Tutorial de Deep Learning

Luiz Gustavo Hafemann
LIVIA
École de Technologie Supérieure – Montréal

Organização do tutorial

- **Dia 1:**
  - Introdução à aprendizagem de máquina
  - Computação simbólica com Theano
- **Dia 2**
  - Redes neurais convolucionais
  - Biblioteca Lasagne

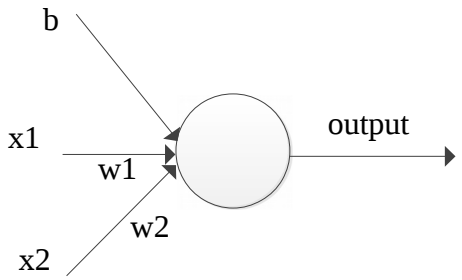
  Dia 3
  - Transfer Learning

Introdução à Redes Neurais

Neurônio artificial:

Combinação linear da entrada: $\mathbf{w}^\intercal \mathbf{x} = w_1 x_1 + w_2 x_2 ... w_m x_m$

Aplicação de uma função não linear: saída = $f(\mathbf{w}^\intercal \mathbf{x} + b)$

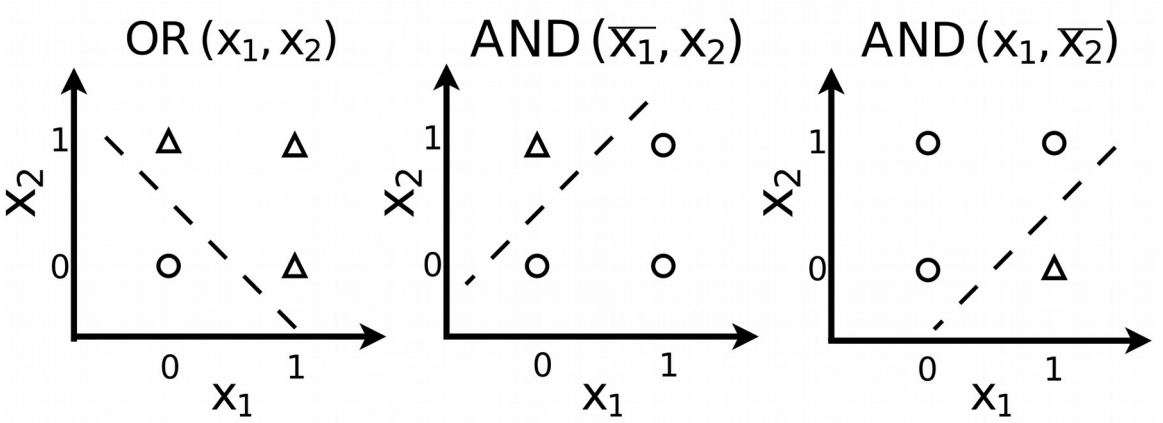


Usando funçao sigmoid → Regressão logística

Introdução à Redes Neurais

Que tipo de problemas podem ser resolvidos?

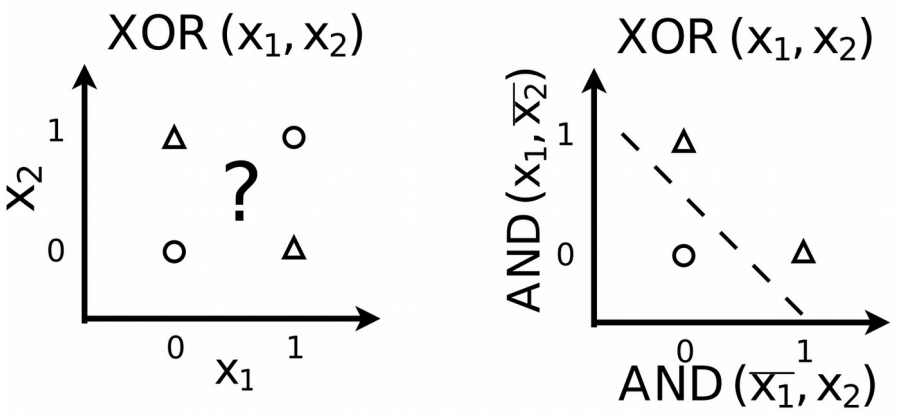
Problemas lineares



(Figura: Hugo Larochelle)

Introdução à Redes Neurais

Não resolve problemas não lineares (e.g: XOR):



A não ser que use uma melhor representação da entrada

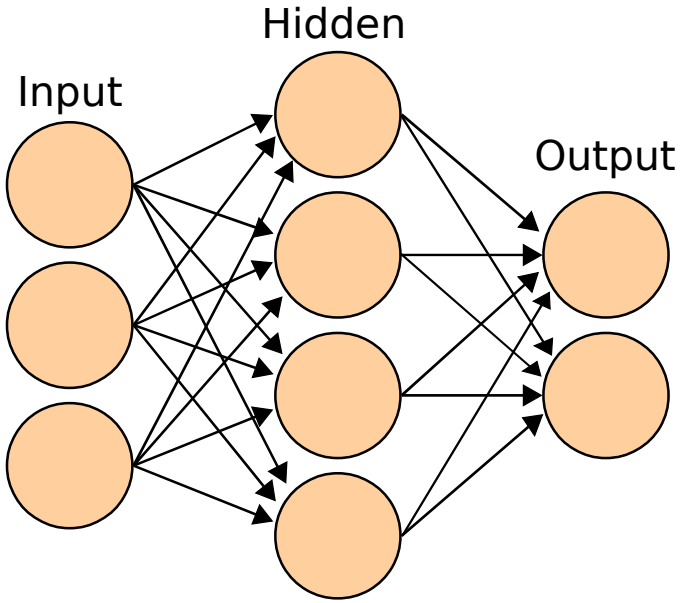
(Figura: Hugo Larochelle)

Redes Neurais com múltiplas camadas

Modelos que utilizaram várias camadas de neurônios

Permite "aprender as representações"

Aprendem funções não-lineares da entrada. Ponto negativo: otimização não-convexa

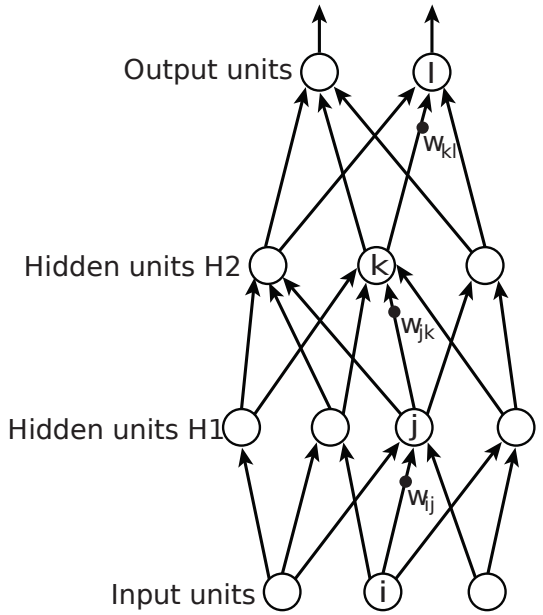


Redes Neurais com múltiplas camadas

Idéia chave: aprendizado baseado em gradiente

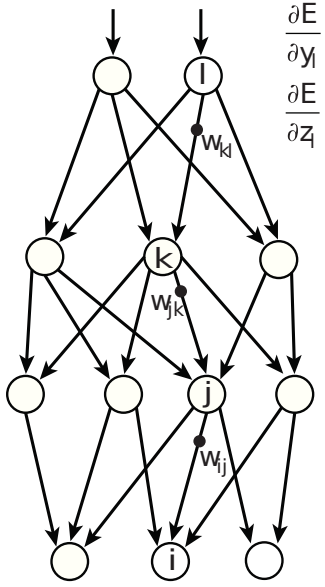
Define-se uma função de custo diferenciável

Calcula-se o gradiente (derivadas parciais do custo referente à cada peso do modelo)



$$y_j = f(z_j)$$

$$z_j = \sum w_{ij} x_i$$



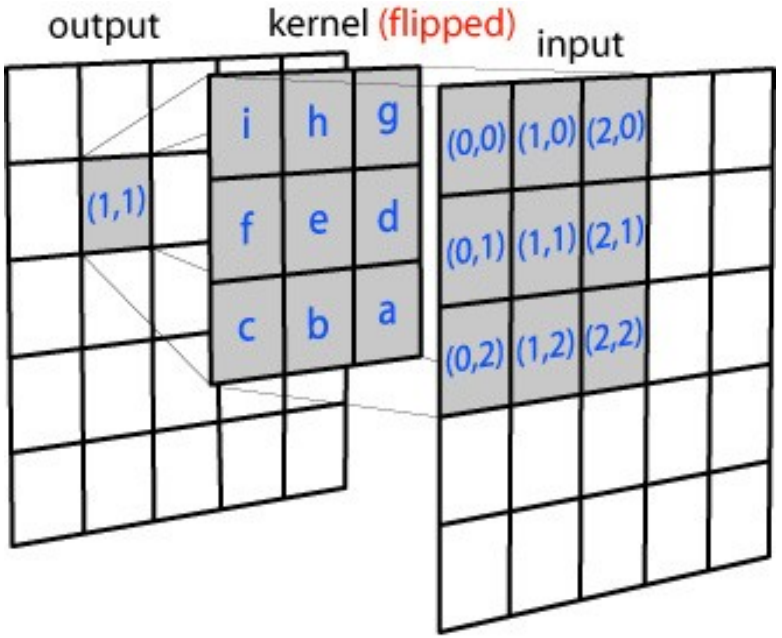
$$\frac{\partial E}{\partial y_l} = y_l -$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}$$

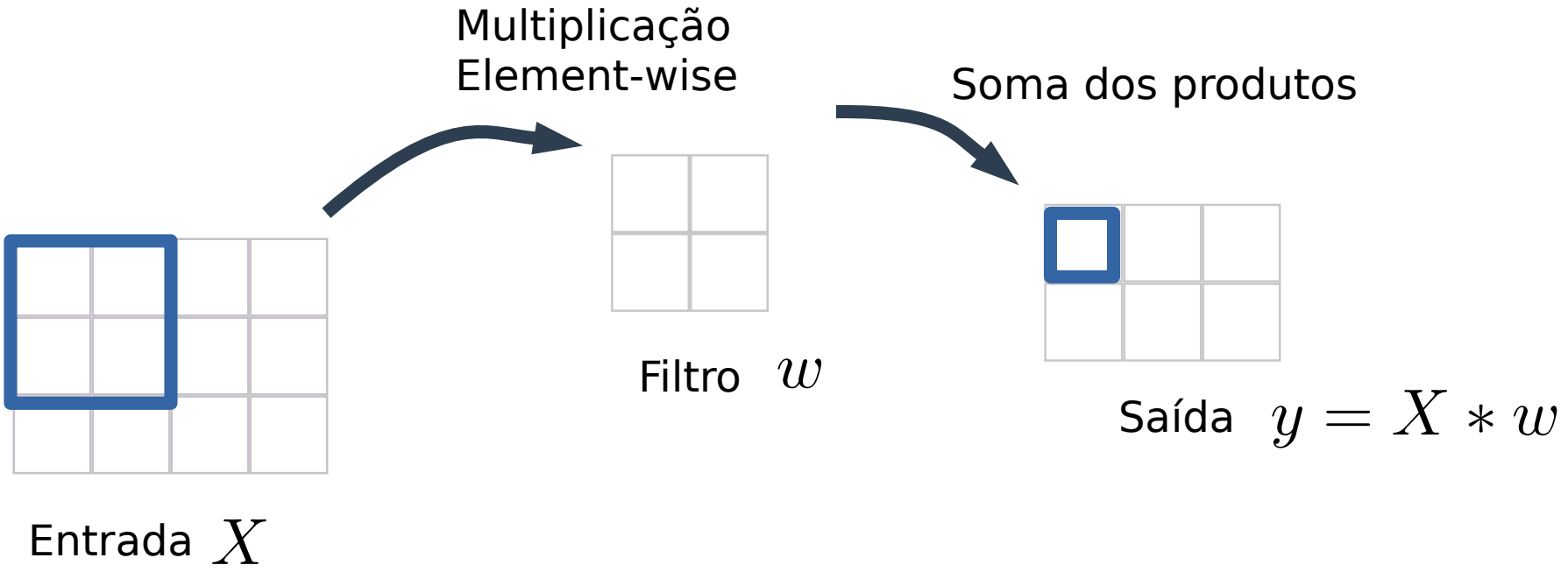
Redes neurais convolucionais

Idéia principal: usar arquiteturas que explorem características presentes em imagens:

- Conexões locais (explora a correlação dos pixels em 2D)
- Pesos compartilhados (mesmo detector é usado em partes diferentes da imagem)

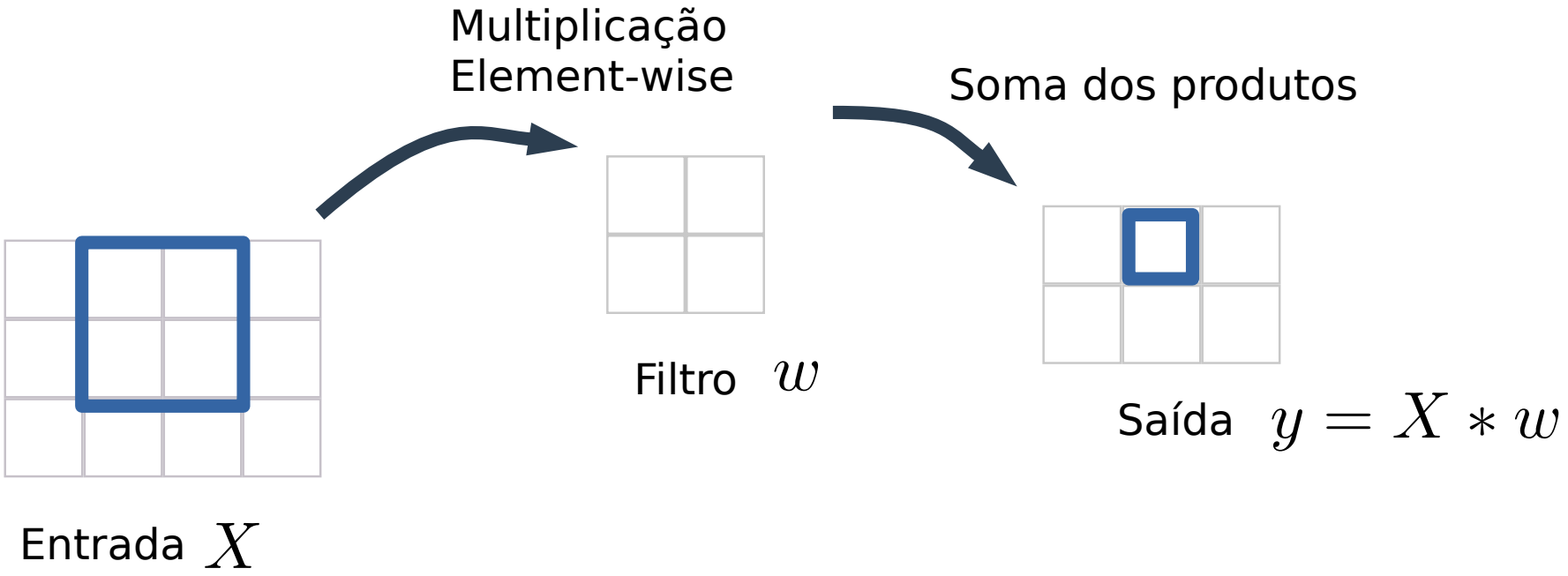


Redes neurais convolucionais



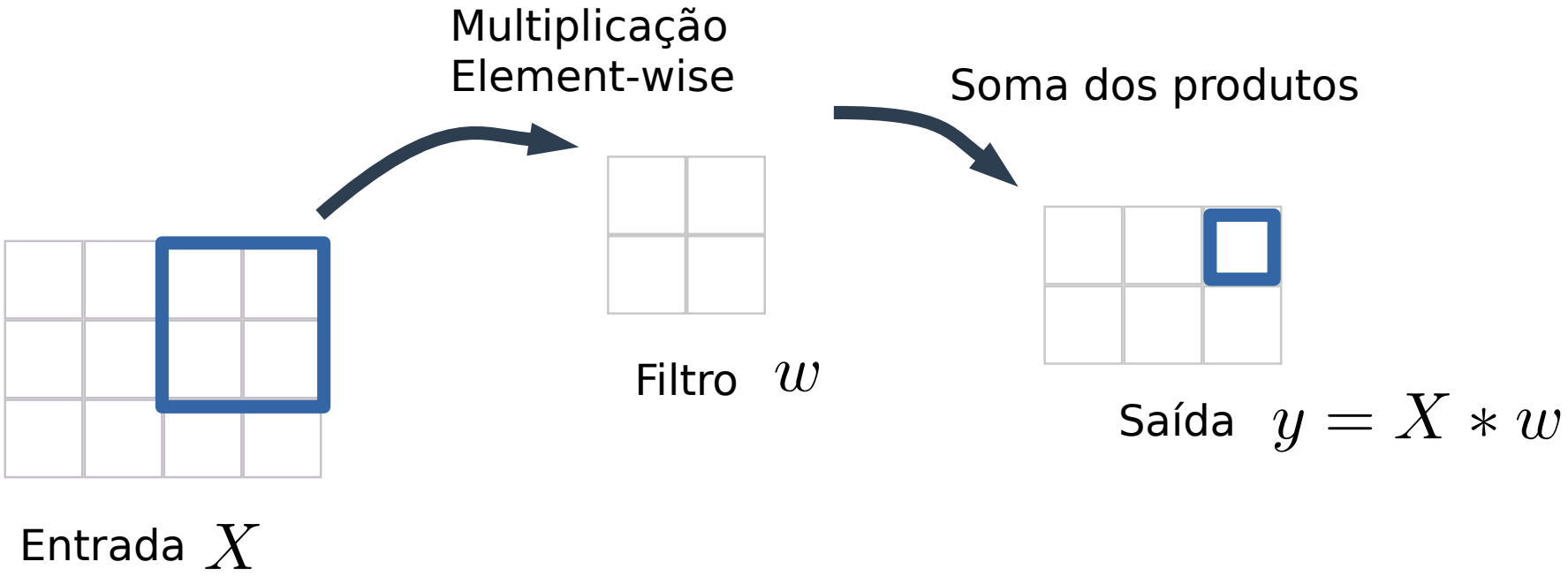
$$y_{11} = X_{11}w_{11} + X_{12}w_{12} + X_{21}w_{21} + X_{22}w_{22}$$

Redes neurais convolucionais



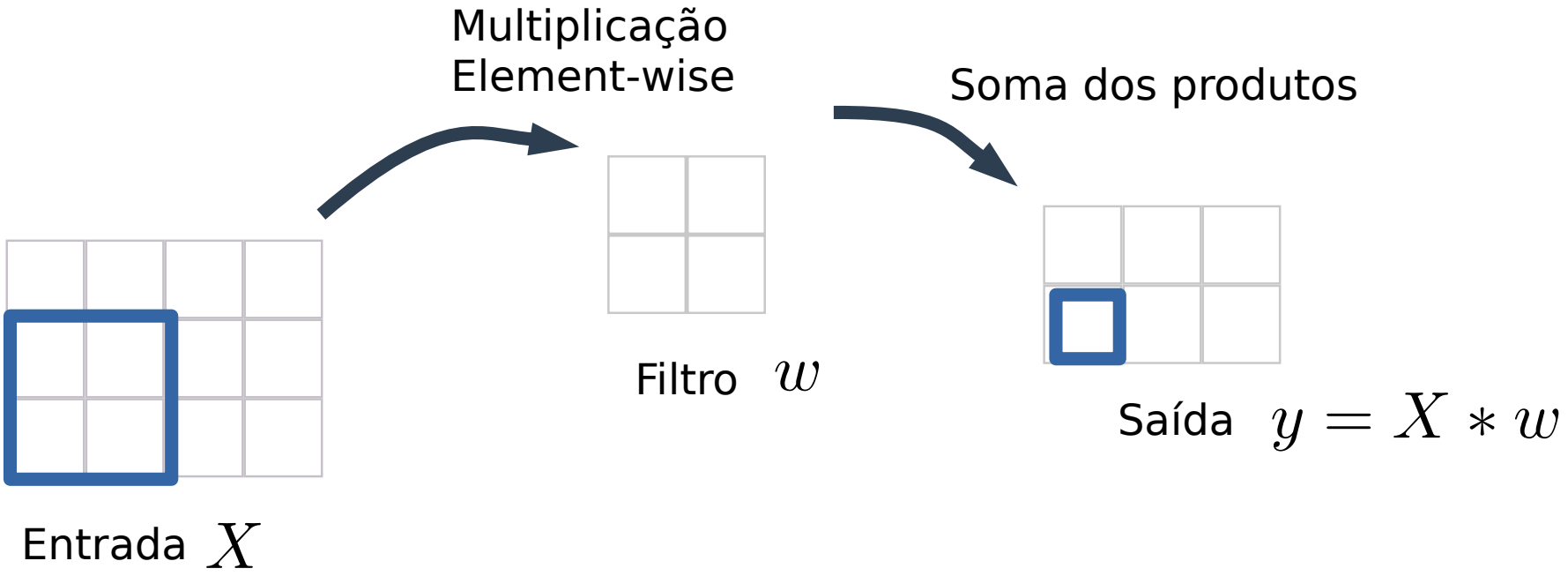
$$y_{12} = X_{12}w_{11} + X_{13}w_{12} + X_{22}w_{21} + X_{23}w_{22}$$

Redes neurais convolucionais

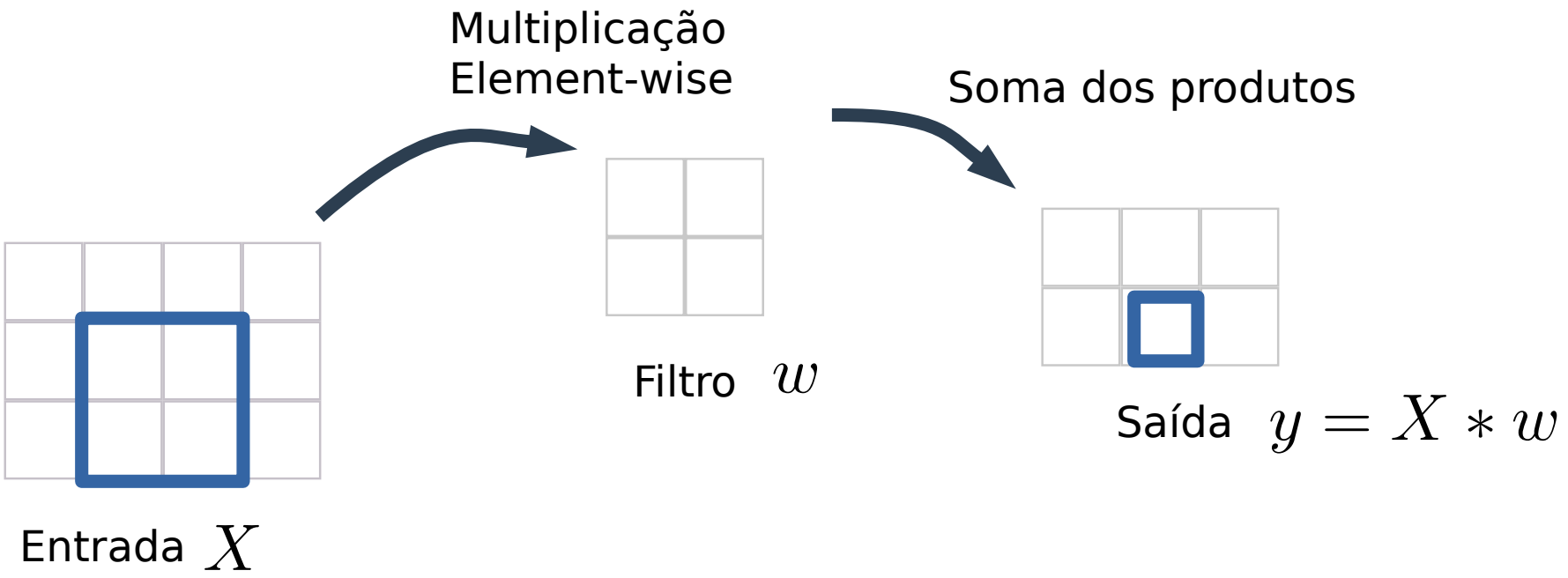


$$y_{13} = X_{13}w_{11} + X_{14}w_{12} + X_{23}w_{21} + X_{24}w_{22}$$

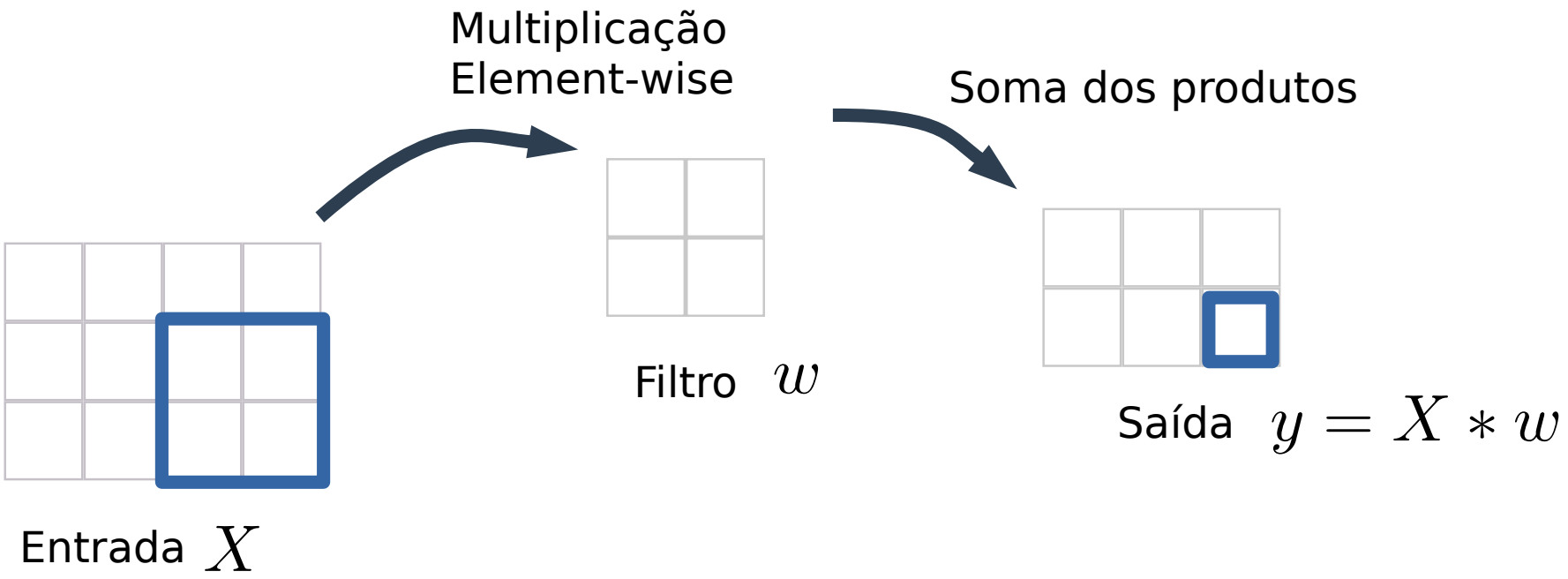
Redes neurais convolucionais



Redes neurais convolucionais



Redes neurais convolucionais



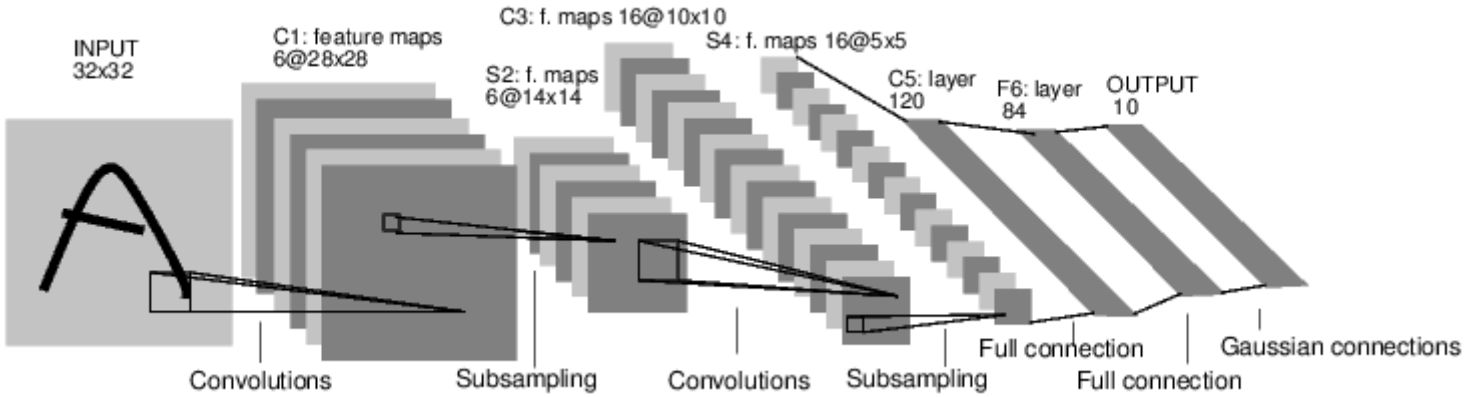
Redes neurais convolucionais

Outras características:

- Operação é linear, e diferenciável
- Computacionalmente caro, mas facilmente paralelizável em GPUs
- Compartilhamento de pesos resulta em menos pesos a serem aprendidos, o que facilita o processo de treinamento (mais robusto à overfitting)

Exemplo de rede convolucional

Lenet-5: usado para reconhecimento de dígitos/letras manuscritas:



Introdução ao Lasagne

Introdução ao Lasagne

- **Usa Theano como back-end, e contém implementações de funcionalidades úteis para Deep Learning:**

Introdução ao Lasagne

- **Usa Theano como back-end, e contém implementações de funcionalidades úteis para Deep Learning:**
  - Camadas convolucionais, max-pooling

Introdução ao Lasagne

- **Usa Theano como back-end, e contém implementações de funcionalidades úteis para Deep Learning:**
  - Camadas convolucionais, max-pooling
  - Funções de custo comuns, e.g. cross-entropy

Introdução ao Lasagne

- **Usa Theano como back-end, e contém implementações de funcionalidades úteis para Deep Learning:**
  - Camadas convolucionais, max-pooling
  - Funções de custo comuns, e.g. cross-entropy
  - Algoritmos de otimização: gradient descent, RMSProp, Adam

Introdução ao Lasagne

- **Usa Theano como back-end, e contém implementações de funcionalidades úteis para Deep Learning:**
  - Camadas convolucionais, max-pooling
  - Funções de custo comuns, e.g. cross-entropy
  - Algoritmos de otimização: gradient descent, RMSProp, Adam

- **Outros frameworks: Keras, Tensorflow, Torch, Caffe**

Introdução ao Lasagne

- **Usa Theano como back-end, e contém implementações de funcionalidades úteis para Deep Learning:**
  - Camadas convolucionais, max-pooling
  - Funções de custo comuns, e.g. cross-entropy
  - Algoritmos de otimização: gradient descent, RMSProp, Adam

- **Outros frameworks: Keras, Tensorflow, Torch, Caffe**
  - https://github.com/zer0n/deepframeworks/blob/master/README.md

Principais conceitos

Principais conceitos

- **Camada (layer):**
  - Representa uma camada da rede neural, especificando o tipo de computação (e.g. convolução) e hiperparâmetros (e.g. tamanho do filtro da convolução).

Principais conceitos

- **Camada (layer):**
  - Representa uma camada da rede neural, especificando o tipo de computação (e.g. convolução) e hiperparâmetros (e.g. tamanho do filtro da convolução).
  - Exemplo:

```
data = InputLayer((None, 100))
hid = DenseLayer(data, 10)
out = DenseLayer(hid, 1, nonlinearity=sigmoid)
```

Principais conceitos

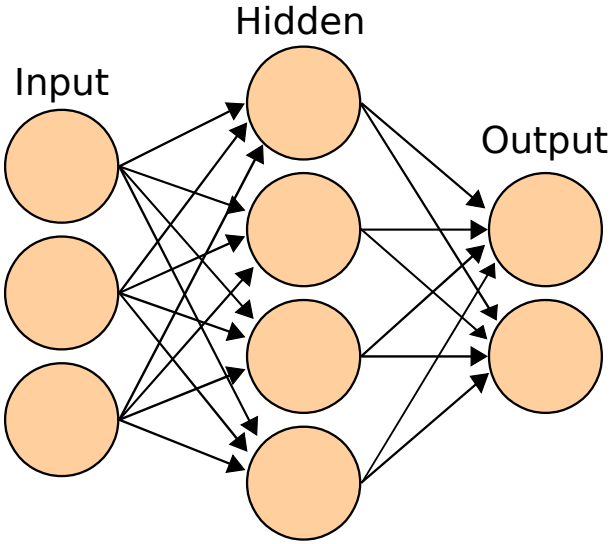
Principais conceitos

- **Não possui conceito de “modelo”:**
  - A última camada possui toda a informação necessária para construir o grafo

Principais conceitos

- **Não possui conceito de “modelo”:**
  - A última camada possui toda a informação necessária para construir o grafo
  - A prática comum é de colocar as camadas em um dicionário de python:

```
net = {}
net['data'] = InputLayer((None, 3))
net['hid'] = DenseLayer(net['data'], 4)
net['out']= DenseLayer(net['hid'], 2,
                       nonlinearity=softmax)
```



Principais conceitos

Principais conceitos

- **Funções e atributos:**
  - Os atributos (e.g. pesos e bias) são acessados dessa forma:

```
net['hid'].W  # Variável compartilhada (Theano) que representa os pesos dessa camada
```

Principais conceitos

- **Funções e atributos:**
  - Os atributos (e.g. pesos e bias) são acessados dessa forma:

    net['hid'].W # Variável compartilhada (Theano) que representa os pesos dessa camada
  - A saída de uma camada é computado como:

    net_output = lasagne.layers.get_output(net['out']) # Retorna uma variável simbólica que contém o resultado da operação da rede

Principais conceitos

- **Funções e atributos:**
  - Os atributos (e.g. pesos e bias) são acessados dessa forma:

    net['hid'].W # Variável compartilhada (Theano) que representa os pesos dessa camada
  - A saída de uma camada é computado como:

    net_output = lasagne.layers.get_output(net['out']) # Retorna uma variável simbólica que contém o resultado da operação da rede
  - Exemplo: uma função para obter a saída da rede (i.e. $\hat{y}$):

    input_var = net['data'].input_var

    predicted = lasagne.layers.get_output(net['out'], inputs=input_var)

    get_predictions = **theano.function([input_var], predicted)**

    y_pred = get_predictions(x)

Principais camadas

Principais camadas

- **Principais Camadas:**
  - DenseLayer: Camada “fully-connected”
  - Conv2DLayer: Camada de convolução
  - MaxPool2DLayer: Camada de max-pooling

Principais camadas

- **Principais Camadas:**
  - DenseLayer: Camada “fully-connected”
  - Conv2DLayer: Camada de convolução
  - MaxPool2DLayer: Camada de max-pooling
- **Principais não-linearidades:**
  - ReLU: REctified Linear Unit (padrão)
  - Sigmoid: interpretado como p(y|x) em problemas binários
  - Softmax: interpretado como p(y|x) em problemas com várias classes

Objetivos

Objetivos

Após definirmos a architetura da rede, precisamos definir a função de custo:

- binary_crossentropy: para funções de classificação de 2 classes
- categorical_crossentropy: para funções de classificação de várias classes
- squared_error: para problemas de regressão

Objetivos

Após definirmos a architetura da rede, precisamos definir a função de custo:

- binary_crossentropy: para funções de classificação de 2 classes
- categorical_crossentropy: para funções de classificação de várias classes
- squared_error: para problemas de regressão

- **Exemplo:**

```
input_var = net['data'].input_var
output_var = T.vector()
predicted = lasagne.layers.get_output(net['out'], inputs=input_var)
loss = lasagne.objectives.categorical_crossentropy(predicted, output_var)
```

Algoritmo de otimização

Algoritmo de otimização

Após definir a função de custo, escolhemos um algorithmo de otimização (e.g. SGD):

```
params = lasagne.layers.get_all_params(net['out'])
updates = lasagne.updates.sgd(loss, params, lr)
```

Algoritmo de otimização

Após definir a função de custo, escolhemos um algorithmo de otimização (e.g. SGD):

```
params = lasagne.layers.get_all_params(net['out'])
updates = lasagne.updates.sgd(loss, params, lr)
```

Podemos então compilar a função de treinamento:

```
train_fn = theano.function([input_var, output_var], loss,
updates=updates)
```

Loop de treinamento

Para treinar a rede, basta chamar a função de treinamento iterativamente.

- Usando toda a base de treinamento (batch)
- Ou usando poucos exemplos da base de treinamento por vez (mini-batch)

Vamos explorar as duas opções durante o exercício

Sumário

Em resumo, precisamos:

1) Definir uma arquitetura
2) Definir uma função de custo
3) Escolher um algoritmo de otimização
4) Compilar a função de treinamento
5) Chamar a função de treinamento até convergência

Prática

Links úteis

Curso de CNNs da Stanford:
http://cs231n.stanford.edu/syllabus.html

Livro de Deep Learning:

http://www.deeplearningbook.org/

Biblioteca		Prós	Contras
Lasagne + Theano	Python	Derivação automática Acesso fácil ao Theano Bom suporte à modelos pré-treinados	Demanda mais trabalho para implementar loop de treinamento
Keras + Theano	Python	Fácil de implementar	“Esconde” o Theano: mais difícil de implementar algo novo
Tensorflow	C++, Python	Derivação automática Fácil de paralelizar	Sintaxe mais complicada se for usar apenas uma GPU;
Torch	Lua	Rápido Não precida “tempo de compilação” para os modelos	Curva de aprendizado mais longa Mais difícil acesso à bibliotecas de imagem, etc (por ser em lua)
Caffe	C++	Rápido Bom suporte à modelos pré-treinados	Mais “engessado” - necessita descrever modelos em arquivos e