Redes Neurais – Parte III*

1. Aprendizado Profundo (*Deep Learning*)

- É provável que o leitor já tenha ouvido algo acerca de redes neurais profundas (*deep neural networks*, DNNs) ou aprendizado profundo (*deep learning*, DL). A razão para isso é o enorme sucesso que essa abordagem vem tendo em domínios tão diversos quanto reconhecimento de imagens/áudio e tradução automática de textos (LECUN ET AL., 2015).

* O material deste tópico é fortemente inspirado em (GOODFELLOW ET AL., 2016), e as figuras são, salvo menção em contrário, extraídas de lá.

- Se fosse necessário fornecer uma explicação para o crescimento acelerado do interesse por DNNs e DL, ela teria de passar por dois fatores cruciais do mundo contemporâneo:
  - O constante aumento da quantidade e disponibilidade de dados de todos os tipos.
  - O significativo desenvolvimento de *hardware* capaz de permitir o treinamento de estruturas massivas (explorando, por exemplo, paralelismo).
- A tônica de *deep learning* é: *deixe os dados "falarem por si mesmos"*. Em outras palavras, a partir da informação disponível, *aprenda diferentes níveis de representação que permitam a solução do problema*. Isso contrasta com diversas abordagens clássicas em que se busca definir as representações dos dados por meio de engenharia de atributos *a priori*.

- Um ponto central é que, embora haja muitos aspectos teoricamente relevantes do conceito de DNN, o paradigma de DL se afirmou principalmente graças ao excelente desempenho obtido em comparação com o estado da arte de diversas áreas. Para ilustrar esse ponto, vejamos a Fig. 1, que traz a evolução da taxa de erro de classificação da ImageNet *Large Scale Visual Recognition Challenge* (GOODFELLOW ET AL., 2016). É nítido o progresso a partir de 2012.

1.1. Profundidade / Número de Camadas: Reflexões

- Se uma rede com uma camada intermediária não-linear já é um aproximador universal, por que uma arquitetura profunda (com um número elevado de camadas) é relevante? Para começarmos a refletir sobre isso, vejamos a Fig. 2.

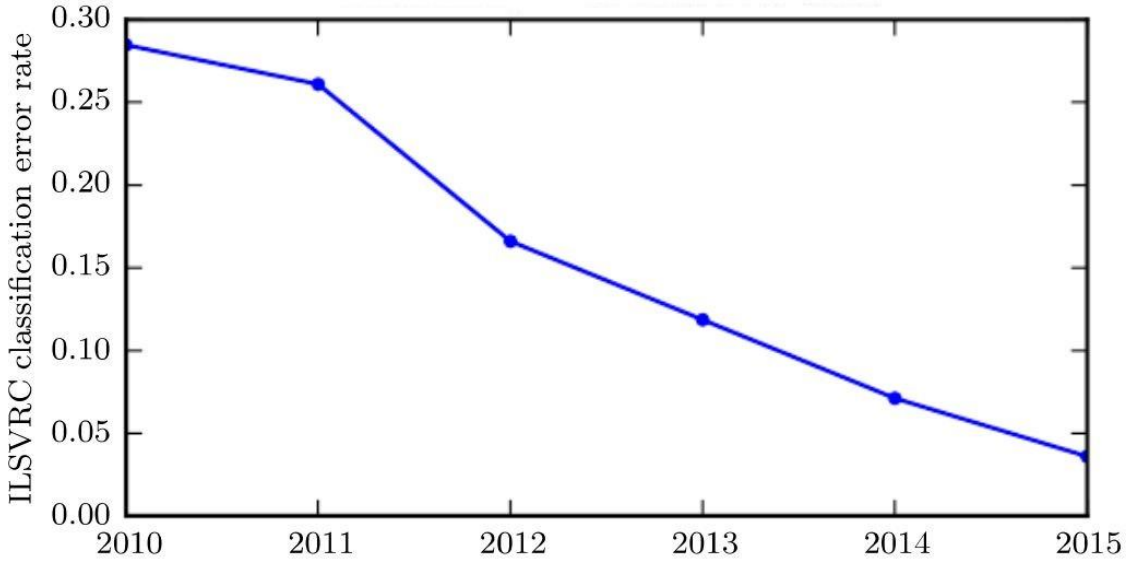

Fig. 1 – Evolução da Taxa de Erro.

- Nota-se que o aumento da quantidade de parâmetros *por meio do aumento do número de camadas* é, no caso, muito mais proveitoso. Esse ganho de desempenho ocorre em diversas circunstâncias: muitas vezes, redes que teriam um número absurdo de neurônios numa arquitetura com uma camada intermediária são praticáveis numa arquitetura mais profunda.

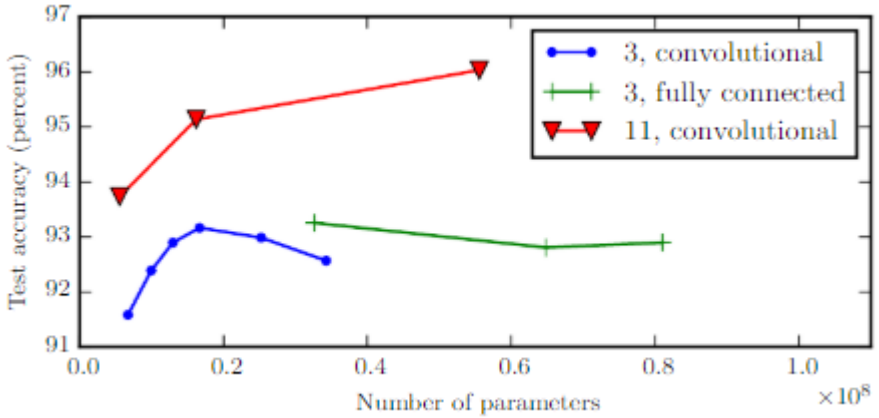

Fig. 2 – Análise do Impacto do No. de Camadas.

- Outro ponto interessante surge quando analisamos o comportamento de uma rede profunda (no caso, uma rede convolucional) do ponto de vista do papel emergente de cada camada numa tarefa como a de reconhecimento de padrões. Em caso de sucesso, espera-se que cada camada termine por ter um papel definido, ou seja, que a solução do problema seja "quebrada em etapas". Vejamos a Fig. 3.

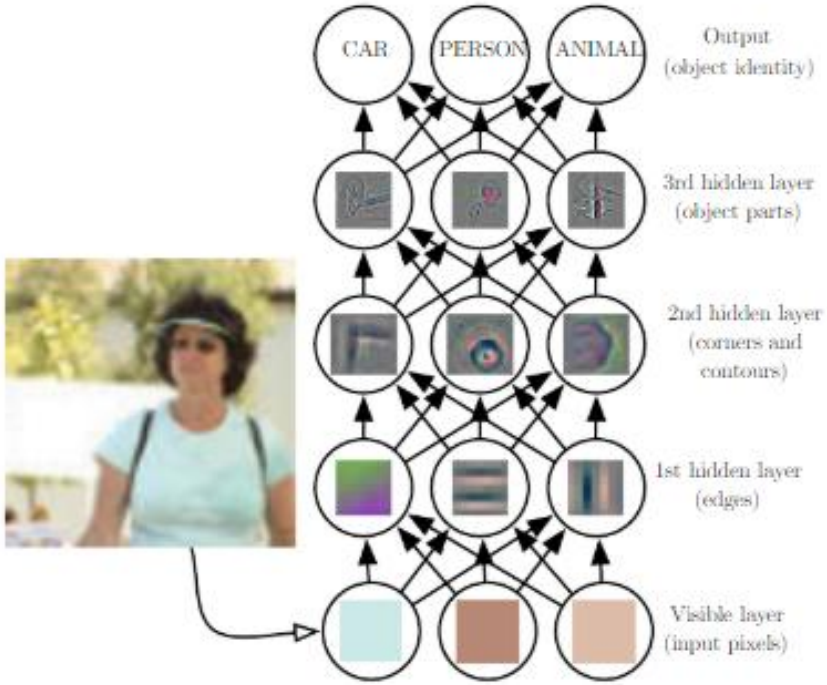

Fig. 3 – Análise da Operação de uma Rede.

2. Redes Convolucionais

- Para que continuemos a refletir sobre redes profundas, convém ter como cenário as paradigmáticas **redes convolucionais**. Muitas redes profundas importantes têm camadas convolucionais, embora se deva ressaltar que é perfeitamente válido trabalhar com redes convolucionais com "poucas camadas" (numa abordagem *shallow*) e com redes profundas sem camadas convolucionais.
- Redes convolucionais empregam a operação de **convolução** no lugar da multiplicação matricial padrão (inerente a uma camada tipo *perceptron*) em pelo menos uma de suas camadas (GOODFELLOW ET AL., 2016). Essa operação, que é linear, permite explorar informações em estruturas organizadas no tempo (como séries temporais) ou no espaço (como imagens).

2.1. Convolução

- Se tivermos duas sequências $x(n)$ e $w(n)$, a convolução entre ambas, $y(n)$, é:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)w(n-k) = (x * w)(n)$$

- Se $x(n)$ for um sinal discreto e $w(n)$ for a resposta ao impulso de um sistema linear e invariante no tempo, pode-se mostrar que a saída desse sistema será exatamente a sequência $y(n)$ como expressa acima. No âmbito de redes, costuma-se dizer que $x(n)$ é a entrada e $w(n)$ é o *kernel* (GOODFELLOW ET AL., 2016).
- A convolução como apresentada acima se harmoniza muito bem com estruturas 1-D como séries temporais. Também é possível estender a ideia para duas dimensões, o que é interessante, por exemplo, quando se lida com imagens:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

- Como a convolução é comutativa, temos:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n K(m,n)I(i-m, j-n)$$

- Uma operação bastante similar é conhecida como correlação-cruzada. Nela, não há inversão de sinal entre $I(\cdot)$ e $K(\cdot)$, o que é pertinente, embora a comutatividade no sentido estrito se perca:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

- Consideraremos aqui que esse caso também é uma operação de convolução. Quando a questão de inversão (*flip*) for importante, isso será informado. Um exemplo sem inversão é mostrado na Fig. 4.

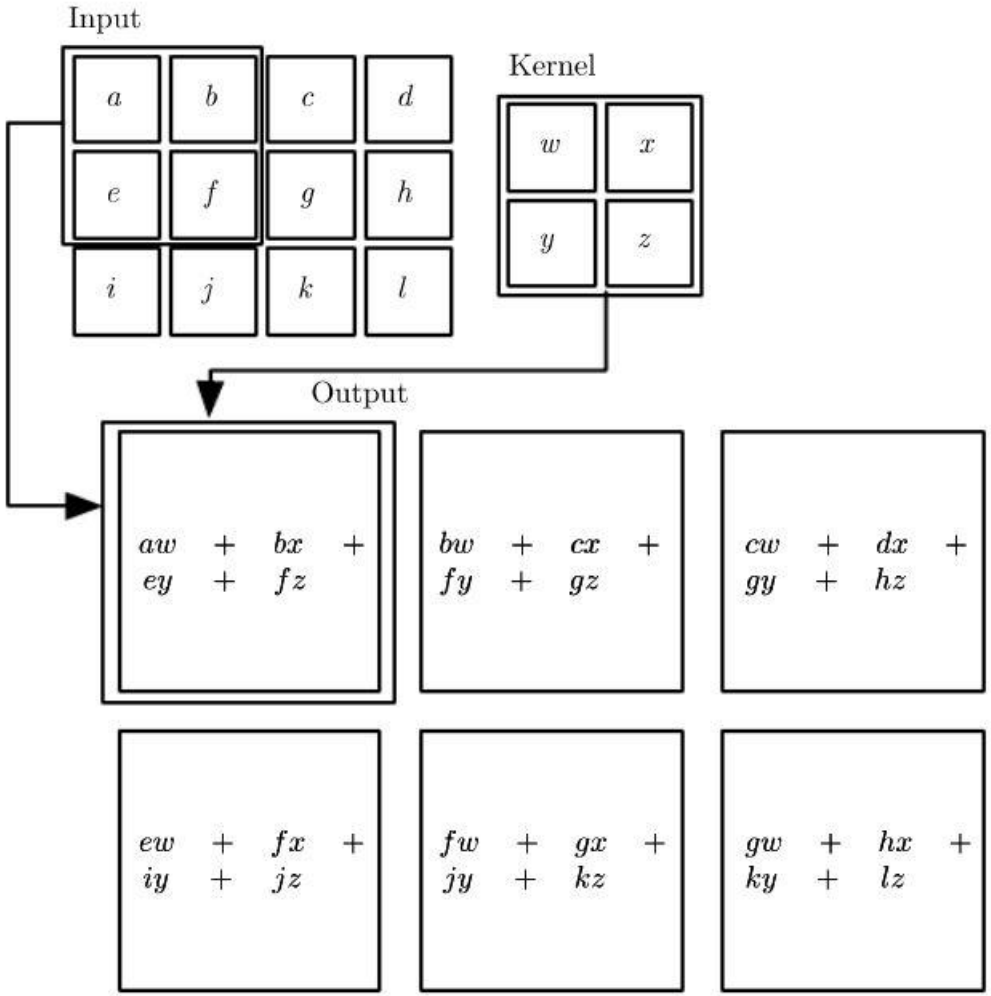

Fig. 4 – Exemplo de Convolução (sem *flip*).

2.1.1. Interações Esparsas

- Um primeiro ponto interessante ligado ao uso de uma camada convolucional é que ela, ao contrário do que ocorre para redes densamente conectadas como a MLP, leva a interações esparsas entrada-saída. Isso tem lugar caso o *kernel* seja menor que a entrada. *Kernels* relativamente pequenos já podem ser capazes de detectar características importantes numa imagem, como bordas. Isso leva a um ganho do ponto de vista de armazenamento de informação.
- Se houvesse m entradas e n saídas, a multiplicação matricial típica de camadas *fully connected* levaria a $O(m \times n)$ parâmetros. Por outro lado, se cada saída tivesse apenas k conexões, com $k << m$, lidar com $O(k \times n)$ seria uma vantagem expressiva. As Figs. 5 e 6 nos ajudarão a entender melhor. Na Fig. 5, analisaremos como uma entrada genérica x_3 afeta os neurônios da camada

seguinte para os casos de uma camada convolucional e de uma densamente conectada.

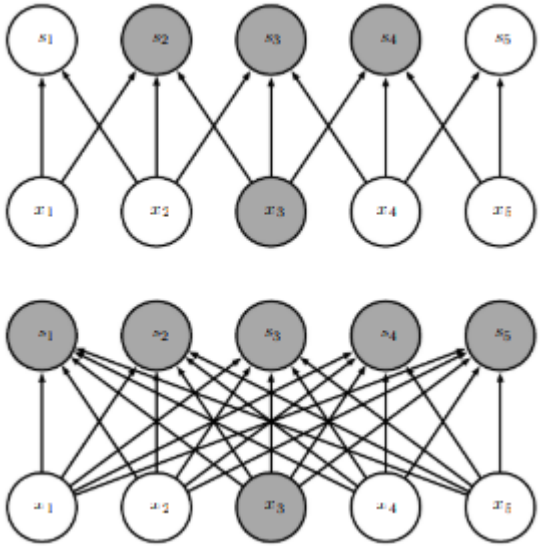

Fig. 5 – Saídas Afetadas pela Entrada x_3 (Convolucional e Densa).

- A perspectiva da saída é contemplada na Fig. 6. As unidades que afetam uma saída são seu campo receptivo (*receptive field*).

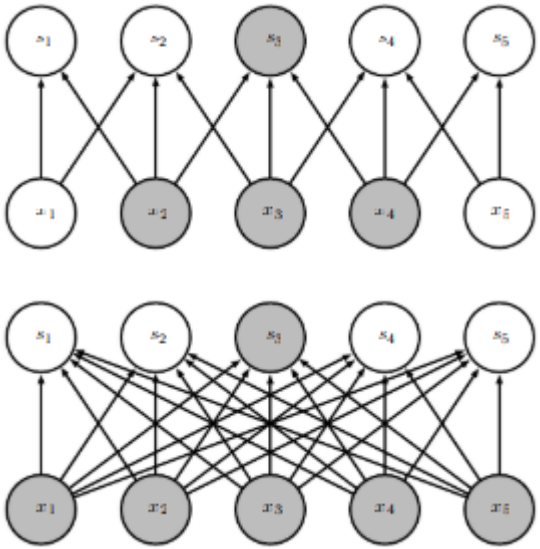

Fig. 6 – Perspectiva da Saída s_3 / Campo Receptivo.

- Para uma rede profunda, pode-se perceber que o campo receptivo vai se ampliando significativamente – mesmo com conexões esparsas. A Fig. 7 mostra isso.

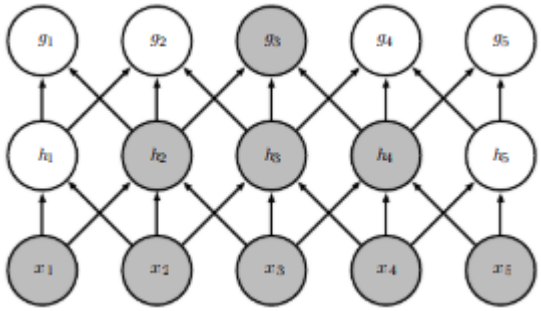

Fig. 7 – Ampliação do Campo Receptivo com um Maior No. de Camadas.

2.1.2. Compartilhamento de Parâmetros

- Em redes totalmente conectadas, cada elemento da matriz de pesos é usado uma única vez no cômputo da saída da camada. Já no caso de redes convolucionais, há compartilhamento de parâmetros (*parameter sharing*). No deslocamento da convolução, todos os coeficientes do *kernel* passam por todas as entradas (exceto nas bordas eventualmente). O mesmo conjunto de parâmetros aprendido é usado em diferentes partes da entrada, o que é excelente do ponto de vista de custo de armazenamento. A Fig. 8 ilustra o conceito.

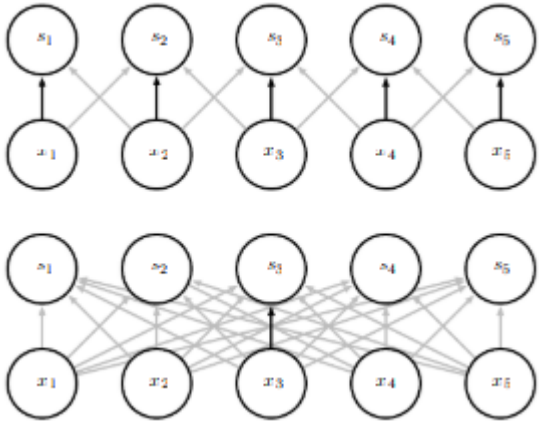

Fig. 8 – Influência de um Parâmetro (Convolucional e Densa).

- Outro ponto importante: equivariância à translação.

2.1.3. Pooling

- Uma camada convolutiva tem, tipicamente, três estágios: a) um estágio de realização de convoluções em paralelo; b) um estágio de aplicação de uma função de ativação não-linear (e.g. a função retificadora); c) um estágio de

aplicação de uma função de *pooling* que modifica a saída da função. A Fig. 9 ilustra isso.

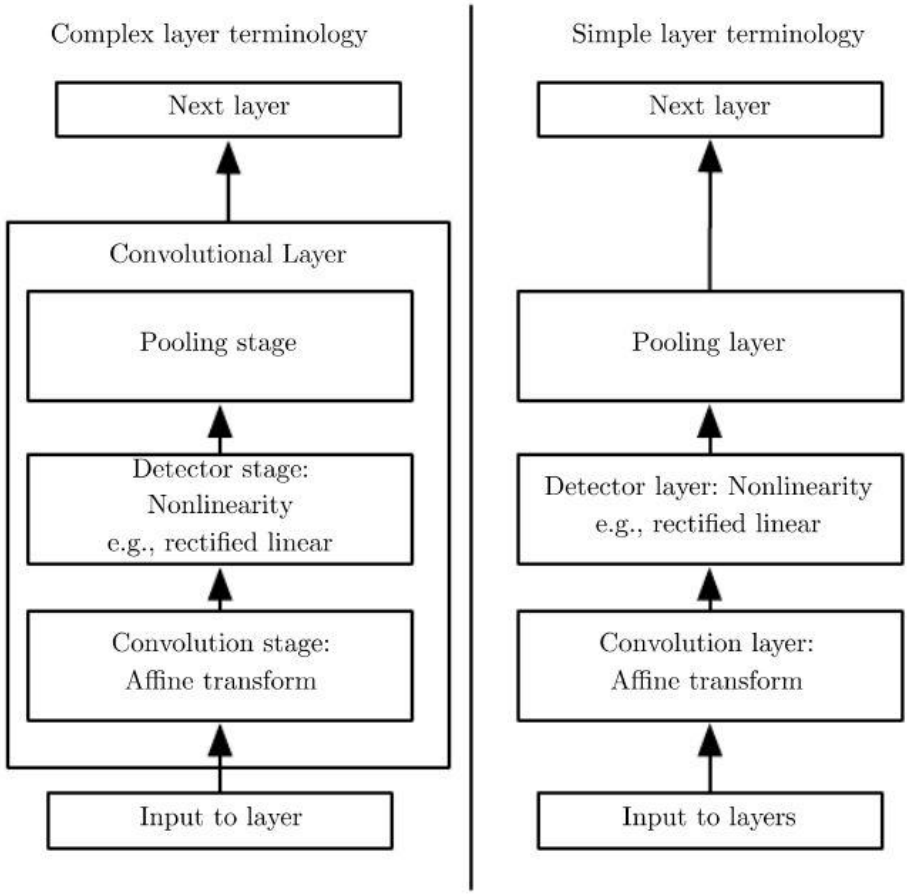

Fig. 9 – Partes de uma Camada Convolucional.

- A razão de ser da função de *pooling* é fazer com que uma saída seja afetada por um "sumário do estado" de saídas vizinhas. No caso de *max-pooling*, a informação reportada é o valor máximo numa vizinhança retangular.
- A operação de *pooling* é importante para tornar a representação aproximadamente invariante a pequenas translações da entrada, ou seja, se a entrada for transladada ligeiramente, os valores das saídas (com *pooling*) ficarão majoritariamente iguais. Isso é muito importante para fazer com que a rede neural consiga reconhecer padrões em diferentes posições de uma imagem.
- Um exemplo bastante simples da resistência do *pooling* a deslocamentos pode ser visto na Fig. 10.

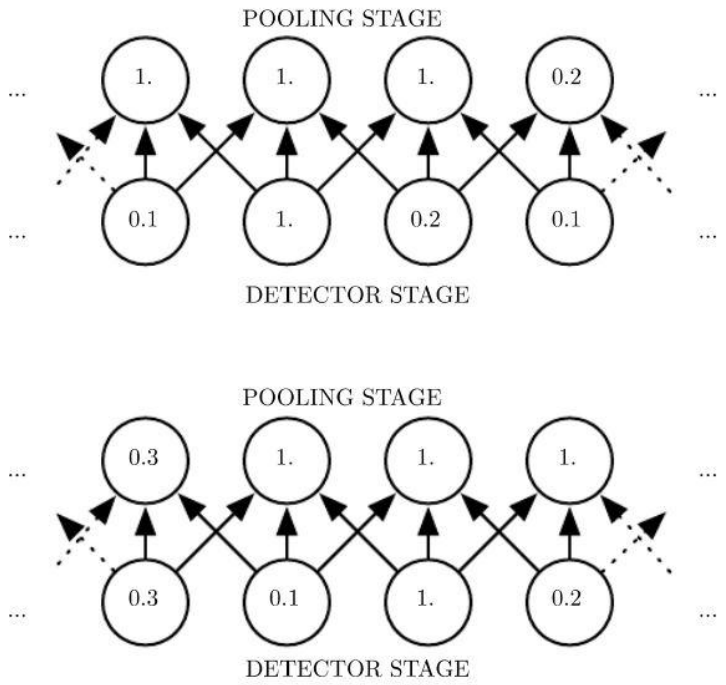

Fig. 10 – Exemplo simples da resistência ao deslocamento de um pixel.

2.1.4. Variações

- Consideremos um *kernel* tensorial **K** com elemento genérico $K_{i,j,k,l}$ que indica a intensidade da conexão entre uma unidade no canal i da saída e uma unidade no canal j da entrada, com um *offset* de k linhas e l colunas entre a unidade de saída e a de entrada. A entrada é um tensor **V** cujo elemento $V_{i,j,k}$ é o valor da

unidade de entrada no canal i, linha j e coluna k. Considerando que o resultado da convolução (sem *flip*) é um tensor $\mathbf{Z}$ de formato igual ao de $\mathbf{V}$, temos:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

- É possível pensar em pular algumas posições do *kernel* para reduzir o custo computacional, com a ressalva de que a extração de características será menos fina. De certo modo, trata-se de um *downsampling* da saída. A convolução modificada seria:

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,(j-1)s+m,(k-1)s+n} K_{i,l,m,n}$$

- O deslocamento s é conhecido como *stride* (também é possível definir *strides* para diferentes direções de percurso na estrutura de dados). A Fig. 11 ilustra a convolução com *stride* de duas formas equivalentes.

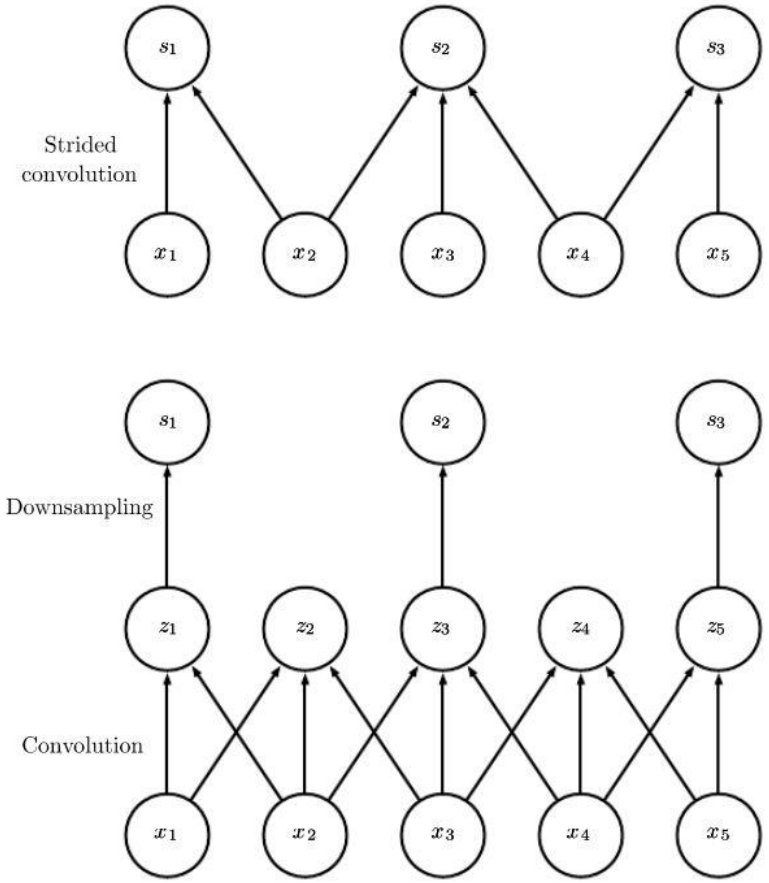

Fig. 11 – Convolução com *stride*.

2.1.5. Repassando os conceitos de CNNs

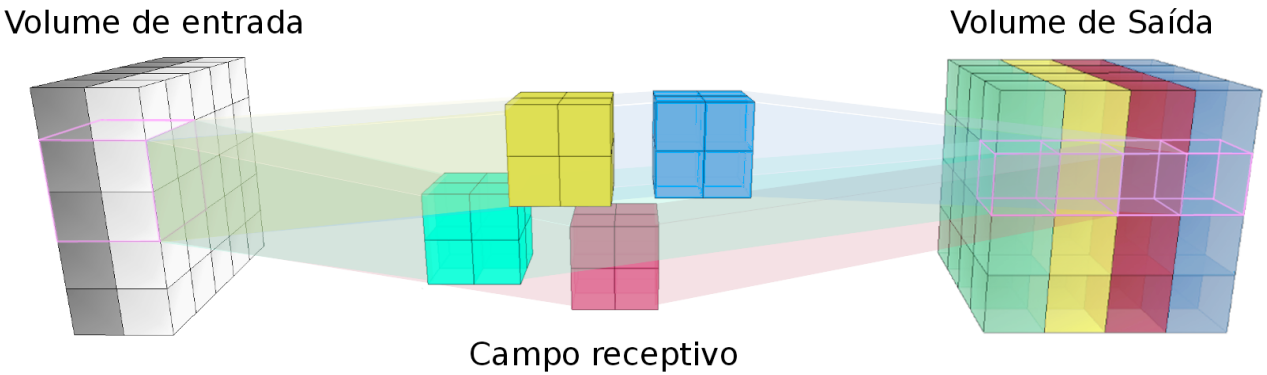

Figura. Ilustração de quatro *kernels* e os campos receptivos.

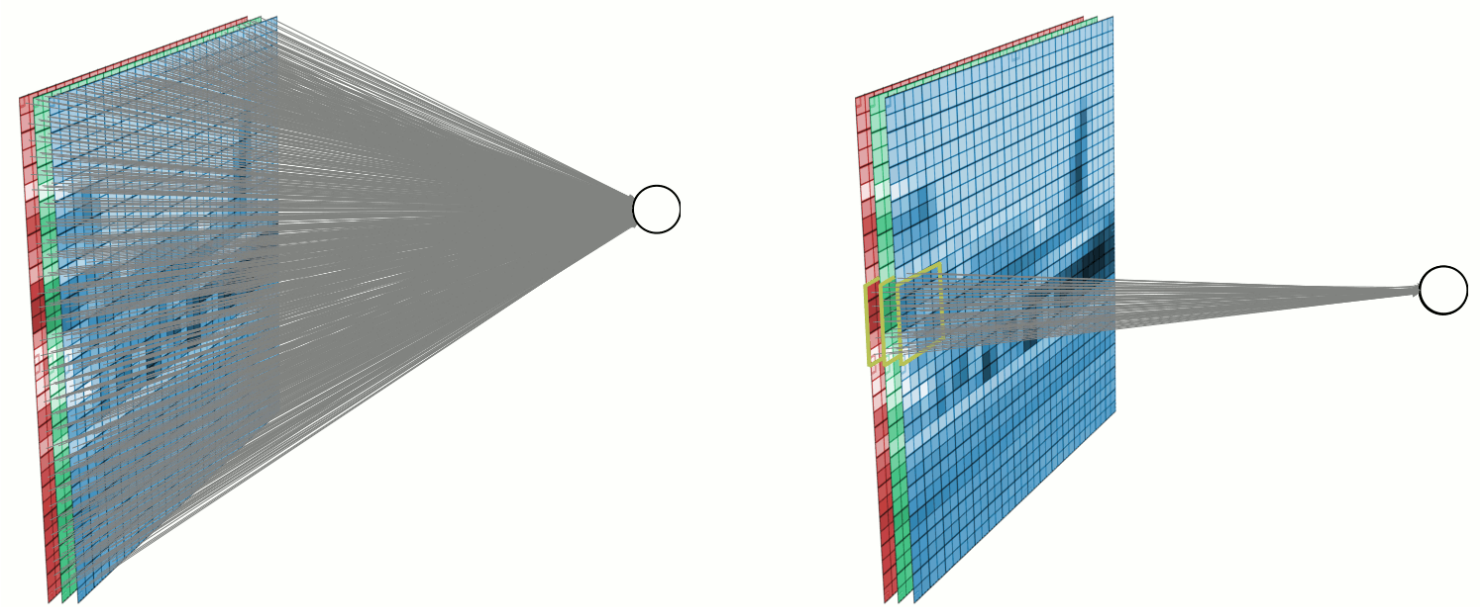

Figura. Contraste entre uma camada densa (*fully connected*) e uma convolucional.

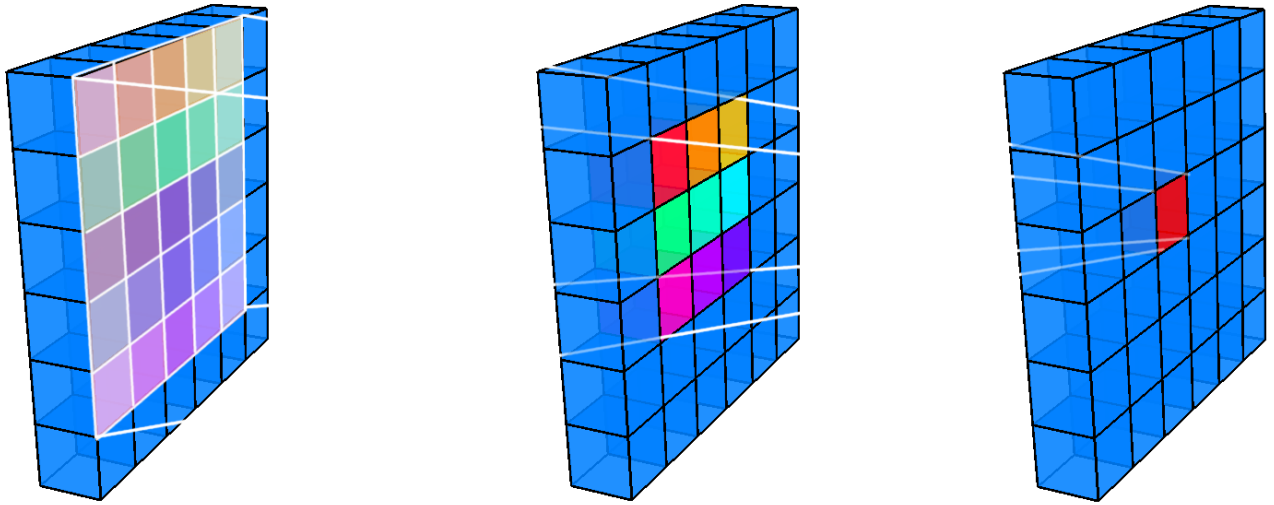

Figura. Ampliação do campo receptivo à medida que a profundidade na rede aumenta.

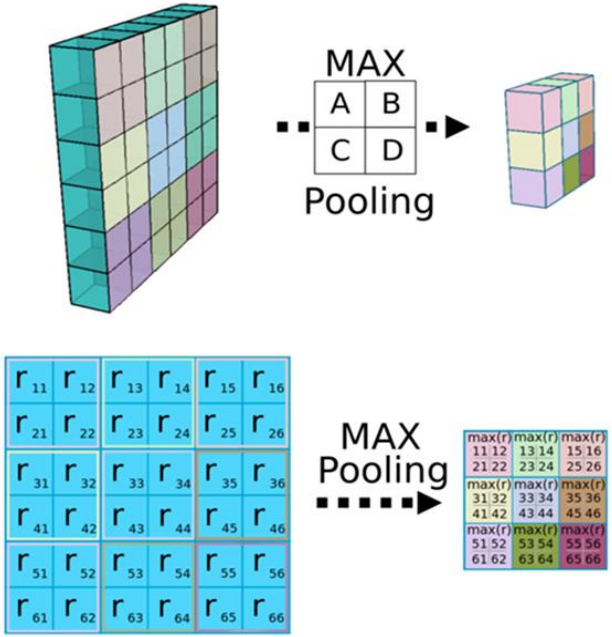

Figura. Operação da camada de *pooling*.

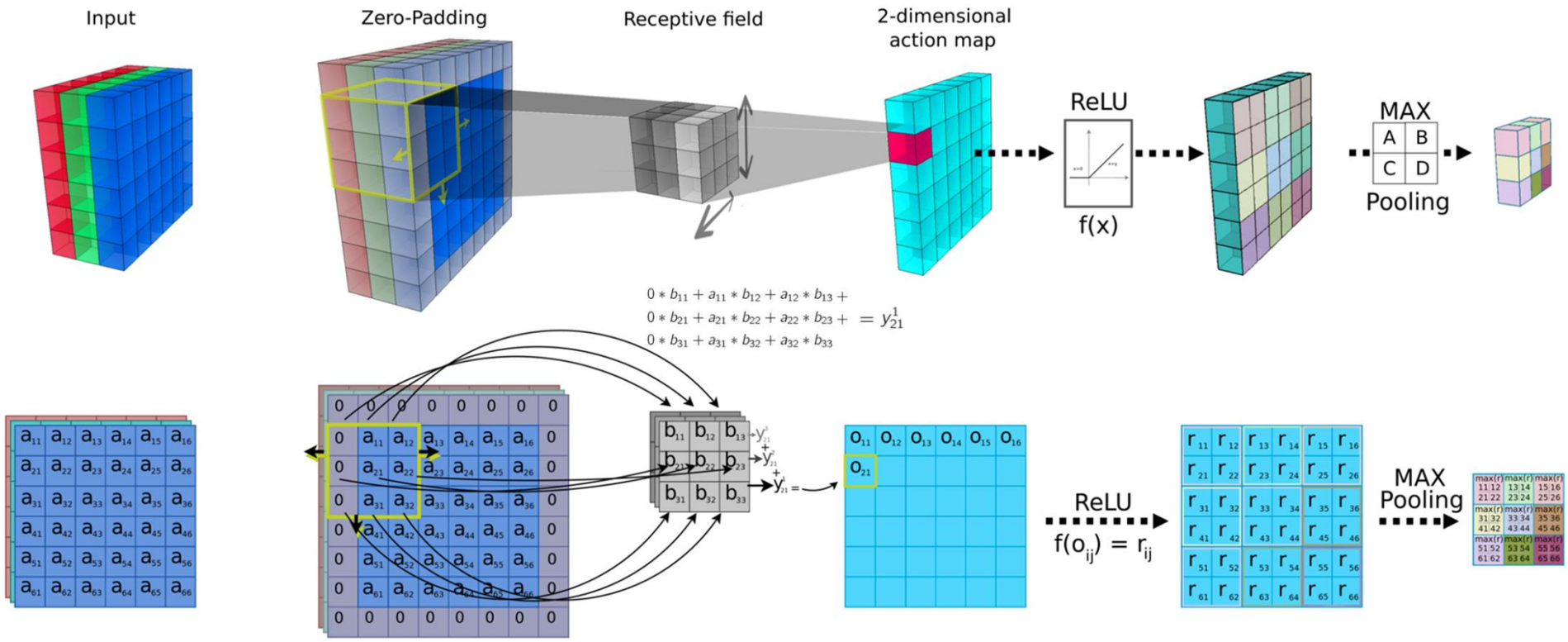

Figura. Operação típica de uma camada convolucional com ativação ReLu seguida de uma camada de *pooling*.

2.1.6. Fazendo História: Revendo a AlexNet

- O trabalho (KRIZHEVSKY ET AL., 2012) ocupa lugar de destaque na história das redes convolucionais profundas. Nele, apresenta-se uma arquitetura que

conseguiu superar os melhores resultados da época, na competição ImageNet LSVRC-2010, por uma larga margem.

- A arquitetura utilizada foi a mostrada na Fig. 12.

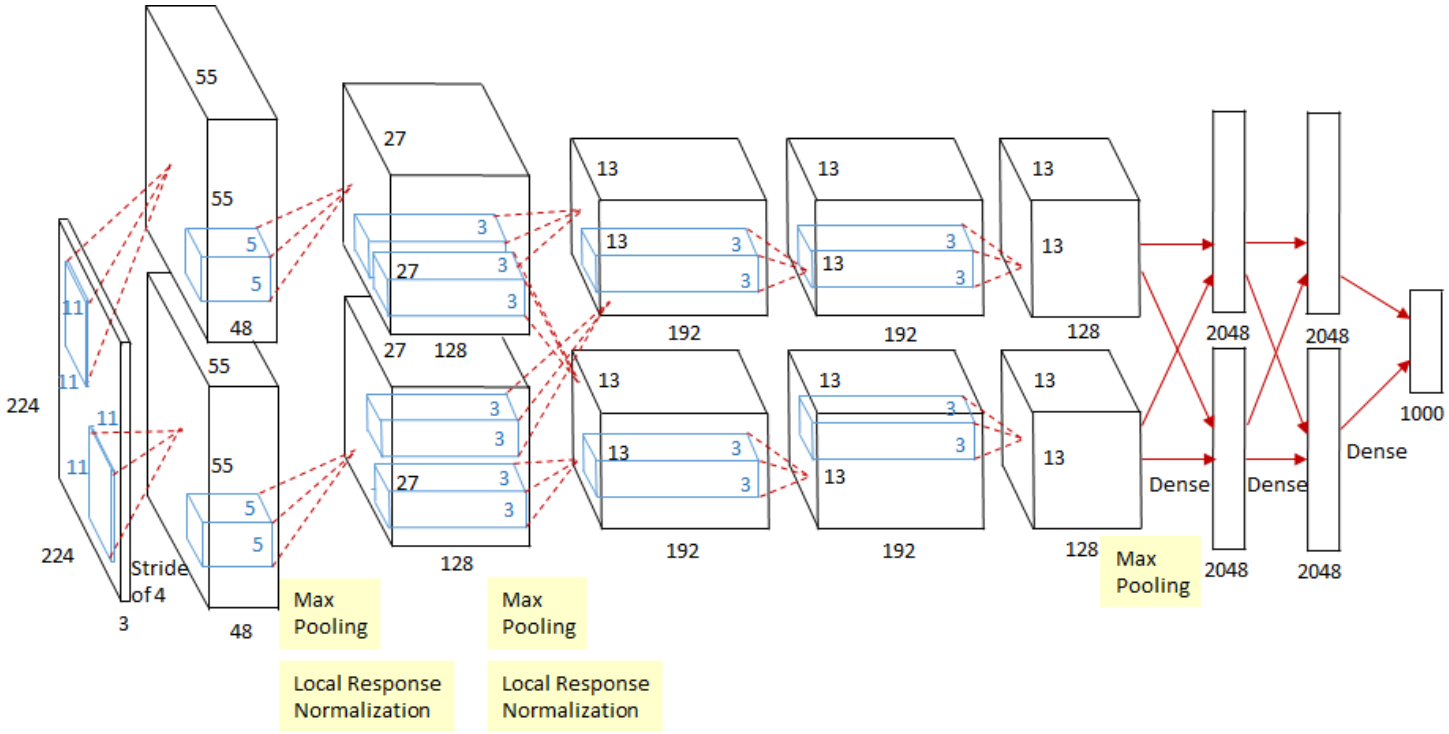

Fig. 12 – Arquitetura – AlexNet (KRIZHEVSKY ET AL., 2012).

- O conjunto de dados, da ImageNet *Large-Scale Visual Recognition Challenge* (LSVRC), possuía cerca de 1.200.000 imagens pertencentes a 1000 categorias. As

imagens foram reamostradas quando necessário para que tivessem sempre o tamanho fixo de 256 × 256 pixels.

- Interessantemente, no artigo, discutem-se as vantagens de se usar ReLUs, o que reforça seu caráter pioneiro. Em seguida, fala-se sobre a implementação computacional e se explica que houve uma divisão dos *kernels* da rede em duas GPUs, o que explica a "simetria" das partes de cima e de baixo da Fig. 12.
- De uma perspectiva global, há oito camadas com pesos sinápticos: cinco camadas convolucionais e três camadas totalmente conectadas (densas ou *fully-connected*). A saída da última camada totalmente conectada é passada a uma camada *softmax* que gera estimativas da probabilidade de que uma entrada pertença a cada uma das 1000 classes.

- A primeira camada convolucional filtra uma imagem 224 × 224 × 3† com 96 *kernels* de tamanho 11 × 11 × 3 (*stride* de 4 pixels); a segunda camada convolucional toma como resposta a saída da primeira camada e a filtra com 256 *kernels* de tamanho 5 × 5 × 48; a terceira camada tem 384 kernels de tamanho 3 × 3 × 256; a quarta camada tem 384 *kernels* de tamanho 3 × 3 × 192 e a quinta camada tem 256 *kernels* de tamanho 3 × 3 × 192. Cada camada densa tem 4096 neurônios.
- A rede tem 60 milhões de parâmetros, o que gera preocupação com sobreajuste (*overfitting*). Uma primeira estratégia para combater esse problema foi realizar *data augmentation*, ou seja, aumentar artificialmente o tamanho do conjunto de dados sintetizando novos dados a partir dos existentes. Isso foi feito extraindo aleatoriamente *patches* de 224 × 224 das imagens 256 × 256 (e reflexões

† Veremos logo a seguir por que a imagem não tem 256 × 256 como se esperava.

horizontais dos mesmos), aumentando o tamanho do conjunto por um fator de 2048. Sem esse esquema, o sobreajuste impediria o uso de uma rede profunda.

- Outra forma de realizar *data augmentation* foi por meio de análise de componentes principais no âmbito dos canais RGB das imagens.
- Para combater o sobreajuste, também foi preciso recorrer a uma estratégia de regularização muito elegante, conhecida como *dropout*. No caso, o *modus operandi* foi forçar a zero a saída de cada neurônio de camada intermediária com probabilidade igual a 0,5 (os neurônios são "*dropped out*"). "Fora de cena", esses neurônios não participam nem da etapa *forward* nem da retropropagação de erro.
- Com esse esquema de *dropout*, cada vez que se apresenta um padrão, a rede neural atua, na prática, com uma arquitetura distinta (com pesos em comum). Não é possível, destarte, "colocar todas as fichas" em co-adaptações complexas.

É preciso buscar mapeamentos robustos às variações causadas pelo sorteio dos neurônios. No artigo, *dropout* foi usado na primeira camada *fully-connected*.

- O treinamento teve por base o algoritmo de gradiente estocástico (SGD, *stochastic gradient descent*) com minibatch de 128 amostras. Usou-se um termo de momento e *weight decay*. O passo de adaptação (*learning rate*) foi igual para todas as camadas e ajustado segundo uma heurística simples baseada no erro de validação. A rede foi treinada por ~90 ciclos (épocas), num período de 5 a 6 dias, em duas GPUs NVIDIA GTX 580 3GB.
- Os resultados obtidos estão nas Figs. 13 e 14.

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

Fig. 13 – Resultados – Parte I (KRIZHEVSKY ET AL., 2012).

- Os resultados mostrados na Fig. 13, para o ILSVRC – 2010, são notáveis. Na Fig. 14, temos os resultados para a edição de 2012.

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

Fig. 14 – Resultados – Parte II (KRIZHEVSKY ET AL., 2012).

- Por fim, na Fig. 15, temos resultados que mostram a operação da rede.

Figure 4: (**Left**) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (**Right**) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Fig. 15 – Resultados – Parte III (KRIZHEVSKY ET AL., 2012).

3. Redes Recorrentes

- As redes *feedforward* que temos visto até agora são, fundamentalmente, modelos genéricos (mesmo que assaz complexos) de mapeamentos entrada – saída. Se a informação for colocada na entrada, tem-se a saída, e o mesmo acontece a qualquer instante. Em outras palavras, se tivermos uma rede desse tipo com m entradas e n saídas, ela representa não mais que uma função $F\colon\ \mathbb{R}^m \rightarrow \mathbb{R}^n$. Não mais que uma função, mesmo que esta seja fabulosamente complexa.
- Entretanto, as redes neurais biológicas possuem funcionalidades que transcendem as de simples funções. Elas, por exemplo, são capazes de *armazenar padrões*, o que se associa a seu comportamento *dinâmico*.

3.1. Sistemas Dinâmicos

- Sistemas dinâmicos são modelos muito mais amplos que mapeamentos entrada – saída. Eles podem ser considerados uma linguagem utilizada pela

matemática e pela física para expressar o comportamento de inúmeros fenômenos.

- Embora seja muito usual trabalhar com sistemas dinâmicos considerando tempo contínuo, no contexto de redes neurais, é mais interessante definir esses sistemas em tempo discreto. Usaremos a representação de estados clássica:

$$\begin{cases} \mathbf{x}(n+1) = f(\mathbf{x}(n), \mathbf{u}(n)) \\ \mathbf{y}(n) = g(\mathbf{x}(n), \mathbf{u}(n)) \end{cases}$$

sendo $\mathbf{x}(n)$ o vetor de estados, $\mathbf{u}(n)$ o vetor de entradas, $\mathbf{y}(n)$ o vetor de observações e $f(\cdot)$ e $g(\cdot)$ mapeamentos genéricos. A chave para entender por que se trata de um sistema dinâmico é que o próximo valor do vetor de estados $\mathbf{x}(n+1)$ depende do valor atual $\mathbf{x}(n)$. Isso significa que há *realimentação*, ou seja, *feedback*. Havendo *feedback*, pode haver **dinâmica** e pode haver **memória**.

3.2. Redes Neurais: Memória / Rede de Hopfield

- Redes recorrentes são usadas em dois contextos que destacamos: como memórias e como estruturas dinâmicas de processamento.
- Como memórias, o exemplo paradigmático é o modelo de memória associativa de Hopfield (HOPFIELD, 1982). Nesse modelo, a proposta era utilizar uma rede recorrente e não-linear para armazenar padrões nas características dinâmicas dessa rede. O acesso a essas memórias deveria ser feito por versões parcialmente corrompidas dos padrões guardados, ou seja, por um mecanismo associativo (endereçamento por conteúdo). A Fig. 16 ilustra uma arquitetura básica.
- A rede da Fig. 16 é um sistema dinâmico, e, portanto, a partir de uma condição inicial, o estado $\mathbf{y} = [y_1, y_2, \ldots, y_s]^T$ evolui até um atrator, que se espera que seja um ponto de equilíbrio (ponto fixo). Se conseguirmos armazenar os padrões

desejados como pontos de equilíbrio e inicializarmos a rede com versões corrompidas dos padrões, a rede pode recuperar os padrões armazenados simplesmente por meio da iteração de sua dinâmica i.e. de sua convergência. A Fig. 17 ilustra isso.

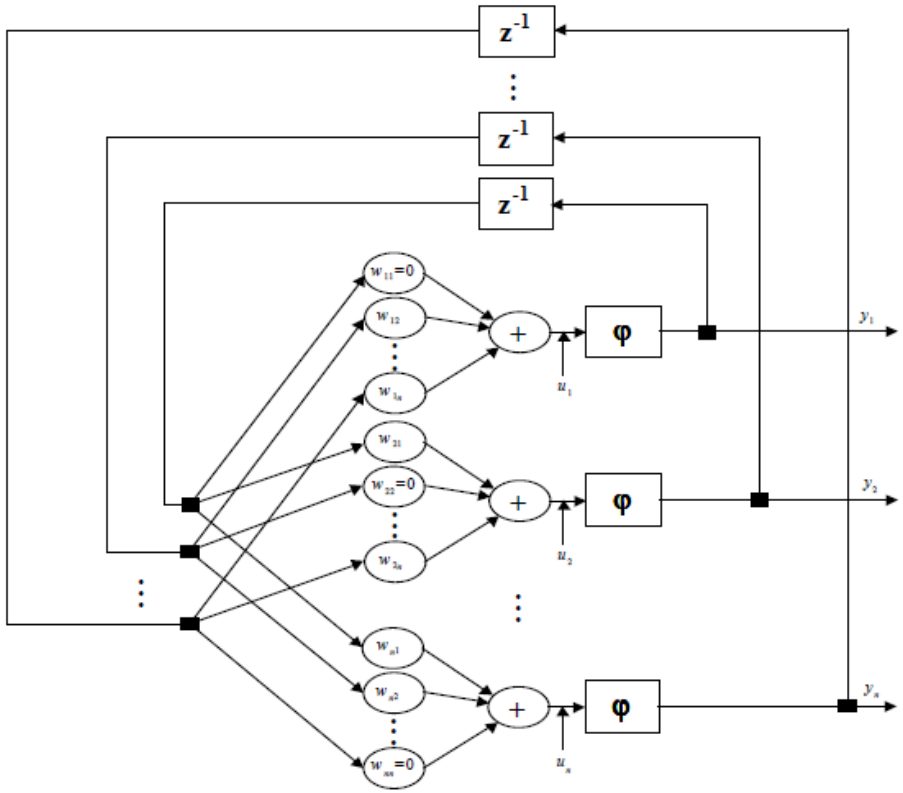

Fig. 16 – Exemplo de Rede de Hopfield.

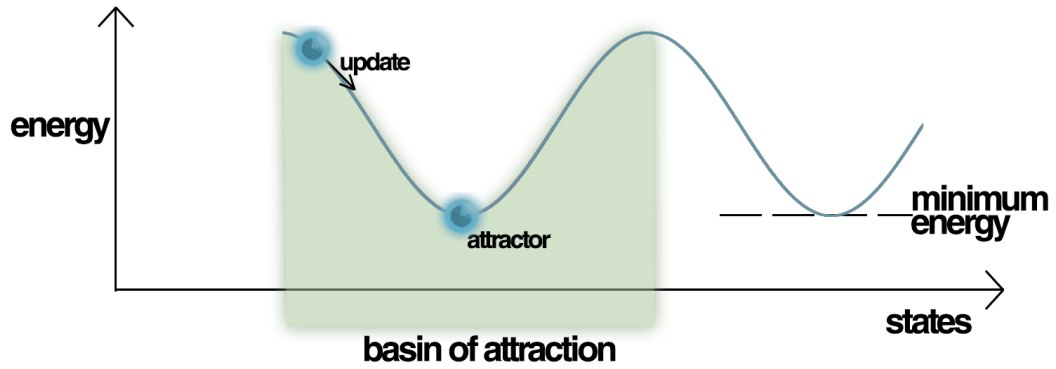

Fig. 17 – Evolução Dinâmica da Rede (*Wikipedia*).

- Pensando em termos do armazenamento e da recuperação de padrões a partir de versões parciais, poderíamos descrever o cenário como na Fig. 18.

Fig. 18 – Exemplo de Operação da Rede de Hopfield (VON ZUBEN, 2003).

- Há fórmulas fechadas para escolha dos pesos sinápticos da rede de Hopfield que armazenam certo conjunto de padrões (VON ZUBEN, 2003). Em outras palavras, o projeto e a operação da rede se dão da seguinte forma: de posse dos padrões (memórias) que se deseja armazenar, calculam-se os pesos sinápticos. Com os pesos, a rede está completa e é possível calcular, para cada configuração de suas saídas (o seu vetor de estado), os próximos valores dessas saídas. Assim, ao inicializarmos as saídas com versões corrompidas dos padrões, espera-se que a rede convirja para as memórias (pontos de equilíbrio).

3.3. Redes Neurais: Processamento Dinâmico

- No contexto do processamento da informação, as redes recorrentes, como já delineado, representam um passo adiante por permitirem à rede apresentar comportamento dinâmico. Isso pode ser decisivo quando as entradas da rede

possuem estrutura sequencial, como é o caso, por exemplo, de séries temporais ou textos (GOODFELLOW ET AL., 2016).

- Uma estrutura neural que expressa de maneira direta as equações de estado em tempo discreto tem as seguintes equações (GOODFELLOW ET AL., 2016):

$$\mathbf{a}(n) = \mathbf{b} + \mathbf{W}\mathbf{x}(n-1) + \mathbf{K}\mathbf{u}(n)$$

$$\mathbf{x}(n) = \tanh(\mathbf{a}(n))$$

$$\mathbf{s}(n) = \mathbf{c} + \mathbf{V}\mathbf{x}(n)$$

$$\mathbf{y}(n) = softmax(\mathbf{s}(n))$$

- A aplicação da metodologia do gradiente para otimização de redes recorrentes possui algumas peculiaridades ligadas à questão de estimação de derivadas ao longo do tempo (GOODFELLOW ET AL., 2016).

3.3.1. Redes Neurais com Estados de Eco

- Certas dificuldades ligadas à aplicação da metodologia do gradiente a estruturas recorrentes têm atraído grande interesse para o campo de redes neurais com estados de eco (ESNs, do inglês *echo state networks*) (JAEGER, 2010; BOCCATO, 2013).
- Essas redes, que possuem semelhanças com algumas ideias hoje pouco lembradas de Alan Turing (BOCCATO, 2013), possuem duas estruturas básicas: o *reservatório de dinâmicas* e a *camada de saída* (ou *readout*). A Fig. 19 traz uma ilustração.

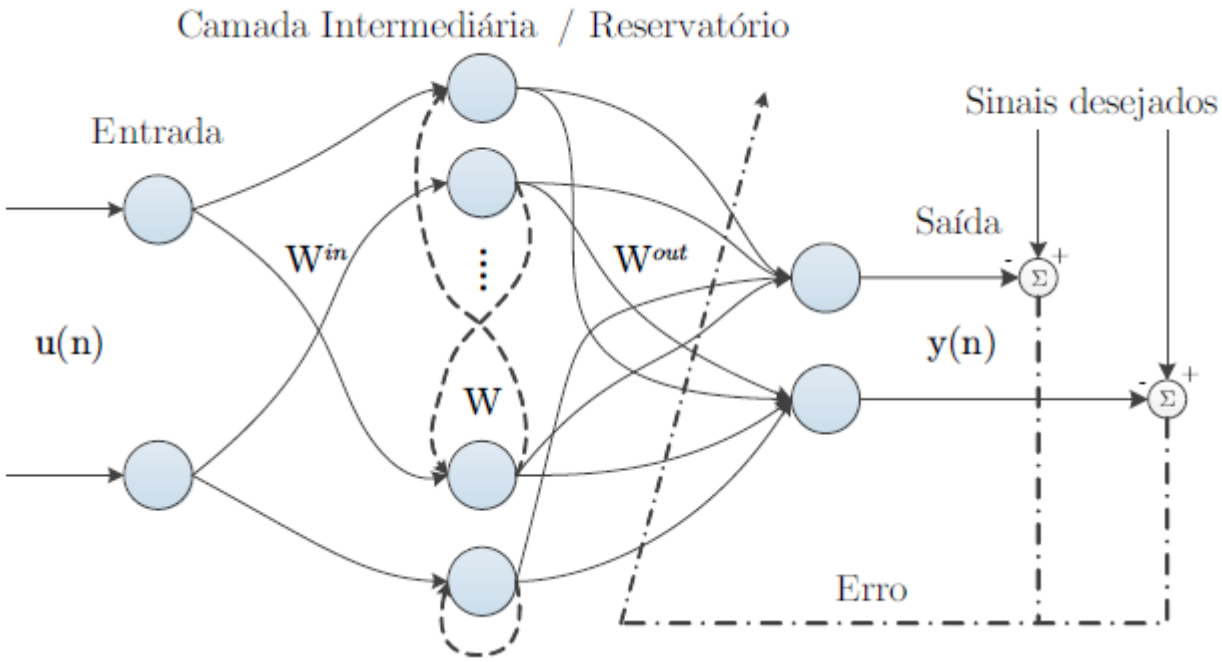

Fig. 19 – Estrutura de uma Rede Neural com Estados de Eco (BOCCATO, 2013).

- A camada intermediária, o reservatório de dinâmicas, contém os laços de realimentação que fazem da rede uma rede recorrente. O ponto fundamental é que, em redes com estados de eco, os pesos desse reservatório não são treinados de acordo com um sinal de referência, mas definidos *a priori* e mantidos fixos.
- Isso facilita imensamente o treinamento dessa rede, pois não é necessário calcular derivadas com respeito a pesos envolvidos em laços de *feedback*. Resta

apenas otimizar os pesos da camada de saída (*readout*), que engendram um modelo linear nos parâmetros e, destarte, podem ser obtidos até mesmo em forma fechada (por mínimos quadrados).

- As equações que regem a ESN são expressas, usualmente, como‡:

$$\mathbf{x}(n) = f(\mathbf{W}\mathbf{x}(n-1) + \mathbf{W}^{\text{in}}\mathbf{u}(n))$$

$$\mathbf{y}(n) = \mathbf{W}^{\text{out}}\mathbf{x}(n)$$

- Há algumas regras e princípios para escolha das matrizes de pesos, especialmente da matriz do reservatório $\mathbf{W}$. Na definição dessa matriz, é preciso levar em conta *inter alia* a propriedade de estados de eco (JAEGER, 2010).

‡ Os termos de *bias* foram omitidos por simplicidade. Eventuais não-linearidades da camada de saída, bem como *feedback* direto dessa camada, foram desconsiderados.

4. Referências bibliográficas

BOCCATO, L. Novas Propostas e Aplicações de Redes Neurais com Estados de Eco. Tese de Doutorado. Faculdade de Engenharia Elétrica e de Computação, UNICAMP, 2013.

GOODFELLOW, I., BENGIO, Y., COURVILLE, A., *Deep Learning*, MIT Press, 2016.

HOPFIELD, J., J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proceedings of the National Academy of Sciences, Vol. 79, pp. 2554 – 2558, 1982.

JAEGER, H., *The "Echo State" Approach to Analysing and Training Recurrent Neural Networks – with an Erratum Note*, Technical Report, 2010.

KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., "ImageNet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems, 2012.

LECUN, Y., BENGIO, Y., HINTON, G., "Deep Learning", Nature, Vol. 521, pp. 436 – 444, 2015.

VON ZUBEN, F., J., *Notas de Aula do Curso IA353*, 2003.