**Universidade Federal de São Carlos**
Aprendizado de Máquina
Prof. Tiago A. Almeida <talmeida@ufscar.br>

# Exercício Prático 5: Redes Neurais Artificiais

## Introdução

Neste exercício, você irá implementar o algoritmo de *backpropagation* para redes neurais artificiais e aplicá-lo na tarefa do reconhecimento de dígitos manuscritos. Antes de iniciar o exercício, é fortemente recomendado que você revise o material apresentado em aula.

## Arquivos incluídos neste exercício

`ex05.m` – Script geral do exercício
`ex05_dados1.mat` – Base de dados de treinamento com os dígitos manuscritos
`ex05_pesos.mat` – Parâmetros da rede neural para o exercício 5
`visualizaDados.m` – Função para plotar e visualizar os dados da base de treinamento
`fmincg.m` – Função de minimização (similar à `fminunc`)
`sigmoide.m` – Função sigmoidal
`gradienteNumerico.m` – Função para calcular o gradiente numérico
`verificaGradiente.m` – Função para checagem da corretude do gradiente
`inicializaPesos.m` – Função para inicializar os pesos
`inicializaPesosAleatorios.m` – Função para inicializar os pesos aleatoriamente
`predicao.m` – Função para predizer a classe dada uma amostra
[ ⋆ ] `gradienteSigmoide.m` – Função para calcular o gradiente da sigmóide
[ ⋆ ] `rnaCusto.m` – Função para calcular a função de custo da rede neural

⋆ indica os arquivos que você precisará completar.

O arquivo `ex05.m` conduzirá todo o processo desse exercício.

# O Problema

Você foi contratado por uma grande empresa para fazer a identificação correta e automática de quais dígitos estão presentes em um conjunto de imagens. Essas imagens têm dimensão de 20 x 20 pixels, onde cada pixel é representado por um ponto flutuante que indica a intensidade de tons de cinza naquela região.

Sabe-se que a aplicação de redes neurais utilizando o algoritmo de *backpropagation* neste tipo de problema obtem resultados satisfatórios. Assim, seu desafio é implementar tal algoritmo e encontrar os pesos ótimos para que a rede seja capaz de identificar automaticamente os dígitos contidos nas imagens, conforme lhe é pedido.

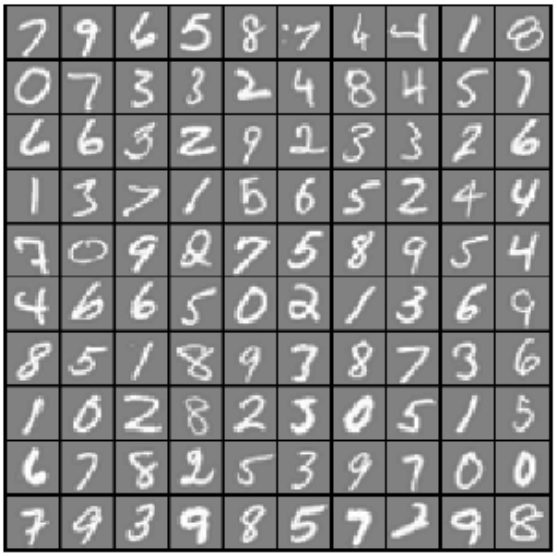

Figura 1: Amostras do conjunto de dados

# Visualização dos Dados

Na primeira etapa do procedimento `ex05.m`, a base de dados é carregada e as amostras são plotadas em 2D para melhor visualização e interpretação dos dados (veja a Figura 1).

Neste contexto, cada imagem tem dimensão de 20 x 20 pixels e cada pixel é representado por um ponto flutuante com a intensidade do tom de cinza naquela região. Deste modo, cada amostra é representada pelo desdobramento dos pixels em um vetor com 400 dimensões.

O conjunto de dados contém 5000 amostras, onde cada amostra é representada por um vetor com 400 dimensões devido ao desdobramento dos pixels explicado anteriormente, podendo ser representado por uma matriz [5000,400].

A segunda parte do conjunto de dados é um vetor `y` com 5000 dimensões, o qual contém os rótulos para cada amostra da base de treino. Imagens contendo dígitos de 1 a 9 recebem, respectivamente, classes de 1 a 9, enquanto imagens contendo o dígito 0 são rotuladas como 10.

# Representação do Modelo

A rede neural proposta para este exercício tem 3 camadas: uma camada de entrada, uma camada oculta e uma camada de saída (Figura 2). É importante lembrar que a camada de entrada possui 400 neurônios devido ao desdobramento dos pixels das amostras em vetores

com 400 dimensões (sem considerar o *bias*, sempre +1). Os dados para treinamento serão carregados nas variáveis `X` e `y` pelo script `ex05.m`.

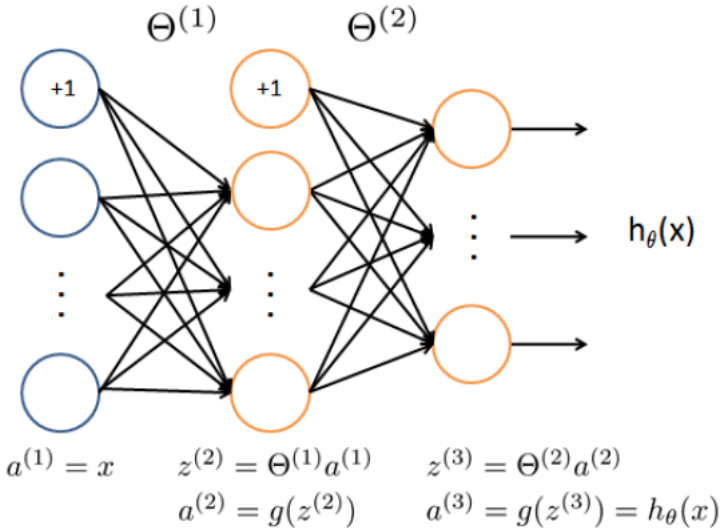


Figura 2: Arquitetura da rede neural

Inicialmente, você também terá acesso aos parâmetros ($\Theta^{(1)}$, $\Theta^{(2)}$) de uma rede neural já treinada, armazenados em `ex05_pesos.mat`. Esses valores serão carregados nas variáveis `Theta1` e `Theta2`. Tais parâmetros têm dimensões condizentes com uma rede neural com 25 neurônios na camada intermediária e 10 neurônios na camada de saída (correspondente às dez possíveis classes).

## *Feedforward* e Função de Custo

Agora, você irá implementar a função de custo e gradiente para a rede neural. Primeiro, complete o código em `rnaCusto.m` para retornar o custo. A função de custo para a rede neural (sem regularização) é descrita a seguir.

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} [-y_k^{(i)} \log((h_\Theta(x^{(i)})_k) - (1 - y_k^{(i)}) \log((1 - h_\Theta(x^{(i)})_k)]$$

Na função $J$, $h_\Theta(x^{(i)})$ é computado conforme representado na Figura 2. A constante $K$ representa o número de classes. Assim, $h_\Theta(x^{(i)})_k = a_k^{(3)}$ é a ativação (valor de saída) da $k$-ésima unidade de saída. Também, é importante lembrar que o vetor de saída precisa ser criado a partir da classe original, tornando-se compatível com a rede neural, ou seja, espera-se vetores com 10 posições contendo 1 para o elemento referente à classe esperada e 0 nos demais elementos. Por exemplo, seja 5 o rótulo de determinada amostra, o vetor `y` correspondente terá 1 na posição $y_5$ e 0 nas demais posições.

Você deve implementar o algoritmo *feedfoward* para calcular $h_\Theta(x^{(i)})$ para cada amostra $i$ e somar o custo de todas amostras. O código deve ser flexível para funcionar com conjuntos de dados de qualquer tamanho e qualquer quantidade de classes, considerando $K \geq 3$.

Completada essa etapa, o *script* `ex05.m` chamará a função `rnaCusto.m` usando os valores carregados nas variáveis `Theta1` e `Theta2`. O resultado do custo deverá ser igual a **0,2876**.

# Função de Custo Regularizada

A função de custo para redes neurais com regularização é descrita a seguir.

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-y_k^{(i)}\log((h_\Theta(x^{(i)})_k) - (1-y_k^{(i)})\log((1-h_\Theta(x^{(i)})_k)]$$

$$+\frac{\lambda}{2m}[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2]$$

Você pode assumir que a rede neural terá 3 camadas – uma camada de entrada, uma camada oculta e uma camada de saída. No entanto, seu código deve ser flexível para suportar qualquer quantidade de neurônios em cada uma dessas camadas. Embora a função $J$ descrita anteriormente contenha números fixos para $\Theta^{(1)}$ e $\Theta^{(2)}$, o código deve funcionar para outros tamanhos.

Também, é importante que a regularização não seja aplicada a termos relacionados ao *bias*. Neste contexto, estes termos estão na primeira coluna de cada matriz $\Theta^{(1)}$ e $\Theta^{(2)}$.

Completada essa etapa, o *script* `ex05.m` chamará a função `rnaCusto.m` usando os valores carregados nas variáveis `Theta1` e `Theta2`, e $\lambda = 1$, onde o custo esperado é **0,3837**.

# *Backpropagation*

Nesta parte do exercício, você implementará o algoritmo de *backpropagation* responsável por calcular o gradiente para a função de custo da rede neural. Neste ponto, a função `rnaCusto.m` já deve estar implementada para que a variável `grad` seja alimentada corretamente. Terminada a implementação do cálculo do gradiente, você poderá treinar a rede neural minimizando a função de custo $J(\Theta)$ usando um otimizador avançado como `fmincg`.

Primeiro, você precisará implementar o gradiente para a rede neural sem regularização. Após ter verificado que o cálculo do gradiente está correto, você implementará o gradiente para a rede neural com regularização.

## Gradiente da Sigmóide

Você deve começar pela implementação do gradiente da sigmóide, o qual pode ser calculado utilizando a equação:

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1-g(z)),$$

sendo que

$$g(z) = \frac{1}{1+e^{-z}}.$$

Ao completar, teste diferentes valores para a função `gradienteSigmoide(z)` utilizando a linha de comando. Para valores grandes de `z` (tanto positivo, quanto negativo), o resultado deve ser próximo a zero. Quando `z = 0`, o resultado deve ser exatamente **0,25**. A função deve funcionar com vetores e matrizes também. No caso de matrizes, a função deve calcular o gradiente para cada elemento.

## Implementando *Backpropagation*

O objetivo do algoritmo de *backpropagation* é encontrar a parcela de responsabilidade que cada neurônio da rede neural teve com o erro gerado na saída. Dada uma amostra de treino $(x^{(t)}, y^{(t)})$, primeiro é executado o passo de *feedforward* para calcular todas ativações na rede, incluindo o valor de saída $h_\Theta(x)$. Então, para cada neurônio $j$ na camada $l$, é calculado o "erro" $\delta_j^{(l)}$ que mede quanto determinado neurônio contribuiu para a diferença entre o valor esperado e o valor obtido na saída da rede.

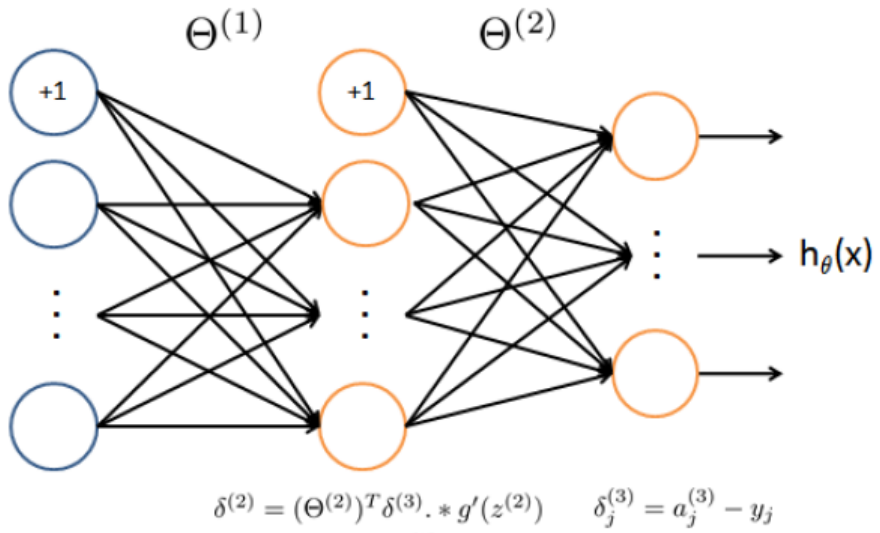


Figura 3: Arquitetura da rede neural

Nos neurônios de saída da rede, a diferença pode ser medida entre o valor esperado (dado pelo rótulo da amostra) e o valor obtido (a ativação final da rede), onde tal diferença será usada para definir $\delta_j^{(3)}$ (visto que a camada 3 é a última camada). Nas camadas ocultas (quando houver mais de uma), o termo $\delta_j^{(l)}$ será calculado com base na média ponderada dos erros encontrados na camada posterior $(l + 1)$.

A seguir, é descrito em detalhes como a implementação do algoritmo *backpropagation* deve ser feita. Você precisará seguir os passos 1 a 4 dentro de um laço, processando uma amostra por vez. No passo 5, o gradiente acumulado é dividido pelas $m$ amostras, o qual será utilizado na função de custo da rede neural.

1. Coloque os valores na camada de entrada $(a^{(1)})$ para a amostra de treino a ser processada. Calcule as ativações das camadas 2 e 3 utilizando o passo de *feedforward*. Observe que será necessário adicionar um termo `+1` para garantir que os vetores de ativação das camadas $(a^{(1)})$ e $(a^{(2)})$ também incluam o neurônio de *bias*.

2. Para cada neurônio $k$ na camada 3 (camada de saída), defina

$$\delta_k^{(3)} = (a_k^{(3)} - y_k),$$

   onde $y_k \in \{0, 1\}$ indica se a amostra sendo processada pertence a classe $k$ $(y_k = 1)$ ou não $(y_k = 0)$.

3. Para a camada oculta $l = 2$, defina

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} . * g'(z^{(2)})$$

4. Acumule o gradiente usando a fórmula descrita a seguir. Lembre-se de não utilizar o valor associado ao bias $\delta_0^{(2)}$.

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

5. Obtenha o gradiente sem regularização para a função de custo da rede neural dividindo os gradientes acumulados por $\frac{1}{m}$:

$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)}$$

Após implementado, o script `ex05.m` prosseguirá com a checagem do gradiente. Esta checagem tem o propósito de certificar que seu código calcula o gradiente corretamente. Neste passo, se sua implementação estiver correta, você deverá ver uma diferença **menor que 1e-9**.

### Redes Neurais com Regularização

A regularização deve ser adicionada após se calcular o gradiente durante o algoritmo de *backpropagation*, lembrando que a regularização não é adicionada quando $j = 0$, ou seja, não deve ser adicionada regularização na primeira coluna de $\Theta$. Portanto, para $j \geq 1$, o gradiente é descrito como:

$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{1}{m}\Theta_{ij}^{(l)}$$

Modifique seu código para calcular `grad` na função `rnaCusto.m` considerando a regularização. Após terminar esta etapa, o script `ex05.m` prosseguirá com a execução da checagem do gradiente. Se sua implementação estiver correta, a diferença observada deverá ser **menor que 1e-9**.

### Aprendendo parâmetros usando `fmincg`

Após ter completado com sucesso todos os passos anteriores, a etapa seguinte do script `ex05.m` usará a função `fmincg` para aprender um bom conjunto de parâmetros para a rede neural.

Uma vez terminado o treinamento, é exibida a porcentagem de acertos com relação às amostras classificadas corretamente. Se sua implementação estiver correta, é esperado um resultado próximo à **95,3%** (podendo variar até 2% devido a inicialização aleatória).

É possível obter resultados ainda melhores ao treinar a rede utilizando mais iterações (variável `MaxIter`) e alterando o valor do parâmetro de regularização $\lambda$.

## Visualização da Camada Oculta

Uma das formas de entender o que a rede neural está aprendendo é visualizar a representação capturada nos neurônios da camada oculta. Informalmente, dado um neurônio de uma camada oculta qualquer, uma das formas de visualizar o que esse neurônio calcula é encontrar uma entrada $x$ que o fará ser ativado (ou seja, um resultado próximo a 1). Para a rede neural que foi treinada, perceba que a $i$-ésima linha de $\Theta^{(1)}$ é um vetor com 401 dimensões, o qual representa os parâmetros para o $i$-ésimo neurônio. Se descartarmos o termo *bias*, teremos um vetor de 400 dimensões que representa o peso para cada pixel a partir da camada de entrada.

Deste modo, uma das formas de visualizar a representação capturada pelo neurônio da camada oculta é reorganizar essas 400 dimensões em uma imagem de 20 x 20 pixels e exibi-la. A etapa seguinte do script `ex05.m` faz exatamente isso, e deverá exibir uma imagem semelhante à Figura 4 com 25 unidades, cada uma correspondendo a um neurônio da camada oculta.

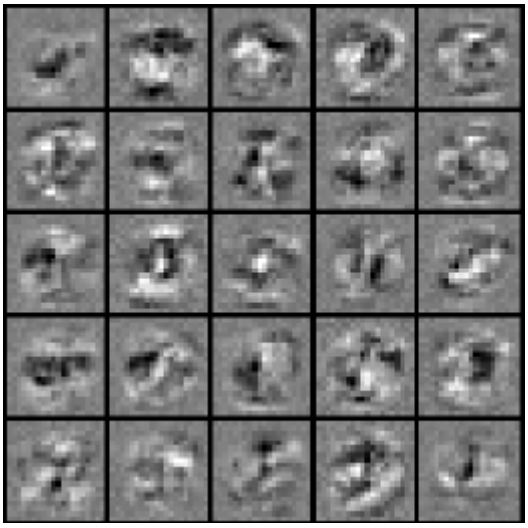

Figura 4: Visualização da rede neural

Na sua rede treinada, você poderá perceber que os neurônios na camada oculta capturam os traços e outros padrões a partir das amostras de entrada.