Mathematical techniques in data science

Lecture 5: Neural networks

Reminders

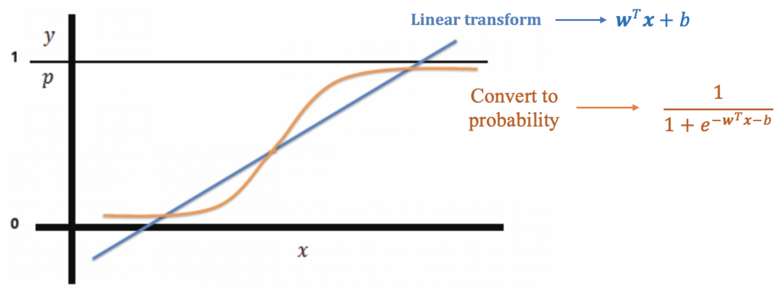
- Homework 1: due 09/29
- Sign up for group projects by the end of Week 5

Logistic regression

- Data point $(\mathbf{x}, y)$ where
  - $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ is a vector with d features
  - y is the label (0 or 1)
- Logistic regression models $P[y = 1 | X = \mathbf{x}]$

Logistic regression

$$\boldsymbol{x} \xrightarrow{\text{Linear transform}} \boldsymbol{w}^T\boldsymbol{x} + b \xrightarrow{\substack{\text{Convert to} \\ \text{probability}}} \frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x} - b}}$$



y
1
p
0
x
Linear transform
$\boldsymbol{w}^T\boldsymbol{x} + b$
Convert to probability
$\frac{1}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x} - b}}$

Logistic regression with more than 2 classes

- Suppose now the response can take any of $\{1, \dots, K\}$ values
- We use the categorical distribution instead of the Bernoulli distribution

$$P[Y = k | X = \mathbf{x}] = p_k(\mathbf{x}), \quad \sum_{k=1}^{K} p_k(\mathbf{x}) = 1.$$

- Model

$$p_k(\mathbf{x}) = \frac{e^{w_k^T \mathbf{x}_k + b_k}}{\sum_{k=1}^{K} e^{w_k^T \mathbf{x}_k + b_k}}$$

Softmax function

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \longrightarrow \boxed{\dfrac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}} \longrightarrow \begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Logistic regression: pros and cons

Pros:

- Simple algorithm
- Prediction is fast
- Easy to implement
- The forward map has a closed-form formula of the derivatives

$$\frac{\partial \ell}{\partial \beta_j}(\beta) = \sum_{i=1}^{n} \left[y_i x_{ij} - x_{ij} \frac{e^{x_i^T \beta}}{1 + e^{x_i^T \beta}} \right].$$

Cons:

- Linear model

How to make logistic regression better?

We want a model that

- computes the derivatives (of the objective function, with respect to the parameters) easily
- can capture complex relationships

This is difficult because complex models often have high numbers of parameters and don't have closed-form derivatives, and computations of

$$\frac{\partial \ell}{\partial \beta_i}(\beta, x) \approx \frac{\ell(\beta + \epsilon_i, x) - \ell(\beta, x)}{\epsilon_i}$$

are costly (and unstable)

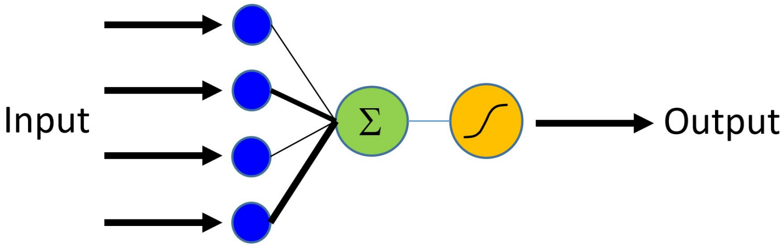
Ideas

- Automatic differentiation and back-propagation
- Ideas:
    - Organizing information using graphs (networks)
    - Chain rule

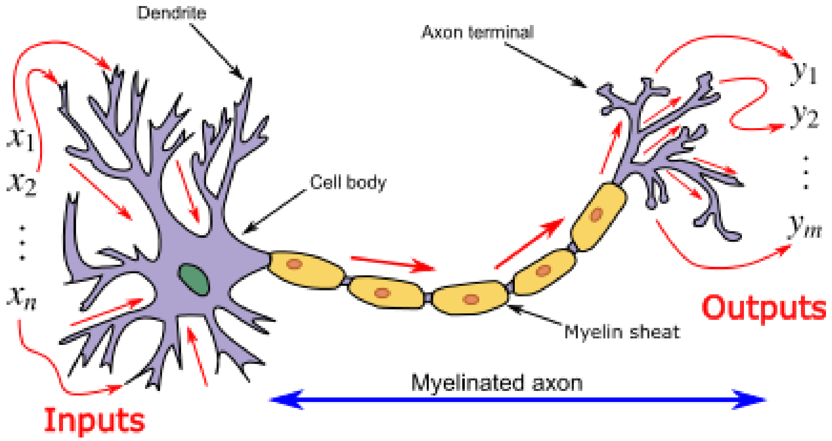
$$(f \circ g)'(x) = f'(g(x))g'(x)$$

Neural networks

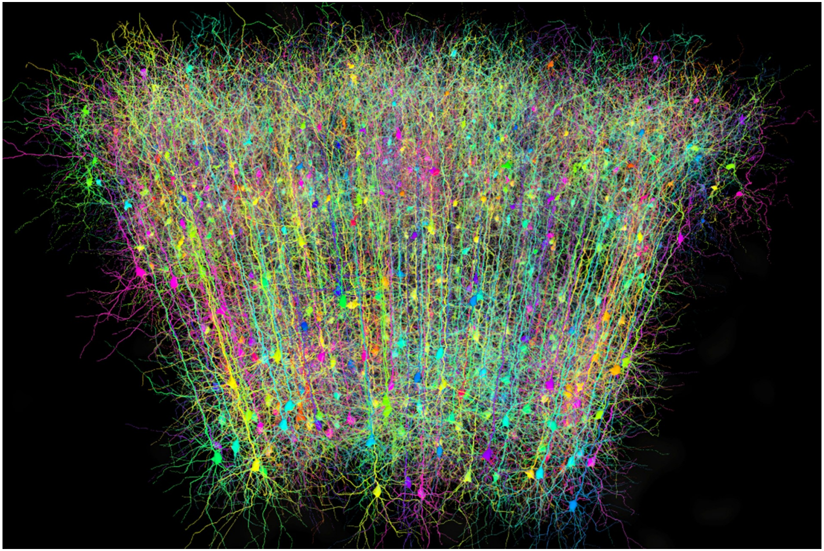
Logistic neuron



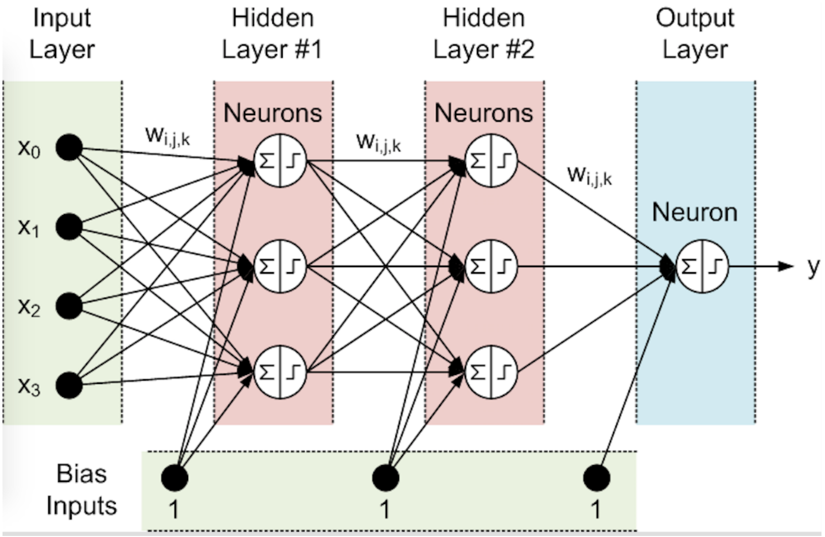
Why neuron?


Dendrite
Axon terminal
x_1
x_2
$\vdots$
x_n
Cell body
y_1
y_2
$\vdots$
y_m
Outputs
Myelin sheat
Myelinated axon
Inputs

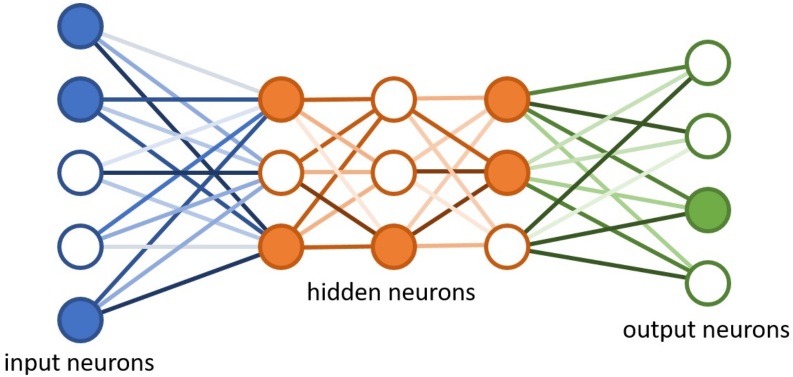
Neural circuit



Feed-forward neural networks



Feed-forward neural networks

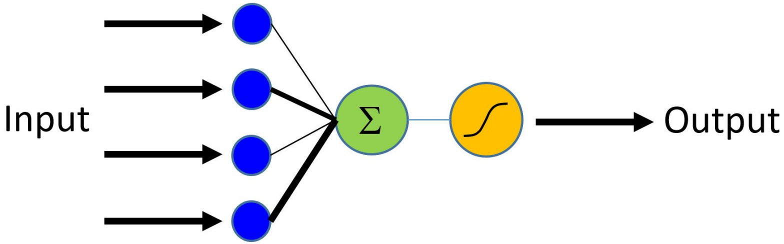


Feed-forward neural networks

- Structure:
    - Graphical representation
    - Activation functions
- Training:
    - Loss functions
    - Stochastic gradient descent
    - Back-propagation

Activation functions

Activation functions



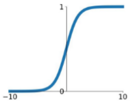
If we do not apply an activation function, then the output signal would simply be a simple linear function of the input signals

Activation functions

Activation Functions

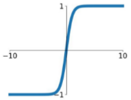
Sigmoid

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



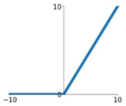
tanh

$\tanh(x)$



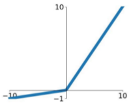
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

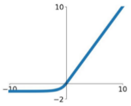


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

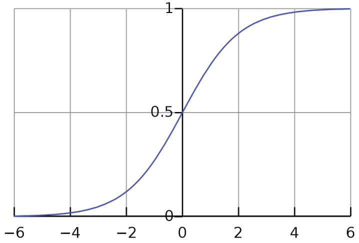
ELU

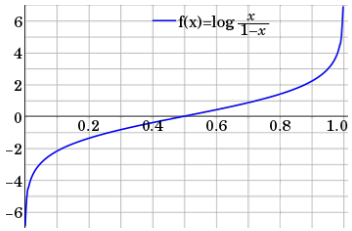
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Logistic function (sigmoid function)

Transformation between $(-\infty, \infty)$ and $[0, 1]$

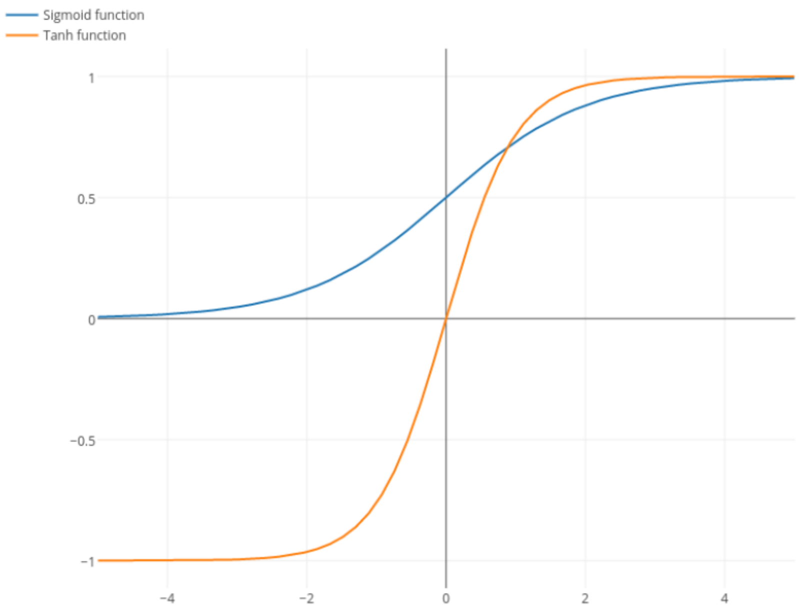




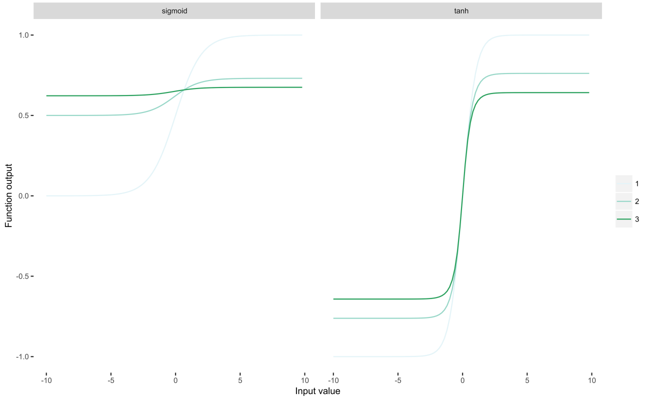
$$f(x) = \frac{e^x}{1 + e^x}$$

$$logit(p) = \log \frac{p}{1 - p}$$

Hyperbolic tangent

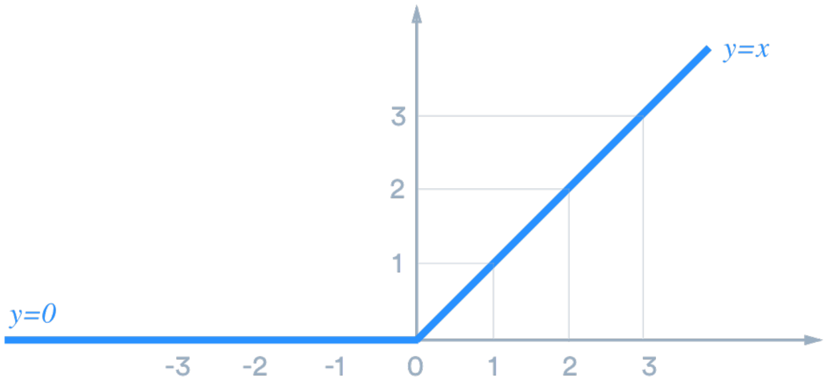


Hyperbolic tangent

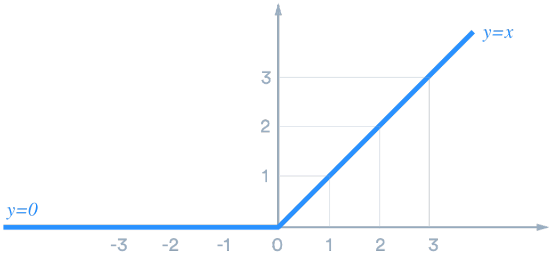


Vanishing gradient problem

Rectified linear unit (ReLU)



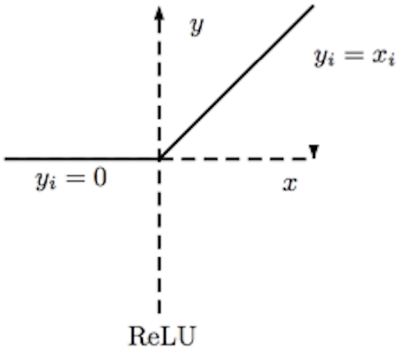
Rectified linear unit (ReLU)



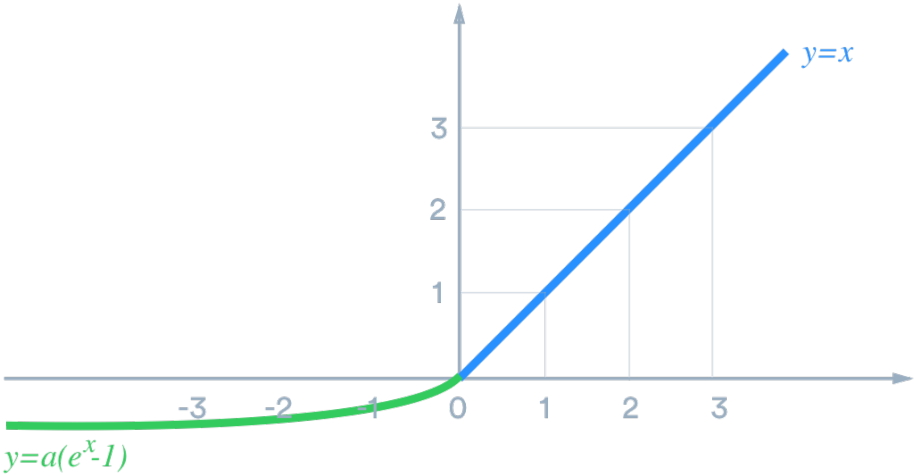
Advantage: model sparsity, cheap to compute (no complicated math), partially address the vanishing gradient problem

Issue: Dying ReLU

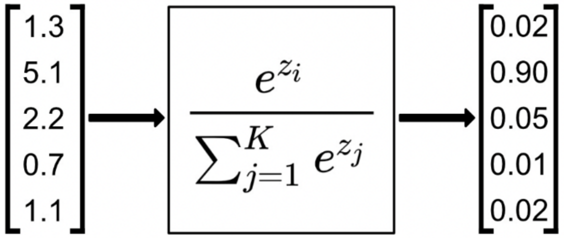
Leaky relu



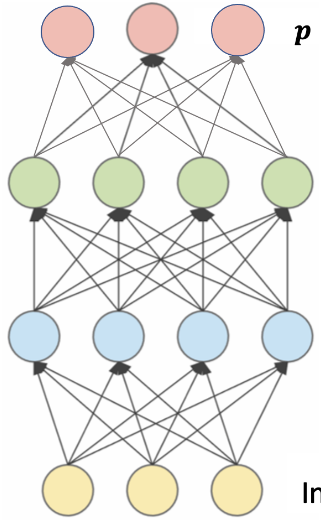
Exponential Linear Unit (ELU, SELU)



Softmax function


$$\begin{bmatrix}1.3\\5.1\\2.2\\0.7\\1.1\end{bmatrix} \longrightarrow \boxed{\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}} \longrightarrow \begin{bmatrix}0.02\\0.90\\0.05\\0.01\\0.02\end{bmatrix}$$

Feed-forward neural networks (multi-class classification)



$\boldsymbol{p} = \text{softmax}(\boldsymbol{W}_3^T \boldsymbol{h}_2 + \boldsymbol{b}_3)$

$\boldsymbol{h}_2 = \sigma(\boldsymbol{W}_2^T \boldsymbol{h}_1 + \boldsymbol{b}_2)$

$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_1)$

Input: $\boldsymbol{x}$

Feed-forward neural networks

- Structure:
  - Graphical representation
  - Activation functions
- Training:
  - Loss functions
  - Stochastic gradient descent
  - Back-propagation

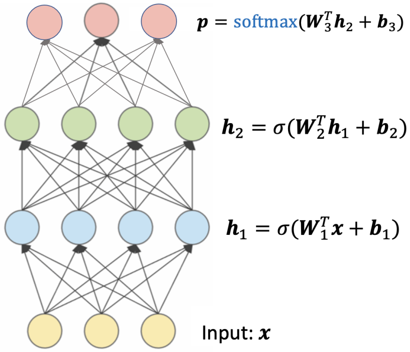
Train feed-forward neural networks

Settings

- Data:
  $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
- Model parameters:

  $$\theta = (W_1, b_1, W_2, b_2, \ldots, W_L, b_L)$$

- Training: Find the best value of θ that fits the data


$\boldsymbol{p} = \text{softmax}(\boldsymbol{W}_3^T \boldsymbol{h}_2 + \boldsymbol{b}_3)$
$\boldsymbol{h}_2 = \sigma(\boldsymbol{W}_2^T \boldsymbol{h}_1 + \boldsymbol{b}_2)$
$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_1)$
Input: $\boldsymbol{x}$

Maximum-likelihood method

- Average log-likelihood

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \log P(y = y_i | \mathbf{x}_i, \theta)$$

- Model parameters:

$$\theta = (W_1, b_1, W_2, b_2, \dots, W_L, b_L)$$

- Training: Maximize $\mathcal{L}(\theta)$

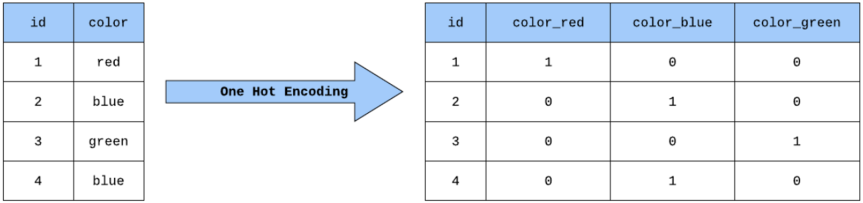
Cross-entropy loss (log loss)

- Cross-entropy loss = negative log-likelihood:

$$\ell(\theta) = -\mathcal{L}(\theta)$$

- Goal: Minimize $\ell(\theta)$

One-hot encoding



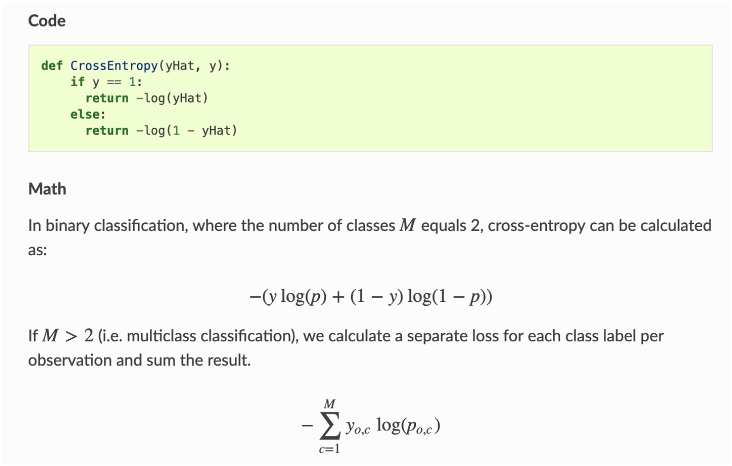
id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Convert a categorical value into a binary vector with exactly one "1" element, and the rest are 0

Loss function for classification: cross-entropy



Code

```
def CrossEntropy(yHat, y):
    if y == 1:
      return -log(yHat)
    else:
      return -log(1 - yHat)
```

Math

In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

$$-(y\log(p) + (1-y)\log(1-p))$$

If $M > 2$ (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

Note: Here $y_{o,:}$ is the one-hot encoding of the label and $p_{o,c}$ is the predicted probability for the observation o is of class c, respectively

Stochastic gradient descent

Gradient descent

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

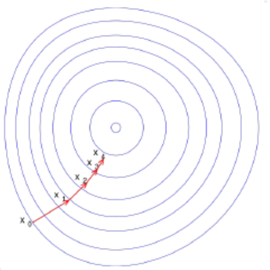
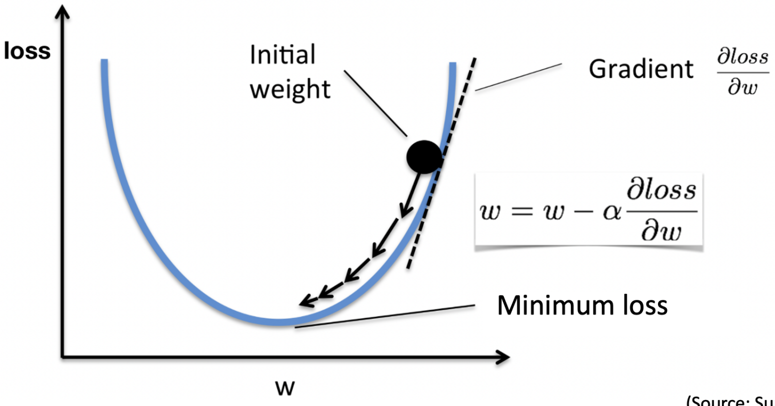


Figure: Gradient Descent. Source:

Gradient descent


loss
Initial weight
Gradient $\frac{\partial loss}{\partial w}$
$w = w - \alpha \frac{\partial loss}{\partial w}$
Minimum loss
w

(Source: Sung Kim)

Stochastic gradient descent

- Recall that our objective function has the form

$$\ell(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(\theta, x_i, y_i)$$

- Mini-batch stochastic gradient descent
  - randomly shuffle examples in the training set, divide them into k mini-batches of data of size m
  - for each batch I_i (i=1, . . . , k), approximate the empirical risk by

$$\hat{\ell}(\theta) = \frac{1}{m}\sum_{j\in I_i} L(\theta, x_j, y_j)$$

    and update θ

$$\theta \leftarrow \theta - \rho\nabla\hat{\ell}(\theta)$$

  - Repeat until an approximate minimum is obtained or a maximum numbers M epochs are done

Stochastic gradient descent: teminology

- Mini-batch stochastic gradient descent
  - randomly shuffle examples in the training set, divide them into k mini-batches of data of size m
  - for each batch I_i (i=1, . . . , k), approximate the objective function by

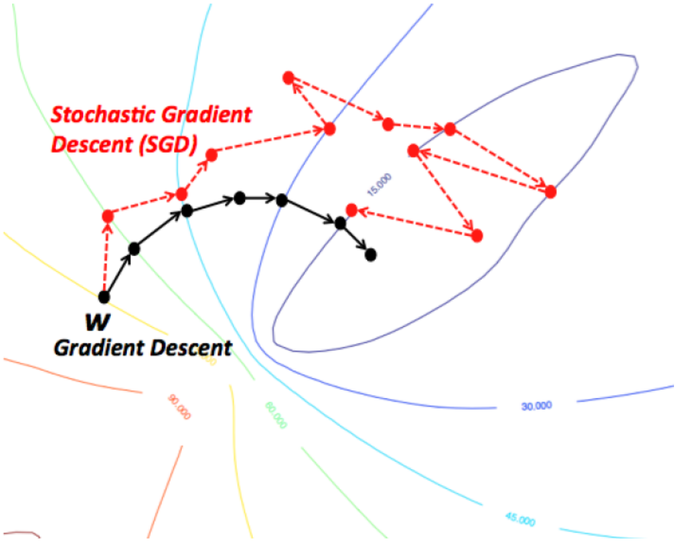
$$\hat{\ell}(\theta) = \frac{1}{m} \sum_{j \in I_i} L(\theta, x_j, y_j)$$

    and update θ

$$\theta \leftarrow \theta - \rho \nabla \hat{\ell}(\theta)$$

  - Repeat until an approximate minimum is obtained or a maximum numbers M epochs are done
- Terminology:
  - m: batch-size
  - ρ: learning rate
  - M: number of epochs

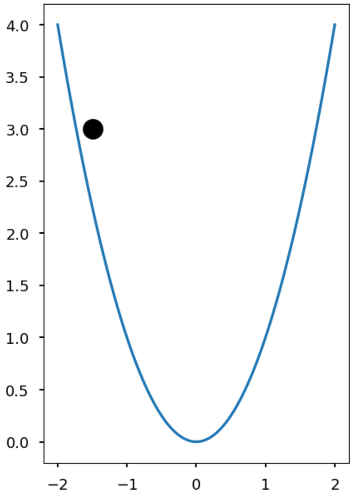
Stochastic gradient descent (SGD)

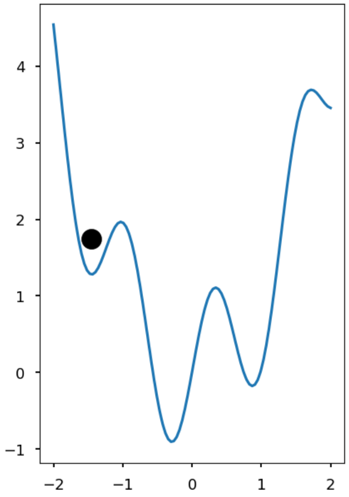


Stochastic gradient descent

- Gradient descent converges to the local minimum, and the fluctuation is small
- SGD's fluctuation is large, but enables jumping to new/better local minima

Escaping local minima





Automatic diffierentiation

Stochastic gradient descent

- The most computationally heavy part in the training of a neural net is to compute
$$\frac{\partial \ell}{\partial \theta_{i,j}}$$
- Numerical differentiation is not realistic, and symbolic differentiation is impossible

Automatic differentiation

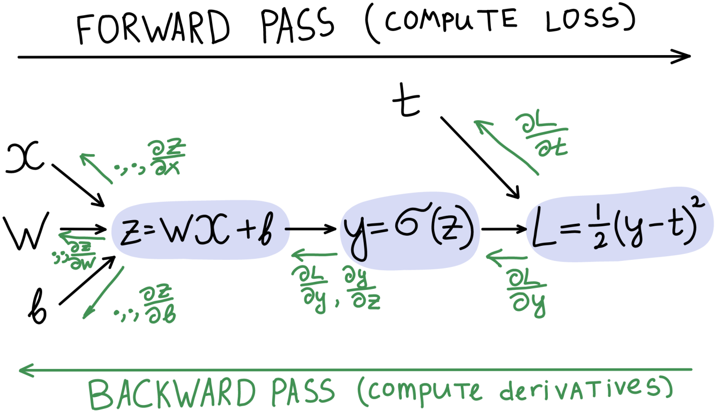
- Assume that

$$y = f(g(h(x)))$$

- Denote $x = u_0$, $h(u_0) = u_1$, $g(u_1) = u_2$, $f(u_2) = u_3 = y$, then

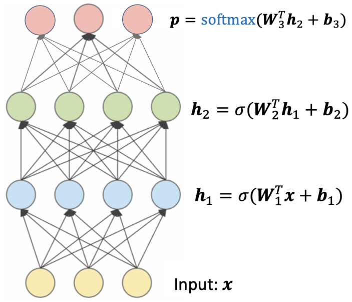
$$\frac{dy}{du_i} = \frac{dy}{du_{i+1}} \frac{du_{i+1}}{du_i}$$

Back-propagation



FORWARD PASS (COMPUTE LOSS)
x
W
b
t
$z = Wx + b$
$y = \sigma(z)$
$L = \frac{1}{2}(y-t)^2$
$\ldots, \frac{\partial z}{\partial x}$
$\ldots, \frac{\partial z}{\partial W}$
$\ldots, \frac{\partial z}{\partial b}$
$\frac{\partial L}{\partial y}, \frac{\partial y}{\partial z}$
$\frac{\partial L}{\partial y}$
$\frac{\partial L}{\partial t}$
BACKWARD PASS (compute derivatives)

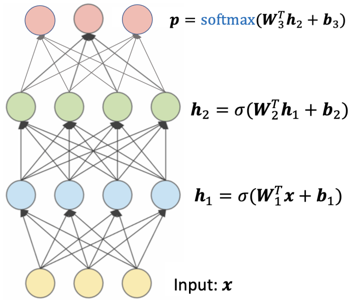
Back-propagation


$\boldsymbol{p} = \text{softmax}(\boldsymbol{W}_3^T \boldsymbol{h}_2 + \boldsymbol{b}_3)$
$\boldsymbol{h}_2 = \sigma(\boldsymbol{W}_2^T \boldsymbol{h}_1 + \boldsymbol{b}_2)$
$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_1)$
Input: $\boldsymbol{x}$

Use chain rule to compute $\nabla \ell(\theta)$

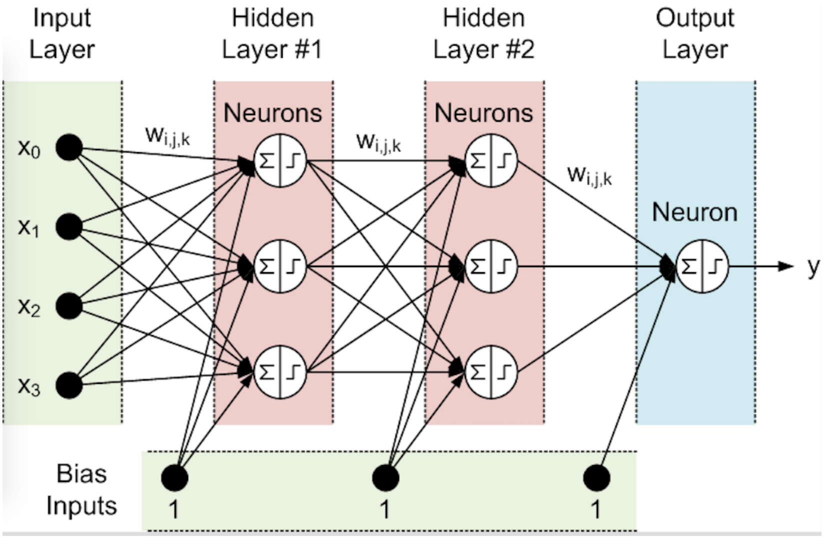
$$\frac{\partial \ell}{\partial b_1} = \frac{\partial \ell}{\partial p}(p) \cdot \frac{\partial p}{\partial h_2}(h_2, W_3, b_3) \cdot \frac{\partial h_2}{\partial h_1}(h_1, W_2, b_2) \cdot \frac{\partial h_1}{\partial b_1}(x, W_1, b_1)$$

Back-propagation


$\boldsymbol{p} = \text{softmax}(\boldsymbol{W}_3^T \boldsymbol{h}_2 + \boldsymbol{b}_3)$
$\boldsymbol{h}_2 = \sigma(\boldsymbol{W}_2^T \boldsymbol{h}_1 + \boldsymbol{b}_2)$
$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_1)$
Input: $\boldsymbol{x}$

- One forward pass to evaluate h_1, h_2, p, ℓ
- One backward pass to compute $\nabla \ell(\theta)$

Feed-forward neural networks



Back-propagation

- Advantage: The cost to compute the partial derivatives with respect to all parameters are just twice the cost of a forward evaluations
- Drawback: The functions used to describe the network (activation functions and loss functions) needs to belong to the class of functions supported by the computational platform

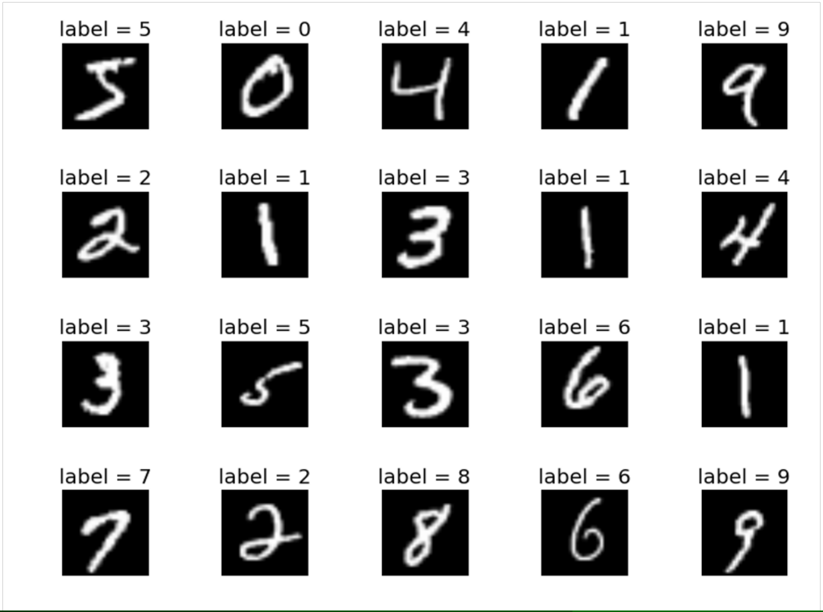
Some intros to computer vision

Computer vision

A field that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs

- Image classification/object recognition
- Object detection
- Image segmentation
- Image generation
- Image style transfer

Image classification



Object detection

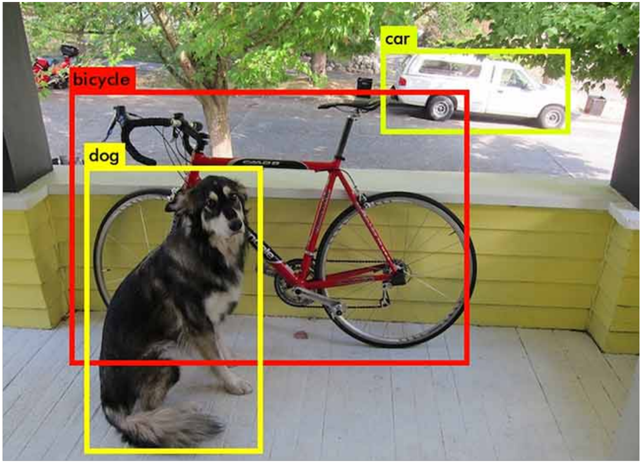


Image segmentation

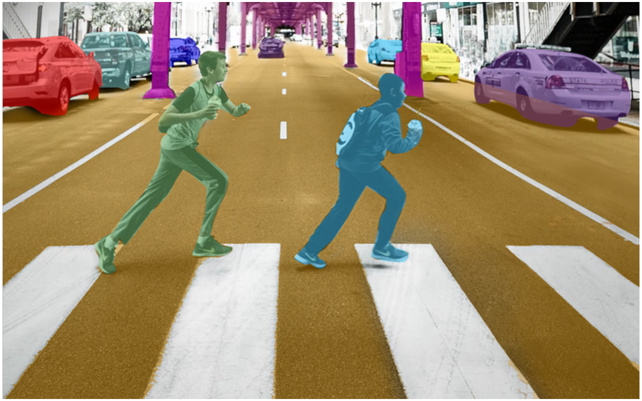


Image generation

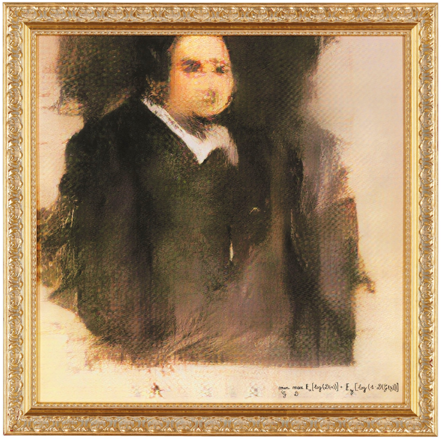
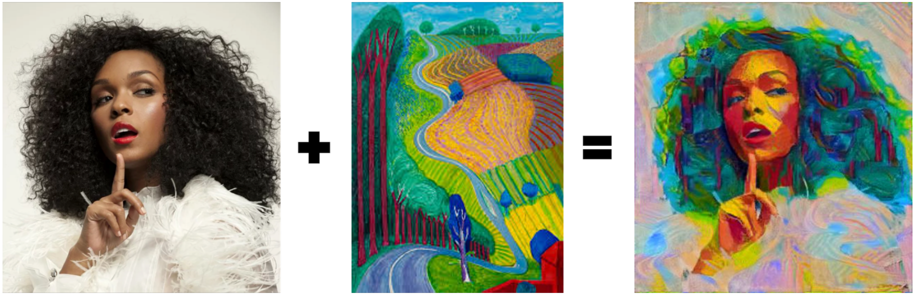
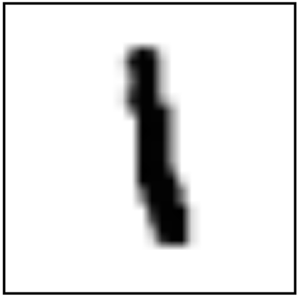

$\min_{G} \max_{D} \mathbb{E}_x[\log(D(x))] + \mathbb{E}_z[\log(1-D(G(z)))]$

Image style transfer

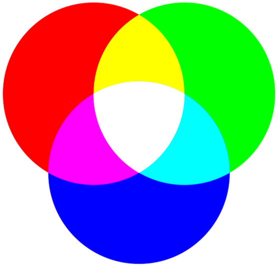


Grayscale image representation



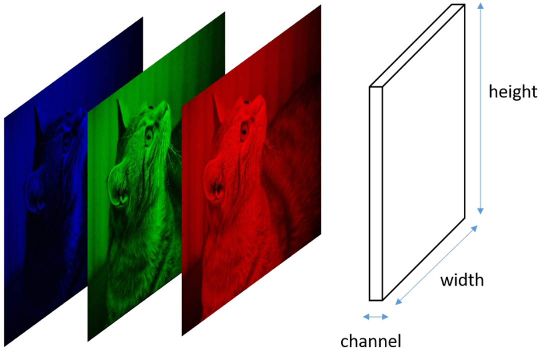
$$
\simeq
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

Color image representation



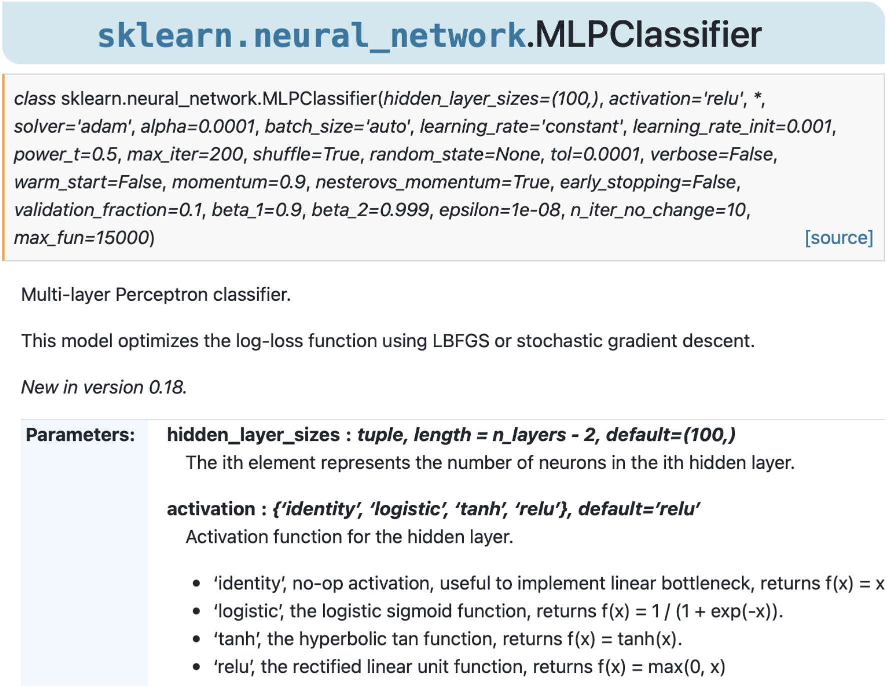
- Use RGB color mode
- Represent a color by 3 values: R (Red) G (Green) B (Blue)
- There are other color modes

Image representation



- An image is an H x W x C matrix: H (height), W (width), C (depth or number of channels)
- Grayscale image: C = 1
- RGB image: C = 3

Demo: train an MLP using Keras



sklearn.neural_network.MLPClassifier

class sklearn.neural_network.MLPClassifier(*hidden_layer_sizes=(100,)*, *activation='relu'*, ***, *solver='adam'*, *alpha=0.0001*, *batch_size='auto'*, *learning_rate='constant'*, *learning_rate_init=0.001*, *power_t=0.5*, *max_iter=200*, *shuffle=True*, *random_state=None*, *tol=0.0001*, *verbose=False*, *warm_start=False*, *momentum=0.9*, *nesterovs_momentum=True*, *early_stopping=False*, *validation_fraction=0.1*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*, *n_iter_no_change=10*, *max_fun=15000*) [source]

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

New in version 0.18.

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*

The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*

Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

K Keras

Simple. Flexible. Powerful.

- High level API for deep learning
- More flexible to define network architecture than sklearn

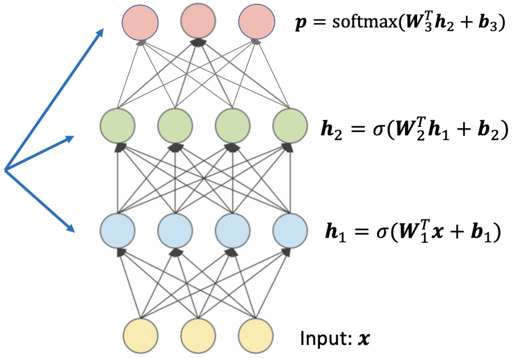
Define network architecture (1)

- Define a network as a Sequential object
- Add layers to it one-by-one

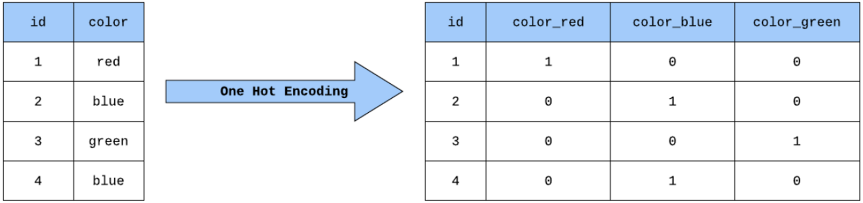
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Define network architecture (2)


Dense /
Fully connected layer
$\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{W}_3^T \boldsymbol{h}_2 + \boldsymbol{b}_3)$
$\boldsymbol{h}_2 = \sigma(\boldsymbol{W}_2^T \boldsymbol{h}_1 + \boldsymbol{b}_2)$
$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{b}_1)$
Input: $\boldsymbol{x}$

One-hot encoding



id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Labels in Keras are usually encoded as one-hot vectors