Deep Neural Networks

Transparência 01: **Pequena História dos Neurônios**

Redes Neurais artificiais tem sido estudadas desde 1940, e, como resultado, existe muita história sobre isto.

Um bom exemplo do progresso desta história é com respeito ao surgimento das funções de ativações dos neurônios. Ao longo desta história, os pesquisadores introduziram a função de ativação *threshold* e criaram as primeiras redes neurais. Isto acarretou o surgimento subsequente das funções de ativação sigmoides, então para as funções tangentes hiperbólicas e agora, recentemente, para as funções ReLU (*Rectified Linear Unit*).

Enquanto muitas literaturas correntes sugerem utilizar a função de ativação ReLU exclusivamente, nós precisamos entender as funções sigmoide e tangente hiperbólica para perceber os benefícios da ReLU.

Transparência 02: **Declínio e Sucesso das Redes Neurais (1/2)**

As Redes Neurais surgiram das cinzas do descrédito várias vezes na sua história. McCulloch, W. e Pitts, W. (1943) foram os primeiros pesquisadores a introduzirem a ideia de redes neurais artificiais. Entretanto, eles não tinham nenhum método para treinar essas redes neurais. Os programadores tiveram que fazer manualmente as matrizes de pesos destas primeiras redes neurais. Por causa deste processo ser tedioso, as redes neurais caíram em desuso pela primeira vez.

Rosenblatt, F. (1958) proveu um algoritmo de treinamento muito necessário chamado de *backpropagation*, que automaticamente criava as matrizes de pesos das redes neurais. De fato, o algoritmo *backpropagation* possui muitas camadas de neurônios que simula a arquitetura do cérebro de animais. Entretanto, o algoritmo *backpropagation* é lento, e, quanto as camadas aumentam, ele torna-se realmente muito mais lento.

Parecia que a adição do poder computacional das décadas de 80 e 90 ajudariam as redes neurais nas tarefas de aprendizagem, mas o hardware o os algoritmos de treinamento desta era não podiam treinar eficientemente redes neurais com muitas camadas, e, assim, pela segunda vez, redes neurais caíram em desuso.

Transparência 03: **Declínio e Sucesso das Redes Neurais (2/2)**

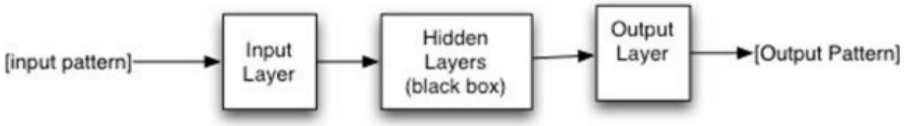
O terceiro surgimento das redes neurais ocorreram quando Hinton (2006) proveu uma nova maneira radical para treinar redes neurais profundas (*Deep Neural Networks*). Assim, o recente avanço no desenvolvimento de unidades de processamento gráficas de alta velocidade (GPUs) permitiu programadores treinarem redes neurais com três ou mais camadas e isto permitiu o resurgimento desta tecnologia.

As redes neurais *feedforward* com muitas camadas tornaram-se as redes neurais profundas (*Deep Neural Networks*). O restante deste material contém métodos, tais como, suporte a GPU para treinar *deep networks*. Nós também exploramos tecnologias relatadas às redes *deep learning*, tais como, *dropou*t, regularização e convolução. As principais aplicações de *deep learning* são em problemas de classificação e reconhecimento de imagens.

Transparência 04: **Redes Neurais São Naturalmente Classificadores de Padrões (1/2)**

Reconhecimento de padrões é a tarefa que redes neurais pode realizar facilmente. Por esta tarefa, você pode comunicar um padrão de entrada para a rede neural e ela comunica de volta para você outro padrão de saída associado ao padrão de entrada.

Figure 8: A Typical Neural Network

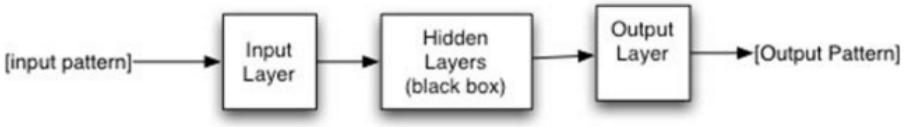


Como pode ser visto na figura acima, uma rede neural aceita um padrão de entrada e retorna outro padrão de saída. Redes Neurais operam sincronamente e retornarão uma saída somente quando ela tem uma entrada bem definida numericamente.

Transparência 05: Redes Neurais São Naturalmente Classificadores de Padrões (2/2)

Redes Neurais consistem de camadas de neurônios similares. Muitas redes tem no mínimo uma camada de entrada e uma camada de saída. O programa apresenta o padrão de entrada para a camada de entrada. Então o padrão de saída é retornado na camada de saída. O que acontece entre a camada de entrada e a camada de saída é uma caixa preta. Por caixa preta, isto significa que você não sabe exatamente como uma rede neural trabalha.

Figure 8: A Typical Neural Network



Transparência 06: **Treinamentos Supervisionado e Não-Supervisionado**

Quando é especificada a saída ideal da rede, então está se utilizando treinamento supervisionado. Se não é provida a saída ideal, então está se utilizando treinamento não-supervisionado. Treinamento supervisionado ensina a rede neural a produzir a saída ideal. Treinamento não-supervisionado usualmente ensina a rede neural localizar o dado de entrada entre um número de grupos definidos previamente, utilizando a contagem realizada pelo neurônio de saída.

Ambos treinamentos supervisionado e não-supervisionado são processos iterativos. Para o treinamento supervisionado, cada iteração de treinamento calcula como a saída atual deve se aproximar da saída ideal e expressa esse aprendizado através de um percentual de erro. Cada iteração modifica as matrizes de pesos internas da rede neural para diminuir a taxa de erro até abaixo de um percentual aceitável.

Treinamento não-supervisionado é também um processo iterativo. Entretanto, calcular o percentual de erro, neste caso, não é tão fácil. Devido ao fato de não se conhecer a saída esperada, não se pode medir quão afastada a rede neural não-supervisionada está da sua saída ideal. Desta forma, não se tem saídas ideais neste caso. Como resultado disto, o usual é treinar a rede num número ideal e fixado de iterações e tentar usar a rede depois disso. Se a rede neural necessitar de mais treinamento, o programa deverá ser capaz de fornecer isso a ela.

Transparência 07: **Dados Parcialmente Rotulados (1/2)**

Muitos algoritmos de treinamento são ou supervisionado ou não-supervisionado. Conjuntos de dados de treinamento supervisionado provêm um resultado esperado para cada item de dado. Conjuntos de dados de treinamento não-supervisionado não provêm um resultado esperado e esse resultado esperado é denominado de um rótulo ou de uma etiqueta. O problema é que muitos conjuntos de dados são uma mistura de itens de dados rotulados e de itens dados não-rotulados.

Para compreender a diferença entre dado rotulado e não-rotulado, considere o seguinte exemplo da vida real. Quando você era criança, você provavelmente viu muitos veículos nas ruas enquanto crescia. Ainda cedo em sua vida, você não conhecia se você estava vendo ou um carro, ou um caminhão ou uma van. Você simplesmente conhecia que você estava vendo algum tipo de veículo. Assim, você pode considerar essa orientação como uma parte não-supervisionada de sua jornada pela vida no aprendizado de tipos diferentes de veículos. Neste momento, você aprendeu pontos comuns das características desses veículos.

Transparência 08: **Dados Parcialmente Rotulados (2/2)**

Mais tarde em sua jornada de aprendizado, você foi dando rótulos ou etiquetas para as coisas. Enquanto você encontrava diferentes veículos, um adulto dizia para você se você estava olhando para um carro, uma caminhão ou uma van. Assim, o treinamento não-supervisonado criou seus fundamentos e o treinamento supervisionado refinou seus conhecimentos. Assim, como você pode ver, aprendizado supervisionado e não-supervisionado são muito comuns na vida real. Em sua própria maneira, aprendizado profundo (*deep learning*) faz bem ao combinar dados de aprendizado supervisionado com dados de aprendizado não-supervisionado.

Algumas arquiteturas de aprendizagem profunda lidam parcialmente com dados rotulados e inicializam os pesos da rede neural por utilizar um conjunto de treinamento inteiro sem os resultados esperados. Você pode treinar independentemente as camadas individualmente sem os rótulos. Devido ao fato que você pode treinar as camadas em paralelo, este processo pode ser escalonado. Uma vez que a fase não-supervisionada tenha inicializado esses pesos, depois disto, a fase supervisionada pode ajustá-los melhor. Esse processo em inglês é denominado de *Patially Labeled Data.*

Transparência 09: **Básico das Redes Neurais (1/22)**

Estas transparências são sobre redes neurais artificiais e como treinar, fazer consulta, estruturar e interpretar com elas.

Treinamento é o processo em que uma rede neural artificial é adaptada para realizar predições adequadas sobre dados numéricos.

Deep Learning, um relativo novo conjunto de técnicas de treinamento para redes neurais com muitas camadas, é também um tópico primário destas transparências. Ela compreende vários algoritmos que podem treinar tipos complexos de redes neurais. Com o desenvolvimento de *Deep Learning*, nós agora temos métodos efetivos para treinar redes neurais com muitas camadas.

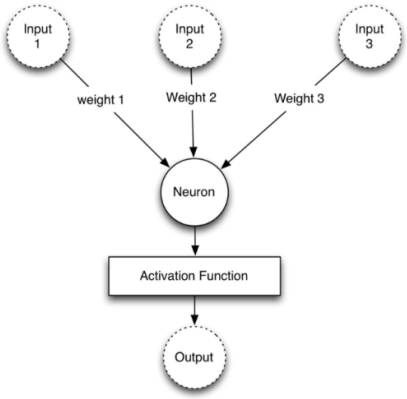
Inicialmente veremos como neurônios formam conexões de pesos sinápticos, como esses neurônios criam camadas e como as funções de ativação afetam a saída das camadas.

Transparência 10: Básico das Redes Neurais (2/22)

Um algoritmo que é denominado de rede neural tipicamente será composto de unidades individuais interconectadas, mesmo embora, essas unidades possam ou não possam ser chamadas de neurônios. De fato, o nome para a unidade de processamento de uma rede neural varia entre as várias fontes encontradas na literatura. Ela pode ser chamada de nó, neurônio ou unidade.

A Figura 1.1 mostra uma estrutura de um neurônio artificial simples.

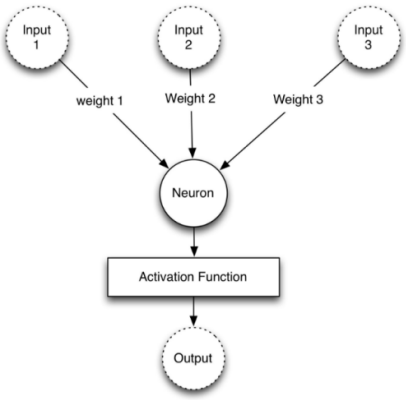
Figure 1.1: An Artificial Neuron



Transparência 11: **Básico das Redes Neurais (3/22)**

Um neurônio artificial multiplica cada uma destas entradas por um peso. Então ele adiciona essas multiplicações e passa essa soma para uma função de ativação. A equação 1.1 sumariza a saída calculada de um neurônio.

Figure 1.1: An Artificial Neuron



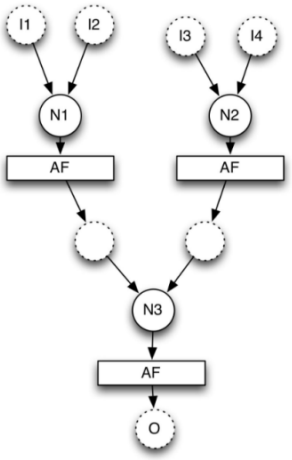
$$f(x_i, w_i) = \phi(\sum_i (w_i \cdot x_i)) \quad (1.1)$$

A equação acima, as variáveis x e w representam a entrada e os pesos do neurônio. A Figura 1.1 mostra a estrutura com somente um bloco construído. Você pode encadear conjuntamente muitos neurônios artificiais para construir uma rede neural artificial (ANN).

Transparência 12: **Básico das Redes Neurais (4/22)**

A Figura 1.2 mostra uma rede neura artificial composta com três neurônios.

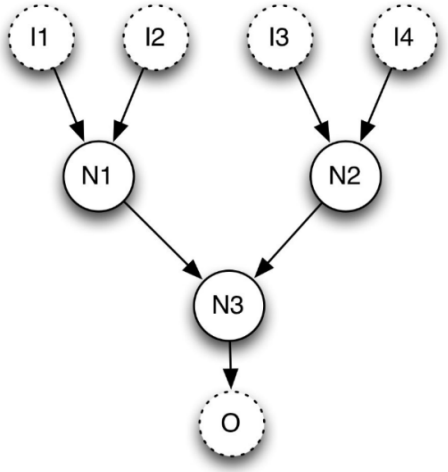
Figure 1.2: Simple Artificial Neural Network (ANN)



Transparência 13: **Básico das Redes Neurais (5/22)**

Os diagramas das redes neurais não mostram tipicamente o nível de detalhes visto na Figura 1.2. Para simplificar o diagrama, nós podemos omitir as funções de ativação e as saídas intermediárias, e este processo resulta na Figura 1.3.

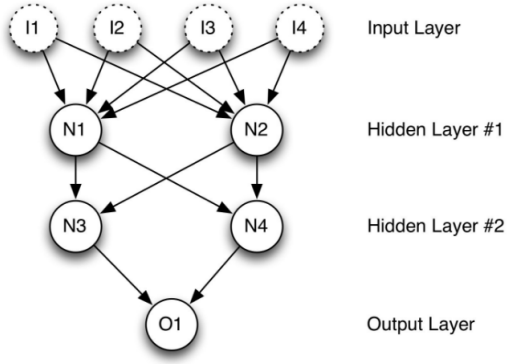
Figure 1.3: Simplified View of ANN



Transparência 14: **Básico das Redes Neurais (6/22)**

A Figura 1.4 é uma nova versão da Figura 1.3, mas que é completamente conectada e com uma camada adicional.

Figure 1.4: Fully Connected Network



Redes neurais como esta sempre terão uma camada de entrada e uma camada de saída. O número total de camadas internas determina o nome da arquitetura da rede. A rede da Figura 1.4 é uma rede com duas camadas internas. Muitas redes terão entre zero e duas camadas internas. Ao menos que você tenha implementado estratégias de aprendizagem profunda (*Deep Learning Strategies*), redes com mais de duas camadas internas são raras.

Transparência 15: **Básico das Redes Neurais (7/22)**

Você pode notar também que as camadas sempre apontam para frente, no sentido da entrada da rede para à saída da rede. Este tipo de rede neural é denominada de redes neurais *feedforward (feedforward neural network).*

Na rede da Figura 1.4 podemos ainda utilizar quatro tipos básicos de neurônios:

- Neurônios de Entrada e Saída;

- Neurônios Internos;

- Neurônios com Bias;

- *Context Neurons.*

Vejamos então, cada um deles separadamente nas próximas transparências.

Transparência 16: **Básico das Redes Neurais (8/22)**

Neurônios de Entrada e Saída. Praticamente toda rede neural tem neurônios de entrada e saída. O neurônio de entrada aceita dados do programa para a rede. O neurônio de saída provem dado processado da rede de volta para o programa. Estes neurônios de entrada e saída serão agrupados pelo programa em camadas separadas chamadas, respectivamente, de camada de entrada e de camada de saída.

Entretanto, para algumas estruturas de rede, os neurônios podem atuar como ambos de entrada e saída ao mesmo tempo. A rede neural de Hopfield é um exemplo de arquitetura de rede neural em que os neurônios são ambos de entrada e saída.

Transparência 17: **Básico das Redes Neurais (9/22)**

Neurônios Internos. Neurônios internos tem duas características importantes. Primeiro, neurônios internos somente recebem entradas de outros neurônios, tais como dos neurônios de entrada ou de outros neurônios internos. Segundo, neurônios internos somente enviam saída para outros neurônios, tais como para os neurônios de saída ou outro neurônios internos.

Uma questão comum para programadores concerne com respeito ao número de neurônios internos de uma rede neural a ser utilizado. Uma resposta a esta questão é complexa.

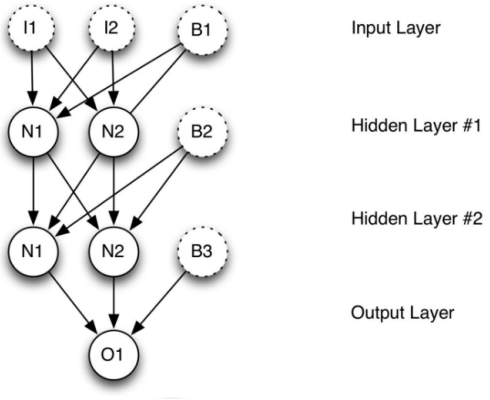
Antes de *Deep Learning*, era geralmente sugerido que para qualquer coisa mais de uma camada interna simples era excessivo (Hornik, 1991). Pesquisadores tem provado que redes neurais com uma única camada interna simples pode funcionar como um aproximador universal. Em outras palavras, esta rede seria capaz de aprender e produzir (ou aproximar) qualquer saída para qualquer entrada, dependendo, é, claro, de se utilizar neurônios internos suficientes na única camada interna.

Outra razão por que pesquisadores usarem a zombar da ideia de se utilizar camadas internar adicionais é que essas camadas impediriam o treinamento da rede neural. Treinamento se refere ao processo que determina bons valores para os pesos sinápticos. Antes dos pesquisadores introduzirem as técnicas de aprendizagem profunda, nós simplesmente não tínhamos uma maneira eficiente de treinar uma rede profunda, que são redes neurais com um grande número de camadas internas. Embora uma rede neural com apenas uma camada interna possa teoricamente aprender qualquer coisa, aprendizagem profunda facilita uma representação mais complexa de padrões de dados.

Transparência 18: **Básico das Redes Neurais (10/22)**

Neurônios de Bias. Programadores adicionam neurônios de bias para as redes neurais ajudá-los aprender padrões. Neurônios de bias funcionam como neurônios de entrada que sempre produzem o valor 1. Por causa dos neurônios de bias terem uma saída constante igual a 1, eles não são conectados com camadas anteriores. O valor de bias pode também ser um número diferente de 1. Entretanto, 1 é a ativação de bias mais comum. A Figura 1.5 mostra uma rede neural com uma camada interna, que utiliza neurônios de bias.

Figure 1.5: Network with Bias Neurons



A rede da Figura 1.5 contem três neurônios de bias. Como pode se observado, exceto para a camada de saída, todas as camadas possui um neurônio de bias. Os neurônios de bias aumentam a flexibilidade de ajuste das funções de ativação.

Transparência 19: **Básico das Redes Neurais (11/22)**

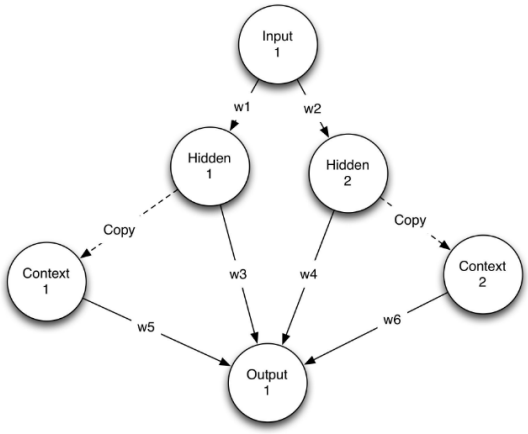
Context Neurons. *Context Neurons* são utilizados em redes neurais recorrentes. Este tipo de neurônio permite a rede neural manter estados. Como resultado, uma dada entrada pode não sempre produzir exatamente a mesma saída. Esta inconsistência é similar aos trabalhos realizados pelos cérebros biológicos.

Considere como fatores de contexto, sua resposta quando você ouve uma buzina alta. Se você ouve esse ruído quando você está atravessando a rua, você pode de sobressalto, parar a caminhada, e olhar na direção da buzina. Se você ouve a buzina enquanto você está assistindo a um filme de aventura em um cinema, você não responderá a esse estímulo da mesma maneira.

Transparência 20: **Básico das Redes Neurais (12/22)**

Context Neurons. Um tipo de rede neural chamada *Simple Recurrent Neural Network (SRN)* utiliza *context neurons*.

Figure 1.6: Context Neurons



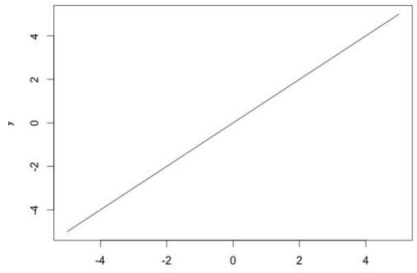
Transparência 21: **Básico das Redes Neurais (13/22)**

Funções de Ativação. Na programação de redes neurais, funções de ativação (ou funções de transferência) estabelecem contornos para a saída dos neurônios. Redes Neurais podem usar muitas funções de ativação diferentes. Nós discutiremos as funções de ativação mais comuns nas próximas transparências.

Equation 1.2: Linear Activation Function

$$\phi(x) = x$$

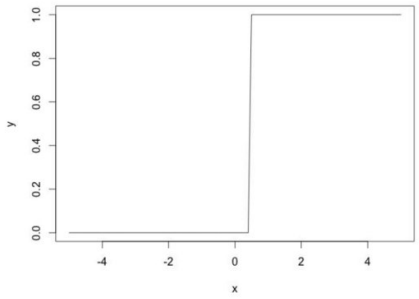
Figure 1.7: Linear Activation Function



Equation 1.3: Step Activation Function

$$\phi(x) = \begin{cases} 1, \text{if } x \geq 0.5. \\ 0, \text{otherwise.} \end{cases}$$

Figure 1.8: Step Activation Function

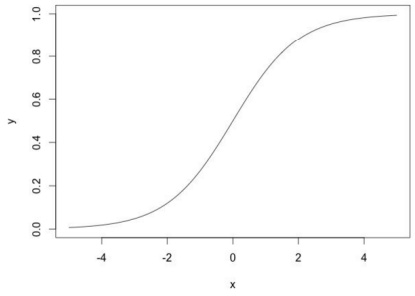


Transparência 22: **Básico das Redes Neurais (14/22)**

Equation 1.4: Sigmoid Activation Function

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

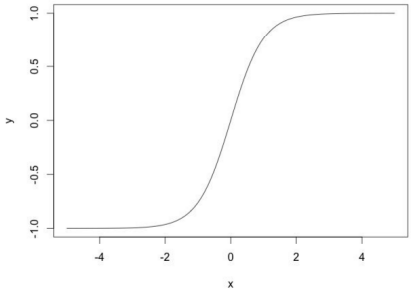
Figure 1.9: Sigmoid Activation Function



Equation 1.5: Hyperbolic Tangent Activation Function

$$\phi(x) = \tanh(x)$$

Figure 1.10: Hyperbolic Tangent Activation Function



***Funções de* Ativação**. Escolher uma função de ativação para sua rede neural é uma consideração importante, pois ela pode afetar como você deve formatar os dados de entrada.

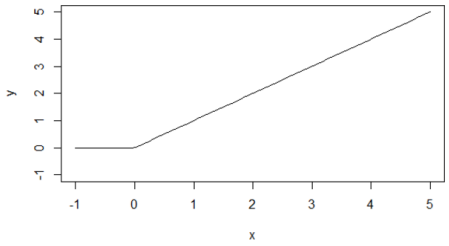
Transparência 23: **Básico das Redes Neurais (15/22)**

Funções de Ativação ReLU (Rectified Linear Units). Introduzida em 2000 por Teh & Hinton, a Unidade Linear Rectificada (ReLU) tem sido muito rapidamente adotada pela comunidade científica. Anterior ao advento da função de ativação ReLU, a função tangente hiperbólica era geralmente a função de ativação mais utilizada. Atualmente muitos pesquisadores contemporâneos recomendam a função ReLU, devido ao superior resultados de treinamento. Como resultado disto, recomenda-se utilizar a função ReLU nas camadas internas das redes neurais e as funções de ativação *softmax* ou linear na camada de saída das redes neurais. A função *softmax* será vista logo em seguida.

Equation 1.6: Rectified Linear Unit (ReLU)

$$\phi(x) = \max(0, x)$$

Figure 1.11: ReLU Activation Function



Transparência 24: **Básico das Redes Neurais (16/22)**

Funções de Ativação ReLU (Rectified Linear Units). Nós agora examinaremos porque a função de ativação ReLU tipicamente executa tarefas melhores do que outras funções de ativação, para as camadas internas das redes neurais. Parte do aumento da performance é devido ao fato de que a função de ativação ReLU é uma função linear não-saturada.

Ao contrário das funções de ativação sigmoide/logística ou tangente hiperbólica, a função ReLU não satura, respectivamente, nem em [0,1] e nem em [-1, 1]. A saturação de uma função de ativação move em alguma direção e eventualmente atinge um valor limite. Por exemplo, a função de ativação tangente hiperbólica satura em -1 quando a variável x decresce e satura em +1 quando a variável x aumenta.

Algumas pesquisas atuais recomendam ou estabelecem que as camadas internas de nossas redes neurais deveriam usar as funções de ativação ReLU. As razões para a superioridade das funções ReLU sobre as funções de ativação convencionais é apresentado na próxima transparência. O problema principal é que no algoritmo de treinamento da retropropagação (*Backpropagation Training*) os neurônios convencionais (logsig e tanh) tem suas derivadas facilmente saturadas em zero, colocando o algoritmo rapidamente em um mínimo local e não em um mínimo global.

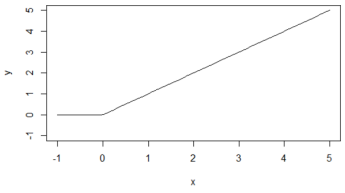
Transparência 25: **Básico das Redes Neurais (17/22)**

A equação 6.11 mostra a função matemática da derivada da função ReLU. Estritamente falando-se, a função ReLU não tem derivada em $x = 0$, pois como pode ser observado na Equação 1.6, trata-se este de um ponto de descontinuidade no valor da derivada. Entretanto, por causa de convenção, o valor zero para o gradiente é utilizado quando $x = 0$.

Equation 1.6: Rectified Linear Unit (ReLU)

$$\phi(x) = \max(0, x)$$

Figure 1.11: ReLU Activation Function



Equation 6.11: Derivative of the ReLU Activation Function

$$\frac{dy}{dx}\phi(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Redes neurais profundas com neurônios sigmoide e tangente hiperbólico podem ter dificuldade de treinamento utilizando o algoritmo da retropropagação. Vários fatores causam esta dificuldade. A anulação do gradiente é uma das causas mais comuns.

Transparência 26: **Básico das Redes Neurais (18/22)**

A Figura 6.3 mostra a função tangente hiperbólica, juntamente com sua função de derivada. A Figura 6.3 mostra que quando a função tangente hiperbólica (linha preta) satura em -1 e 1, a derivada da tangente hiperbólica (linha vermelha) anula-se em zero. As funções de ativação sigmoide e tangente hiperbólica tem ambas este mesmo problema, mas a função ReLU não. A Figura 6.4 mostra o mesmo gráfico para a função de ativação sigmoide com a anulação de sua derivada nos dois extremos.

Figure 6.3: Tanh Activation Function & Derivative

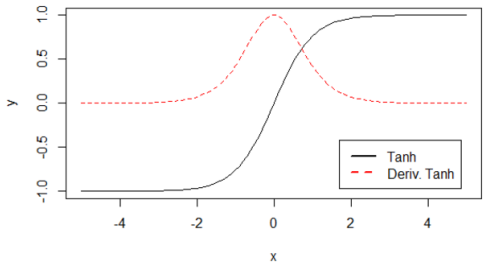
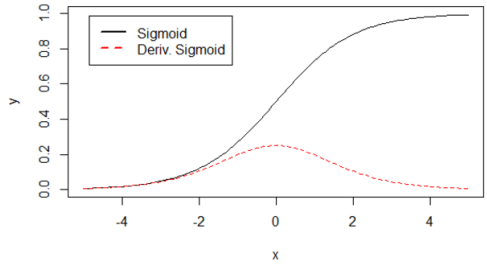


Figure 6.4: Sigmoid Activation Function & Derivative



Transparência 27: **Básico das Redes Neurais (19/22)**

Função de Ativação Softmax. A última função de ativação que examinaremos é a função de ativação *softmax*. Juntamente com a função de ativação linear, *softmax* é usualmente utilizada na camada de saída de uma rede neural. Veja abaixo, um exemplo de aplicação desta função de ativação.

Equation 1.7: The Softmax Function

$$\phi_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

```
Neuron 1: setosa: 0.9
Neuron 2: versicolour: 0.2
Neuron 3: virginica: 0.4
```

```
[0.9,0.2,0.4]

[0.47548495534876745 , 0.2361188410001125 , 0.28839620365112]

sum=exp(0.9)+exp(0.2)+exp(0.4)=5.17283
056695839
j0= exp(0.9)/sum = 0.47548495534876745
j1= exp(0.2)/sum = 0.2361188410001125
j2= exp(0.4)/sum = 0.28839620365112
```

Transparência 28: **Básico das Redes Neurais (20/22)**

Função de Ativação Softmax. A função *softmax* é utilizada numa rede neural que realiza *classificação de padrões*. O neurônio que tiver o valor maior, reivindica, a entrada da rede, como membro de sua classe. Por causa disso, essa função é um método preferível. A função de ativação *softmax* força a saída da rede neural representar a *probabilidade,* em que a, respectiva entrada, pertence em cada uma das classes. Sem a função *softmax*, os neurônios de saída são simplesmente valores numéricos, com o valor mais alto indicando a classe vencedora.

Para ver como a função de ativação *softmax* é utilizada, o exemplo da transparência anterior, tratou de um problema comum de classificação de padrões. Este exemplo tratou da classificação dos diferentes tipos das íris das flores. Quando você fornece as medidas de uma flor, a função *softmax* permite à rede neural dar para você a probabilidade dessas medidas pertencerem, cada uma delas, em uma entre três espécies distintas, a saber, *setona, virginica* ou *versicolour.*

Transparência 29: **Básico das Redes Neurais (21/22)**

Função de Ativação Softmax. Para classificar os padrões de entrada em uma das três espécies, você necessitará de um neurônio para cada uma das três espécies. Os neurônios de saída não identificam automaticamente a probabilidade de cada uma das três espécies. Portanto, é desejável fornecer probabilidades que soma no total 1 unidade ou 100%. Para garantir esse probabilidade é que se utiliza a função *softmax*.

É importante observar, que quando *softmax* é a função de ativação utilizada na saída da rede, a saída de cada neurônio da camada de saída depende dos outros neurônios de saída.

Equation 1.7: The Softmax Function

$$\phi_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}$$

```
Neuron 1: setosa: 0.9
Neuron 2: versicolour: 0.2
Neuron 3: virginica: 0.4
```

```
[0.9,0.2,0.4]

[0.47548495534876745 , 0.2361188410001125 , 0.28839620365112]

sum=exp(0.9)+exp(0.2)+exp(0.4)=5.17283
056695839
j0= exp(0.9)/sum = 0.47548495534876745
j1= exp(0.2)/sum = 0.2361188410001125
j2= exp(0.4)/sum = 0.28839620365112
```

Transparência 30: **Básico das Redes Neurais (22/22)**

Nestas últimas 20 transparências, nós vimos que uma rede neural é composta de neurônios, camadas e funções de ativação. Fundamentalmente, os neurônios em uma rede neural podem ser de natureza de entrada, internos ou de saída. Neurônios de entrada e saída passam informação, respectivamente, para dentro e para fora das redes neurais. Neurônios internos ocorrem entre os neurônios de entrada e de saída e ajudam no processo de informação.

As funções de ativação escalam a saída de um neurônio. Nós também introduzimos várias funções de ativação. As duas funções de ativação mais comuns são as sigmoides e as tangentes hiperbólicas.

Transparência 31: **Máquinas de Boltzmann (1/6)**

Existem duas grandes categorias de redes neurais com aprendizagem não-supervisionada: a Rede Neural de Hopfield e a Máquina de Boltzmann. Estes tipos de redes neurais são semelhantes, pois ambas utilizam uma ***função de energia*** durante seu processo de treinamento. A função de energia mede a quantidade de energia na rede. Enquanto o treinamento progride, a energia deve decrescer enquanto a rede aprende.

Embora nenhuma desta redes neurais clássicas é utilizada extensivamente em aplicações modernas de Inteligência Artificial (IA), ambas são fundamentais para aprender algoritmos mais modernos.

A Máquina de Boltzmann forma o fundamentos das *Deep Belief Neural Network* (DBNN), que é um dos algoritmos fundamentais de aprendizagem profunda (*deep learning*).

É importante ter alguma ideia de como essas redes convergem para um estado estável. Por exemplo, você pode calcular um valor de energia para as redes de Hopfield. Este valor decresce enquanto a rede se move para um estado mais estável. *Para avaliar a estabeilidade da rede de Hopfield, você pode utilizar a função de energia.*

Transparência 32: **Máquinas de Boltzmann (2/6)**

Hinton & Sejnowski (1985) foram os primeiros a introduzirem as Máquinas de Boltzmann, mas este tipo de rede neural não tem apreciado uso difundido até recentemente. Um tipo especial de Máquina de Boltzmann, A Máquina de Boltzmann Restrita (RBM), é uma das tecnologias fundacional da aprendizagem profunda e das *Deep Belief Neural Network* (DBNN).

A Máquina de Boltzmann é essencialmente uma rede neural de duas camadas completamente conectada. Nós nos referimos a essa camadas de camada visual e camada interna. A camada visual é análoga a camada de entrada de uma rede *Multi-Layer Perceptrons* (MLP). Apesar do fato de que uma Máquina de Boltzmann tem uma camada interna, ela funciona mais como uma camada de saída.

A Máquina de Boltzmann não tem camada interna entre as camadas de entrada e de saída.

Transparência 33: **Máquinas de Boltzmann (3/6)**

A Figura 3.3 mostra uma estrutura muito simples de uma Máquina de Boltzmann. A Máquina de Boltzmann da Figura 3.3 tem três neurônios internos e quatro neurônios visíveis. A Máquina de Boltzmann é completamente conectada porque cada neurônio tem uma conexão para todos os outros neurônios. Entretanto, nenhum neurônio é conectado consigo mesmo.

Figure 3.3: Boltzmann Machine

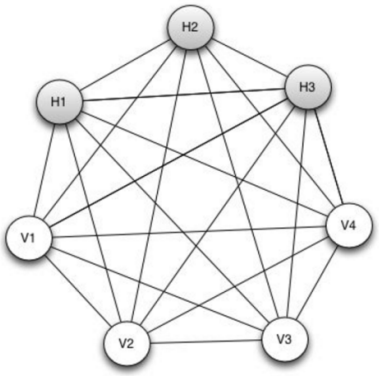
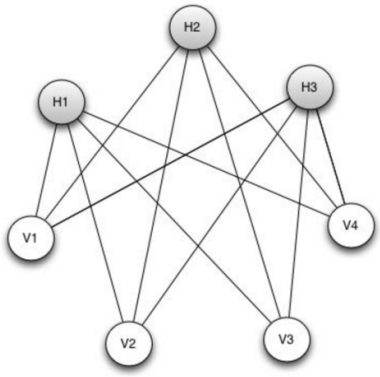


Figure 3.4: Restricted Boltzmann Machine (RBM)



A Figura 3.3 apresenta um esquema simplificado de uma Máquina de Boltzmann Restrita (RBM). A RBM não é completamente conectada. Cada os neurônio interno é conectado com todos os neurônios visíveis. Entretanto, não há nenhuma conexão entre si dos neurônios internos nem entre si dos neurônios visíveis.

Transparência 34: **Máquinas de Boltzmann (4/6)**

As redes neurais de Hopfield e as Máquinas de Boltzmann têm duas finalizadas: 1) resolver problemas de otimização, por exemplo, o problema do caixeiro viajante e 2) funcionarem como mapa de memórias (reconhecimento de padrões).

Tanto as redes de Hopfield como as Máquinas de Boltzmann admitem somente estados binários, ou zero ou um. Embora haja alguma pesquisa sobre Máquinas de Boltzmann capazes de atribuir números decimais para os neurônios, aproximadamente toda pesquisa sobre Máquina de Boltzmann é centrada em unidades binárias.

Embora a Máquina de Boltzmann e as redes neurais de Hopfield possuam algumas características em comum, existem várias diferenças importantes entre elas:

a) as redes de Hopfield sofrem de certos reconhecimentos de padrões falsos;

b) as Máquinas de Boltzmann podem armazenar uma maior capacidade de padrões do que as redes de Hopfield;

c) as Máquinas de Boltzmann podem ser empilhadas para formar camadas.

Transparência 35: **Máquinas de Boltzmann (5/6)**

Principio de funcionamento das Máquinas de Boltzmann para problemas de otimização:

Para criar uma Máquina de Boltzmann que possa fornecer uma solução para o problema do caixeiro viajante (Traveling Salesman Problem), o programa deve alinhar os pesos e bias de uma tal maneira que permita os estados dos neurônios da Máquina de Boltzmann estabilizarem em um ponto que minimize a distância total entre as cidades. Este objetivo é alcançado com um algoritmo que minimize a função de energia.

Transparência 36: **Máquinas de Boltzmann (6/6)**

Rede Neural de Hopfield. É um tipo simples de rede neural que pode reconhecer padrões (versão discreta da rede) ou resolver problemas de otimização (versão contínua da rede). Você deve criar uma função especial de energia para cada tipo de problema de otimização que requer a rede neural de Hopfield. Por causa desta qualidade, é preferível utilizar outras técnicas de otimização mais simples, tais como, têmpera simulada, algoritmos genéticos, enxame de partículas, etc.

As Máquinas de Boltzmann. São uma arquitetura de rede neural que divide muitas características com a rede neural de Hopfield. Entretanto, ao contrário da rede de Hopfield, você pode empilhar as Máquinas de Boltzmann para formar as *Deep Belief Neural Network (DBNN)*. Este empilhamento habilita as Máquinas de Boltzmann funcionarem como um papel central na implementação das DBNN, ou seja, a base da aprendizagem profunda.

As Redes Neurais Feedforward. Estas redes são consideradas como sendo as mais populares dentro do campo das redes neurais artificiais. Dentro desta classe podemos citar, as redes *Multi-Layer Perceptrons (MLP)*, as redes *Radial Basis Functions (RBF)* e as redes de *Wavelets*. Novos algoritmos de treinamento, novos tipos de camadas, novas funções de ativação e outras inovações permitiram que as clássicas redes neurais *feedforward* poderem ser utilizadas também em aprendizagem profunda.

Transparência 37: Deep Learning (1/40)

Aprendizagem Profunda (*Deep Learning*) é relativamente um novo avanço em programação de redes neurais artificiais e representa uma maneira para treinar redes neurais profundas. Essencialmente, qualquer rede neural com *mais de duas camadas* é profunda. A habilidade para criar redes neurais profundas existe desde que Pitts (1943) introduziu o *perceptron* de múltiplas camadas. Entretanto, não se tem sido capaz de efetivamente treinar redes neurais profundas até Hinton (1984) tornar-se o primeiro pesquisador a treinar com sucesso estas redes neurais mais complexas.

Transparência 38: Deep Learning: Artigo Histórico Bastante Importante (2/40)

Deep Learning in Neural Networks: An Overview

Technical Report IDSIA-03-14 / arXiv:1404.7828 v4 [cs.NE] (88 pages, 888 references)

Jürgen Schmidhuber
The Swiss AI Lab IDSIA
Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
University of Lugano & SUPSI
Galleria 2, 6928 Manno-Lugano
Switzerland

8 October 2014

Transparência 39: Deep Learning: Principais Tipos de Redes Neurais Profundas (3/40)

Deep Learning está ganhando mais e mais popularidade devido ao seu sucesso alcançado em várias aplicações, tais, como, em Processamento de Linguagem Natural (*Natural Language Processing*, NLP), Reconhecimento de Imagens e outros paradigmas de Aprendizado de Máquinas (*Machine Learning*, ML).

Existem Três abordagens convencionais que formam as bases para a compreensão teórica adequada de *Deep Learning*:

1. *Convolutional Neural Networks (CNNs)* proposta por Lecun*;*

2. *Deep Belief Networks (DBNs)* proposta por Hinton (2006)*;*

3. *Stacked Auto-encoders* proposta por Bengio (ao redor de 2007).

Transparência 40: ***Convolutional Neural Networks (CNNs)*** **proposta por Lecun (4/40)**

LeCun, Y. (1985). Une procédure d'apprentissage pour réseau à seuil asymétrique. *Proceedings of Cognitiva 85, Paris*, pages 599–604.

LeCun, Y. (1988). A theoretical framework for back-propagation. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28, CMU, Pittsburgh, Pa. Morgan Kaufmann.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Back-propagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990a). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 396–404. Morgan Kaufmann.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990b). Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann.

LeCun, Y., Muller, U., Cosatto, E., and Flepp, B. (2006). Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems (NIPS 2005)*.

LeCun, Y., Simard, P., and Pearlmutter, B. (1993). Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors. In Hanson, S., Cowan, J., and Giles, L., editors, *Advances in Neural Information Processing Systems (NIPS 1992)*, volume 5. Morgan Kaufmann Publishers, San Mateo, CA.

Transparência 41: ***Deep Belief Networks (DBNs)*** **proposta por Hinton (2006) (5/40)**

Hinton, G. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial intelligence*, 40(1):185–234.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comp.*, 14(8):1771–1800.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158–1160.

Hinton, G. E., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012a). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97.

Hinton, G. E. and Ghahramani, Z. (1997). Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society* **B**, 352:1177–1190.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hinton, G. E. and Sejnowski, T. E. (1986). Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing*, volume 1, pages 282–317. MIT Press.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. Technical Report arXiv:1207.0580.

Hinton, G. E. and van Camp, D. (1993). Keeping neural networks simple. In *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam*, pages 11–18. Springer.

Transparência 42: *Stacked Auto-encoders* proposta por Bengio (6/40)

Bengio, Y. (1991). *Artificial Neural Networks and their Application to Sequence Recognition*. PhD thesis, McGill University, (Computer Science), Montreal, Qc., Canada.

Bengio, Y. (2009). *Learning Deep Architectures for AI. Foundations and Trends in Machine Learning, V2(1)*. Now Publishers.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 19 (NIPS)*, pages 153–160. MIT Press.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Transparência 43: **Deep Learning: Alguns Termos Técnicos Importantes (7/40)**

Alguns termos técnicos relacionados com a tecnologia de aprendizagem profunda são:

- *Partially Labeled Data* (Dados Parcialmente Rotulados);

- *Rectified Linear Units* (ReLU) (Unidades Lineares Retificadas);

- *Dropout* (cair fora);

- *Deep Belief Neural Networks* (Redes Neurais de Crença Profunda);

- *Convolutional Neural Networks* (Redes Neurais Convolucionais) & *Dropout* (cair fora);

- *Constrative Divergence* (Divergência Constrativa);

- *Gibb's Sampling* (Amostragem de Gibb's);

- *GPU Training* (Treinamento de GPU);

- *Stacked Auto-encoders* (Auto-Codificadores Empilhados).

Transparência 44: **Deep Learning: Partially Labeled Data (8/40)**

Muitos algoritmos de treinamento são ou supervisionado ou não-supervisionado. Conjuntos de dados de treinamento supervisionado provêm um resultado esperado para cada item de dado. Conjuntos de dados de treinamento não-supervisionado não provêm um resultado esperado e esse resultado esperado é denominado de um rótulo ou de uma etiqueta. O problema é que muitos conjuntos de dados são uma mistura de itens de dados rotulados e de itens dados não-rotulados.

Para compreender a diferença entre dado rotulado e não-rotulado, considere o seguinte exemplo da vida real. Quando você era criança, você provavelmente viu muitos veículos nas ruas enquanto crescia. Ainda cedo em sua vida, você não conhecia se você estava vendo ou um carro, ou um caminhão ou uma van. Você simplesmente conhecia que você estava vendo algum tipo de veículo. Assim, você pode considerar essa orientação como uma parte não-supervisionada de sua jornada pela vida no aprendizado de tipos diferentes de veículos. Neste momento, você aprendeu pontos comuns das características desses veículos.

Transparência 45: **Deep Learning: Partially Labeled Data (9/40)**

Mais tarde em sua jornada de aprendizado, você foi dando rótulos ou etiquetas para as coisas. Enquanto você encontrava diferentes veículos, um adulto dizia para você se você estava olhando para um carro, uma caminhão ou uma van. Assim, o treinamento não-supervisonado criou seus fundamentos e o treinamento supervisionado refinou seus conhecimentos. Assim, como você pode ver, aprendizado supervisionado e não-supervisionado são muito comuns na vida real. Em sua própria maneira, aprendizado profundo (*deep learning*) faz bem ao combinar dados de aprendizado supervisionado com dados de aprendizado não-supervisionado.

Algumas arquiteturas de aprendizagem profunda lidam parcialmente com dados rotulados e inicializam os pesos da rede neural por utilizar um conjunto de treinamento inteiro sem os resultados esperados. Você pode treinar independentemente as camadas individualmente sem os rótulos. Devido ao fato que você pode treinar as camadas em paralelo, este processo pode ser escalonado. Uma vez que a fase não-supervisionada tenha inicializado esses pesos, depois disto, a fase supervisionada pode ajustá-los melhor. Esse processo em inglês é denominado de *Patially Labeled Data.*

Transparência 46: **Deep Learning: Rectified Linear Units (ReLU) (10/40)**

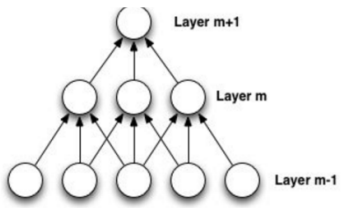
A função ReLU tem tornado-se a função de ativação *standard* para as camadas internas de redes neurais profundas. Entretanto, a máquina de Boltzman restrita (RBM) é o padrão para *Deep Belief Neural Network (DBNN).*

Em adição às funções de ativação ReLU para as camadas internas, redes neurais profundas utilizarão uma função de ativação linear (regressão) ou softmax (classificação de padrões) para a camada de saída, dependendo se a rede neural suportará um problema de regressão ou um problema de classificação de padrões.

Transparência 47: **Deep Learning: Convolutional Neural Networks (11/40)**

Convolução é uma importante tecnologia que é frequentemente combinada com *deep learning*. Hinton (2014) introduziu a convolução para permitir, redes de reconhecimento de imagens, funcionarem similarmente à sistemas biológicos e, assim, alcançar resultados mais acurados. Uma destas abordagens é a conectividade esparsa ou escassa (***sparse connectivity***) em que nós não criamos e nem utilizamos todos os pesos sinápticos possíveis. A Figura 9.1 mostra uma rede com conectividade escassa.

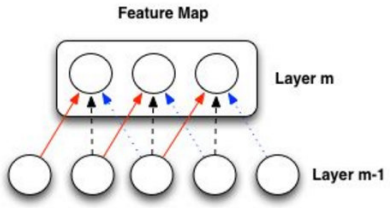
Figure 9.1: Sparse Connectivity



Transparência 48: **Deep Learning: Convolutional Neural Networks (12/40)**

Uma rede neural *feedforward* regular usualmente cria e utiliza todas as conexões sinápticas entre duas camadas. Na tecnologia empregada em *deep learning*, nós nos referimos a estas camadas de camadas densas (***dense layers***). Em adição a possibilidade de não precisar representar todos os pesos possíveis da rede, redes neurais convolucionais também poderão compartilhar seus pesos, como visto na Figura 9.2.

Figure 9.2: Shared Weights

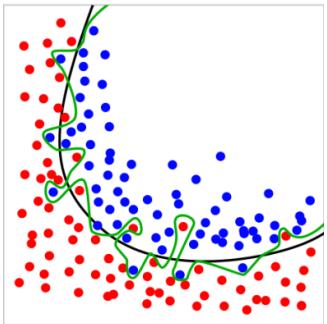


Como você pode ver na Figura 9.2, os neurônios compartilham somente três pesos individualmente. As linhas vermelhas, pretas e azuis indicam os pesos individuais. Compartilhamento de pesos permite ao programa armazenar estruturas complexas, enquanto mantêm a eficiência de memória e processamento.

Transparência 49: **Deep Learning: Neuron Dropout and Overfitting (13/40)**

Nas estatísticas e na aprendizagem de máquinas, uma das tarefas mais comuns é adequar um "modelo" a um conjunto de dados de treinamento, de modo a poder fazer previsões confiáveis em dados gerais não treinados. No ***overfitting***, um modelo estatístico descreve o erro ou ruído aleatório de um modelo em vez do relacionamento subjacente propriamente dito, ou seja, o modelo de aprendizagem empregado aprende de forma inapropriada. Na Figura abaixo, a linha verde representa um modelo com efeito de *overfitting* e a linha preta representa um modelo regularizado.

Enquanto a linha verde segue melhor os dados de treinamento, é provável que a linha verde tenha uma taxa de erro maior nos novos dados não vistos, em comparação com a linha preta.



Transparência 50: **Deep Learning: Neuron Dropout and Overfitting (13/40)**

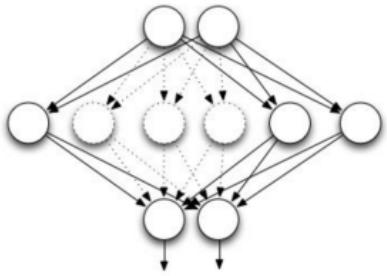
Dropout é uma *técnica de regularização* que conduz a muitos benefícios para *deep learning*. Assim como muitas técnicas de regularização, *dropout* pode prevenir *overfitting*. Você também pode aplicar *dropout* em uma rede neural de camada em camada, assim como é feita na convolução. Em geral, se aplicada a técnica de *dropout*, em uma rede de camada simples, essa camada recebe o nome de camada *dropout* (*dropout layer*). De fato, você pode misturar camadas *dropout* com camadas regulares e camadas convolucionais em uma única rede neural. Entretanto, nunca se deve misturar técnicas de *dropout* e técnicas de convolução numa mesma camada.

Hinton (2012) introduziu *dropout* como um simples e efetivo algoritmo de regularização para reduzir *overfitting*. *Dropout* trabalha por remover certos neurônios na camada de *dropout*. O ato de remover estes neurônios previne outros neurônios de tornarem-se excessivamente dependentes dos neurônios retirados. O programa também remove todos esses neurônios escolhidos, juntamente com todas as suas conexões, sem causar prejuízo algum na nova arquitetura gerada e faz isso de forma harmoniosa.A Figura 9.3 ilustra esse processo.

Transparência 51: **Deep Learning: Neuron Dropout and Overfitting (13/40)**

A rede neural da Figura 9.3 contêm uma camada de entrada, uma camada *dropout* e uma camada de saída. A camada *dropout* tem vários de seus neurônios removidos. Os círculos, feitos de linhas pontilhadas, indicam os neurônios que o algoritmo de *dropout* removeu. As linhas pontilhadas de conexões indicam os pesos que o algoritmo *dropout* removeu quando ele removeu os neurônios.

Figure 9.3: Dropout Layer



Ambos *dropout* e outras formas de regularização são extensivos tópicos no campo de redes neurais. O Capítulo 12 intitulado *Dropout and Regularization* cobre a teoria de regularização com foco particular em *dropout*. Este capítulo também contêm uma pequena explanação sobre os algoritmos de regularização L1 e L2. L1 e L2 desencorajam as redes neurais de um extensivo uso de muitos pesos e a inclusão de certas entradas irrelevantes.

Transparência 52: **Deep Learning: GPU Training (14/40)**

Hinton (1987) introduziu uma nova maneira para treinar *Deep Belief Neural Network (DBNN)* de forma eficiente. *Deep Neural Networks* tem existido quase tão longamente quanto as próprias redes neurais convencionais. Entretanto, até o surgimento do algoritmo de Hilton, nenhuma maneira eficiente para treinar *Deep Neural Networks* existia. Os algoritmos *Backpropagation* e suas variantes são muito lentos, e, o problema, de anulação do gradiente em mínimos locais, dificulta enormemente o treinamento.

As unidades de processamento gráfico (GPU), a parte do computador que é responsável pela exibição gráfica, é a maneira que pesquisadores resolveram o problema do treinamento de redes neurais *feedforward*. Muitos de nós estamos familiarizados com GPUs por causa que os modernos vídeo games utilizam gráficos 3D.

A representação destas imagens gráficas é matematicamente denso, e, para melhorar essas operações matemáticas, os computadores muito cedo confiaram esse processamento na unidade de processamento central (CPU). Entretanto, este abordagem não é efetiva. Os sistemas gráficos nos vídeo games modernos requerem circuitos dedicados, que se tornaram a GPU, ou *video card*. Essencialmente, modernas GPUs são computadores que funcionam dentro dos nossos computadores.

Transparência 53: **Deep Learning: GPU Training (15/40)**

Como os pesquisadores já descobriram, o poder de processamento contido em uma GPU pode ser aproveitado para tarefas matemáticas densas, tais como treinar redes neurais artificiais. Desta forma, GPUs podem executar tarefas de uso geral, além do processamento gráfico de imagens. Estas tarefas gerais de processamento recebem o nome especial de *General-Purpose Use of the GPU (GPGPU)*. Quando aplicada em *Deep Learning*, a performance melhora extraordinariamente. A combinação de GPU com funções de ativação ReLU, regularização e o algoritmo *backpropagation* regular pode produzir resultados notáveis.

Entretanto, GPGPU pode ser difícil de usar. Programas escritos para GPU empregam uma linguagem de programação de baixo nível denominada de C99. Esta linguagem é muito similar a linguagem de programação regular em C/C++. Entretanto, em muitas maneiras, A linguagem C99 requerida pela GPU é muito mais difícil do que a linguagem de programação regular C/C++.

Transparência 54: **Deep Learning: GPU Training (16/40)**

Felizmente, você não precisa aprender programação em GPU para explorar seu poder de processamento. Assim, nós não recomendamos que você aprenda a programar GPU por causa que ela é quase completamente diferente de programação de CPU. Boas técnicas de programação de CPU podem produzir programas horrivelmente ineficientes em GPU. Se você quiser utilizar GPU, você deve trabalhar com um pacote que suporta isto.

GPUs podem alcançar resultados excepcionais baseado simplesmente em poder de processamento puro. Em 2010, *the Swiss AI Lab IDSIA* mostrou que, apesar do problema de anulação do gradiente, o poder de processamento superior das GPUs tornou o algoritmo *backpropagation* tratável para redes neurais *feedforward* (Ciresan et al, 2010).

Transparência 55: **Deep Learning: Tools for Deep Learning (17/40)**

Um dos desafios primários de *Deep Learning* é o tempo de processamento para treinar a rede. Nós frequentemente treinamentos esses algoritmos durante muitas horas, ou mesmo dias, procurando redes neurais que se adaptam bem aos conjuntos de dados.

Nós podemos utilizar várias bibliotecas (*frameworks*) para serem empregadas em nossas pesquisas e nos modelos de predição. Muitas destas bibliotecas são ajustadas para treinar muito rapidamente. Entre as principais bibliotecas, citam-se: Tensorflow, H2O, Theano, Lasagne, NoLearn e ConvNetJS. Vejamos, a seguir, brevemente cada uma destas bibliotecas.

Transparência 56: **Deep Learning: Tools for Deep Learning (18/40)**

Tensorflow...

Transparência 57: **Deep Learning: Tools for Deep Learning (19/40)**

H2O é uma *framework* de aprendizado de máquina que suporta uma vasta variedade de linguagens de programação. H2O é implementada em JAVA. H2O pode ser utilizada com R, Python, Scala, Java, e qualquer outra linguagem que se comunica com H2O's REST API. Em adição aos pacotes de *Deep Learning*, H2O suporta uma variedade de outros modelos de aprendizagem de máquinas, tais como, regressão logística e árvores de decisão. Para maiores informações sobre H2O, entre na seguinte URL: https://www.h2o.ai/download/.

Theano é um pacote de matemática para Python. Theano primariamente objetiva a matemática. Embora Theano não implementa diretamente *Deep Learning Networks*, ele provem todas as ferramentas matemáticas necessárias para o programador criar aplicações com redes de aprendizagem profunda. Theano também suporta diretamente GPGPU. Você pode achar o pacote de Theano na seguinte URL: http://deeplearning.net/software/theano/.

Os criadores de Theano também escreveram um extenso tutorial para *Deep Learning*, usando Theano e que pode ser achado na seguinte URL: http://deeplearning.net/.

Transparência 58: **Deep Learning: Tools for Deep Learning (20/40)**

Pelo fato de que Theano não suporta diretamente técnicas e algoritmos de *Deep Learning*, vários pacotes tem sido construídos para tornar a utilização de Theano mais fácil para os programadores de *Deep Learning*. Um par de pacotes, frequentemente utilizados juntos, é Lasagne e NoLearn. Nolearn é um pacote para Python que provem abstrações para vários algoritmos de aprendizagem de máquinas. NoLearn é especializado em redes neurais. Um dos pacotes de redes neurais suportado por NoLearn é Lasagne, que provem suporta *Deep Learning* e pode ser achado na seguinte URL: https://pypi.python.org/pypi/Lasagne/0.1dev.

Você pode acessar o pacote de NoLearn na seguinte URL: https://github.com/dnouri/nolearn.

A biblioteca de *Deep Learning* denominada Lasange leva este nome por causa da comida italiana conhecida como lasanha. Essa analogia, embora um pouco estranha, é bastante pertinente, pois observe que uma lasanha é feita de muitas camadas.

Suporte para *Deep Learning* tem também sido criado para Javascript. O pacote ConvNetJS implementa muitos algoritmos de aprendizagem profunda, particularmente na área de redes neurais convolucionais. Esse pacote pode ser encontrado na seguinte URL: http://cs.stanford.edu/people/karpathy/convnetjs/.

Transparência 59: **Deep Learning: Deep Belief Neural Networks (21/40)**

A rede neural denominada *Deep Belief Neural Network (DBNN)* foi uma das primeiras aplicações de *Deep Learning*. Uma DBNN é simplesmente uma rede de crença regular com muitas camadas. *Belief Networks*, introduzida por Neil em 1992 são diferentes das redes neurais *Feedforward* regulares.

Hinton (2007) descreve as DBNNs como "modelos generativos probabilísticos que são compostos de múltiplas camadas estocásticas com variáveis latentes". Por causa desta descrição técnica ser um pouco complicada, nós definimos alguns termos importantes, a seguir:

Transparência 60: **Deep Learning: Deep Belief Neural Networks (22/40)**

Modelo Probabilístico: DBNNs são utilizadas somente para classificar e sua saída é a probabilidade que uma entrada pertence a cada classe.

Modelo Generativo: DBNNs podem produzir valores plausíveis, e, de forma aleatória, para os valores de entrada. Algumas literaturas se referem a isso como um "fenômeno de idealização".

Múltiplas Camadas: como numa rede neural regular, DBNNs podem ser projetadas com múltiplas camadas.

Variáveis Latentes Estocásticas: DBNNs são projetadas a partir das Máquinas de Boltzmann que produzem valores aleatórios (estocásticos) que não podem ser diretamente observados (são latentes).

Transparência 61: **Deep Learning: Deep Belief Neural Networks (23/40)**

As diferenças primárias entre DBNN e a tradicional rede neural *feedforward* (FFNN) são sumarizadas a seguir:

a) As entradas para a DBNN devem ser somente binárias; as entradas para a FFNN podem ser binárias ou decimais.

b) A saída de uma DBNN é a classe que a entrada pertence; a saída de uma FFNN pode ser uma classe (problemas de classificação) ou uma predição numérica (problemas de regressão).

c) DBNNs podem gerar entradas plausíveis baseado em um dado resultado. FFNNs não podem funcionar como as DBNNs.

Transparência 62: **Deep Learning: Deep Belief Neural Networks (24/40)**

Estas são diferenças importantes. O fato de que uma DBNN pode aceitar somente entrada binária frequentemente limita severamente o tipo de problema que ela pode atacar. É importante observar também que uma DBNN pode ser utilizada somente com problemas de classificação e não para problemas de regressão. A rede DBNN não pode ser aplicada em problemas de predição.

Se o interesse for problemas de predição, a técnica mais adequada são as RDFN (*Regular Deep Feedforward Network*).

Comparada com as FFNNs, DBNNs podem inicialmente parecer de alguma forma mais restrita. Entretanto, elas fazem por ter a habilidade de gerar uma entrada plausível baseada numa dada saída. Um dos primeiros experimentos realizados com DBNN foi a classificação de dez dígitos manuscritos (clássico MNIST *problem*). Este problema foi primeiro relatado no paper de Hinton de 2006.

Transparência 63: **Deep Learning: Deep Belief Neural Networks (25/40)**

A primeira linha da Figura 9.5 mostra a variedade de diferentes zeros que a DBNN conseguiu gerar dos seus dados de treinamento.

Figure 9.5: DBNN Dreaming of Digits

As Máquinas de Boltzmann Restritas (RBM) é o centro ou o cerne da DBNN. As entradas providas para a DBNN passam através de uma série de RBMs empilhadas que formam as camadas desta rede neural. A criação de camadas adicionais de RBM resultam em DBNNs mais profundas. ***Embora as RBMs são de aprendizagem não supervisionada, a DBNN resultante deste processo é de aprendizagem supervisionada***. A seguir são descritos brevemente alguns algoritmos utilizados para treinar as DBNNs.

Transparência 64: **Deep Learning: Deep Belief Neural Networks (26/40)**

Para ser...

Transparência 65: **Deep Learning: Deep Belief Neural Networks (27/40)**

Para ser...

Transparência 66: **Deep Learning: Deep Belief Neural Networks (28/40)**

Para ser...