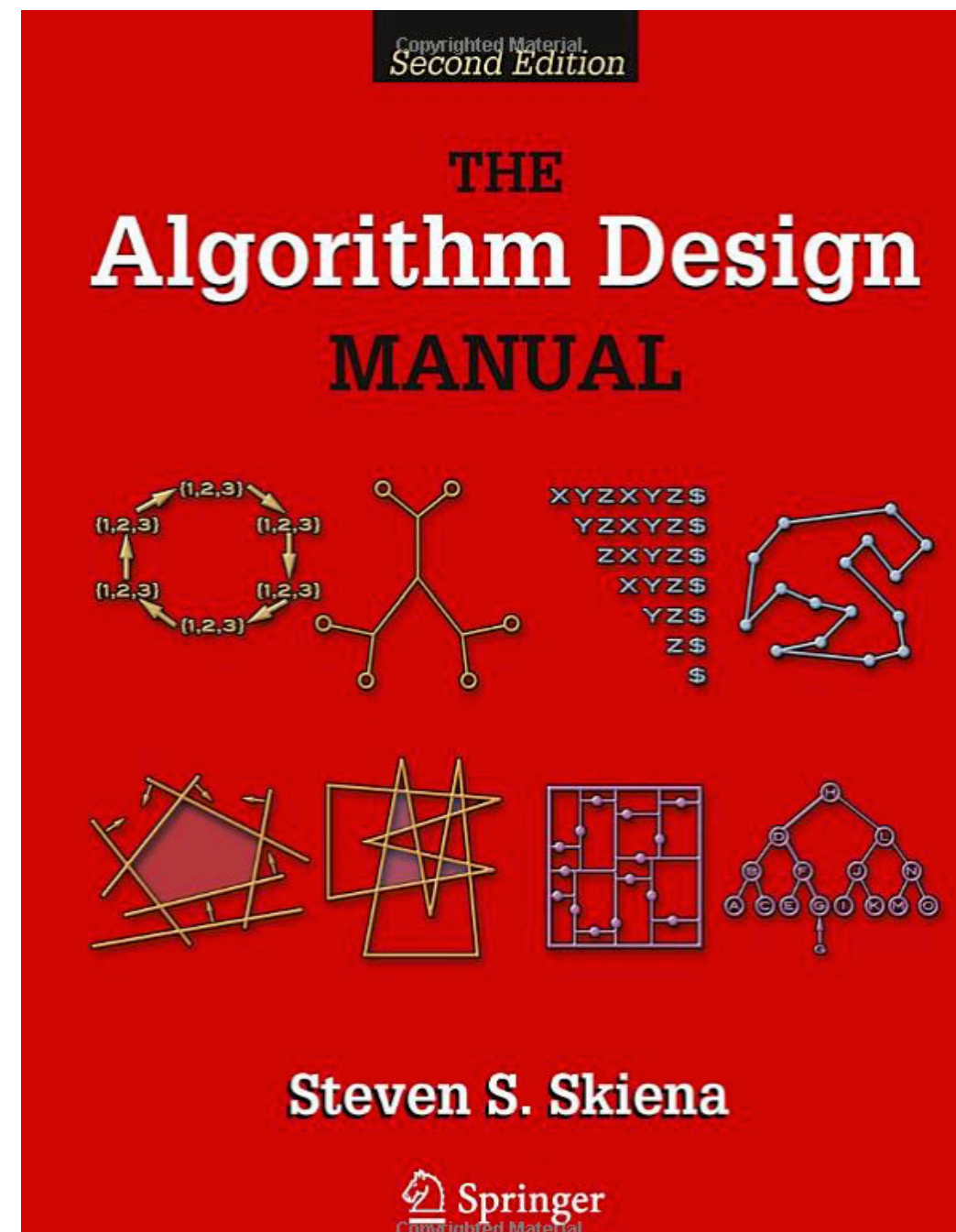
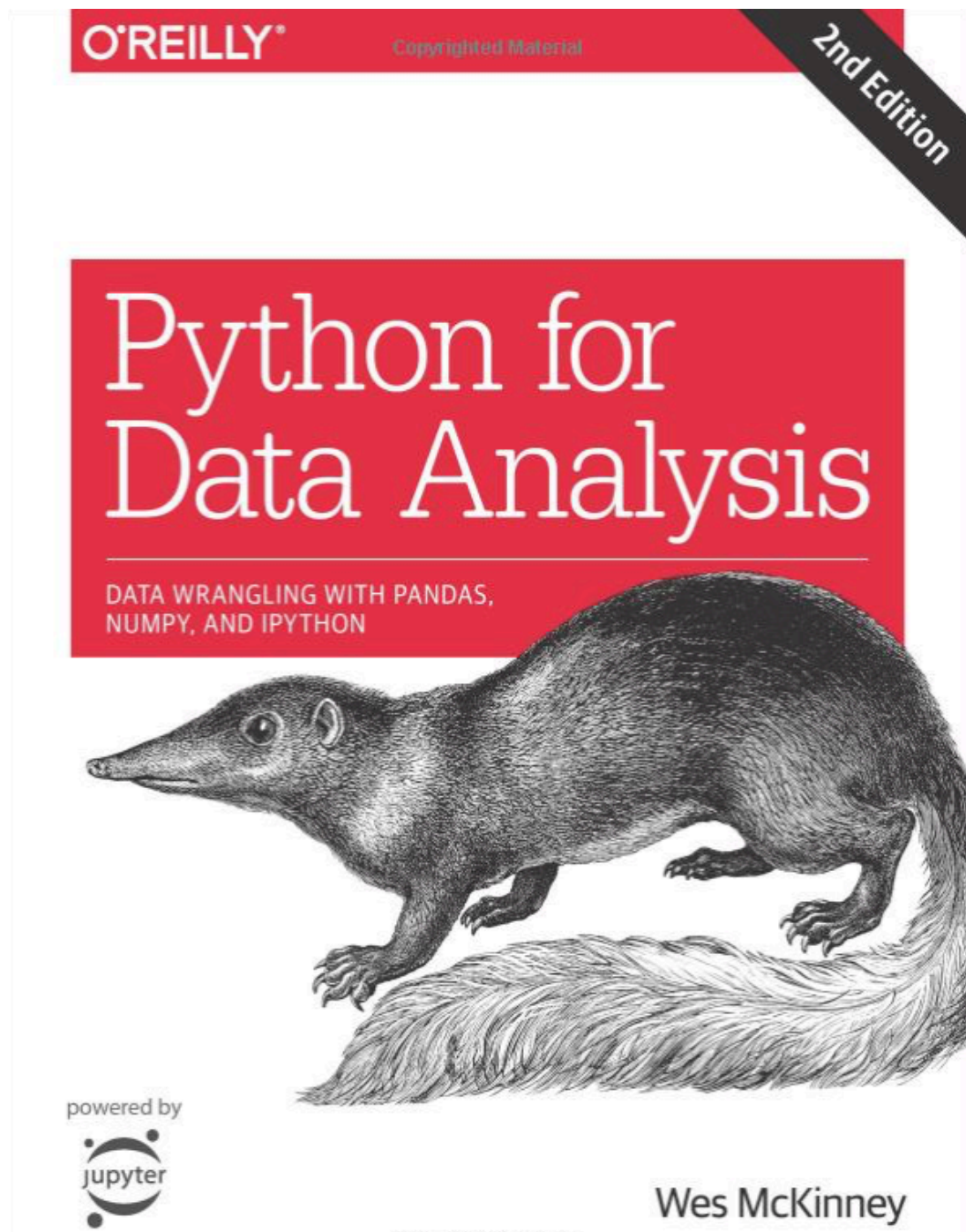


Workshop Intensivo de Aplicações Modernas de Ciência de Dados com Machine Learning

Nono dia

Paulo Cysne Rios Jr.

Livros Recomendados



Publicado em 20 de outubro de 2017

Modelagens Analíticas mais usadas

Usando Estatística Clássica

- Linear Regression Simples e com Polinômios
- Linear Regression Regularizada: Ridge Regression, Lasso Regression, Elastic Net, Early Stopping
- Logistics Regression, Softmax Regression
- Linear Discriminant Analysis

Machine Learning

- Support Vector Machines com kernel linear
- Support Vector Machines com kernel não linear
- Decision Trees

Machine Learning: Ensemble Methods

- Bagging e Pasting
- Random Forests
- Boosting: AdaBoost
- Boosting: Gradient Boosting

Machine Learning: Deep Learning

- Theano
- TensorFlow

Grid Search

***Busca* dos melhores
hyperparametros (hyperparameters)
do modelo analítico
para o conjunto de dados usado**

Usando RandomForest no Dataset de Imóveis

```
>>> from sklearn.ensemble import RandomForestRegressor  
>>> forest_reg = RandomForestRegressor()  
>>> forest_reg.fit(housing_prepared, housing_labels)
```


Usando RandomForest no Dataset de Imóveis com Grid Search

```
from sklearn.model_selection import GridSearchCV
```



```
param_grid = [  
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},  
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},  
]
```

```
forest_reg = RandomForestRegressor()
```



```
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,  
                           scoring='neg_mean_squared_error')
```

```
grid_search.fit(housing_prepared, housing_labels)
```



Usando RandomForest no Dataset de Imóveis com Grid Search

Os melhores hyperparameters encontrados

```
>>> grid_search.best_params_  
{'max_features': 8, 'n_estimators': 30}
```

Usando RandomForest no Dataset de Imóveis com Grid Search

Os melhores hyperparameters encontrados

```
>>> grid_search.best_estimator_  
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,  
                        max_features=8, max_leaf_nodes=None, min_impurity_decrease=0.0,  
                        min_impurity_split=None, min_samples_leaf=1,  
                        min_samples_split=2, min_weight_fraction_leaf=0.0,  
                        n_estimators=30, n_jobs=1, oob_score=False, random_state=42,  
                        verbose=0, warm_start=False)
```

Usando RandomForest no Dataset de Imóveis com Grid Search

Todos os resultados

```
>>> cvres = grid_search.cv_results_  
>>> for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
```

```
...     print(np.sqrt(-mean_score), params)  
...  
63647.854446 {'n_estimators': 3, 'max_features': 2}  
55611.5015988 {'n_estimators': 10, 'max_features': 2}  
53370.0640736 {'n_estimators': 30, 'max_features': 2}  
60959.1388585 {'n_estimators': 3, 'max_features': 4}  
52740.5841667 {'n_estimators': 10, 'max_features': 4}  
50374.1421461 {'n_estimators': 30, 'max_features': 4}  
58661.2866462 {'n_estimators': 3, 'max_features': 6}  
52009.9739798 {'n_estimators': 10, 'max_features': 6}  
50154.1177737 {'n_estimators': 30, 'max_features': 6}  
57865.3616801 {'n_estimators': 3, 'max_features': 8}  
51730.0755087 {'n_estimators': 10, 'max_features': 8}  
49694.8514333 {'n_estimators': 30, 'max_features': 8}  
62874.4073931 {'n_estimators': 3, 'bootstrap': False, 'max_features': 2}  
54643.4998083 {'n_estimators': 10, 'bootstrap': False, 'max_features': 2}  
59437.8922859 {'n_estimators': 3, 'bootstrap': False, 'max_features': 3}  
52735.3582936 {'n_estimators': 10, 'bootstrap': False, 'max_features': 3}  
57490.0168279 {'n_estimators': 3, 'bootstrap': False, 'max_features': 4}  
51008.2615672 {'n_estimators': 10, 'bootstrap': False, 'max_features': 4}
```

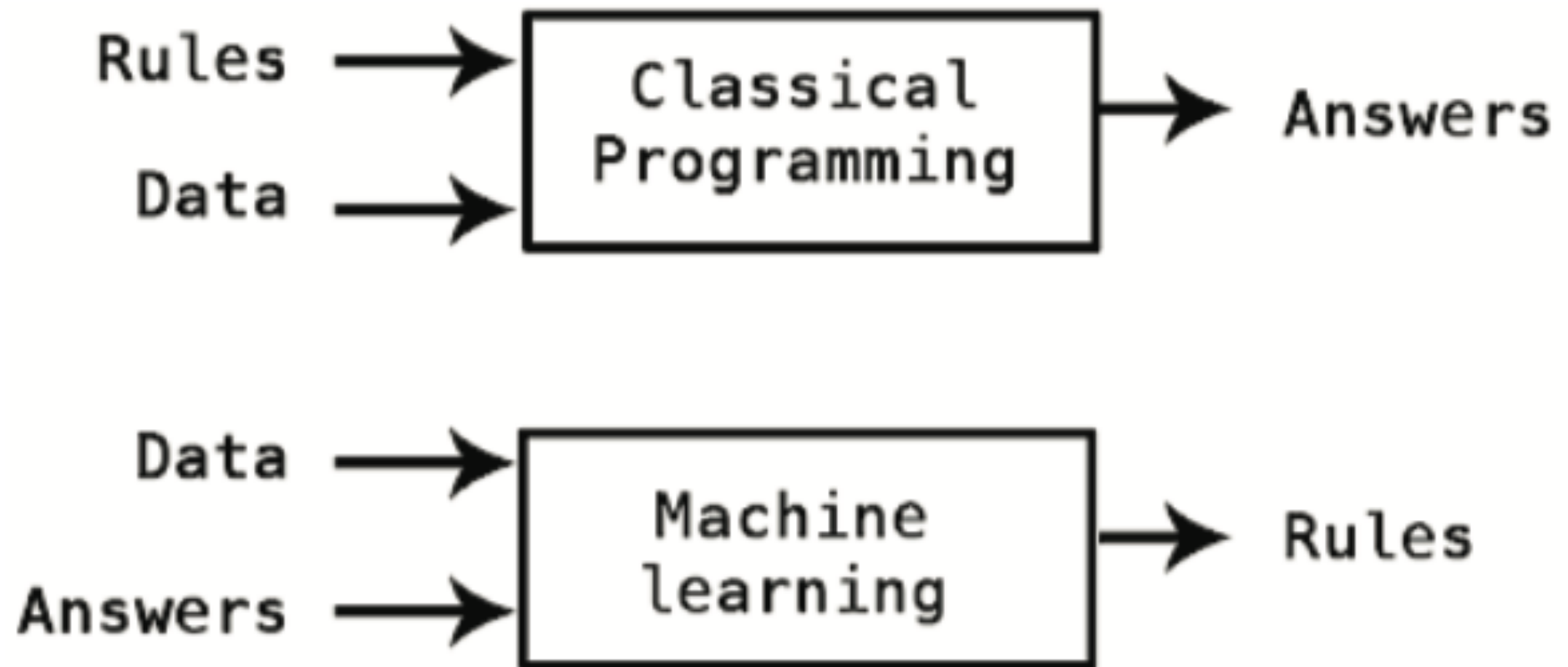
Usando RandomForest no Dataset de Imóveis com Grid Search

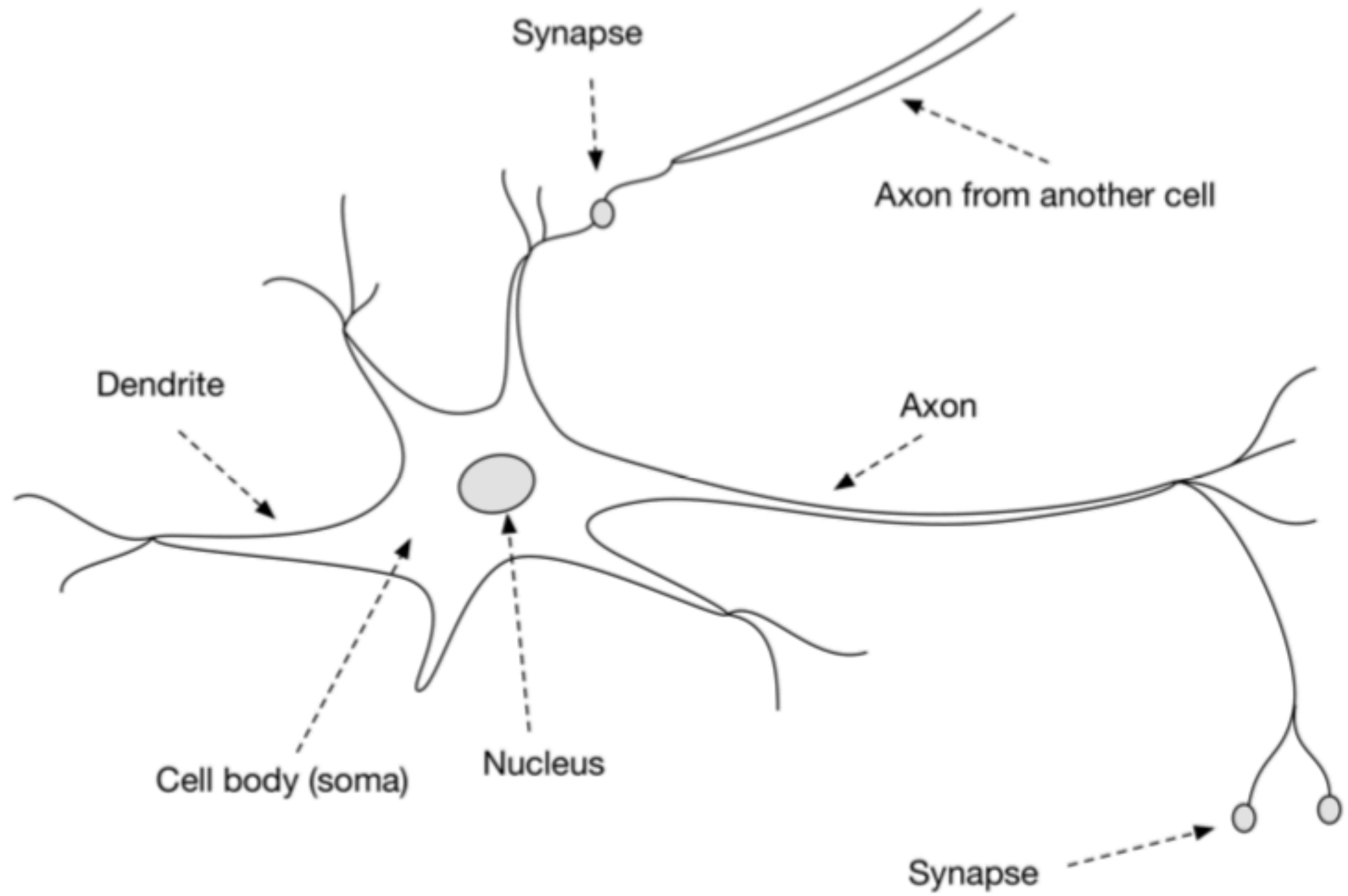
- Neste exemplo, obtemos a melhor solução com o:
- Hiperparâmetro **max_features = 8**
- Hiperparâmetro **n_estimators = 30**
- O score RMSE para esta combinação é **49.694**.
- Que é um pouco melhor do que o resultado obtido sem grid_search, usando os valores padrão do hiperparâmetro, que é de um RMSE = **52.564**.

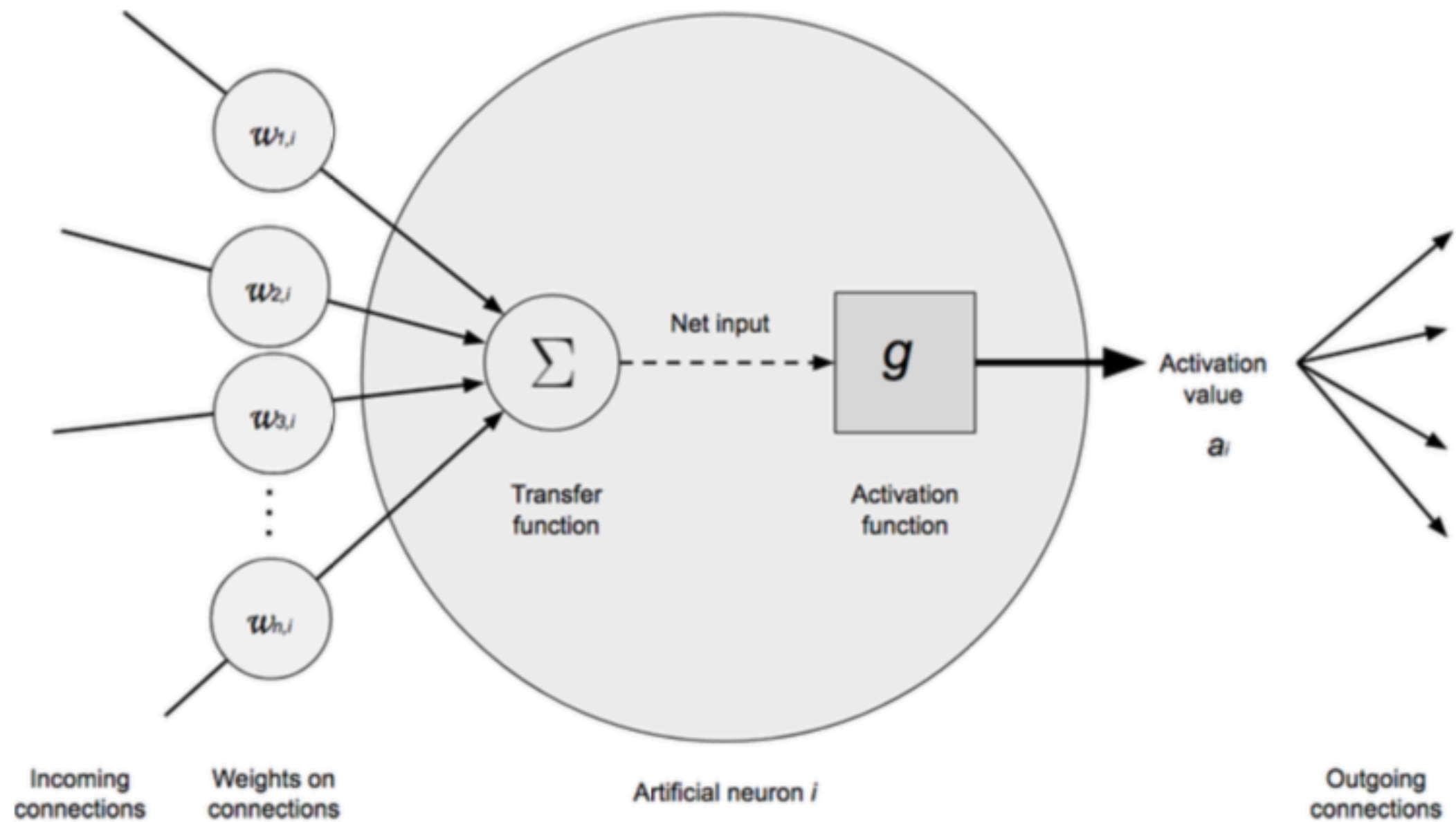
Deep Learning

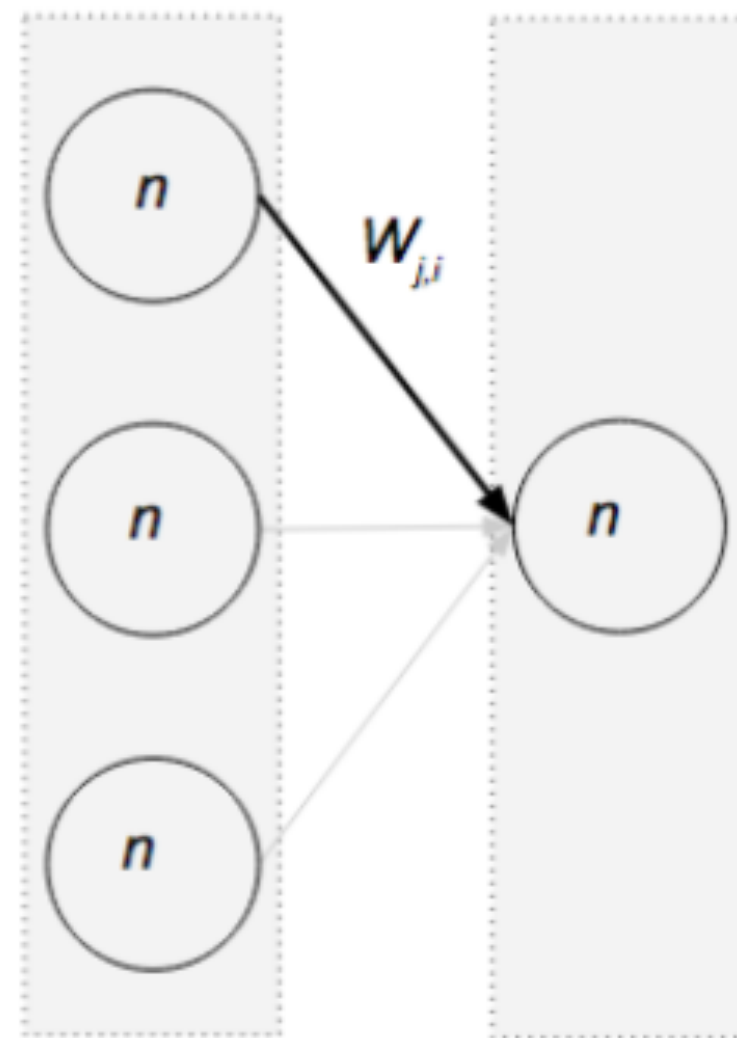
Inovação





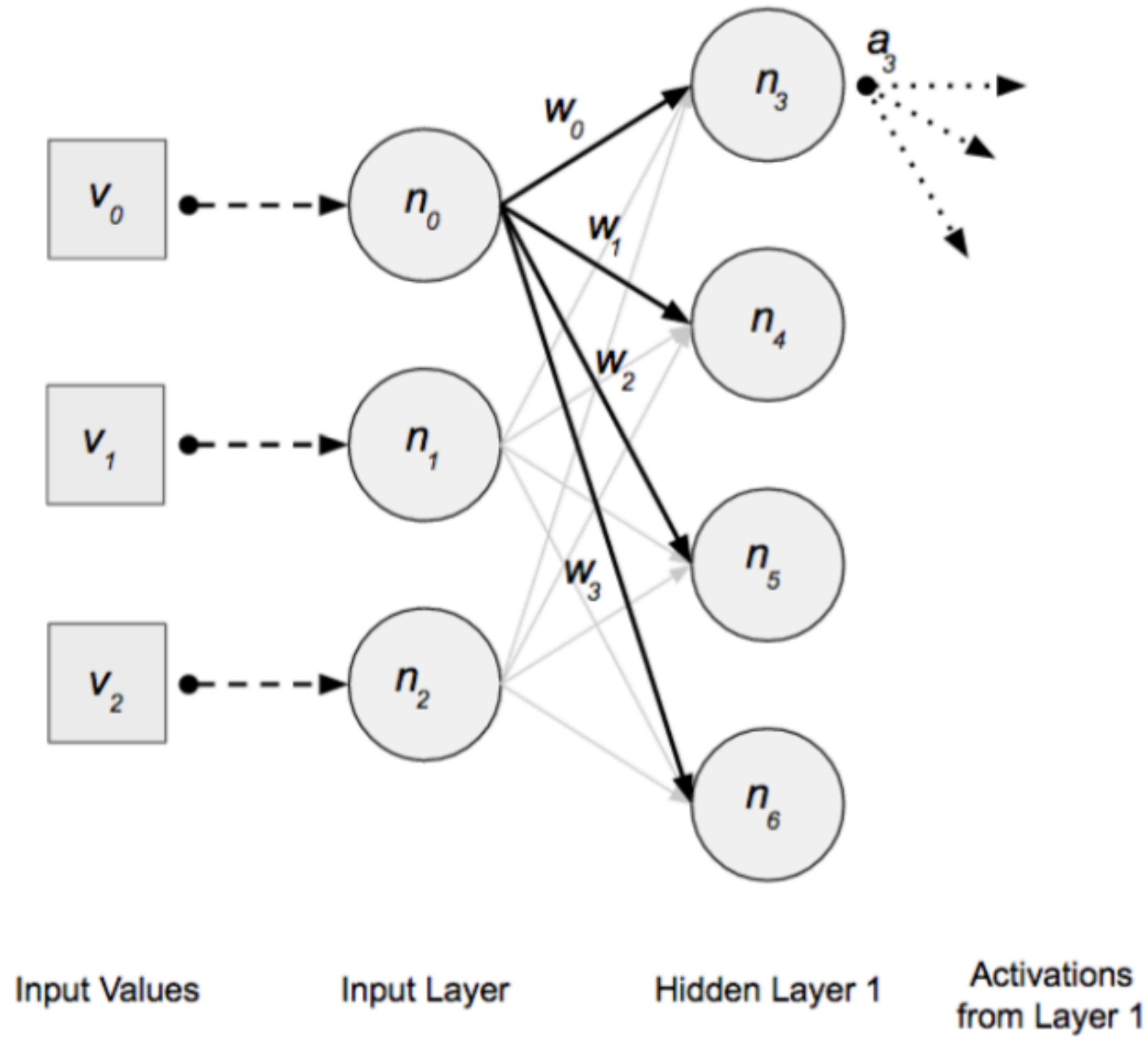


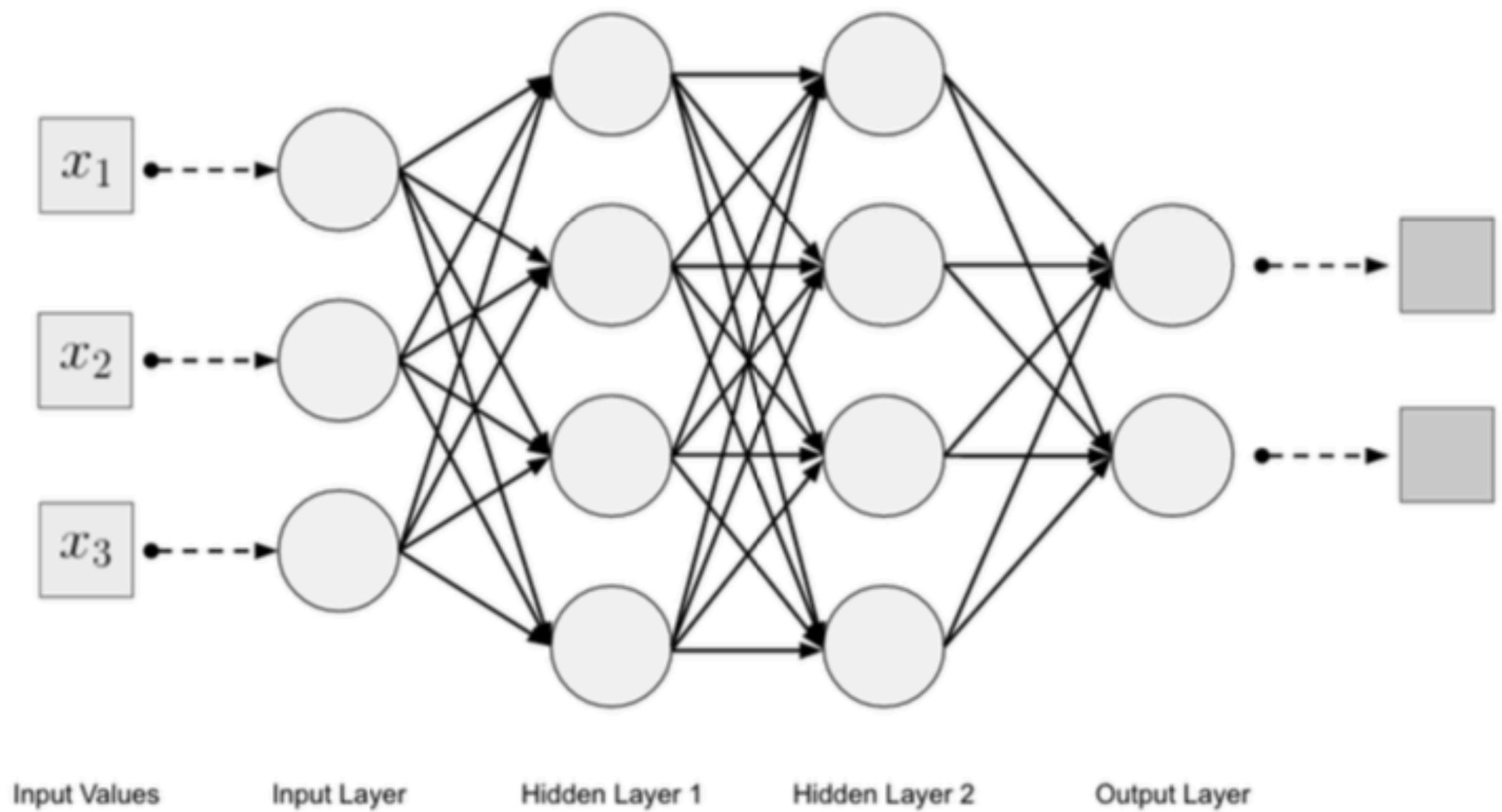


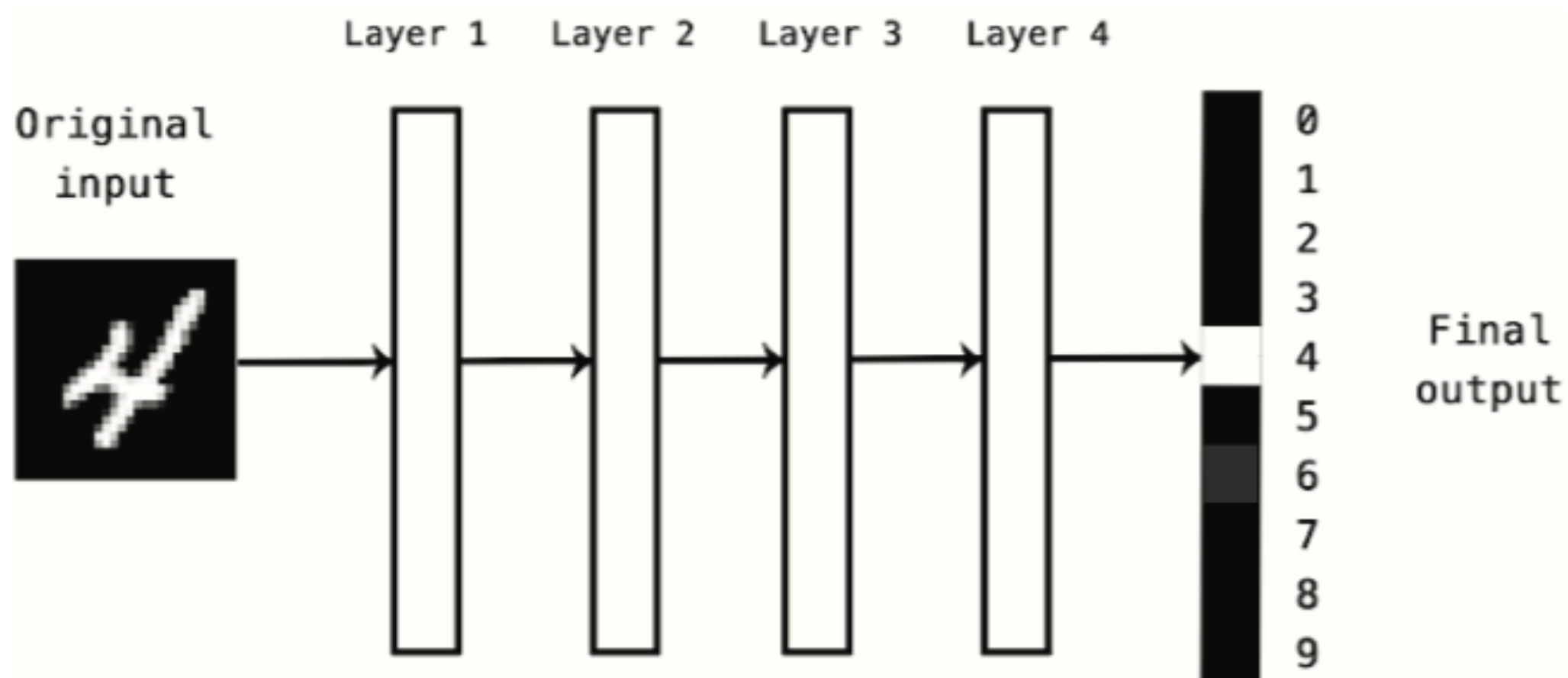


Hidden Layer
(layer j)

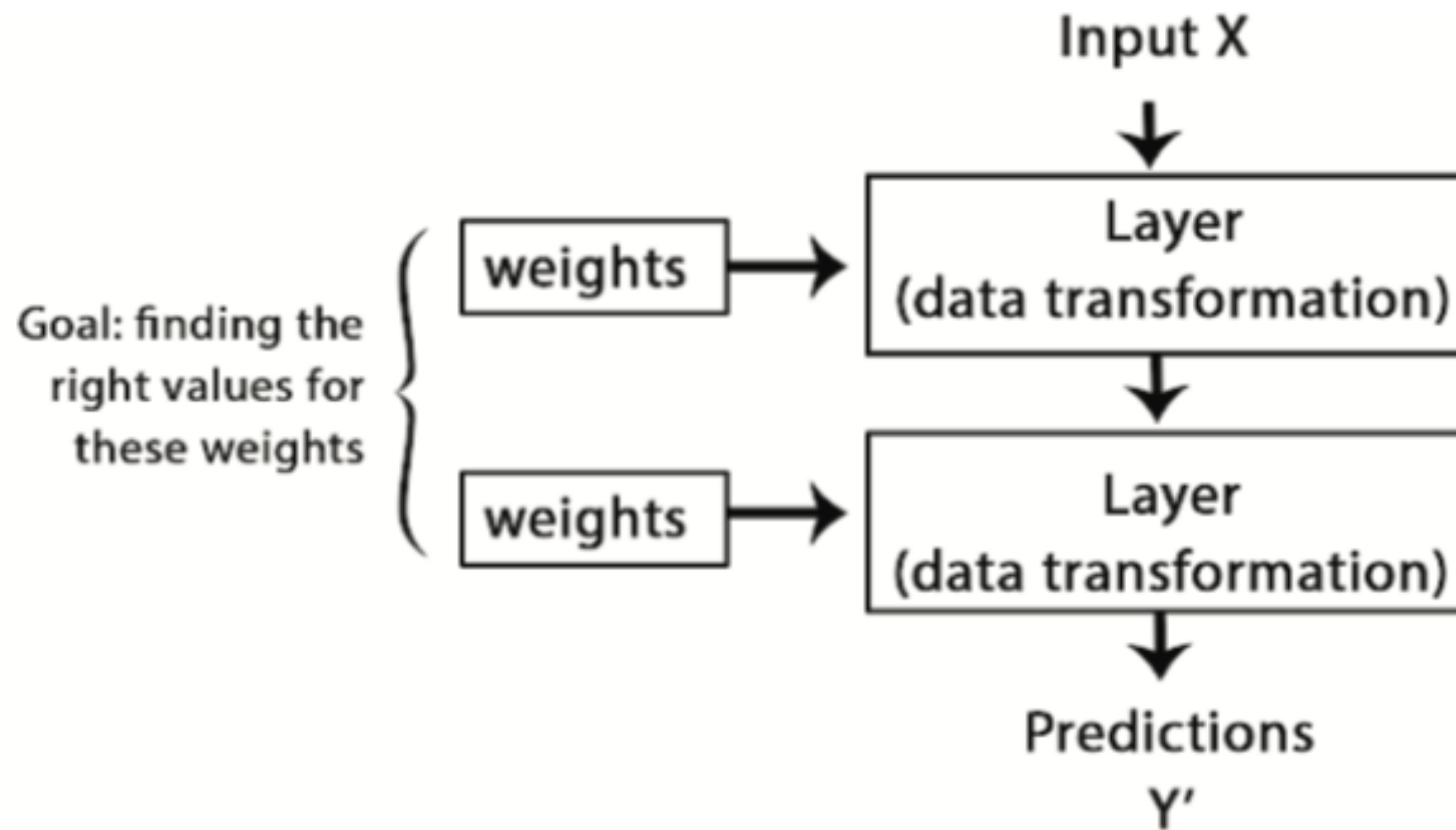
Output Layer
(layer i)



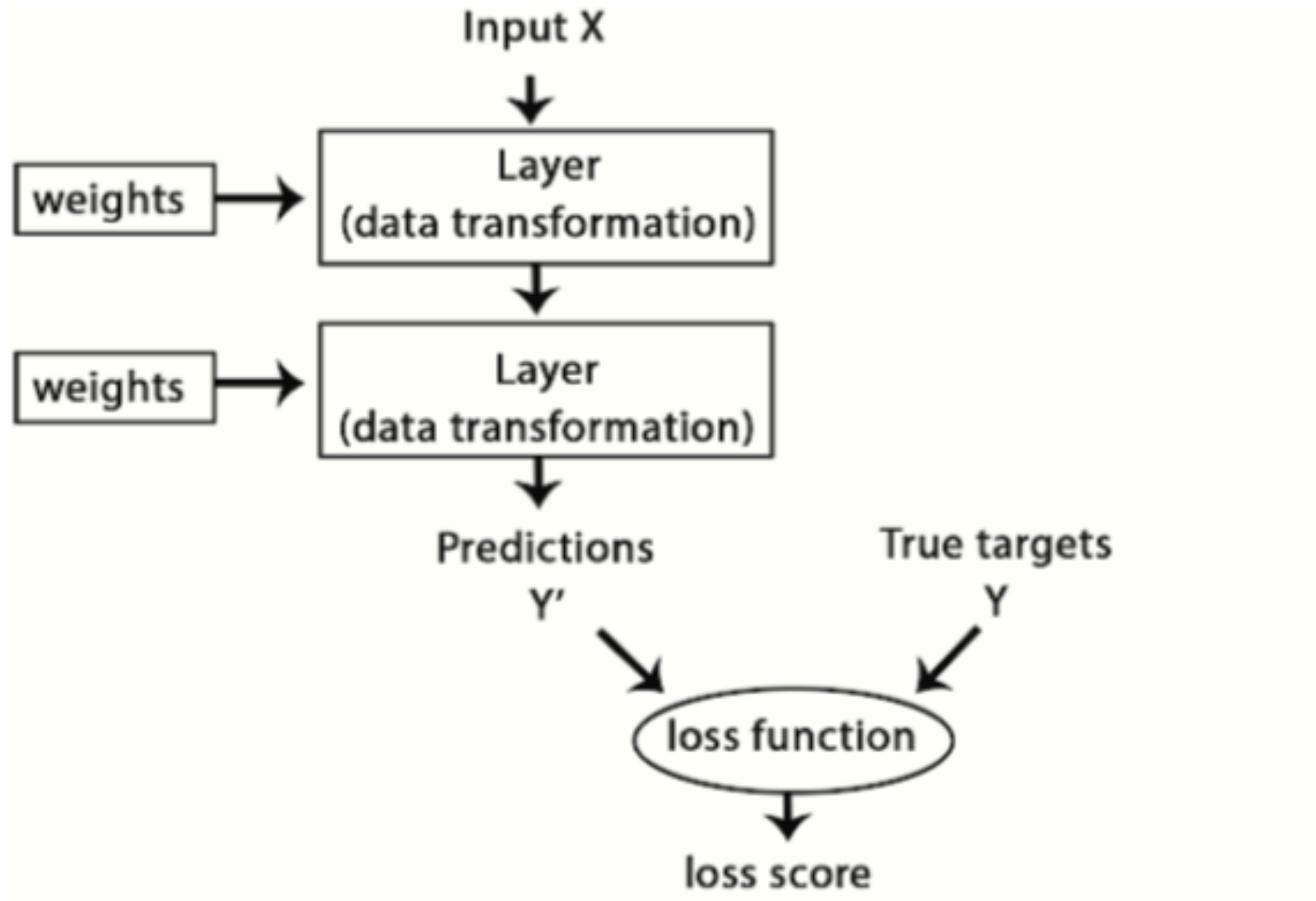




Uma rede neural é parametrizada por seus pesos



Uma função de perda mede a qualidade da saída da rede

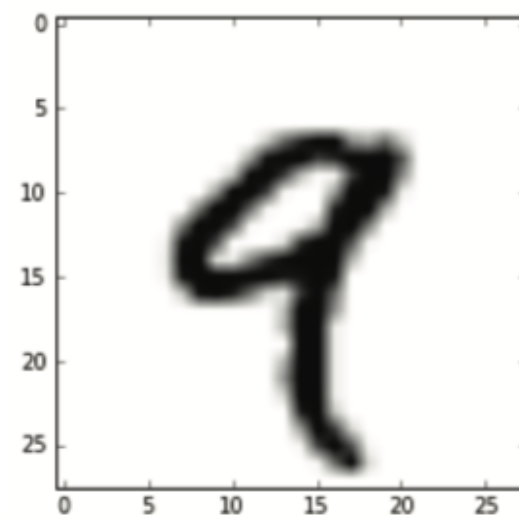


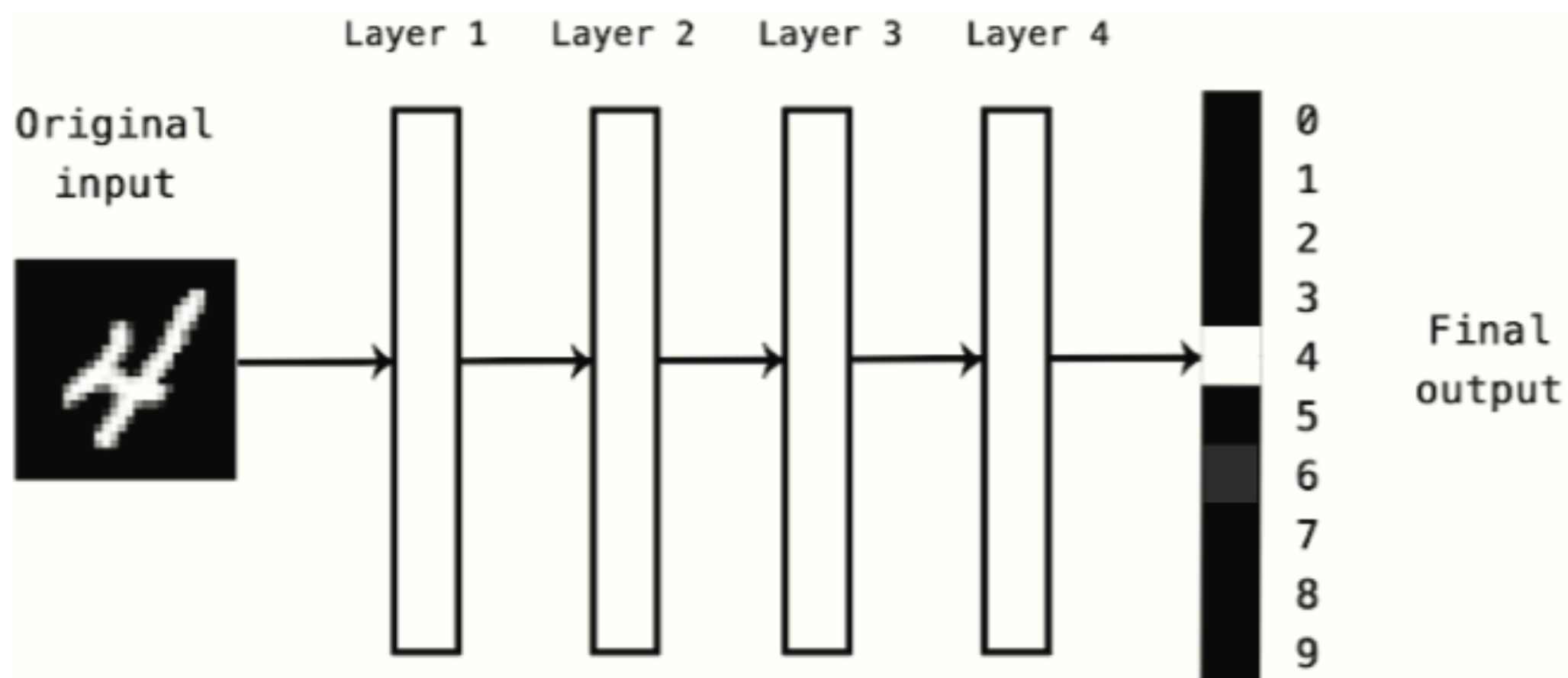
Exemplo: MSNIST

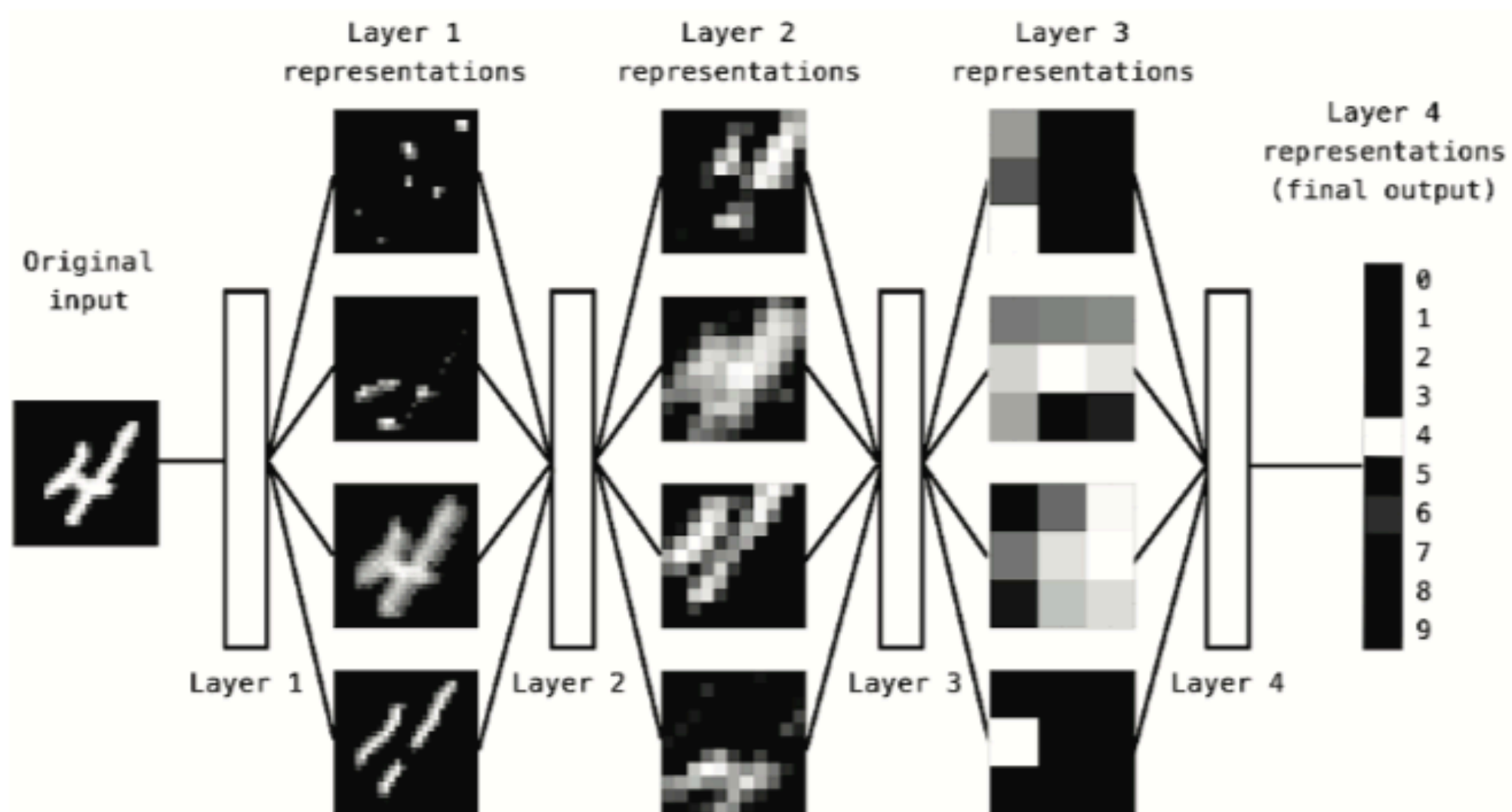
- O problema que vamos tentar resolver neste exemplo é o seguinte:
- Classificar imagens em grayscale de dígitos manuscritos (28 pixels por 28 pixels), em suas 10 categorias (0 a 9).
- O conjunto de dados que usamos é o conjunto de dados MNIST, um conjunto de dados clássico na comunidade de aprendizagem de máquinas, que tem sido em torno de quase tanto quanto o campo em si e tem sido muito estudado.

Exemplo: MNIST

- É um conjunto de 60.000 imagens de treinamento, mais 10.000 imagens de teste, montadas pelo Instituto Nacional de Padrões e Tecnologia (o NIST em MNIST) na década de 1980.
- Você pode considerar em "resolver" MNIST como sendo o "Olá Mundo" ("Hello World") de Deep Learning - é o que você faz para verificar se seus algoritmos funcionam como esperado. À medida que você se tornar um profissional de machine learning, você verá MNIST surgir muitas vezes, em artigos científicos, postagens de blog e assim por diante.







Packages usadas

- Conda install tensorflow
- Conda install keras


```
Zirelium:Workshop Zirilie$ conda install tensorflow
```

```
Fetching package metadata .....
```

```
Solving package specifications: .
```

```
Package plan for installation in environment /Users/Zirilie/anaconda
:
```

```
The following NEW packages will be INSTALLED:
```

```
[ libprotobuf: 3.2.0-0 ]
[ protobuf: 3.2.0-py36_0 ]
[ tensorflow: 1.1.0-np112py36_0 ]
```

```
The following packages will be UPDATED:
```

```
conda: 4.3.21-py36_0 --> 4.3.27-py36hb556a21_0
```

```
Proceed ([y]/n)? y
```

libprotobuf-3.2.0	100%	#####	Time: 0:00:01	4.39 MB/s
protobuf-3.2.0	100%	#####	Time: 0:00:00	6.56 MB/s
tensorflow-1.1.0	100%	#####	Time: 0:00:04	6.06 MB/s
conda-4.3.27-py36_0	100%	#####	Time: 0:00:00	6.53 MB/s

```
[Zirelium:Workshop Zirilie$ conda install keras ]
Fetching package metadata .....
Solving package specifications: .

Package plan for installation in environment /Users/Zirilie/anaconda
:

The following NEW packages will be INSTALLED:

    keras:      2.0.8-py36h39110e4_0

The following packages will be UPDATED:

    conda:      4.3.27-py36hb556a21_0 --> 4.3.30-py36h173c244_0

The following packages will be SUPERSEDED by a higher-priority channel:

    conda-env:  2.6.0-0 --> 2.6.0-h36134e3_0

Proceed ([y]/n)? y

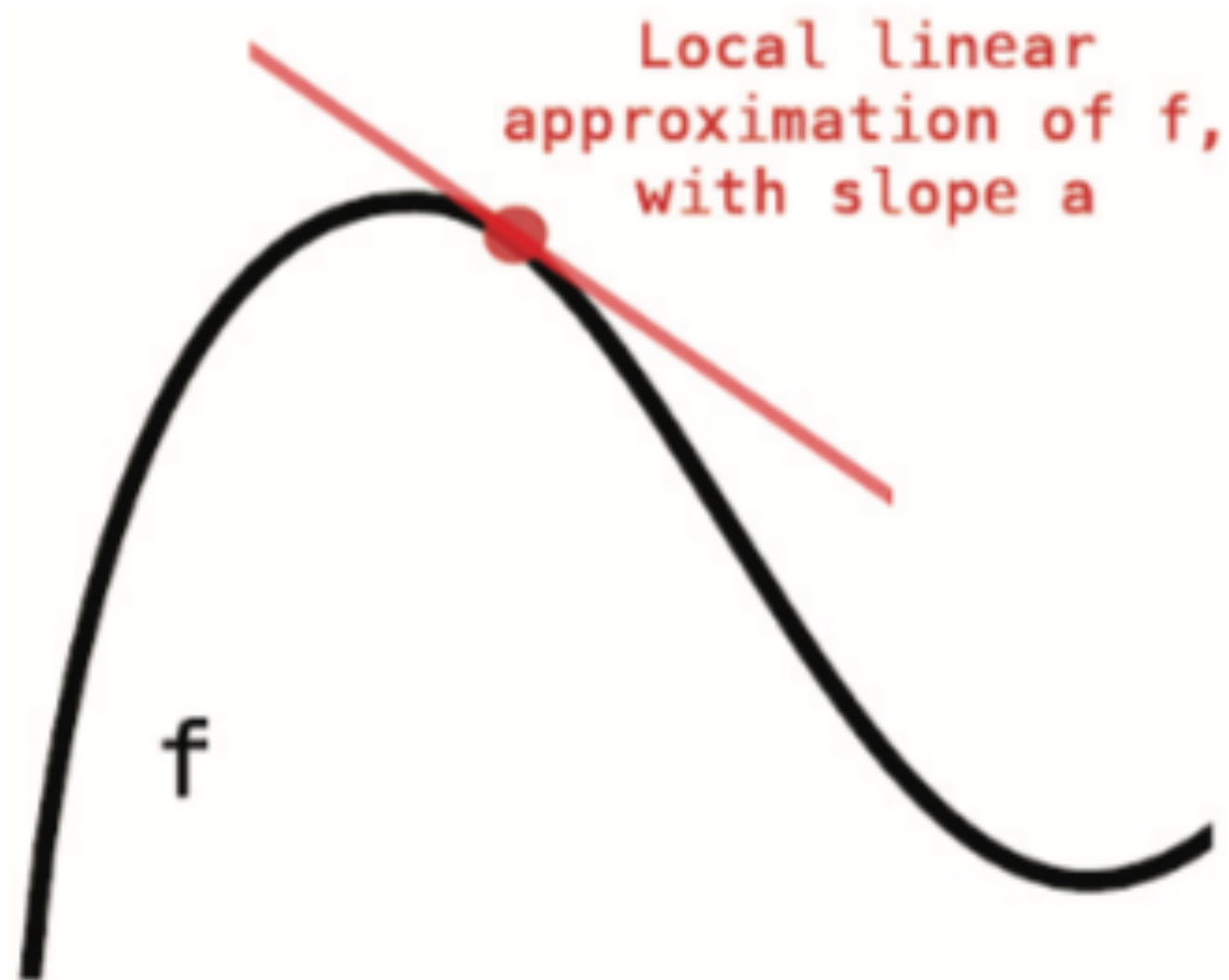
conda-env-2.6. 100% |#####| Time: 0:00:00 388.97 kB/s
keras-2.0.8-py 100% |#####| Time: 0:00:01 260.33 kB/s
conda-4.3.30-p 100% |#####| Time: 0:00:01 356.39 kB/s
Zirelium:Workshop Zirilie$ _
```

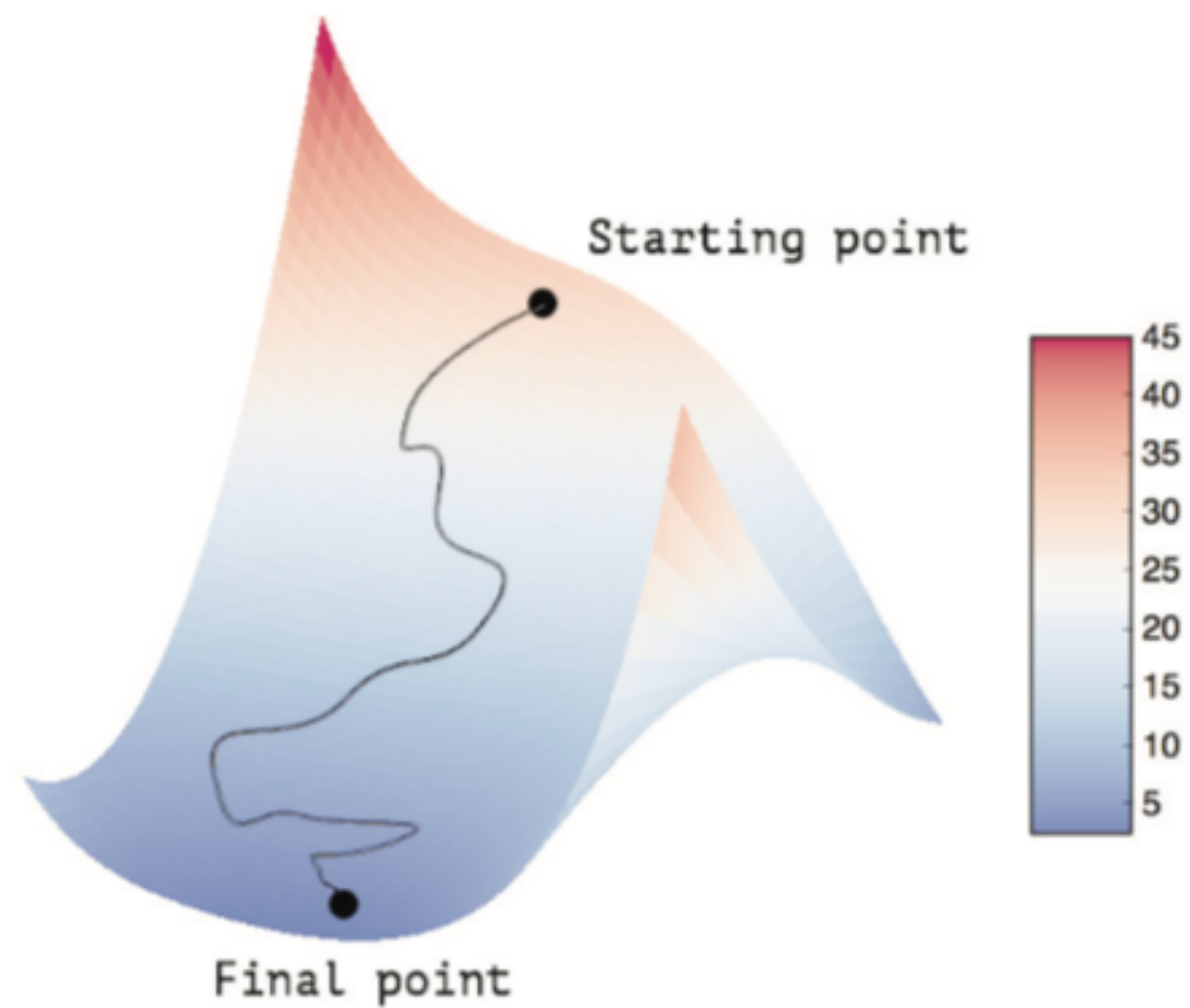
Treinando

- Para tornar nossa rede pronta para treinamento, precisamos escolher mais três coisas, como parte da etapa de "compilação":
 - - **Uma função de perda (a loss function)**: é como a rede será capaz de medir o quão bom é o trabalho que está fazendo nos dados de treinamento e, assim, como será capaz de orientar-se na direção certa.
 - - **Um otimizador (an optimizer)**: este é o mecanismo através do qual a rede se atualizará com base nos dados que vê e na sua perda acima.
 - - **Métricas (metrics)**: para monitorar durante treinamento e testes. Aqui, nos preocupamos somente com a precisão.

Algunas bases matemáticas de Deep Learning

Derivada de uma função f num ponto p





A gradient descent (descida do gradiente) numa superfície de perda 2D

- Dada uma **função diferenciável**, é teoricamente possível encontrar seu mínimo analiticamente:
- Sabe-se que **uma função é mínima num ponto em que a sua derivada é 0**, então tudo o que você tem que fazer é encontrar todos os pontos onde a derivada vai para 0 e verificar em quais desses pontos a função tem o valor **mais baixo**.
- Aplicado a uma rede neural, isso significaria encontrar analiticamente **a combinação de valores de pesos que produzem a menor função de perda possível**.

```
network.compile(optimizer='rmsprop',  
                loss='categorical_crossentropy',  
                metrics=[ 'accuracy' ])
```

- **categorical_crossentropy** é uma *função de perda* que é usada como sinal de feedback para aprender os pesos dos tensores, que a fase de treinamento tentará minimizar.
- Essa diminuição da perda ocorre por meio de uma **diminuição de gradiente estocástica**. As regras exatas que regem o uso específico da descida de gradiente são definidas pelo otimizador **rmsprop** passado como o primeiro argumento.
- A métrica usada é a **acurácia**.

A seguir

- Recomendamos fazer a seguir os seguintes workshops que ofereceremos:
- **Modelagem Analítica de Machine Learning**
- **Deep Learning**