التعلم العميق Deep Learning

- **الدرس الأول : التعلم العميق و الشبكات العصبية**
  - الأسبوع الأول : مقدمة للتعلم العميق
  - الأسبوع الثاني : أساسيات الشبكات العصبية
  - الأسبوع الثالث : الشبكات العصبية المجوفة
  - الاسبوع الرابع : الشبكات العصبية العميقة
- **الدرس الثاني : تطوير الشبكات العميقة : المعاملات العليا**
  - الأسبوع الأول : السمات العملية للتعلم العميق
  - الأسبوع الثاني : الحصول علي القيم المثالية
  - الأسبوع الثالث : ضبط قيم الشبكات العميقة
- **الدرس الثالث : هيكلية مشاريع الـ ML**
  - الأسبوع الأول : استراتيجيات الـ ML - 1
  - الأسبوع الثاني : استراتيجيات الـ ML - 2
- **الدرس الرابع : الشبكات العصبية الملتفة CNN**
  - الأسبوع الأول : أساسيات الشبكات العصبية الملتفة
  - الأسبوع الثاني : حالات عملية من الشبكات العصبية الملتفة
  - الأسبوع الثالث : التعرف علي الأشياء
  - الاسبوع الرابع : التعرف علي الوجه
- **الدرس الخامس : الشبكات العصبية المتكررة RNN**
  - الأسبوع الأول : مفهوم الشبكات العصبية المتكررة
  - الأسبوع الثاني : المعالجة اللغوية الطبيعية NLP
  - الأسبوع الثالث : نماذج التتابع

درس 2: تطوير الشبكات العميقة : المعاملات العليا

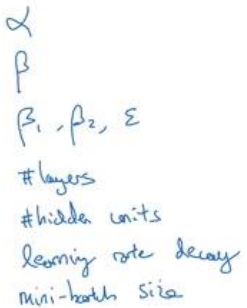
الأسبوع الثالث : ضبط قيم الشبكة العميقة

==========

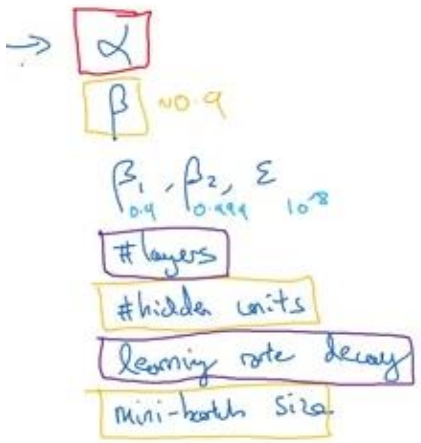
سنتناول في هذا الكورس عددا من الخطوات المطلوب معرفتها , لتحديد قيم الـ , hyperparameters , وهي القيم العليا الخاصة بالشبكة , والتي علي اساسها نقوم بضبطها (مثل معامل التعلم الفا)

أحد الصعاب الخاصة بالتعلم العميق , وهو تحديد عدد من العوامل المطلوب منك تحديدها , مثل :

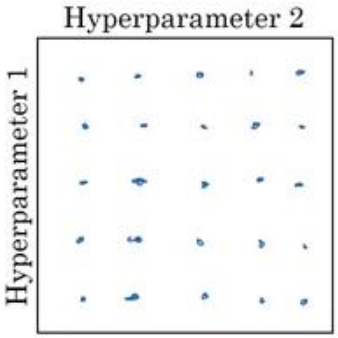
- الفا : معامل التعلم learning rate
- بيتا : معامل الزخم momentum term
- بيتا 1 و بيتا 2 : معاملات آدم adam optimization algorithm
- عدد الطبقات
- عدد الوحدات الخفية لكل طبقة
- معدل تقليل معامل التعلم learning rate decay
- حجم الاجزاء المقسمة mini-batch size



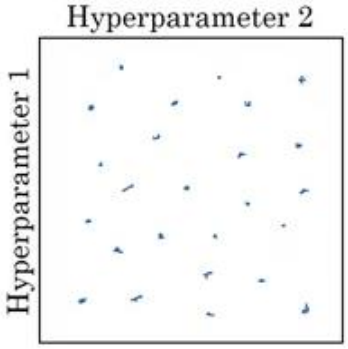
ولاحظ إن بيكون فيه عوامل أهم من عوامل , يعني العوامل ذات اللون الأحمر (الفا) هي اكثرهم أهمية , يليها ذات اللون الأصفر , يليها ذات اللون القرمزي



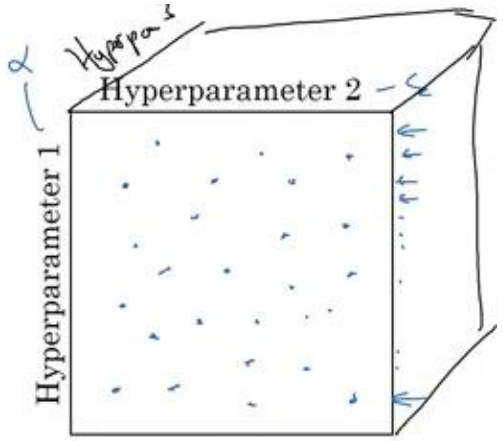
وهنا فيه نصيحة مهمة , متعلقة باختيار المعاملات . .



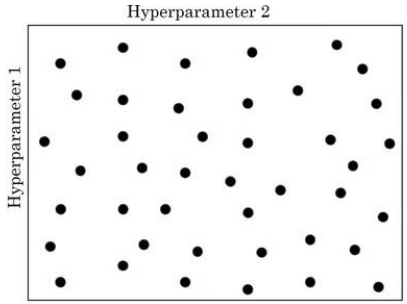
في الماضي كانو بيعملو جدول منظم من المعاملات المختلفة , بين نوعين من المعاملات (مثلا الف و ابسلون), لاختيار انسب معامل فيهم اللي يحافظ علي القيم المثلي هنا و هنا



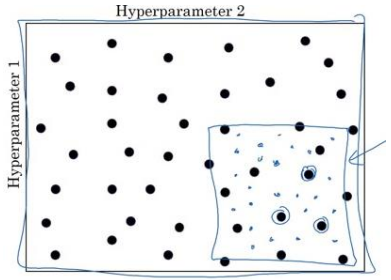
لكن حديثا وجدو ان عمل جدول عشوائي من القيم بيكون له نتائج افضل , و ده لتجنب تضييع الوقت في الفحص بشكل طويل القيم بالترتيب , خاصة لو كان هناك معامل شديد الأهمية (الفا) مقابل معامل اقل أهمية (ابسلون) . .



ومن الخطوات الذكية في البحث الادق , لو كان عندي مثلا 100 قيمة للمعامل الأول , و100 قيمة للمعامل الثاني , فيكون لدي 10 الاف قيمة مشتركة , فبدلا من فحص 10 الاف قيمة , اقوم فقط بإظهار 50 منهم بالشكل التالي . .



وبفرض أني وجدت أن أفضل قيم موجودة تحت علي اليمين , وقتها اعمل فوكس علي هذا الربع من القيم و اقوم بتجاهل الباقي , واقوم فقط بفحص المزيد من القيم في هذا المكان , ثم اقوم بعمل زوم اكتر و اظهار قيم ادق , حتي اصل لافضل قيمة في اسرع وقت



والفكرة تتطور اذا ما كان هناك اكتر من معاملين , مثلا 3 او 4 , وقتها لابد من رص القيم بشكل عشوائي , والفحص العشوائي بينهم يكون افضل

-

و هنا هيظهر سؤال , لو كنت أريد تحديد القيم الخاصة بهذه المعاملات , (مثلا قيمة الفا) , فلابد ساقوم بالبحث خلال مدي من الارقام (مثلا من 0.001 إلي 0.8) , فكيف اقوم بتحديد هذا المدي و البحث خلاله في مدة قصيرة و بشكل دقيق

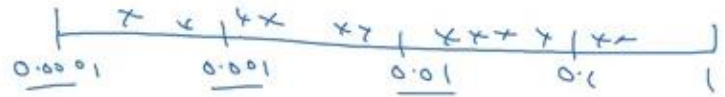
فلو قلنا أنني اريد تحديد عدد خلايا طبقة معينة , وأن المدي المقبول لها هو من 50 إلي 100 , فيسهل رص الأرقام من 50 إلي 100 و وفحص كلا منهم للوصول للرقم المناسب . .

لكن ماذا عن الأرقام التي يصعب حصرها خاصة و انها عشرية او كسرية و ليست صحيحة , مثل ألفا ؟

فلو كان المدي المقبول لألفا هو من 0.0001 إلي 1 مثلا , فلو قمنا بعمل رص خطي للأرقام هكذا :

فسيكون هذا توزيع غير مناسب للأرقام , ولا يتناسب مع الفحص الدقيق لها , حيث أن القيم المطلوب فحصها ما بين 0.0001 إلي 0.1 سيكون فقط 10 % من الوقت , بينما هي تحتوي علي الاف الارقام الهامة . .

بينما لو قمنا بعمل رص "أسي" للأرقام بهذا الشكل :



فنجد أن الأرقام أكثر منطقية , والتوزيع نفسه أكثر دقة في الوصول للرقم المناسب

و هذا يمكن عمله بسهولة في بايثون عبر السطرين :

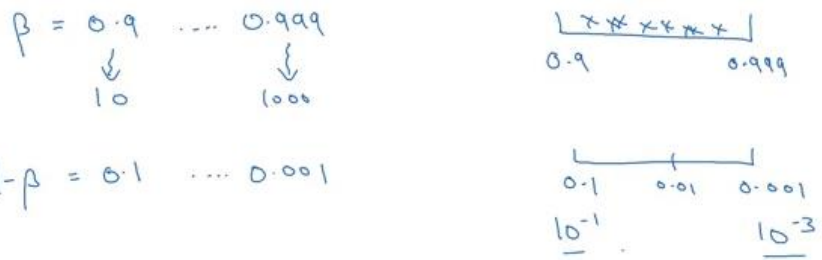
```
r = -4 * np.random.rand()
alpha = 10**r
```

ففي السطر الأول , قيمة r ستكون ارقام عشوائية تتراوح بين سالب 4 و 0 , وحينما يتم رفعها أس للعشرة , تكون قيمة الفا تتراوح بين 0.0001 و 1

وبالتالي كقاعدة عامة , لو عندك العامل المطلوب يتراوح بين 10^a و 10^b , وقتها اعمل (لوج) للرقمين , ستظهر كلا من a , b وتكون r تتراوح بين a , b حيث أن العامل المطلوب هو 10 أس r

فكرة كمان في نفس النطاق , حينما اقوم بتحديد قيمة بيتا (وهو المعامل الأسي) , والتي غالبا تتراوح بين 0.9 (متوسط 10 قيم سابقة) و 0.999 (متوسط الف قيمة سابقة)

فيمكن عملها بسهولة , عبر ان نقول ان بيتا هي 1 ناقص نفس فكرة القيم السابقة والتي سنقول انها تتراوح بين 0.1 و 0.001 بهذا الشكل :



لاحظ ان الان القيمة الاكبر علي اليسار 0.1 , والاصغر علي اليمين 0.001 , وذلك بسبب انها بعد الطرح سيتم عكس الترتيب . .

فنقول ان r تتراوح بين 3- و 1- و أن قيمة بيتا تساوي (1-10^r)

والسبب اني استخدم النظام الأسي في رص الأرقام , أن الحساسية تزداد جدا للشبكة , مع اقتراب المعامل مع القيم التي تزود في تفاعلها . .

فمثلا الفرق الذي تقوم به بيتا في تغيرها من 0.9000 إلي 0.9005 هو فرق هامشي لا يكاد يذكر
بينما الفرق الذي تقوم به في التغير من 0.999 إلي 0.9995 هو فرق ضخم جدا في الشبكة

لأن أساسا استخدام البيتا يكون عبر قانون :

1/(1- β)

لهذا السبب , فرص الارقام بشكل خطي غير صحيح , ولابد من رصها بشكل أسي .

-

فيه مبدأ في العلم اسمه : cross fertilization

ترجمته الحرفية : التخصيب المتقاطع . .

لكن معناه الاصطلاحي : تعميم الفائدة . .

و اللي معناه هنا , إن في تطبيقات الـ ML المختلفة , كتير من الافكار اللي نجحت في مجال معين (قراءة الحروف) ممكن يتم تطبيقها بنجاح في مجال تاني (التعرف علي الوجوه مثلا) .

طبعا الكلام ده مش عام , فيه افكار و تطبيقات في مجال معين , لو طبقناها في مجال تاني مش هتنفع و الدنيا هتبوظ فيها .

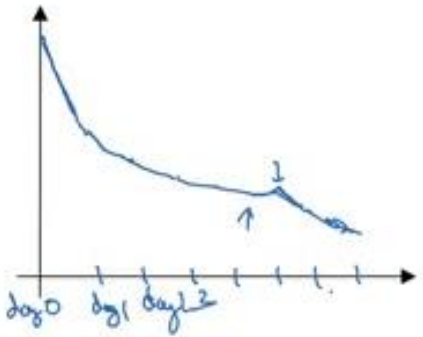
و ديه نفس الفكرة اللي بيتم تطبيقها في تحديد المعاملات , ان ممكن يتم تطبيق نفس القيم التقريبية للمعاملات العليا واللي نجحت في مجال معين ,ممكن يتم تطبيقها في مجال تاني .

و بشكل عام فيه اتجاهين في التناول ده . .

التناول الأول : نموذج جليسة الأطفال baby sit model

وده بيكون في حالة ان عندك كمية ضخمة من البيانات , وبالتالي هتاخد وقت كتير وفي نفس الوقت مش عندك قدرة علي استئجار cloud processor لو انك تجيب عدد من اجهزة الكومبيوتر عشان تتعامل معاها , فده هيجعل المعالجة بطيئة للغاية , وممكن تحتاج علي ايام كثيرة . .

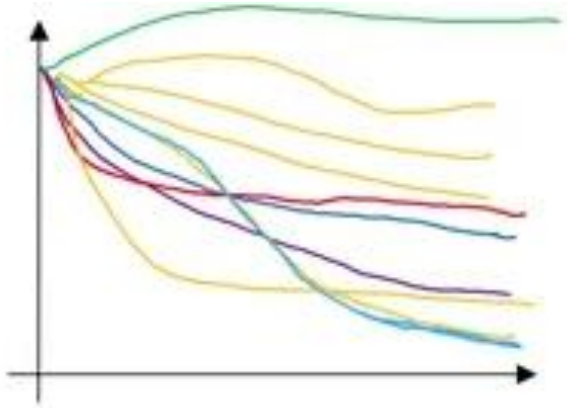
و هنا يتم تفعيل اسلوب "baby sit" يعني انك تراقب المعالجة علي مدار كذا يوم , بحيث مرة في اليوم تراقب الأداء , وعلي اساس اداء كل يوم , وعلي اساس قيمة الـ j تقوم بتغيير المعاملات . .



فتأتي كل يوم للعملية , وتقوم بتغيير قيم الفا , او بيتا , او ابسلون ,و غيره , علي اساس هل الـ J تزداد ام تقل , ولو وجدت في يوم رقم 7 ان زيادة معامل الفا مثلا , قام بزيادة قيمة J فتقوم بارجاع الفا في اليوم 8 , الي قيمتها في يوم 6 . .

والطريقة ديه احيانا بتسمي : طريقة الباندا , لأن الباندا بيكون ليها طفل واحد , بترعاه من غير ما تنتبه لغيره

الطريقة التانية , في حالة إن عندك أكتر من بروسيسور , يقدر يعمل معالجة للبيانات بالتوازي , وقتها ممكن نعمل شئ افضل . .



وهو اننا نعمل معالجات مختلفة بقيم معاملات عليا متنوعة , علي اكتر من جهاز بالتوازي , ونعمل مراقبة لكل جهاز فيهم , لنري ايهم افضل عبر مراقبة كلا منهم , سنري بوضوح ان هناك معالجات افضل من معالجات اخري , وبالتالي يتم استخدام القيم الافضل و التعامل معها في باقي المعالجات . .

و هذا يتم علي جزء من البيانات و ليس كلها , بالطبع لضيق الوقت .

و هذه الطريقة تسمي طريقة الاسماك , حيث ان الاسماك تضع عدد كبير من البيض و تنتظر منها من الذي سيفقس و ترعاه .

-

كنا شرحنا قبل كدة ازاي اننا ممكن نعمل تسوية للبيانات "normalization" , وازاي انها بتعمل تسريع كبير في الوقت المطلوب لعمل المعالجة . . بحيث الكونتير يتحول من بيضاوي لدائري . .

عن طريق الخطوات :

$$\mu = \frac{1}{m} \sum_i x^{(i)}$$
$$X = X - \mu \quad \leftarrow \text{elementwise}$$
$$\sigma^2 = \frac{1}{m} \sum_i x^{(i)2}$$
$$X = X / \sigma^2$$

الكلام ده شرحناه بالفعل بالنسبة للمدخلات البسيطة . .

فماذا عن الـ NN , واللي بيكون فيها مش مجرد مدخلات من الاول و بس , لكن ايضا بيكون فيه مدخلات في داخل الـ NN وهي عبارة عن الـ activations , وهي مخارج كل طبقة من الطبقات . .

وكأن , نريد ان نقوم بعمل تسوية للـ activations كي تقوم بس بتسريع عملية معالجة البيانات لكل طبقة علي حدة . .

و هذا هو ما بيسمي batch normalization

و ان كان هناك خلاف بين علماء الـ ML حول ايهم الذي يفضل ان اقوم بعمل تسوية له , a or z يعني قبل دالة السيجمويد (او التانش) او بعده , لكن الاغلب يفضلون الـ z , لضبطها قبل دخول دالة السيجمويد .

$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{(i)} - \mu)^2$$

$$z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$$

وهنا نقوم بعمل تسوية لقيم z بهذا الشكل , مع مراعة اننا قمنا باضافة قيمة ابسلون مضافة الي سيجما تربيع تحت الجذر , وذلك خوفا من ان تكون سيجما بصفر في اي حالة . .

و غالبا بيتم التحكم في قيمة z النهائية ,واللي بنسميها z tilde (وهو الرمز ~ يتم كتابته فوق الـ z لتكون $\tilde{z}$) بالمعادلة :

$$\tilde{z} = \gamma\, z + \beta$$

ان الزي تيلدا , تساوي جاما , مضروبة في زي بعد ما عملنا لها تسوية (norm) مجموعة لبيتا (وهي غير بيتا السابقة) .

فضرب الزي في قيمة , وجمعها علي قيمة يمكن التحكم فيها .

فلو قلنا ان جاما تساوي $\sqrt{\sigma^2 + \varepsilon}$ (وهو نفس المقام في معادلة التسوية) , وان بيتا تساوي الميو (وهي المطروحة من معادلة التسوية) , فنجد ان قيمة $\tilde{z}$ ستكون مساوية للـ z

وعبر اختلاف قيم جاما و بيتا تختلف $\tilde{z}$.

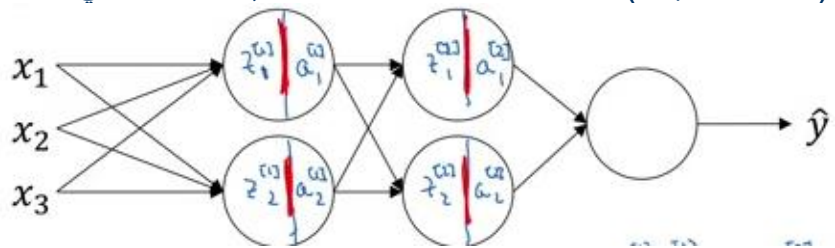
والسبب الرئيسي لجعل قيم $\tilde{z}$ تختلف عن z ان التسوية العادية بتجعل متوسط الارقام صفر , والارقام تتراوح حولها (مثلما قمنا في المدخلات inputs) لكن تطبيق هذا الامر في الـ z الداخلية تؤدي لعدم دقة البيانات الخارجة .

-

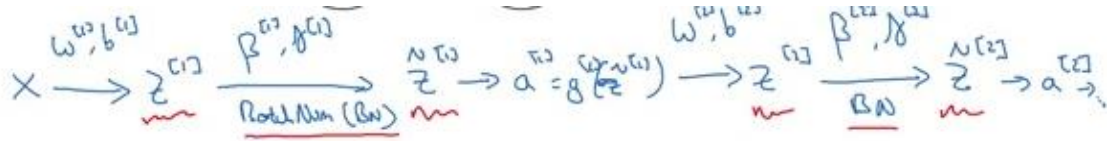
ما سبق كان عن تطبيق تسوية الـ activations لخلية واحدة .

فكيف يتم تعميمها علي باقي الشبكة ؟

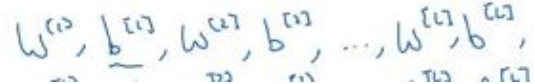
في الحالة العادية , نستخدم x لإيجاد قيمة z (عبر الـ w , b) و منها a عبر دالة السيجمود , و من a نأتي بـ z التالية و هكذا



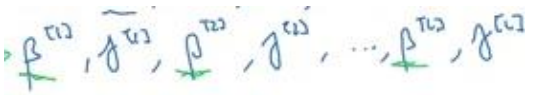
الشئ المختلف هنا اننا سنتوقف عند مرحلة z لنأتي بـ $\tilde{z}$, عبر استخدام جاما و بيتا , من $\tilde{z}$ نطبق السيجمويد لنأتي بـ a و نكرر العملية



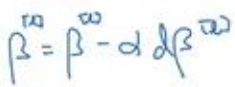
فبدلا من ان يكون المطلوب تعيين فقط الـ w , b لكل طبقة



سيكون مطلوب ايضا تحديد قيم جاما و بيتا لكل طبقة



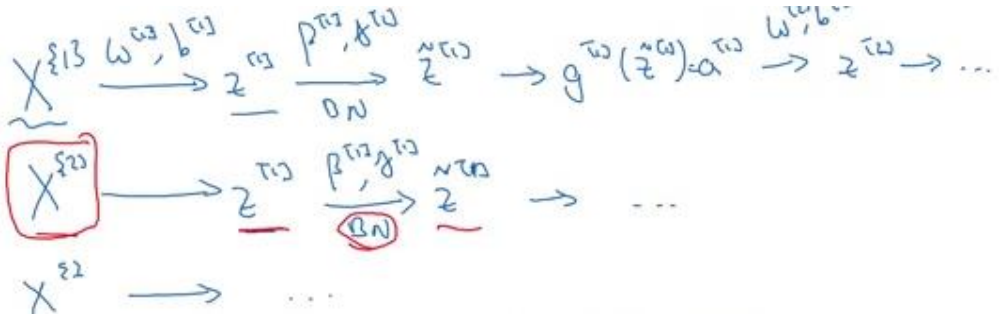
أيضا نقوم بعمل update لقيم بيتا بشكل مستمر عبر ضرب اشتقاقها في الفا و طرحها من الاصل



وكل هذه العمليات الرياضية لن نقوم بها بالفعل , إذ أن تينسورفلو يقوم بكل هذا في سطر واحد .

ولا تنس أن هذا سيطبق مع تكنيك mini-batches (وهو الذي يقوم بتقسيم العينات لاجزاء بحيث نقوم بالتدريب علي كل جزء علي حدة , ثم عمل updates للاوزان w , b)

و يكون بهذه الطريقة :



و هناك ملحوظة ذكية . .

في الأحوال العادية , قيمة z تكون حاصل ضرب w في a (او في x) مجموعة علي b

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

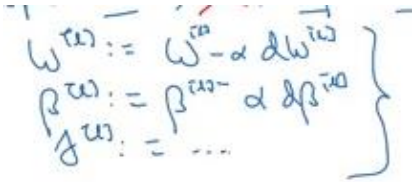
لكن حينما نقوم بعمل تسوية لقيمة z , فسيتم طرح اي قيمة لها من المتوسط , فنجد ان قيمة b سواء جمعناها او كانت بصفر , ستكون النتيجة واحدة , لانها رقم ثابت

فيمكن حينها الاستغناء عن كل قيم b (او جعلها صفر) وتكون المعادلة هي ان الـ z تساوي حاصل ضرب w في a (او في x) , ومن بعدها نأتي بـ znorm ثم $\tilde{z}$

ايضا لا تنس ان ابعاد مصفوفات كلا من b (لو تم استخدامها) و جاما و بيتا , هي نفس ابعاد مصفوفة الـ z او $\tilde{z}$, وهي (1 , n(L)) اي عدد صفوف يساوي عدد الخلايا الخفية في الطبقة , وعمود واحد .

و بالتالي الخطوات الكاملة تكون كالتالي :

- يتم تطبيق المسار الأمامي بداية من x للوصول لـ z
- يتم ايجاد $\tilde{z}$ من قيمة z
- يتم استخدام $\tilde{z}$ للوصول للطبقة التالية , حتي نصل الي المخرج output
- نقوم بعمل المسار الخلفي , وحساب قيم dw , dβ , dγ (تم تجاهل db لحذفها كما ذكرنا)
- يتم عمل تعديل لقيم w , β , γ , عبر القوانين :



- هذا اذا ما تم استخدام الـ gradient descent اما اذا تم استخدام تكنيكات اخري (momentum , RMS , adams) فيتم تطبيقه
- يتم تكرار الخطوات السابقة لكل جزء من اقسام الـ mini-batch

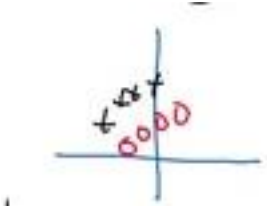
-

و هنا هنسأل سؤال , ما الذي بيجعل batch norm يعني تسوية بيانات الـ z الداخلية في الـ NN يكون لها تاثير ايجابي ؟

لو قلنا مثال , اني ساقوم ببناء خوارزم , للتفريق بين صور القطط و غير القطط, وقمت باعطائه كـ (training data) عدد من صور القطط (السوداء فقط) حينما تكون y تساوي 1 , وصور اخري حينما تكون y= 0

فإذا قمت بتجريبه علي الـ test data حينما تكون القطط ملونة , فنتوقع ان كفائتها ستكون قليلة , لانها لم تدخل في التدريب

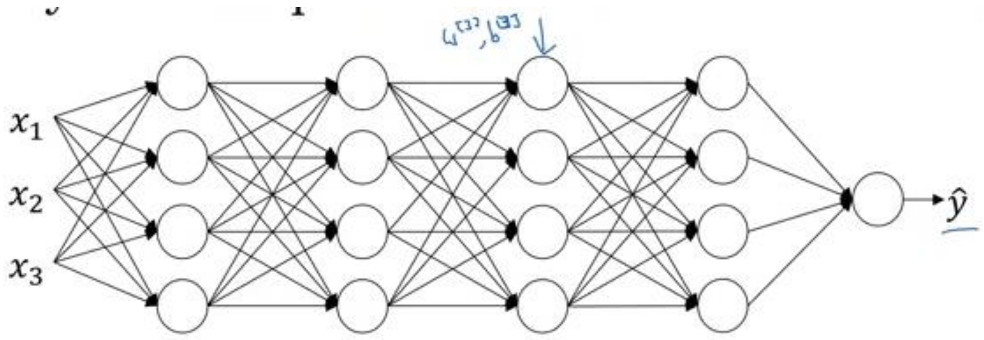
وكأن توزيع صور القطط و غيرها في اثناء التدريب كان هكذا :



بينما توزيع صور القطط و غيرها في التدريب كان هكذا :

والتأثير ده اسمه covariate shift واللي معناه : ان لو فيه شئ في البداية تم تغييره ,فلابد اني اقوم بعمل اعادة للتدريب مرة اخري , لان التغيير في البدايات , سيصاحبه تغيير حتمي في النهايات .

و لتطبيق هذه الفكرة علي الـ NN للوصول لأهمية عمل تسوية للبيانات , نتخيل ان لدينا شبكة عميقة هكذا . .



فنلاحظ أن الطبقة الرابعة ستتأثر حتما و بشكل كبير باي تغيير يحدث في المدخلات في الطبقة الأولي X , و هنا تأتي أهمية تسوية البيانات الداخلة , وكذلك تسوية قيم الـ z الداخلية . .

لان تسويتها و جعل متوسطها هو الصفر يسهل كثيرا علي الشبكة عملها , ويجعلها اقرب لتحقيق نتائج افضل .

-

كل ما تم ذكره عن تسوية قيم الـ z كان علي بيانات التدريب training data. .

فماذا عن بيانات الاختبار test data ؟

اثناء التدريب كنا نتعامل باسلوب الـ mini-batches (تقسيم العينة لاجزاء ومعالجة كل جزء بالترتيب) , لكن اثناء الاختبار سيكون كل جزء علي حدة .

وقتها نقوم بتحديد قيم μ (ميو) و σ (سيجما) التي تم حسابها من الاجزاء batches اثناء التدريب , واستخدامها لتحديد قيمة مناسبة لـ ميو و سيجما , التي سنستخدمها اثناء الاختبار .

و غالبا ما يتم اختيار تكنيك (exponentially weighted average) السالف ذكره في السابق .

اذن اثناء التدريب نحتفظ بقيم ميو و سيجما المستخدمة , ومنها يتم استنتاج ميو و سيجما مناسبة اثناء الاختبار .

-

سنتعرف علي ما يسمي softmax classification

وهو الخاص بالتصنيف بين اكثر من متغير , ففي الامثلة السابقة تحدثنا عن متغير واحد (الصورة قطة او ليست قطة) , حينما يتراوح المخرج بين 0 و 1

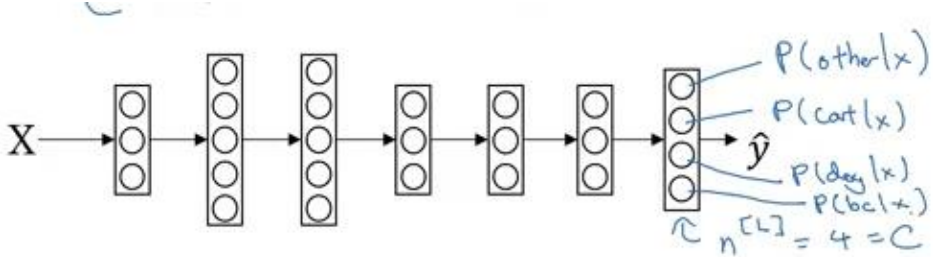
بينما هنا سنتناول لو كانت الصورة قطة , او كلب , او كتكوت , او صورة اخري .

و لا تنس انه من المنطقي دائما عمل فئة "أخري" او "غير ذلك" في التصنيف المتعدد , لأن المدخل قد لا يكون ايا من التصنيفات التي ذكرتها , وغالبا ما يأخذ هو رقم 0 بينما التصنيفات التي لديك تأخذ ارقام 1 , 2 ,3 . . .

ففي هذا المثال , سنقول ان الصورة قد تكون قطة (1) او كلب (2) او كتكوت (3), او اخري (0)

وأول ما يتم تحديده هو رقم C وهو عدد التصنيفات لديك , ولا تنس إضافة رقم 0 للاخري , فيكون هنا الـ C تساوي 4 , حيث هي (0,1,2,3)

سنقوم ببناء الشبكة , بحيث يكون لها مدخل واحد (الصورة) , واخر طبقة يكون فيها 4 وحدات

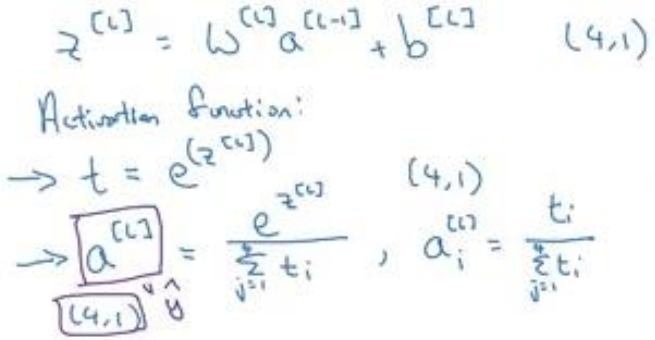


بحيث كل وحدة منهم تشير الي الاختيارات الاربعة (0 1 2 3) (اخري , قط , كلب , كتكوت)

و مع التصنيف المتعدد , يتم استخدام طريقة الـ softmax classification كدالة الـ activationبديلا عن السيجمويد او التانش .

وتكون عبارة عن , في آخر طبقة (الـ output) , يتم أولا حساب قيمة z من عبر المعادلة التقليدية (W a + b) حيث الـ a هي مخرج الطبقة السابقة, ة بدلا من استخدام السيجمويد او التانش , يتم احتساب ما يسمي الـ t و هي اكسبونينشيال e لقيمة الـ z . .

فحينما يكون لدي 4 قيم للـ z يتم احتساب 4 قيم للـ t كل واحدة هي اكسبونينشيال للـ z , ثم جمعهم معا , ثم قسمة كل t فيهم علي المجموع . .



وكأن كل مخرج من المخارج , هو اكسبونينشيال الـ z علي المجموع لهم , بالقانون :

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

ثم تكون الاحتمالية الأكبر , هي للرقم الأكبر فيهم , بالنسبة المئوية . .

فإذا كانت الـ z الخاصة بها هي :

$$z^{[L]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$$

فنقوم بحساب الـ t وهو اكسبونينشيال الارقام :

$$\mathrm{t} = \begin{bmatrix} e^{5} \\ e^{2} \\ e^{-1} \\ e^{3} \end{bmatrix}$$

والذي سيكون مجموعها : 176

و بقسمة كل t علي المجموع

$$g^{[L]}(z^{[L]}) = \begin{bmatrix} e^{5}/(e^{5}+e^{2}+e^{-1}+e^{3}) \\ e^{2}/(e^{5}+e^{2}+e^{-1}+e^{3}) \\ e^{-1}/(e^{5}+e^{2}+e^{-1}+e^{3}) \\ e^{3}/(e^{5}+e^{2}+e^{-1}+e^{3}) \end{bmatrix}$$

يكون الاربع مخارج هي :

$$\begin{bmatrix} 0.842 \\ 0.042 \\ 0.002 \\ 0.114 \end{bmatrix}$$

وبتفعيل النسبة المئوية , يظهر لنا ان نسبة ان تكون الصورة في التصنيف الأول (0 أي اخري) , هو 84% , ونسبة أن تكون في التصنيف الثاني (1 اي قطة) 4 % , وهكذا 0.2% و 11 %

و عبر هذه الارقام نتمكن من معرفة ناتج تقريبي للصورة .

ولا تنس ان هذه اول مرة نتناول اكثر من رقم و نخرج اكثر من رقم , في الامثلة السابقة كنا نتناول رقم واحد و ندخله في السيجمويد او التانش و نخرج منه رقم واحد اما صفر او 1 .

_

بداية لنفهم لم سميت softmax ؟

عكسها هو الـ hardmax وهو التصنيف العادي بين متغيرين , حينما يكون هناك مخرج اما 1 او 0 . فيكون الخيار "حاد" "hard" . .

اما هنا فالارقام تكون مختلفة , وليست 1 و 0 , لذا سميت softmax .

اذن كيف يتم حساب الـ cost function في حالة الـ softmax ؟

معادلة الخطأ بشكل عام تكون الفارق بين القيمة الحقيقية , والقيمة المتوقعة . .

ويكون لها القانون :

$$L(\hat{y},y) = - \Sigma y_j \log \hat{y}_j$$

ولكي نفهمها لابد من تناول مثال . .

فإذا قلنا ان الصورة هي لقطة , إذن ستكون قيمة الـ y (القيمة الحقيقية) مصفوفة كالتالي :

0
1
0

0

حيث الخانة الثانية بـ 1 و الباقي اصفار

اذا فرضنا ان الخوارزم جاء بقيمة جيدة هكذا

0.1
0.8
0.05
0.05

اذا قمنا بالتطبيق في القانون , نجد ان قيمة y في الخانة الاولي و الثالثة و الرابعة هي صفر , فقيمتها صفر , فتكون فقط في القيمة الثانية , وهي 1

نقوم بضربها في لوج القيمة الثانية و هي 0.8 , والتي تساوي 0.09- , وحينما نضربها في 1- تساوي 0.09 , وهي قيمة قليلة و جيدة , بسبب ان قيمة ŷ كانت عالية

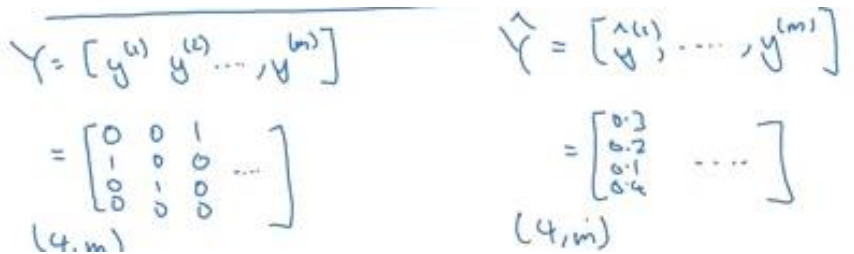
أما إن كانت قيمة ŷ قليلة , مثلا 0.2 , فنجد أن لوج قيمتها هي 0.69- وحينما نضربها في 1- ستساوي 0.69 , وهي قيمة كبيرة . .

و نقوم بجمع قيم الاخطاء لكل عناصر العينة لدي و قسمتها علي عدد العينة m لنأتي بالخطأ النهائي

$$J(w^{(i)}, b^{(i)}, \dots) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

ولا تنس أن ابعاد مصفوفة Y كابيتال , وهي كل قيم y لكل عناصر العينة , ستكون ابعادها (4xm)

وكذلك مصفوفة Ŷ نفس الابعاد



وأخيرا . .

لما نحسب المسار الخلفي ,و نحتاج لإيجاد قيمة dz تكون قيمتها ببساطة هي

$$dz = \hat{y} - y$$

-

و هنا سنتكلم عن أهمية لغة البرمجة و المكتبات في التعامل مع الـ DL

فالتعقيد البالغ في المعادلات السابقة , يحتم علينا ان نعتمد بشكل كبير علي المكتبات الجاهزة , والمصممة بكل تلك المعادلات . .

و من أشهر المكتبات في هذا المجال :

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

ومن المهم أن يتم انتقاء المكتبة المختار بعناية , حتي لا تضيع مجهود كبير في تعلم والتدريب علي شئ غير مفيد . .

و من العوامل التي يمكن اعتبارها في تقييم اي مكتبة سيتم استخدامها هي :

- مدي سهولة التعامل مع المكتبة
- مدي سرعتها مع البيانات الضخمة
- هل هي مفتوحة المصدر أم لا ,و هل ستستمر مفتوحة ام لا , لان عدد من المكتبات تكون مفتوحة لفترة , حتي يعتادها الناس ثم يجعلونا محدودة

-*

نهاية الاسبوع الثالث و الكورس الثاني