



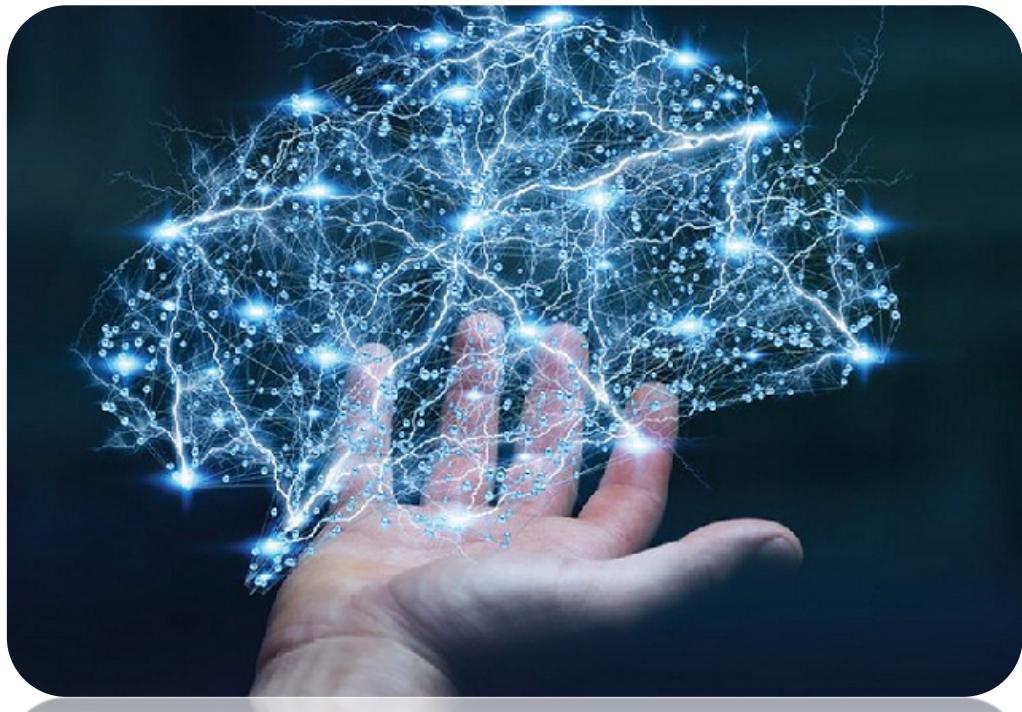
به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر
شبکه عصبی

گزارش پروژه ۳

ملیکه احلاقی	سید علی طباطبایی آل طه	نام و نام خانوادگی
۸۱۰۱۹۴۲۵۴	۸۱۰۱۹۴۴۶۲	شماره دانشجویی
۹۷/۲/۲۷		تاریخ ارسال گزارش

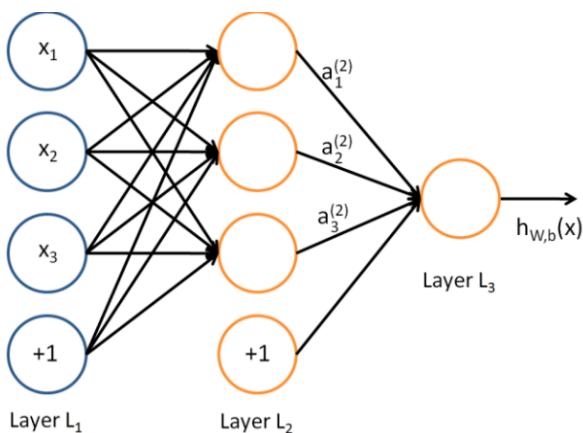


1-1 چکیده

یک شبکه عصبی مصنوعی ایده‌ای برای پردازش اطلاعات است که از سیستم عصبی زیستی الهام گرفته و مانند مغز به پردازش اطلاعات می‌پردازد. این سیستم از شمار زیادی عناصر پردازش به هم پیوسته به نام نورون تشکیل شده که برای حل یک مساله باهم هماهنگ عمل می‌کنند. شبکه‌های عصبی مصنوعی با پردازش داده‌های تجربی، دانش یا قانون نهفته در ورای داده‌ها آنها را به ساختار شبکه منتقل می‌کنند که به این عمل یادگیری می‌گویند. اصولاً توانایی یادگیری مهم ترین ویژگی یک سیستم هوشمند است. سیستمی که قابلیت یادگیری داشته باشد، منعطف‌تر است و ساده‌تر برنامه‌ریزی می‌شود، بنابراین بهتر می‌تواند در مورد مسایل و معادلات جدید پاسخ‌گو باشد. در این ساختار با ایجاد شبکه‌ای بین گره‌ها و اعمال یک الگوریتم آموزشی به آن، شبکه را آموزش می‌دهند. در این حافظه یا شبکه عصبی گره‌ها (نورونها) دارای دو حالت فعال (روشن یا 1) و غیرفعال (خاموش یا 0) اند و هر یال (سیناپس یا ارتباط بین گره‌ها) دارای یک وزن می‌باشد. یالهای با وزن مثبت، موجب تحریک یا فعال کردن گره غیر فعال بعدی می‌شوند و یالهای با وزن منفی، گره متصل بعدی را غیرفعال یا مهار (در صورتی که فعال بوده باشد) می‌کنند. در واقع تنظیم وزنهای ورودی هر نورون عصبی باعث یادگیری کل شبکه می‌شود. پس از اینکه مجموعه‌ای از داده‌ها به منظور آموزش به شبکه داده می‌شود با هر بار دریافت خروجی و مقایسه با جواب مورد انتظار اگر به تشخیص صحیح رسیده بود مسیرهایی که به این تشخیص منجر شده است از طریق شبکه تقویت می‌شود (از طریق نرمالیزه کردن وزن یالها). شبکه‌های عصبی مصنوعی می‌توانند دارای لایه‌های متعددی باشند و یا یک لایه باشند. از کاربردهای شبکه عصبی عبارت اند از حل مسایل شناسایی الگو و پردازش تصویر و متن، پردازش زبان‌های طبیعی، مسایل دسته‌بندی و ... است. در این پژوهه پس از پیاده‌سازی شبکه‌ی عصبی حروف A تا Z را دسته‌بندی می‌نماییم. هدف از پیاده‌سازی این پژوهه آموزش شبکه‌ی عصبی در جهت تشخیص حروف براساس تصویری از یک حرف است.



شکل ۱: نمونه‌ای از ارقام موجود در پایگاه داده notMNIST



شکل ۲: معماری سه لایه شامل یک لایه‌ی ورودی یک لایه‌ی خروجی و یک لایه مخفی.

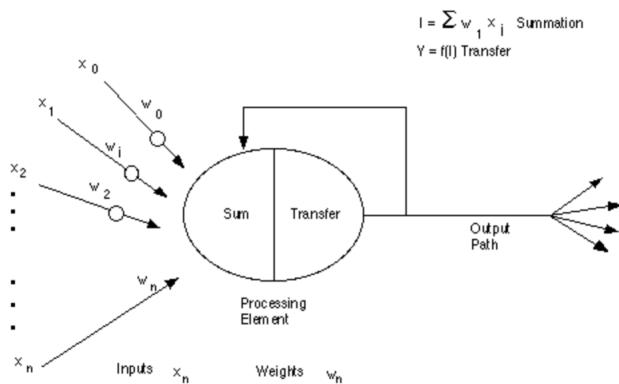
1-2 ارائه‌ی روش

پیاده‌سازی این شبکه‌ی عصبی از چند فاز تشکیل می‌شود. ورودی‌های این شبکه شامل `training` و `test` از مجموعه داده‌های `notMNIST` استفاده می‌کند که با تعدادی از تصاویر مانند شکل ۱ شبکه را آموزش می‌دهیم و با کمک تصاویری مشابه آن را تست می‌نماییم. این ورودی‌ها تصاویر ۲۸*۲۸ پیکسل هستند که در نتیجه لایه‌ی ورودی از ۷۸۴ نورون متناظر این پیکسل‌ها تشکیل می‌شود. در ابتدا توسط کاربر انتخاب‌هایی صورت می‌گیرد که اولاً روش بهینه‌سازی `GD` و یا `Adam` باشد. هم‌چنان کاربر می‌تواند تابع فعال ساز را از بین تابع خطی و `sigmoid` انتخاب نماید. نهایتاً روش `regularization` نیز از بین `L2Norm` و `dropout` توسط کاربر مشخص خواهد شد. بسته به انتخاب‌هایی که کاربر دارد آموزش و تست شبکه عصبی شروع می‌شود. با دریافت `option`‌های کاربر و تشکیل `test_set` و `training_set` شبکه تشکیل خواهد شد. این شبکه از سه لایه‌ی ورودی و خروجی همچنین یک لایه‌ی درونی تشکیل شده است.

لایه‌ی ورودی با دریافت `initialize test_set` و `training_set` می‌شود. در اینجا چک می‌شود که اگر روش بهینه‌سازی `stochastic` باشد ابتدا `shuffle` training_set را می‌کند. همچنین لایه‌ی میانی و خروجی نیز به ترتیب از ۳۰ (عدد تجربی) و ۱۰ نورون تشکیل می‌شود. که این ۱۰ نورون متناظر با هریک از حروف خواهد بود. در ۱۰۰۰ مرحله `train` و `test` تکرار خواهد شد.

Train

در این مرحله به تعداد عناصر `training set` آموزش شبکه صورت می‌گیرد. در هر مرحله تصویر مربوطه به همراه نام حرف دریافت می‌شود. در صورتی که انتخاب کاربر روش `dropout` باشد برای اعداد ورودی یک احتمال تصادفی بین ۰ و ۱ انتخاب می‌شود که بودن آن به منزله‌ی حذف نورون متناظر است. سپس محاسبات لایه‌ی میانی انجام می‌شود که بسته به اینکه `sigmoid` و یا `linear` باشد خروجی نورون‌های میانی محاسبه می‌شود.



همین مراحل برای لایه‌ی خروجی انجام می‌شود. سپس loss_function را محاسبه می‌نماییم که از فرمول زیر محاسبه می‌شود:

$$\sum_1^n \frac{1}{2} (\bar{z}_i - z_i)^2$$

در صورتی که روش regularization L2Norm باشد این مقدار به صورت زیر محاسبه می‌شود:

$$\sum_{i=1}^n (y_i - \sum_{j=1}^p x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

Cost function

سپس backpropagation روی لایه‌ی خروجی و بعد لایه‌ی میانی انجام می‌شود. برای انجام این مرحله درصورتی که تابع SGD باشد در لایه‌ی ورودی به میانی از فرمول زیر به منظور محاسبه می‌شود:

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}} \right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

برای محاسبه میانی delta بین لایه‌ی میانی و خروجی نیز از فرمول زیر بهره برده ایم:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

تفاوت روش SGD و GD در شکل های زیر قابل ملاحظه است:

② Vanilla (Batch) G.D.

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

$\frac{1}{m} \sum_{i=1}^m (\hat{y}^i - y^i) x^i_j$

② Stochastic G.D.

for i in range(M):

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} \text{only one example}$$

$(\hat{y}^i - y^i) x^i_j$

با بهره گیری از این فرمول ها و همچنین اعمال L2Norm در صورت انتخاب شدن آن توسط کاربر مراحل backpropagation به پایان خواهد رسید.

L2 Regularization

$$E = \frac{1}{2} * \sum (t_k - o_k)^2 + \frac{\lambda}{2} * \sum w_i^2$$

plain error

weight penalty

elegant math

simple math

$$\frac{\partial E}{\partial w_{jk}}$$

$$\Delta w_{jk} = \eta * [x_j * (o_k - t_k) * o_k * (1 - o_k)] + [\lambda * w_{jk}]$$

learning rate

signal

Test

بعد از پایان مرحله **feed forward** در طی یک **test** خواهیم شد. در این مرحله در طی یک **iteration** برابر با خروجی مطلوب بود به تعداد خروجی های درست یکی می افزاییم. در نهایت برای به دست آوردن دقت شبکه از درصد **predict** های درست نسبت به تعداد کل تست ها استفاده می کنیم.

در نهایت بعد از اتمام `train` و `test` با محاسبه‌ی `loss function` به طور میانگین می‌توان نمودار `loss` را با کمک `LossAccPlotter()` به نمایش گذاشت.

```
def train(self):
    loss = 0
    for i in range(self.__inLayer.training_set_size()):

        desired_output = self.get_desired_output(self.__inLayer.get_training_label(i))
        self.__outLayer.set_desired_output(desired_output)

        inp = self.__inLayer.get_image(i)
        if self.option.is_dropout():
            prob = np.random.randint(0, 2, (1, 784))
            inp = np.multiply(inp, prob)
        self.__hidLayer.calc(inp)

        hid = self.__hidLayer.get_output()
        if self.option.is_dropout():
            prob = np.random.randint(0, 2, (1, 30))
            hid = np.multiply(hid, prob)
        self.__outLayer.calc(hid)

        loss += self.__outLayer.loss_function()

        self.__outLayer.back_propagate(hid)
        self.__hidLayer.back_propagate(inp, self.__outLayer)

    return loss / self.__inLayer.training_set_size()
```

```
def test(self):
    count = 0
    loss = 0
    for i in range(self.__inLayer.test_set_size()):

        desired_output = self.get_desired_output(self.__inLayer.get_test_label(i))
        self.__outLayer.set_desired_output(desired_output)

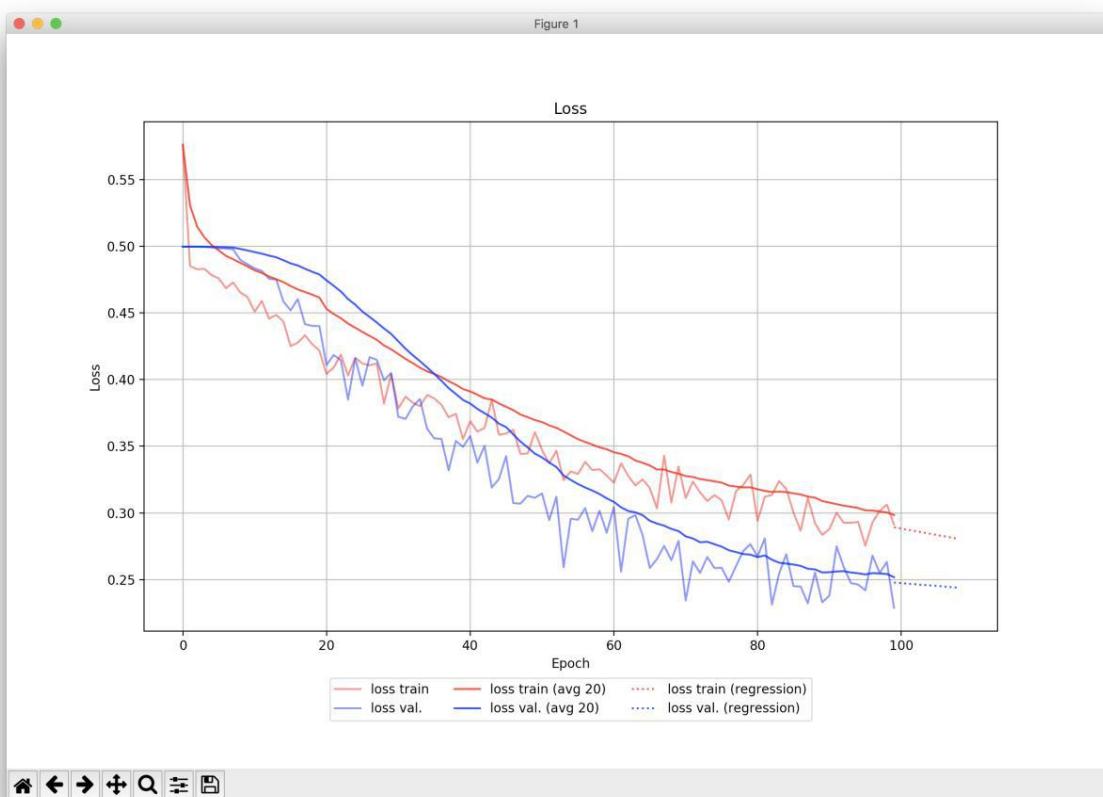
        inp = self.__inLayer.get_test_image(i)
        self.__hidLayer.calc(inp)
        self.__outLayer.calc(self.__hidLayer.get_output())

        loss += self.__outLayer.loss_function()

        must_be = self.__inLayer.get_test_label(i)
        y_predict = np.zeros(10, dtype=np.int)
        max_i = np.argmax(self.__outLayer.get_output())
        y_predict[max_i] = 1
        prediction = self.prediction(tuple(y_predict))
        if must_be == prediction:
            count += 1

    print("avg: %f" % (count / self.__inLayer.test_set_size()))
    return loss / self.__inLayer.test_set_size()
```

1-3 ارائه نتایج



2

Options:

Size of training set: 250

Size of test set: 1000

Number of iterates: 100

Optimization: Stochastic Gradient Descent

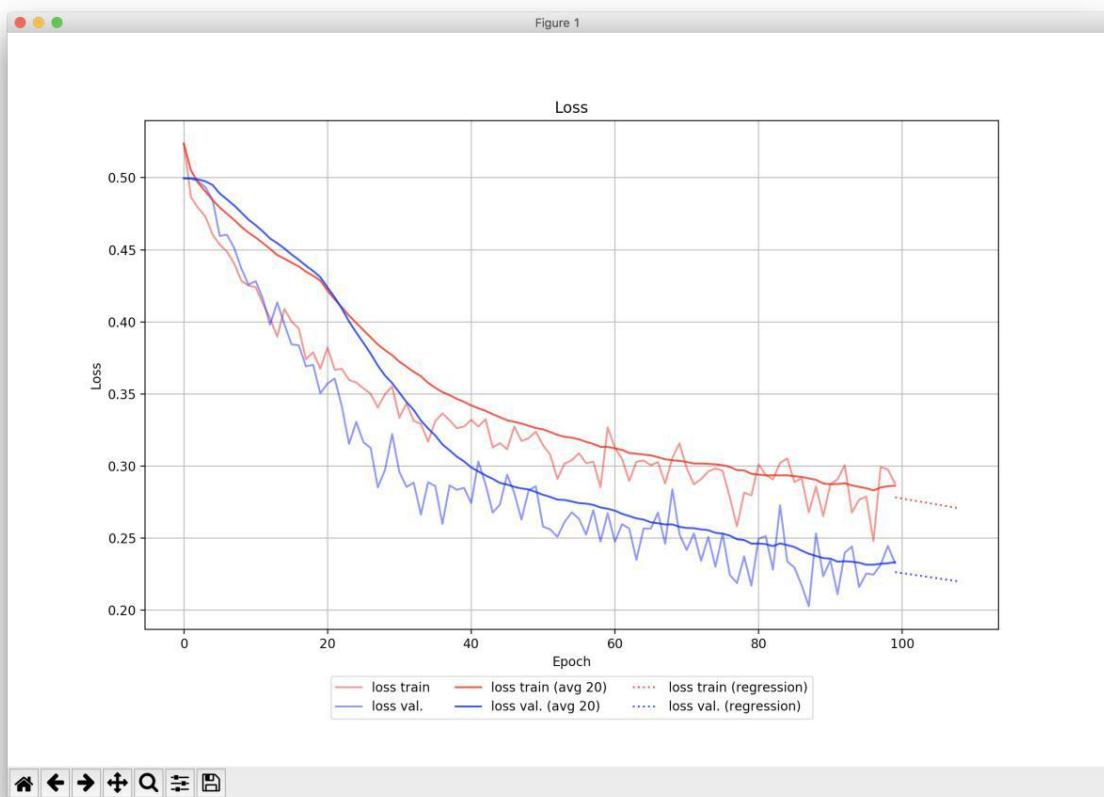
Activation Function: Sigmoid

Regularization: Drop out

accuracy: 81.60%

3

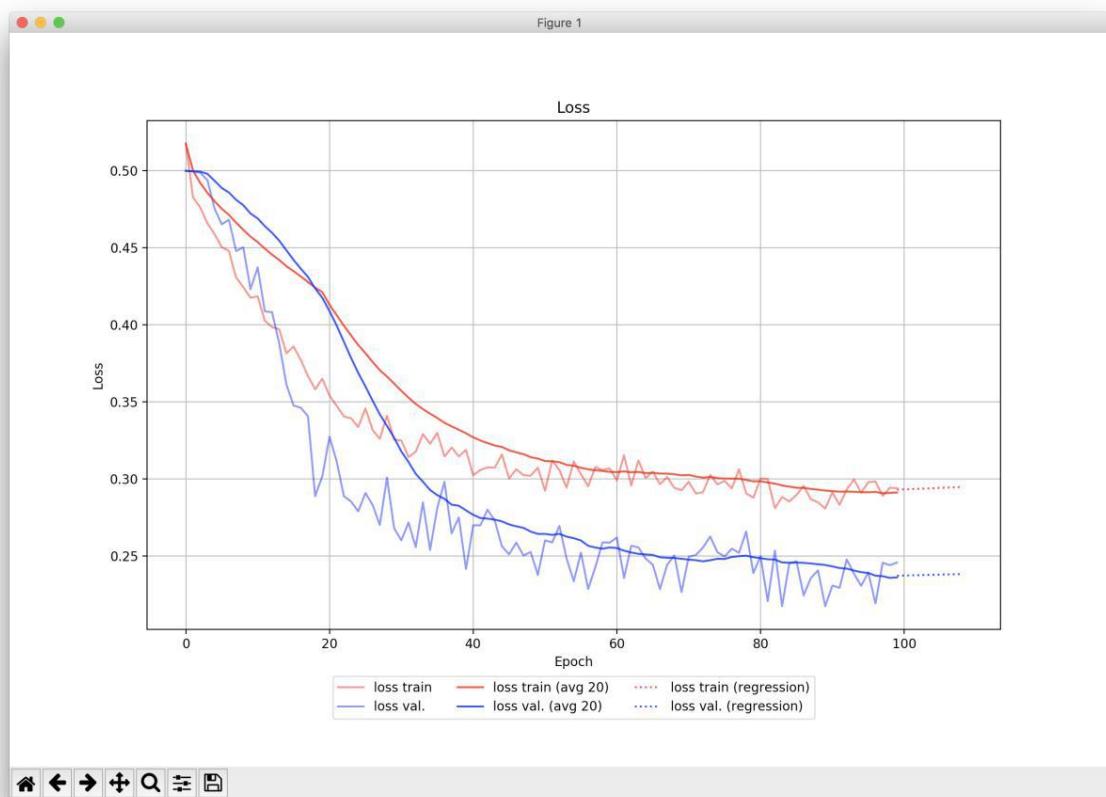
4



5

Options:**Size of training set:** 500**Size of test set:** 1000**Number of iterates:** 100**Optimization:** Stochastic Gradient Descent**Activation Function:** Sigmoid**Regularization:** Drop out**accuracy:** 81.90%

6



7

Options:

Size of training set: 750

Size of test set: 1000

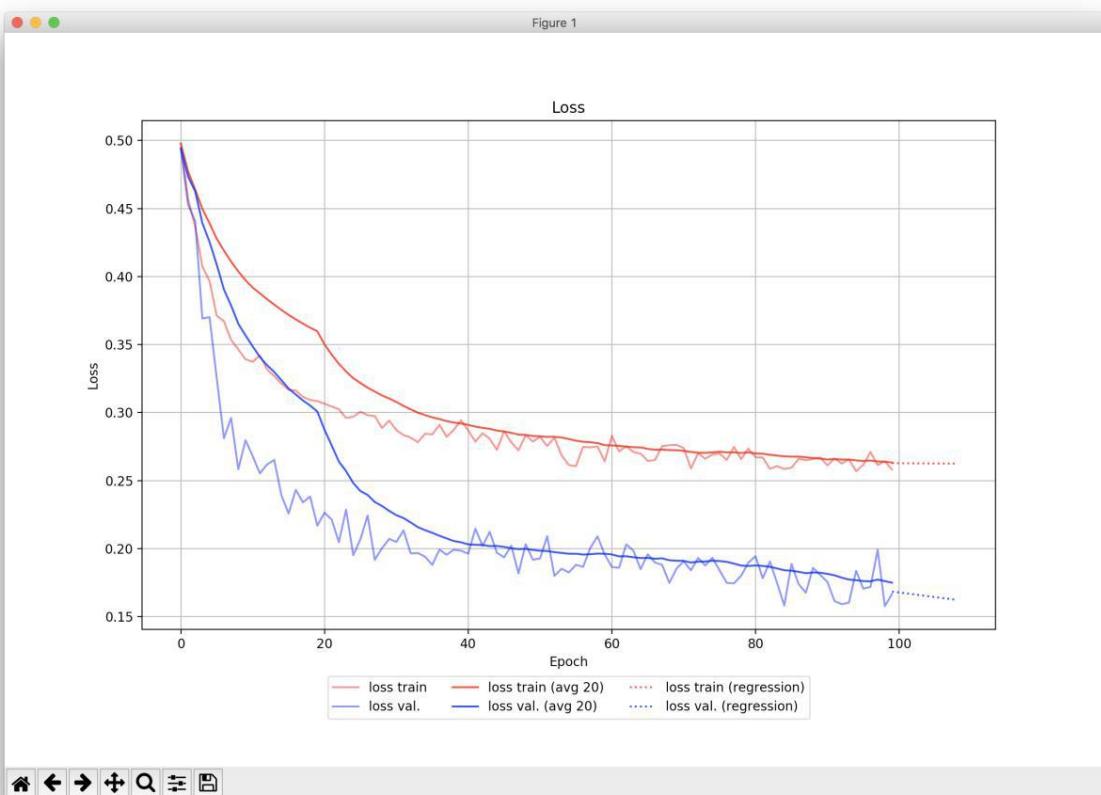
Number of iterates: 100

Optimization: Stochastic Gradient Descent

Activation Function: Sigmoid

Regularization: Drop out

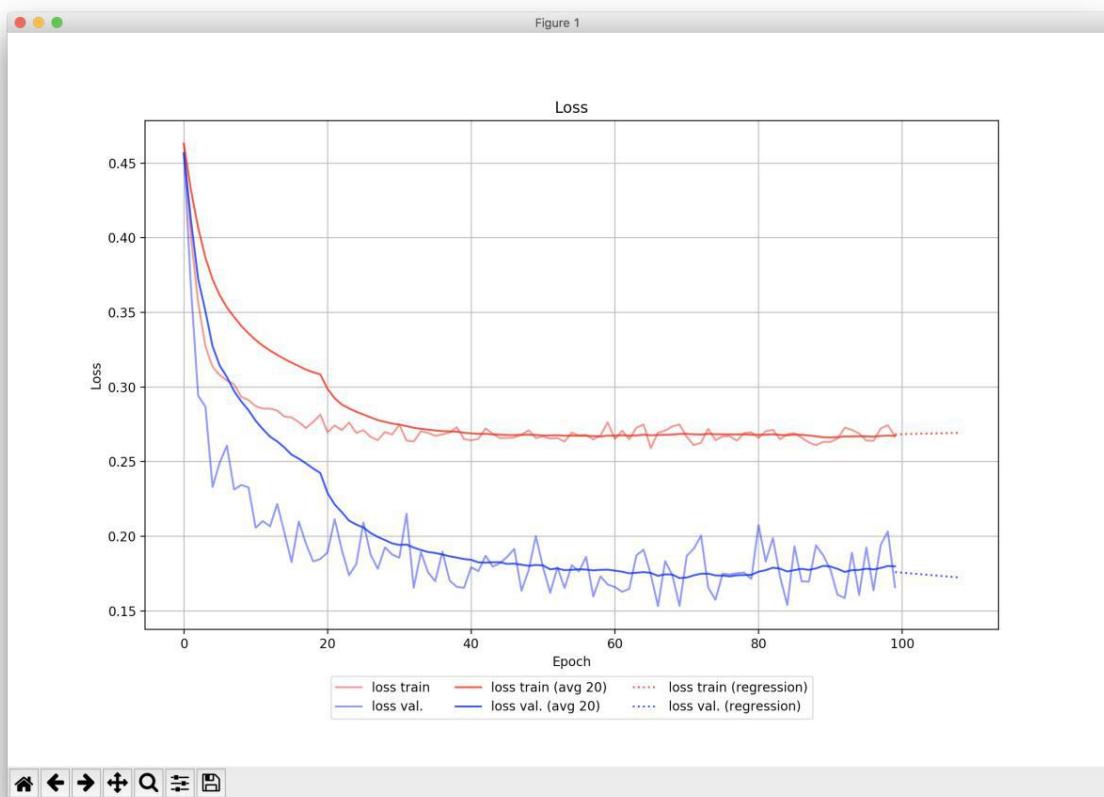
8 accuracy: 80.80%



9

Options:**Size of training set: 1500****Size of test set: 1000****Number of iterates: 100****Optimization: Stochastic Gradient Descent****Activation Function: Sigmoid****Regularization: Drop out****accuracy: 86.80%**

10



11

Options:

Size of training set: 5000

Size of test set: 1000

Number of iterates: 100

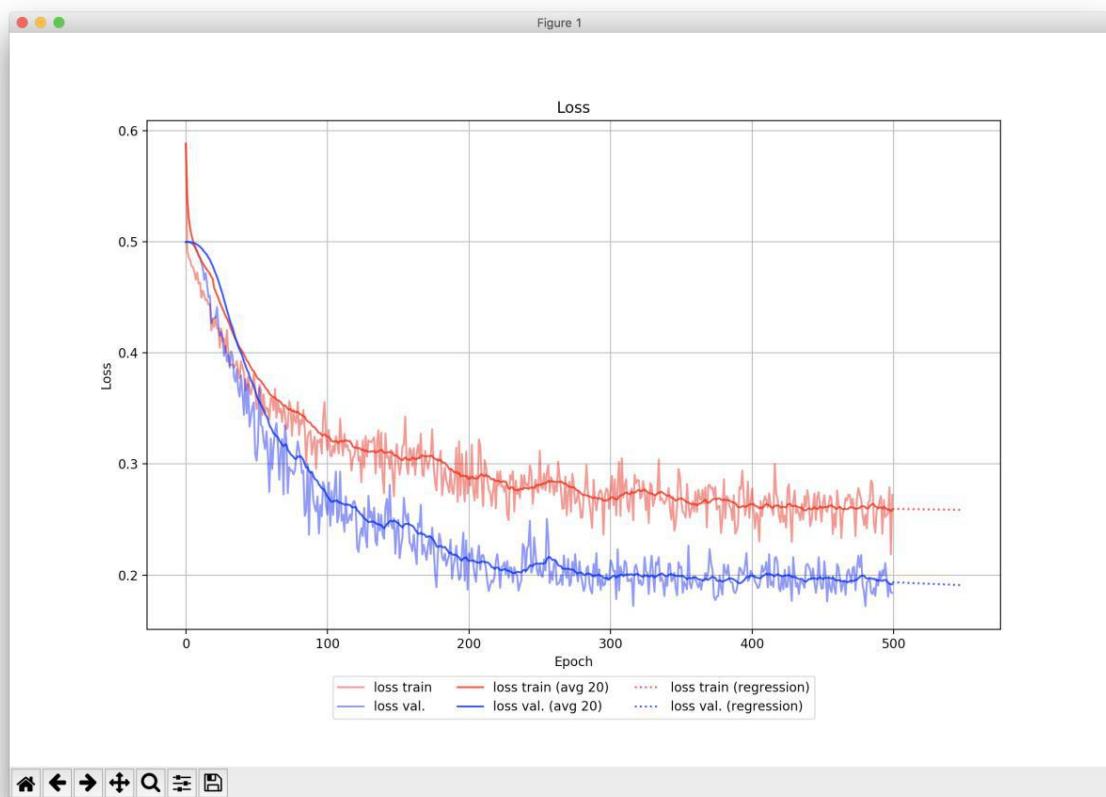
Optimization: Stochastic Gradient Descent

Activation Function: Sigmoid

Regularization: Drop out

accuracy: 83.20%

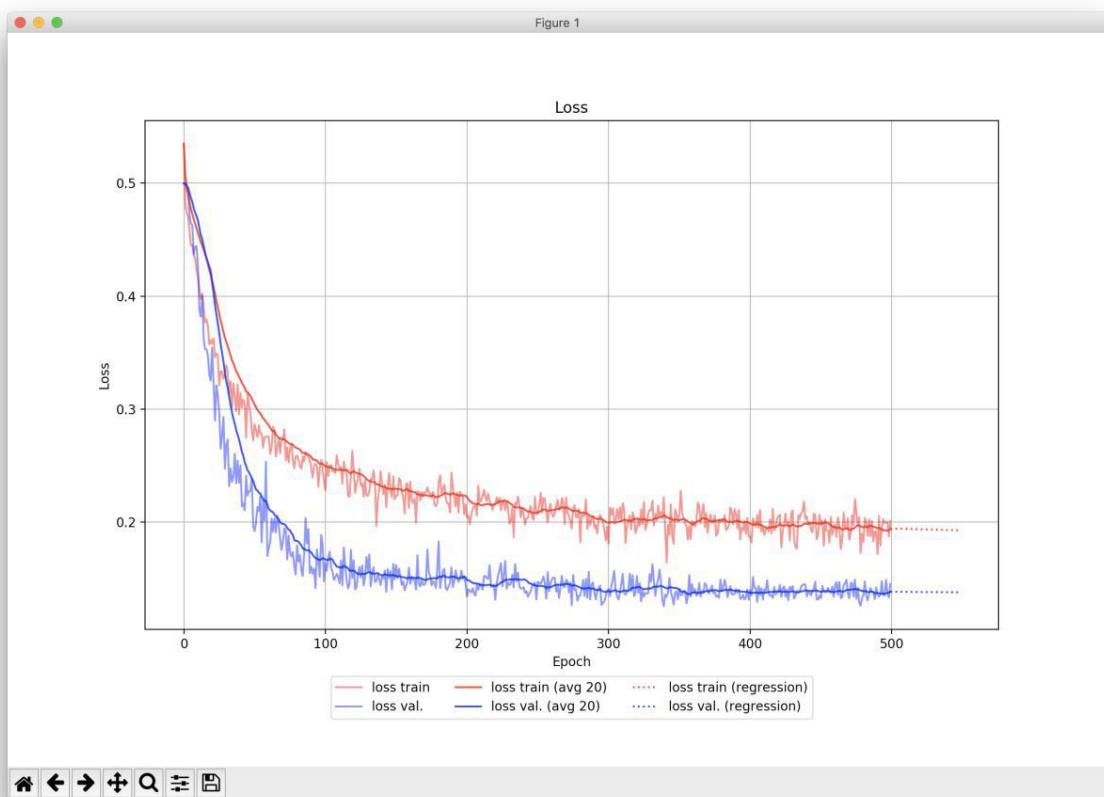
12



13

Options:**Size of training set:** 250**Size of test set:** 1000**Number of iterates:** 500**Optimization:** Stochastic Gradient Descent**Activation Function:** Sigmoid**Regularization:** Drop out**accuracy:** 84.30%

14



15

Options:

Size of training set: 500

Size of test set: 1000

Number of iterates: 500

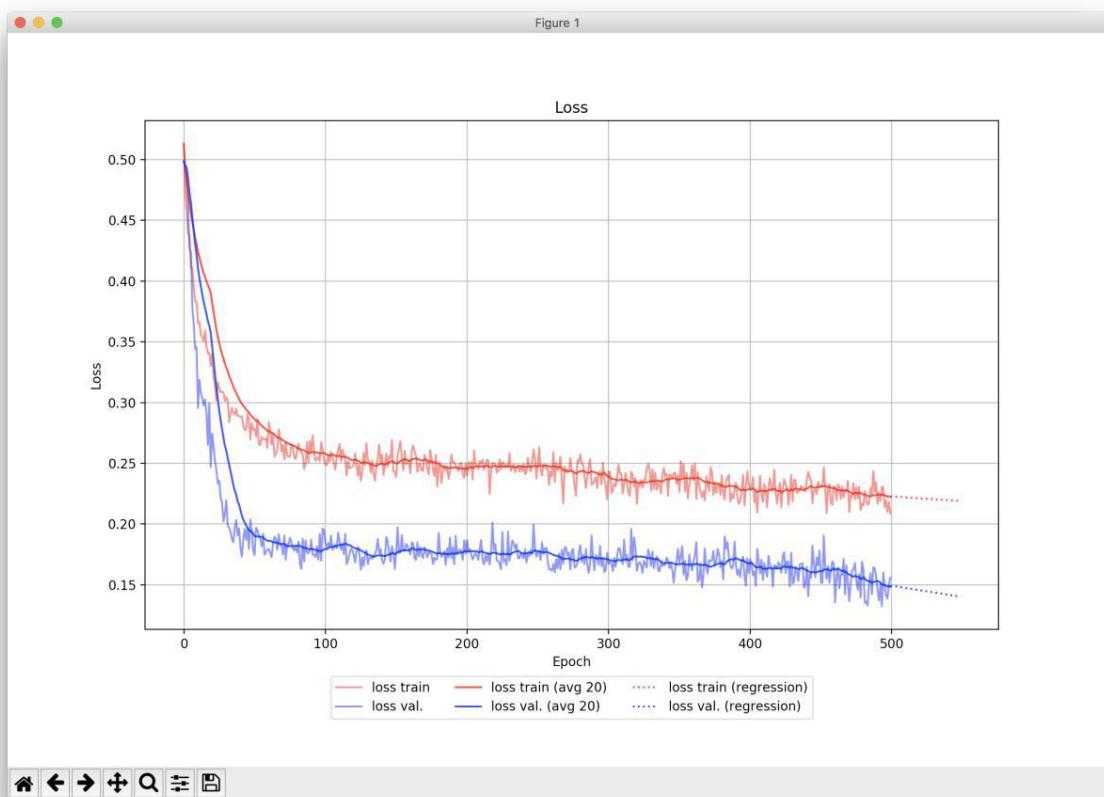
Optimization: Stochastic Gradient Descent

Activation Function: Sigmoid

Regularization: Drop out

accuracy: 84.20%

16



17

Options:

Size of training set: 750

Size of test set: 1000

Number of iterates: 500

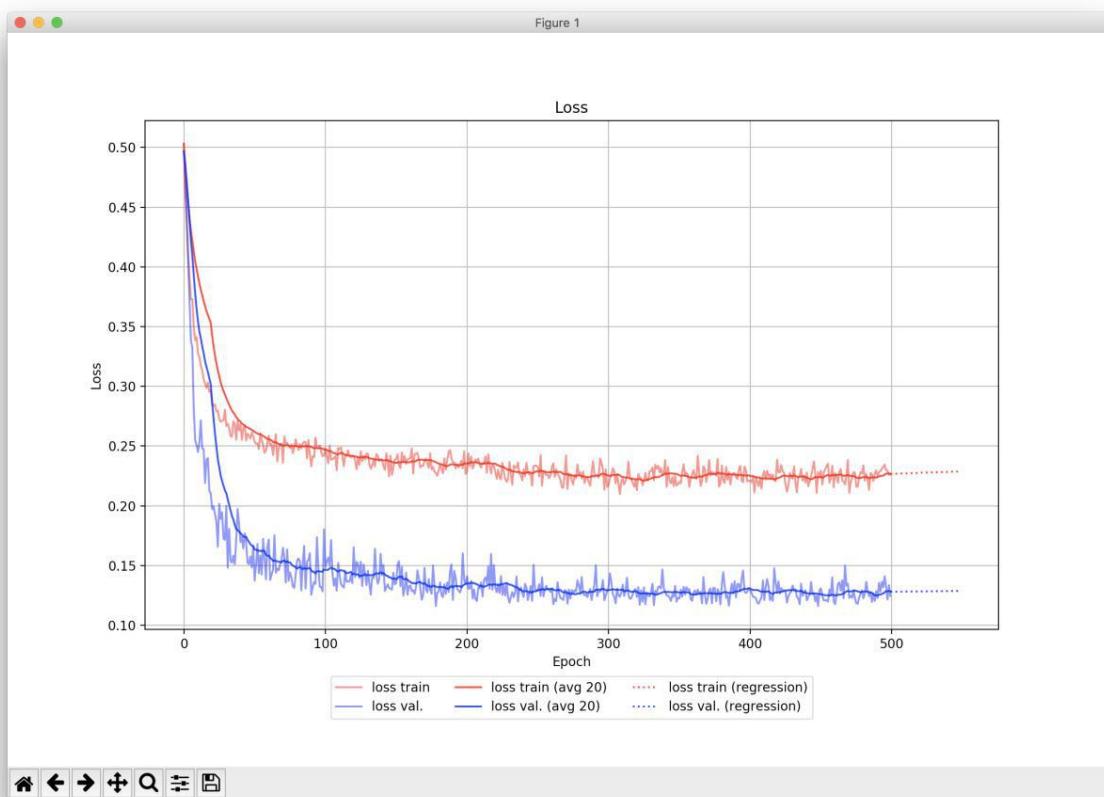
Optimization: Stochastic Gradient Descent

Activation Function: Sigmoid

Regularization: Drop out

accuracy: 86.30%

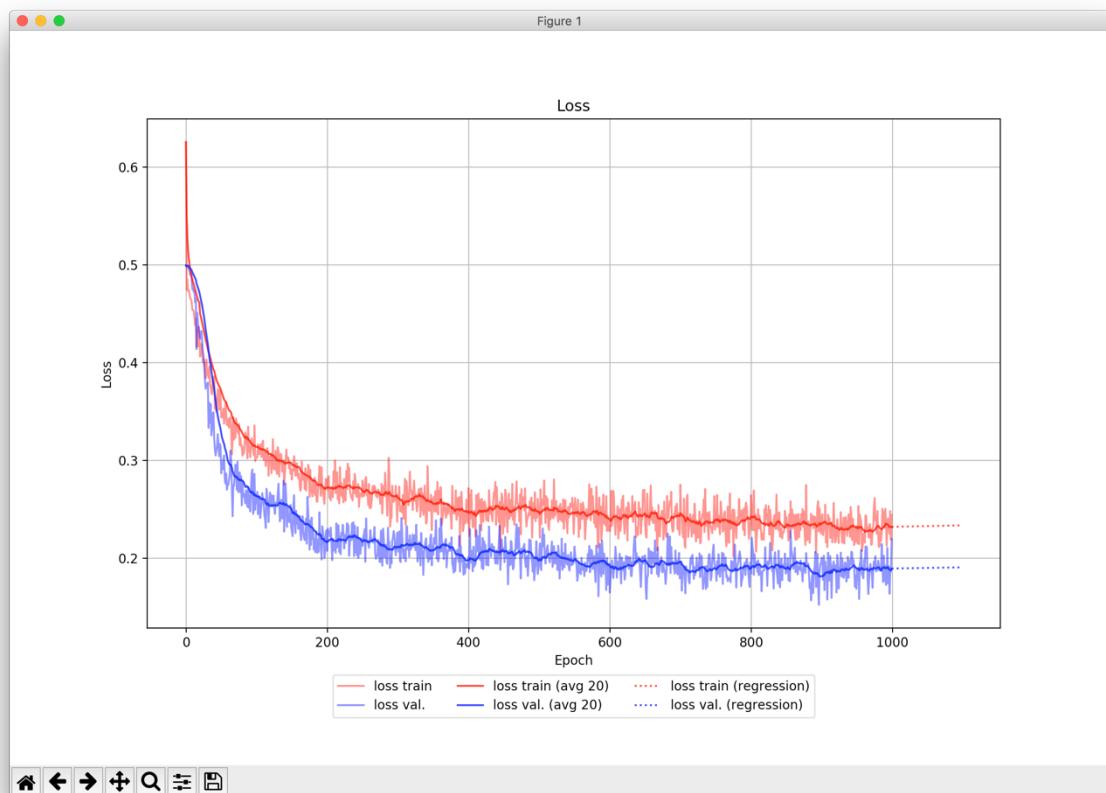
18



19

Options:**Size of training set:** 1500**Size of test set:** 1000**Number of iterates:** 500**Optimization:** Stochastic Gradient Descent**Activation Function:** Sigmoid**Regularization:** Drop out**accuracy:** 88.00%

20

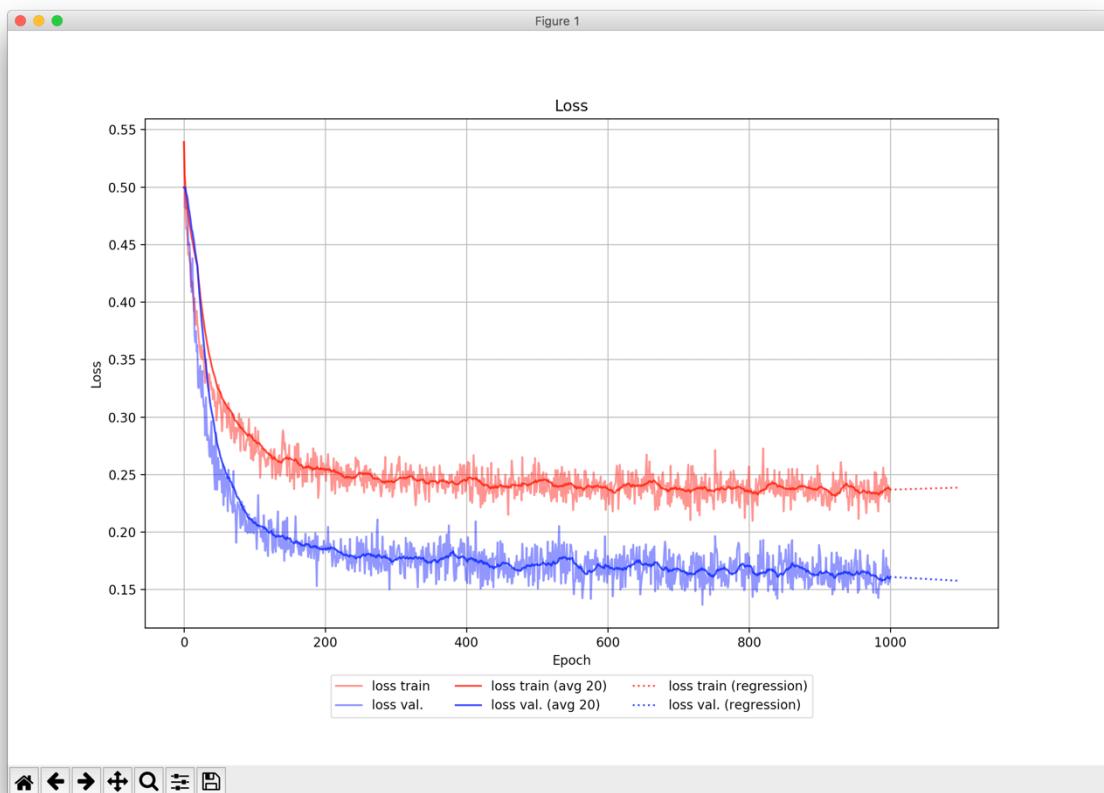


21

Options:

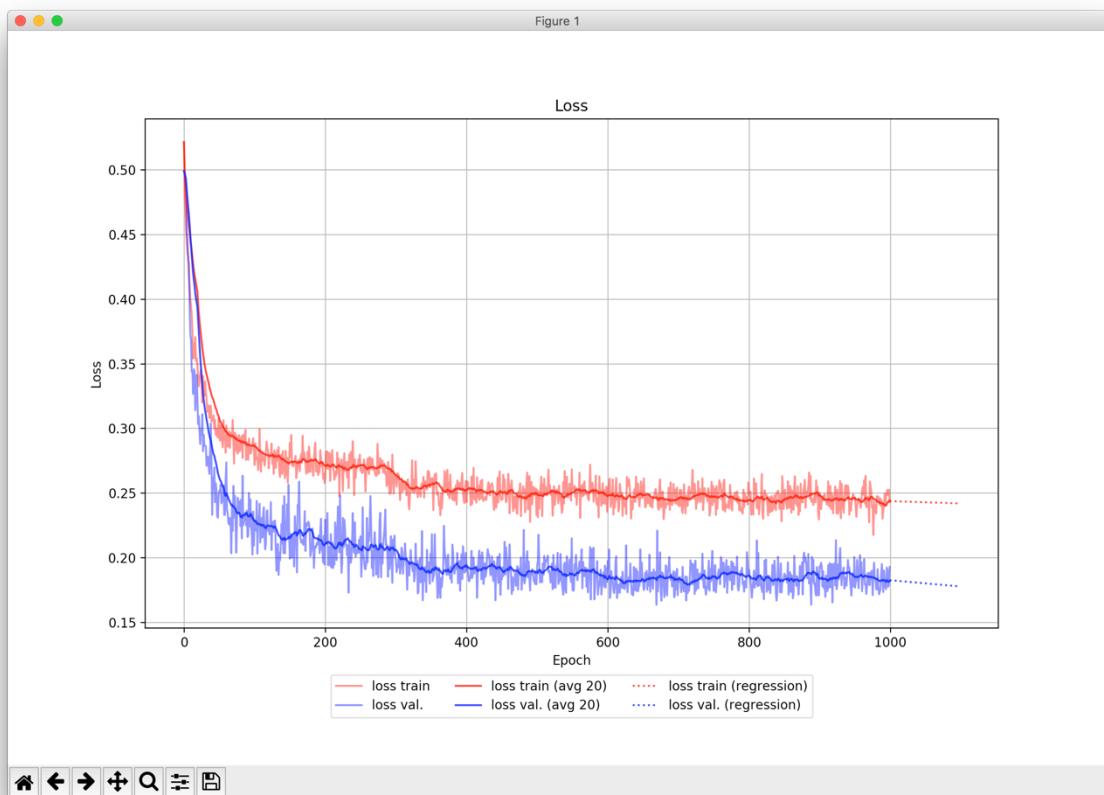
Size of training set: 250
Size of test set: 1000
Number of iterates: 1000
Optimization: Stochastic Gradient Descent
Activation Function: Sigmoid
Regularization: Drop out

22 accuracy: 84.20%



23

Options:**Size of training set: 500****Size of test set: 1000****Number of iterates: 1000****Optimization: Stochastic Gradient Descent****Activation Function: Sigmoid****Regularization: Drop out**24 accuracy: 82.20%



25

Options:

Size of training set: 750

Size of test set: 1000

Number of iterates: 1000

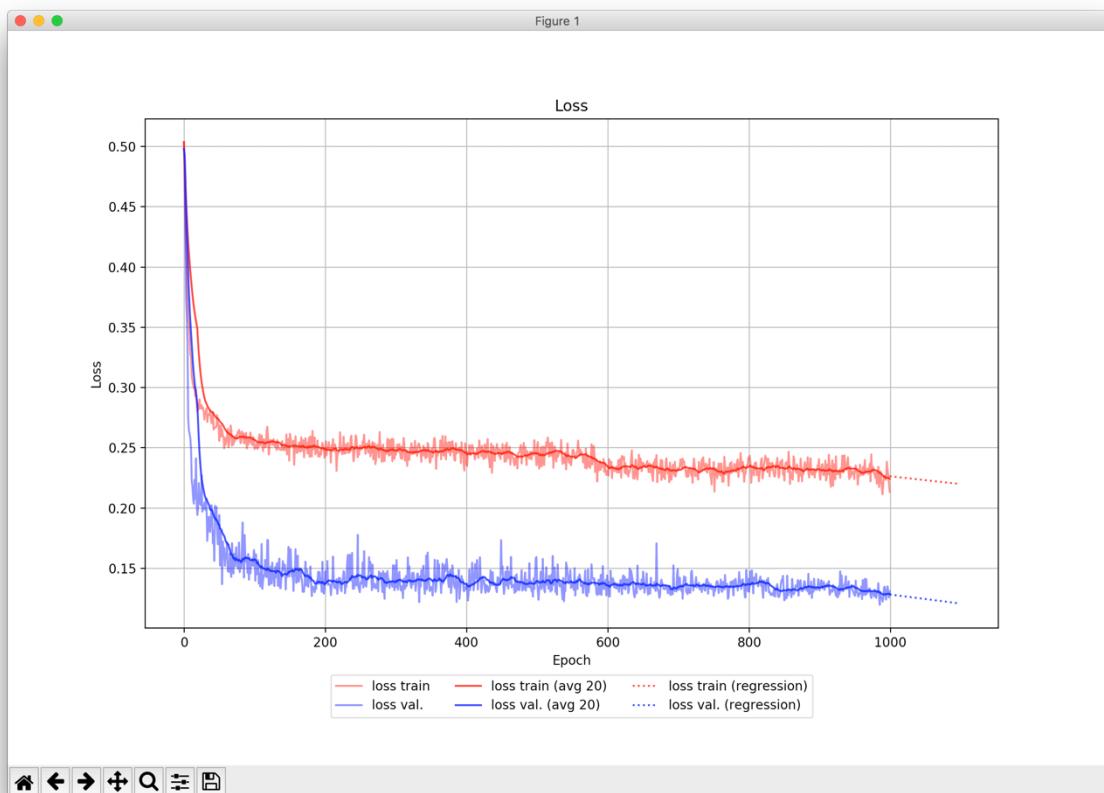
Optimization: Stochastic Gradient Descent

Activation Function: Sigmoid

Regularization: Drop out

accuracy: 84.10%

26 ■



27

Options:

Size of training set: 1500

Size of test set: 1000

Number of iterates: 1000

Optimization: Stochastic Gradient Descent

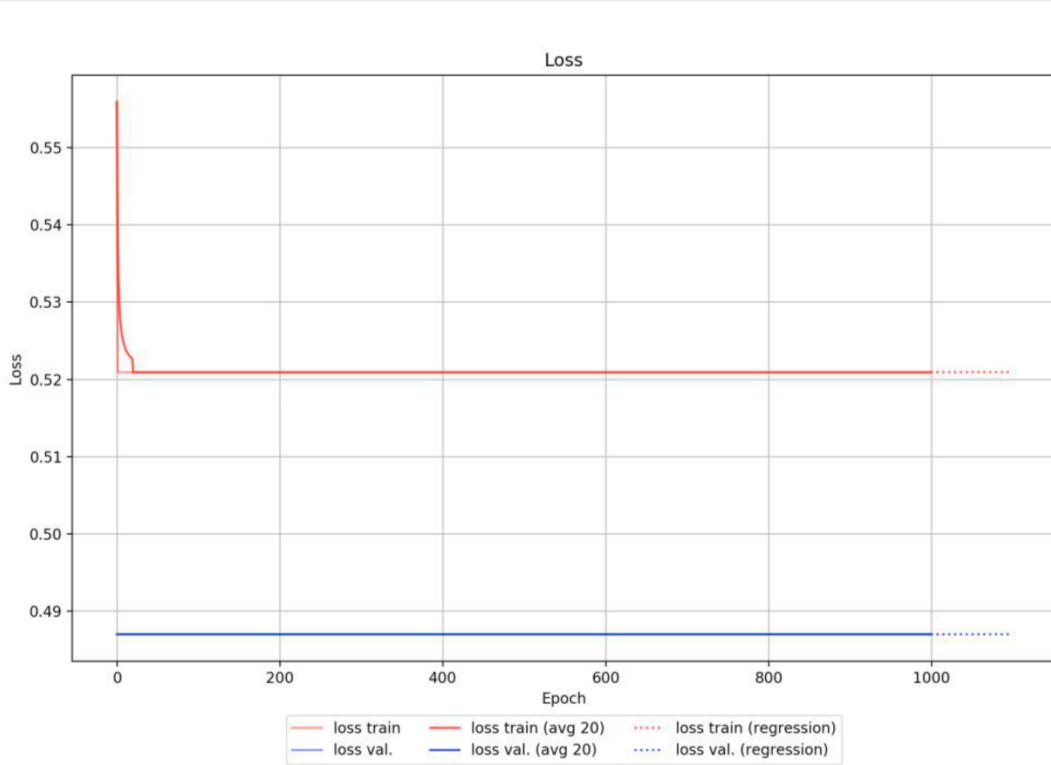
Activation Function: Sigmoid

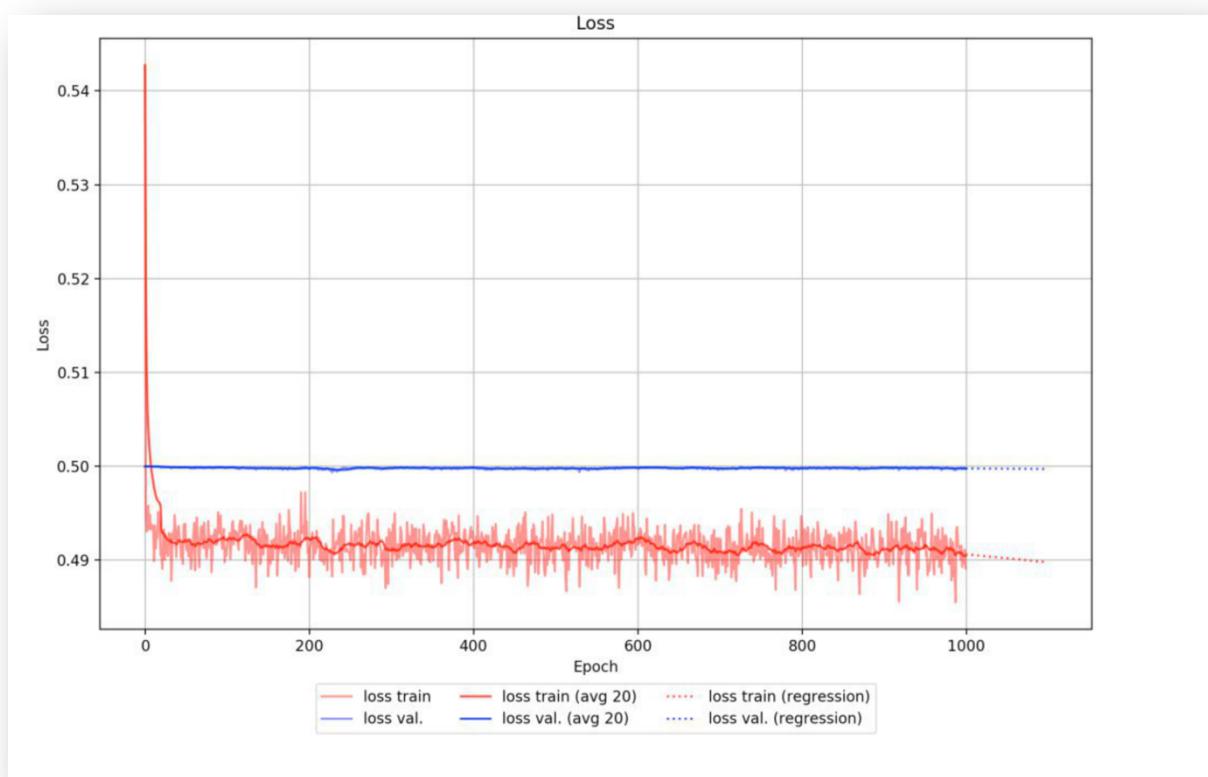
Regularization: Drop out

accuracy: 86.20%

28

29

**Options:****Size of training set:** 750**Size of test set:** 1000**Number of iterates:** 1000**Optimization:** Gradient Descent**Activation Function:** Sigmoid**Regularization:** Drop out**accuracy:** 9.90%

**Options:**

Size of training set: 750

Size of test set: 1000

Number of iterates: 1000

Optimization: Gradient Descent

Activation Function: Sigmoid

Regularization: Drop out

accuracy: 9.90%

تحلیل نتایج

در پایان باید گفت در این مساله **option** هایی که به کاربر داده می شود خود میتوانند موضوع بحث باشد که در چنین مساله ای به طور خاص چه روش هایی برای **regularization** , **optimization** و یا **activation function** بهینه خواهد بود. هم چنین متغیر هایی هم چون اندازه i دقت پیش بینی شبکه را تحت تاثیر قرار دهند. طبق آزمون و خطای ازای مقداردهی های مختلف و نیز (dropout/SGD/sigmoid). تحقیقاتی که در این زمینه انجام دادیم روش های دیفالتی را به کار گرفتیم. در چراکه مثلا **dropout** با بزرگ شدن شبکه **robustness** و **performance** بالاتری خواهد داشت. در مورد انتخاب تابع **sigmoid** هم بایستی گفت که در صورت استفاده از تابع فعالساز $A = cx$ قابلیت چند

لایه شدن شبکه بی فایده خواهد بود. چرا که در واقع خروجی لایه‌ی آخر به طور مستقیم تابعی خطی از ورودی اولیه خواهد بود و هر تعداد لایه‌ی میانی در این شرایط می‌تواند با یک لایه جایگزین شود. از طرفی مشکل این است که مشتق اینتابع نسبت به x برابر یک مقدار ثابت است و عملاً backpropagation مفهومی نخواهد داشت. در آخر هم می‌توان گفت علت انتخاب SGD به عنوان دیفالت این است که در صورت رخداد local minima این روش از گیرافتادن در آن جلوگیری می‌کند. همچنین روش GD بار محاسباتی بیشتری خواهد داشت. با بررسی پارامترهای مختلف و رسم loss function به ازای انتخاب‌های مختلف توانستیم مقادیر پیشفرضی نیز برای یک سری ثابت در نظر بگیریم.

29-1 جمع‌بندی و نتیجه‌گیری

در این آزمایش توانستیم با انتخاب‌های نسبتاً بهینه و همچنین مورد آزمایش قرار دادن پارامترهای مختلف انتخاب‌های نسبتاً خوبی داشته باشیم تا شبکه توانایی این را داشته باشد که در صورت مقابله با یک تصویر از حرف با احتمال بالایی توانایی تشخیص آن را داشته باشند. در آخر با اعمال این تغییرات هم چنین به کارگیری فرمول‌های مورد نیاز توانستیم به یک شبکه‌ی عصبی دست یابیم که با دریافت یک عکس و تنظیمات دیفالت با احتمال بالایی آن را حدس بزند.

29-2 پاسخ سوالات تء وری

۱) در مرحله‌ی feed-forward از لایه‌ی input شروع و در طول لایه‌ها به انجام محاسبات روی آن می‌پردازیم و خروجی واقعی مدل را نهایتاً در لایه‌ی خروجی به دست می‌آوریم. از آن جا که جریان محاسبات از لایه‌ی خروجی به سمت لایه‌ی خروجی انجام می‌شود به این مرحله feed-forward گفته می‌شود.

در مرحله‌ی feed-backward در واقع از loss function استفاده می‌کنیم و با انجام مشتق گیری‌های لازم از نقطه‌ی پایان به سمت نقطه شروع حرکت می‌کنیم و update های لازم را در وزن‌ها و بایاس‌ها ایجاد می‌کنیم

Backpropagation Steps:

Input -> Forward calls -> Loss function -> derivative -> backpropagation

(۲)

: Supervised ✓

در یادگیری با ناظر ممکن است روابط داخلی میان داده‌های در حال پردازش از پیش مشخص نباشد اما ما می‌دانیم که هدف ما مدل کردن چه خروجی خواهد بود. آن‌چه از پیش نمی‌دانیم این هست که خروجی مطلوب را چگونه به دست آوریم بنابراین شما می‌توانید مجموعه‌ای از داده‌های موجود را برای "training" یک مدل برای پیش‌بینی‌ها یا پاسخ‌های خاص استفاده کنید. آموزش مدل معمولاً از بخشی از داده‌ها برای یادگیری استفاده می‌کند و بخشی از داده‌ها برای اعتبار سنجی و اندازه‌گیری دقیق مدل است.

: Unsupervised ✓

در الگوریتم های بدون ناظر، هنوز نمی دانید که خروجی مدل چه خواهد بود. شما احتمالاً حدس می زنید که برخی روابط بین داده های شما وجود دارد، اما داده ها برای حس زدن بسیار پیچیده است. بنابراین در این موارد شما داده های خود را **normalize** می کنید که قابل مقایسه باشند و اجازه می دهید که مدل به کار خود ادامه دهد و برخی از این روابط را به دست آورد. یکی از ویژگی های خاص این مدل این است که در حالی که در تواند راه های مختلفی برای دسته بندی یا مرتب سازی داده های شما پیشنهاد کند، این بر عهده شماست که اجازه دهید که جست و جو را ادامه داده و نتایج مفید بیشتری به دست آورد. ادامه می این فعالیت می تواند به عنوان تقویت اطلاعات راجع به روابط داخلی تلقی شود.

: Reinforcement ✓

این الگوریتم مساله ها و تکنیک هایی را که به منظور ارتقاء آن تلاش می کند، مدل سازی می کند. به منظور انجام این کار، RL نیاز به توانایی sense دارد و به صورت خودکار تصمیم گیری در مورد یک action انجام می دهد و سپس نتیجه را با یک تعریف از reward مقایسه می کند. RL تلاش می کند تا چگونگی به حداقل رساندن این reward ها را بفهمد، اما این کار را به خودی خود (با دستور العمل مستقیم) انجام نمی دهد. Agent با توجه به حالت environment و شرایط action ای را انتخاب می کند که به بیشترین پاداش را دریافت کند. این action ها state و environment را عامل را تغییر می دهند. هم چنین با دادن پاداش به agent وقفه در آن ها ایجاد خواهد شد. با اجرای این حلقه و تکرار شدن آن agent رفتار خود را بهبود می بخشد.

: overfitting (۳)

این مشکل زمانی پیش می آید که آنالیزی از یک مجموعه داده ی خاص به دست آید به گونه ای که دقیقاً و با اختلاف خیلی کمی با آن دسته ی خاص تطابق دارد. همان طور که در شکل زیر مشاهده می شود خط سیاه یک مدل **regularized** و خط سبز یک مدل **overfitted** از داده ها است که بیش از به آن دسته خاص از داده وابسته است و احتمالاً در ازای مواجه شدن با داده ی ورودی جدید با خطای بیش تری مواجه می شود.

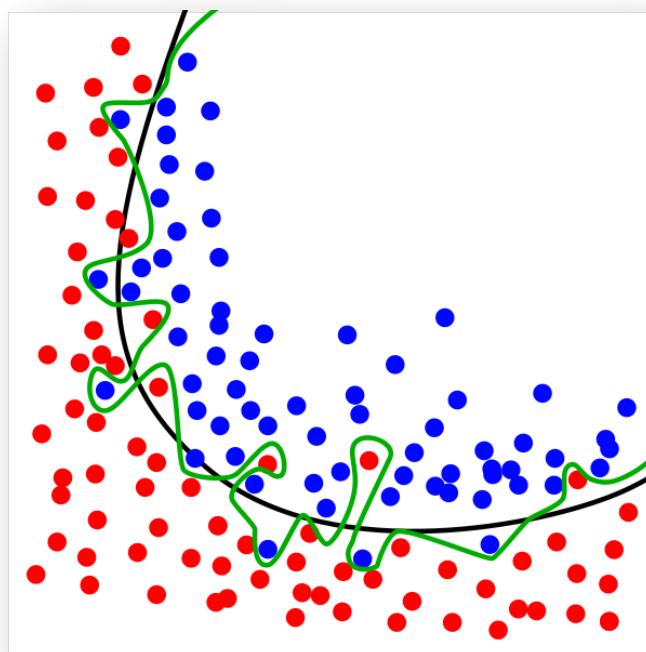


Figure2-overfitting

برای مشکل overfitting راه حل های زیادی وجود دارد که در ادامه به یک سری از آن ها اشاره شده است:

Cross Validation ✓

در روش استاندارد k -fold cross-validation داده هارا به k دسته افزایش می نماییم. $K-1$ دسته ای اول را در یک حلقه ای $k-1$ تایی train می کنیم و از داده های دسته ای k به عنوان test استفاده می کنیم.

Train more data ✓

این روش همیشه پاسخگو نیست اما training با داده ای بیشتر کمک میکند که سیگنال بهتری detect شود. این تکنیک با اضافه شدن داده های noisy راه حل مناسبی نخواهد بود.

Remove features ✓

یک سری از الگوریتم ها یک انتخاب built-in feature از input feature های غیر مرتبط را به منظور بالابردن generalizability حذف می نماییم.

Early stopping ✓

در iteration های مختلف از جایی به بعد توانایی generalization مدل کم می شود و خروجی شروع به overfit می کند. در این الگوریتم قبل از رسیدن به آن نقطه جلوی iteration های بعدی گرفته خواهد شد.

Regularization ✓

Ensembling ✓

.

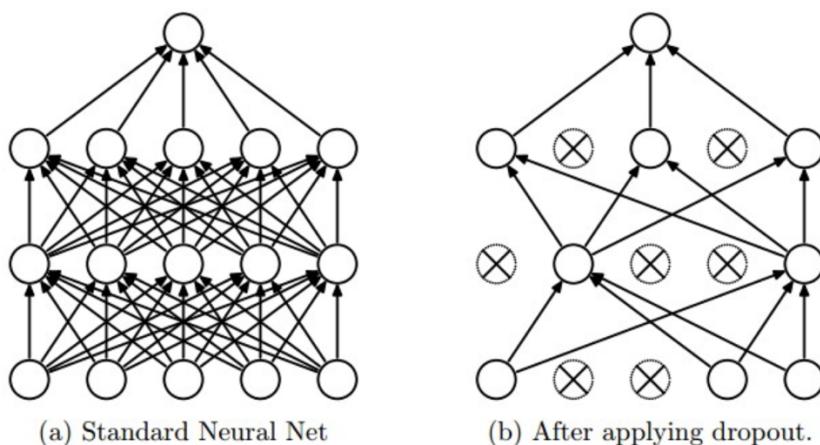
.

.

(۴)

Drop out ✓

در این تکنیک با احتمال $p=1-q$ تعدادی از گره های شبکه را از آن خارج می کنیم به این صورت که تمام اتصالات آن به لایه ای قبل و بعد نادیده گرفته شود. در این روش به طور موقت تعدادی از نورون ها را غیر فعال می نماییم. این روش در فاز training به منظور کاهش overfit feature به کار گرفته می شود. با dropout شبکه می تواند تمرکز بیشتری روی های دیگر داشته باشد.



L2Norm ✓

در این روش مربع اندازه ای ضرایب به عنوان penalty term به تابع اضافه می شود. بخش highlight شده المان regularization را در روش L2 نمایش می دهد. در صورتی که لامدا برابر صفر باشد به روش قبل بازمی گردیم و در صورتی که خیلی بزرگ باشد وزن زیادی را اضافه می کند که موجب underfitting خواهد شد. پس انتخاب لامدا اهمیت پیدا می کند.

$$\sum_{i=1}^n (y_i - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

Cost function

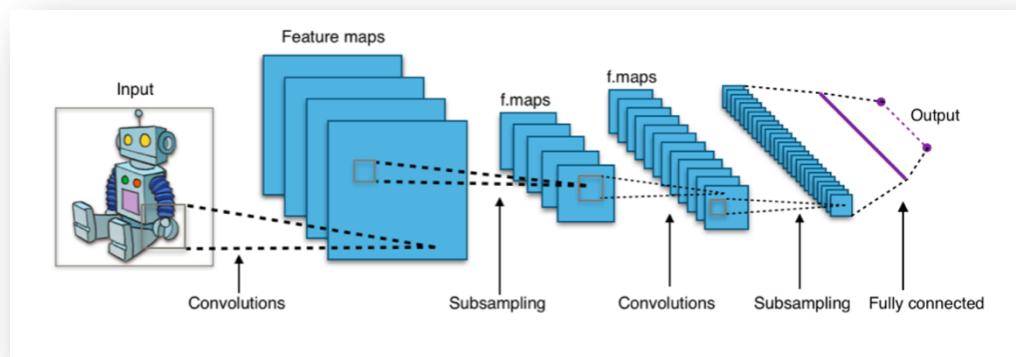
فایده ای اصلی dropout در این است که اجازه می دهد که یک شبکه تعداد زیادی از زیرشبکه ها را به صورتی کم هزینه و آسان شامل دو بخش traing و test را مدل می کند. نتایج آزمایشات نشان می دهد که dropout نه تنها کارایی بهتری در شبکه های بزرگ دارد بلکه

از روش L2 دارای robustness بیشتری خواهد شد. در مقابل L2 دقیق پیش‌بینی بالاتری در شبکه‌های کوچک‌تر دارد چرا که میانگین learning model می‌تواند کارایی کلی را وقتی تعداد submodel‌ها زیاد است بالا ببرد و هر یک از این submodel‌ها بایستی از دیگری متفاوت باشد.

۵) در بخش نتایج خروجی‌های loss function نمایش داده شده است.

(۶)

Convolutional •



در این گونه شبکه‌های عصبی از یک یا بیشتر لایه‌ی convolutional تشکیل شده است که هر یک از یک convolution operation روی ورودی که نتایج را به لایه‌ی بعدی دهد استفاده می‌کند و این عملیات اجزا می‌دهد که شبکه‌ای عمیق‌تر با تعداد کم تری از پارامترها ایجاد شود. این گونه شبکه‌ها خروجی‌های بسیار خوبی در اپلیکیشن‌های تصویری و صوتی دارند.

Recurrent •

در این گونه شبکه‌ها اتصالات بین نورون‌ها به گونه‌ای است که یک حلقه‌ی جهت دار ایجاد می‌کند. این به این معناست که خروجی نه تنها به ورودی فعلی بلکه به مراحل قبلی state فعلی نورون وابسته است. این حافظه به حل مسائل NLP همچون connected speech recognition و handwriting recognition کمک می‌کند.

مقایسه:

CNN

- ۱-CNN تعداد ثابت ورودی می‌گیرد و تعداد ثابتی خروجی می‌دهد.
- ۲-CNN یک نوع از feed-forward که تنوعی از perceptron های چندلایه دارند که به منظور استفاده در حجم کمی از پیش پردازش طراحی شده‌اند.
- ۳-CNN ها از الگو های اتصال بین نورون‌ها استفاده می‌کنند که الهام گرفته از ساختار visual cortex در حیوانات است که نورون‌های آن به این منظور ترتیب داده شده‌اند که به نواحی overlapping که کنار یک دیگر می‌تواند یک visual field را تشکیل دهند پاسخ می‌دهند.
- ۴-CNN ها در پردازش تصاویر و ویدیو‌ها ایده‌آل هستند.

RNN

- ۱-RNN طول ورودی و خروجی اش دلخواه است.
- ۲-RNN برخلاف feedforward از حافظه‌ی داخلی خود برای پردازش دنباله‌ی دلخواهی از ورودی‌ها استفاده می‌کند.
- ۳-این گونه شبکه‌ها از یک سری اطلاعات در طول زمان بهره می‌گیرند.(time-series).
- ۴-RNN ها در تحلیل متن و کلام ایده‌آل هستند.