Direct Use of Neural Networks for Decision Making

Ernest Aksen, PhD, ScD
Professor at Department of Mathematical Methods in Economics,
Belarus State Economic University (Minsk, Belarus)
eaksen@hotmail.com

In most cases the output of a neural network produces a predicted value for the relevant output factor or probabilities for different values for the relevant output factor, and a decision is made based on these values (Bishop, 2006; Goodfellow, 2016; Haykin, 2008). (Examples: a neural network can predict the future demand for a commodity, and based on this value a decision is made on the quantity of output; a neural network can predict a future price of a commodity, and based on this value a decision to buy or not to buy it is made; a neural network can estimate probabilities of defaults for a loan, and based on these values the decision to give or reject the loan is made). This paper develops a methodology of using neural networks directly for making decisions without preliminary obtaining the predicted value for the relevant output factor (or probabilities for different values of the relevant output factor). The presented methodology is related to reinforcement learning (Sutton, 2018).

Notation

Let $x=(x_1,\ldots,x_m)$ denote a vector of input factors, y – an observed output factor, z – a decision (which is made after x get known but before y gets known), $u(z,y)$ – utility (benefit) for decision z if the output y occurs.

Let $f(x,\beta)$ denote a neural network where β is an array of estimated parameters. The function $f(x,\beta)$ can be scalar or vector-valued (or array-valued). If the function $f(x,\beta)$ is vector-valued, we denote its components as $f_k(x,\beta)$. Thus $f(x,\beta)=\left[f_1(x,\beta),\ldots,f_s(x,\beta)\right]$.

Parameters β are estimated based on n observations (x_i,y_i), $i\in\{1,2,\ldots,n\}$.

Let X, Y and Z denote the respective series of observations and decisions:

$$X=(x_1,\ldots,x_n),\qquad Y=(y_1,\ldots,y_n),\qquad Z=(z_1,\ldots,z_n). \tag{1}$$

Denote:

$$F(X,\beta)=\left[f(x_1,\beta),\ldots,f(x_n,\beta)\right]. \tag{2}$$

Consider separately two cases: continuous and discrete decision variable z.

Continuous variable z

In this case we interpret the output $f(x,\beta)$ of the relevant network as the value for the decision variable z:

$$z = f(x,\beta). \tag{3}$$

Using formula (3) we can obtain the decision z_i corresponding to the *i*-th observation and to arbitrary values of parameters β:

$$z_i = f(x_i,\beta), \quad i \in \{1,2,\ldots,n\}. \tag{4}$$

Remark 1. The decision (4) is not a real decision that was actually made for the *i*-th observation. It is the decision which would have been made if the neural network (with the given values of parameters β) had been used for making decisions.

In accordance with formulas (1), (2), (4) and (5) we have:

$$Z = F(X,\beta). \tag{5}$$

Let $U(Z,Y)$ denote a total utility (benefit) function (over the given series of observations). For example function $U(Z,Y)$ can be of the form:

$$U(Z,Y) = \sum_{i=1}^{n} u(z_i, y_i), \tag{6}$$

where $u(z_i, y_i)$ denotes the utility (benefit) from decision z_i for the state y_i.

Substituting (5) into the total utility function $U(Z,Y)$ we get:

$$U(Z,Y) = U\left[F(X,\beta),Y\right]. \tag{7}$$

It is quite natural to estimate parameters β by maximizing (7):

$$U\left[F(X,\beta),Y\right] \to \max. \tag{8}$$

The optimal values β^* of parameters β provide the decisions $Z^* = F(X,\beta^*)$ which maximize the total utility for the given observations (x_i, y_i), $i \in \{1,2,\ldots,n\}$.

Example 1.

Suppose a firm produces a commodity. If its output *z* exceeds the demand *y* for the commodity, the remainder is sold at a discount price. The regular and discount prices *p* and *d* and the unit cost *c* are supposed to be constant. The demand *y* for the commodity depends (stochastically) on the factors $x = (x_1,\ldots,x_m)$. When the firm decides how much to produce, it knows the values of factors $x = (x_1,\ldots,x_m)$ but does not know the (future) demand *y*. Observations (x_i, y_i), $i \in \{1,2,\ldots,n\}$ for the input factors $x = (x_1,\ldots,x_m)$ and demand *y* are known (over *n* past periods). Using this data we want to teach a neural network to recommend us the optimal output size based on the current values of input factors $x = (x_1,\ldots,x_m)$.

According to (3) we interpret the output $f(x,\beta)$ of a relevant neural network as the firm's output size.

Within the above setting, the firm's (one-period) profit $u(z, y)$ is calculated as follows:

$$u(z, y) = \min(z, y)\, p + \max(z - y, 0)\, d - cz\,, \tag{9}$$

and therefore the total profit $U(Z, Y)$ over *n* periods equals:

$$U(Z, Y) = \sum_{i=1}^{n} \left\{ \min(z_i, y_i)\, p + \max(z_i - y_i, 0)\, d - cz_i \right\}, \tag{10}$$

where $Y = (y_1, \ldots, y_n)$, $Z = (z_1, \ldots, z_n)$ according to notation (1).

Substituting (4) and (5) into (10) we have:

$$U\left[F(X, \beta), Y\right] = \sum_{i=1}^{n} \left\{ \min\left[f(x_i, \beta), y_i \right] p + \max\left[f(x_i, \beta) - y_i, 0 \right] d - cf(x_i, \beta) \right\}. \tag{11}$$

The optimal values β^* of parameters β provide the outputs $Z^* = F(X, \beta^*)$ which maximize the total profit (11) (over *n* periods) for the given explanatory values x_i and demands y_i, $i \in \{1, 2, \ldots, n\}$.

A relevant implementation in Keras is given below for the simplest case of an one-layer network with the rectified linear activation function:

```
model = Sequential()
model.add(Dense(1, input_dim=m, activation='relu'))
def fun_loss(Y,Z):
   profits=p*tf.minimum(Z,Y)+d*tf.maximum(Z-Y,0)-c*Z
   sum_profit=tf.reduce_sum(profits)
   return -sum_profit
model.compile(loss=fun_loss, optimizer='adam')
model.fit(X,Y)
```

Discrete variable *z*

Let *s* be the number of possible decisions, and let the decision variable *z* take values from 1 to *s*:

$$z \in \{1, 2, \ldots, s\}. \tag{12}$$

In accordance with above, a decision is made after *x* gets known but before *y* gets known.

Let p_k denote the probability of taking the *k*-th decision, and let *p* denote the vector of such probabilities:

$$p = (p_1, \ldots, p_s). \tag{13}$$

Vector (13) is known as a mixed strategy.

In this case we interpret the output $f(x,\beta)$ of a relevant network as the vector of probabilities of taking different decisions.

So in this case

$$f(x,\beta)=\left[f_1(x,\beta),\ldots,f_s(x,\beta)\right], \tag{14}$$

$$p_k = f_k(x,\beta), \qquad k\in\{1,2,\ldots,s\}, \tag{15}$$

$$p = f(x,\beta). \tag{16}$$

Vector (14) can be obtained by using the softmax activation function at the output layer of a neural network.

Using formula (16) we can obtain the mixed strategy $p_i=\left(p_{i1},\ldots,p_{is}\right)$ corresponding to the *i*-th observation and any given values of parameters β:

$$p_i = f(x_i,\beta), \qquad i\in\{1,2,\ldots,n\}, \tag{17}$$

$$p_{ik} = f_k(x_i,\beta), \qquad i\in\{1,2,\ldots,n\}, \quad k\in\{1,2,\ldots,s\}. \tag{18}$$

Remark 2. In the case of $s=2$ the output of a neural network can be scalar and can be obtained by using sigmoid (logistic) activation function.

As above, we use the notation $u(z,y)$ for the utility from the decision *z* if the state *y* occurs.

Let $v(p,y)$ denote the expected utility from using a mixed strategy (13) if the state *y* occurs:

$$v(p,y)=\sum_{k=1}^{s}u(k,y)\,p_k\ . \tag{19}$$

Let P denote a sequence of mixed strategies for the given series of observations:

$$P=\left(p_1,\ldots,p_n\right). \tag{20}$$

Using notation (2) and (20) formulas (17) can be written as

$$P = F(X,\beta). \tag{21}$$

According to (19) the expected utility from using a mixed strategy (13) for the *i*-th observation is as follows:

$$v(p,y_i)=\sum_{k=1}^{s}u(k,y_i)\,p_{ik}\ . \tag{22}$$

Let $V(P,Y)$ denote a total expected utility (benefit) function over the given series of n observations:

$$V(P,Y)=\sum_{i=1}^{n}v(p_i,y_i). \tag{23}$$

Substituting (22) into (23) yields:

$$V(P,Y)=\sum_{i=1}^{n}\sum_{k=1}^{s}u(k,y_i)\,p_{ik}\ . \tag{24}$$

Substituting (18) and (21) into (24) we have:

$$V\left[F(X,\beta),Y\right]=\sum_{i=1}^{n}\sum_{k=1}^{s}u(k,y_i)\,f_k(x_i,\beta)\,. \tag{25}$$

The optimal values β^* of parameters β provide the mixed strategies $P^*=F(X,\beta^*)$ which maximize the total expected utility (25) for the given observations (x_i,y_i), $i\in\{1,2,\ldots,n\}$.

While using a relevant trained neural network in real world, the decision can be chosen based on the maximum probability in the obtained mixed strategy, i.e.

$$z^*=\arg\max_{k\in\{1,2,\ldots,s\}} f_k(x,\beta)\,. \tag{26}$$

Remark 3. In the case of $s=2$ the function $f(x,\beta)$ can be scalar (in a accordance with Remark 2), in which case formula (25) takes the form:

$$V\left[F(X,\beta),Y\right]=\sum_{i=1}^{n}\left\{u(1,y_i)\,f(x_i,\beta)+u(2,y_i)\left[1-f(x_i,\beta)\right]\right\}, \tag{27}$$

and the rule (26) takes the form:

$$z^*=\begin{cases}1, & \text{if } f(x,\beta)\ge 0.5,\\ 2, & \text{if } f(x,\beta)<0.5.\end{cases} \tag{28}$$

Example 2

A financial institution gives loans whose outcomes (i.e. defaults or absence of defaults) can be of *r* kinds:

$$y\in\{1,2,\ldots,r\}\,. \tag{29}$$

For each kind of outcome the relevant expected profit a_y is known. The outcome *y* for a loan depends (stochastically) on the indicators $x=(x_1,\ldots,x_m)$ describing the borrower. Using the values of the indicators $x=(x_1,\ldots,x_m)$ for the potential borrower, the financial institution should decide whether to give or reject the loan. Observations (x_i,y_i), $i\in\{1,2,\ldots,n\}$ for the indicators $x=(x_1,\ldots,x_m)$ and outcomes *y* are known over *n* past loans. Using this data we want to teach a neural network to recommend the financial institution whether to give or reject the loan based on the current values of indicators $x=(x_1,\ldots,x_m)$ for the relevant potential borrower.

In this case there are two kinds of decisions. So $s=2$ and $z\in\{1,2\}$. Let us agree that $z=1$ if the loan is given, and $z=2$ if the loan is rejected. According to the above, the profits $u(z,y)$ are determined as follows:

$$u(1,y)=a_y, \qquad u(2,y)=0, \qquad y\in\{1,2,\ldots,r\}. \tag{30}$$

We interpret the (scalar) output $f(x,\beta)$ of a relevant neural network as the probability of giving the loan. So in this case formula (27) gives the total expected profit (for *n* past loans). Due to (30) formula (27) reduces to

$$V\left[F(X,\beta),Y\right]=\sum_{i=1}^{n}u(1,y_i)\,f(x_i,\beta). \tag{31}$$

The optimal values β^* of parameters β provide the probabilities $P^*=F(X,\beta^*)$ of giving the loans which maximize the total expected profit (31) (for *n* past loans) for the given indicators $x_i=(x_{i1},\ldots,x_{im})$ and outcomes y_i, $i\in\{1,2,\ldots,n\}$.

A relevant implementation in Keras is given below for the simplest case of an one-layer network with the sigmoid activation function:

```
model = Sequential()
model.add(Dense(1, input_dim=m, activation='sigmoid'))
def fun_loss(profits,probabilities):
   expected_profits=profits*probabilities
   sum_expected_profits=tf.reduce_sum(expected_profits)
   return -sum_expected_profits
model.compile(loss=fun_loss, optimizer='adam')
profits=np.array([A[y-1] for y in Y])
model.fit(X,profits)
```

Example 3

One unit of a financial asset can be bought (or not) at time *t*. If the financial asset is bought at time *t*, it is necessarily sold at the next time $t+1$. We want to teach a neural network to recommend us buying or not buying one unit of the asset based on the asset's current and past prices. A sequence $S=(S_1,S_2,\ldots,S_n)$ of the asset's prices is available for the network's training.

Let us use *m* prices (current and latest past ones) for modeling the decision whether to buy or not to buy one unit of the asset, and use the following notation:

$$x_i=(S_i,S_{i-1},\ldots,S_{i-m+1}), \qquad i\in\{m,m+1,\ldots,n-1\}, \tag{32}$$

$$y_i=S_{i+1}-S_i, \qquad i\in\{m,m+1,\ldots,n-1\}. \tag{33}$$

In this case there are two kinds of decisions. So $s = 2$ and $z \in \{1, 2\}$.

Let us agree that $z = 1$ if one unit of the asset is bought, and $z = 2$ otherwise (if it is not traded). According to the setting of the problem and formula (33), the profits $u(z, y_i)$ are determined as follows:

$$u(1, y_i) = y_i, \qquad u(2, x_i, y_i) = 0, \qquad i \in \{m, m+1, \ldots, n-1\}. \tag{34}$$

We interpret the (scalar) output $f(x, \beta)$ of a relevant neural network as the probability of buying one unit of the asset. So in this case formula (27) gives the total expected profit for $n - m$ past trading periods (from m to $n-1$). Due to (34) formula (27) reduces to

$$V\left[F(X, \beta), Y\right] = \sum_{i=m}^{n-1} y_i f\left(S_i, S_{i-1}, \ldots, S_{i-m+1}; \beta\right). \tag{35}$$

The optimal values β^* of parameters β provide the probabilities $P^* = F(X, \beta^*)$ of buying one unit of asset (at times $i \in \{m, m+1, \ldots, n-1\}$), which maximize the total expected profit (35) for the given series of the asset's prices.

Remark 4. If the trading of fractions of the asset is allowed, a probability of buying one unit of the asset can be interpreted as the relevant fraction of the asset to be bought. (For example, $p = 0.7$ would mean that the relevant fraction of the asset is bought. If one unit of the asset consists of 100 smaller units, than $p = 0.7$ would mean that 70 smaller units are bought.) In such a case formula (35) would give the total profit for $n - m$ trading periods (from m to $n-1$).

A relevant implementation in Keras is given below for the simplest case of an one-layer network with the sigmoid activation function:

```
model = Sequential()
model.add(Dense(1, input_dim=m, activation='sigmoid'))
def fun_loss(Y,probabilities):
   expected_profits=Y*probabilities
   sum_expected_profits=tf.reduce_sum(expected_profits)
   return -sum_expected_profits
model.compile(loss=fun_loss, optimizer='adam')
X=np.array([[S[i-j+m-1] for j in range(m)] for i in range(n-m)])
Y=S[m:]-S[m-1:-1]
model.fit(X,Y)
```

References

1. Bishop, C.M., 2006. *Pattern Recognition and Machine Learning*. Springer.

2. Goodfellow, I., Y. Bengio, A. Courville, 2016. *Deep Learning.* MIT Press.
3. Haykin, S., 2008. *Neural Networks and Learning Machines.* Pearson.
4. Sutton, R.S., A.G. Barto, 2018. *Reinforcement Learning: An Introduction*. Bradford Books.