandy Korpusik, and Andrew Forney

Abstract

Deep learning has seen a meteoric rise in the machine learning community and has vastly changed the landscape of many fields like computer vision and natural language processing. Yet, for all of its successful applications, connecting deep learning theory to practice remains a challenge in the classroom, with students typically only seeing parts of the data-collection, training, and deployment process at a time. To address these difficulties, this work presents Pacman Trainer, a web application that can be used as a class activity in which students provide the best move for Pacman to take in a given maze and prompt, thus generating a labeled dataset for supervised deep-imitation learning. By experiencing first-hand how training data is labeled, sanitized, vectorized, used during training, and then deployed to control Pacman in the same environment, students grasp the entirety of the deep learning pipeline concretely. This experience also highlights the shortcomings of deep imitation learning, which segues to discussions of overfitting, generalizability, and reinforcement learning alternatives, in which Pacman agents can be trained online in the same environment to juxtapose learning paradigms. Classroom-ready instructions, examples, and accessory exercises are provided using Pytorch, complete with clonable repositories suitable for GitHub Classroom integration.

Introduction

Deep neural networks have been studied for decades in many domains of machine learning, yet initially lacked the required training data, computational power, and backpropagation algorithm [1] for training the models that are applied widely today. With large datasets of speech, text, and images publicly available online (or easily harvestable from crowdsourcing platforms like Mechanical Turk), free deep learning (DL) frameworks like Pytorch and Tensorflow, and fast graphical processing units (GPUs), modelers can train expressive, deep neural networks with ease in a number of popular applied domains. In the field of computer vision, convolutional neural networks (CNNs) are used for image segmentation (i.e., outlining all the objects within an image) [2], object recognition (identifying specific objects, such as animals or furniture) [3, 4], and scene classification [5]. DL is used in self-driving cars (e.g., object detection), conversational agents such as Siri and Alexa, Google search, and translation from a source language to a target language [6]. These instances are but several applications of the many in the supervised learning domain, as DL has also found footholds in reinforcement learning with successes like Google's AlphaStar capable of playing the complex game of Starcraft [7].

Yet, for all of these applications, challenges still exist in the deep-learning educational pipeline. Focusing herein on the DL introduced in undergraduate artificial intelligence (AI) and machine learning (ML) courses, students may be left with several blind spots: they rarely encounter the finer details of the data gathering/labeling process nor appreciate the massive amounts often required for complex tasks; they may find the typical performance metrics like classification accuracy, precision, etc. or the tasks of reproducing others' models to be unengaging [8]; they may be left with abstract (rather than concrete) warnings about overfitting and a model's ability to generalize or transport to different environments on which it was trained. It is these pain points, among others, that the present work hopes to ameliorate. We approach these by first reviewing related efforts in ML and DL education, outlining this paper's contributions, establishing a cursory background on the technical details that follow, and finally present a classroom-ready deep learning trainer and agent situated in the engaging and familiar Pacman environment.

Related Work

AI educational tools in the Pacman environment have been employed in the past, most notably by Berkeley's "Pacman Projects" to teach introductory lessons in topics like search, filtering, and q-learning [9]. However, these lessons are largely within the online-learning domain and stop short of discussing deep learning either in the imitation or reinforcement learning contexts. Others have picked up this thread and adapted deep-Q-networks to learn in the Pacman environment online [10], but in ways that focus less on the particulars of deep learning's fundamentals (let alone from an educational bearing) and with more of a focus on the details of online learning.

Contributions & Outline

The distinguishing contributions of Pacman Trainer (PT), and likewise, the portions of the DL educational pipeline that it addresses, are as follows:

Contribution	Problem	Proposed Solution
C_1	**Example problems can be large, intimidating, and unfamiliar** in typical deep learning application tutorials.	PT is situated in the **familiar Pacman environment** with which the vast majority of students are familiar.
C_2	The **data collection and labeling pipeline** is left as a mystery without exposing students to the process and labor involved.	PT provides an interface for **in-class data collection** that can be instantly culled and distributed to its members.
C_3	The **effects of different model, structural, and hyperparameter choices** can be obscured by simple accuracy reports.	Students get to **observe the effects of their choices** on the Pacman agent that performs in real time.
C_4	The **overfitting / generalizability discussion** is mentioned as a warning without hands-on illustration of its challenges.	PT **enables experimentation with the richness of environments** that contrast between training and deployment (e.g., different pellet positions, with and without ghosts, etc.)

Background

This work is intended primarily for educators already familiar with deep learning, so we provide only a cursory overview of its many components herein. These definitions and components also comprise the various "choice-points" that students will be able to experiment with during the design of their Pacman agents, though the intuitions and details of each are left for the context of the full course in which Pacman Trainer is intended to be situated.

Neural network models are called "neural" because they are inspired by the neurons in the human brain. They are more expressive than previous statistical models used in machine learning since they apply *non-linear* activation functions to each layer and can stack multiple layers in sequence (which is where the name "deep" learning comes from), enabling the models to learn more complex functions than with simpler linear models.

Network Structures: The standard neural network (employed herein), is a fully-connected, feed-forward neural network composed of a sequence of dense layers of neurons that are each connected to each of the neurons in the subsequent layer, as shown in Figure 1. In the PyTorch [11] tool detailed in the Pacman Agent tutorial of this work, these are called `linear` layers.

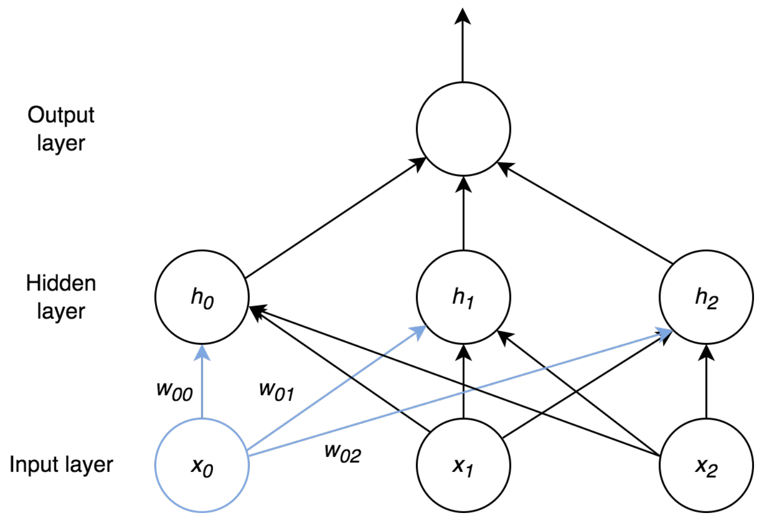


Figure 1: A traditional fully-connected neural network. Each input neuron x_i is connected to each hidden layer neuron h_i, which is computed as the weighted sum of the input neurons.

The bottom layer is the input layer, where each input neuron represents one feature. In natural language processing, the input may be a sentence, and each neuron represents one word; in computer vision, the input is an image, and each neuron is one pixel value. The intermediate layers are called "hidden" layers, and the final layer generates the output, which may be a single value between 0 and 1 in a binary classification task or the probability for each class in a categorical task (e.g., the probability that an input sentence is negative, positive or neutral in a sentiment analysis task). In the Pacman Agent setting, the input layer composes the maze state, and the output, an activation for each of the 4 directions in which Pacman may move.

The output of each hidden layers' neurons h_i in the sample diagram are computed as follows:

$$h_i = f(w_{0i}x_0 + w_{1i}x_1 + w_{2i}x_2 + b_h) \tag{1}$$

where w is a learned weight, x is the input, b_h is a learned bias term, and f is an activation function.

Activation Functions: Activation functions decide the means by which a weighted input from a previous layer is propagated to the next at each unit/neuron. Typically, the non-linear activation function f is a variant of the rectified linear unit (ReLU) function [12], $max(0, x)$, which is preferred over formerly used sigmoid or hyperbolic tangent (tanh) functions since it is less prone to the vanishing gradients problem because its slope does not plateau when it gets large.

Training and Hyperparameters: Neural networks are trained using an algorithm known as back-propagation, which works by sequentially computing gradients of the loss function with respect to the weights at that layer, starting with the final output layer and working backwards to the first input layer by applying the chain rule from calculus. The weights are updated through gradient descent, where the gradient with respect to a given weight is multiplied by the step size and subtracted from the current value of that weight: $w^{t+1} = w^t - \alpha\frac{dL}{dw}$, where w^t is the weight at timestep t, L is the loss, and α is the learning rate, or step size. The loss function is one hyperparameter that may be tuned on the development set. Simple loss functions include mean-squared error (MSE), which is the sum of each training example's squared error between the true label y and the predicted output of the model, or the cross-entropy loss for classification.

Another hyperparameter that should be tuned is the optimizer. Stochastic gradient descent (SGD) is the classic approach, which also requires tuning the learning rate α and a momentum parameter that determines the tradeoff between how heavily to weight the current sample's update versus the *momentum* from previous samples' updates. The preferred optimizer today is the *Ad*am optimizer, which dynamically *ad*apts the learning rate as training progresses [13].

Overfitting and Transportability: A common issue in machine learning, and deep learning in particular due to the models' power, is that of overfitting [14, 15]. If there is insufficient training data, or the model is too powerful, the model may memorize the training dataset and generalize poorly to an unseen test set. This challenge is often related to the *transportability* problem in which a model trained in an environment with different constraints than the one in which it is deployed may suffer degraded performance [16, 17]. Students should be able to juxtapose these different challenges and appreciate how a lack of sample diversity contributes to each.

Methods

Outline: This work aims to provide students with concrete, hands-on experience with *all* stages of the deep learning pipeline (including its challenges and limits) as intimated in the previous section. Illustrated in Figure 2, we partition this effort as follows:

- *Tasks*: decompose the deep learning pipeline into three, sequentially-dependent conceptual categories: data collection, model training / tuning, and tests for generalizability.
- *Learning Tools:* the main deliverables of the present work: we provide exercises and accompanying code for each of the given tasks.

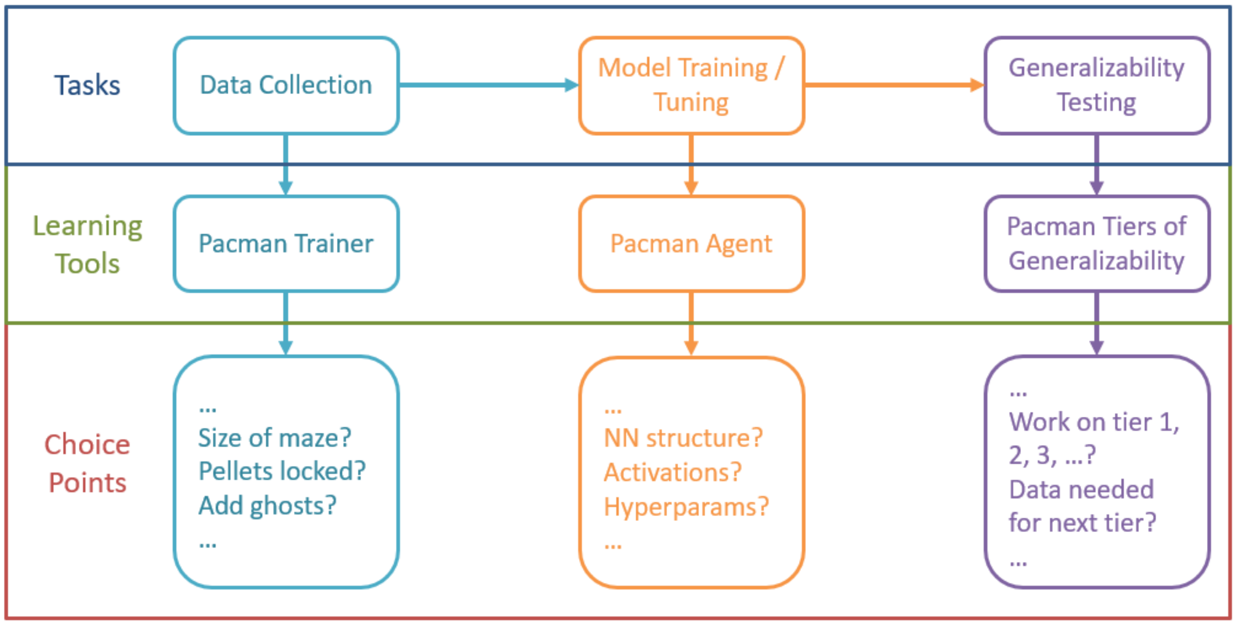

Tasks
Data Collection
Model Training / Tuning
Generalizability Testing
Learning Tools
Pacman Trainer
Pacman Agent
Pacman Tiers of Generalizability
Choice Points
... Size of maze? Pellets locked? Add ghosts? ...
... NN structure? Activations? Hyperparams? ...
... Work on tier 1, 2, 3, ...? Data needed for next tier? ...

Figure 2: Outline of methods by task, associated learning tools, and student choice-points.

- *Choice Points:* experimental steps or conceptual questions that accompany each of the learning tools. These serve as explicit points at which students must make important decisions that will affect the resulting model or to validate its generalizability.

Environment: For each of these tasks, students will find examples in the familiar game of Pacman, a gridworld with simple objectives:

- The model controls the movements of Pacman in a grid maze in any of the cardinal directions, $[Up, Down, Left, Right]$.
- The objective is to collect all of the "food pellets" (white dots within the maze) while avoiding adversarial ghosts that seek to eat Pacman.
- A single game ends if either Pacman eats all of the maze's pellets or is eaten by a ghost.

Student Objective: The student's overall task at-hand is to craft a **policy** that successfully[1] controls Pacman, though is learned through the tenets of *supervised / imitation learning* from samples that are gathered from the class itself. Just as most *classification tasks* must assign some label to some input, the policy's (π) job is to map some state (s) of the Pacman board to some action (a) that is best to take: $\pi(s) = a$

What follows is a detailed description of each learning tool presented herein alongside suggested choice points with which students may experiment for the purposes of deeply understanding the deep-learning pipeline (and its limitations).

[1]Here, "successful" control of Pacman is dictated by the transportability tier being solved (detailed later). In general, the idea of a successful policy in this environment is one that efficiently collects pellets while avoiding ghosts.

Pacman Trainer (PT)

Pacman Trainer (PT)[2] is a web application akin to data crowdsourcing platforms like Amazon's Mechanical Turk [https://www.mturk.com/] and the more academically-oriented Prolific [https://prolific.co/]. The general recipe for these platforms is to provide some *job* that has been posted by a *requester* and is then completed by *workers* who are compensated by the requester for their efforts. PT follows this same recipe, though without the hassle of requiring student workers to set up accounts, and is restricted to tasks only within the Pacman environment. The general steps of the PT workflow are depicted in Figure 3 and are detailed next.

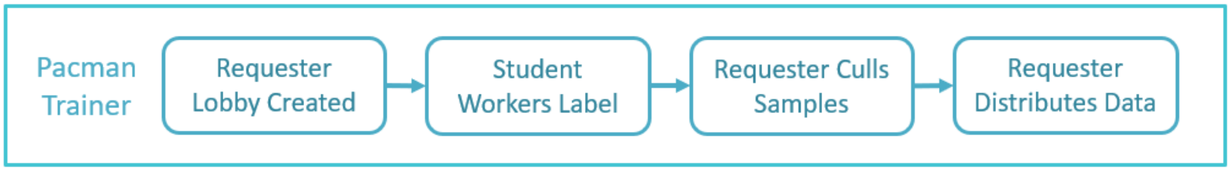
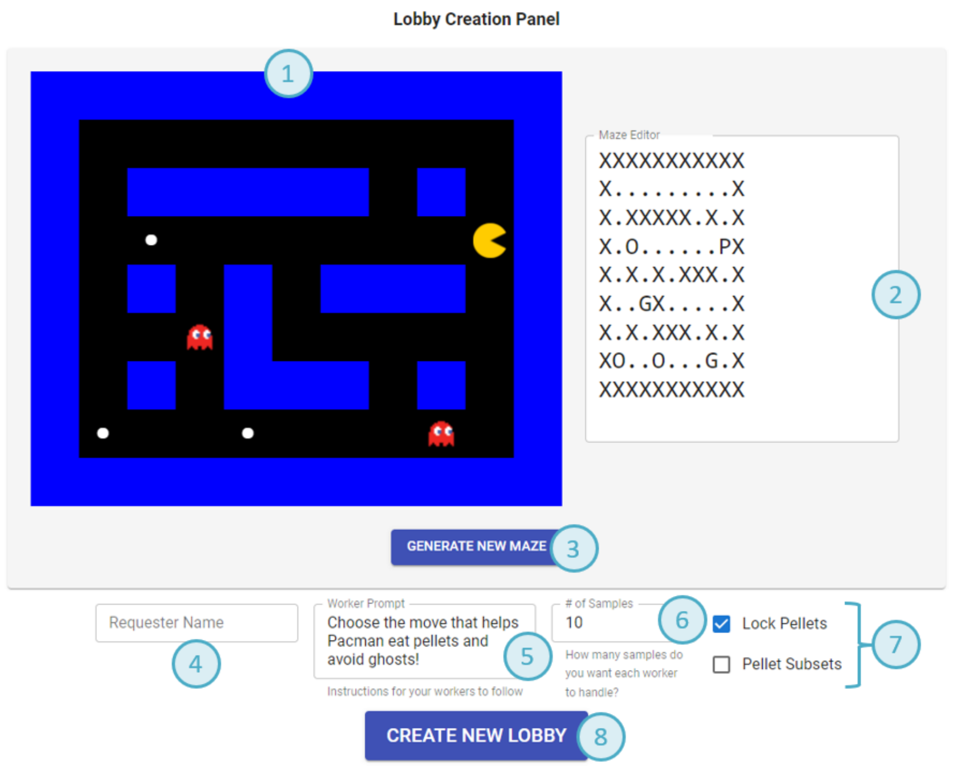


Figure 3: Outline of Pacman Trainer steps.

PT Requester Interface



Maze Editor

```
XXXXXXXXXXX
X.........X
X.XXXXX.X.X
X.O......PX
X.X.X.XXX.X
X..GX.....X
X.X.XXX.X.X
XO..O...G.X
XXXXXXXXXXX
```

Requester Name

Worker Prompt: Choose the move that helps Pacman eat pellets and avoid ghosts!
Instructions for your workers to follow

# of Samples: 10
How many samples do you want each worker to handle?

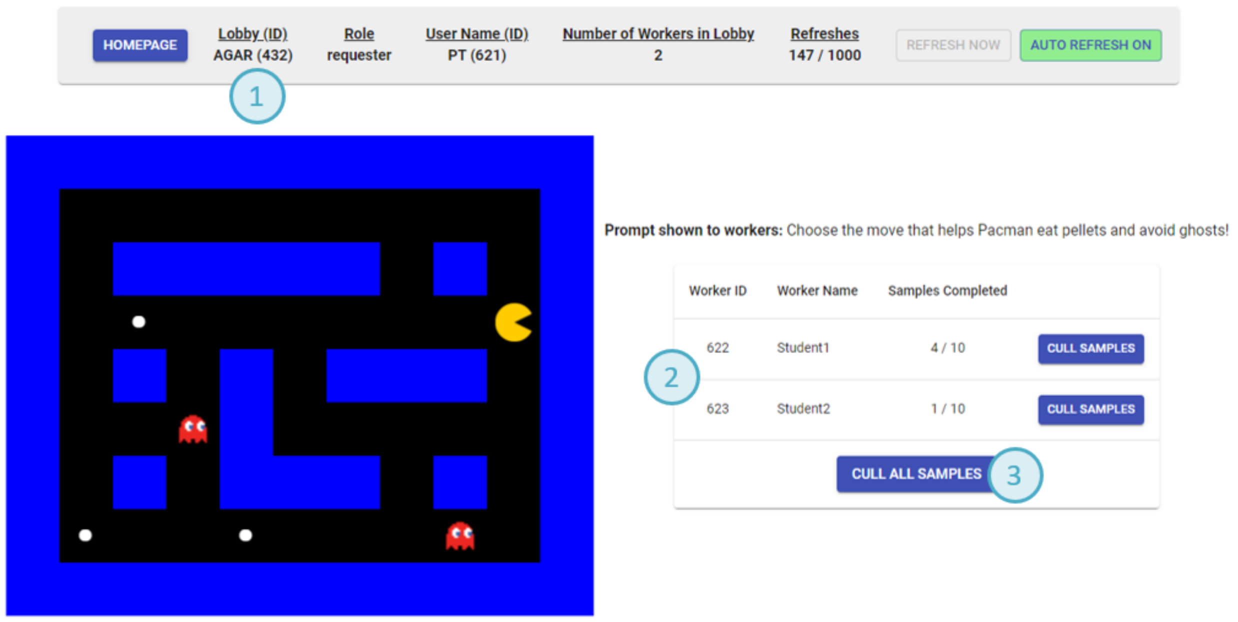
☑ Lock Pellets
☐ Pellet Subsets

Figure 4: Pacman Trainer Requester interface with enumerated components.

[2]https://github.com/masaok/pacman-trainer-web

Lobby Creation: The first step in using Pacman Trainer [https://pacmantrain.com/] is for a Requester (the instructor) to create a lobby that students will join to produce labeled data that will be used to train Pacman on the task at-hand. The lobby creation phase is parameterized by a variety of options, detailed as-follows:

1. *Maze Preview:* an example of how student workers will see the maze in the Maze Editor (2).
2. *Maze Editor:* row-by-row specification of the maze's contents (which will change according to certain parameters described later), with the following options: *Walls [X]* defining the maze's bounds and cells in which neither Pacman nor ghosts can move, *Pellets [O]* defining Pacman's chief objective (to eat these), *Ghosts [G]* defining the adversaries that will hunt Pacman, and *Open cells [.]* containing none of the above, but which allow movement.
3. *Random Maze Generator:* generates a new, random maze configuration.
4. *Requester Name:* the name of the lobby owner / instructor.
5. *Worker Prompt:* the prompt that will be shown to students as they label mazes.
6. *Desired Samples Per Student:* students will be considered "done" labeling when they have completed the specified number of samples.
7. *Sampling Options:* presently, only two options that will change the behavior of samples shown to students for labeling (locking pellets in-place and generating subsets of these for when Pacman has eaten some; detailed ahead).
8. *Lobby Launch Button:* Once all of the parameters above are set, launches an active lobby that students may join and begin the labeling task within.



Lobby (ID)	Role	User Name (ID)	Number of Workers in Lobby	Refreshes
AGAR (432)	requester	PT (621)	2	147 / 1000

Prompt shown to workers: Choose the move that helps Pacman eat pellets and avoid ghosts!

Worker ID	Worker Name	Samples Completed
622	Student1	4 / 10
623	Student2	1 / 10

Figure 5: Pacman Trainer Requester lobby monitor with enumerated components.

Lobby Monitoring: Once the lobby has been constructed, the requester is sent to the Lobby Monitoring page depicted in Figure 5. This page can be used to monitor the class' progress in joining the lobby and labeling samples, with the key components enumerated as follows:

1. *Lobby ID:* the identifier associated with this lobby. The four-letter code (in this example, AGAR) is given to the students to be able to join.
2. *Worker Panel:* Table containing all students who have joined the lobby. Students decide their displayed worker name when joining, though their identity is not coupled with their responses in the final dataset. Individual student responses can be culled immediately by pushing the button next to their name. The number of completed samples from each student is listed next to their name, up to the maximum specified in the lobby creation parameters.
3. *Sample Culling:* Once the lobby has reached a desired level of completion, the requester may "Cull All Samples" to obtain the full labeled dataset complete with all student responses.

Training Dataset: The culled dataset consists of two columns:

- X, the input maze state with positions of all relevant entities represented as a string with new-line characters separating each row.
- y, the labeled action assigned by a student participant to the corresponding maze state.

Note one of the initial challenges of the task at hand: the state space of even small mazes combines all of the possible locations that Pacman, pellets, pellet subsets (from Pacman eating them), and ghosts can be found in, requiring thousands of samples through which the traditional deep learning pipeline will train. Despite this challenge, and depending upon the difficulty of the task (discussed in Transportability Tiers section ahead), even small classes can make short work of the labeling process. Anecdotally, in 5 minutes, a class of 30 students generated roughly 6000 labeled data points using Pacman Trainer.

PT Student Interface

After entering the lobby, students will be shown consecutive sample maze states X_i and click the action that best accomplishes the given prompt, as depicted in Figure 6. Each sample maze state is generated according to a simple ruleset that features into the generalizability tiers that follow.

1. For all samples: all walls are kept in their locations specified in the maze editor.
2. At each sample:
    - If the Lock Pellets option was selected during lobby creation, pellets will only appear in the locations specified in the maze editor. Otherwise, the same number of pellets will appear in randomly chosen non-wall locations.
    - If the Pellet Subsets option was selected during lobby creation, some non-empty subset of the number of pellets specified in the maze editor will appear.
    - The locations of any ghosts and Pacman are randomly assigned to remaining open cells.

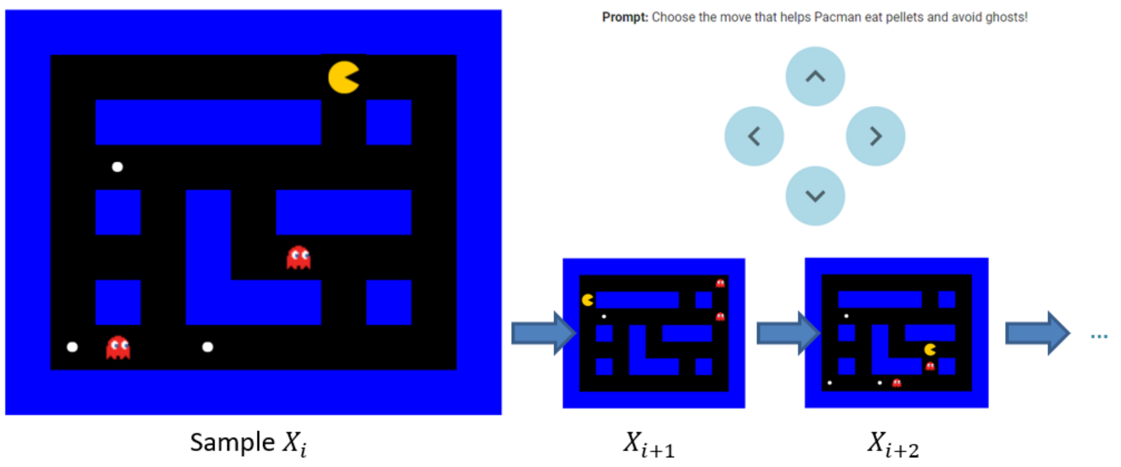

Prompt: Choose the move that helps Pacman eat pellets and avoid ghosts!
Sample X_i
X_{i+1}
X_{i+2}
...

Figure 6: Pacman Trainer Student Labeler interface in which students in a lobby click the move button for Pacman that best accomplishes the prompt for the presently shown maze state, X_i. Upon creating a label, they are then shown the next state X_{i+1} until reaching desired number of samples.

Pacman Agent (PA)

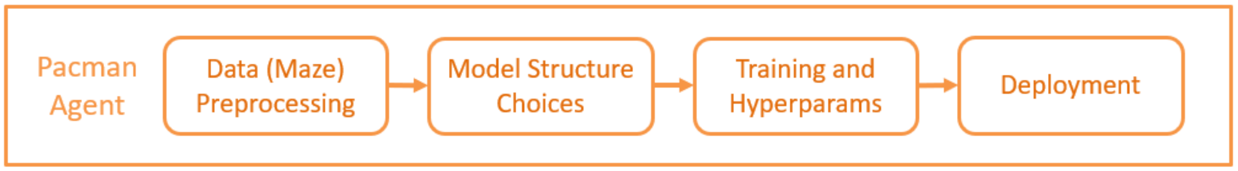


Figure 7: Outline of Pacman Agent steps.

Following successful data collection using the Pacman Trainer app, students are now tasked with training a deep imitation learner given the samples that they just generated. This phase is accomplished in a templated Pacman Agent (PA) Python implementation: The full Pacman Agent activity, including the environment and agent scaffolding, setup instructions, and steps of the exercise can be found at https://github.com/masaok/pacman-agent. As part of this tutorial, students are tasked with familiarizing themselves with each part of the basic Pytorch documentation: https://bit.ly/3BgCkyR. Having read this guide, students are prepared to map each section to the Pacman Trainer problem as outlined in Figure 7 and scaffolded in the `pac_trainer.py` module:

1. *Maze Preprocessing:* Requires students to convert the 2-D mazes of Strings in their training data to a 1-D one-hot encoded representation amenable as an input layer for the network. This is accomplished within the `PacmanMazeDataset` class with methods to vectorize both maze and its associated label/move.

2. *Model Structure Choices:* Students will then choose the structure of the dense neural network, which has "correct" answers for the input layer (which must have dimensionality equal to the product of rows, cols, and possible maze entities) and output layer (one unit for each of the four movement actions), but allows experimentation with the hidden layers. Students are also able to juxtapose different activation function, though the standard ReLU will suffice.

All of the above is packaged in the `PacNet` class.

3. *Training and Hyperparameters:* The choice of learning rate, batch size, number of training epochs, loss function, and optimizer can all be made at this stage of training with little need to deviate from the Pytorch tutorial apart from choosing the Adam optimizer over SGD.
4. *Deployment:* Once trained, the model can be inserted within the `PacmanAgent` class within the `pacman_agent.py` module to choose actions given the current maze state. The model's performance can then be visualized by running the `environment.py` module to watch it make choices in real time.

Readers should consult the `EXERCISE.md` guide in the project repository to see detailed instructions on all of the above. As a final feature of this package, users may optionally use the `maze_gen.py` module, which can generate a faux-training set of the same format as culled from the Pacman Trainer; this may be used if the data collection exercise is infeasible, or to demonstrate the increased data needs for more complex tasks in the Pacman environment (see following section for tiers of difficulty in this environment).

Transportability and Activity Outlines

Although deep learning is an impressive tool for imitation learning, students should likewise be aware of this technique's risks, shortcomings, and data requirements so as to appreciate that it is not a magic-wand that can be waved at a dataset to accomplish any task.

Pacman Tiers of Transportability

Herein, we discuss several "tiers" of transportability in the Pacman environment that can be demonstrated to students in-class or left as exercises for them to explore. By no means is this an exhaustive partition of the tasks possible in this environment, but can scaffold discussions surrounding differences in training and deployment environments and the accompanying differences in required data for more complex tasks.

In any given machine learning task, it is important to explicitly define the *training* and *deployment* environments, which this exercise allows students to easily juxtapose. Successful deployment can be loosely defined as one in which a trained agent successfully controls Pacman in the environment to which it's deployed. As such, within each tier:

- If training and deployment settings are the same, a properly constructed model should intelligently control Pacman during deployment.
- The same should be said for an agent that is trained at a higher tier and deployed to a lower one (i.e., when trained on a complex task and deployed to a simpler).
- However, an agent trained in a lower tier will falter in a higher one (i.e., when trained on a simpler task and deployed to a more complex one).

We depict a suggested set of transportability tiers in Figure 8 with the following properties: (I) *Locked Pellets:* The pellets and walls will always have the same position in training as during deployment, though Pacman can start the game anywhere in the maze. (II) *Ghostly Adversaries:*

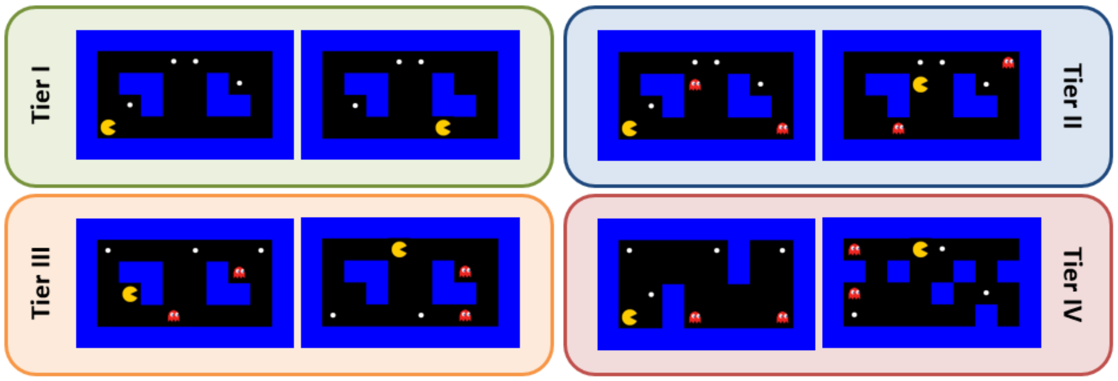


Figure 8: Example samples required to accomplish the tasks described in each transport tier.

Tier 1 plus ghosts that are added to the maze and must be avoided. (III) *Dynamic Pellets:* Tier 2 except pellets can be found in any maze location. (IV) *Shifting Walls:* Tier 3 except walls may be found anywhere in the maze.

Note, for instance, that an agent trained on Tier 1 samples will sufficiently find the pellets in its maze during deployment, but the addition of ghosts in Tier 2 will cause Pacman to walk blithely into certain doom having never encountered them while training. Note also that this is just one possible tier list, e.g., the novel introduction from Tiers 2 and 3 can be swapped, first requiring Pacman to find dynamically placed pellets before accounting for ghosts.

Suggested Activities

Depending on the size of a class, instructors may choose to begin the exercise in Tiers 1 or 2, given the escalating data requirements of higher tiers and class time they wish to allocate. The present work assumes a simple start at Tier 1 and a small maze like those depicted in Figure 8. With in-class data collected through the trainer, students can complete the network training exercise outlined in the Pacman Agent repository.

Having students deploy their Tier X agent on a Tier Y task with $X \neq Y$ and record the results in a lab report. Moreover, students may be tasked with amending the `MazeGen` module to manufacture their own training sets for higher tiers; this lesson can tie in to traditional topics in AI like A* search and also helps students appreciate the data requirements for even modestly higher tiers. Students may also experiment with the parameters of the PacNet, including the number of hidden layers, units per layer, activation functions, and other learning hyperparameters to witness the effects on Pacman's performance, time-to-train, or generalizability. The data collection can serve as a day-one teaser of an AI course, in which imitation learning is demonstrated immediately by training the model in-class via the instructor's solution.

Results

Although this introductory work with Pacman Trainer does not provide empirical results for in-class reception, anecdotal offerings in an undergraduate artificial intelligence class were positive with students enjoying the intuitive walkthrough in the familiar Pacman environment. In a class of roughly 35 students, all but one team successfully completed the activity outlined in the Pacman Agent description, and several explicitly commented on how glad they were to have had the activity for understanding advanced topics in follow-on courses. By way of "solution" to the environment, we do provide an example network, generated training set, and MazeGen implementation that can be amended to create training data for any tier independent of human-data obtained through the trainer. Readers may see the EXERCISE.md readme in the Pacman Agent repository for more information on this solution.

Discussion

While providing many insights into the power and caveats surrounding deep learning, this project also has the benefits of leading to, or transitioning from, adjacent topics in machine learning (ML) and artificial intelligence (AI). We briefly discuss its limits and these "educational adjacencies" that can enrich AI curricula.

Limitations: The primary drawback of Pacman Trainer is that the Pacman environment is one that is best solved by earlier tools in AI, like search for pathfinding and minimax for adversarial problems. Without the context that deep learning deployed in this environment is for illustrative purposes only, and to serve as an intuitive sandbox in which to experiment with different stages of the deep learning pipeline, students may mistake the types of problems for which deep learning is better suited (like machine vision and natural language processing). However, we argue that Pacman Trainer serves as a more engaging and visual introduction to deep learning beyond examining traditional classification accuracy metrics and that students easily grasp this caveat.

Educational Adjacencies: Apart from its own merits, the Pacman Agent exercise can serve as a launchpad between two adjacent topics: (1) *Reinforcement Learning:* once students encounter the data and person-power required to train deep learning systems on even modest problems, this pain point can be used to motivate reinforcement learning applications like deep-Q-networks, which can also be applied to learn optimal policies in the same Pacman environment through online exploration. (2) *Causal Inference and Transportability:* although the "transportability tiers" given in this work rested on intuition, modern courses in AI and ML may choose to integrate topics from causal inference wherein structural formalisms for transportability can determine which results from training to deployment are transportable or not.

Conclusion

By using this educational tool, students gain hands-on experience with the entirety of the deep, supervised, imitation learning pipeline: by labeling moves for Pacman to make in the Pacman Trainer, using these in training a Pacman Agent, and lastly testing generalized deployment, students gain familiarity with the data required for deep learning, how this data is integrated into Pytorch implementations of neural networks, and witness firsthand the power and limits of deep learning as a supervised learning tool. These lessons motivate adjacent topics in modern AI education into reinforcement learning and causal inference, and provide an unintimidating invitation to the more complex topics of deep learning that welcomes the next generation of data scientists.

References

[1] Y. Chauvin and D. E. Rumelhart, *Backpropagation: Theory, architectures, and applications*. Psychology press, 2013.

[2] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[5] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern recognition*, vol. 37, no. 9, pp. 1757–1771, 2004.

[6] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[8] T. W. Smith and S. A. Colby, "Teaching for deep learning," *The clearing house: A journal of educational strategies, issues and ideas*, vol. 80, no. 5, pp. 205–210, 2007.

[9] J. DeNero and D. Klein, "Teaching introductory artificial intelligence with pac-man," in *First AAAI Symposium on Educational Advances in Artificial Intelligence*, 2010.

[10] F. Fallas-Moya, J. Duncan, T. Samuel, and A. Sadovnik, "Measuring the impact of memory replay in training pacman agents using reinforcement learning," in *2021 XLVII Latin American Computing Conference (CLEI)*. IEEE, 2021, pp. 1–8.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[12] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[13] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[14] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," *Advances in neural information processing systems*, pp. 402–408, 2001.

[15] P. Baldi and P. J. Sadowski, "Understanding dropout," *Advances in neural information processing systems*, vol. 26, pp. 2814–2822, 2013.

[16] E. Bareinboim and J. Pearl, "Transportability of causal effects: Completeness results," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, 2012.

[17] Y. Chung, P. J. Haas, E. Upfal, and T. Kraska, "Unknown examples & machine learning model generalization," *arXiv preprint arXiv:1808.08294*, 2018.