richer outputs: quantifying uncertainty

By the end of this section, you'll be able to
- define a class of linear, probabilistic hypotheses appropriate to a given classification or regression task
- compute the loss suffered by a probabilistic hypothesis on given data

TWO THIRDS BETWEEN DOG AND COW — Remember: our Unit 1 motto is to *learn linearities flanked by hand-coded nonlinearities*:

$$\begin{tikzcd} \mathcal{X} \arrow[r, "\text{featurize}", "\text{not learned}"'] & \mathbb{R}^2 \arrow[r, "\text{linearly combine}", "\textbf{learned!}"'] & \mathbb{R}^1 \arrow[r, "\text{read out}", "\text{not learned}"'] & \mathcal{Y} \end{tikzcd}$$

We design the nonlinearities to capture domain knowledge about our data and goals. Here we'll design nonlinearities to help model *uncertainty* over $\mathcal{Y}$. We can do this by choosing a different read-out function. For example, representing distributions by objects `{3:prob_of_three, 1:prob_of_one}`, we could choose:

```
prediction = {3 : 0.8 if threeness[0]>0. else 0.2,
              1 : 0.2 if threeness[0]>0. else 0.8 }
```

If before we'd have predicted "the label is 3", we now predict "the label is 3 with 80% chance and 1 with 20% chance". This hard-coded 80% *could* suffice.° But let's do better: intuitively, a 3 is more likely when `threeness` is huge than when `threeness` is nearly zero. So let's replace that 80% by some smooth function of `threeness`. A popular, theoretically warranted choice is $\sigma(z) = 1/(1+\exp(-z))$:°

← As always, it depends on what specific thing we're trying to do!

← σ, the **logistic** or **sigmoid** function, has linear log-odds: $\sigma(z)/(1-\sigma(z)) = \exp(z)/1$. It tends exponentially to the step function. It's symmetrical: $\sigma(-z) = 1-\sigma(z)$. Its derivative concentrates near zero: $\sigma'(z) = \sigma(z)\sigma(-z)$. Food For Thought: Plot $\sigma(z)$ by hand.

```
sigma = lambda z : 1./(1.+np.exp(-z))
prediction = {3 :    sigma(threeness[0]),
              1 : 1.-sigma(threeness[0]) }
```

Given training inputs x_i, a hypothesis will have "hunches" about the training outputs y_i. Three hypotheses $h_{\text{three!}}$, h_{three}, and h_{one} might, respectively, confidently assert $y_{42} = 3$; merely lean toward $y_{42} = 3$; and think $y_{42} = 1$. If in reality $y_{42} = 1$ then we'd say h_{one} did a good job, h_{three} a bad job, and $h_{\text{three!}}$ a very bad job on the 42nd example. So the training set "surprises" different hypotheses to different degrees. We may seek a hypothesis $h_\star$ that is minimally surprised, i.e., usually confidently right and when wrong not confidently so. In short, by outputting probabilities instead of mere labels, we've earned this awesome upshot: *the machine can automatically calibrate its confidence levels!* It's easy to imagine how important this calibration is in language, self-driving, etc.

HUMBLE MODELS — Let's modify logistic classification to allow for *unknown unknowns*. We'll do this by allowing a classifier to allot probability mass not only among labels in $\mathcal{Y}$ but also to a special class $\star$ that means "no comment" or "alien input". A logistic classifier always sets $p_{y|x}[\star|x] = 0$, but other probability models may put nonzero mass on "no comment". Different probability models give different learning programs. In fact, two simple probability models give us the perceptron and hinge losses we saw previously!

	LOGISTIC	PERCEPTRON	SVM
$p_{y\|x}[+1\|x]$	$\oplus/(\ominus+\oplus)$	$\oplus\cdot(\ominus\wedge\oplus)/2$	$\oplus\cdot(\ominus\wedge\oplus/e)/2$
$p_{y\|x}[-1\|x]$	$\ominus/(\ominus+\oplus)$	$\ominus\cdot(\ominus\wedge\oplus)/2$	$\ominus\cdot(\ominus/e\wedge\oplus)/2$
$p_{y\|x}[\star\|x]$	$1-$ above $=0$	$1-$ above	$1-$ above
outliers	responsive	robust	robust
inliers	sensitive	blind	sensitive
acc bnd	good	bad	good
loss name	$\text{softplus}(\cdot)$	$\text{srelu}(\cdot)$	$\text{hinge}(\cdot)$
formula	$\log_2(1+e^{(\cdot)})$	$\max(1,\cdot+1)$	$\max(0,\cdot+1)$
update	$1/(1+e^{+y\eth})$	$\text{step}(-y\eth)$	$\text{step}(1-y\eth)$

Table 1: **Three popular models for binary classification. Top rows:** Modeled chance given x that $y=+1,-1,\star$. We use $\eth=\vec{w}\cdot\vec{x}$, $\oplus=e^{+\eth/2}$, $\ominus=e^{-\eth/2}$, $a\wedge b=\min(a,b)$ to save ink. **Middle rows:** All models respond to misclassifications. But are they robust to well-classified outliers? Sensitive to well-classified inliers? **Bottom rows:** For optimization, which we'll discuss later, we list (negative log-probability) losses. An SGD step looks like

$$\vec{w}_{t+1}=\vec{w}_t+\eta\cdot\text{update}\cdot y\vec{x}$$

Food For Thought: For each of the three models, plot $p_{y|x}[+1|x]$ and $p_{y|x}[-1|x]$ against the decision function value $\eth$. For which decision function values $\eth$ does the svm model allocate chance to the "no-comment" outcome $\star$? Relate to the margin.

MLE with the perceptron model or svm model minimizes the same thing, but with $\text{srelu}(z)=\max(1,1+z)$ or $\text{hinge}(z)=\max(0,1+z)$ instead of $\text{softplus}(z)$.

Two essential properties of softplus are that: (a) it is convex° and (b) it upper bounds the step function. Note that srelu and hinge also enjoy these properties. Property (a) ensures that the optimization problem is relatively easy — under mild conditions, gradient descent will find a global minimum. By property (b), the total loss on a training set upper bounds the rate of erroneous classification on that training set. So loss is a *surrogate* for (in)accuracy: if the minimized loss is nearly zero, then the training accuracy is nearly 100%.°

← A function is **convex** when its graph is bowl-shaped rather than wriggly. It's easy to minimize convex functions by 'rolling downhill', since we'll never get stuck in a local wriggle. Don't worry about remembering or understanding this word.

← The perceptron satisfies (b) in a trivial way that yields a vacuous bound of 100% on the error rate.

So we have a family of related models: **logistic**, **perceptron**, and **SVM**. In Project 1 we'll find hypotheses optimal with respect to the perceptron and SVM models (the latter under a historical name of **pegasos**), but soon we'll focus mainly on logistic models, since they fit best with deep learning.

RICHER OUTPUTS: MULTIPLE CLASSES — We've explored hypotheses $f_W(x) = \text{readout}(W\cdot\text{featurize}(x))$ where W represents the linear-combination step we tune to data. We began with **hard binary classification**, wherein we map inputs to definite labels (say, $y=\text{cow}$ or $y=\text{dog}$):

$$\text{readout}(\eth) = \text{"cow if } 0<\eth \text{ else dog"}$$

We then made this probabilistic using σ. In such **soft binary classification** we return (for each given input) a *distribution* over labels:

$$\text{readout}(\eth) = \text{"chance } \sigma(\eth) \text{ of cow; chance } 1-\sigma(\eth) \text{ of dog"}$$

Remembering that $\sigma(\mathfrak{d}) : (1-\sigma(\mathfrak{d}))$ are in the ratio $\exp(\mathfrak{d}) : 1$, we rewrite:

$$\text{readout}(\mathfrak{d}) = \text{“chance of cow is } \exp(\mathfrak{d})/Z_{\mathfrak{d}}\text{; of dog, } \exp(0)/Z_{\mathfrak{d}}\text{”}$$

I hope some of you felt bugged by the above formulas' asymmetry: W measures "cow-ishness minus dog-ishness" — why not the other way around? Let's describe the same set of hypotheses but in a more symmetrical way. A common theme in mathematical problem solving is to trade irredundancy for symmetry (or vice versa). So let's posit both a W_{cow} *and* a W_{dog}. One measures "cow-ishness"; the other, "dog-ishness". They assemble to give W, which is now a matrix of shape $2\times$number-of-features. So $\mathfrak{d}$ is now a list of 2 numbers: $\mathfrak{d}_{\text{cow}}$ and $\mathfrak{d}_{\text{dog}}$. Now $\mathfrak{d}_{\text{cow}} - \mathfrak{d}_{\text{dog}}$ plays the role that $\mathfrak{d}$ used to play.

Then we can do hard classification by:

$$\text{readout}(\mathfrak{d}) = \text{argmax}_y \mathfrak{d}_y$$

and soft classification by:

$$\text{readout}(\mathfrak{d}) = \text{“chance of } y \text{ is } \exp(\mathfrak{d}_y)/Z_{\mathfrak{d}}\text{”}$$

To make probabilities add to one, we divide by $Z_{\mathfrak{d}} = \sum_y \exp(\mathfrak{d}_y)$.

Behold! By rewriting our soft and hard hypotheses for binary classification, we've found formulas that also make sense for more than two classes! The above readout for **soft multi-class classification** is called **softmax**.

RICHER OUTPUTS: REGRESSION — By the way, if we're trying to predict a real-valued output instead of a binary label — this is called **hard one-output regression** — we can simply return $\mathfrak{d}$ itself as our readout:

$$\text{readout}(\mathfrak{d}) = \mathfrak{d}$$

This is far from the only choice! For example, if we know that the true ys will always be positive, then $\text{readout}(\mathfrak{d}) = \exp(\mathfrak{d})$ may make more sense. I've encountered a learning task (about alternating current in power lines) where what domain knowledge suggested — and what ended up working best — were trigonometric functions for featurization and readout! There are also many ways to return a distribution instead of a number. One way to do such **soft one-output regression** is to use normal distributions:

$$\text{readout}(\mathfrak{d}) = \text{“normal distribution with mean } \mathfrak{d} \text{ and variance 25”}$$

By now we know how to do **multi-output regression**, soft or hard: just promote W to a matrix with more output dimensions.

As usual, we can define the goodness-of-fit (of a probabilistic model to data) to be the log of the probability that model would have given to the data. Since log of a gaussian is quadratic in the distance between mean and data, our loss is quadratic in the same. But in our readout the mean is just the decision function value $\mathfrak{d}$. So minimizing loss means minimizing the squared distance $(\mathfrak{d}-y)^2$. Food For Thought: We can avoid hardcoding constant variances by making $\mathfrak{d}$ two-dimensional and saying $\cdots$ mean $\mathfrak{d}_0$ and variance $\exp(\mathfrak{d}_1)$. Express loss in terms of the training data and the $2\times$ (features) weight matrix W.

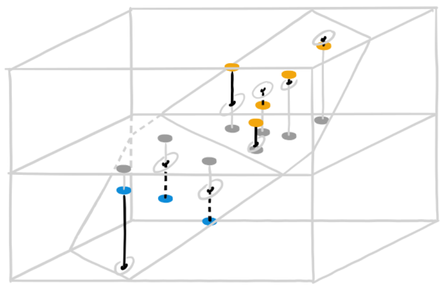
Figure 16: Minimizing loss means minimizing the squared distance $(\mathfrak{d}-y)^2$. We shade those distances **black**. Vertical axis: decision function value $\mathfrak{d}$. Horizontal axes: featurespace. Gray hexagon: (part of) the graph of the decision function ($x \mapsto w\cdot x = \mathfrak{d}$); the graph intersects $\mathfrak{d} = 0$ at the decision boundary (gray line). Inputs x are in dark gray; labeled pairs (x,y) are in orange and blue depending on y's sign. In regression, y can be any real number.

RICHER OUTPUTS: BEYOND CLASSIFICATION — Okay, so now we know how to use our methods to predict discrete labels or real numbers. But what if we want to output structured data like text? A useful principle is to factor the task of generating such "variable-length" data into many smaller, simpler predictions, each potentially depending on what's generated so far. For example, instead of using W to tell us how to go from (features of) an image x to a whole string y of characters, we can use W to tell us, based on an image x together with a partial string y', either what the next character is OR that the string should end. So if there are 27 possible characters (letters and space) then this is a $(27+1)$-way classification problem: ← VERY OPTIONAL PASSAGE

$$(\text{Images} \times \text{Strings}) \to \mathbb{R}^{\cdots} \to \mathbb{R}^{28} \to \text{DistributionsOn}(\{\text{'a'}, \cdots, \text{'z'}, \text{' '}, \text{STOP}\})$$

We could decide to implement this function as some hand-crafted featurization function from Images $\times$ Strings to fixed-length vectors, followed by a learned W, followed by softmax. Deciding on this kind of thing is part of **architecture**.

Food For Thought: A "phylogenetic tree" is something that looks like

```
(dog.5mya.(cow.2mya.raccoon))
```

or

```
((chicken.63mya.snake).64mya.(cow)).120mya.(snail.1mya.slug)
```

That is, a tree is either a pair of trees together with a real number OR a species name. The numbers represent how long ago various clades diverged. Propose an architecture that, given a list of species, predicts a phylogenetic tree for that species. Don't worry about featurization.