МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

Кафедра дифференциальных уравнений и системного анализа

НЕЙРОБАЙЕСОВСКИЕ МЕТОДЫ

Курсовая работа

Афанасенко Григория Сергеевича
студента 2-го курса
специальности 6-05-0533-08
«Компьютерная математика
и системный анализ»

Научный руководитель:
кандидат физ.-мат. наук,
доцент А. Э. Малевич

Минск, 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В современном мире развитие нейронных сетей и глубинного обучения происходит очень быстро. Регулярно появляются новые методы обучения нейронный сетей, новые архитектуры, новые слои или трюки, каждые из которых имеют свои особенные полезные эффекты и поведение. К сожалению, из-за многомерности данных, их сложной структуры и общей сложности задач обучения нейронных сетей, люди всё меньше и меньше могут объяснить, почему тот или иной трюк, слой работают. В данном случае речь не идёт об интерпретации нейронных сетей, а о поиске строгих математических доказательств и обоснований их работы. Наши открытия в этой области обгоняют нашу способность их постичь. Позднее некоторые исследования в этой области и полученные методы получили название — Нейробайесовские методы.

Получение истинного понимания принципа работы нейронных сетей позволяют усовершенствовать уже существующие методы и изобрести новые. По этой теме сделано большое количество исследований и работ на английском языке. На русском языке популяризатором этих методов стал Ветров Дмитрий Петрович, который ведёт исследования в этой области.

Целью курсовой работы будет разобрать самые базовые концепции и понятия нейробайесовских методов, собрать их в одной работе и предоставить весь материал, чтобы другие заинтересованные люди могли ознакомиться с ними на русском языке. Более подробно, в этой работе будет проведён разбор со всеми математическими выкладками ряда самых известных зарубежных статей по этой теме с дальнейшим объяснением их на более понятном языке. Как результат, курсовая работа будет собой представлять небольшой учебник по нейробайесовским методам.

Помимо вышесказанного, также будет реализован полноценный фреймворк для обучения байесовских нейронных сетей на C++ и протестирован на нескольких наборах данных.

ГЛАВА 1

Теоретические сведения.

1.1 Байесовский подход.

1.1.1 Основы байесовской статистики.

Перед тем, как приступить к сетям, вспомним основы байесовской статистики.

Байесовская статистика противопоставляется частотной статистике, как альтернатива. Основное отличие в двух подходах состоит в том, что в байесовской статистике вероятность интерпретируется, как степень уверенности в истинности суждения. Другими словами, как мера незнания или неопределённости. В частотной статистике вероятность определяется как частота события. Байесовскую вероятность ещё иногда называют «логической» вероятностью, поскольку её проще применять в реальных задачах.

Байесовский подход же к оценке параметров заключается в утверждении, что априорные знания влияют на апостериорные знания. Данное утверждение наиболее ярко видно в формуле Байеса:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} = \frac{P(D|H)P(H)}{\int_{\mathcal{H}} P(D|\widetilde{H})P(\widetilde{H})d\widetilde{H}}$$

$$P(H|D) \propto P(D|H)P(H)$$

В данной формуле,

H — некоторая гипотеза, вероятность которой мы хотим узнать, при помощи известных данных D.

$P(H|D)$ — апостериорная вероятность гипотезы после того, как мы пронаблюдали данные D.

$P(H)$ — это априорная вероятность или априорные знания о нашей гипотезе, которые мы знаем до того, как пронаблюдали данные D.

$P(D|H)$ — плотность распределения, которое называется правдоподобием наблюдаемых данных D, если гипотеза верна.

$P(D)$ — можно проинтерпретировать, как шум в данных. Особенно хорошо это заметно, когда мы записываем его в интегральной форме, где, как-бы, перебираем возможную гипотезу $\widetilde{H}$, которая должна наилучшим образом объяснить наши данные.

Нижняя запись, опуская знаменатель, который не зависит от H, обозначает пропорциональную зависимость между апостериорной вероятностью и априорной вероятностью.

Байесовские статические методы использует Теорему Байеса для вычисления и обновления вероятности после получения новых данных.

1.1.2 Байесовский подход к оценке параметров.

Теперь перейдем к задаче оценке параметров статистических(вероятностных) моделей. К таким моделям можно отнести почти все модели машинного обучения, нейронные сети, марковские цепи и др.

В общем случае обозначим за $a_\theta(x)$ — статистическую модель из параметризованного семейства $\{a_\theta(x) : \theta \in \Theta\}$, где θ — параметры модели. В случае линейной регрессии $y = w^T x + b$, $\theta = \{w, b\}$; для нейронных сетей θ — веса промежуточных слоёв; для марковских цепях θ — вероятности на рёбрах и т.д.

Когда мы занимаемся выбором модели $a_\theta(x)$, которая наилучшим образом(в некоторым смысле) описывают неизвестную закономерость в данных, то мы занимаемся подбором параметров θ. В классическом подходе мы хотим найти *точечную оценку* на параметры θ, то есть найти значения параметров $\widehat{\theta}$, при котором качество нашей статистической модели будет наилучшим, т.е. $\widehat{\theta} = \underset{\theta}{\operatorname{argmin}}\, loss_{D_y, D_x}(\theta)$. Тут важно отметить, что нас интересует единственное такое оптимальное значение, даже если их может быть несколько. В этом и заключается точечный подход к оценке параметров.

Однако такой подход ничего не говорит про устойчивость найденного решения или, на языке байесовской статистики, уровня уверенности в том, что найденные $\widehat{\theta}$ является наилучшими. Чтобы лучше понять, рассмотрим пример с линейной регрессией $y = w^T x + b$ на двух ситуациях(Рисунок 1.1, Рисунок 1.2).

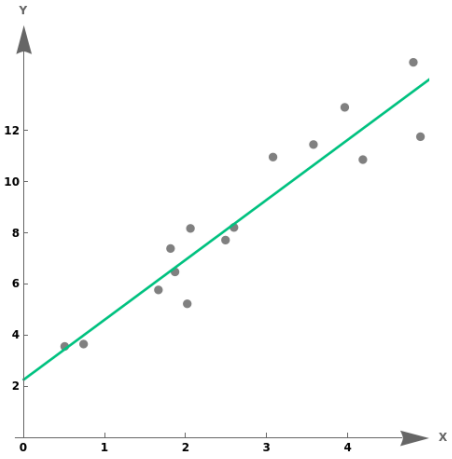

Рисунок 1.1 Уверенная модель

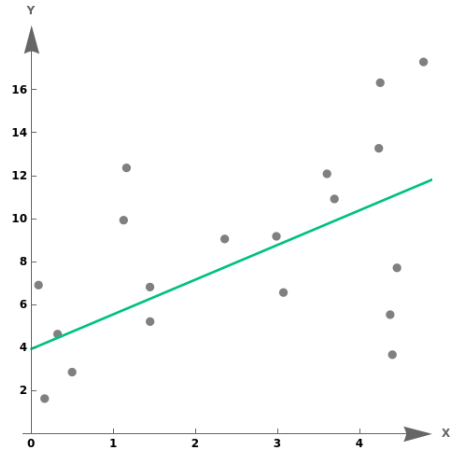

Рисунок 1.2 Неуверенная модель

И в первом, и во втором случае мы нашли оптимальную оценки $\widehat{w}, \widehat{b}$, однако во втором случае в данных сильно больше шума, что увеличвает неуверенность модели в том, что найденные оценки является наилучшими. Для того, чтобы лучше понимать ситуацию рассмотрим распределение параметров, а не их значение. Например, для примеров выше распределения параметров могли быть следующими:

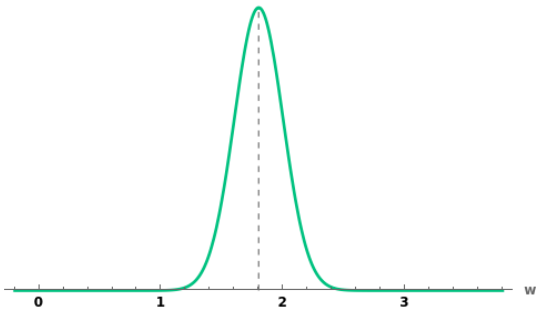

Рисунок 1.3 Уверенная модель

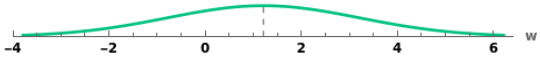

Рисунок 1.4 Неуверенная модель

Таким образом, одного лишь значения оценки параметров $\widehat{\theta}$ может быть недостаточно для того, чтобы правильно решить поставленную задачу. Гораздо больше информации даёт распределение параметров $p(\theta)$, поэтому в байесовском подходе **оценивается не сами параметры, а их распределение.**

Далее мы применим основную байесовскую парадигму и добавим априорные знания в нашу статическую модель. Вернёмся к точечной оценке параметров:

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}}\, loss_{D_y, D_x}(\theta)$$

Чаще всего в качество $loss_{D_y,D_x}(\cdot)$ предполагается брать $-\log p(D_y|D_x,\theta)$ и получаем:

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}}(-\log p(D_y|D_x,\theta)) = \underset{\theta}{\operatorname{argmax}} \log p(D_y|D_x,\theta)$$

Получившаяся оценка $\widehat{\theta}$ называется *оценкой максимума правдоподобия*(MLE). В данном случае θ рассматривается, как неизвестный, но неслучайный параметр.

Теперь будем рассматривать θ, как случайный параметр. Тогда будем искать значение θ, которое максимизирует апостериорное распределение $p(\theta|D_y, D_x)$(или его логарифм, что тоже самое):

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmax}}(\log p(\theta|D_x,D_y)) = \underset{\theta}{\operatorname{argmax}}(\log p(D_y|\theta,D_x)p(D_x|\theta)p(\theta) - \log p(D_y,D_x))$$

$$\widehat{\theta} = \Big[p(D_x|\theta) = p(D_x), \text{т.к. } D_x \perp\!\!\!\perp \theta\Big] = \underset{\theta}{\operatorname{argmax}}(\log p(D_y|\theta,D_x)p(D_x)p(\theta))$$

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmax}}(\underbrace{\log p(D_y|\theta,D_x)}_{-\mathcal{L}_{D_y,D_x}(\theta)} + \underbrace{\log p(\theta)}_{-\mathcal{R}(\theta)})$$

Таким образом мы получили *оценку апостериорного максимума*(MAP) на параметры θ. Стоит заметить, что разница между итоговым оптимизируемым функционалом в случае MLE и случае MAP заключется в добавлении функционала $\mathcal{R}(\theta)$, который выступает в роли *регуляризатора* весов.

Таким образом добавление априорных знаний или априрорного распределения параметров также добавляет естественный регуляризатор в функционал. Следовательно, просто меняя подход на байесовский наша модель становится более устойчива к переобучению.

Замечание. В будущем мы ещё вернёмся к нашему оптимизируемому функционалу, добавляя в него дополнительные слагаемые и тем самым получая новые свойства. Здесь также можно проследить интересный переход от вероятностной постановки задачи к оптимизационной, которую мы решаем с помощью градиентных методов.

1.2 Вариационный вывод.

Теперь попробуем совместить все предыдущие идеи в одну. Такое решение было предложено в [6].

Рассмотрим вероятностную модель $p(\underbrace{D_x, D_y}_{\mathcal{D}}, \theta) = p(\theta|D_x, D_y)p(D_x, D_y)$, где $D_x = (x_1, x_2, ..., x_n)$, $D_y = (y_1, y_2, ..., y_n)$, $n \gg 1$, $\theta \in \mathbb{R}^d$

Для получения апостериорного распределения на параметры θ воспользуемся формулой Байеса:

$$p(\theta|D_x, D_y) = \frac{p(D_x, D_y|\theta)p(\theta)}{p(D_x, D_y)} = \frac{p(D_y|\theta, D_x)p(\theta)}{p(D_x, D_y)} + const$$

$$p(\theta|D_x, D_y) = \frac{\sum_{i=1}^{n} p(y_i|\theta, x_i)p(\theta)}{\int p(D_x, D_y|\widetilde{\theta})p(\widetilde{\theta})d\widetilde{\theta}} + const$$

Подсчёт истинного апостериора таким способом требует взятие интеграла в знаменателе. В случаях, когда это возможно, проблем нет. Однако в большинстве случаев, которые используются на практике и которые нас интересуют, такой интеграл не берётся.

1.2.1 KL-дивергенция и вариационная нижняя оценка.

Для решения проблемы будем пробовать приблизить апостериорное распределение $p(\theta|D_x, D_y)$ параметризованным распределением $q(\theta|\varphi)$ путём минимизации дивергенции Кульбака-Лейблера $KL(q(\theta|\varphi) \,||\, p(\theta|D_x, D_y))$. То есть

$$\widehat{\varphi} = \underset{\varphi}{\operatorname{argmin}} KL(q(\theta|\varphi) \,||\, p(\theta|D_x, D_y))$$

Чем плоха такая оптимизационная задача? Для того, чтобы считать значение самой функции и её градиента, нам бы потребовалось уметь считать значение $p(\theta|D_x, D_y)$, которое мы и пытаемся найти. Если бы могли его считать, то и смысла в нашей задаче не было бы. Поэтому требуется найти альтернативный оптимизируемый функционал.

Распишем дивергенцию:

$$KL(q(\theta|\varphi) \;||\; p(\theta|D_x, D_y))) = \int q(\theta|\varphi) \log \frac{q(\theta|\varphi)}{p(\theta|D_x, D_y)} d\theta = \int q(\theta|\varphi) \log q(\theta|\varphi) -$$

$$- \int q(\theta|\varphi) \log p(\theta|D_x, D_y) = \mathbb{E}_{\theta \sim q(\theta|\varphi)}[\log q(\theta|\varphi)] - \mathbb{E}_{\theta \sim q(\theta|\varphi)}[\log p(\theta|D_x, D_y)] =$$

$$= \underbrace{\mathbb{E}_{\theta \sim q(\theta|\varphi)}[\log q(\theta|\varphi)] - \mathbb{E}_{\theta \sim q(\theta|\varphi)}[\log p(\theta, D_x, D_y)]}_{-ELBO(\varphi, \mathcal{D})} + \underbrace{\log p(D_x, D_y)}_{const}$$

Тогда получаем следующее:

$$KL(q(\theta|\varphi) \;||\; p(\theta|D_x, D_y)) + ELBO(\varphi, \mathcal{D}) = \log p(D_x, D_y)$$

Получившееся значение $ELBO(\varphi, \mathcal{D})$ называется *вариационной нижней оценкой* или *evidence lower bound*. Оно также записывается, как $L(\varphi, \mathcal{D})$.

Свойства вариационной нижней оценки:

- $\log p(D_x, D_y) \geq L(\varphi, \mathcal{D}) = \mathbb{E}_{\theta \sim q(\theta|\varphi)}[\frac{\log p(\theta, D_x, D_y)}{q(\theta|\varphi)}]$, т.к. дивергенция Кульбака-Лейблера неотрицательна. Данное неравенство верно для любого распределения q.

- $L(\varphi, \mathcal{D}) \leq 0$, т.к. $\log p(D_x, D_y) \leq 0$. Т.е. вариационная нижняя оценка всегда неположительна.

Следовательно, задача о минимизации дивергенции Кульбака-Лейблера по параметрам φ эквивалентна задаче о максимизации вариационной нижней оценки по тем же параметрам φ. Этим фактом мы можем пользоваться, чтобы перейти от предыдущей оптимизационной задаче, которая требовала подсчёта апостериорной вероятности $p(\theta \mid D_x, D_y)$, к задаче, где этого не требуется. Тем самым это упрощает нашу оптимизационную задачу:

$$\widehat{\varphi} = \underset{\varphi}{\operatorname{argmin}}\, KL(q(\theta|\varphi) \;||\; p(\theta|D_x, D_y)) = \underset{\varphi}{\operatorname{argmax}}\, L(\varphi, \mathcal{D})$$

$$\widehat{\varphi} = \underset{\varphi}{\operatorname{argmin}} - L(\varphi, \mathcal{D})$$

Распишем $-L(\varphi, \mathcal{D})$:

$$-L(\varphi, \mathcal{D}) = \mathbb{E}_{\theta \sim q(\theta|\varphi)}[\log q(\theta|\varphi)] - \mathbb{E}_{\theta \sim q(\theta|\varphi)}[\log p(D_y|\theta, D_x)] - \mathbb{E}_{\theta \sim q(\theta|\varphi)}[p(\theta)] - \log p(D_x)$$

Теперь попробуем провести аналогию между оптимизационной задачей в предыдущем пункте и новой. Это будет лишь приблизительная аналогия с теоретической точки зрения, однако на практике её всегда можно считать истинной:

$$\underbrace{\mathbb{E}_{\theta\sim q(\theta|\varphi)}[\log q(\theta|\varphi)]}_{-\mathcal{H}(\varphi)} - \underbrace{\mathbb{E}_{\theta\sim q(\theta|\varphi)}[\log p(D_y|\theta, D_x)]}_{-\mathcal{L}_{D_x,D_y}(\varphi)} - \underbrace{\mathbb{E}_{\theta\sim q(\theta|\varphi)}[\log p(\theta)]}_{-\mathcal{R}(\varphi)} - \underbrace{\log p(D_x)}_{-noise}$$

Тогда наша функция потерь переписывается в виде:

$$-L(\varphi, \mathcal{D}) = \mathcal{L}_{D_x,D_y}(\varphi) + \mathcal{R}(\varphi) + noise - \mathcal{H}(\varphi),$$

где

$\mathcal{L}_{D_x,D_y}(\varphi)$ — функция ошибки предсказания модели.

$\mathcal{R}(\varphi)$ — естественный регуляризатор, получившийся от передачи априорных знаний в модель.

$\mathcal{H}(\varphi)$ — энтропия Шеннона параметрического распределения $q(\theta|\varphi)$. Чем выше этот параметр, тем более "хаотичным"можно считать получившееся распредление. Вместе с тем уровень энтропии показывает уровень уверенности в значение параметров нашей модели. Например, значение энтропии Шеннона δ-распределения равно 0.

$noise$ — шум в данных, от которого нам не избавиться, но он является константой, поэтому из оптимизируемого функционала его можно исключить.

1.2.2 Стохастический вариационный вывод.

Для решения оптимизационной задачи можно использовать различные методы: методы 1-го порядка(градиентные методы), методы второго порядка(метод Ньютона), генетические и эволюционные алгоритмы. Поскольку мы работаем с большими объёмами данных, как по количеству самих данных, так и по количеству признаков, то самыми эффективными будут градиентные методы.

Тогда посчитаем градиент по φ:

$$\nabla_\varphi(-L(\varphi, \mathcal{D})) = \nabla_\varphi\mathcal{L}_{D_x,D_y}(\varphi) + \nabla_\varphi\mathcal{R}(\varphi) - \nabla_\varphi\mathcal{H}(\varphi) =$$

$$= -\nabla_\varphi\mathbb{E}_{q(\theta|\varphi)}[\log p(D_y|\theta, D_x)] - \nabla_\varphi\mathbb{E}_{q(\theta|\varphi)}[\log p(\theta)] + \nabla_\varphi\mathbb{E}_{q(\theta|\varphi)}[\log q(\theta|\varphi)]$$

Вернёмся к интегральной форме:

$$-\nabla_\varphi \int q(\theta|\varphi) \log p(D_y|\theta, D_x) d\theta - \nabla_\varphi \int q(\theta|\varphi) \log p(\theta) d\theta + \nabla_\varphi \int q(\theta|\varphi) \log q(\theta|\varphi) d\theta$$

Нам было бы удобно, если бы функция плотности $q(\theta|\varphi)$ не зависила от параметров, поскольку тогда мы могли бы поднести знак дифференцирования под символ матожидания. Однако пока что мы так делать не можем.

Вместо этого попробуем внести дифференцирование под знак интеграла. Это возможно при соблюдении определённых *условий регулярности*, которые мы накладываем на статистическую модель. Такие модели также называются *регулярными* [10]. Сначал рассмотрим последний интграл:

$$\int \frac{\partial}{\partial \varphi}(q(\theta|\varphi) \log q(\theta|\varphi)) d\theta = \int \frac{\partial}{\partial \varphi} q(\theta|\varphi) \log q(\theta|\varphi) d\theta + \int q(\theta|\varphi) \frac{\partial}{\partial \varphi} \log q(\theta|\varphi) d\theta =$$

$$= \int \frac{\partial}{\partial \varphi} q(\theta|\varphi) \log q(\theta|\varphi) d\theta \; + \; \int \frac{\partial}{\partial \varphi} q(\theta|\varphi) d\theta = \int \frac{\partial}{\partial \varphi} q(\theta|\varphi) \log q(\theta|\varphi) d\theta =$$

$$= \int q(\theta|\varphi) \frac{1}{q(\theta|\varphi)} \frac{\partial}{\partial \varphi} q(\theta|\varphi) \log q(\theta|\varphi) d\theta = \int q(\theta|\varphi) \frac{\partial(\log q(\theta|\varphi))}{\partial \varphi} \log q(\theta|\varphi) d\theta =$$

$$= \mathbb{E}_{\theta \sim q(\theta|\varphi)} \Big[\frac{\partial(\log q(\theta|\varphi))}{\partial \varphi} \log q(\theta|\varphi) \Big]$$

Таким образом мы смогли занести $\frac{\partial}{\partial \varphi}$ под матожидание. В последних переходах использовался *log-derivative trick*, который позволяет перейти к производной от логарифма плотности, оставив при этом, матожидание. Аналогично поступим для двух остальных интегралов:

$$- \int \frac{\partial}{\partial \varphi} q(\theta|\varphi)(\log p(D_y|\varphi, D_x) + \log p(\theta)) d\theta =$$

$$= - \int q(\theta|\varphi) \frac{\partial(\log q(\theta|\varphi))}{\partial \varphi} (\log p(D_y|\theta, D_x) + \log p(\theta)) d\theta$$

Тогда итоговый градиент функции потерь будет равен:

$$\frac{\partial}{\partial \varphi}(-L(\varphi, \mathcal{D})) = \int q(\theta|\varphi) \frac{\partial \log q(\theta|\varphi)}{\partial \varphi} (\log q(\theta|\varphi) - \log p(D_y|\theta, D_x) - \log p(\theta)) d\theta$$

Этот интеграл мы не можем посчитать аналитически, поэтому воспользуемся

методом Монте-Карло для получения приближенной оценки интеграла [8]:

$$\theta_k \sim q(\theta|\varphi),\ k = \overline{1...K}$$

$$\text{grad}(-L(\varphi, \mathcal{D})) \approx \frac{1}{K}\sum_{k=1}^{K}\frac{\partial(\log q(\theta_k|\varphi))}{\partial\varphi}(\log q(\theta_k|\varphi) - \log p(D_y|\theta_k, D_x) - \log p(\theta_k))$$

Однако на практике время сэмплирования θ_k занимает достаточно много времени, поэтому часто берут $K = 1$. Для больших данных мы часто пользуются стохастическим градиентом:

$$j \sim U[1...N]$$

$$\theta_k \sim q(\theta|\varphi),\ k = \overline{1...K}$$

$$\text{grad}(-L(\varphi)) \approx \frac{1}{K}\sum_{k=1}^{K}\frac{\partial(\log q(\theta_k|\varphi))}{\partial\varphi}(\log q(\theta_k|\varphi) - N\log p(y_j|\theta_k, x_j) - \log p(\theta_k))$$

Также для нейронных сетей используется пакетный градиентный спуск, чтобы мы могли обрабатывать данные небольшими пакетами. Такой подход уменьшает дисперсию:

$$i_b \sim U[1...N],\ b = \overline{1...B}$$

$$\theta_k \sim q(\theta|\varphi),\ k = \overline{1...K}$$

$$-\frac{dL}{\partial\varphi} \approx \frac{1}{K}\sum_{k=1}^{K}\Big[\frac{\partial(\log q(\theta_k|\varphi))}{\partial\varphi}\big(\log q(\theta_k|\varphi) - \frac{N}{B}\sum_{b=1}^{B}(\log p(y_{i_b}|\theta_k, x_{i_b})) - \log p(\theta_k)\big)\Big]$$

Однако данный метод перестаёт работать при $N \gg 1$ и $|\varphi| \gg 1$ из-за очень большой дисперсии градиента. Поскольку $\mathbb{E}_{\theta\sim q(\theta|\varphi)}\left[\frac{\partial(\log q(\theta|\varphi))}{\partial\varphi}\right] = 0$, то в после сэмплирования θ_k мы будем получать числа разных знаков. При этом второй множитель из-за домножения на $\frac{N}{B}$ будет очень большим по норме, и из-за этого дисперсия будет очень большого порядка с постоянно меняющимся знаком.

Из-за этого при больших данных данный метод неприменим и нужно искать ему замену. Оригинально данный алгоритм был назван REINFORCE, который являются разновидностью policy-gradient алгоритмов, и был придуман в обучении с подкреплением [9].

1.2.3 Трюк с репараметризацией.

На замену этому методу был придуман метод, который называется *трюк с репараметризацией* (*reparametrization trick*) [5]. Предварительно запишем несколько утверждений утверждений:

Утверждение 1.2.1. *(Law of the unconscious statistician) Пусть z - случайный вектор, $z \sim p(z)$. При этом существует случайный вектор $\varepsilon \sim r(\varepsilon)$ и существует диффеоморфизм g: $z = g(\varepsilon)$ такой, что $g(\varepsilon) \sim p(z)$. Тогда*

$$\forall f - \text{ измеримая функция случ. величины}$$

$$\mathbb{E}_{p(z)}[f(z)] = \mathbb{E}_{r(\varepsilon)}[f(g(\varepsilon))]$$

$$\int p(z)f(z)dz = \int r(\varepsilon)f(g(\varepsilon))d\varepsilon$$

Таким образом данное утверждение позволяет заменять одну сэмплируемую величину на другую, сохраняя матожидание. Это свойство применимо для нашей предыдущей опимизационной задаче:

$$-L(\varphi) = \mathbb{E}_{q(\theta|\varphi)}[\log q(\theta|\varphi)] - \mathbb{E}_{q(\theta|\varphi)}[\log p(D_y|\theta, D_x)] - \mathbb{E}_{q(\theta|\varphi)}[\log p(\theta)]$$

Предположим, что мы можем некоторым способом репараметризовать нашу модель. То есть найти диффеоморфизм g: $\theta = g(\varepsilon, \varphi)$, $\varepsilon \sim r(\varepsilon)$. При этом $g(\varepsilon, \varphi) \sim q(\theta|\varphi)$. Тогда, используя 1.2.1 мы можем переписать функцию потерь:

$$-L(\varphi) = \mathbb{E}_{r(\varepsilon)}[\log q(g(\varepsilon, \varphi)|\varphi)] - \mathbb{E}_{r(\varepsilon)}[\log p(D_y|g(\varepsilon, \varphi), D_x)] - \mathbb{E}_{r(\varepsilon)}[\log p(g(\varepsilon, \varphi))]$$

В полученной формуле мы добились того, чего хотели раньше — плотность больше не зависит от оптимизируемых параметров φ, и следовательно мы можем занести производную под символ матожидания без ошибки:

$$\frac{\partial}{\partial\varphi}\mathbb{E}_{r(\varepsilon)}[\log q(g(\varepsilon, \varphi)|\varphi)] - \frac{\partial}{\partial\varphi}\mathbb{E}_{r(\varepsilon)}[\log p(D_y|g(\varepsilon, \varphi), D_x)] - \frac{\partial}{\partial\varphi}\mathbb{E}_{r}[\log p(g(\varepsilon, \varphi))] =$$

$$= \mathbb{E}_{r(\varepsilon)}\big[\frac{\partial}{\partial\varphi}\log q(g(\varepsilon, \varphi)|\varphi)\big] - \mathbb{E}_{r(\varepsilon)}\big[\frac{\partial}{\partial\varphi}\log p(D_y|g(\varepsilon, \varphi), D_x)\big] - \mathbb{E}_{r(\varepsilon)}\big[\frac{\partial}{\partial\varphi}\log p(g(\varepsilon, \varphi))\big]$$

Теперь для получения приближения градиента мы используем метод Монте-

Карло для оценки матожиданий:

$$\varepsilon_k \sim r(\varepsilon),\ k = \overline{1...K}$$

$$\theta_k = g(\varepsilon_k, \varphi)$$

$$\frac{\partial(-L(\varphi))}{\partial\varphi} \approx \frac{1}{K}\sum_{k=1}^{K}\frac{\partial}{\partial\varphi}\log q(\theta_k|\varphi) - \frac{\partial}{\partial\varphi}\log p(D_y|\theta_k, D_x) - \frac{\partial}{\partial\varphi}\log p(\theta_k)$$

Аналогично мы можем написать стохастическую оценку на градиент:

$$\varepsilon_k \sim r(\varepsilon),\ k = \overline{1...K}$$

$$\theta_k = g(\varepsilon_k, \varphi)$$

$$j \sim U[1...N]$$

$$\frac{\partial(-L(\varphi))}{\partial\varphi} \approx \frac{1}{K}\sum_{k=1}^{K}\frac{\partial}{\partial\varphi}\log q(\theta_k|\varphi) - N\frac{\partial}{\partial\varphi}\log p(y_j|\theta_k, x_j) - \frac{\partial}{\partial\varphi}\log p(\theta_k)$$

И пакетный градиент, который чаще используется при обучении моделей для работы с большими данными:

$$\varepsilon_k \sim r(\varepsilon),\ k = \overline{1...K}$$

$$\theta_k = g(\varepsilon_k, \varphi)$$

$$i_b \sim U[1...N],\ b = \overline{1...B}$$

$$\frac{\partial(-L(\varphi))}{\partial\varphi} \approx \frac{1}{K}\sum_{k=1}^{K}\Big(\frac{\partial}{\partial\varphi}\log q(\theta_k|\varphi) - \frac{N}{B}\sum_{b=1}^{B}\frac{\partial}{\partial\varphi}\log p(y_{i_b}|\theta_k, x_{i_b}) - \frac{\partial}{\partial\varphi}\log p(\theta_k)\Big)$$

1.3 Модификации стохастического вариационного вывода.

1.3.1 Эмпирический Байес.

Байесовский вывод требует от аналитика очень аккуратного выбора предположений для хорошего моделирования. Как ранее было сказано, одним из ключевых отличий байесовского подхода является добавление априорных знаний. Выбор правильного априорного распределения является важным шагом,

поскольку оно должно давать корректное приближение данных. Неверное априорное распредление будет давать ложную информацию, как для аналитика, так и для статистической модели.

Однако, задача выбора правильного априорного распределения является невероятно сложной и в основном зависит от опыта и знаний аналитика.

Эмпирический Байес — это метод, который позволяет частично уменьшить эту проблему по средствам выбора априрорного распредления напрямую из данных. С одной стороны это противоречиво, поскольку априрорное распредление должно быть выбрано до наблюдаемых данных, что говорит само название. С другой стороны такой способ выбор априорного распределения позволяет находить более полезные и практичные распределения.

Рассмотрим применение эмпирического Байеса для стохастического вариационного вывода.

Теперь будет предполагать, что априорное распредление на параметры $p(\theta)$ также является параметризованным $p_\xi(\theta)$ с параметрами ξ. Запишем новую функцию потерь:

$$Loss(\varphi, \xi) = \mathbb{E}_{q(\theta|\varphi)}[\log q(\theta|\varphi)] - \mathbb{E}_{q(\theta|\varphi)}[\log p(D_y|\theta, D_x)] - \mathbb{E}_{q(\theta|\varphi)}[\log p_\xi(\theta)]$$

Будем оптимизировать параметры ξ с помощью градиентного спуска. Запишем сразу формулы для градиента по φ и по ξ:

$$\frac{\partial(Loss(\varphi, \xi))}{\partial\varphi} \approx \frac{1}{K}\sum_{k=1}^{K}\Big(\frac{\partial}{\partial\varphi}\log q(\theta_k|\varphi) - \frac{N}{B}\sum_{b=1}^{B}\frac{\partial}{\partial\varphi}\log p(y_{i_b}|\theta_k, x_{i_b}) - \frac{\partial}{\partial\varphi}\log p_\xi(\theta_k)\Big)$$

$$\frac{\partial(Loss(\varphi, \xi))}{\partial\xi} \approx \frac{1}{K}\sum_{k=1}^{K}\Big(-\frac{\partial}{\partial\xi}\log p_\xi(\theta_k)\Big)$$

Замечание. В данном алгоритме параметры ξ оцениваются точечно, то есть мы не задаём распределения на параметры, как в случае с параметрами θ.

1.4 Свойства стохастического градиента вариационного вывода.

1.4.1 Несмещённость стохастического градиента.

Теперь рассмотрим вероятностно-статистические свойства стохастического градиента вариационного вывода [4]. Наиболее пристальное внимание уделим дисперсии и матожиданию, поскольку они играют наиболее важные роли. Сосредоточимся на функции ошибки модели $\mathcal{L}_{D_x,D_y}(\varphi) = \mathbb{E}_{\theta\sim q(\theta|\varphi)}[\log p(D_y|D_x,\theta)]$.

В предыдущей главе была получена несмещённая дифференцируемая Монте-Карло оценка с помощью мини пакетов для функции ошибки модели через репараметризацию:

$$\mathcal{L}_{D_x,D_y}(\varphi) \simeq \widehat{\mathcal{L}}^{RT}_{D_x,D_y}(\varphi) = \frac{N}{B}\sum_{b=1}^{B}\log p(y_{i_b}|x_{i_b},\theta = g(\varphi,\varepsilon))$$

Данная оценка является несмещённой в силу несмещённости самой Монте-Карло оценки, а также дифференцируемой, следовательно градиент этой оценки также будет являться несмещённой оценкой градиента $\frac{\partial}{\partial\varphi}\mathcal{L}_{D_x,D_y}(\varphi)$:

$$\mathbb{E}_{i_b}[\frac{\partial}{\partial\varphi}\widehat{\mathcal{L}}^{RT}_{D_x,D_y}\varphi] = \frac{\partial}{\partial\varphi}\mathcal{L}_{D_x,D_y}(\varphi)$$

1.4.2 Дисперсия стохастического градиента.

Теория стохастической аппроксимации доказывает, что стохастический градиентный спуск асимптотически сходится к локальному минимуму при правильном выборе шага. Однако, несмотря на это, на практике ключевое влияние на производительность(скорость сходимости, найденные оптимум) оказывает дисперсия стохастического градиента. Если дисперсия градиента слишком большая, то стохастический градиентный спуск не сделает значимых улучшений за разумное время, из-за чего он будет слишком долго сходится.

Введём дополнительное обозначение:

$$\mathcal{L}_i = \log p(y_i|x_i,\theta = g(\varphi,\varepsilon))$$

Тогда мы можем переписать нашу последнюю оценка:

$$\widehat{\mathcal{L}}^{RT}_{D_x,D_y}(\varphi) = \frac{N}{B}\sum_{i=1}^{B}\mathcal{L}_i$$

Посчитаем дисперсию этой оценки:

$$\mathbb{D}[\widehat{\mathcal{L}}^{RT}_{D_x,D_y}(\varphi)] = \mathbb{D}\big[\frac{N}{B}\sum_{i=1}^{B}\mathcal{L}_i\big] = \frac{N^2}{B^2}\Big(\sum_{i=1}^{B}\mathbb{D}[\mathcal{L}_i] + \sum_{i,j=1,i\neq j}^{B}\operatorname{cov}[\mathcal{L}_i,\mathcal{L}_j]\Big) =$$

$$= \frac{N^2}{B}\Big(\mathbb{D}[\mathcal{L}_i] + (B-1)\operatorname{cov}[\mathcal{L}_i,\mathcal{L}_j]\Big) = N^2\Big(\frac{1}{B}\mathbb{D}[\mathcal{L}_i] + \frac{B-1}{B}\operatorname{cov}[\mathcal{L}_i,\mathcal{L}_j]\Big)$$

Теперь проведём анализ полученной формулы. Заметим, что с возрастанием размера пакета B уменьшается вклад отдельной дисперсии $\mathbb{D}[\mathcal{L}_i]$ в общую дисперсию оценки. Вместе с тем вклад ковариаций не уменьшается с ростом B, поскольку коэффициент ковариации $\frac{B-1}{B}$ стремится к 1. Таким образом общая ковариация не уменьшается с ростом B, хотя именно такой эффект ожидается от пакетной оптимизации.

Таким образом, требуется попробовать придумать другую оценку, которая бы уменьшила ковариацию для разных примеров почти до нуля.

1.5 Полезные применения нейробайесовского подхода.

В этом пункте мы рассмотрим, как описанные ранее формулы могут помочь в рассмотрении с другой точки зрения, и объяснении работоспособности других моделей, слоёв, которые ранее не имели достаточно строгих доказательств своей работы. Этот пункт, в некотором роде, даёт обоснование того, почему стоит заниматься нейробайесовскими методами, и почему они не бесполезны в современном мире.

1.5.1 Слой Dropout.

Как известно использование `Dropout` слоя позволяет бороться с переобучением(*overfitting*) глубоких нейронных сетей. Однако возникает вопрос, как именно этот слой влияет на переобучение и возможно ли его улучшить? Ответ на по-

следний вопрос даёт [4].

Вспомним, как выглядит наш стохастический градиент функции ошибки для случая, когда $B = 1$:

$$j \sim U[1...N]$$
$$\text{stoch grad}\,\mathcal{L}_{D_x,D_y} = N\frac{\partial}{\partial W_{i,l}}\log p(y_j|x_j, W)$$

В данной формуле я выделил матрицу весов какого-нибудь слоя нейронной сети, из которого мы хотим повыбрасывать веса для регуляризации. Опишем процесс `Dropout` аналогичным способом:

$$\delta_{i,l} \sim Bernoulli[p] = r(\delta)$$

$$j \sim U[1...N]$$
$$\text{stoch grad}\,\mathcal{L}_{D_x,D_y} = N\frac{\partial}{\partial W_{i,l}}\log p(y_j|x_j, W \circ \delta)$$

Как видно, эти формулы очень напоминают формулы полученные в 1.2.2. Значит мы можем перейти от стохастического градиента к обычному, проведя обратную операцию той, которая была в 1.2.2:

$$\text{grad}\,\mathcal{L}_{D_x,D_y} = \sum_{j=1}^{N}\int r(\delta)\frac{\partial}{\partial W_{i,l}}\log p(y_j|x_j, W \circ \delta)d\delta$$

$$\text{grad}\,\mathcal{L}_{D_x,D_y} = \frac{\partial}{\partial W_{i,l}}\sum_{j=1}^{N}\int r(\delta)\log p(y_j|x_j, W \circ \delta)d\delta$$

Теперь становится понятно, что функционал $\mathcal{L}_{D_x,D_y}$ выглядит:

$$\mathcal{L}_{D_x,D_y} = \sum_{j=1}^{N}\int r(\delta)\log p(y_j|x_j, W \circ \delta)d\delta$$

Таким образом, теперь мы можем утверждать, что добавление `Dropout` даёт эффект зашумление функции ошибки. То есть этот интеграл придаёт шума модели в соответствии с функцией плотности $r(\theta)$.

Однако можно попробовать заменить распределение Бернулли на другое распределение. Действительно, ведь это распределение указывает на источник

и тип шума и никто не мешает взять его другим. Для того, чтобы понять какое распределение стоят взять, давайте рассмотрим ещё одну особенность слоя `Dropout`. Рассмотрим обычный линейный слой с `Dropout`:

$$\delta_i \sim Bernoulli[p]$$

$$y = (x \cdot W^T) \circ \delta$$

$$y_i = \sum_{k=1}^{D} x_k \cdot W_{i,k} \cdot delta_k$$

То есть каждое y_i есть сумма D независимых одинаково распределённых случайных величин, то есть можно применить *Центральную Предельную Теорему* и сказать, что y_i будет распределено нормально. В данном случае распределение на y_i возникает из того факта, что мы пропускаем сквозь наш слой очень большое количество данных. Таким образом y_i будут иметь *примерно* нормальные распределения. Зная этот факт можно в качестве шума брать гауссовский шум:

$$\delta_i \sim \mathcal{N}(1, \alpha)$$

Именно такой подход рассмотрен в статье, упомянутой выше.

Стоит заметить, что мы смогли прийти ко всем этим выводом лишь после того, как прошли полный путь от интегралов до формулы градиента в предыдущих пунктах, чтобы теперь пойти в обратную сторону, и быстро найти ответ. Вероятно, что другого более простого способа строго объяснить работоспособность `Dropout` может и не быть. Таким образом это показывает небесполезность нейробайесовских методов.

Дальнейшее развитие нейробайесовских методов может дать понимание ещё большего числа эффектов обучение и работы нейронных сетей.

ГЛАВА 2

Виды нейронных сетей.

2.1 Детерминированные нейронные сети.

Сначала напомним, что такое обычные(детерминированные) нейронные сети и как они обучаются.

Основная задача обычных искусственных нейронных сетей(ANN) в том, чтобы аппроксимировать некоторую зависимость выхода y от входа x: $y = \Phi(x)$. Зависимость $\Phi(x)$ аппроксимируем через композицию последовательных преобразований.

Для простоты будем рассматривать обычные *полносвязные* сети со входом x, скрытыми(промежуточными) состояниями слоёв $\boldsymbol{h_i}$, функциями активации $a_i(\cdot)$ и выходом y:

$$\boldsymbol{h_0} = \boldsymbol{x}$$

$$\boldsymbol{h_i} = a_i(\boldsymbol{W_i} \cdot h_{i-1} + \boldsymbol{b_i}), i = \overline{1...n}$$

$$\boldsymbol{h_n} = \widehat{y}$$

$$L = \mathcal{L}(\widehat{y}, y),$$

где $\mathcal{L}(\cdot, \cdot)$ - функция ошибки.

Обозначим параметры модели на i-ом слое $\boldsymbol{\theta_i} = (\boldsymbol{W_i}, \boldsymbol{b_i})$, а параметры всей модели через $\boldsymbol{\Theta} = \{\boldsymbol{\theta_i} : \ i \ = \ \overline{1...n}\}$. Чаще всего нейронные сети принято рассматривать, как вычислительный граф/граф вычислений. Такой подход удобен с инженерной точки зрения, поскольку позволяет воспользоваться инструментом автоматического дифференцирования, и используется во всех современных фреймоворках: PyTorch, TensorFlow и прочие. Граф вычислений является ациклическим ориентированным графом, составленным из вершин-переменных и вершин-операций(Рисунок 2.1).

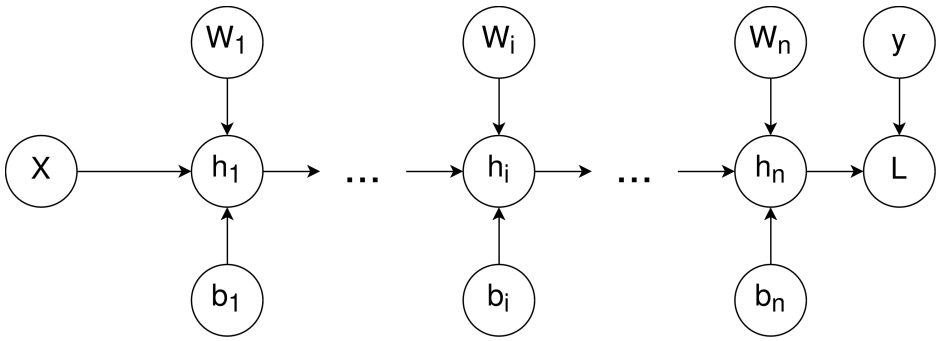

Рисунок 2.1 *Полносвязная* сеть в виде графа вычислений

Далее будем называть модели, основанные на графах вычислений, — графовыми моделями. Графы вычислений могут разных типов: статическими/динамическими, детерминированными/вероятностными и т.д. Для обучения/настройки параметров детерминированных графовых моделей используется метод *обратного распространения ошибки(back propagation)*, который широко используется в современном мире. Вкратце напомним алгоритм:

После прямого выполнения графа(*forward pass*), то есть в соответствии с направлениями рёбер на выходе мы получаем L -значение функции ошибки, которые в зависимости от задач мы хотим либо минимизировать, либо максимизировать. Для этого мы пользуемся градиентными методами оптимизации, что требует вычисление градиентов $\frac{dL}{dW_i}, \frac{dL}{db_i}$ по нашим параметрам модели, где $i = \overline{1,n}$. В общем случае это трудная задача, однако в случае детерминированных графовых моделей мы можем использовать цепное правило(*chain rule*) для того, чтобы последовательно проталкивать градиенты, начиная с концевой вершины, содержащей L.

Например, для подсчёта градиентов $\frac{dL}{dW_n}, \frac{dL}{db_n}$ мы представим его в виде

$$\frac{dL}{dW_n} = \frac{dL}{dh_n} \cdot \frac{dh_n}{dW_n}$$

$$\frac{dL}{db_n} = \frac{dL}{dh_n} \cdot \frac{dh_n}{db_n}$$

Аналогично для всех остальных параметров модели мы будем проталкивать накопленный с концевой вершины градиент до соответствующих вершин и с помощью этого градиента высчитывать градиент по параметрам модели. Схему работы алгоритма обратного распространения ошибки можно увидеть на Рису-

нок 2.2.

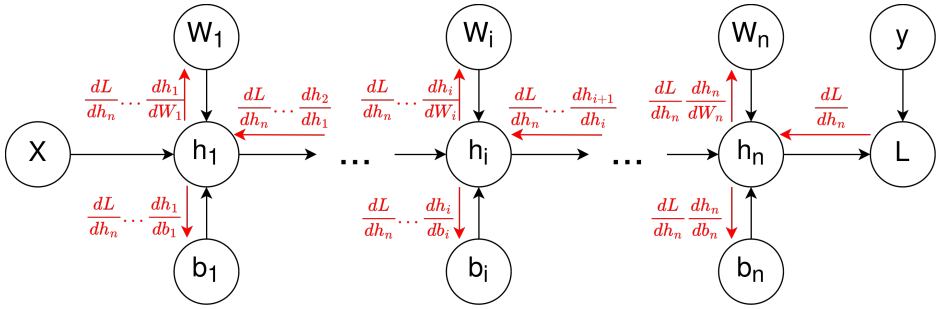

Рисунок 2.2 Обратное распространение ошибки по графу вычислений детерминированной полносвязной сети

Однако детерминированные нейронные сети обладают несколькими проблемами:

- Переобучение.
- Низкая интерпретируемость.
- Завышенная/заниженная уверенность модели в предсказаниях, даже если они неверные.
- Низкий уровень откалиброванности модели.
- Избыток параметров.

Указанные проблемы попытаемся решить с помощью байесовского подхода к нейронным сетям, который рассмотрим далее.

2.2 Байесовские нейронные сети.

2.2.1 Вероятностные графы вычислений.

Перед тем, как приступить к байесовским нейронным сетям, рассмотрим *вероятностные графы вычислений*, на которых основаны байесовские сети. В литературе также часто вместо названия *вероятностные графы вычислений* встречается *вероятностные графические модели*. Второе название является более общим, в то время как первое более специфично именно для байесовских нейронных сетей. Такие графы вычислений широко используются и известны

достаточно давно. Они лежат в основе, например, Марковских цепей, которые ранее активно использовались в различных задачах машинного предсказания, распознавания образов и т.п.

Основная мотивация в использовании вероятностного подхода состоит в том, что в реальном мире мы чаще имеем дело с неопределённостью в данных и знаниях и не можем детерминированно описать все приходящие переменные для решения задачи. Для решения проблем с неопределённостью можно попробовать собрать большие объёмы данных для того, чтобы попытаться "понять"эту неопределённость. С другой стороны мы можем использовать байесовский подход, который напрямую оперирует с неопределённостью.

Рассмотрим структуру *вероятностных графовых моделей*. В отличие от детерминированных моделей в граф добавляются вершины со случайными переменными. Таким образом в нашем совместно существуют детерминированные вершины и случайные (Рисунок 2.3). Стоит отметить, что после вступления в контакт детерминированных переменных и случайных, весь дальнейший результат будет случайным. При работе с такими моделями нужно различать *наблюдаемые* и *скрытые/латентные* переменные. Различия в этих двух понятиях естественны: в реальной жизни у нас есть некоторые известные данные и те, которые мы не может измерить явно, а лишь вычислить в результате работы модели.

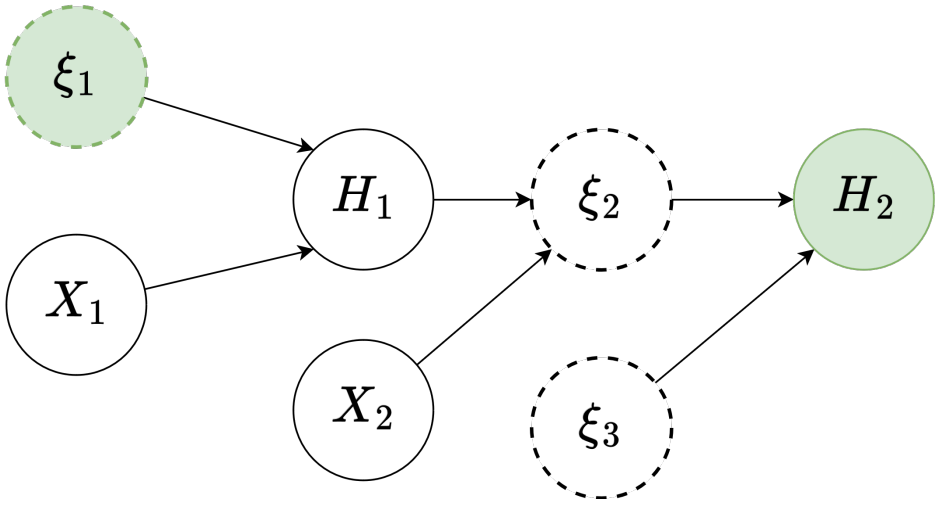

Рисунок 2.3 Вероятностная графическая модель. Здесь круги с пунктирной границей являются сэмплируемыми случайными величинами. Зелёным цветом обозначены наблюдаемые случайные переменные.

Стоит сделать замечание, что детерминированные переменные также можно представить, как случайные величины с δ-функцией плотности распределения $\delta(\cdot)$, где $\delta(\cdot)$ — δ-функция Дирака. Данный факт позволяет рассматривать все вершины в вероятностной графовой модели, как случайные.

Введём более строгое определение. Пусть $(x_1, x_2, ..., x_n)$ - множество случайных величин, представляющих вершины ориентированного графа. Тогда *вероятностная графическая модель* — это семейство условных распределений $p(x_1|...)$, $p(x_2|...)$ и т.д. над данными случайными величинами $x_1, x_2, x_3, ..., x_n$.

В случае графовых моделей каждая случайная величина x_i зависит не от всех других случайных величин, а лишь от некоторого множество её предков $ancestors(x_i)$. Таким образом мы можем вычислить полную условную плотность величины x_i так:

$$p(x_i|x_n, x_{n-1}, ...x_1) = p(x_i|ancestors(x_i))$$

Используя *цепное правило* для совместного распределения $p(x_1, x_2, ..., x_n)$ мы можем расписать его через частные распределения и условные:

$$p(x_1, x_2, ..., x_n) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)...p(x_n|x_{n-1}, ..., x_1)$$

Выбирая порядок множителей справа удобным образом мы можем вычислить совместное распределение.

Подобные *вероятностные графические модели* позволяют узнавать неочевидные взаимосвязи в данных, если в качестве вершин принять, например, признаки из какого-нибудь набора данных. При достаточном времени, потраченном на составлении связей в данном графе, аналитик данных способен в удобной форме отлавливать закономерности и проверять гипотезы о распределении данных. Также возможно их использование в системном или бизнес анализах, однако придётся потратить больше времени для дизайна нашего графа, поскольку мы можем столкнуться с не числовыми вершинами, а, например, событийными.

Другая полезная особенность таких моделей в том, что вместо какого-то конкретного значения интересующей нас величины мы получаем её распределение(Рисунок 2.4). Это даёт сильно больше информации, чем одно значение и позволяет оценивать *риски*, связанные с этой величиной. Существует много за-

дач, где определение рисков важнее какого-то одного ответа. Примеры: задача кредитного скоринга, большинство задач по работе с финансами(определение стоимости ценных бумаг, курса валют и т.д.), задачи в области медицины и здравоохранения, транспорт на автопилоте и т.п.

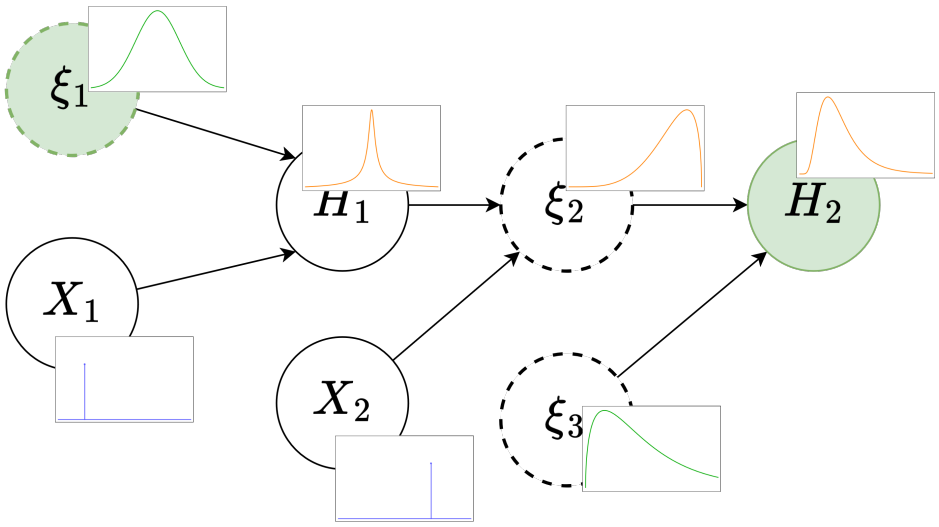

Рисунок 2.4 Та же графическая модель, но с видимыми распределениями значений в вершинах. Детерминированные вершины имеют δ-функцию распределения.

Существуют несколько инструментов для работы с такими моделями: Bayes Net Toolbox (`MATLAB`), pgmpy (`Python`) и др.

2.2.2 Байесовские нейронные сети.

Теперь рассмотрим байесовский подход к нейронным сетям. Такие сети в некоторой литературе называются – *байесовские нейронные сети.*

Стоит отметить, что байесовские сети/графы(bayes nets, bayes networks) и байесовские нейронные сети(bayesian neural networks) не одно и то же. Обычные байесовские сети в литературе чаще обозначают те же самые вероятностные графы вычислений, рассмотренные выше.

Байесовские нейронные сети [1] являются расширением классических нейронных сетей с применением байесовского подхода, то есть добавление априорной информации на параметры модели, а также оценивание распределения параметров вместо их конкретного значения. Такой подход позволяет в какой-то мере решить проблемы нейронных сетей, указанных ранее: как уже отмечалось

в Главе 1, добавление априорной информации эквивалентно добавлению регуляризации к параметрам модели.

Замечание. Кроме того, при обучении байесовских нейронных сетей распределения каких-то параметров могут вырождаться в почти детерминированные распределения. Это позволяет выбросить эти параметры из набора сэмплируемых параметров и установить им единственное константное значение. Таким образом мы можем проводить прореживание(sparse) параметров наших сетей. Тут также стоит отметить, что прореживание параметров вовсе не означает, что такие параметры можно полностью выкинуть из обучения. Если их зафиксировать и обучить нейронную сеть заново, то итоговое качество упадёт. Это происходит, в частности, из-за стохастической природы наших обучающих методов.

Теперь рассмотрим принцип работы байесовских нейронных сетей. Для простоты ограничимся линейными слоями нейронных сетей и на их примере рассмотрим принцип работы.

Будем считать, что параметры линейного слоя W_i и b_i являются случайными величинами (Рисунок 2.5):

$$W_i \sim p(W_i \mid h_{i-1})$$

$$b_i \sim p(b_i \mid h_{i-1})$$

$$h_i = W_i \cdot h_{i-1} + b_i$$

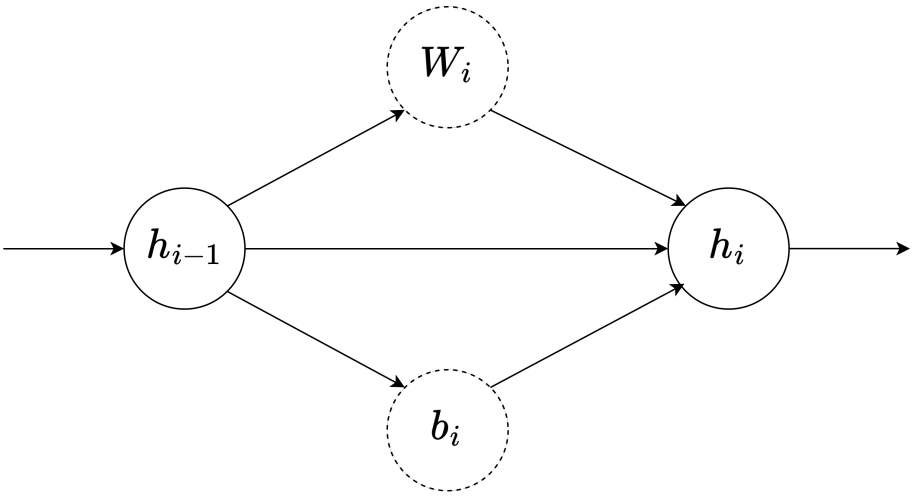

Рисунок 2.5 Вероятностный граф вычислений байесовской нейронной сети.

Для аппроксимации распределений $p(W_i \mid h_{i-1})$, $p(b_i \mid h_{i-1})$ воспользуемся сначала базовым методом(REINFORCE) из Главы 1 — будем приближать $p(W_i \mid h_{i-1})$ с помощью $q(W_i \mid \varphi^i_W)$(аналогично для b_i). Тогда наш граф вычислений принимает следующий вид(Рисунок 2.6).

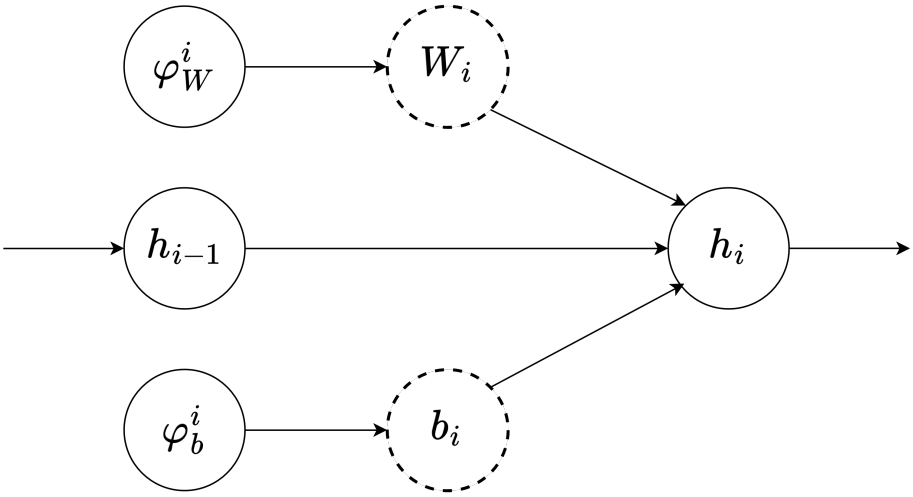

Рисунок 2.6 Вероятностный граф вычислений байесовской нейронной сети с параметризованными распределениями.

Для сэмплирования матрицы можно сэмплировать как отдельно все ячейки, так и столбцы или строки. Однако существует и другой метод сэмплирования целиком всей матрицы [2].

Как уже отмечалось такой метод имеет недостатки в виде большой дисперсии стохастического градиента. На эту проблему можно взглянуть с другой стороны: на схеме графа вычислений, когда градиент протаскивается с будущих слоёв до текущего, он должен пройти сквозь случайную вершину W_i и дойти до φ^i_W. Такой градиент будет иметь большую дисперсию, поскольку зависит от сэмплируемой величины W_i.

Теперь воспользуемся *репараметризационным трюком*, добавив две новые случайные величины $\varepsilon_W \sim r(\varepsilon_W)$, $\varepsilon_b \sim r(\varepsilon_b)$ и их функции преобразования для получения параметров $W = g_W(\varepsilon_W)$, $b = g_b(\varepsilon_b)$. Здесь стоит отметить, что ε_b и ε_W могут быть из пространств, размерности которых не совпадают с размерностями W и b. В этом случае граф вычислений преобразуется в такой вид(Рисунок 2.7).

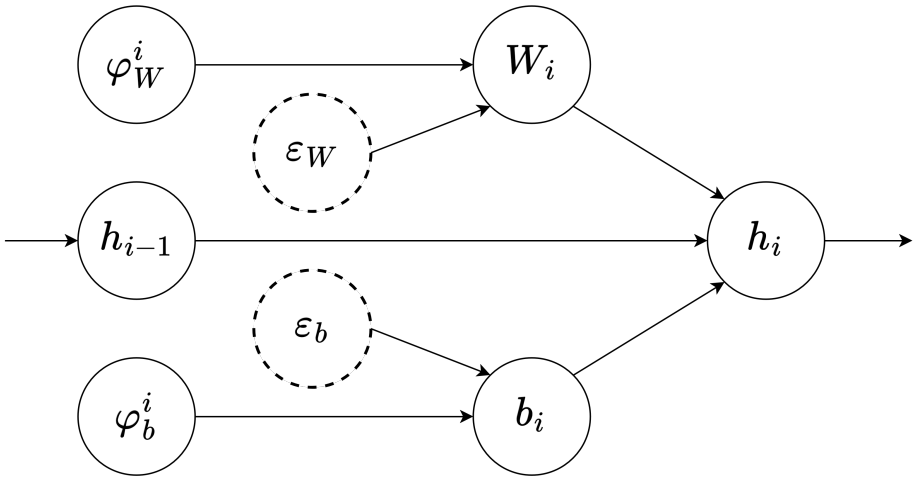

Рисунок 2.7 Байесовская нейронная сеть после применения репараметризации.

После репараметризации градиент будет беспрепятственно проходить только по детерминированным вершинам и избегать случайных вершин, что сильно улучшает качество процесса обучения.

Следующим шагом требуется алгоритм обучения, который бы позволял использовать идею "цепного правила"для нашей сети.

Предположим, что мы уже получили градиент $\frac{dL}{dh_i}$ со следующего слоя и хотим пересчитать градиенты весов текущего слоя, а также пробросить градиент далев предыдущий слой. Тогда формулы пересчёта градиента будут следующие:в предыдущий слой.

Тогда формулы градиента будут следующие:

$$\frac{d\mathcal{L}}{d\varphi_W^i} = \frac{d\mathcal{L}}{dh_i} \cdot \frac{dh_i}{dW_i} \cdot \frac{dW_i}{d\varphi_W^i}$$

$$\frac{d\mathcal{L}}{d\varphi_b^i} = \frac{d\mathcal{L}}{dh_i} \cdot \frac{dh_i}{db_i} \cdot \frac{db_i}{d\varphi_b^i}$$

$$\frac{d\mathcal{L}}{dh_{i-1}} = \frac{d\mathcal{L}}{dh_i} \cdot \frac{dh_i}{dh_{i-1}}$$

Однако, помимо градиента по функции ошибки, нам также нужно учесть и другие слагаемые из вариационной нижней оценки $L(\varphi)$. Будем предполагать, что параметры разных слоёв независимы, и тогда

$$q(\theta \mid \varphi) = \prod_i q(\theta_i \mid \varphi_i)$$

$$p(\theta) = \prod_i p(\theta_i)$$

И тогда формулы для градиента по параметрам распределения будут следующие:

$$\frac{d(-L)}{d\varphi_W^i} = \frac{d\mathcal{L}}{d\varphi_W^i} - \frac{d(\log p(W_i))}{d\varphi_W^i} + \frac{d(\log q(W_i \mid \varphi_W^i))}{d\varphi_W^i} =$$

$$= \Big(\frac{d\mathcal{L}}{dh_i} \cdot \frac{dh_i}{dW_i} - \frac{d(\log p(W_i))}{dW_i} + \frac{d(\log q(W_i \mid \varphi_W^i))}{dW_i}\Big)\frac{dW_i}{d\varphi_W^i} + \frac{d(\log q(W_i \mid \varphi_W^i))}{d\varphi_W^i}$$

Аналогичную формулу получаем для вектора b_i.

И отдельно алгоритмы прямого хода и обратного хода:

Algorithm 1 Прямой ход байесовского линейного слоя.

procedure FORWARD(h_{i-1})
- $\varepsilon_W \sim r(\varepsilon_W)$
- $\varepsilon_b \sim r(\varepsilon_b)$
- $W_i = g_W(\varphi_W^i, \varepsilon_W)$
- $b_i = g_b(\varphi_b^i, \varepsilon_b)$
- $Y = W_i \cdot h_{i-1} + b_i$
- **return** Y

Algorithm 2 Обратный ход байесовского линейного слоя.

procedure BACKWARD(h_{i-1}, g_o)
- $g_w^\varepsilon = \frac{d(g_W(\varphi_W^i, \varepsilon_W))}{d\varphi_W^i}$
- $g_b^\varepsilon = \frac{d(g_b(\varphi_b^i, \varepsilon_b))}{d\varphi_b^i}$
- $g_w = g_o \cdot \frac{dh_i}{dW_i} - \frac{d(\log p(W_i))}{dW_i} + \frac{d(\log q(W_i \mid \varphi_W^i))}{dW_i}$
- $g_b = g_o \cdot \frac{dh_i}{db_i} - \frac{d(\log p(b_i))}{db_i} + \frac{d(\log q(b_i \mid \varphi_b^i))}{db_i}$
- $g_{\varphi_W^i} = g_w \cdot g_w^\varepsilon + \frac{d(\log q(W_i \mid \varphi_W^i))}{d\varphi_W^i}$
- $g_{\varphi_b^i} = g_b \cdot g_b^\varepsilon + \frac{d(\log q(b_i \mid \varphi_b^i)}{d\varphi_b^i}$
- $\varphi_W^i = \varphi_W^i - \alpha_W^i g_{\varphi_W^i}$
- $\varphi_b^i = \varphi_b^i - \alpha_b^i g_{\varphi_b^i}$
- $g_i = g_o \cdot \frac{dh_i}{dh_{i-1}}$
- **return** g_i

Стоит отметить, что при использовании фреймоворков с функционалом автоматического дифференцирования, нет потребности в явном записывании этих

формул, поскольку все вычисления такой фреймоворк выполнит сам.

Стоит отметить ещё одно свойство таких моделей — ансамблевый эффект. Каждый раз, когда мы запускаем нашу модель для предсказания, она генерирует новый набор параметров, от которого считает значение предсказания. Таким образом, в некотором смысле наше предсказание является результатом работы ансамбля нескольких разных моделей. То есть байесовские нейронные сети имитируют *ансамблевое обучение(ensemble learning)*. Как известно ансамбли моделей часто дают качество в разы лучше, чем качество одной модели. Также некоторые виды ансамблей позволяют уменьшать *variance* в *bias-variance разложении*, что стабилизирует обучение и предсказание модели.

2.3 Трюк с локальной репараметризацией.

Теперь вернёмся к проблеме из 1.4.2. Для линейного слоя рассмотрим метод, который уменьшит ковариацию в оценке функции ошибки и тем самым уменьшим дисперсию самой оценки. Это метод [4] называется *трюк с локальной репараметризацией*. Он может быть применим для широкого семейства слоёв, но в данной ситуации рассмотрим упрощённый байесовский линейный слой:

$$Y = X \cdot W$$

$$y_{i,j} = \sum_{k=1}^{n} x_{i,k} \cdot W_{k,j},\ j = \overline{1...m}$$

В данном случае предположим веса распределены нормально, то есть

$$W_{i,j} \sim \mathcal{N}(\mu_{i,j}, \sigma_{i,j}^2),\ \forall W_{i,j} \in W$$

При этом значение весов получается за счёт метода репараметризации:

$$\varepsilon_{i,j} \sim \mathcal{N}(0, 1)$$

$$W_{i,j} = \mu_{i,j} + \sigma_{i,j}\varepsilon_{i,j}$$

Ненулевая ковариация между разными примерами образуется из-за того, что мы используем одну и ту же матрицу весов для всех примеров из мини-батча. Одним из способов её уменьшения будет сэмплирование отдельной матрицы

весов для **каждого** обучающего примера. Однако, данный способ является вычислительно неэффективным.

Вместо этого распишем распределение на Y:

$$\mathbb{E}[y_{i,j}] = \mathbb{E}\Big[\sum_{k=1}^{n} x_{i,k} W_{k,j}\Big] = \sum_{k=1}^{n} x_{i,k}\mathbb{E}[W_{k,j}] = \sum_{k=1}^{n} x_{i,k}\mu_{k,j}$$

$$\mathbb{D}[y_{i,j}] = \mathbb{D}\Big[\sum_{k=1}^{n} x_{i,k} W_{k,j}\Big] = \sum_{k=1}^{n} \mathbb{D}[x_{i,k} W_{k,j}] = \sum_{k=1}^{n} x_{i,k}^2\mathbb{D}[W_{k,j}] = \sum_{k=1}^{n} x_{i,k}^2\sigma_{k,j}^2$$

Введём обозначения:

$$\alpha_{i,j} = \sum_{k=1}^{n} x_{i,k}\mu_{k,j}$$

$$\beta_{i,j}^2 = \sum_{k=1}^{n} x_{i,k}^2\sigma_{k,j}^2$$

Каждое $y_{i,j}$ является линейной комбинацией нормально распределённых случайных величин $W_{k,j}$, значит и $y_{i,j}$ распределено тоже нормально, причём,

$$y_{i,j} \sim \mathcal{N}(\alpha_{i,j}, \beta_{i,j}^2)$$

И $y_{i,j}$ может быть вычислено с помощью уже известного трюка с репараметризацией:

$$\nu_{i,j} \sim \mathcal{N}(0, 1)$$

$$y_{i,j} = \alpha_{i,j} + \beta_{i,j}\nu_{i,j}$$

Замечание. Все описанные свойства дублируются для градиентов до параметрам.

Таким образом вместо сэмплирования матрицы весов и вычисления Y, можно сэмплировать выход линейного слоя напрямую. Такой метод имеет меньшую дисперсию, поскольку $y_{i,j}$ теперь независимы, то есть попарные ковариации из 1.4.2 теперь равно нулю. Это происходит в силу того, что для каждого выхода мы сэмплируем свою шумовую переменную $\nu_{i,j}$. В предыдущем методе происходило сэмплирование шумовой переменной $\varepsilon_{i,j}$ для одной матрицы весов, которая затем использовалось в линейном слое, из чего следовала зависимость между $y_{i,j}$.

Теперь посчитаем дисперсию для новой оценки:

$$\mathcal{L}_{D_x,D_y}(\varphi) \simeq \widehat{\mathcal{L}}^{LRT}_{D_x,D_y}(\varphi) = \frac{N}{B}\sum_{i=1}^{B}\widehat{\mathcal{L}}_i^{LRT}$$

Поскольку $\widehat{\mathcal{L}}_i^{LRT}$ независимы, то ковариация любых двух различных $\widehat{\mathcal{L}}_i^{LRT}$ будет равно нулю:

$$\mathbb{D}[\widehat{\mathcal{L}}^{LRT}_{D_x,D_y}(\varphi)] = \mathbb{D}\Big[\frac{N}{B}\sum_{i=1}^{B}\widehat{\mathcal{L}}_i^{LRT}\Big] = \frac{N^2}{B^2}\mathbb{D}\Big[\sum_{i=1}^{B}\widehat{\mathcal{L}}_i^{LRT}\Big] = \frac{N^2}{B^2}\sum_{i=1}^{B}\mathbb{D}[\widehat{\mathcal{L}}_i^{LRT}] = \frac{N^2}{B}\mathbb{D}[\widehat{\mathcal{L}}_i^{LRT}]$$

Таким образом, с увеличением размера мини-батча дисперсия оценки будет только уменьшаться.

2.4 Байесовский подход к свёрточным слоям.

Чаще всего в контексте байесовского подхода на практике рассматриваются линейные слои(впрочем, как и в этой работе), однако не будет лишним упомянуть и то, как можно задавать априорные распределения на другие виды слоёв. Например, можно рассмотреть свёрточные слои. По своей природе свёрточные слои являются набором фильтров, которые представляют собой двумерные матрицы(в некоторых случаях трёхмерные).

Одним из способов задать параметрическое семейство фильтров, подобные тем, что получаются в процессе обучения сети — фильтры Габора.

Чтобы задать фильтр Габора нужно знать параметры: w — размеры фильтра, θ, σ, λ, ψ, γ:

$$G_{i,j} = \exp\left(-\frac{x^2+\gamma y^2}{2\sigma^2}\right)\cos\left(\frac{2\pi x}{\lambda}+\psi\right)$$

$$x = i\cdot\cos\theta + j\cdot\sin\theta$$

$$y = -i\cdot\sin\theta + j\cdot\cos\theta$$

$$i,j \in \{1,2,3,...,w\}$$

В [7] путём тюнинга параметров, перечисленных выше, были предложены сле-

дующие распределения на параметры:

$$\theta \sim \mathcal{U}(0, \pi)$$

$$\sigma \sim \mathcal{U}(2, 10)$$

$$\lambda \sim \mathcal{U}(1, w)$$

$$\psi \sim \mathcal{U}(-\pi, \pi)$$

$$\gamma \sim \mathcal{U}(0, 1.5)$$

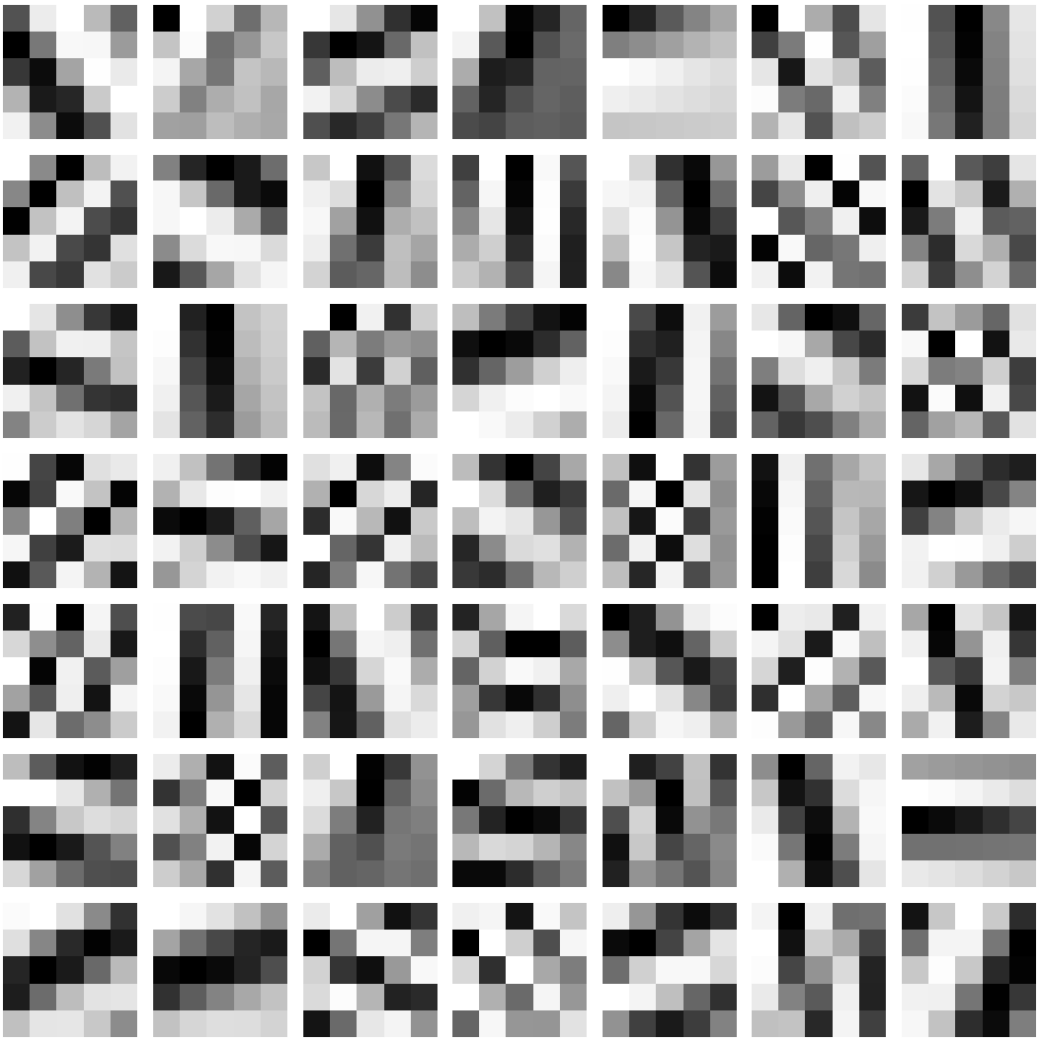

Рисунок 2.8 Примеры фильтров Габора, созданные в соответствии с вышеуказанными распределениями.

ГЛАВА 3

Фреймворк на C++ для нейробайесовских моделей.

3.1 Описание фреймворка.

3.1.1 Компиляция и сборка фреймворка.

Реализация фреймворка будет происходить на `C++20` с дальнейшим биндингом этого фреймворка в `Python 3.12`, как полноценного пакета-расширения. В качестве системы сборки плюсовой части будет использован `CMake`, а для биндинга внутрь `Python` — `PyBind11`. Чтобы создать пакет-расширение с с возможностью дальнейшего распространения в интернете требуется создать `wheel`-пакет с расширением `.whl`, который будет возможно установить обычной командой `pip install *.whl`.

Для сборки такого `wheel`-пакета будет использован `scikit-build 0.18`, который автоматически компилирует `C++` часть проекта и собирает её в файл общей библиотеки `.so`, а также собирает итоговой `.whl`-пакет, в который подсоединяет файлы общей библиотеки.

Поскольку определение и описание всего функционала происходит на чистом `C++`, то после биндинга в `Python` будут отсутствовать возможности корректного автодополнения кода и подсказки. Для того, чтобы эти возможности вернуть, существуют `stub`-файлы формата `.pyi`. Их основная цель в том, чтобы предоставлять информацию о типах для нетипизированных пакетов и модулей `Python`. Особенно это полезно для случаев, когда мы отделяем описание(*declaration*) функционала в отдельные файлы, которые могут быть использованы, как документация. В моём случае для автоматического создания `stub`-файлов будет использован `pybind11-stubgen`. Получившиеся `.pyi` файлы требуется поместить внутрь того же `wheel`-пакета, и после этого станет доступна возможность корректного автодополнения и подсказок.

3.1.2 Функционал фреймворка.

Основным классом фреймворка является класс `Tensor`, который реализует основной функционал многомерного массива для работы с данными. Также `Tensor` поддерживает функционал автоматического дифференцирования. Кроме того, `Tensor` имеет совместимость с уже известными многомерными массивами другой популярной библиотеки — `NumPy`.

Помимо этого фреймворк содержит класс `Module`, от которого будут наследоваться все слои нейронной сети, включая байесовские слои. Помимо этого есть класс `Optimizer`, который является классом, от которого будут наследоваться все отпимизаторы: `SGD`, `Adam` и т.д.

С точки зрения функционала фреймворка, то он покрывает базовые потребности в обучении нейронных сетей подобно уже известным `PyTorch`, `TensorFlow`. При проектировании архитектуры было уделено внимание тому, чтобы интерфейс функций, классов и методов был знаком всем пользователям уже известных библиотек. Также было не мало важно не уступать по времени выполнения математических функций агрегирования, умножения матриц и прочего уже известным библиотекам. Для того, чтобы добиться значительного ускорения, потребовалось воспользоваться специфичными инструкциями процессора семейства `AVX512` [3]. Подобные и другие инструкции, которые можно найти по ссылке, добиваются ускорения вычислений за счёт использования специальных типов регистров на процессоре, которые могут оперировать сразу с несколькими переменными вещественного типа, занимающие подряд находящиеся байты в памяти.

3.1.3 Юнит-тестирование фреймворка.

В силу большого объёма проекта и большого количества кода — требуется тестирование его отдельных частей и компонентов, и это называется юнит-тестированием. Поскольку фреймворк одновременно реализовывался для двух языков программирования `Python` и `C++`, то следует покрывать тестирование оба языка. Для `Python` использовалась библиотека `Pytest`, а для `C++` — `Catch2`. Юнит-тестирование позволяет значительно сократить время, затрачиваемое на поиск ошибок в коде, которые могли бы возникнуть в будущем.

3.2 Тестирование фреймворка и нейробайесовских сетей.

Для тестирования самих нейробайесовских сетей и фреймворка использовались два набора данных - `FashionMNIST`, `CIFAR10`. В качестве архитектуры были выбраны достаточно простые сети, состоящие только из линейных байесовских слоёв. Использование байесовского подхода для свёрточных слоёв в данной работе не рассматривается. Нейронная сеть состоит из двух полносвязных слоёв по 1200 нейронов в скрытых слоях.

Ниже на картинках представлены значение метрики `Accuracy` для обычной модели и нейробайесовской. Обучение происходило в одинаковых условиях, то есть значение скорости обучения и прочих гиперпараметров равны.

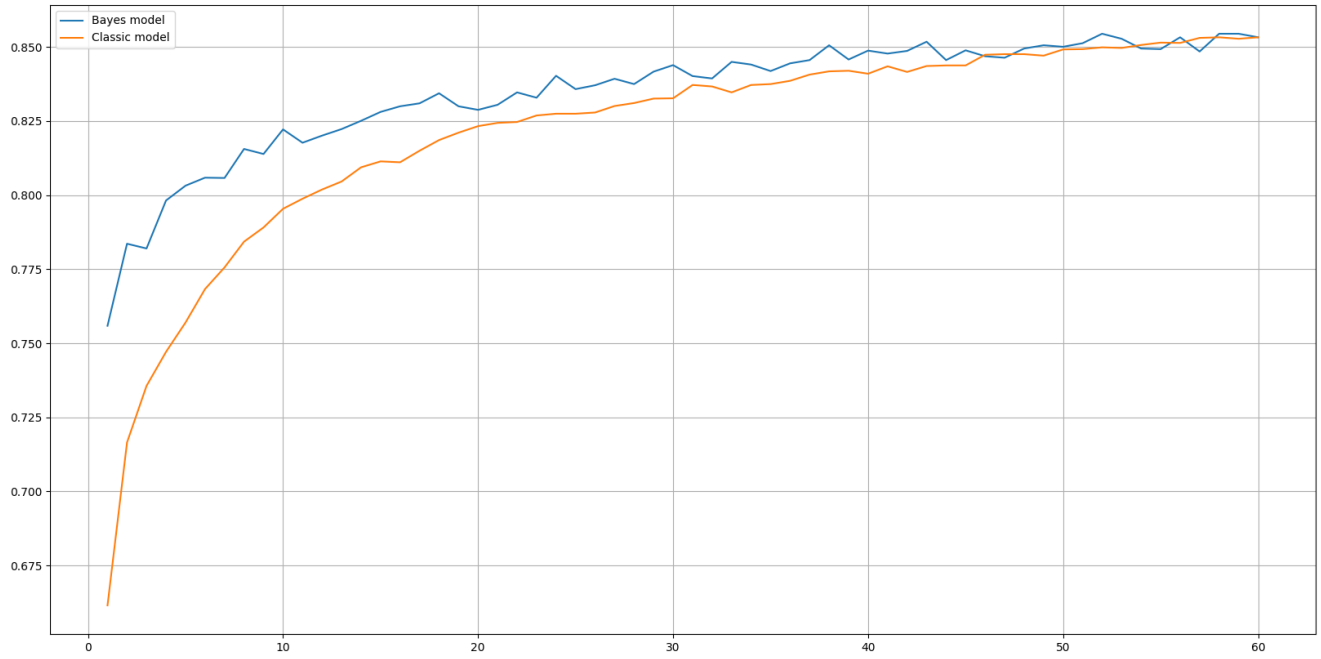

Рисунок 3.1 Значение метрики *Accuracy* на *FashionMNIST* во время процесса обучения.

Как можно увидеть, байесовская модель быстрее сходится к оптимальному решению и у неё лучше обобщаю способность. В то время, как детерминированная сеть начинает с более худшего решения, однако по итогу сходится к тому же решению, что и байесовская сеть.

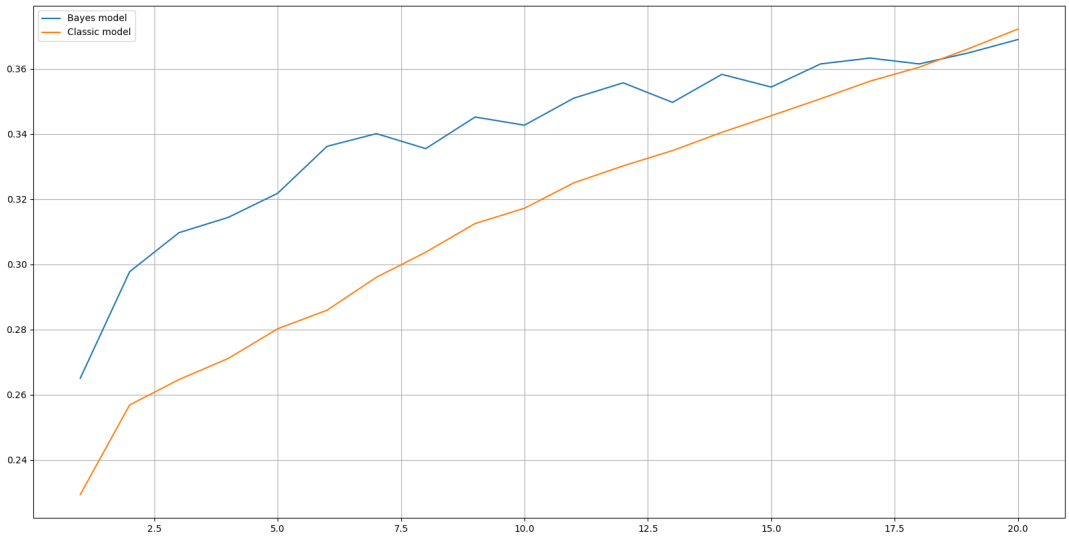

Рисунок 3.2 Значение метрики *Accuracy* на *CIFAR10* во время процесса обучения.

На этом же наборе данных наблюдается похожее поведение, что и в предыдущем наборе данных, однако детерминированная модель имеет более стабильный процесс обучения, чем у байесовской сети.

Поведение, связанное с тем, что байесовская модель начинает из более благоприятных точек, чем детерминированная модель, может быть объяснено за счёт использования корректных априорных распределений. Такие распределения также называют *информативными*, поскольку они дают некоторую полезную информацию модели ещё до начала обучения. Однако, детерминированная модель способна найти такое же хорошее решение, но на это ей потребуется время. Поиск информативных распределений достаточно нетривиальная задача, поэтому, несмотря на вероятное улучшение процесса обучения, использование байесовского подхода усложняет обучения нейронных сетей.

ЗАКЛЮЧЕНИЕ

В ходе работы были разобраны основные научные статьи по нейробайесовским методам, выполнены математические выкладки и объяснения на русском языке.

Дано объяснение о том, что из себя представляют байесовские сети и байесовские нейронные сети, в том числе сравнение с классическими нейронными сетями. Также было дано несколько обоснований полезности нейробайесовских методов, которые были подкреплены практическими экспериментами на наборах данных CIFAR10, FashionMNIST.

Также был реализован основной функционал фреймворка для обучения нейронных сетей, который, однако, требует будущей доработки.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Blundell C., Cornebise J., Kavukcuoglu K., and Wierstra D. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 1613–1622, 2015.

[2] Louizos C. and Welling M. Structured and efficient variational deep learning with matrix gaussian posteriors. *International Conference on Machine Learning*, 2016.

[3] Intel. Intel intrinsics guide. https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html. Accessed: 2025-01-20.

[4] Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, volume 28, 2015.

[5] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *2nd International Conference on Learning Representations*, 2014.

[6] Hoffman M.D. Stochastic variational inference. *Journal of Machine Learning Research 14*, 2013.

[7] Tim Pearce, Andrew Y.K. Foong, and Alexandra Brintrup. Structured weight priors for convolutional neural networks. *ICML 2020 Workshop on Uncertainty and Robustness in Deep Learning*, 2020.

[8] Ranganath R. Black box variational inference. *17th International Conference on Artificial Intelligence and Statistics*, 2014.

[9] Williams R. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[10] Ивченко Г.И. и Медведев Ю.И. *Математическая статистика.* Издательство "Высшая школа 1984.