```
logLik.bamlss <- function(object, ..., optimizer = FALSE, samples = FALSE)
{
  Call <- match.call()
  Call <- Call[!(names(Call) %in% c("optimizer", "samples"))]
  mn <- as.character(Call)[-1L]
  object <- list(object, ...)
  mstop <- object$mstop
  if(any(names(object) != "")) {
    i <- names(object) == ""
    object <- object[i]
    mn <- mn[i]
  }
  object <- object[mn != "mstop"]
```

Advanced Bayesian Methods: Theory and Applications in R

Machine Learning

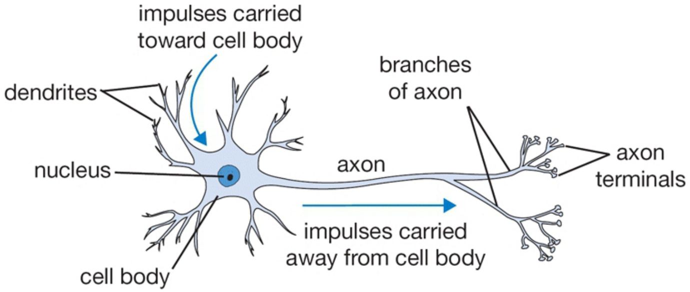
Nikolaus Umlauf
https://nikum.org/abm.html

Neural Networks

- Neural networks (NN) are models that mimic the interconnected structure of neurons in the human brain.
- They are flexible nonlinear regression models capable of capturing intricate dependencies in data.
- NNs excel in handling high-dimensional data and can learn and represent features from raw data during the training process.
- They are suitable for a variety of tasks including classification, regression, and clustering.
- NNs are universal function approximators (Hornik 1991).
- In statistical modeling, NNs are often used alongside traditional techniques for improved predictive performance.

Inspiration: The Brain

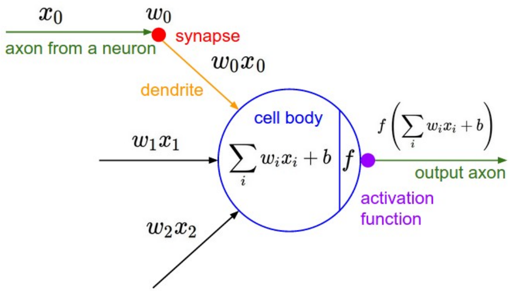
- Many machine learning methods are inspired by biology, including the human brain.
- The human brain has around 86 billion neurons on average, with each communicating with about 10,000 other neurons.


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

Source (2024-09-17): `http://cs231n.github.io/neural-networks-1/`

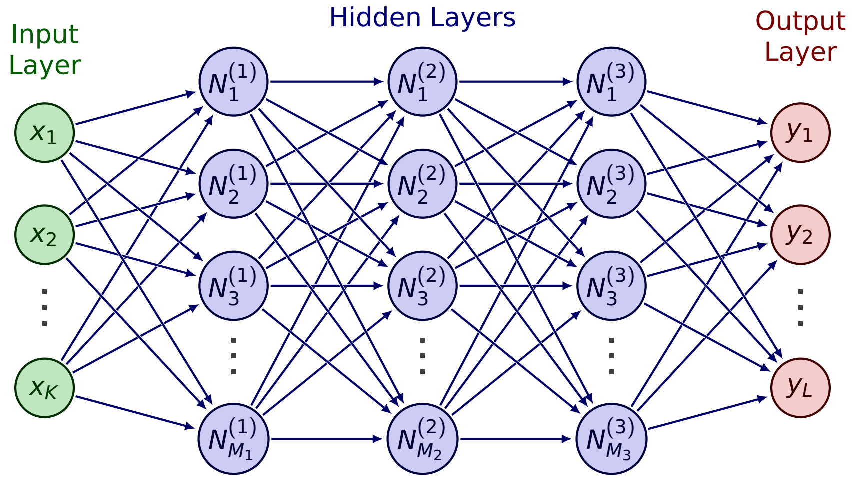
Mathematical Model

- NNs define functions of the inputs, which represent hidden features and are computed by neurons.
- The individual components of neural networks, also known as artificial neurons, are called units.

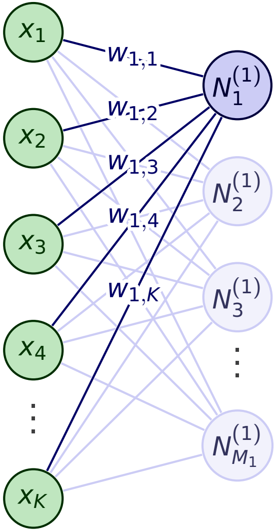


Source (2024-09-17): `http://cs231n.github.io/neural-networks-1/`

General Architecture



General Architecture



$$= \sigma\left(w_{1,0} + w_{1,1}x_1 + w_{1,2}x_2 + \ldots + w_{1,K}x_K\right)$$

$$= \sigma\left(w_{1,0} + \sum_{k=1}^{K} w_{1,k}x_k\right)$$

$$\begin{pmatrix} N_1^{(1)} \\ N_2^{(1)} \\ \vdots \\ N_{M_1}^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,0} \\ w_{2,0} \\ \vdots \\ w_{M_1,0} \end{pmatrix} + \begin{pmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,K} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M_1,1} & w_{M_1,2} & \ldots & w_{M_1,K} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix}\right]$$

$$\mathbf{N}^{(1)} = \sigma\left(\mathbf{w}_0^{(1)} + \mathbf{W}^{(1)}\mathbf{X}\right)$$

Single Hidden Layer NN

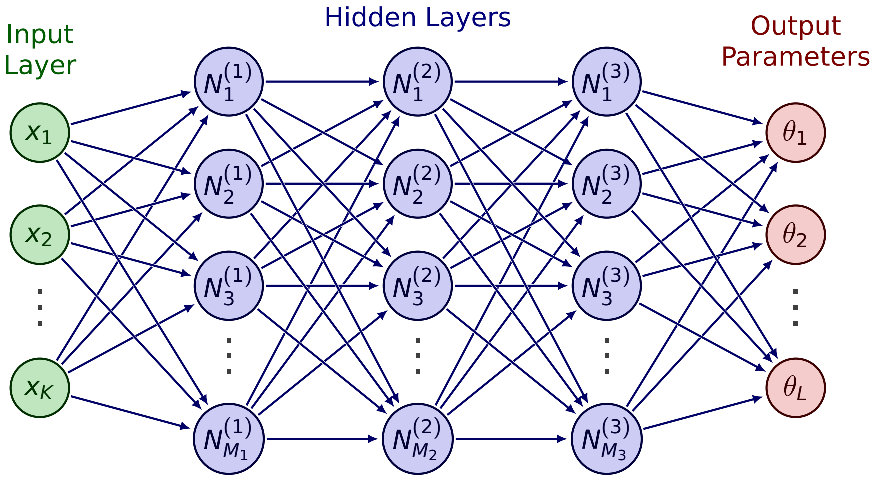
Given response vector $\mathbf{y} = (y_1, \ldots, y_n)^\top$ and input feature matrix $\mathbf{X}$, a single hidden layer NN can be expressed as

$$\underset{n\times 1}{\mathbf{y}} = \left(\underset{1\times M_1}{\mathbf{W}}^{(2)} \sigma \left[\underset{M_1\times 1}{\mathbf{w}_0}^{(1)} + \underset{M_1\times K}{\mathbf{W}}^{(1)} \underset{K\times n}{\left(\mathbf{X}^\top\right)} \right] \right)^\top = \left(\underset{1\times M_1}{\mathbf{W}}^{(2)} \underset{M_1\times n}{\mathbf{N}}^{(1)} \right)^\top = \underset{n\times M_1}{\left(\mathbf{N}^{(1)\top}\right)} \underset{M_1\times 1}{\left(\mathbf{W}^{(2)\top}\right)},$$

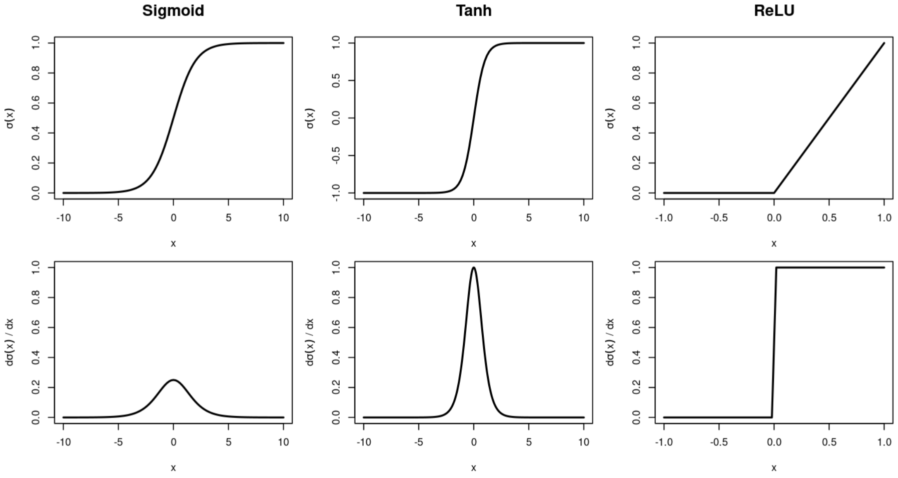
where

- $\mathbf{X}$ is the input data matrix with dimensions $n \times K$,
- $\mathbf{y}$ is the output vector with dimensions $n \times 1$,
- $\sigma(\cdot)$ is the activation function applied to the output of the hidden layer,
- $\mathbf{W}^{(1)}$ is the weight matrix for the hidden layer with dimensions $M_1 \times K$,
- $\mathbf{w}_0^{(1)}$ is the bias vector for the hidden layer with dimensions $M_1 \times 1$,
- $\mathbf{W}^{(2)}$ is the weight matrix for the output layer with dimensions $1 \times M_1$.

Deep Distributional NN



Activation Functions $\sigma(x)$



Estimation in Neural Networks

Estimating the parameters of a neural network involves optimizing the model to minimize the difference between predicted and actual values.

- **Objective Function**: Define a loss function that quantifies the difference between predicted and actual values. Common choices include mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks.
- **Optimization Algorithm**: Use an optimization algorithm, such as stochastic gradient descent (SGD) or Adam, to minimize the loss function by adjusting the weights and biases of the network iteratively.
- **Backpropagation**: Compute the gradients of the loss function with respect to the parameters of the network using backpropagation. This allows for efficient computation of parameter updates that minimize the loss.

Estimation in Neural Networks

- **Estimation with *nnet* Package**: The *nnet* package provides functionality for fitting neural network models using the method of maximum likelihood estimation. The model parameters are optimized using a form of gradient descent called the BFGS algorithm, which iteratively updates the weights and biases to minimize the negative log-likelihood of the observed data.

The training process involves repeatedly adjusting the parameters of the network until convergence, where the model performs well on a validation dataset or a predefined stopping criterion is met.

NN for GAMLSS

Example:

```
R> library("gamlss2")
R> data("rent", package = "gamlss.data")
```

Model formula including NN (see `?special_terms`).

```
R> f <- R ~  H + loc + n(~Fl+A,size=30,decay=0.8) |
+      H + loc + n(~Fl+A,size=30,decay=0.8)
```

Estimate model.

```
R> set.seed(123)
R> b <- gamlss2(f, data = rent, family = GA)
```

Plot estimated effects.

```
R> plot(b)
```

Residual diagnostics.

```
R> plot(b, which = "resid")
```

DDNN

Example:

```
R> library("bamlss")
R> library("keras")
R> data("rent", package = "gamlss.data")
```

Data preparation.

```
R> rent$R <- sqrt(rent$R)
R> rent$Fl <- scale(rent$Fl)
R> rent$A <- scale(rent$A)
```

Model formula.

```
R> f <- R ~  H + loc + Fl + A
```

Set python environment.

```
R> Sys.setenv(RETICULATE_PYTHON = "/usr/bin/python3")
```

DDNN

Estimate model.

```
R> set.seed(123)
R> b <- ddnn(f, data = rent, family = gaussian,
+    nlayers = 2, units = 100, activation = "relu",
+    learning_rate = 0.01, epochs = 1000,
+    validation_split = 0.2, early_stopping = TRUE,
+    patience = 50)
```

Plot history.

```
R> plot(b$history)
```

Residual diagnostics.

```
R> e <- residuals(b)
R> plot(e)
```

DDNN

Gaussian model from scratch:

Packages.

```
R> library("keras")
R> library("tensorflow")
R> Sys.setenv(RETICULATE_PYTHON = "/usr/bin/python3")
```

Response and covariate data.

```
R> data("rent", package = "gamlss.data")
R> Y <- cbind(scale(sqrt(rent$R)), 1)
R> X <- cbind(1, scale(rent$Fl))
```

DDNN

Define loss function.

```
R> gaussian_loss <- function(y_true, y_pred) {
+     K = keras::backend()
+     mu = y_pred[, 1]
+     sigma = K$exp(y_pred[,2])
+     sigma2 = K$pow(sigma, 2)
+     ll = -0.5 * K$log(6.28318530717959 * sigma2) -
+       0.5 * K$pow((y_true[,1] - mu), 2) / sigma2
+     ll = K$sum(ll)
+     return(-1 * ll)
+   }
```

DDNN

Define architecture.

```
R> input_mu <- keras::layer_input(shape = ncol(X))
R> input_sigma <- keras::layer_input(shape = ncol(X))
R> output_mu <- input_mu %>%
+    layer_dense(units = 100, activation = "relu") %>%
+    layer_dense(units = 100, activation = "relu") %>%
+    layer_dense(units = 1)
R> output_sigma <- input_sigma %>%
+    layer_dense(units = 100, activation = "relu") %>%
+    layer_dense(units = 100, activation = "relu") %>%
+    layer_dense(units = 1)
R> inputs <- list(input_mu, input_sigma)
R> outputs <- list(output_mu, output_sigma)
R> final_output <- keras::layer_concatenate(outputs)
R> model <- keras::keras_model(inputs, final_output)
```

DDNN

Compile.

```
R> model <- keras::compile(model,
+     loss = gaussian_loss,
+     optimizer = keras::optimizer_adam(learning_rate = 0.01)
+   )
R> callbacks <- list(
+     keras::callback_early_stopping(patience = 50)
+   )
```

DDNN

Estimate model.

```
R> set.seed(123)
R> history <- keras::fit(model,
+    x = list(X, X),
+    y = Y,
+    epochs = 1000,
+    batch_size = 10,
+    verbose = 1,
+    validation_split = 0.2,
+    callbacks = callbacks
+  )
```

Plot loss.

```
R> plot(history)
```

DDNN

Predict and visualize.

```
R> par <- predict(model, list(X, X))
R> par <- as.data.frame(par)
R> names(par) <- c("mu", "sigma")
R> par$sigma <- exp(par$sigma)
R> i <- order(X[, 2])
R> plot(Y[, 1] ~ X[, 2], pch = 16, col = rgb(0.1, 0.1, 0.1, alpha = 0.3))
R> lines(par$mu[i] ~ X[i, 2], col = 4, lwd = 2)
R> q <- NULL
R> for(p in c(0.01, 0.1, 0.9, 0.99))
+    q <- cbind(q, qnorm(p, mean = par$mu, sd = par$sigma))
R> matplot(X[i, 2], q[i, ], type = "l",
+    lty = c(3, 2, 2, 3), lwd = 2, col = 4,
+    add = TRUE)
```

Two-Stage Approach

- **Stage 1**: Neural Network Prediction.
    - Train a neural network to capture complex, non-linear patterns in the data.
    - Use the trained network to generate predictions for the outcome variable.
- **Stage 2**: Bayesian Modeling with JAGS.
    - Use the predictions from the neural network as inputs in a Bayesian model.
    - Estimate the Bayesian model using JAGS to quantify uncertainty.
- **Benefits**:
    - Combines the flexibility of neural networks with the rigor of Bayesian inference.
    - Provides robust predictions with uncertainty estimates.