03:13
Good evening, and welcome to Lecture 5.
03:18
So up to now, we have been talking about just one
03:24
task, one NLP task, which is text classification.
03:27
And through other calls, we'll introduce
03:30
other tasks like machine translation,
03:33
question entering,
03:34
code generation, and so on. But for today, we're
03:38
going to stick with text classification. The
03:40
only thing that's going to change today is we're
03:42
going to introduce a different kind of model.
03:44
So, so far we've introduced linear models, and we're
03:48
going to introduce a more expressive type of model,
03:53
feed-forward neural networks, and we'll see that
03:56
the definition of these models and sort of how the
03:59
math works follows very nicely from the linear
04:03
models that we have introduced so far. so that's going
04:07
to be kind of like just a light extension of
04:11
what we talked about so far but in terms of how
04:14
powerful the models are it's actually not a very light
04:18
extension so these are much more powerful models
04:22
alright
04:23
a brief recap of what we talked
04:25
about last time we introduced the
04:30
last function prior to last lecture and in last lecture
04:34
we said okay, now we want to find a parameter setting
04:37
of our parameters down view that minimize the loss.
04:41
And for that, we introduced an algorithm,
04:44
the gradient descent algorithm,
04:45
which iteratively finds the best setting of Rw by
04:50
simply saying, okay, we're going to start somewhere,
04:53
maybe random initialization,
04:55
initialize our parameters to some numbers, and then
04:59
we compute the gradient of the loss with respect to
05:03
the parameters. and the gradient is the instantaneous
05:06
rate of change it's just telling us about the local
05:10
point where we are right now and so because
05:13
this is a local indicator we have to take
05:16
small steps so it's the instantaneous rate of
05:19
change so we take small steps from there and
05:22
we move in the direction of steepest descent
05:25
and then we repeat until convergence where we said
05:28
convergence can be perhaps performance on the validation
05:31
side is no longer improving or we have set a fixed
05:36

number of epochs and the epochs have come to an end.
05:40
All right, so that's the idea of gradient descent.
05:44
One last recap of last lecture is that we
05:47
derived the analytic gradient of the softmax loss
05:51
and we came up with these nice expressions.
05:56
This is the loss of the single example. We said the
05:59
partial derivative of that loss with respect to the
06:04
parameters of the target class is this expression here,
06:08
which is that, so we move in the opposite
06:11
direction of the gradient, so we are saying,
06:12
we are going at each step, we are making the class
06:17
vector of the target class to be more like the examples
06:21
from that class, and then we are making it less
06:25
like the examples from that class to the extent that
06:29
the model is already predicting the labels correctly.
06:32
So this way we are not overdoing the
06:37
thing of kind of moving the weights
06:39
in the direction of the target class.
06:42
So one way to think about this interpretation is
06:45
also that the scoring function of the linear model
06:50
is simply computing the dot products between the
06:55
class vectors and the feature vectors, right?
06:59
And so for the target class, we want
07:02
the dot product between the class vector
07:06
and the feature vector to be high.
07:09
And so we can ask ourselves, what does it take
07:13
for the dot product of two vectors to be large?
07:19
What does it take for the dot
07:21
products of two vectors to be large?
07:24
Any thoughts?
07:28
yeah
07:30
right exactly the components have
07:32
to be similar, similar direction
07:35
and so that's why we want basically that
07:39
vector to be similar to the vectors of the
07:41
feature vectors of the examples from that class
07:46
yeah we want the dot products to be large
07:48
so the vectors have to be kind of similar
07:54
right okay so that's the recap of what we did last
07:57
time and this is our plan for today we're going to
08:03
introduce a simple processing unit called a neuron and
08:09
we'll see that the definition of a neuron follows quite
08:12
nicely from the linear models we have defined so far
08:17
and once we have defined a neuron we will combine
08:22

neurons and by just sort of repeating the same
08:25
computation but arranged in different ways we
08:29
will see that we'll come up with much more
08:31
powerful models called neural networks in this case
08:35
we are looking at one particular architecture
08:37
of neural network, feed forward neural network
08:41
and as you will see there's not a whole lot of new
08:46
math or computation happening here, but what
08:49
is going to be kind of new and a little bit
08:53
difficult in neural networks is really kind
08:55
of how to train neural networks, how to get
08:57
them to actually work. So I'll give you some
09:01
notes on sort of things to keep in
09:03
mind when training neural networks.
09:07
Although these days it's a little bit easier with
09:10
frameworks like PyTorch, in the early days when
09:14
deep learning first started working on, it was
09:15
only sort of a few groups that were publishing good
09:18
results because they knew how to train deep
09:20
neural networks and no one else knew how to do it.
09:22
These days it's a little bit easier
09:24
for everyone to do it, but we still have
09:27
to keep in mind a number of things.
09:29
And lastly, I will give an example of a simple
09:35
feed-forward neural network which was developed
09:37
for NLP, which is the deep averaging network, which
09:41
is the enough work we are implementing in PA1.
09:45
And so once we do this lecture, and then
09:48
the next lecture on Thursday, then
09:51
we'll have everything we need to do PA1.
09:56
All right, any kind of comments
09:59
or questions up to now?
10:03
Okay, cool. Let's dive right in.
10:07
And all right, let's begin with a single neuron.
10:12
so so far we've looked at the case of multi-class
10:17
classification the single neuron actually kind
10:22
of behaves like a simpler case of multi-class
10:24
classification which is binary classification so
10:27
let's define binary classification so for multi
10:32
-class classification we use the softmax function to
10:36
transform the scores to probabilities right? And
10:41
when we have only two classes, when you only
10:45
have two classes then we only need one probability
10:51
and then that probability is going to determine
10:54

the probability of the target class and then the
10:57
probability of the non -target class because we
11:00
just have two classes so probability of the target
11:03
class once we know that compute the non-target
11:05
class probability by 1 minus this probability.
11:09
So in that binary classification
11:12
case, we just need a single score.
11:14
So once we have that score, we then
11:17
want to squash it to be between 0 and 1.
11:20
And the way we're going to squash
11:21
that is by using the sigmoid function.
11:25
We will see that there are other ways to squash the
11:29
numbers. But for now, we are going to say, OK, we're
11:32
going to use the sigmoid function here if we're
11:35
looking for probability for a number between 0 and 1.
11:38
And so here on the x-axis will be the score that
11:42
comes from our classifier, but we just have a
11:45
single score because it's a two-class situation.
11:48
And then if the number is negative and large,
11:52
not negative, then probability is 0. If
11:55
it's large and positive, probability is 1
11:57
according to this sigmoid function, right?
12:00
All right. Cool.
12:03
and so here let me spell out sort of the
12:06
parameters so you can see that here we end up
12:09
with a lot fewer parameters in the binary
12:13
classification case so for the multi-class setting
12:18
we have the scoring function as follows, we simply
12:22
multiply the weight matrix by the feature vector and out
12:26
comes a vector that gives scores for all the classes
12:30
the output is a vector and then we compute
12:33
the probabilities, exponentiate, normalize.
12:37
For binary classification,
12:39
we have just a weight vector. We no longer have a
12:45
matrix. We have just a single vector and we do the
12:49
product between that vector and the feature vector.
12:51
And out comes the scalar. So we just
12:54
have a single number as opposed to a
12:56
vector of numbers. Out comes the scalar.
12:58
And then we compute the probability by putting that
13:01
scalar through the sigmoid function. And so it's
13:06
1 over 1 plus the credential of minus the score.
13:10
And then, as I mentioned,
13:12
only two classes,
13:13

that probability determines the
13:16
probability of the non-target class.
13:18
Okay, cool.
13:22
So that binary linear classifier
13:26
is actually just a single neuron.
13:31
So here, the pink circle, we're going to
13:35
call it our neuron. So this is our neuron.
13:37
And the neuron is doing exactly what the binary
13:41
linear classifier is doing. It takes a bunch
13:44
of numbers corresponding to the feature vector,
13:48
multiplies them by a bunch of numbers. It
13:51
starts the elements from the weight vector.
13:55
Out comes a single number. the scalar that we're
13:59
getting in the case of the binary classifier this
14:04
was the score for the target class but here for
14:08
the neuron we'll actually see that we don't know
14:13
what the number actually represents we we just
14:17
know that this is going to be emitting some number
14:19
and it's just going to be a number that's going
14:21
to be useful for predicting the target class all
14:30
right so this is the neuron the elements of a neuron
14:34
are we have the weight vector we have an activation
14:37
function which just takes a real number maps it to
14:41
another real number and then so this is the representation
14:46
here we have outcomes the number, activation
14:50
function the product between the weight vector and
14:54
the feature vector so this activation function can be
14:59
any kind of non -linear function,
15:03
ReLU, 10, and others, not just a sigmoid function.
15:09
All right.
15:10
Once we have a neuron, we can now compose it and
15:14
combine it in different ways to come up with a network.
15:18
But the computation is exactly the same. We have
15:21
the same sort of very simple processing unit, which
15:25
in isolation is not very powerful. but once we
15:28
combine it in different ways and just repeat sort of
15:32
the same thing, we'll come up with a powerful model.
15:38
So here, we are not limited to a single neuron here.
15:45
We can actually have multiple neurons,
15:52
each taking the same input that we
15:56
have, So this is the feature vector, and
15:59
each is simply doing its own thing,
16:03
which is it has its own associated set of weights,
16:07
and it computes the number, puts it through
16:11

the null linearity, and then we have what is
16:14
called the activation of the neuron by doing that.
16:18
So here we can really go to town. we are not even
16:23
limited to have the same number of neurons as the
16:26
dimensionality of the input we can have like 100 neurons
16:29
here, they are all just a computing number and the only
16:34
thing we'll have by introducing more neurons is we'll
16:38
have more parameters because each neuron has its own
16:42
vector of parameters, so more
16:45
neurons more parameters in the model
16:51
but I kind of mentioned this
16:53
already but these numbers here
16:56
each neuron is emitting a number we don't
17:00
have to decide ahead of time what these
17:04
units are actually trying to predict
17:08
so suppose that we are doing spam, non
17:10
-spam classification and maybe the useful
17:14
features in that task are like, okay,
17:16
does the email contain the word free? Does it,
17:20
you know, the email address that's coming from,
17:23
that's sending the email, does it have weird
17:25
characters in the email? So these kinds of things.
17:27
We are not specifying these beforehand.
17:31
These neurons are just extracting some useful
17:34
features, are just computing some numbers that are
17:37
indicating some features, but we don't specify what
17:40
they are, and generally we don't know what they are.
17:43
they're just computing these
17:45
numbers to be useful for predictions
17:49
so
17:50
exactly, so here
17:55
what defines sort of what
17:58
is a good number here to have is really the loss
18:01
function and so the loss function is going to
18:05
determine what these hidden values here, these
18:08
outputs of these neurons, what they should
18:10
be and And so what we end up doing,
18:14
okay, so suppose this is the output layer,
18:16
we are simply saying, okay, whatever we predict
18:20
here should be the correct target class.
18:22
And if it should be the correct target class,
18:25
these numbers here should be useful for
18:30
predicting the target class. So that's the idea.
18:43
right
18:45

and
18:47
before we know it we have what is
18:49
called a multi -layer neuron network
18:51
which is that we can have this intermediate
18:55
we can have multiple layers here not just one
18:59
not just one layer of neurons we can
19:03
have multiple layers here and then we
19:06
can make a prediction here with the
19:10
the final layer
19:15
yeah so this is sort of generally
19:17
what it means that the network is
19:21
it's a deep network usually in deep
19:24
learning it just means that you have more
19:26
than three layers or three layers or more
19:29
but
19:30
made deep learning successful is this idea of saying
19:33
that okay one option we have is we can design a
19:37
network which just has the input here and then maybe
19:39
a giant layer here and then we directly from that
19:44
layer we go to the output. So that would be kind
19:47
of a shallow network but what deep learning has
19:50
actually shown is that by learning these sort of
19:54
hierarchical representations so you learn some features
19:57
and then you re-represent those features again.
20:01
So each one of these is a sort of a representation
20:04
of the input that is derived from earlier
20:07
representations and so by doing this kind of
20:09
thing where you're learning sort of a hierarchical
20:11
representation you're actually learning
20:13
different features at each of the layers.
20:17
Who has used something like
20:19
BERT, BERT representations?
20:21
BERT representations?
20:26
Okay so we haven't talked about pre-training yet
20:29
but when you pre-train a model such as BERT which
20:33
is like a language model those models have many,
20:36
many layers. I think BERT has over 700 layers, and
20:42
each of those is actually learning a representation
20:44
of your input. Say this is your sentence,
20:46
each of the layers is learning a representation.
20:50
In BERT, and generally, what has been
20:52
found in these models is that usually the
20:56
middle layers are the most useful, like they are
21:00
learning very sort of general purpose features of your
21:02

input. You can use those, use them for whatever you
21:06
like but the idea is that really that this layers here
21:10
this layer here this layer here and this layer here
21:12
is learning some representation of the input and
21:16
usually they are learning different representations and
21:28
so that's why we're sort of saying for network
21:31
for neural networks the final classifier here
21:33
is highly non-linear with respect to the original
21:37
input it's linear with respect to the layer
21:41
just before this classification layer but then
21:45
we don't know what these features are, so
21:48
that's why this classifier is not interpretable.
21:51
Whereas in a linear model,
21:53
this weight of the classifier were directly
21:56
interacting with the feature vectors here, the feature
22:00
vectors. So we could directly kind of say, okay,
22:03
this feature here has a high weight for this class,
22:08
so we can kind of talk about what is actually
22:10
contributing to the predictions. But now, not anymore,
22:14
because there's pods here that we don't know.
22:18
All right, cool.
22:21
What you'll find in books and in
22:26
papers, if you're reading papers,
22:28
note these cartoon figures here. You will
22:32
generally find this matrix notation. and
22:36
so we said that essentially what we have
22:41
is a vector associated with each neuron
22:46
and if we kind of stack those vectors together for
22:52
neurons in a single layer then we end up with a weight
22:56
matrix associated with that layer so the first row
23:02
is going to be the weights for the first neuron,
23:05
and so computing the activation for that neuron is
23:09
simply we grab the first row from the matrix and apply
23:14
the non-linearity, and then we have the activation.
23:17
Same thing for the second neuron. We take the
23:22
weights from the second row, do the product with the
23:26
feature vector, and then we apply the non-linearity.
23:30
So essentially what's going on is whatever computation
23:34
that we were doing for each neuron is simply now
23:38
done via this matrix operation. And so we can kind
23:43
of write this out in a single line of code, usually.
23:51
All right.
23:54
One critical point is that these activations here,
23:59
they are applied element -wise. so when we have neurons
24:06

in the same layer it's not that we are applying sigmoid
24:10
or whatever to all the numbers to get out some weird
24:12
thing we are actually saying each number is independently
24:15
computing its own activation and we squash it
24:19
independent of all the other numbers in the neuron so
24:23
these are independent computations they're not tied
24:26
together element wise application of the activations.
24:33
All right. This is the thing that
24:37
you might see in papers, actually,
24:39
when describing feedforward networks or networks in
24:43
general. So you have something like the following,
24:45
where for the first hidden layer, you have a weight
24:49
matrix, W superscript 1, and you do the computation.
24:55
for the second hidden layer you do that computation
24:59
but not using the feature vector but the representation
25:02
from the first hidden layer and so on so for
25:06
the last hidden layer the representation that we are
25:10
working on is the representation from the L minus 1
25:14
layer and so on
25:19
yeah so this is the notation for a feed forward
25:28
all right let's think about this a little bit
25:32
I'm going to give this is not super yeah
25:36
it should be nice and light so let's start
25:40
with this light question here I'll give
25:44
you maybe three minutes to think about it
27:03
you're welcome come to talk to people about it.
28:20
30 more seconds.
28:50
Okay, great.
28:52
Does anyone want to share what they're thinking?
28:56
Any thoughts?
28:58
Yeah, sure.
29:16
Yeah, very good.
29:18
Yeah, so the answer you said, as I see,
29:20
which is that without the nonlinearities,
29:23
the model collapses to a linear model.
29:27
Yeah, so we need these guys.
29:32
All right, yeah, so here's kind of an example of kind
29:35
of how we can think about it, which is that if we say
29:40
we have these data points that in the original input
29:44
space, they are not linearly separable, right? So they
29:48
are kind of complicated like this and a linear model
29:51
will not succeed at separating them because we are
29:55
directly going to be working with these representations.
29:58
So the idea is that if we add nonlinearities and
30:03

do sort of put these through a bunch of layers,
30:07
for example, and just sort of get a different
30:09
representation, we can end up with a representation
30:12
that is much easier to separate linearly.
30:15
So when we then apply that linear classifier
30:18
at the very end of a neural network,
30:20
we have obtained a representation
30:22
that's much easier to work with,
30:25
a representation that's easier to separate
30:27
with the linear classifier. So that's
30:29
kind of what's happening with the network.
30:32
we end up with that representation. It's not
30:35
interpretable, but it's nice and well-behaved
30:40
when we apply the classifier. We are able to do
30:44
a good job at predicting the observed labels.
30:48
Right.
30:52
So far we have talked about sort
30:55
of the sigmoid activation function.
30:58
Generally, what you're going to see
31:01
is the ReLU activation function or variations
31:06
of this activation function are much
31:08
more common than the sigmoid function.
31:11
One of the reasons here, you can see that this
31:13
activation function is simply computing the max between
31:18
the activation and zero. So it's kind of like this.
31:21
Whereas here we have to compute the exponential function.
31:25
So this is much more expensive, whereas this one
31:27
is just computing a simple max. and also the sigmoid
31:31
function suffers from what is called these saturating
31:36
gradients because when the activations are large
31:38
everything kind of like teppers off like that whereas
31:41
this one here just keeps going it doesn't kind of
31:45
chop off the activations it just keeps
31:47
going so much easier for training
31:52
and also this one kind of looks it's almost
31:55
linear so it's also easier to train in that
31:59
sense it's kind of nice and almost linear so
32:02
usually you'll see other variations
32:04
like leaky relo and so on
32:09
that sort of have the properties
32:11
that I just talked about as well so
32:14
yeah just that you will probably not see the
32:18
sigmoid tan and others are not super common anymore
32:24
alright so
32:27

here is another question um so with feed forwards and
32:33
because there's not a whole lot to uh kind of there's
32:36
not a whole lot new you might think okay i really
32:38
understand this uh but there are these things that
32:41
you might not think about so i want to have a little
32:44
bit more a few more check -ins than i usually have
32:47
just to make sure that we're thinking about sort of
32:49
what's really going on here all right so this is okay
35:48
30 more seconds
36:21
okay cool um does anyone want to share what yeah sure
36:39
right uh awesome uh cool thanks for reasoning
36:43
by elimination um yeah so in general
36:48
we do not want the the sort of the
36:51
neurons to interfere with each other
36:53
so if we are forcing them to kind of
36:57
share this probability mass by applying
37:00
the softmax, then you're saying basically
37:03
you're allowing the neurons, sort of different
37:06
features, to interfere with each other. So you want
37:09
the activation function to be applied element-wise.
37:14
We want to leave the features alone. Each
37:17
one should be doing its own thing. Element
37:18
-wise application of the activation function.
37:32
All right, so here is Python code, just Python
37:37
code for implementing this two-layer feed forward.
37:41
This is just basic Python code.
37:45
So we can, if we are using the sigmoid activation
37:49
function, we can begin off by defining
37:52
it here. So this is the activation function.
37:55
And then we have input here x. Here it's randomly
37:59
initialized. Typically, you don't want to do that
38:01
because you actually want your feature vector to
38:05
semantically express something, some data point, right?
38:09
And then the first hidden layer would apply
38:13
the nonlinearity and dot products between the
38:17
weights for that layer and the feature vector.
38:20
Second hidden layer,
38:22
dot products between the weights of that layer and
38:25
the hidden representation from the first layer,
38:28
nonlinearity.
38:29
And then the output here,
38:31
linear, just dot product between the weights of that
38:35
and the hidden representation from the second layer.
38:38
And then these are kind of the scores of the different
38:41

classes and we put them through a softmax, right?
38:45
So this is nice and easy. So we're just
38:48
doing dot products and then applying this non
38:54
-linearity here element -wise and that's it.
38:57
So it's very easy. and it's even easier with
39:01
deep learning frameworks now so I don't even have
39:04
to think about dot products and so on I just go
39:07
into PyTorch and say give me a linear layer of
39:09
these dimensions and then I get a linear layer
39:12
same thing for activation functions
39:14
there are many different activation functions
39:16
I can just say give me the sigmoid
39:18
leru and so on
39:20
you can do it even with fewer lines
39:23
of code with deep learning frameworks
39:26
but I can say that this is a kind of
39:31
simple computation.
39:37
So instead of designing the
39:39
architectures of neural networks,
39:42
we can really go to town and kind of do
39:45
different things, different exciting things.
39:47
One of the simpler things that we can do is kind
39:50
of determine how many neurons we have in a layer.
39:53
And so we are seeing here that the more
39:55
neurons, the more capacity the model has with
39:58
three neurons we can kind of see that okay
40:00
this is some kind of smooth decision boundary
40:04
six neurons it gets a little bit more complicated
40:07
and twenty neurons it's really kind of fitting really
40:12
overfitting to the data so once we have many
40:16
many neurons you have a lot of parameters and that
40:19
gives the network the capacity to even just memorize
40:22
the training data so the more neurons the more
40:25
capacity and the more likely that the network has
40:29
enough numbers to actually store your training data
40:33
and then just memorize it and do well in that way.
40:39
Alright on to the training loss.
40:43
The training loss is very similar to the loss that
40:47
we defined for the linear model. So here's the loss
40:52
of a single data point is, again, the negative log
40:55
probability of the target class. And then we
41:00
normalize, basically, by exponentiate, normalize. The
41:03
thing that's changing here is this scoring function.
41:06
In a linear model, the scoring function was simply we
41:09

had this expression here, whereas now we have a more
41:13
complex scoring function where we have the non
41:17
-linearity here. So that's the only thing that's changing.
41:21
and the training of objective again we can
41:24
express it in a similar way where we have the loss
41:29
of each example, we add the, we sum them up
41:33
normalize over the training examples we add a
41:36
regularization term over all the weights that we
41:39
have in that model here we have just W1 and W2
41:44
so that's the loss, we will not work through the
41:47
analytic gradient, for this you're welcome to
41:50
if you want so the thing that changes here notice
41:54
is just the scoring function so you can write
41:57
that out and figure it out if you like but PyTorch
42:02
can also easily do that for you in your code
42:09
here
42:11
regularization matters even more than in linear
42:14
models so for example so here if we have a tiny
42:19
regularization strength, which
42:21
means that we are regularizing
42:26
not too much. Basically, we have this irregular
42:29
kind of overfitting decision boundary, but
42:32
if we regularize a little bit more, then we
42:37
have a much smoother decision boundary here.
42:42
In neural networks, often what you have is not L2
42:48
or L1. What you might see more often is something
42:51
called dropout. And this was introduced in 2014.
42:56
The idea is the following. We say at training time,
43:00
we want to set some activations to zero. So we
43:03
essentially go in there and recall that each of
43:07
these arrows is just a weight. It's just a number
43:10
associated with a particular neuron. So these are the
43:13
weights for this neuron so we can knock out some
43:15
of these edges by setting them to zero randomly.
43:19
So for example in the first epoch we
43:24
decide which weights to set to zero. Second epoch
43:28
we decide a different set of weights to set
43:30
to zero and bring the others back in and so on.
43:34
The idea of doing this is just to introduce redundancy
43:37
in the network so that the network does not
43:39
just rely on a single path to make the predictions
43:45
instead we have multiple paths for making predictions
43:49
and so that this is a much more robust thing that
43:53
you end up with in PyTorch this is just a single
43:57

line you can just say dropout and then you specify
44:01
that so usually with dropout you specify the dropout
44:06
rate so you can say 0.3 or 0.8 or whatever but
44:12
single line, specify the rate, the dropout
44:15
rate, and you're done. It will be
44:17
dropping out things for you during training.
44:20
At inference time, we still use the entire network.
44:24
We still use the entire network, and the idea is
44:26
that, okay, this entire network knows how to
44:29
predict. It's very robust. It's great, and so on.
44:33
Great.
44:38
All right. So, we have the last function, and we have
44:43
understood that we need to really do
44:45
regularization in our work, neural networks.
44:48
One thing that we have to keep in mind is the
44:52
following, that we can still use gradient descent
44:55
with neural networks here. And the only thing
44:59
that is different from linear models is that we
45:02
are no longer doing convex optimization, where
45:06
there's a single global minimum for the function.
45:10
There are this local minima that we can get stuck
45:15
in when we are doing gradient and descend. You
45:25
can get stuck in this local minimum.
45:30
And if you initialize differently, if
45:33
you initialize the network differently,
45:36
you'll see that you get different results.
45:39
You'll see that you get different results. And so
45:42
in papers, for example, at the top conferences,
45:46
they usually now ask you that you report your
45:49
results with multiple initializations so that you
45:53
can't just say, oh, I have these great results
45:54
to report to the community, and it ends up being
45:57
just you got lucky once with gradient descent.
45:59
So we want to show that your result is robust
46:02
and we should trust it. So you have to specify,
46:07
okay, these are the, I have tried
46:09
multiple measurements of my network, so I
46:11
found different solutions, and so on.
46:14
In general,
46:16
even though we are working with non-convex optimization
46:18
with neural networks, these networks are actually
46:21
working they are doing a great job there is no theory
46:25
as to why this is sort of working but there are many
46:30
there are empirical results that actually
46:33

tell us why this might actually be working
46:38
so this is a very good paper, one paper
46:41
that was published at ICLEA 2019,
46:44
one of the machine learning conferences
46:48
called the lottery ticket hypothesis is one
46:52
of very good empirical results sort of showing
46:55
us why even though we are working with non
46:57
-convex optimization networks are, we are
47:01
able to train these high performance networks
47:03
we have the LLMs they are working great,
47:06
trained with gradient descent they're
47:08
huge and you know these are complex
47:12
you know problems and somehow we are able
47:15
to get a good solution Anyway, so this paper,
47:18
The Lottery Ticket Hypothesis, says the
47:20
following. It says, in these neural networks,
47:26
in a big neural network, there are sub
47:29
-networks that can be initialized such that,
47:35
when trained in isolation,
47:38
their performance matches
47:41
or actually outperforms the original larger network.
47:47
so there are sub-networks
47:51
that can be initialized so the initializing part
47:54
is actually doing a lot of work here so you
47:56
have to get the initialization right but they are
48:00
in there, there are multiple networks in there
48:03
so one way to think about it is that there is
48:05
a lot of redundancy in these neural networks
48:09
and so there are many winning tickets that's
48:12
why they talk about the tickets there are many
48:14
winning tickets and so we just need to find one
48:17
of them and if you find a winning ticket we have
48:20
good performance and so this is one result
48:24
kind of explaining kind of why the networks are
48:28
kind of able to train these
48:31
and they kind of
48:33
have a simple algorithm for kind
48:36
of finding these sub-networks
48:38
by
48:40
yeah by doing a bunch of experiments, dropping
48:43
out some of the layers and so on um yeah all right
48:50
um oh we kind of talked about this already uh
48:56
so let me see so maybe i'll just give you like
49:00
two minutes to think about it so you can also
49:02

just appreciate it but i think we talked about it
51:06
okay cool let's see yeah um all right so this one um
51:13
yeah, we kind of talked about it
51:14
already. I just wanted you to think about
51:16
it as well, which is that really,
51:22
even though it's not clear, we
51:25
don't have that interpretability,
51:26
still, we're actually just adjusting the parameters
51:30
to be good at making predictions
51:34
and maximizing the likelihood of the observed data.
51:38
Yeah, so A is the answer.
51:43
okay cool
51:45
so that's the
51:49
the feed forward and the way we would
51:51
do text classification with it which is
51:54
what we are doing oh sorry about that
51:58
okay the way we'll do text classification with it
52:02
is the following we have our input now we put the
52:07
input through a bunch of layers and And then the
52:11
classifier works with the outputs of those layers as
52:15
opposed to the original feature vector. And then we
52:19
apply the softmax at the top layer of the sigmoid
52:23
function if we just have a two-class setup. So in
52:28
PyTorch, you would literally say, OK, I have a two
52:30
-class setup, sigmoid, and I just call the sigmoid.
52:33
Or I have a multi-class setup, so I call the softmax
52:37
function. So that would be like just one line.
52:41
So now I want to show you an example of a simple
52:47
architecture for the feed-forward network
52:51
architecture in NLP, the deep averaging network.
52:56
And here on this slide is the entire architecture.
53:04
So we start off with the words in our input, So, for
53:10
example, predator is a masterpiece, and we are going
53:16
to grab vectors corresponding to each of these words.
53:23
We have not yet talked about how to generate vectors
53:28
for words. We'll talk about that in the next lecture,
53:31
but assume that we have some vectors, so vectors of
53:35
real numbers representing each of the words. so deep
53:39
averaging networks is I'm going to average these
53:42
vectors and I now end up with a representation of my text
53:49
as a single vector and that vector which is just an
53:54
average of these vectors goes into the feed forward
54:00
so it's deep in that I have here multiple layers
54:05
usually three or two layers it's deep, and it's averaging
54:09

it's taking the word embeddings here the vector
54:12
representations of these words and averaging them
54:17
so this representation here we can also think of
54:21
it as a continuous bag of words representation
54:26
in the sense that it's still a bag because we're
54:31
getting rid of the sentence order it does not solve
54:34
the problem of the bag of words in the sense that
54:37
If you have a sentence like, the movie was not good,
54:41
it was bad, and another one, the movie was not good,
54:47
it was bad. The movie was not bad, it was good.
54:50
They're going to have the same representation in
54:53
this continuous bag of words. The same problem you
54:56
have in the discrete bag of words we talked about.
54:59
Before, the difference between this
55:02
continuous bag of words and the discrete bag
55:05
of words we talked about is that for the
55:09
discrete bag of words, we had counts here.
55:11
We had literally like, okay, I've seen this
55:14
word two times, three times, but here we are
55:16
going, and that discrete bag of words is a
55:19
high-dimensional sparse vector. But this
55:22
continuous one is a low -dimensional dense vector.
55:27
So there are no zeros here. You are going to
55:30
find that and it's going to be low-dimensional.
55:32
And you can specify the dimensionality of
55:35
that through the dimensions of the embeddings
55:40
it's a very simple idea
55:43
simply average the embeddings
55:47
so who has
55:49
seen sort of word embeddings, algorithms
55:52
for learning what embeddings here
55:54
algorithms for learning what embeddings oh not many
55:58
people okay good that's good, we will cover those
56:03
algorithms in the next lecture and then once we
56:07
have covered those you have everything you need
56:09
for PA1 in the same lecture we will talk about
56:13
tokenization which is the last spot of PA1 as well
56:17
so everything will be covered in the last lecture
56:19
alright so that's the deep averaging network and
56:23
what was kind of interesting about
56:26
deep averaging networks was that
56:29
in NLP and in language, we appreciate the fact
56:32
that syntactic structure matters and it does matter,
56:36
but the Deep Averaging Network was doing really
56:38

well with a caveat. Okay, so these tasks are
56:43
standard analysis and textual entailment.
56:47
So in some sense, so these are
56:50
tasks where maybe structure you know,
56:53
you're getting like 80% performance, the other 20
56:56
% might be where the model is struggling because
56:59
the structure has been thrown out or some other
57:02
thing like that. But you are seeing that regardless
57:07
of the structure being thrown out,
57:09
you are doing just as well close to the
57:13
performance you are getting from a model that respects
57:16
structure. At that time, these were LSDMs.
57:18
For example, here, these are LSDMs.
57:22
And
57:23
notice that this is actually super fast,
57:26
136 seconds compared to the state of the
57:29
art at that time from this CNN model here
57:32
that's taking literally
57:35
way longer than this deep averaging network.
57:39
So you'll see in this assignment that
57:41
it's actually something that runs super fast
57:43
and you can get improvements from it.
57:46
So having said that, as I mentioned, structure
57:49
is important and these tasks are like sentiment
57:52
analysis where you can do well by doing some
57:55
keyword things. So if you're doing a task like
57:59
say, saw-cost intersection,
58:02
you might not do as well as you might be a
58:05
bit more far off from a model that respects
58:10
structure and sequential order
58:12
like an LSDM or transformer.
58:18
Anyway, so on these benchmarks, it was
58:21
doing kind of well on these simple tasks.
58:26
alright so let's talk a little bit about
58:30
training
58:31
neural networks
58:35
alright for training neural networks there are many
58:38
different ways we can come up with different architectures
58:42
and so on so there are many different things that
58:45
we have to think about so in just the feed forward
58:49
space we can think about two things when it comes to
58:53
the architecture, sort of the number of layers, the
58:56
number of neurons in each layer, so the depth, the
58:59
number of layers, the width, the number of neurons,
59:01

and
59:03
non-linearities we can pick from different options,
59:07
we can use different kinds of
59:09
regularization, like dropouts,
59:12
layer norm, batch norm,
59:15
and optimization.
59:17
We have a number of important things that we can
59:21
tune there, like learning rate and so on. All these
59:24
different things affect performance quite a lot.
59:28
So often, again, in papers, in deep learning,
59:31
usually you have to specify kind of what things
59:33
you actually tried and so on so people can
59:36
reproduce and also can see how robust results are.
59:41
One very important type of parameter that
59:45
we have to think about is the learning rate.
59:49
So here is, on the y -axis here is the epoch
59:56
number, and then the y-axis is the loss.
59:58
So if we have two large alerting rates, we
1:00:04
can see that you might get divergence right
1:00:07
away. So we are overshooting the solution,
1:00:10
and we basically are not going to converge.
1:00:13
Typically, you are looking for something
1:00:16
like this red curve here, so it's usually
1:00:18
good practice to plot out your loss function
1:00:22
your loss
1:00:24
as you train so you can see how it's behaving
1:00:28
so if you get it to be kind of too low your loss
1:00:32
might be behaving like this so slow convergence
1:00:34
and if it's not overshooting but it's still
1:00:38
kind of high you might get something like this
1:00:40
where you're not really going to converge as well
1:00:44
but this is a hyperparameter that's, if I'm
1:00:50
going to think about one hyperparameter when you're
1:00:52
training your models, it's a learning rate.
1:00:55
So this is actually, this makes a difference.
1:00:58
And there's something called
1:01:00
the learning rate schedule,
1:01:03
which says instead of kind of keeping the learning
1:01:07
rates fixed throughout training, we can have a schedule
1:01:11
that kind of adjusts the learning rate as training
1:01:15
progresses. so this is a simple one called the step schedule
1:01:19
where we reduce the learning rate at fixed epochs
1:01:26
so for example here we find that we are actually not
1:01:29
converging but then if we put down if we lower the
1:01:34

learning rate we see that we are making more progress and
1:01:37
lower it again so it's usually a good thing to do and
1:01:42
the cosine schedule is a good one because it
1:01:47
smoothly decreases the learning rate over time.
1:01:51
Again, in PyTorch you can literally specify
1:01:55
this in two lines that here's the starting
1:01:59
learning rate and I want the cosine learning rate
1:02:04
schedule and it will do everything for you.
1:02:08
So this is usually a good default
1:02:11
one that we can use in PyTorch.
1:02:22
Cool.
1:02:23
Let's think about this question here about
1:02:26
sort of network design and training dynamics.
1:02:29
So I'm going to give you three
1:02:32
minutes to think about this one.
1:04:55
Three more seconds.
1:05:44
Cool.
1:05:45
Does anyone want to share what they're thinking?
1:05:50
Yeah, sure.
1:06:01
Right, exactly. Awesome. Yeah, so you said
1:06:04
the answer is P, which is correct. So this
1:06:07
kind of determines what functions we can
1:06:11
actually learn, sort of the capacity, the
1:06:14
expressiveness, the expressiveness of the model.
1:06:18
Yeah, okay, I'm going to skip that one.
1:06:20
All right, so putting everything together,
1:06:24
when we are training neural networks,
1:06:27
we think about the architecture in terms of, so for
1:06:31
our simple fit forwards, we have to think in terms
1:06:33
of the number of layers, the layer width activation
1:06:37
functions and so on but you can actually be more
1:06:40
creative and come up with your own architectures we'll
1:06:44
look at creative architectures that people thought
1:06:49
about like Recurrent neural network where you have
1:06:57
basically connections going
1:07:00
backwards so here we are feeding forward but
1:07:03
you can have RNN RNNs were actually developed
1:07:06
by, the simplest RNN was developed by Jeff
1:07:10
Ellman, who was a professor here until he passed
1:07:14
away recently. He was in cognitive science.
1:07:18
So you can be creative about it. Come up with
1:07:20
some network architecture that people might use.
1:07:23
Transformer networks.
1:07:25
You know, somebody was super creative and came up
1:07:28

with a complex network that, you know, People have
1:07:32
tried to tweak and tune and try to outperform, but so
1:07:36
far it has stood the test of time. So that can be
1:07:38
you. You can come up with different architectures.
1:07:40
And in terms of optimization,
1:07:42
we're still using gradient descent.
1:07:45
It's very useful to think about the learning
1:07:48
rate when you're training and also the
1:07:51
learning rate schedules, though you can default
1:07:54
to the cosine learning rate schedule here.
1:07:57
Regularization, very important here.
1:07:59
you can do dropout by default if you don't
1:08:05
want to think too much about how you want
1:08:07
to regularize dropout is good these days
1:08:13
great
1:08:14
so yeah I just talked about sort of the network
1:08:18
design and how you can kind of be creative about this
1:08:20
so here is kind of I like this question here which
1:08:24
is going to get us thinking about bottleneck layers
1:08:28
bottleneck layers will come back we'll talk
1:08:30
about them later when you talk about parameter
1:08:32
efficient fine-tuning but let's let's think
1:08:35
about them for a second just here well I'll just
1:08:43
put maybe about three minutes two and a half
1:11:32
Does anyone want to share what they're thinking?
1:11:36
Thinking, any thoughts?
1:11:41
Yeah.
1:11:44
Correct, yes.
1:11:46
So the network here is doing some compression.
1:11:51
We have to throw out some information when
1:11:55
we introduce a bottleneck layer like this.
1:11:59
So, yeah, this is kind of worth thinking about
1:12:02
and
1:12:04
usually bottleneck layers are used so because
1:12:08
here we are getting perhaps we're getting
1:12:09
features sort of maybe some high level features
1:12:13
so typically they are used in connection with
1:12:16
what's called a skip connection so you would
1:12:18
go from this layer and then to the next layer
1:12:21
but you would also have a connection going
1:12:23
from the layer prior directly to the next one
1:12:27
so that's another kind of architecture design
1:12:31
sort of these skip connections I think they
1:12:33
were introduced by Hinton and in his group
1:12:38

and
1:12:40
bottleneck layers will also make a comeback when we
1:12:43
start talking about parameter efficient fine tuning,
1:12:47
BAFT so let's get them in the back of our minds
1:12:54
alright so I kind of talked about this
1:12:56
already throughout which is that you can
1:12:59
easily implement
1:13:00
feed-forward neural networks and all
1:13:02
kinds of neural networks in PyTorch.
1:13:04
A lot of things can just come with a single line.
1:13:10
Different
1:13:11
optimizers, different
1:13:13
learning rate schedules.
1:13:14
It's super, super nice and easy.
1:13:19
You'll find that pleasant.
1:13:21
On the other hand, I think I kind of
1:13:23
I'm a bit conflicted because it's like you're just
1:13:26
specifying things and you don't know what's going on
1:13:28
it's like okay give me a dropout it's like okay what
1:13:31
is going on I don't know dropout right so yeah it's
1:13:36
it's good and bad but mostly good all right one thing
1:13:42
to keep in mind is that for a lot of minor tweaks like
1:13:44
if I introduce dropout am I going to get like a 10%
1:13:47
improvement in general not only for PA1 but in general
1:13:51
usually those give you some minor improvements,
1:13:54
like maybe less than a percent or something like that.
1:13:58
If you're really super off, maybe by like 10% for the
1:14:03
target performance in the assignment, it's likely
1:14:05
because perhaps you have the wrong network size or
1:14:10
optimization is completely off, or maybe you have bugs
1:14:14
in your code, but it's not because you did not add
1:14:17
like a cosine learning rate schedule or some other
1:14:21
thing, or you didn't drop out or anything like that.
1:14:25
So where can I find, like, where can I find, like,
1:14:28
text classification data sets that I can use for,
1:14:32
like, an experiment with? So Hugging
1:14:34
Face is a super great resource.
1:14:37
You will find lots of data sets there, not only for
1:14:40
text classification, but for a bunch of NLP tasks.
1:14:43
So, yeah, Hugging Face data sets.
1:14:46
So this brings us to the end of text classification.
1:14:50
we'll start a different task on Thursday and we've
1:14:55
defined the problem we gave some examples we also
1:14:59
gave two types of models sort of linear models also
1:15:04

called logistic regression and feed forward networks
1:15:07
and next time we'll look at how
1:15:11
we can learn word representations
1:15:14
awesome
1:15:53
right
1:15:58
Those are not activations per se, but you're right.
1:16:04
For layer norm, you are saying that
1:16:08
you want to basically make the
1:16:11
activations to be within a certain range.
1:16:14
Right.
1:16:16
These are not activations.
1:16:18
Yeah, these are not activations.
1:16:21
Yeah, exactly. Okay, no worries. That's so good. So
1:16:29
I had a question regarding the dropouts.
1:16:32
So the illustration kind of shows the feed-forward
1:16:36
network, but a couple of the neurons are extra just
1:16:39
because we're dropping out of the path, essentially.
1:16:42
So I guess it propagates a lot of neurons later down
1:16:44
the path that are also deactivated for, let's say,
1:16:47
because we cut out two of the neurons on the lower
1:16:50
layer, the start layer, so we have neurons in the later
1:16:53
layers. What normally would have been passed and
1:16:55
maybe they're cut, right? Or I guess we're cutting
1:16:57
on the path, so whatever neurons don't have any inputs
1:17:00
or outputs, we're cutting on the path, right? Yeah,
1:17:02
exactly. So they're just not part of the computation
1:17:05
because they zero out. So the network specification,
1:17:09
everything stays the same. It's just that the math
1:17:11
is kind of turning out to be zero on those paths.
1:17:14
Right.
1:17:15
Right. So I guess my question is, it says that we're
1:17:22
supposed to encourage redundancy and robustness with
1:17:24
this path. So whereas maybe you said maybe a path in
1:17:28
the network on the left may have maybe remembered a path,
1:17:31
a specific path for some certain feature. Right. It
1:17:33
must just kind of rely on exactly just one path and
1:17:38
then kind of forget everything else and not use that.
1:17:43
So discouraging over reliance on
1:17:45
just a small part of the network.
1:17:46
So is that to say that the network on the
1:17:49
right would try to consolidate multiple,
1:17:51
I guess, paths into one more general path?
1:17:55
So maybe some path that no longer exists
1:17:57
from the left network is now consolidated
1:17:59

maybe in a way in the right network.
1:18:00
Yeah.
1:18:03
Some paths, basically,
1:18:06
you end up with...
1:18:09
Okay, think about it this way.
1:18:12
We might end up just relying on a single
1:18:16
feature, for example, in this network because it
1:18:18
does so well, the network just prefers it.
1:18:20
But if we are actually dropping out that
1:18:23
feature sometimes, we are also saying,
1:18:25
you know, you can also rely on these features
1:18:27
and learn how to predict with those. So that way
1:18:30
you generalize better because you don't only
1:18:32
rely on that one feature. When it's not there,
1:18:35
we, yeah. So now you can see the
1:18:36
features are the neurons here.
1:18:38
Mm-hmm. The different part because they are
1:18:41
doing features extraction at each layer.
1:18:43
Right. Right. Yeah. So like, for example, if you're
1:18:46
classifying images, instead of relying on color,
1:18:48
maybe relying on shapes, that kind of thing. Mm-hmm. I
1:18:51
see. Okay, that makes sense. Thank you. Sure, yeah.
1:18:54
Hi, I just have some questions
1:18:56
about the data application.
1:18:59
I'm a statistics student, and
1:19:01
I want to enroll in this class,
1:19:03
and I see there are some available fees. Yes. So
1:19:08
is there any possible I can be enroll in? Oh, yeah.
1:19:11
I think you should be able to
1:19:13
because, as you said, there are.
1:19:15
So this is week three, right? Oh, yeah. You're a
1:19:18
graduate student? Oh, yeah. Oh, yeah, you should be
1:19:20
fine. I think because for graduate students, I think
1:19:23
you should be able to add in third week at most.
1:19:29
So I actually don't know then how it
1:19:32
would work, but let me put it on this one.