Exploring Traditional and Deep Learning Approaches to Online Machine Learning in NBA Game Prediction

Jerry Sun[1] **Peter Wu**[1]

Abstract

Comparisons between offline learning methods and online methods have seldom been studied within the same problem setting, often being unfeasible given the nature of online tasks, usually requiring extreme computational and memory needs, as well as typical machine learning algorithms being unsuitable for streams of data. We confront this issue by using data from the NBA, where new data is released day-by-day, but notably with low data throughput: this makes the problem tractable in both online and offline contexts. In this paper, we evaluate the performance of Online Neural Networks (ONN), a novel online deep learning algorithm utilizing Hedge Backpropagation (Sahoo et al., 2018) in comparison to basic algorithms adapted for use in online settings, the Perceptron and Logistic Regression. We further compare the ONN with several traditional variations of DNNs. These algorithms are evaluated on their ability to predict NBA game outcomes from year 2000 to 2020. We discover some interesting results with our baseline models actually outperforming the ONN by an average of $\approx 6\%$ and on average $\approx 60\%$ faster wall clock times. Similarly, we discover that almost all of our basic DNN architectures outperform the ONN by an average of $\approx 7\%$ and on average $\approx 80\%$ faster wall clock times.

1. Introduction

In machine learning, we often encounter cases—especially in practical applications—where an entire dataset is not fully available. Online machine learning approaches are suitable for tasks that involve incoming streams of data, without any notion of a training or test set. Formally, we seek to find an approximate solution to the optimization problem:

$$min_w \, \mathbf{E}_{\tilde{x} \sim D}[\mathcal{L}(w; \tilde{x})]$$

for some distribution D and loss function $\mathcal{L}$. However, D's true distribution is not known, and is only represented by samples, the number of which can change with time. This makes the problem relevant in situations where we may suffer limitations with (i) availability of data, (ii) memory issues when considering a large-scale dataset, and (iii) scalability. In this paper, we consider a problem that does not significantly suffer from these three limitations in order to enable a comparison of online and offline learning methods with respect to their accuracy performance.

A major possible testbed for online learning is sports, where new data is well-documented. The NBA, in particular, is the most popular basketball league in the world and presents a unique challenge due to the uncertain nature of regular season games. Historically, methods for sports prediction have been based more in statistical simulation and inference, given the stochastic nature of sports data (Min et al., 2008), though recent work has argued for the validity of ML techniques in classification for sports games, such as the SRP-CRISP-DM framework for developing artificial neural networks on sports-related tasks (Bunker & Thabtah, 2019). In the NBA specifically, countless number of factors play into the games' results such as player health, trades, naturally volatile player performances, player improvement, coaching, and so forth. Thus, this problem is a naturally heavily sought after problem with respect to not only gambling odds and team monetary evaluations but also sports science within each team for them to be able to identify areas to improve and expectations for their team. As such, this problem will naturally have a lot of noise and relatively high variance and bias as well, only contributing to its unsolved nature.

1.1. Problem Setting

We ground the investigations of this paper in the task of predicting the outcomes of NBA regular season matches as a binary classification problem based on whether the home team wins, or the away team wins. This problem is specifically amenable to the use of online learning, because of

[1]Cornell University, Department of Computer Science, Ithaca, New York, USA.

the stream-like nature of daily games, resulting in a constantly evolving dataset. Recent data is especially important because rosters are often in flux, and a team's performance can vary wildly from season to season with both player and team management changes. With this objective in mind, we seek to begin training with examples from the year 2000 season up to the present, evaluating the performance of several online models on daily streams of new data for the regular season.

1.2. Data Acquisition and Pre-processing

While several sites host NBA game data, we refer to `basketball-reference.com`, as they have a reliable stream of box score information on their site dating back into the earliest years of the sport. Due to the changes in the game and inconsistencies in the metrics that were kept track of in the past, we only consider the seasons from 2000 to the present in order to represent the "modern" NBA. Overall, there were roughly 25,000 games that were played within the past 20 years, but only around 22,000 of the games were able to be considered after adopting a three game-running average in per-player statistics. Players often have slumps and highs and their performance is often tied to their momentum from previous games; thus, a three game-running average was chosen in order to accurately reflect the most immediate playing ability of the player.

The features that were decided on were the "starting five" players for each team and their respective running 3 game averages in the categories of team record indicator, minutes, field goals made, field goals attempted, three pointers made, three pointers attempted, free throws made, free throws attempted, offensive rebounds, defensive rebounds, assists, steals, blocks, turnovers, and points. These sets of features were selected to maintain simplicity as coaching and other health and injury features were either not featured on the source website or would have created a complex player mapping for the models to learn and train on which would have made the problem extremely complex with high dimensionality, sparse, and limited data.

In pre-processing, the player features were all relatively standardized by accumulating 3 game totals for each feature and then min-max normalizing them. Team records are trickier as the same team could have drastically different number of wins and losses throughout the season simply due to what part of the season they are in; thus, we normalized the records to be within 0 and 1, representing their likelihood of winning. The more wins than losses a team had, the closer the value was to 1 and vice-versa with 0. If a team had the same number of wins and losses then their value would be 0.5 as they have a naive 50% chance of winning. Within the online context, we decided to batch games to evaluate running accuracy averages by day such that there were 3,904 days of basketball to consider and loop through. When all said and done, there were 21,656 games (3,904 game days) and 142 features per game to train each model on. Another notable aspect about the data is that it is unbalanced with 66% of the wins coming from the home team so we expect the models to potentially predict mostly home team wins. A balanced accuracy measurement is thus used.

2. Methods

Part of the line of inquiry is to determine the efficacy of the novel ONN architecture and its ability to perform against other well known online machine learning algorithms, as well as typical DNNs. We provide a conceptual basis of the algorithms for experimentation in the subsections below.

2.1. Perceptron

The Perceptron is an algorithm with a simple geometric intuition, and is given by the following update rule, where w defines a separating hyperplane at epoch time t for label y and corresponding data point x (Rosenblatt, 1958):

$$\overrightarrow{w}_{t+1} = \overrightarrow{w}_t + y_i x_i$$

We implement the Online Perceptron using the Scikit-learn machine learning library (Pedregosa et al., 2018). Scikit-learn has a `Perceptron` class which exposes a `partial_fit` function, intended for training on subsets of a dataset. This allows us to train the model in an online manner and fit a new batch of games for each day, representing that day's daily slate.

Being such a simple algorithm to implement, the Perceptron suffers from two major drawbacks in this problem setting. First, the model assumes that the data is linearly separable, which is unlikely especially when considering the noisiness of the data. Additionally, the update rule considers all misclassifications equally, so it may overreact to recency if the latest online update results in several errors.

2.2. Logistic Regression

Logistic regression is a discriminative algorithm, i.e. it models the probability $P(y_i|x_i)$ of a point given label vector y and data x in question. This expression in logistic regression is equated to

$$P(y_i|x_i) = \frac{1}{1 + e^{-y_i(w^T x_i + b)}}$$

Importantly, we apply stochastic gradient descent (SGD) on the loss function of logistic regression in training, because it is more conducive to online learning: SGD considers a random example or minibatch of the dataset, and as such we can train on streams of incoming data with it. The update

rule for a weight vector w and learning rate α_t using logistic regression is given as follows:

$$w_{t+1} = w_t + \alpha_t \frac{x_i y_i}{1 + e^{w_t^T x_i y_i}}$$

We implement the algorithm using Scikit-learn's `SGDClassifier`, which can operate on log loss as an option. Once again, we use a similar training workflow with a similar `partial_fit` function call on every day of NBA game data.

We use logistic regression as a second baseline because it is an additional linear classifier, but is more perceptive to the degree of misclassification of certain examples while training; this gives the expectation that logistic regression will outperform the Perceptron.

2.3. Online Neural Networks (ONNs)

One of the first online learning algorithms of its kind, we evaluate Sahoo et al.'s online neural network which uses the Hedge Backpropagation algorithm (2018). This method of using deep learning for online cases is markedly different from past attempts: rather than only adapting the optimization algorithms for the network itself, they present a framework allowing networks to learn a depth while training. As such, model selection, which presents a major difficulty when implementing DNNs for online learning because of the uncertainty of incoming data, is no longer a major issue. The limitations of backpropagation as a method for training online deep learning are improved upon by Sahoo et al. through Hedge Backpropagation, which uses a hedging strategy to address fixed depth, the vanishing gradient problem, and diminishing feature reuse (Sahoo et al., 2018). We write the Hedge Backpropagation algorithm for reference.

Algorithm 1 ONN Using Hedge Backpropagation

Input: Discounting parameter $\beta \in (0, 1)$, learning rate η, smoothing parameter s, number of epochs T
Initialize $F(x) =$ DNN with L hidden layers and $L + 1$ classifiers f^l; $\alpha^l = \frac{1}{L+1}$
for $i = 1$ **to** T **do**
  Predict $\hat{y}_t = F_t(x_t) = \sum_{l=0}^{L} \alpha_t^l f_t^l$, where $f^l = softmax(\sigma(W^l h^{l-1})\Theta^l)$ and Θ^l is a parameter to be learned.
  Reveal true value of y_t
  Set $\mathcal{L}_t^l = \mathcal{L}(f_t^l(x_t), y_t)\ \forall l, ..., L$
  Update $\Theta_{t+1}^l \leftarrow \Theta_t^l - \eta\alpha^l \nabla_{\Theta_t^l} \mathcal{L}(f^l, y_t)\ \forall l, ..., L$
  Update $W_{t+1}^l \leftarrow W_t^l - \eta \sum_{j=l}^{L} \alpha^j \nabla_{w^l} \mathcal{L}(f^j, y_t)$ $\forall l, ..., L$
  Update $\alpha_{t+1}^l = max(\alpha_{t+1}^l, \frac{s}{L})\ \forall l, ..., L$
  Normalize $\alpha_{t+1}^l = \frac{\alpha_{t+1}^l}{Z_t}$ where $Z_t = \sum_{l=0}^{L} \alpha_{t+1}^l$
end for

We implement the online neural network using the `onn` implementation of Hedge Backpropagation using PyTorch. This library is open-source and is licensed with the Apache-2.0 license, and exposes a similar API as Scikit-learn's `partial_fit` and `predict` functions. Sahoo et al. adopt the intuition that networks should be trained "shallow to deep", as shallow models converge faster. Since Hedge Backpropagation adapts the depth of the network according to the classifier's performance at a particular depth, and can be viewed as an ensemble method consolidating the results of multiple networks of varying depths, we initialize the classifier to a relatively shallow initial 3 layers and 32 neurons per hidden layer.

2.4. Deep Neural Networks (DNNs)

Lastly, we evaluate our algorithms against traditional DNNs. The DNNs we construct follow the same model architecture as the ONN: 3 hidden layers (all with ReLu activation functions applied to them) and 32 nodes in each hidden layer. We implemented this using PyTorch (Paszke et al., 2019), which allows for a "model as code" implementation strategy. Using PyTorch, we create three linear layers using `torch.nn.Linear` and apply `torch.nn.LeakyReLu`, because the gradients may be relatively sparse. The DNNs constructed differ with respect to their optimizers and are as follows: (i) SGD without momentum, (ii) SGD with momentum, (iii) SGD with Nesterov momentum, and (iv) Adam (Kingma & Ba, 2017), an optimizer currently commonly used in practice for data science implementations. All SGD models were established with a learning rate of 0.1 and weight decay of $5e{-}6$ with momentum of 0.9. For the Adam model, we found that a learning rate of 0.05 and weight decay of $1e{-}5$ performed better than other hyperparameter combinations. As for the loss function, we use binary cross entropy loss due to the binary nature of the problem. The hyperparameters were largely either default values or arbitrarily tested and selected upon without extensive searching, with the rationale that the dataset is constantly changing, and choosing "good" hyperparameters is made a significant engineering challenge that is out of the scope of this paper.

3. Experiments

Two experiments were primarily conducted: one between online machine learning algorithms and one between the ONN and traditional DNNs. With respect to the first experiment, all 3 models (Perceptron, Logistic Regression, and ONN) ran on the entire length of the dataset while computing running accuracy averages every 100 days in order to capture the changes and trends in the game as it evolves. With respect to the second experiment, we had the ONN run in an online fashion while the DNNs ran in a offline

fashion with batch training and compared those. The ONN was still evaluated on a running average basis whereas the DNNs utilized a train validation split with 500 of the last days being the validation set. The DNN models were run for however long it took for them to appear to converge which ended up being 800 epochs.

4. Analysis of Results

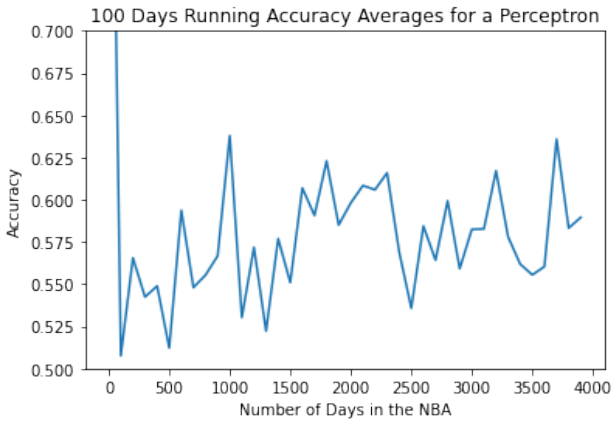

Figure 1. Running Accuracy for Perceptron

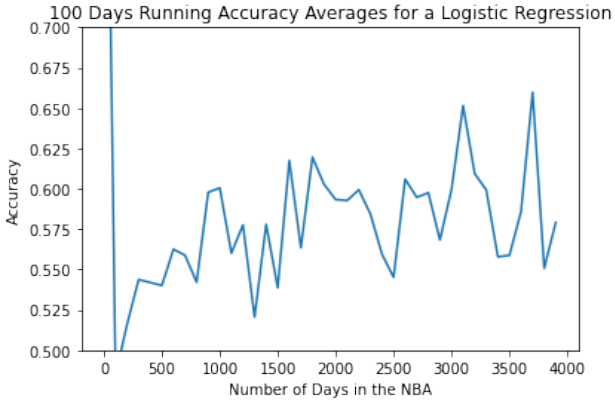

Figure 2. Running Accuracy for Logistic Regression with SGD

4.1. Online Learning Methods

What we can see from the figures above is that all the models perform roughly in the same ballpark which is somewhat expected since the problem is an incredibly complex and noisy one, resulting in both high bias and high variance. We observe that the ONN appears to be fairly less noisy and variant compared to both the Perceptron and Logistic Regression which are both equally noisy. With respect to wall clock time, we can see Logistic Regression ($\approx 16s$) runs the fastest, followed closely by the Perceptron ($\approx 21s$) and finally a much slower ONN ($\approx 47s$). This can primarily be explained by their implementation which is mostly Python for the Perceptron and ONN whereas the Logistic Regression is implemented with Cython and thus should run faster.

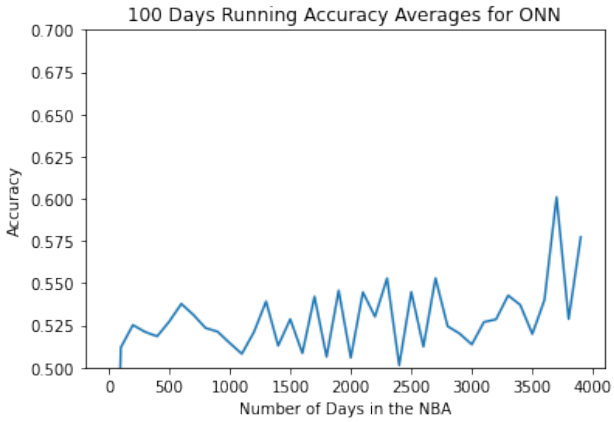

Figure 3. Running Accuracy for ONN

With regards to performance, the Logistic Regression actually performs the best on average throughout the entire learning period with a peak accuracy reaching $\approx 66\%$ while the Perceptron follows closely behind with a peak accuracy of $\approx 64\%$. The ONN surprisingly does fairly worse with a peak of $\approx 60\%$ accuracy, but all 3 models show no sign of converging just yet and are still improving their accuracy, lending the belief that the dataset is not extensive enough to provide an adequate measurement of final converged accuracies if they ever do. While many typical online learning settings involve extremely large-scale datasets, it was a bit jarring to see the ONN perform significantly worse than these linear models, given its tendency to first start out with shallower models.

4.2. Deep Learning Methods

As one can see from the graphs, the DNNs that use the SGD optimizer don't fare well without the help of momentum. It appears as though the model gets stuck inside a local optimum and is not able to exit. The model is stuck predicting all games as having home team wins, for which there is a resulting bias. Both the SGD momentum-based models do quite well, eventually converging at around $\approx 67\%$ accuracy, with the Nesterov momentum based method appearing to exit the local optimum much faster. However, the non-Nesterov based model appears to be less noisy in its convergence. With the Adam optimizer based method, we see an exit from the local optimum much faster than either of the SGD momentum based models with a much faster convergence as well. The Adam optimizer converges in around 300 epochs whereas the SGD momentum based models converged at around 600 epochs which is twice as

Table 1. Wall clock time of training and accuracy of various ML models on the NBA regular season from 2000-2020.

MODEL TYPE	TRAINING WALL CLOCK TIME (S)	ACCURACY (%)
PERCEPTRON	21.7	63.6
LOGISTIC REGRESSION WITH SGD	16.3	65.9
ONLINE NEURAL NETWORKS	47.5	60.1
DNN WITH SGD	9.6	59.3
DNN WITH SGD + MOMENTUM	10.1	67.4
DNN WITH SGD + NESTEROV	10.4	67.8
DNN WITH $Adam$	10.2	67.5

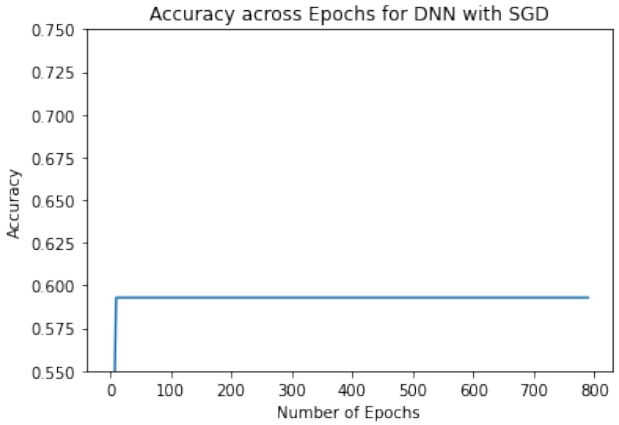

Figure 4. Validation Accuracy for DNN with SGD

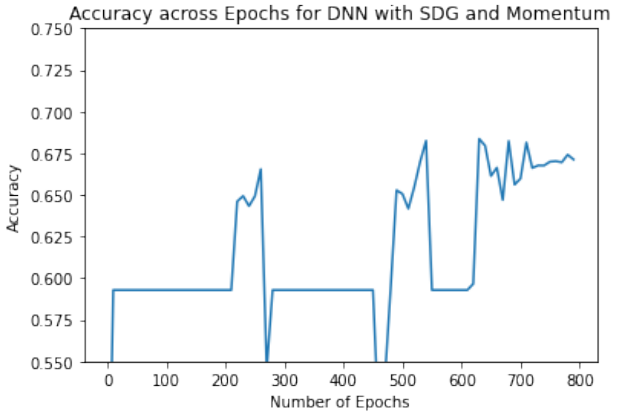

Figure 5. Validation Accuracy for DNN with SGD & Momentum

long. The Adam optimizer however does not achieve any significantly different results with regards to accuracy.

All of the DNNs took $\approx 9s$ to run through all 800 epochs with Adam performing the best due to its much faster convergence while maintaining the same converged accuracy results. Comparing this to ONN, we can see that obviously the DNNs, although performing many more computations than ONN, executes much faster due to its implementation. On top of that, we see the DNNs achieve a much higher accuracy without even having to exhaustively search for hyperparameters, leading to the conclusion that for problems where online learning may be suitable but not necessary, current offline deep learning approaches still trump online deep learning methods with both higher accuracies and lower variances.

5. Conclusion

The results for this particular problem and set of experiments show that the ONN model with Hedge Backpropagation is outperformed by simpler and more naive models such as the Perceptron and Logistic Regression when it comes to the average accuracy performance for tabular data similar to NBA regular seasons from 2000 to 2020. It is, however, a much more stable algorithm in its adjustment of weights, resulting in less variance between each period of evaluation. When treating online learning methods as an alternative to offline learning methods, it appears as though offline deep learning methods still outperform online methods in every regard within the scope of the problem discussed in this paper: wall clock time, accuracy, variance and presence of noise in the performance, and speed to convergence.

However, our research was not devoid of potential error sources. There are some possible flaws that exist within this paper's experimental setup and depth of study. The various DNNs were not extensively experimented with when it comes to the hyperparameters as no standardized search process was established or used. There was also a lack of foresight in the collection of data that could have yielded better results as more data would have given the plain SGD model more time to break out of the local optimum and also would have given the online learning methods more time to try and see a converged upon accuracy. Data could have been scraped and collected from probably even earlier as most of the stats we track were indeed recorded even until the 80s with some smaller adjustments to how some of the data was represented. Past data aside, data could've at least been scraped up until the present day as scraping till the conclusion of a season does not matter.

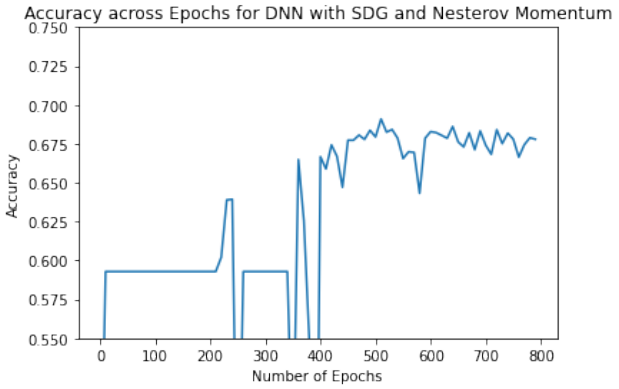

Figure 6. Validation Accuracy for DNN with SGD & Nesterov Momentum

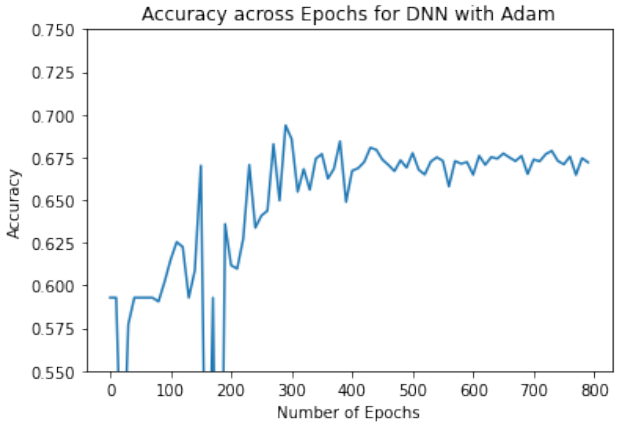

Figure 7. Validation Accuracy for DNN with Adam

Some future work in this area could entail potentially adding additional hyperparameters to the ONN model for it to replicate those of traditional offline learning such as momentum acceleration, weight decay, and learning rate. These hyperparameters arguably made the difference between the DNNs and ONN as without tuning these hyperparameters, the DNNs would have all suffered from being stuck in a local optimum or kept on overshooting the optimum. Thus, being able to fairly compare these methods would allow for a more accurate understanding as to whether the online nature of the problem has any impact on the performance of these deep learning techniques.

References

Bunker, R. P. and Thabtah, F. A machine learning framework for sport result prediction. *Applied Computing and Informatics*, 15(1):27–33, 2019. ISSN 2210-8327. doi: https://doi.org/10.1016/j.aci.2017.09.005. URL https://www.sciencedirect.com/science/article/pii/S2210832717301485.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization, 2017.

Min, B., Kim, J., Choe, C., Eom, H., and (Bob) McKay, R. A compound framework for sports results prediction: A football case study. *Knowledge-Based Systems*, 21(7):551–562, 2008. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2008.03.016. URL https://www.sciencedirect.com/science/article/pii/S0950705108000609.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2018.

Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Sahoo, D., Pham, Q., Lu, J., and Hoi, S. C. H. Online deep learning: Learning deep neural networks on the fly. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 2660–2666. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi: 10.24963/ijcai.2018/369. URL https://doi.org/10.24963/ijcai.2018/369.