

NeuralMath

ایک عصبی نیٹ ورک کی ریاضیاتی
ساخت

اردو

پرتگالی اصل کا اردو ترجمہ

.Americo Cunha Jr

/neuralmath.org/ur

$$\hat{y} = \sigma(Wx + b)$$

اشاعتی معلومات

ایک عصبی نیٹ ورک کی ریاضیاتی ساخت
Americo Cunha Jr

Laboratório Nacional de Computação Científica (LNCC), Petrópolis, Brasil
Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, Brasil
ORCID: 0000-0002-8342-0363 · americocunha@lncc.br



neuralmath.org/ur/

حوالہ دینے کا طریقہ

Cunha Jr, A. (2026). A anatomia matemática de uma rede neural (Version 1.0). Zenodo.
<https://doi.org/10.5281/zenodo.23212381>

اصل مقالہ

Cunha Jr., A. A anatomia matemática de uma rede neural.
Professor de Matemática Online (PMO). Manuscrito
.submetido, 2026

لائسنس

Creative Commons Attribution 4.0 International (CC BY 4.0)
یہ متن لائسنس کے تحت تقسیم کیا گیا ہے۔ CC-BY-4.0

(10-2026) 1.0 · نسخہ: DOI: 10.5281/zenodo.23212381

ضمنی مواد

یہ ایڈیشن ویسپر
قابل اجرا نوٹ بک (Google Colab)
ماخذ کوڈ (GitHub)

کوئی غلطی ملی؟ اصلاحات اور تجاویز یہاں خوش آمدید ہیں:
<https://github.com/amicocunhajr/NeuralMath/issues>

دیباچہ

یہ متن ایک سادہ خیال سے پیدا ہوا: ایک چھوٹے عصبی نیٹ ورک کے اندر ہونے والی ریاضیاتی کارروائیوں کو نمایاں کرنا۔ بہت سے ابتدائی تعارفوں میں عصبی نیٹ ورک طاقور مگر غیر شفاف اوزار کے طور پر سامنے آتے ہیں۔ قاری کسی لائبریری کو استعمال کرنا، تربیت چلانا اور نقصان کی مچھنی دیکھنا سیکھتا ہے، مگر ضروری نہیں کہ وہ ان کارروائیوں کی پوری ترتیب دیکھے جو ڈیٹا کو فیصلے میں بدلنے ہے۔ یہاں ہم الٹا راستہ اختیار کرتے ہیں۔ ہم ایک بنیادی ہندسی مسئلہ سے آغاز کرتے ہیں، خطی درجہ بند کو مصنوعی نیورون کے طور پر سمجھتے ہیں، فعالی افعال متعارف کرتے ہیں، نیورمز کو ایک چھوٹے نیٹ ورک میں جوڑتے ہیں اور قدم بہ قدم گراڈیان، گراڈیان نزول اور بیک پروپاگیشن تک پہنچتے ہیں۔

یہ متن ان قارئین کے لیے لکھا گیا ہے جن کی کیکولس اور شماریات میں نسبتاً محدود تیاری ہے: طلبہ، اساتذہ اور خود سیکھنے والے۔ مقصد موضوع کو اتنا آسان کرنا نہیں کہ ریاضی غائب ہو جائے، بلکہ بیان کو اس طرح منظم کرنا ہے کہ برنی بات کسی مانوس تصور پر قائم ہو: مستوی میں نقطہ، سیدھی لکیر، افعال کی ترکیب، مشتق اور زنجیری قاعدہ۔ Python اور NumPy کا کوڈ اس تعمیر کا قدرتی نتیجہ بنتا ہے؛ مساوات اور الگورتھم ایک بھرپور شے کی دو وضاحتیں ہیں۔ اس لیے بہترین طریقہ یہ ہے کہ نوٹ بک ساتھ چلی ہو اور ہر مرحلے کو چلا کر تبدیل کیا جائے۔

یہ متن NeuralMath مجموعے کا حصہ ہے جو 26 زبانوں میں دستیاب ہے۔ ہر نسخہ وہی مساوات، اشکال، عددی مثالیں اور حصوں کی نمونگ برقرار رکھتا ہے تاکہ مختلف زبانوں کے قارئین مل کر مطالعہ کر سکیں۔ ریاضی اور میڈیونک سیکھنے میں ایک اہم رکاوٹ صرف تصویری نہیں بلکہ لسانی بھی ہے، اور ہر قاری اپنی زبان میں داخلے کے ایک راستے کا حق دار ہے۔

میرا مقصد یہ ہے کہ قاری سفر کے اختتام پر عصبی نیٹ ورک کو کم پراسرار انداز میں دیکھے۔ ہمیں یہ دکھانا کرنے کی ضرورت نہیں کہ جدید عصبی نظام سادہ ہیں۔ اتنا دکھانا کافی ہے کہ آج کی اپیلی کیشنز کے پیمانے اور پیچیدگی کے نیچے اب بھی ایسی ریاضیاتی تصورات کام کرتے ہیں جن کا معائنہ، اخذ، نفاذ اور مہم ممکن ہے۔

اس ایڈیشن کے بارے میں



Google Colab · UR

یہ NeuralMath مجموعے کا اردو ایڈیشن ہے، جو پرتگالی اصل کا ترجمہ ہے۔
FILL IN: how the translations were produced. پرتگالی
متن حوالہ جاتی نسخہ رہے گا؛ کسی اختلاف کی صورت میں وہی معتبر
ہوگا۔

اس ترجمے کا ابھی کسی مادری زبان بولنے والے نے جائزہ نہیں لیا۔
اصلاحات
پر <https://github.com/amicocunhajr/NeuralMath/issues>
خوش آمدید ہیں۔

نوٹ بک کو ساتھ کھول کر پڑھیں: متن کا ہر حصہ ایسے کوڈ سے مطابقت
رکھتا ہے جسے چلایا اور تبدیل کیا جا سکتا ہے۔ اس ایڈیشن کی نوٹ بک
Google Colab میں کھولنے کے لیے QR کوڈ اسکرین کریں۔ دوسری
25 زبانوں کے نسخے آخری صفحے پر درج ہیں۔

ایک عصبی نیٹ ورک کی ریاضیاتی ساخت

ib Americo Cunha Jr

خلاصہ

ہم ایک سادہ درجہ بندی کے مسئلے کے ذریعے مصنوعی اعصابی نیٹ ورکس کا ریاضیاتی تعارف پیش کرتے ہیں۔ ایک خطی درجہ بند فطری طور پر مصنوعی نیورون کے تصور تک لے جاتا ہے؛ اس کے بعد ہم اوزان، بائس، فعال سازی فنکشنز، اور نیورونز کو ایک چھوٹے نیٹ ورک میں جوڑنے کے طریقے پر بحث کرتے ہیں۔ تربیت کو لاس فنکشنز، مشتقات، گریڈینٹ ڈیسنٹ اور بیک پروپیگیشن کے ذریعے متعارف کرایا جاتا ہے۔ آخر میں ماڈل کو براہ راست Python میں نافذ کیا جاتا ہے، تاکہ مساوات اور الگورتھم کے درمیان تعلق واضح رہے۔ مقصد یہ دکھانا ہے کہ تحلیلی ہندسہ، خطی الجبرا اور حسابان کے مانوس تصورات اعصابی نیٹ ورکس کے بنیادی طریقہ کار کو سمجھنے کے لیے ایک قابل رسائی بنیاد فراہم کرتے ہیں۔

کلیدی الفاظ: مصنوعی اعصابی نیٹ ورکس؛ مشین لرننگ؛ ریاضی کی تعلیم؛ بیک پروپیگیشن؛ Python۔

1. کمپیوٹر کب سیکیسکتا ہے؟

روزمرہ زندگی میں ہم تقریباً غیر محسوس طور پر درجہ بندی کے فیصلے کرتے رہتے ہیں۔ کسی پھل کو دیکھ کر ہم فوراً پہچان لیتے ہیں کہ وہ سیب ہے یا سترہ؛ فون پر پیغام آنے پر ہم حقیقی گفتگو کو ناپسندیدہ اشتہار سے الگ کرتے ہیں؛ طبی تجزیے میں کئی پیمائشوں کو ملا کر کسی فیصلے کی مدد کی جا سکتی ہے؛ اور مالیاتی اداروں میں قرض دینے سے پہلے متعدد معلومات کا جائزہ لیا جاتا ہے۔ ان سب مثالوں میں جواب پیدا کرنے کے لیے کئی خصوصیات کو ایک ساتھ دیکھا جاتا ہے۔

بہت سے روایتی کمپیوٹر پروگراموں میں وہ قواعد جن سے جواب پیدا ہوتا ہے، پروگرامر واضح طور پر لکھتا ہے۔ جب قواعد معلوم ہوں اور انہیں صاف انداز میں بیان کیا جا سکے تو یہ طریقہ اچھا کام کرتا ہے۔ لیکن جب ڈیٹا بہت متغیر ہو، یا اچھے فیصلے کے لیے درکار تمام شرائط پہلے سے لکھنا دشوار ہو، تو یہ طریقہ کم مؤثر ہو جاتا ہے۔ اس سے ایک فطری سوال پیدا ہوتا ہے:

کیا ہم ایسا پروگرام بنا سکتے ہیں جو فیصلہ کرنے کا قاعدہ پہلے سے حاصل کرنے کے بجائے مثالوں سے سیکھے؟

یہ مشین لرننگ کے مرکزی تصورات میں سے ایک ہے، جو مصنوعی ذہانت کا وہ شعبہ ہے جس میں ڈیٹا سے ماڈل موزوں کرنے کے طریقوں کا مطالعہ کیا جاتا ہے [2, 3]۔ زیر نگرانی سیکھنے کے مسئلے میں الگورتھم کو درست جوابات کے ساتھ مثالیں دی جاتی ہیں، اور ہم ایسا ماڈل تلاش کرتے ہیں جو تربیت میں استعمال نہ ہونے والے معاملات کے لیے بھی مفید پیش گوئیاں کر سکے۔ اس مقصد کے لیے بنائے گئے بہت سے ماڈلز میں مصنوعی اعصابی نیٹ ورکس تصویر شناسی، زبان کی پروسیسنگ، سگنل تجزیے اور زمانی سلسلوں کی پیش گوئی جیسی تطبیقات میں اہم ہو چکے ہیں [6]۔

اعصابی نیٹ ورکس کو ریاضی، کمپیوٹنگ اور اسکولی تعلیم کے درمیان پل کے طور پر استعمال کرنے پر Batista اور Martins نے Professor de Matemática Online میں پہلے ہی بحث کی ہے [1]؛ انہوں نے اساتذہ کی تربیت، Python اور ہندسوں کی شناخت پر مشتمل ایک سرگرمی پیش کی۔ موجودہ مضمون ایک تکمیلی راستہ اختیار کرتا ہے۔ ہمارا بنیادی مقصد ایک چھوٹے اعصابی نیٹ ورک کی اندرونی ریاضی کو واضح کرنا ہے؛ تحلیلی ہندسہ سے آغاز کرتے ہوئے قدم بہ قدم گریڈینٹ کے حساب اور بیک پروپیگیشن کی دستی تنفیذ تک پہنچنا۔

جدید تطبیقات کی پیچیدگی کے باوجود، اعصابی نیٹ ورک کا ریاضیاتی جوہر نسبتاً مانوس تصورات سے متعارف کرایا جا سکتا ہے۔ بنیادی سطح پر اعصابی نیٹ ورک فنکشنز کا ایک مرکب ہے جس کے پیرامیٹرز ڈیٹا سے ایڈجسٹ کیے جاتے ہیں۔ ہم ایک

سادہ ہندسی مسئلے سے شروع کرتے ہیں، جدا کرنے والی لکیر کو مصنوعی نیورون کے طور پر دوبارہ تعبیر کرتے ہیں، کئی نیورونز کو جوڑتے ہیں، فعال سازی فنکشنز کے کردار کا مطالعہ کرتے ہیں، اور دکھاتے ہیں کہ مشتقات نیٹ ورک کے پیرامیٹرز کو خودکار طور پر ایڈجسٹ کرنے کے لیے کیسے استعمال ہو سکتے ہیں۔ آخر میں ہم Python میں ایک مکمل مگر چھوٹی مثال نافذ کرتے ہیں، جس میں مساوات اور کوڈ کے درمیان براہ راست مطابقت برقرار رہتی ہے۔

2. مسئلے کی ریاضیاتی نمائندگی

اعصابی نیٹ ورک کے کام کرنے کے طریقے کو سمجھنے کے لیے ہم جان بوجھ کر ایک سادہ مسئلے سے آغاز کرتے ہیں: سیب اور سنترے میں فرق کرنا۔ کسی پھل کی بہت سی خصوصیات اس کام کے لیے مفید ہو سکتی ہیں، مثلاً رنگ، کمیت، قطر، بناوٹ اور خوشبو۔ تاہم ابتدائی ریاضیاتی ماڈل میں یہ تمام پیچیدگی شامل کرنا ضروری نہیں۔ ہم صرف دو ایسی مقداریں استعمال کریں گے جنہیں آسانی سے ناپا جا سکتا ہے: کمیت، گرام میں، اور قطر، سینٹی میٹر میں۔

لہذا ہر پھل کو ایک مرتب جوڑے سے ظاہر کیا جائے گا

$$(x_1, x_2),$$

جہاں

$$x_1 = \text{کمیت} \quad \text{اور} \quad x_2 = \text{قطر}.$$

پس جوڑا (155, 7.2) ایک ایسے پھل کی نمائندگی کرتا ہے جس کی کمیت 155 g اور قطر 7.2 cm ہے۔ طبعی شے کی جگہ دو عددی پیمائشیں رکھ کر ہم ہر پھل کو کارتیسی سطح پر ایک نقطے سے وابستہ کرتے ہیں: پہلی محدود کمیت اور دوسری قطر بتاتی ہے۔ شکل 1 فرضی سیبوں اور سنتروں کے ایک مجموعے کے لیے اس نمائندگی کو دکھاتی ہے۔

یہ تبدیلی مسئلے کی ظاہری نوعیت بدل دیتی ہے۔ سیب یا سنترے کو براہ راست پہچاننے کا سوال پوچھنے کے بجائے ہم ایک ہندسی سوال پوچھ سکتے ہیں: دونوں جماعتوں سے تعلق رکھنے والے نقاط کو سطح میں کیسے جدا کیا جا سکتا ہے؟ زاویہ نظر کی یہ تبدیلی مشین لرننگ میں بنیادی اہمیت رکھتی ہے۔ تصاویر، متون، آوازیں اور تجرباتی سگنلز کسی الگورتھم کے ذریعے پروسیس ہونے سے پہلے عددی نمائندگیوں میں تبدیل کیے جانے چاہییں۔ ہماری مثال میں ہر شے کی صرف دو محدود ہیں: حقیقی تطبیقات میں ایک نمائندگی سینکڑوں، ہزاروں، بلکہ لاکھوں اجزا پر مشتمل ہو سکتی ہے۔ اصول وہی رہتا ہے: ہر مثال کو اعداد کے ایک مجموعے سے وابستہ کیا جاتا ہے، اور ماڈل ایسا قاعدہ تلاش کرتا ہے جو اس نمائندگی کو مطلوبہ جواب سے ملائے۔

ہم سب سے سادہ دستیاب ہندسی قاعدے سے آغاز کرتے ہیں اور پوچھتے ہیں کہ کیا ایک واحد سیدھی لکیر سیب کے نقاط کو سنترے کے نقاط سے مناسب طور پر جدا کر سکتی ہے۔

3. سیدھی لکیر بطور درجہ بند

جب ہر پھل کو کارتیسی سطح پر ایک نقطے سے ظاہر کر دیا جائے تو درجہ بندی ایک ہندسی مسئلہ بن جاتی ہے: کیا ایک سیدھی لکیر سیبوں کو سنتروں سے جدا کر سکتی ہے؟ یہ سوال مشین لرننگ کے ایک اہم خاندان، جسے خطی درجہ بند کہا جاتا ہے، کے بنیادی تصور کو بیان کرتا ہے۔

ڈیٹا کے بادل کو کاٹتی ہوئی ایک لکیر پر غور کریں، جیسا کہ شکل 2 میں دکھایا گیا ہے۔ یہ لکیر سطح کو دو نصف سطحوں میں تقسیم کرتی ہے۔ اگر زیادہ تر سیب ایک طرف اور زیادہ تر سنترے دوسری طرف ہوں، تو کسی نئے پھل کی لکیر کے نسبت پوزیشن کو درجہ بندی کے قاعدے کے طور پر استعمال کیا جا سکتا ہے۔ کمیت اور قطر ناپنے کے بعد ہم پھل کو ایک نقطے سے ظاہر کرتے ہیں اور طے کرتے ہیں کہ یہ نقطہ لکیر کے کس طرف ہے۔

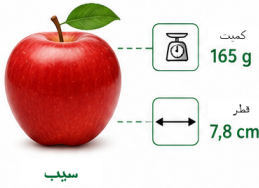
کارتیسی سطح (x_1, x_2) میں ایک لکیر کو یوں بیان کیا جا سکتا ہے

$$w_1 x_1 + w_2 x_2 + b = 0, \quad (w_1, w_2) \neq (0, 0).$$

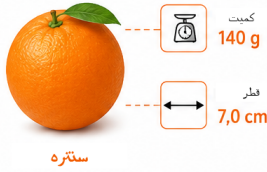
ویکٹر (w_1, w_2) لکیر پر عمود ہے، اس لیے اس کی سمت متعین کرتا ہے، جبکہ پیرامیٹر b اس کی جگہ میں حصہ ڈالتا ہے۔ اس نمائندگی میں ایک معمولی زائد پن ہے: w_1 اور w_2 کو ایک ہی غیر صفر مستقل سے ضرب دینے پر بالکل وہی لکیر حاصل

1. طبیعی اشیا

حقیقی پھل



سیب



سنترہ

2. عددی ڈیٹا

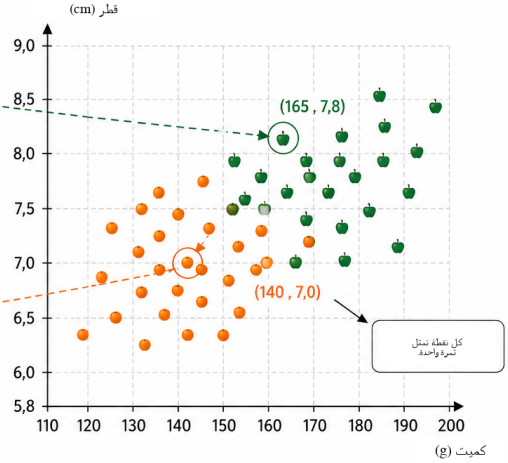
(کمیت، قطر)

(165, 7.8)
(x_1, x_2)

(140, 7.0)
(x_1, x_2)

3. ریاضیاتی نمائندگی

کارٹیزی سطح پر نقاط



سیب

سنترہ

شکل 1: طبعی مشاہدے سے ریاضیاتی ماڈلنگ تک۔ ہر پہل کو پہلے دو قابل پی مائش مقداروں --- کمیت اور قطر --- سے بی ان کی اجاتا ہے، پھر مرتب جوڑے (x_1, x_2) سے ظاہر کی اجاتا ہے۔ کمیت---قطر کی سطح میں ہر مشاہدہ ایک نقطہ بن جاتا ہے، اس لی سے سیب اور سنترے میں فرق کرنے کے مسئلے کو دو جماعتوں کو جدا کرنے کے ہندسی مسئلے کے طور پر دوبارہ تعبیر کی اجاتا سکتا ہے۔

ہوتی ہے۔ چنانچہ حد کو بیان کرنے والے پیرامیٹرز یکتا نہیں۔ ان کی عددی تعبیر متغیرات کے پیمانے پر بھی منحصر ہے۔ چونکہ کمیت گرام اور قطر سینٹی میٹر میں ناپے جاتے ہیں، اس لیے مختلف اکائیوں کو مدنظر رکھے بغیر w_1 اور w_2 کی عددی قدروں کا براہ راست موازنہ نہیں کرنا چاہیے۔

لکیر کی مساوات خود یہ معلوم کرنے کا سادہ طریقہ دیتی ہے کہ نقطہ حد کے کس طرف ہے۔ اسکو کو یوں تعریف کریں

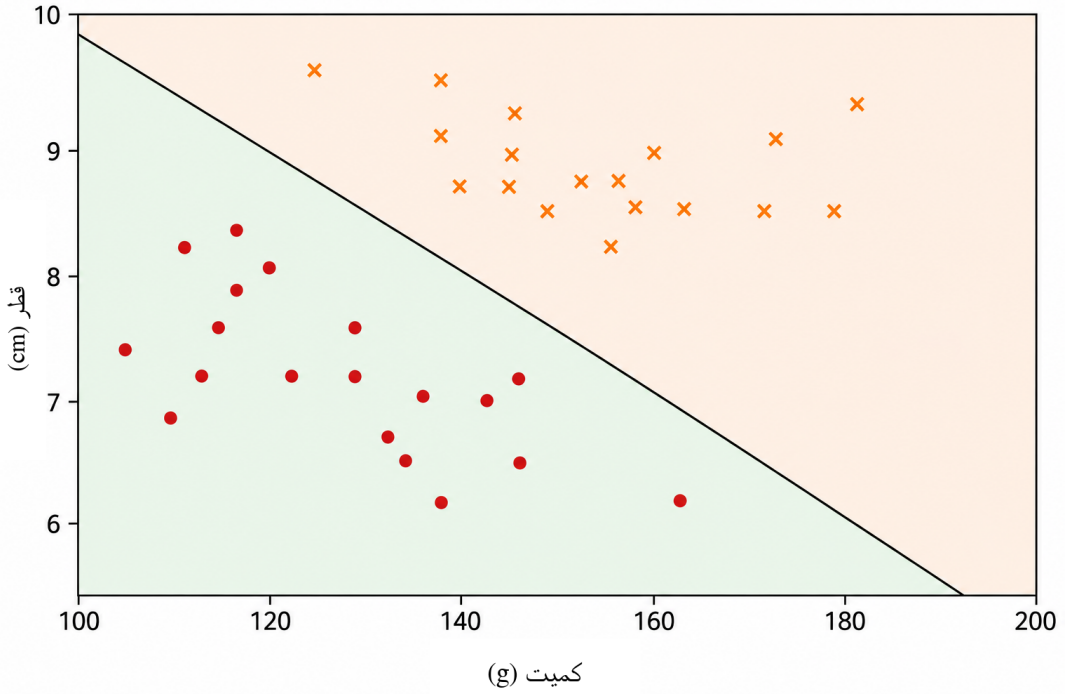
$$s(x_1, x_2) = w_1 x_1 + w_2 x_2 + b.$$

اگر $s(x_1, x_2) = 0$ ہو تو نقطہ لکیر پر ہے۔ اگر $s(x_1, x_2) > 0$ ہو تو وہ ایک نصف سطح میں، اور اگر $s(x_1, x_2) < 0$ ہو تو دوسری نصف سطح میں ہے۔ اگر ہم طے کریں کہ مثبت اسکو سیب اور منفی اسکو سنترے کے مطابق ہیں، تو قاعدہ بنتا ہے

$$\begin{cases} s(x_1, x_2) \geq 0 & \Rightarrow \text{سیب}, \\ s(x_1, x_2) < 0 & \Rightarrow \text{سنترہ}. \end{cases}$$

مثال کے طور پر اس حد پر غور کریں

$$w_1 = \frac{1}{10}, \quad w_2 = 2, \quad b = -30.$$



شکل 2: کمیت-قطر کی سطح میں خطی درجہ بند کی مثال۔ سی دھمی لکیری فیصلہ حد کے طور پر کام کرتی ہے: ایک نصف سطح کے نقاط کو ایک جماعت اور مخالف نصف سطح کے نقاط کو دوسری جماعت دی جاتی ہے۔ لہذا فیصلہ صرف اس بات پر منحصر ہے کہ نقطہ لکیری کے کس طرف واقع ہے۔

160 g کمیت اور 7.8 cm قطر والے پھل کے لیے

$$s = \frac{1}{10} \times 160 + 2 \times 7.8 - 30$$

$$= 16 + 15.6 - 30$$

$$= 1.6.$$

چونکہ اسکور مثبت ہے، نقطہ اس نصف سطح میں واقع ہے جسے ہم نے سیبوں سے وابستہ کیا ہے۔ لہذا یہ درجہ بند اس پھل کو سیب قرار دے گا۔

w_1 , w_2 اور b کا ہر انتخاب اچھی درجہ بندی نہیں دیتا۔ کچھ لکیریں دونوں گروہوں کو خراب انداز میں جدا کرتی ہیں، جبکہ کچھ کم غلطیاں کرتی ہیں۔ سیکھنے کا مسئلہ یہی ہے کہ پہلے سے لیبل شدہ مثالوں کو استعمال کر کے ایسے پیرامیٹرز تلاش کیے جائیں جو ماڈل کی خطا کے کسی پیمانے کو کم کریں۔ لکیر کی جگہ براہ راست پروگرام کرنے کے بجائے ہم چاہتے ہیں کہ الگورتھم ڈیٹا سے اس کے معاملات ایڈجسٹ کرے۔

اگر جماعتیں تقریباً خطی طور پر جدا ہونے کے قابل ہوں تو ایک لکیر اچھی کارکردگی دکھا سکتی ہے۔ زیادہ پیچیدہ ڈیٹا سیٹس کے لیے غیر خطی حد درکار ہو سکتی ہے۔ اس محدودیت پر بات کرنے سے پہلے ہم ابھی بنائے گئے درجہ بند کو ایک بنیادی حسابی اکائی، یعنی مصنوعی نیورون، کے طور پر دوبارہ تعبیر کرتے ہیں۔

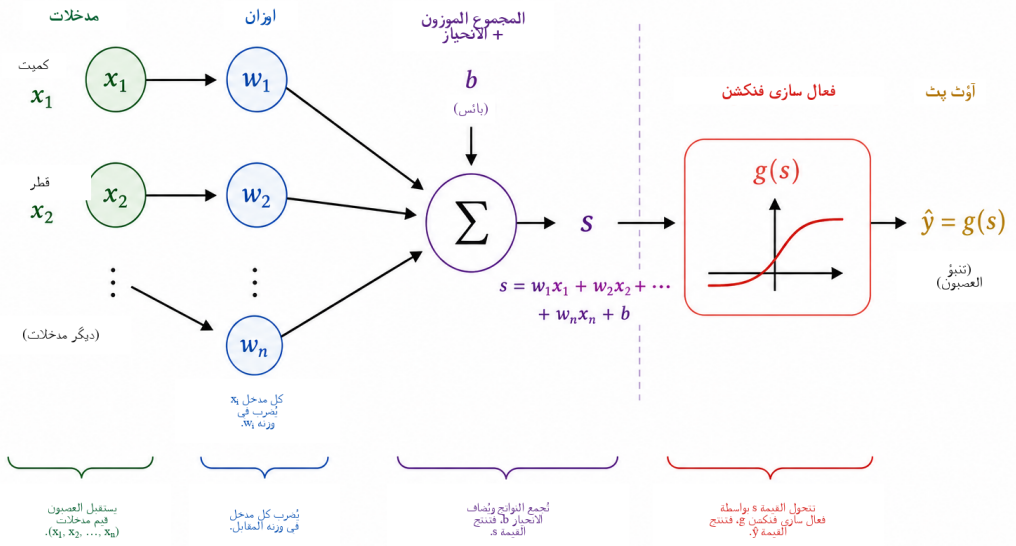
4. مصنوعی نیورون

پچھلے حصے میں اظہار

$$s = w_1x_1 + w_2x_2 + b$$

اس بات کا تعین کرنے کے لیے استعمال ہوا کہ کوئی پھل لکیر کے کس طرف واقع ہے۔ اسی اظہار کو مصنوعی نیورون کے حساب کے پہلے مرحلے کے طور پر بھی دیکھا جا سکتا ہے۔ ایک نیورون عددی مدخلات وصول کرتا ہے، انہیں اوزان اور بائس کی مدد سے جوڑتا ہے، پھر اپنا آؤٹ پٹ حاصل کرنے کے لیے ایک ریاضیاتی فنکشن لاگو کرتا ہے۔

ہماری مثال میں مدخلات کمیت x_1 اور قطر x_2 ہیں۔ ہر مدخل کو ایک پیرامیٹر سے ضرب دی جاتی ہے جسے وزن کہتے ہیں؛ حاصل ضربوں کو ایک مستقل جزو b کے ساتھ جمع کیا جاتا ہے، جسے بائس کہتے ہیں۔ شکل 3 اس حساب کو خاکے کی صورت میں دکھاتی ہے۔



نیورون انجام دیتا ہے: خطی امتزاج (وزنی مجموعہ + بائس)، اس کے بعد غیر خطیت (فعال سازی فنکشن)۔

شکل 3: مصنوعی نیورون کے کام کا خاکہ۔ ہر مدخل x_i کو وزن w_i سے ضرب دی جاتی ہے؛ حاصل ضربوں کو بائس b کے ساتھ جمع کر کے اس کو s حاصل کیا جاتا ہے۔ پھر فعال سازی فنکشن g اس کو $\hat{y} = g(s)$ میں تبدیل کرتا ہے، جو درمیانی مفیداری آخری پیش گوئی کے طور پر استعمال ہو سکتا ہے۔

اوزان طے کرتے ہیں کہ مدخلات میں تبدیلی اس کو کیسے متاثر کرتی ہے۔ اگر x_2 اور باقی تمام مقادیر مستقل رکھی جائیں تو Δx_1 کی تبدیلی s کو $w_1 \Delta x_1$ سے بدلتی ہے۔ اسی طرح Δx_2 کی تبدیلی اس کو $w_2 \Delta x_2$ سے بدلتی ہے۔ وزن کا نشان بھی اہم ہے: مثبت وزن کے ساتھ مدخل میں اضافہ اس کو بڑھاتا ہے، جبکہ منفی وزن اس کے برعکس اثر پیدا کرتا ہے۔ جب مختلف متغیرات کی اکائیاں اور پیمانے مختلف ہوں تو اس تعبیر کو احتیاط سے استعمال کرنا چاہیے۔ اسی لیے عملی تنفیذات میں تربیت سے پہلے مدخلاتی ڈیٹا کو اکثر نارملائز یا اسٹینڈرائز کیا جاتا ہے۔

بائس b افائن تبدیلی کا مستقل جزو ہے۔ ہندسی طور پر یہ فیصلہ حد کو منتقل ہونے دیتا ہے۔ اگر $b = 0$ ہو تو مساوات

$$w_1x_1 + w_2x_2 = 0$$

لکیر کو مبدا سے گزرنے پر مجبور کرتی ہے۔ بائس کی موجودگی میں حد دوسری جگہوں پر منتقل ہو سکتی ہے۔ مساوی طور پر، جب $x_1 = x_2 = 0$ ہو تو اس کو صرف $s = b$ ہوتا ہے۔

ضروری نہیں کہ اسکور ہی نیورون کا آخری آؤٹ پٹ ہو۔ عام طور پر اس پر ایک فعال سازی فنکشن g لاگو کیا جاتا ہے:

$$\hat{y} = g(s) = g(w_1x_1 + w_2x_2 + b).$$

یوں حساب کے دو مراحل ہیں،

$$s = w_1x_1 + w_2x_2 + b,$$

$$\hat{y} = g(s).$$

پچھلے حصے کے پیرامیٹرز استعمال کرنے پر $s = 1.6$ ملتا ہے۔ اگر ہم اسٹیپ فنکشن منتخب کریں

$$g(s) = \begin{cases} 1, & s \geq 0, \\ 0, & s < 0, \end{cases}$$

تو $\hat{y} = g(1.6) = 1$ ۔ ہماری اختیار کردہ روایت کے تحت پہلے کو سیب درجہ بند کیا جاتا ہے۔ اس فعال سازی کے ساتھ فیصلہ اس وقت بدلتا ہے جب s صفر کو عبور کرتا ہے، اس لیے جماعتی حد بالکل

$$w_1x_1 + w_2x_2 + b = 0$$

ہے۔

پس ایک واحد نیورون قابل ایڈجسٹ پیرامیٹرز رکھنے والی ایک سادہ ریاضیاتی شے ہے۔ اس کی حقیقی اہمیت تب سامنے آتی ہے جب ایسی بہت سی اکائیوں کو باہم جوڑا جائے۔ تاہم نیٹ ورک بنانے سے پہلے یہ سمجھنا ضروری ہے کہ فعال سازی فنکشن کیوں بنیادی اہمیت رکھتا ہے۔

5. فعال سازی فنکشن کی ضرورت کیوں ہے؟

ایک نیورون پہلے افائن تبدیلی کے ذریعے اسکور حساب کرتا ہے اور پھر فعال سازی فنکشن لاگو کرتا ہے۔ اگر اعصابی نیٹ ورک کو پیچیدہ تعلقات کی نمائندگی کرنی ہو تو یہ دوسرا مرحلہ لازمی ہے۔ وجہ سمجھنے کے لیے ایسے نیٹ ورک کا تصور کریں جس میں کوئی فعال سازی فنکشن استعمال نہ ہو۔ تب ہر اکائی کی صورت یہ ہوگی

$$y = w_1x_1 + w_2x_2 + b.$$

ایک بعدی مثال کے لیے فرض کریں کہ پہلی اکائی

$$h = ax + b$$

پیدا کرتی ہے، اور دوسری

$$y = ch + d$$

حساب کرتی ہے۔ بدلنے پر

$$y = c(ax + b) + d = acx + (cb + d),$$

جو اب بھی x کا افائن فنکشن ہے۔ یہی بات کئی ابعاد میں بھی درست ہے: افائن تبدیلیوں کا مرکب بھی افائن ہی رہتا ہے۔ لہذا غیر خطی عمل کے بغیر کئی تہیں شامل کرنے سے نیٹ ورک بنیادی طور پر زیادہ اظہاری نہیں ہو جاتا؛ پوری ترکیب کو ایک واحد افائن تبدیلی سے بدلا جا سکتا ہے۔

فعال سازی فنکشن اس محدودیت کو توڑتا ہے۔ اسکور s حساب کرنے کے بعد نیورون عموماً ایک غیر خطی نقشہ لاگو کرتا ہے

$$h = g(s).$$

جب ایسی کئی اکائیاں جوڑی جاتی ہیں تو حاصل شدہ ترکیب کو ایک واحد افائن تبدیلی میں سمیٹا نہیں جا سکتا۔ یوں نیٹ ورک زیادہ پیچیدہ فیصلہ حدود بنا سکتا ہے۔

سب سے سادہ فعال سازی اسٹیپ فنکشن ہے،

$$g(s) = \begin{cases} 1, & s \geq 0, \\ 0, & s < 0. \end{cases}$$

یہ دوئی فیصلہ پیدا کرتا ہے اور درجہ بندی کو سمجھنے کے لیے بہت مفید ہے۔ تاہم اس کا مشتق $s \neq 0$ کے لیے صفر اور $s = 0$ پر غیر معین ہے۔ چونکہ گریڈینٹ پر مبنی تربیت مشتقات سے یہ طے کرتی ہے کہ پیرامیٹرز کیسے بدلیں، اس لیے اسٹیپ فنکشن اس تربیتی طریقے کے لیے موزوں نہیں۔

ایک متبادل سگموئنڈ فنکشن ہے،

$$\sigma(s) = \frac{1}{1 + e^{-s}}.$$

یہ ہر حقیقی عدد کو 0 اور 1 کے درمیان ایک قدر میں نقش کرتا ہے۔ بہت منفی s کے لیے آؤٹ پٹ 0 کے قریب جاتا ہے، بہت مثبت s کے لیے 1 کے قریب، اور ان دونوں انتہاؤں کے درمیان منتقلی ہموار ہوتی ہے۔ سگموئنڈ ہر جگہ قابل اشتقاق ہے، اور

$$\sigma'(s) = \sigma(s)(1 - \sigma(s)).$$

چونکہ اس کا آؤٹ پٹ 0 اور 1 کے درمیان رہتا ہے، اسے دوئی درجہ بند کی آؤٹ پٹ تہ میں اکثر استعمال کیا جاتا ہے۔ مناسب لاس فنکشن کے ساتھ احتمالی تشکیل میں آؤٹ پٹ کو اس جماعت کے احتمال کے تخمینے کے طور پر سمجھا جا سکتا ہے جو 1 سے ظاہر ہوتی ہے۔ تاہم پوشیدہ تہوں میں سگموئنڈ ایک مشکل پیدا کر سکتا ہے: بڑی قدر والے مدخلات کے لیے اس کا مشتق بہت چھوٹا ہو جاتا ہے، جس سے گہرے نیٹ ورکس میں vanishing-gradient مسئلہ پیدا ہو سکتا ہے۔

پوشیدہ تہوں کے لیے وسیع طور پر استعمال ہونے والا فعال سازی فنکشن (ReLU) (Rectified Linear Unit) ہے،

$$\text{ReLU}(s) = \max(0, s) = \begin{cases} 0, & s < 0, \\ s, & s \geq 0. \end{cases}$$

اگرچہ یہ دو خطی حصوں سے بنا ہے، ReLU مجموعی طور پر خطی نہیں۔ اس کا مشتق $s < 0$ کے لیے 0 اور $s > 0$ کے لیے 1 ہے۔ $s = 0$ پر معمول کا مشتق موجود نہیں، اور سافٹ ویئر تنفیذات گریڈینٹ کے لیے ایک طے شدہ اصطلاح اختیار کرتی ہیں۔ مثبت مدخلات کے لیے s بڑھنے پر مشتق سکڑتا نہیں، اس لیے سگموئنڈ میں پائے جانے والے تشعب کے ایک طریقہ کار سے بچا جاتا ہے۔

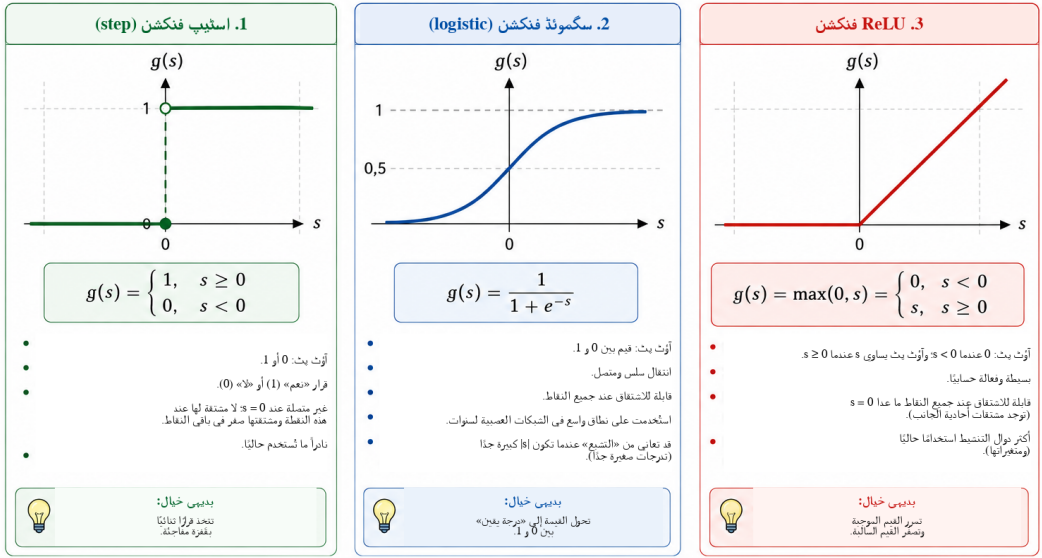
شکل 4 ان تین فعال سازی فنکشنز کا بصری موازنہ کرتی ہے: اسٹیپ فنکشن کی اچانک چھلانگ، سگموئنڈ کی ہموار منتقلی، اور ReLU میں ڈھلوان کی تبدیلی۔

ان کی بنیادی خصوصیات ذیل میں خلاصہ کی گئی ہیں۔

عام استعمال	مشتق	آؤٹ پٹ	فنکشن
تاریخی ماڈلز اور تدریسی مثالیں	$s \neq 0$ کے لیے صفر؛ $s = 0$ پر غیر معین	$\{0, 1\}$	Step
دوئی درجہ بندی میں آؤٹ پٹ تہ	s کے لیے موجود	$(0, 1)$	Sigmoid
پوشیدہ تہیں	$s < 0$ کے لیے 0 اور $s > 0$ کے لیے 1	$[0, \infty)$	ReLU

ان فنکشنز کی تفصیلات مختلف ہیں، مگر ان کا مشترک تصویری کردار غیر خطیت متعارف کرانا ہے۔ یہی غیر خطیت اگلے حصے کے چھوٹے نیٹ ورک کو ایسے تعلقات کی نمائندگی کرنے دیتی ہے جنہیں ایک سیدھی لکیر بیان نہیں کر سکتی۔

مزید مطالعہ کے خواہش مند قارئین کے لیے Haykin [5] اعصابی نیٹ ورکس کا کلاسیکی بیان پیش کرتے ہیں، جبکہ Goodfellow، Bengio اور Courville [3] وسیع تر تناظر میں فعال سازی فنکشنز اور گہرے نیٹ ورکس کی تربیت پر گفتگو کرتے ہیں۔ Nielsen کا آزادانہ دستیاب متن [7] نیورونز، فعال سازیاں اور سیکھنے کے بارے میں بدیہی فہم پیدا کرنے کے لیے خاص طور پر قابل رسائی ہے۔



خلاصة: تدخل فعال ساري فممكن الانطلاق إلى العصبون.
من دونها (أو إذا كانت جميعها خطية)، تنصرف الشبكة كلها لتمييز خطي واحد (خط مستقيم).

شکل 4: اعصابی نیٹ ورکس کی ابتدائی بحث میں عام طور پر استعمال ہونے والے تین فعال سازی فنکشنز کا موازنہ۔ اسٹیپ فنکشن اچانک فیصلہ کرتا ہے؛ سگموئڈ 0 اور 1 کے درمیان ہموار منتقلی فراہم کرتا ہے؛ اور ReLU منفی مدخلات کو دباتا جبکہ مثبت مدخلات کو برقرار رکھتا ہے۔ غیر خطی فعال سازی اس تہوں کے ڈھیری کو ایک واحد افائن تبدیلی میں سمیٹنے سے روکتی ہے۔

6. ایک چھوٹا اعصابی نیٹ ورک بنانا

اب ہم ایسے نیٹ ورک پر غور کرتے ہیں جس میں دو مدخلات، تین نیورونز پر مشتمل ایک پوشیدہ تہ، اور ایک آؤٹ پٹ نیورون ہے۔ مدخلات بدستور

$$x_1 = \text{کمیت}, \quad x_2 = \text{قطر},$$

ہیں، اور شکل 5 کے مطابق دونوں مدخلات ہر پوشیدہ نیورون کو دیے جاتے ہیں۔

پوشیدہ تہ کے آؤٹ پٹس کو h_1 ، h_2 اور h_3 کہیں۔ ہر نیورون کو یہی دو مدخلات ملتے ہیں لیکن اس کے اپنے اوزان اور بائس ہوتے ہیں:

$$h_1 = g(w_{11}x_1 + w_{12}x_2 + b_1),$$

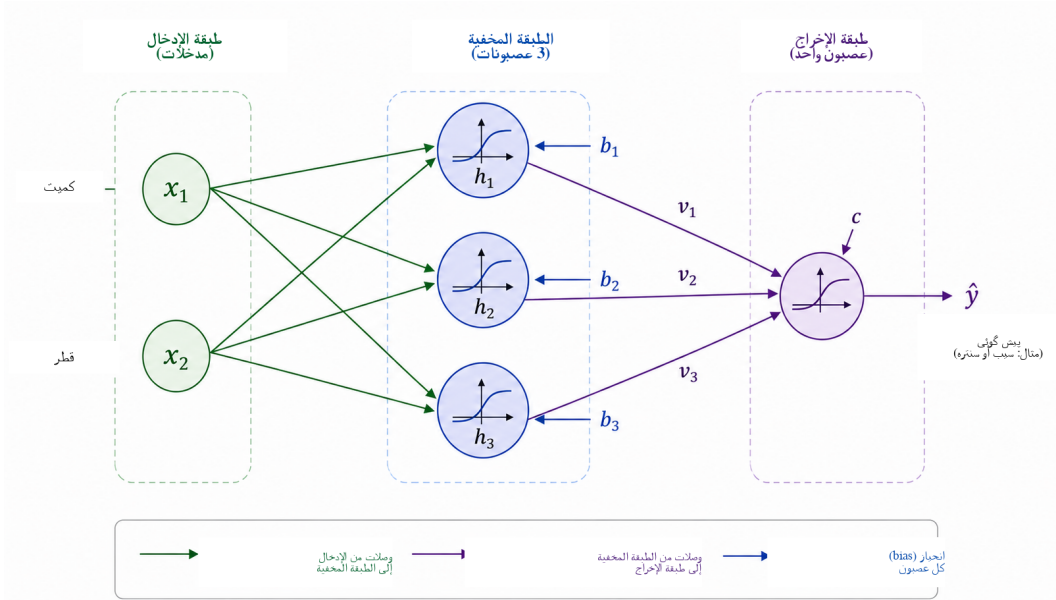
$$h_2 = g(w_{21}x_1 + w_{22}x_2 + b_2),$$

$$h_3 = g(w_{31}x_1 + w_{32}x_2 + b_3).$$

اگرچہ تینوں نیورونز کو ایک ہی کمیت اور قطر ملتے ہیں، وہ مختلف جوابات پیدا کر سکتے ہیں کیونکہ ان کے پیرامیٹرز مختلف ہیں۔ تربیت کے دوران ان پیرامیٹرز کو اس طرح ایڈجسٹ کیا جاتا ہے کہ درمیانی فعال قدریں درجہ بندی کے لیے مفید ہو جائیں۔

پھر تین مقداریں h_1 ، h_2 اور h_3 آؤٹ پٹ نیورون کو دی جاتی ہیں، جو

$$s_{\text{out}} = v_1h_1 + v_2h_2 + v_3h_3 + c$$



شکل 5: اعصابی نیٹ ورک کے حساب کو متعارف کرانے کے لیے استعمال ہونے والی تدریسی 2-3-1 ساخت۔ دو مدخلات --- کمیت اور قطر --- پوشیدہ تہہ کے تین نیورونز تک پہنچتی ہیں۔ ان کی فعال قدریں h_1, h_2, h_3 پھر آؤٹ پٹ نیورون میں جمع ہو کر پیش گوئی $\hat{y}$ پیدا کرتی ہیں۔ ہر رابطے کا ایک قابل ایڈجسٹ وزن اور ہر نیورون کا ایک بائس ہوتا ہے۔

حساب کرتا ہے اور

$$\hat{y} = g_{\text{out}}(s_{\text{out}})$$

پیدا کرتا ہے۔ ہمارے دوئی مسئلے کے لیے ہم سگموئڈ آؤٹ پٹ استعمال کرتے ہیں:

$$\hat{y} = \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3 + c), \quad 0 < \hat{y} < 1.$$

سیب کے لیے $y = 1$ اور سنترے کے لیے $y = 0$ استعمال کرتے ہوئے، 0.5 کی حد جماعتی فیصلہ دیتی ہے

$$\hat{y} \geq 0.5 \Rightarrow \text{سیب}, \quad \hat{y} < 0.5 \Rightarrow \text{سنترہ}.$$

اس لیے 1 کے قریب قدریں سیب سے مضبوطی سے وابستہ ہیں، جبکہ 0 کے قریب قدریں سنترے سے۔ جیسا کہ پہلے ذکر ہوا، $\hat{y}$ کو عددی طور پر احتمال سمجھنے کے لیے ماڈل اور اس کی تربیت سے متعلق اضافی مفروضات درکار ہیں۔ یہاں استعمال ہونے والے درجہ بندی کے قاعدے کے لیے اسے مسلسل نیٹ ورک آؤٹ پٹ سمجھنا کافی ہے۔

اس ساخت کی مرکزی خصوصیت درمیانی نمائندگی ہے

$$(x_1, x_2) \rightarrow (h_1, h_2, h_3) \rightarrow \hat{y}.$$

نیٹ ورک دو خصوصیات وصول کرتا ہے اور آخری جواب دینے سے پہلے تین نئی مقادیر بناتا ہے۔ یہ فعال قدریں دستی طور پر منتخب نہیں کی جاتیں: یہ اوزان اور بائس پر منحصر ہوتی ہیں اور اس لیے تربیت کے دوران پیرامیٹرز کے ساتھ بدلتی رہتی ہیں۔

اپنے چھوٹے حجم کے باوجود، اس نیٹ ورک میں پہلے ہی تیرہ قابل ایڈجسٹ پیرامیٹرز ہیں۔ تین پوشیدہ نیورونز میں

$$3(2 + 1) = 9$$

پیرامیٹرز ہیں، اور آؤٹ پٹ نیورون

$$3 + 1 = 4$$

مزید دیتا ہے۔ لہذا

$$9 + 4 = 13.$$

یہی گنتی میٹرکس نوٹیشن میں بالکل واضح ہو جاتی ہے۔ تعریف کریں

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \mathbf{W}^{(1)} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}, \quad \mathbf{b}^{(1)} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

تب پوشیدہ تہ کو مختصر طور پر یوں لکھا جا سکتا ہے

$$\mathbf{h} = g \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right),$$

جہاں g ہر جزو پر الگ الگ لاگو ہوتا ہے۔ آؤٹ پٹ کے لیے

$$\hat{y} = \sigma \left(\mathbf{W}^{(2)} \mathbf{h} + \mathbf{b}^{(2)} \right),$$

جہاں

$$\mathbf{W}^{(2)} = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix}.$$

یہ نوٹیشن اس ساخت کی پیش گوئی کرتا ہے جو Python پروگرام میں سامنے آئے گی۔ جب کوئی پھل مدخل پر دیا جاتا ہے اور ہم یکے بعد دیگرے فعال قدریں حساب کرتے ہوئے $\hat{y}$ تک پہنچتے ہیں، تو ہم فارورڈ پروپیگیشن یا فارورڈ پاس انجام دیتے ہیں۔ بنیادی سوال اب بھی یہی ہے: تیرہ پیرامیٹرز خودکار طور پر کیسے متعین کیے جائیں تاکہ پیش گوئیاں ڈیٹا کے مطابق ہو جائیں؟

7. اعصابی نیٹ ورک کی سی سی کہتا ہے؟

اوزان اور بائس پہلے سے معلوم نہیں ہوتے۔ انہیں ایسی مثالوں کی مدد سے ایڈجسٹ کرنا پڑتا ہے جن کے درست جواب معلوم ہوں۔ اس عمل کو تربیت کہتے ہیں اور اسے اس چکر سے خلاصہ کیا جا سکتا ہے

$$\boxed{\text{پیش گوئی}} \rightarrow \boxed{\text{خطا ناپیں}} \rightarrow \boxed{\text{پیرامیٹرز ایڈجسٹ کریں}} \rightarrow \boxed{\text{دہرائیں}}.$$

140 g کمیت اور 7.0 cm قطر والے ایک پھل پر غور کریں جس کی درست جماعت سنترہ ہے، یعنی $y = 0$ ۔ فرض کریں تربیت کے کسی مرحلے پر نیٹ ورک $\hat{y} = 0.80$ آؤٹ پٹ دیتا ہے۔ پیش گوئی اور ہدف کے درمیان فرق کی ایک سادہ پیمائش مربع خطا ہے

$$E = \frac{1}{2} (y - \hat{y})^2.$$

اس مثال کے لیے

$$E = \frac{1}{2} (0 - 0.80)^2 = 0.32.$$

یہ اظہار ایک مثال کے لیے لاس فنکشن ہے۔ چونکہ یہ مربع ہے، $E \geq 0$ ، اور ہدف سے زیادہ دور پیش گوئیوں پر زیادہ جرمانہ عائد ہوتا ہے۔ عامل $1/2$ کم از کم کی جگہ نہیں بدلتا، لیکن مشتقات کو سادہ بناتا ہے۔ دوئی درجہ بندی میں دوسرے لاس فنکشنز،

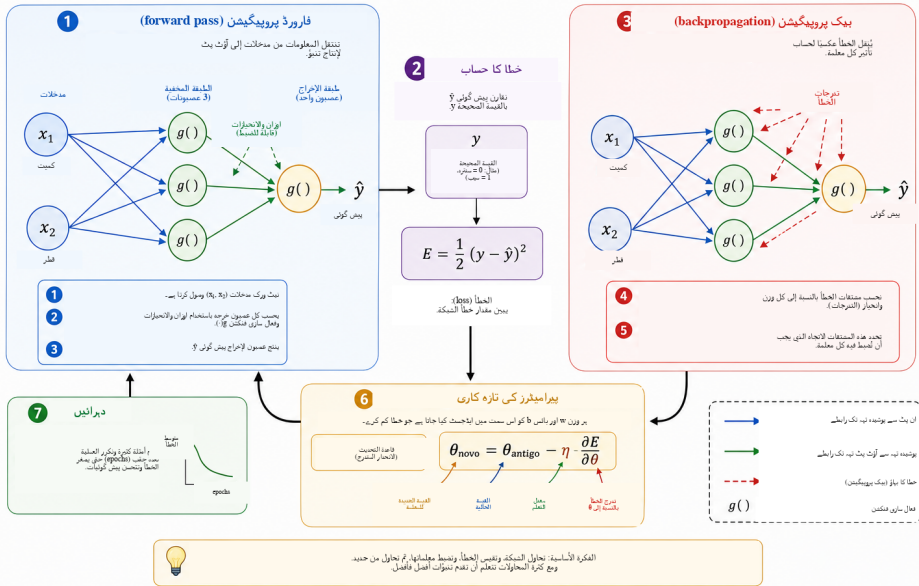
خاص طور پر cross-entropy، بھی کثرت سے استعمال ہوتے ہیں۔ یہاں ہم مربع خطا استعمال کرتے ہیں کیونکہ اس کی الجبرا شفاف ہے اور optimization اور backpropagation متعارف کرانے کے لیے کافی ہے [2, 3]۔

اگر تربیتی مجموعے میں N مثالیں ہوں تو ایک مجموعی پیمانہ اوسط لاس ہے

$$J = \frac{1}{N} \sum_{i=1}^N E_i.$$

ہر پیش گوئی نیٹ ورک کے اوزان اور بائس پر منحصر ہے، اس لیے J بھی ان تمام پیرامیٹرز پر منحصر ہے۔ تربیت کا مطلب ایسے پیرامیٹرز تلاش کرنا ہے جو J کو چھوٹا کریں۔

ایڈجسٹمنٹ کے طریقے کو سمجھنے کے لیے پہلے فرض کریں کہ ایک وزن w کے سوا تمام پیرامیٹرز مستقل ہیں۔ تب لاس کو $J = J(w)$ ، فنکشن سمجھا جا سکتا ہے۔ شکل 6 تربیت کے بنیادی مراحل کا خلاصہ کرتی ہے جنہیں اب ہم دیکھتے ہیں۔



شکل 6: اعصابی نیٹ ورک کی تربیت کا بنیادی چکر۔ فارورڈ پاس مدخلات کو پیش گوئی میں تبدیل کرتا ہے؛ لاس فنکشن پیش گوئی اور ہدف کے اختلاف کو ناپتا ہے؛ بیک پروپیگیشن chain rule کے ذریعے گریڈیئنٹس حساب کرتا ہے؛ اور گریڈیئنٹ ڈیسنٹ ان ہی گریڈیئنٹس سے اوزان اور بائس کو اپ ڈیٹ کرتا ہے۔ کئی epochs تک یہ چکر دہرانہی وہ عمل ہے جسے ہم سیکنڈا کہتے ہیں۔

جن قارئین کا حسابان سے کم تعارف ہے، ان کے لیے مشتق کو یہاں مقامی حساسیت کی پیمائش کے طور پر سمجھا جا سکتا ہے: یہ بتاتا ہے کہ کسی ایک پیرامیٹر میں چھوٹی تبدیلی لاس کو کس طرح بدلنے کا رجحان رکھتی ہے۔ اس کا نشان بتاتا ہے کہ کون سی سمت مقامی طور پر فنکشن کو بڑھاتی یا گھٹاتی ہے، جبکہ اس کی قدر اس مقامی تبدیلی کی شدت بتاتی ہے۔

مشتق

$$\frac{\partial J}{\partial w}$$

مقامی ڈھلوان دیتا ہے۔ اگر یہ مثبت ہو تو w میں تھوڑا اضافہ عموماً J بڑھاتا ہے، اس لیے وزن کو مخالف سمت میں منتقل کرنا چاہیے۔ اگر یہ منفی ہو تو w بڑھانے سے مقامی طور پر J کم ہونے کا رجحان ہوتا ہے۔ دونوں حالتیں اس قاعدے میں سمٹ جاتی

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial J}{\partial w},$$

جو گریڈینٹ ڈیسٹ کا بنیادی تصور ہے۔ مثبت عدد η سیکھنے کی شرح ہے؛ یہ قدم کے حجم کو کنٹرول کرتا ہے۔ بہت چھوٹی قدر تربیت کو سست کر سکتی ہے، جبکہ بہت بڑی قدر ارتعاش یا عدم استحکام پیدا کر سکتی ہے۔
کئی پیرامیٹرز والے نیٹ ورک کے لیے θ کو کوئی بھی وزن یا بائس سمجھیں۔ اپ ڈیٹ یوں ہو جاتا ہے

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \frac{\partial J}{\partial \theta}.$$

تمام جزوی مشتقات کا مجموعہ لاس کا گریڈینٹ بناتا ہے۔ ہندسی طور پر گریڈینٹ تیز ترین مقامی اضافے کی سمت بتاتا ہے، اس لیے منفی گریڈینٹ مقامی کمی کی سمت فراہم کرتا ہے۔
اب بھی یہ حساب کرنا باقی ہے کہ جب نیٹ ورک کئی فنکشنز کا مرکب ہو تو یہ مشتقات کیسے حاصل کیے جائیں۔ فارورڈ پاس میں معلومات اس ترتیب سے گزرتی ہے

اؤٹ پٹ → پوشیدہ تہ → مدخلات

ابتدائی تہ کے کسی پیرامیٹر کا لاس پر اثر معلوم کرنے کے لیے ہم انحصارات کے سلسلے کو ریاضیاتی طور پر الٹی سمت میں دنبال کرتے ہیں۔ اس عمل کو بیک پروپیگیشن کہتے ہیں۔ بنیادی طور پر یہ chain rule کا منظم اور مؤثر استعمال ہے۔ Rumelhart، Hinton اور Williams [8] پوشیدہ تہوں والے نیٹ ورکس میں اس طریقے کی مقبولیت کا کلاسیکی حوالہ فراہم کرتے ہیں، جبکہ Nielsen [7] ایک مفصل اور خاص طور پر تدریسی اخذ پیش کرتے ہیں۔

ایک سادہ انحصاری زنجیر میں، جہاں E کا انحصار $\hat{y}$ پر، $\hat{y}$ کا فعال قدر h پر، اور h کا وزن w پر ہو، chain rule دیتا ہے

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial w}.$$

ہر عامل آخری لاس پر w کے اثر کے ایک مرحلے کو بیان کرتا ہے۔ بڑے نیٹ ورکس میں ایسی بہت سی زنجیریں ہوتی ہیں، اور بیک پروپیگیشن حساب کو اس طرح منظم کرتا ہے کہ درمیانی مشتقات کو ہر پیرامیٹر کے لیے الگ سے دوبارہ حساب کرنے کے بجائے دوبارہ استعمال کیا جا سکے۔

عملی تربیت میں بہت سی مثالیں استعمال ہوتی ہیں۔ پورے تربیتی مجموعے سے ایک مکمل گزر کو epoch کہتے ہیں۔ پیرامیٹرز ہر مثال کے بعد اپ ڈیٹ کیے جا سکتے ہیں، جیسا کہ اس مضمون کے بعد کے تدریسی پروگرام میں ہوگا، یا مثالوں کے چھوٹے گروہوں کے بعد جنہیں mini-batches کہتے ہیں، جو بڑے نیٹ ورکس میں عام ہیں۔ دونوں صورتوں میں پیش گوئی، لاس، گریڈینٹ اور اپ ڈیٹ کا چکر کئی epochs تک دہرایا جاتا ہے۔

اس ارتقا کو ٹھوس طور پر دیکھنے کا ایک طریقہ یہ ہے کہ ایک ہی مثال کی پیش گوئی کو تربیت کے مختلف مراحل میں دیکھا جائے۔ دوبارہ اس پھل پر غور کریں جس کی کمیت 140 g، قطر 7.0 cm اور درست جماعت سنترہ ہے۔ جدول 1 ایک فرضی پیش رفت دکھاتا ہے۔ یہ قدریں کسی مخصوص run سے نہیں، صرف طریقہ کار کی وضاحت کے لیے ہیں۔

ابتدا میں اؤٹ پٹ 1 کے قریب ہے اور پھل غلطی سے سیب درجہ بند ہوتا ہے۔ پیرامیٹرز ایڈجسٹ ہوتے جاتے ہیں تو پیش گوئی کم ہوتی ہے اور آخر کار 0.5 کی حد عبور کرکے درست جماعت پیدا کرتی ہے۔ اس عمل میں اوزان اور بائس کی عددی قدریں بدلتی ہیں؛ نیٹ ورک کی ساخت مستقل رہتی ہے۔

تربیتی ڈیٹا پر لاس کم کرنا آخری مقصد نہیں۔ ایک مفید نیٹ ورک کو ان مثالوں پر بھی اچھی پیش گوئیاں کرنی چاہئیں جو پیرامیٹرز موزوں کرنے میں استعمال نہیں ہوئیں۔ اس صلاحیت کو generalization کہتے ہیں۔ جب ماڈل تربیتی مجموعے کی مخصوص تفصیلات کے ساتھ ضرورت سے زیادہ مطابقت اختیار کر لے اور نئے ڈیٹا پر خراب کارکردگی دکھائے تو اسے overfitting کہتے ہیں۔ زیادہ محتاط جائزوں میں ڈیٹا کو عموماً training، validation اور test مجموعوں میں تقسیم کیا جاتا ہے: پہلا پیرامیٹرز موزوں کرتا ہے، دوسرا ماڈل بنانے وقت فیصلوں کی رہنمائی کرتا ہے، اور تیسرا آخری جائزے کے لیے محفوظ رہتا ہے۔ generalization اور ماڈل evaluation پر وسیع تر بحث [2] Faceli et al اور [3] Courville و Goodfellow، Bengio میں ملتی ہے۔

جدول 1: تربیت کے دوران ایک ہی پھل کے لیے نیٹ ورک کی پیش گوئی کا فرضی ارتق۔ بار بار پیرامیٹر آپ ڈیٹس ابتدا میں غلط آؤٹ پٹ کو ہدفی لیبل کے مطابق درجہ بندی کی طرف لے جا سکتی ہیں۔

Epoch	پیش گوئی $\hat{y}$	لاس E	درجہ بندی
1	0.86	0.37	سیب
5	0.71	0.25	سیب
10	0.55	0.15	سیب
20	0.31	0.05	سن ترہ
40	0.09	0.004	سن ترہ

یوں ہم اعصابی نیٹ ورک میں سیکھنے کے ریاضیاتی معنی تک پہنچتے ہیں۔ اس تناظر میں سیکھنے کا مطلب ایک مرکب فنکشن کے پیرامیٹرز کو اس طرح ایڈجسٹ کرنا ہے کہ ڈیٹا پر خطا کی پیمائش کم ہو اور، مثالی طور پر، نئے نمونوں پر بھی اچھی عمومی کارکردگی حاصل ہو۔ نیٹ ورک پیش گوئی کرتا ہے، لاس حساب کرتا ہے، مشتقات سے طے کرتا ہے کہ پیرامیٹرز اس لاس کو کیسے متاثر کرتے ہیں، اور پھر چھوٹی چھوٹی اصلاحات کا سلسلہ انجام دیتا ہے۔

8. ریاضی سے Python تک

اب ہم مساوات سے ایک قابل اجرا پروگرام کی طرف بڑھتے ہیں۔ مقصد کسی مخصوص اعصابی نیٹ ورک لائبریری کو استعمال کرنا نہیں، بلکہ مضمون بھر میں تیار کیے گئے ریاضیاتی عملیات کو NumPy کے ذریعے براہ راست نافذ کرنا ہے [4]۔ اس طرح سطر بہ سطر دیکھا جا سکتا ہے کہ خطی امتزاج، فعال سازی فنکشنز، لاس کا حساب، بیک پروپیگیشن اور گریڈینٹ ڈیسنٹ کوڈ میں کس طرح ظاہر ہوتے ہیں۔

زیادہ قائل کرنے والی آخری مثال کے لیے ہم ایک مصنوعی ڈیٹا سیٹ استعمال کرتے ہیں جو کمیت-قطر کے زیادہ وسیع خطے میں پھیلا ہوا ہے۔ ہم اب بھی صرف دو طبعی خصوصیات — گرام میں کمیت اور سینٹی میٹر میں قطر — استعمال کرتے ہیں، لیکن اب 64 تربیتی مثالوں کو سطح کے ایک بڑے حصے میں پھیلاتے ہیں اور آٹھ اضافی مثالیں ایک وضاحتی آزمائش کے لیے الگ رکھتے ہیں۔ ڈیٹا مصنوعی ہے اور سیبوں اور سنتروں کے معقول حدود میں رکھا گیا ہے۔ اسے اس طرح بنایا گیا ہے کہ تمام نقاط کو چند غیر فطری گچھوں میں سمیٹے بغیر ایک غیر خطی فیصلہ حد حاصل ہو۔

ہندسی تصور سادہ ہے۔ ہم کمیت-قطر کی سطح میں ایک حوالہ منحنی تعریف کرتے ہیں، سیبوں کو اس منحنی کے اوپر دو پٹیوں میں، اور سنتروں کو نیچے دو پٹیوں میں رکھتے ہیں۔ مشاہدات کو وسیع تر خطے میں پھیلانے کے لیے کمیت اور قطر دونوں میں چھوٹی متعین تبدیلیاں شامل کی جاتی ہیں۔ حاصل شدہ مثال پچھلی تصویروں سے زیادہ بھرپور ہے: اس میں زیادہ نقاط ہیں، ڈیٹا خطے میں پھیلا ہوا ہے، اور مناسب فیصلہ حد سیدھی لکیر سے واضح طور پر زیادہ منحنی ہے۔

اس آخری حسابی تجربے کے لیے ہم پوشیدہ تہ کو بارہ ReLU نیوروز تک بڑھاتے ہیں۔ اس سے پچھلے حصوں میں مطالعہ کی گئی تصوری ساخت نہیں بدلتی؛ صرف زیادہ پیچیدہ حد کی نمائندگی کرنے کی نیٹ ورک کی صلاحیت بڑھتی ہے۔ تین نیوروز والے تدریسی نیٹ ورک کے 13 پیرامیٹرز کے بجائے اب ہمارے پاس 49 قابل ایڈجسٹ پیرامیٹرز ہیں: پہلی تہ میں 24 اوزان اور 12 بائس، پھر آؤٹ پٹ تہ میں 12 اوزان اور ایک بائس۔ مساوات بالکل وہی صورت رکھتی ہیں؛ صرف ویکٹر اور میٹرکس بڑے ہو جاتے ہیں۔

مکمل پروگرام ذیل میں دیا گیا ہے۔

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

def relu(s):
    return np.maximum(0, s)
```

```

def relu_derivative(s):
    return (s > 0).astype(float)

def sigmoid(s):
    return 1 / (1 + np.exp(-s))

def base_boundary(m):
    return 6.68 + 0.74*np.exp(-((m-155)/20)**2) \
        + 1.50/(1 + np.exp(-(m-202)/5))

def build_data():
    masses_1 = np.linspace(118, 212, 16)
    masses_2 = masses_1 + np.array([
        1.2,-1.1,0.8,-1.3,1.0,-0.8,1.1,-1.0,
        1.3,-0.9,0.9,-1.2,1.0,-1.0,0.8,-0.7
    ])
    k = np.arange(16)

    a1 = np.column_stack([
        masses_1,
        base_boundary(masses_1) + 0.22 + 0.05*np.sin(0.8*k)
    ])
    a2 = np.column_stack([
        masses_2,
        base_boundary(masses_2) + 0.58 + 0.06*np.cos(0.45*k)
    ])

    o1 = np.column_stack([
        masses_1,
        base_boundary(masses_1) - (0.22 + 0.04*np.cos(0.7*k))
    ])
    o2 = np.column_stack([
        masses_2,
        base_boundary(masses_2) - (0.55 + 0.05*np.sin(0.5*k))
    ])

    X_train = np.vstack([a1, a2, o1, o2])
    y_train = np.array([1.0]*32 + [0.0]*32)

    X_test = np.array([
        [121.0, 7.20], [135.0, 7.45],
        [152.0, 7.75], [171.0, 7.32],
        [121.0, 6.28], [152.0, 6.86],
        [189.0, 6.48], [208.0, 7.55]
    ])
    y_test = np.array([1,1,1,1,0,0,0], dtype=float)

    return X_train, y_train, X_test, y_test

def predict(X, mean, std, W1, b1, W2, b2):
    Xn = (X - mean) / std
    outputs = []
    for x in Xn:
        h = relu(W1 @ x + b1)
        y_hat = sigmoid(W2 @ h + b2)
        outputs.append(y_hat)

```

```

return np.array(outputs)

X_train, y_train, X_test, y_test = build_data()

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
Xn = (X_train - mean) / std

rng = np.random.default_rng(42)
W1 = 0.1 * rng.standard_normal((12, 2))
b1 = np.zeros(12)
W2 = 0.1 * rng.standard_normal(12)
b2 = 0.0
eta = 0.05
n_epochs = 6000
history = []

for epoch in range(n_epochs):
    for i in rng.permutation(len(Xn)):
        x = Xn[i]
        target = y_train[i]

        z1 = W1 @ x + b1
        h = relu(z1)
        z2 = W2 @ h + b2
        y_hat = sigmoid(z2)

        delta2 = (y_hat - target) * y_hat * (1 - y_hat)
        dW2 = delta2 * h
        db2 = delta2
        delta1 = (delta2 * W2) * relu_derivative(z1)
        dW1 = np.outer(delta1, x)
        db1 = delta1

        W1 -= eta * dW1
        b1 -= eta * db1
        W2 -= eta * dW2
        b2 -= eta * db2

    errors = []
    for x, target in zip(Xn, y_train):
        h = relu(W1 @ x + b1)
        y_hat = sigmoid(W2 @ h + b2)
        errors.append(0.5 * (target - y_hat)**2)
    history.append(np.mean(errors))

def make_figure():
    masses = np.linspace(116, 214, 500)
    diameters = np.linspace(6.0, 8.85, 500)
    M, D = np.meshgrid(masses, diameters)
    grid = np.column_stack([M.ravel(), D.ravel()])
    P = predict(grid, mean, std, W1, b1, W2, b2).reshape(M.shape)

    fig, ax = plt.subplots(figsize=(9.2, 6.3))
    ax.contourf(
        M, D, P,
        levels=[0.0, 0.25, 0.50, 0.75, 1.0],
        colors=["#f3e0cb", "#ecd5c4", "#dbe7cf", "#cde3bf"],

```

```

alpha=0.95
)
ax.contour(M, D, P, levels=[0.5],
           colors=["#5a1a8a"], linewidths=2.2)

mask = y_train == 1.0
ax.scatter(X_train[mask,0], X_train[mask,1], marker="o",
           s=48, facecolors="white", edgecolors="black",
           linewidths=1.1, label="Apple - training")
mask = y_train == 0.0
ax.scatter(X_train[mask,0], X_train[mask,1], marker="s",
           s=48, facecolors="white", edgecolors="black",
           linewidths=1.1, label="Orange - training")
mask = y_test == 1.0
ax.scatter(X_test[mask,0], X_test[mask,1], marker="^",
           s=92, color="#1f77b4", label="Apple - test")
mask = y_test == 0.0
ax.scatter(X_test[mask,0], X_test[mask,1], marker="x",
           s=110, color="#ff7f0e", linewidths=2.0,
           label="Orange - test")

ax.set_xlabel("Mass (g)")
ax.set_ylabel("Diameter (cm)")
ax.set_title("Neural-network output, after training")

handles = [
    Line2D([0],[0], marker='o', markersize=9,
            markerfacecolor='white', markeredgcolor='black',
            linestyle='None', label='Apple - training'),
    Line2D([0],[0], marker='s', markersize=9,
            markerfacecolor='white', markeredgcolor='black',
            linestyle='None', label='Orange - training'),
    Line2D([0],[0], marker='^', markersize=9, color='#1f77b4',
            linestyle='None', label='Apple - test'),
    Line2D([0],[0], marker='x', markersize=10, color='#ff7f0e',
            linestyle='None', label='Orange - test'),
    Line2D([0],[0], color='#5a1a8a', lw=2.2,
            label='Neural-network decision boundary')
]
ax.legend(handles=handles, loc='center left',
          bbox_to_anchor=(1.02, 0.5), frameon=True)

fig.tight_layout()
plt.show()

```

اگرچہ کوڈ مضمون میں پہلے مطالعہ کی گئی مساوات سے لمبا ہے، اس کی ساخت شفاف رہتی ہے۔ `build_data()` فنکشن خودکار طور پر 64 تربیتی مثالیں اور آٹھ وضاحتی آزمائشی مثالیں بناتا ہے۔ پھر ہم دونوں متغیرات کو اس قاعدے کے مطابق معیاری بناتے ہیں

$$x_j^* = \frac{x_j - \mu_j}{\sigma_j},$$

جہاں μ_j اور σ_j صرف تربیتی مجموعے سے حساب کیے جاتے ہیں۔ اس سے آزمائشی مجموعہ پیرامیٹرز کی fitting پر اثر انداز نہیں ہوتا اور کمیت و قطر کو قابل موازنہ عددی پیمانوں پر بھی لایا جاتا ہے۔

فارورڈ پاس اپنی اصل میں نہیں بدلتا۔ فرق صرف یہ ہے کہ پہلی تہ اب بارہ پوشیدہ فعال قدروں کا ویکٹر پیدا کرتی ہے۔ میٹرکس نوٹیشن میں،

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \mathbf{h} = \text{ReLU}(\mathbf{z}^{(1)}),$$

$$z^{(2)} = W^{(2)}h + b^{(2)}, \quad \hat{y} = \sigma(z^{(2)}).$$

یہ چار سطریں اب بھی الگورتھم کا ریاضیاتی جوہر رکھتی ہیں۔ تربیت بھی پہلے متعارف کرائی گئی منطق برقرار رکھتی ہے: پیش گوئی حساب کریں، خطا ناپیں، مشتقات کی مدد سے حساسیت کی معلومات پیچھے منتقل کریں، اور چھوٹے گریڈینٹ ڈیسنٹ قدموں سے پیرامیٹرز اپ ڈیٹ کریں۔

کوڈ میں آؤٹ پٹ تہ پر بیک پروپیگیشن اس سطر میں ظاہر ہوتی ہے

$$\text{delta2} = (\text{y_hat} - \text{target}) * \text{y_hat} * (1 - \text{y_hat})$$

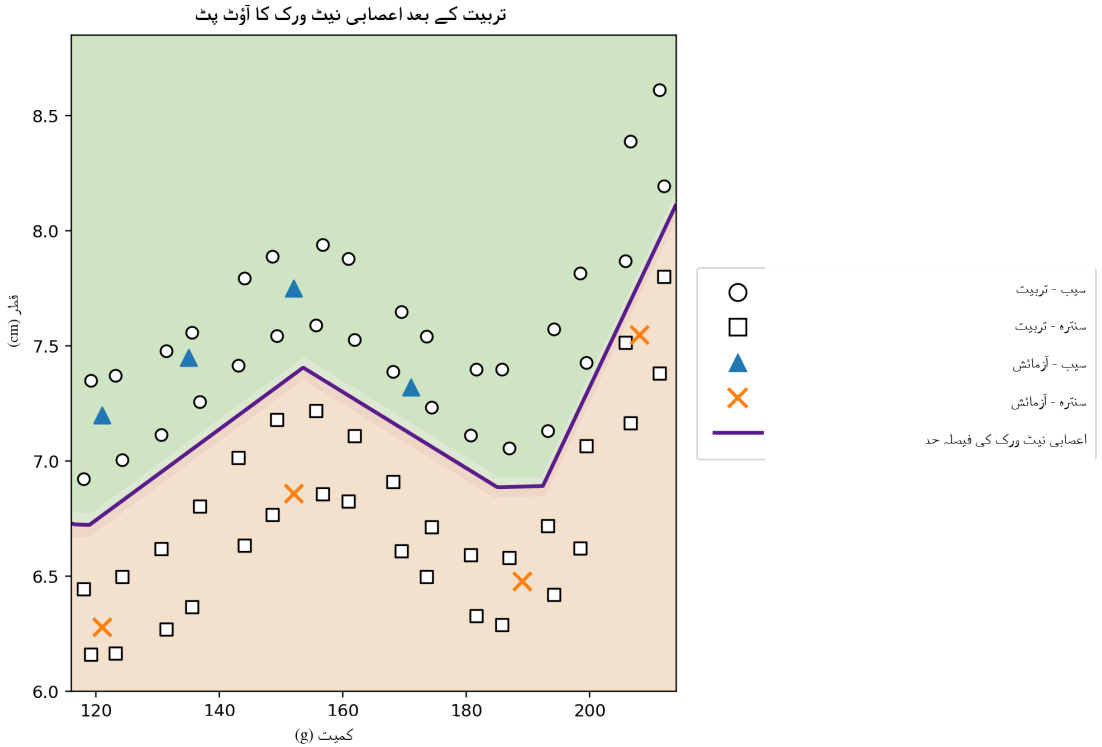
اور پوشیدہ تہ تک منتقلی اس سطر میں

$$\text{delta1} = (\text{delta2} * W2) * \text{relu_derivative}(z1)$$

نظر آتی ہے، جس کے بعد تمام اوزان اور بائس اپ ڈیٹ ہوتے ہیں۔ چونکہ ہر epoch میں مثالوں کی ترتیب بدلی جاتی ہے اور ہر مشاہدے کے بعد پیرامیٹرز اپ ڈیٹ ہوتے ہیں، یہ تنفیذ stochastic gradient descent کی ایک بنیادی شکل ہے۔

یہاں استعمال ہونے والے تصادفی بیج اور ہائپر پیرامیٹرز کے ساتھ آخری اوسط لاس 10^{-5} سے 10^{-4} کے درجے میں ہے، اور تمام 64 تربیتی مثالیں اور آٹھ وضاحتی آزمائشی مثالیں متوقع درجہ بندی حاصل کرتی ہیں۔ اسے سخت شماریاتی کارکردگی کے جائزے کے طور پر نہیں سمجھنا چاہیے: ڈیٹا مصنوعی ہے اور تدریسی مقصد کے لیے بنایا گیا ہے۔ اس کا کردار ایک کم معمولی صورت حال میں یہ دکھانا ہے کہ ایک چھوٹا اعصابی نیٹ ورک واضح طور پر غیر خطی فیصلہ حد سیکھ سکتا ہے۔

شکل 7 کمیت-قطر کی سطح میں پروگرام کا آخری آؤٹ پٹ دکھاتی ہے۔ تمام تربیتی نقاط دکھائے گئے ہیں۔ نمایاں منحنی سطح $\hat{y} = 0.5$ ہے اور نیٹ ورک کی سیکھی ہوئی فیصلہ حد کی نمائندگی کرتا ہے؛ سایہ دار خطے نیٹ ورک کی پیش گوئی کردہ دو جماعتیں دکھاتے ہیں۔ دائرے تربیتی سیب، مربع تربیتی سنترے، مثلث آزمائشی سیب، اور ضرب کے نشان آزمائشی سنترے ہیں۔ وضاحتی کلید کو مرکزی رسم کے باہر رکھا گیا ہے تاکہ کوئی ڈیٹا نقطہ چھپ نہ جائے۔



شکل 7: اعصابی نیٹ ورک کی تربیت کے بعد حاصل ہونے والے Python آؤٹ پٹ۔ مصنوعی ڈیٹا سیٹ میں کمی-تقطر کی سطح کے وسیع خطے میں پھیلی 64 تربیتی مثالیں اور وضاحتی آزمائش کے لیے محفوظ آٹھ اضافی مثالیں شامل ہیں۔ نمایاں منحنی سطح $\hat{y} = 0.5$ ہے اور سیکیوٹی ہوئی فیصلہ حد کی نمائندگی کرتا ہے؛ سایہ دار خطے نیٹ ورک کی پیش گوئی کردہ جماعتیں دکھاتے ہیں۔ دائرے تربیتی سیب، مربع تربیتی سنترے، مثلث آزمائشی سیب، اور ضرب کے نشان آزمائشی سنترے ظاہر کرتے ہیں۔

ہم نے دو سادہ نقطہ گروہوں کو جدا کرنے والی سیدھی لکیر سے آغاز کیا اور ایک زیادہ بھرپور حسابی مثال پر اختتام کیا جس میں نیٹ ورک کے پیرامیٹرز کی خودکار ایڈجسٹمنٹ سے ایک منحنی حد نمودار ہوتی ہے۔ ریاضی میں کوئی پراسرار چیز شامل نہیں ہوئی: تنفیذ صرف افائن امتزاج، فعال سازی فنکشنز، مشتقات، میٹرکس ضرب اور گریڈینٹ ڈیسنٹ اپ ڈیٹس کو بڑے پیمانے اور تیز رفتاری سے منظم کرکے دہراتی ہے [3, 7]۔

اس مضمون سے متعلق تمام حسابی مواد — کوڈ اور قابل اجرا Google Colab notebook سمیت — <https://neuralmath.org> پر دستیاب ہے، تاکہ مثالوں کو دیکھنے، دوبارہ پیدا کرنے یا اپنی ضرورت کے مطابق ڈھالنے والے قارئین کے لیے ایک واحد رسائی مقام فراہم ہو۔

مصنوعی ذہانت کے استعمال کا بیان۔

مصنف نے OpenAI کی تیار کردہ ChatGPT کو مسودہ سازی، لسانی نظر ثانی، توضیحی تنظیم اور تدریسی اشکال بنانے میں معاون آلے کے طور پر استعمال کیا۔ ریاضیاتی مواد کے تصور اور ترقی، کوڈ اور استعمال شدہ حوالہ جات سمیت کام کے آخری مواد کی مکمل ذمہ داری مصنف پر ہے۔

حوالہ جات

- [1] M. Batista and A. O. Martins. “Redes Neurais no Ensino Básico”. *Professor de Matemática Online*, v. 10, n. 4, p. 454–481, 2022. doi: 10.21711/2319023x2022/pmo1032
- [2] K. Faceli, A. C. Lorena, J. Gama, T. A. de Almeida, and A. C. P. L. F. de Carvalho. *Inteligência Artificial: Uma Abordagem de Aprendizado de Máquina*. 2nd ed. Rio de Janeiro: LTC, 2021.
- [3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Cambridge, MA: MIT Press, 2016. Available at: <http://www.deeplearningbook.org>
- [4] C. R. Harris, K. J. Millman, S. J. van der Walt et al. “Array programming with NumPy”. *Nature*, v. 585, p. 357–362, 2020. doi: 10.1038/s41586-020-2649-2
- [5] S. Haykin. *Redes Neurais: Princípios e Prática*. Translation by Paulo Martins Engel. 2nd ed. Porto Alegre: Bookman, 2000.
- [6] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. *Nature*, v. 521, p. 436–444, 2015. doi: 10.1038/nature14539
- [7] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. Available at: <https://neuralnetworksanddeeplearning.com>
- [8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. “Learning representations by back-propagating errors”. *Nature*, v. 323, p. 533–536, 1986. doi: 10.1038/323533a0

Americo Cunha Jr

NeuralMath

ریاضی · عصبی نیٹ ورک · Python

دوسری زبانوں میں پڑھیں

ja · 日本語

ar · العربية

ko · 한국어

bn · বাংলা

nl · Nederlands

cs · Čeština

pl · Polski

de · Deutsch

pt · Português (اصل)

el · Ελληνικά

ro · Română

en · English

ru · Русский

es · Español

sv · Svenska

fa · فارسی

tr · Türkçe

fr · Français

uk · Українська

hi · हिन्दी

ur · اردو

hu · Magyar

vi · Tiếng Việt

id · Bahasa Indonesia

zh · 简体中文

it · Italiano



<https://neuralmath.org>

1.0 نسخه · DOI: 10.5281/zenodo.23212381 · CC-BY-4.0