Computação Natural - Trabalho Prático 3
Redes Neurais Artificiais

Yuri Diego Santos Niitsuma

November 21, 2018

1 Introdução

O trabalho prático 3 tem compo objetivo em por em prática conceitos relacionados a redes neurais aplicado em uma classificação supervisionada. Iremos modelar uma rede neural capaz de classificar entre uma galáxia, estrela ou quasar com através de uma amostra de dados do Sloan Digital Sky Survey. Pode-se se informar mais do que se trata o SDSS no link [1].

2 Análise da base de dados

A base de dados contém algumas colunas que são redundantes ou não apresentam informação de descrição para a classificação. Por exemplo, os atributos **objid** e **rerun** possuem desvio padrão 0 mostrado na tabela de descrição dos dados.

	objid	ra	dec	u	g	r	i	z
count	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0
mean	1.23765e+18	176.02	14.5	18.61	17.36	16.83	16.57	16.42
std	0.0	47.69	25.05	0.84	0.95	1.08	1.13	1.21
min	1.23765e+18	8.25	-5.37	12.99	12.8	12.43	11.95	11.61
25%	1.23765e+18	159.27	-0.55	18.18	16.8	16.17	15.85	15.6
50%	1.23765e+18	181.01	0.37	18.84	17.5	16.86	16.56	16.4
75%	1.23765e+18	201.22	14.67	19.25	18.0	17.5	17.25	17.13
max	1.23765e+18	260.88	68.54	19.6	19.92	24.8	24.36	22.83

	run	rerun	camcol	field	specobjid	redshift	plate	mjd	fiberid
count	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0
mean	979.33	301.0	3.64	303.24	1.65e+18	0.14	1468.37	52955.51	354.99
std	272.36	0.0	1.66	162.4	2.00e+18	0.39	1780.92	1508.68	207.38
min	308.0	301.0	1.0	11.0	2.99e+17	-0.0	266.0	51578.0	1.0
25%	752.0	301.0	2.0	187.75	3.37e+17	0.0	300.0	51900.0	190.0
50%	756.0	301.0	4.0	300.0	4.96e+17	0.04	441.0	51999.0	352.5
75%	1331.0	301.0	5.0	412.0	2.88e+18	0.09	2559.0	54468.0	511.0
max	1412.0	301.0	6.0	768.0	9.31e+18	5.35	8277.0	57401.0	1000.0

[1] `https://pt.wikipedia.org/wiki/Sloan_Digital_Sky_Survey`

Podemos ver que a quantidade de amostras é desbalanceadas no gráfico de barras na figura 1. Vamos analisar como contornar este problema na seção em que respondemos os itens do guia de implementação.

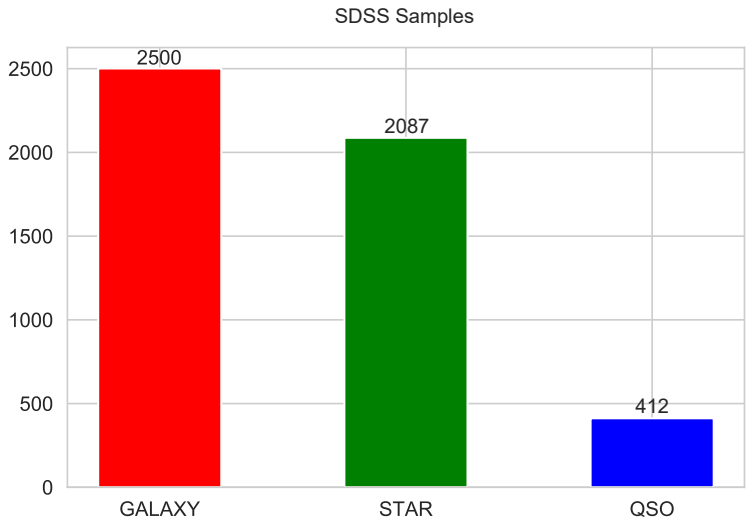

Figure 1: Quantidade das amostras

Suspeitei que apenas os atributos **u**, **u**, **g**, **r**, **i** e **z**, que são bandas de comprimento de onda, fossem capazes de distinguir entre as 3 classes, tendo como objetivo diminuir a quantidade de atributos para diminuir o custo de processamento. Com isso tentei analisar os dados projetando em duas componentes do PCA[1] que é mostrado na figura 2.

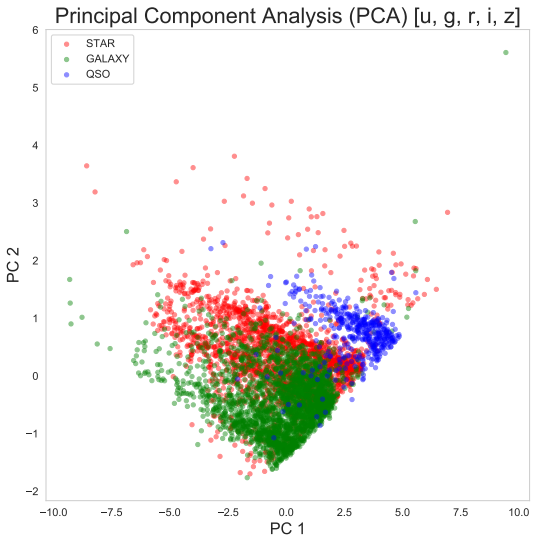

Figure 2: PCA com atributos (u,g,r,i,z)

FAIL!!! Não é possível separar os dados visualmente apenas com estes atributos, o que pode ser reforçado projetando com todos os atributos na figura 3.

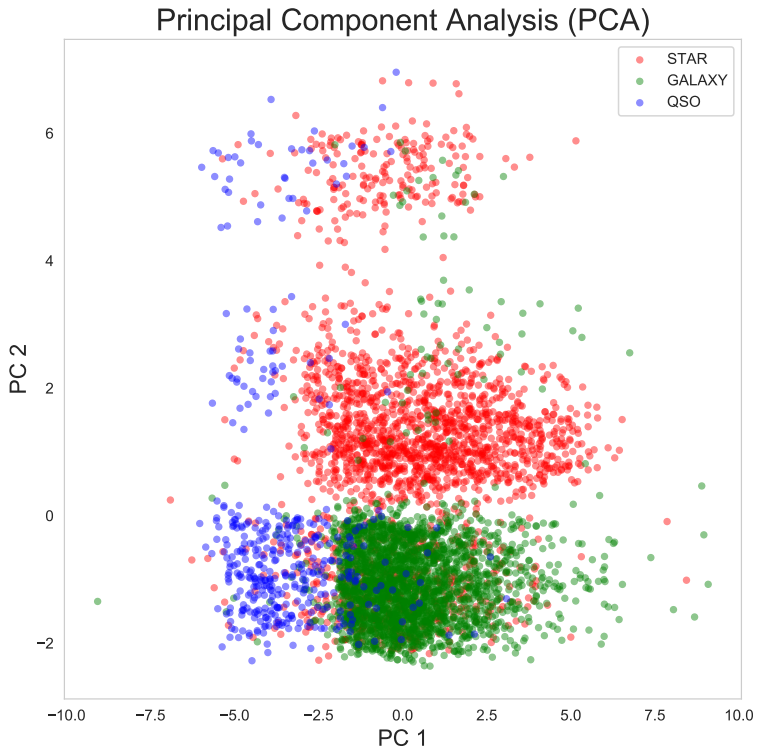

Figure 3: PCA com todos os atributos

Por curiosidade foi analisado também a correlação entre os atributos na figura 4. O que explica a falha da tentativa anterior.

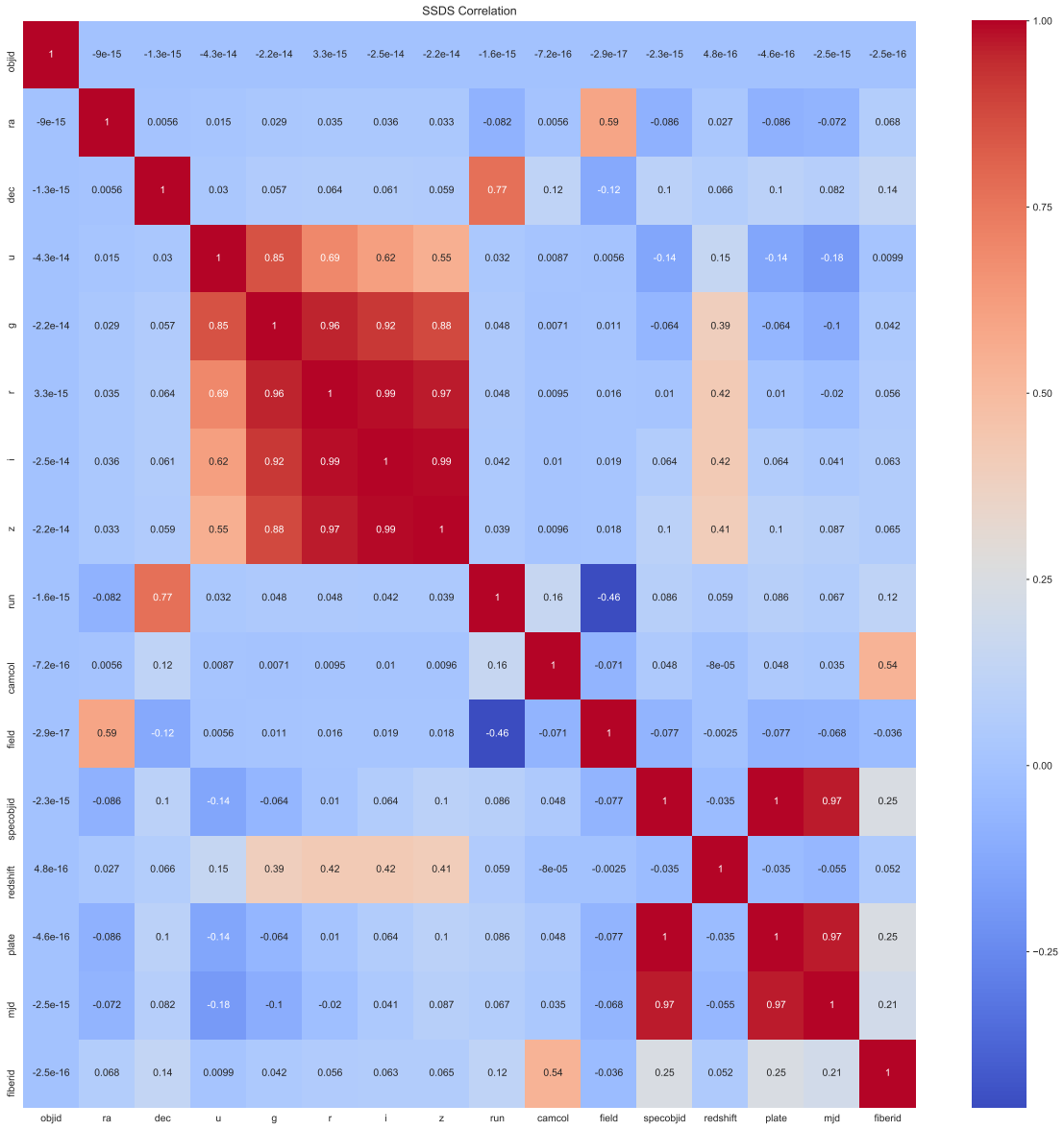

Figure 4: Correlação entre os atributos

2.1 Carregamento dos dados

Os dados são carregados e embaralhados para evitar vícios durante a etapa de experimentos.

Os dados são transformados para uma normal padrão, transladando para a média, e os valores entre 0 e 1. Isso evita valores altos e discrepantes entre as colunas.

Os dados são separados em 3 partições:

- **Train**: Conjuntos dos dados utilizados para o treinamento da rede.
- **Validation**: Conjunto dos dados utilizados para a validação da rede a cada época durante o treinamento.
- **Test**: Conjunto dos dados utilizados para testar a rede (detecção de *overfitting*).

Dado que o *dataset* provido possui 5000 elementos, cada partição ficará com 1666, 1666 e 1668 respectivamente.

Se parâmetro **normalize** for passada como **True**, então os dados da classe **QSO** será replicado 5 vezes para a base de dados.

3 Implementação

A biblioteca escolhida para implementar a rede neural foi o Keras contido no Tensorflow[2].

Toda a implementação está inclusa no arquivo **tp3.py** contendo a leitura dos dados e criação do modelo.

- A versão do **Python 3.6.7**. A versão do Python 3.7.X apresenta problemas no momento com o TensorFlow.
- A versão do **TensorFlow 1.12.0**.
- Utilizei o **scikit-learn** para utilizar duas funções para normalizar os valores dos dados. (**LabelEncoder**, **MinMaxScaler**)
- O arquivo contendo as dependências **requiriments.txt**.

O **main** do **tp3.py** contém um exemplo de execução. Ele executa a tarefa responsável pela função run_job e depois os resultados são gravados em um gráfico em **pdf** na pasta **output**.

O **doc** da função explica detalhadamente em como passar os parâmetros.

```
def run_job(
                dataset='datasets/sdss.csv',
                lr=0.01,
                layers=[(16, 'relu'), (16, 'relu')],
                batch_size=100,
                decay=0.0,
                momentum=0.9,
                epochs=100,
                normalize_qso=False,
                dropout_rate=0.0
    ):
    """Run a entire job.
```

[2] https://keras.io/

```
    Load the dataset
    Split the dataset in 3 partitions (train, validation, test) as describe in docu
    Create the Multi Layer Perceptron model
    Training the model with the params
    Return results

    Keyword Arguments:
        dataset {str} -- path for the dataset file
                                    (default: {'datasets/sdss.csv'})
        lr {float} -- Learning Rate (default: {0.01})
        layers {list of tuples (int, str)} -- units and activation function
                                    (default: {[(16, 'relu'), (16, 'relu')]})
        batch_size {int} -- how many samples for each update the weights
                                    (default: {100})
        decay {float} -- decay rate for learning rate (default: {0.0})
        momentum {float} -- param to help to choose the best direction
                                    for the gradient (default: {0.9})
        epochs {int} -- how many epochs (default: {100})
        normalize_qso {bool} - (Specific) If True then replics 5 times
                                    the data of QSO class
        dropout_rate {float} - rate of how many units will be deactivate
                                    in the training step

    Returns:
        Return the entire history of accuracy of train, validation and
                                    the final test and params information
    """
```

3.1 MLP - Multilayer Perceptron

Definição:

Seja:

- $\boldsymbol{D}$ o conjuntos de dados de entrada para o treinamento.
- h número de camadas escondidas
- n_i número de neurônios em cada i-ésima cadama escondida. ($i \in \{1, \ldots, h\}$)
- f^i a função de ativação na i-ésima camada.
- $\boldsymbol{x_i}$ o vetor de entradas
- $\boldsymbol{z_i}$ o vetor de de bias
- $\boldsymbol{t}$ e $\boldsymbol{W_o}$ matrizes de pesos das camadas internas e saída respectivamente.
- $\boldsymbol{\delta_o} = (\delta_1, \delta_2, \ldots, \delta_p)^T$ o vetor gradiente da rede das saídas dos neurônios e $\boldsymbol{\delta_k} = (\delta_1, \delta_2, \ldots, \delta_m)^T$ vetor gradiente das saídas da camada interna.
- η a taxa de aprendizagem (*Learning Rate*).

A rede escolhida foi a Multilayer Perceptron, com a atualização dos pesos sendo atualizados pelo **backpropagation** utilizando a descida do gradiente descendente estocástico[3] aplicado na função de erro da classificação para atualizar os pesos dos neurônicos.

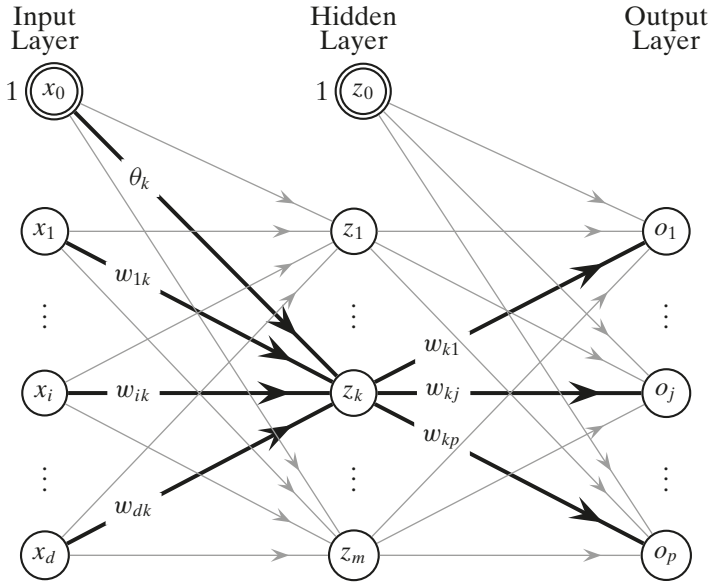

Figure 5: Exemplo de uma rede neural com uma camada escondida

O *backpropagation* é um método usado em redes neurais artificiais para calcular um gradiente que é necessário no cálculo dos pesos a serem usados na rede. *backpropagation* é uma abreviação para "a propagação retrógrada de erros", já que um erro é calculado na saída e distribuído para trás em todas as camadas da rede. [2][3]

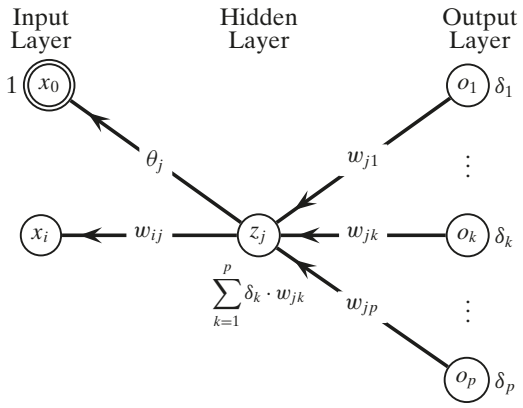

Figure 6: Backpropagation dos gradientes da camada de saída para camadas antecessores

3.2 Funções de ativação

As funções de ativação utilizadas nas camadas escondidas é a **ReLU**, pois foi o que obtive os melhores resultados entre outras funções de ativação (*sigmoid*, *tanh*). Dois benefícios principais

[3] `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD`

adicionais das ReLUs são a dispersão e uma *reduced likelihood* do problema do *vanishing gradient*[4].

$$\text{ReLU(x)} = \begin{cases} x & if\ x \geq 0 \\ 0 & x < 0 \end{cases}$$

A camada de saída é composto por 3 neurônios contendo a função de ativação **Softmax**. Cada neurônio tem como saída a probabilidade do vetor de entrada pertencer a classe que este neurônio representa. *Softmax* é uma generalização da *sigmoid* ou função de ativação logística. Em resumo é uma saída exponencial normalizado com a soma de todas as saídas. Seja c_j a j-ésima classe sendo que temos K classes, temos:

$$P(c_j = 1|\boldsymbol{x}) = o_j = \frac{\exp[net_j]}{\sum_{i=1}^{k} \exp[net_i]}, \forall j = 1, 2, \ldots, K$$

3.3 Função de Erro

A função de erro escolhida $\boldsymbol{\delta}$ é a **Cross-Entropy Error** que é tipicamente utilizada com a função de ativação. *Softmax*.

$$\delta_j^{h+1} = o_j - y_j$$

Em que $y_i = 1$ se para i-ésima classe já rotulada e $y_i = 0$ para o restante. Com este vetor de erro é calculado a derivada da matriz **f** das funções de ativação dos neurônios das camadas internas. Daí se tem a necessidade da função de ativação ser derivável.

3.4 Treinamento

Segue-se um pseudo-código generalizado com h camadas escondidas.

Deep MLP training: stochastic gradient descent.

DEEP-MLP-TRAINING $(\mathbf{D}, h, \eta, \texttt{maxiter}, n_1, n_2, \cdots, n_h, f^1, f^2, \cdots, f^{h+1})$**:**

$n_0 \leftarrow d$ // input layer size

$n_{h+1} \leftarrow p$ // output layer size

// Initialize weight matrices and bias vectors

for $l = 0, 1, 2, \cdots, h$ **do**

  $\boldsymbol{\theta}_l \leftarrow$ random n_{l+1} vector with small values

  $\mathbf{W}_l \leftarrow$ random $n_l \times n_{l+1}$ matrix with small values

$t \leftarrow 0$ // iteration counter

repeat

  foreach $(\mathbf{x}_i, \mathbf{y}_i) \in \mathbf{D}$ *in random order* **do**

    // Feed-Forward Phase

    $\mathbf{z}^0 \leftarrow \mathbf{x}_i$

    for $l = 0, 1, 2, \ldots, h$ **do**

      $\mathbf{z}^{l+1} \leftarrow f^{l+1}\left(\mathbf{W}_l^T \cdot \mathbf{z}^l\right)$

    $\mathbf{o}_i \leftarrow \mathbf{z}^{h+1}$

    // Backpropagation Phase

    $\boldsymbol{\delta}^{h+1} \leftarrow \partial \mathbf{f}^{h+1} \odot \partial \boldsymbol{\mathcal{E}}_{\mathbf{x}}$ // net gradients at output

    // use $\partial \mathbf{F}^{h+1}\, \partial \boldsymbol{\mathcal{E}}_{\mathbf{x}}$ for softmax

    for $l = h, h-1, \cdots, 1$ **do**

      $\boldsymbol{\delta}^l \leftarrow \partial \mathbf{f}^l \odot \left(\mathbf{W}_l \cdot \boldsymbol{\delta}^{l+1}\right)$ // net gradients at layer l

    // Gradient Descent Step

    for $l = 0, 1, \cdots, h$ **do**

      $\nabla_{\mathbf{W}_l} \leftarrow \mathbf{z}^l \cdot \left(\boldsymbol{\delta}^{l+1}\right)^T$ // weight gradient matrix at layer l

      $\nabla_{\boldsymbol{\theta}_l} \leftarrow \boldsymbol{\delta}^{l+1}$ // bias gradient vector at layer l

    for $l = 0, 1, \cdots, h$ **do**

      $\mathbf{W}_l \leftarrow \mathbf{W}_l - \eta \cdot \nabla_{\mathbf{W}_l}$ // update $\mathbf{W}_l$

      $\boldsymbol{\theta}_l \leftarrow \boldsymbol{\theta}_l - \eta \cdot \nabla_{\boldsymbol{\theta}_l}$ // update $\boldsymbol{\theta}_l$

  $t \leftarrow t + 1$

until $t \geq \texttt{maxiter}$

Figure 7: Multilayer Perceptron etapa de treinamento

4 Guia de experimentações

Todas os experimentos utilizam os seguintes parâmetros:

- LearningRate = 0.01
- Hidden Layers = Duas camadas escondidas de 16 neurônios cada com função de ativação **relu**
- DecayingRate = 0.0
- Epochs = 100

Com exceção quando é citado o próprio parâmetro mantendo os restantes constantes.

Na especificação é dado um guia de experimentações e vamos analisar cada questão. Todos os gráficos é o resultado da média de 10 execuções para uma melhor confiabilidade de dados obtidos.

1. O que acontece quando se aumenta o número de neurônios da camada escondida da rede? Isso afeta o número de épocas necessárias para convergência?

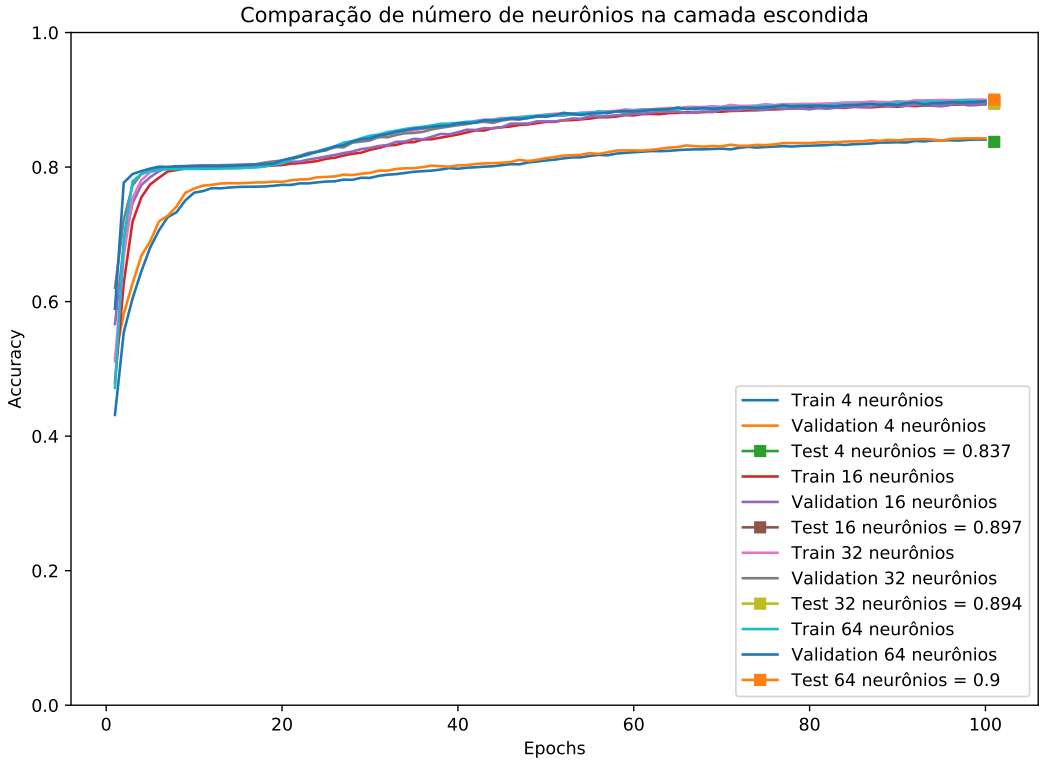

Figure 8: Comparação de número de neurônios na camada escondida

Vemos na Figura 8 que a quantidade de neurônios impactam, mas após uma certa quantidade o ganho não é muito significativo. Verifica-se também que com uma quantidade maior de neurônios há uma convergência mais rápida exigindo uma quantidade menor de **epochs** para convergir.

2. O que acontece quando se aumenta o número de camadas escondidas? O ganho no erro é grande o suficiente para justificar a adição de uma nova camada?

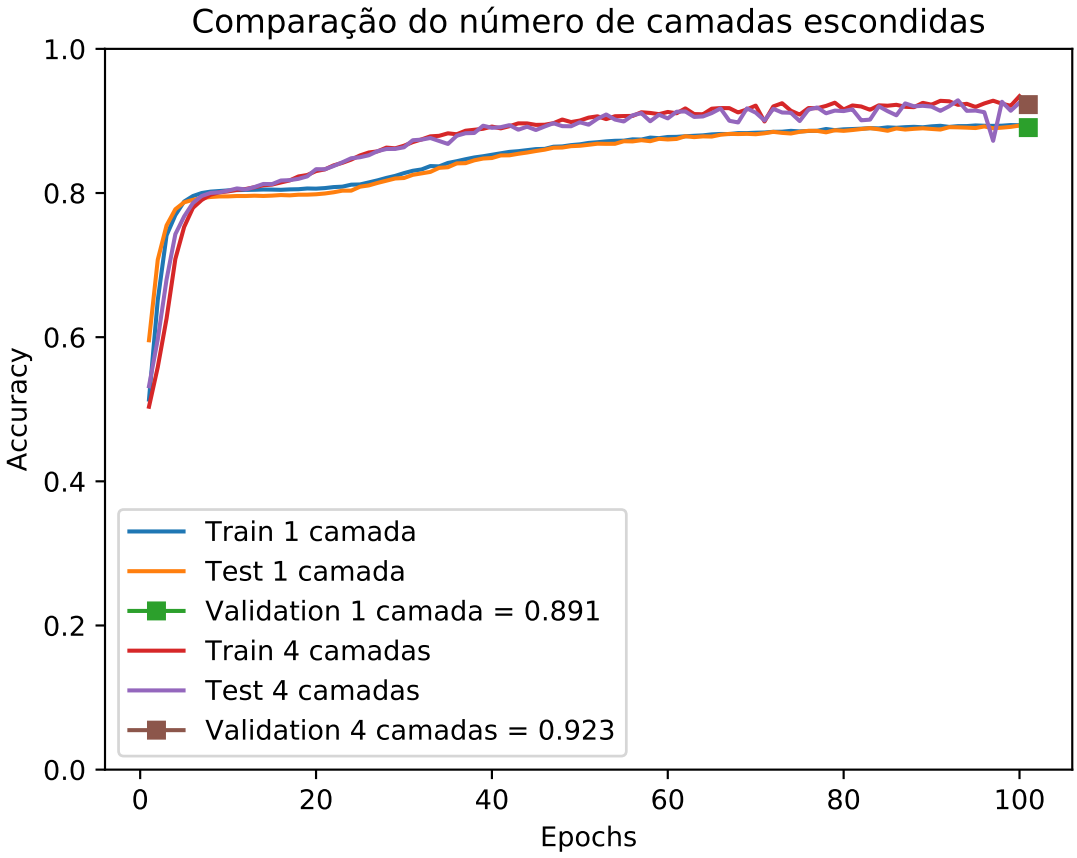

Figure 9: Comparação de número de camadas escondidas

A quantidade de camadas ajuda a combinar os limiares de características, o que pode ser melhor visualizado em um exemplo em 2D interativo na ferramenta Playground Tensorflow[5]. No gráfico da Figura 9 percebemos que uma maior quantidade de camadas da uma maior complexidade pois aumenta o espaço de busca dos pesos dos neurônios. Pode-se ver que o crescimento de acurácia não é tão suavizado quanto a quantidade menor de camadas.

3. Qual o impacto da variação da taxa de aprendizagem na convergência da rede? Oque acontece se esse parâmetro for ajustado automaticamente ao longo das diferentes épocas?

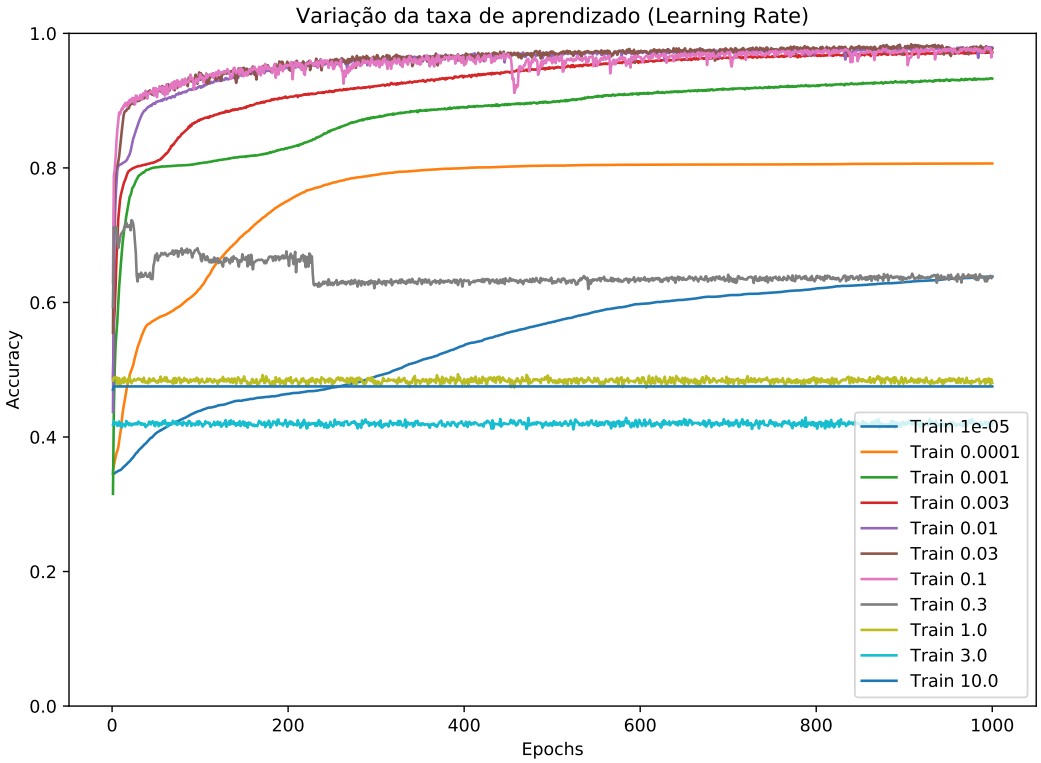

Figure 10: Comparação de diferentes valores do Learning Rate

Na figura 10 destaca vários valores da taxa de aprendizagem através de 1000 *epochs*.

Um valor muito baixo resulta em uma convergência muito lenta o que exige muitas *epochs* para atingir um resultado satisfatório.

Conforme os valores são incrementados, a convergência é melhorada mas em excesso é gerado um ruído que torna ruim o resultado da convergência. Com valor muito alto, a função de erro "passeia" dando saltos muitos altos o que pode nunca atingir o resultado ótimo ou piorar os pesos devido a imprecisão.

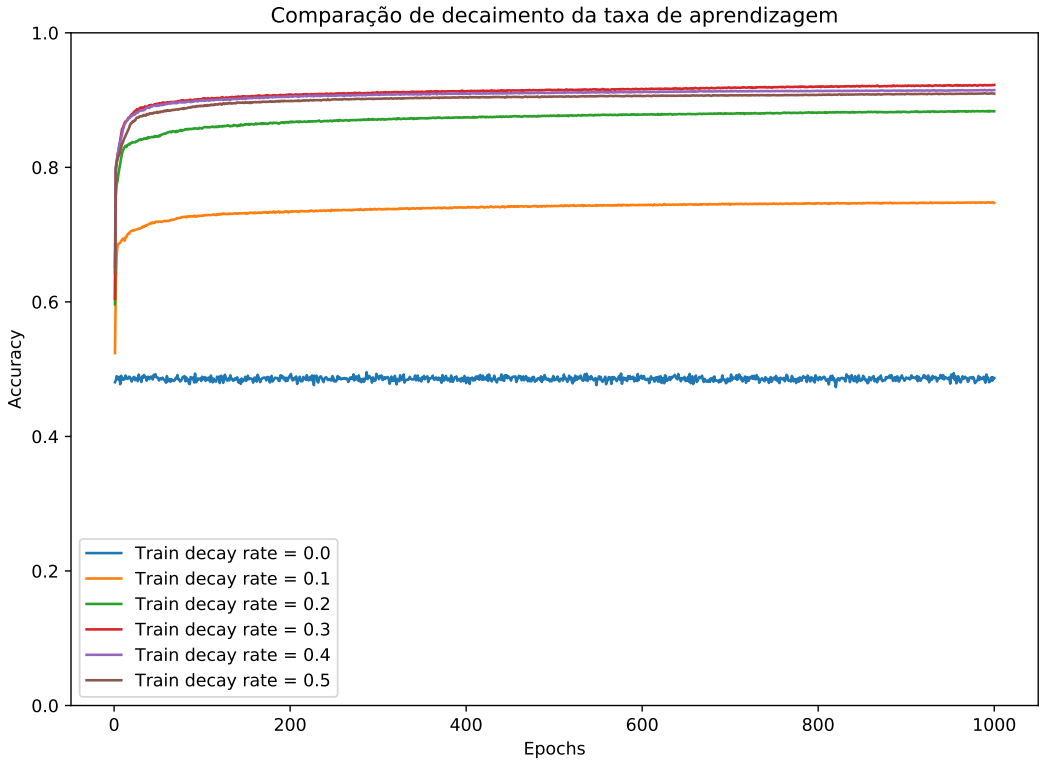

Figure 11: Comparação de decaimento da taxa de aprendizagem

Na Figura 11, a taxa de aprendizagem inicial é 1.0, um valor alto que se pode ver o efeito colateral quando $decay = 0.0$. E vemos que a o decaimento da taxa de aprendizagem ocasiona numa convergência forçada após algumas *epochs*.

4. Compare o treinamento da rede com *gradient descent estocástico* com o *mini-batch*. A diferença em erro de treinamento versus tempo computacional indica que qual deles deve ser utilizado?

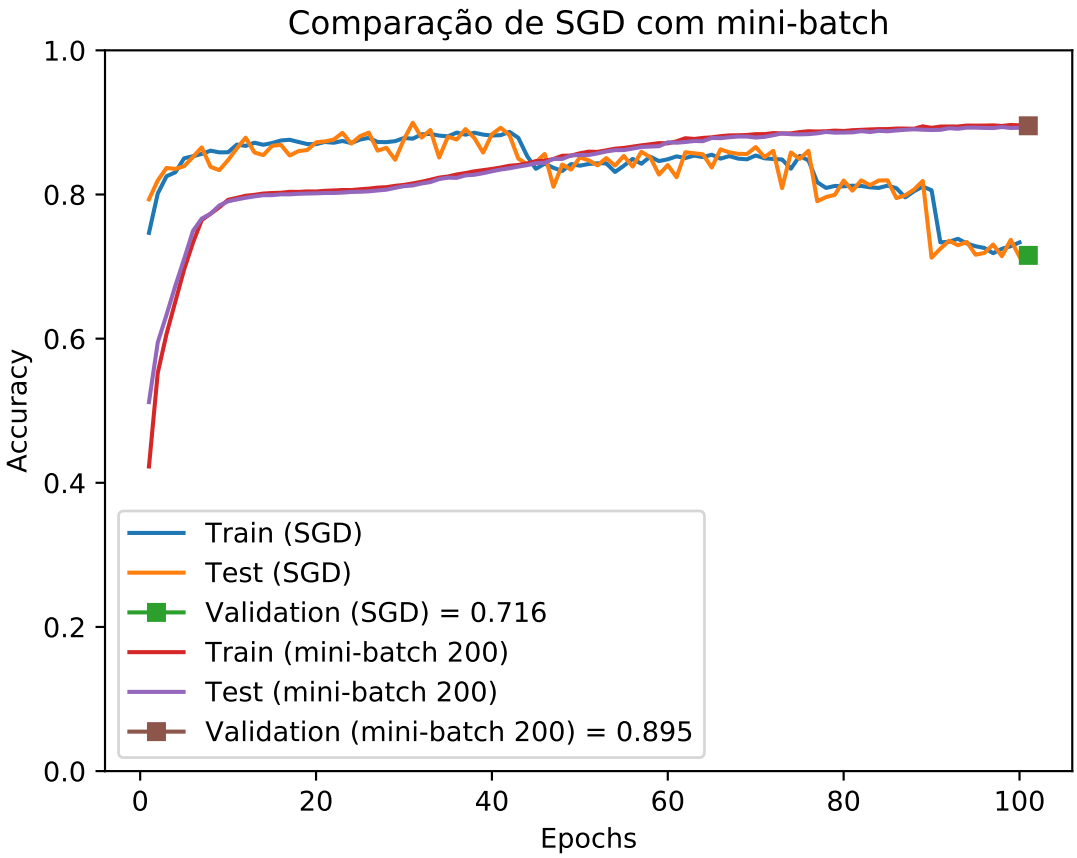

Figure 12: Comparação de SGD com mini-batch

A Figura 12 mostra o comportamento da acurácia quando com o SGD e mini-batch. Observa-se que com o SDG padrão há um decaimento da acurácia porque a atualização dos pesos é feita a cada avaliação de um elemento, enquanto o mini-batch o crescimento é suavizado pois a atualização é feita após 200 avaliações se tornando o passo do gradiente mais generalizado.

5. Qual a diferença do erro encontrado pela rede no conjunto de treinamento ou validação em relação ao erro encontrado no teste? Existe *overfitting*? Como ele pode ser evitado?

   Como os dados são embaralhados na leitura do *dataset*, é possível que a maior os dados da classe *QSO* não estejam na partição de treinamento, ocasionando um erro na avaliação da classe de que não foi treinada.

   Se o *overfitting* for causado pela rede, uma possível alternativa é a **Dropout Regularization**[6], em que durante o processo de treinamento, uma certa quantidade de neurônios são selecionados para serem desativados. Assim a rede fica mais robusta e evita o *overfitting* ao mesmo tempo.

   A função está disponível na biblioteca padrão do Keras[4].

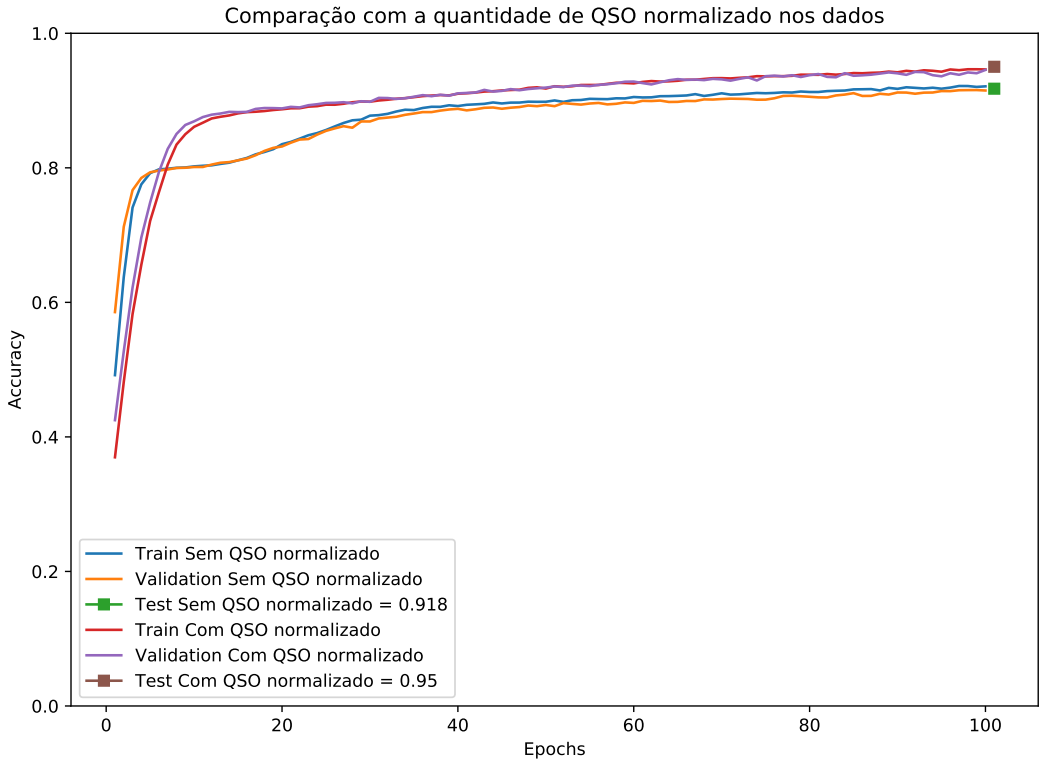

Figure 13: Comparação utilizando Dropout Regularization

Analisando este recurso na Figura 13, vemos que a acurácia da validação e do teste teve um bom ganho em que muitos casos tem a curva acima do valor de treinamento.

[4] `https://keras.io/layers/core/#dropout`

6. A base com que você trabalhou é um pouco desbalanceada. Você pode tentar contornar esse problema usando a técnica de oversampling, ou seja, fazendo cópias dos exemplos das classes minoritárias para balancear melhor a base. Por exemplo, a classe QSO tem 412 exemplos, e a classe GALAXY 2501. De forma simples, você poderia fazer 6 cópias de cada exemplo da classe QSO, aumentando o número de exemplos dessa classe para 2472, e utilizando todos eles no treinamento da rede. Fazendo um oversampling das classes minoritárias e retreinando a rede com os melhore parâmetros encontrados, o erro diminuiu? Por quê?

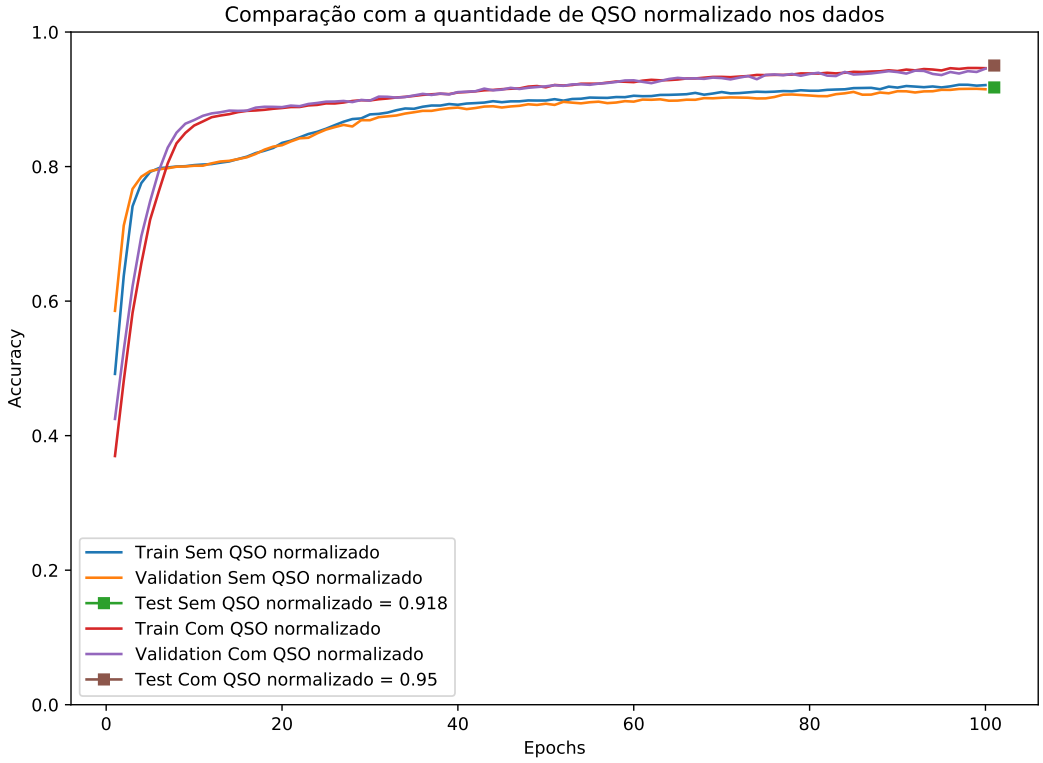

Figure 14: Comparação com a quantidade de QSO normalizado nos dados

Pela Figura 14 verificamos que a cruva de acurácia é normalizada após a aplicação do *oversampling*. O *oversampling* utilizado foi o sugerido na especificação de replicar 5 vezes a quantidade dos dados da classe **QSO**.

O erro diminui pois na etapa de treinamento, as classes **QSO** ganham um maior peso devido pela quantidade maior que foi avaliada.

5 Conclusão

O trabalho prático ajudou em aprender conteúdos profundos e novos que ainda era desconhecido para mim, como o *Dropout Regularization*, que vem de paper recente de 2014 e foi incluído na biblioteca Keras após comprovada a sua eficácia. Também foi importante em verificar alguns das variações de parâmetros, o seu papel e seu comportamento na modelagem.

Por fim, o trabalho tem a sua importância em aprender os conceitos principais e por em prática em uma das principais bibliotecas do mercado.

References

[1] A. Mackiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers and Geosciences*, vol. 19, pp. 303–342, 1993.

[2] S. Sathyanarayana, "A gentle introduction to backpropagation," *Numeric Insight, Inc Whitepaper*, p. 16, 07 2014.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[4] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013.

[5] D. Smilkov and S. Carter, "A neural network playground," http://playground.tensorflow.org/.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.