Universidad de Sevilla

Federated Learning y Algoritmos de Agregación.

Fernando Cañizares Romero

10 de febrero de 2024

Índice general

1 Introducción al Federated Learning.

Un producto de Machine Learning se caracteriza por tres elementos esencialmente:

1. **El modelo matemático usado**. Podemos encontrar múltiples de ellos tanto definidos en textos científicos como en distintas APIs y lenguajes de programación, para poder usarlos en las computadoras.

2. **Los parámetros del modelo**. Estos serían los elementos del modelo que no dependen del entrenamiento, son fijos y pueden variar los resultados finales. Por ejemplo, el número de árboles de un modelo tipo bosque aleatorio.

3. **Los datos para el entrenamiento**. Gracias a los datos, los modelos se pueden entrenar y se consigue ajustar sus pesos (valores internos que dependen del entrenamiento). Estos son los elementos más importantes del producto, ya que mediante ellos se puede generalizar el problema en cuestión.

   Por ello, es importante que los datos estén bien definidos (evitando los outliers, sesgos, etc...), que se tenga una cantidad adecuada para que el entrenamiento sea consistente y es vital tener disponibilidad de los datos.

Es en la dificultad de disponer de los datos donde el Federated Learning toma importancia, ya que en la vida real, generalmente, es difícil acceder a los datos. Ya porque exista escasez, estén repartidos en diferentes lugares, por problemas de privacidad, por sensibilidad de los datos, etc...

El Federated Learning se basa en dar la vuelta a la forma habitual de crear un producto Machine Learning. En la que los equipos tendrían acceso directo a los datos, o bien porque se hayan compartido o porque se disponga de forma remota. Por consiguiente, la idea del Federated Learning consiste en enviar los modelos (prentrenados o no) a la máquina donde se encuentran los datos, entrenar allí el modelo y por último, recuperar éste ya entrenado en nuestra máquina.

Por lo que el Federated Learning es la forma de aplicar Machine Learning en situaciones donde el proceso de entrenamiento no pueda centralizarse haciendo que no sea necesario compartir conjuntos de datos con una entidad central. Los modelos se entrenan de forma colaborativa entre múltiples partes o clientes. El término *partes* puede referirse a entidades tan diferentes como los centros de datos de una empresa, clústeres de cómputo en diferentes nubes, teléfonos móviles, automóviles o distintas empresas y organizaciones. El proceso de aprendizaje requerirá una unificación de los distintos modelos obtenidos en cada una de las partes dentro de un nodo (o varios) central o nodo de agregación, esta unificación tendría que recoger los pesos de los modelos resultantes y después mediante un algoritmo agregar todos los pesos para crear un modelo general (algoritmo de fusión o agregación) en el que se recoja la información de los resultados, este proceso se llamará **agregador**.

Los algoritmos de fusión más generales son:

1. **Federated stochastic gradient descent** (FedSGD): Este algoritmo de fusión es la transposición directa del descenso del gradiente estocástico aplicado sobre un entorno federado. Utiliza una fracción aleatoria C de las partes, usando todos los datos, con los que el servidor promedia los gradientes proporcionalmente (o no ponderadas) al número de muestras de entrenamiento en cada parte realizando así un paso del descenso de gradiente [15].

2. **Federating Average** (FedAVG): Este algoritmo es una generalización de FedSGD. Permite que las partes puedan realizar más de una actualización de los batches de datos locales y a continuación intercambiar los pesos actualizados en lugar de los gradientes. El fundamento FedAVG es que en FedSGD, si todas las partes comienzan con la misma inicialización, promediar los gradientes es estrictamente equivalente a promediar los pesos de los modelos locales. Además, promediar ponderaciones ajustadas (o no ponderadas) provenientes de la misma inicialización no necesariamente perjudica el desempeño del modelo promediado resultante [5].

Se comentarán estos métodos con detalle en secciones posteriores.

1.1. Conceptos y Terminología

Los productos basado en Federated Learning (**FL**) entrenan un modelo $\mathcal{M}$ sobre un conjunto de datos $\mathcal{D}$, $\mathcal{D}$ está dividido en n partes, donde cada parte P_i tiene su propio

conjunto de entrenamiento D_i. Un proceso **FL** incluye un agregador A que no tiene constancia de ningún conjunto de datos.

Para entrenar un modelo global $\mathcal{M}_g$ el agregador y las partes forman parte del algoritmo FL creando una comunicación entre ellos. El proceso se ejecuta de la siguiente forma:

1. Para obtener $\mathcal{M}_g$, el agregador usa una función $\mathcal{Q}$ que tiene como entrada el modelo o estado actual del entrenamiento $\mathcal{M}_t$ en la ejecución t, y genera la consulta q_{t+1} [1].

2. la consulta q_t, obtiene información sobre un modelo local, o la información agregada sobre el conjunto de datos de cada parte. Por ejemplo, información sobre los gradientes, los pesos de una red neuronal o conteos para los árboles de decisión.

3. El proceso de entrenamiento local aplica una función $\mathcal{L}$ que usa la consulta q_t y el conjunto de datos local D_i para devolver un modelo actualizado $r_{i,t}$. Normalmente la consulta q_t, contiene información que cada parte puede usar para ejecutar el entrenamiento local, por ejemplo los pesos del modelo con los que iniciar el entrenamiento local.

4. Se manda el resultado $r_{i,t}$ desde la parte P_i al agregador A, recogiendo así los resultados de todas las partes.

5. Cuando se tienen todos los modelos $r_{i,t}$ en el agregador, definimos $R_t = \{r_{1,t}, \ldots, r_{n,t}\}$ como el conjunto de todos los modelos entrenados en la ejecución t. Se aplica entonces en el agregador la función de agregación $\mathcal{F}$ que tiene como entrada R_t y devuelve el modelo $\mathcal{M}_t$.

Este proceso puede ser ejecutado varias veces, y continuar hasta verificar un criterio de terminación, por ejemplo, un número máximo de rondas de entrenamiento k, lo que daría como resultado $\mathcal{M}_g = \mathcal{M}_k$.

La función de entrenamiento local $\mathcal{L}$, la función de fusión $\mathcal{F}$ y la función de generación de consultas $\mathcal{Q}$ generalmente se diseñan para funcionar en conjunto. Por ejemplo, para el caso de una red neuronal, $\mathcal{L}$ sería la red neuronal local. $\mathcal{F}$ sería un algoritmo de agregación que promediara los pesos de los modelos de R_t generando $\mathcal{M}_t$. Y para la siguiente ejecución, $\mathcal{Q}$ pasaría $\mathcal{M}_t$ como parte de la nueva consulta q_{t+1}.

[1] Algunos algoritmos FL pueden incluir entradas adicionales para $\mathcal{Q}$ y adaptar las consultas a cada parte, aunque sin perdida de generalidad se usará esta notación más simple

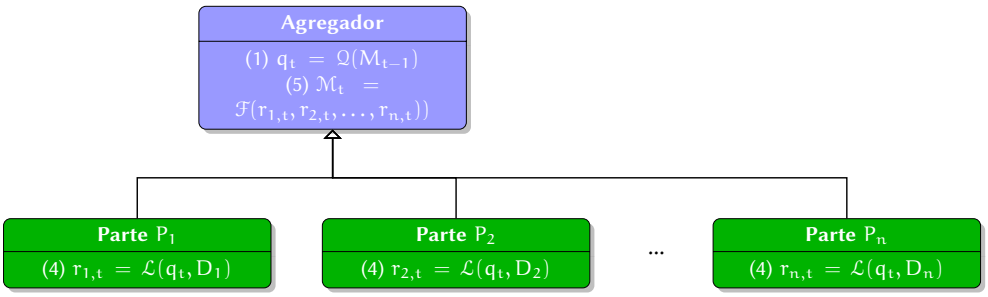

Figura 1.1: *Algoritmo Federated Learning*

El modelo observado en la figura 1.1, puede complicarse más, por ejemplo, asociando a cada parte P_i un agregador A_i, de manera que vaya solicitando información a las demás partes. Existiendo en cada agregados una función de agregación diferente, distintos criterios de parada o La función Q podría determinar las partes a consultar en cada ronda, basándose por ejemplo en los méritos de las contribuciones.

Además el problema FL canónico implica entrenar un modelo global único a partir de los datos almacenados en múltiples dispositivos remotos. Nuestro objetivo es entrenar este modelo bajo la restricción de que los datos generados por el dispositivo se almacenan y procesan localmente, y solo las actualizaciones intermedias del modelo son enviadas periódicamente con un servidor central.

Como conclusión podemos considerar el FL como un problema de optimización, por lo que el FL sería esencialmente un problema de optimización federada. Partimos de un conjunto fijo de n partes, cada uno con un conjunto de datos local fijo D_i. Al comienzo de cada ejecución t_i, el servidor envía el estado del algoritmo global actual w_t a cada una de las partes (por ejemplo, los parámetros del modelo). Luego, cada cliente seleccionado realiza un cálculo local basado en el estado global y su conjunto de datos local $f_i(w)$ y envía una actualización al nodo central. A continuación, el nodo central aplica estas actualizaciones a su estado global w_{t+1} repitiendo el proceso sino se ha alcanzado el factor de parada. Por lo que el problema puede formularse de la siguiente forma:

$$\min_{w \in \mathbb{R}} f(w) \qquad \text{donde} \qquad f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w) \tag{1.1}$$

En un problema de *machine learning*, se suele tomar $f_i(w) = l(w; x_i, y_i)$ como la *loss* de la predicción en la muestra (x_i, y_i) con los pesos w.

Si existen K clientes, y definimos $\mathcal{P}_k$ como el conjunto de los índices de las observaciones en el cliente k, con $n_k = |\mathcal{P}_k|$. Podemos reescribir el problema 1.1 como

sigue:

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \qquad \text{donde} \qquad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w) \tag{1.2}$$

Si la partición $\mathcal{P}_k$ se hubiera formado repartiendo las muestras de entrenamiento sobre los clientes de manera aleatoria, entonces tendríamos $E_{\mathcal{P}_k}(Fk(w)) = f(w)$, donde el valor esperado se calcula sobre el conjunto de muestras asignadas a un cliente k fijo. Ésta es la suposición de IID que suelen realizar los algoritmos de optimización distribuidos, pero en nuestro caso puede ocurrir que esto no se cumpla (es decir, que F_k pudiera ser una aproximación arbitrariamente mala de f).

1.2. Consideraciones a Destacar

Los algoritmos de **FL** nos aportan ventajas muy interesantes, ya que se abre el espectro de colaboración, se dividen los entrenamientos en distintas máquinas, también mantiene la privacidad de los datos sensibles... Pero igualmente se añaden complicaciones, ya que pueden existir ataques para obtener información de datos que generalmente tienen una alta sensibilidad, ya sea porque pertenecen al entorno privado de una compañía u organización, o porque afecten a personas físicas reales. Esto hace que sean necesaria la aplicación de otras herramientas como el *Cifrado Homomórfico* o *Differential Privacy* incrementando por consiguiente el coste computacional y en ciertos casos puede ser un aumento considerable.

El FL requiere una comunicación frecuente entre nodos durante el entrenamiento. Por lo tanto, requiere suficiente potencia de cálculo local, memoria y conexiones con ancho de banda adecuado para poder intercambiar parámetros del modelo.

Sin embargo, el FL evita la comunicación de datos, que puede requerir un alto nivel de recursos antes de iniciar el entrenamiento del modelo centralizado. No obstante, los dispositivos que normalmente se emplean en el aprendizaje federado tienen limitaciones de comunicación, por ejemplo, los dispositivos IoT (*Internet of Things*) o los smartphones generalmente están conectados a redes Wi-Fi, por lo que, aunque los modelos suelan ser menos difíciles de compartir en comparación con los datos sin procesar, los mecanismos del FL pueden no ser adecuados. [2]

En el FL encontramos varios problemas estadísticos a resolver:

1. El problema de heterogeneidad entre los conjuntos de datos de las partes. Cada nodo puede tener algún sesgo con respecto a la población general, y el tamaño de los conjuntos de datos puede variar significativamente. En secciones posteriores se tratará este tema con más detalle. [10].

2. La heterogeneidad también se ve afectada durante el tiempo, ya que la distribución de cada conjunto de datos local puede variar si se van agregando nuevos registros.

3. La interpretabilidad del conjunto de datos de cada parte.

4. Los conjuntos de datos pueden requerir procesos de mantenimiento regulares.

5. Ocultar los datos de entrenamiento podría permitir a los atacantes utilizar *backdoors* en el modelo global; [11].

6. No poder acceder a los datos de entrenamiento hace que sea más difícil identificar sesgos no deseados.

7. Pérdida parcial o total de las actualizaciones del modelo debido a fallas en los nodos que afectan el modelo global. [2]

Pese a estos problemas abiertos el FL está en auge a lo que investigación se refiere, por los beneficios que aporta.

El concepto de ”federación”surge al rededor de 1990 en artículos sobre *federated database systems* (FDBSs), en el que se expone un sistema para obtener beneficio de la cooperación de diferentes bases de datos autónomas [6]. Posteriormente con el nacimiento de los entornos *cloud* se aplica la misma idea y nace el concepto de *federated cloud computing* FC, permite una optimización de costos debido a la externalización parcial de los servicios a regiones más rentables. La migración de recursos y la duplicidad de éstos son dos características básicas de FC [4].

La figura 1.2 muestra el número de artículos en cada año para estas tres áreas de investigación. Aquí contamos los artículos buscando las palabras clave ”*federated database systems*”, ”*federated cloudz* ”*federated learning*.[en] *Google Scholar*. Aunque la base de datos federada se propuso hace 30 años, todavía hay alrededor de 400 artículos que la mencionaron en los últimos años. La popularidad de FC es más notable que FDBS al principio, mientras que disminuye en los últimos años. Para FL, el número de artículos relacionados está aumentando rápidamente. Además, existe una creciente preocupación por la privacidad. Así, se espera que la popularidad de FL siga aumentando durante

algunos años hasta que se alcance cierta madurez y los principales frentes abiertos se hayan cerrado.

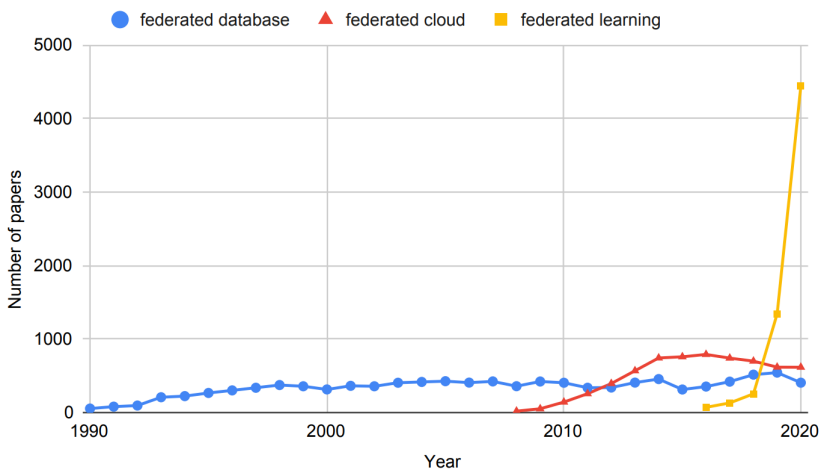

Figura 1.2: *El número de artículos relacionados sobre FDBSs, FC y FL*

Algoritmos de Agregación 2

Los algoritmos de agregación son fundamentales en el contexto del FL, ya que es la única forma de componer un modelo global a partir de los modelos entrenados en las distintas partes, en esta sección se presentarán diferentes algoritmos de agregación presentados en los últimos años junto con el pseudocódigo y algunos de los resultados teóricos más relevantes.

En primer lugar nos referiremos al problema de optimización implícito en el FL como **optimización federada** (*federated optimization*) creando una conexión (y poniéndolo en contraste) con la optimización distribuida. Éste tiene varías propiedades fundamentales clave que diferencian de un problema de Optimización Distribuida:

1. **No IID** (no independientes e idénticamente distribuidos): El conjunto de entrenamiento en un cliente dado está normalmente basado en las características particulares del éste, por ejemplo el uso de que cierta persona puede hacer de una determinada aplicación en su dispositivo. Por lo tanto, el conjunto de datos local de cualquier usuario en particular no será representativo de la distribución de la población.

2. **No balanceado**: Como en el caso anterior, algunos clientes pueden haber creados más registros que otros, lo que lleva a cantidades variables de datos de entrenamiento entre las partes.

3. **Masivamente distribuidos**: Se espera que el número de clientes participando en la optimización sea mayor que el número promedio de muestras por clientes.

4. **Comunicación limitada**: Se puede dar el caso de que los dispositivos se encuentren apagados o con una conexión lenta.

Un sistema de optimización federado también debe abordar otros problemas prácticos: conjuntos locales de los clientes que cambian mientras se agregan y eliminan datos;

disponibilidad de los clientes que participan en un problema con su distribución local de datos (por ejemplo, los dispositivos de estadounidense probablemente se conectarán en una franja horaria diferente que británicos para un problema relacionado con el idioma inglés); y clientes que nunca responden o envían actualizaciones dañadas.

Todos estos asuntos están más allá del alcance del problema práctico que se presentará en este texto.

En un entorno federado, no existe un gran coste temporal al involucrar a más clientes, los experimentos de Chen et al. [32] demuestran que este enfoque es vanguardista (en el entorno de los centro de datos), donde se superan a los enfoques asíncronos. Para aplicar este enfoque en el entorno federado, seleccionamos una fracción de clientes en cada ronda y calculamos el gradiente de la función pérdida sobre todos los datos disponibles en estos clientes. Por lo tanto, C controla el tamaño de lote global. Con $C = 1$ se computaría al descenso de gradiente con el lote completo (no estocástico). [1] En adelante se nombrará a este algoritmo como FederatedSGD (o FedSGD).

2.1. Federated Stochastic Gradient Descent

El éxito del *Deep Learning* se ha basado en la aplicación del algoritmo del descenso del gradiente estocástico (SGD) y sus variantes para la optimización. De hecho, muchos avances se obtienen mediante adaptación de la estructura del modelo (por tanto, la función de pérdida) para que éste sea más sensible a la optimización mediante métodos basados en el gradiente [Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.]. Por lo que es natural empezar con el SGD para construir algoritmos de optimización federada.

Una implementación típica de FedSGD con $C = 1$ y un factor de aprendizaje fijo η hace que cada cliente calcule $g_k = \nabla F_k(w_t)$ el gradiente promedio en los datos locales del modelo w_t, el servidor central agrega los gradientes obtenidos y genera una actualización del modelo $w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} g_k \to w_{t+1}$. Ya que $\sum_{k=1}^{K} \frac{n_k}{n} g_k = \nabla f(w_t)$. Se actualiza equivalentemente para todo k, $w_t - \eta g_k \to w_{t+1}^k$ y luego $\sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \to w_{t+1}$. Es decir, cada cliente realiza localmente un paso de descenso de gradiente en el modelo actual utilizando sus datos locales, y luego el agregador toma un promedio ponderado de los modelos resultantes.

[1] Si bien el mecanismo de selección de lotes es diferente a seleccionar un lote eligiendo individuos al azar, los gradientes del lote g calculados por FedSGD cumplen que $E(g) = \nabla f(w)$.

Para objetivos generales no convexos, promediar modelos en el espacio de parámetros podría producir un modelo arbitrariamente malo.

2.2. Federated Averaging

Podemos agregar más complejidad a FSGD iterando la actualización local $w^k - \eta \nabla F_k(w^k) \rightarrow w^k$ varias veces antes de realizar el promedio. Este enfoque genera el algoritmo *Federated Averaging* (o FedAvg). El modelo depende de tres parámetros clave: C, la fracción de clientes que participaran el cada iteración; E, el número de veces que cada cliente entrena sobre su conjunto de datos local en cada iteración antes de enviar la actualización al servidor de agregación; y B, el tamaño del *minibatch* local para las actualizaciones del cliente. Escribimos $B = \infty$ para indicar que el conjunto de datos local completo se trata como un solo *minibatch*. Si $B = \infty$ y $E = 1$ obtenemos exactamente FedSGD.

Para un cliente con n_k muestras, el número de actualizaciones locales por iteración viene dado por $u_k = E\frac{n_k}{B}$. El pseudocódigo completo se proporciona en el algoritmo 1.

Algorithm 1: Algorithm 1

Server executes:

Initialize w_0

for *each round* $t = 1, 2, \dots$ **do**

  $m \leftarrow \text{máx}(C, K, 1)$ $S_t \leftarrow$ (random set of m clients)

  for *each client* $k \in S_t$ ***in parallel*** **do**

    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

Function `ClientUpdate(k, w):`

  `// Run on client k`

  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)

  for *each local epoch i from 1 to* E **do**

    for *batch* $b \in \mathcal{B}$ **do**

      $w \leftarrow w - \eta \nabla l(w; b)$

  return w to server

2.3. Probabilistic Federated Neural Matching

Para este procedimiento se asume que hay disponibles datos en los clientes. Luego se procede entrenando modelos locales para cada cliente, en paralelo. A continuación, el algoritmo empareja los parámetros estimados de los modelos locales (grupos de vectores de peso en el caso de redes neuronales) a través los clientes para construir un modelo global. Este algoritmo de agregación se base en un procedimiento Beta-Bernoulli (BBD), que es un tipo de modelo bayesiano no paramétrico que regula la coincidencia entre los parámetros locales y los globales (ya existentes), y permite que se creen nuevos parámetros globales si los existentes no son compatibles con los locales. Como estos conceptos son matemáticas avanzadas, definamos qué es cada uno.

Definición 2.3.1. Sea $y, x \in \mathbb{C}$ con $\Re(x) > 0$ y $\Re(y) > 0$, entonces se define la función Beta o integral euleriana de primera especie como sigue:

$$\beta(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,dt \tag{2.1}$$

Mediante el cambio de variable $s = 1 - t$ es fácil ver que :

$$\beta(x, y) = \beta(y, x) \tag{2.2}$$

Otra propiedad relevante es que la función Beta tiene una relación muy estrecha con la Gamma:

Proposición 2.3.1.

$$\beta(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}, \quad x, y \in \mathbb{C} \tag{2.3}$$

Demostración. Consideremos el producto

$$\Gamma(x)\Gamma(y) = \int_0^\infty e^{-u}u^{x-1}\,du \int_0^\infty e^{-v}v^{y-1}\,dv$$

Tomando $u = p^2$, $v = q^2$ se tiene que

$$\begin{aligned}\Gamma(x)\Gamma(y) &= 4\int_0^\infty e^{-p^2}p^{2x-1}\,dp \int_0^\infty e^{-q^2}q^{2y-1}\,dq \\ &= 4\int_0^\infty \int_0^\infty e^{-(p^2+q^2)}p^{2x-1}q^{2y-1}\,dp\,dq\end{aligned}$$

Mediante un cambio a coordenadas polares, $x = r\cos\theta$, $y = r\sin\theta$.

$$\Gamma(x)\Gamma(y) = 4\int_0^\infty e^{-r^2} r^{2(x+y)-1}\,dr \int_0^{\frac{\pi}{2}} (\cos\theta)^{2x-1}(\sin\theta)^{2y-1}\,d\theta \tag{2.4}$$

Haciendo $r^2 = t$ en la primera integral:

$$\int_0^\infty e^{-r^2} r^{2(x+y)-1}\,dr = \frac{1}{2}\int_0^\infty e^{-t} t^{x-y-1}\,dt = \frac{1}{2}\Gamma(x+y)$$

Si en la segunda integral sustituimos $\cos\theta = s$:

$$\int_0^{\frac{\pi}{2}} (\cos\theta)^{2x-1}(\sin\theta)^{2y-1}\,d\theta = \frac{1}{2}\int_0^1 s^{x-1}(1-s)^{y-1}\,ds = \frac{1}{2}\beta(x,y)$$

Por lo que sustituyendo en 2.4 y despejando el valor de $\beta(x,y)$ se obtiene el resultado.

■

Se define ahora la distribución beta, esta es para una variable aleatoria continua que toma valores en el intervalo $[0,1]$, lo

Definición 2.3.2. Sea $a, b \in (0,\infty)$ y $x \in (0,1)$, entonces la distribución Beta es una distribución continua con función de densidad

$$f(x) = \frac{1}{\beta(a,b)} x^{a-1}(1-x)^{b-1}$$

Donde $\beta(a,b)$ es la función Beta con parámetros a, b. Y con función de distribución

$$I_x(a,b) = \frac{\int_0^x t^{a-1}(1-t)^{b-1}\,dt}{\beta(a,b)} \tag{2.5}$$

Nota 2.3.1. El numerador de 2.5 se conoce como la función beta incompleta y es una generalización de la función beta. Esta depende también del valor de x y se denota por $\beta(x;a,b)$. $I_x(a,b)$ se conoce como la función beta incompleta regularizada. Se pueden encontrar ejemplos y un desarrollo más detallado en [33] y [21].

Ya estamos en condiciones de definir qué es un procedimiento beta-Bernoulli:

Definición 2.3.3. Sea P una distribución beta con parámetros a y b. Luego, sea $\mathbb{X} = (X_1, X_2, \dots)$ una serie de variables aleatorias con la propiedad de que dado $P = p \in (0, 1)$, $\mathbb{X}$ es una serie condicionalmente independiente con

$$\mathbb{P}(X_i = 1 | P = p) = p, \quad i \in \mathbb{N}_+$$

Entonces $\mathbb{X}$ es un procedimiento beta-Bernoulli con parámetros a y b.

Nota 2.3.2. El concepto de independencia condicional hacer referencia a que dos sucesos son independientes si se da cierta condición.

En resumen, dado $P = p$, la serie $\mathbb{X}$ es una serie de experimentos de Bernoulli con probabilidad de éxito p. X_i es el resultado del experimento i, donde 1 denota éxito y 0 error.

Ejemplo 2.3.1.
Supongamos que seleccionamos una probabilidad de obtener caras en una moneda de acuerdo a la distribución beta con parámetros a y b. Lazando la moneda con esta probabilidad repetidamente obtendríamos

Definición 2.3.4. Sea ν una medida sobre el espacio $(\Omega, \mathcal{A})$ se dice que un conjunto $A \subset \Omega$ en $\mathcal{A}$ es atómico si verifica que $\nu(A) > 0$ y para cualquier subconjunto medible $B \subset A$ con $\nu(B) < \nu(A)$ el conjunto B tiene medida cero.

Ejemplo 2.3.2.
Por ejemplo, si consideramos el conjunto de números del 1 al 10 $X = \{1, \dots, 10\}$, con el conjunto de las partes de X como sigma sigma álgebra asociado y con medida ν definida como el cardinal del conjunto. Entonces los conjuntos unitarios de $\mathcal{P}(X)$ serían atómicos.

Definición 2.3.5. Sea $(X, \mathcal{A}, \nu)$ se dice finito si $\nu(X)$ es finito. Y se dice $\sigma -$ finito si X es la unión numeracle de conjuntos medibles con medida finita. Un conjunto en un espacio de medida tiene medida $\sigma -$ finita su es una unión numerable de conjuntos con medida finita.

Definición 2.3.6. Una medida $\sigma-$finita sobre un espacio medible $(X, \mathcal{A})$ se dice atómica o puramente atómica si cada conjunto medible de medida positiva contiene un conjunto atómico. Esto es quivalente a decir que existe una partición numerable de X formada por juntos atómicos

El ejemplo anterior nos vale también para este caso.

Definición 2.3.7. Una medida ν es no atómica si para cualquier conjunto medible A con $\nu(A) > 0$ existe un conjunto medible $B \| subset A$ tal que $\nu(A) > \nu(B) > 0$.

Una medida no atómica con al menos un valor positivo tiene un número infinito de valores diferentes. Si tomamos un conjunto A con $\nu(A) > 0$ se puede construir una serie decreciente de conjuntos medibles $A = A_1 \supset A_2 \supset A_3 \dots$ tal que $\nu(A) = \nu(A_1) > \nu(A_2) > \nu(A_3) > \dots > 0$. Esto puede no ser cierto para las medidas que tienen conjuntos atómicos, véase el ejemplo anterior.

Se tiene que las medidas no atómicas en realidad forman un continuo de valores. Y se puede demostrar (Waclaw Sierpinski) que si ν es una medida no atómica y A un conjunto medible con $\nu(A) > 0$, para cualquier número real b con $\nu(A) > b > 0$ existe entonces un subconjunto medible B de A tal que $\nu(B) = b$.

Definición 2.3.8. Sea H_0 una medida continua no atómica sobre el espació $(X, \mathcal{A})$, y $H_0(\Omega) = \gamma$ con gamma finito. Sea a, b dos escalares positivos. Se define un procedimiento H_k como

$$H_k(\theta) = \sum_{k=1}^{K} \pi_k \delta_{\theta_k}(\theta)$$

donde

$$\pi_k|_{a,b,\gamma} \overset{iid}{\sim} \text{BETA}(\frac{a\gamma}{k}, \frac{b(K-\gamma)}{K}); \quad \theta_k \overset{iid}{\sim} \frac{1}{\gamma} H_0$$

Cuando $K \to \infty$, $H_k \to H$ donde H sería un procedimiento beta, denota por $H \sim BP(a, b, H_0)$.

Ya estamos en condiciones de definir un procedimiento beta Bernoulli.

Definición 2.3.9. sea Q una medida distribuida por un procedimiento beta con parámetro γ_0 y medida base H. Esto es que $Q|_{\gamma_0,H} \sim BP(1, \gamma_0, H)$. Se sigue que Q es una medida discreta (no de probabilidad) con $Q = \sum_i q_i \delta_{\theta_i}$ formado por un conjunto infinito y numerable de pares (peso, átomo) $(q_i, \theta_i) \in [0, 1] \times \Omega$.

Los pesos se distribuyen segun un procedimiento *stick-breaking* [Stick-breaking Construction for the Indian Buffet Process]: $c_i \sim \beta(\gamma_0, 1)$, $q_i = \prod_{j=1}^{i} c_j$ y los conjuntos atómicos se extraen de la medida base H normalizada $\theta_i \sim H/H(\Omega)$ con dominio Ω.

En el texto Ω será R^D para algún D. Los subconjuntos atómicos de la medida Q se seleccionan usando un procedimiento de Bernoulli con medida base Q. Es decir, cada subconjunto $\mathcal{T}_j$, con $j = 1, \ldots, J$ están caracterizados por un procedimiento de Bernoulli con medida base Q, $\mathcal{T}_j|_Q \sim \text{BeP}(Q)$. Cada subconjunto $\mathcal{T}_j$ es también una medida discreta formado por pares $(b_{ji}, \theta_i) \in \{0, 1\} \times \Omega$, $\mathcal{T}_j := \sum_i b_{ij}\delta_{\theta_i}$ donde $b_{ij}|q_i \sim \text{Bernoulli}(q_i)$ para todo i es una variable aleatoria binaria que indica si el conjunto atómico θ_i pertenece al subconjunto $\mathcal{T}_j$. Así se dice que la colección de tales subconjuntos se distribuyen mediante un procese beta Bernoulli

2.3.1. ALGORITMO PROBABILISTIC FEDERATED NEURAL MATCHING

A continuación, describimos cómo se puede aplicar la maquinaria no paramétrica bayesiana al problema del aprendizaje federado con redes neuronales. Nuestro objetivo será identificar subconjuntos de pesos en cada uno de los modelos locales J que coinciden con pesos en otros modelos locales. Luego combinaremos apropiadamente los pesos emparejados para formar un modelo global.

El enfoque del aprendizaje federado se basa en el siguiente problema básico. Supongamos que hemos entrenado J perceptrones multiplica (MLP) con una capa oculta cada uno. Para el j—ésimo MLP con $j = 1, \ldots, J$, sea $V_j^{(0)} \in \mathbb{R}^{D \times L_j}$ y $\tilde{\nu}_j^{(0)} \in \mathbb{R}^{L_j}$ los pesos y los sesgos de la capa oculta, $V_j^{(1)} \in \mathbb{R}^{L_j \times K}$ y $\tilde{\nu}_j^{(1)} \in \mathbb{R}^K$ los pesos y los sesgos de la capa *softmax*. D es es la dimensión de los datos, L_i el número de neuronas en la capa oculta y K el número de clases. Se considera la siguiente arquitectura:

$$f_j(x) = \text{softmax}(\sigma(xV_j^{(0)} + \tilde{\nu}_j^{(0)})V_j^{(1)} + \tilde{\nu}_j^{(1)})$$

donde $\sigma(\cdot)$ es una función de regularización (sigmoide, ReLU, etc...). Dada la colección de pesos y sesgos $\{V_j^{(0)}, \tilde{\nu}_j^{(0)}, V_j^{(1)}, \tilde{\nu}_j^{(1)}\}_{j=1}^J$, queremos entrenar un modelo global con pesos y sesgos $\Theta^{(0)} \in \mathbb{R}^{D \times L}, \tilde{\theta}^{(0)} \in \mathbb{R}^L, \Theta^{(1)} \in \mathbb{R}^{L \times K}, \tilde{\theta}^{(1)} \in \mathbb{R}^K$ donde $L \ll \sum_{j=1}^J L_j$ es un número desconocido de unidades ocultas de la red global que se quiere inferir.

Se puede probar que el orden de las neuronas de la capa oculta de un MLP es invariante a cualquier permutación. Por lo que toda reordenación de las columnas de $V_j^{(0)}$, sesgos $\tilde{\nu}_j^{(0)}$ y filas de $V_j^{(1)}$ no afectará al resultado de $f_j(x)$. Por lo que estos pueden ser vistos como una colección no ordenada de vectores $V_j^{(0)} = \{v_{jl}^{(0)} \in \mathbb{R}^D\}_{l=1}^{L_j}$, $V_j^{(1)} = \{v_{jl}^{(1)} \in \mathbb{R}^{L_j}\}_{l=1}^K$ y escalares $\tilde{\nu}_j^{(0)} = \{\tilde{\nu}_{jl}^{(0)} \in \mathrm{mathddR}\}_{l=1}^{L_j}$.

Las capas ocultas en las redes neuronales pueden veer como extractores de características. Esto se debe al hecho de que la última capa de un clasificador de redes neuronales realiza una regresión softmax. Generalmente las redes neuronales obtienen mejores resultados que la regresión softmax básica, por lo que deben ser capaces de extraer de los datos sin procesar muestras de características de alta calidad.

En la construcción de la arquitectura, cada neurona oculta en el j—ésimo MLP representa una nueva característica $\tilde{x}\,(v_{jl}^{(0)}, \tilde{\boldsymbol{\nu}}_{jl}^{(0)}) = \sigma(\langle x, v_{jl}^{(0)}\rangle + \tilde{\boldsymbol{\nu}}_{jl}^{(0)})$, como segunda observación, notar que cada $(v_{jl}^{(0)}, \tilde{\boldsymbol{\nu}}_{jl}^{(0)})$ parametriza.

Dado que los J modelos MLP se entrenan con el mismo tipo de datos (aunque no necesariamente homogéneos), se asume que comparten al menos algunos extractores de características con el mismo propósito. Sin embargo, debido a la invariancia a permutaciones discutido anteriormente, es poco probable que un extractor de características con índice l del j—ésimo MLP corresponda a un extractor de características con el mismo índice de un MLP diferente. Para construir un conjunto de extractores de características globales (neuronas) $\{\Theta_i^{(0)} \in \mathbb{R}^D, \tilde{\boldsymbol{\Theta}}_i^{(0)} \in \mathbb{R}\}_{i=1}^L$ se debe modelar el proceso de agrupación y combinación de extractores de características de los J MLPs.

El desarrollo teórico que se presenta sobre los algoritmos de *Neural Matching* es largo y presta de conocimientos profundos de matemática compleja. Como el objetivo de este trabajo no es profundizar en las definiciones y resultados necesarios para llegar a la construcción rigurosa de los algoritmos, sino los algoritmos en si, en [19], [20], [21], [22] y [24] podemos encontrar toda esta teoría al detalle.

Se presenta la idea principal del *Neural Matching*, que sería el procedimiento aplicado a una sola capa. Partimos de un modelo basado en un procedimiento beta Bernoulli de con base en los pesos de un modelo MLP. En el algoritmo se asume el siguiente proceso generativo. Primero preparamos una colección de conjuntos atómicos globales (neuronas de la capa oculta) de un procedimiento Beta previo con medida base H y parámetro de masas γ_0, $Q = \sum_i q_i \delta_{\theta_i}$. Se toma $H = \mathcal{N}(\mu_0, \Sigma_0)$ como la medida base con $\mu_0 \in \mathbb{R}^{D+1+K}$ y diagonal Σ_0. Cada $\theta_i \in \mathbb{R}^{D+1+K}$ es un vector formado por $[\theta_i^{(0)} \in \mathbb{R}^D, \tilde{\boldsymbol{\Theta}}_i^{(0)} \in \mathbb{R}, \theta_i^{(1)} \in \mathbb{R}^K]$ formado a partir de los pares sesgo-pesos de la extracción de características correspondientes de la regresión softmax. En lo que sigue se usará el término *batch* para referirnos a una partición de los datos.

Luego para cada $j = 1, \ldots, J$ se toma un subconjunto de conjuntos atómicos globales para el batch j mediante un procedimiento de Bernoulli:

$$\mathcal{T}_j := \sum_i b_{ji}\delta_{\theta_i}, \quad \text{donde} \quad b_{ij}| : q_i \sim \text{Bern}(q_i)\forall i. \tag{2.6}$$

$\mathcal{T}_j$ está respaldado por los conjuntos atómicos de la forma $\{\theta_i : b_{ij} = 1, i = 1, 2, \dots\}$, que identifican las neuronas por lote j. Por último, supongamos que los conjuntos atómicos locales observados son medidas con cierto ruido relacionados directamente con los conjuntos atómicos globales.

$$\nu_{jl}|\mathcal{T}_j \sim \mathcal{N}(\mathcal{T}_{jl}, \Sigma_j) \quad \text{para} \quad l = 1, \dots, L_j;\ L_j := \text{card}(\mathcal{T}_j), \tag{2.7}$$

con $\nu_{jl} = [\nu_{jl}^{(0)}, \tilde{\boldsymbol{\nu}}_{jl}^{(0)}, \nu_{jl}^{(1)}]$ los pesos, sesgos y los pesos de la regresión softmax correspondiente con la l—-ésima neurona del j—ésimo MLP entrenado con L_i neuronas en los datos del batch j.

Bajo este modelo, el número clave que se infiere es la colección de variables aleatorias que coinciden con los los conjuntos atómicos observados (neuronas) en cualquier otro batch con los conjuntos atómicos globales. Se denota la colección de estas variables aleatorias como $\{B^j\}_{j=1}^J$, donde $B_{i,l}^j = 1$ implica que $\mathcal{T}_{jl} = \theta_i$, por lo que existe una correspondencia biyectiva entre $\{b_{ji}\}_{i=1}^\infty$ y B^j.

Estimación máxima a posteriori (MAP). Ahora veamos un algoritmo para la estimación MAP de los conjuntos atómicos globales para el modelo presentado anteriormente. La función objetivo a maximizar es la posteriori de $\{theta_i\}_{i=1}^\infty$ y $\{B^j\}_{j=1}^J$:

$$\arg_{\{\theta_i\},} \max_{\{B^j\}} P(\{\theta_i\}, \{B^j\}|\{\nu_{jl}\}) \alpha P(\{\nu_{jl}\}|\{\theta_i\}, \{B^j\}) P(\{B^j\}) P(\{\theta_i\}) \tag{2.8}$$

Notar que la siguiente proposición se sigue fácilmente de la conjugación gaussiana-gaussiana:

Proposición 2.3.2. Dado $\{B^j\}$, la estimación MAP de $\{\theta_i\}$ viene dado por

$$\hat{\Theta}_i = \frac{\mu_0/\sigma_0^2 + \Sigma_{j,l} B_{i,l}^j \nu_{jl}/\sigma_j^2}{1/\sigma_0^2 + \Sigma_{j,l} B_{i,l}^j/\sigma_j^2}, \quad \text{con} \quad i = 1, \dots, L, \tag{2.9}$$

Donde por simplicidad se asume que $\Sigma_0 = I\sigma_0^2$ ay $\Sigma_j = I\sigma_j^2$.

Proposición 2.3.3. El coste (negativo) requerido para encontrar B^j es:

$$-C^{j}_{i,l} = \begin{cases} \frac{||\frac{\mu_0}{\sigma_0^2}+\frac{\nu_{jl}}{\sigma_j^2}+\sum_{-j,l} B^{j}_{i,l}\frac{\nu_{jl}}{\sigma_j^2}||^2}{\frac{1}{\sigma_0^2}+\frac{1}{\sigma_j^2}+\sum_{-j,l} B^{j}_{i,l}/\sigma_j^2} - \frac{||\frac{\mu_0}{\sigma_0^2}+\sum_{-j,l} B^{j}_{i,l}\frac{\nu_{jl}}{\sigma_j^2}||^2}{\frac{1}{\sigma_0^2}+\sum_{-j,l} B^{j}_{i,l}/\sigma_j^2} + 2\log\frac{m_i^{-j}}{J-m_i^{-j}}, & i \leqslant L-j \\ \frac{||\frac{\mu_0}{\sigma_0^2}+\frac{\nu_{jl}}{\sigma_j^2}||^2}{\frac{1}{\sigma_0^2}+\frac{1}{\sigma_j^2}} - \frac{||\frac{\mu_0}{\sigma_0^2}||^2}{\frac{1}{\sigma_0^2}} - 2\log\frac{i-L-j}{\gamma_0/J}, & L_{-j} < i \leqslant L_{-j}+L_j \end{cases} \tag{2.10}$$

Donde $L_{-j} = \text{máx}\{i : B^{-j}_{i,l} = 1\}$ denota el número activo de pesos globales fuera del grupo j.

Luego aplicando eñ algoritmo húngaro para encontrar el minimizador de $\Sigma_i\Sigma_l B^{j}_{i,l}C^{j}_{i,l}$ y se obtienen las asignaciones concordantes entre las neuronas.

Se proporciona a continuación el pseudocódigo para el algoritmo *Single Layer Neural Matching*:

Algorithm 2: Single Layer Neural Matching

Server executes:

1 Se toman los pesos y los sesgos de los J batches y de ν_{jl}.

2 De la matriz de asignación de costes 2.10

3 Se calculan las asignaciones concordantes Bj utilizando el algoritmo húngaro.

4 Enumerar todas las neuronas globales únicas resultantes y utilizar 2.9 para inferir los vectores de peso global asociados de todas las instancias de las neuronas globales en los J batches.

5 Concatenar las neuronas globales con los pesos y sesgos inferidos para formar la nueva capa oculta global.

El algoritmo multicapa basado en *Neural Matching* es el *Multilayer PFNM*.

Algorithm 3: Multilayer PFNM

Server executes:

1 $L^{C+1} \leftarrow$ number of outputs

// Iterar a través de capas de arriba hacia abajo

2 **for** *layers* $c = C, C-1, \ldots, 2$ **do**

3 Tomar todas las capas ocultas c de los J batches y de ν^c_{jl}.

4 Llamar al algoritmo Single Layer Neural Matching con dimensión de salida L^{c+1} y dimensión de entrada 0 ya que no se usan los pesos que están relacionados con las capas inferiores.

5 Formar la capa de neuronas global c a partir de la salida de la coincidencia de una sola capa.

6 $L^c \leftarrow \text{card}(\cup_{j=1}^{J} \mathcal{T}^c_j)$

// Empareja la capa anterior usando pesos que se conecten tanto a la entrada como a la capa superior

7 Llamar al algoritmo Single Layer Neural Matching con dimensión de salida L^2 y dimensión de entrada igual al número de datos de entrada.

8 Devolver las asignaciones globales y formar un modelo global de capas múltiples.

2.4. ALGORTIMO DE KRUM

El *Federated Learning* se encuentra muy relacionado con aprendizaje distribuido. Pero tiene un enfoque mucho más descentralizado y general. Esto añade más complejidad a los problemas que se presentaban en el aprendizaje distribuido y además crea nuevos desafíos y retos que a día de hoy siguen abiertos. Uno de los problemas principales es el nivel de descentralización [?], [30] que se presentan en algunas situaciones como hemos visto en algunos de los ejemplos de aplicación. Las máquinas en la que generalmente se realizan los entrenamientos son más impredecibles y pueden estar más expuestas a ataques externos. Debido a esta impredecibilidad y comportamiento errático (a veces incluso adverso), se ha modelado generalmente como fallo bizantino [14], lo que significa que algunas máquinas pueden comportarse arbitrariamente y enviar cualquier mensaje

al agregador que mantiene y actualiza una estimación del vector de pesos del modelo general. Los fallos bizantinos pueden provocar una gran degradación del rendimiento del aprendizaje. Es bien sabido que los modelos estándar basados en la agregación de los modelos pueden ser sesgados simplemente por una la existencia de una sola máquina con fallos bizantinos. Incluso cuando los modelos obtenidos de las máquinas bizantinas toman valores moderados (por lo tanto son difíciles de detectar) y cuando el número de máquinas es pequeño la pérdida de rendimiento puede ser significativa (Sección 7, [25]).

En este caso se estudia la resistencia a fallos bizantinos de las implementaciones del SGD, se parte de que tenemos un número n de clientes de los cuales f pueden tener un comportamiento bizantino, eso quiere decir que se comportan aleatoriamente.

Mencionar que durante toda la sección la norma considerada es la euclídea.

Supongamos que partimos de la ronda t en un sistema síncrono, los pesos del servidor de agregación $x_t \in \mathbb{R}^d$ son enviados a todos los clientes. Cada cliente i realiza una estimación con sus datos locales $V_p^t = G(x_t, D_i^t)$ del gradiente $\nabla Q(x_t)$ de la función de coste Q, donde D_i^t es una variable aleatoria, como la muestra de un conjunto de datos (o un mini-batch de muestras). Un cliente bizantino b genera un vector V_b^t que puede desviarse arbitrariamente del vector que debería enviar si fuera un cliente normal, según un algoritmo fijo. Dado que estamos en un sistema síncrono, si el servidor de agregación no recibe V_b^t para un cliente bizantino dado b. Entonces por defecto el sistema le dará el valor $V_b^t = 0$.

El servidor de agregación calcularé entonces $F(V_1^t, \ldots, V_n^t)$, donde F es una función de agregación. Por último se actualizaría el estado de los pesos del modelo usando la siguiente ecuación

$$x_{t+1} = x_t - \gamma_t \dot{F}(V_1^t, \ldots, V_n^t)$$

En esta sección se asume que los clientes no bizantinos calculan estimaciones insesgadas del gradiente $\nabla Q(x_t)$. Mas precisamente, en cada iteración t, los vectores V_i^t propuestos por los clientes no bizantinos son vectores aleatorios independientes y están idénticamente distribuidos, $V_i^t \sim G(x_t, D_i^t)$ con $(E)_{D_i^t} G(c_t, D_i^t) = \nabla Q(x_t)$. Esto se deduce de que cada muestra de datos usada para calcular el gradiente se extrae de forma uniforma e independiente. Los clientes bizantinos tiene total conocimiento del sistema, tanto el algoritmo de agregación F, como los vectores propuestos por los demás clientes. Además puede existir colaboración entre sí.

El 2.4.1 muestra como ninguna combinación lineal de los vectores de entrada es

tolerante a ningún nodo bizantino. En particular, el promediado de los pesos no sería tolerante a la existencia de uno o varios nodos bizantino.

Lema 2.4.1. Sea F_{lin} un algoritmo de agregación de la forma $F_{lin}((V_1^t, \dots, V_n^t)) = \sum_{i=1}^{n} \lambda_i V_i$ con $\lambda_i \neq 0 \forall i = 1, \dots, n$. Sea un vector $U \in \mathbb{R}$. Entonces un nodo bizantino puede hacer que siempre F tome el valor U. En particular, un solo nodo bizantino puede impedir la convergencia del algoritmo.

Demostración. La prueba is inmediata si el resultado dado por el nodo bizantino $V_n = \frac{1}{\lambda_n} U - \sum_{i=1}^{n-1} \frac{\lambda_i}{\lambda_n} V_i$ implica que $F = U$. ■

La siguiente definición otorga dos las condiciones necesaria para que una función de agregación tenga la propiedad de ser tolerante bizantina.

Definición 2.4.1. Sea $0 \leqslant \alpha \leqslant \pi/2$ y un entero f con $0 \leqslant f \leqslant n$. Sea $V_1, \dots, V_n$ vectores aleatorios iid de $\mathbb{R}^d$ con $V_i \sim G$ donde $\mathbb{E}G = g$. Sea $B_1, \dots, B_f$ in$\mathbb{R}^d$ vectores aleatorios cuales quiera (posiblemente dependientes de los V_i). Una función de agregación F se dice que es (α, f)—tolerante bizantina si, para cualquier $1 \leqslant j_1 \leqslant \cdots \leqslant j_f \leqslant n$ el vector

$$F = F(V_1, \dots, \underbrace{B_1}_{j_1}, \dots, \underbrace{B_f}_{j_f}, \dots, V_n)$$

satisface las siguientes condiciones:

1. $\langle \mathbb{E}F, g \rangle \geqslant (1 - \sin \alpha)\|g\|^2 > 0$.

2. Para $r = 2, 3, 4$, $\mathbb{E}\|F\|^r$ está acotada superiormente por una combinación lineal en términos de $\mathbb{E}\|G\|^{r_1} \dots \mathbb{E}\|G\|^{r_{n-1}}$ con $r_1 + \cdots + r_{n-1} = r$.

La condición 1 muestra que si el valor esperado de F se encuentra dentro de la bola de centro g y radio r, el producto escalar entre el valor esperado de F y g esta acotado inferiormente por $(1 \sin \alpha)\|g\|^2$ siendo $\alpha = r/\|g\|$.

La condición 2 es más técnica y obliga a los momentos de la función F a estar controlados por los momentos del estimador del gradiente G.

Pasamos ahora a definir la función de Krum, el agregador que protagoniza esta sección, se muestra además que satisface las condiciones para ser (α, f)—tolerante bizantina.

Definición 2.4.2. Para cada $i \neq j$, denotamos que $i \to j$ si V_j pertenece a los $n - f - 2$ vectores más cercanos a V_i. Entonces, se define para cada nodo i el *score* $s(i) = \sum_{i \to j} \|V_i - V_j\|^2$ donde la suma recorre los $n - f - 2$ vectores más cercanos a V_i. Finalmente la función de Krum $KR(V_1, \ldots, V_n) = V_{i_*}$ donde i_* se refiere al nodo que minimiza la función *score*, $s(i_*) \leqslant s(i)$ para todo i

Nota 2.4.1. Si dos o más nodos tienen la *score* mínima, se tomará el que tiene el índice más pequeño.

Lema 2.4.2. La complejidad temporal esperada de la función Krum $KR(V_1, \ldots, V_n)$, donde $V_1, \ldots, V_n$ de $\mathbb{R}^d$, es de $O(n^2 \cdot d)$.

La proposición **??** muestra que, si $2f + 2 < n$ y el estimador del gradiente es lo suficientemente preciso (su desviación típica es relativamente pequeña comparada con la norma del gradiente), entonces la función Krum es (α, f)—tolerante bizantina, donde el ángulo α depende de la proporción de la desviación sobre el gradiente.

Proposición 2.4.1. Sea $V_1, \ldots, V_n \in \mathbb{R}^d$ vectores aleatorios iid cualesquiera donde $V_i \sim G$ con $\mathbb{E}G = g$ y $\mathbb{E}\|G - g\|^2 = d\sigma^2$. Sea $B_1, \ldots, B_f$ f vectores aleatorios (posiblemente dependientes de los V_i). Si $2f + 2 < n$ y $\eta(n, f)\sqrt{d} \cdot \sigma < \|g\|$, donde

$$\eta(n, f) := \sqrt{2(n - f + \frac{f \cdot (n - f - 2) + f^2 \cdot (n - f - 1)}{n - 2f - 2})} = \begin{cases} O(n) & \text{si } f = O(n) \\ O(\sqrt{n}) & \text{si } f = O(1) \end{cases},$$

Entonces la función de Krum KR es (α, f)—tolerante bizantina y se tiene que $\alpha \in [0, \pi/2)$ está definido por:

$$\sin \alpha = \frac{\eta(n, f) \cdot \sqrt{d} \cdot \sigma}{\|g\|}.$$

La condición sobre la norma del gradiente, $\eta(n, f) \cdot \sqrt{d} \cdot \sigma < \|g\|$, puede satisfacerse hasta cierto punto haciendo que los nodos calcules las estimación del gradiente en mini-batches [31]. En efecto el promedio de las estimaciones del gradiente sobre un mini-batch divide la desviación típica σ por la raíz cuadrada del tamaño del mini-batch. Podemos encontrar la prueba de esto y de los resultados anteriores en [27], además de un análisis detallado de la convergencia y de resultados sobre experimentaciones mas minuciosas sobre este método.

2.5. COORDINATE WISE MEDIAN

En esta sección se presenta otro modelo tolerante a fallos bizantinos. Nos basamos en los estudios realizados por [25] donde trata de responder las sigueintes preguntas ¿cuál es el mejor rendimiento estadístico que se puede alcanzar con un modelo siendo tolerante bizantino? y ¿qué algoritmos logran este rendimiento?

Para formalizar esta cuestión, consideramos un problema de minimización empírica del riesgo (ERM). En este caso, una muestra de nm focos de datos independiente siguiendo una determinada distribución y que se distribuyen uniformemente entre m máquinas, de las cuales αm son bizantinas. El objetivo es aprender un modelo minimizando alguna función de loss. En este contexto, se espera que el error de aprendizaje de el parámetro, medido según una métrica apropiada, disminuya cuando la cantidad de datos nm aumente y el número de máquinas bizantinas αm sea menor. De hecho, se demuestra en [27] que, al menos para los problemas fuertemente convexos, ningún algoritmo puede lograr un error inferior a

$$\tilde{\Omega}\left(\frac{\alpha}{\sqrt{n}} + \frac{1}{\sqrt{nm}}\right) = \tilde{\Omega}\left(\frac{1}{\sqrt{n}}\left(\alpha + \frac{1}{\sqrt{m}}\right)\right) \quad ,$$

independientemente de los costes de comunicación (ver la sección 6 del artículo mencionado anteriormente). Intuitivamente, la tasa de error anterior es la tasa óptima a la que se debe aspirar, ya que $\frac{1}{\sqrt{n}}$ es la desviación típica efectiva de cada máquina con n datos, α es el efecto de sesgo de las máquinas bizantinas, y $\frac{1}{\sqrt{m}}$ es el efecto del promediado de las m máquinas normales.

Cuando no existen máquinas bizantinas o hay pocas, vemos que el valor $\frac{1}{\sqrt{nm}}$ con el número total de datos; cuando algunas máquinas son bizantinas, su influencia sigue estando acotada, y además es proporcional a α. Si se garantiza que un algoritmo alcanza este límite, tenemos la seguridad de que no sacrificamos la calidad del aprendizaje al tratar de evitar de los fallos bizantinos, sino que pagamos un precio que es inevitable, pero por lo demás conseguimos la mejor precisión estadística posible en presencia de fallos bizantinos.

2.5.1. CONSTRUCCIÓN DEL PROBLEMA

Supongamos que se tiene una muestra de datos según una distribución $\mathcal{D}$ sobre una espacio muestral $\mathcal{Z}$. Sea $f((w; z))$ la función *loss* de un vector de parámetros (pesos) $w \in \mathcal{W} \subset \mathbb{R}^d$ asociado a z, donde $\mathcal{W}$ es el espacio de parámetros, y $F(w) := \mathbb{E}_{z\sim\mathcal{D}}[f(w; z)]$

es ña correspondiente función *loss* de la población. El objetivo es entrenar un modelo que minimice la función *loss* de la población:

$$w^* = \arg\min_{w \in \mathcal{W}} F(w). \tag{2.11}$$

1. Se parte de que $\mathcal{W}$ es un conjunto convexo y compacto con diametro D. Es decir $\|w - w'\|_2 \leqslant D$, $\forall w, w' \in \mathcal{W}$.

2. El problema se modela mediante computación distribuida con una máquina de agregación y m nodos. Cada nodo almacena n datos, cada uno de los cuales se obtiene independientemente de la distribución $\mathcal{D}$.

3. Se denota por $z^{i,j}$ el j—ésimo dato en el i—ésimo nodo, y

$$F_i(w) := \frac{1}{n} \sum_{j=1}^{n} f(w; z^{i,j}),$$

   la función de riesgo empírica para el i—ésimo nodo.

4. Se supone también que se tiene una fracción α de m nodos bizantinos, y el resto $1 - \alpha$ son normales. Con la notación $[N] := \{1, 2, \ldots, N\}$, indexamos el conjunto de nodos por $[m]$, y se denota el conjunto de máquinas bizantinas por $\mathcal{B} \subset [m]$ (luego $|\mathcal{B}| = \alpha m$).

Nota 2.5.1. El agregador se comunica con los demás nodos usando un protocolo predefinido. Las máquinas bizantinas no tienen por qué obedecer este protocolo y pueden enviar mensajes arbitrarios al agregador; en particular, pueden tener un conocimiento completo del sistema y de los algoritmos de aprendizaje (como se menciona anteriormente).

2.5.2. Algoritmos

Se introduce las operaciones de *coordinate-wise median* y *trimmed mean operations*, que sirven como base para construir nuestro algoritmo.

Definición 2.5.1. Sea $x^i \in \mathbb{R}^d$, $i \in [m]$. Entonces se define el procedimiento de *coordinate-wise median* como un vector $g := \text{med}\{x^i : i \in [m]\}$ de forma que su k—ésima coordenada $g_k = \text{med}\{x_k^i : i \in [m]\}$ para cada $k \in [d]$, y donde la función 'med' respresenta la mediana usual unidimensional.

Esto iría construyendo el vector fijando cada término k y tomando mediana de cada gradiente.

Definición 2.5.2. Sea $\beta \in [0, \frac{1}{2})$ y $x^i \in \mathbb{R}^d$, $i \in [m]$. Entonces se define el procedimeinto de *Coordinate-wise* $\beta-$*trimmed mean* como un vector $g := \text{trmean}_\beta\{x^i : i \in [m]\}$ de forma que su $k-$ésima coordenada $g_k = \frac{1}{(1-2\beta)m} \sum_{x \in U_k}$ para cada $k \in [d]$, y donde U_k es un subconjunto de $\{x_k^!, \ldots, x_k^m\}$ que se obtiene eliminando la mayor y menor fracción β de sus elementos.

Basándonos en estas dos definiciones podemos construir dos algoritmos de agregación en función de la métrica que elijamos.

Algorithm 4: Robust Distributed Gradient Descent

Se requiere: Inicializar el vector de parámetros $w^0 \in \mathcal{W}$ y los parámetros para los algoritmos, β (para la opción II), η y T.

for $i = 0, 1, 2, \ldots, T-1$ **do**

Agregador: Envía w^t a todos los nodos.

for ***para todo*** $i \in [m]$ ***en paralelo*** **do**

Nodo i: Calcular los gradientes locales

$$g^i(w^t) \leftarrow \begin{cases} \nabla F_i(w^t) & \text{En las maquinas normales,} \\ * & \text{En las máquinas Bizantinas} \end{cases},$$

enviar $g^i(w^t)$ al agregador

Agregador: Se calcular el gradiente agregado

$$g(w^t) \leftarrow \begin{cases} \text{med}\{g^i(w^t) : i \in [m]\} & \text{Opción I,} \\ \text{trmean}_\beta\{g^i(w^t) : i \in [m]\} & \text{Opción II} \end{cases},$$

Actualizar los parámetros $w^{t+1} \leftarrow \prod_{\mathcal{W}}(w^t - \eta g(w^t))$.

2.6. ZENO

Zeno es otro algoritmo para agregación de modelos tolerante a comportamientos bizantinos, es un algoritmo presentado en [Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance]. La construcción del problema es similar a la anterior.

La idea es tratar cada estimador del gradiente como sospechoso. Se calcula una puntuación utilizando un oráculo estocástico de orden cero. Esta puntuación indica cómo de fiable es el nodo dado en una iteración. A continuación, tomamos la media entre los candidatos con las puntuaciones más altas. Esto permite tolerar un gran número de resultados incorrectos dados por los nodos bizantinos. Se puede demostrar que la convergencia es tan rápida como la del SGD sin nodos bizantinos [Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance]. Además, la varianza disminuye a medida que aumenta el número de nodos no defectuosos.

2.6.1. Construcción del problema

Se considera el problema de optimización:

$$\min_{x \in \mathbb{R}^d} F(x),$$

donde $F(x) = \mathbb{E}_{x \sim \mathcal{D}}[f(x; z)]$ donde z es una muestra de una distribución desconocida $\mathcal{D}$.

Se supone que existe x^* que minimiza a $F(x)$ y además que existe un agregador y m nodos.

En cada iteración los nodos con una muestra de n datos iid que siguen la distribución desconocida $\mathcal{D}$, se calcula el gradiente de de la función *loss* local $F_i(x) = \frac{1}{n} \sum_{j=1}^{n} f(x; z^{i,j})$, $\forall i \in [m]$, donde $z_{i,j}$ es el j—ésimo elemento de la muestra del nodo i. El agregador recibirá todos los gradientes calculados en los distintos nodos, creará un modelo global agregando los gradientes los locales, y actualizará el gradiente del modelo global creando una nueva iteración:

$$x^{t+1} = x^* t - \gamma^t \, \mathtt{Aggr}(\{g_i(x^t) : i \in [m]\}),$$

donde $\mathtt{Aggr}$ es una regla de agregación (por ejemplo la media) y

$$g_i(x^t) = \begin{cases} * & \text{para un nodo bizantino,} \\ \nabla F_i(x^t) & \text{en otro caso} \end{cases} \tag{2.12}$$

$*$ representa un valor arbitrario.

2.6.2. Algoritmo

Como se ha mencionado antes, este algoritmo requiere de un oráculo estocástico de orden zero, esté dará una puntuación a cada gradiente obtenido de los nodos. Luego se agregan las mejoras puntuaciones, la puntuación indica, a grandes rasgos , el grado de confianza de cada candidato.

Definición 2.6.1. Sea $f_r(x) = \frac{1}{n_r}\sum_{i=1}^{n_r} f(x; z_i)$, donde los Z_i son iid y sginden una distribución $\mathcal{D}$, y n_r el tamaño del batch de $f_r(\cdot)$. $\mathbb{E}][f_r(x)] = F(x)$. Para cada actualización (estimación del gradiente) u, basada en el parámetro x, factor de aprendizaje γ y peso constante ρ, se define su puntuación estocástica descendiente como sigue:

$$\text{Score}_{\gamma,\rho}(u, x) = f_r(x) - f_r(x - \gamma u) - \rho\|u\|^2 .$$

El valor definido en 2.6.1 consta de dos partes:

1. El valor descendiente estimado de la función *loss*.
2. La norma de la actualización.

La puntuación aumenta cuando, $f_r(x) - f_r(x - \gamma u)$ crece y disminuye cuando la norma de u al cuadrado crece. Intuitivamente un valor descendente más grande nos sugiere una convergencia más rápida y un valor más pequeño indica un menor cambio. Incluso si un gradiente viene de un nodo bizantino, un cambio más pequeño lo hace menos dañino y más fácil de eliminar por lo gradientes obtenidos de nodos normales.

Usando la puntuación definida anteriormente. Se define la siguiente regla de agregación. Sin perdida de generalidad se prescinde del término t índice de agregación.

Definición 2.6.2. **??** Supongamos que entre los estimadores del gradiente $\{\tilde{v}_i : i \in [m]\}$, q elementos son bizantinos y x el valor actual de los parámetros. Ordenamos la secuencia usando la puntuación estocástica descendiente definida en 2.6.1, obteniendo $\{\tilde{v}_{(i)} : i \in [m]\}$, donde

$$\text{Score}_{\gamma,\rho}(\tilde{v}_{(1)}, x) \geqslant \cdots \geqslant \text{Score}_{\gamma,\rho}(\tilde{v}_{(m)}, x) .$$

$\tilde{v}_{(i)}$ es el vector con la i—ésima puntuación más alta. La regla de agregación propuesta, Zeno, agrega los estimadores del gradiente tomando la media de los primeros $m - b$ de los elementos con las puntuaciones más altas. Con $m > b \geqslant q$:

$$\text{Zeno}_b(\{\tilde{v}_i : i \in [m]\}) = \frac{1}{m-b} \sum_{i=1}^{m-b} \tilde{v}_{(i)}$$

Algorithm 5: Zeno

Server executes:

Entrada: Definimos los parámetros rho de 2.6.1 y b de ??.
$x^0 \leftarrow \text{rand}()$ (Inicialización).
for $t = 1, \ldots, T$ **do**
  El agregador envía x^{t-1} a todos los nodos.
  El agregador espera a recibir todos los $\{\tilde{v}_i^t : i \in [m]\}$
  Se toman las muestras para la evaluación de las puntuación estocástica descendiente $f_r^t(\cdot)$ como se expone en 2.6.1.
  Se calcula $\bar{\tilde{v}}^t = \text{Zeno}(\{\tilde{v}_i^t : i \in [m]\})$ como se expone en ??.
  Se actualizan los parámetros $x^t \leftarrow x^{t-1} - \gamma^t \bar{\tilde{v}}^t$.

Node executes $i = 1, \ldots, m$:

for $t = 1, \ldots, T$ **do**
  Se recibe x^{t-1} desde el agregador.
  Se obtienen las muestras. Y se calcula y envían el gradiente $v_i^t = \nabla F_i^t(x^{t-1})$ al agregador.

2.7. Statical Parameter Aggregation via Heterogeneous Matching (SPAHM)

En [Statistical Model Aggregation via Parameter Matching] encontramos todo el desarrollo teórico matemático para la construcción del algoritmo y el estudio de sus propiedades, dado el nivel de complejidad teórica que entraña este algoritmo en particular y que el objetivo del presente trabajo no consiste en mostrar al lector los fundamentos para justificar el funcionamiento del algoritmo, sino el algoritmo en sí. Solamente se proporcionará el pseudocódigo del algoritmo además de algunos comentarios y observaciones.

SPAHM es otro algoritmo basado en la coincidencia de parámetros. Al igual que una de las secciones anteriores, se desarrolla un modelo bayesiano no paramétrico que combina los modelos entrenados en los diferentes nodos. Tomando como base inferencia bayesiana no paramétrica (BNP), el modelo presentado en esta sección trate los parámetros de los modelos locales como experimentos sesgados de ciertos parámetros globales subyacentes (pudiendo existir infinitos).

El experimento se modela mediando un procedimiento beta bernoulli (BBP). La fusión de modelo se basa en la probabilidad a posteriori sobre el modelo. Regido por el BBP, el modelo busca que los parámetros locales coincidan con los parámetros globales existentes y mantenerlo o que se creen unos nuevos.

Algorithm 6: SPAHM

Entrada: Los parámetros de entrada a la función son ν_{jl}, el número de itereciones M, e supuestos valores para hiperparámetros iniciales $\hat{\tau}$, $\hat{n}_0$.
while *no haya convergencia* **do**
  for M *iteraciones* **do**
    $j \sim \text{Unif}(\{1, \ldots, J\})$.
    Para la matriz de costes C^j. Se usa el algoritmo húngaro para optimizar las asignaciones B^j, manteniendo el resto de asignaciones fijas.
  Dado B, optimizar la función de optimización de hiperparámetros para $\hat{\tau}$, $\hat{n}_0$ con el fin de mejorar sus valores.
Resultado Parámetros coincidentes B, estimaciones de los conjuntos atómicos globales θ_i.

1. ν_{jl} es el resultado de los parámetros locales del parámetro global θ_i y $B^j_{il} = 1$ denota que ν_{jl} se corresponde con θ_i. $B^j_{il} = 0$ indica lo contrario.
2. $B = \{B^j_{il}\}$ son las variables de asignación.

2.8. Fed+

Fed+ unifica varios algoritmos de agregación. La principal ventaja es que se ajuste mejor a las características de los problemas reales. Como hemos mencionado anteriormente el federated learning plantea un contexto de aprendizaje diferente. Esto incluye problemas

y retos extra que debemos tener en cuenta, por ejemplo que los datos con sean iid entre los clientes, pueden existir clientes cuyos datos sean atípicos, y los algoritmos pueden tener malos resultados cuando los datos entre los clientes son demasiado heterogéneos. La personalización a los diferentes clientes de los modelos federados presenta un mayor beneficio y permite una mayor precisión de los datos en cada cliente.

Fed+ está diseñado para abordar estas cuestiones y aumentar la robustez frente a los valores atípicos y modelos poco desarrollados. Fed+ se basa en una formulación del problema que permite al servidor de agregación aplicar reglas robustas de agregación de modelos. Es importante mencionar que Fed+ no necesita que todos los modelos converjan a un único punto central. Formalicemos la formulación del problema de federated learning en la que se basa Fed+:

2.8.1. Construcción del problema

Sea K el número de nodos con función local de *loss* $f_k : \mathbb{R}^d \longrightarrow \mathbb{R}$, $k = 1, 2, \ldots, K$. A diferencia de FedAvg que define un problema de minimización central promediando las K *loss* locales, en Fed+ se plantea una penalización y se propone el siguiente problema de mínimos:

$$\min_{w \in \mathbb{R}^{d \times K}} F_\mu(W) := \frac{1}{K} \sum_{k=1}^{K} [f_k(w_k) + \mu \mathcal{B}(w_k, \mathcal{A}(W))], \tag{2.13}$$

donde $\mu > 0$ es una constante de penalización fija. $\mathcal{A}$ es una función de agregación que devuelve un modelo central $\tilde{w} \in \mathbb{R}^d$ de $W := \{w_1, \ldots, w_K\}$, y $\mathcal{B}(\cdot, \cdot)$ una distancia que penaliza la desviación de un modelo local del modelo central agregado $\tilde{w} = \mathcal{A}(W)$. Notar que $\tilde{w}$ puede ser la media, mediana, etc...

Cuando $\mu = 0$, el problema 2.13 se reduce al plano no federado donde cada nodo minimiza independientemente su función objetivo local. Por otra parte, para $\mu > 0$ y $\cdot \mathcal{A}(W) = \frac{1}{K} \sum_k w_k$, si se toma $\mathcal{B}$ tal que $\mathcal{B}(w, \tilde{w}) = \infty$ si $w \neq \tilde{w}$ y $\mathcal{B}(w, \tilde{w}) = 0$ en caso contrario, entonces el problema 2.13 es equivalente al problema presentado en FedAvg.

2.8.2. Tratamiento de la Agregación Robusta

Se pretende explotar la función $\mathcal{B}$ para definir una familia de funciones de agregación, entre las que se incluyen la media, la mediana, la mediana geométrica, la *coordinate-wise*

media, etc... Es decir, el modelo global $\tilde{w}$ se calcula agregandos los modelos locales $\{w_1, \ldots, w_k\}$ mediante

$$\tilde{w} \leftarrow \mathcal{A}(W) := \arg\min_{w \in \mathbb{R}} \frac{1}{K} \sum_{k=1}^{K} \mathcal{B}(w_k, w) \tag{2.14}$$

Eligiendo $\mathcal{B}$ se pueden definir diferentes funciones de agregación, por ejemplo tomando $\mathcal{B} = \|w - w'\|_2$ o $\mathcal{B} = \|w - w'\|_1$, se obtiene la mediana geométrica y *coordinate-wise media* respectivamente. Para unificar los métodos de agregación que incluyen funciones no diferenciables $\mathcal{B}$ en 2.13, se define la siguiente familia de funciones paramétricas $\mathcal{B}$ donde se toma una función convexa $\phi : \mathbb{R}^d \longleftrightarrow [0, \infty]$ y un parámetro de suvidad-solidez $\rho > 0$:

$$\begin{aligned} &\mathcal{B}(w_k, \tilde{w}) = \Phi_\rho(w_k - \tilde{w}), \\ &\Phi_\rho(w) := \min_{w' \in \mathbb{R}^d} \left[\phi(w') + \frac{1}{2_\rho} \|w - w'\|_2^2 \right]. \end{aligned} \tag{2.15}$$

Se llama al problema de mínimos en 2.15 el operados proximal de ϕ y lo denotamos por $\text{prox}_\phi^\rho(w)$. Nótese que Φ_ρ es una función infinitamente diferenciale diferenciable conocida como la envolvente de Moreau [MEDIAPROXIMAL Y LAS ENVOLTURAS DE MOREAU Y GOEBEL]

2.8.3. NODOS LOCALES

Cada nodo resuelve su propia versión del problema ejecutando, ejecutando E_k iteraciones de las posterior actualización, con un factor de aprendizaje $\nu > 0$

$$w_k \leftarrow \theta[w_k - \nu \nabla f_k(w_k)] + (1 - \theta) z_k, \quad i = 1, \ldots, E_k. \tag{2.16}$$

El factor de personalización del aprendizaje se consigue mediante el parámetro de regularización z_k y la constante $\theta \in (0, 1]$ controla el grade de regularización mientras se entrena el modelo local usando 2.16. En la práctica, el gradiente exacto $\nabla f_k(w_k)$ en 2.16 se sustituye por una estimación insesgada. En los métodos estándar, zk se define en el modelo global $\tilde{w}$. Sin embargo, Fed+ propone una personalización robusta.

2.8.4. Reformularización y Unificación

Para obtener un marco unificado que abarque la robustez y la personalización, consideremos de nuevo FedAvg, expresado equivalentemente como:

$$\begin{aligned} \min_{W,Z\in\mathbb{R}^{d\times K},\,\tilde{w}\in\mathbb{R}^d} \quad & \frac{1}{K}\sum_{k=1}^{K} f_k(w_k) \\ \textbf{s.t.} \quad & \\ & w_k = z_k; \\ & z_k = \tilde{w}; \\ & k = 1,\dots,K, \end{aligned} \tag{2.17}$$

Con $Z := \{z_1,\dots,z_K\}$ con $z_k \in \mathbb{R}^d$. Mientras que el método de multiplicación de dirección alterna ADMM [3] puede utilizarse para resolver problemas con restricciones de igualdad de la forma anterior, Fed+ adopta un enfoque basado en penalizaciones, adecuado tanto para entornos convexos como no convexos. A continuación, para manejar datos entre las partes, sustituimos las restricciones de igualdad en 2.17 por funciones de penalización:

$$\min_{W,Z\in\mathbb{R}^{d\times K},\,\tilde{w}\in\mathbb{R}^d} H_{\mu,\alpha}(W,Z,\tilde{w}) := \frac{1}{K}\sum_{k=1}^{K}\left[f_k(w_k) + \frac{\alpha}{2}\|w_k - z_k\|_2^2 + \mu\phi(z_k-\tilde{w})\right], \tag{2.18}$$

α es un parámetro predefinido fijo y $\phi : \mathbb{R}^d \to [0.\infty]$ es una función de penalización convexa. La siguiente proposición vincula la formulación 2.18 con el problema 2.13

Proposición 2.8.1. El problema 2.18 es un caso especial de 2.13, donde las funciones $\mathcal{A}$ y $\mathcal{B}$ se definen como en 2.14 y 2.15 respectivamente con $\rho = \mu/\alpha$, y se tiene la siguiente relación entre los dos problemas de optimización:

$$F_\mu(W) = \min_{W,Z\in\mathbb{R}^{d\times K},\,\tilde{w}\in\mathbb{R}^d} H_{\mu,\alpha}(W,Z,\tilde{w}). \tag{2.19}$$

Además, la formulación de Fed+ 2.18 sugiere una elección natural para la función de personalización $\mathcal{R} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ la cual calcula $z_k := \mathcal{R}(\tilde{w}, w_k)$ como una (solida) combinación del modelo local y el global. Para ser más precisos, Fed+ propone fijar $\mathcal{R}(\tilde{w}, w_k)$ minimizando 2.18 con respecto a z_k mientras se mantiene fijo w_k y $\tilde{w}$. Así se tiene lo siguiente:

$$z_k \leftarrow \mathcal{R}(\tilde{w}, w_k) = \tilde{w} + \text{prox}_{\phi}^{\rho}(w_k - \tilde{w}), \quad \rho = \mu/\alpha. \tag{2.20}$$

Con la siguiente proposición se relaciona la actualización local del cliente a la formulación del problema 2.18.

Proposición 2.8.2. Sea $\theta := \frac{1}{1+\alpha\nu}$. Entonces, la actualización local 2.16 es una iteración del gradiente descendiente con factor de aprendizaje $\nu' := \frac{\nu}{1+\alpha\nu}$ aplicada al siguiente subproblema:

$$\min_{w_k \in \mathbb{R}^d} F_k(w_k; z_k, \tilde{w}) := f_k(w_k) + \frac{\alpha}{2}\|w_k - z_k\|_2^2 + \mu\phi(z_k - \tilde{w}), \tag{2.21}$$

donde z_k y $\tilde{w}$ están fijos definidos por z_k^{t-1} y $\tilde{w}^{t-1}$ respectivamente.

La prueba de estas dos proposiciones y un estudio detallado del algoritmo Fed+ se pueden encontrar en [26].

2.8.5. ALGORITMO

Fed+ se define como una familia de métodos para abordar el problemas del federated learning para resolver el problema de mínimos 2.13 con $\mathcal{B}$ definida como en 2.15 y $\mathcal{A}$ como se define en 2.14. Fed+ está diseñado para construir funciones de agregación solidas $\mathcal{A}$.

Para abarcar casos especiales importantes, el algoritmo **??** introduce una serie de parámetros: $\lambda \in [0, 1]$, $\theta \in (0, 1]$, y $\mathcal{R} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$. Fed+ prone inicializar los modelos locales con los pesos de estos en la ronda anterior (lambda = 0 en la línea 10 del

algoritmo). Esto mitiga cambios bruscos en los modelos locales.

Algorithm 7: Fed+

Fed+: Clientes $k = 1, \ldots, K$; función de agregación $\mathcal{A}$; iteraciones locales por iteración global en k, E_k; factor de aprendizaje ν; y $\theta \in (0, 1])$, $\lambda \in [0, 1]$; y función de personalización sólida $\mathcal{R} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$.

Inicialización: Cada cliente k envía un modelo inicial local w_k^0 al agregador, que calcula el modelo central $\tilde{w}^0 \leftarrow \mathcal{A}(W^0)$.

Server executes:

for $t = 1, \ldots, T$ **do**

  Tomar una muestra de clientes $S_t \subset \{1, \ldots, K\}$.

  Se envía el modelo global $\tilde{w}^{t-1}$ a cada partido $k \in S_t$.

  for *cada partido* $k \in S_t$ ***en paralelo*** **do**

    $w_k^t \leftarrow$ Solver-local$(k, t, \tilde{w}^{t-1}, w_k^{t-1})$ `// Cada cliente` $k \notin S_t$ `se define` $w_k^t \leftarrow w_k^{t-1}$.

    Se manda el modelo local w_k^t al agregador.

  Se calcular el modelo global: $\tilde{w}^t \leftarrow \mathcal{A}(W^t)$.

Solver-local$(k, t, \tilde{w}^{t-1}, w_k^{t-1})$:

`// Se ejecuta sobre cada nodo activo` $k \in S_t$

Se calcula un modelo local robusto: $z_k^{t-1} := \mathcal{R}(\tilde{w}^{t-1}, w_k^{t-1})$.

Se actualiza el modelo local: $w_k^t \leftarrow (1 - \lambda) w_k^{t-1} + \lambda \tilde{w}_k^{t-1}$.

for $i = 1, \ldots, E_k$ **do**

  $w_k^t \leftarrow \theta[w_k^t - \nu \nabla f_k(w_k^t)] + (1 - \theta) z_k^{t-1}$.

Herramientas y Experimento Practico 3

En este capítulo realizaremos una aplicación de este nuevo concepto de aprendizaje. Tomaremos una de las bases de datos disponibles en Tensorflow para realizar el entrenamiento de varios modelos Federated Leanring usando varios algoritmos de agregación.

3.1. Herramientas

Las herramientas de más populares que implementan soluciones para el Federated Learning son Pysyft de Openmined, Tensorflow Federated de Google e IBM Federated Learning. Las dos primeras son de código abierto, debido a que siguen en desarrollo se encuentran en un estado bastante inestable. IBM Federated Learning, aunque es de código privado, presenta una solución más completa y bien estructurada, por lo que usaremos la API que ofrece IBM para el desarrollo del problema.

Además usaremos una librería llamada MLflow. MLflow es una API de Databricks para la monitorización del ciclo de vida completo de modelos. Mlflow ofrece una API y una interfaz de usuario, mediante las cuales podremos realizar la monitorización, agrupar ejecuciones, visualizar métricas, agregar los datos de entrenamiento, añadir anotaciones, registrar los modelos, etc... Para este propósito MLflow consta de cuatro componentes:

1. **MLflow Tracking** (MLT):

   MLT se basado y construido entorno al concepto de ejecución, que representa la ejecución de un fragmento de código. Cada ejecución puede almacenar la siguiente información:

   a) Versión del código.

   b) hora de inicio y fin de una ejecución.

   c) El Source, es decir, el nombre del archivo o el nombre del proyecto si se ha obtenido de un MLP.

d) Parámetros que representan pares clave-valor en los que las dos componentes son cadenas de caracteres.

e) Métricas, al igual que los parámetros son paras claves-valor pero en este caso el valor es de tipo numérico y puede ir actualizándose durante el transcurso de una ejecución, esto lo hace clave para ir registrando los valores de una función *loss*. Después MLflow nos permite acceder al historial completo, visualizar los registros y comprar los resultados obtenidos con los que se han almacenado de ejecuciones previas.

f) Artefactos, estos son los archivos resultantes de una ejecución y que no son ni parámetros ni métricas, pueden ser los datos de entrenamiento, imágenes, modelos... Se permite que estén en cualquier formato

Si se registran ejecuciones en un MLP, MLflow almacena la URI y la versión del proyecto origen. Un factor importante a tener en cuenta es que se pueden organizar y agrupar distintas ejecuciones en conjuntos de estas llamados **experimentos**. Estos agrupan las ejecuciones lo que los hace muy útiles para agruparlas en diferentes problemas o proyectos que estemos tratando. La API de MLflow y su interfaz permite al usuario crear y buscar experimentos, además de consultar ejecuciones utilizando parámetros o métricas.

Una de las herramientas más interesantes de MLflow nos ofrece la oportunidad de realizar un registro automático de parámetros, métricas y artefactos si usamos algunas de las siguientes librerías:

a) Scikit-learn

b) Tensorflow y Keras

c) Gluon

d) XGBoost

e) LightGBM

f) Statsmodels

g) Spark

h) Fastai

i) Pytorch

2. **MLflow Proyects** (MLP): Este componente proporciona un formato de empaquetar el código de forma que sea reutilizable y reproducible. Incluye una API e instrucciones en línea de comandos que permiten ejecutar proyectos machine learning. Esto hace posible encadenar diferentes proyectos en flujos de trabajo.

3. **MLflow Models** (MLM): MLM es un formato estándar de almacenamiento de modelos, permite que el modelo una vez almacenado puede utilizarse desde una gran cantidad de herramientas intermedias, servir un modelo en *real-time* mediante un *REST API* o Realizar inferencia en Apache Spark. Este componente crea una convención para almacenar modelos de diferentes ”variedades”(esto hace referencia a la librería con la que se ha construido un modelo, por ejemplo, tensorflow, spark o pytorch) y permite que pueda ser interpretado por otro tipo de herramientas.

4. **MLflow Registry** (MLR): Por último Registry, sería un almacén de modelos. Proporciona un conjunto de APIs y una interfaz de usuario para realizar conjuntamente la gestión, administración y mantenimiento del ciclo de vida de los modelos. Registra el historial completo del modelo (ejecución y experimento del que procede), realiza un control de versiones, transiciones de etapa (por ejemplo de la etapa, preproducción a producción) y además permite añadir anotaciones.

Se usará también **Docker**. Esta herramienta permite crear contenedores de software y se utilizará para crear un sistema distribuido que simule el punto de partida del Federated Learning. Se usará Python como lenguaje base.

3.2. Experimento

Bibliografía

[1] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, Bingsheng He (2021) *A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection.* arXiv:1907.09693v6

[2] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G.L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, Sen Zhao (2021) *Advances and Open Problems in Federated Learning.* arXiv:1912.04977v3

[3] Sören Bartels, Marijo Milicevic (2017) *Alternating direction method of multipliers with variable step sizes.* arXiv:1704.06069v1

[4] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze (2011) *Cloud federation.* in Proc. of the 2nd Intl. Conf. on CloudComputing, Grids, and Virtualization, 2011, pp. 32–38.

[5] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Agüera y Arcas (2017) *Communication-Efficient Learning of Deep Networks from Decentralized Data.* arXiv:1602.05629v3

[6] Amit P. Sheth, James A. Larson (1990) *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases.* ACM Computing Surveys, 22, pp 183-236

[7] Arash Mehrjou (2021) *Federated Learning as a Mean-Field Game.* arXiv:2107.03770v1

[8] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, Virginia Smith (2019) *Federated Learning: Challenges, Methods, and Future Directions.* arXiv:1908.07873v1

[9] Tian Li and Maziar Sanjabi and Virginia Smith (2020) *Fair Resource Allocation in Federated Learning.* arXiv:1905.10497v2

[10] Enmao Diao, Jie Ding, Vahid Tarokh (2021) *HeteroFL: Computation and Communication Efficient Federated Learning for Heterogeneous Clients.* arXiv:2010.01264v2

[11] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, Vitaly Shmatikov (2019) *How To Backdoor Federated Learning.* arXiv:1807.00459v3.

[12] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, Mark Purcell, Ambrish Rawat, Tran Minh, Naoise Holohan, Supriyo Chakraborty, Shalisha Whitherspoon, Dean Steuer, Laura Wynter, Hifaz Hassan, Sean Laguna, Mikhail Yurochkin, Mayank Agarwal, Ebube Chuba, Annie Abay (2020) *IBM Federated Learning: an Enterprise Framework White Paper V0.1.* arXiv:2007.10987v1.

[13] Marlo Carranza Purca, Tomás Alberto Núñez Lay (2014) *MEDIA PROXIMAL y LAS ENVOLTURAS DE MOREAU y GOEBEL.* Pesquimat, 15, 10.15381/pes.v15i1.9600.

[14] Leslie Lamport, Robert Shostak, Marshall Pease (1982) *The Byzantine Generals Problem.* ACM Transactions on Programming Languages and Systems, Volume 4, Issue 3, July 1982, pp 382–401, https://doi.org/10.1145/357172.357176

[15] Reza Shokri, Vitaly Shmatikov (2015) *Privacy-Preserving Deep Learning.* Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, October 2015, pp 1310–1321 https://doi.org/10.1145/2810103.2813687

[16] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, Jason Roselander (2019) *Towards Federated Learning at Scale: System Design.* arXiv:1902.01046v2

[17] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, Dave Bacon (2017) *Federated Learning: Strategies for Improving Communication Efficiency.* arXiv:1610.05492v2

[18] Tamara Broderick, Michael I. Jordan, Jim Pitman (2011) *Beta processes, stick-breaking, and power laws.* arXiv:1106.0539v2

[19] Johnson, R.A. (1970) *Atomic and nonatomic measures.* Proceedings of the American Mathematical Society Vol. 25, No. 3, pp. 650-655. Published By: American Mathematical Society.

[20] Thibaux, Romain and Jordan, Michael (2007) *Hierarchical Beta Processes and the Indian Buffet Process. Journal of Machine Learning Research.* Journal of Machine Learning Research - Proceedings Track. 2. 564-571.

[21] Francisco Javier Merino Cabrera (2016) *Las funciones eulerianas Gamma y Beta complejas.* Universidad de la Laguna, Grado en Matemáticas.

[22] Yee Whye Teh, Dilan Grür, Zoubin Ghahramani (2007) *The Exact Inference of Beta Process and Beta Bernoulli Process From Finite Observations.* Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, PMLR 2, pp 556-563.

[23] Thomas L. Griffiths, Zoubin Ghahramani (2011). *The Indian Buffet Process: An Introduction and Review.* Journal of Machine Learning Research 12, pp 1185-1224. Editor: David M. Blei.

[24] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang, Yasaman Khazaeni (2019) *Bayesian Nonparametric Federated Learning of Neural Networks.* arXiv:1905.12022v1.

[25] Dong Yin, Yudong Chen, Kannan Ramchandran, Peter Bartlett (2021) *Byzantine-Robust Distributed Learning: Towards Optimal Statistical Rates.* arXiv:1803.01498v2.

[26] Pengqian Yu, Achintya Kundu, Laura Wynter, Shiau Hong Lim (2020) *Fed+: A Unified Approach to Robust Personalized Federated Learning.* arXiv:2009.06303v2.

[27] Peva Blanchard and El Mahdi El Mhamdi and Rachid Guerraoui and Julien Stainer (2017) *Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent.* Published in NIPS.

[28] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang (2019) *Statistical Model Aggregation via Parameter Matching.* arXiv:1911.00218v1

[29] Cong Xie, Oluwasanmi Koyejo, Indranil Gupta (2018) *Zeno: Distributed Stochastic Gradient Descent with Suspicion-based Fault-tolerance.* arXiv:1805.10032v3

[30] Brendan McMahan and Daniel Ramage (2017) *Federated Learning: Collaborative Machine Learning without Centralized Training Data.* Google AI Blog. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[31] Bottou, L. (1999) *On-line learning and stochastic approximations.*

[32] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, Rafal Jozefowicz (2016) *Revisiting Distributed Synchronous SGD.* arXiv:1604.00981v3

[33] Laura Alejandra Caicedo Suárez e Isabella Burbano García *DISTRIBUCIÓN BETA.* https://rstudio-pubs-static.s3.amazonaws.com/166233_44a100ae858948c89b6e20ae657088e9.html