



Redes Neuronais

Classificação de expressões faciais

Inteligência Artificial

2016/2017

Mestrado Integrado em Engenharia Informática e
Computação

Maria João Mira Paulo - up201403820@fe.up.pt
Nuno Miguel Mendes Ramos - up201405498@fe.up.pt
Pedro Duarte da Costa - up201403291@fe.up.pt

21 de Maio de 2017

Conteúdo

1	Objetivo	3
2	Especificação	3
2.1	Análise do Trabalho	3
2.2	Ilustração de cenários	4
2.3	Explicação de Dataset	4
2.4	Modelo de Aprendizagem a aplicar	5
2.4.1	Algoritmo de <i>Backpropagation</i>	6
2.4.2	Arquitetura das redes neuronais	6
2.4.3	Métodos de treino de uma rede neuronal	7
2.4.4	Funções de ativação aplicáveis às <i>Hidden Layers</i>	7
2.4.5	Funções de perda aplicáveis à <i>Output Layer</i>	7
3	Desenvolvimento	8
3.1	Ferramentas utilizadas	8
3.2	Pré-Processamento do Dataset	8
3.3	Estrutura da aplicação	11
3.3.1	Carregamento de dados	11
3.3.2	Configuração da rede	12
3.3.3	Treino e teste da rede	13
3.4	Detalhes relevantes da implementação	14
4	Experiências	15
4.1	Resultados de várias configurações	16
4.1.1	Experiência 1: Caso Base	16
4.1.2	Experiência 2: Aumento do número de medidas fornecidas	18
4.1.3	Experiência 3: Redução do número de expressões neutras	20
4.1.4	Experiência 4: Construção de uma rede neuronal por expressão	22
4.1.5	Experiência 5: Adição da medida z aos dados fornecidos	23
5	Conclusões	24
6	Melhoramentos	25
7	Recursos	26
7.1	Bibliografia	26
7.2	Software	26
7.3	Percentagem do trabalho efetivo de cada elemento do grupo	26
8	Apêndice	27
8.1	Manual de utilizador	27
8.2	Resultados de teste das experiências	27

1 Objetivo

Este trabalho consiste na criação de uma Rede Neuronal capaz de detetar diferentes expressões faciais.

A Linguagem Gestual é uma língua de sinais, uma língua que utiliza gestos, sinais e expressões faciais e corporais, ao invés de sons na comunicação. As expressões faciais e corporais são o que permite a existência de uma expressão gramatical, caso contrário seria impossível detetar as diferenças entre, por exemplo, uma frase afirmativa e uma frase interrogativa.

Existe um limite de expressões faciais que podem ser demonstradas pelos seres humanos e, por isso, existem técnicas de reconhecimento de *Grammatical Facial Expressions*. Estas técnicas tornam-se complexas na medida em que uma expressão facial pode variar de pessoa para pessoa e que, muitas vezes a co-ocorrência de *Grammatical Facial Expressions* (GFE) pode causar conflitos de interpretação.

Machine Learning é o que permite a uma máquina reconhecer padrões, é o que dá aos computadores a habilidade de aprender sem serem explicitamente programados.

Desta forma, foi criada, treinada e testada uma rede de modo a ser possível o reconhecimento e classificação expressões realizadas por outras pessoas.

2 Especificação

2.1 Análise do Trabalho

As expressões faciais na comunicação em Libras (Língua Brasileira de Sinais) são um fator importante e muitas vezes decisivo no reconhecimento de expressões gramaticais. As expressões faciais são um componente fundamental da comunicação pois proporcionam informações sobre o caráter, emoções e reações das pessoas.

Existem nove tipos de identificadores gramaticais capazes de transformar uma simples frase numa expressão gramatical:

- Questões "WH", por exemplo *Quem, Onde, Como ou Porquê*
- Questões fechadas, de resposta *Sim ou Não*
- Questões de dúvida
- Tópico
- Negação
- Afirmação
- Cláusula Condicional
- Realce
- Cláusula Relativa

2.2 Ilustração de cenários

A linguagem é um instrumento de pensamento usado para exprimir conceitos e transmitir ideias e valores. A expressão corporal e facial desempenha um papel de grande importância no contexto da comunicação, servindo como meio de reforçar uma ideia, chegando até muitas vezes a ser confundida com o próprio argumento. Assim, as expressões faciais permitem uma melhor comunicação com o outro.

Desta forma, a Língua de Sinais permitiu a integração e abertura social do surdo à sociedade, com uma extrema importância no desenvolvimento de uma identidade social.

2.3 Explicação de Dataset

A base de dados é composta por um conjunto de 18 vídeos, conseguidos através do uso do *Microsoft Kinect sensor*, um sensor de movimentos. Um utilizador expressa uma determinada *Grammatical Facial Expression*, 5 vezes em cada vídeo, e são extraídos ficheiros de texto com informação relativa a cada um desses vídeos. Estes são etiquetados por outra, um tradutor humano fluente em Libras, por forma obter uma maior imparcialidade. Ao todo existem 27965 instâncias capturadas através desses vídeos.

A base de dados usada é então constituída por 36 ficheiros, 18 *Datapoint Files* e 18 *Target Files*. A cada *Datapoint File* corresponde um *Target File*, sendo que o nome dos ficheiros contém a letra A ou a letra B, simbolizando um utilizador A e um utilizador B respetivamente.

Desta forma, um *Datapoint File* é constituído por *frames*, cada uma contendo um *FrameId* e 100 pontos, cada um representado pela sua coordenada x e coordenada y, em pixéis e pela coordenada z, em milímetros.

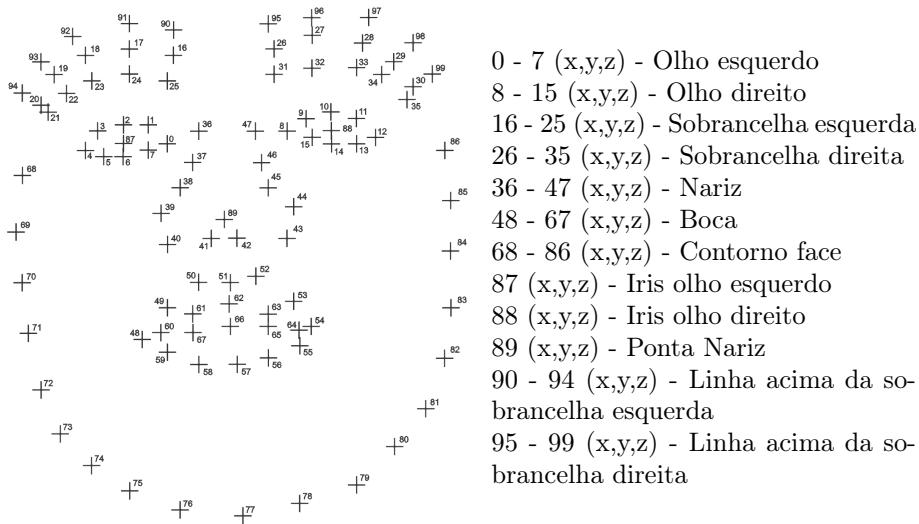


Figura 1: Imagem representativa dos pontos extraídos

Esta base de dados baseia-se no pressuposto de que uma GFE é composta por uma GFE específica e por Expressões Neutras.

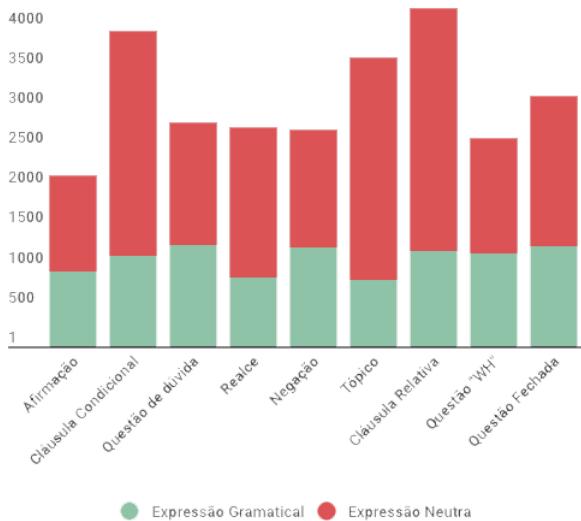


Figura 2: Instâncias por expressão facial

Assim, a cada instância do *Datapoint File*, que caracteriza uma CGF específica, é atribuído no *Target Files* o valor **1** caso corresponda a uma expressão característica da CGF ou **0** caso seja uma Expressão Neutra.

2.4 Modelo de Aprendizagem a aplicar

Neste trabalho foi utilizada uma rede neuronal *Multilayer Perceptron* (MLP). Este modelo utiliza um tipo de aprendizagem supervisionada indutiva com recurso a *backpropagation*.

O modelo MLP é uma rede neuronal artificial do tipo *feedforward*, ou seja, as conexões entre os neurónios não formam ciclos. A informação move num único sentido. Este modelo consiste em múltiplas camadas de neurónios, num grafo direcionado, todo ele conectado. Cada neurónio numa dada camada, tem conexões diretas a neurónios da camada seguinte.

Aprendizagem supervisionada, é uma tarefa de *machine learning* que infere uma função a partir de um conjunto de dados de treino categorizados. Aplica-se neste projeto, pois recebe um conjunto de dados de entrada, as expressões faciais, e o correspondente valor de output desejado para cada entrada, tipo de expressão. Este tipo de aprendizagem indutiva permite, generalizar os dados de entrada e obter uma função que permite inferir, com alguma aproximação, o tipo de expressão facial através de novas expressões.

2.4.1 Algoritmo de *Backpropagation*

A rede neural configurada aplica o algoritmo *Back-propagation*. *Back-propagation* é um método de treino a duas fases, propagação e depois atualização dos pesos das arestas por retrocesso.

Numa primeira fase o vetor de entrada é propagado desde o inicio da rede, camada a camada, até ao chegar à ultima, à camada de saída. O resultado desta propagação é comparado com o valor desejado, usando uma função de perda, sendo calculado um valor de erro para cada um dos neurónios da camada de saída. O valor de erro é propagado por retrocesso, a partir da camada de saída, até todos os neurónios terem um valor de erro associado que representa de forma grosseira a sua contribuição em relação aos dados de entrada originais. Através destes valores de erro os pesos das arestas da rede são atualizados por forma a minimizar a função de perda. Numa segunda fase e após receber dados de entrada suficientes a rede converge para um estado em que os erros são cada vez menores, dizendo-se que "aprendeu"um certo padrão.

Numa tentativa de ajustar corretamente os pesos das arestas, este método é usado em conjunto com o método de otimização *gradient descent*, sendo usada a derivada da função de perda, minimizando a mesma e apontando a direção da função de erro para "baixo"(*downhill*). Esta otimização recorre a uma variável *learning rate*, que corresponde ao quanto "descemos a colina" a cada iteração.

2.4.2 Arquitetura das redes neuronais

Os neurónios são organizados em redes de neurónios, conectados num grafo acíclico. Em redes *feedforward* o *output* de uma camada serve de *input* para a camada seguinte.

Desta forma, as redes neuronais são sempre constituídas por:

1. Uma **Input Layer**, primeira camada da rede neuronal, constituída por neurónios de input e responsável por receber os dados, apresentar os padrões de reconhecimento e declarar o tamanho da dataset;
2. Uma ou mais **Hidden Layers** responsáveis por grande parte do processamento, mais concretamente pela extração de características;
3. Uma **Output Layer** constituída por neurónios de output e responsável por apresentar resultados. O número de neurónios de output depende dos diferentes valores desejados como resultado. Além disso é uma camada que não contém nenhuma função de ativação;

2.4.3 Métodos de treino de uma rede neuronal

Existem diferentes métodos de treino aplicáveis a uma rede neuronal:

- **Stochastic Gradient Descent (SGD):** Este método é um gradiente incremental, é uma aproximação estocástica do método de otimização por gradiente descendente (método usado para encontrar mínimos), ou seja, este método tenta encontrar mínimos ou máximos de uma função objetivo.
- **Adagrad:** Este método baseia-se em gradientes de otimização. A *learning rate*, ao ser ajustada e adaptada conforme os parâmetros escolhidos, proporciona a realização de grandes atualizações por parâmetros pouco habituais e pequenas atualizações por parâmetros mais frequentes. Uma das grandes vantagens deste método é que elimina a necessidade de modificar a *learning rate*.
- **Adadelta:** Este método é uma extensão do Adagrad e pretende reduzir a agressividade gerada nessa técnica quando é ajustada a *learning rate*.

2.4.4 Funções de ativação aplicáveis às *Hidden Layers*

Além disso foram estudadas diferentes funções de ativação aplicáveis à ***Hidden Layer***, entre elas:

- **Rectified Linear Units (ReLU)** $f(x) = \ln(1 + e^x),$
- **Sigmoid** $S(x) = \frac{1}{1 + e^{-x}}.$
- **Tanh** $\tanh(x) = \sinh(x)/\cosh(x) = (e^x - e^{-x})/(e^x + e^{-x})$
- **Maxout** $h_i(x) = \max_{j \in [1, k]} z_{ij}$

2.4.5 Funções de perda aplicáveis à *Output Layer*

A ***Output Layer*** podem ser aplicadas diferentes funções de perda, entre elas:

- **Softmax:** A função de custo Softmax é usada em Redes Neurais de classificação uma vez que retorna uma probabilidade associada a cada classe num problema com múltiplas classes associadas.
- **Support Vector Machines (SVM):** Retorna, para cada dado de entrada, uma classe. É um classificador binário não probabilístico e, por esta razão, funciona bem em domínios onde existem dados com uma clara margem de separação e, por consequente, com pouco ruído. Em situações onde as classes se encontram muito sobrepostas deve-se utilizar apenas evidências independentes.

3 Desenvolvimento

3.1 Ferramentas utilizadas

Para representar a rede neuronal foi escolhida uma biblioteca de *javascript* denominada ***ConvNetJS***. Esta plataforma pode ser executada num navegador *web* e além disso não requer qualquer tipo de *software*, compilador ou instalação.

Para pré-processamento de dados foi programado um *parser* na linguagem ***Python***.

3.2 Pré-Processamento do Dataset

A partir dos pontos extraídos dos vídeos e através do *parser* construído foram pré-processadas um conjunto de medidas que representam os determinados identificadores, *features*, de uma determinada expressão facial.

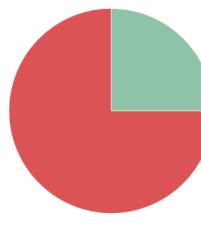
Semantic Functions	Eyebrows	Eyes	Mouth	Head
WH-question	↑			↑
Yes/no question	↑			↓
Doubt question	↓	*	*	⊖
Topic		◊		↓
Negation	↓		∩	↔
Assertion				†
Conditional clause		As in yes/no question		
Focus		As in topic		
Relative clause	↑			

↑ – upward head; ↓ – downward head
† – up and downward head; ↔ – left and rightward head
* – compressed mouth; ◊ – open mouth; ∩ – downward mouth
⊖ – approximation; ⊖ – detachment

Figura 3: Tabela com os identificadores associados a cada expressão

O *parser* lê cada um dos ficheiros de *datapoints* e *targets* do *dataset* e, extrai, para ficheiros de JSON correspondentes, um conjunto de pontos e distâncias por expressão e o seu *target* correspondente. Estes pontos foram previamente escolhidos de forma a representar da melhor forma as *features* anteriormente referidas.

O *parser* é ainda responsável por separar $\frac{3}{4}$ das expressões para uma pasta de expressões de **treino** e os restantes $\frac{1}{4}$ para uma pasta de expressões de **teste**. Foi escolhido o formato JSON por ter uma fácil integração com *Javascript*.



Dados de Teste Dados de Treino

Figura 4: Utilização dos dados

Para a primeira experiência obtivemos as seguintes distâncias e ângulos:

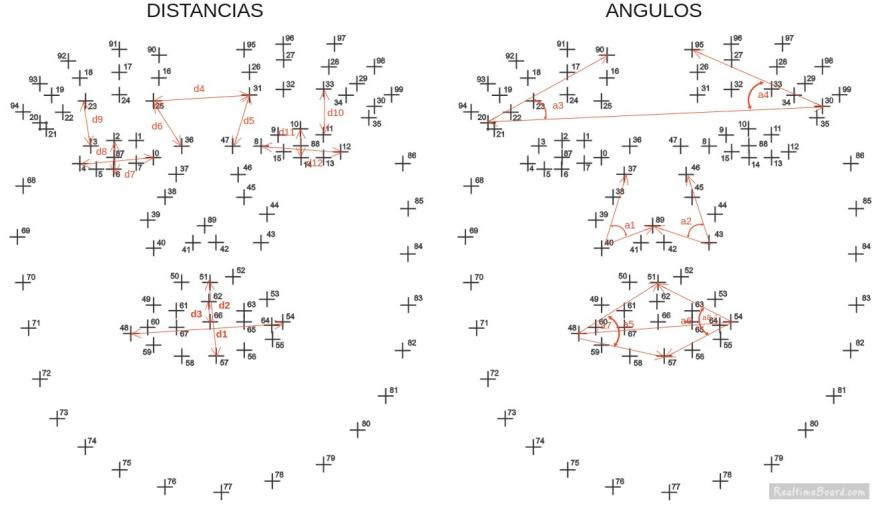


Figura 5: Representação das medidas extraídas para a experiência 1

Para a segunda e terceira experiência obtivemos assim as seguintes distâncias e ângulos:

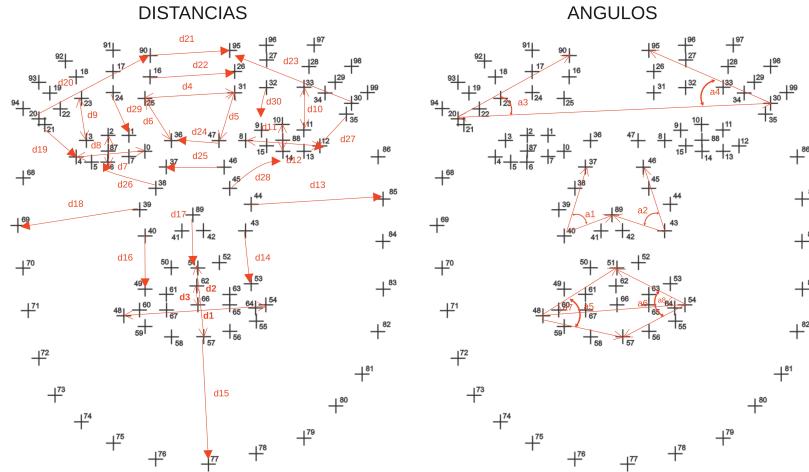


Figura 6: Representação das medidas extraídas para a experiência 2

Para a quarta experiência obtivemos as seguintes distâncias, ângulos e alguns valores de z:

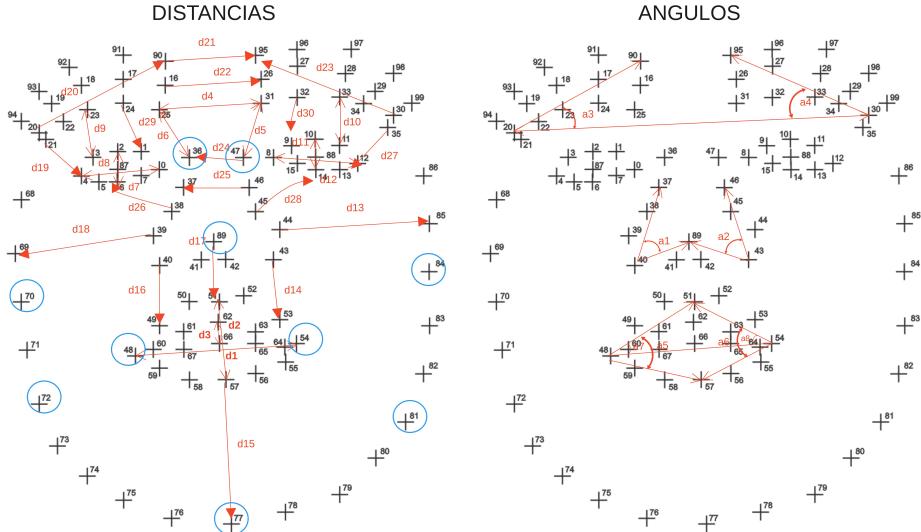


Figura 7: Representação das medidas extraídas para a experiência 3 e 4

Explicação da escolha das várias medidas

Distâncias:

- d1,d2, d3, d14, d15, d16 e d17 permitem verificar se a boca está aberta, fechada ou cerrada;
- d4, d5, d6, d9, d10, d19, d29, d30, d27, d22, d21, d20, d23, d24 e d30 permitem verificar se as sobrancelhas estão cerradas, descontraídas ou levantadas;
- d7,d8,d11,d12 correspondem a olhos abertos ou fechados;
- d18, d13, d26, d28 e d25 permitem perceber a forma do nariz.

Estas distâncias e ângulos são depois colocados num *array* e fornecidos à rede.

Ângulos:

- a1 e a2 permitem verificar se a face está levantada ou baixa;
- a3 e a4 permitem verificar se as sobrancelhas estão cerradas, descontraídas ou levantadas;
- a5 e a6 verificam se a boca está aberta ou fechada;
- a7 e a8 verificam se a boca está curvada negativamente.

Medidas em z:

- Estas medidas permitem verificar o movimento vertical e horizontal do rosto.

3.3 Estrutura da aplicação

A biblioteca usada estrutura a rede nos seguintes objetos:

- **convnetjs.Net()**: Objeto que representa a rede neuronal e recebe as configurações da mesma.
- **convnetjs.Vol()**: Objeto que representa um conjunto de volumes de dados a fornecer à rede.
- **convnetjs.Trainer()**: Objeto que recebe a rede a treinar e as configurações de treino.

Desta forma, o nosso trabalho é composta por três fases distintas:

- Carregamento de dados;
- Configuração da rede;
- Treino e teste da rede;

3.3.1 Carregamento de dados

Sempre que a página é carregada a função apresentada em baixo é chamada. Esta função guarda em memória todos os dados a serem usados, percorrendo linha a linha o objeto JSON e inserindo todos os vetores e ângulos num *array* denominado *train_data* e a respetiva expressão facial, *target*, no *array* *train_labels*. O mesmo processo é aplicado aos dados de teste.

```
1 function prepare_train_data(response) {
2     jsonResponse = JSON.parse(response);
3     var label;
4
5     for (line of jsonResponse) {
6         var lineData = [];
7         for (element in line) {
8
9             if (element == 'target') {
10                 label = line[element];
11             } else if (element != 'index') {
12                 lineData.push(line[element]);
13             }
14         }
15         train_data.push(lineData);
16         train_labels.push(label);
17     }
18 }
```

Listing 1: Código de load dos dados de treino.

Após os dados serem carregados, é chamada a função *shuffleData()*, que baralha aleatoriamente os conjuntos de dados, por forma a que a sua introdução na rede seja aleatória e equilibrada.

3.3.2 Configuração da rede

Através da *webapp* é possível definir as seguintes configurações da rede:

- Número de *hidden layers*;
- Número de neurónios nas *hidden layers*;
- Função de ativação das *hidden layers*;
- Função de classificação na camada de *output*.

Após a receção destes valores a rede é inicializada a partir do código apresentado de seguida.

```
1  var net = new convnetjs.Net();
2
3  /* Input layer of size 1x1x38, 38 input points (30-distances + 8
4   angles)*/
5  layer_defs.push({
6    type: 'input',
7    out_sx: 1,
8    out_sy: 1,
9    out_depth: input_neurons
10   });
11
12  /* Fully connected layers */
13  for (let i = 0; i < number_of_hidden_layers; i++) {
14    layer_defs.push({
15      type: 'fc',
16      num_neurons: number_of_neurons,
17      activation: activation_function
18    });
19  }
20
21  /* Output layer */
22  layer_defs.push({
23    type: activation_function_output,
24    num_classes: output_neurons
25  });
26
27  /* Creates network from given configurations*/
28  net.makeLayers(layer_defs);
```

Listing 2: Exemplo de código de configuração da rede

3.3.3 Treino e teste da rede

A biblioteca ConvNetJS, utiliza o objeto *Trainer*, referido anteriormente, para treinar a rede de acordo com as configurações de treino escolhidas na *webapp*. Podem ser escolhidas as seguintes configurações:

- Método de treino
- *Momentum*
- *Learning rate*
- *L1 decay*
- *Batch size*
- *L2 decay*

Após inicialização da rede, é chamada a seguinte função para inicializar o *Trainer* com as configurações indicadas.

```
1 function initTrainer() {  
2     if (training_method == 'sgd') {  
3         trainer = new convnetjs.Trainer(net, {  
4             method: 'sgd',  
5             learning_rate: learning_rate,  
6             l2_decay: l2_decay,  
7             momentum: momentum,  
8             batch_size: batch_size,  
9             l1_decay: l1_decay  
10        });  
11    } else {  
12        trainer = new convnetjs.Trainer(net, {  
13            method: training_method,  
14            l2_decay: l2_decay,  
15            batch_size: batch_size  
16        });  
17    }  
18}  
19}
```

Listing 3: Exemplo de código de inicialização do objecto de treino da rede

De forma a verificar a eficácia da rede ao longo do tempo, a rede é treinada e testada em simultâneo. Para isto, é chamada, em ciclo infinito, a função `load_and_step()`. Esta função treina a rede usando o *Trainer*, criado previamente através de um volume de dados de treino. Simultaneamente, testa a mesma com uma expressão pertence ao conjunto de dados de teste. De seguida encontra-se parte da função `load_and_step()`.

```

1  /* Volume com os dados de treino*/
2  netx = new convnetjs.Vol(1, 1, 20);
3  /* Atribuição de uma expressão de treino ao volume a partir do
   conjunto de dados de treino */
4  netx.w = train_data[trainIteration];
5
6
7  /* Chamada de treino da rede, onde se fornece o volume de treino
   e o resultado (label) esperado */
8  var stats = trainer.train(netx, train_labels[trainIteration]);
9  avloss = stats.loss;
10 var yhat = net.getPrediction();
11 trainAccWindows[train_labels[testIteration]].add(yhat ===
12   train_labels[trainIteration] ? 1.0 : 0.0);
13 lossWindows[train_labels[trainIteration]].add(avloss);
14 var x = new convnetjs.Vol(1, 1, 20);
15 x.w = test_data[testIteration];
16 var scores = net.forward(x); // pass forward through network
17 var yhat_test = net.getPrediction();
18 testAccWindows[test_labels[testIteration]].add(yhat_test ===
19   test_labels[testIteration] ? 1.0 : 0.0);

```

Listing 4: Código de exemplo do treino e teste de uma rede

3.4 Detalhes relevantes da implementação

Foi criada uma página *web* onde é possível inicializar a rede neuronal, guardar a rede e, mais tarde, carrega-la. Desta forma pode-se retomar um ponto de treino e teste anterior. Além disso é possível analisar, em tempo real, o estado de teste e treino da rede.

Inteligência Artificial Feup

Neural Network for Facial Expressions detection

Training Stuff	Hidden Layer Stuff
Training method: <input type="button" value="adadelta"/> Learning Rate(if you use sgd as a training method): <input type="text" value="0.01"/> L1-Decay(if you use sgd as a training method): <input type="text" value="0.01"/> L2-Decay: <input type="text" value="0.01"/> Batch-Size: <input type="text" value="10"/> Momentum(if you use sgd as a training method): <input type="text" value="0.9"/>	Number of hidden layers: <input type="text" value="1"/> Number of hidden layers neurons: <input type="text" value="32"/> Activation function for hidden layers: <input type="button" value="relu"/> Output Layer Stuff Activation function for output layer: <input type="button" value="softmax"/>
<input type="button" value="Init Network"/> <input type="button" value="Stop Network"/> <input type="button" value="Resume Network"/> <input type="button" value="Save Network"/> <input type="button" value="Load Network"/> Data Loaded	

Figura 8: Interface

4 Experiências

Nas várias experiências mantiveram-se constantes as seguintes configurações:

- **Uma *hidden layer*.** Este valor foi decidido após alguma pesquisa e testes, sendo suficiente para o caso em estudo.
- O **número de neurónios** das camadas intermédias (*Hidden Layers*) é obtido pela seguinte formula: **(Número de *inputs* + *outputs*) x 2/3.**
- Para **ativação** das camadas intermédias usamos a função *ReLU*.
- Para a **classificação** dos resultados de output, utilizamos a função *softmax* sendo que esta devolve a probabilidade associada a cada expressão.
- Para as experiências indicadas neste relatório utilizou-se sempre a **função de treino adadelta** em preferência a *adagrad* ou *sgd*. A função *adadelta*, ao contrário das anteriores, não é tão dependente dos valores de configuração para obter uma boa performance uma vez que se adapta dinamicamente aos mesmos ao longo do tempo de execução. Efetuamos alguns testes à rede usando a função *adadelta* e *adagrad*, obtendo resultados muito similares mas ligeiramente melhores na primeira.

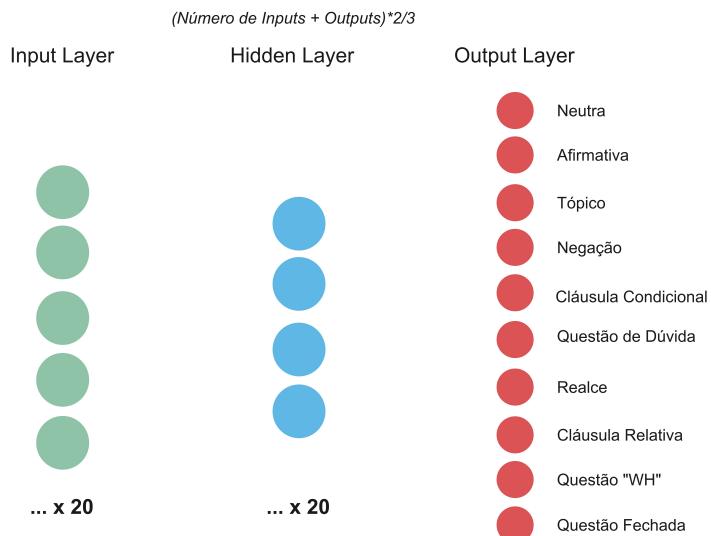


Figura 9: Estrutura base da Rede Neuronal

4.1 Resultados de várias configurações

4.1.1 Experiência 1: Caso Base

Nesta experiência começamos por selecionar 12 distâncias e 8 ângulos que permitiriam identificar as várias expressões. Foi criada uma rede capaz de identificar as 9 expressões faciais, assim como a expressão neutra.

Inicialmente foi usada a configuração base da rede, indicada anteriormente. De seguida foram realizados vários testes, onde se fez variar o número de *hidden layers* em 1, 5 e 10 e o número de neurónios nas mesmas em 20 (valor *default*) e 40.

Usamos o conjunto de dados na sua totalidade de expressões:

- **13600** Expressões Neutras
- **727** Expressões Negativas
- **938** Expressões de Cláusula Condicional
- **938** Expressões de Realce
- **576** Expressões de Tópico
- **955** Expressões de Questão Fechada, de resposta Sim ou Não :
- **901** Expressões de Questão de dúvida
- **571** Expressões Afirmativas
- **891** Expressões de Questão "WH"
- **944** Expressões de Cláusula Relativa

Encontram-se, de seguida, os resultados da configuração que obteve melhores resultados.

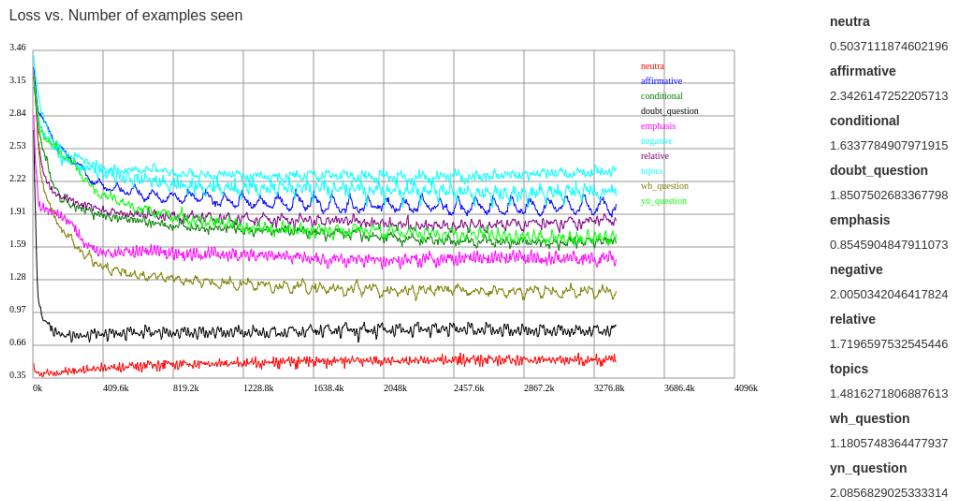


Figura 10: Gráfico de Perda da experiência 1



Figura 11: Gráfico de Treino da experiência 1

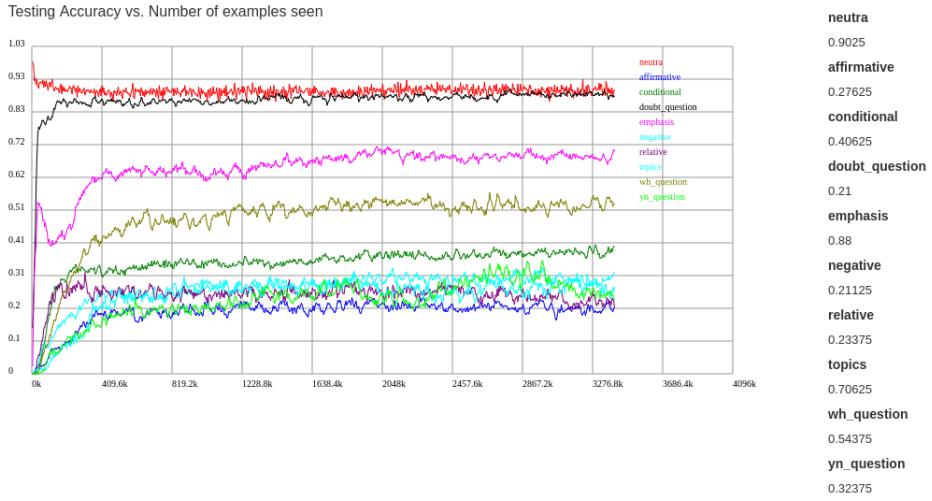


Figura 12: Gráfico de Teste da experiência 1

Os resultados das restantes configurações da rede encontram-se nos Anexos.

Conclui-se, pela análise dos gráficos, que esta rede tem uma baixa eficácia.

Isto deve-se ao facto do conjuntos de pontos não ser representativo para todas as expressões e, além disso, do facto do conjunto de dados não ser equilibrado. Existe um elevado número de expressões neutras comparativamente com as restantes o que pode ter levado a um problema de *overfitting*.

4.1.2 Experiência 2: Aumento do número de medidas fornecidas

Esta experiência parte de uma rede neuronal com 38 neurónios de *input*, 30 distâncias e 8 ângulos, com uma *hidden layer* e com 10 neurónios de *output*. O conjunto de dados usado é o mesmo da primeira experiência.

Encontra-se de seguida, os resultados da configuração que obteve melhores resultados.

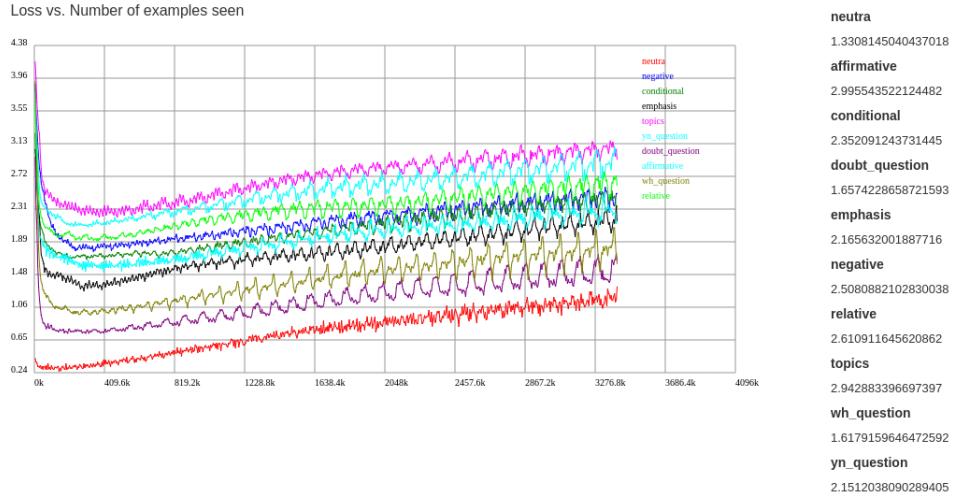


Figura 13: Gráfico de Perda

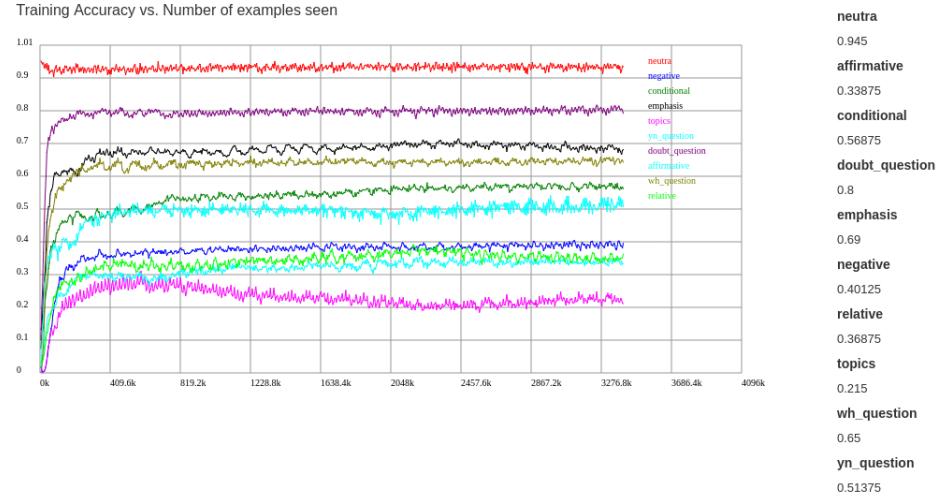


Figura 14: Gráfico de Treino

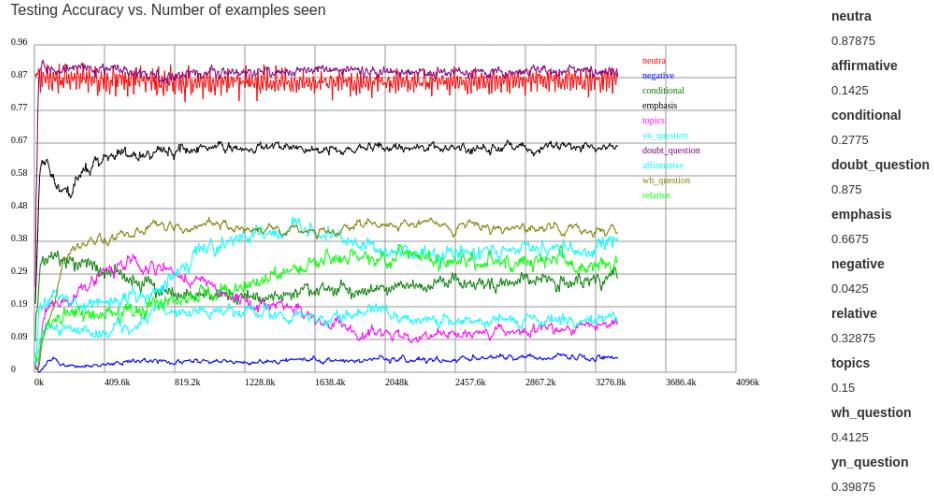


Figura 15: Gráfico de Teste

Os resultados das restantes configurações encontram-se nos Anexos.

Em comparação com a primeira experiência, os resultados pioraram ligeiramente. Desta forma, é possível concluir que as distâncias e ângulos escolhidos no primeiro caso eram suficientes para definir uma expressão e que, tal como na experiência anterior, também nesta há um problema de *overfitting*.

Os problemas da primeira experiência mantêm-se: o conjunto de pontos escolhido pode não ser representativo para todas as expressões e o conjunto de dados não é equilibrado.

Por fim, foi possível concluir que, para obter melhores resultados necessitámos de mais expressões de treino e teste e não, de mais distâncias ou ângulos para cada expressão.

4.1.3 Experiência 3: Redução do número de expressões neutras

Na experiência anterior, o *dataset* não estava equilibrado, ou seja, existia um elevado número de expressões neutras comparativamente com as restantes. De forma a resolver este problema de *overfitting*, foi decidido testar o impacto da redução do número de expressões neutras nos resultados finais.

Foi usado o seguinte volume de dados:

- **891** Expressões Neutras
- **995** Expressões Negativas
- **576** Expressões de Cláusula Condicional
- **576** Expressões de Realce
- **571** Expressões de Tópico
- **944** Expressões de Questão Fechada, de resposta Sim ou Não :
- **938** Expressões de Questão de dúvida
- **727** Expressões Afirmativas
- **891** Expressões de Questão "WH"
- **901** Expressões de Cláusula Relativa

De seguida, encontram-se, os resultados da configuração que obteve melhores resultados.

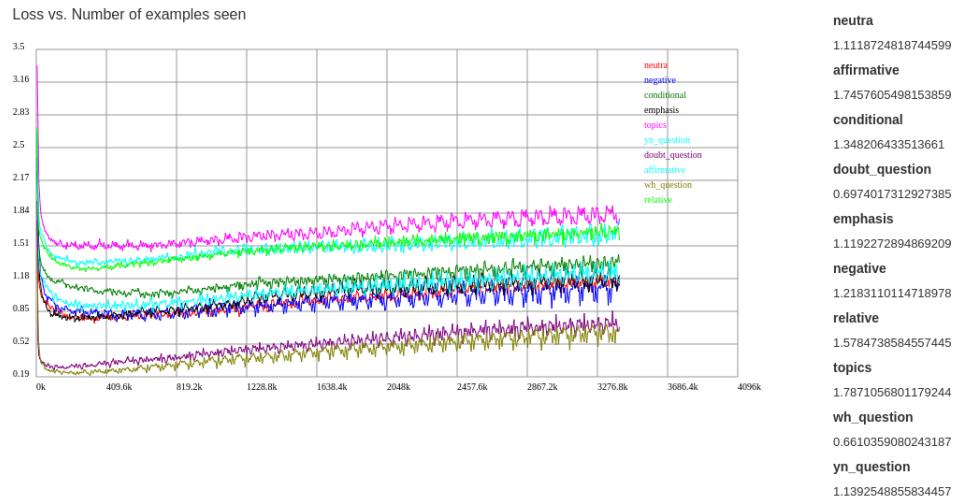


Figura 16: Gráfico de Perda

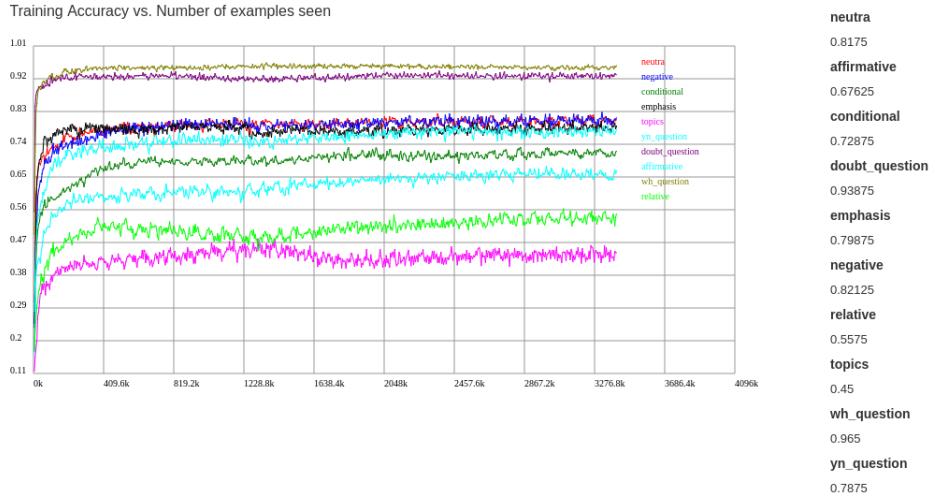


Figura 17: Gráfico de Treino

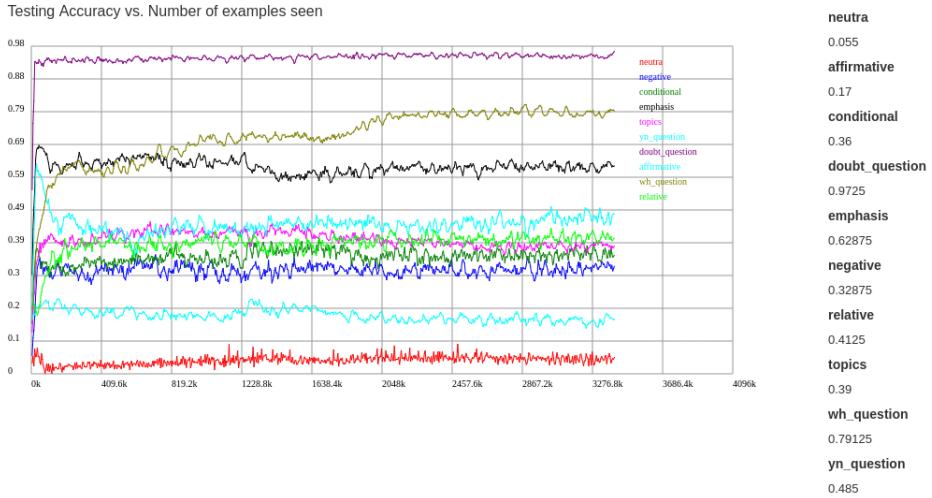


Figura 18: Gráfico de Teste

Os resultados das restantes configurações encontram-se nos Anexos.

Neste caso, é importante comparar esta experiência com a primeira. As condições mantiveram-se em ambos os casos, à exceção do número de expressões neutras, que foram reduzidas para uma quantidade semelhante à das outras expressões. Na globalidade houve uma melhoria dos resultados.

Contudo, desta vez,a eficácia de reconhecimento da expressão neutra passou de valores acima de 90% para 6,5%. Este resultado pode ser explicado pelo facto da expressão neutra não ter características específicas, sendo necessário muitos dados para a rede detetar um padrão.

4.1.4 Experiência 4: Construção de uma rede neuronal por expressão

Com o intuito de avaliar a capacidade de deteção de expressões individualmente, procedeu-se à construção de 9 redes independentes, cada uma responsável por analisar um ficheiro do *dataset* associado a uma expressão específica.

Uma vez que cada ficheiro se encontra relativamente equilibrado em relação ao número de expressões específicas e ao número de expressões neutras, não houve necessidade de reduzir o número de expressões neutras.

Foram construídas as varias redes com as configurações *default*, fazendo variar o número de *hidden layers* em 1, 5 e 10 e o número de neurónios nas mesmas em 32 (valor *default*) e 64.

Apresenta-se, de seguida, o melhor resultado da experiência.

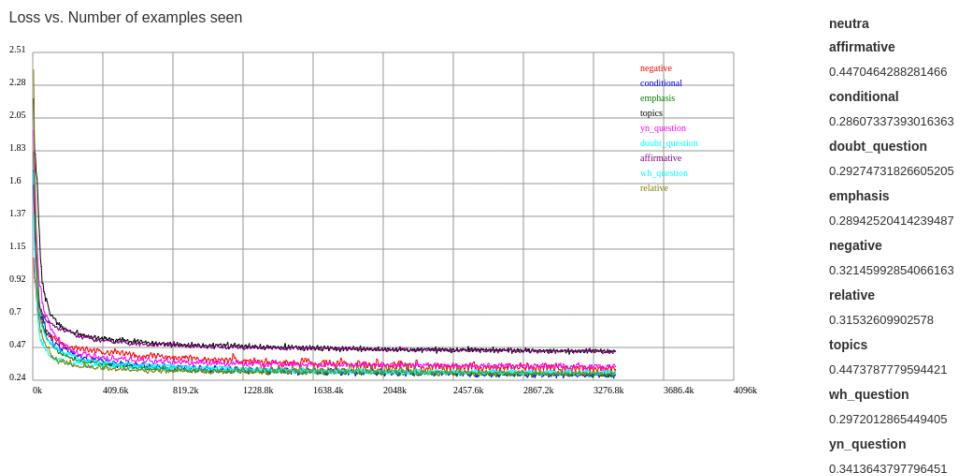


Figura 19: Gráfico de Perda

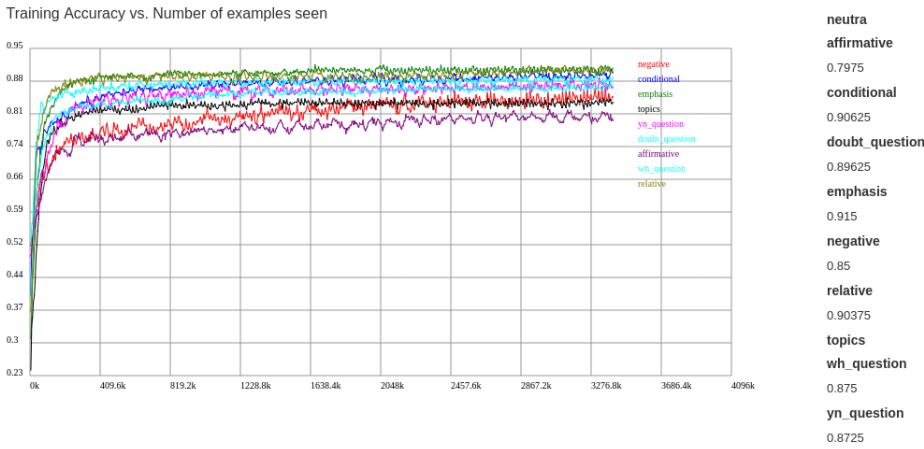


Figura 20: Gráfico de Treino

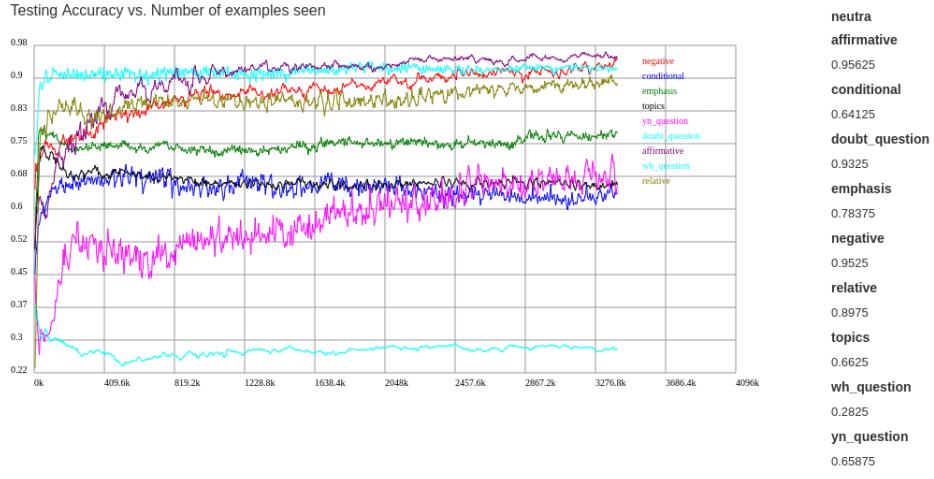


Figura 21: Gráfico de Teste

Os restantes resultados encontram-se nos anexos.

Esta experiência obteve resultados muito favoráveis. Estes resultados devem-se ao facto de cada rede ser treinada apenas para detetar a expressão em causa e a neutra e, por isso, o conjunto de dados fornecidos a cada uma delas torna-se muito mais equilibrado. A baixa eficácia na deteção da expressão *wh_question* pode ser explicada pela seleção de pontos que não representem, otimamente, a expressão em causa.

4.1.5 Experiência 5: Adição da medida z aos dados fornecidos

A quinta experiência baseia-se na primeira mas com uma pequena variação ao nível dos neurónios de entrada. Foi criada uma rede capaz de identificar as 9 expressões mais a neutra, mas desta vez foram usados 48 neurónios de entrada, 30 distâncias, 8 ângulos e 10 coordenadas z.

Não foram obtidos bons resultados nesta experiência. Foi pensada uma forma de contornar estes maus resultados. Contudo, por falta de tempo, não foi possível a aplicação desta melhoria. Foi possível concluir que se tratou de um problema de **normalização** dos dados.

A normalização refere-se a normalização das dimensões da data de forma a que tenham todas, aproximadamente, a mesma escala. O *Data Set* usado era composto por frames cada uma com 100 pontos, e cada ponto constituído por 3 coordenadas. A normalização da coordenada z é necessária, porque, em comparação com as distâncias e os ângulos obtidos através das coordenadas x e y, é representada por valores muito grandes o que leva a um desequilíbrio dos dados.

5 Conclusões

Os gráficos seguintes permitem observar a eficácia média de deteção das várias configurações usadas na rede. E quais a expressões que cada configuração (n^{o} de *hidden layers* x n^{o} de neurónios por *hidden layer*) melhor deteta.

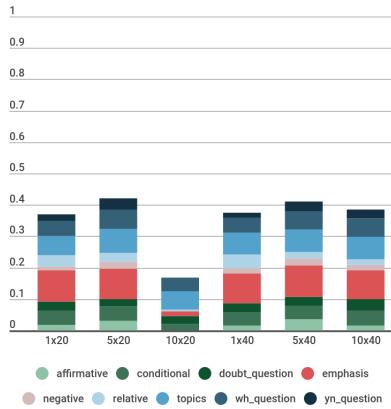


Figura 22: Resultados Experiência 1

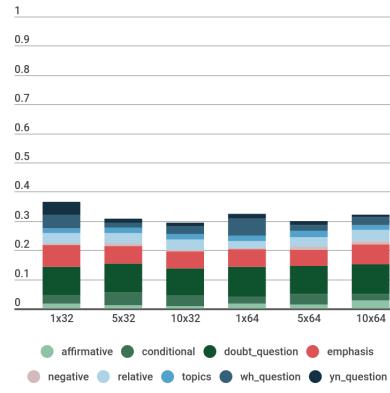


Figura 23: Resultados Experiência 2

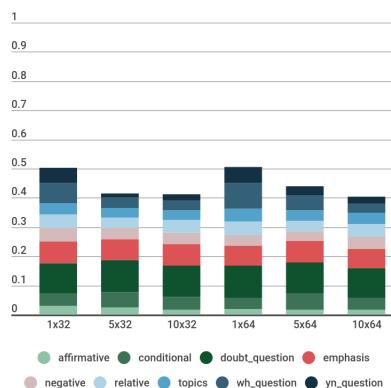


Figura 24: Resultados Experiência 3

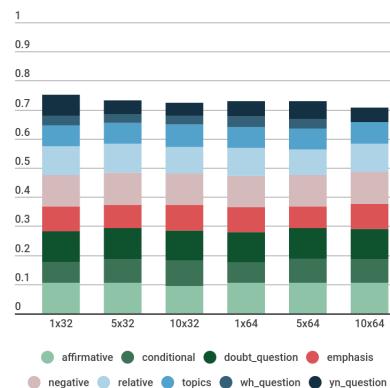


Figura 25: Resultados Experiência 4

Através dos gráficos podemos concluir que, a Experiência 4, obteve resultados consideravelmente melhores que qualquer uma das outras experiências realizadas.

Nesta experiência foi criada uma rede neuronal para cada expressão facial gramatical. O volume de dados usado para cada rede diminuiu significativamente e, por consequência, recolheram-se dados bastante mais equilibrados. Além disso, quando dividimos a rede neuronal em pequenas redes neuronais, restringimos possibilidades e, por isso aumentamos a probabilidade de acertar numa certa expressão.

Concluindo, excluímos o problema de *overfitting* porque conseguimos regularizar a data ao diminuir a complexidade dos dados.

É possível inferir ainda, que o aumento do número de neurónios, para além dos recomendados, e o aumento do número de camadas intermédias, não influenciaram significativamente os resultados obtidos.

O gráfico seguinte demonstra a eficácia da rede neuronal da Experiência 4 em cada expressão facial gramatical. Permite comparar com os valores obtidos no *paper* de referência indicado nos recursos.

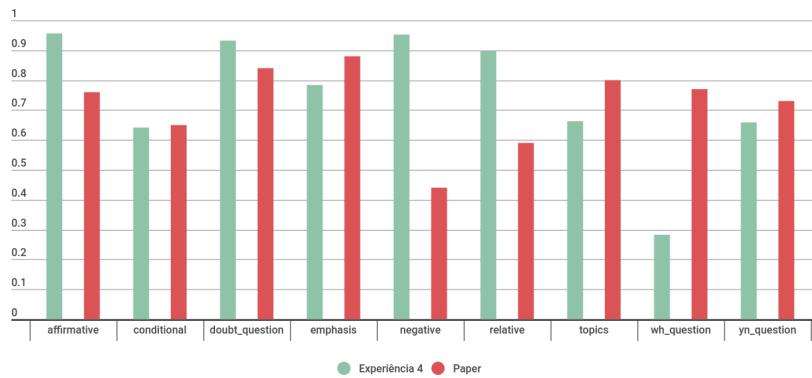


Figura 26: Comparação com os resultados do *paper* de referência

Esta análise permite validar alguns dos resultados obtidos, no sentido em que quase todas as expressões possuem valores semelhantes aos obtidos no *paper*. Comparamos apenas com a Experiência 4 pois no *paper* foi igualmente realizada uma classificação binária em expressão facial gramatical e expressão neutra. Ou seja, uma rede neuronal por expressão.

6 Melhoramentos

Alguns melhoramentos poderiam ser aplicados futuramente ao projeto, tais como a seleção de **vetores temporais** com várias *frames*. Desta forma conseguíamos detetar movimento, essencial no reconhecimento de certas expressões como, por exemplo, a de Negação e Afirmação.

Além disso, uma vez que através da Experiência 5, de adição da medida z aos dados fornecidos à rede, não foram obtidos bons resultados, seria interessante uma melhor seleção e tratamento dessas coordenadas z, **normalização** dos dados. Normalizar significa ajustar uniformemente valores do *dataset* de forma a estarem aproximadamente todas na mesma escala.

Por fim, seria oportuno a **obtenção de pontos a partir de imagens**. Desta forma, seria possível o carregamento de uma imagem, conversão para ficheiro de texto com os pontos respetivos e classificação da expressão facial realizada.

7 Recursos

7.1 Bibliografia

1. FREITAS, F. A. ; Peres, S. M. ; Lima, C. A. M. ; BARBOSA, F. V. . Grammatical Facial Expressions Recognition with Machine Learning. In: 27th Florida Artificial Intelligence Research Society Conference (FLAIRS), 2014, Pensacola Beach. Proceedings of the 27th Florida Artificial Intelligence Research Society Conference (FLAIRS). Palo Alto: The AAAI Press, 2014. p. 180-185;
2. Grammatical Facial Expressions Data Set - <http://archive.ics.uci.edu/ml/datasets/Grammatical+Facial+Expressions>
3. ConvNetJS Documentation - [http://cs.stanford.edu/people/karpathy/ convnetjs/docs.html](http://cs.stanford.edu/people/karpathy/convnetjs/docs.html)
4. Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition - <http://cs231n.github.io/>

7.2 Software

- Biblioteca ConvNetJS - Deep Learning in your browser;
- IDE - PyCharm
- IDE - VSCode
- Linguagem de programação Python
- Linguagem de programação Javascript

7.3 Percentagem do trabalho efetivo de cada elemento do grupo

Maria João Mira Paulo: 33%

Nuno Miguel Mendes Ramos: 33%

Pedro Duarte da Costa: 33%

8 Apêndice

8.1 Manual de utilizador

1. Abrir um terminal dentro da pasta da experiência desejada;
2. Correr o comando: php -S localhost:8080 ou python -m SimpleHTTPServer 8080;
3. Abrir o browser e entrar no endereço: http://localhost:8080/;
4. (Opcional) Definir as configurações da rede;
5. Esperar que aparece a informação "Data Loaded" e carregar no botão Init Network;
6. (Opcional) Guardar a rede no estado atual carregando no botão Save Network;
7. (Opcional) Parar de treinar e testar a rede carregando no botão Stop Network;
8. (Opcional) Carregar uma rede treinada, colocando as configurações da mesma na caixa de texto e pressionando o botão Load Network.

8.2 Resultados de teste das experiências

Configuração: Número de Hidden Layer x Número de Neurónios por Layer

Tabela 1: Resultados Experiência 1

Configuração	Experiência 1					
	1x20	5x20	10x20	1x40	5x40	10x40
affirmative	0,1663	0,2763	0,0038	0,1275	0,3175	0,1325
conditional	0,3975	0,4063	0,1738	0,3975	0,4013	0,4438
doubt_question	0,2738	0,2100	0,2275	0,2663	0,2538	0,3088
emphasis	0,8763	0,8800	0,1375	0,8500	0,8863	0,8475
negative	0,1175	0,2113	0,0038	0,1575	0,1988	0,1425
relative	0,3175	0,2338	0,0538	0,3838	0,1863	0,1775
topics	0,5688	0,7063	0,5325	0,6188	0,6500	0,6388
wh_question	0,4388	0,5438	0,3813	0,4263	0,5175	0,5225
yn_question	0,1638	0,3238	0,0000	0,1563	0,2913	0,2450
Média	0,3689	0,4213	0,1682	0,3760	0,4114	0,3843

Tabela 2: Resultados Experiência 2

Configuração	Experiência 2					
	1x32	5x32	10x32	1x64	5x64	10x64
affirmative	0,1425	0,1025	0,0713	0,1600	0,1275	0,2550
conditional	0,2775	0,3888	0,3263	0,2150	0,3238	0,1950
doubt_question	0,8750	0,8863	0,8388	0,9200	0,8713	0,9150
emphasis	0,6675	0,5538	0,5138	0,5425	0,4775	0,6163
negative	0,0425	0,0725	0,0575	0,0138	0,1113	0,0838
relative	0,3288	0,3250	0,3088	0,2213	0,2850	0,3700
topics	0,1500	0,1638	0,1863	0,1800	0,2025	0,1375
wh_question	0,4125	0,1575	0,2338	0,5138	0,1688	0,2538
yn_question	0,3988	0,1200	0,1050	0,1538	0,1288	0,0713
Média	0,3661	0,3078	0,2935	0,3244	0,2996	0,3219

Tabela 3: Resultados Experiência 3

Configuração	Experiência 3					
	1x32	5x32	10x32	1x64	5x64	10x64
affirmative	0,2700	0,2163	0,1475	0,1700	0,1475	0,1450
conditional	0,3650	0,4713	0,3863	0,3600	0,4900	0,3675
doubt_question	0,9588	0,9838	0,9800	0,9725	0,9650	0,9338
emphasis	0,6550	0,6600	0,6738	0,6288	0,6663	0,5900
negative	0,4125	0,3388	0,3350	0,3288	0,3150	0,3588
relative	0,4413	0,3125	0,3925	0,4125	0,3213	0,3950
topics	0,3338	0,3138	0,3075	0,3900	0,3125	0,3488
wh_question	0,6075	0,3025	0,2900	0,7913	0,4738	0,2638
yn_question	0,4688	0,1225	0,2050	0,4850	0,2688	0,2375
Média	0,5014	0,4135	0,4131	0,5043	0,4400	0,4044

Tabela 4: Resultados Experiência 4

Configuração	Experiência 4					
	1x32	5x32	10x32	1x64	5x64	10x64
affirmative	0,9563	0,9463	0,8563	0,9338	0,9563	0,9563
conditional	0,6413	0,7213	0,7775	0,6338	0,7375	0,7125
doubt_question	0,9325	0,9563	0,9200	0,9413	0,9463	0,9413
emphasis	0,7838	0,7425	0,8013	0,7763	0,6738	0,7750
negative	0,9525	0,9725	0,9738	0,9663	0,9575	0,9800
relative	0,8975	0,8950	0,8300	0,8638	0,8063	0,8750
topics	0,6625	0,6475	0,6850	0,6513	0,6313	0,6725
wh_question	0,2825	0,2913	0,2738	0,3250	0,2988	0,0000
yn_question	0,6588	0,4238	0,3925	0,4800	0,5463	0,4513
Média	0,7519	0,7329	0,7233	0,7301	0,7282	0,7071