Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

CUANTIFICACIÓN DE LA INCERTIDUMBRE EN REDES BAYESIANAS PARA LA CLASIFICACIÓN DE IMÁGENES USANDO PYTHON

Israel Rivera Zárate
Instituto Politécnico Nacional-CIDETEC
irivera@ipn.mx

Miguel Hernández Bolaños
Instituto Politécnico Nacional-CIDETEC
mbolanos@ipn.mx

Patricia Pérez Romero
Instituto Politécnico Nacional-CIDETEC
promerop@ipn.mx

Abstract

The Bayesian convolutional neural networks (BayesCNNs) with variational inference, are a variant of convolutional neural networks (CNNs), in which the intractable posterior probability distributions over weights are inferred by Bayes by Backprop. These networks are often unable to correctly assess uncertainty in the training data and therefore make overly confident decisions about the correct class, prediction, or action. In the present work, we propose a variational inference method that achieves performances equivalent to frequentist inference in identical architectures on several datasets. Two architectures are compared: AlexNet versus VGG16 under BayesCNN learning process. We estimate their performance using the MNIST, CIFAR-10 and CIFAR-100 datasets for the task of image classification.

Palabras clave: Inferencia Bayesiana, incertidumbre, redes neuronales convolucionales.

Como lo señala Charles Blundell et al. (2015), la incertidumbre en una red mide la certeza del modelo con respecto a su predicción. En el modelado bayesiano, existen dos tipos principales de incertidumbre que se pueden modelar: incertidumbre aleatoria e incertidumbre epistémica. La incertidumbre aleatoria mide el ruido inherente a las observaciones. Este tipo de incertidumbre está presente en el método de recopilación de datos, como el ruido del sensor o el ruido de movimiento, que es uniforme en todo el

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

conjunto de datos. No se puede reducir si se recopilan más datos. La incertidumbre epistémica, por otro lado, representa la incertidumbre causada por el modelo. Esta incertidumbre se puede reducir con más datos y a menudo se denomina incertidumbre del modelo. La incertidumbre aleatoria se puede clasificar en incertidumbre homocedástica, incertidumbre que se mantiene constante para diferentes entradas e incertidumbre heterocedástica. La incertidumbre heterocedástica depende de las entradas del modelo, y algunas entradas pueden tener salidas más ruidosas que otras. Esta incertidumbre es particularmente importante para evitar que el modelo genere decisiones demasiado seguras. Los trabajos actuales miden las incertidumbres mediante la aplicación de distribuciones de probabilidad sobre los parámetros o los resultados del modelo. La incertidumbre epistémica se modela aplicando una distribución a priori a los pesos de un modelo y luego intentando capturar cuánto varían estos pesos dados ciertos datos. La incertidumbre aleatoria, por otro lado, se modela aplicando una distribución sobre los resultados del modelo. Ver figura 1.

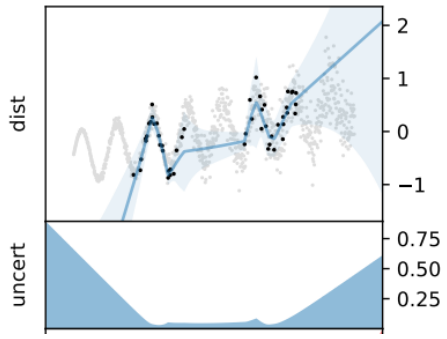

Figura 1. Arriba: ajuste del modelo. Abajo: incertidumbre. Elaboración propia.

I. INFERENCIA BAYESIANA

Como lo indica Adji Bousso et al. (2017), se define una función $y = f(x)$ que estima las entradas $\{x_1, \dots, x_N\}$ y sus correspondientes salidas $\{y_1, \dots, y_N\}$ y produce una salida predictiva. Mediante la inferencia bayesiana, se utiliza una distribución a priori en el espacio de funciones $p(f)$. Esta distribución representa la creencia previa sobre qué funciones probablemente generaron a los datos. La verosimilitud se define como $p(Y \mid f, X)$ para capturar el proceso mediante el cual se genera la observación de una función. Utilizamos la regla de Bayes para hallar la distribución a posteriori dado el conjunto de datos: $p(f \mid X, Y)$. Ver figura 2.

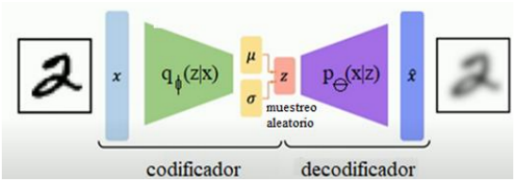

Figura 2. Inferencia Variacional Bayesiana. Elaboración propia.

La nueva salida se puede predecir para un nuevo punto de entrada x^* integrando todas las funciones posibles de f:

$$p(y^*|x^*, X, Y) = \int p(y^*|f^*)p(f^*|x^*, X, Y)df^*$$

… (1)

La ecuación (1) es insoluble, pero se puede aproximar tomando un conjunto finito de variables aleatorias ω (pesos en los nodos) y condicionando el modelo a éstas. Sin embargo, se basa en el supuesto de que el modelo depende únicamente de estas variables,

Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

convirtiéndolas en estadísticas suficientes en nuestro modelo aproximado.

De este modo, minimizamos la divergencia de Kullback-Leibler (KL), que intuitivamente es una medida de similitud entre dos distribuciones: $\mathrm{KL}(q(\omega) \,||\, p(\omega \mid X, Y))$, lo que resulta en la distribución predictiva aproximada.

$$q(y^*|x^*) = \int p(y^*|f^*)p(f^*|x^*, w)q(w)df^*dw.$$

… (2)

Minimizar la divergencia de Kullback-Leibler equivale a maximizar el límite inferior de la evidencia logarítmica (ELBO).

$$KL_{\mathrm{VI}} := \int q(w)p(F|X, w)\log p(Y|F)dFdw$$

$$- KL(q(w)||p(w)) \quad \ldots (3)$$

Con respecto a los parámetros variacionales que definen $q(\omega)$. Esto se conoce como inferencia variacional, una técnica estándar en el modelado bayesiano. Maximizar la divergencia KL entre la distribución a posteriori y la distribución a priori sobre ω dará como resultado una distribución variacional que aprende una buena representación de los datos (obtenida a partir del logaritmo de la verosimilitud) y se acerca más a la distribución a priori. En otras palabras, puede prevenir el sobreajuste. Ver figura 3.

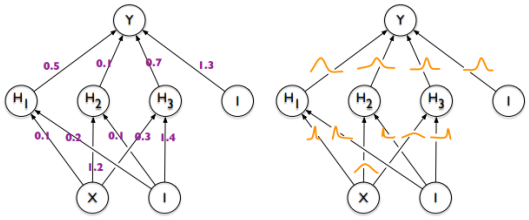

Figura 3. Inferencia Izquierda: modelo CNN. Derecha: modelo BayesCNN. Elaboración propia.

Estimación de la incertidumbre en CNNs

En las tareas de clasificación, nos interesa la distribución predictiva $p_D\,(y^* \mid x^*)$ donde x^* es un ejemplo de datos no vistos e y^* su clase predicha. Para una red neuronal bayesiana, esta cantidad viene dada por:

$$p_{\mathcal{D}}(y^*|x^*) = \int p_w(y^*|x^*)\; p_{\mathcal{D}}(w)\; dw$$

… (4)

En Bayes por retropropagación, las distribuciones gaussianas $q_\theta(\omega \mid D) \sim \mathcal{N}\,(\omega \mid \mu, \sigma^2)$, donde $\theta = \{\mu, \sigma\}$ se aprenden con un conjunto de datos $D = \{x_i, y_i\}_{i=1}^{n}$ como se explicó anteriormente. Debido a la naturaleza discreta y finita de la mayoría de las tareas de clasificación, se suele asumir que la distribución predictiva es categórica. Incorporar este aspecto a la distribución predictiva nos da:

$$p_{\mathcal{D}}(y^*|x^*) = \int \mathrm{Cat}(y^*|f_w(x^*))\mathcal{N}(w|\mu, \sigma^2)\; dw$$

$$= \int \prod_{c=1}^{C} f(x_c^*|w)^{y_c^*} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w-\mu)^2}{2\sigma^2}}\; dw \quad \ldots (5)$$

donde C es el número total de clases $\sum_c f(x_C^* \mid \omega) = 1$.

Como no existe una solución de forma cerrada debido a la falta de conjugación entre las distribuciones categórica y gaussiana, no podemos recuperar esta distribución. Sin embargo, podemos construir un estimador imparcial de la expectativa mediante muestreo de $q_\theta(\omega \mid D)$:

$$\mathbb{E}_q[p_{\mathcal{D}}(y^*|x^*)] = \int q_\theta(w|\mathcal{D})\ p_w(y|x)\ dw \quad \ldots (6)$$

$$\approx \frac{1}{T}\sum_{t=1}^{T} p_{w_t}(y^*|x^*) \quad \ldots (7)$$

Donde T es el número predefinido de muestras. Este estimador nos permite evaluar la incertidumbre de nuestras predicciones mediante la definición de varianza, por lo que se denomina varianza predictiva y se denota como Var_q:

$$\mathrm{Var}_q\big(p(y^*|x^*)\big) = \mathbb{E}_q[yy^T] - \mathbb{E}_q[y]\mathbb{E}_q[y]^T \quad \ldots (8)$$

Esta cantidad puede ser descompuesta en incertidumbre aleatoria y epistémica:

$$\mathrm{Var}_q\big(p(y^*|x^*)\big) = \underbrace{\frac{1}{T}\sum_{t=1}^{T}\mathrm{diag}(\hat{p}_t) - \hat{p}_t\,\hat{p}_t^T}_{\text{aleatoric}} + \underbrace{\frac{1}{T}\sum_{t=1}^{T}(\hat{p}_t - \bar{p})(\hat{p}_t - \bar{p})^T}_{\text{epistemic}} \quad \ldots (9)$$

Con

$$\bar{p} = \tfrac{1}{T}\textstyle\sum_{t=1}^{T}\hat{p}_t \text{ and } \hat{p}_t = \mathrm{Softmax}\big(f_{w_t}(x^*)\big).$$

… (10)

II. DESARROLLO

La implementación se desarrolló en Python (https://www.python.org, versión 3.6.1) con la ayuda de las librerías gratuitas PyTorch (https://pytorch.org, versión 2.0) y Tensorflow (https://www.tensorflow.org, versión 1.2.1), ampliamente aceptadas por la comunidad. El entorno de desarrollo empleado ha sido el Notebook de Jupyter (http://jupyter.org, versión 5.0.0).

Se compararán los desempeños en la clasificación de imágenes de una arquitectura convolucional basada en pesos como es AlexNet y la red VGG16, contra sus modelos entrenados mediante aprendizaje Bayesiano. Como los modelos frecuentistas ya están entrenados la metodología se centra exclusivamente en los modelos Bayesianos. Cabe señalar que estos modelos fueron elegidos por poseer un alto desempeño en la tarea de clasificación de imágenes empleando las bases de datos MNIST, CIFAR-10 y CIFAR-100 respectivamente. Ver figura 4 y 5.

Layer type	width	stride	padding	input shape	nonlinearity
convolution (11×11)	64	4	5	$M \times 3 \times 32 \times 32$	Softplus
max-pooling (2×2)		2	0	$M \times 64 \times 32 \times 32$	
convolution (5×5)	192	1	2	$M \times 64 \times 15 \times 15$	Softplus
max-pooling (2×2)		2	0	$M \times 192 \times 15 \times 15$	
convolution (3×3)	384	1	1	$M \times 192 \times 7 \times 7$	Softplus
convolution (3×3)	256	1	1	$M \times 384 \times 7 \times 7$	Softplus
convolution (3×3)	128	1	1	$M \times 256 \times 7 \times 7$	Softplus
max-pooling (2×2)		2	0	$M \times 128 \times 7 \times 7$	
fully-connected	128			$M \times 128$	

Figura 4. Arquitectura de la red AlexNet. Elaboración propia.

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

No	Convolution	Output Dimension	Pooling	Output Dimension
layer 1&2	convolution layer of 64 channel of 3x3 kernel with padding 1, stride 1	224x224x64	Max pool stride=2, size 2x2	112x112x64
layer3&4	convolution layer of 128 channel of 3x3 kernel	112x112x128	Max pool stride=2, size 2x2	56x56x128
layer5,6,7	convolution layer of 256 channel of 3x3 kernel	56x56x256	Max pool stride=2, size 2x2	28x28x256
layer8,9,10	Convolution layer of 512 channel of 3x3 kernel	28x28x512	Max pool stride=2, size 2x2	14x14x512
layer11,12,13	Convolution layer of 512 channel of 3x3 kernel	14x14x512	Max pool stride=2, size 2x2	7x7x512

Figura 5. Arquitectura de la red VGG16. Elaboración propia.

Metodología:

La metodología adoptada en el presente trabajo consiste en los siguientes pasos:

[Paso 1:] Creación de los modelos.

[Paso 2:] Aprendizaje de la red.

[Paso 3:] Cuantificación de la incertidumbre.

PASO 1. Creación de los Modelos

En todos los experimentos realizados, implementamos redes neuronales bayesianas con inferencia variacional y frecuentistas (basada en pesos). Los fragmentos de código de los modelos de AlexNet aparecen en las figuras 6 y 7 respectivamente. Una modificación mínima se utilizó para el código de la arquitectura VGG16.

```
class AlexNet(nn.Module):

    def __init__(self, num_classes, inputs=3):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(inputs, 64, kernel_size=11, stride=4, padding=5),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.5),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.classifier = nn.Linear(256, num_classes)
```

Figura 6. Fragmento de código modelo AlexNet frecuentista. Elaboración propia.

```
if activation_type=='softplus':
    self.act = nn.Softplus
elif activation_type=='relu':
    self.act = nn.ReLU
else:
    raise ValueError("Only softplus or relu supported")

self.conv1 = BBBConv2d(inputs, 64, 11, stride=4, padding=5, bias=True, priors=self.priors)
self.act1 = self.act()
self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv2 = BBBConv2d(64, 192, 5, padding=2, bias=True, priors=self.priors)
self.act2 = self.act()
self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv3 = BBBConv2d(192, 384, 3, padding=1, bias=True, priors=self.priors)
self.act3 = self.act()

self.conv4 = BBBConv2d(384, 256, 3, padding=1, bias=True, priors=self.priors)
self.act4 = self.act()

self.conv5 = BBBConv2d(256, 128, 3, padding=1, bias=True, priors=self.priors)
self.act5 = self.act()
self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

self.flatten = FlattenLayer(1 * 1 * 128)
self.classifier = BBBLinear(1 * 1 * 128, outputs, bias=True, priors=self.priors)
```

Figura 7. Fragmento de código modelo AlexNet Bayesiano. Elaboración propia.

PASO 2. Aprendizaje de la red

A. Inicialización de parámetros:

Utilizamos una distribución gaussiana y almacenamos los valores de media y varianza en lugar de un solo peso. La varianza no puede ser negativa, lo que se garantiza utilizando softplus como función de activación.

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957

Expresamos la varianza como $\sigma_i = softplus\ (\rho_i)$, donde ρ es un parámetro sin restricciones.

$$\mathrm{Softplus}(x) = \frac{1}{\beta} \cdot \log\left(1 + \exp(\beta \cdot x)\right) \quad \ldots (11)$$

Tomamos la distribución gaussiana e inicializamos la media μ como 0 y la varianza σ (y, por lo tanto, ρ) de forma aleatoria. Observamos una media centrada en 0 y una varianza que comienza con un número grande y disminuye gradualmente con el tiempo. Una buena inicialización también puede consistir en restringir la varianza e inicializarla a un valor pequeño. Sin embargo, podría depender de los datos, por lo que aún no se ha descubierto un buen método para la inicialización de la varianza. Realizamos un descenso de gradiente sobre θ = (μ, ρ) y un peso individual $\omega_i \sim \mathcal{N}\ (\omega_i \mid \mu_i, \sigma_i)$. Ver figura 8 fragmento de código inicialización de parámetros.

```
def forward(self, x, sample=True):

    self.W_sigma = torch.log1p(torch.exp(self.W_rho))
    if self.use_bias:
        self.bias_sigma = torch.log1p(torch.exp(self.bias_rho))
        bias_var = self.bias_sigma ** 2
    else:
        self.bias_sigma = bias_var = None

    act_mu = F.conv2d(
        x, self.W_mu, self.bias_mu, self.stride, self.padding, self.dilation, self.groups)
    act_var = 1e-16 + F.conv2d(
        x ** 2, self.W_sigma ** 2, bias_var, self.stride, self.padding, self.dilation, self.groups)
    act_std = torch.sqrt(act_var)

    if self.training or sample:
        eps = torch.empty(act_mu.size()).normal_(0, 1).to(self.device)
        return act_mu + act_std * eps
    else:
        return act_mu
```

Figura 8. Fragmento de código: Inicialización de parámetros. Elaboración propia.

B. Entrenamiento

Se utilizaron tasa de aprendizaje de 0.001, épocas del orden de 100, tamaño del batch de 256, tamaño del muestreo de 10 a 25, función de pérdida entropía cruzada, posterior aproximado $q_\theta(\omega \mid D)$ de -10, optimizador Adam, normalización L2 de 0.0005.

B.1 Forward Pass (Pase hacia adelante):

Como indica Alex Graves (2011). El problema principal es que el término de la esperanza $E_{q(\omega|\theta)}[\log p(D|\omega)]$ es difícil de diferenciar con respecto a θ θ directamente, porque la esperanza se toma sobre $q(\omega|\theta)$ que depende de θ. "Bayes por Backpropagation" resuelve esto usando el truco de reparametrización. Ver figura 9. En lugar de muestrear ω directamente de $q(\omega|\theta)$, se muestrea un ruido ε de una distribución simple (ej. $\mathcal{N}$ (0,1)). Ahora, la dependencia de θ = (μ, σ) está fuera de la operación de muestreo, lo que permite que los gradientes de retropropagación fluyan a través de esta transformación.

$$\omega = \mu + \sigma \odot \varepsilon \ldots (12)$$

```
def train_model(net, optimizer, criterion, trainloader, num_ens=1, beta_type=0.1, epoch=None, num_epochs=None):
    net.train()
    training_loss = 0.0
    accs = []
    kl_list = []
    for i, (inputs, labels) in enumerate(trainloader, 1):

        optimizer.zero_grad()

        inputs, labels = inputs.to(device), labels.to(device)
        outputs = torch.zeros(inputs.shape[0], net.num_classes, num_ens).to(device)

        kl = 0.0
        for j in range(num_ens):
            net_out, _kl = net(inputs)
            kl += _kl
            outputs[:, :, j] = F.log_softmax(net_out, dim=1)

        kl = kl / num_ens
        kl_list.append(kl.item())
        log_outputs = utils.logmeanexp(outputs, dim=2)

        beta = metrics.get_beta(i-1, len(trainloader), beta_type, epoch, num_epochs)
        loss = criterion(log_outputs, labels, kl, beta)
        loss.backward()
        optimizer.step()

        accs.append(metrics.acc(log_outputs.data, labels))
        training_loss += loss.cpu().data.numpy()
    return training_loss/len(trainloader), np.mean(accs), np.mean(kl_list)
```

Figura 9. Fragmento de código: Entrenamiento Bayesiano. Elaboración propia.

B.2 Función de pérdida:

Cabe señalar que el modelo se entrena maximizando el límite inferior de la evidencia (ELBO) que en este caso corresponde con la función de pérdida establecida en la ecuación (3). El ELBO se optimiza típicamente por el gradiente respecto a θ. Ver figura 10. Donde la Divergencia KL tiene una solución analítica y K es la dimensionalidad de la variable latente:

$$D_{KL}[q(\omega|\theta)||p(\omega)] =$$

$$\frac{1}{2}\sum_{j=1}^{K}(\mu_j^2 + \sigma_j^2 - \log\sigma_j^2 - 1)\ldots \quad (13)$$

```
def acc(outputs, targets):
    return np.mean(outputs.cpu().numpy().argmax(axis=1) == targets.data.cpu().numpy())


def calculate_kl(mu_q, sig_q, mu_p, sig_p):
    kl = 0.5 * (2 * torch.log(sig_p / sig_q) - 1 + (sig_q / sig_p).pow(2) + ((mu_p - mu_q) / sig_p).pow(2)).sum()
    return kl


def get_beta(batch_idx, m, beta_type, epoch, num_epochs):
    if type(beta_type) is float:
        return beta_type

    if beta_type == "Blundell":
        beta = 2 ** (m - (batch_idx + 1)) / (2 ** m - 1)
    elif beta_type == "Soenderby":
        if epoch is None or num_epochs is None:
            raise ValueError('Soenderby method requires both epoch and num_epochs to be passed.')
        beta = min(epoch / (num_epochs // 4), 1)
    elif beta_type == "Standard":
        beta = 1 / m
    else:
        beta = 0
    return beta
```

Figura 10. Fragmento de código: KL Divergencia. Elaboración propia.

B.3 Backward Pass (Retropropagación):

Los parámetros (μ, ρ) de las distribuciones variacionales se actualizan usando un optimizador (como Adam o SGD) y los gradientes calculados:

$$\mu \leftarrow \mu + \alpha\nabla_{\mu}\text{ELBO}$$
$$\rho \leftarrow \rho + \alpha\nabla_{\rho}\text{ELBO}$$

PASO 3. Cuantificación de la Incertidumbre

Para todas nuestras tareas, utilizamos el optimizador de Adam como lo recomienda Charles Blundell et al. (2015), para optimizar los parámetros. También aplicamos el método de reparametrización local y calculamos el gradiente de la función de pérdida combinada con respecto a los parámetros variacionales (μ, ρ). Que corresponde con las expresiones de la ecuación (9). Ver figura 11.

```
def get_uncertainty_per_image(model, input_image, T=15, normalized=False):
    input_image = input_image.unsqueeze(0)
    input_images = input_image.repeat(T, 1, 1, 1)

    net_out, _ = model(input_images)
    pred = torch.mean(net_out, dim=0).cpu().detach().numpy()
    if normalized:
        prediction = F.softplus(net_out)
        p_hat = prediction / torch.sum(prediction, dim=1).unsqueeze(1)
    else:
        p_hat = F.softmax(net_out, dim=1)
    p_hat = p_hat.detach().cpu().numpy()
    p_bar = np.mean(p_hat, axis=0)

    temp = p_hat - np.expand_dims(p_bar, 0)
    epistemic = np.dot(temp.T, temp) / T
    epistemic = np.diag(epistemic)

    aleatoric = np.diag(p_bar) - (np.dot(p_hat.T, p_hat) / T)
    aleatoric = np.diag(aleatoric)

    return pred, epistemic, aleatoric
```

Figura 11. Fragmento: Cuantificación de la Incertidumbre. Elaboración propia.

III. PRUEBAS Y RESULTADOS

Caso práctico 1: Conjuntos de datos pequeños (MNIST, CIFAR-10)

Como lo señala Yann LeCun and Corinna Cortes (2010). Entrenamos las redes con los conjuntos de datos MNIST de dígitos manuscritos y CIFAR-10, ya que estos conjuntos de datos sirven ampliamente como referencia para el rendimiento de las CNN.

MNIST:

La base de datos MNIST de dígitos manuscritos cuenta con un conjunto de entrenamiento de 60 000 ejemplos y un conjunto de prueba de 10 000 ejemplos. Es un subconjunto de un conjunto mayor disponible en el NIST. Los dígitos se han normalizado en tamaño y se han centrado en una imagen de tamaño fijo de 28 x 28 píxeles. Cada imagen está en escala de grises y etiquetada con su clase correspondiente, que va del cero al nueve.

CIFAR-10:

Los CIFAR-10 son subconjuntos etiquetados del conjunto de datos de 80 millones de imágenes diminutas. El conjunto de datos CIFAR10 cuenta con un conjunto de datos de entrenamiento de 50,000 imágenes en color en 10 clases, con 5,000 imágenes de entrenamiento por clase, cada una de 32 x 32 píxeles. Hay 10,000 imágenes para pruebas.

Resultados:

En primer lugar, evaluamos el rendimiento de nuestro método propuesto, las redes neuronales bayesianas con inferencia variacional. La Tabla 1 muestra una comparación de las precisiones de validación (en porcentaje) para arquitecturas entrenadas con dos enfoques bayesianos distintos: la inferencia variacional, es decir, Bayes por retropropagación y abandono, según lo propuesto por Yarin Gal (2016).

Tabla 1. Comparación de la precisión en porcentaje entre los modelos AlexNet y VGG16.

	MNIST	CIFAR-10
Bayesian AlexNet (with VI)	99	73
Frequentist AlexNet	99	73
Bayesian VGG16 (with VI)	98	69
Frequentist VGG16	98	68
Bayesian VGG16 (with Dropout)	99.5	83

Comparamos los resultados de estos dos enfoques con el enfoque de inferencia frecuentista para ambos conjuntos de datos. Las redes neuronales complejas bayesianas entrenadas mediante inferencia variacional alcanzan precisiones de validación comparables a las de su contra partes entrenadas mediante inferencia frecuentista. En MNIST, las precisiones de validación de los dos enfoques bayesianos dispares son comparables, pero un enfoque bayesiano VGG16 con Dropout alcanza una precisión de validación considerablemente mayor en CIFAR-10, aunque no pudimos reproducir estos resultados. Ver figura 12.

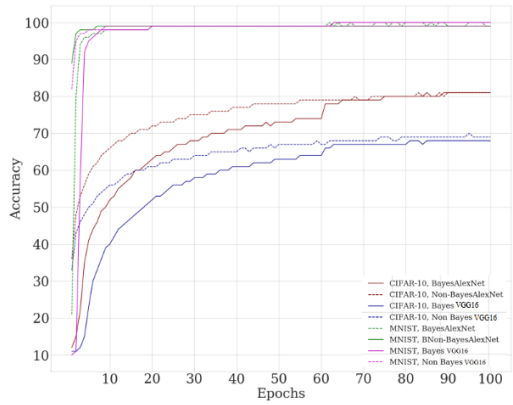

Figura 12. Comparación de las precisiones de validación de AlexNet y VGG16 con el enfoque frecuentista y Bayesiano en los conjuntos de datos MNIST y CIFAR-10. Elaboración propia.

Caso Práctico 2: Conjunto de Datos Grande (CIFAR-100)

CIFAR-100:

Este conjunto de datos es similar al CIFAR-10 y constituye un subconjunto etiquetado del conjunto de datos de 80 millones de imágenes diminutas. El conjunto de datos consta de 100 clases con 600 imágenes cada una. Cada clase incluye 500 imágenes de entrenamiento y 100 imágenes de validación. Las imágenes están coloreadas con una resolución de 32 x 32 píxeles.

Resultados:

En la Figura 12, mostramos cómo las redes bayesianas incorporan de forma natural los efectos de la regularización, ejemplificados en AlexNet. Mientras que una AlexNet entrenada mediante inferencia frecuentista sin regularización presenta un sobreajuste considerable en CIFAR-100, una AlexNet entrenada mediante inferencia bayesiana en CIFAR-100 no lo presenta. Esto se evidencia por la alta precisión de entrenamiento para el enfoque frecuentista sin pérdida de datos o con pérdida de datos de una capa. La CNN bayesiana tiene un rendimiento equivalente al de una red AlexNet entrenada por inferencia frecuentista con tres capas de Dropout después de la primera, cuarta y sexta capa de la arquitectura. Cabe destacar que la CNN bayesiana con 100 muestras presenta un sobreajuste ligeramente menor en comparación con la CNN bayesiana con 25 muestras. Sin embargo, un mayor número de muestras en un conjunto de datos más pequeño no resultó útil, por lo que nos mantuvimos en 25 como número de muestras para todos los demás experimentos. Ver figura 13.

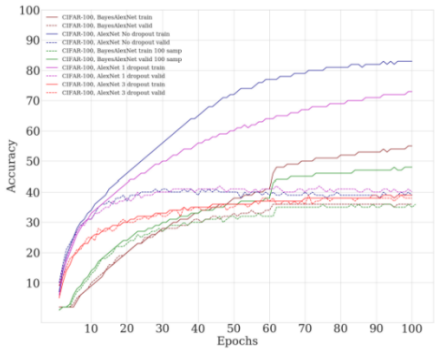

Figura 13. Comparación de las precisiones de entrenamiento y validación de AlexNet frecuentista y bayesiano con y sin pérdidas (Dropout) en los conjuntos de datos CIFAR-100. Elaboración propia.

La Tabla 2 muestra una comparación de las precisiones de validación (en porcentaje) para arquitecturas entrenadas con dos enfoques bayesianos distintos: la inferencia variacional, es decir, Bayes por retropropagación y abandono, según lo propuesto por Yarin Gal (2016).

Tabla 2. Comparación de la precisión en porcentaje entre los modelos AlexNet y VGG16.

	CIFAR-100
Bayesian AlexNet (with VI)	36
Frequentist AlexNet	38
Bayesian VGG16 (with VI)	31
Frequentist VGG16	33

Estimación de la Incertidumbre:

Finalmente, la Tabla 3 compara las medias de las incertidumbres aleatoria y epistémica para un modelo bayesiano VGG16 con inferencia variacional en MNIST y CIFAR-10. La incertidumbre aleatoria de CIFAR-10 es aproximadamente veinte veces mayor que la de MNIST. Considerando que la incertidumbre aleatoria mide la variabilidad irreducible y depende de los valores predichos, una mayor incertidumbre aleatoria para CIFAR-10 se puede deducir directamente de su menor precisión de validación y podría deberse, además, al menor número de ejemplos de entrenamiento. La incertidumbre epistémica de CIFAR-10 es aproximadamente quince veces mayor que la de MNIST, lo cual anticipamos, ya que la incertidumbre epistémica disminuye proporcionalmente a la precisión de la validación.

Tabla 3. Comparación de la incertidumbre aleatoria y epistémica para la red VGG16.

	Aleatoric uncertainty	Epistemic uncertainty
Bayesian VGG16 (MNIST)	0.0096	0.0026
Bayesian VGG16 (CIFAR-10)	0.1920	0.0404

IV. CONCLUSIONES

Se propusieron las redes neuronales convolucionales bayesianas utilizando Bayes por retropropagación como un método de inferencia variacional fiable. Además, se compararon resultados con arquitecturas frecuentistas. En comparación, se aplicaron distribuciones de probabilidad a priori a todos los parámetros del modelo y se actualizaron según el teorema de Bayes con inferencia variacional. Se demostró que estas redes neuronales logran resultados de vanguardia, similares a los obtenidos por las mismas arquitecturas de red entrenadas mediante inferencia frecuentista. Además, se examinaron cómo se pueden calcular las incertidumbres para el método propuesto y se mostró cómo se pueden reducir las incertidumbres epistémicas con más datos de entrenamiento.

V. REFERENCIAS

Adji B. Dieng, Dustin Tran, Rajesh Ranganath, John Paisley & David M. Blei (2017). Variational inference via χ upper bound minimization. In Advances in Neural Information Processing Systems, pages 2729–2738.

Alex Graves (2011). Practical variational inference for neural networks. In Advances in Neural Information Processing Systems, pages 2348–2356.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu & Daan Wierstra (2015). Weight uncertainty in neural networks. In International Conference on Machine Learning, pages 1613–1622.

Yann LeCun and Corinna Cortes (2010). MNIST handwritten digit database.

Yarin Gal (2016). Uncertainty in Deep Learning. PhD thesis, University of Cambridge.