به نام خدا

دانشگاه تهران - دانشگاه صنعتی خواجه نصیرالدین طوسی تهران

دانشکده مهندسی برق و کامپیوتر

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق - گروه مهندسی کنترل

تشخیص اندیشه

نام و نام خانوادگی	محمدجواد احمدی
شمارهٔ دانشجویی	۴۰۱۰۰۰۸۶

فهرست مطالب

۱ پاسخ پرسش دوم ۳
۱.۱ پاسخ قسمت ۱ . . . . . ۳
۲.۱ پاسخ قسمت ۲ . . . . . ۷
۳.۱ پاسخ قسمت ۳ . . . . . ۱۳
۴.۱ پاسخ قسمت ۴ . . . . . ۳۲

فهرست تصاویر

۱ مقایسهٔ ساختاری RNN و LSTM ۴
۲ فرآیند تبدیل کلمه به بردار ۵
۳ لایهٔ Embedding ۶
۴ Word2Vec ۶
۵ Glove ۷
۶ نمایی از مدل اول ۱۳
۷ نمودار دقت و اتلاف – مدل اول – h=25 . . . . . ۱۸
۸ نمودار دقت و اتلاف – مدل اول – h=50 . . . . . ۱۹
۹ نمودار دقت و اتلاف – مدل اول – h=75 . . . . . ۱۹
۱۰ نمودار دقت و اتلاف – مدل اول – h=100 . . . . . ۱۹
۱۱ ماتریس درهم‌ریختگی – مدل اول – h=25 . . . . . ۲۰
۱۲ ماتریس درهم‌ریختگی – مدل اول – h=50 . . . . . ۲۰
۱۳ ماتریس درهم‌ریختگی – مدل اول – h=75 . . . . . ۲۱
۱۴ ماتریس درهم‌ریختگی – مدل اول – h=100 . . . . . ۲۱
۱۵ نمایی از مدل دوم ۲۲
۱۶ نتیجهٔ تابع دقت و اتلاف در حالت اعتبارسنجی با داده‌های تست (منطبق بر مقاله) ۲۹
۱۷ ماتریس درهم‌ریختگی در حالت اعتبارسنجی با داده‌های تست (منطبق بر مقاله) ۳۰
۱۸ نتیجهٔ تابع دقت و اتلاف در حالت اعتبارسنجی با داده‌های تست (با ترم تنظیم) ۳۱
۱۹ ماتریس درهم‌ریختگی در حالت اعتبارسنجی با داده‌های تست (با ترم تنظیم) ۳۲
۲۰ نتیجهٔ تابع دقت و اتلاف در حالت داده‌های اعتبارسنجی جداگانه ۳۳
۲۱ ماتریس درهم‌ریختگی در حالت داده‌های اعتبارسنجی جداگانه ۳۴
۲۲ نتیجهٔ تابع دقت و اتلاف در حالت داده‌های اعتبارسنجی جداگانه ۳۵
۲۳ ماتریس درهم‌ریختگی در حالت داده‌های اعتبارسنجی جداگانه ۳۶
۲۴ نمایی از مدل پاسخگو ۳۶
۲۵ نمایی از مدل Prototype Responder ۴۰

پرسش ۲. تشخیص اندیشه

۱ پاسخ پرسش دوم

توضیح پوشهٔ کدهای تشخیص اندیشه

کدهای مربوط به این قسمت، علاوه بر پوشهٔ محلی کدها در این لینک گوگل‌کولب آورده شده است.

۱.۱ پاسخ قسمت ۱

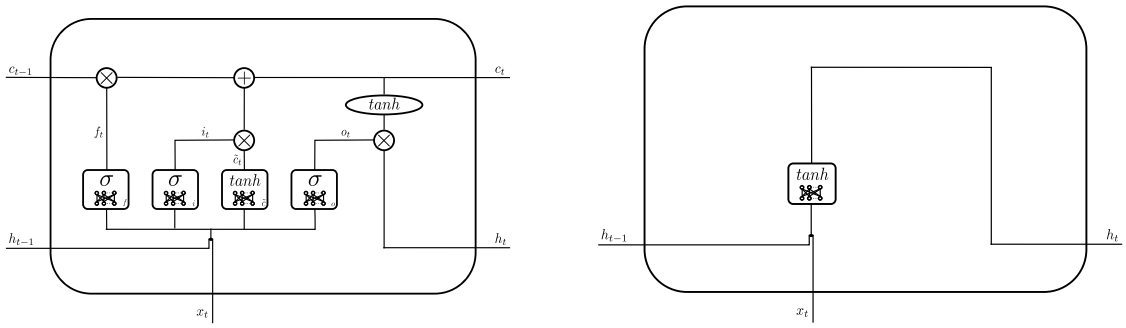
ابتدا توضیحاتی از مقاله ذکر می‌کنیم. مقاله این‌طور عنوان می‌کند که درک زبان طبیعی مستقیماً به اندیشهٔ انسان مرتبط است و به همین دلیل ماندگار است. در حین خواندن، درک یک متن، به‌سادگی و از طریق درک یک کلمهٔ تنها رخ نمی‌دهد، بلکه اغلب از طریق نحوهٔ چینش کلمات اتفاق می‌افتد. به عبارت دیگر، به مدل‌سازی پویایی نیاز است که با آن کلمات فردی به وجود می‌آیند. متأسفانه، شبکه‌های عصبی پیش‌خور سنتی به طور مستقیم قابلیت استخراج اطلاعات از ترتیب زمانی که ورودی‌ها رخ می‌دهند را ندارند، زیرا تنها به در نظر گرفتن ورودی فعلی برای پردازش خود محدود هستند. ایدهٔ در نظر گرفتن بلوک‌های کلمات ورودی به صورت مستقل از یکدیگر در زمینهٔ پردازش زبان طبیعی خیلی محدود است. در واقع، همانطور که فرآیند خواندن برای انسان منجر به حفظ تمام کلمات متن نمی‌شود، بلکه به استخراج مفاهیم اساسی بیان شده می‌انجامد، ما به یک «حافظه» نیاز داریم که فقط به در نظر گرفتن صریح ورودی‌های قبلی محدود نباشد، بلکه تمام اطلاعات مرتبط کسب‌شده در هر مرحله را به متغیرهای وضعیت تزریق می‌کند.

مجموعهٔ این دلایل باعث به وجود آمدن شبکه‌های عصبی بازگشتی (RNN) شده است که با معرفی مفهوم "وضعیت داخلی" (بر اساس ورودی‌های قبلی)، به ویژه در وظایف پردازش زبان طبیعی، قابلیت بسیاری را نشان داده است. شبکه‌های عصبی بازگشتی حلقه‌های داخلی دارند که اجازه می‌دهند اطلاعات قبلی از طریق مراحل مختلف تحلیلی عبور کنند. می‌توان با بازکردن فعالیت‌های شبکه و در نظر گرفتن زیرشبکه‌های فردی به عنوان نسخه‌هایی (در هر مرحله به‌روز شده) از همان شبکه، که به هم متصل شده‌اند و یک زنجیره را تشکیل می‌دهند، یک دید صریح از سیستم بازگشتی به دست آورد. در این زنجیره هر یک اطلاعاتی را به جانشین خود ارسال می‌کنند.

در بحث‌های نظری، شبکه‌های عصبی بازگشتی باید قادر باشند اطلاعاتی را که الگوریتم یادگیری در یک دنباله آموزشی پیدا می‌کند، در وضعیت‌های خود نگه‌دارند. متأسفانه، در انتشار گرادیان به عقب در طول زمان، تأثیر مقادیر خروجی مطلوب که به تابع هزینه کمک می‌کنند، به حدی کوچک می‌شود که پس از تنها چند مرحله، چالش و مسألهٔ محو گرادیان‌ها پدیدار می‌شود. این مشکل، در برخی پژوهش‌ها به‌صورت مشخص مورد بررسی قرار گرفته است، به این معنی است که شبکه‌های عصبی بازگشتی تنها می‌توانند یک حافظهٔ کوتاه‌مدت داشته باشند، زیرا بخش‌های دنباله که در زمان دورتر قرار دارند، به تدریج کم‌اهمیت‌تر می‌شوند. این باعث می‌شود که شبکه‌های عصبی بازگشتی تنها برای دنباله‌های بسیار کوتاه مفید باشند که این مطلوب نیست.

برای غلبه بر مشکل حافظهٔ کوتاه‌مدت (حداقل به طور جزئی)، ساختارهای با حافظهٔ بلندمدت (LSTM) معرفی شدند. برخلاف یک شبکهٔ RNN کلاسیک (تصویر ۲۳(ه))، LSTM یک ساختار سلولی بسیار پیچیده‌تر دارد (تصویر ۲۳(و)) که در

آن یک شبکهٔ عصبی تکی با چهار شبکه جایگزین شده که با یکدیگر تعامل دارند. با این حال، عنصر ممتاز LSTM سلول حالتِ (Cell State) c است که به اطلاعات اجازه می‌دهد از طریق عملیات خطی ساده در طول زنجیره جریان یابند. اضافه یا حذف‌کردنِ اطلاعات از حالت توسط سه ساختار، به نام «دروازه‌ها»، که هر کدام هدف‌های خاصی دارند، تنظیم می‌شود. اولین



RNN (آ) LSTM (ب)

شکل ۱: مقایسهٔ ساختاری RNN و LSTM.

دروازه که «دروازهٔ فراموشی» نام دارد، برای تصمیم‌گیری دربارهٔ اطلاعاتی که باید از حالت حذف شود، استفاده می‌شود. برای دستیابی به این هدف، حالت به طور نقطه‌ای با مقدار زیر ضرب می‌شود:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \quad ، \tag{۱}$$

این مقدار از یک بلوک خطی (درواقع affine به دلیل وجود انحرافات و بایاس‌ها) به دست می‌آید که بردار مشترک ورودی را با خروجی قبلی ($[h_{t-1}, x_t]$) ترکیب می‌کند، و سپس تابع فعال‌سازی سیگموئیدی معمولی ($\sigma(x) = \frac{1}{1+e^{-x}}$) را را وارد داستان می‌کند. دروازهٔ فراموشی امکان حذف (ارزش‌های نزدیک به صفر) یا حفظ (ارزش‌های نزدیک به یک) مؤلفه‌های هر بردار حالت را فراهم می‌کند. دروازهٔ دوم که دروازهٔ ورودی» نام دارد، هدفی دارد که شرایط اضافه‌کردن اطلاعات جدید به حالت را فراهم کند. این عملیات از طریق ضرب نقطه‌ای بین دو بردار به دست می‌آید:

$$\begin{aligned} i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right), \\ \tilde{c}_t &= \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \end{aligned} \tag{۲}$$

بردار اول (همواره از طریق یک فعال‌سازی سیگموئیدی به دست می‌آید) که برای به‌روزرسانی مقادیر تصمیم می‌گیرد، و بردار دوم (از طریق یک لایه با فعال‌سازی تانژانت هایپربولیک به دست می‌آید) که هدف آن ایجاد کاندیدای‌های جدید است. توجه کنید که تابع تانژانت هایپربولیک نیز هدفی دارد که آن، تنظیم جریان اطلاعات و اجبار فعال‌سازی‌ها برای باقی‌ماندن در بازه منفی‌یک و یک است. باید توجه داشت که به‌روزرسانی وضعیت سلول تنها به دو دروازه‌ای که تازه تعریف شده‌اند وابسته است و در واقع توسط رابطهٔ ۳ نمایش داده می‌شود:

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{۳}$$

در نهایت، «دروازهٔ خروجی» کنترل تولید خروجی جدید h_t برای این سلول را به‌صورت وابسته به ورودی کنونی و حالت قابل مشاهدهٔ قبلی کنترل می‌کند:

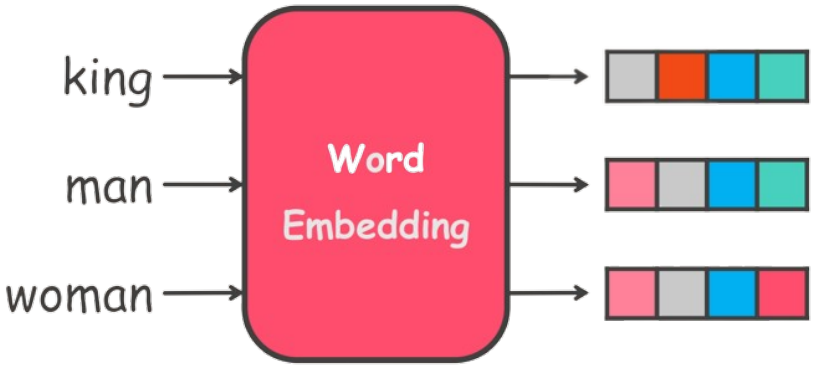
$$\begin{aligned} o_t &= \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right), \\ h_t &= o_t * \tanh\left(c_t\right). \end{aligned} \tag{۴}$$

هم‌چنین، در این حالت نیز یک لایهٔ فعال‌سازی سیگموئیدی وجود دارد که قسمتی از حالت را مشخص می‌کند و آن‌ها را در مقادیر بین صفر و یک ضرب می‌کند. توجه کنید که، به‌منظور محدودکردن مقادیر خروجی، تابع تانژانت هایپربولیک به هر عنصر از بردار حالت اعمال می‌شود (این یک تابع ساده بدون هیچ لایهٔ عصبی دیگری است)؛ پیش از ضرب در بردار مشخص‌شده توسط دروازه.

بنابراین با توجه به موارد ذکرشده، اولین مشکل RNN مسألهٔ محو گرادیان است که یکی از متهم‌های اصلی آن همان تانژانت هایپربولیک به‌کاررفته در قسمت فعال‌سازی است. این مسأله منجر به ضعف در حفظ وابستگی‌های بلندمدت می‌گردد.

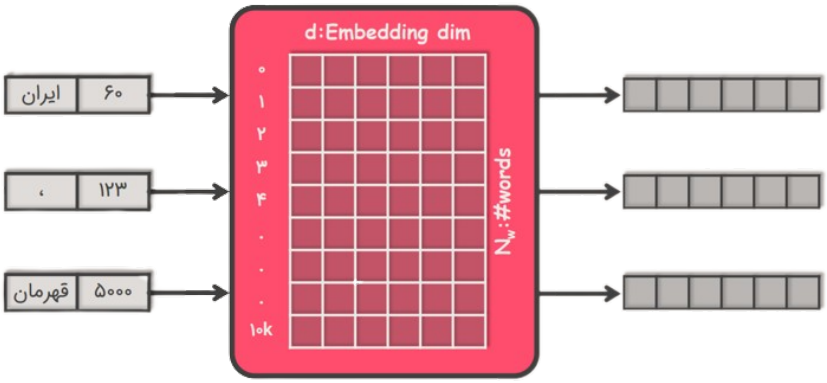
مقاله در مورد روش‌های کدگذاری این‌گونه بیان می‌کند که بدون شک و بدون وابستگی به نوع شبکه عصبی استفاده‌شده، لازم است تعیین کنیم که چه نوع تعبیری از جملات استفاده می‌شود. در واقع، برخلاف زبان‌های رسمی که به طور کامل مشخص هستند، زبان طبیعی از نیازهای سادهٔ ارتباطی بشر نشأت می‌گیرد و به همین دلیل دارای تعداد زیادی ابهام است. برای حداقل سعی در فهمیدن آن، لازم است نوعی «نزدیکی معنایی» بین اصطلاحات مختلف مشخص شود، مثل تبدیل‌کردن هر کلمه به برداری با مقادیر واقعی و به صورت مناسب. بنابراین، تعبیر به‌دست‌آمده می‌تواند تک‌تک کلمات را به یک نمایش عددی تبدیل کند که معنای آن‌ها را حفظ می‌کند و در بعضی معانی، آن را قابل فهم برای رایانه نیز می‌کند. امروزه روش‌های مختلفی برای به‌دست آوردن این فضای معنایی وجود دارد که به طور کلی به عنوان تکنیک‌های تعبیر کلمه شناخته می‌شوند. در این مقاله، تصمیم گرفته شده که از یک تعبیر پیش‌آموزش‌دیده به نام GloVe استفاده شود، که بر اساس مجموعه‌واژگان ۴۰۰۰۰۰ کلمه‌ای در یک فضای ۳۰۰ بعدی نگاشت شده است.

تبدیل کلمات به بردار از اهمیت بالایی برخوردار است. ما می‌خواهیم کلمات را همانند آن‌چه که در ؟؟ نشان داده شده به یک لایهٔ Word Embedding بدهیم و از آن طرف بردار تحویل بگیریم. آین بردارهای خروجی نباید One-Hot باشند؛ چراکه، مثلاً در حالت عادی ارتباطاتی بین کلمات وجود دارد. مثلا بین کلمات مرد و زن و یا خواهر و برادر ارتباطاتی است. دو بردار One-Hot همواره ضرب نقطه‌ای و تشابه صفر دارند و به درد ما نمی‌خورند. بنابراین خروجی بردار باید چیزی مثل همان خروجی برداری در شکل ۲ باشد. بنابراین، ما با برداری‌کردن مناسب کلمات به قدرت مناسب پردازش هم دست پیدا می‌کنیم و این بخش اهمیت بالایی دارد. لایهٔ Embedding اصولاً یک ماتریس است که دو بعد دارد و تعداد سطرهای آن معادل تعداد کلمات



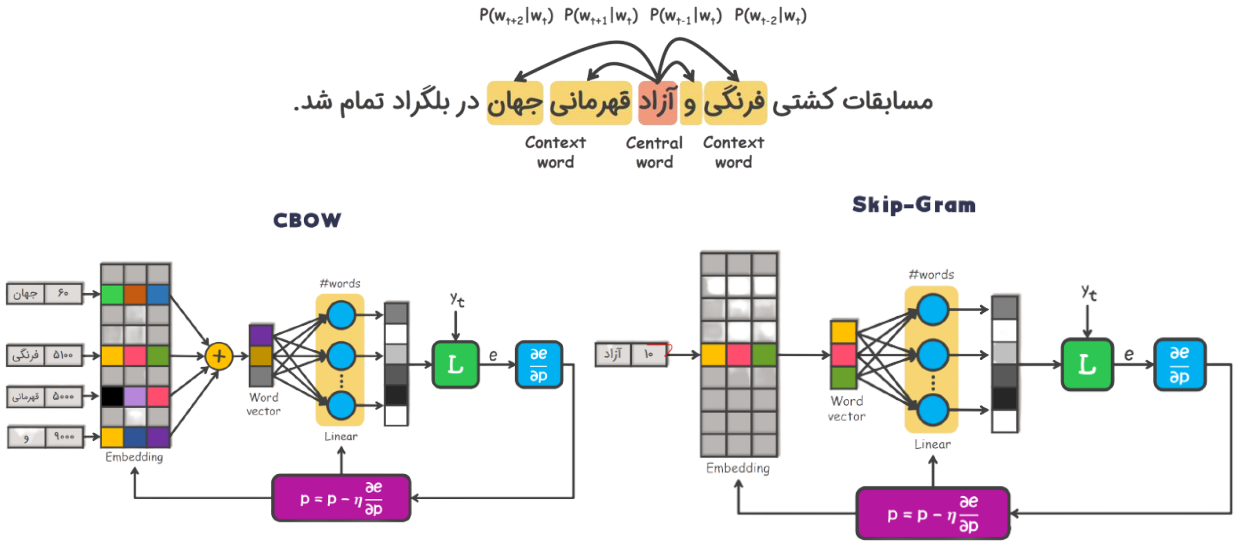
شکل ۲: فرآیند تبدیل کلمه به بردار.

دیکشنری است و هر سطر معادل یک کلمه است. و هم‌چنین تعداد ستون‌ها نشان‌دهندهٔ طول برداری است که می‌خواهیم برای هر کلمه داشته باشیم. مثالی از این موضوع در شکل ۳ نشان داده شده است. این لایه شبیه به همان لایهٔ خطی است و این ماتریس هم قابل آموزش است و باید آموزش داده شود تا مقادیر آن معنا دارد شود (مثلاً کلمات مرد و زن به هم شبیه شوند).



شکل ۳: لایهٔ Embedding.

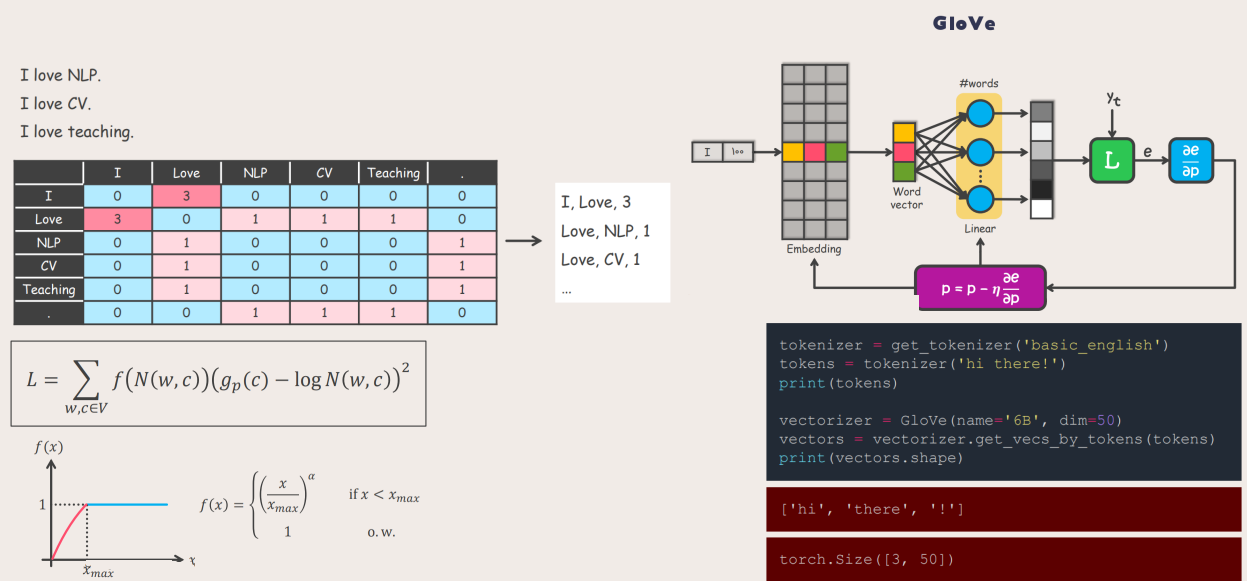
در ادامه بد نیست به نحوهٔ فهم کلمات جدید برای خودمان به‌عنوان یک انسان توجه کنیم. ما وقتی معنای کلمه‌ای را نمی‌دانیم و آن را در جملاتی مشاهده می‌کنیم، بر اساس کلمات قبل و بعدی که از آن می‌بینیم در مورد مفهوم آن کلمهٔ ناشناخته فهمی پیدا می‌کنیم. بر اساس همین ایده یک الگوریتم قدیمی به نام Word2Vec ایجاد شده که در آن می‌توان مفهوم یک کلمهٔ مرکزی (Central Word) را بر اساس پنجره‌ای از کلمات همسایه (Context Word) را حدس زد. این کار با الگوریتم CBOW انجام می‌شود. این رابطه دوطرفه است و از کلمهٔ مرکزی هم می‌توان به کلمات همسایه رسید. این کار با الگوریتم Skip-Gram انجام می‌شود. شمایی از این الگوریتم در شکل ۴ نشان داده شده است. در این الگوریتم کلمات کم‌کاربرد مانند حروف ربط را معمولاً حذف می‌کنیم. می‌توان از اوزان آماده ماتریس نهفته هم استفاده کرد. Glove هم یکی از روش‌هایی است که می‌توانیم با آن



شکل ۴: Word2Vec.

یک لایهٔ Embedding معنادار و خوبی داشته باشیم. اوزان پیش‌آموزش‌دیدهٔ آن هم در پای‌تورچ آمادهٔ استفاده است. تفاوت کوچک این روش با Word2Vec این است که در این‌جا زوج کلماتی که می‌خواهیم بسازیم بر اساس یک ماتریس هم‌رخ‌داد یا Co-occurance Matrix ساخته می‌شود. فرض کنید یکسری جمله داشته باشیم و در آن می‌خواهیم بررسی کنیم کلمات چندبار

و به چه صورتی در کنار هم قرار گرفته‌اند. نمونه‌ای از این فرآیند ساخت ماتریس هم‌رخداد در شکل ۵ نشان داده شده است. این ماتریس اطلاعات ارزشمندی به فرآیند آموزش اضافه می‌کند. ساختار آموزش این روش کاملاً مشابه قبل است اما تابع اتلاف آن به‌جای آنتروپی متقابل بر پایهٔ MSE (معادلهٔ داخل شکل ۵) است. در این معادله زوج‌کلمات مدنظر ما هستند. N نمایان‌گر همان ماتریس هم‌رخداد است و g هم خروجی کلاسیفایر پیش‌بینی و لایهٔ نهفته است. در نهایت یک وزن هم در این رابطه داریم. این وزن عمدهٔ تمرکزش بر واژگانی است که ممکن است اتفاقی چندبار کنار هم قرار گرفته باشند و اهمیت بالایی هم نداشته باشند. درواقع این وزن می‌گوید این زوج‌کلمات باید مثلا حداقل ۱۰۰بار کنار هم دیده شوند که اهمیت بالایی پیدا کنند.



شکل ۵: Glove.

نقطهٔ ضعف پررنگ روش‌هایی که تاکنون اشاره کردیم فهم معنای کلمات چندمعنا است و لایهٔ Embedding به‌ازای آن کلمه بدون توجه به معنای آن تنها یک بردار واحد نسبت می‌دهد. روش شبکهٔ ELMO بر پایهٔ LSTM سعی می‌کند برای هر کلمه با توجه به جمله‌ای که در آن قرار گرفته یک بردار معنادار بسازد. در واقع ما به این شبکه جمله می‌دهیم و می‌گوییم به دنبال معنا و بردارمعنادار کلمات هستیم. اما بعد از آمدن ترنسفورمرها شبکه‌ای بر پایهٔ ترنسفورمر به نام BERT آمد که سعی داشت تمام مشکلات مانند همین مشکل چندمعنایی را حل کند (BERT Word Embeddings).

۲.۱ پاسخ قسمت ۲

مقاله در مورد مجموعه‌داده این‌گونه بیان می‌کند که به منظور آموزش و آزمایش مدل‌های مختلف، از یک مجموعه‌دادهٔ ناهمگن استفاده شده است. این مجموعه‌داده شامل سوالاتی است که به صورت دستی ساخته شده و همچنین سوالاتی که توسط TREC و USC منتشر شده‌اند. این مجموعه داده شامل ۵۵۰۰ سوال در زبان انگلیسی در مجموعه آموزش و ۵۰۰ سوال در مجموعه آزمایش است. هر سوال دارای یک برچسب است که پاسخ را در یک سلسله مراتب دو سطحی دسته‌بندی می‌کند. سطح بالاتر شامل شش کلاس اصلی است (کوتاه‌نویسی، موجودیت، توصیف، انسان، مکان، عدد)، هر یک از این کلاس‌ها در نتیجه به زیرکلاس‌های متمایزی تقسیم می‌شوند که با هم در سطح دوم سلسله مراتب قرار می‌گیرند (مانند: انسان/گروه، انسان/شخصیت، مکان/شهر، مکان/کشور و غیره). به طور کلی، از ترکیب همه دسته‌ها و زیردسته‌ها، ما ۵۰ برچسب مختلف برای دسته‌بندی پاسخ

به سوالات در اختیار داریم. یک مثال که از مجموعه‌داده استخراج شده است به صورت زیر است (برچسب با حروف توپر و سوال به‌صورت ایتالیک درج شده است):

HUMAN:ind *Who was the 16th President of the United States?*

همانطور که مشاهده می‌شود، برچسب از دو بخش تشکیل شده است که توسط علامت ":" از هم جدا شده‌اند و به ترتیب کلاس اصلی و زیرکلاس را نمایش می‌دهند. در این مثال، کلاس اصلی نشان می‌دهد که پاسخ باید یک شخص یا یک گروه را مشخص کند، در حالی که زیرکلاس نشان می‌دهد که ما می‌خواهیم یک فرد خاص را (به طور طبیعی از طریق نام او) شناسایی کنیم. لازم به ذکر است که بازنمایی ارائه شده توسط GloVe شامل تمامی کلمات (۹۱۲۳ کلمه) موجود در مجموعه‌داده می‌شود و بدین ترتیب نیازی به کارهای بیشتر در این زمینه نیست. با این حال، پس از استخراج داده‌ها از مجموعه داده، ابتدا یک پیش‌پردازش اولیه انجام می‌شود که شامل پاکسازی داده‌ها از هر نویسه و علامت نگارشی غیرمفید (به استثنای علامت سوال) می‌شود. در جلوتر مشاهده می‌شود که برای عمل به این گفته از دستور زیر استفاده شده است تا علامت سوال باقی بماند:

```
text = re.sub(r'[^a-zA-Z0-9\?]+', ' ', text)
```

علاوه بر این، تمام کلمات موجود در رشته‌ها به حروف کوچک تبدیل می‌شوند تا از تکرار کدگذاری‌های چندگانه برای کلمات و عبارات یکسان جلوگیری شود. حال در گام اول و برای راحتی، مجموعه‌داده را در گوگل‌درایو بارگذاری می‌کنیم. هم‌چنین، فراخوانی این فایل بدون نیاز به Mountکردن و استفاده از gdown در محیط گوگل‌کولب ممکن است با خطا مواجه شود که با استفاده از دستورات زیر مشکل حل می‌شود (منبع). در ادامه هم دستوراتی برای دریافت اطلاعات مربوط به نهفته‌سازی Glove نوشته‌ایم.

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1lDpsLB-erPd4rvvRguui6i06h1hTKE1K
!gdown 1HctYMsZ-V7t7ipdLOOO5S-WNWH5X_l4b
!gdown 1nZ-JnAIQ5fku0FhIFUHCFKguCtEyuNNB

# Load pre-trained GloVe embeddings
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove*.zip

nltk.download('punkt')
```

در ادامه دستورات مختلفی را برای آماده‌سازی و پیش‌پردازش داده‌ها می‌نویسیم. در این دستورات، متن داده‌های آموزشی و آزمون پاکسازی می‌شود. ابتدا تمامی کاراکترهای خاص و ناخواسته از متن حذف می‌شوند و سپس متن به حروف کوچک تبدیل می‌شود. در ادامه، متن پاکسازی‌شده توسط تابع word_tokenize توکن‌بندی می‌شود. این فرایند شامل تقسیم متن به واحدهای جداگانه که معمولاً کلمات هستند می‌باشد. سپس، باید مجموعه‌ی کلمات توقف یا متوقف را دریافت کنیم. کلمات متوقف یا stop words در پردازش زبان طبیعی به کلماتی اشاره دارند که اغلب در متن‌ها به صورت تکراری و بدون ارزش اطلاعاتی ظاهر می‌شوند. این کلمات عموماً از نظر معنایی کم اهمیت هستند و بیشتر برای ایجاد جریان و قواعد گرامری در جملات استفاده می‌شوند. مثال‌هایی از کلمات متوقف شامل "the"، "is"، "and"، "a"، "an" و "in" می‌باشند. در پردازش متن، حذف کلمات متوقف می‌تواند منجر به بهبود عملکرد الگوریتم‌ها و مدل‌ها در وظایفی مانند تحلیل احساسات، تشخیص موضوع و دسته‌بندی متن شود. با حذف کلمات متوقف، تمرکز بیشتری بر کلمات کلیدی و اطلاعات معنایی موجود در متن داریم. با استفاده از nltk.download('stopwords')، منابع مربوط به کلمات متوقف دانلود می‌شوند. سپس کلمات متوقف انگلیسی در stop_words قرار می‌گیرند. در نهایت، این کلمات از متن پاکسازی‌شده حذف می‌شوند و نتیجه در ستون filtered_text قرار می‌گیرد. در ادامه، متن فیلترشده و برچسب‌های داده‌های آموزشی و آزمون را از داده‌های اصلی استخراج می‌کنیم و آن‌ها را به

ترتیب در متغیرهای مربوط به خود قرار می‌دهیم. ابتدا تابع Tokenizer را با تعداد کلمات مجاز برابر با ۴۰۰۰۰۰ ایجاد می‌کنیم و با استفاده از fit_on_texts متن آموزشی را برای توکن‌بندی آماده می‌کنیم. سپس با استفاده از texts_to_sequences، متن‌های آموزشی و آزمون را به دنباله‌های عددی تبدیل می‌کنیم و در متغیرهای X_train و X_test ذخیره می‌کنیم. حداکثر طول دنباله را برابر با بیشترین طول دنباله در X_train در نظر می‌گیریم و سپس با استفاده از pad_sequences، دنباله‌های عددی را پُر می‌کنیم تا طول یکسانی داشته باشند و در متغیرهای مربوطه ذخیره می‌کنیم. تعداد دسته‌ها را در متغیر num_classes ذخیره می‌کنیم. سپس با استفاده از np.eye، برچسب‌های آموزشی و آزمون را به بردارهای دودویی تبدیل می‌کنیم و در متغیرهای y_train و y_test ذخیره می‌کنیم. در ادامهٔ کار، مسیر فایل glove_path را با مسیر فایل Embeddings GloVe مدنظر خود جایگزین می‌کنیم. سپس فایل را خوانده و در آرایهٔ embedding_matrix ذخیره می‌کنیم. در این آرایه، بردارهای embedding برای کلمات موجود در word_index قرار می‌گیرند. در صورتی که برداری برای یک کلمه موجود نباشد، مقدار صفر به آن اختصاص داده می‌شود. لازم به ذکر است که برای نرمال‌سازی از تابع normalize_text استفاده می‌شود. دستورات به شرح زیر هستند:

```
## Method1

import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import warnings
# Ignore all warnings
warnings.filterwarnings("ignore")

train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')


# Clean the text by removing special characters and converting to lowercase
train_data['cleaned_text'] = train_data['text'].str.replace('[^\w\s\?]', '').str.lower()
test_data['cleaned_text'] = test_data['text'].str.replace('[^\w\s\?]', '').str.lower()

# Tokenize the text
train_data['tokenized_text'] = train_data['cleaned_text'].apply(word_tokenize)
test_data['tokenized_text'] = test_data['cleaned_text'].apply(word_tokenize)

# Remove stopwords
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
train_data['filtered_text'] = train_data['tokenized_text'].apply(lambda tokens: [word for word in
    tokens if word not in stop_words])
test_data['filtered_text'] = test_data['tokenized_text'].apply(lambda tokens: [word for word in
```

```
    tokens if word not in stop_words])


# Get the filtered text and labels
X_train = train_data['filtered_text'].values
y_train = train_data['label-fine'].values
X_test = test_data['filtered_text'].values
y_test = test_data['label-fine'].values

# Tokenize the text and convert to sequences
tokenizer = Tokenizer(num_words=400000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

# Pad the sequences to have the same length
max_sequence_length = max(len(seq) for seq in X_train)
X_train = pad_sequences(X_train, maxlen=max_sequence_length)
X_test = pad_sequences(X_test, maxlen=max_sequence_length)

# Convert labels to categorical
num_classes = train_data['label-fine'].nunique()
y_train = np.eye(num_classes)[y_train]
y_test = np.eye(num_classes)[y_test]

# Define function for text normalization
def normalize_text(text):
    text = re.sub(r'[^a-zA-Z0-9\?]+', ' ', text)
    text = text.lower()
    return text

# Normalize the filtered text
X_train = [normalize_text(text) for text in X_train]
X_test = [normalize_text(text) for text in X_test]

# Replace 'glove_path' with the path to your GloVe embeddings file
glove_path = '/content/glove.6B.300d.txt'
embeddings_index = {}
with open(glove_path, encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
```

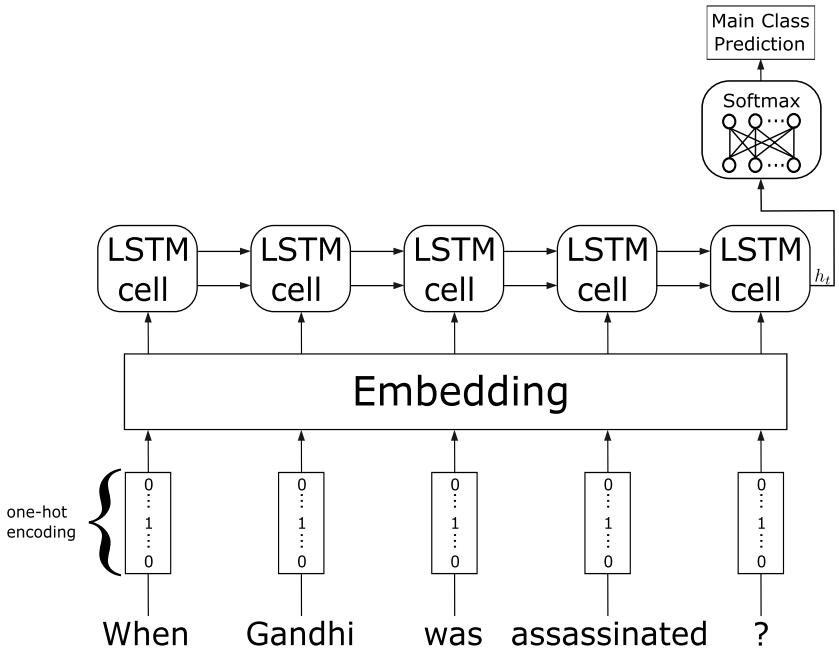
```

# Create an embedding matrix
embedding_dim = 300
word_index = tokenizer.word_index
num_words = min(400000, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, embedding_dim))
for word, i in word_index.items():
    if i >= num_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector


## Method2

# Import necessary libraries
import pandas as pd
import numpy as np
import re
import nltk
from nltk.tokenize import word_tokenize
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder

# Load dataset
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Read 'QA_data.csv' with 'errors' parameter and pass the file object to 'pd.read_csv()'
with open('QA_data.csv', 'r', encoding='utf-8', errors='replace') as file:
    qa_data = pd.read_csv(file)

# Define function for text normalization
def normalize_text(text):
    text = re.sub(r'[^a-zA-Z0-9\?]+', ' ', text)
    text = text.lower()
    return text

# Normalize text
train_data['text'] = train_data['text'].apply(normalize_text)
test_data['text'] = test_data['text'].apply(normalize_text)
qa_data['text'] = qa_data['text'].apply(normalize_text)

```

```
# Tokenize the text
nltk.download('punkt')
train_data['tokens'] = train_data['text'].apply(word_tokenize)
test_data['tokens'] = test_data['text'].apply(word_tokenize)
qa_data['tokens'] = qa_data['text'].apply(word_tokenize)

# Load pre-trained GloVe embeddings
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove*.zip

# Load GloVe embeddings into a dictionary
embeddings_index = {}
with open('glove.6B.300d.txt') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

# Prepare tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data['tokens'])

# Convert tokens to sequences
train_data['sequences'] = tokenizer.texts_to_sequences(train_data['tokens'])
test_data['sequences'] = tokenizer.texts_to_sequences(test_data['tokens'])
qa_data['sequences'] = tokenizer.texts_to_sequences(qa_data['tokens'])

# Pad sequences
maxlen = max(train_data['sequences'].apply(len))
train_padded_sequences = pad_sequences(train_data['sequences'], maxlen=maxlen)
test_padded_sequences = pad_sequences(test_data['sequences'], maxlen=maxlen)
qa_padded_sequences = pad_sequences(qa_data['sequences'], maxlen=maxlen)

# Encode labels
encoder = LabelEncoder()
encoder.fit(train_data['label-coarse'])
train_data['encoded_labels'] = encoder.transform(train_data['label-coarse'])
test_data['encoded_labels'] = encoder.transform(test_data['label-coarse'])
qa_data['encoded_labels'] = encoder.transform(qa_data['label-coarse'])

# Create an embedding matrix
embedding_dim = 300
vocab_size = len(tokenizer.word_index) + 1
embedding_matrix = np.zeros((vocab_size, embedding_dim))
```

```
for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

۳.۱ پاسخ قسمت ۳

مقاله در مورد مدل اول این‌گونه بیان می‌کند که تمام دنباله‌ها را که به شبکه وارد می‌شوند، به یک بردار با ابعاد ثابت، که ابعاد آن برابر با تعداد دسته‌بندی‌های اصلی است نگاشت می‌کنیم. به عبارت دیگر در این روش، هر کلمهٔ تشکیل‌دهندهٔ سوال (طبق ترتیب زمانی و پس از انجام نگاشت از طریق لایهٔ تعبیه) وارد شبکه LSTM شود و حالت و طبقهٔ اصلی نهایی آن از طریق یک شبکهٔ عصبی کلاسیک با فعال‌سازی Softmax در یکی از شش کلاس پیش‌بینی اصلی پیش‌بینی می‌شود. الگوریتم از Backpropagation through time (BPTT) و تابع هزینهٔ Categorical Cross-Entropy استفاده می‌کند. نمایی از این مدل در شکل ۶ نشان داده شده است. برای پیاده‌سازی دستورات مربوط به این مدل، ابتدا داده‌های آموزش و آزمون که از مرحلهٔ قبل ایجاد شده‌اند را



شکل ۶: نمایی از مدل اول.

به‌صورت کامل معین و آماده می‌کنیم. تابع `encode_labels` در زبان برنامه‌نویسی پایتون، وظیفهٔ تبدیل برچسب‌ها به بردارهای One-Hot را دارد. این تابع دو ورودی دریافت می‌کند: `labels` که برچسب‌های ورودی را نشان می‌دهد و `num_classes` که تعداد کلاس‌ها را نمایش می‌دهد. در داخل تابع، با استفاده از تابع tf.keras.utils.to_categorical، برچسب‌ها به بردارهای One-Hot تبدیل می‌شوند (مطابق خواست مقاله در شکل ۶). هر برچسب به یک بردار با طول `num_classes` تبدیل می‌شود. در این بردار، مکان متناظر با شمارهٔ کلاس برچسب، با مقدار ۱ و سایر مکان‌ها با مقدار ۰ پر می‌شود. به عبارت دیگر، هر برچسب به صورت یک بردار با مقدار ۱ در مکان متناظر با کلاس و مقادیر ۰ در سایر مکان‌ها نمایش داده می‌شود.

برای تعریف مدل یک تابع با نام `create_lstm_model` تعریف می‌کنیم که یک مدل LSTM را ایجاد می‌کند. مدل از ساختار Sequential استفاده می‌کند، که به ترتیب لایه‌های Embedding، LSTM و Dense را اضافه می‌کند. لایهٔ Embedding از واژگان و جملات ورودی استفاده می‌کند و وزن‌های جدول تعبیه (`embedding_matrix`) را استفاده می‌کند. وزن‌های جدول تعبیه از قبل آموزش دیده‌اند و برای نمایش برداری واژگان و تبدیل واژگان به بردارهای عددی استفاده می‌شوند. ورودی لایهٔ Embedding دارای اندازهٔ واژگان (`vocab_size`) و بعد بردارهای تعبیه (`embedding_dim`) است و همچنین طول ورودی (`input_length`) را نیز تعیین می‌کنیم. لایهٔ بعدی یک LSTM است که تعداد واحدهای LSTM را با پارامتر `h_dim` تعیین می‌کند. این لایه به صورت توالی بازگشتی (`return_sequences=True`) است تا مطابق شکل ۶ خروجی‌ها را به لایه بعدی انتقال دهد. تابع فعال‌سازی لایه LSTM هم تانژانت هیپربولیک (tanh) در نظر گرفته‌ایم. سپس، می‌توان یک لایهٔ دیگر LSTM با تعداد واحدهای LSTM مشابه قبلی اضافه کرد. این لایه در حالت پیش‌فرض تک لایه LSTM است که خروجی نهایی را تولید می‌کند. در نهایت، یک لایهٔ Dense با تابع فعال‌سازی softmax برای تولید احتمالات کلاس‌ها اضافه می‌شود. تعداد کلاس‌ها با پارامتر `num_classes` مشخص می‌شود. مدل با استفاده از تابع هزینه `categorical_crossentropy` و بهینه‌ساز `Adam` (با نرخ یادگیری ۰.۰۰۱) تنظیم می‌شود. معیارهای دقت و اتلاف نیز برای مدل مشخص شده است. در نهایت، مدل ساخته شده با استفاده از توابع `compile` و `return` آمادهٔ استفاده و آموزش می‌شود. سپس فراپارامترها را تعیین می‌کنیم.

در ادامه توابعی برای رسم نمودارها و محاسبات لازم برای حلّ سوال ایجاد می‌کنیم. تابع `save_plots_to_pdf` را برای ذخیرهٔ نمودارها به صورت فایل PDF نوشته‌ایم. در این تابع نمودارهای دقت و اتلاف برای داده‌های آموزش و اعتبارسنجی رسم شده و پارامترهای مهم مدنظر سوال (مانند `h_dim`) در قسمت عنوان نوشته می‌شوند. نمودارهای هم ذخیره می‌شوند و هم در محیط کولب نمایش داده می‌شوند. با استفاده از `plt.close`، منابع گرافیکی آزاد می‌شوند و نمودارها بسته می‌شوند. تابع `save_confusion_matrix_to_pdf` هم برای رسم ماتریس درهم‌ریختگی به شکلی مناسب و به صورتی که در آن پارامترهای مهم مدنظر سوال (مانند `h_dim`) نشان داده شود نوشته شده است. با نوشتن این توابع به سراغ نوشتن بخش اصلی کد می‌رویم. این قسمت با استفاده از مجموعه‌ای از پارامترهای انتخاب شده و با استفاده از مدل LSTM پیش‌تر تعریف‌شده، مدل‌های مختلف را آموزش می‌دهد و نتایج عملکرد آن‌ها را گزارش می‌دهد. در بخش آموزش این کد، برای هر مقدار ابعاد h، یک مدل LSTM با استفاده از تابع `create_lstm_model` ایجاد می‌شود و با استفاده از داده‌های آموزش و اعتبارسنجی، مدل آموزش داده می‌شود. در بخش ارزیابی هم دقت مدل روی مجموعه‌های آموزش و آزمون محاسبه می‌شود. همچنین، از معیارهای دیگر اشاره‌شده در صورت سوال هم برای ارزیابی عملکرد مدل استفاده می‌شود. درنهایت نمودارهای مختلف از جمله نمودارهای دقت، ماتریس‌های درهم‌ریختگی و معیارهای دیگر برای هر مدل در یک فایل PDF ذخیره می‌شوند. قبل از ادامه به ابعاد بعدی h، برنامه طوری تنظیم شده که ۲ ثانیه منتظر می‌ماند. این کد به صورت سیستماتیک و در چندین مرحله، مدل LSTMپایه را با ابعاد مختلف آموزش می‌دهد و عملکرد آن را بررسی می‌کند. همچنین، نمودارها و ماتریس‌های درهم‌ریختگی برای هر مدل ذخیره می‌کند تا بتوان ارزیابی دقیق‌تری انجام داد. با ارائهٔ این توضیحات تمام دستورات مربوطه در برنامهٔ ۱ آورده شده است. به‌دلیل عدم شفافیت کامل دستورات مورد استفادهٔ مقاله از قبیل عدم مشخص‌بودن اندازهٔ دسته، تعداد دوره، درنظرگرفتن دادهٔ اعتبارسنجی و غیره، از تجربیات اندوختهٔ تمرین‌های گذشته برای رسیدن به یک نتیجهٔ مناسب و قابل قبول استفاده کرده‌ایم. مثلاً مدل را در چند حالت مختلف تشکیل داده‌ایم و بهترین نتیجه را این‌جا ذکر کرده‌ایم. هم‌چنین دو حالت در نظر گرفته‌ایم. یک حالت نزدیک به آن چیزی که احتمالاً مقاله پیش رفته (بدون دادهٔ اعتبارسنجی و با حدود ۵۰ دورهٔ آموزش) و یکی هم با دادهٔ اعتبارسنجی. دستورات آورده‌شده در برنامهٔ ۱ حالت کامل‌تر و با دادهٔ اعتبارسنجی است و برای جلوگیری از ازدحام بیش‌تر دستورات حالتی که در آن دادهٔ اعتبارسنجی در نظر نگرفته‌ایم آورده نشده است. برای قسمت لایهٔ Embedding آموزش را غیرفعال می‌کنیم چون ممکن است آموزش و ایجاد ارتباط بین کلمات نسبت به حالت قدرتمند پیش‌آموزش‌دیدهٔ موجود خوب انجام نپذیرد.

Program 1: Model One, Training, and Evaluation

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.optimizers import Adam
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix
import seaborn as sns
from matplotlib.backends.backend_pdf import PdfPages
import time

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Prepare the data
X_train = train_padded_sequences
y_train = train_data['encoded_labels']
X_test = test_padded_sequences
y_test = test_data['encoded_labels']

# Function to encode labels as one-hot vectors
def encode_labels(labels, num_classes):
    return tf.keras.utils.to_categorical(labels, num_classes=num_classes)

from keras.regularizers import l2

# Function to create the LSTM model
def create_lstm_model(input_length, h_dim, num_classes):
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=input_length
    , weights=[embedding_matrix], trainable=False))
    model.add(LSTM(h_dim, return_sequences=True, activation='tanh'))
    model.add(LSTM(h_dim, activation='tanh'))
    model.add(Dense(num_classes, activation='softmax', kernel_regularizer=l2(0.001)))
    model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=[
    'accuracy'])
    return model
# def create_lstm_model(input_length, h_dim, num_classes):
#     model = Sequential()
#     model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=
    input_length, weights=[embedding_matrix], trainable=False))
#     model.add(LSTM(h_dim, activation='tanh'))
```

```
#     model.add(Dense(num_classes, activation='softmax'))
#     model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
#     return model

# Function to save plots to PDF
def save_plots_to_pdf(filename):
    with PdfPages(filename) as pdf:
        plt.figure(figsize=(10, 4))

        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'], label='Training')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.legend()
        plt.title(f'Accuracy (h_dim = {h_dim})')

        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'], label='Training')
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.legend()
        plt.title(f'Loss (h_dim = {h_dim})')

        pdf.savefig(bbox_inches='tight')

        # Show the plots in Colab output
        plt.show()

    plt.close()

# Function to save confusion matrix to PDF
def save_confusion_matrix_to_pdf(y_true, y_pred, filename):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.title(f'Confusion matrix (h_dim = {h_dim})')
    plt.savefig(filename, bbox_inches='tight')

    # Show the plot in Colab output
    plt.show()
    plt.close()

```

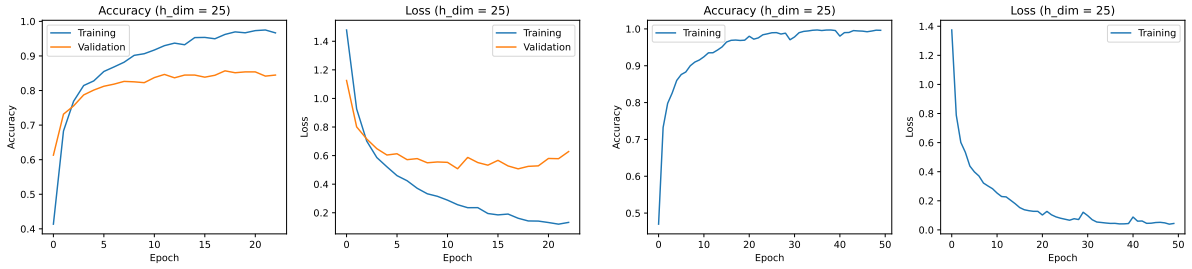
```
# Set hyperparameters
h_dimensions = [25, 50, 75, 100]
epochs = 50
num_classes = 6
batch_size = 64

# Iterate over h_dimensions and train the models
for h_dim in h_dimensions:
    print(f"Training model with h_dim = {h_dim}")
    model = create_lstm_model(maxlen, h_dim, num_classes)

    # Encode labels as one-hot vectors
    y_train_encoded = encode_labels(y_train, num_classes)

    history = model.fit(X_train, y_train_encoded, epochs=epochs, batch_size=batch_size, verbose
    =2)
    train_accuracy = model.evaluate(X_train, encode_labels(y_train, num_classes), verbose=0)[1] *
     100
    test_accuracy = model.evaluate(X_test, encode_labels(y_test, num_classes), verbose=0)[1] *
    100
    print(f"Training set accuracy: {train_accuracy:.2f}%")
    print(f"Test set accuracy: {test_accuracy:.2f}%")

    y_pred_train = np.argmax(model.predict(X_train), axis=1)
    y_pred_test = np.argmax(model.predict(X_test), axis=1)

    train_f1 = f1_score(y_train, y_pred_train, average='weighted')
    test_f1 = f1_score(y_test, y_pred_test, average='weighted')

    train_precision = precision_score(y_train, y_pred_train, average='weighted')
    test_precision = precision_score(y_test, y_pred_test, average='weighted')

    train_recall = recall_score(y_train, y_pred_train, average='weighted')
    test_recall = recall_score(y_test, y_pred_test, average='weighted')

    print(f"Training set F1-score: {train_f1:.2f}")
    print(f"Test set F1-score: {test_f1:.2f}")

    print(f"Training set Precision: {train_precision:.2f}")
    print(f"Test set Precision: {test_precision:.2f}")

    print(f"Training set Recall: {train_recall:.2f}")
    print(f"Test set Recall: {test_recall:.2f}")

    save_plots_to_pdf(f"model_{h_dim}_plots.pdf")
```

```
128
129     print("Confusion matrix (training set):")
130     save_confusion_matrix_to_pdf(y_train, y_pred_train, f"model{h_dim}confusion_matrix_train.pdf"
        )
131
132     print("Confusion matrix (test set):")
133     save_confusion_matrix_to_pdf(y_test, y_pred_test, f"model{h_dim}confusion_matrix_test.pdf")
134
135     print("\n")
136
137     time.sleep(2)  # Wait for 2 seconds before proceeding to the next h_dim
```

نتایج در حالت نزدیک به مقاله، به صورتی است که در در جدول ۱ و شکل ۷ تا شکل ۱۴ نشان داده شده است. همان‌طور که مشاهده می‌شود. تفکیک ۶ کلاس اصلی با دقت قابل قبولی انجام شده است. هرچند برخی نتایج نشان‌داده‌شده برای داده‌های آموزش اهمیت و معنای خاصی ندارند (مانند ماتریس درهم‌ریختگی و...) آن‌ها را هم نشان دادیم تا صرفاً به برخی نکات مانند چالش در تشخیص برچسب ۲ به‌دلیل کم‌تربودن داده در آن دسته اشاره کنیم.

جدول ۱: نتایج مربوط به مدل اول

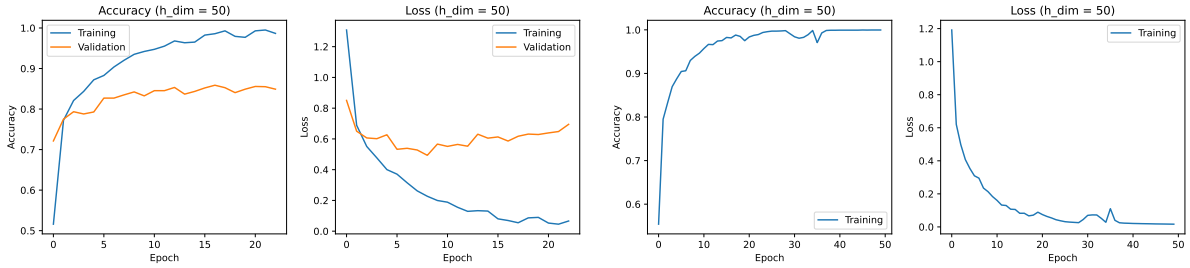
Test Set (%)				Training Set (%)				h Dimension
F1-score	Recall	Precision	Accuracy	F1-score	Recall	Precision	Accuracy	
۸۹	۸۹	۸۹	۸۹.۰۰	۱۰۰	۱۰۰	۱۰۰	۹۹.۸۳	25
۹۰	۹۰	۹۱	۹۰.۲۰	۱۰۰	۱۰۰	۱۰۰	۹۹.۹۶	50
۹۱	۹۲	۹۲	۹۱.۶۰	۱۰۰	۱۰۰	۱۰۰	۹۹.۹۸	75
۹۱	۹۱	۹۱	۹۱	۱۰۰	۱۰۰	۱۰۰	۹۹.۹۸	100



(ب) با اعتبارسنجی (آ) بدون اعتبارسنجی

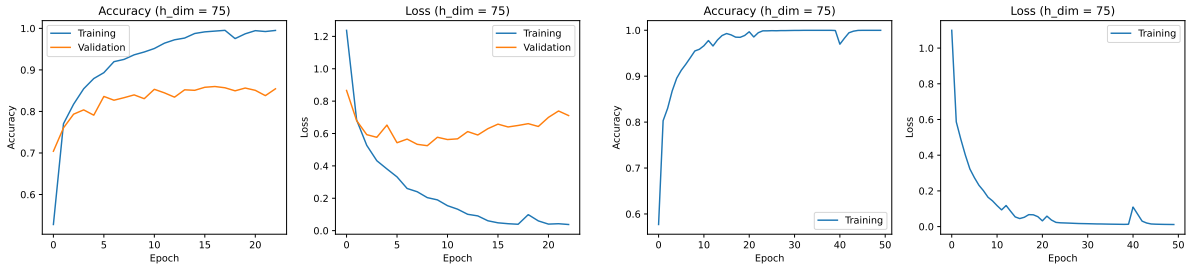
شکل ۷: نمودار دقت و اتلاف – مدل اول – h=25.

مقاله این‌گونه بیان می‌کند که نتایج خوبی که در پیش‌بینی کلاس اصلی به دست آمد، انگیزه‌ای برای توسعهٔ مدل با حفظ بخش اول و اصلی آن شد. در بخش دوم، پیش‌بینی زیرکلاس نیز در نظر گرفته شده است. زیرکلاس باید به عنوان یک ویژگی تخصصی از کلاس اصلی تأثیر بگیرد. ایدهٔ اصلی مقاله برای بخش دوم این است که یک عنصر دیگر به انتهای سوال اضافه شود. این عنصر پرکننده اطلاعات مفیدی ندارد ولی تنها برای ارزیابی خروجی پس از خروجی متناظر با آخرین عنصر سوال لازم است. برای این کار، خروجی قبل از آخرین عنصر سوال مانند حالت قبلی می‌تواند در نظر گرفته شود، در حالی که خروجی آخر تنها



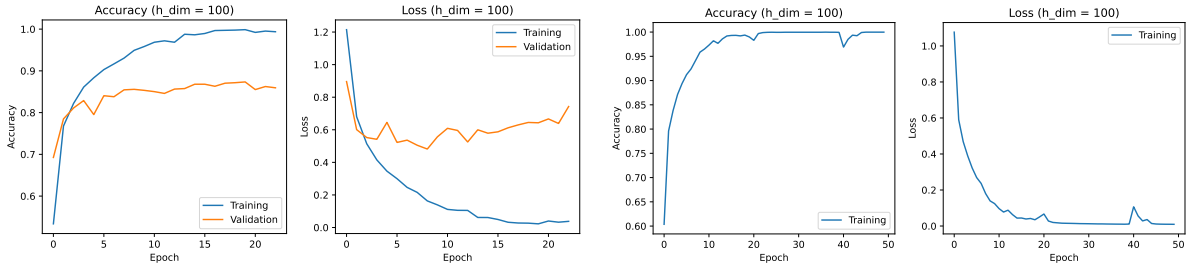
(آ) بدون اعتبارسنجی (ب) با اعتبارسنجی

شکل ۸: نمودار دقت و اتلاف – مدل اول – h=50.



(آ) بدون اعتبارسنجی (ب) با اعتبارسنجی

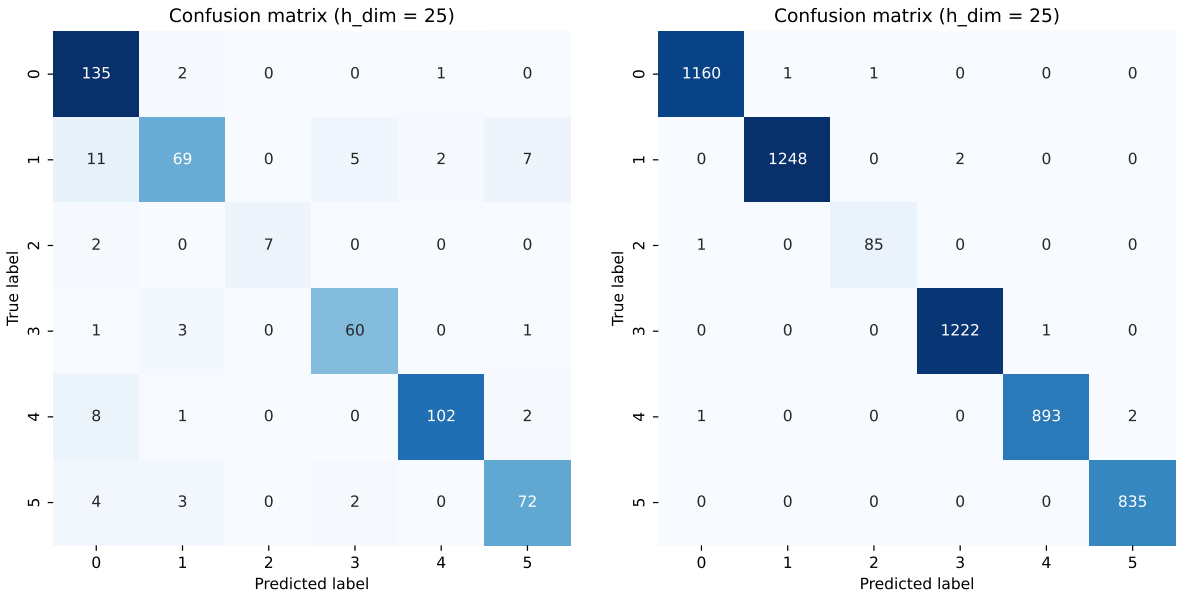
شکل ۹: نمودار دقت و اتلاف – مدل اول – h=75.



(آ) بدون اعتبارسنجی (ب) با اعتبارسنجی

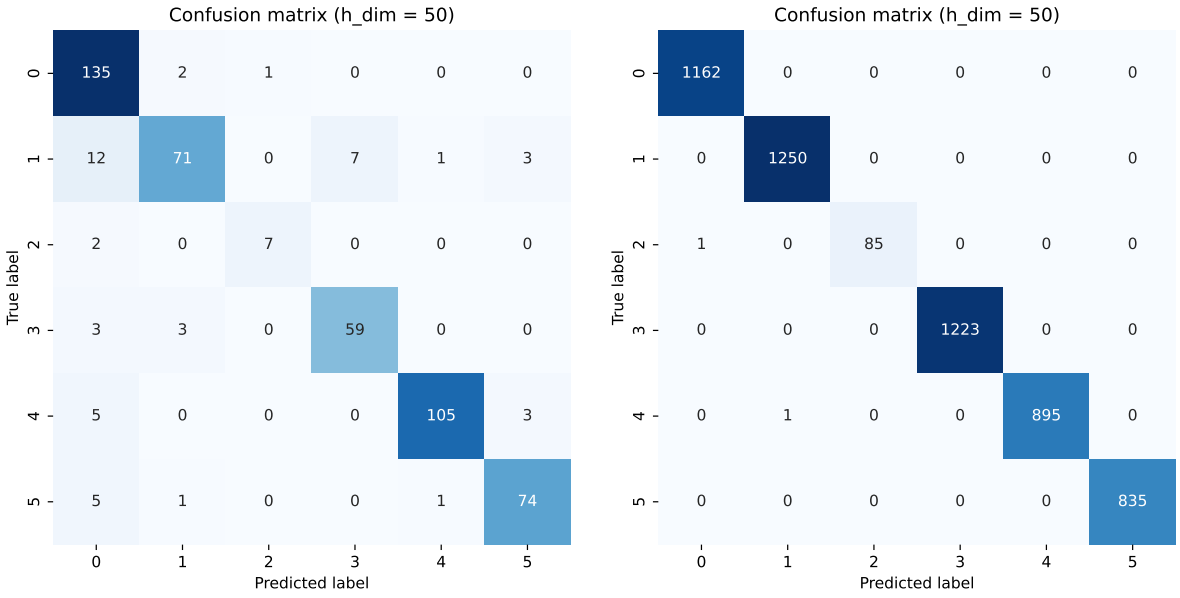
شکل ۱۰: نمودار دقت و اتلاف – مدل اول – h=100.

به حالت قبلی وابسته است (که هر دو با h_{t-1} و c_{t-1} نمایش داده می‌شوند) زیرا لایهٔ نهفته‌ساز به محاسبهٔ h_t کمکی نمی‌کند. با آموزش شبکه به صورت نظارت شده برای ارتباط زیرکلاس با این خروجی، وابستگی‌ای به هر دو دسته‌بندی اصلی و خود سوال ایجاد خواهد شد. دو اطلاعاتی که از بخش بازگشتی به بیرون می‌آیند، توسط دو لایهٔ تماماً‌متصل با فعال‌ساز Softmax تمییز داده و طبقه‌بندی می‌شوند؛ بنابراین، دنبالهٔ ورودی را به دو بردار با اندازهٔ ثابت که نمایندهٔ کلاس اصلی و زیرکلاس مرتبط با آن است تبدیل می‌کنند. نمایش شماتیک مدل دوم که توصیف شد را می‌توان در شکل ۱۵ مشاهده کرد. الگوریتم BPTT همچنان برای آموزش مدل استفاده می‌شود، اما با تفاوتی نسبت به مورد قبلی؛ که به تابع اتلاف دوگانه (به‌دلیل وجود دو نوع دسته‌بندی) بازمی‌گردد. برای پیاده‌سازی مدل دوم دستورات آورده‌شده در برنامهٔ ۲ را نوشته‌ایم. ابتدا مدل را به چند صورت پیاده‌سازی می‌کنیم. اولین و مهم‌ترین شکل پیاده‌سازی مدل که نتایج خوبی به دست داده را در این‌جا معرفی می‌کنیم. ابتدا لایهٔ ورودی را



Test (ب) Train (آ)

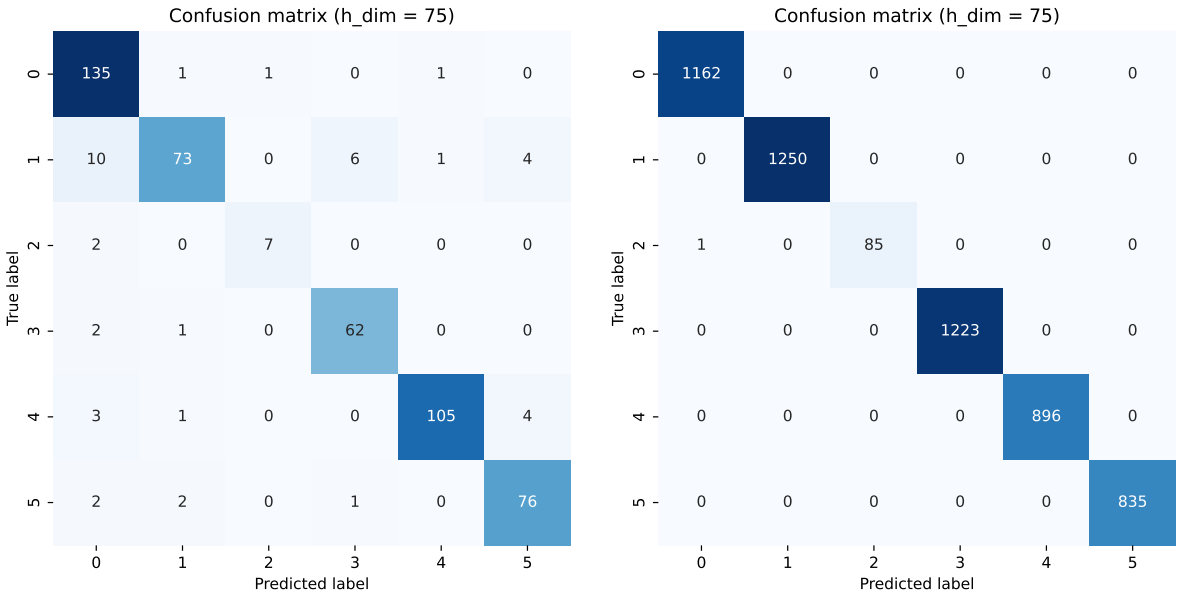
شکل ۱۱: ماتریس درهم‌ریختگی – مدل اول – h=25.



Test (ب) Train (آ)

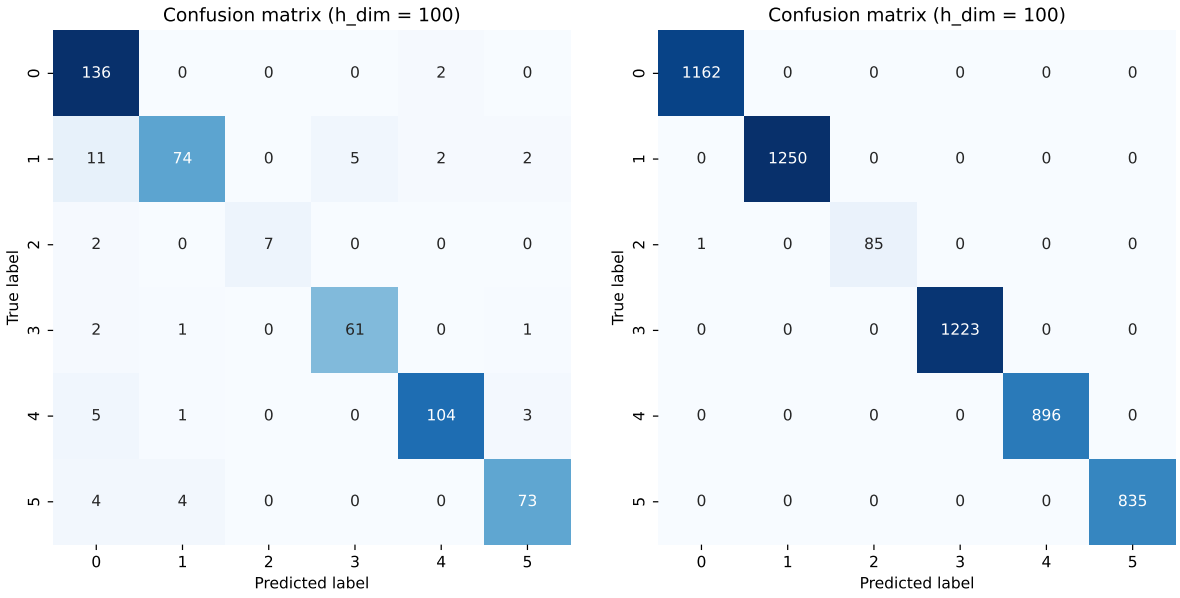
شکل ۱۲: ماتریس درهم‌ریختگی – مدل اول – h=50.

ایجاد می‌کنیم که اندازهٔ ورودی آن با توجه به `input_length` تعیین می‌شود. سپس لایهٔ Embedding را ایجاد می‌کنیم که برای تبدیل ورودی به بردارهای تعبیه استفاده می‌شود. ابعاد واژگان واحد تعبیه را با `vocab_size` و ابعاد بردارهای واحد تعبیه را با `embedding_dim` نشان می‌دهیم. ماتریس وزن‌های تعبیه `embedding_matrix` برای تعبیهٔ واژگان استفاده می‌شود. این لایه در



Test (ب) Train (آ)

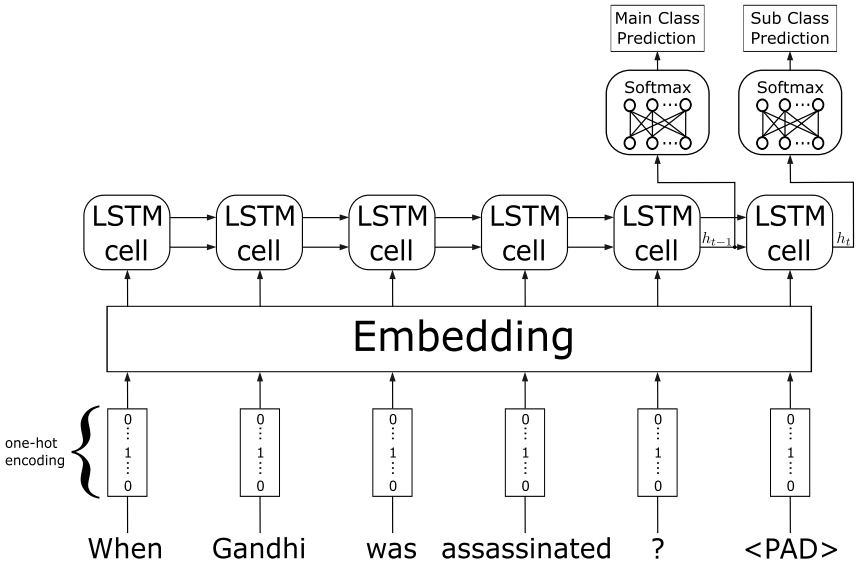
شکل ۱۳: ماتریس درهم‌ریختگی – مدل اول – h=75.



Test (ب) Train (آ)

شکل ۱۴: ماتریس درهم‌ریختگی – مدل اول – h=100.

حالت غیرقابل آموزش قرار داده شده است. در ادامه یک لایه LSTM ایجاد می‌کنیم و آن را روی ورودی تعبیه‌شده اِعمال می‌کنیم. تعداد واحدهای LSTM در این لایه با `h_dim` تعیین می‌شود. با تنظیم پارامتر `return_sequences=True` خروجی LSTM به صورت توالی برگردانده می‌شود و با تنظیم `return_state=True` وضعیت پنهان آخرین زمان برگردانده می‌شود. درواقع، این



شکل ۱۵: نمایی از مدل دوم.

لایهٔ LSTM با دریافت ورودی دنباله‌ای (بردارهای تعبیه)، خروجی‌ها و وضعیت‌های آخرین زمان در دنباله را تولید می‌کند. نتایج این عملیات در سه متغیر `lstm_outputs`، `state_h` و `state_c` ذخیره می‌شود. متغیر `lstm_outputs` حاوی خروجی‌های LSTM برای تمام زمان‌ها در دنباله است. اگر طول دنباله T باشد، `lstm_outputs` یک تنسور با ابعاد (batch_size, T, h_dim) است. هر عنصر از این تنسور نشان‌دهندهٔ خروجی LSTM در زمان مربوطه است. متغیر `state_h` نمایندهٔ حالت پنهان آخرین زمان در دنباله است. اگر `h_dim` اندازهٔ بردار حالت پنهان باشد، `state_h` یک تانسور با ابعاد (batch_size, h_dim) است. متغیر `state_c` حاوی وضعیت سلول آخرین زمان در دنباله و نیز یک تنسور با ابعاد (batch_size, h_dim) است. در قسمتی از دستورات مقداردهی چندگانه به کمک عملگر تجزیه‌ای (=) استفاده شده است؛ بنابراین مقادیر محاسبه‌شده توسط لایهٔ LSTM به‌ترتیب در سه متغیر `lstm_outputs`، `state_h` و `state_c` قرار می‌گیرند. در ادامه لایه‌های تماماً‌متصل را تعریف می‌کنیم. تعداد واحدها را برابر با تعداد کلاس‌های موجود در متغیر `encoder_main` در نظر می‌گیریم و از فعالساز softmax برای تولید احتمالات دسته‌بندی استفاده می‌کنیم. خروجی این لایه برچسب‌های اصلی است. هم‌چنین یک لایهٔ تماماً‌متصل دیگر بر اساس تعداد کلاس‌های موجود در متغیر `encoder_sub` در خروجی زمان آخر LSTM تعریف می‌کنیم. بعد از تعریف لایه‌ها، مدل کلی را تعریف و مشخصات آموزش، از نوع و پارامترهای بهینه‌ساز و تابع اتلاف تا سایر موارد را به دقت تعریف می‌کنیم. در ادامه تابع `save_plots_to_pdf` را که در قسمت قبل ایجاد کرده بودیم برای نمایش نمودارهای مربوط به دقت و خطا برای دو برچسب اصلی و فرعی بازنویسی می‌کنیم. دستورات مربوط به رسم ماتریس درهم‌ریختگی هم در تابع `save_confusion_matrix_to_pdf` به‌صورتی مناسب بازنویسی شده‌اند که دو ماتریس درهم‌ریختگی را برای کلاس‌های یکتای موجود در برچسب‌های اصلی و فرعی نشان دهند. باید دقت داشت که برخی شماره‌های کلاس در برچسب فرعی دیتاست تست ناموجود هستند؛ مانند کلاس ۱.

```
Available classes in the test dataset:
[ 0  2  3  4  5  6  7  8  9 10 11 12 13 14 17 18 19 20 21 22 23 24 25 26
 27 28 29 30 31 33 34 35 37 38 40 41 44 45 46]
```

در ادامه و پس از نوشتن تابعی برای محاسبه و نمایش معیارهای ارزیابی، مشابه قسمت قبل بخشی را برای آماده‌سازی داده‌ها

می‌نویسیم. برای این کار ابتدا داده‌های آموزش و آزمون به‌صورت جداگانه بارگذاری می‌شوند. سپس تابعی برای تصحیح و نرمال‌سازی متن ایجاد می‌شود. ابتدا تمام کاراکترهای غیرضروری و اعداد را حذف و با فاصله جایگزین می‌کنیم و سپس تمام حروف را به حروف کوچک تبدیل می‌کنیم. در ادامه، از کلاس Tokenizer برای تبدیل متن به توکن‌ها استفاده می‌کنیم. ابتدا یک نمونه از کلاس Tokenizer ایجاد می‌کنیم و با استفاده از تابع fit_on_texts، توکن‌ها از متن‌های آموزش استخراج می‌شوند و در داخل تابع ایجاد شده ذخیره می‌شوند. سپس توکن‌ها را به توالی‌ها تبدیل می‌کنیم. از تابع texts_to_sequences برای تبدیل متن‌های آموزش و آزمون به توالی‌های عددی استفاده می‌شود. طول بزرگترین توالی در توالی‌های آموزش را مبنا گرفته و با استفاده از تابع pad_sequences، توالی‌ها را به همان طول پدینگ می‌کنیم. هم‌چنین، برچسب‌های اصلی و فرعی را به‌صورت One-Hot رمزگذاری می‌کنیم. از کلاس LabelEncoder برای رمزگذاری برچسب‌ها استفاده می‌کنیم. در انتهای بخش مربوط به آماده‌سازی داده، برای بارگیری و استفاده از توکن‌های GloVe به منظور ساختن ماتریس تعبیه برای استفاده در مدل دستوراتی مشابه قبل نوشته‌ایم. در اینجا، ماتریس تعبیه نشان می‌دهد که هر توکن یا کلمه در مدل با یک بردار عددی نشان داده می‌شود. در نهایت یک حلقه می‌نویسیم تا به‌ازای پارامترهای تعیین‌شده مدل را آموزش دهد، ارزیابی کند و تمامی موارد مورد نیاز را در خروجی نمایش دهد.

Program 2: Model Two, Training, and Evaluation

```python
import pandas as pd
import numpy as np
import re
import nltk
import tensorflow as tf
from nltk.tokenize import word_tokenize
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
import time
import warnings
from tensorflow.keras.utils import to_categorical

# Ignore all warnings
warnings.filterwarnings("ignore")

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Set hyperparameters
h_dimensions = [25, 100]
```

```
epochs = 50
batch_size = 32

# Function to create the LSTM model
def create_lstm_model(input_length, h_dim):
    input_layer = Input(shape=(input_length,))
    embedding_layer = Embedding(vocab_size, embedding_dim, weights=[embedding_matrix], trainable=
    False)(input_layer)
    lstm_layer = LSTM(h_dim, activation='tanh', return_sequences=True, return_state=True)
    lstm_outputs, state_h, state_c = lstm_layer(embedding_layer)
    main_output = Dense(len(encoder_main.classes_), activation='softmax', name='main_output')(
    state_h)
    sub_output = Dense(len(encoder_sub.classes_), activation='softmax', name='sub_output')(
    lstm_outputs[:, -1, :])
    model = Model(inputs=input_layer, outputs=[main_output, sub_output])
    model.compile(optimizer=Adam(learning_rate=0.001), loss={'main_output': '
    sparse_categorical_crossentropy', 'sub_output': 'sparse_categorical_crossentropy'}, metrics=[
    'accuracy'])
    return model

# Function to save plots to PDF
def save_plots_to_pdf(filename, history, h_dim):
    with PdfPages(filename) as pdf:
        fig, axs = plt.subplots(2, 2, figsize=(10, 8))

        axs[0, 0].plot(history.history['main_output_accuracy'], label='Training')
        axs[0, 0].plot(history.history['val_main_output_accuracy'], label='Validation')
        axs[0, 0].set_xlabel('Epoch')
        axs[0, 0].set_ylabel('Accuracy')
        axs[0, 0].legend()
        axs[0, 0].set_title(f'Main Accuracy (h_dim = {h_dim})')

        axs[0, 1].plot(history.history['main_output_loss'], label='Training')
        axs[0, 1].plot(history.history['val_main_output_loss'], label='Validation')
        axs[0, 1].set_xlabel('Epoch')
        axs[0, 1].set_ylabel('Loss')
        axs[0, 1].legend()
        axs[0, 1].set_title(f'Main Loss (h_dim = {h_dim})')

        axs[1, 0].plot(history.history['sub_output_accuracy'], label='Training')
        axs[1, 0].plot(history.history['val_sub_output_accuracy'], label='Validation')
        axs[1, 0].set_xlabel('Epoch')
        axs[1, 0].set_ylabel('Accuracy')
        axs[1, 0].legend()
        axs[1, 0].set_title(f'Sub Accuracy (h_dim = {h_dim})')
```

```
        axs[1, 1].plot(history.history['sub_output_loss'], label='Training')
        axs[1, 1].plot(history.history['val_sub_output_loss'], label='Validation')
        axs[1, 1].set_xlabel('Epoch')
        axs[1, 1].set_ylabel('Loss')
        axs[1, 1].legend()
        axs[1, 1].set_title(f'Sub Loss (h_dim = {h_dim})')

        plt.subplots_adjust(hspace=0.5)  # Adjust the vertical spacing between subplots

        pdf.savefig(bbox_inches='tight')

        # Show the plots
        plt.show()

    plt.close()

# Function to save confusion matrix to PDF
def save_confusion_matrix_to_pdf(y_true, y_pred, filename, label_type):
    unique_classes = np.unique(y_true)
    num_classes = len(unique_classes)

    cm = confusion_matrix(y_true, y_pred, labels=unique_classes)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  # Normalize the confusion
     matrix

    # Calculate the appropriate figure size based on the number of classes
    figsize = max(10, num_classes / 2)

    plt.figure(figsize=(figsize, figsize))
    sns.heatmap(cm_normalized, annot=True, fmt='.2f', cmap='Blues', cbar=False)

    # Adjust the font size of the numbers inside the heatmap
    annot_fontsize = max(6, 120 // num_classes)
    plt.tick_params(axis='both', labelsize=annot_fontsize)

    # Map the class indices to their corresponding labels
    class_labels = encoder_sub.inverse_transform(unique_classes)

    plt.xticks(np.arange(num_classes) + 0.5, class_labels, rotation='vertical')
    plt.yticks(np.arange(num_classes) + 0.5, class_labels, rotation='horizontal')

    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.title(f'Normalized Confusion Matrix ({label_type})')
```

```
    plt.savefig(filename, bbox_inches='tight')

    # Show the plot
    plt.show()
    plt.close()


# Print the available classes in the test dataset
print("Available classes in the test dataset:")
print(np.unique(test_encoded_sub_labels))

# Function to print evaluation metrics
def print_evaluation_metrics(y_true, y_pred, label_type):
    accuracy = np.mean(y_true == y_pred)
    precision = precision_score(y_true, y_pred, average='macro')
    recall = recall_score(y_true, y_pred, average='macro')
    f1 = f1_score(y_true, y_pred, average='macro')

    print(f'Evaluation metrics ({label_type}):')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1-score: {f1:.4f}')

# Load dataset
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Define function for text normalization
def normalize_text(text):
    text = re.sub(r'[^a-zA-Z0-9\?]+', ' ', text)
    text = text.lower()
    return text

# Normalize text
train_data['text'] = train_data['text'].apply(normalize_text)
test_data['text'] = test_data['text'].apply(normalize_text)

# Tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data['text'])

# Convert tokens to sequences
train_sequences = tokenizer.texts_to_sequences(train_data['text'])
test_sequences = tokenizer.texts_to_sequences(test_data['text'])
```

```
159
160 # Pad sequences
161 maxlen = max(len(seq) for seq in train_sequences)
162 train_padded_sequences = pad_sequences(train_sequences, maxlen=maxlen)
163 test_padded_sequences = pad_sequences(test_sequences, maxlen=maxlen)
164
165 # Encode main labels
166 encoder_main = LabelEncoder()
167 encoder_main.fit(train_data['label-coarse'])
168 train_encoded_main_labels = encoder_main.transform(train_data['label-coarse'])
169 test_encoded_main_labels = encoder_main.transform(test_data['label-coarse'])
170
171 # Encode sub labels
172 encoder_sub = LabelEncoder()
173 encoder_sub.fit(train_data['label-fine'])
174 train_encoded_sub_labels = encoder_sub.transform(train_data['label-fine'])
175 test_encoded_sub_labels = encoder_sub.transform(test_data['label-fine'])
176
177 # Convert main labels to one-hot vectors
178 y_main_train = to_categorical(train_encoded_main_labels)
179 y_main_test = to_categorical(test_encoded_main_labels)
180
181 # Convert sub labels to one-hot vectors
182 y_sub_train = to_categorical(train_encoded_sub_labels)
183 y_sub_test = to_categorical(test_encoded_sub_labels)
184
185 # Load GloVe embeddings into a dictionary
186 embeddings_index = {}
187 with open('glove.6B.300d.txt') as f:
188     for line in f:
189         values = line.split()
190         word = values[0]
191         coefs = np.asarray(values[1:], dtype='float32')
192         embeddings_index[word] = coefs
193
194 # Create an embedding matrix
195 embedding_dim = 300
196 vocab_size = len(tokenizer.word_index) + 1
197 embedding_matrix = np.zeros((vocab_size, embedding_dim))
198 for word, i in tokenizer.word_index.items():
199     embedding_vector = embeddings_index.get(word)
200     if embedding_vector is not None:
201         embedding_matrix[i] = embedding_vector
202
203 # Iterate over h_dimensions and train the models
```

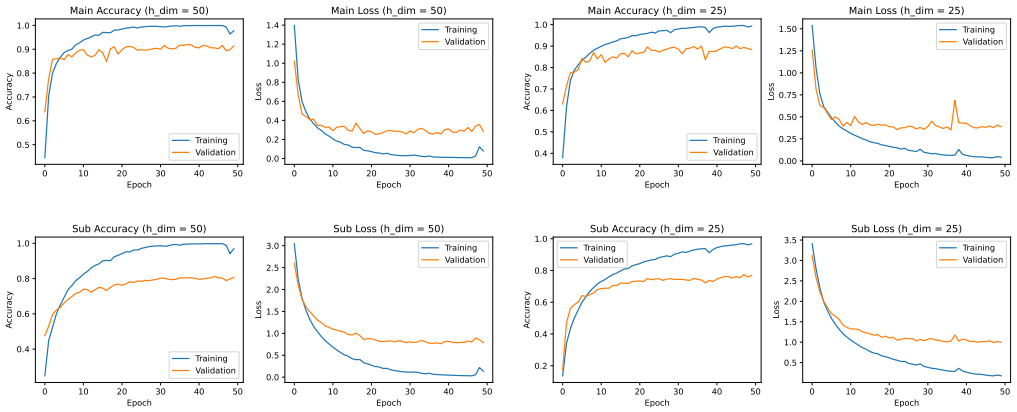
```
for h_dim in h_dimensions:
    print(f"Training model with h_dim = {h_dim}")
    model = create_lstm_model(maxlen, h_dim)

    # Fit the model
    history = model.fit(
        train_padded_sequences,
        {'main_output': train_encoded_main_labels, 'sub_output': train_encoded_sub_labels},
        validation_data=(test_padded_sequences, {'main_output': test_encoded_main_labels, '
    sub_output': test_encoded_sub_labels}),
        epochs=epochs,
        batch_size=batch_size,
        verbose=1
    )

    # Save plots and confusion matrices
    timestamp = str(int(time.time()))
    save_plots_to_pdf(f'plots_{h_dim}_{timestamp}.pdf', history, h_dim)

    # Only plot confusion matrices for the test data
    main_output_predictions, sub_output_predictions = model.predict(test_padded_sequences)
    main_output_predictions = np.argmax(main_output_predictions, axis=1)
    sub_output_predictions = np.argmax(sub_output_predictions, axis=1)

    save_confusion_matrix_to_pdf(
        test_encoded_main_labels,
        main_output_predictions,
        f'confusion_matrix_test_main_{h_dim}_{timestamp}.pdf',
        'main'
    )
    save_confusion_matrix_to_pdf(
        test_encoded_sub_labels,
        sub_output_predictions,
        f'confusion_matrix_test_sub_{h_dim}_{timestamp}.pdf',
        'sub'
    )

    print_evaluation_metrics(test_encoded_main_labels, main_output_predictions, 'Main')
    print_evaluation_metrics(test_encoded_sub_labels, sub_output_predictions, 'Sub')
```

با اجرای این دستورات در نزدیک‌ترین حالت به مقاله نتایج به صورتی است که در جدول ۲ و شکل ۱۶ و شکل ۱۷ نشان داده شده است. همان‌طور که مشاهده می‌شود نتایج در حالت h_dim=100 به مراتب بهتر است. دلیل بدشدن نتایج برای کلاس و برپسبی مانند ۶ یا ۲۹ فرعی آن است که فقط یک داده از این برچسب در مجموعه‌داده‌ی تست وجود دارد و رخ‌دادن هم‌چین اتفاقی دور از انتظار نیست. عملکرد کلی اما قابل قبول است.

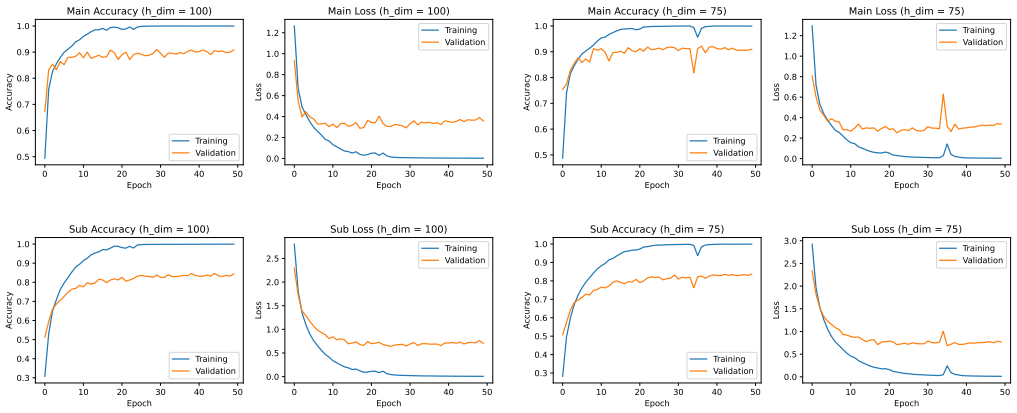
.

جدول ۲: نتایج مربوط به مدل دوم – پیاده‌سازی اول

Test Set (%)								Training Set (%)		h Dimension
F1-score		Recall		Precision		Accuracy		Accuracy		
Sub	Main	Sub	Main	Sub	Main	Sub	Main	Sub	Main	
۶۱.۶۲	۸۶.۵۴	۶۲.۴۸	۸۶.۱۹	۶۶.۴۵	۸۷.۹۱	۷۶.۸۰	۸۸.۴۰	۹۶.۷۴	۹۹.۲۸	25
۶۰.۷۲	۸۹.۹۱	۶۱.۱۷	۸۹.۰۱	۶۶.۱۶	۹۱.۰۰	۸۰.۶۰	۹۱.۴۰	۹۶.۹۴	۹۷.۷۶	50
۶۹.۲۶	۹۰.۳۱	۶۷.۶۵	۸۸.۷۳	۷۸.۰۱	۹۲.۸۷	۸۳.۶۰	۹۱.۰۰	۹۹.۹۳	۹۹.۹۶	75
۶۳.۱۲	۹۰.۱۶	۶۵.۶۵	۸۸.۴۰	۶۵.۹۲	۹۲.۶۷	۸۴.۴۰	۹۰.۸۰	۹۹.۹۳	۹۹.۹۶	100



(ب) h_dim=50 (آ) h_dim=25

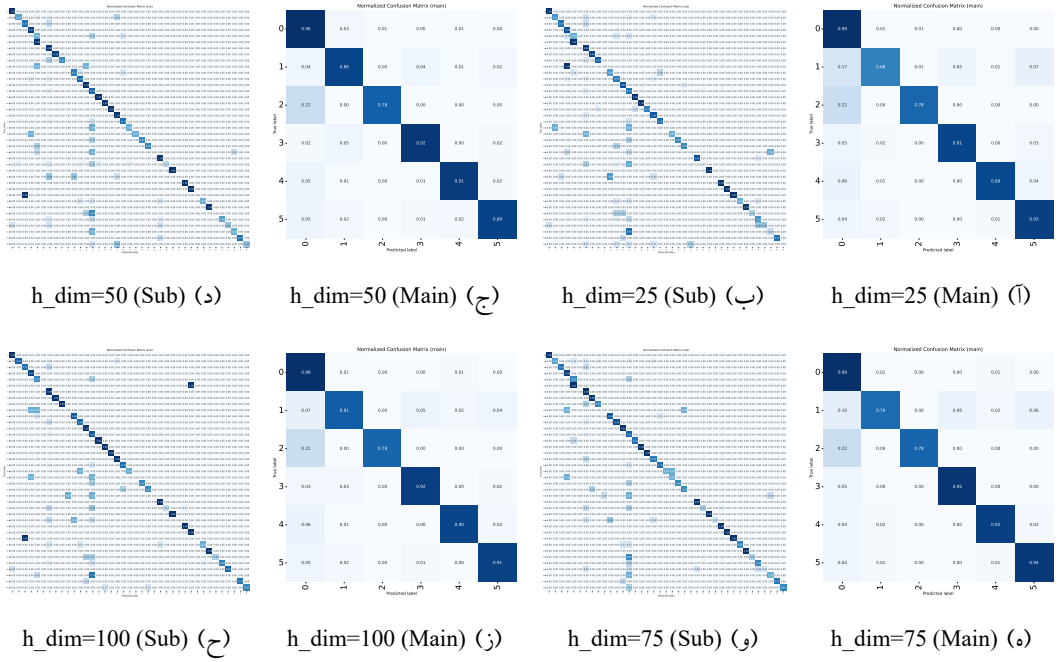


(د) h_dim=100 (ج) h_dim=75

شکل ۱۶: نتیجهٔ تابع دقت و اتلاف در حالت اعتبارسنجی با داده‌های تست (منطبق بر مقاله).

نتایج را در حالتی که در لایه‌های تماماً‌متصل از رگولاریزیشن نرم L2 استفاده کرده‌ایم نیز تست کردیم. نتایج به صورتی است که در جدول ۳ و شکل ۱۸ و شکل ۱۹ نشان داده شده است.

هم‌چنین نتایج را برای یک حالتی که احتمالاً با آنچه مقاله درخصوص دادهای اعتبارسنجی در آن به‌صورت گنگ اشاره کرده کمی فاصله دارد نیز امتحان کردیم (انتخاب بخشی از دادهٔ آموزش به‌عنوان اعتبارسنجی). نتایج به صورتی است که در جدول ۴



(آ) h_dim=25 (Main)　(ب) h_dim=25 (Sub)　(ج) h_dim=50 (Main)　(د) h_dim=50 (Sub)

(ه) h_dim=75 (Main)　(و) h_dim=75 (Sub)　(ز) h_dim=100 (Main)　(ح) h_dim=100 (Sub)

شکل ۱۷: ماتریس درهم‌ریختگی در حالت اعتبارسنجی با داده‌های تست (منطبق بر مقاله).

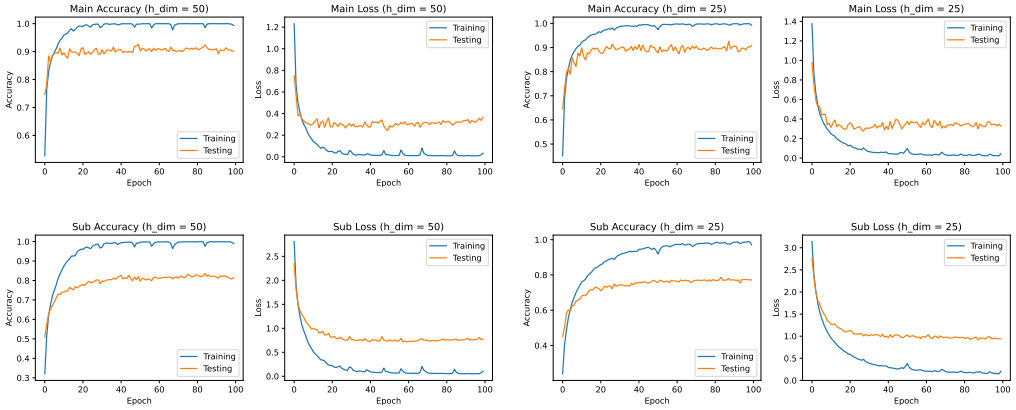
جدول ۳: نتایج مربوط به مدل دوم – پیاده‌سازی دوم

Test Set (%)								Training Set (%)		h Dimension
F1-score		Recall		Precision		Accuracy		Accuracy		
Sub	Main	Sub	Main	Sub	Main	Sub	Main	Sub	Main	
۵۰.۶۶	۸۹.۱۲	۵۲.۴۳	۸۸.۵۶	۵۳.۹۸	۸۹.۸۸	۷۷.۲۰	۹۰.۸۰	۹۷.۱۰	۹۹.۲۷	25
۶۸.۱۲	۸۸.۷۸	۶۷.۶۲	۸۶.۴۳	۷۷.۶۸	۹۳.۲۰	۸۱.۴۰	۹۰.۲۰	۹۸.۹۷	۹۹.۳۲	50
۶۴.۸۷	۸۹.۴	۶۵.۴۶	۸۷.۴۵	۶۸.۲۴	۹۲.۴۳	۸۲.۸۰	۸۹.۸۰	۹۹.۹۳	۹۹.۹۸	75
۶۷.۰۴	۹۰.۰۸	۶۶.۳۸	۸۸.۷۶	۷۱.۹۵	۹۳.۱۰	۸۲.۴۰	۹۱.۲۰	۹۹.۴۹	۹۹.۷۴	100

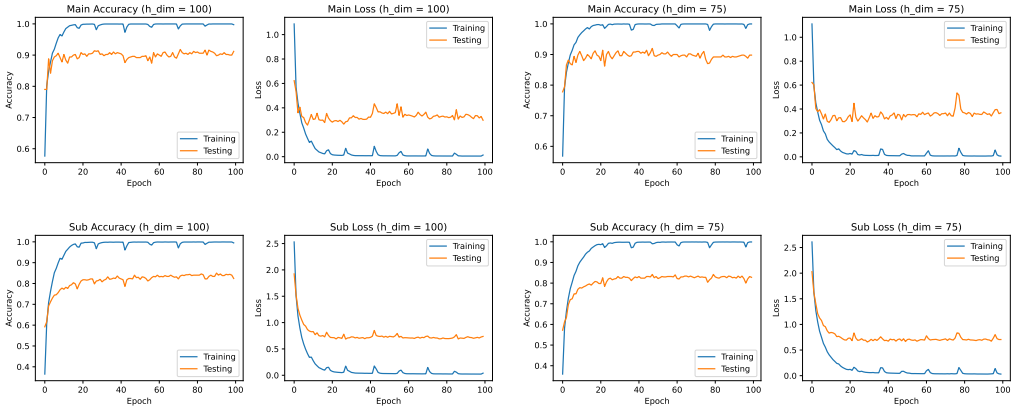
و شکل ۲۰ و شکل ۲۱ نشان داده شده است. در نهایت، حالت قبل را با ترم تنظیم تکرار می‌کنیم. نتایج به صورتی است که در

جدول ۴: نتایج مربوط به مدل دوم – پیاده‌سازی سوم

Test Set (%)								Training Set (%)		h Dimension
F1-score		Recall		Precision		Accuracy		Accuracy		
Sub	Main	Sub	Main	Sub	Main	Sub	Main	Sub	Main	
۴۶.۲۹	۸۷.۸۳	۴۷.۴۹	۸۴.۱۷	۵۱.۲۱	۸۶.۵۹	۷۵.۲۰	۸۷.۶۰	۹۵.۰۵	۹۹.۳۸	25
۶۵.۵۲	۸۸.۲۶	۶۴.۹۳	۸۶.۴۶	۷۰.۳۰	۹۱.۱۳	۸۱.۲۰	۸۸.۸۰	۹۹.۸۲	۹۹.۹۳	50
۶۲.۴۴	۸۸.۷۹	۶۰.۸۲	۸۶.۶۸	۶۸.۵۱	۹۲.۲۱	۷۹.۲۰	۸۹.۰۰	۹۹.۸۹	۹۹.۹۵	75
۶۳.۷۳	۸۸.۸۹	۶۳.۴۸	۸۷.۷۶	۶۹.۱۴	۹۰.۸۹	۸۲.۴۰	۹۰.۲۰	۹۹.۹۳	۹۹.۹۵	100



(آ) h_dim=25 (ب) h_dim=50



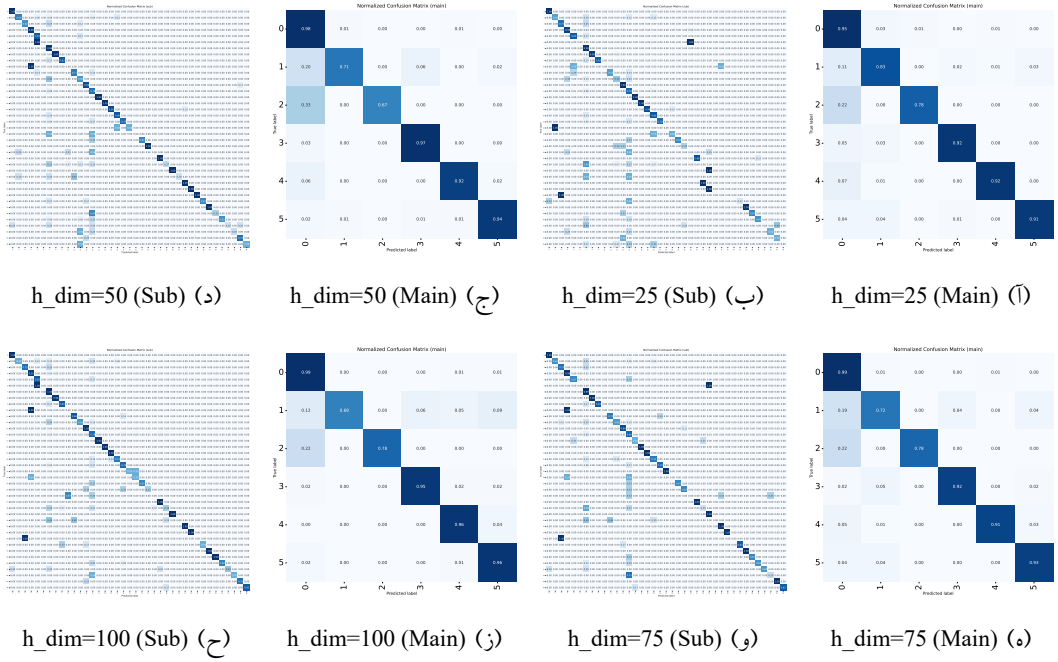
(ج) h_dim=75 (د) h_dim=100

شکل ۱۸: نتیجهٔ تابع دقت و اتلاف در حالت اعتبارسنجی با داده‌های تست (با ترم تنظیم).

جدول ۵ و شکل ۲۲ و شکل ۲۳ نشان داده شده است.

جدول ۵: نتایج مربوط به مدل دوم – پیاده‌سازی چهارم

Test Set (%)								Training Set (%)		h Dimension
F1-score		Recall		Precision		Accuracy		Accuracy		
Sub	Main	Sub	Main	Sub	Main	Sub	Main	Sub	Main	
۴۸.۵۵	۸۸.۴۴	۵۰.۱۵	۸۶.۴۰	۵۴.۴۵	۹۱.۶۹	۷۶.۲۰	۸۸.۸۰	۹۷.۶۶	۹۹.۸۲	**25**
۶۵.۸۰	۹۰.۰۴	۶۵.۹۱	۸۸.۰۴	۷۰.۸۴	۹۲.۹۸	۸۱.۰۰	۹۰.۴۰	۹۹.۸۹	۹۹.۹۸	**50**
۵۹.۹۸	۸۸.۳۷	۶۰.۰۷	۸۶.۸۶	۶۴.۲۲	۹۰.۶۴	۷۸.۰۰	۸۹.۴۰	۹۸.۳۰	۹۹.۱۵	**75**
۶۴.۹۸	۸۷.۴۲	۶۴.۰۸	۸۵.۰۲	۷۱.۴۳	۹۱.۶۸	۸۰.۸۰	۸۹.۰۰	۹۹.۳۸	۹۹.۵۹	**100**

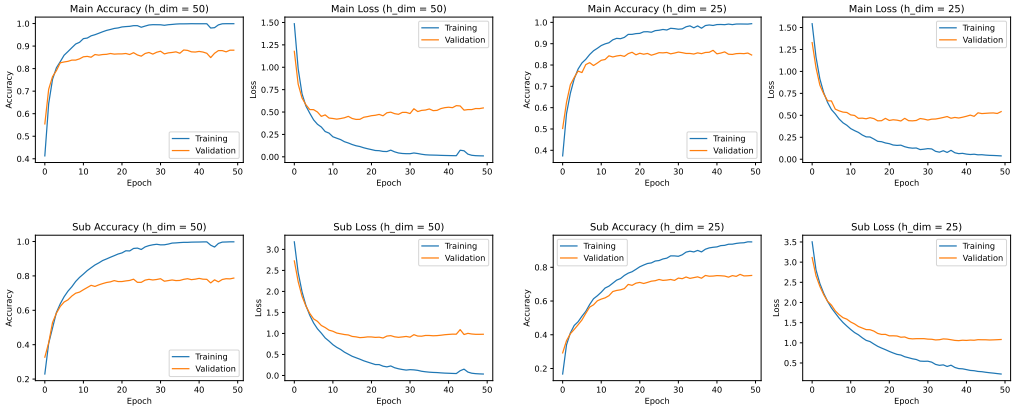


شکل ۱۹: ماتریس درهم‌ریختگی در حالت اعتبارسنجی با داده‌های تست (با ترم تنظیم).

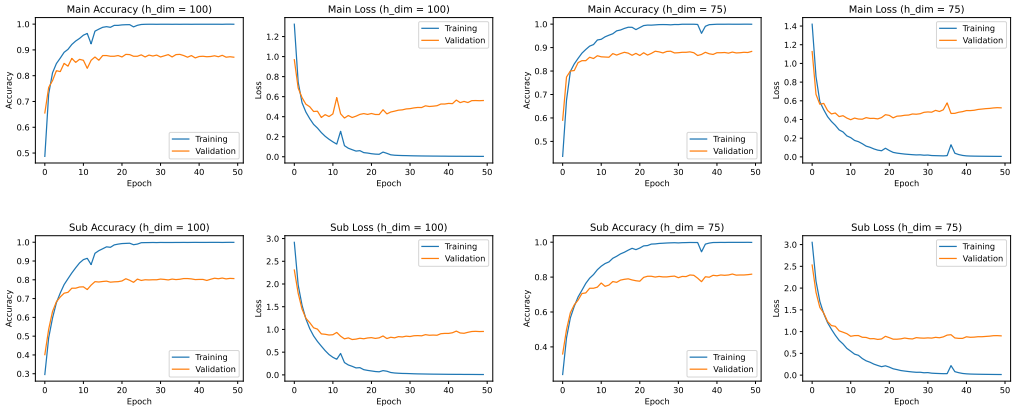
۴.۱ پاسخ قسمت ۴

در مقاله این‌طور ذکر شده که در نهایت و به منظور آزمایش (حداقل به طور خلاصه) اهمیت آنچه در طبقه‌بندی اهداف به منظور خلق یک پاسخ‌دهنده به دست آمده، تصمیم گرفته شد یک نمونهٔ اولیه از یک مدل پاسخ‌دهنده ایجاد شود که بتواند از ساختار به دست آمده استفاده کند و یک پاسخ تولید کند. این نمونهٔ اولیه از یک شبکهٔ LSTM دوطرفه (BLSTM) استفاده می‌کند تا بتواند بررسی ورودی را به صورت مناسبی انجام دهد، به طوری که تنها پس از دریافت کامل سوال، بتواند پاسخی تولید کند. وضعیت این شبکهٔ BLSTM توسط پیش‌بینی بازگشتی توسط مدل قبلی تغذیه می‌شود. دو بردار که نمایندهٔ کلاس اصلی و فرعی هستند در واقع به هم متصل شده‌اند و یک بردار تکی را تشکیل می‌دهند که وضعیت اولیهٔ شبکهٔ BLSTM را مشخص کنند. خروجی‌های شبکه، که کلمات پاسخ را تشکیل می‌دهند، به عبارت دیگر توسط تحلیل هر دو متن قبلی و بعدی تولید می‌شوند و تحت تأثیر دسته‌بندی فراهم‌شده قرار می‌گیرند. نمایی ازاین مدل پاسخ‌دهنده (پاسخ‌گو) در شکل ۲۵ نشان داده شده است.

برای تشکیل ساختار آموزش و ارزیابی این بخش هم مانند قبل ابتدا بخشی را برای پردازش و آماده‌سازی داده‌ها در نظر می‌گیریم. ابتدا فایل با فرمت CSV به نام `QA_data.csv` را بارگیری می‌کنیم و آن را در متغیر `qa_data` ذخیره می‌کنیم. پارامتر `encoding=latin1` برای تعیین کدگذاری متن استفاده می‌شود که در اینجا از latin1 استفاده شده است. در ادامه یک تابع با نام `normalize_text` تعریف می‌کنیم. این تابع وظیفهٔ نرمال‌سازی یک متن را بر عهده دارد و ضمن کوچک‌کردن حروف، واژگان و نشانه‌های اضافی را حذف می‌کند. در ادامه سایرکارهای مربوط به توکنی‌سازی، نهفته‌سازی، پدگذاری و غیره مشابه آن‌چه که در پاسخ قسمت ۳ ذکر شد انجام می‌شود. در ادامه مدل را مطابق آن‌چه از مقاله فهمیده‌ایم تعریف و پیاده‌سازی می‌کنیم. مشابه قسمت قبل چند مدل مختلف ایجاد کرده‌ایم که نزدیک‌ترین مدل به مقاله را این‌جا معرفی می‌کنیم. ابتدا یک لایهٔ ورودی با شکل مشخص که به عنوان ورودی دنباله‌های توکن‌شدهٔ با طول بیشینهٔ `max_sequence_length` را دریافت می‌کند، ایجاد می‌کنیم. سپس یک لایهٔ embedding ایجاد می‌کنیم که مسئول نمایش عددی لغات است. با ارائهٔ پارامترهای `vocab_size` و `embedding_dim`، لایهٔ embedding با سایز و ابعاد مشخص ایجاد می‌شود. همچنین با ارائهٔ اوزان `embedding_matrix`



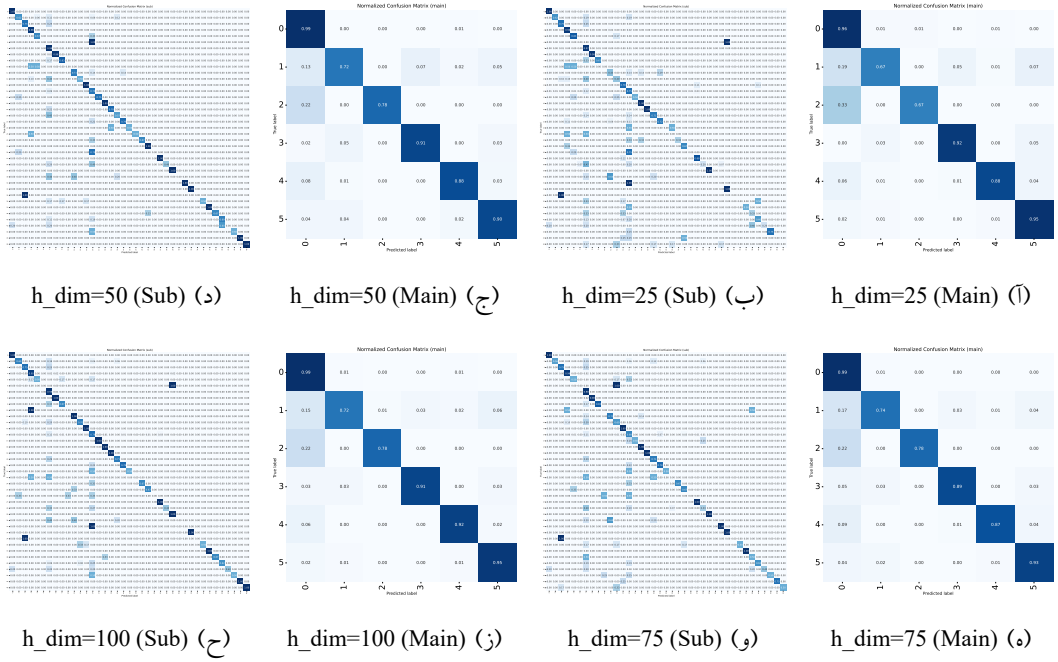
(آ) h_dim=25 (ب) h_dim=50



(ج) h_dim=75 (د) h_dim=100

شکل ۲۰: نتیجهٔ تابع دقت و اتلاف در حالت داده‌های اعتبارسنجی جداگانه.

به‌عنوان ورودی، لایهٔ embedding از این وزن‌ها استفاده می‌کند. پارامتر `trainable=False` نیز به معنی این است که وزن‌های این لایه قابل آموزش نباشند. در ادامه یک لایهٔ LSTM با ۱۰۰ واحد حافظه (memory units) ایجاد می‌کنیم که دنباله‌های ورودی را دریافت می‌کند و دنباله خروجی را ضمن داشتن پارامتر `return_sequences=True` تولید می‌کند. آخرین وضعیت مخفی (hidden state) و آخرین وضعیت سلول (cell state) لایهٔ LSTM را استخراج می‌کنیم. این وضعیت‌ها معمولاً در زمان پایان دنباله معنادار هستند. سپس یک لایهٔ BiLSTM (LSTM دوطرفه) ایجاد می‌کنیم. این لایه دو لایهٔ LSTM را به صورت موازی در دو جهت (جلو و عقب) اجرا می‌کند و خروجی را تولید می‌کند. پارامتر `merge_mode=concat` نیز نشان‌دهندهٔ این است که خروجی‌های دو لایهٔ LSTM دوطرفه به صورت ترکیب‌شده (concatenated) باشند. این لایه از خروجی لایهٔ embedding و بازگشتی می‌تواند استفاده کند و حالات مختلف در پیاده‌سازی تست شده‌اند. در بخش وضعیت اولیه (`initial_state`) هم، آرایه‌ای شامل آخرین وضعیت مخفی و آخرین وضعیت سلول لایهٔ LSTM آمده است (منطبق بر مقاله). این وضعیت‌ها از لایهٔ LSTM قبلی استخراج شده و به عنوان وضعیت اولیه برای لایهٔ BiLSTM مورد استفاده قرار می‌گیرند. درنهایت، یک لایهٔ خروجی ایجاد می‌کنیم که دارای تابع فعال‌سازی softmax است که برای تبدیل خروجی‌های عددی به احتمالات برای هر کلاس استفاده می‌شود. تعداد کلاس‌ها با استفاده از پارامتر `num_classes` مشخص شده است. در انتها مدل را آماده‌به‌کار می‌کنیم. در ادامه دستوراتی را برای آموزش مدل می‌نویسیم. با فراخوانی تابع `fit` روی مدل، داده‌های آموزش



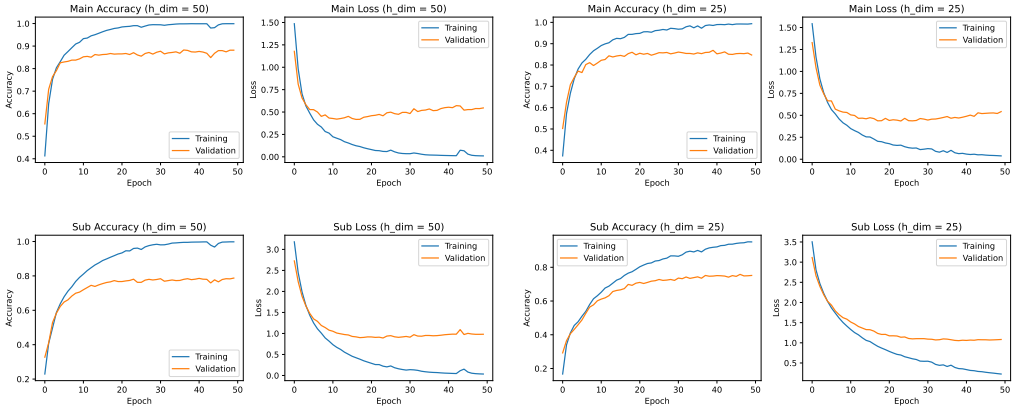
شکل ۲۱: ماتریس درهم‌ریختگی در حالت داده‌های اعتبارسنجی جداگانه.

X و برچسب‌های آموزش answers_encoded به مدل داده می‌شوند. فراپارامترها با سعی و خطای فراوان تنظیم شده‌اند. در ادامه لیستی از سوالات مدنظر تمرین ایجاد می‌کنیم و این سوالات آزمایشی را به‌صورت نرمال‌شده (توسط تابع normalize_text) در آرایهٔ test_questions ذخیره می‌کنیم. این عمل باعث حذف نویزهای غیرضروری و استانداردسازی متن سوالات می‌شود. در انتها سوالات آزمایشی به توکن‌ها تبدیل می‌شوند.

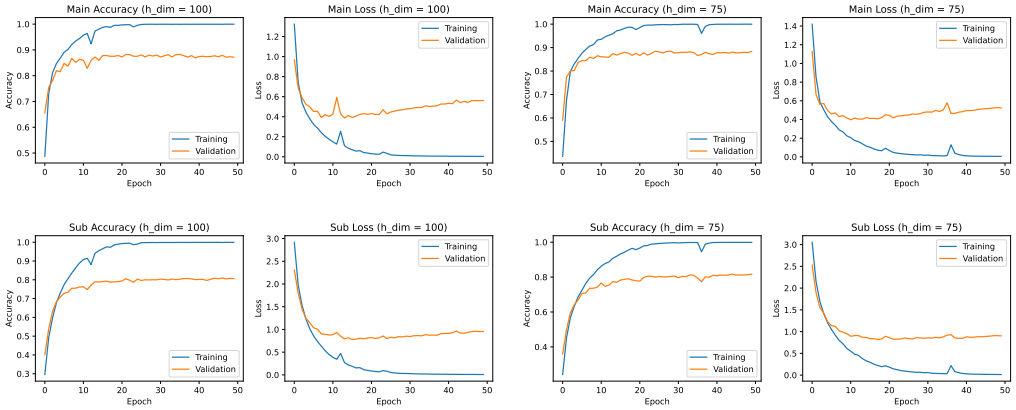
درادامه ابتدا تابع texts_to_sequences از توکن‌بندی‌کنندهٔ tokenizer روی سوالات آزمایشی اعمال می‌شود و سپس با استفاده از تابع pad_sequences، دنباله‌های توکن‌ها با طول یکسان (max_sequence_length) تنظیم می‌شوند. نتیجهٔ نهایی در آرایهٔ test_X قرار می‌گیرد. با انجام این کار و با استفاده از مدل آموزش‌دیده، پیش‌بینی‌های مدل روی داده‌های آزمایشی صورت می‌گیرد. تابع predict روی test_X اجرا شده و خروجی پیش‌بینی‌ها در متغیر predictions قرار می‌گیرد. سپس با استفاده از np.argmax، برچسب‌هایی که احتمال بالاتری دارند انتخاب می‌شوند و در آرایهٔ predicted_labels ذخیره می‌شوند. در نهایت، با استفاده از حلقهٔ for، سوالات آزمایشی و پاسخ‌های پیش‌بینی‌شده به‌صورت همزمان چاپ می‌شوند. دستورات در برنامهٔ ۳ آورده شده است.

Program 3: Prototype Responder, Training, and Evaluation

```
import pandas as pd
import numpy as np
import re
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, LSTM, Bidirectional, Dense

```



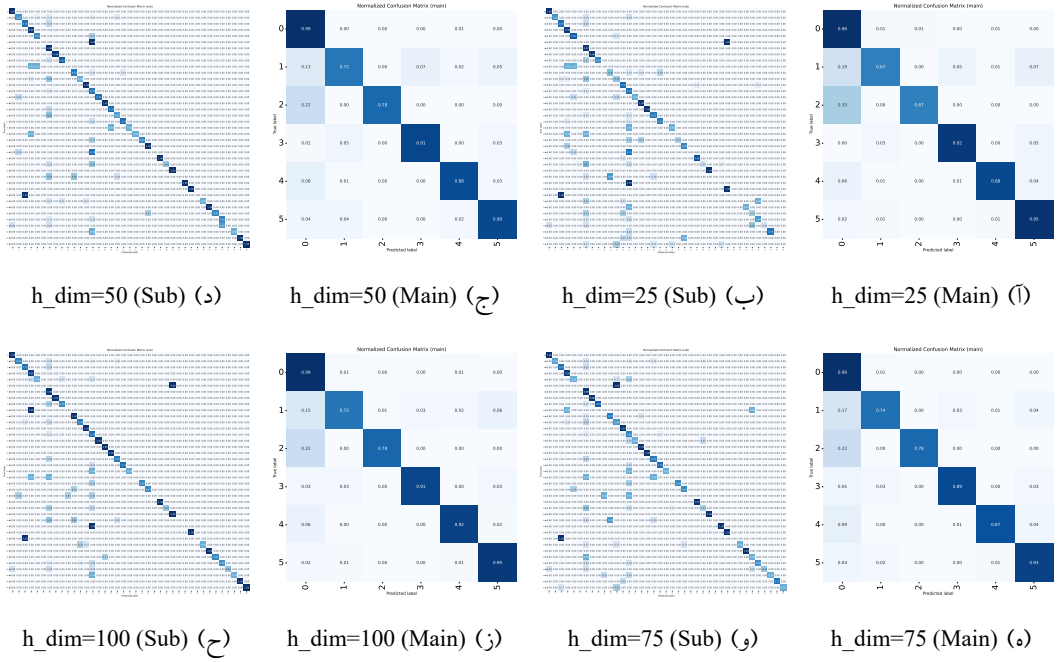
(آ) h_dim=25 (ب) h_dim=50



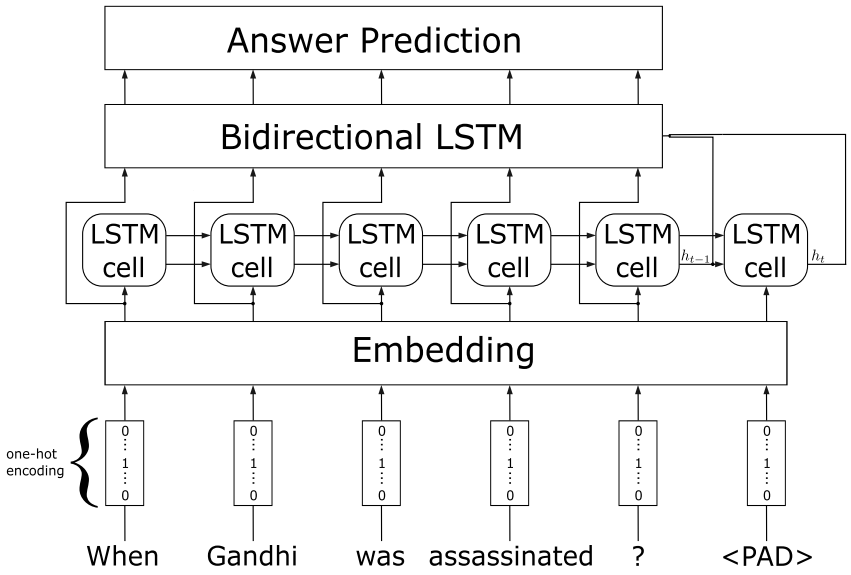
(ج) h_dim=75 (د) h_dim=100

شکل ۲۲: نتیجهٔ تابع دقت و اتلاف در حالت داده‌های اعتبارسنجی جداگانه.

```
# Load the QA_data
qa_data = pd.read_csv('QA_data.csv', encoding='latin1')

# Define function for text normalization
def normalize_text(text):
    text = re.sub(r'[^a-zA-Z0-9\?]+', ' ', text)
    text = text.lower()
    return text

# Normalize text
qa_data['text'] = qa_data['text'].apply(normalize_text)

# Preprocess the data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(qa_data['text'])
sequences = tokenizer.texts_to_sequences(qa_data['text'])
```



شکل ۲۳: ماتریس درهم‌ریختگی در حالت داده‌های اعتبارسنجی جداگانه.



شکل ۲۴: نمایی از مدل پاسخگو.

```
word_index = tokenizer.word_index
max_sequence_length = max(len(seq) for seq in sequences)

X = pad_sequences(sequences, maxlen=max_sequence_length)

```

```
# One-hot encode the answers
answers = qa_data['answer']
answer_labels = np.unique(answers)
label_to_index = {label: index for index, label in enumerate(answer_labels)}
answers_encoded = np.array([label_to_index[answer] for answer in answers])

num_classes = len(answer_labels)

# Load GloVe embeddings into a dictionary
embeddings_index = {}
with open('glove.6B.300d.txt') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

# Create an embedding matrix
embedding_dim = 300
vocab_size = len(word_index) + 1
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

# Define the Responder model
input_layer = Input(shape=(max_sequence_length,))
embedding_layer = Embedding(vocab_size, embedding_dim, weights=[embedding_matrix], trainable=
    False)(input_layer)
lstm_layer = LSTM(100, return_sequences=True)(embedding_layer)
lstm_last_hidden_state = lstm_layer[:, -1, :]
lstm_last_cell_state = lstm_layer[:, -1, :]
bilstm_layer = Bidirectional(LSTM(100, return_sequences=False), merge_mode='concat')(lstm_layer,
    initial_state=[lstm_last_hidden_state, lstm_last_cell_state, lstm_last_hidden_state,
    lstm_last_cell_state])
# bilstm_layer = Bidirectional(LSTM(100, return_sequences=False), merge_mode='concat')(
    embedding_layer, initial_state=[lstm_last_hidden_state, lstm_last_cell_state,
    lstm_last_hidden_state, lstm_last_cell_state])
output_layer = Dense(num_classes, activation='softmax')(bilstm_layer)
model = Model(inputs=input_layer, outputs=output_layer)

# model = Sequential()
# model.add(Embedding(vocab_size, embedding_dim, input_length=max_sequence_length, weights=[
    embedding_matrix], trainable=False))
```

```
# model.add(LSTM(100, return_sequences=True))
# model.add(Bidirectional(LSTM(100, return_sequences=True)))
# model.add(Bidirectional(LSTM(100)))
# model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X, answers_encoded, epochs=100, batch_size=8)

# Plot loss and accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'])
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.tight_layout()
plt.savefig('lossaccuracyplot2.pdf')
plt.show()

# Test the model on Table 3 questions
test_questions = [
    'How many people speak French?',
    'What day is today?',
    'Who will win the war?',
    'Who is Italian first minister?',
    'When World War II ended?',
    'When Gandhi was assassinated?'
]

# Normalize test questions
test_questions = [normalize_text(question) for question in test_questions]

test_sequences = tokenizer.texts_to_sequences(test_questions)
test_X = pad_sequences(test_sequences, maxlen=max_sequence_length)

predictions = model.predict(test_X)
predicted_labels = [answer_labels[np.argmax(pred)] for pred in predictions]

```

```
# Save predictions as pandas DataFrame
results = pd.DataFrame({'Question': test_questions, 'Predicted Answer': predicted_labels})
results.to_csv('predictions.csv', index=False)

# Print the predicted answers
for question, answer in zip(test_questions, predicted_labels):
    print(f'Question: {question}')
    print(f'Predicted Answer: {answer}\n')
```

برخی نتایج به صورتی است که در جدول ۶، شکل ۲۵ و برنامهٔ ۴ نشان داده شده است:

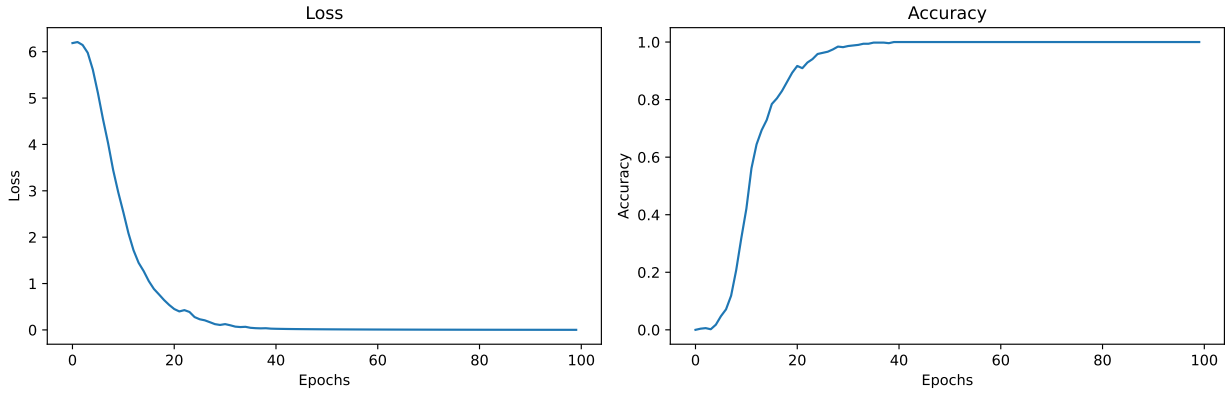
Program 4: Prototype Responder: Some Results

```
Question: how many people speak french?
Predicted Answer: 2000

Question: what day is today?
Predicted Answer: tennis tournament

Question: who will win the war?
Predicted Answer: 16th US president

Question: who is italian first minister?
Predicted Answer: Immanuel Kant

Question: when world war ii ended?
Predicted Answer: 1945

Question: when gandhi was assassinated?
Predicted Answer: 1830
----------------------------------------
Question: how many people speak french?
Predicted Answer: 5,280

Question: what day is today?
Predicted Answer: deliberate killing of a people

Question: who will win the war?
Predicted Answer: Paris

Question: who is italian first minister?
Predicted Answer: 16th US president

Question: when world war ii ended?
Predicted Answer: June 21st

Question: when gandhi was assassinated?
```

```
Predicted Answer: 1743
```

جدول ۶: برخی نتایج مربوط به مدل Prototype Responder

Question	Answer
How many people speak French?	2000 | 5280 | ...
What day is today?	tennis tournament
Who will win the war?	16th US president
Who is Italian first minister?	Immanuel Kant
When World War II ended?	1945 | June 21st | ...
When Gandhi was assassinated?	1830 | 1743 | ...



شکل ۲۵: نمایی از مدل Prototype Responder.