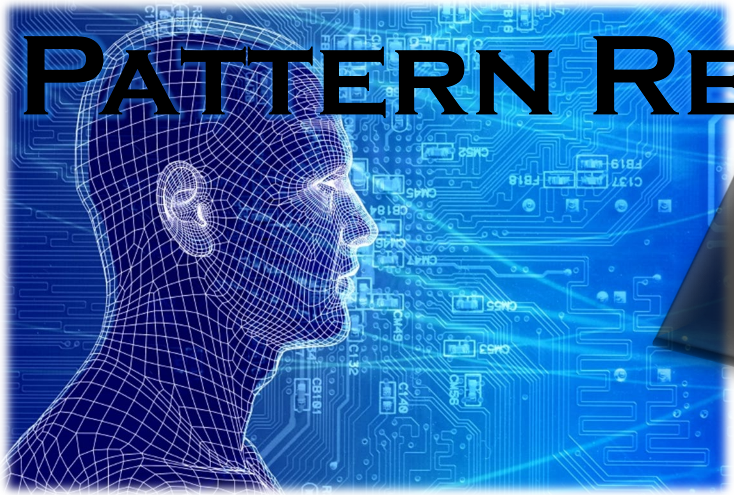
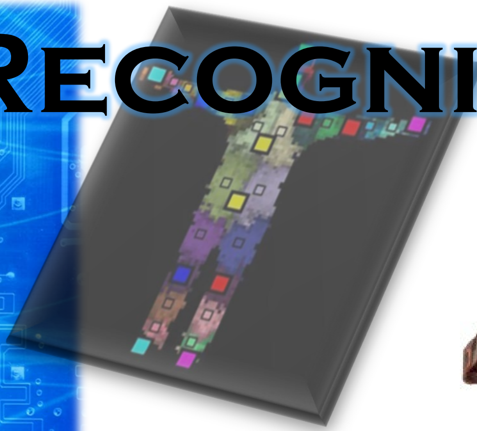
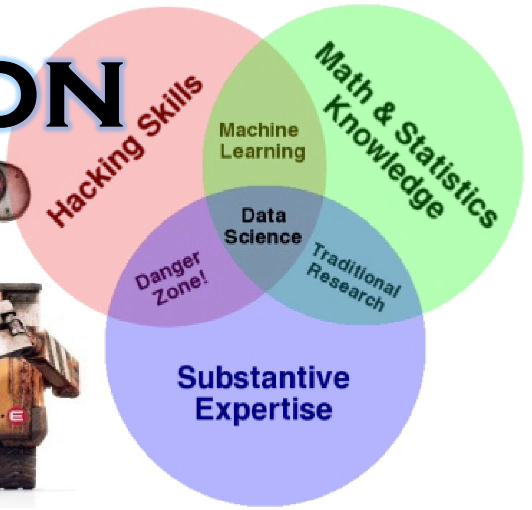
Pattern Recognition



# Neural Networks

PHYS 453

Dr Daugherity

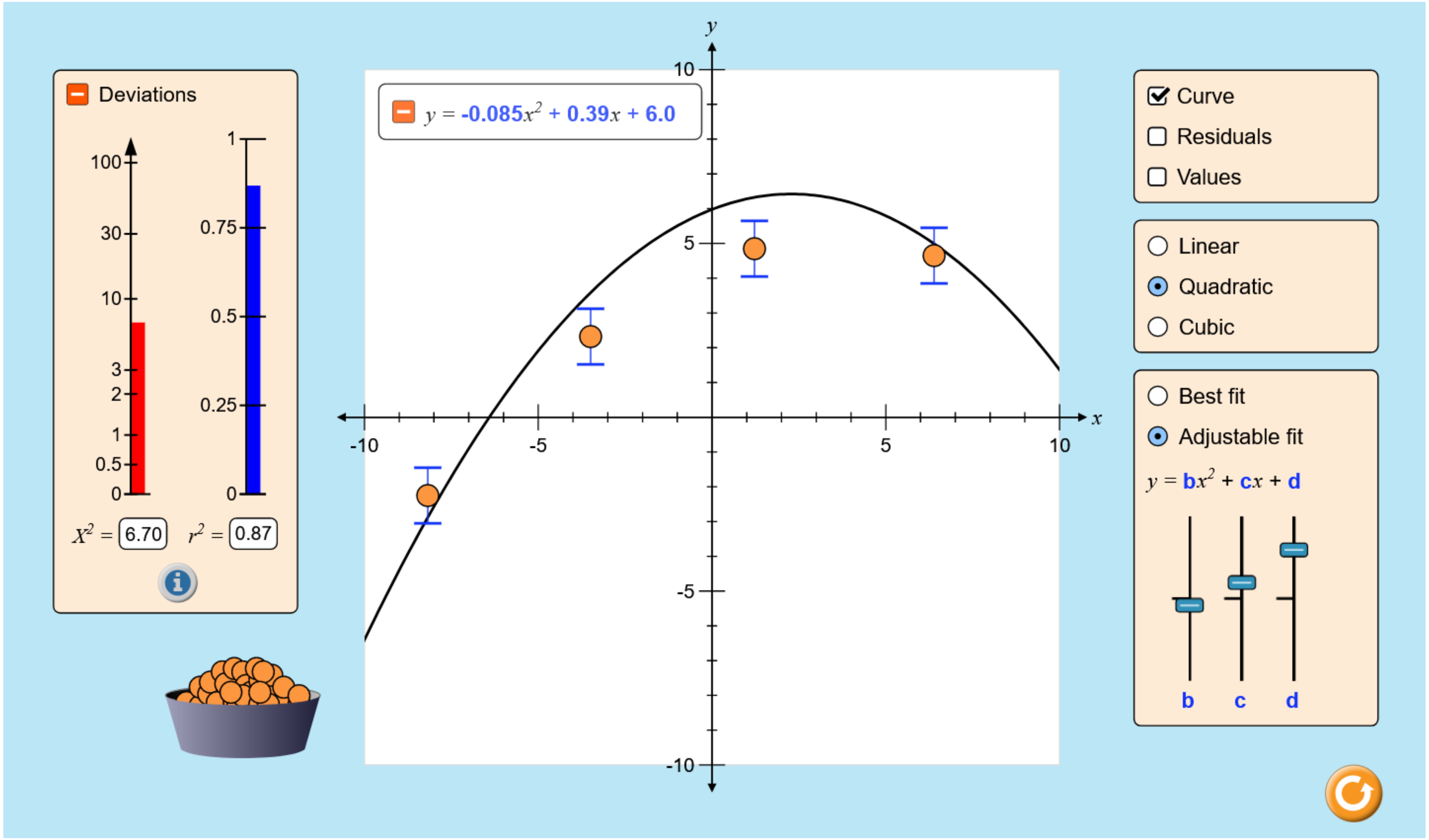
ACU ABILENE CHRISTIAN UNIVERSITY

# Neural Networks

**Key Concepts**

- Perceptron: linear model $\hat{y} = \mathrm{X} \cdot w + b$
- Gradient Descent: search to find $w$’s and $b$
- Combine multiple Perceptrons in non-linear way

https://phet.colorado.edu/sims/html/curve-fitting/latest/curve-fitting_en.html



$X^2 = 6.70$ $r^2 = 0.87$

$y = -0.085x^2 + 0.39x + 6.0$

☑ Curve
☐ Residuals
☐ Values

Linear [✓ Quadratic] Cubic

Best fit [✓ Adjustable fit]

$y = bx^2 + cx + d$

## Perceptron

## 1.5.8. Mathematical formulation

We describe here the mathematical details of the SGD procedure. A good overview with convergence rates can be found in [12].

Given a set of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^m$ and $y_i \in \mathcal{R}$ ($y_i \in -1, 1$ for classification), our goal is to learn a linear scoring function $f(x) = w^T x + b$ with model parameters $w \in \mathbf{R}^m$ and intercept $b \in \mathbf{R}$. In order to make predictions for binary classification, we simply look at the sign of $f(x)$. To find the model parameters, we minimize the regularized training error given by

$$E(w, b) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \alpha R(w)$$

where $L$ is a loss function that measures model (mis)fit and $R$ is a regularization term (aka penalty) that penalizes model complexity; $\alpha > 0$ is a non-negative hyperparameter that controls the regularization strength.

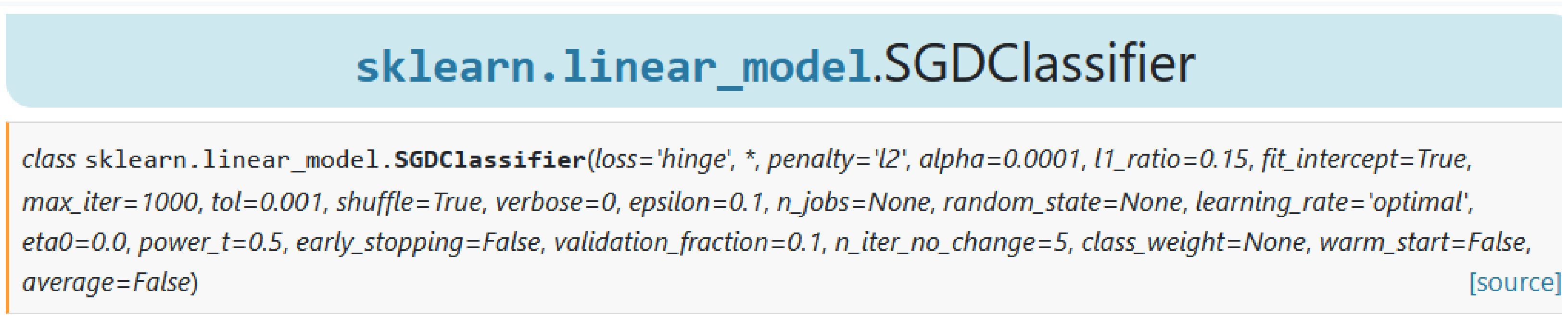
Different choices for $L$ entail different classifiers or regressors:

- Hinge (soft-margin): equivalent to Support Vector Classification. $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))$.
- Perceptron: $L(y_i, f(x_i)) = \max(0, -y_i f(x_i))$.
- Modified Huber: $L(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i))^2$ if $y_i f(x_i) > 1$, and $L(y_i, f(x_i)) = -4 y_i f(x_i)$ otherwise.
- Log Loss: equivalent to Logistic Regression. $L(y_i, f(x_i)) = \log(1 + \exp(-y_i f(x_i)))$.
- Squared Error: Linear regression (Ridge or Lasso depending on $R$). $L(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2$.
- Huber: less sensitive to outliers than least-squares. It is equivalent to least squares when $|y_i - f(x_i)| \leq \varepsilon$, and $L(y_i, f(x_i)) = \varepsilon |y_i - f(x_i)| - \frac{1}{2}\varepsilon^2$ otherwise.
- Epsilon-Insensitive: (soft-margin) equivalent to Support Vector Regression. $L(y_i, f(x_i)) = \max(0, |y_i - f(x_i)| - \varepsilon)$.

Popular choices for the regularization term $R$ (the `penalty` parameter) include:

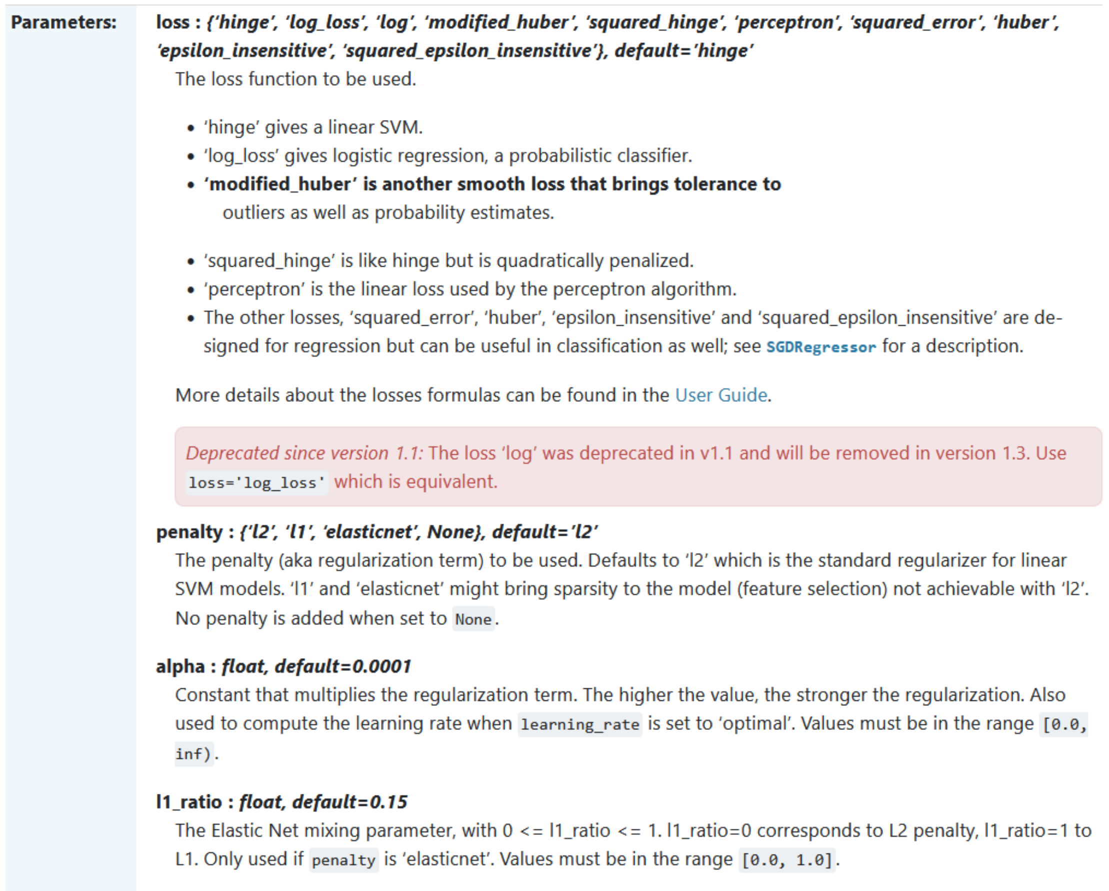
- L2 norm: $R(w) := \frac{1}{2} \sum_{j=1}^{m} w_j^2 = ||w||_2^2$,
- L1 norm: $R(w) := \sum_{j=1}^{m} |w_j|$, which leads to sparse solutions.
- Elastic Net: $R(w) := \frac{\rho}{2} \sum_{j=1}^{n} w_j^2 + (1 - \rho) \sum_{j=1}^{m} |w_j|$, a convex combination of L2 and L1, where $\rho$ is given by `1 - l1_ratio`.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier



## sklearn.linear_model.SGDClassifier

*class* `sklearn.linear_model.`**`SGDClassifier`**(*loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False*) [source]



**Parameters:**

**loss : *{'hinge', 'log_loss', 'log', 'modified_huber', 'squared_hinge', 'perceptron', 'squared_error', 'huber', 'epsilon_insensitive', 'squared_epsilon_insensitive'}, default='hinge'***

The loss function to be used.

- 'hinge' gives a linear SVM.
- 'log_loss' gives logistic regression, a probabilistic classifier.
- **'modified_huber' is another smooth loss that brings tolerance to**

  outliers as well as probability estimates.

- 'squared_hinge' is like hinge but is quadratically penalized.
- 'perceptron' is the linear loss used by the perceptron algorithm.
- The other losses, 'squared_error', 'huber', 'epsilon_insensitive' and 'squared_epsilon_insensitive' are designed for regression but can be useful in classification as well; see `SGDRegressor` for a description.

More details about the losses formulas can be found in the User Guide.

*Deprecated since version 1.1:* The loss 'log' was deprecated in v1.1 and will be removed in version 1.3. Use `loss='log_loss'` which is equivalent.

**penalty : *{'l2', 'l1', 'elasticnet', None}, default='l2'***

The penalty (aka regularization term) to be used. Defaults to 'l2' which is the standard regularizer for linear SVM models. 'l1' and 'elasticnet' might bring sparsity to the model (feature selection) not achievable with 'l2'. No penalty is added when set to `None`.

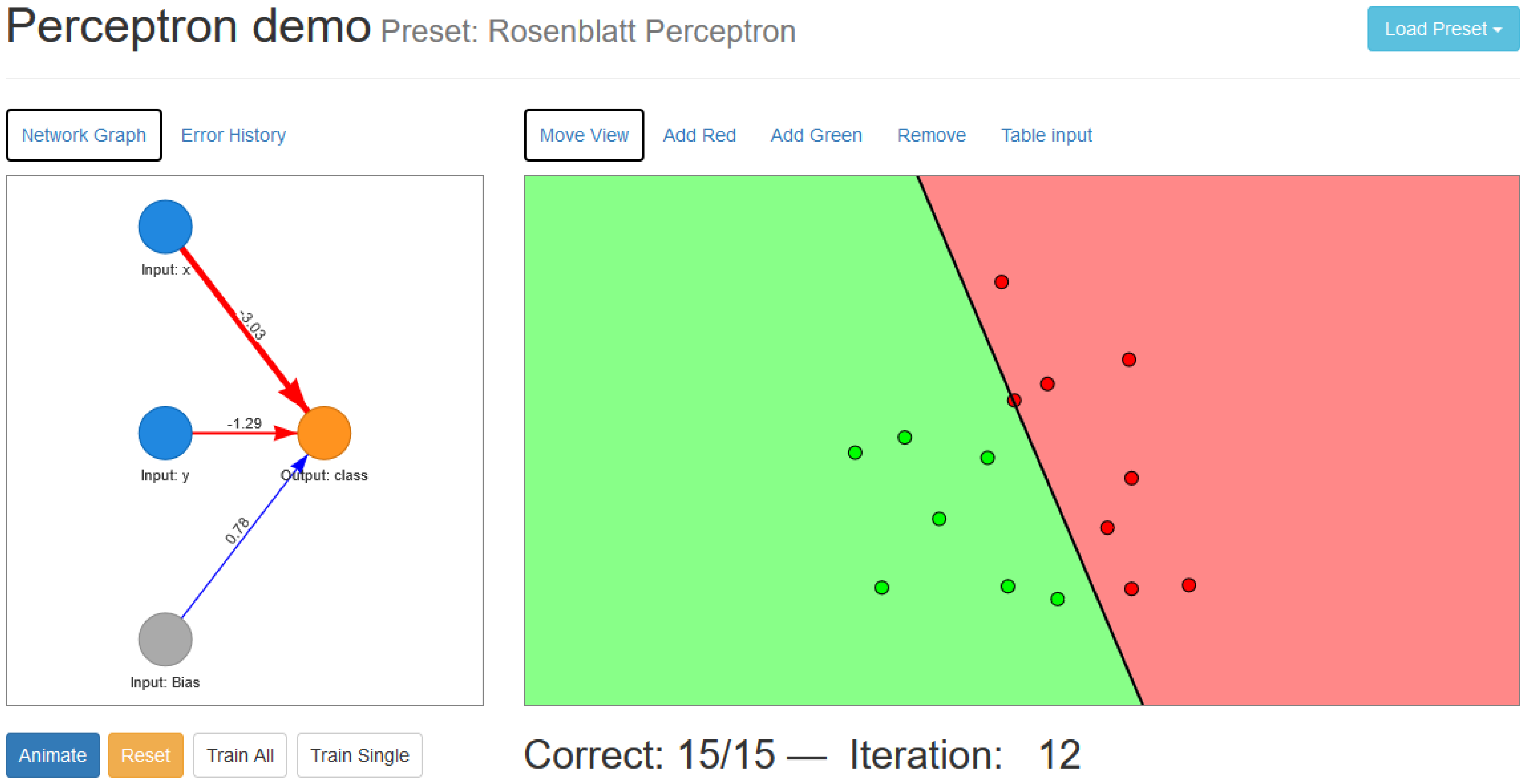
**alpha : *float, default=0.0001***

Constant that multiplies the regularization term. The higher the value, the stronger the regularization. Also used to compute the learning rate when `learning_rate` is set to 'optimal'. Values must be in the range `[0.0, inf)`.

**l1_ratio : *float, default=0.15***

The Elastic Net mixing parameter, with 0 <= l1_ratio <= 1. l1_ratio=0 corresponds to L2 penalty, l1_ratio=1 to L1. Only used if `penalty` is 'elasticnet'. Values must be in the range `[0.0, 1.0]`.

https://phiresky.github.io/kogsys-demos/neural-network-demo/?preset=Rosenblatt%20Perceptron


Perceptron demo Preset: Rosenblatt Perceptron

Correct: 15/15 — Iteration: 12

## Theory

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

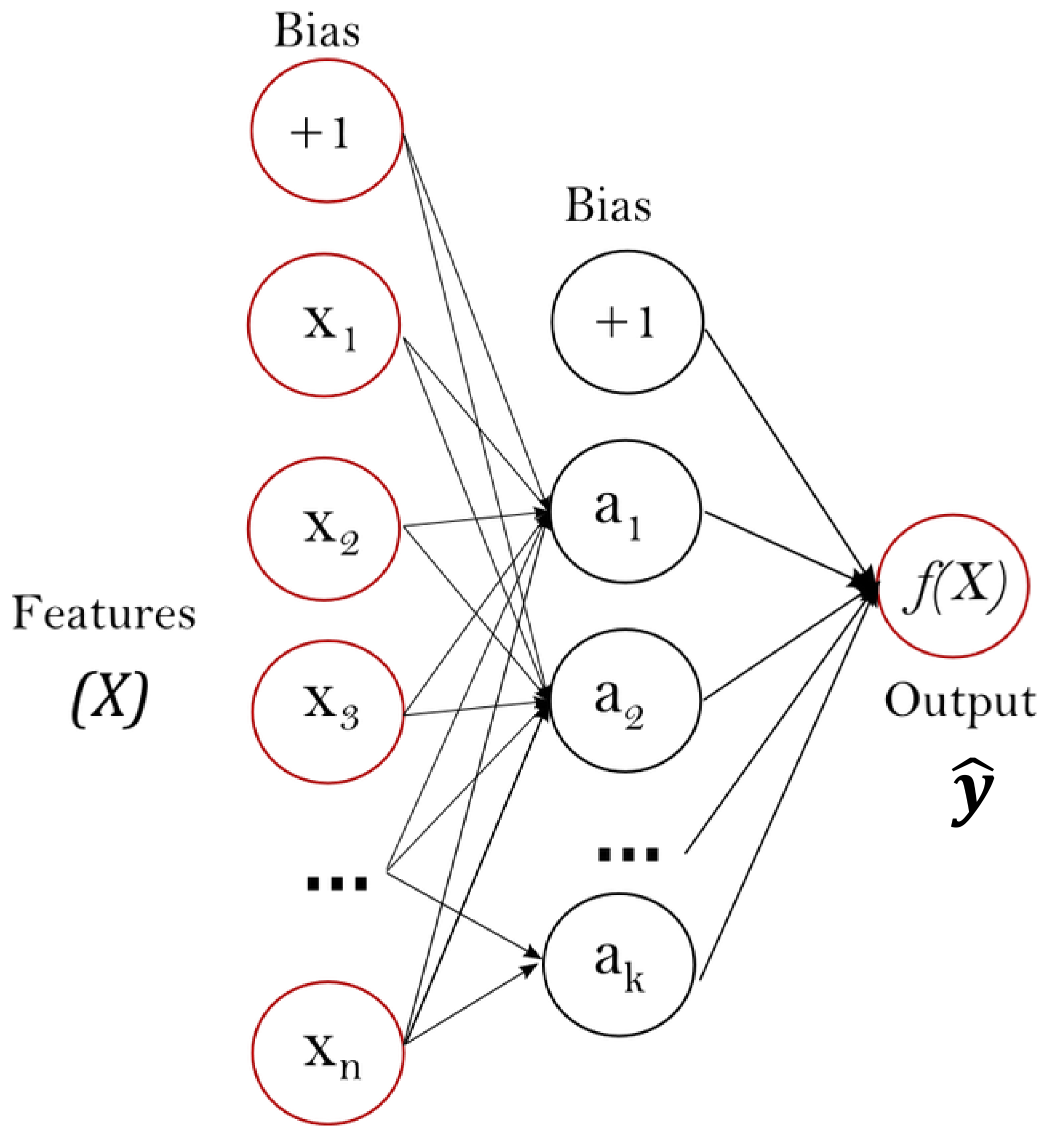
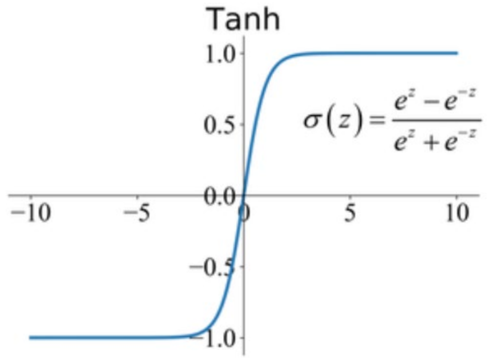


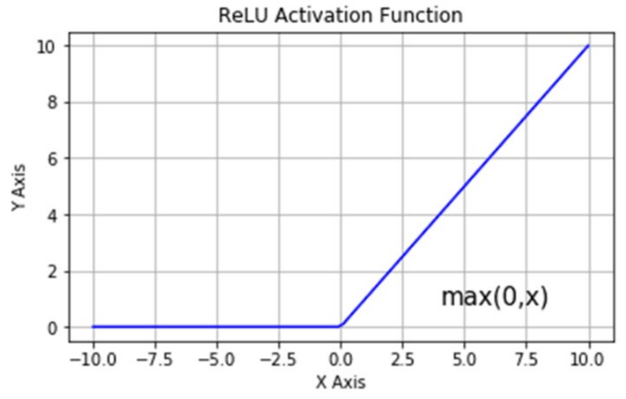
Figure 1 : One hidden layer MLP.

Hidden: $h = g(w \cdot x + b)$

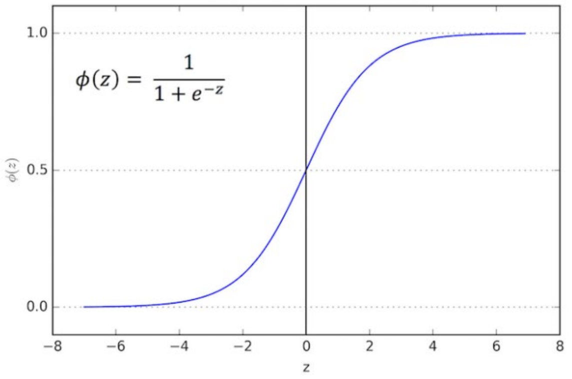
Output: $\hat{y} = f\big(g(w \cdot h + b)\big)$

$g(\quad)$ is activation function

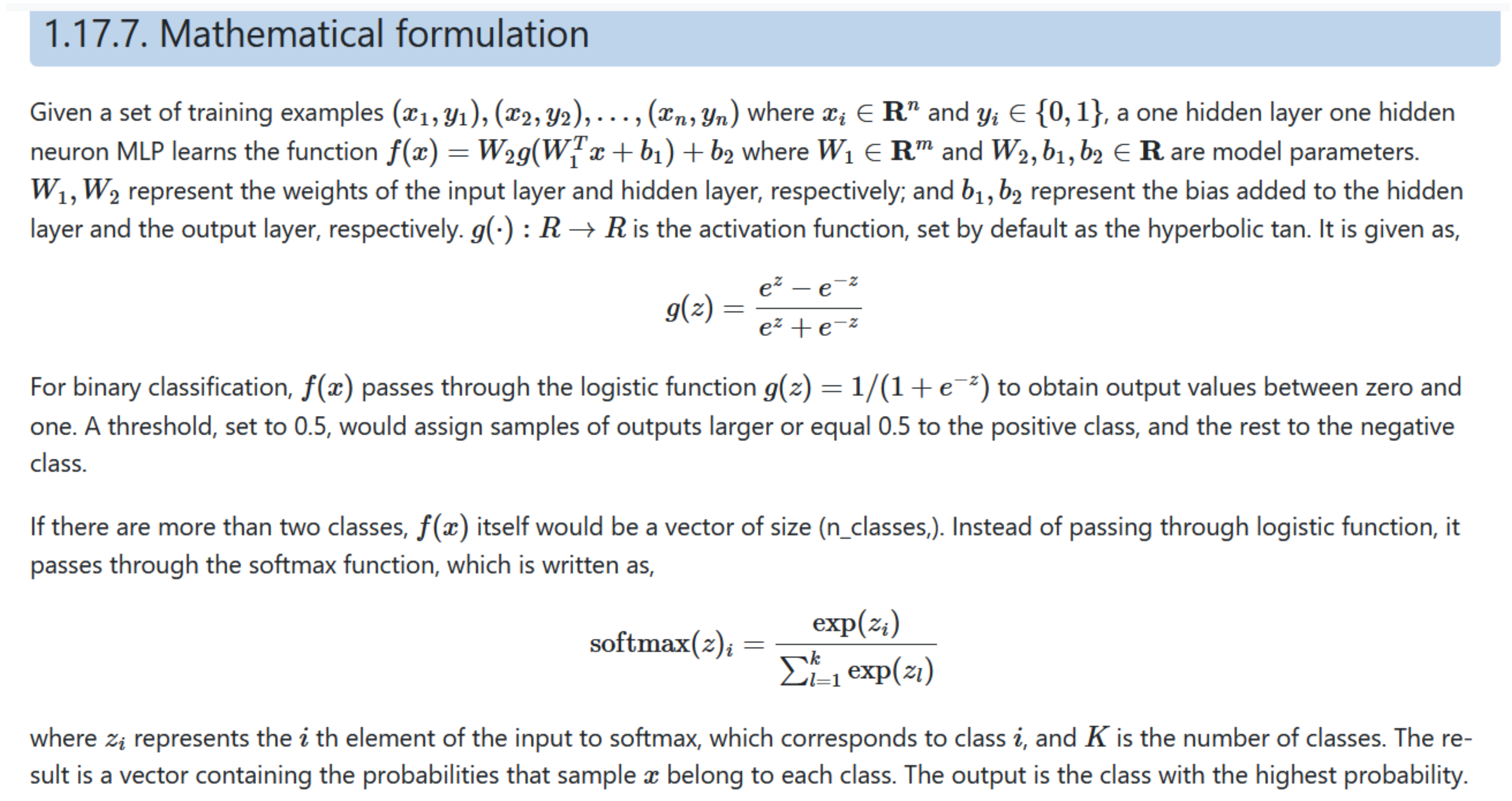




$f(\quad)$ is logistic function



https://scikit-learn.org/stable/modules/neural_networks_supervised.html



## 1.17.7. Mathematical formulation

Given a set of training examples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ where $x_i \in \mathbf{R}^n$ and $y_i \in \{0, 1\}$, a one hidden layer one hidden neuron MLP learns the function $f(x) = W_2 g(W_1^T x + b_1) + b_2$ where $W_1 \in \mathbf{R}^m$ and $W_2, b_1, b_2 \in \mathbf{R}$ are model parameters. $W_1, W_2$ represent the weights of the input layer and hidden layer, respectively; and $b_1, b_2$ represent the bias added to the hidden layer and the output layer, respectively. $g(\cdot) : R \rightarrow R$ is the activation function, set by default as the hyperbolic tan. It is given as,

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

For binary classification, $f(x)$ passes through the logistic function $g(z) = 1/(1 + e^{-z})$ to obtain output values between zero and one. A threshold, set to 0.5, would assign samples of outputs larger or equal 0.5 to the positive class, and the rest to the negative class.

If there are more than two classes, $f(x)$ itself would be a vector of size (n_classes,). Instead of passing through logistic function, it passes through the softmax function, which is written as,

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{l=1}^{k} \exp(z_l)}$$

where $z_i$ represents the $i$ th element of the input to softmax, which corresponds to class $i$, and $K$ is the number of classes. The result is a vector containing the probabilities that sample $x$ belong to each class. The output is the class with the highest probability.

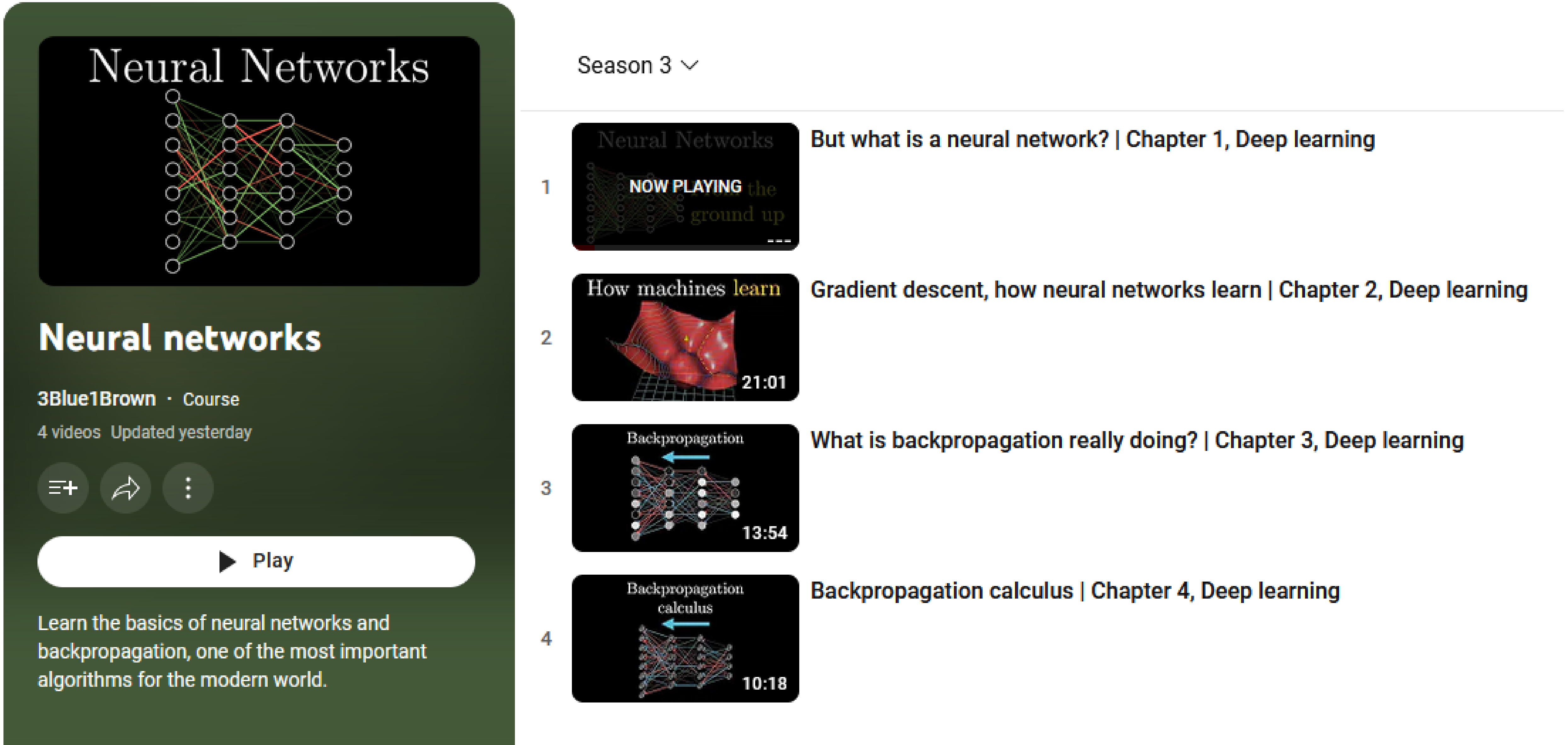
## Cross-Entropy Loss Function

$$Loss(\hat{y}, y, W) = -\frac{1}{n} \sum_{i=0}^{n} (y_i \ln \hat{y}_i + (1 - y_i) \ln (1 - \hat{y}_i)) + \frac{\alpha}{2n} ||W||_2^2$$

$y$ is true from training data

$\hat{y}$ is prediction from model

https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi



Neural networks

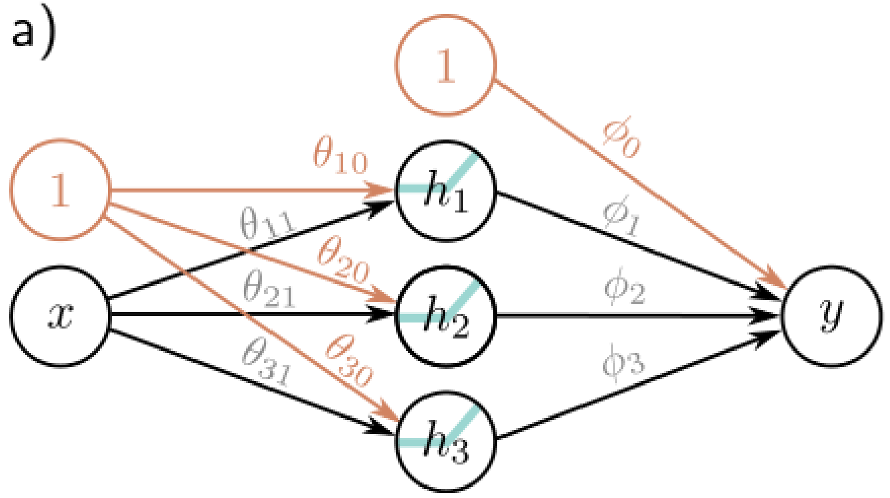
3Blue1Brown · Course

4 videos Updated yesterday

Learn the basics of neural networks and backpropagation, one of the most important algorithms for the modern world.

1. But what is a neural network? | Chapter 1, Deep learning
2. Gradient descent, how neural networks learn | Chapter 2, Deep learning (21:01)
3. What is backpropagation really doing? | Chapter 3, Deep learning (13:54)
4. Backpropagation calculus | Chapter 4, Deep learning (10:18)

$$\begin{aligned} y &= \mathrm{f}[x, \boldsymbol{\phi}] \\ &= \phi_0 + \phi_1 \mathrm{a}[\theta_{10} + \theta_{11} x] + \phi_2 \mathrm{a}[\theta_{20} + \theta_{21} x] + \phi_3 \mathrm{a}[\theta_{30} + \theta_{31} x]. \end{aligned}$$



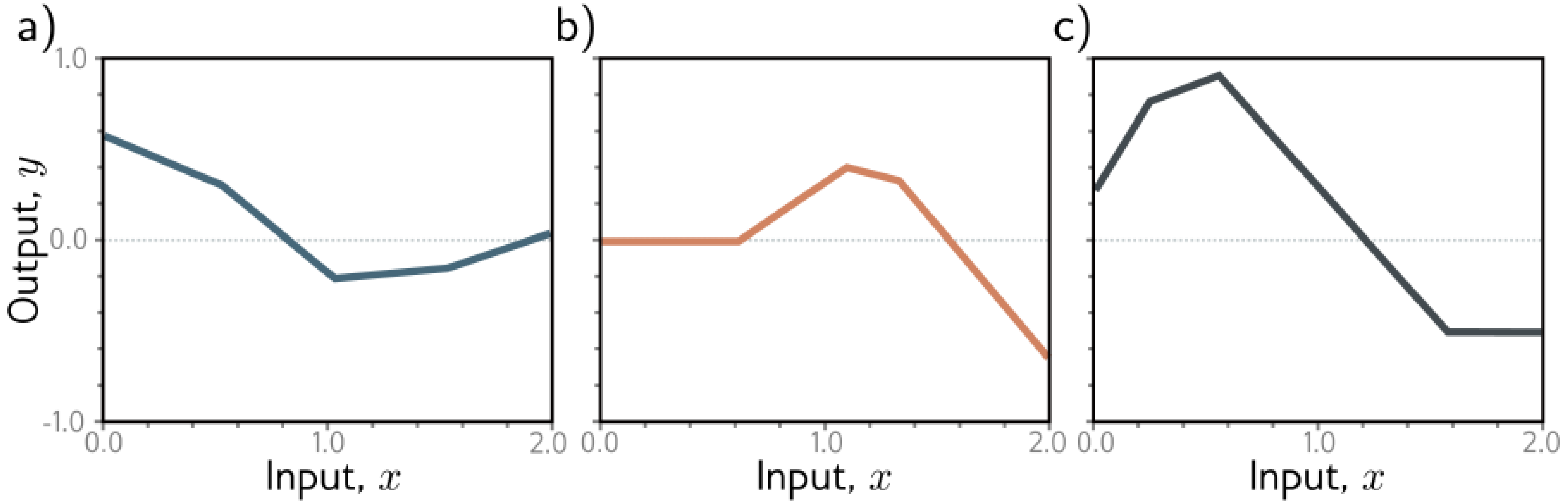
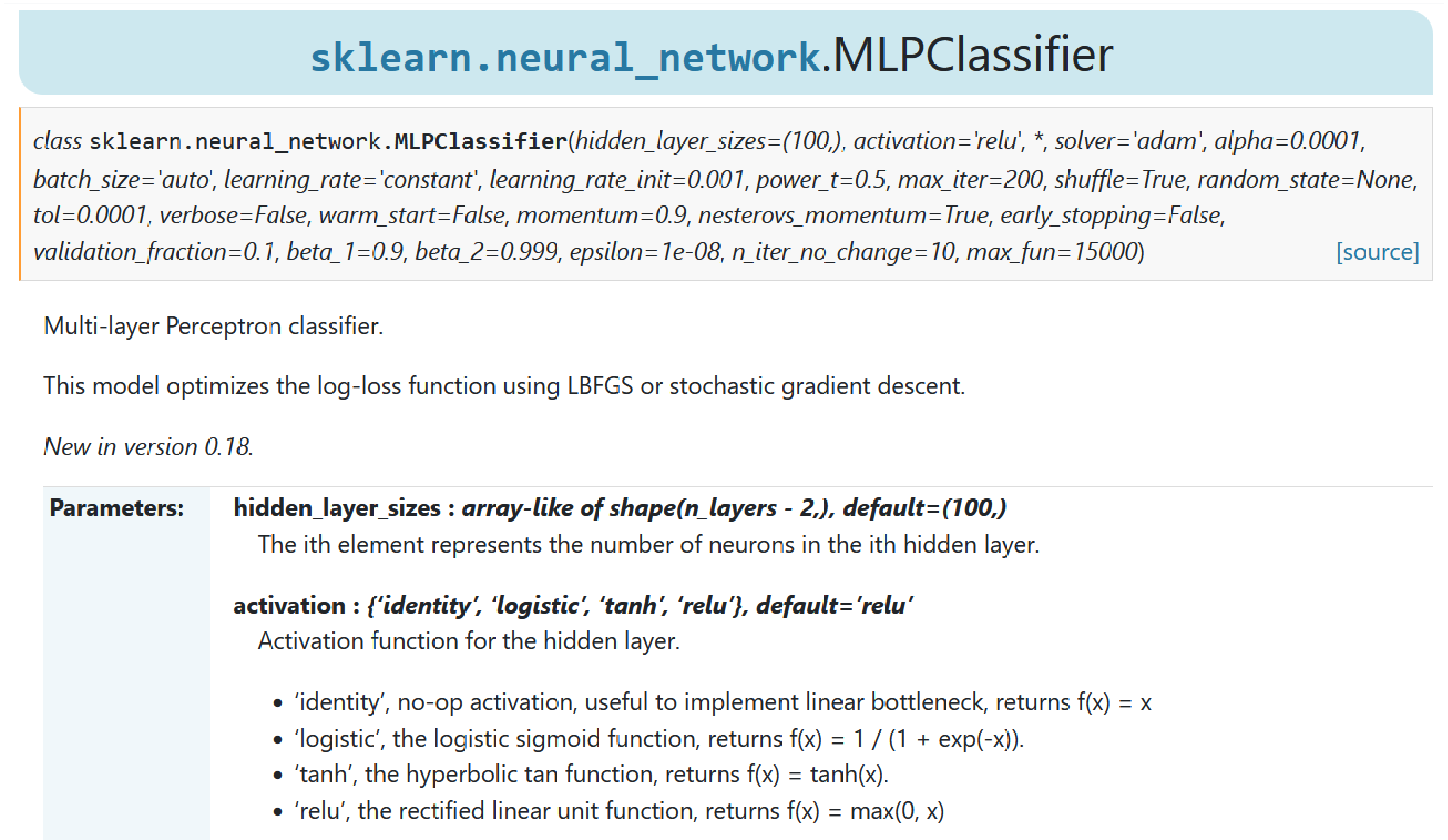


**Figure 3.2** Family of functions defined by equation 3.1. a–c) Functions for three different choices of the ten parameters $\boldsymbol{\phi}$. In each case, the input/output relation is piecewise linear. However, the positions of the joints, the slopes of the linear regions between them, and the overall height vary.

## Practice

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html



## 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 : *array-like of shape(n_layers - 2,), default=(100,)***

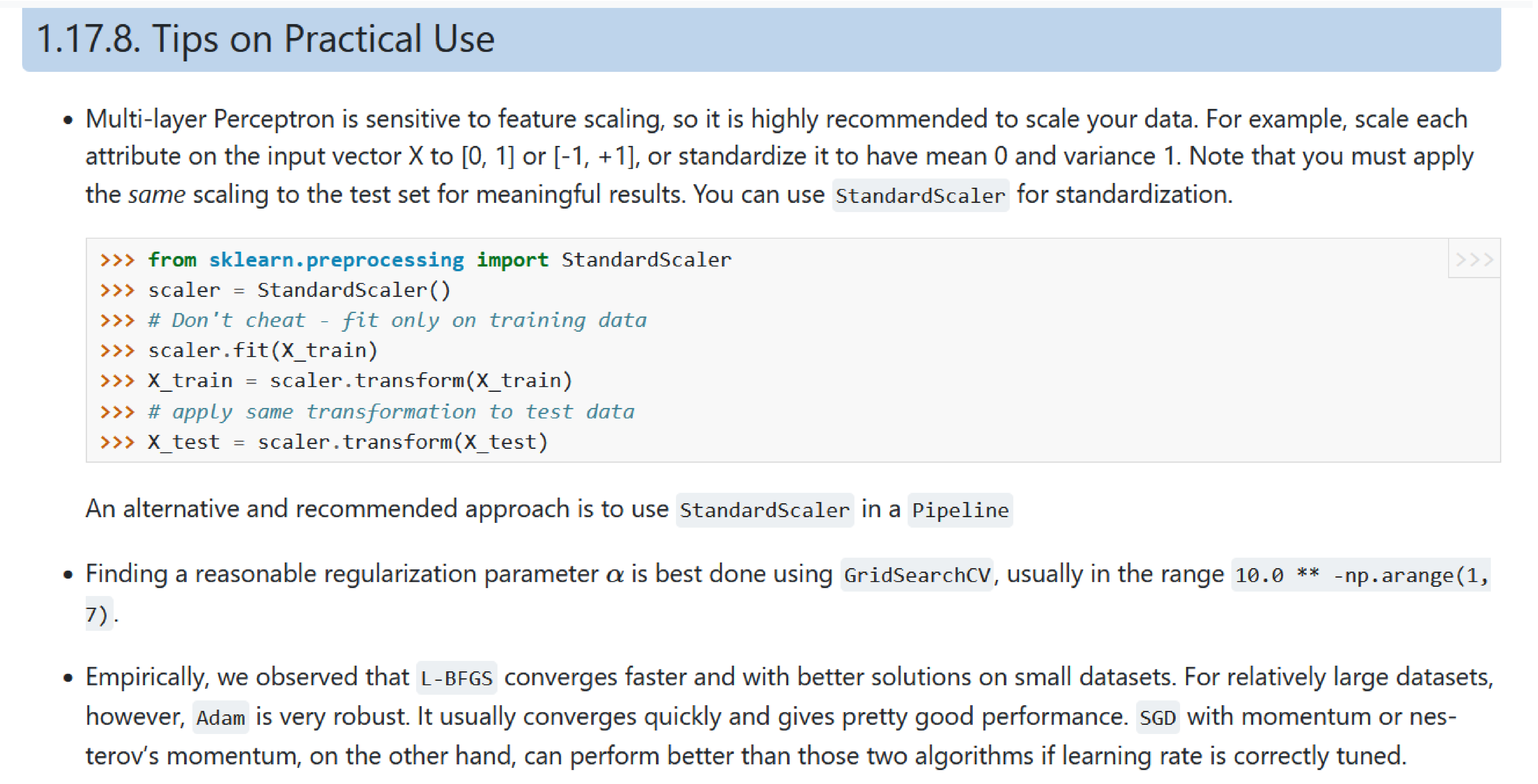
The ith element represents the number of neurons in the ith hidden layer.

**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)

https://scikit-learn.org/stable/modules/neural_networks_supervised.html



## 1.17.8. Tips on Practical Use

- Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data. For example, scale each attribute on the input vector X to [0, 1] or [-1, +1], or standardize it to have mean 0 and variance 1. Note that you must apply the *same* scaling to the test set for meaningful results. You can use `StandardScaler` for standardization.

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> # Don't cheat - fit only on training data
>>> scaler.fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> # apply same transformation to test data
>>> X_test = scaler.transform(X_test)
```

  An alternative and recommended approach is to use `StandardScaler` in a `Pipeline`

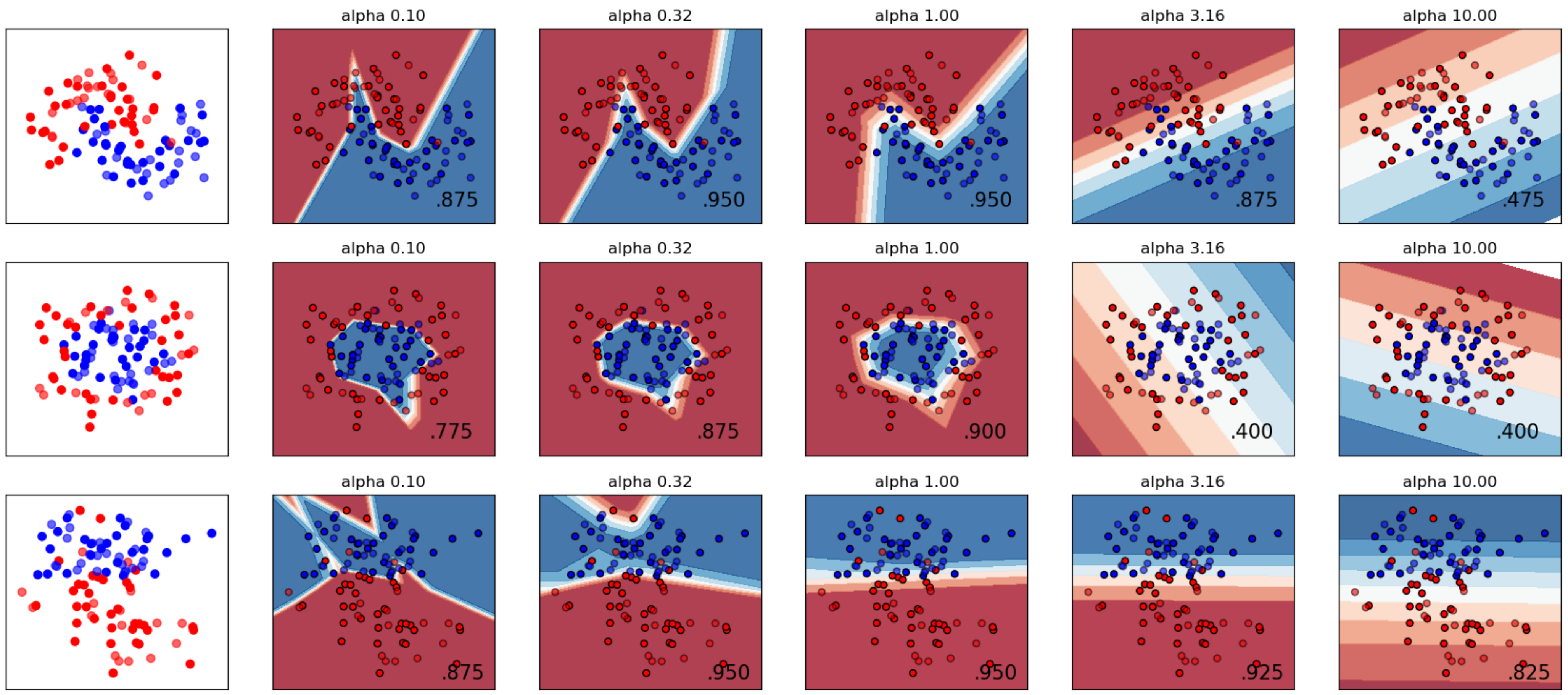
- Finding a reasonable regularization parameter $\alpha$ is best done using `GridSearchCV`, usually in the range `10.0 ** -np.arange(1, 7)`.

- Empirically, we observed that `L-BFGS` converges faster and with better solutions on small datasets. For relatively large datasets, however, `Adam` is very robust. It usually converges quickly and gives pretty good performance. `SGD` with momentum or nesterov's momentum, on the other hand, can perform better than those two algorithms if learning rate is correctly tuned.

# Varying regularization in Multi-layer Perceptron

A comparison of different values for regularization parameter 'alpha' on synthetic datasets. The plot shows that different alphas yield different decision functions.

Alpha is a parameter for regularization term, aka penalty term, that combats overfitting by constraining the size of the weights. Increasing alpha may fix high variance (a sign of overfitting) by encouraging smaller weights, resulting in a decision boundary plot that appears with lesser curvatures. Similarly, decreasing alpha may fix high bias (a sign of underfitting) by encouraging larger weights, potentially resulting in a more complicated decision boundary.

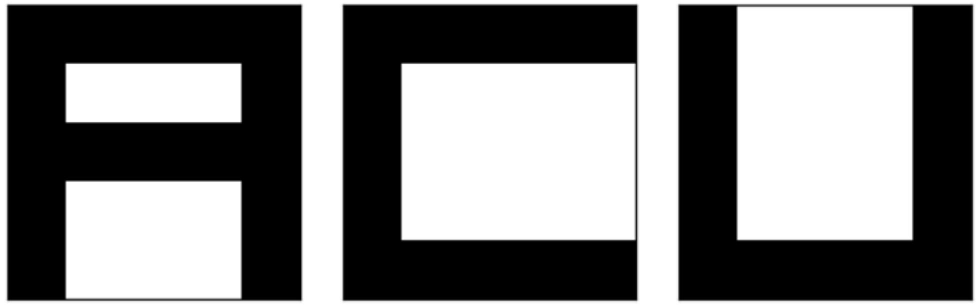


https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html#sphx-glr-auto-examples-neural-networks-plot-mlp-alpha-py

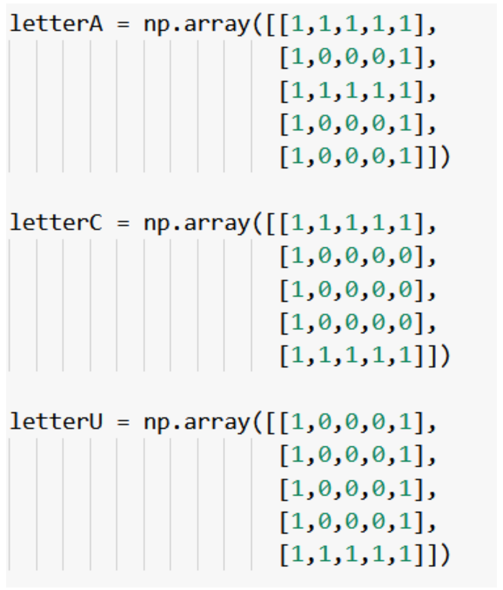
## Examples

# The ACU Classifier

Usually we don't know (and don't care) how the neural network works. The weights typically have no interpretation. However, you can make simple examples where they do have meaning.

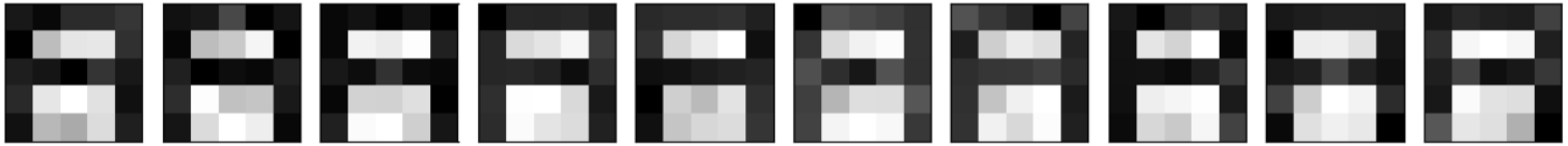


Our 3 "digits"



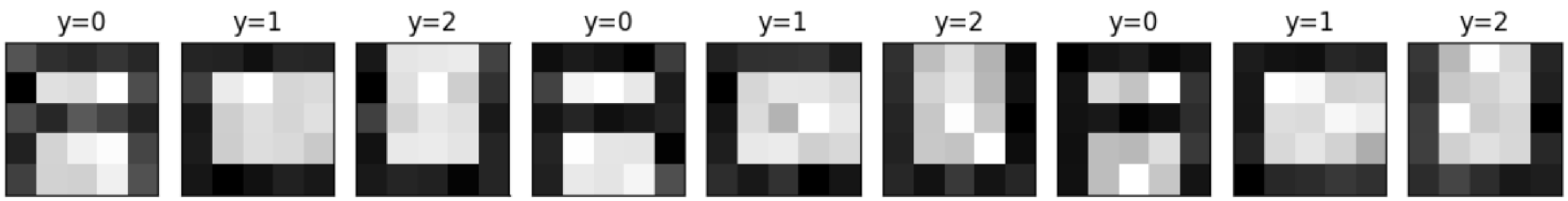
```
letterA = np.array([[1,1,1,1,1],
                    [1,0,0,0,1],
                    [1,1,1,1,1],
                    [1,0,0,0,1],
                    [1,0,0,0,1]])

letterC = np.array([[1,1,1,1,1],
                    [1,0,0,0,0],
                    [1,0,0,0,0],
                    [1,0,0,0,0],
                    [1,1,1,1,1]])

letterU = np.array([[1,0,0,0,1],
                    [1,0,0,0,1],
                    [1,0,0,0,1],
                    [1,0,0,0,1],
                    [1,1,1,1,1]])
```

To make training data, I take each letter and add some random gaussian noise:



# The ACU Classifier

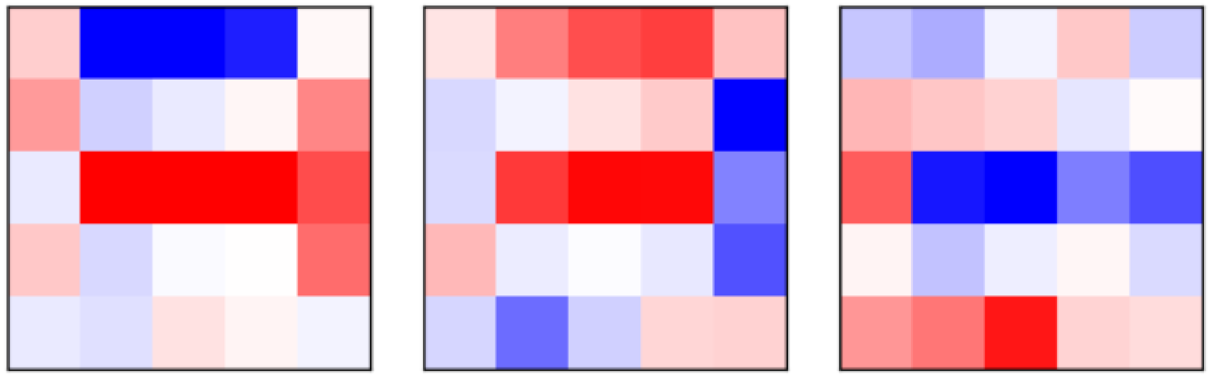
**Training data**



Train a neural network with 3 hidden nodes:

```
NUMH=3
mlp = MLPClassifier(hidden_layer_sizes=(NUMH), activation='tanh',max_iter=1000, random_state=1)
mlp.fit(X, y)
print('TRAIN', mlp.score(X, y))
```

Draw the input-to-hidden weights:



Results are different every time, but sometimes it “learns” to look for the cross pieces.

Also see: https://scikit-learn.org/stable/auto_examples/neural_networks/plot_mnist_filters.html#sphx-glr-auto-examples-neural-networks-plot-mnist-filters-py