Graph Neural Network Fundamentals

Mathematics and implementation

Dr. Arafat Khan

December 10, 2022

Presentation Overview

1. Node, edge and graph labeling
   - Labels of a graph
   - Local transition and output functions
   - Global transition and output functions
2. Mathematics of GNN
   - Computation of State
   - The Learning Algorithm
   - Gradient computation
   - Contractive transition function
3. Implementation
   - Pseudocode
   - Demo

This discussion is based on the paper *"The Graph Neural Network Model"* [SGT$^+$09].

Presentation Overview

1. Node, edge and graph labeling
   - Labels of a graph
   - Local transition and output functions
   - Global transition and output functions
2. Mathematics of GNN
   - Computation of State
   - The Learning Algorithm
   - Gradient computation
   - Contractive transition function
3. Implementation
   - Pseudocode
   - Demo

Labels of a graph (node, edge and graph labels)

A graph G is a pair (N, E), where $N = \{n_1, n_2, \dots, n_{|N|-1}, n_{|N|}\}$ is the set of nodes and $E = \{e_1, e_2, \dots, e_{|E|-1}, e_{|E|}\}$ is the set of edges.

$$\text{label for node } n \;\; \ell_n : n \mapsto \mathbb{R}^{d_N}, \quad \text{where } n \in N$$

$$\text{label for edge } e \;\; \ell_e : e \mapsto \mathbb{R}^{d_E}, \quad \text{where } e \in E$$

For $\mathfrak{N} = \{\mathfrak{n}_1, \mathfrak{n}_2, \dots, \mathfrak{n}_{|\mathfrak{N}|-1}, \mathfrak{n}_{|\mathfrak{N}|}\} \subseteq N$. we define,

$$\ell_{\mathfrak{N}} : \mathbb{R}^{d_N \times |\mathfrak{N}|} \mapsto \mathbb{R}^{d_N} \;\; \text{as} \;\; \left(\ell_{\mathfrak{n}_1}, \ell_{\mathfrak{n}_2}, \dots, \ell_{\mathfrak{n}_{|\mathfrak{N}|-1}}, \ell_{\mathfrak{n}_{|\mathfrak{N}|}}\right) \mapsto \mathbb{R}^{d_N}$$

For $\mathfrak{E} = \{\mathfrak{e}_1, \mathfrak{e}_2, \dots, \mathfrak{e}_{|\mathfrak{E}|-1}, \mathfrak{e}_{|\mathfrak{E}|}\} \subseteq E$, we define,

$$\ell_{\mathfrak{E}} : \mathbb{R}^{d_E \times |\mathfrak{E}|} \mapsto \mathbb{R}^{d_E} \;\; \text{as} \;\; \left(\ell_{\mathfrak{e}_1}, \ell_{\mathfrak{e}_2}, \dots, \ell_{\mathfrak{e}_{|\mathfrak{E}|-1}}, \ell_{\mathfrak{e}_{|\mathfrak{E}|}}\right) \mapsto \mathbb{R}^{d_E}$$

For the entire graph,

$$\ell : \mathbb{R}^{d_N \times |N|} \times \mathbb{R}^{d_E \times |E|} \mapsto \mathbb{R}^{D}$$

$$\text{as} \;\; \left(\ell_{n_1}, \ell_{n_2}, \dots, \ell_{|N|-1}, \ell_{|N|}, \ell_{e_1}, \ell_{e_2}, \dots, \ell_{|E|-1}, \ell_{|E|}\right) \mapsto \mathbb{R}^{D}$$

States of a nodes

Let us take $n \in N$ from the graph $G = (N, E)$ and define the following,

- $\ell_n \in \mathbb{R}^{d_N}$: node label of $n \in N$.
- $\ell_{\mathcal{N}(n)} \in \mathbb{R}^{d_N}$: label of the neighborhood $\mathcal{N}(n) = \{\nu_1, \nu_2, \ldots, \nu_{|\mathcal{N}(n)|-1}, \nu_{|\mathcal{N}(n)|}\} \subseteq N$.
- $\mathcal{E}(n)$: edges connected to n i.e $\mathcal{E}(n) = \{(\nu_1, n), (\nu_2, n), \ldots, (\nu_{|\mathcal{N}(n)|-1}, n), (\nu_{|\mathcal{N}(n)|}, n)\}$.
- $\ell_{\mathcal{E}(n)}$: edge label of $\mathcal{E}(n)$.

Now we can define something called the *state* of n, denoted as $x_n \in \mathbb{R}^s$, which captures the the local information of n using its own node label $\ell_n \in \mathbb{R}^{d_N}$, node label of the neighbors $\ell_{\mathcal{N}(n)} \in \mathbb{R}^{d_N}$, connected edges lalel $\ell_{\mathcal{E}(n)}$, and the *states* of the neighbors $x_{\mathcal{N}(n)} \in \mathbb{R}^s$.

Q. How to calculate x_n? **Ans.** *local transition function*.

Local transition and output functions

local transition function

$$f_w : \mathbb{R}^{d_N} \times \mathbb{R}^{d_E} \times \mathbb{R}^{s} \times \mathbb{R}^{d_N} \mapsto \mathbb{R}^{s}$$

$$\text{as} \quad \left(\ell_n, \ell_{\mathcal{E}(n)}, x_{\mathcal{N}(n)}, \ell_{\mathcal{N}(n)}\right) \mapsto \mathbb{R}^{s}.$$

$$\text{where} \quad \left(\ell_{(\nu_1,n)}, \ell_{(\nu_2,n)}, \ldots, \ell_{(\nu_{|\mathcal{N}(n)|-1},n)}, \ell_{(\nu_{|\mathcal{N}(n)|},n)}\right) := \ell_{\mathcal{E}(n)}$$

$$\left(x_{\nu_1}, x_{\nu_2}, \ldots, x_{\nu_{|\mathcal{N}(n)|-1}}, x_{\nu_{|\mathcal{N}(n)|}}\right) := x_{\mathcal{N}(n)}$$

$$\left(\ell_{\nu_1}, \ell_{\nu_2}, \ldots, \ell_{\nu_{|\mathcal{N}(n)|-1}}, \ell_{\nu_{|\mathcal{N}(n)|}}\right) := \ell_{\mathcal{N}(n)}$$

local output function

$$g_w : \mathbb{R}^{s} \times \mathbb{R}^{d_N} \mapsto \mathbb{R}^{r} \;\text{ as }\; (x_n, \ell_n) \mapsto \mathbb{R}^{r}$$

$$\textit{local transition function}, \;\; x_n = f_w\left(\ell_n, \ell_{\mathcal{E}(n)}, x_{\mathcal{N}(n)}, \ell_{\mathcal{N}(n)}\right)$$

$$\textit{local output function}, \;\; o_n = g_w(x_n, \ell_n)$$

Global transition and output functions

Global transition function

$$
\begin{aligned}
x_{n_1} &= f_w\left(\ell_{n_1}, \ell_{\mathcal{E}(n_1)}, x_{\mathcal{N}(n_1)}, \ell_{\mathcal{N}(n_1)}\right)\\
x_{n_2} &= f_w\left(\ell_{n_2}, \ell_{\mathcal{E}(n_2)}, x_{\mathcal{N}(n_2)}, \ell_{\mathcal{N}(n_2)}\right)\\
&\vdots\\
x_{n_{|N|}} &= f_w\left(\ell_{n_{|N|}}, \ell_{\mathcal{E}(n_{|N|})}, x_{\mathcal{N}(n_{|N|})}, \ell_{\mathcal{N}(n_{|N|})}\right)
\end{aligned}
$$

$$x = F_w(x, \ell)$$

Global output function

$$
\begin{aligned}
o_{n_1} &= g_w(x_{n_1}, \ell_{n_1})\\
o_{n_2} &= g_w(x_{n_2}, \ell_{n_2})\\
&\vdots\\
o_{n_{|N|}} &= g_w(x_{n_{|N|}}, \ell_{n_{|N|}})
\end{aligned}
$$

$$o = G_w(x, \ell_N)$$

Presentation Overview

1. Node, edge and graph labeling
   - Labels of a graph
   - Local transition and output functions
   - Global transition and output functions
2. Mathematics of GNN
   - Computation of State
   - The Learning Algorithm
   - Gradient computation
   - Contractive transition function
3. Implementation
   - Pseudocode
   - Demo

Contraction mapping theorem

Definition: Contraction map

If (X, d_X) and (Y, d_Y) are two metric spaces, then $T : X \to Y$ is a contractive mapping if there is a constant $q \in [0, 1)$ such that

$$d_Y(T(x), T(x')) \leq q\, d_X(x, x')$$

for all x and x' in X.

Lemma

Contraction map $\implies$ Lipschitz continuous map $\implies$ Uniformly continuous map $\implies$ Continuous map.

Proof: Use definitions. For further detail follow Appendix A of the note.

Contraction mapping theorem

Theorem: Contraction mapping theorem

Let (X, d) be a non-empty complete metric space with a contraction mapping $T : X \to X$. Then

- T admits a unique fixed-point $x^* \in X$ (i.e. $T(x^*) = x^*$).
- x^* can be found by starting with an arbitrary element $x_0 \in X$ and defining a sequence $(x_n)_{n \in \mathbb{N}}$ by $x_n = T(x_{n-1})$ for $n \geq 1$. Then $\lim_{n\to\infty} x_n = x^*$.

Proof: For $m, n \in \mathbb{N}$ with $m > n$, we have,

$$d(x_m, x_n) \leq q^n d(x_1, x_0) \implies (x_n) \text{ Cauchy.}$$

Prove uniqueness by contraction. $\lim_{n\to\infty} x_n = x^*$ no matter where $x_0 \in X$ is [see Appendix A of the note for the detail].

Computation of the State

Assuming the transition function is a contractive map the outputs and the states can be computed by iterating,

$$x_n(t+1) = f_w\left(\ell_n, \ell_{\mathcal{E}(n)}, x_{\mathcal{N}(n)}(t), \ell_{\mathcal{N}(n)}\right)$$
$$o_n(t) = g_w(x_n(t), \ell_n)$$

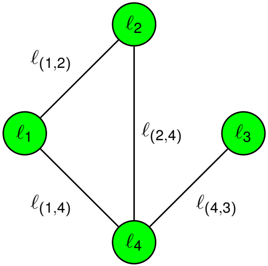
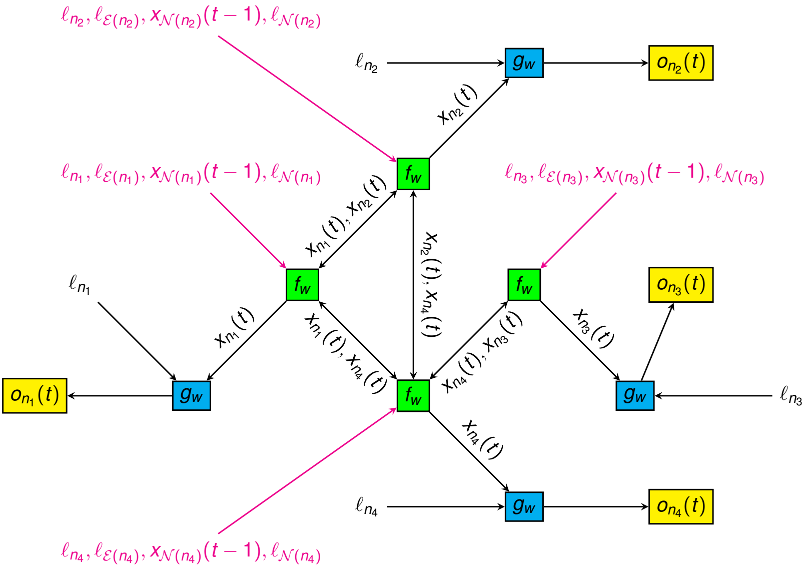
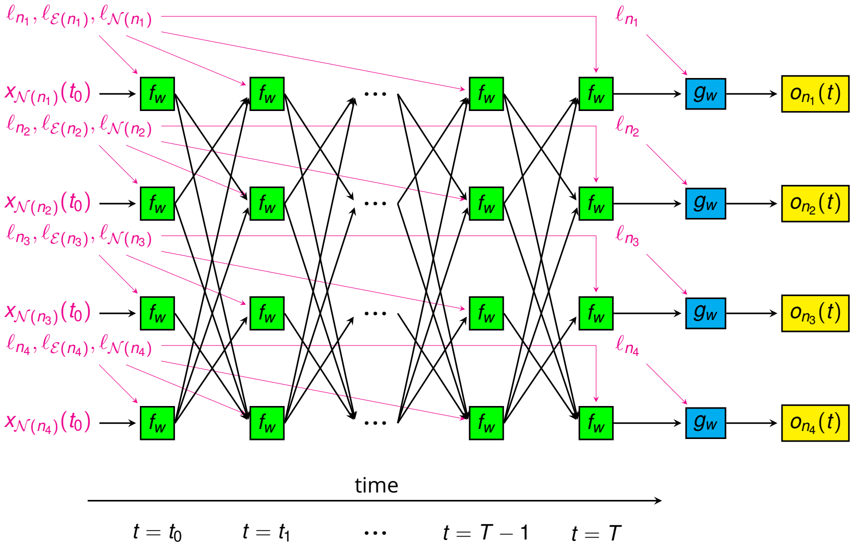


Figure: Example with a graph

Computation of the State (encoding network)



Computation of the State (unfolded network)



The Learning Algorithm (domain)

Let us sonsider an arbitrary set of graphs $\mathcal{G}$.

$$\mathcal{G} = \bigcup_{\alpha \in \Lambda} G_\alpha, \quad \text{where } G_\alpha = (N_\alpha, E_\alpha)$$

Let us also consider,

$$\mathcal{N} = \bigcup_{\alpha \in \Lambda} N_\alpha$$

Definition: domain of a supervised learning framework

The domain is the set of pairs of a graph and a node, i.e. $\mathcal{D} = \mathcal{G} \times \mathcal{N}$.

The Learning Algorithm (learning set)

Definition: learning Set

We take a finite set of graphs from $\mathcal{G}$ as,

$$\{G_1, G_2, \ldots, G_i, \ldots, G_{p-1}, G_p\} \subseteq \mathcal{G} \implies p \leq |\mathcal{G}|.$$

Let us take a subset of the node set of $G_i = (N_i, E_i)$ as,

$$\{n_{i,1}, n_{i,2}, \ldots, n_{i,j}, \ldots, n_{i,q_i-1}, n_{i,q_i}\} \subseteq N_i \in \mathcal{N} \implies q_i \leq |N_i|.$$

We assume a supervised learning framework with the following set,

$$\mathcal{L} = \{(G_i, n_{i,j}, t_{i,j}) | G_i = (N_i, E_i) \in \mathcal{G}, t_{i,j} \in \mathbb{R}^m\}$$

where $n_{i,j} \in N_i, 1 \leq i \leq p, 1 \leq j \leq q_i$. $\mathcal{L}$ is called the learning set.

The Learning Algorithm (loss function)

Learning in GNNs consists of estimating the parameter w such that the following function,

$$\varphi_w : \mathcal{D} \rightarrow \mathbb{R}^m$$

approximates the data in the learning set. $\varphi_w(G, n)$ is essentially the result of the global output function corresponding to the state of node n. The learning task can be posed as the minimization of a quadratic cost function,

$$e_w = \sum_{i=1}^{p} \sum_{j=1}^{q_i} \left(t_{i,j} - \varphi(G_i, n_{i,j})\right)^2 + \text{regularization terms.}$$

As common in neural network applications, the cost function may include a penalty term to control other properties of the model.

The Learning Algorithm

Repeat the following steps over and over again until some stopping criterion is satisfied.

1. **`Forward (w)`:** Update the states $x_n(t)$ iteratively by

$$x_n(t) = f_w\left(\ell_n, \ell_{\mathcal{E}(n)}, x_{\mathcal{N}(n)}(t-1), \ell_{\mathcal{N}(n)}\right)$$
$$o_n(t) = g_w(x_n(t), \ell_n)$$

   until at time $t = T$ they approach the fixed point solution of $x(T) \approx x^*$. The hypothesis that F_w is a contraction map ensures the convergence to the fixed point.

2. **`Backward` (x^*, w):** Compute he gradient $\dfrac{\partial e_w(T)}{\partial w}$.

3. **`Update weights`:** The weights w are updated according to the gradient

$$w \leftarrow w - \lambda \frac{\partial e_w}{\partial w}$$

Gradient computation

Fundamental theorem of GNN

Let $F_w(x, \ell)$ and $G_w(x, \ell_N)$ are continuously differentiable w.r.t. x and w. Then we have the following,

1. **Differentiability:** φ_w is continuously differentiable w.r.t. w.
2. **Backpropagation:** Let $z(t)$ be defined as,

$$z(t) = z(t+1) \cdot \left(\frac{\partial F_w}{\partial x}\right)(x, \ell) + \frac{\partial e_w}{\partial o} \cdot \left(\frac{\partial G_w}{\partial x}\right)(x, \ell_N).$$

Then, $z = \lim_{t \to -\infty} z(t)$. The convergence is exponential and independent of the initial state $z(T)$. Then the gradient of e_w w.r.t. w is,

$$\frac{\partial e_w}{\partial w} = \frac{\partial e_w}{\partial o} \cdot \left(\frac{\partial G_w}{\partial w}\right)(x, \ell_N) + z \cdot \frac{\partial F_w}{\partial x}.$$

Differentiability of φ_w

Proof: Steps of the proof are,

- Define $\Theta(x, w) = x - F_w(x, \ell) \implies \Theta$ is continuously differentiable and $\left(\frac{\partial \Theta}{\partial x}\right)(x, w) = I_{s|N|} - \left(\frac{\partial F_w}{\partial x}\right)(x, \ell)$.
- Using the convergence criterion of *Neumann series*,

$$\left(I_{s|N|} - \left(\frac{\partial F_w}{\partial x}\right)\right)\left(I_{s|N|} + \left(\frac{\partial F_w}{\partial x}\right) + \left(\frac{\partial F_w}{\partial x}\right)^2 + \cdots\right) = I_{s|N|}$$

$I_{s|N|} - \left(\frac{\partial F_w}{\partial x}\right)$ is invertible so is $\left(\frac{\partial \Theta}{\partial x}\right)(x, w)$.

- Using *implicit function theorem* $\exists$ open U containing w such that $\exists! \Psi \in \mathcal{C}^1 \ni \Psi(w) = x$, and $\Theta(\Psi(w), w) = 0, \forall w \in U$. $\forall w$ this argument is true $\implies \Psi \in \mathcal{C}^1$ on whole domain. As $\varphi_w(G, n) = [G_w(\Psi(w), \ell_N)]_n \implies \varphi_w$ is the composition of $\mathcal{C}^1$ functions.

Convergence of $z(t)$

Proof: Sketch of the proof is,

$$z(t) = z(t+1) \cdot \left(\frac{\partial F_w}{\partial x}\right)(x, \ell) + \frac{\partial e_w}{\partial o} \cdot \left(\frac{\partial G_w}{\partial x}\right)(x, \ell_N)$$

$$\implies z(t-1) - z(t) = (z(t) - z(t+1)) \cdot \left(\frac{\partial F_w}{\partial x}\right)(x, \ell)$$

Using operator norm property and the fact that F_w contractive, we have for $\mu \in [0, 1)$,

$$\|z(t-1) - z(t)\| \leq \mu \|(z(t) - z(t+1))\|$$

$$\implies \|z(t-m) - z(t-m+1)\| \leq \mu^m \|(z(t) - z(t+1))\|$$

$$\implies z(t) \text{Cauchy in complete metric space.}$$

Calculation of $\frac{\partial e_w}{\partial w}$

Proof: $z(t) \approx z(t+1)$ for some sufficiently large t. Then we have,

$$z = \frac{\partial e_w}{\partial o} \cdot \left(\frac{\partial G_w}{\partial x}\right)(x, \ell_N) \cdot \left(I_{s|N|} - \left(\frac{\partial F_w}{\partial x}\right)(x, \ell)\right)^{-1}$$

Using $\Psi(w) = F_w(\Psi(w), \ell)$ and differentiating $\Theta(\Psi(w), w) = \mathbf{0}$,

$$\frac{\partial \Psi}{\partial w} = \left(I_{s|N|} - \left(\frac{\partial F_w}{\partial x}\right)(x, \ell)\right)^{-1} \cdot \left(\frac{\partial F_w}{\partial w}\right)(x, \ell)$$

Thus finally we have,

$$\begin{aligned}\frac{\partial e_w}{\partial w} &= \frac{\partial e_w}{\partial o} \cdot \frac{\partial G_w}{\partial w}(x, \ell_N) + \frac{\partial e_w}{\partial o} \cdot \frac{\partial G_w}{\partial x}(x, \ell_N) \cdot \frac{\partial \Psi}{\partial w} \\ &= \frac{\partial e_w}{\partial o} \cdot \frac{\partial G_w}{\partial w}(x, \ell_N) + z \cdot \left(\frac{\partial F_w}{\partial w}\right)(x, \ell)\end{aligned}$$

Contractive transition functions - two examples

1. **Linear:** Any global transition function of the form $F_w(x, \ell) = Ax + b$ is contractive for all w. But not necessarily this simple linear function will provide enough predictive power.
2. **Nonlinear:** If F_w is the three-layered neural network we guarantee the function to be a universal approximator. However, not all the parameters w ensures that F_w is a contraction map. This can be solved by adding a penalty term to the loss function. e.g.

$$e_w = \sum_{i=1}^{p} \sum_{j=1}^{q_i} \left(t_{i,j} - \varphi(G_i, n_{i,j})\right)^2 + \beta L\left(\left\|\frac{\partial F_w}{\partial w}\right\|\right).$$

where, with desired contraction constant $\mu \in (0, 1)$ we define L,

$$L(y) = \begin{cases} (y-\mu)^2 \text{if } y > \mu, \\ 0 \text{otherwise.} \end{cases}$$

Presentation Overview

1. Node, edge and graph labeling
   - Labels of a graph
   - Local transition and output functions
   - Global transition and output functions
2. Mathematics of GNN
   - Computation of State
   - The Learning Algorithm
   - Gradient computation
   - Contractive transition function
3. Implementation
   - Pseudocode
   - Demo

Pseudocode - Main Procedure

Algorithm 1 Main procedure

$\mathcal{S} \leftarrow$ some stopping criterion depends typically on requirements
$\lambda \leftarrow$ learning rate
$\epsilon_f \leftarrow$ convergence tolerance of contractive map F_w
$\epsilon_z \leftarrow$ convergence tolerance of contractive map z
$w \leftarrow$ random weight initialization
$x, T \leftarrow$ **FORWARD**(w)
REPEAT:
    $\frac{\partial e_w}{\partial w} \leftarrow$ **BACKWARD**(x, w, T)
    $w \leftarrow w - \lambda \frac{\partial e_w}{\partial w}$
    if $\mathcal{S}$ satisfied
        return w

Pseudocode - Forward propagation

Algorithm 2 Forward procedure

1: **FORWARD**(w)
2: $t \leftarrow 0$
3: $x(0) \leftarrow$ # random initialization of initial states.
4: **REPEAT:**
5: $x(t+1) \leftarrow F_w(x(t), \ell)$
6: **if** $\|x(t) - x(t+1)\| \leq \epsilon_f$ **then return** $x(t+1), t$
7: **else** $t \leftarrow t + 1$

In line 3 random initialization is OK, as the fixed point does not depend on the initial state. Line 5 ensures the construction of Cauchy sequence thus by contraction mapping theorem we must have the condition in Line 6 satisfied at some point of time for any $\epsilon_f \in \mathbb{R}_{>0}$.

Pseudocode - Backward propagation

Algorithm 3 Backward procedure

BACKWARD(x, w, T)

$o \leftarrow G_w(x, \ell_N), \quad A \leftarrow \frac{\partial F_w}{\partial x}(x, \ell), \quad b \leftarrow \frac{\partial e_w}{\partial o} \cdot \frac{\partial G_w}{\partial x}(x, \ell_N)$

$z(T) \leftarrow$ random initialization, $\quad t \leftarrow T - 1$

REPEAT:

$z(t) \leftarrow z(t+1) \cdot A + b$

if $\|z(t) - z(t+1)\| \leq \epsilon_z$ **then break**

else $\quad t \leftarrow t - 1$

$C \leftarrow \frac{\partial F_w}{\partial w}(x, \ell), \quad d \leftarrow \frac{\partial e_w}{\partial o} \cdot \frac{\partial G_w}{\partial w}(x, \ell_N)$

$\frac{\partial e_w}{\partial w} \leftarrow z(t) \cdot C + d$

return $\frac{\partial e_w}{\partial w}$

Python example

Let's see everything in action...

The End

Questions? Comments?

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini, *The graph neural network model*, IEEE Transactions on Neural Networks **20** (2009), no. 1, 61–80.